Étude de l’interopérabilité de deux langages de programmation basée sur la machine virtuelle de Java.

Mémoire présenté par Frédéric Minne (FSA3DS/IN)

Promoteur : Baudouin Le Charlier

Lecteurs : Gustavo Ospina, Kim Mens

Année académique 2002-2003

Table des matières

Introduction

→ Introduction

- Interopérabilité de deux langages de programmation
- Mécanismes d'interopérabilité
- Un mécanisme d'interopérabilité basé sur la JVM

- Machines virtuelles et conversion de données
- Mécanisme d'interopérabilité
- Applications
- Conclusion

Interopérabilité de deux langages de programmation

- Interopérabilité de deux langages de programmation

$\longrightarrow$ capacité d'un programme écrit dans un langage de programmation X d'utiliser un programme écrit dans un langage de programmation Y différent de X.

- Java et Prolog
  - Prolog

    $\longrightarrow$ IA, Systèmes experts, langue naturelle...
  - Java

    $\longrightarrow$ GUI, réseaux, services web, applets...

Mécanismes d’interopérabilité

- Mécanismes basés sur la JVM
  - Méthodes natives

    → JNI
  - Interpréteur écrit en Java
- Autres mécanismes
  - Redirection des I/O
  - Client-serveur
  - XML (XML, XML-RPC, SOAP...)
  - CORBA

Un mécanisme basé sur la machine virtuelle de Java

Utilisation d'un mécanisme basé sur la JVM et utilisant un compilateur Prolog vers JCode

→ Pourquoi ce choix ?

- Mécanisme basé sur la JVM :
  - Granularité plus petite
  - Portabilité
  - Mécanismes de la JVM (GC, sécurité...)
- Compilateur Prolog vers JCode :
  - Format des fichiers classes

  → Instructions de la JVM

  → Persistance

Machines virtuelles et conversion de données

- Introduction
- → Machines virtuelles et conversion de données
  - ○ Machines virtuelles et partage de la mémoire
  - ○ Conversion de données
- Mécanisme d'interopérabilité
- Applications
- Conclusion

Machine virtuelle et partage de la mémoire

Interaction Java et Prolog

- Interaction JVM et machine Prolog
- Partage de mémoire entre Java et Prolog
  - ○ Espaces mémoires séparés

  → cas typique d'un interpréteur natif
  - ○ Espace mémoire partagé

  → cas d'un interpréteur écrit en Java
- Partage des structures de données internes
  - ○ Structures de données partagées

  → compilateur Prolog vers JCode
  - ○ Structures de données non partagées

  → interpréteur = boîte noire (GNUProlog, tuProlog...)

Conversion de données (I)

Java est fortement typé

Vérification de types statique

$\longrightarrow$ compilation

Types de données en Java

- Types primitifs
  - Types numériques
    - Types entiers (byte, char, short, int, long)
    - Types virgule flottante (float, double)
  - booléens
- Classes, interfaces et objets

Conversion de données (II)

Prolog est un langage faiblement typé

$\longrightarrow$ termes

Vérification de types dynamique

$\longrightarrow$ exécution

Toutes les données = termes mais on distingue :

- Constantes
  - Atomes
  - Numériques
    - Entiers
    - Virgule flottante
- Variables logiques
- Structures (termes composés)
  - Listes

Conversion de données (II)

Représentation des termes Prolog pour interopérabilité

$\longrightarrow$ 2 solutions

- Utilisation de types/classes Java existants
  - $+$: plus simple, plus performant...
  - $-$: moins naturel, perte info sur les termes...
- Création de classes adhoc
  - $+$: préserve info sur les termes, héritage/polymorphisme, plus élégant...
  - $-$: développer de nouvelles classes, plus complexe...

