



I.U.T CLERMONT-FERRAND

PROJET N°14
OPENTHERMOSTAT
RAPPORT

Rapport de projet tuteuré

Élèves :

Yannis MAHIOU
Gabin SALABERT
Benoît LOUVEAU
Elouan RAFFRAY
Adrien LENOIR

Tuteur :

Sébastien SALVA

Décembre 2017

Sommaire

1	Autorisation à diffuser sur l'intranet de l'IUT.	3
2	Remerciements	4
3	Introduction	5
4	Environnement	6
5	L'Existant	7
6	Objectifs	8
7	Patrons de conception	9
7.1	Patron "Memento"	9
7.2	Patron "Singleton"	10
7.3	Patron "Fabrique simple"	11
7.4	Patron "Facade"	12
7.5	Patron "Adaptateur"	13
7.6	Patron "Poids-mouche"	14
7.7	Patron "Itérateur"	15
7.8	Patron "Chaîne de responsabilité"	16
7.9	Patron "Composite"	17
8	Implémentation des patrons de conception	18
8.1	Code source	18
8.2	Implémentation du Singleton	19
8.2.1	Détails des données système (Singleton)	20
8.3	Implémentation du Memento	21
8.3.1	Détails des données système (Memento)	22
9	Exécution des tests de pénétration	23
9.1	Utilisation de (Airmon/Airodump/Aircrack)-ng	23
9.2	Utilisation de ZaProxy	24
9.2.1	Précisions	25
9.3	Présentation de Krack-Attack	28
9.4	Burpsuite	29
9.5	Injectons HTML JavaScript	30
10	Patrons de sécurité	31
10.1	Patron "Input Guard"	31
10.2	Patron "Secure Logger"	33
10.3	Patron "Single Access Point"	34
10.4	Patron "Application firewall"	36
10.5	Patron "Security context"	38
10.6	Patron "Audit Interceptor"	39
10.7	Patron "Authentication Enforcer"	40
10.8	Patron "Firewall"	41

10.9 Patron "Output Guard"	42
10.10Patron "TimeOut"	43
10.11Patron "Session"	45
10.12Patron "Encrypted Storage"	48
 11 Résumé en anglais	 49
 12 Bibliographie	 50
 13 Lexique	 50
 14 Annexes	 50

1 Autorisation à diffuser sur l'intranet de l'IUT.

J'autorise/nous autorisons la diffusion de mon/notre rapport sur l'intranet de l'IUT

2 Remerciements

Nous remercions monsieur Salva pour nous avoir permis de travailler sur un projet original qui nous a fait découvrir l'I.O.T.

Mais aussi pour le prêt de carte arduino et l'aide apportée tout au long du projet.

3 Introduction

Ce projet est encadré par M.Salva et est réalisé par Gabin Salabert, Yannis Mahiou, Adrien Lenoir, Benoit Louveau et Elouan Raffray. Il s'inscrit dans le cadre de l'IOT (Internet of Things, regroupant toutes manifestations physiques d'Internet) et concerne un objet connecté, composé d'un microcontrôleur avec Wifi (programmable avec Arduino), d'un capteur de présence, d'un capteur de température et d'un émetteur/récepteur infrarouge. L'objet communique avec des pompes à chaleurs pour en permettre le contrôle automatique. Le code de la carte était déjà existant et nous a été donné.

Le sujet est centré sur l'étude et la modification du code existant sur deux axes principaux, la conception et la sécurité. Cette démarche est principalement effectuée afin que nous puissions approfondir nos connaissances dans ces domaines mais aussi améliorer notre démarche d'analyse et de résolution de défauts. Tout cela en analysant l'application et les problèmes que son code contient.

Pour ceci, une compréhension générale du code par l'équipe a été nécessaire. Le sujet, réalisé en deux temps, comprend deux équipes : sécurité et conception. Dans un premier lieu, une analyse générale du code existant a été réalisée, puis nous avons réparti les tâches entre ces deux équipes afin d'aborder les travaux en conception et en sécurité.

Pour l'équipe conception, il fallait analyser les patrons, justifier leur nécessité ou leur non-nécessité puis implémenter ceux potentiellement indispensable. Pour l'équipe sécurité, les étapes étaient de tester les outils de pénétration, utiliser les plus aptes à détecter les failles existante, puis analyser les patrons permettant de corriger ces dernières et enfin les implémenter.

Nous allons donc présenter les résultats de ces recherches, en partant de l'analyse générale, puis en expliquant les travaux réalisés en conception et en sécurité.

4 Environnement

Le système étudié possède une interface web avec laquelle l'utilisateur peut interagir pour paramétrer le mode de fonctionnement des pompes à chaleurs.

La carte peut être reliée à un réseau WiFi pour faciliter sa liaison avec d'autres appareils de domotique.

Le code est en C++ avec une base Arduino, et nous l'avons modifié en utilisant Visual Micro.

Pour l'analyse du code existant et la mise en place des implémentations de patrons, des diagrammes de classes ont été mis en place avec StarUML.

5 L'Existant

Le projet existe depuis 6 mois et possède un code fonctionnel disponible sur Github.

Comme précisé dans l'introduction, c'est un système codé en C++ et doté d'une base Arduino qui permet de gérer plusieurs pompes à chaleur via un émetteur/récepteur infra-rouge. Les modifications portent principalement sur la conception et la sécurité.

En effet, sont présentes sur ce projet des failles de sécurité ainsi qu'une architecture potentiellement améliorable en terme de performances.

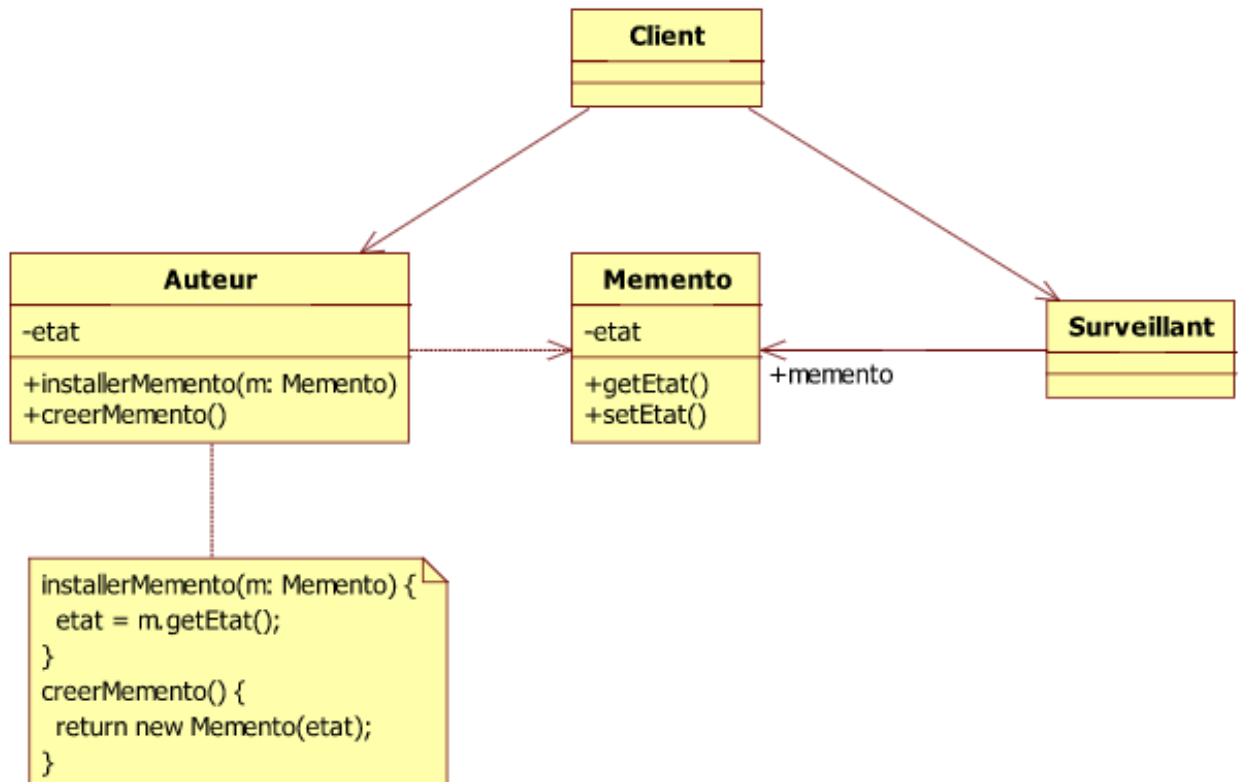
Le projet comprend aussi une interface web permettant à l'utilisateur de gérer les paramètres des pompes et d'accéder aux données comme la température et l'humidité.

6 Objectifs

7 Patrons de conception

7.1 Patron "Memento"

Sans violer l'encapsulation, le Memento permet de saisir et de transmettre à l'extérieur d'un objet l'état interne de celui-ci, dans le but de pouvoir ultérieurement le restaurer dans cet état.



Ici, il nous est presque indispensable dans le sens où de nombreux éléments doivent être sauvegardés de façon à pouvoir les restaurer ultérieurement. En effet, nous aurions alors des sauvegardes de l'état à des moments précis et décisifs dans le cas où un problème surviendrait.

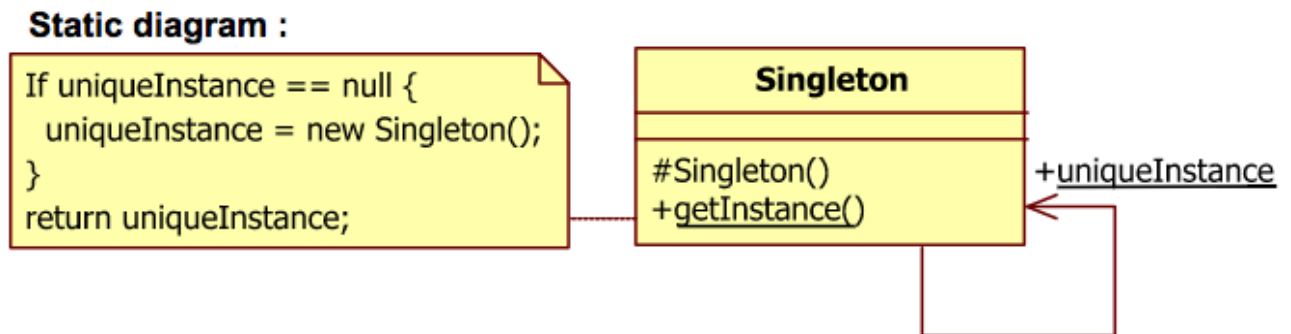
Memento : Mémoire de l'état interne de l'objet original. Il peut stocker autant d'informations d'état interne de l'original que nécessaire, à la discrétion de ce dernier. Protège des incursions d'objets autres que l'auteur. Les mementos ont effectivement deux interfaces. Le Surveillant voit l'interface étroite du Memento - il ne peut que faire passer le memento aux autres objets.

Auteur : Crée un objet Memento contenant un instantané de son état interne courant et utilise Memento pour restaurer son état interne.

Surveillant : Est responsable de la sauvegarde de Memento. N'agit jamais sur Memento, ni ne l'examine.

7.2 Patron "Singleton"

Le principe de ce patron est de garantir qu'une classe n'a qu'une seule instance et fournir un point d'accès de type global à cette classe.



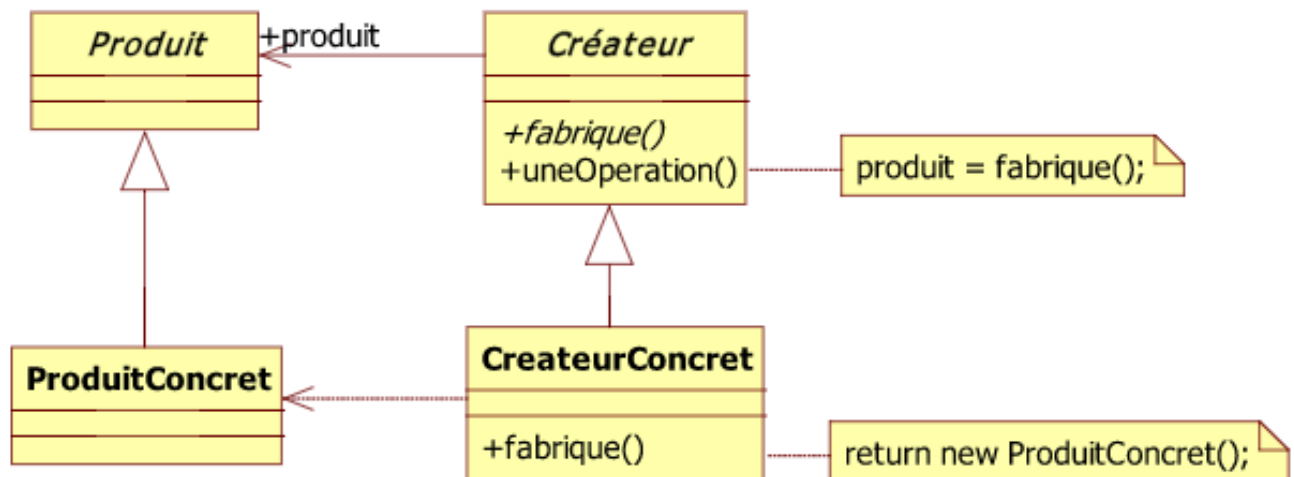
Dans le cadre de notre projet, il ne doit y avoir exactement qu'une instance de la classe Application, sinon la carte ne pourrait pas le supporter.

De plus, l'instance unique doit être accessible aux autres classes qui l'utiliseront afin d'accéder aux variables.

Singleton : Définit une opération Instance qui donne au client l'accès à son unique instance. Instance est une opération de classe (c'est-à-dire, une méthode de classe en Smalltalk, et une fonction membre statique en C++). Il peut avoir la charge de créer sa propre instance unique.

7.3 Patron "Fabrique simple"

Ce patron définit une interface pour la création d'un objet, mais en laissant à des sous-classes le choix des classes à instancier. La Fabrique simple permet à une classe de déléguer l'instanciation à des sous-classes.



L'implémentation de ce patrons permettrait de gérer l'instanciation de différents objets dans des cas bien spécifiques. La plupart du temps, il est en effet utilisé dans les cas où l'instanciation d'un objet x ou y n'est pas prévisible. Or cette situation ne se présente pas au sein de notre projet étant donné que chaque objet est instancié et joue un rôle.

Produit : Définit l'interface des objets créés par la fabrique.

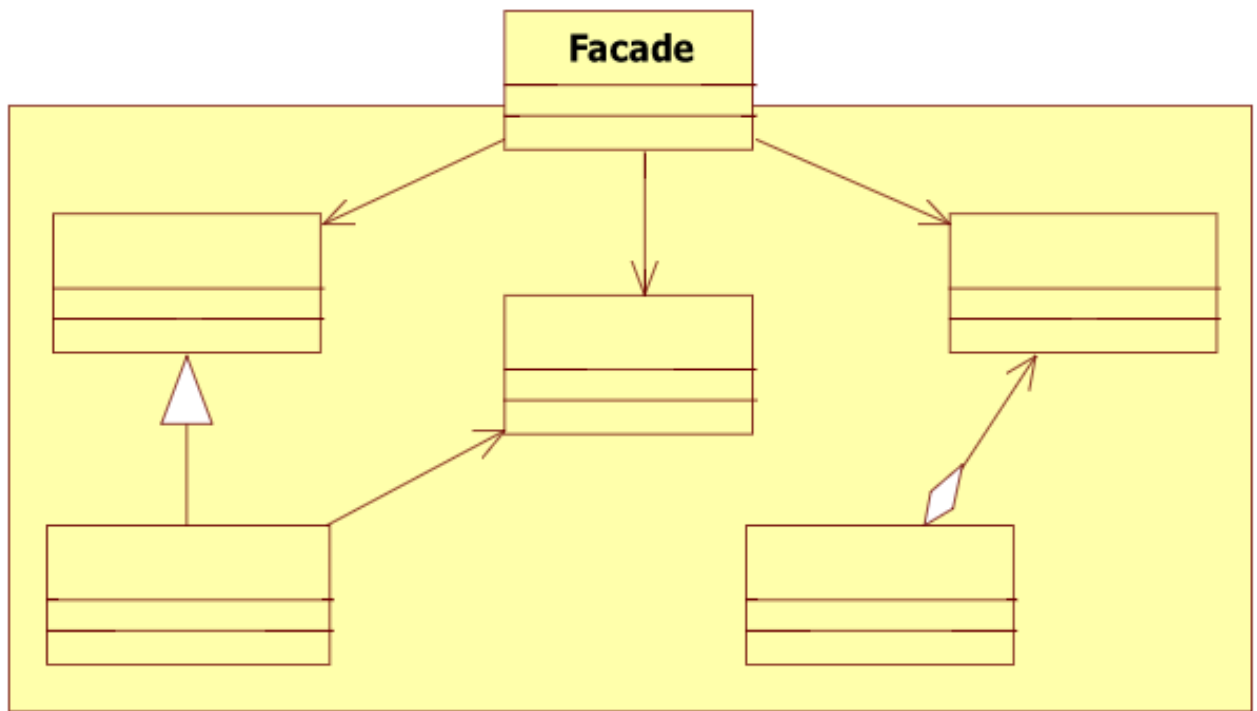
Produit Concret : Implémente l'interface Produit.

