**Projet Final:**

**Système Logiciel Auto-Adaptant dans l’Internet des Objets**

Michael Morin et Germain Besson

Département d’Informatique et Mathématique, UQAC

8INF853: Architecture des applications d'entreprise

Dr Hamid Mcheick

Dec 19, 2023

**Système Logiciel Auto-Adaptant dans l’Internet des Objets**

L’Internet des Objets (Internet of Things - IoT) est un paradigme de plus en plus important en informatique où des objets de la vie courante reçoivent une puce pour communiquer sur les réseaux. Ainsi, une machine à laver pourrait communiquer avec la sécheuse afin de déterminer si cette dernière est disponible pour la prochaine étape de la lessive, ou quand elle le sera pour notifier le propriétaire. Pour ce faire, chaque Objet utilise un protocole de communication, une batterie et une gamme de fonction qui varie d’Objet en Objet. Pour pallier à l’hétérogénéité des environnements dans de tels réseaux, les systèmes logiciels auto-adaptant (self-adapting software system - SASS) sont parfois utilisés pour permettre une adaptation du logiciel en temps réel pour optimiser l’utilisation des ressources. Bien que cette solution offre de nombreuses possibilités, dans un contexte générique en IoT, l’interopérabilité, la modifiabilité et l'utilisabilité sont des attributs de qualités importants à assurer. Après une courte revue de la littérature, un modèle architectural sera présenté avec les résultats d’une implémentation afin de discuter de comment notre proposition adresse ces attributs.

**Revue de la littérature**

Implémenter une architecture dans un contexte général de l’IoT requiert avoir une bonne compréhension des défis couramment observés. Ainsi, l’article de Sobin écrit en 2020 discutant d’architecture, des protocoles et des défis dans un tel contexte semble crucial pour bien situer le choix de notre solution et notre focus. Après avoir introduit les différentes difficultés en IoT, l’auteur présente la multitude de contextes tels que les maisons ou les villes intelligentes pour expliquer l’immense hétérogénéité des réseaux en IoT. De cet exposé, Sobin liste sept exigences génériques, dont cinq sont couvertes par notre projet: extensibilité, efficacité énergétique, gestion des données, interopérabilité et capacité d’auto-organisation. Après une revue des réseaux et taxonomies disponibles, l’auteur propose une revue importante des difficultés (extensibilité, efficacité énergétique, sécurité et interopérabilité) en plus des opportunités (réseau sociaux, protocoles et architectures) disponibles en IoT, mais notre sommaire ici ne va que s’attarder aux aspects liés au projet courant. Selon Sobin, les architecture proposés jusqu'à offrent plusieurs bénéfices mais l’évolution doit continuer. Il note, entre autres, la nécessité d’avoir une passerelle entre les Objets et le réseau pour traduire les protocoles en plus d’offrir une interface commune pour identifier et gérer les objets selon les besoins de l’utilisateur. Au niveau de l’extensibilité, une solution proposée consiste en une abstraction des données sur une couche additionnelle alors que les deux autres proposent plutôt une architecture orientée services ou l’utilisation du pattern publication/abonnement. Pour ce qui est de la consommation d’énergie, les solutions proposées ciblent souvent les protocoles de communications, mais certaines emploies des stratégies applicative basées sur la Théorie de Jeu (Game Theory). Finalement, l’auteur conclut sur l’importance de la sécurité, de l’extensibilité, de la mémoire ainsi que les considérations architecturales comme aspects à améliorer en IoT.

Plusieurs des points soulevés par Sobin (2020) sont couverts par les systèmes logiciels auto-adaptant (self-adapting software system - SASS) selon Ismail et al. (2021). Après avoir détaillé l’évolution des systèmes logiciels adaptants, les auteurs expliquent que l’adaptation trouvée dans ces systèmes est souvent une arrière pensée dans un logiciel standard mais que, considérant la difficulté de bien définir le contexte avant le développement, il est crucial de prendre en compte la variabilité interne et externe du système. En fait, lors qu’ils expliquent comment un SASS est construit, on peut imaginer avoir une couche supplémentaire à un logiciel standard: le logiciel de base est constitué du coeur du système mais les adaptations dû aux changements dans l'environnement sont abstrait dans un sous-système qui ajuste le système de base selon les besoins. Après une classification extensive des sources d’incertitudes dans un SASS, ces dernières sont définies à la fois selon leurs caractéristiques et mathématiquement. Après une discussion sur sept façon d'adresser l’incertitude dans un SASS, les auteurs discutent en profondeur de l’incertitude dans un contexte général en IoT. Avec la présentation d’un modèle en couche, Ismail et al. décrivent les sources d’incertitudes tout en les classifiant selon leur modèle générique de l’IoT. Finalement, des solutions spécifiques en IoT sont vaguement suggérées avant de conclure sur les difficultés des SASS.

