**Projet Final:**

**Remaniement Automatique pour l’Internet des Objets**

Michael Morin

Département d’Informatique et Mathématique, UQAC

8INF957: Programmation objet avancée

Dr Hamid Mcheick

Dec 19, 2023

**Remaniement Automatique pour l’Internet des Objets**

Alors qu’Internet bénéficie de protocoles de communication standardisés, l’Internet des Objets (Internet of Things - IoT) utilise une variété de protocoles pour communiquer dans ses réseaux. En fait, comme les premiers ordinateurs qui tentaient de communiquer ensemble, des Objets de la vie ordinaire sont maintenant dotés de puces qui leur permette de communiquer entre eux localement ou à travers Internet. Cependant, les protocoles de communication en IoT sont nombreux ce qui rend le développement de systèmes logiciels générique difficile. Au lieu de présenter un système spécifique à un contexte comme plusieurs autres, l’implémentation présentée ci-bas permet d’améliorer à la fois la réutilisabilité des systèmes existants, ou une partie d’entre eux, tout en assurant l’interopérabilité du système final grâce au remaniement automatique (automatic refactoring).

**Revue de la littérature**

L’idée d’un remaniement automatique en IoT provient de l’article de Zhang et al. (2020) dans lequel les auteurs présentent une telle implémentation. Avec un focus sur la concurrence des données (data race), les méthodes en Java pour réduire automatiquement les problèmes liés à cet attribut de qualité sont explorées. En comparant la meilleure solution disponible avec la plus récente méthode permettant d’assurer la fiabilité des données, Zhang et al. démontre un manque à gagner important. Ainsi, un framework est présenté afin de changer automatiquement les cadenas associés au donnés d'un programme. Une fois le nouveau framework implanté, les auteurs évaluent le nouveau système à l’aide de cinq logiciels de références pour déterminer l’efficacité de leur travail comparé aux originaux. Avec des résultats satisfaisants, la conclusion ne présente qu’une limitation quant à la généralisabilité de leur système logiciel.

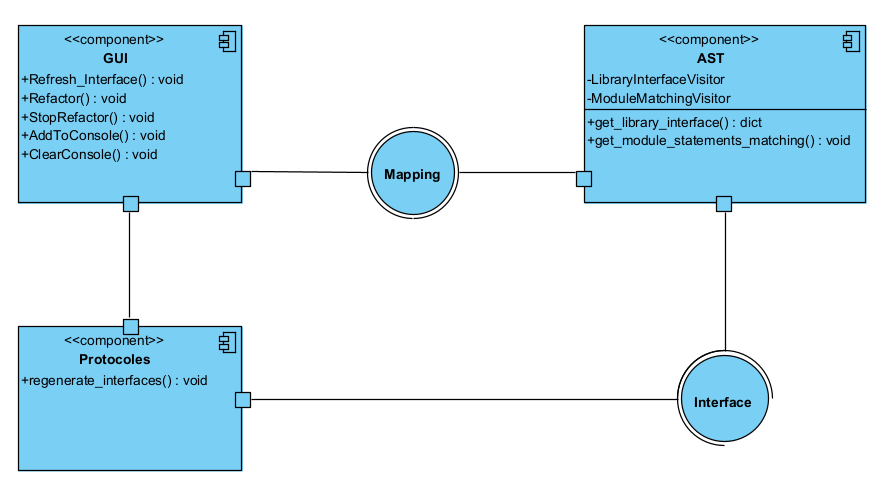
La revue de la littérature dans ce sujet a permis d’obtenir un autre article de Zhang et al. écrit en 2019 sur le remaniement en IoT. Selon eux, le remaniement a tendance à créer des problèmes de sécurité tels que des changements aux résultats finaux ou des problèmes de concurrence. Après avoir discuté de solutions existantes et de leurs limitations, les auteurs rappellent l’importance du remaniement tant qu’il maintient la sécurité des données et de l’application. Présentant leur processus de travail, les auteurs détaillent les trois étapes nécessaires pour assurer la sécurité dans le système refactorisé. Pour ce qui est de l’algorithme de détection des fautes, il adresse à la fois les ‘’deadlocks’’, les réutilisations d’objet ainsi que le partage des champs statiques. Évalué sur quatre logiciels repères, l’auteur présente un remaniement effectif à plus de 99%. Notant la limitation des tests, Zhang et al. propose d’étendre la capacité du programme au-delà de l’IoT.