Créateur : Déclare la fabrique; celle-ci renvoie un objet de type Produit. Le Créateur peut également définir une implémentation par défaut de la fabrique, qui renvoie un objet ProduitConcret par défaut. Peut appeler la fabrique pour créer un objet Produit.

Créateur Concret : Surcharge la fabrique pour renvoyer une instance d'un Produit Concret.

7.4 Patron "Facade"

Elle fournit une couche d'encapsulation supplémentaire, à l'ensemble des objets d'un sous-système. La Façade fournit donc une interface de plus haut niveau, qui rend le sous-système plus facile à utiliser.



L'implémentation d'une façade permettrait donc d'améliorer la lisibilité du code et du modèle. Cependant, dans notre cas, elle n'est pas implémentable dans le sens où nous n'avons pas besoin de lier les objets entre eux via un point d'accès unique. Il n'y a pas d'intérêt à ce qu'ils communiquent de cette façon étant donné que nous n'avons pas d'informations ni de système extérieurs (Stub, etc..) qui pourraient être récupérés et utilisés par tous.

Façade : Connaît les classes du sous-système compétentes pour une requête. Délègue le traitement des requêtes clients aux objets appropriés du sous-système.

Classes du sous-système : Implémentent les fonctionnalités du sous-système. Gèrent les travaux assignés par l'objet Façade. Ne connaissent pas la façade ; c'est-à-dire qu'elles n'ont pas de références à celle-ci.

7.5 Patron "Adaptateur"

Convertit l'interface d'une classe en une autre conformément à l'attente du client. L'Adaptateur permet à des classes de collaborer, alors qu'elles n'auraient pas pu le faire du fait d'interfaces incompatibles.

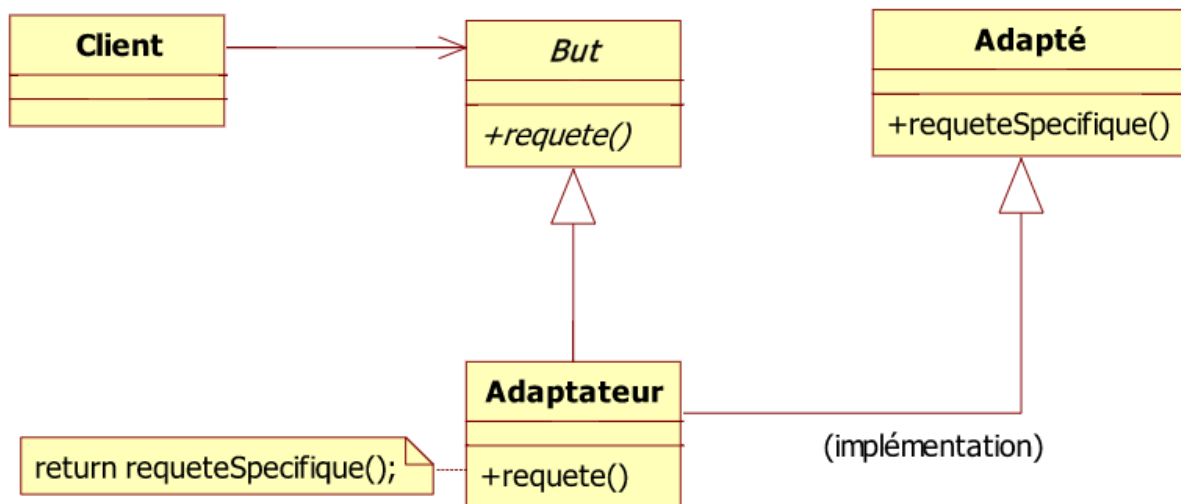
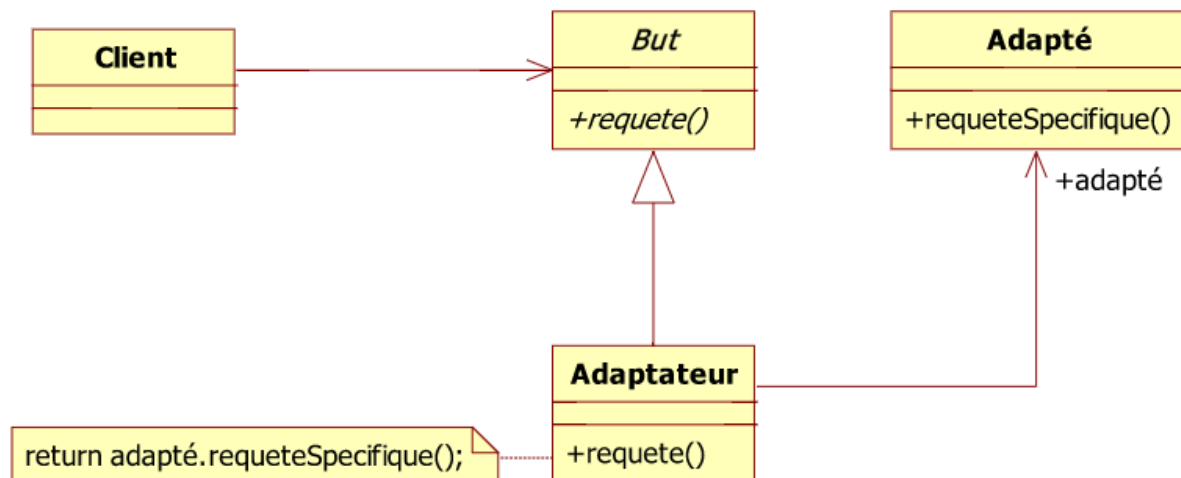


Diagramme de classes pour l'adaptateur d'objets :



Tout d'abord, nous n'avons pas d'interface, ce qui n'est donc pas compatible avec ce patron.

De plus, nos classes collaborent déjà par le biais de la classe Application, point où elles convergent toutes.

But : Définit une interface spécifique du domaine qu'utilise le client. Délègue le traitement des requêtes clients aux objets appropriés du sous-système.

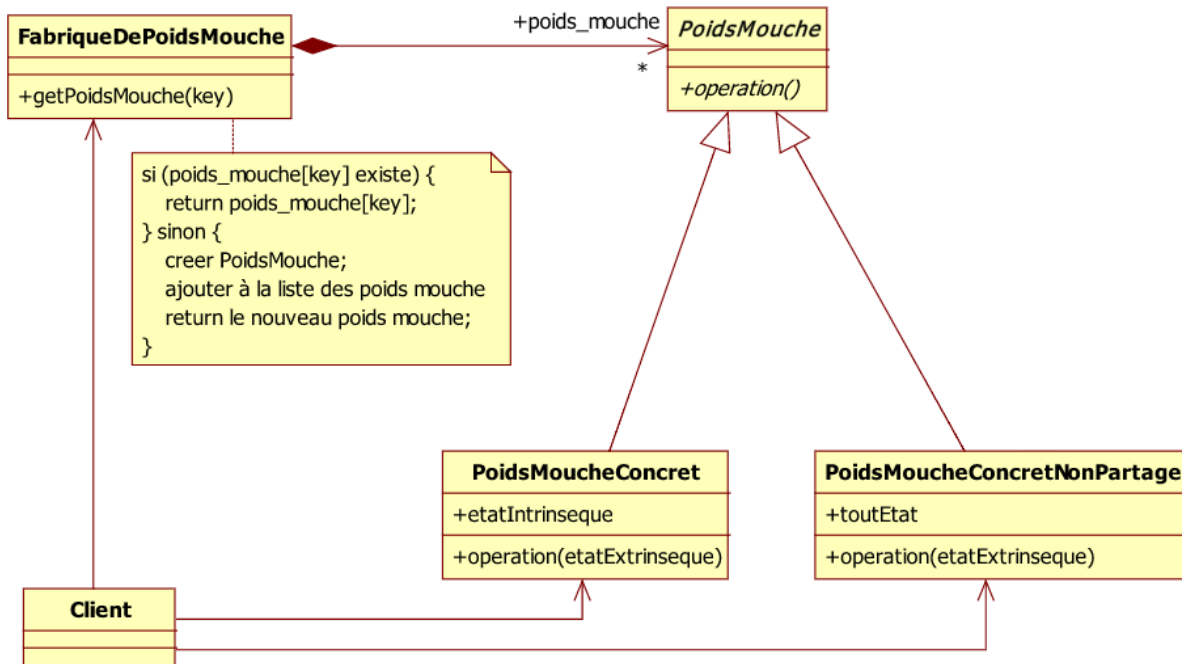
Client : Collabore avec les objets en se conformant à l'interface du But.

Adapté : Définit une interface existante qui demande adaptation.

Adaptateur : Adapte l'interface de l'adapté à l'interface But..

7.6 Patron "Poids-mouche"

Utilise une technique de partage qui permet la mise en oeuvre efficace d'un grand nombre d'objets.



Nous n'avons pas d'intérêt à utiliser ce patron dans ce projet. Nous n'avons pas de prolifération d'instances d'objets à gérer et à manipuler. Ces objets n'ont, par ailleurs, pas de point commun intrinsèque. L'utilisation de ce patron de conception n'est donc pas utile.

PoidsMouche : Déclare une interface à travers laquelle les poids mouche peuvent recevoir les états extrinsèques et agir sur eux.

PoidsMoucheConcret : Implémente l'interface d'un poids mouche et stocke éventuellement les états intrinsèques. Un objet **PoidsMoucheConcret** doit être partageable. Tout état qu'il contient doit être intrinsèque ; c'est-à-dire être indépendant du contexte de l'objet.

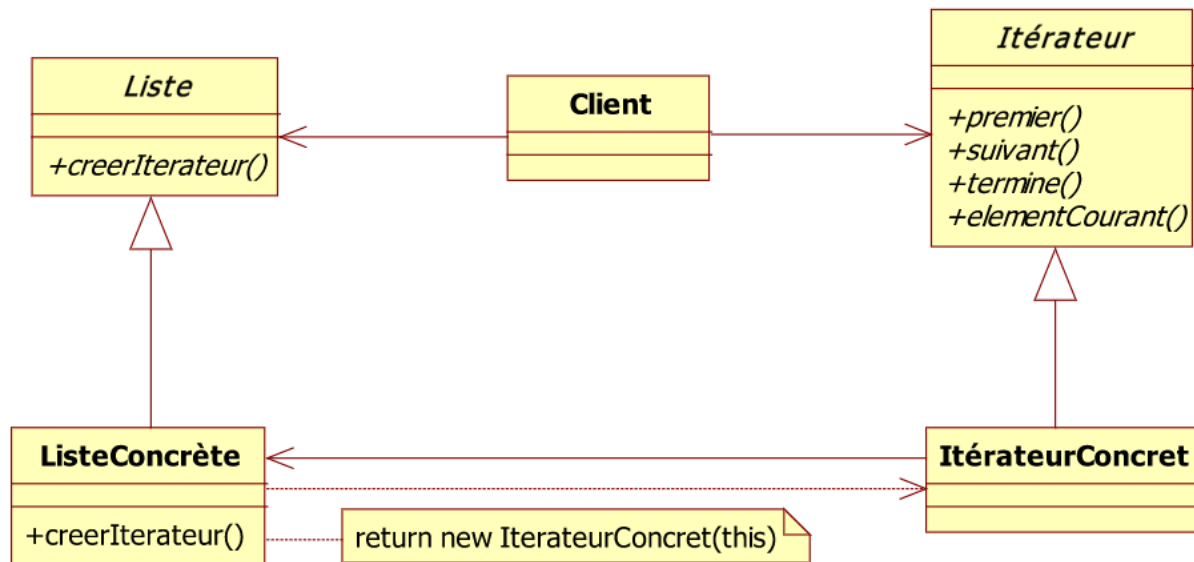
PoidsMoucheConcretNonPartagé : Toutes les sous-classes de **PoidsMouche** ne sont pas nécessairement partagées. L'interface **PoidsMouche** permet le partage, il ne l'impose pas. Il est courant pour des objets **PoidsMoucheConcretNonPartage** d'avoir pour enfants des objets **PoidsMoucheConcret** à certains niveaux de la structure d'objets **PoidsMouche**.

FabriqueDePoidsMouche : Crée et gère des objets **PoidsMouche**. S'assure que les objets **PoidsMouche** sont convenablement partagés. Si un client réclame un Poids Mouche, l'objet **FabriqueDePoidsMouche** fournit une instance existante ou en crée une s'il n'y en a pas.

Client : Gère une référence aux Poids Mouche. Calcule ou mémorise l'état extrinsèque des Poids Mouche.

7.7 Patron "Itérateur"

Permet d'accéder à une liste itérée et de l'utiliser même durant le parcours. Il fournit donc un moyen d'accès séquentiel aux éléments de cette liste.



Nous n'avons tous simplement pas de liste au sein de notre projet. L'implémentation n'est donc pas possible.

Itérateur : Définit un interface pour accéder aux éléments et les parcourir.

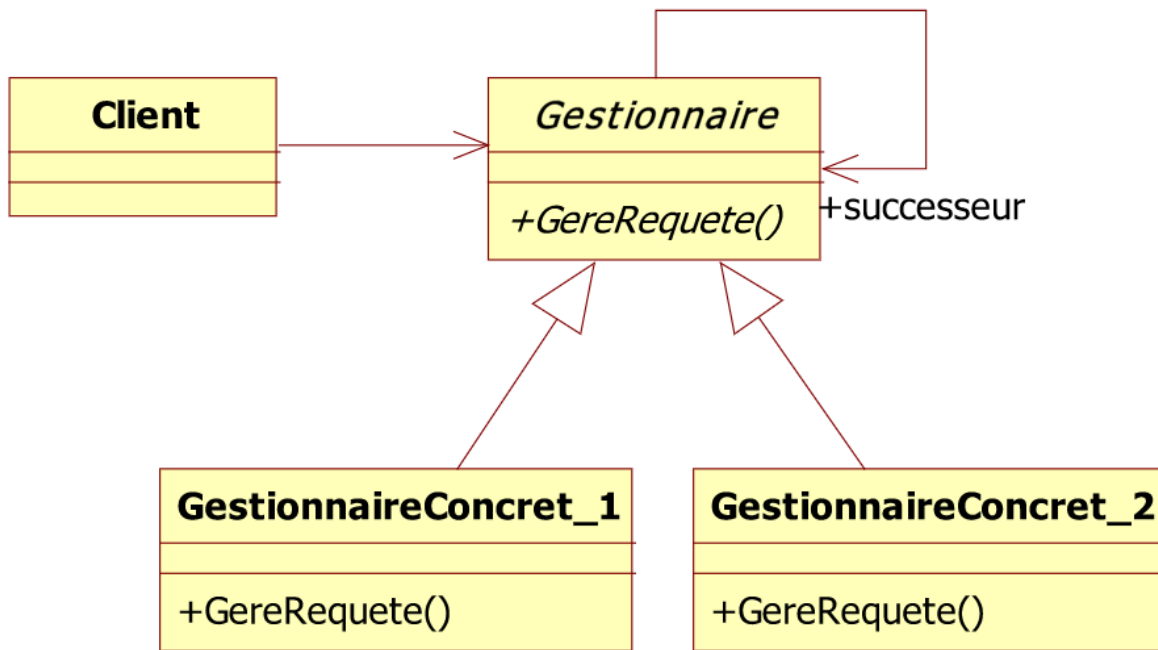
ItérateurConcret : Implémente l'interface Itérateur. Assure le suivi de l'élément courant lors de la traversée d'un agrégat.

Liste : Définit une interface pour la création d'un objet Itérateur.

ListeConcrète : Implémente l'interface de création de Itérateur, afin de retourner l'instance adéquate de ItérateurConcret.

7.8 Patron "Chaîne de responsabilité"

Chaîne les objets récepteurs et fait passer la requête tout au long de la chaîne, jusqu'à ce qu'un objet la traite.



L'implémentation de ce patron ne servirait à rien étant donné que chaque objet a sa propre utilité spécifique. C'est à dire qu'une requête ne pourra pas être réalisée par plusieurs objets ni proposée à plusieurs objets. Chacun d'eux répondent à des demandes bien précises qu'ils sont les seuls à pouvoir réaliser.

