# Open Source Access Controller

*Architecture Software*

*24/03/2014*

# Introduction

L'OSAC est une machine embarquée pilotant des périphériques physiques d'accès, comme des portes à ouverture électrique, des lecteurs de badges etc.

Cette solution de contrôle d'accès se veut open-source et évolutive, afin de s'adapter au mieux aux demandes des clients. Un système Linux embarqué se chargera de piloter ce système, qui lui-même s'exécutera sur un PC open-source.


BeagleBone Black

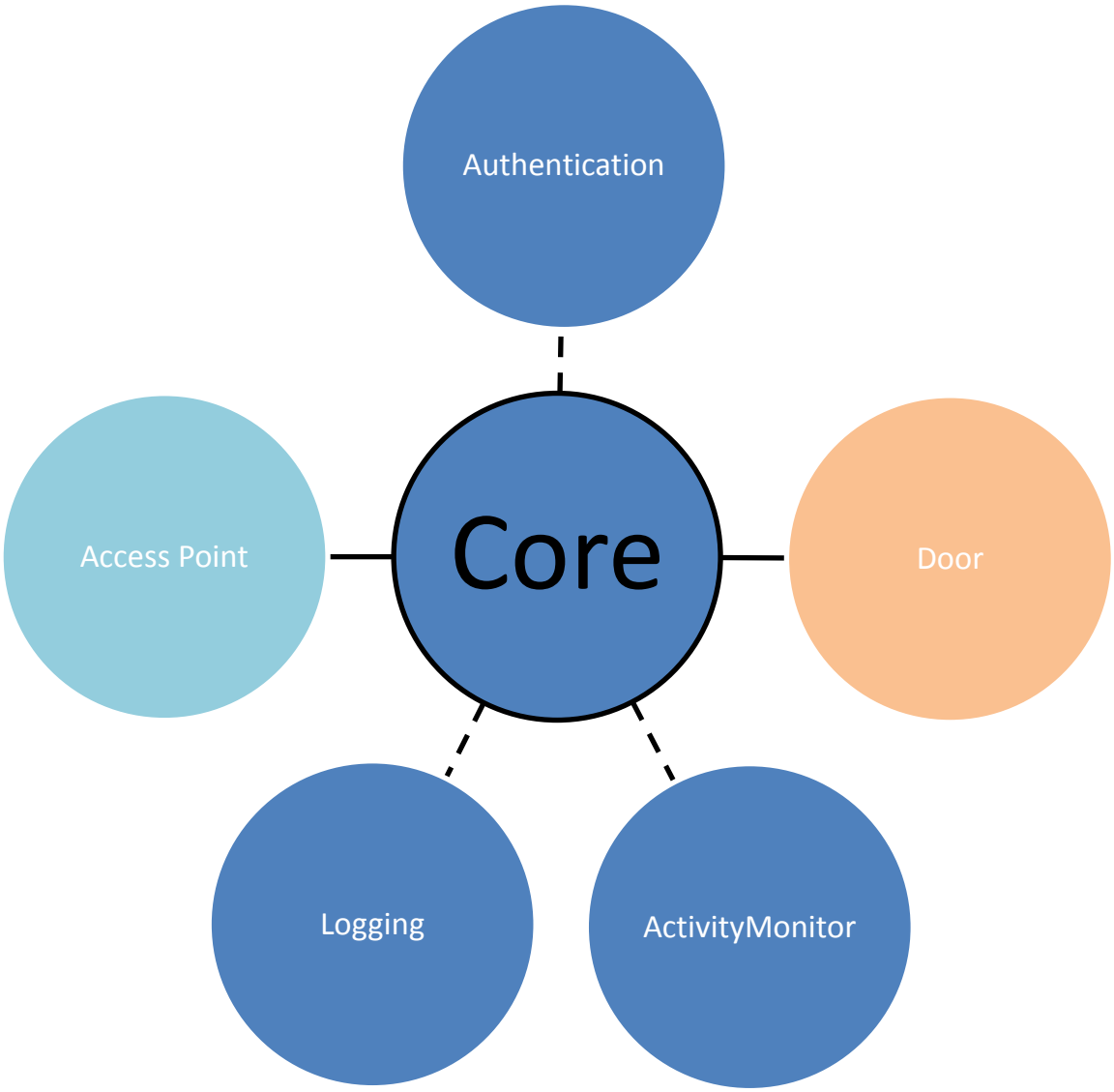
Ici c'est une carte BeagleBone Black qui sera utilisée, mais la solution OSAC ne vise pas à se limiter à cette plateforme uniquement. Potentiellement toutes les cartes capables de faire tourner un Linux et ayant un port Ethernet et suffisamment d'entrées/sorties seront supportées.

