Cégep du Vieux Montréal

Event Reactor

Document de conception

Réalisé par

Antoine Deschênes

Présenté à

Jean-Christophe Demers

Cours B65

Projet Synthèse

Table des matières

[Présentation du projet 4](#_Toc412542248)

[En bref 4](#_Toc412542249)

[Composantes du système 5](#_Toc412542250)

[Serveur 5](#_Toc412542251)

[Fournisseur de services 5](#_Toc412542252)

[Services 5](#_Toc412542253)

[Événements 5](#_Toc412542254)

[Réactions 5](#_Toc412542255)

[Interface de gestion 5](#_Toc412542256)

[Détails du projet 6](#_Toc412542257)

[Contraintes, besoins et spécifications 6](#_Toc412542258)

[Communication avec le matériel : Raspberry Pi 6](#_Toc412542259)

[Fonctionnement sans écran : gestion par le web 6](#_Toc412542260)

[Communication inter-appareils : WebSocket 6](#_Toc412542261)

[Communication inter-réseaux : serveur-relai 6](#_Toc412542262)

[Communication inter-langages : Autobahn 6](#_Toc412542263)

[Fonctionnement interne 7](#_Toc412542264)

[Serveur 7](#_Toc412542265)

[Fournisseur de services 7](#_Toc412542266)

[Services 7](#_Toc412542267)

[Événements 7](#_Toc412542268)

[Réactions 7](#_Toc412542269)

[Interface de gestion 8](#_Toc412542270)

[Cas d’usage 9](#_Toc412542271)

[Classes 11](#_Toc412542272)

[Serveur 11](#_Toc412542273)

[Fournisseur de services 11](#_Toc412542274)

[Interface de gestion 13](#_Toc412542275)

[Interface utilisateur 14](#_Toc412542276)

[Démonstration 16](#_Toc412542277)

[Table des figures 17](#_Toc412542278)

# Présentation du projet

## En bref

Le but de ce projet est de créer un système d’événement-réaction programmable. Le système pourra capter continuellement des données provenant du monde réel et virtuel. Lorsque ces données respecteront certaines conditions, des événements seront produits. Un événement pourra engendrer plusieurs réactions autant au niveau matériel que logiciel. De plus, un événement produit à un endroit pourra réagir à un autre endroit en communicant par internet.



Figure 1. Diagramme du système

## Composantes du système

### Serveur

Le serveur est un élément passif qui héberge l’interface de gestion et sert de relais pour les appels de méthodes entre les fournisseurs de service connectés et la page web. Il est situé à l’extérieur des réseaux fermés afin de rendre la communication possible n’importe où.

### Fournisseur de services

Les fournisseurs de services sont le centre du système. Ce sont eux qui collectent les données (les services), gèrent les événements et émettent les réactions. Un fournisseur de service n’est pas associé à une machine en particulier, c’est une instance d’un logiciel qui peut fonctionner sur n’importe quel ordinateur répondant aux conditions minimales. Un seul ordinateur pourrait même avoir plusieurs instances en fonctionnement.

#### Services

Un service est un élément du système donnant accès à des données et à des actions matérielles ou logicielles. Par exemple, un service de thermomètre permet de lire une température, alors qu’un service de courriel permet d’envoyer un courriel. La variété des services disponibles sera limitée à un certain nombre pour ce projet, mais le modèle sur lequel le système est basé permettrait facilement l’ajout de nouveaux types de services.

#### Événements

Un événement est causé lorsque des données valident ou invalident un ensemble de conditions. L’événement peut appeler plusieurs réactions sur n’importe quel fournisseur de services. Par exemple, un événement pourrait survenir lorsque la température moyenne de deux thermostats dépasse 25°C.

#### Réactions

Une réaction est l’appel à un service provenant d’un événement. La réaction est en fait une modification d’un attribut d’un service. Par exemple, une réaction à un événement pourrait être d’activer un signal électrique afin d’allumer une lumière.

### Interface de gestion

L’interface de gestion est accessible à partir d’un navigateur web. C’est avec celle-ci qu’on peut voir tous les fournisseurs de services, ajouter, modifier et supprimer les services disponibles, les événements et les réactions, puis afficher les données collectées par les services.

# Détails du projet

## Contraintes, besoins et spécifications

Le projet présente plusieurs défis techniques et certains choix ont été faits afin de répondre au mieux aux besoins du projet.