Manipulation dans un programme Java

$\longrightarrow$ méthodes de conversion/extraction

Mécanisme d’interopérabilité

- Introduction
- Machines virtuelles et conversion de données

→ Mécanisme d’interopérabilité

  - Remarques générales
  - Mécanisme de conversion de données
  - Mécanisme de Java vers Prolog
  - Mécanisme de Prolog vers Java

- Applications
- Conclusion

Remarques générales

Propriétés

- transparent
- générique (neutre)
- non symétrique

3 parties

- conversion des données
- Java vers Prolog
- Prolog vers Java

Mécanisme de conversion de données

Principe

- Termes
  - ◇ Interfaces
  - ◇ Classes abstraites

  → ? instances
- Méthodes de conversion

→ Fabrique de termes

Java vers Prolog

Principe

- permettre l'appel à un programme Prolog depuis Java
  - ○ Interpréteur Prolog abstrait(générique)
  - → Cacher implémentation sous-jacente
  - ○ Classes annexes
    - ◇ Solutions
    - ◇ Clauses
    - ◇ Théories
  - → Interfaces

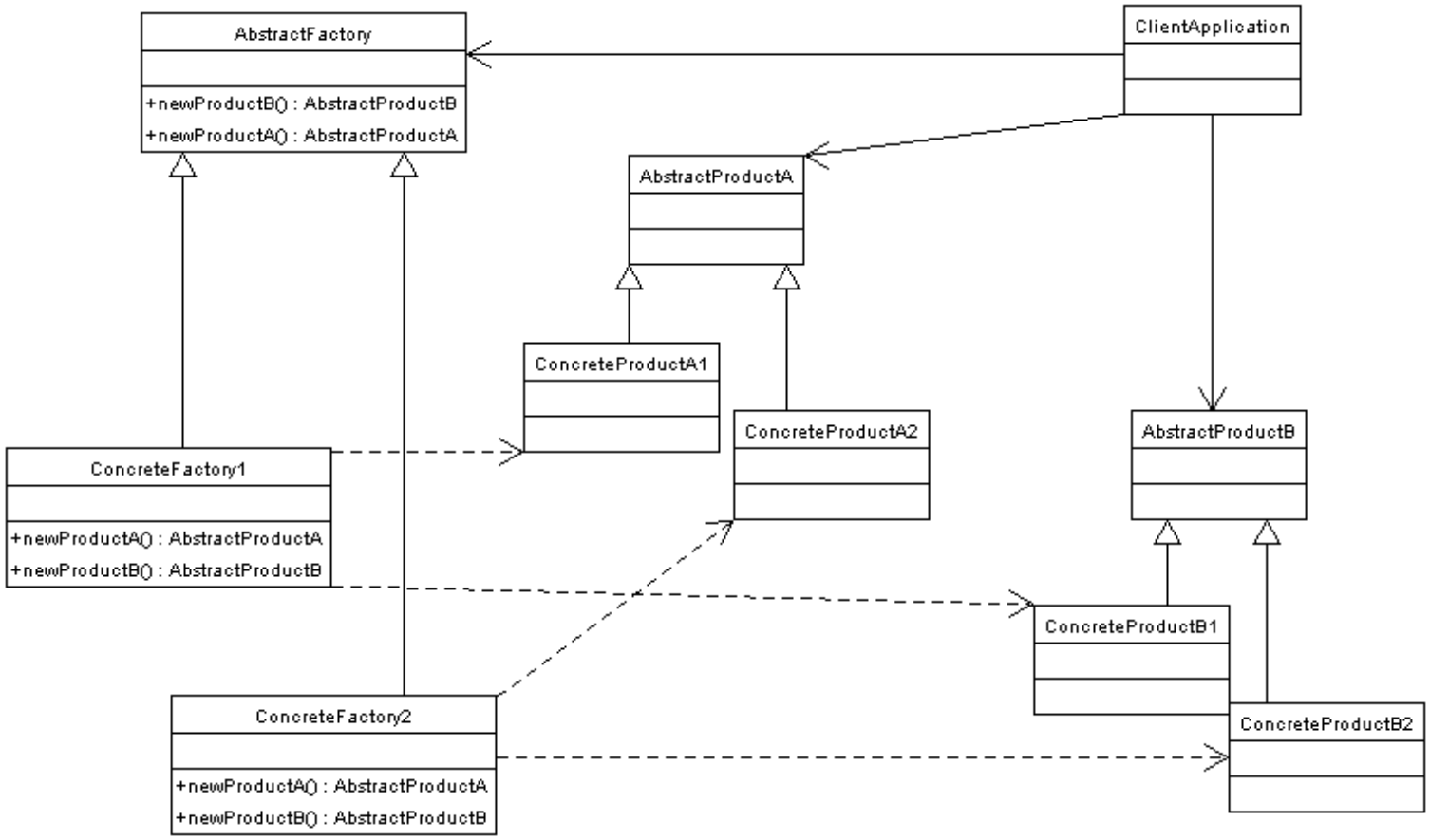
Java vers Prolog (II)