# Architecture

Le rôle du programme de contrôle sera de récupérer les données des différents lecteurs de cartes, gérer un service d'authentification qui traitera ces données, et au final décider ou pas de l'ouverture de l'accès.

Pour assurer un niveau suffisant de flexibilité, des modules plugins pourront être greffés au programme principal à des niveaux différents, pour pouvoir redéfinir ou modifier différents comportements. Prenons l'exemple d'un nouveau lecteur RFID qui arriverait sur le marché. Pour ajouter le support de ce lecteur au sein du programme, il suffira de créer un nouveau module capable de piloter le périphérique et de le greffer, sans changer le code principal.

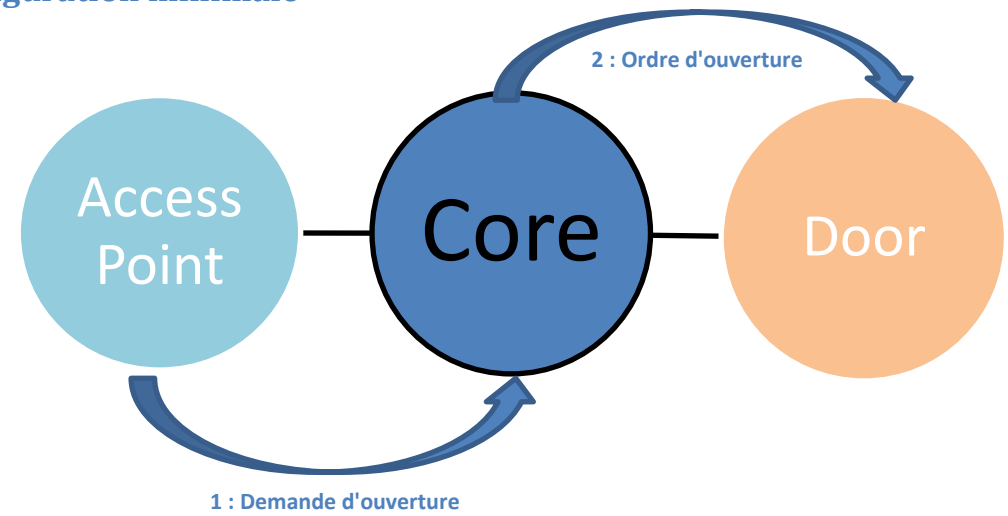
## Schéma global




L'architecture globale consiste en un block principal nommé ici **Core** et de ses sous-modules. Ce block est responsable de la logique principale, de relayer les informations et évènements entre modules, et de s'occuper de leur bon chargement.