### Communication avec le matériel : Raspberry Pi

Un des buts principaux du système est d’interagir avec le monde réel. Communiquer avec des capteurs et des actuateurs n’est pas une tâche facile sur un ordinateur conventionnel. De plus, il comporte des contraintes de mobilité et n’est généralement pas disponible pour effectuer une tâche dédiée, 24h sur 24. Un microcontrôleur peut accéder facilement au matériel, mais n’est pas très évolué au point de vue logiciel. Le Raspberry Pi a été choisi car il est peu coûteux, comporte un plein système d’exploitation, un accès facile au matériel, prend peu d’espace et peut être dédié à une seule tâche. Sur cet appareil, le moyen le plus simple de communiquer avec le matériel est le langage Python. La partie du projet donnant accès au matériel, le fournisseur de services, fonctionnera avec ce langage.

### Fonctionnement sans écran : gestion par le web

Les fournisseurs de services et le serveur fonctionneront sans écran, clavier, ni souris. Une interface universelle permettra la configuration des services et des événements. Une page web dynamique répond bien à ce besoin : l’utilisateur n’a besoin que d’un navigateur web et une connexion réseau pour gérer le système, sans avoir à installer d’applications.

### Communication inter-appareils : WebSocket

Plusieurs fournisseurs de services et l’interface web devront communiquer entre eux. Un des seuls moyen d’effectuer une communication bidirectionnelle en temps réel avec un navigateur web est le WebSocket avec JavaScript. Vu l’utilisation du protocole WebSocket pour la page de gestion, il sera aussi utilisé pour permettre aux appareils fournisseurs de services de communiquer activement entre eux.

### Communication inter-réseaux : serveur-relai

La plupart des réseaux ont des pare-feu fermés qui empêchent d’initier les connexions entrantes. Si on veut gérer le système à distance, ou même déclencher un événement à distance, il faut s’assurer que la communication est possible. Un serveur ouvert aux connexions entrantes sera déployé sur le web afin de permettre d’initier des connexions en utilisant une adresse fixe et servira de relai pour la communication entre les appareils. De plus, il hébergera la page de gestion afin de ne pas avoir besoin des fichiers HTML et JS sur son appareil.

### Communication inter-langages : Autobahn

La page web devra utiliser JavaScript et les fournisseurs de services utiliseront Python, qui n’est pas capable de communiquer par WebSocket sans librairies externes. La plupart de celles-ci sont obsolètes. La librairie Autobahn sera utilisée parce qu’elle est tenue à jour et disponible sur les deux langages, même si la documentation de l’API laisse à désirer. Recevoir des événements du réseau demande de programmer de façon asynchrone avec une boucle événementielle, ce qui est natif et facile avec JavaScript, mais demande beaucoup de nouveaux concepts avec Python. Ce-dernier ne fournit pas ses propres librairies et le style de programmation devient assez différent. Twisted sera utilisée étant donné qu’elle est la référence en applications réseau événementielle pour *Python*.

## Fonctionnement interne

### Serveur

La machine du serveur utilisera le système d’exploitation Linux CentOS et sera hébergé chez Cloud At Cost. Le port 8080 (non bloqué par les pare-feu) sera ouvert aux connexions http et *WebSocket*. Il exécutera une instance du serveur web Crossbar, qui est capable de servir les deux protocoles qu’on souhaite utiliser. La répartition des appels entre les appareils sera ainsi faite automatiquement, mais il serait aussi possible de coder le serveur *WebSocket*.

### Fournisseur de services

Le fournisseur de service pourra contenir de 0 à plusieurs services et de 0 à plusieurs événements. Chaque fournisseur devra porter un identifiant unique. Le nom de l’hôte sera pris par défaut mais il y aura moyen de modifier l’identifiant. Plusieurs méthodes seront disponibles à travers les WebSocket : Ajout, suppression et accesseur pour les événements et les services. Le fournisseur devra aussi écouter les demandes d’informations à partir de la page web. Il enverra alors une liste des événements et des services qu’il contient. Une boucle appellera la mise à jour des événements et des services de façon séquentielle. Elle devra fonctionner assez rapidement pour sembler fonctionner en temps réel, tout en essayant de ne pas étrangler les ressources.

#### Services

Lors de la mise à jour d’un service par la boucle, une lecture de données sera faite sur le matériel, puis la valeur sera copiée dans une variable du service. La page web et les événements liront la copie locale des données. La mise à jour des services est nécessaire afin d’empêcher les lectures simultanées sur le même port. De plus, certains services plus critiques devront constamment faire des vérifications. Par exemple, le service qui permet de contrôler une plaque thermoélectrique devra s’assurer que la plaque ne dépasse pas les limites de température et de courant spécifiées par le manufacturier.

