Manuel d'utilisateur

Csp To Java

Compilateur de programme CSP en JAVA

Benjamin Le Bozec

24 juin 2005

version 1.0

Table des matières

Introduction

L'application présentée ici est un compilateur de programme CSP en JAVA. Afin de faciliter son utilisation, une interface graphique a été réalisée, et ce document a pour mission d'en expliciter le fonctionnement.

De plus, une fenêtre *contrôle de l'exécution* a été créée afin de permettre à l'utilisateur de suivre le déroulement du programme en temps réel. Cette seconde interface est aussi décrite dans ce manuel.

1 Installation et lancement

Installation

Tout d'abord, récupérez et installez le jdk (Java Developpement Kit) 1.5 ou une version ultérieur (sur le site http ://www.sun.com par exemple). Il est nécessaire au bon fonctionnement de l'application, d'une part parce qu'elle a été développée sous JAVA, et d'autre part parce que le compilateur `javac` est nécessaire pour exécuter le programme JAVA résultant de la compilation.

Ensuite récupérez le fichier `CTJ_x.zip` contenant l'application, puis décompressez-le dans le répertoire de votre choix. Il contient :

- l'application CTJ (`CTJ.jar`) ;
- un dossier `csp_runtime`, nécessaire à l'exécution du programme compilé ;
- et un dossier `examples`, contenant quelques exemples de programmes CSP.

Lancement

Pour lancer CTJ, il suffit de lancer le fichier `CTJ.jar`. Actuellement, la plupart des systèmes d'exploitations gèrent automatiquement les exécutables jar. Si ce n'est pas le cas, il suffit d'utiliser la commande :

```
java -jar CTJ.jar
```

dans une console DOS ou LINUX par exemple.

2 Interface avec le compilateur

Lors de l'exécution de l'application, la première fenêtre qui apparaît constitue l'interface avec le compilateur :

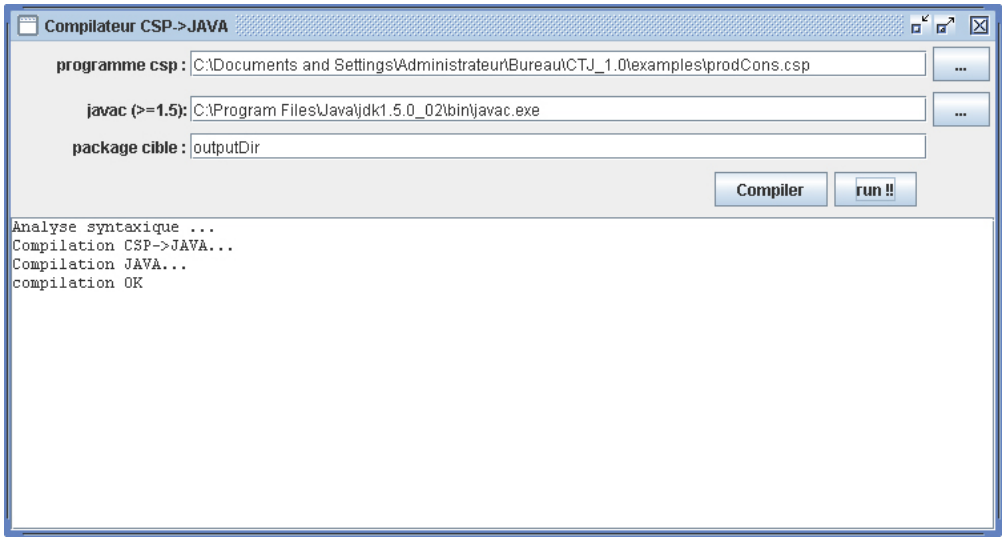


FIG. 1 – Fenêtre du compilateur

Les trois champs de texte en haut de la fenêtre permettent un paramétrage rapide du compilateur :

- `programme CSP` est le chemin d'accès au fichier contenant le programme CSP à compiler ;
- `javac` est le chemin d'accès du compilateur JAVA fournit avec le jdk1.5 ;
- `package cible` correspond au package dans lequel les fichiers JAVA créés seront placés. Plus concrètement, il s'agit du répertoire dans lequel les fichiers seront placés. Attention cependant : lors de la compilation, le répertoire est entièrement vidé de son contenu.

Les deux boutons `"..."` ouvrent une boite de dialogue `parcourir`, permettant de sélectionner facilement le programme CSP et le compilateur `javac`.

Deux boutons sont aussi disponibles :

- le bouton `compilation` exécute le compilateur ;
- le bouton `run`, exécute le programme JAVA résultant de la compilation.

Enfin, la zone de texte située en bas de la fenêtre permet d'afficher les différents messages résultant de la compilation.