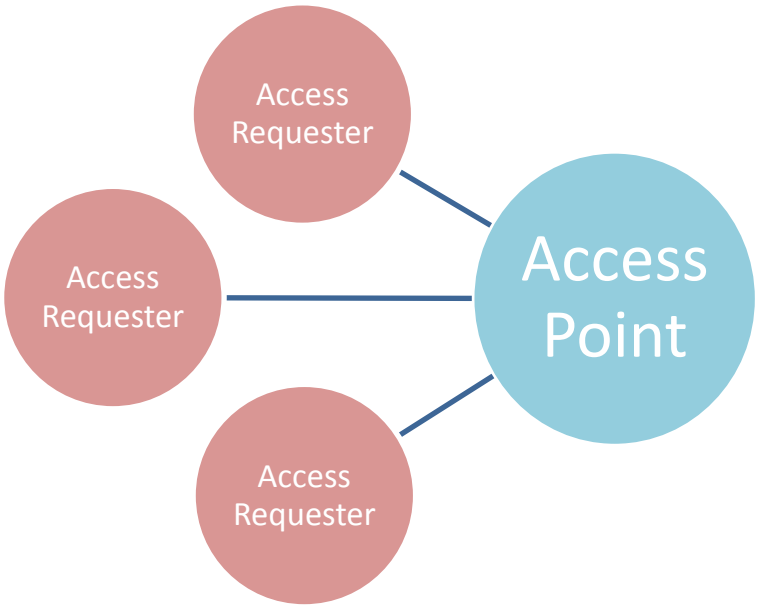
Ici les modules optionnels sont reliés en pointillés, et ceux obligatoires en trait plein. Il sera possible d'avoir plusieurs instances pour certains modules, comme le module **Door**. Le **Core** sera chargé de les gérer car on peut facilement imaginer un contrôle d'accès capable de gérer l'ouverture de multiples portes.

## Configuration minimale

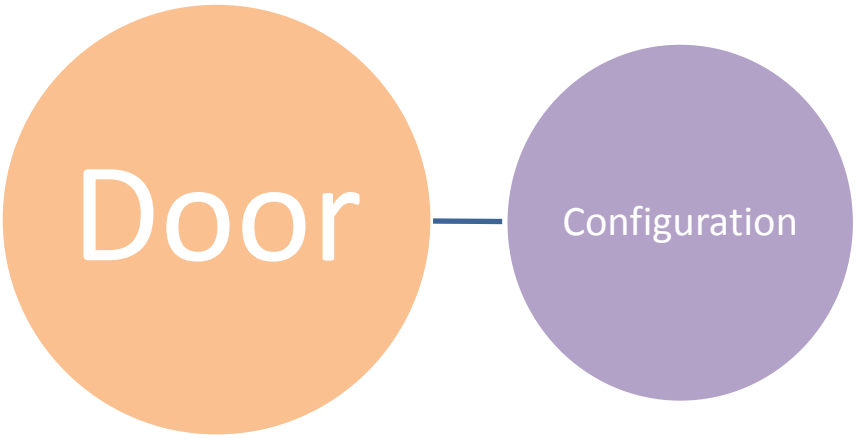



Cette configuration représente la façon la plus simple dont le **Core** est susceptible de fonctionner. On retrouve d'un côté un module **Access Point**, capable de demander l'ouverture d'une porte, et d'une **Door**, qui représente l'accès physique, pouvant être un coffre, un portail ou tout simplement une porte.

Quand un **Access Point** veut ouvrir une porte, il envoie d'abord sa demande au **Core**, qui va la traiter, et ensuite ouvrir la porte. Dans ce cas particulier de configuration, aucun traitement n'est fait par le **Core**, il se contente d'autoriser **l'Access Point** à ouvrir la porte. Nous verrons plus tard quel rôle le **Core** aura dans le cas de l'authentification.




Un **Access Point** contiendra un ou plusieurs **Access Requester**, qui peuvent typiquement représenter un lecteur de badges, mais aussi tout système permettant à l'utilisateur de demander l'ouverture d'une porte comme un clavier numérique mural ou un lecteur d'empreintes. Dans les cas d'authentification à plusieurs facteurs, un **Access Point** pourra représenter l'ensemble d'un lecteur de badges et d'un combiné mural avec deux **Access Requester**.