**Méthode**

Le problème important relevé ici provient de la spécificité des solutions logicielles en IoT. Ainsi, un système a été conçu pour remanier la manière dont un logiciel communique avec les objets pour que le protocole de communication puisse être changé automatiquement. Au niveau des exigences non-fonctionnelles, un accent a été placé sur l’interopérabilité, la facilité d’utilisation et la réutilisabilité. Pour bien mesurer ces exigences, le temps d’exécution, les modifications nécessaires pour ajouter un nouveau protocole ainsi que la simplicité de l’interface utilisateur seront évalués après la description de l'implémentation. Au niveau des exigences fonctionnelles, elles sont énumérées ici-bas:

* Ajouter des protocoles réseaux facilement
* Remanier rapidement les solutions logicielles
* Assurer la sécurité et l’utilisabilité du logiciel remanié
* Supporter plus d’un langage de programmation
* Être efficace sur les petits et les gros projets
* Déployer sur des systèmes monolithe ou distribués
* Maintenir les comportements logiciels initiaux

Pour satisfaire ces exigences, la solution utilisera le diagramme suivant. Dans ce dernier, on peut voir que, pour obtenir les protocoles disponibles, le module GUI fait appel au module Protocoles qui va chercher et retourne les interfaces des protocoles. Si l’interface n’existe pas, elle est créée par le module AST à la demande de Protocoles. Pour ce qui est du remaniement, le GUI fournit un dictionnaire cartographié à AST afin de rendre le remaniement automatique.



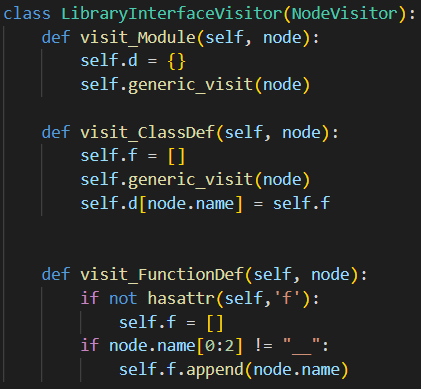
Dans ce projet, deux patrons seront principalement utilisés: le rappel et le visiteur. Ce qui est appelé rappel ici est une instance du patron observateur. Dans ce dernier, un objet a le rôle d'abonné alors qu’un autre a le rôle de publieur. En appelant une fonction qui permet de s'abonner chez le publieur, l’abonné transmet le type d'événement à lequel il souscrit ainsi qu’un pointeur vers lui-même comme arguments. Ainsi, lorsque le publieur atteint un certain événement, il appelle une fonction prédéfinie sur le pointeur donnée. Dans notre cas de rappel, un pointeur vers une fonction est passé en argument au lieu d’un pointeur vers un objet et le publieur décide lui-même quand la fonction est appelée.

Pour ce qui est du visiteur, il possède une liste de fonctions de visite pour différents éléments qui agit différemment selon le type d’élément. De son côté, l’élément a une méthode permettant au visiteur de venir visiter. Puisque le visiteur et l’élément sont interdépendants, ce sont deux interfaces ayant chacune une ou plusieurs implémentations. Une fois le visiteur et l’élément instancier, l’élément utilise sa fonction permettant la visite en prenant le visiteur comme argument. À l’intérieur de cette fonction, l’élément fait alors appel à la fonction du visiteur supportant son type. Ainsi, le visiteur implémente une logique derrière la visite sans que l’élément n'ait à changer quoi que ce soit. Dans notre cas, le visiteur permet de remanier automatiquement les codes de la librairie source vers l’équivalent en librairie ciblée.