#### Événements

La mise à jour des événements consistera à vérifier si les conditions de déclenchement sont atteintes ou perdues. Puisque les événements seront programmables pendant l’exécution du logiciel, la condition ne pourra pas être simplement insérée dans un if. *Python* permet d’évaluer n’importe quelle expression située dans une chaîne de caractères, mais cette fonction ne sera pas utilisée, car elle peut être mal utilisée. Une classe contenant des helpers aura une méthode qui remplacera les noms de variables valides par leur valeur respective, et détectera si la valeur peut être lue localement ou si un appel distant doit être effectué. Cela aura comme avantage de donner une certaine autonomie aux fournisseurs de services. De plus, la fonction helper fera quelques corrections syntaxiques (obliger le double égal, remplacement de mots) avant de passer l’expression à une fonction capable d’évaluer uniquement les expressions simples (sans appels de fonctions). Lorsque la validité de la condition sera différente par rapport à la vérification précédente, les réactions seront déclenchées.

#### Réactions

Les réactions surviendront au déclenchement d’un événement, tel que décrit plus haut. Il y aura deux composantes à une réaction : la variable de service à affecter et la valeur qu’on souhaitera lui donner. Comme les événements, la valeur à donner pourra provenir de variables et d’expressions mathématiques.

### Interface de gestion

La page de gestion affichera la liste des fournisseurs et de leur services, les événements et les réactions, ainsi que toutes les configurations et données rattachées. Une interface dynamique de type vue en arbre sera utilisée. Au démarrage de la page, un appel à tous sera envoyé afin de découvrir quels appareils sont connectés, quels services et quels événements sont disponibles. On pourra demander d’afficher plus de détails. Les détails affichables pour un service sont ses données actuelles et sa configuration. La page demandera régulièrement les données aux services dont les détails sont affichés afin de suivre en temps réel le fonctionnement. Les détails affichables pour un événement sont son état actuel et sa configuration. Un double-clic sur un paramètre de configuration de service ou d’événement permettra de le modifier. Des boutons permettant de supprimer et d’ajouter les services, les événements et les réactions seront disponibles. La structure des classes ressemblera à la hiérarchie utilisée par un fournisseur de services.

## Cas d’usage

Il y a principalement deux acteurs dans le système. Au frontend, il y a l’utilisateur et au backend, le fournisseur de services. Pour l’utilisateur, uniquement les cas d’usage de la page de gestion sont documentés. Les périphériques externes sont fournis par l’utilisateur ; Les cas d’utilisation de ceux-ci ne peuvent être tous considérés.

