11 APPELER UN PROGRAMME JAVA DEPUIS PROLOG IV

(version 1.0béta - Octobre 1998)

CE CHAPITRE décrit comment, depuis Prolog IV, effectuer des appels à des méthodes ou classes JAVA définies sur le poste local ou accessible sur une autre machine par un réseau de style Internet.

11.1 PRIMITIVES

Un très petit nombre de primitives permet de lier Prolog IV à des méthodes JAVA définies localement ou à distance.

11.1.1 init_server/1

Ce prédicat permet d'initialiser l'adresse du serveur. Il doit être appelé avant tout accès au serveur, c'est-à-dire avant l'utilisation de tout prédicat écrit en JAVA.

L'adresse du serveur doit être fournie sous la forme de quatre nombres séparés par des points (adresse IP) : a.b.c.d.

Exemples :

```
>> init_server("128.246.150.249").
true.
>> init_server("128.246.150").
false.
```

11.1.2 new_java_pred/5

Ce prédicat permet de créer un nouveau prédicat dont l'implémentation est écrite en JAVA sur un poste local ou réseau. Pour l'utiliser, vous devez disposer du paquetage JAVA « *prolog* ». Vous devez également lancer le serveur JAVA fourni avec le paquetage « *prolog* ».

Ce serveur peut être lancé sur le poste local ou sur un autre poste pourvu que la machine possède une adresse Internet (adresse IP de la forme a.b.c.d) et que vous la connaissiez. Les fonctions JAVA qui représenteront les prédicats Prolog doivent toujours avoir une déclaration comme indiquée ci-après :

```
public static P4Goal nom_de_la_fonction (P4Goal pred_courant)
```

L'argument « pred_courant » représente le prédicat de départ de Prolog IV et le terme « *P4Goal* » renvoyé par la fonction représente le prédicat qui sera unifié avec celui de Prolog IV (il doit donc avoir la même arité). Vous devez connaître le nom de la fonction JAVA et de la classe la contenant ainsi que l'URL (adresse Internet) de l'ordinateur où est située cette classe (sauf dans le cas où celle-ci se trouve sur le poste local).

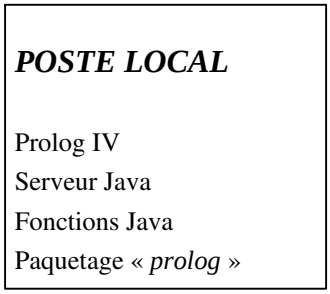
Le fichier « *exemples.java* » contient la classe « *exemples* » qui est munie de plusieurs fonctions utilisables pour l'appel de fonctions JAVA depuis Prolog IV.

Pour l'utilisation du paquetage « *prolog* », reportez-vous à la documentation API JAVA fournie avec celui-ci.

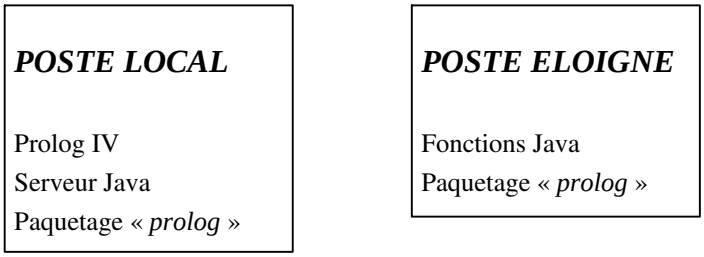
Ce paquetage doit être présent sur la machine où se trouve le serveur JAVA ainsi que sur la (les) machine(s) où se trouvent les fonctions JAVA appelées.

Les schémas suivants présentent toutes les configurations possibles pour la disposition des différents éléments :

(1) Protocoles utilisables : file, http, ftp.

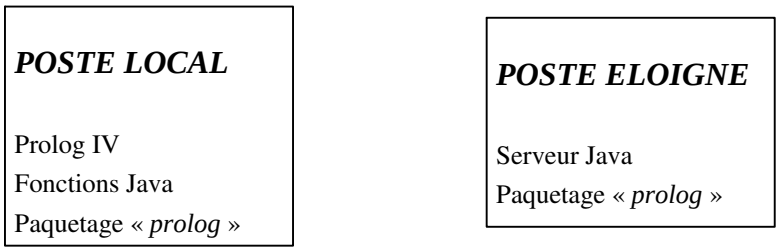


(2) Protocoles utilisables : http, ftp.

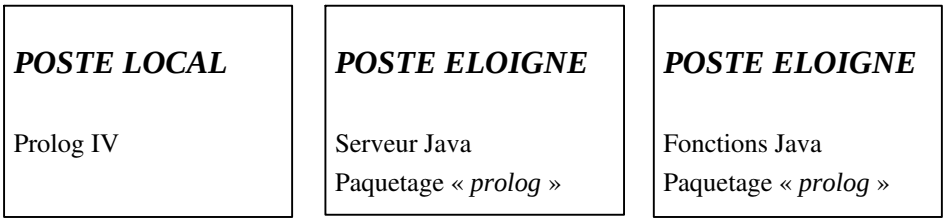


(3) Protocoles utilisables : file, http, ftp.