**Résultat**

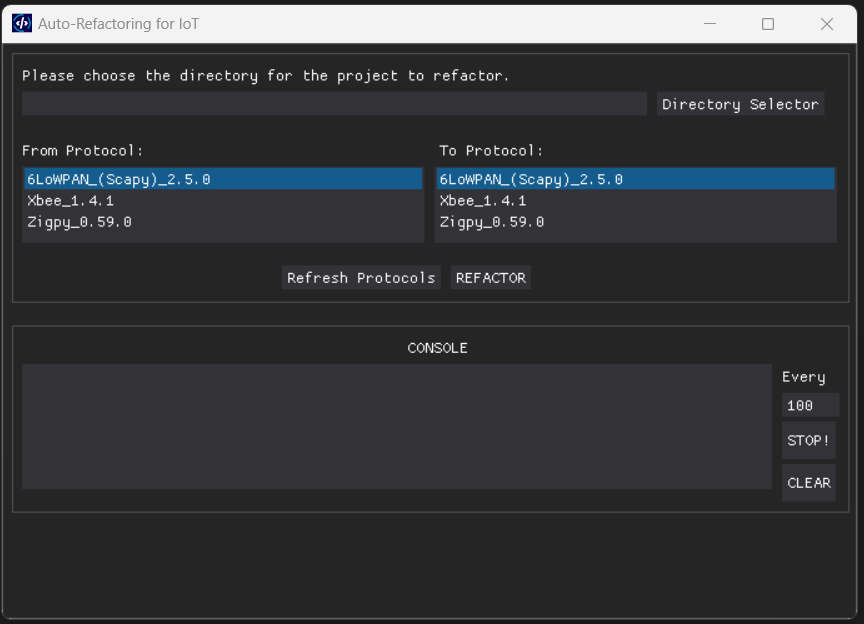
L'implémentation de la solution fut plus complexe que prévu, mais le développement a été complété. Au final, trois modules ont été conçus: Protocoles, AST et GUI. Dans le premier, les fonctions de génération d’interface et de récupération de ces dernières ont vu une implémentation. Pour la génération, le module cherche le dossier .\Protocols\Libraries et il itère sur chaque dossier trouvé pour vérifier si le dossier .\Protocols\Interfaces contient une interface sous la forme YAML. Dans le cas où aucune interface n’existe pour une librairie trouvée, un arbre AST est généré pour obtenir l’interface.



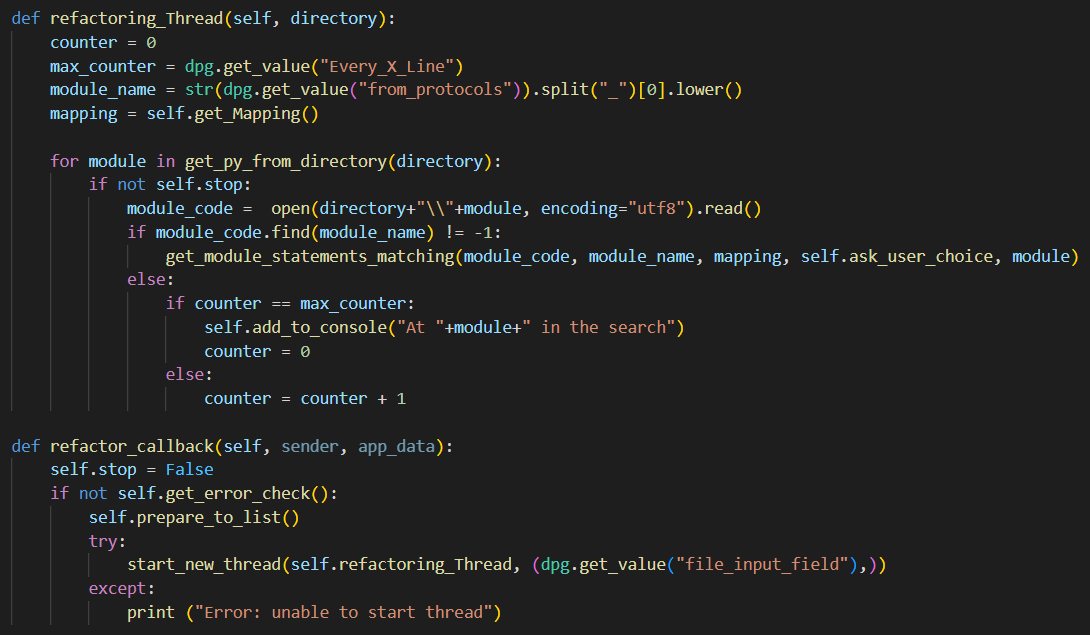
Le module AST (ast\_custom.py), quant à lui, se situe dans le dossier util utilise la librairie ast de Python. Dans le module précédent, on peut voir que la fonction pour créer l’interface d’une librairie utilise le code d’un module pour créer un arbre syntaxique abstrait à l’aide de la fonction ‘parse’ de la librairie native. Ensuite, un visiteur est créé en étendant le visiteur natif afin de définir un attribut ‘d’ comportant un dictionnaire de l’interface. Pour se faire, le visiteur ‘LibraryInterfaceVisitor’ utilise des fonctions de visite afin d’extraire les noms de classes et de fonctions définies dans le module.