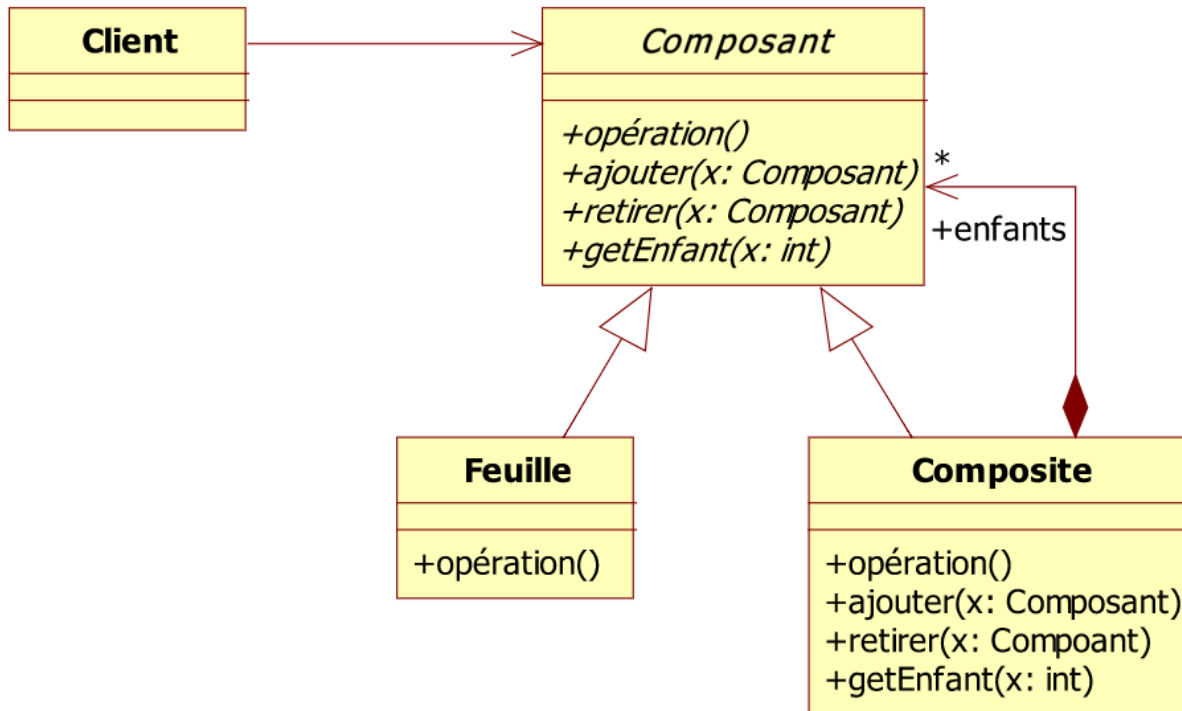
Gestionnaire : Définit une interface pour gérer les requêtes. Comporte (éventuellement) le code de liaison avec le successeur.

GestionnaireConcret : Gère les requêtes dont il a la charge. Sait accéder à son successeur. Traite la requête s'il le peut, sinon la transmet à son successeur.

Client : Propose initialement la requête à un objet GestionnaireConcret de la chaîne.

7.9 Patron "Composite"

Réuni des objets en des structures arborescentes pour représenter des hiérarchies. Le client peut donc traiter d'une façon unique les objets et combinaisons d'objets.



Il serait peut être intéressant d'utiliser le patron composite dans notre cas. En effet, on peut remarquer que nous avons plusieurs composants de la loop qui pourraient être factorisés. Par exemple, Web/NTP/Wifi dans un composite Network. Ou encore Sensor-Manager/Thermostat, Ir dans Sensor.

De plus, le composite crée de nombreuses possibilités d'extension car l'ajout de feuilles ou la suppression de composite n'implique pas de modification du code, nous pourrions donc ajouter des capteurs (ou autres) sans difficultés.

Composant : Déclare l'interface des objets entrant dans la composition. Implémente le comportement par défaut qui convient pour l'interface commune à toutes les classes. Déclare une interface pour accéder à ses composants enfants et les gérer.

Feuille : Représente des objets feuille dans la composition. Une feuille n'a pas d'enfants. Définit le comportement d'objets primitifs dans la composition.

Composite : Définit le comportement des composants dotés d'enfants. Il stocke les composants enfants. Il implémente les opérations liées aux enfants dans l'interface Composant.

Client : Manipule les objets de la composition à l'aide de l'interface Composant.

8 Implémentation des patrons de conception

8.1 Code source

Au départ, nous avions un code source sans patrons implémenté, que ce soit de conception ou de sécurité. Nous avons alors pris l'initiative de sauvegarder la taille de build, la taille mémoire, la taille des sketches ainsi que l'utilisation CPU afin de pouvoir les comparer après l'implémentation de patrons.

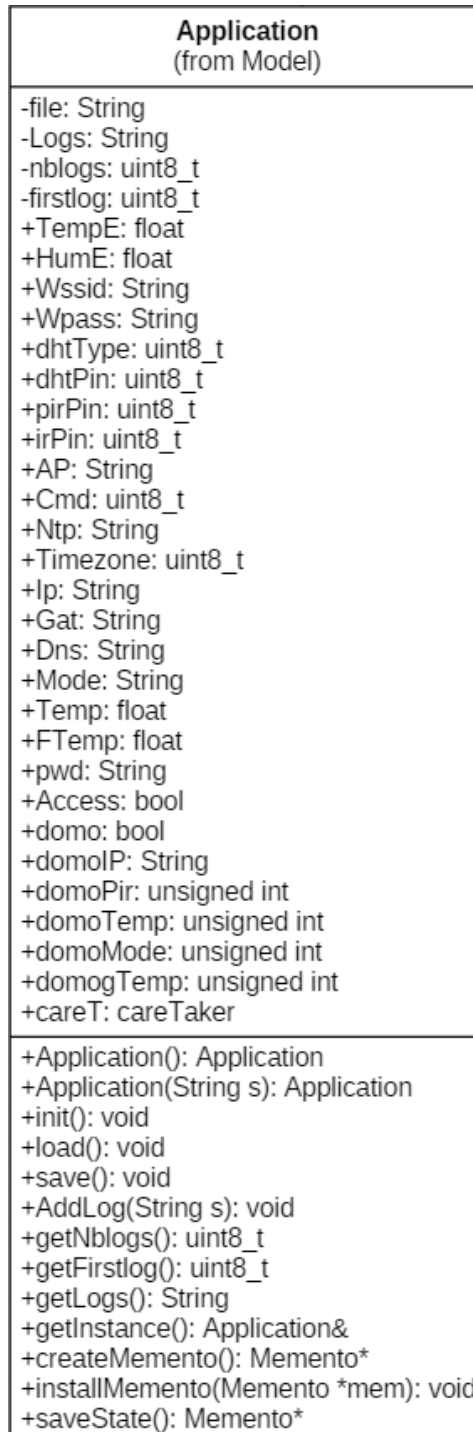
Voici donc le détail des données systèmes à partir du code que nous avons à la base :

```
Sortie
Afficher la sortie à partir de : Micro Build
Compiling debug version of 'WebServerESP8266' for 'WeMos D1 R2 & mini'
Program size: 348 325 bytes (used 33% of a 1 044 464 byte maximum) (5,20 secs)
Minimum Memory Usage: 42164 bytes (51% of a 81920 byte maximum)

Uploading 'WebServerESP8266' to 'WeMos D1 R2 & mini' using 'COM8'
Uploading 352480 bytes from to flash at 0x00000000
.....
The upload process has finished.
```

8.2 Implémentation du Singleton

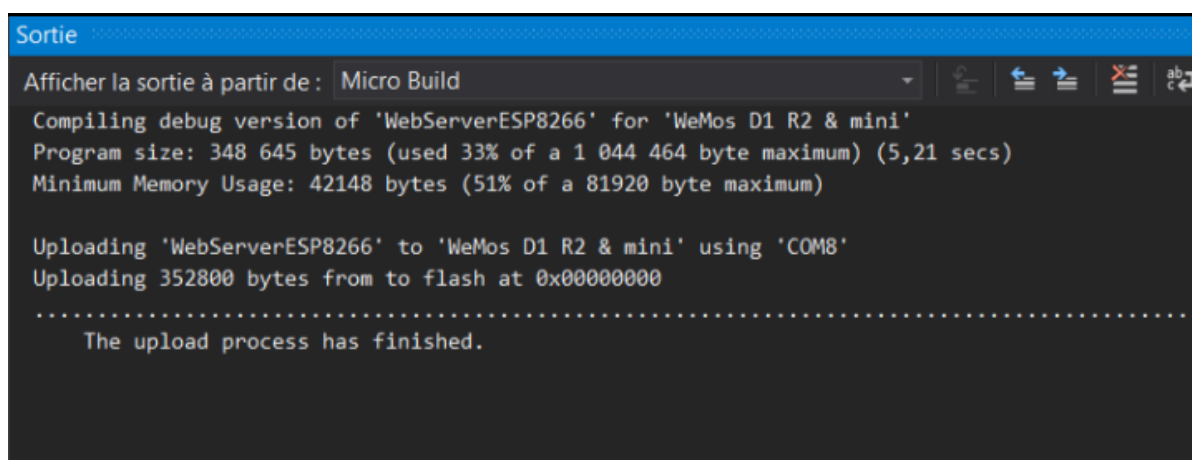
Nous avons donc retiré toutes les instances d'Application qui étaient passée en paramètre de constructeurs ou de méthodes. Il n'existe dorénavant plus qu'une seule et unique instance d'Application. Nous avons donc aussi implémenté la méthode getInstance, permettant de récupérer l'unique existante.



8.2.1 Détails des données système (Singleton)

Si l'on compare les données obtenues après l'implémentation du patron Singleton, on observe que seuls la taille du programme, le temps d'exécution et le nombre de bytes uploadés ont très, très légèrement augmenté. Quand à la taille mémoire, elle a très légèrement diminué. Singleton a donc une implémentation redondante mais reste très léger.

Voici donc le détail des données système après l'implémentation de Singleton :

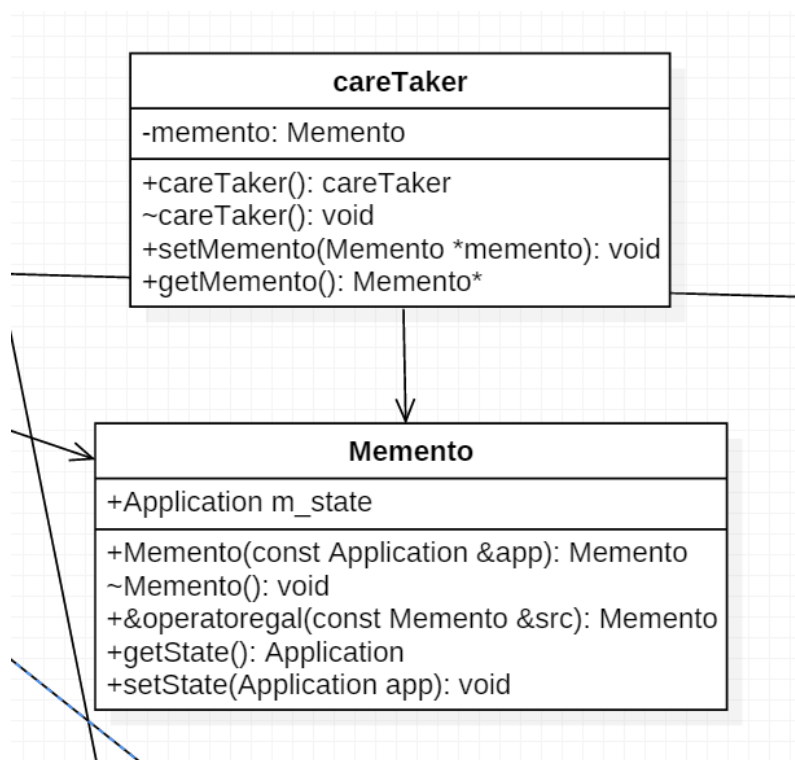
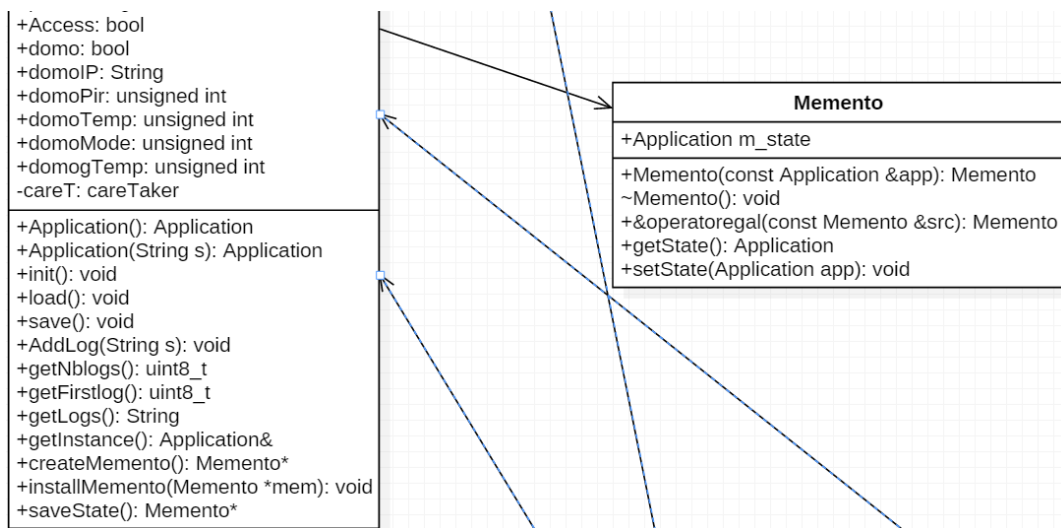


```
Sortie
Afficher la sortie à partir de : Micro Build
Compiling debug version of 'WebServerESP8266' for 'WeMos D1 R2 & mini'
Program size: 348 645 bytes (used 33% of a 1 044 464 byte maximum) (5,21 secs)
Minimum Memory Usage: 42148 bytes (51% of a 81920 byte maximum)

Uploading 'WebServerESP8266' to 'WeMos D1 R2 & mini' using 'COM8'
Uploading 352800 bytes from to flash at 0x00000000
.....
The upload process has finished.
```

8.3 Implémentation du Memento

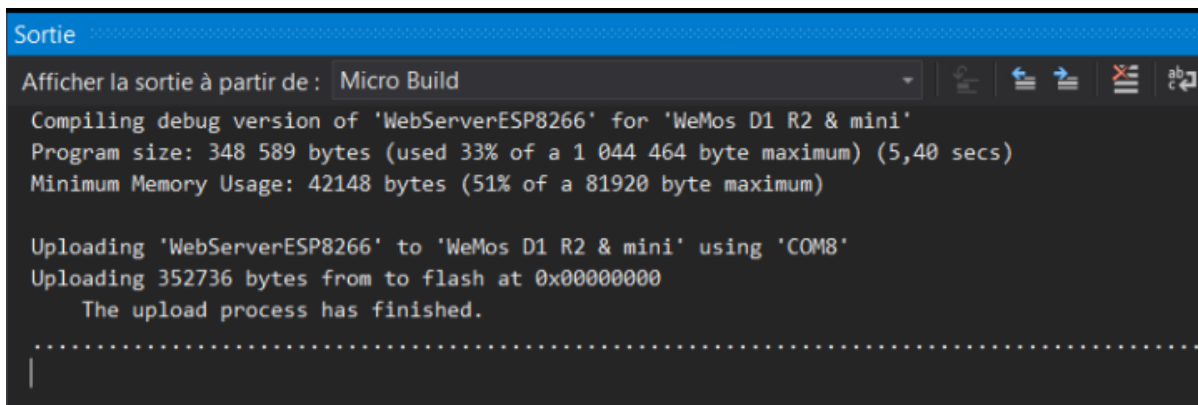
Nous avons donc codé les classes careTaker et Memento. Et nous avons modifié les méthodes save et load de la classe Application dans le but de sauvegarder l'état. Etat qui correspond à la chaîne formatée. Nous pouvions aussi sauvegarder l'instance de la classe, cependant elle contenait plus que ce dont nous avons besoin comme les logs, etc.



8.3.1 Détails des données système (Memento)

Si l'on compare les données obtenues après l'implémentation du patron Memento, on observe que seuls la taille du programme et le nombre de bytes uploadés ont très, très légèrement augmenté. La taille mémoire a elle très légèrement diminué. Quand au temps d'exécution, il a augmenté de 0.20s. Memento a donc une implémentation très légère aussi malgré les 2 nouvelles classes codées.