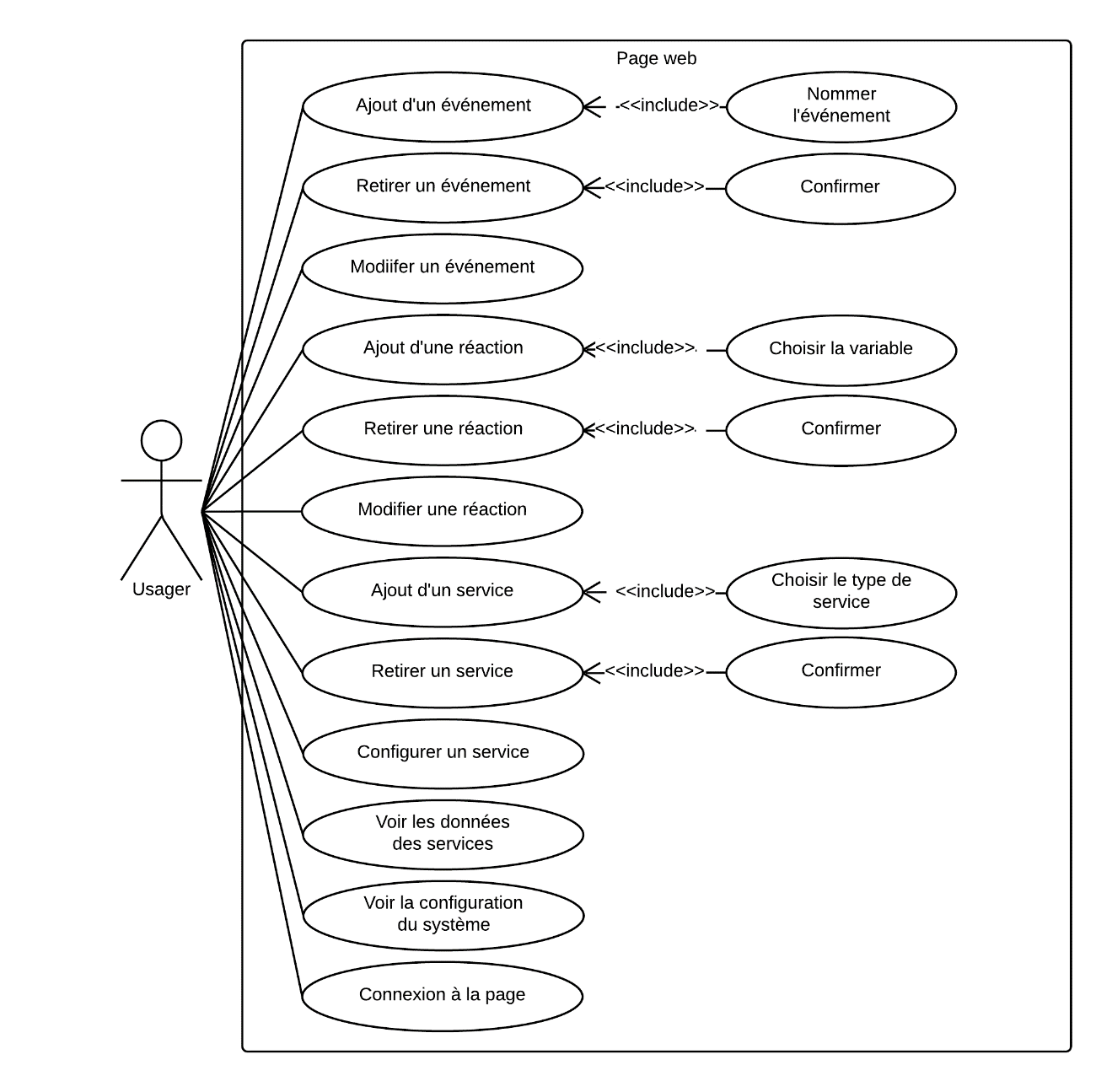


Figure 2. Cas d'usage pour l'usager de la page web

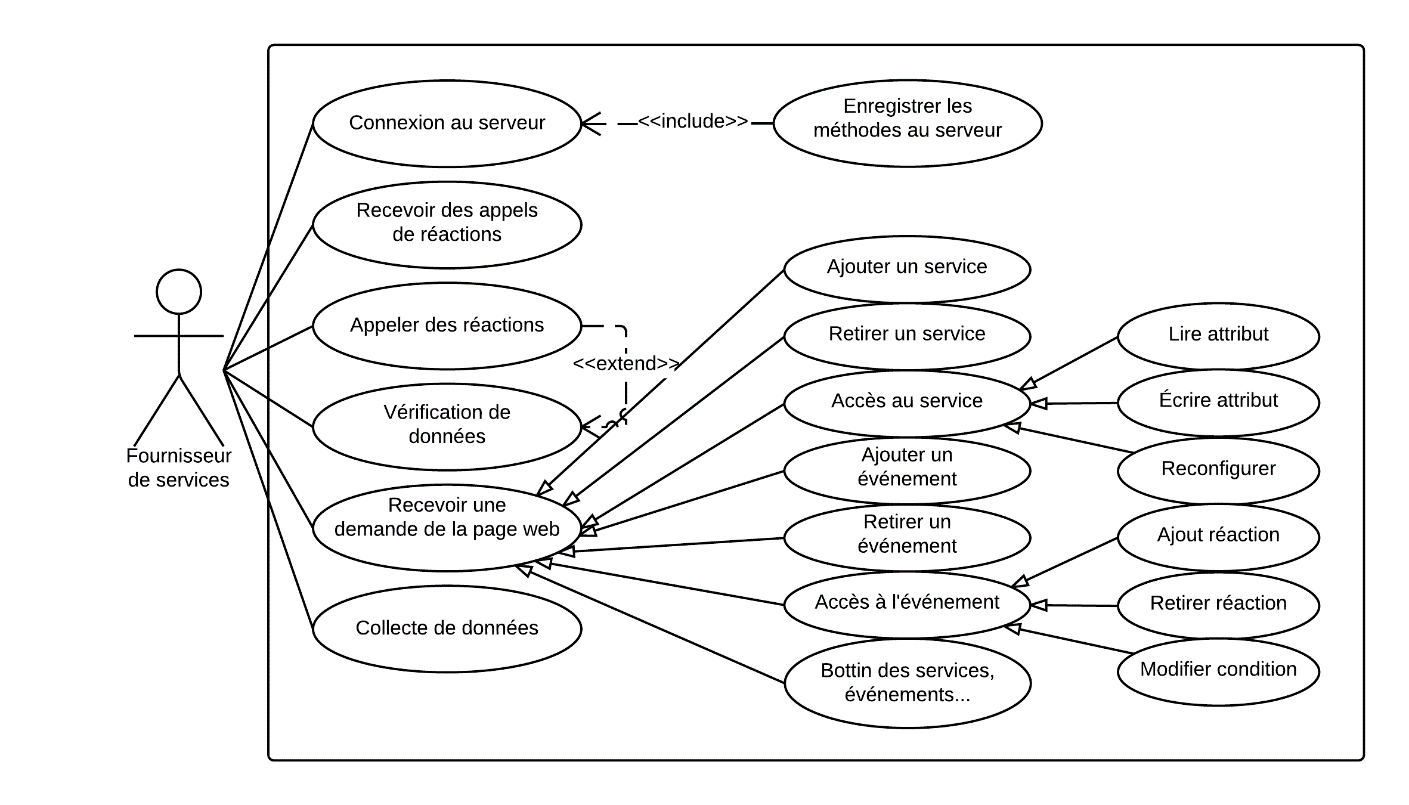


Figure 3. Cas d'usage pour le fournisseur de services

## Classes

### Serveur

Le serveur n’a pas d’« intelligence » et ne possédera donc pas de classes. Si la connexion et la déconnexion des clients deviennent trop difficile à gérer dans les autres processus, un système de gestion minimal pourra être ajouté au serveur.

