MULTI-TACHES EN ADA

Souvent sous-exploitées les capacités du langage Ada dans le domaine du multi-tâche sont impressionnantes de facilitées. Nul besoin d'utiliser des pointeurs et autres mécanismes délicats à coder, tout est pris en compte par le langage. Outre le fait premier de faire tourner plusieurs tâches avec une sensation qu'elles s'exécutent concurremment ou réellement concurremment pour les heureux possesseurs de système multi-processeurs, le multi-tâche permet facilement de lancer des actions asynchrones qui demandent des contorsions de programmation pas possible sinon. Le cas typique consiste à prendre en compte à la suite plusieurs actions de l'utilisateur et de lancer une tâche à chaque action sans attendre la fin de la précédente.

1) Déclaration d'une tâche en Ada

Une tâche en Ada se défini comme un type de donnée complété par le code même de la tâche.
Pour commencer voici une tâche toute simple qui ajoute un dans une variable et s'arrête à 99 :

```
task type Compte;
task body Compte is
  begin
    loop
      Valeur := Valeur + 1;
      delay 0.5;
    exit when Valeur > 99;
    end loop;
  end Compte;
```

La déclaration de la tâche s'effectue comme une variable :

```
C : Compte;
```

Ainsi que les autres objets Ada la déclaration provoque l'élaboration de l'objet. Autrement dit, la tâche démarre à ce moment. Bien entendu le programme principal ne se terminera qu'une fois toutes les tâches terminées.

Pour finir voici le code complet d'un petit exemple qui comporte une deuxième tâche affichant le compteur :

```
with Text_IO; use Text_IO;
procedure Taches is

Valeur : Natural := 0;

task type Compte;
task body Compte is
  begin
    loop
      Valeur := Valeur + 1;
      delay 0.5;
    exit when Valeur > 99;
    end loop;
  end Compte;

task type Affiche;
task body Affiche is
  begin
    loop
      Put_Line (Natural'Image(Valeur));
      delay 1.0;
    exit when Valeur > 99;
    end loop;
  end Affiche;

C : Compte;
A : Affiche;

begin
Put_Line ("Compte et affiche concurremment.");
end Taches;
```

La compilation s'effectue de façon classique :

```
% gnatmake taches
```

Vous pouvez faire varier les valeurs des délais d'attente dans les boucles pour voir le comportement du multi-tâche Ada sur Mac OS X.

2) Tâches Ada et Carbon :

Voici une petit programme qui permet de combiner Carbon et le multi-tâche :

```
with Text_IO; use Text_IO;
with CoreServices.CarbonCore.MacTypes;
use CoreServices.CarbonCore.MacTypes;
with CoreServices.CarbonCore.DateTimeUtils;
use CoreServices.CarbonCore.DateTimeUtils;

procedure Hello is

task type Heure;
task body Heure is
  DT : aliased UInt32;
  begin
    loop
      GetDateTime(DT'access);
      put_line("Date - Time : " & UInt32'Image(DT));
      delay 1.0;
    end loop;
  end Heure;

AffHeure : Heure;

begin
put_line("Hello again, with tasking Ada and now Carbon.");
end;
```

La bibliothèque Carbon est requise pour la compilation :
% gnatmake hello -largs /System/Library/Frameworks/Carbon.framework/Carbon

Comme dit en préambule l'étape suivante est de combiner le multi-tâche avec les interactions de l'utilisateur.
Un second exemple est la traduction en Ada d'un exemple d'Apple affichant deux fenêtres comportant une balle rebondissant à l'intérieur. L'utilisateur peut déplacer les fenêtres à l'écran avec la balle rebondissant en même temps. Le programme "buffered windows" est présent sur le site "http://www.adapower.net/macos". Il s'utilise avec ProjectBuilder.

Pascal Pignard, avril, mai 2002.