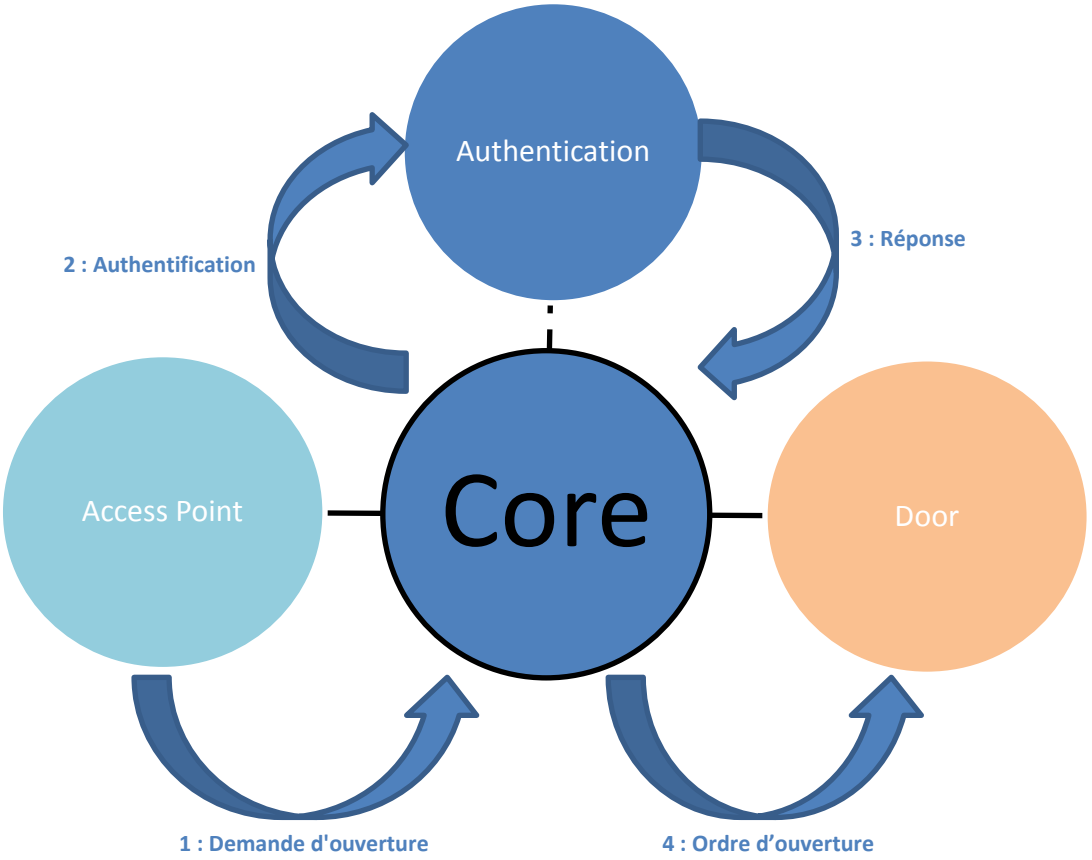



Le module **Door** est simplement capable d'ouvrir physiquement un accès. On peut demander son ouverture ainsi que sa fermeture, mais aussi consulter son état si un capteur d'ouverture est installé. Sa **Configuration** contient de multiples informations :

- Alias de la porte
- Mode d'ouverture (Toujours ouverte, toujours fermée, mode normal)
- Ses différentes **Devices** (voir **Hardware Manager**)
  - **Device** qui gère l'ouverture de la porte
  - Bouton poussoir pour ouvrir sans authentification (optionnel)
  - Capteur d'ouverture de la porte (optionnel)

Le comportement de la porte est entièrement personnalisable, et on peut très bien implémenter une **Door** qui ne s'ouvre jamais, ou qui déclenche une alerte au **Core** quand son capteur d'ouverture reste trop longtemps en état ouvert.

## Module d'authentification




Le module d'authentification vient s'insérer sur le **Core** pour interfacer deux modules **Access Point** et **Door**. C'est ce module qui va décider si l'utilisateur de l'**Access Point** a le droit ou non d'ouvrir la porte à laquelle il essaie d'accéder.