### Fournisseur de services

Le fournisseur de services aura une classe principale Provider qui devra absolument hériter d’une classe fournie par les librairies afin de fonctionner de manière asynchrone. Celle-ci possédera des méthodes de gestion et d’accès aux objets Service et Event et deux tables contenant ces objets. Elle contiendra la boucle de vérification des services et des événements qui sera appelée à l’initialisation.

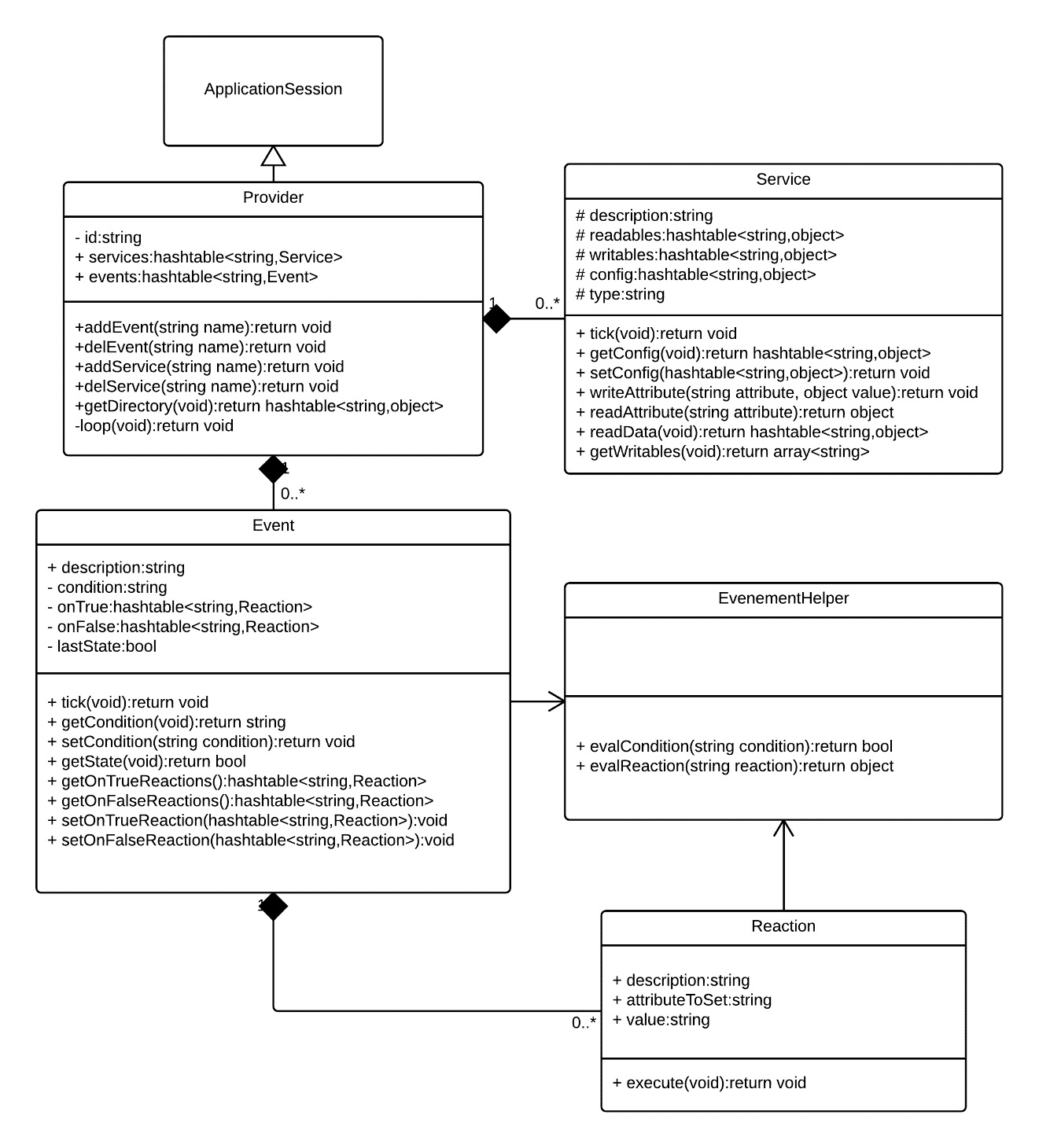


Figure 4. Diagramme de classes du fournisseur de services

Une classe héritera de Service pour chaque type de capteurs et d’actuateurs, car les paramètres et les accès à ceux-ci seront nécessairement différents dans chaque cas. Les services devront fournir un accès aussi