Une autre fonction importante définie dans ce module est celle permettant le remaniement. Avec un fonctionnement similaire à la fonction précédente, les seuls ajouts sont l'utilisation de la fonction ‘unparse’ native afin d’obtenir le code de l’arbre remanié et l’écriture du code sous forme de module dans le dossier ‘refactored’. Tout comme la création d’interface avait sa propre implémentation d’un visiteur, cette fonction fait appel au visiteur ‘ModuleMatchingVisitor’ qui visite principalement les appels de fonction et les importations de librairies. Dans chaque visite, le nom de la fonction ou de la librairie est comparée au module source identifié par l’utilisateur afin de soit remanier automatiquement, soit de demander à l’utilisateur de lier l’objet du protocole source à l’objet du protocole ciblé. Finalement, pour effectuer ses tâches, le visiteur est initialisé avec un dictionnaire cartographiant les objets sources aux objets ciblés ainsi qu’avec un objet de rappel (callback) pour faire les demandes à l’utilisateur.

Pour la première fois, l’interface utilisateur que j’ai créé est sous forme graphique contenue dans le module GUI. À l'aide d’une fonction ‘setup’, le programme crée l’interface graphique selon les besoins de la librairie DearGUIPy. Afin de dessiner les composantes de gestion du remaniement, de console et d’ajout d’un thème, des fonctions emboîtées sont appelées pour chaque composante. L’image suivante permet de bien comprendre ce que les composants représentent:



Avec la librairie GUI choisie, chaque bouton fait appel à une fonction de rappel implémenté par le développeur afin d'exécuter une action lorsque le bouton est appuyé. Pour le ‘Directory Selector’, la fonction permet de récupérer le chemin absolu choisi afin de la placer dans le champ de texte à sa gauche. Pour ‘Refresh Protocols’, la fonction de régénération d’interface discutée plus tôt est utilisée et une nouvelle liste de protocoles est présentée dans les champs ‘From Protocol’ et To Protocol’. Pour la section CONSOLE, le conteur à droite permet d’ajuster le nombre de modules itérés avant qu’une ligne soit ajoutée sur la console. Pour les boutons d'arrêt et de nettoyage, ils permettent respectivement d’arrêter la refactorisation et de vider le champ de texte à gauche. Finalement, le bouton ‘Refactor’ prépare le dictionnaire cartographiant les objets sources aux objets ciblés avant de passer de créer un nouveau fil d'exécution (thread) qui va effectuer le remaniement. Ce nouveau fil d'exécution permet de s’assurer que les boutons continue de fonctionner mais aussi de créer l’AST pour chaque module, de visiter les arbres ainsi que d’afficher périodiquement l’état du remaniement à la console.