Compilation

Rappelons que la compilation est composée de trois phases :

- une phase **analyse syntaxique**, détectant les erreurs syntaxiques (comme par exemple l'oubli d'un point-virgule, etc.) ;
- une phase **compilation CSP->JAVA**, créant les fichiers JAVA et indiquant d'éventuels problèmes dans le programme CSP (tels que la re-déclaration d'une variable ou l'utilisation d'un indiçage incorrect pour un tableau) ;
- et la phase **compilation JAVA**, correspondant à l'exécution du compilateur `javac` et créant les fichiers .class associés aux fichiers .java précédemment compilés. Les erreurs non prises en charge par le compilateur précédent seront donc détectées ici.

Les différents messages résultant de ces phases sont affichés dans la zone de texte de la fenêtre.
Le message `Compilation OK` indique que la compilation a réussi et que le programme est prêt a être exécuté.

Lancement du programme

Il est possible grâce au bouton `run` de faire apparaître une fenêtre *à usage unique* (c'est-à-dire qu'une fois l'exécution terminée, elle ne pourra être relancée qu'en appuyant une nouvelle fois sur le bouton `run`), permettant le contrôle de l'exécution du programme ainsi que le suivi de son déroulement. Cette fenêtre est décrite dans la partie suivante.

3 Fenêtre de suivi et de contrôle de l'exécution

Elle apparaît lorsque le bouton `run` de la fenêtre précédente est appuyé. Elle se présente de la manière suivante :

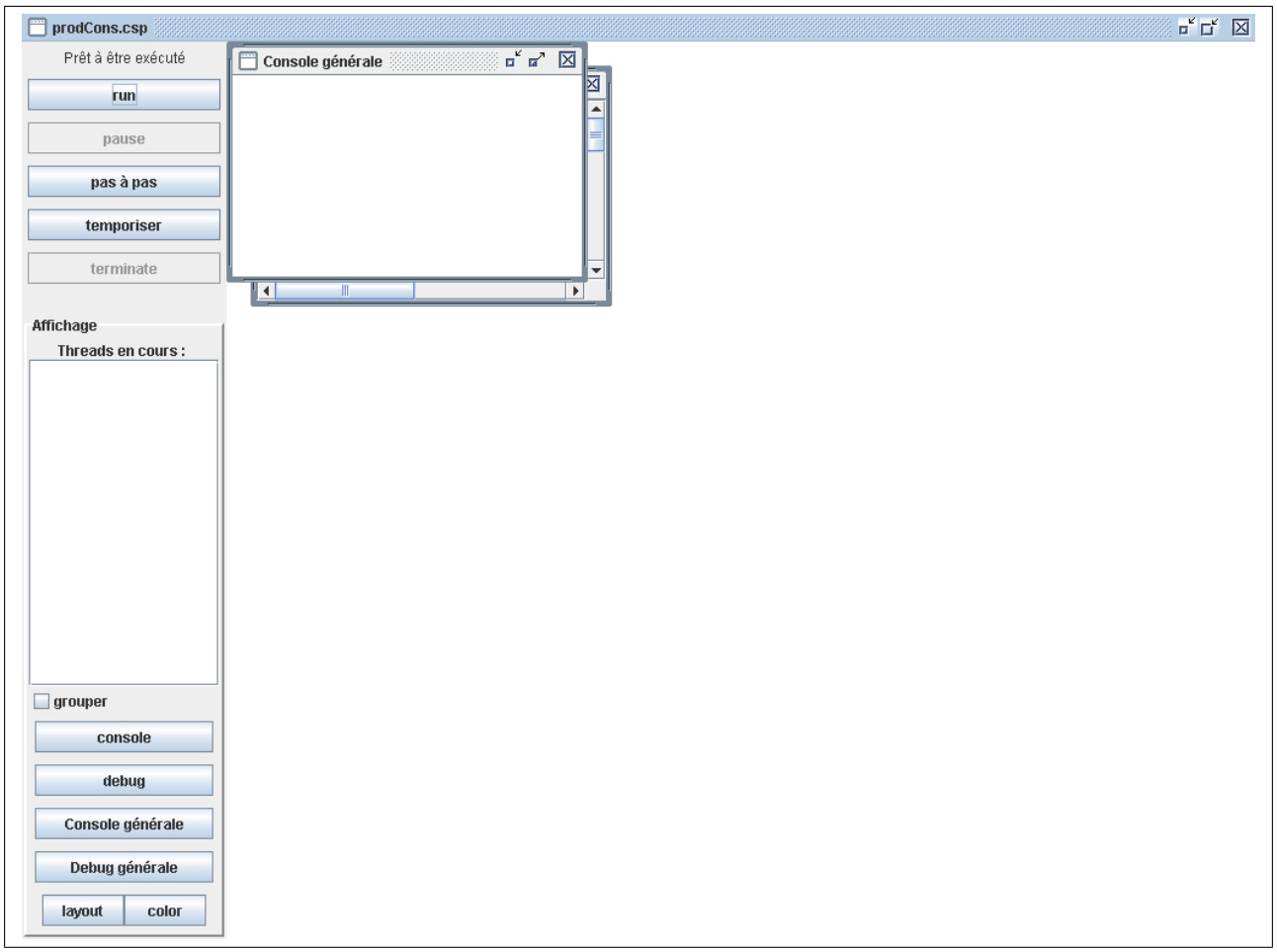


FIG. 2 – Fenêtre de contrôle de l'exécution

On peut remarqué qu'elle est divisée en deux panneaux :