Afin de mieux comprendre les SASS mais aussi d'approfondir les solutions architecturales disponibles, l’article publié en 2020 par Garcés et al. a été étudié. Après un bref rappel de l’utilité de ces systèmes, les auteurs présentent treize architecture en discutant de leurs différences selon leurs constituants (manager vs managed system), le type de contrôle utilisé ainsi que le besoin en adaptation. Après une analyse poussée de chacune de ces architectures, les auteurs proposent quatre modèles de SASS générique basés sur les modèles spécifiques présentés. À la suite de cette présentation, un scénario de cas (case scenario) où le niveau d’une rivière est observé permet de mieux comprendre comment les modèles génériques peuvent servir de base pour une implémentation. Afin de s’assurer de la valeur de ces modèles, quinze architectes spécialisés dans les SASS ont répondu à un questionnaire et les résultats compilés indiquent une bonne satisfaction. Ainsi, les auteurs concluent sur les limitations de leur étude mais surtout les nouvelles pistes que leur travaille ouvrent.

**Méthode**

La création d’un SASS, même en spécifiant un contexte d’IoT, demande plus de spécificité au niveau des exigences. Pour ce faire, l’article de Ismail et al. (2021) nous permet d’extraire les exigences fonctionnelles suivantes:

* Gérer les différences au niveau des débits de communication
* Assurer la continuité des services lorsque l'environnement devient inhospitalier
* Permettre la connexion aux réseaux pour une grande diversité d’Objet.
* Être extensible de manière horizontale et verticale
* Utiliser l’énergie des Objets efficacement
* Faciliter la configuration du système par l’utilisateur
* Stocker les données de manière ordonnée avec le maximum d’information
* Maintenir un bon standard quant à la fiabilité des données

Cependant, il est aussi important de prendre en compte les attributs de qualité que notre système devrait présenter. Dans l’article de Sobin (2020), sept attributs sont mentionnés comme étant particulièrement importants en IoT mais cinq ont été ciblé comme intéressant pour notre projet: extensibilité, efficacité énergétique, gestion des données, interopérabilité et capacité d’auto-organisation. Avec l’utilisation d’un SASS, l’efficacité énergétique, la gestion des données et l’auto-organisation semble prendre une place moins importante puisqu’elles sont incluses facilement dans un tel système. Pour ce qui est de l’extensibilité, l’architecture que nous présentons ici bas utilise le patron des micro-services qui permet, entre autres, d’améliorer l’extensibilité horizontale. De cette réflexion, les attributs d’interopérabilité, de modifiabilité et d’utilisabilité ont été sélectionnés comme focus principal pour les exigences non-fonctionnelles. Pour les mesurés, les ratios d’échanges et d’opérations seront utilisés en plus d’une discussion sur le nombre d’erreurs résolus, l’interface utilisateur ainsi que l’effort nécessaire pour effectuer une modification au système.

**Architecture**

Pour réaliser ce SASS nous avons fait un système de structure en couche. Avoir une architecture en couche apporte de nombreux avantages. Chaque couche est responsable d’une responsabilité ce qui facilite la modularité. L’architecture en couche facilite l’évolutivité, une évolution dans une des couches ne doit pas affecter les autres.Les composants de chaque couche peuvent être réutiliser dans d'autres systèmes.

Notre architecture est composée de quatre couches que l’on nomme UI, Manager, Managed et Object. Chaque couche à ses propres responsabilités.



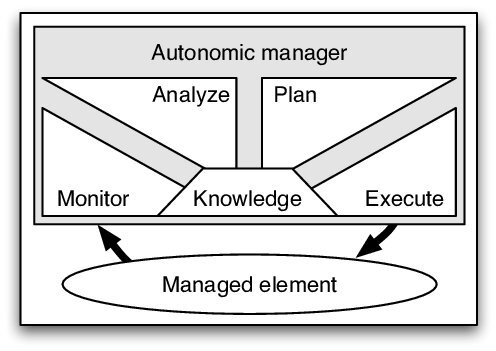
*Architecture en couche du système*

L’UI comme son nom l’indique est la couche qui gère les intéractions avec l’utilisateur. C’est la partie front-end de notre système. L’utilisateur peut visualiser les données, modifier ou ajouter des règles sur le système. Par exemple, il peut demander à ce que les données d’un objet connecté lui soient transmises toutes les 15 minutes au lieu de 30 minutes.