Voici donc le détail des données système après l'implémentation de Memento :



```
Sortie
Afficher la sortie à partir de : Micro Build
Compiling debug version of 'WebServerESP8266' for 'WeMos D1 R2 & mini'
Program size: 348 589 bytes (used 33% of a 1 044 464 byte maximum) (5,40 secs)
Minimum Memory Usage: 42148 bytes (51% of a 81920 byte maximum)

Uploading 'WebServerESP8266' to 'WeMos D1 R2 & mini' using 'COM8'
Uploading 352736 bytes from to flash at 0x00000000
The upload process has finished.
.....
|
```

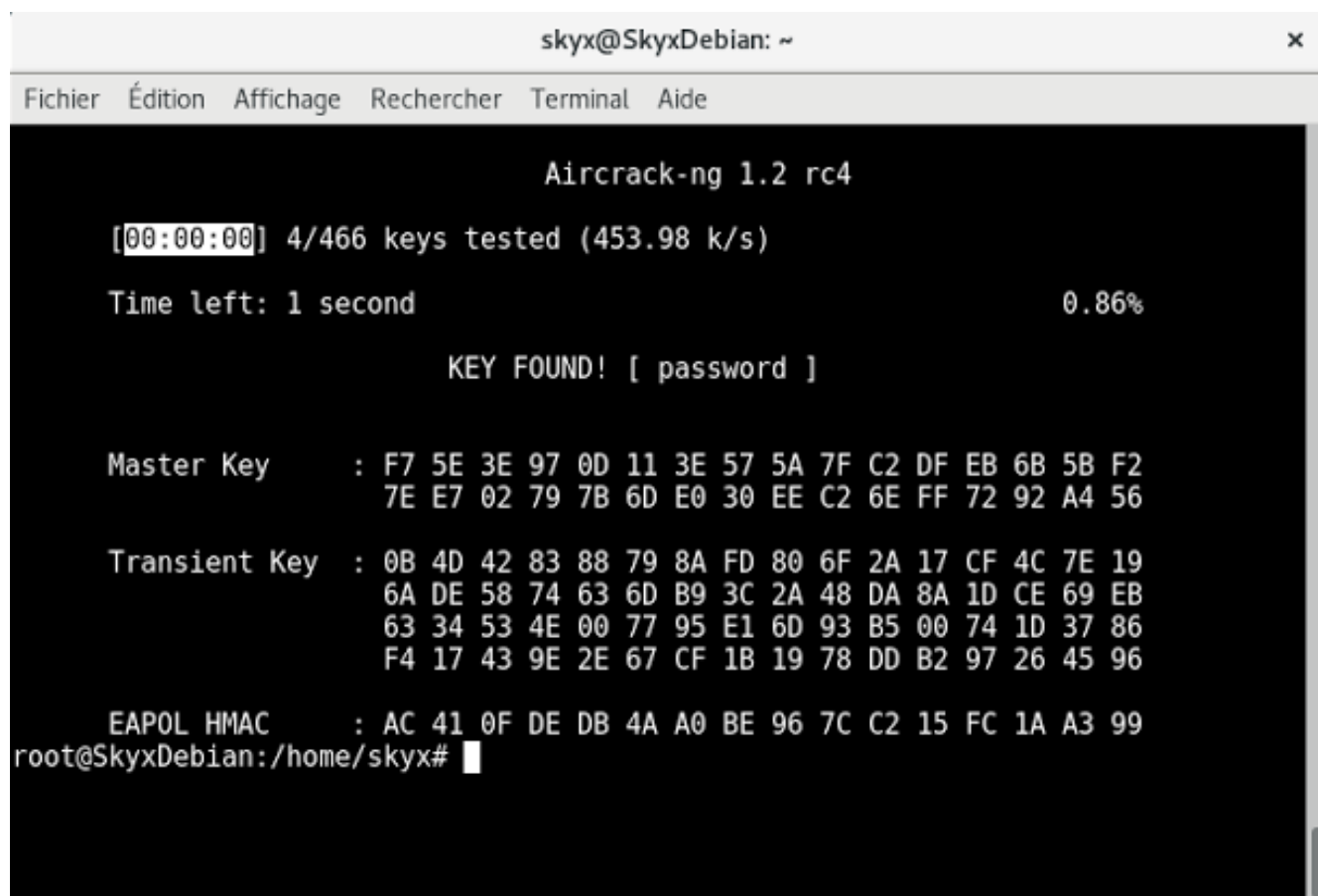
9 Exécution des tests de pénétration

9.1 Utilisation de (Airmon/Airodump/Aircrack)-ng

La suite Air*, fait partie de la distribution Kali Linux, spécialisée dans le PenTesting. Elle est notamment connue pour ses capacités à “casser” les clés WPA/WEP.

Déroulement :

- 1 : Préparation de l'interface WIFI avec Airmon-ng (Mode monitor).
- 2 : Repérage des AP alentours puis récupération de paquets (sniffing) avec Airodump.
- 3 : Envoi de requêtes de désauthentification vers l'AP, pour récupérer le handshake lors de la reconnexion des équipements connectés sur le WIFI. (Il faut connaître l'adresse MAC d'un équipement connecté à l'AP)
- 4 : Attaque par dictionnaires sur le handshake sniffé précédemment.
- 5 : Résultats :



```
skyx@SkyxDebian: ~  
Fichier  Édition  Affichage  Rechercher  Terminal  Aide  
Aircrack-ng 1.2 rc4  
[00:00:00] 4/466 keys tested (453.98 k/s)  
Time left: 1 second 0.86%  
KEY FOUND! [ password ]  
Master Key      : F7 5E 3E 97 0D 11 3E 57 5A 7F C2 DF EB 6B 5B F2  
                  7E E7 02 79 7B 6D E0 30 EE C2 6E FF 72 92 A4 56  
Transient Key   : 0B 4D 42 83 88 79 8A FD 80 6F 2A 17 CF 4C 7E 19  
                  6A DE 58 74 63 6D B9 3C 2A 48 DA 8A 1D CE 69 EB  
                  63 34 53 4E 00 77 95 E1 6D 93 B5 00 74 1D 37 86  
                  F4 17 43 9E 2E 67 CF 1B 19 78 DD B2 97 26 45 96  
EAPOL HMAC      : AC 41 0F DE DB 4A A0 BE 96 7C C2 15 FC 1A A3 99  
root@SkyxDebian:/home/skyx#
```

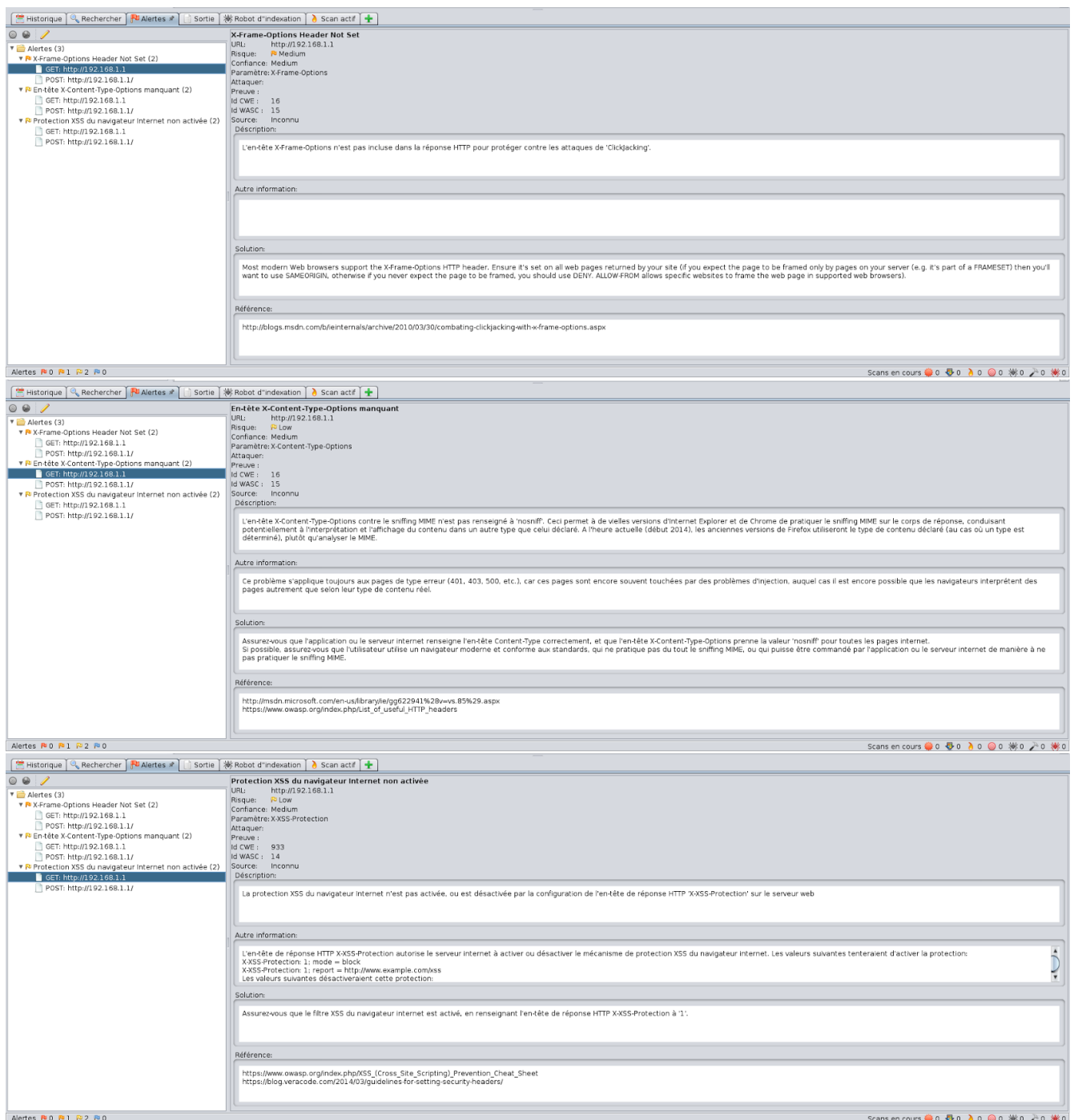
Il est possible de scanner les réseaux alentours avec Angry IP Scanner pour obtenir l'adresse IP du point d'accès puis faire un arping vers l'adresse obtenue pour récupérer une adresse MAC nécessaire à l'attaque.

9.2 Utilisation de Zaproxy

Zed Attack Proxy est un outil de sécurité Open-Source qui permet de trouver automatiquement des failles de sécurité dans des applications Web.

Déroulement :

- 1 : On donne l'adresse web (192.168.1.1) de la carte pour lancer l'attaque.
- 2 : Zaproxy enchaîne les requêtes HTTP pour découvrir des failles.
- 3 : Résultat -> 3 failles trouvées :



The image displays three screenshots of the Zaproxy alert details window, showing the results of a security scan on the target URL <http://192.168.1.1>.

Alert 1: X-Frame-Options Header Not Set

- URL:** <http://192.168.1.1>
- Risque:** Medium
- Confiance:** Medium
- Paramètre:** X-Frame-Options
- Attaquer:**
 - Preuve: 16
 - Id CWE: 15
 - Source: Inconnu
- Description:** L'en-tête X-Frame-Options n'est pas incluse dans la réponse HTTP pour protéger contre les attaques de 'Clickjacking'.
- Autre information:**
- Solution:** Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. ALLOW-FROM allows specific websites to frame the web page in supported web browsers).
- Référence:** <http://blogs.msdn.com/b/enternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx>

Alert 2: En-tête X-Content-Type-Options manquant

- URL:** <http://192.168.1.1>
- Risque:** Low
- Confiance:** Medium
- Paramètre:** X-Content-Type-Options
- Attaquer:**
 - Preuve: 16
 - Id CWE: 15
 - Source: Inconnu
- Description:** L'en-tête X-Content-Type-Options contre le sniffing MIME n'est pas renseigné à 'nosniff'. Ceci permet à de vieilles versions d'Internet Explorer et de Chrome de pratiquer le sniffing MIME sur le corps de réponse, conduisant potentiellement à l'interprétation et l'affichage du contenu dans un autre type que celui déclaré. A l'heure actuelle (début 2014), les anciennes versions de Firefox utiliseront le type de contenu déclaré (au cas où un type est déterminé), plutôt qu'analyser le MIME.
- Autre information:** Ce problème s'applique toujours aux pages de type erreur (401, 403, 500, etc.), car ces pages sont encore souvent touchées par des problèmes d'injection, auquel cas il est encore possible que les navigateurs interprètent des pages autrement que selon leur type de contenu réel.
- Solution:** Assurez-vous que l'application ou le serveur internet renseigne l'en-tête Content-Type correctement, et que l'en-tête X-Content-Type-Options prenne la valeur 'nosniff' pour toutes les pages internet. Si possible, assurez-vous que l'utilisateur utilise un navigateur moderne et conforme aux standards, qui ne pratique pas du tout le sniffing MIME, ou qui puisse être commandé par l'application ou le serveur internet de manière à ne pas pratiquer le sniffing MIME.
- Référence:** <http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx>
https://www.owasp.org/index.php/List_of_useful_HTTP_headers

Alert 3: Protection XSS du navigateur internet non activée

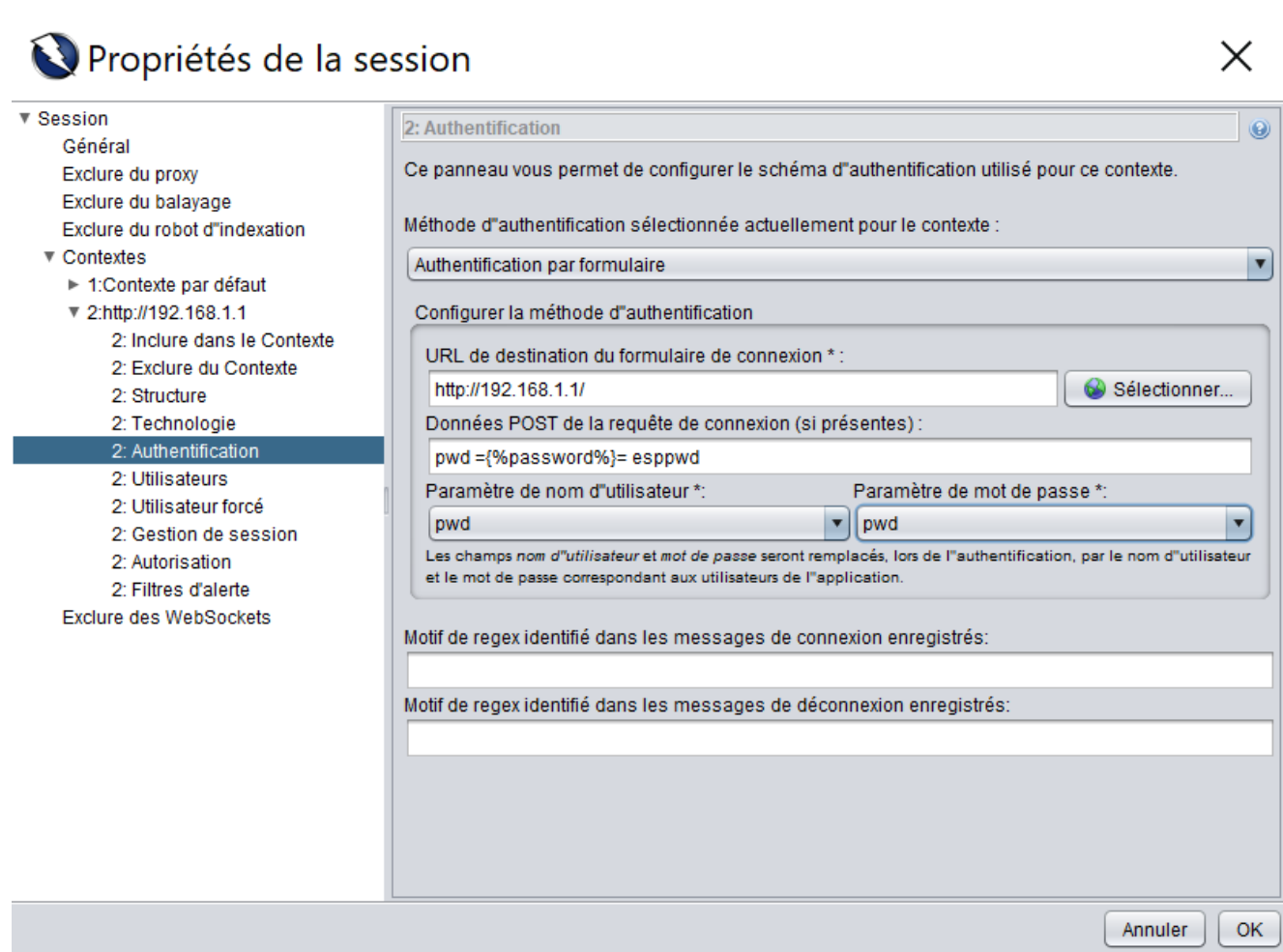
- URL:** <http://192.168.1.1>
- Risque:** Low
- Confiance:** Medium
- Paramètre:** X-XSS-Protection
- Attaquer:**
 - Preuve: 933
 - Id CWE: 14
 - Source: Inconnu
- Description:** La protection XSS du navigateur internet n'est pas activée, ou est désactivée par la configuration de l'en-tête de réponse HTTP 'X-XSS-Protection' sur le serveur web.
- Autre information:** L'en-tête de réponse HTTP X-XSS-Protection autorise le serveur internet à activer ou désactiver le mécanisme de protection XSS du navigateur internet. Les valeurs suivantes tenteraient d'activer la protection: X-XSS-Protection: 1; mode = block
X-XSS-Protection: 1; report = <http://www.example.com/xss>
Les valeurs suivantes désactiveraient cette protection:
- Solution:** Assurez-vous que le filtre XSS du navigateur internet est activé, en renseignant l'en-tête de réponse HTTP X-XSS-Protection à '1'.
- Référence:** [https://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)
<https://blog.veracode.com/2014/03/guidelines-for-setting-security-headers/>

