Guillaume RIPOLL
L3 -> L3

Présentation : 31/07/2014

Stage facultatif sur la génération de code parallèle pour les langages synchrones à l'IRIT avec l'équipe ACADIE

In : *Fichier Signal*

```
type integer;
type boolean = enum (true, false);
type test = enum (un, deux, trois);

process P = (? integer a; boolean b; ! integer u1, u2, v1, 2, x, y, z;)(
|x := a when b
|y := x +1
|z := a +3
|u1 := (y $1 init 0)
|v1 := (y $1 init 1)
|u2:= u1 default 0
|v2:= v1 default 0
|)end;
```

Remarque :
On suppose un code aplatit.

analyse syntaxique

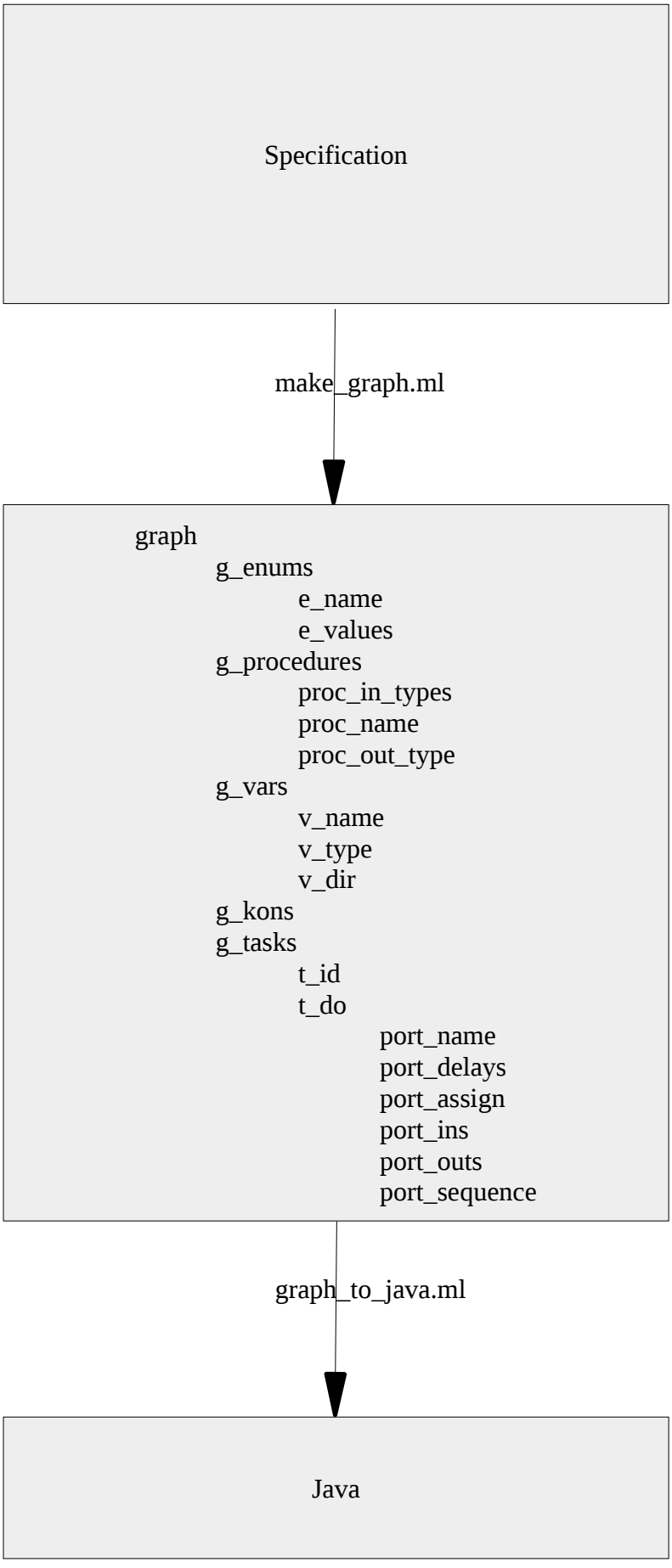
Ter_chk_spec
Ter_no_submod
Ter_arith_to_call

```
Specification
    process_list
        header
            process_name
            signal_declarations
                input_signal_list
                output_signal_list;
                local_signal_list
            local_process_list
        body
            assignment_list
                assigned_signal_name
                signal_expression
            constraint_list
                constraint_kind
                left_signal_name
                right_signal_name
            instantiation_list
                instance_process_name
                instance_output_signals
                instance_input_expressions
    type_declaration_list
    procedure_declaration_list
```

Ter_chk_spec
Garantit une spécification valide en vérifiant les différentes contraintes du langage.

Ter_no_submod
Minimise le programme de sorte à n'avoir plus qu'un seul process, et que la local_process_list soit vide. Ajout de variables locales, assignations, contraintes et instantiations.

Ter_arith_to_call
Convertit les appels aux opérateurs binaires +, -, * par des appels de fonctions.


Specification
make_graph.ml
graph
g_enums
e_name
e_values
g_procedures
proc_in_types
proc_name
proc_out_type
g_vars
v_name
v_type
v_dir
g_kons
g_tasks
t_id
t_do
port_name
port_delays
port_assign
port_ins
port_outs
port_sequence
graph_to_java.ml
Java

Template Java :

data
- Constantes.java
- Count.java
- GlobalData.java
- Signal.java
- Signal_Int.java
- Signal_Bool.java
- Signal_... .java

exceptions
- Uncoded_function.java

main
- Main.java

thread
- Ter_Runnable.java
- ... .java

usable
- Usable.java

Légende :
Vert : fichiers constants

Main :

création globale de GlobalData et de Count
création de l'exécuteur des tâches
création des tâches, ajoutées à la liste de GlobalData
définition des dépendances entre tâches
tant qu'on peut lire le fichier In
    pour chaque tâche n'ayant pas de dépendances
        incrémenter Count
        lancement de la tâche
    on attend Count
    fin de cycle : reset du GlobalData
fin : shutdown de executor

Chaque tâche :

I - hérite de Ter_Runnable, avec un run qui garantit
  1- l'exécution d'une méthode compute et en fonction du résultat
    a - la décrémentation du nombre de dépendances des tâches dépendantes
    b- l'exécution des tâches dont le nombre de dépendance devient null
  2- la décrémentation de Count
II - implémente compute, méthode chargée du calcul spécifique et renvoyant un booléen pour indiquer si oui ou non le calcul s'est bien passé

Usable :

1 - Un enum est créé dans Usable pour chaque enum du graph.
Pour chaque ajout d'enum, une classe héritant de Signal est créée qui implémente le type de variables Signal<NomEnum>
2 - Une fonction est créée pour chaque procedure du graph.
Quand on ne connaît pas le code de la fonction, ce dernier consiste en une levée de l'exception Uncoded_function.
Codes connus : add, mul et sub qui prennent 2 entiers et renvoient un entier.

Syntaxe du fichier In :

Une ligne par cycle.
Chaque ligne a la forme
nom1=valeur;^nom1=présence;nom2=valeur2; ... où '^' est un espace
valeur et présence sont soit converties, soit considérées comme non présentes si on ne les trouve pas dans la ligne, ou si on détecte la valeur '*'.