La couche inférieure nommée Manager est le cœur du système. C’est ici que la boucle MAPE-K est implémentée. Nous reviendrons en détails sur la boucle MAPE-K dans la section suivante. Elle est responsable de la gestion globale du système. Elle prend des décisions en fonction des données reçues et des exigences de l’utilisateur. Elle interagit donc à la fois avec l’UI, pour recevoir des instructions à l’utilisateur, et avec la couche Managed qui lui envoie des données. En fonction de l’analyse des données réalisé et des exigences utilisateur, elle va envoyer des instructions à la couche inférieur Managed

La couche Managed est l’interface entre la couche objet et la couche Manager. Elle reçoit les données des objets connectés en prenant en compte les différents protocoles et les transmet à la couche Manager. De plus, elle reçoit des instructions de la part de la couche Manger et transmet donc les commandes nécessaires aux objets connectés du système.

Enfin, la couche Objet représente les objets du système, ils transmettent les objets selon différents protocoles à la couche Managed et exécute les commandes reçues. Pour simuler les objets connectés dans notre système, nous utilisons des datasets et les objets transmettent les données du dataset à une certaine fréquence.

**MAPE-K**

*Ayeb, Neil. (2020). Administration autonomique et décentralisée de flottes d'équipements de l'Internet des Objets.*

La boucle MAPE-K est présente dans la couche Manager. Cette boucle décrit le cycle auto-adaptant de notre système. La boucle MAPE-K est constituée de cinq éléments, le Monitor, l’analyse, Plan et Execute. Dans cette section nous expliquerons le rôle de chaque élément présent dans le schéma ci-dessus.

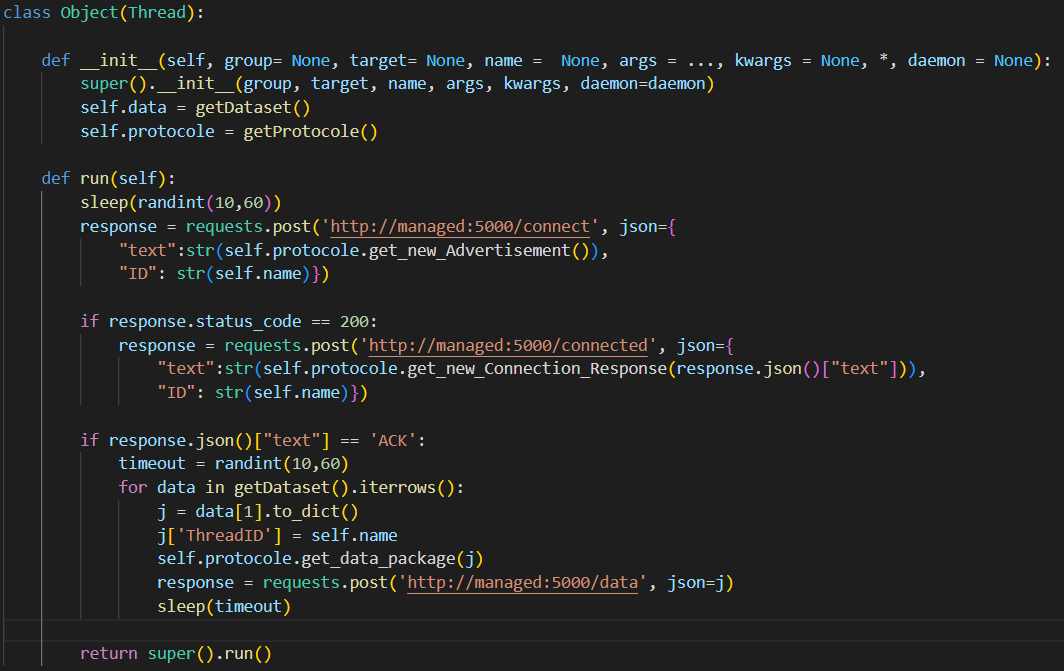
La boucle commence par la réception des données envoyées par la couche Managed. La partie Monitor (Filter) est chargée de récupérer ces données et en extrait le contexte de l’objet mais aussi apprend le motif envoyé dans les données. Vient ensuite la phase de l'analyse. L’objectif de l’analyse est de détecter des problèmes. Pour cela, elle utilise les données historiques, le contexte et les motifs pour vérifier que les objectifs définis par l’utilisateur ( exemple : fréquence de donnée = 15 minutes) sont respectés et qu’il n’y a pas d’erreur dans le système. Lorsqu’un problème est détecté c’est au Plan (planner) de déterminer le meilleur plan d’action. Pour récupérer le meilleur plan d'action, il utilise la Policy. Si aucun plan n’est trouvé, il demande à l’utilisateur de définir un plan d’action. Le planner élabore les scripts en fonction du plan d’action sélectionné. Ces scripts vont être utilisés par l’élément exécute (executor). L’executor peut transmettre des commandes nécessaires à la couche Managed si un objet doit changer son comportement. Si des erreurs interviennent l’utilisateur en sera informé.