9.2.1 Précisions

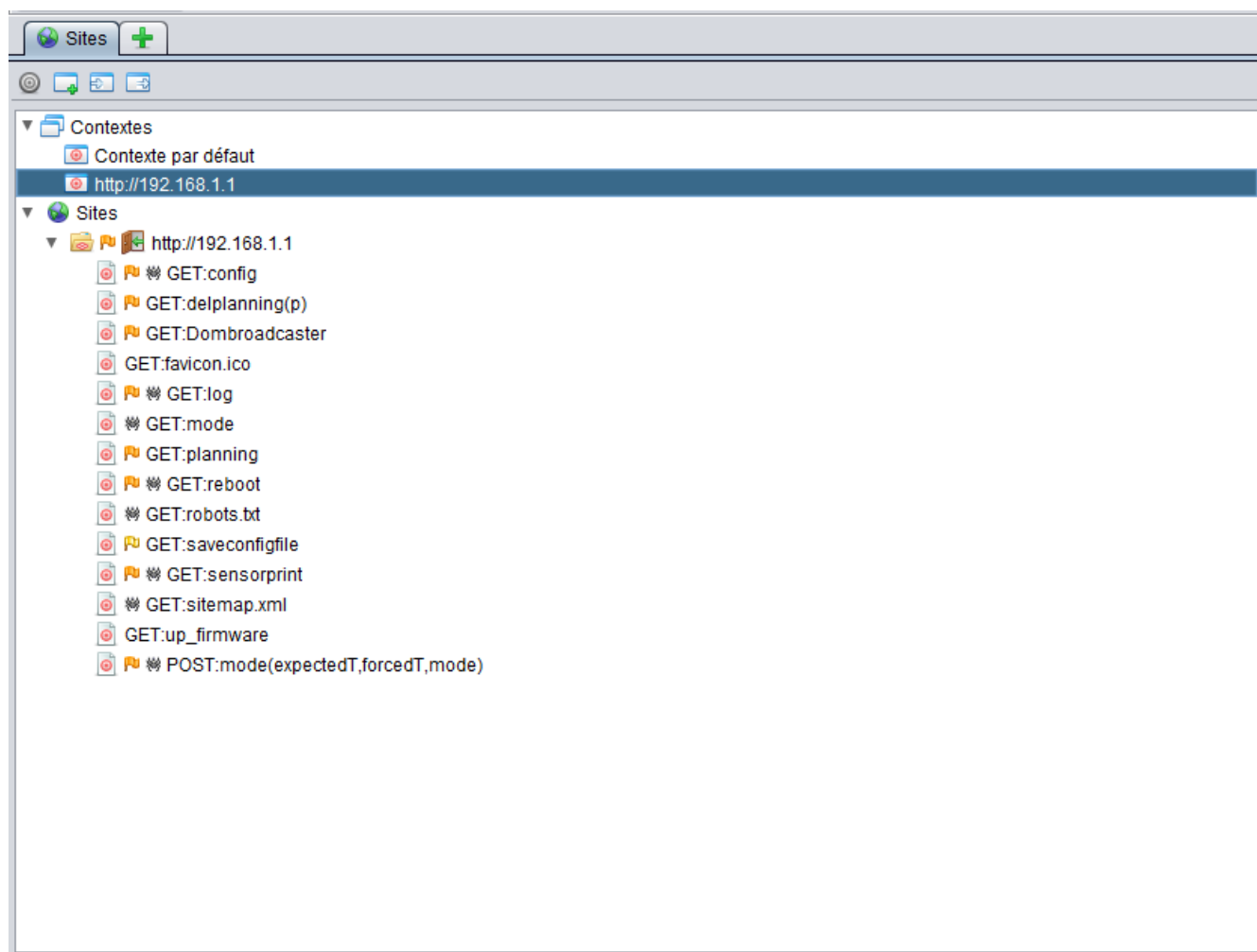
Quelques informations supplémentaires sur l'utilisation de ZaProxy.

Approfondissements :

1 : On peut donner un contexte à ZaProxy, c'est-à-dire des identifiants qu'il peut tester sur le site.



2 : De cette manière, AJAX Spider peut parcourir le site de façon plus ample. Voilà un exemple du parcours plus ample :



3 : Résultat : Une requête reboot réinitialise le système, détruisant les fichiers de config et permettant donc de se connecter à la carte sans mot de passe.

Open Thermostat [Main](#) [Config](#) [Sensor State](#) [Log&Info](#) [Time 0:0](#)

No password set!

Wifi Info
Wifi SSID:

Thermostat

Mode	Off ▼
Current Temperature	-1.00
Expected Temperature	<input type="text" value="0.00"/>
Forced Temperature	<input type="text" value="0.00"/>

Misc

9.3 Présentation de Krack-Attack

Krack-Attack (Key Reinstallation Attack) se base sur une faille de sécurité du protocole WIFI WPA2. Lors de l'échange de clés (handshake), il est possible de forcer une réinstallation de la clé, alors composée uniquement de 0. On peut par conséquent écraser la clé WIFI actuelle.

Déroulement :

- 1** : Utilisation de krackattacks-scripts (<https://github.com/vanhoefm/krackattacks-scripts>) qui est un ensemble de scripts permettant la détection de vulnérabilités relatives à la faille WPA2 et leur exploitation.
- 2** : Il suffit de désactiver le cryptage matériel, le WIFI et de lancer une commande permettant au script d'utiliser le WIFI.
- 3** : Par la suite, on crée un fichier de conf wpa-supplisant en donnant les données de notre borne à tester pour faciliter la connexion.
- 4** : L'étape suivante consiste à exécuter le script principal en passant en paramètre une commande wpa-supplisant pour la connexion. Celui-ci va se connecter à la borne et effectuer les tests.
- 5** : Malheureusement, à l'heure actuelle, lors des tests, le script bloque et n'effectue pas les tests pour une raison encore inconnue.
- 6** : Cependant, il est fort probable que le protocole n'est pas été encore patché sur l'IOT et l'expose donc à cette vulnérabilité encore assez peu corrigé.

9.4 Burpsuite

Burp Suite est une application Java qui peut être utilisée pour la sécurisation ou effectuer des tests de pénétration sur les applications web. La suite est composée de différents outils comme un serveur proxy (Burp Proxy), robot d'indexation (Burp Spider), un outil d'intrusion (Burp Intruder), un scanner de vulnérabilités (Burp Scanner) et un répéteur HTTP (Burp Repeater).

Objectif : Cracker le mot de passe de l'interface Web.

Démarche :

BurpSuite crée un proxy, que l'on fait utiliser par le navigateur. Ce proxy lui permet d'intercepter les requêtes envoyées et reçues par le navigateur. En interceptant une requête de connexion, on peut ensuite marquer le champ contenant le mot de passe, et lancer une attaque, qui va renvoyer cette requête avec des mots de passe différent. Dans notre cas, une attaque en bruteforce sur les caractères alphanumériques a été lancée, uniquement sur des mots a 4 caractères. A l'ordinaire, BurpSuite trie les résultats en fonction de leur code de retour, mais notre application renvoie 200, que le mot de passe soit bon ou mauvais. J'ai du rajouter un filtre qui recherchait la chaîne "Wrong password" dans la réponse de l'objet, afin de sélectionner les bonnes réponses.

Résultats :

BurpSuite a donc trouvé le mot de passe "sasa", mais a apparemment réussi a accéder a l'interface avec de nombreux autres mot de passe (11067 pour être exact), mais en en testant une dizaine a la main, aucun d'eux ne fut une réussite, je pense donc a un bug ou une mauvaise configuration. J'ai d'abord pensé au fait qu'une fois le mot de passe trouvé, les résultat suivants furent eux aussi concluant, mais il y a un mot de passe qui fut testé avant "sasa" qui a marché (d'après BurpSuite), et certains ont été testé après sans pour autant marcher. Une partie de la liste des mot de passe testés est visible ici. (j'ai arrêté le bruteforce en voyant qu'il avait trouvé le mot de passe) Dans tout les cas, le mot de passe a été trouvé au bout de quelques heures seulement. Certes le bruteforce a été effectué sur un ordinateur portable milieu/bas de gamme, mais de toute façon la puissance de la carte limite l'utilisation du bruteforce, qui a été effectué sur 1 thread seulement.

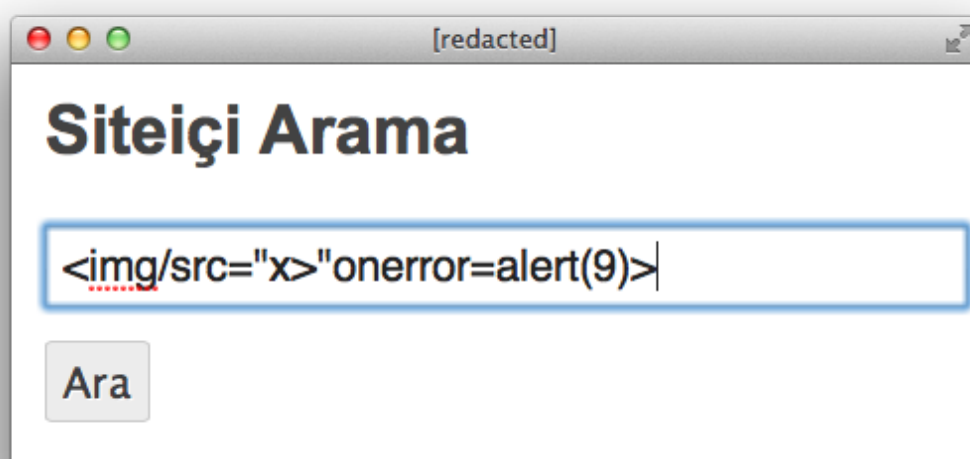
9.5 Injections HTML JavaScript

Tous les champs de l'application Web ne sont pas protégés. Il est donc possible d'injecter du code (HTML/JavaScript) dans ces derniers.

(ex : "><script>alert('hello')</script>") Cet exemple est basique et bénin, mais cette faille peut permettre à un utilisateur d'afficher images, texte comme il le souhaite, mais aussi exécuter des scripts JavaScript avec de mauvaises intentions, ou encore de récupérer des informations sur les utilisateurs (cookies phpsessionid, etc).

Objectif : Cracker le mot de passe de l'interface Web.

Solutions : Il est nécessaire de protéger ces champs, une solution potentielle est le patron d'Input Guard. D'autres solutions impliquent l'utilisation de PHP pour vérifier les inputs rentrés par l'utilisateurs avec des classes comme validation et nettoyage.



10 Patrons de sécurité

10.1 Patron "Input Guard"

C'est un patron de sécurité que l'on place à chaque point d'accès d'un composant dans le but de vérifier la validité de l'entrée.

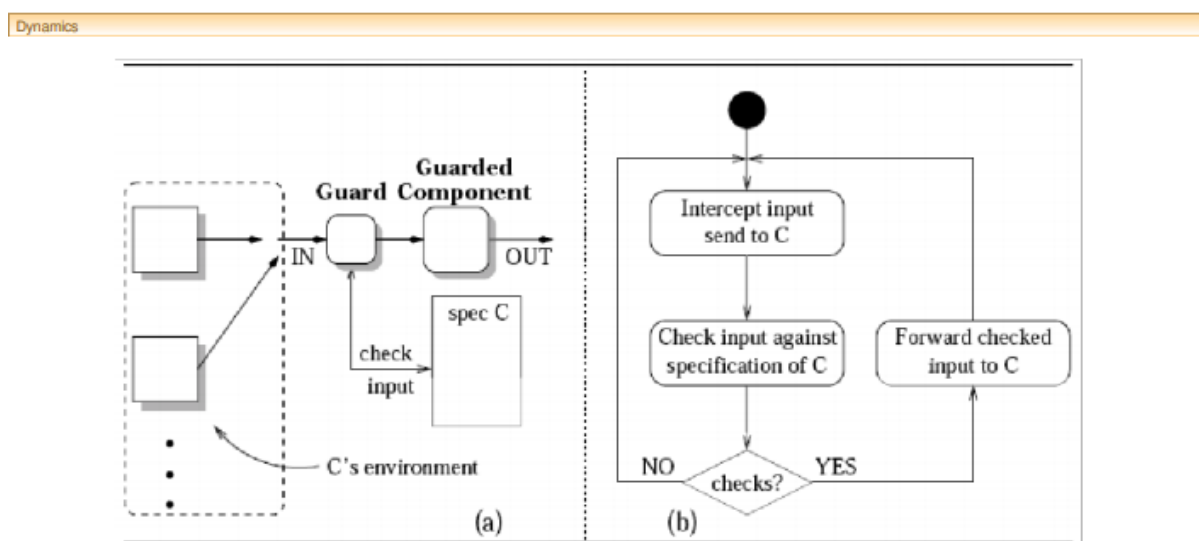
Fonction : Protéger le système contre des inputs non conforme aux specifications demandées.

Analyse : D'après la documentation fournie, le patron Input Guard s'appliquent aux systèmes qui possèdent les caractéristiques suivantes :

1 : Le système est composé de composants distincts, qui peuvent chacun être fautif et qui interagissent entre eux en input/output.

2 : Les erreurs qui peuvent être propagées dans un composant système ont la forme d'inputs erronés. Les erreurs internes (causées par des perturbations électromagnétiques) ne sont pas prises en charge par ce patron car pas sous la forme d'un input erroné.

Notre application correspond à cette description, car il y a de nombreux endroits où l'utilisateur peut entrer du texte, pouvant aboutir à de l'injection de code, etc... La récupération automatique de l'heure (ntp) et les services REST font aussi partie de l'input récupéré. Par exemple, l'utilisateur peut configurer la connexion wifi de l'appareil, et rentrant des valeurs comme "><script>alert(" hello ")</script> ", il peut injecter du code malicieux dans la page, qui sera exécuté lors de l'affichage de la page. Il est donc nécessaire de protéger les input/output, afin d'éviter ce genre de problèmes. Le patron Input Guard pourrait donc être utile à notre application.



3 Implementations de ce patrons sont possibles :

1 : Le garde est implemente comme un nouveau composant du systeme, et son nombre est proportionnel aux nombres de points d'accès a proteger. Cet maniere rajoute du temps d'execution et augmente l'utilisation memoire car l'invocation de garde pour chaque point d'accès est couteux

2 : Le garde est implemente dans le composant a garder. Cet methode est couteuse en temps d'execution car pour rendre le systeme robuste, on check l'input a chaque appel du composant a garder.

3 : Le garde est implemente dans le composant qui envoie l'input. Le nombre d'appels du garde est proportionnel aux nombres de composant qui envoient des donnees au composant a garder. Le probleme est que les garde ne connaissent pas forcement les specifications du composant a garder afin de traiter l'input correctement. Les erreurs de communication ne sont pas geres non plus, car le check de l'input se fait avant l'envoi de celui-ci.

Dans le cadre de notre projet, les inputs sont recuperes/utilises dans les classes : Web et NTP. Pour faciliter l'utilisation de ce patron avec le Single Access Point, il est preferable d'utiliser la 1ere methode. En creant une classe ne contenant que des methodes statiques, on reduit ainsi l'utilisation memoire, et cette methode d'implementation est preferable a la 2eme, car la classe Web est suffisamment lourde et sa responsabilite n'est pas de nettoyer/valider l'input qu'elle recoit.

Notre implementation : La creation d'une classe InputGuard est necessaire, son role sera de traiter l'input avant de l'envoyer au composant a garder. Elle devra utiliser les specifications du composant qui recevra l'input traite. Ces specifications seront contenues dans une classe a part. Deux classes utiliseront l'inputguard, mais leurs specifications sont les memes, car elle sont basees sur le protocole HTTP, ce qui permet d'eviter de creer une classe de specifications supplementaire.

10.2 Patron "Secure Logger"

C'est un patron de sécurité permettant de centraliser et de sécuriser les logs.

Fonctions : Rassembler les log dans un seul endroit et rendre ses logs impossible a alterer.

Analyse :

-> Ce patron n'est utile que si :

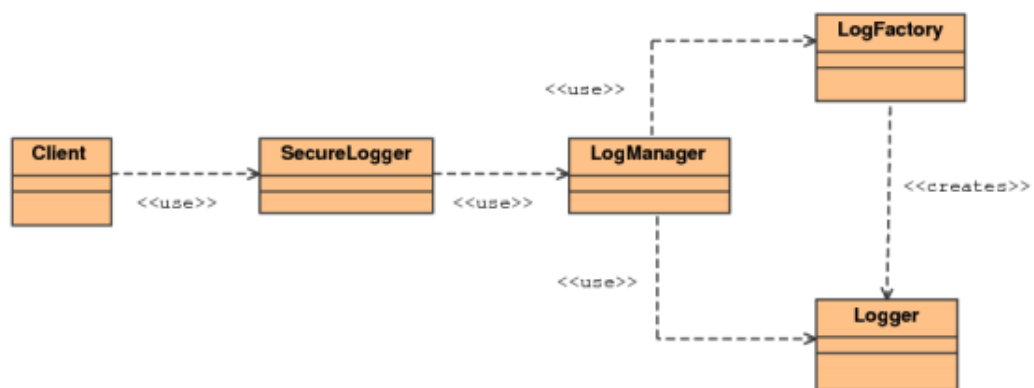
1 : On a besoin de log des informations sensibles qui ne doivent pas être accessibles à tous.