standardisé que possible afin d’éviter les cas d’exception. Le nombre de méthodes sera possiblement réduit, car il faudrait autrement enregistrer six méthodes par services sur le réseau. Les variables et configurations des services seront stockées dans les hashtables correspondantes. Elles sont directement dans les classes sur le diagramme UML pour une question de clarté.

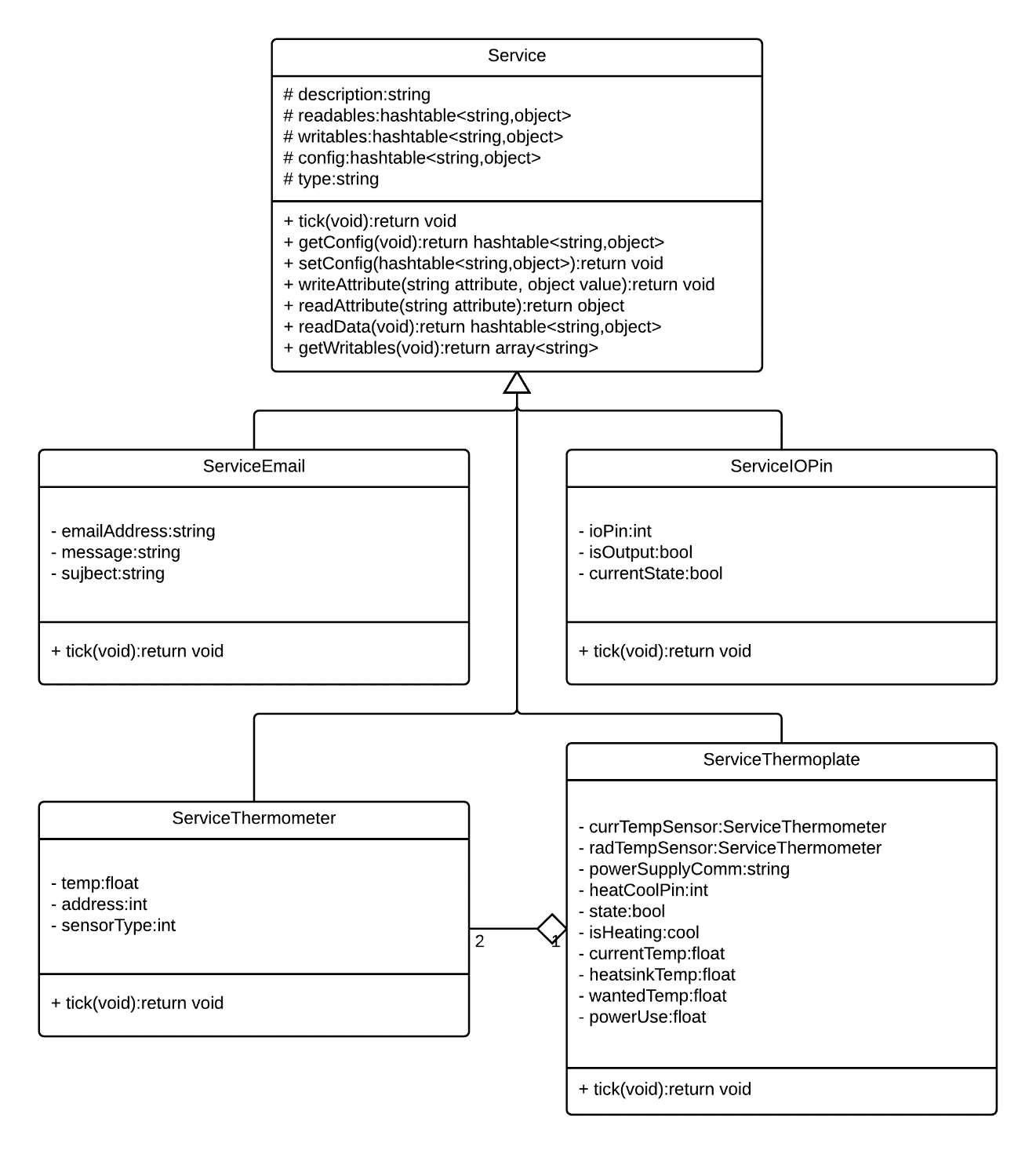


Figure 5. Diagramme de classes du fournisseur de services : Hiérarchie des services

### Interface de gestion

La hiérarchie des classes de la page de gestion réplique localement la structure du réseau, car elle va afficher en arborescence les événements et les services un peu comme ils sont déjà disposés dans le fournisseur de services.