****

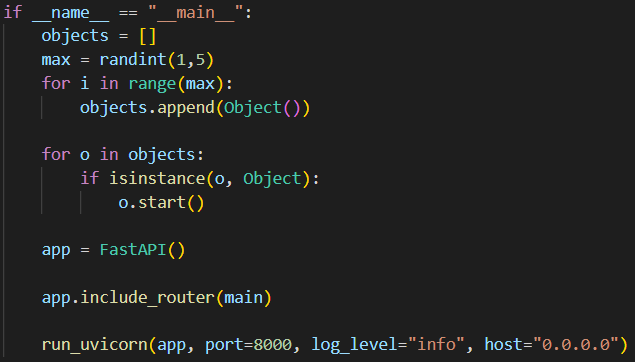
**Résultat**

Pour ce qui est de l’implémentation, elle a été conçue de façon à ce que chaque couche soit déployée comme un microservice. De ce fait, la solution finale inclut quatre microservices et une base de données déployés avec docker-compose. Pour exécuter la solution, il faut donc utiliser la commande ‘docker-compose up --build’ à partir de la racine du projet. La description du déploiement peut être trouvée dans le fichier ‘docker-compose.yml’. Pour la base de données, l’image officielle de InfluxDB à été utilisée avec des accès par le manager et le UI.

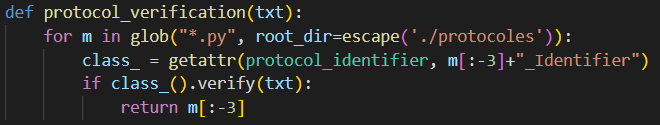
En commençant avec la plus basse couche, le service Objet permet l’émulation d’un objet IoT selon des données réelles. Dans le module ‘Object’, la classe émulant l’objet est définie avec la classe ‘Thread’ de la librairie Threading pour que chaque objet émulé ait son propre fil d’exécution. Lors de la création d’un Objet, un dossier contenant des datasets et un autre contenant des protocoles de communication sont parcourus afin de choisir une entité de chaque type de façon aléatoire comme attributs. Une fois activés et passé une attente de quelques secondes, les objets tentent une connexion vers le service Managed à l’aide du protocole de communication choisi. Une fois la connexion établie, les Objets envoient les données extraites du dataset choisi au Managed et, une fois la réponse reçue, ils attendent un nombre aléatoire de secondes avant d'envoyer la prochaine valeur.

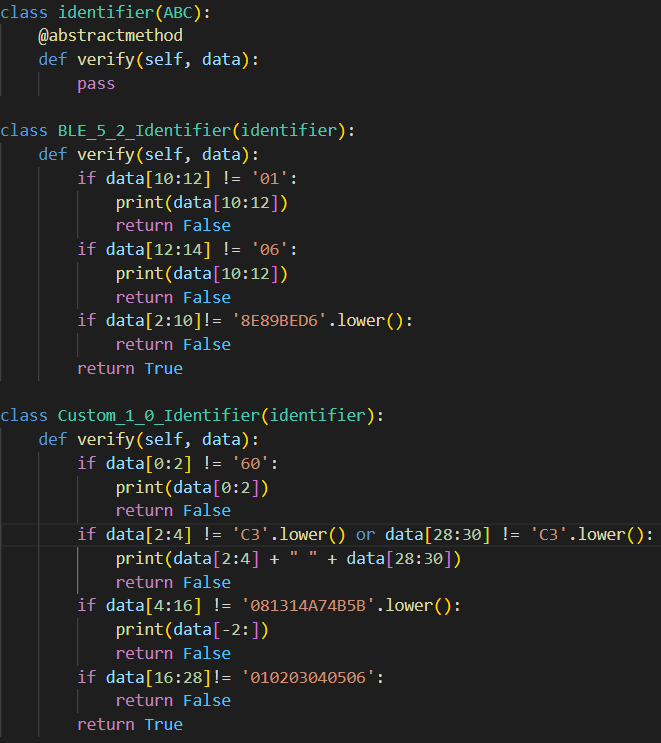


En plus de la création des objets, le service Objet initialise un serveur web permettant de recevoir les requêtes pour les objets. Puisque chaque objet utilise le même port du conteneur, le serveur permet de recevoir une requête et de la rediriger vers le bon fil d’exécution.