2 : On doit pouvoir être sûr que le sdonnées loggées sont entières et n'ont pas pû être altérées par un utilisateur malicieux.

3 : On peut centraliser le système de log pour améliorer la gestion de l'application. **4 :** On veut crypter les données du log pour en assurer l'intégrité et la sécurité.

Dans notre application, les donnees qui sont loggees ne contiennent pas d'informations sensible. Le systeme de log est deja bien centralise (passage par la classe Application). Surtout, l'implementation de ce patron impliquerait la creation d'au minimum 4 classes, ce qui serait trop lourd pour notre carte, avec un apport minime sur le fonctionnement.

Structure



10.3 Patron "Single Access Point"

C'est un patron de sécurité permettant de protéger le système et certaines de ses transactions.

But : Apporter un point d'accès unique à l'ensemble du modèle pour l'utilisateur. Cela permet de protéger le système et ses sous-systèmes entièrement de l'extérieur. Il permet de sécuriser la connexion de l'utilisateur au système et gère les autorisations d'accès au modèle.

Utilité :

Ce système, facilement implémentable, est très utile dans une application dont le lancement ne se fait que d'une seule manière. Ce point d'accès à l'application peut permettre de contrôler l'initialisation de toutes les variables (e.g configs, etc ..). C'est ce point d'accès unique, qui contrôle si un utilisateur peut se connecter, peut se coupler avec un pattern tel que Input Guard pour vérifier les credentials d'un utilisateur. En revanche, il rend l'accès à l'application moins flexible. Dans notre cas, il pourrait permettre l'encapsulation complète du modèle pour l'extérieur en gérant les requêtes vers le modèle. En pratique, toutes les classes auraient leur constructeur privé, et le SAP serait friend de toutes les classes. Ainsi, personne à part le SAP ne pourrait instancier les classes, il s'assurerait ainsi de leur bonne initialisation et renverrait la référence de l'objet. Une requête faite pour accéder à la classe Web par exemple serait nettoyé puis validé si elle contient des inputs puis instancierait la classe et la renverrait de telle manière qu'on puisse la manipuler par la suite via ses méthodes publiques. On ne pourrait donc pas instancier la classes à l'extérieur du modèle sans passer par le SAP en ayant son autorisation.

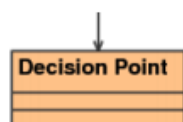
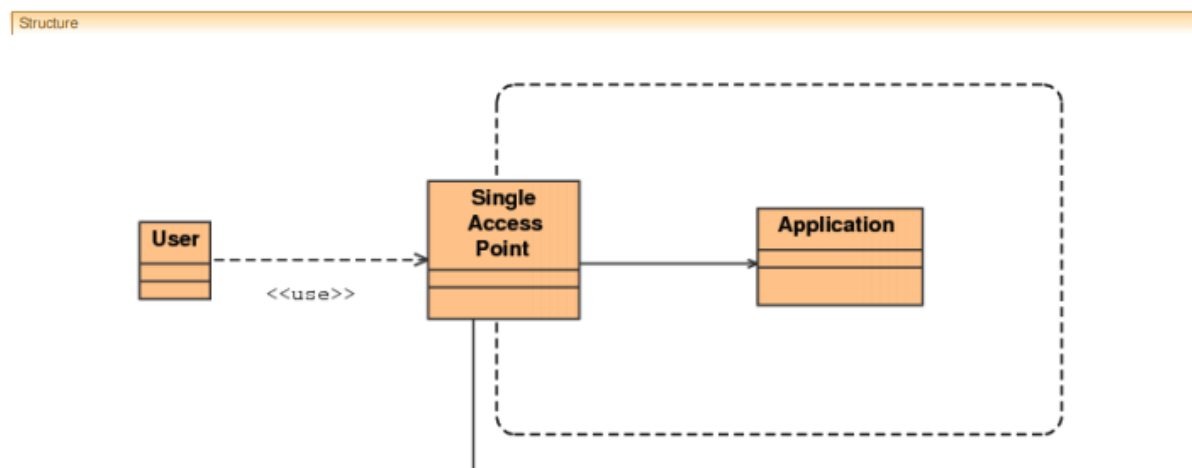
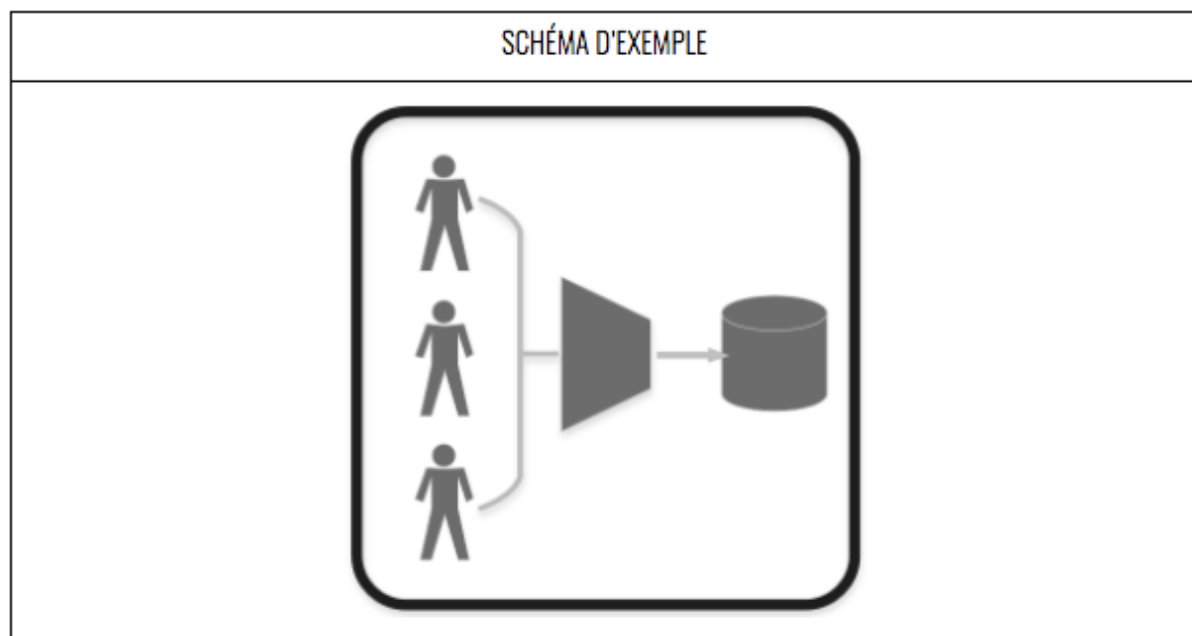


Figure 1: Single Access Point structure.

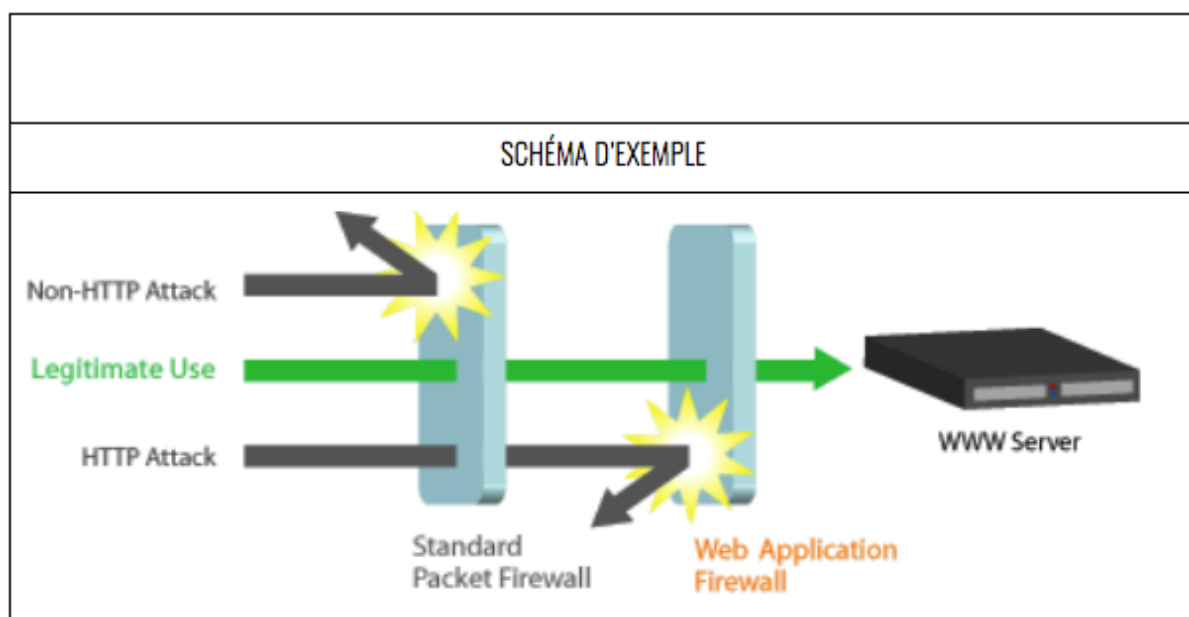


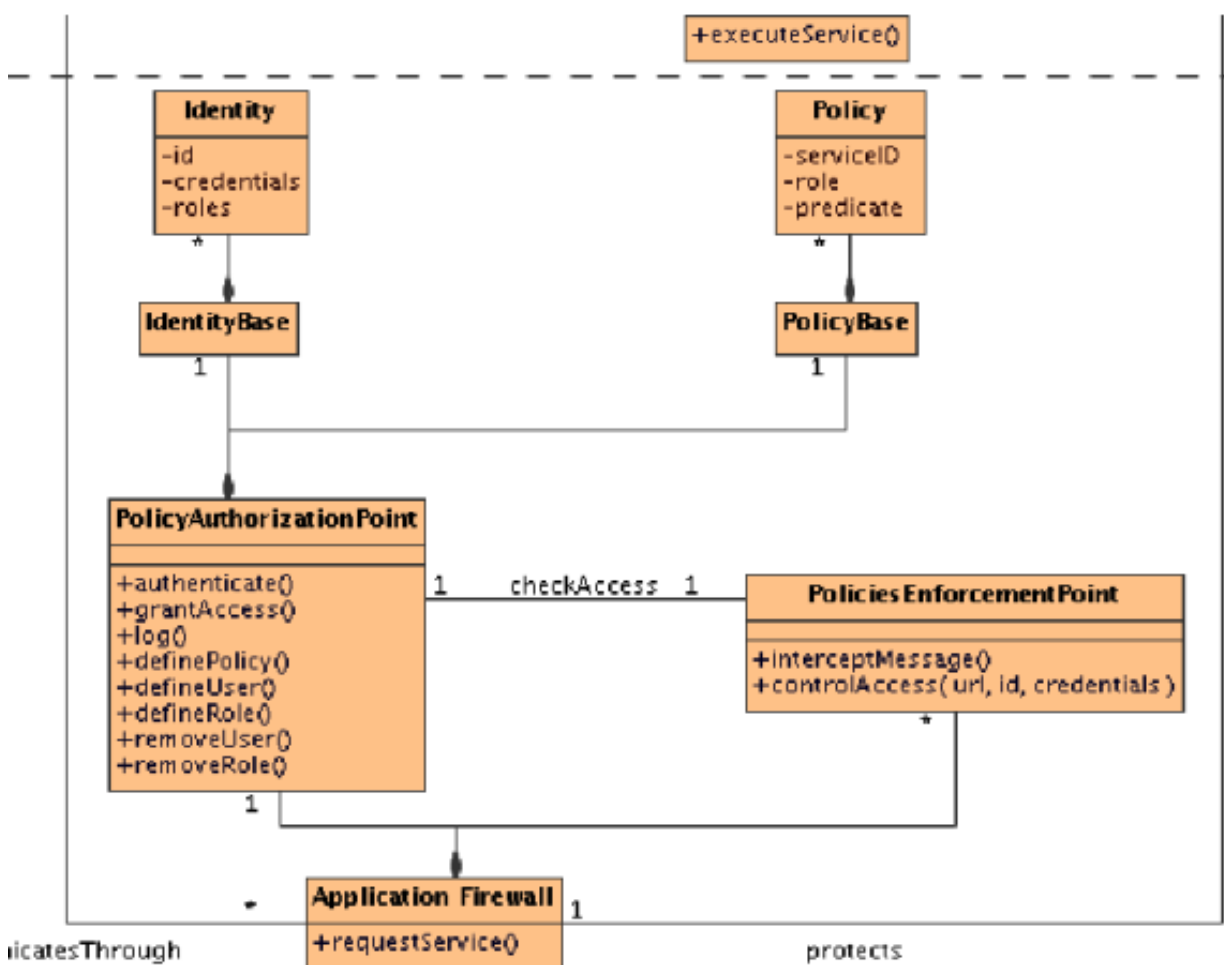
10.4 Patron "Application firewall"

C'est un patron de sécurité permettant de contrôler la circulation des données en fonction de normes prédéfinies.

But : Pare-feu contrôlant les entrées/sorties de l'application selon des règles établies (politiques) et les utilisateurs.

Utilité : Un tel système est intéressant pour notre IOT, car il permet de filtrer les entrées/sorties d'informations et de différencier plusieurs rôles d'utilisateurs. De plus, il protège bien des diverses attaques envers l'IOT rendant le système plus protégé des pirates. En revanche, son implémentation est lourde et irréalisable sur un si petit modèle de carte (nombre d'instances trop grand).



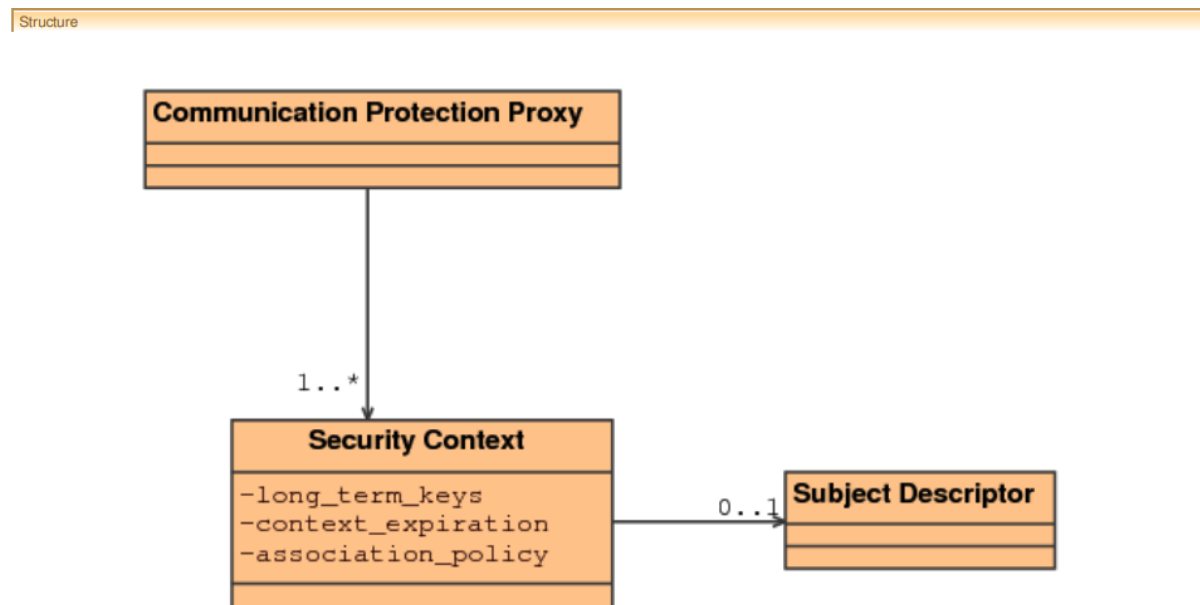


10.5 Patron "Security context"

C'est un patron de sécurité permettant d'encapsuler les attributs de sécurité ainsi que les données relatant d'exécutions particulières.

But : Ajoute une couche d'abstraction sur un objet dont on souhaite obtenir un accès régulier mais protégé. Le pattern prend tout son sens dans une situation où un objet interagit avec plusieurs autres pour former le contexte de l'application. La couche d'abstraction décrite plus haut, permet de stocker des données relatives à l'identification de l'utilisateur telles des timers d'expiration pour la session ou des clés d'identification.