- Un premier panneau situé à gauche de la fenêtre et permettant de manipuler exécution et affichage ;
- Un second panneau s'étendant sur le reste de la fenêtre et contenant de plus petites fenêtres.

Contrôle de l'exécution

Il est possible grâce aux cinq boutons situés en haut à gauche :

FIG. 3 – Contrôle de l'exécution

- le bouton **run** permet d'exécuter le programme en temps réel ;
- le bouton **pause** permet de stopper l'exécution ;
- le bouton **pas à pas** permet de n'exécuter qu'une commande à la fois ;
- le bouton **temporiser** permet de ralentir l'exécution en exécutant une commande à la seconde ;
- et enfin le bouton **terminate** permet d'arrêter définitivement le programme.

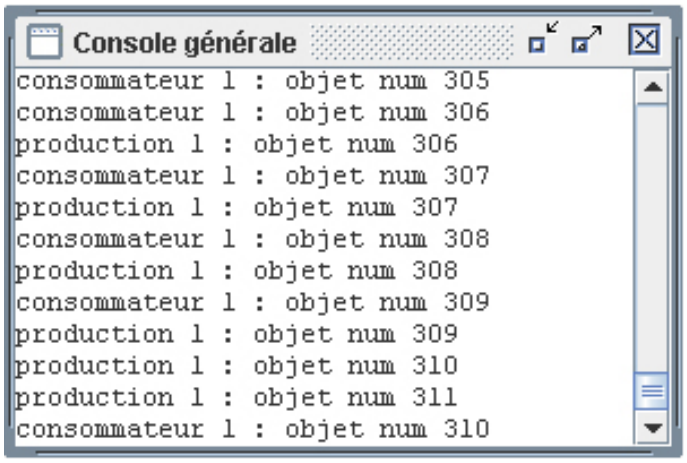
Notons aussi que l'état actuel de l'exécution est indiqué en haut à gauche de la fenêtre.

La partie affichage du panneau gauche permet quant à elle de gérer l'affichage, c'est-à-dire de gérer les sous-fenêtres apparaissant dans le panneau de droite. Elle est composée d'une liste des processus en cours, d'un *checkbox* et de six boutons :

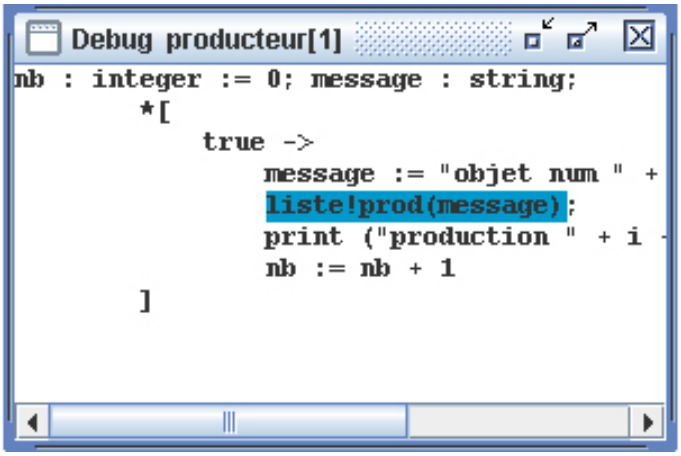
Avant d'expliquer le fonctionnement de ces différents boutons, précisons d'abord que les sous-

fenêtres situées dans le panneau de droite peuvent être de deux types :

- ce peut être une fenêtre *console*, affichant les différentes sorties des processus (c'est-à-dire affichant les messages émis par la commande `print`) :



– ou une fenêtre *debug*, permettant le suivi de l'exécution en indiquant la commande que chaque processus s'apprête à exécuter :



Chacune de ces sous-fenêtres peut alors être associée à un ou plusieurs processus. Une `console` peut être associée à plusieurs processus, qu'il soit ou non de même type, tandis qu'un `debug` ne peut être associé qu'à des processus de types identiques.