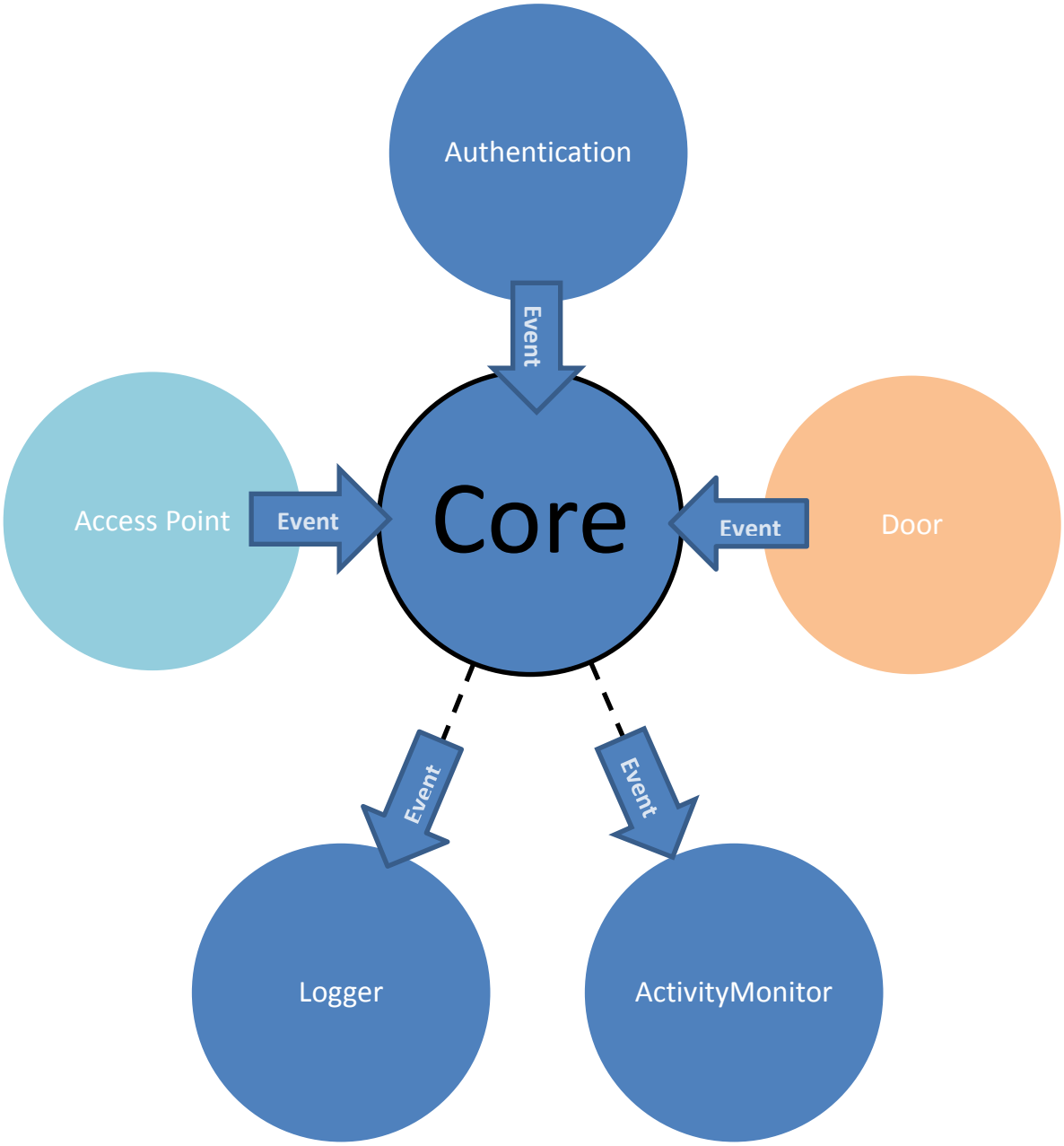
Un détail qui n'a pas été évoqué précédemment car n'important pas encore, l'**Access Point** accompagne chacune de ses demandes d'une donnée représentant l'information que l'utilisateur a envoyé au système pour faire sa demande. Dans le cas d'un lecteur RFID, c'est l'information contenue sur la carte, le code tapé si l'on a un combiné, etc.