- Problèmes
  - ? instances
  - Générique(neutre)

→ Solution

  - Fabrique d’objets
  - Motif de fabrique abstraite

Java vers Prolog (III)



Applications

- Introduction
- Machines virtuelles et conversion de données
- Mécanisme d'interopérabilité

→ Applications

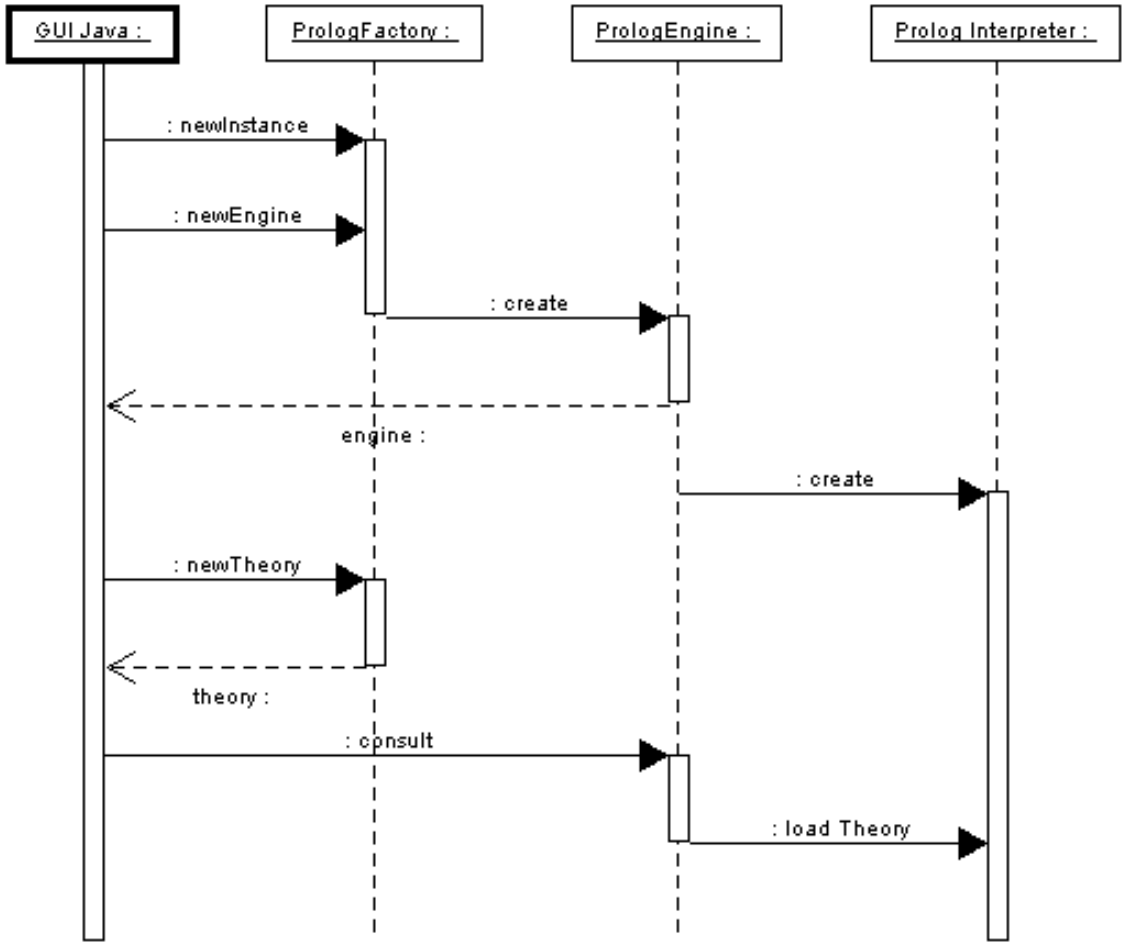
  - Scénarios
  - Choix du mécanisme

- Conclusion

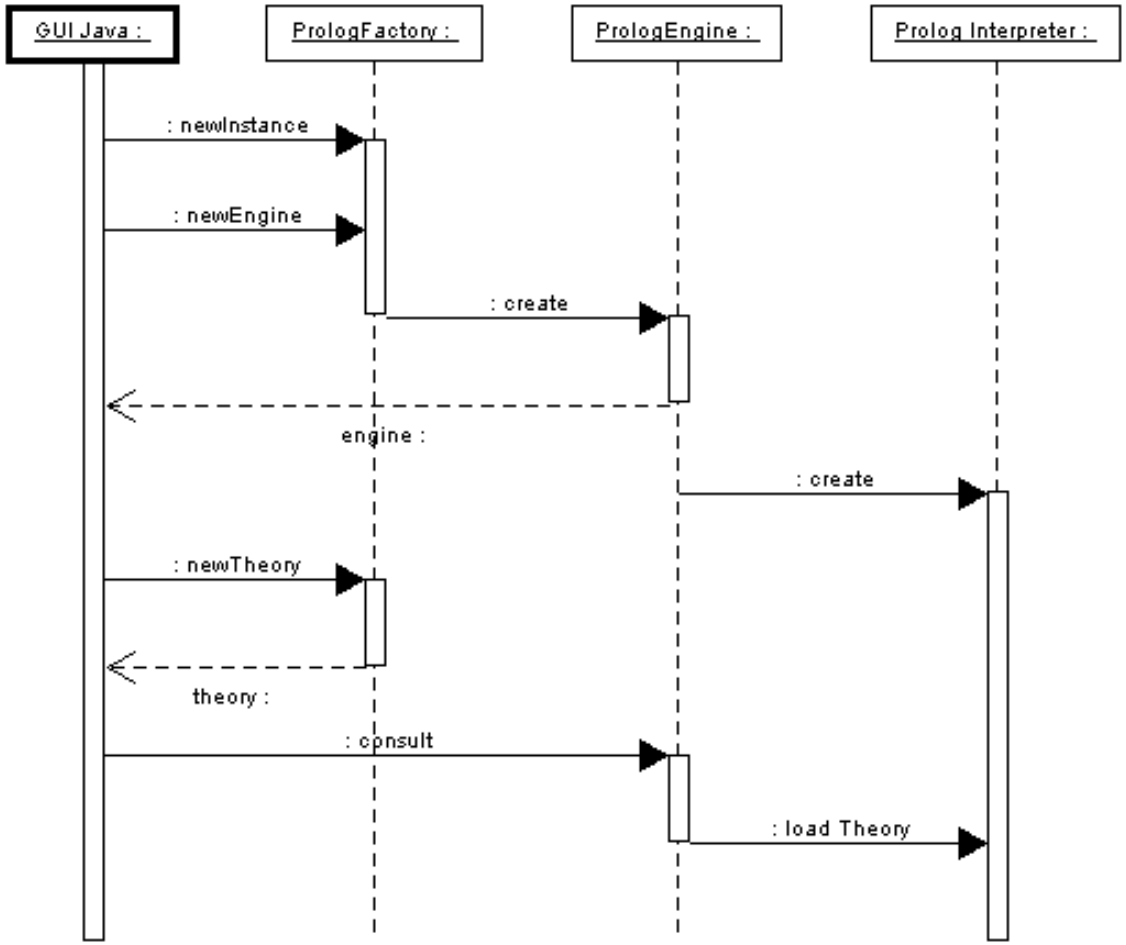
Scénarios

- 2 scénarios
  - ○ Java vers Prolog
  - → Interface graphique pour un programme Prolog
  - ○ Prolog vers Java
  - → Prolog et JDBC
- Un interpréteur Prolog