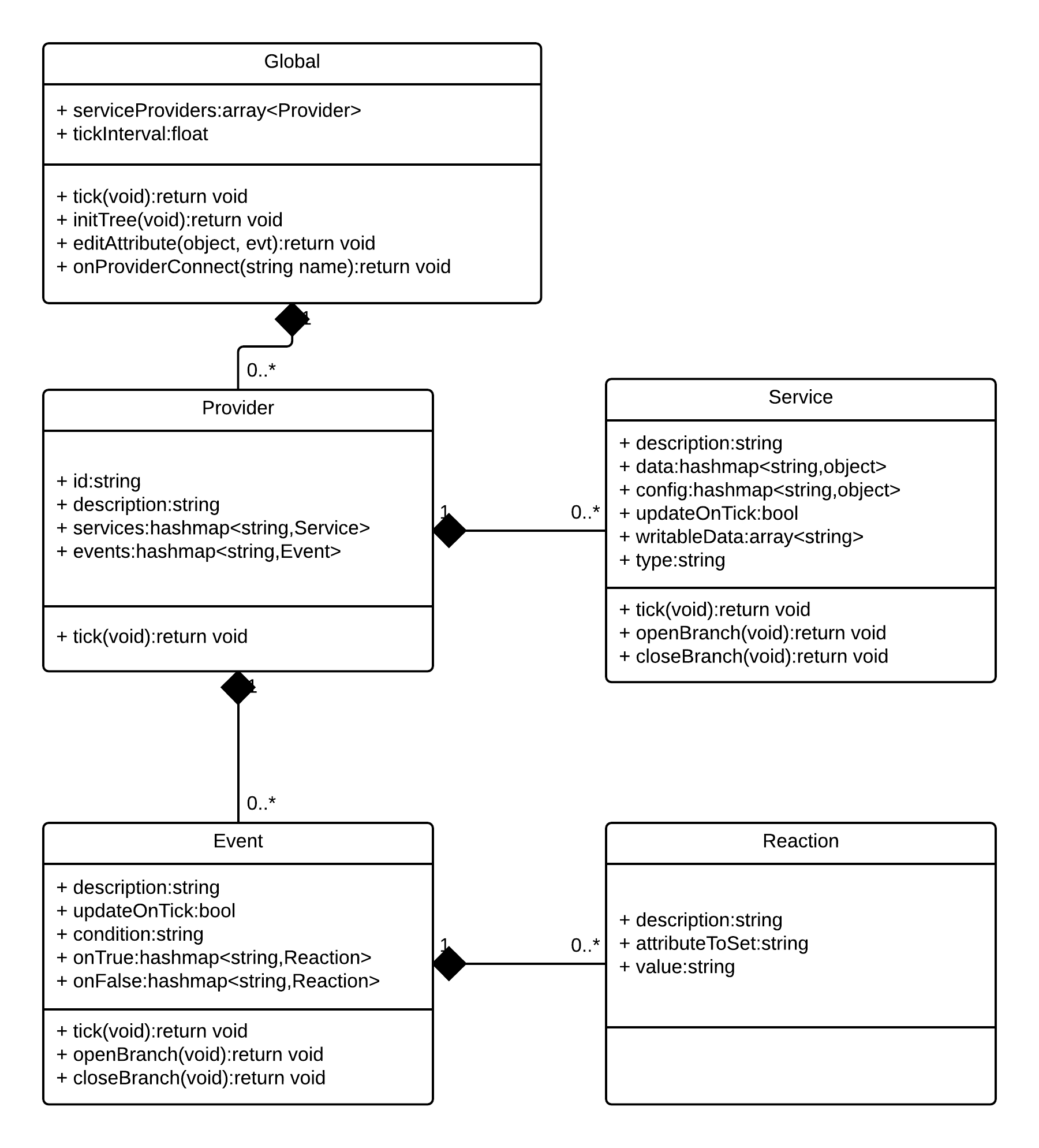


Figure 6. Diagramme de classes de l'interface de gestion

## Interface utilisateur

L’interface de gestion se tiendra sur une seule page accessible autant par un appareil mobile que par un ordinateur de bureau. Deux listes seront affichées, soit les services et les événements. Ils seront regroupés par appareil fournisseur. Une vue en arborescence sera utilisée et il sera possible de cliquer n’importe quel de ces éléments pour afficher ses détails. Les variables et les configurations seront modifiables par un double-clic qui fera apparaître une boîte d’entrée. Les données lues par les services et l’état des événements seront mises à jour continuellement afin de suivre en temps réel l’état du système.