(4) Protocoles utilisables : http, ftp.



(5) Protocoles utilisables : http, ftp.



Exemple général :

Imaginons que vous ayez écrit une fonction JAVA déclarée comme suit :

```
public static P4Goal nom_de_la_fct(P4Goal pred_courant)
```

Si cette fonction fait partie d'une classe nommée « classe_java » à l'adresse « http://nom_de_l_hote/chemin/ » alors vous pourrez créer le prédicat Prolog correspondant à cette fonction comme suit :

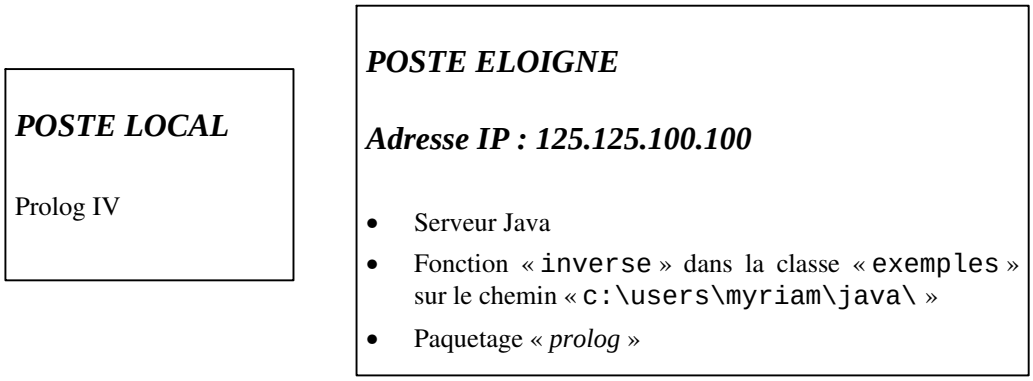
```
>> init_server(«emplacement_du_serveur»).
true.

>> new_java_pred("«nom_du_pred»", arite, "«nom_de_la_fct»",
                 "«classe_java»",
                 "«http://nom_de_l_hote/chemin/»").
true.

>> nom_du_pred(...).
...
```

Exemple précis :

Imaginons que nous soyons dans l'architecture suivante :



Voilà comment procéder pour créer un prédicat Prolog IV qui utilise la fonction JAVA `inverse` pour inverser une liste :

```
>> init_server("125.125.100.100")
true.
>> new_java_pred("inv_liste", 2, "inverse", "exemples",
                 "file:/c:/users/myriam/java/").
true.
>> inv_liste([1,2,3,4],A).
A = [4,3,2,1].
```

Le choix du protocole :

Il existe trois protocoles utilisables :

http (HyperText Transfer Protocol)

ftp (File Transfer Protocol)

file : système de fichiers local

Le plus rapide est le protocole « *file* » mais il ne peut être utilisé que lorsque le serveur JAVA et les fonctions JAVA appelées se trouvent sur le même poste.

Dans tous les autres cas, seuls les deux autres protocoles fonctionnent. On préférera néanmoins le protocole « *http* » qui semble être plus rapide.

Traitement des erreurs dans les fonctions JAVA appelées :

Lors de la création d'une fonction Java destinée à être utilisée comme prédicat Prolog, il est préférable de traiter correctement toutes sortes d'erreurs de façon à ce que Prolog sache lorsqu'une erreur est survenue et fasse échouer le prédicat.

La classe `PrologGoal` dispose d'un champ permettant de ranger un numéro d'erreur. Ce champ peut-être visualisé grâce à la méthode `getError()` et il peut être modifier à l'aide de la méthode `setError()`. Si le numéro vaut zéro, Prolog considère qu'il n'y a pas d'erreur et dans le cas contraire il échoue.

L'exemple ci-dessous décrit brièvement la manière de procéder :

```
public static P4Goal nom_de_la_fonction(P4Goal predicat_courant)
  {
    …
    try
    {
       …
      return le_predicat;
    }
    catch (une_premiere_exception e)
    {
       …
      le_predicat.setError(MA_PREMIERE_ERREUR);
      return le_predicat;
    }
    catch (une_seconde_exception e)
    {
       …
      le_predicat.setError(MA_SECONDE_ERREUR);
      return le_predicat;
    }
    catch …
    …
    catch (Exception e)
    {
       …
      le_predicat.setError(MON_ERREUR_INCONNUE);
      return le_predicat;
    }
  }
```

11.1.3 java_pred_list/0

Ce prédicat affiche la liste de tous les prédicats JAVA enregistrés dans Prolog IV. Ce prédicat réussit toujours.

Exemples :

Si aucun prédicat n'est enregistré :

```
>> java_pred_list.

true.
```

Si des prédicats JAVA sont déjà enregistrés, toutes les informations les concernant sont affichées.

```
>> java_pred_list.

------------------------------------------------------------------
Nom du predicat         :  permuter_arg
Arite du predicat       :  2
Fonction JAVA associee  :  permute
Classe JAVA associee    :  qSort
URL de la classe JAVA   :  file:/export/home/caba/transfert/fonct/
------------------------------------------------------------------

------------------------------------------------------------------
Nom du predicat         :  inverser_liste
Arite du predicat       :  2
Fonction JAVA associee  :  inverse
Classe JAVA associee    :  qSort
URL de la classe JAVA   :  file:/export/home/caba/transfert/fonct/
------------------------------------------------------------------
true.
```

11.1.4 remove_all_java_pred/0

Ce prédicat permet d'effacer d'un seul coup tous les prédicats JAVA connus de Prolog IV.