Le **Core** relaie ensuite cette info à l'authentification et attends sa réponse. Si l'utilisateur présente les droits suffisants, on lui cède l'accès.

Un peut remarquer qu'à aucun moment la partie **Core** n'a conscience permissions des utilisateurs, et ne réponds jamais toute seule aux demandes des **Access Points**. De cette façon on peut facilement personnaliser la gestion des droits, voire même déporter sa gestion sur une autre machine en changeant le module d'authentification.

Attention : Dans les cas où la porte a un fonctionnement spécial comme « toujours ouverte », la demande d'ouverture de l'**Access Point** n'est pas transmise au module **Authentication** en premier, mais directement acceptée par la porte.

## Modules complémentaires




Ce schéma introduit deux nouveaux modules au système actuel : **Logger** et **ActivityMonitor**. Ces deux modules ont pour rôle de suivre l'état du système et de collecter des informations sur le fonctionnement.

Jusqu'à présent, les modules **Access Point**, **Authentication** et **Door** ne communiquaient seulement entre eux. En réalité, quand ces trois modules conversent avec le **Core**, ils génèrent aussi des **Events**. Ceux-ci possèdent des informations comme :

- le niveau d'alerte
- le message
- la date
- l'émetteur

Le Core va ensuite les rediriger vers les modules **Logger**, pour qu'ils soient traités. L'implémentation de ce module est laissée libre, car beaucoup de choses différentes sont possibles, mais elle pourra par exemple servir de journal, ou chaque **Event** est enregistré dans un fichier.

Le module **ActivityMonitor** est quant à lui un petit peu différent de **Logger**. Son rôle est de récolter un plus petit ensemble d'informations, concernant l'activité du programme et des périphériques, et de les faire transparaître sur les diodes lumineuses de la platine. On pourra par exemple visualiser par un clignotement une personne qui active le point d'accès, ou l'état de la connexion avec le module **Authentication**.

