Jean Privat
privat.jean@uqam.ca
PK-4830

Projet NIT

Un langage à objet puissant, propre, intuitif, efficace et robuste.

Étienne Gagnon
gqgnon.etienne@uqam.ca
PK-4730

Spécification des langages : qu'est-ce qu'un langage proche du mode de pensée humain ?

Implémentation des langages : comment compiler efficacement de tels langages ?

Utilisabilité

Développement d'un produit utilisable dans la vrai vie par des vrais gens

- **KISS**
  Keep it simple stupid
- **Langage de script**
  Utilisable pour prototypes, scripts shell et programmes d'envergure
- **Efficace**
  Optimisation des temps de développement, de déploiment, d'exécution et de maintenance

Robustesse

Prévention des bogues

- Typage statique
- Plus de NullPointerException
- Isolation

2 modes d'exécution

- **Strict**
  Pour tous les jours
- **Encore plus strict**
  Pour les applications critiques
  (ex : temps réél)

Langages à objets

Bon paradigme

- Qualités d'ingénierie des logiciels
- Qualités intuitives de représentation

Le meilleur de la théorie actuelle

- Tout est objet
- Spécialisation multiple
- Raffinement de classes
- Types virtuels
- Vrai généricité

Syntaxe de NIT

En (très) gros

- Langage de script statiquement typé
- Plus simple et plus robuste que Java
- Plus rapide et plus expressif que C++

```
# Polite objects that say hello.
class Hello
  # Hello what?
  attr _title: String

  redef to_s do return "Hello {_title}"
end

var h = new Hello("World")
print h # Output "Hello World".
```

Plus d'info

http ://nitlanguage.org

Le compilateur nitc

Objectifs

- Pour la **recherche** sur les techniques d'implémentation
- Pour pouvoir utiliser NIT

Caractéristiques

- Autogène (écrit en NIT)
- Simple et modulaire → facile à comprendre, à modifier, à améliorer
- Permet facilement l'intégration de nouvelles extentions
- Permet facilement la comparaison de techniques d'implémentations

Sujets de recherche

La plupart de ces exemples de sujets de recherche peuvent se concevoir du côté de la spécification des langages, du côté de leur implémentation, voire des deux côtés à la fois.

Moteurs d'execution

- **Compilateur global**
  Plus d'analyses, plus de performances...
- **Compilateur séparé**
  Édition de lien dynamique
  (au chargement, au runtime)
- **Interpréteur**
  Dévelopement d'un interpréteur
- **Mixte**
  Utilisation les bibliothèques compilées par l'interpréteur

Isolation

Idée : Contraindre le graphe des objets en composantes connexes

- **Parallélisme**
  Manipulation concurente des composantes (sans mutex)
- **Exceptions transactionelles**
  Lors d'un `catch` l'état du système est celui d'avant le `try`
- **Temps réel**
  La gestion de la mémoire peut être finement contrôlée et garantie

Objets universels

Représentation des objets de type donnée pure (primitifs comme les entiers ou définis par l'utilisateur)

- Spécification dans le langage
- Implémentation efficacement

Interface native

Communication contrôlée C↔NIT

- Communication bidirectionelle
- Simplicité de l'interfaçage
  - → Intégration dans le langage NIT
  - → Extraction semi-automatique
- Contrôle de ce que C peut demander à NIT de faire
  - → Meilleure robustesse
  - → Meilleures optimisations

Autodocumentation et IDE

- Présentations des nouveaux concepts de NIT
- Extraction de documentation
- Intégration dans des IDE
- Navigation dans les programmes
- Outils d'aide à l'édition
  (refactorisation...)

Autres sujets de recherche

- **Programmation par contrat**
  Contrat et raffinement de classes font-ils bon ménage ?
- **Bibliothèques standards**
  Quoi écrire pour profitier des caractéristiques originales de NIT ?
- **Généricité et types virtuels**
  Comment les implémenter efficacement ?
- **Spécialisation de classes génériques**
  Comment faire pour que `Array[String]` sache faire plus de chose que `Array[Object]` ?