Utilité : Permet l'encapsulation de l'objet dans un autre qui contient les clés cryptographique, les timers et tous autres attributs relatifs à la sécurité de l'objet. Il permet l'instauration d'un point d'accès qui contrôle les autorisations et protège les données sensibles. Ce security pattern est un dérivé du Proxy pattern en conception. Il permet à une classe de se faire passer pour une autre en ajoutant une protection à celui-ci. On appelle cette encapsulation le "contexte de sécurité". Ce pattern assez léger, peut permettre une authentification par token d'accès aux objets dont WebServer.ino a la responsabilité et permettre d'ajouter une couche d'abstraction au modèle et obtenir une API OpenThermostat propre et réutilisable. Il pourrait donc être facilement implémenter en rendant les constructeurs et méthodes de chaque classe privés et la couche d'abstraction friend de chacune d'elles.

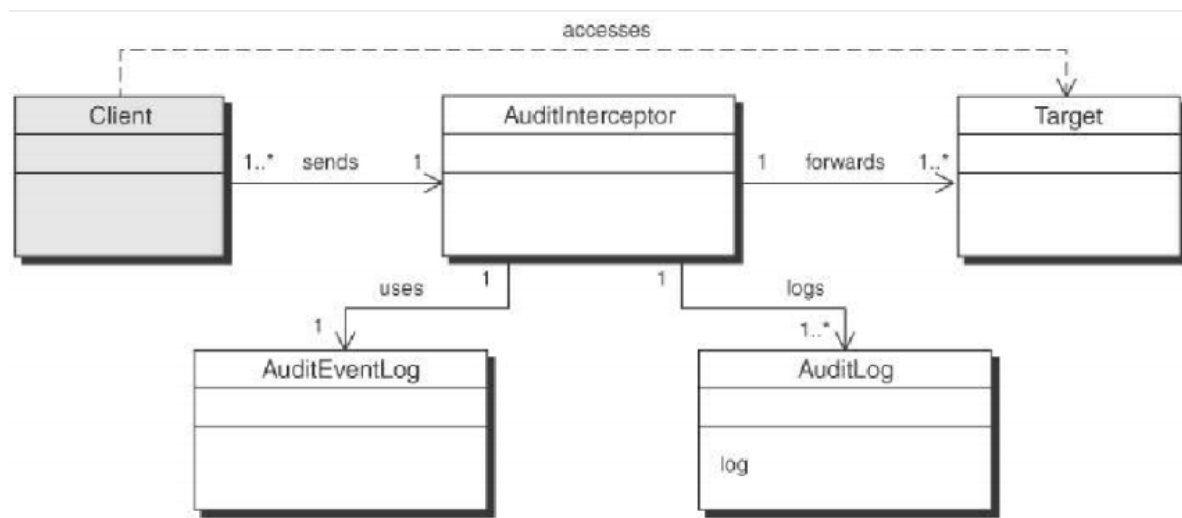


10.6 Patron "Audit Interceptor"

C'est un patron de sécurité qui attrape et vérifie les requêtes et réponses.

But : Crée un intermédiaire entre le client et la cible de la requête sans que le client ne connaisse son existence. Il permet au client d'accéder aux différents services sans que celui-ci n'ait à prendre en compte les différentes API ou interfaces permettant de communiquer avec le système et offre un event tracking des requêtes et réponses transitant entre le client et le serveur. Ce pattern offre de la flexibilité côté développement du système pour gérer les accès et offre une certaine rigidité pour le client qui lui évite des mises à jours constantes lors de modification (si on parle d'une API publique).

Utilité : Un tel pattern requiert une gestion des messages par flux ce qui n'est pas le cas actuellement dans notre application. Coder un tel système revient à coder une couche entière d'interception de messages qui agirait en amont du WebServer. L'évent tracking que fournit ce pattern peut s'avérer intéressant pour logger les éventuels accès suspects ou les attaques sur la carte voire les différentes modifications apportées à OpenThermostat. Il reste cependant à déterminer si rajouter une couche de lecture des requêtes et réponses HTTP NTP ne va pas ralentir l'interface web et son fonctionnement global surtout si celui-ci est couplé à d'autres patterns de sécurité qui peuvent nécessiter du traitement de requêtes HTTP ou NTP.

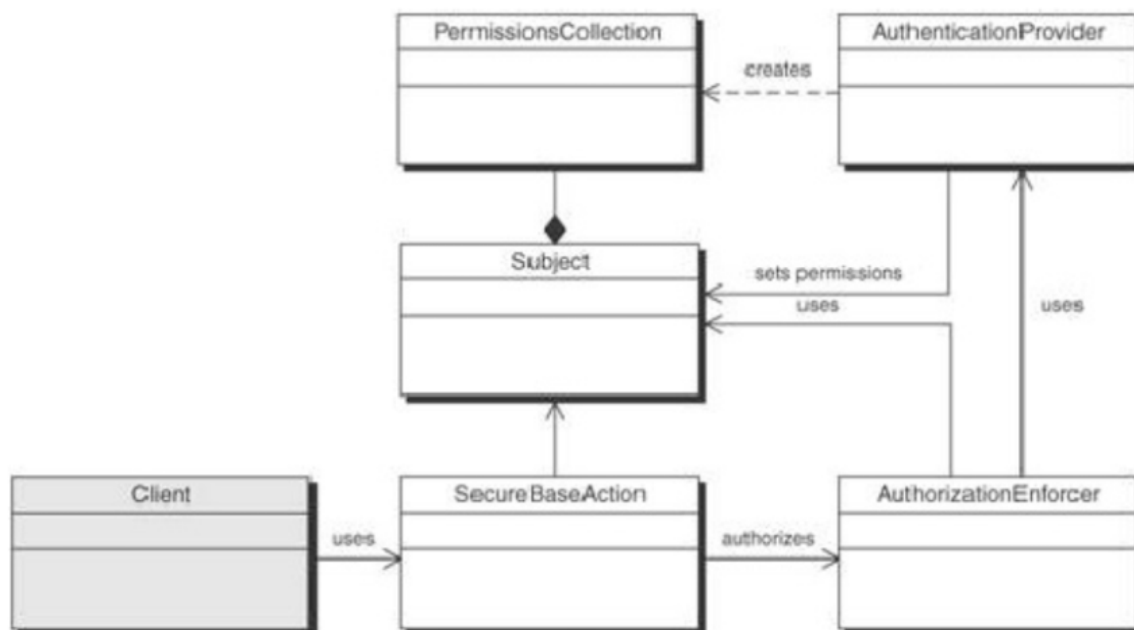


10.7 Patron "Authentication Enforcer"

C'est un patron de sécurité qui centralise l'authentification et encapsule les détails de son mécanisme.

But : Ce pattern permet de gérer différents rôles utilisateurs et leur accès aux ressources de l'application. Il est particulièrement utile pour la gestion d'accès aux URLs d'un site web. En application, ce pattern agit au niveau des méthodes du modèle et permet l'autorisation d'accès même lorsqu'un composant du système peut être accédé de plusieurs endroits.

Utilité : Ce patron de sécurité est en pratique facilement implémentable et plutôt simple à coder. Pour notre projet, il ne semble pas adapté puisque l'interface web ne dispose pas de plusieurs rôles. En effet, l'accès est uni-directionnel pour les composants du système et se fait par autorisation de connexion directe. Il n'y a pas de distinction de rôles étant donné qu'un utilisateur doit avoir accès à toutes les ressources de l'interface pour les paramétrer.

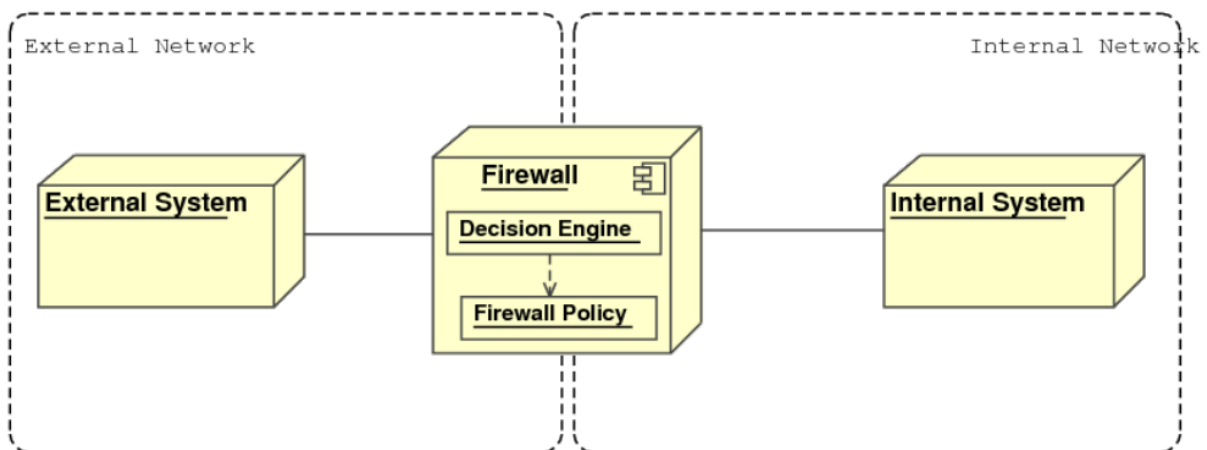


10.8 Patron "Firewall"

C'est un patron de sécurité qui joue un rôle d'administrateur, de filtre. On peut prendre l'exemple d'un garde au sein d'une prison, il peut examiner chacune des lettres qui transitent et donc décider si elles sont aptes à passer ou non.

But : Contrôle les connections réseaux entrantes et sortantes, restreint l'accès à certains hôtes au niveau du réseau.

Utilité : Ce patron n'est applicable que sur un réseau de systèmes afin de gérer leur accès, il ne concerne donc pas notre application.



10.9 Patron "Output Guard"

C'est un patron de sécurité qui, contrairement à l'input guard, est souvent utilisé car il est parfois souhaitable de stopper l'erreur lorsqu'elle se produit plutôt que lorsqu'elle est sur le point de se propager.

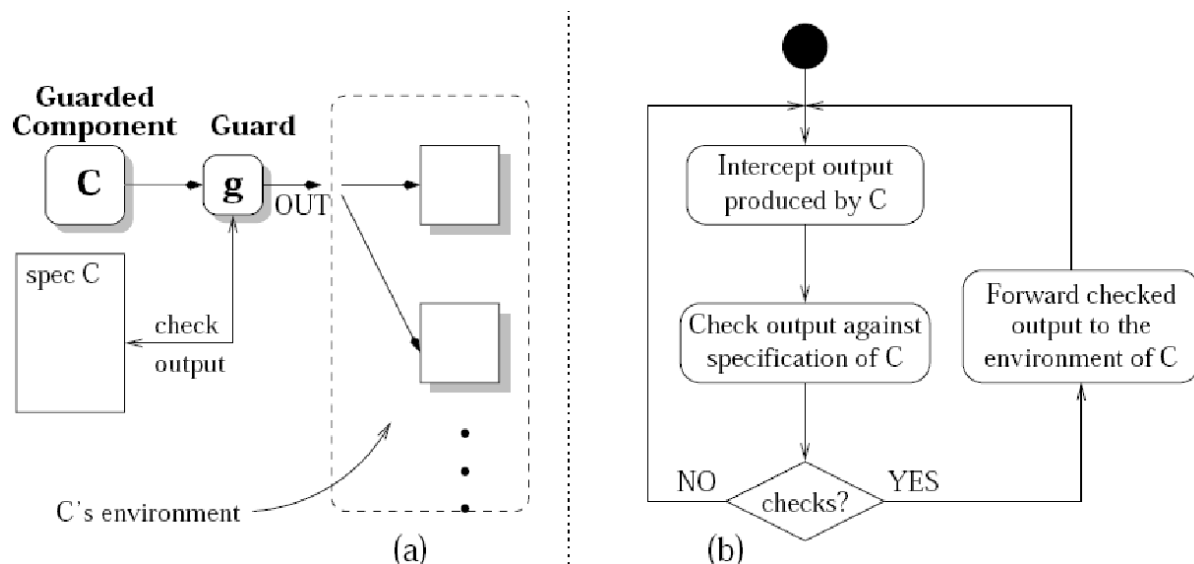
But : Confine les erreurs dans les composants qui en sont à l'origine.

Utilité : L'Input Guard prévient les erreurs d'input pour ne pas contaminer les composants cibles, l'Output Guard confine les erreurs dans le composant qui les a créées, empêchant ainsi leur propagation.

D'après la documentation fournie, le patron Output Guard s'applique aux systèmes qui possèdent les caractéristiques suivantes :

1. Le système est composé de composants distincts, qui peuvent chacun être fautif et qui interagissent entre eux en input/output.
2. Les erreurs qui se produisent dans le système sont exprimées comme une sortie de système erronée selon ses spécifications.

Dans notre système, peu de composants pourraient produire des outputs erronées, surtout depuis l'implémentation de l'Input Guard. L'Output Guard serait une complication supplémentaire sans réel avantage à l'utilisation.



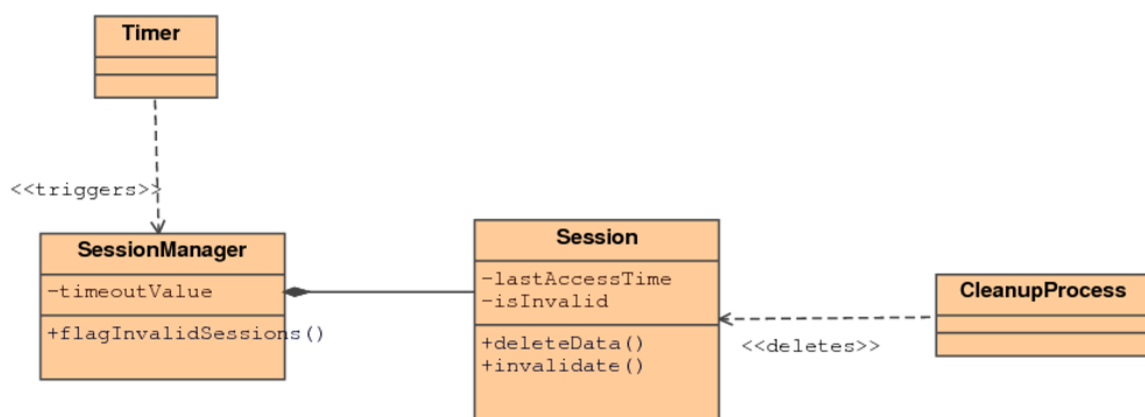
10.10 Patron "TimeOut"

Il dépend du patron session. Ce modèle étend le modèle de session et est nécessaire lorsque les données de session sont conservées sur le serveur (pour limiter la quantité de données sur le serveur). Il requiert aussi des méthode bien définies pour invalider une session et pour supprimer ses données. Ces méthodes peuvent également être utilisées lorsque le client se déconnecte explicitement.

But : Faire un gestionnaire de session. Périodiquement, il parcourra toutes les sessions du système à la recherche de sessions qui n'ont pas été consultées depuis plus d'un certain temps. Lorsqu'une telle session est trouvée, elle est immédiatement signalée comme non valide, elle ne sera donc plus utilisée.

A savoir : Il est là pour empêcher le système de manquer de ressources car les sessions abandonnées ne sont pas nettoyées. Cependant, les utilisateurs peuvent abandonner leurs sessions sans que le système en soit averti. L'utilisateur peut ne pas signaler au système qu'il n'a plus l'intention d'accéder à la session (déconnexion). Lorsqu'une session n'a pas été utilisée pendant un délai fixe moyen on peut considérer la session comme abandonnée avec une probabilité élevée. Accéder ou même effectuer des requêtes sur le serveur en utilisant la session d'un autre utilisateur, peut être utile aux acteurs malveillants. Si la session abandonnée est fermée le plus rapidement possible, la probabilité qu'elle soit détournée est réduite.

Utilité : Ce patron me semble intéressant puisqu'il permet d'invalider la connexion de l'utilisateur au bout d'un certain délais puisque sinon à partir du moment où une personne s'est connectée toute personne qui essayera d'accéder à l'interface pourra y accéder sans mot de passe du moment que la personne est connectée au même réseau wifi que l'ESP8266.