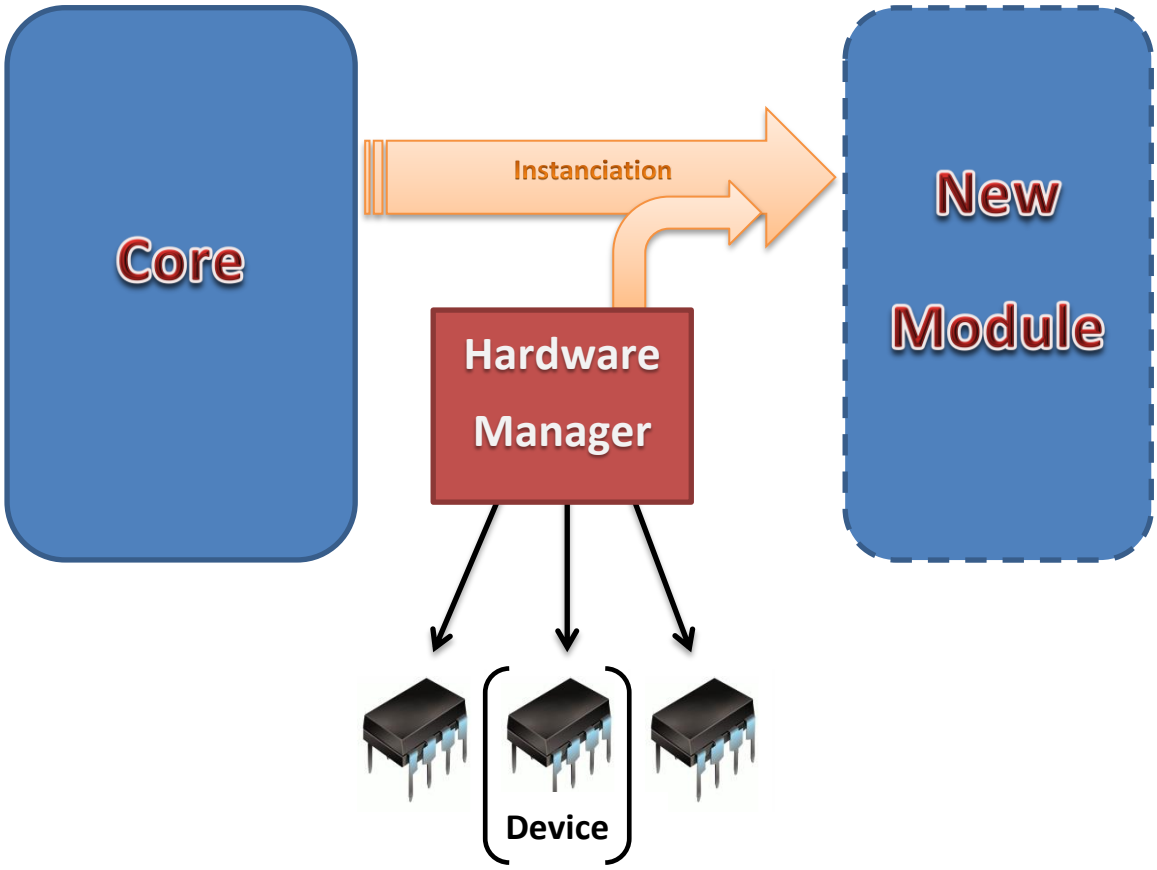
Le but de ce module est surtout de faciliter l'installation du dispositif OSAC, pour attester du bon fonctionnement du système et de ses périphériques.

Enfin, selon la platine utilisée, le nombre de LEDs disponibles ne seront pas les mêmes et donc le module **ActivityMonitor** sera potentiellement spécifique à chaque plateforme.

## Interaction Hardware

Pour que le système ait accès aux différents périphériques matériels (appelés ici **Devices**), un élément **Hardware Manager** sera instancié par le **Core**. En lui-même, le **Core** n'agit jamais sur les **Devices**, mais les différents modules en ont besoin.

### Instanciation module




Afin de rendre le matériel accessible aux modules, l'instance d'**Hardware Manager** sera passée lors de l'instanciation des modules concernés. Evidemment tous les modules ne sont pas concernés, ceux qui n'ont aucune interaction avec le microcontrôleur n'auront pas cette instance.

Les **Devices** en elles-mêmes peuvent représenter plusieurs types de périphériques :

- Un pin GPIO
- Un port série
- Un bus I²C
- Etc…

Au-dessus de ces objets de base qui représentent simplement une liaison physique, on pourra construire une couche **Protocole** qui permettra d'envoyer des informations à travers les **Devices** sans se soucier des détails d'implémentation. Ceci s'avérera pratique dans le cas d'une connexion RS485 par exemple, où plusieurs protocoles existent, mais utilisant la même interface physique.

### Interaction niveau module

Pour synthétiser le concept de **Protocole**, voyons un exemple d'utilisation par le module **Access Point**, qui essayera d'accéder à un lecteur de badge type R10 pour en lire l'état.


Pour le **Module**, le lecteur est simplement représenté par une interface de communication suivant un **Protocole**. A cette couche spécifique du soft, la **libLogicalAccess** pourra être utilisée afin d'interfacer les lecteurs avec la logique du code.

## Configuration du système

Pour permettre aux modules d'être configurables, il sera essentiel de proposer une interface à l'administrateur. Un service **REST** tournera donc sur la platine pour permettre un accès et une configuration à distance.

## Résumé du block Core