Exemple :

```
>> java_pred_list.

------------------------------------------------------------------
Nom du predicat         :  permuter_arg
Arite du predicat       :  2
Fonction JAVA associee  :  permute
Classe JAVA associee    :  qSort
URL de la classe JAVA   :  file:/export/home/caba/transfert/fonct/
------------------------------------------------------------------

------------------------------------------------------------------
Nom du predicat         :  inverser_liste
Arite du predicat       :  2
Fonction JAVA associee  :  inverse
Classe JAVA associee    :  qSort
URL de la classe JAVA   :  file:/export/home/caba/transfert/fonct/
------------------------------------------------------------------
true.

>> remove_all_java_pred.

true.

>> java_pred_list.

true.
```

Dans cet exemple, deux prédicats JAVA sont déjà enregistrés (`permuter_arg` et `inverser_liste`). Une fois le prédicat `remove_all_java_pred` utilisé, plus aucun prédicat n'est enregistré. On peut alors définir de nouveaux prédicats portant le même nom que les prédicats qui viennent d'être effacés.

11.1.5 remove_java_pred/1

Ce prédicat permet d'effacer un prédicat donné de la liste des prédicats JAVA connus de Prolog IV.

Exemple :

```
 >> java_pred_list.

------------------------------------------------------------------------
Nom du predicat         :  permuter_arg
Arite du predicat       :  2
Fonction JAVA associee  :  permute
Classe JAVA associee    :  qSort
URL de la classe JAVA   :  file:/export/home/caba/transfert/fonct/
------------------------------------------------------------------------

------------------------------------------------------------------------
Nom du predicat         :  inverser_liste
Arite du predicat       :  2
Fonction JAVA associee  :  inverse
Classe JAVA associee    :  qSort
URL de la classe JAVA   :  file:/export/home/caba/transfert/fonct/
------------------------------------------------------------------------
true.

>> remove_java_pred(«inverser_liste»).

true.

>> java_pred_list.

------------------------------------------------------------------------
Nom du predicat         :  permuter_arg
Arite du predicat       :  2
Fonction JAVA associee  :  permute
Classe JAVA associee    :  qSort
URL de la classe JAVA   :  file:/export/home/caba/transfert/fonct/
------------------------------------------------------------------------
true.
```

Dans cet exemple, deux prédicats sont enregistrés au départ (`permuter_arg` et `inverser_liste`). Ensuite, le prédicat `inverser_liste` est effacé grâce au prédicat `remove_java_pred`, et il ne reste donc plus que le prédicat JAVA `permuter_arg`.

11.2 INFORMATIONS DIVERSES

11.2.1 Qu'est ce qu'une URL ?

Une URL (Uniform Resource Locator) donne l'information permettant de localiser une ressource sur l'Internet. En général, une URL est composée de trois parties :

- le protocole utilisé,
- la description de l'hôte,
- le chemin de la ressource chez l'hôte.

11.2.1.1 Le protocole utilisé pour accéder à l'information.

Le protocole est toujours suivi de « : ». Les protocoles les plus connus sont ftp (File Transfer Protocol), http (HyperText Transfer Protocol). Si le protocole est file, il indique que la ressource se trouve sur le système de fichiers local et pas sur l'Internet.

Exemples :

ftp://ftp-serveur.domaine.fr/pub/

http://www.domaine.fr/divers/

file:/home/public/

11.2.1.2 Information sur l'hôte.

Cette partie donne l'information nécessaire pour trouver l'hôte sur lequel se trouve la ressource. L'information sur l'hôte commence par un double slash (« // ») si on a un protocole Internet comme ftp ou http, et commence par un simple slash (« / ») si on a un protocole de type file. L'information sur l'hôte se termine par un simple slash (« / ») quel que soit le protocole utilisé. L'information sur l'hôte est elle-même divisée en trois parties :

le *nom de domaine* de l'hôte si la ressource réside sur l'Internet (dans le cas d'une ressource locale, il n'y a pas de nom de domaine),

Exemple :

http://prologianet.univ-mrs.fr/

le *nom d'utilisateur* et le *mot de passe* si nécessaire,

Exemple :

http://prologianet.univ-mrs.fr.«myriam».«myriam00»/langage/fr/

le *numéro de port* si nécessaire (qui suit le nom d'hôte et un « : »).

Exemple :

http://prologianet.univ-mrs.fr:80/langage/fr/

11.2.1.3 Le chemin de la ressource (« path »).

Suite de répertoires qui doivent être séparés par des slashs (« / »).

Exemples :

http://prologianet.univ-mrs.fr/pub/prolog4/

file:/c:/users/myriam/

·♦·♦·♦·♦·♦·♦·