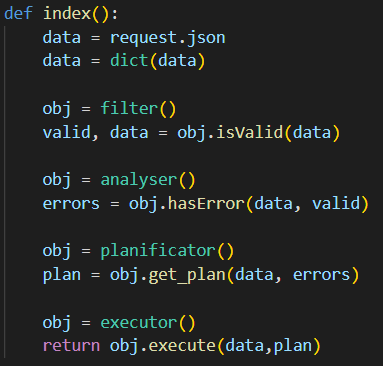


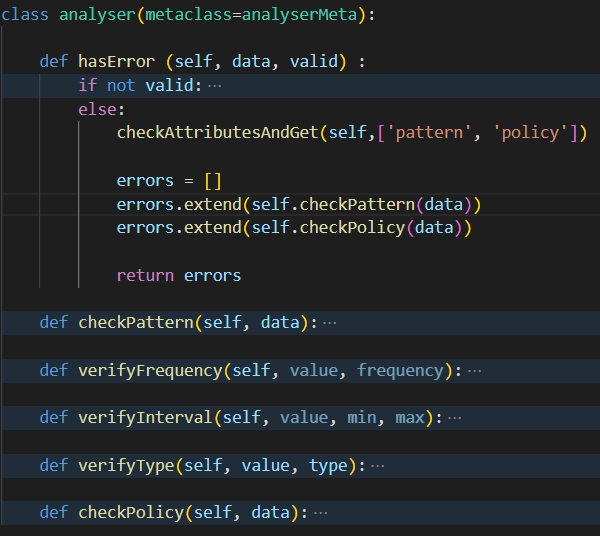
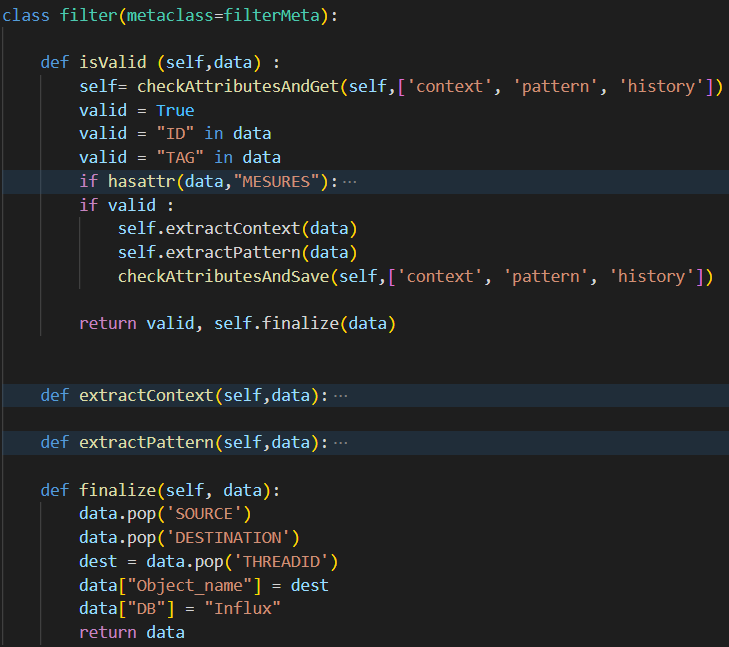
De son côté, le service Managed implémente trois routes: connect, connected et data. Dans la première route, connect, l’objet tente une connexion au réseau mais le service ne connaît pas l’objet ou son protocole. Ainsi, la première étape lors d’une requête sur cette route est de vérifier le protocole à l’aide du module ‘protocol\_identifier’. Pour ce faire, le nom des protocoles est extrait du dossier et un objet pour son identification est instancié. Une fois le protocole découvert, l’objet est cartographié (mapped) vers son protocole et le tout est sauvegardé dans un fichier. Pour les requêtes sur connected, une vérification de la cartographie est effectuée avant d’envoyer une confirmation alors que la route data ne sert qu’à faire le lien entre l’Objet et Manager.



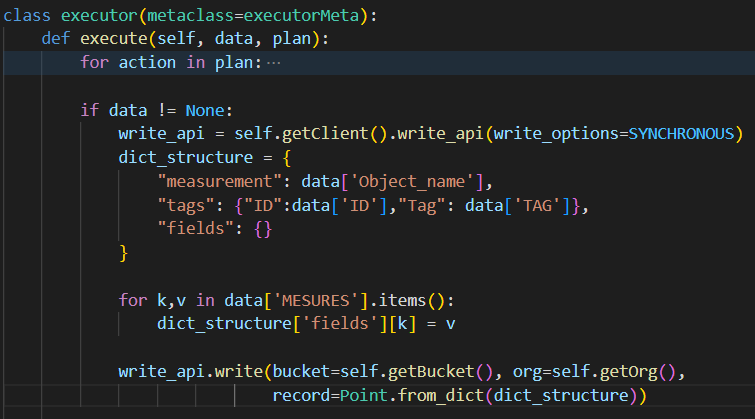
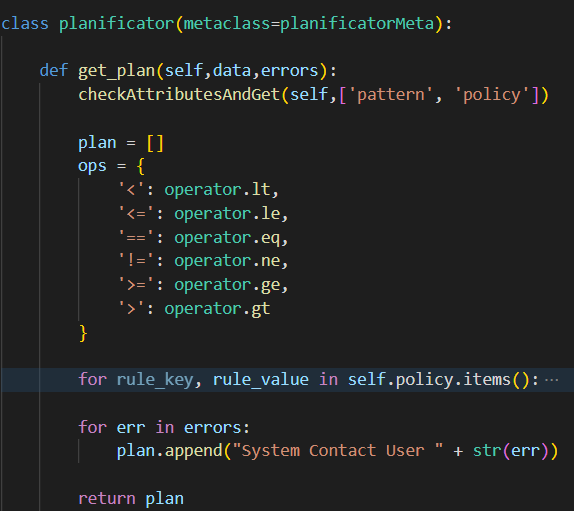