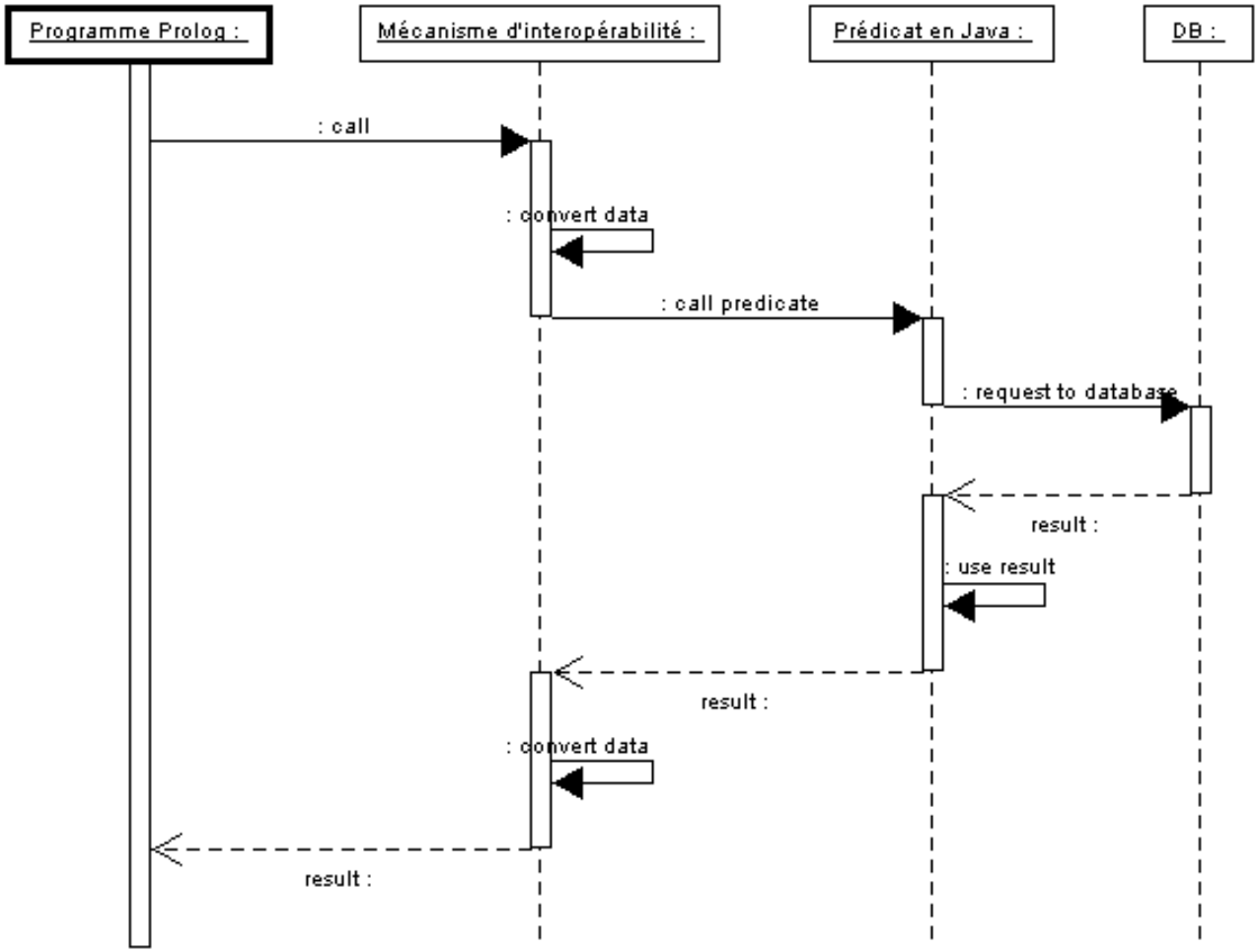
Scénario J2P (I)



Scénario J2P (II)