**Discussion**

L'implémentation a permis d’explorer la structure de données qu’est l’arbre syntaxique abstrait, mais aussi les patrons de rappel et de visiteur. En fait, l’ajout de l’interface graphique a propulsé l’étude du patron de rappel beaucoup plus loins que prévu puisque son utilisation est omniprésente dans la librairie. Au niveau du module ‘ast\_custom’ le patron de rappel a permis de lier une fonction en GUI au visiteur ‘ModuleMatchingVisitor’. Ainsi, après avoir défini une fonction affichant une demande à l’utilisateur en GUI, la fonction fût passée en argument au constructeur du visiteur ce qui a permis au visiteur de transférer le nom de l’objet source au GUI pour la demande. Au niveau du visiteur, il a permis d’implémenter un algorithme de remaniement selon le nœud (node) rencontré dans l’objet sans avoir à connaître la structure interne de l’arbre.

Au-delà de l'implémentation, les attributs de qualités d’interopérabilité, de facilité d’utilisation et de réutilisabilité avaient été ciblés. Pour ce qui est du premier attribut, le but était d’assurer l’échange de données dans un environnement hétérogène. Pour ce faire, les librairies en Python n’ont qu’à être placées dans le dossier pour pouvoir être utilisées. Ainsi, n’importe quelle librairie pourrait être utilisée. De plus, lorsque la librairie source est cartographiée à la cible, le logiciel peut être utilisé même si l’utilisateur n'a aucune connaissance des librairies. Au niveau du programme source, la taille ou la structure du projet importe peu puisque le programme demande à l’utilisateur de choisir le dossier source avant de le parcourir de façon récursive. De ces faits, la seule limite au niveau de l’interopérabilité est que le programme et les librairies soient en Python. Cependant, une deuxième itération pourrait facilement apporter un meilleur support pour une diversité de langages de programmation.

Pour ce qui est de la facilité d’utilisation, l’interface en soit à été construite pour être simple et intuitive. L’ajout d’une librairie ne demande qu’un téléchargement et le projet est choisi facilement par l’utilisateur. D’un autre côté, si les librairies ne sont pas cartographiées, l’utilisateur doit avoir une bonne compréhension de chacune. Pour ce qui est de la réutilisation, malheureusement, n’a pas reçu l’attention nécessaire. Au niveau des modules, la division permet une réutilisation de ‘ast\_custom’, ‘glob\_custom’ et de ‘Procoles. Au niveau des classes et fonctions, le couplage est plutôt important ce qui prévient une bonne réutilisation du code. Dans une nouvelle itération, le code de la console recevrait son propre module avec une fonction pour le GUI en plus des fonctions de rappel. De plus, l’arbre des interfaces ainsi que les fonctions de rappel liées à ses fonctionnalités serait transféré dans le module Protocoles pour une meilleure réutilisabilité.

**Conclusion**

Avec un article de remaniement automatique des cadenas en Java, l’idée d’une application similaire permettant le changement d’un protocole à l’autre a vue une implémentation. Au niveau de l’architecture, l’emphase fût placée sur les patrons de rappel et de visiteur et les attributs de qualité d’interopérabilité, de facilité d’utilisation et de réutilisabilité ont été sélectionnés. À l’aide de trois modules distincts, la solution logicielle fût implémentée mais, au final, l’attention donnée aux attributs fût décroissante respectivement. Bien que des suggestions pour une implémentation future ont été exposées, pousser la recherche au niveau des tests et de la sécurité suivant le remaniement automatique permettrait d’assurer qu’une future implémentation soit plus robuste au niveau de sont utilité.

**Références**

ast — Abstract Syntax Trees. (n.d.). Python Documentation. https://docs.python.org/3 /library/ast.htm

Boadzie, D. (2023, September 11). Introduction to Abstract Syntax trees in Python. Earthly Blog. https://earthly.dev/blog/python-ast/

Implementing a callback in Python - passing a callable reference to the current function. (n.d.). Stack Overflow. https://stackoverflow.com/questions/4689984/implementing -a-callback-in-python-passing-a-callable-reference-to-the-current

Refactoring.Guru. (n.d.). Design patterns. https://refactoring.guru/design-patterns/