Pour ce qui est de Manager, le service offre une route à l’index pour les données ainsi qu’une API RESTful pour la gestion des policies, mais l’API ne sera pas discuté ici. Dans ce service, le MAPE-K est implémenté. Pour la partie Knowledge, un dossier est disponible pour toute la boucle sous forme de fichier yml ou csv (context, history, pattern, policy) et des fonctions de get et set dans le module ‘Util’ permet d’utiliser le knowledge facilement. Le reste du service consiste à une transition entre chaque étape du MAPE pour chaque donnée reçue.

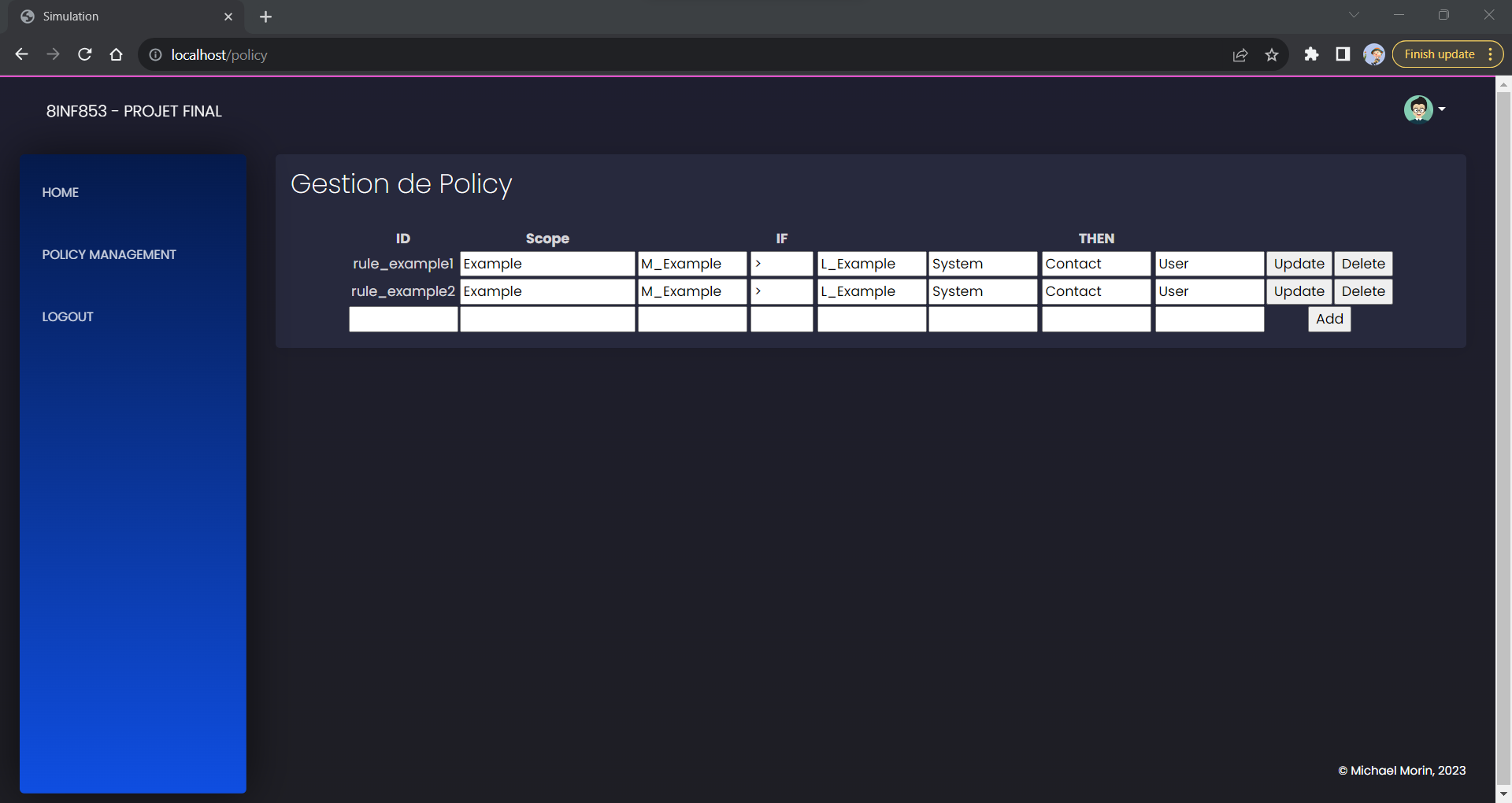
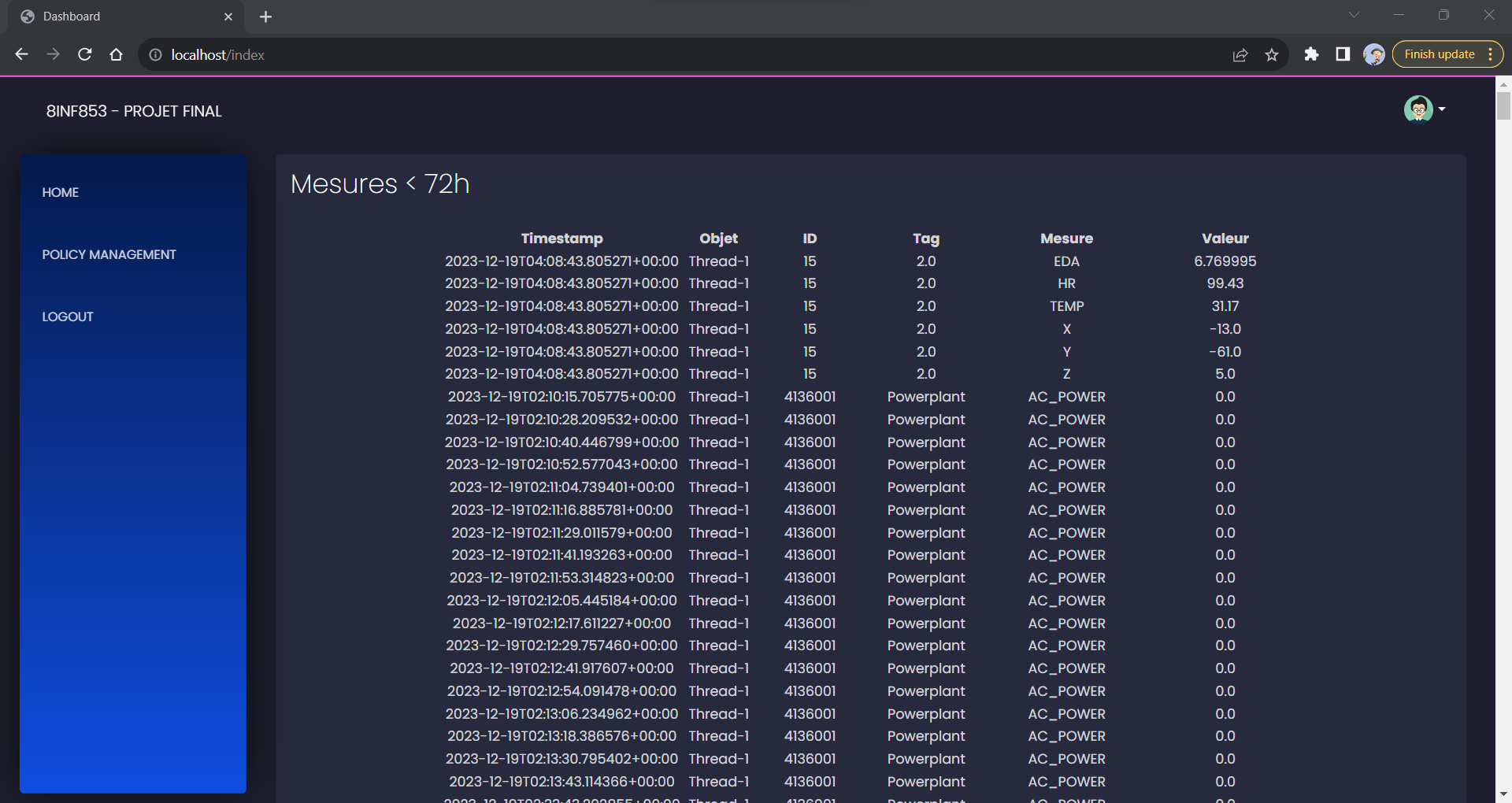


La première étape, Monitor, est implémentée dans l’objet Filter. Dans cet objet, les données sont étudiées pour en vérifier la validité, en extraire le contexte et les patterns avant de modifier la structure pour la prochaine étape. Cette dernière est exécutée à l’aide de l’objet Analyser. Dans cet objet, les patterns et les policies sont vérifiés pour s'assurer que les mesures n’ont pas d’erreurs. Dans le cas où des erreurs sont détectées, elles sont passées à la prochaine étape.

