Université Montpellier-II
UFR des Sciences - Département Informatique
Master Informatique - UE HAI931I

Méta-Programmation et Réflexivité.

PROJETS

Projet en deux étapes.

1 Etape 1

1.1 Modalités et rendu

L'étape 1 sera réalisée par votre travail individuel avec mon soutien en TP.

A rendre pour l'étape 1 : Travail individuel. A envoyer par courriel, avant les congés de noël, à l'adresse **christophe.dony@umontpellier.fr** un document joint nommé impérativement **HAI931I-votreNom-Etape1.pdf** contenant un pdf (fait avec le traitement de texte de votre choix) les codes que vous avez écrit pour traiter l'exercice Etape 1 avec pour chaque code un petit commentaire personnel montrant votre vocabulaire et votre compréhension des choses.

1.2 Sujet pour l'étape 1

Réaliser l'exercice de construction d'un noyau objet réflexif, selon le programme guidé présenté au chapitre 2 du livre "A simple reflective Object Kernel" (`http://books.pharo.org/booklet-ReflectiveCore`), également accessible ici. Répondre à l'ensemble des questions posées.

2 Etape 2

2.1 Modalités et rendu

Etape 2 : à choisir une projet dans une des deux catégories "Langages" ou "Articles". Chaque article traite d'une problématique liée à la méta-programmation ou réflexion.

Faire un Compte-Rendu de lecture ou d'étude reflétant : - votre compréhension du langage étudié (du point de vue méta-programmation et réflexivité) ou de l'article lu, - et/ou toute idée que la lecture vous a inspiré, - et/ou toute implantation dans le langage de votre choix que la lecture vous a donné envie de réaliser (et que vous avez réalisée).

Si les articles sont des thèses, l'important est de comprendre l'introduction, l'idée générale et une des contributions ; il n'est pas demandé (mais pas interdit) de tout lire.

Choix : Choisir un des langages L_1 à L_n ou un des articles numérotés. Venez me voir pendant le TP pour m'indiquer votre choix que j'inscrirai dans la base, un choix par étudiant ou par groupe de deux étudiants, premier arrivé, premier servi.

Rendu : A rendre pour l'étape 2 par courriel à **christophe.dony@umontpellier.fr** pour le 8 janvier 2024 avant minuit une archive zip nommée **HAI931I-votreNomOuVosDeuxNoms-Etape2.zip** contenant a minima un texte personnel de 2 pages minimum (pas de limite maximum) reflétant votre analyse du sujet, non généré par une IA, contenant aussi possiblement des exemples sous forme de codes commentés. Placez également dans l'archive les outils (compilateur, machine virtuelle, ...) utilisés pour faire tourner les programmes ou si impossible indiquez le lien pour obtenir le langage ou l'environnement utilisé pour les tests. Si vous choisissez d'utiliser une IA pour vos explications, indiquez le puis définissez

avec vos propres mots chacun des termes qui y sont utilisés et ce que vous subcomprenez de ce texte généré. De plus si le texte généré contient des erreurs, des non-sens ou des contre-sens, il est nécessaire que vous les indiquiez.

2.2 Choix des sujets pour l'étape 2

2.2.1 Sujet Orientation "Langages"

Common-Lisp-Object-System Reprendre les exercices des TDs/TPs dans le langage (capacitaire) de votre choix (L1 Javascript, L2 Python, L3 Objective-C, L4 Php, L5 Ruby, L6 C# and .net, L7 autre de votre choix non déjà pris). Voir également http://newspeaklanguage.org/.

2.2.2 Sujet Orientation "Articles"

Il s'agit ici :

- soit de lire un article ou un document et d'en produire un résumé intéressant de deux pages minimum décrivant une idée clé que vous avez comprise, illustrée par un exemple,
- soit d'essayer un système ou un langage et d'en produire un résumé descriptif avec un exemple de code.

Voici ci-dessous des sujets ouverts à l'étude.

Articles généraux

1. ADAPTIVENESS OF SOFTWARE SYSTEMS USING REFLECTION Jan KOLLAR, Michal FORGAC, Jaroslav PORUBAN. article à lire : notes-etudes/Kollar.pdf et étendre aux articles cités.

Reflexion au moment du chargement ("load-time") en Java

2. Shigeru Chiba, "Load-time Structural Reflection in Java," ECOOP 2000 Object-Oriented Programming, LNCS 1850, Springer Verlag, page 313-336, 2000.
   https://en.wikipedia.org/wiki/Javassist
   http://www.javassist.org/
   article à lire : notes-etudes/Javassist.pdf

Les intergiciels (middleware) réflexifs

Les intergiciels sont les logiciels qui font la liaison entre les clients et les serveurs dans les architectures distribuées. Pourquoi ne seraient-ils pas réflexifs.

3. Donmaine : Adaptive and Reflexive Middleware.
   Thomas Ledoux, "Reconfiguration dynamique d'architectures logicielles : des métaclasses aux "nuages verts"", https://tel.archives-ouvertes.fr/tel-01864344.
4. La réflexivité dans les architectures multi-niveaux : application aux systèmes tolérant les fautes.
   Thèse de François Taïani.

Réflexion comportementale

Comment donner au méta-programmeur le contrôle sur l'exécution des méthodes, donc sur les comportements des objets.