On note aussi que deux sous-fenêtres `debug général` et `console générale` sont déjà présente avant le lancement de l'application. Ces deux fenêtres correspondent respectivement à un `debug` associé à tous les processus (et affichant par conséquent le programme CSP dans sa totalité), et à une `console`, elle aussi associée à tous les processus. Le `debug général` est un cas exceptionnel de `debug` où des processus de types différents peuvent être associée à la même sous-fenêtre.

Expliquons alors le fonctionnement des différents boutons de l'affichage.

Les boutons `console générale` et `debug général` permettent d'afficher respectivement la `console générale` et le `debug général` s'ils avaient été fermés.

Les deux boutons `console` et `debug` nécessitent que des processus soient sélectionnés dans la liste située dans le panneau de gauche. Ils permettent alors de créer de nouvelles sous-fenêtres qui seront automatiquement associées à ces processus. Si l'option `grouper` est déselectionnée, une sous-fenêtre sera créée pour chaque thread, sinon une seule sous-fenêtre à laquelle seront associés tous les processus sera créée.

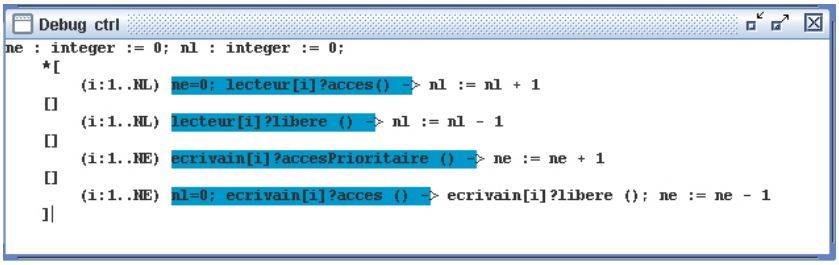
Enfin, deux autres boutons sont disponibles :

- le bouton `layout` permet de répartir les différentes sous-fenêtre sur la totalité du panneau de droite ;
- le bouton `color` permet de choisir la couleur avec laquelle les commandes seront surlignées dans un `debug`.

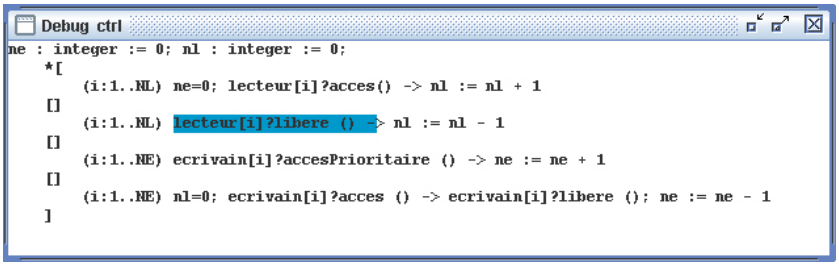
Précisions sur la sous-fenêtre `debug`

Comme ceci a été expliqué précédemment, un `debug` permet au processus d'indiquer la prochaine commande qu'il va exécuter.

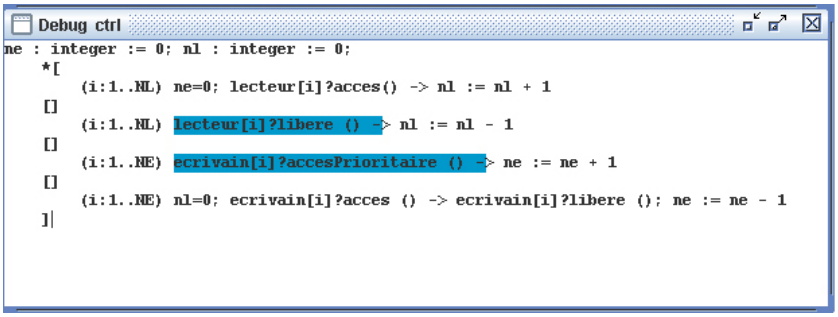
Précisons cependant qu'une commande alternative est gérée un peu différemment. En effet, Lorsque l'exécution arrive à une telle commande, la fenêtre `debug` surligne dans un premier temps les gardes qui vont être évaluées :



Ensuite, seule la garde choisie par l'alternative est surlignée, et l'exécution continue dans la liste de commandes correspondante :



Dans le cas où aucune garde n'est évaluée à `vraie`, mais où certaines sont évaluées à **neutres**, les commandes d'e/s correspondantes sont alors surlignées :



4 Sauvegarde des paramètres

Afin de ne pas avoir à répéter les étapes de configuration à chaque lancement de l'application, un mécanisme de sauvegarde au format XML a été implémenté. Ainsi, lors de la fermeture, un fichier `cspConfig.xml` est créé dans le même dossier que l'application.

Les paramètres sauvegardés sont :

- les trois paramètres de la première fenêtre (fenêtre de compilation) ;
- et la couleur de surlignage des commandes de la fenêtre d'exécution.