En troisième étape, le Planificator sert d’objet pour la planification. Dans cet objet, la librairie operator est importée car, tout comme avec analyser pour vérifier les policies. En fait, les règle des policies suivent la forme IF…THEN et la clause ‘if’ inclut un sujet et un objet à comparer à l’aide d’un opérateur booléen. En utilisant un dictionnaire et les fonctions correspondant à l’opérateur, on peut vérifier que le sujet et l’objet sous formes de variable ou de constante suivent la policy. Sinon, la clause ‘then’ est assemblée et placée dans une liste passé à l'exécuteur. De plus, si une erreur n’est pas visée par une policies, une clause est assemblée pour demander à l’utilisateur de diriger l’action pour résoudre l’erreur. Du côté de l’executor, les clauses sont décodées et exécutées et, à moins d’indication contraire, les données sont sauvegardées dans la base de données.



Finalement, l’UI permet à l’utilisateur de voir les données dans un tableau ou encore de gérer les policies. Considérant un manque de temps, la présentation est plutôt simple mais efficace.



**Discussion**

En plus d’une implémentation satisfaisante, nous avions pour objectif d’avoir une solution logicielle faisant preuve d’interopérabilité, de modifiabilité et d’utilisabilité. Pour ce qui est de l’interopérabilité, deux protocoles ont été implémentés comme simulation d’objet et dans managed. Cependant, l’utilisation de librairies aurait facilité grandement ce travail mais l’interaction avec des objets n’était pas possible. Ainsi, une deuxième version serait nécessaire pour vérifier cet attribut en pratique.

Au niveau de la modifiabilité, des efforts ont été fait. Au niveau des protocoles, le logiciel ne nécessite que l’implémentation de l’interface pour ajouter un nouveau protocoles. Avec les microservices, le nombre de managed et de manager peut être modifié très facilement. Pour l’interface graphique, la librairie Flask a été utilisée ce qui permet de réutiliser les templates et le code facilement. De plus, le knowledge a été sauvegardé en fichier yml au lieu de json pour qu’il soient plus facile à lire et l’historique utilise le format csv afin de pouvoir utiliser facilement la librairie pandas. Cependant, une prochaine itération devrait pouvoir considérer plus d’opérateur et possiblement utiliser un dictionnaire dans knowledge afin de traduire les policies en instruction exécutable.

Finalement, au niveau de l’utilisabilité, la solution logicielle fonctionne plutôt bien. Avec la simulation d’objet, il suffit de régler les paramètres du générateur de nombre aléatoire afin d’augmenter le nombre d'objets et d’ajouter un dataset ou un protocole pour diversifier les objets. Cependant, le dataset n’a pas encore de module pour le mettre automatique dans le format nécessaire. Finalement, la simplicité de l’interface utilisateur permet une utilisation amicable pour le client.

**Conclusion**

Bien que l’article initial fût quelque peu décevant, l’idée d’un système logiciel auto-adaptant pour l’Internet des Objets a semblé intéressante. Avec une implémentation en microservice d’une architecture en couche, une simulation d’objet, une passerelle pour les protocoles, une boucle MAPE-K et une interface utilisateur furent développés. Avec un focus sur l’interopérabilité, la modifiabilité et l’utilisabilité, la solution finale offre une base sur tous ces attributs mais une seconde itération répondrait mieux à ces besoins non-fonctionnels. Pour pousser encore plus loin, une recherche sur l’implémentation de l’intelligence artificielle dans un système auto-adaptant pourrait aboutir à une solution nettement plus fiable et performante.

**Références**

Afaneh, M. (2023, March 30). A Deep Dive into BLE Packets and Events. Novel Bits. https://novelbits.io/deep-dive-ble-packets-events/

Ayeb, N. (2020, November 25). Administration autonomique et décentralisée de flottes d’équipements de l’Internet des Objets. https://theses.hal.science/tel-03152166

Catalogue of parametrised CRC algorithms. (n.d.). https://reveng.sourceforge.io/crc- catalogue/all.htm

Core Specification Working Group. (2019). Bluetooth Core Specification, v 5.2. In bluetooth.com. Bluetooth SIG. Retrieved December 19, 2023, from https://www.bluetooth.com/specifications/specs/core-specification-5-2/

MalteGruber. (n.d.). GitHub - MalteGruber/zigpy\_standalone: A minimal demo of a standalone zigpy application that serves as an introduction to zigpy and the ZigBee. GitHub. https://github.com/MalteGruber/zigpy\_standalone/tree/main

Müller. (Spring 2013). Welcome to SENG 371 Software Evolution [Slide show; Powerpoint]. Faculty of Engineering and Computer Science, Univeristy of Victoria. https://www.engr.uvic.ca/~seng371/lectures/L6-371-S13-bw.pdf

Welcome to CASAS datasets. (n.d.). https://casas.wsu.edu/datasets/