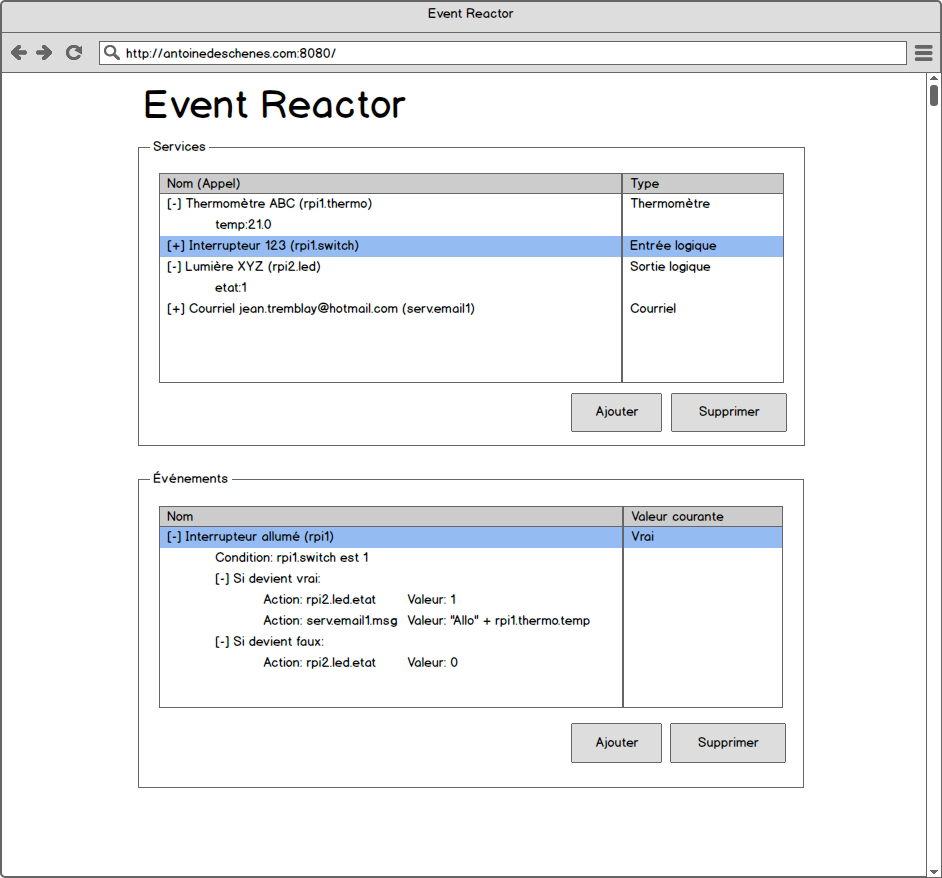


Figure 7. Interface de gestion : Version ordinateur

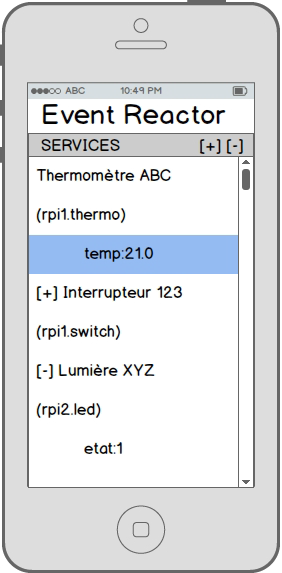


Figure 8. Interface de gestion : Version mobile

# Démonstration

Afin de montrer les possibilités du logiciel, une démonstration sera montée avec les composantes suivantes :

* Un Raspberry Pi
* Avertisseur sonore
* Un thermomètre infrarouge
* Un thermomètre numérique
* Interrupteurs et DELs
* Plaque thermoélectrique montée à un pot en métal et un radiateur
* Un ordinateur pour la page de gestion

La démonstration consistera en un refroidisseur de bière à température contrôlée muni d’un antivol et d’autres fonctionnalités.

# Table des figures

[Figure 1. Diagramme du système 4](#_Toc412542209)

[Figure 2. Cas d'usage pour l'usager de la page web 9](#_Toc412542210)

[Figure 3. Cas d'usage pour le fournisseur de services 10](#_Toc412542211)

[Figure 4. Diagramme de classes du fournisseur de services 11](#_Toc412542212)

[Figure 5. Diagramme de classes du fournisseur de services : Hiérarchie des services 12](#_Toc412542213)

[Figure 6. Diagramme de classes de l'interface de gestion 13](#_Toc412542214)

[Figure 7. Interface de gestion : Version ordinateur 14](#_Toc412542215)

[Figure 8. Interface de gestion : Version mobile 15](#_Toc412542216)