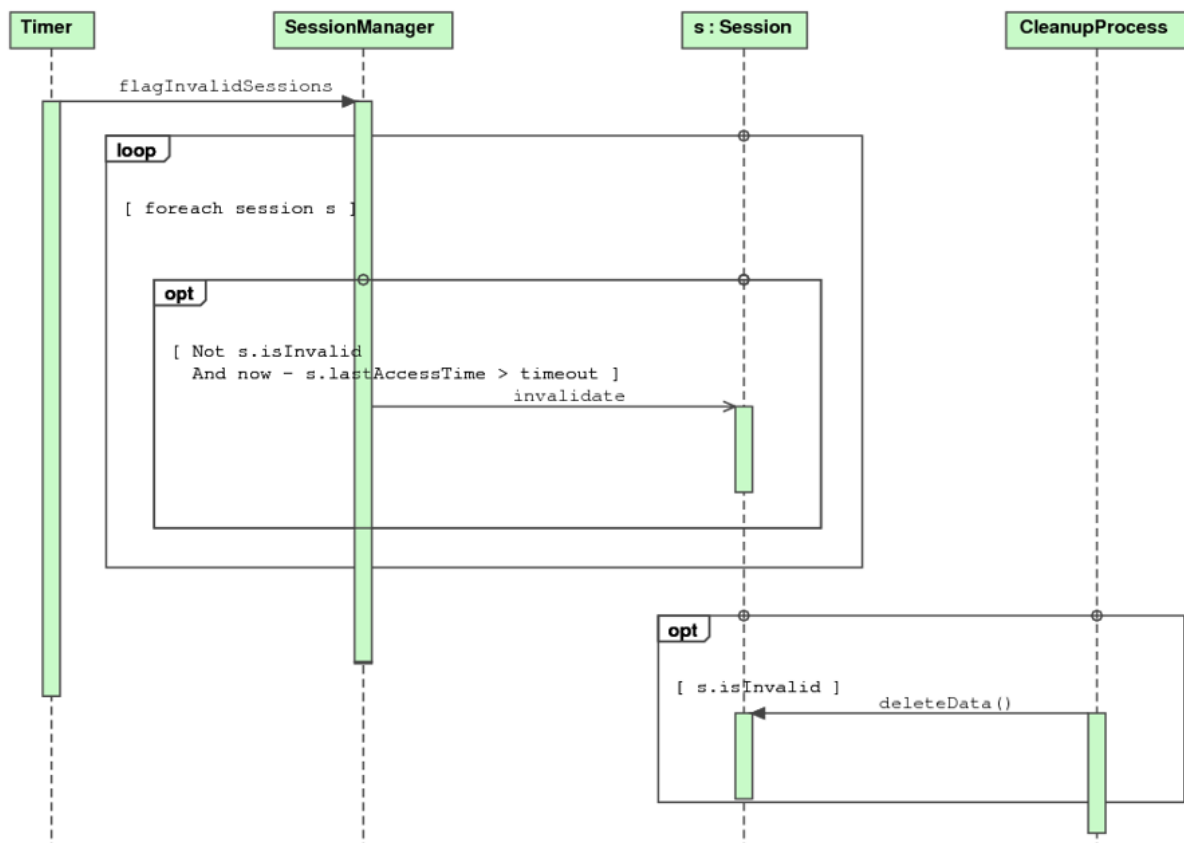


Détails du rôle de chacune des classes du diagramme ci-dessus :

Précisions :

- ❑ **SessionManager** : Le gestionnaire de session contient une référence à chaque session du système.
- ❑ **Session** : Le gestionnaire de session contient une référence à chaque session du système.
- ❑ **Minuteur** : Le minuteur appelle périodiquement l'opération pour invalider les sessions inactives.
- ❑ **CleanupProcess** : Le processus de nettoyage supprime les sessions invalidées.

Le temporisateur déclenche la méthode `flagInvalidSessions` de `SessionManager`. Le `SessionManager` parcourt toutes les sessions, en comparant la dernière heure d'accès de la session à l'heure actuelle. Si la différence est supérieure au délai d'expiration de la session on invalide la session puis `CleanupProcess` recherche les sessions invalidées et supprime leurs données.



10.11 Patron "Session"

Le patron session est nécessaire pour pouvoir implémenter le patron session timeout puisque celui-ci dépend du patron session.

But : Plusieurs objets doivent avoir accès aux valeurs partagées, mais toutes les valeurs sont différentes et ne sont pas uniques.

Avantages :

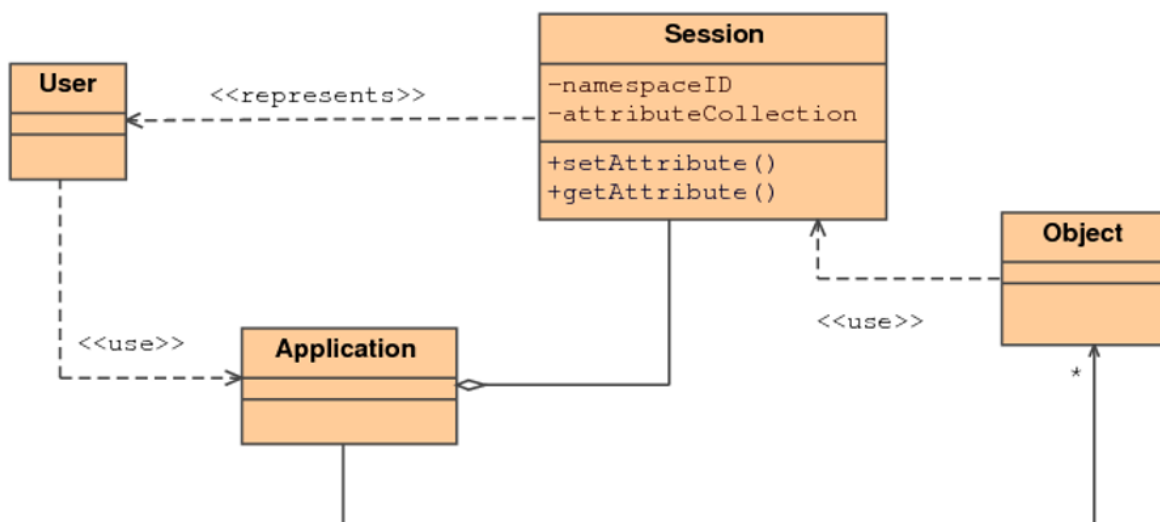
- L'objet Session fournit une interface commune à tous les composants pour accéder aux variables importantes.
- Au lieu de transmettre de nombreuses valeurs autour de l'application séparément, un seul objet Session peut être transmis.
- Chaque fois qu'une nouvelle variable ou objet partagé est nécessaire, il peut être placé dans l'objet Session, puis tous les composants ayant accès à l'objet y auront accès.
- La propagation des modifications est simplifiée car chaque objet d'un thread ou d'un processus ne dépend que d'un seul objet Session partagé.

Inconvénients :

- Alors qu'un objet peut ne pas avoir besoin d'une session, il peut créer ultérieurement un objet nécessitant la session. Lorsque c'est le cas, le premier objet doit toujours garder une référence à la Session pour pouvoir le passer au nouvel objet. Parfois, il peut sembler que chaque objet a une session. La prolifération des variables d'instance de session tout au long de la conception est une conséquence malheureuse, mais nécessaire, du modèle de session.
- Ajouter une session tard dans le processus de développement peut être difficile. Toute référence à un Singleton doit être modifiée. Cela est également vrai lorsque vous essayez de consolider de nombreuses variables globales qui ont été transmises en tant que paramètres dans une session.
- Lorsque de nombreuses valeurs sont stockées dans la session, il faudra une structure

Utilité :

- Le référencement de variables globales peut garder le code propre et direct.
- Chaque objet peut seulement avoir besoin d'accéder à certaines des valeurs partagées.
- Les valeurs partagées peuvent changer avec le temps.
- Plusieurs applications qui s'exécutent simultanément peuvent ne pas partager les mêmes valeurs.
- Passer de nombreux objets partagés dans toute l'application rend les API plus compliquées.
- Alors qu'un objet peut ne pas avoir besoin de certaines valeurs, il peut changer ultérieurement pour avoir besoin de ces valeurs.



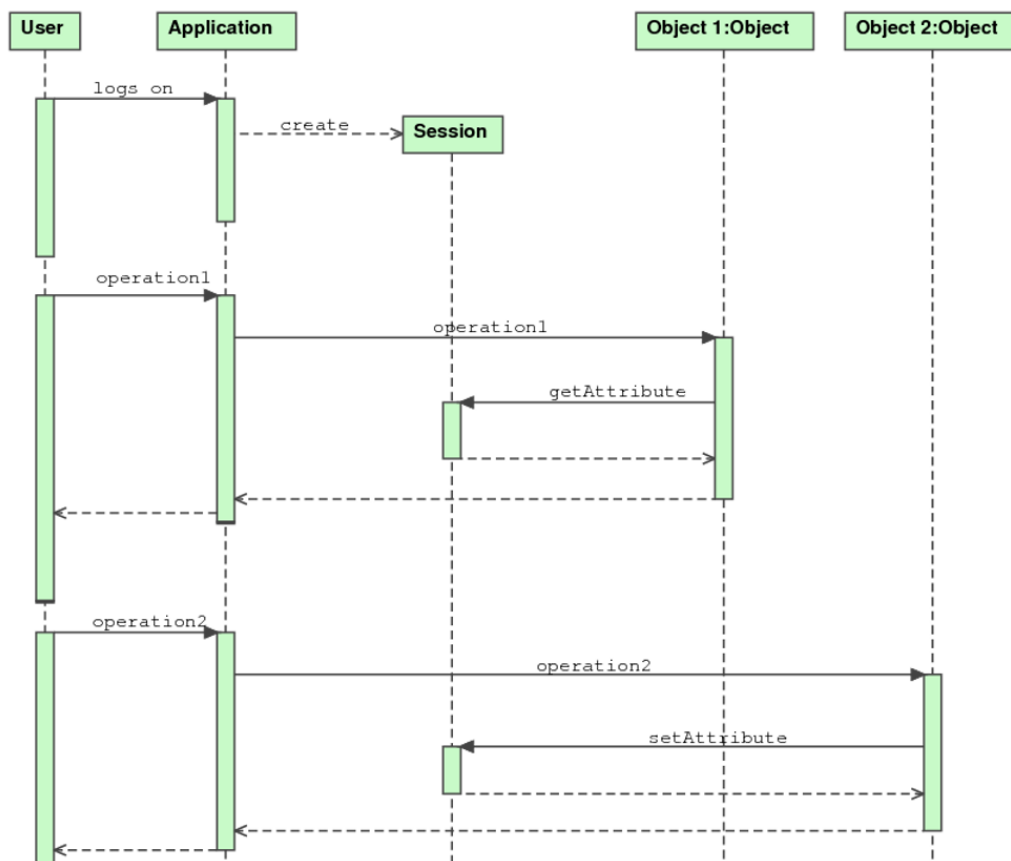
Détails du rôle de chacune des classes du diagramme ci-dessus :

Précisions :

- **User** : L'utilisateur de l'application. Une session sera une représentation des attributs pertinents ou liés à cet utilisateur.
- **Application** : L'application que l'utilisateur utilise. L'application définit le contexte de la session et peut gérer les sessions de plusieurs utilisateurs simultanés.
- **Session** : L'objet session encapsule les attributs liés à un utilisateur. Une session a généralement un identifiant unique dans son contexte.
- **Objet** : Les objets sont des objets exécutant des opérations pendant que l'utilisateur utilise l'application. Les objets peuvent avoir besoin de récupérer ou de stocker des informations dans l'objet de session.

Deux façons de mise en œuvre pour stocker les données de sessions :

1. Stocker les données de session sur le serveur
2. Stocker les données de session sur le client (les données spécifiques à la session pourront être modifiées!!!)



10.12 Patron "Encrypted Storage"

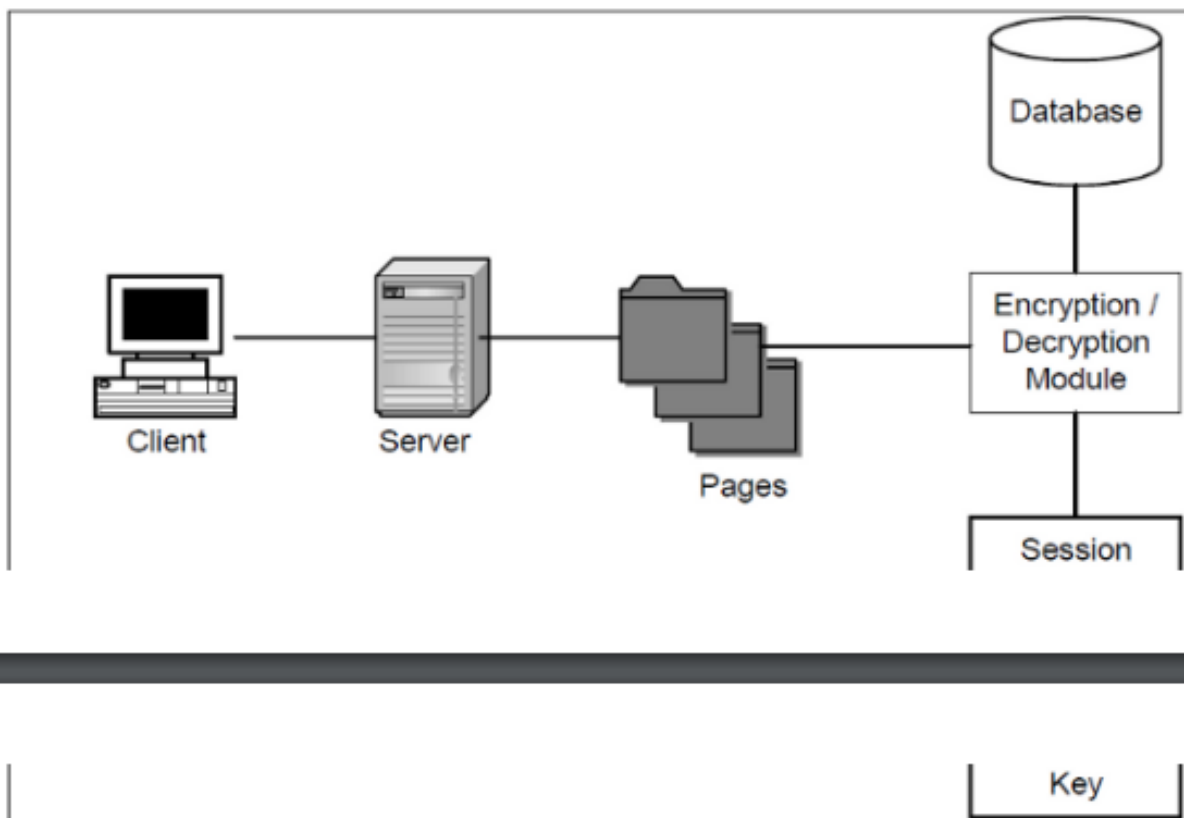
Le patron Encrypted Storage crypte les données les plus critiques avant qu'elles soient commit sur le disque. Puis, avant d'être utilisées, elles sont décryptées en mémoire. Si un voleur tente d'accéder aux données, il n'aura pas accès à celles cryptées.

But : Le pattern Encrypted Storage s'assure que si des données sont volées, elles resteront inaccessibles aux voleurs.

Utilité :

Notre système possède un fichier de configuration avec toute la configuration de l'appareil stockée dessus. Ce fichier est en clair, il pourrait donc lui facilement si récupéré. L'utilisation d'un stockage encrypter éviterait ce problème, puisque le fichier récupéré ne contiendrait pas de données en clair, elles seraient décryptées à l'exécution avec une clé contenue dans le code.

Implémentation : La sauvegarde et le chargement du fichier de config sont gérées par la classe Application. Il est donc nécessaire de faire des changements aux méthodes concernées pour qu'elles utilisent des fonctions de cryptage et de décryptage. Le module de cryptage/décryptage sera une nouvelle classe, et une autre classe s'occupera de fournir la clé de cryptage.



11 Résumé en anglais

This project was assigned by M.Salva, and realised by Gabin Salabert, Yannis Mahiou, Adrien Lenoir, Benoit Louveau and Elouan Raffray. The Internet Of things is the network of all objects embedded with electronics. This project is about one of them. We were given a arduino programmable microcontroller, linked with a temperature sensor, a presence sensor and an infrared emitter/receiver. This device is used to control heat pumps in a house in a totally independent way. The code for it was already existing and was given to us at the beginning of the assignment. The goal of this project was mainly for us to get more experience and knowledge in code designing. Indeed, we had to look at two main aspects of this code, conception and security.

At the beginning of the project, we established a comprehensive documentation for the existing code, so that we could all understand how the device works. After that, the team was divided in 2 groups, one in charge of the design (Yannis Mahoui and Gabin Salabert), the other of the security (Adrien Lenoir, Benoit Louveau and Elouan Raffray). Both aspects were studied in a similar way, which had us looking at design and security patterns and their potential usefulness in the code. Depending on their potential, we had to implement them or not. Beforehands, the existing code was analysed, to help us pin down which problems needed to be solved. On the security side, this work was partially achieved by using security tools to find flaws in the code. This pentesting lead us to find a code injection issue, a bruteforcing issue and others network related flaws.

But not all the patterns we analysed were useful, and most of them didn't end up implemented for various reasons. Some of them were just useless in our system, but some of them were trickier to decide. As we are working on a small microcontroller, available resources were not infinite, and in fact needed to be watched upon closely. Some patterns were just too big to fit on the card, they would have slowed its performance immensely for very little benefit. This wasn't obvious for all patterns, so after implementation we had to check the performance of the card again (CPU, RAM available, sketch size), to check if the pattern did deserve its place in the code.

At the end of the project, numerous patterns were successfully applied to the code, with the example of singleton, memento, input guard, and single access point. All the security flaws that were found are now fixed, and the code has a better design. Overhaul, we are certain to have gained experience in this field of study and the existing code has been improved, making our goal fulfilled.

12 Bibliographie

13 Lexique

14 Annexes