5. Le système *Reflectivity* de *Pharo* permet d'associer un méta-objet à toute instruction d'une méthode.
   Sub-method, partial behavioral reflection with Reflectivity, Looking back on 10 years of use.
6. Modélisation formelle de la réflexion de comportement dans le cadre d'un langage à prototypes, sans classes.
   A Semantic of Introspection in a Prototype-Based Language.

Approfondir la question de la composition et de la compatibilité des méta-classes

7. Une proposition pour mieux contrôler les métaclasses explicites, *Safe Metaclass Programming*
   notes-etudes/Bour98a-SafeMetaclassProg.pdf

Système réflexifs pour la construction d'outils de mise au point des programmes

8. Comment réaliser des logiciels que l'on peut déboguer en Pharo sans interrompre leur exécution, (Dynamic Software Update).
   Voir l'article : Reflection as a Tool to Debug Objects et aussi la thèse
   notes-etudes/these-S-Costiou.pdf
9. Comment faire en Pharo du pas-à-pas dans l'exécution d'un programme, en avant (vers le futur) ou en arrière (vers le passé).
   notes-etudes/thesis-maximilian-willembrinck-2023.pdf

Variations sur l'adaptation dynamique pour les non spécialistes des systèmes réflexifs

Comment réaliser l'adaptation dynamique (ou transformation de programmes ou de modèles à l'exécution) sans utiliser et sans être un spécialiste des langages réflexifs. Différentes études traitent de cette question.

10. The Vision of Autonomic Computing (create self-managing computing systems - MAPE-K architecture).
    notes-etudes/autonomic-computing.pdf.

    Modeling and Analyzing MAPE-K Feedback Loops for Self-adaptation.
    notes-etudes/Make-K-Loop.pdf.
11. Software Engineering for Self-Adaptive Systems : A Second Research Roadmap.
    notes-etudes/sefsas2.pdf
12. Software Engineering of Self-Adaptive Systems : An Organised Tour and Future Challenges.
    notes-etudes/2017-SE-SAS.pdf.
13. Reflecting on Self-Adaptive Software Systems.
    notes-etudes/Reflecting-Self-Adaptive-Software-Systems.pdf.

Miroirs versus Méta-Objets

Les méta-objets sont des objets représentant des entités du méta-niveau utilisables au niveau de base. Certains travaux en ont proposé une version différente de la version standard et les ont nommé miroirs. Que font-ils de plus ou de différent ?

14. Mirrors : Design Principles for Meta-level Facilities of Object-Oriented Programming Languages. Gilad Bracha, David Ungar.
    notes-etudes/mirrors.pdf

Langages et Interpréteurs réflexif, tours réflexive

15. A Component-based meta-level architecture and prototypical implementation of a Reflective Component-based Programming and Modeling language.
16. Un interpréteur méta-circulaire capable d'interpréter l'intégralité de son propre code, amorçage compris.
    notes-etudes/ReflectiveInterpreter.pdf
    notes-etudes/mystery-tower-revealed.pdf
17. Un langage réflexif pour la conception et la vérification de systèmes électroniques (hardware).
    notes-etudes/reflective_functional_language_for_hardware_design_and_theorem_proving.pdf

Explication de la démonstration du théorême de Gödel

18. Le théorème de Gödel montre que tout système formel contenant a minima l'arithmétique de Peano autorise l'expression d'énoncés indémontrables. Sa démonstration par l'absurde est réalisée en encodant dans un tel système

formel une expression contenant une contradiction. Cet encodage nécessite un plongement du méta-niveau dans le niveau de base et cela relève donc conceptuellement de ce cours.

Le site http://www.felderbooks.com/papers/godel.html propose une explication synthétique et claire de cette démonstration, reprenant l'explication fournie par Douglas Hofstadter dans son livre "Gödel, Escher, Bach : les brins d'une guirlande éternelle

Etc, etc

Si un autre article non listé lié à la méta-programmation et réflexivité vous intéresse, faites moi une suggestion. Vous pouvez notamment choisir des articles qui vous font poursuivre l'étude de *Smalltalk* et *Common-Lisp-Object-System* vus pendant les cours et TPs.

Post Scriptum

(a) Je tiendrai compte des éventuelles difficultés différentes des sujets dans mon analyse de vos retours.

(b) Pour mettre du code pharo dans du latex :

```
\documentclass[a4wide,10pt,frenchb]{article}
...
\usepackage{listings}
\lstdefinelanguage{Smalltalk}{
alsoletter={:},
  classoffset=0,
  morekeywords={class, subclass:, instanceVariableNames:, classVariableNames:, new, new:,
  ifTrue:, ifFalse:, to:, do:, value:, value},
keywordstyle=\myKeywordStyleA,
  classoffset=1,
morekeywords={self, super, true, false, nil, Transcript, Smalltalk},
  keywordstyle=\myKeywordStyleB,
 classoffset=0,
morecomment=[s]{"}{"},
morestring=[b]',
}
...
\begin{document}
...
\begin{lstlisting}[language=Smalltalk,label={},caption={Exemple d'attribut d'instance partagé}]
Object subclass: #Citoyen
instanceVariableNames: 'nom age adresse'
classVariableNames: 'president'
package: 'HAI931'
\end{lstlisting}
...
\end{document}
```