Scénario P2J

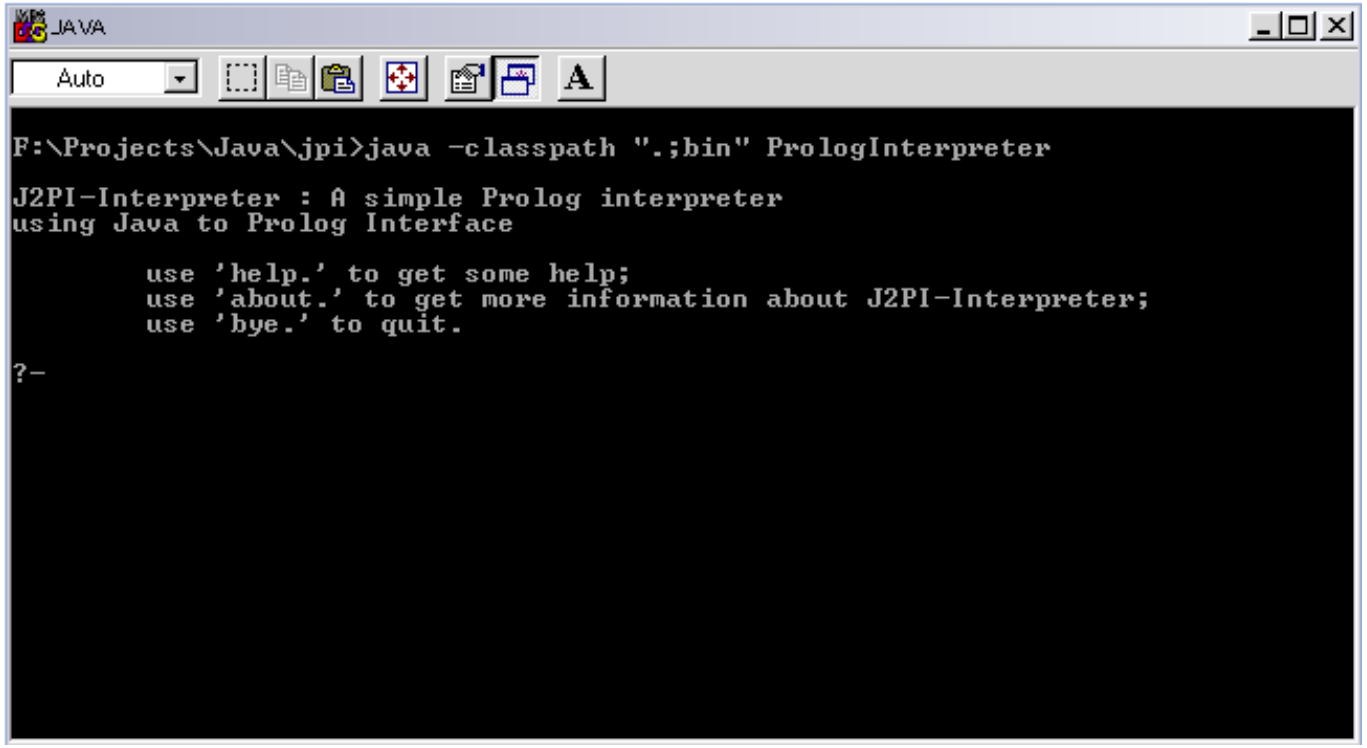


Scénarios

- Java vers Prolog
  - → Couche de présentation pour un programme Prolog
  - → IA dans un programme Java
- Prolog vers Java
  - → ajout de fonctionnalités au langage Prolog

Un interpréteur Prolog

Démo



Conclusion

- Introduction
- Machines virtuelles et conversion de données
- Mécanisme d'interopérabilité
- Applications

→ Conclusion

  - Conclusion
  - Perspectives

Conclusion

- Ce que j’ai fait
  - Mécanisme de conversion
  - Java vers Prolog
- ce qu’il reste à faire
    - Prolog vers Java
    - Corriger bugs du compilateur

Perspective

- améliorer Java vers Prolog
- améliorer Prolog vers Java
- vers un langage hybride