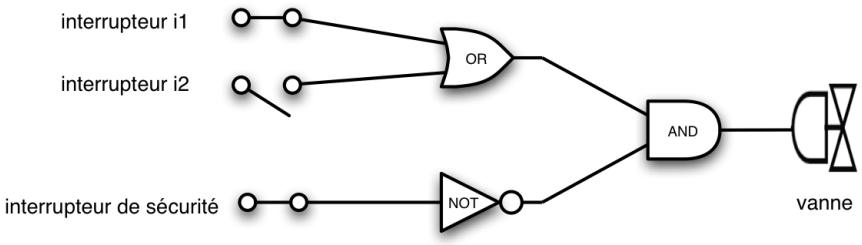
Bases de la conception orientée objet
Portes logiques et simulation de circuit

Steven Costiou
Stéphane Ducasse
Inria

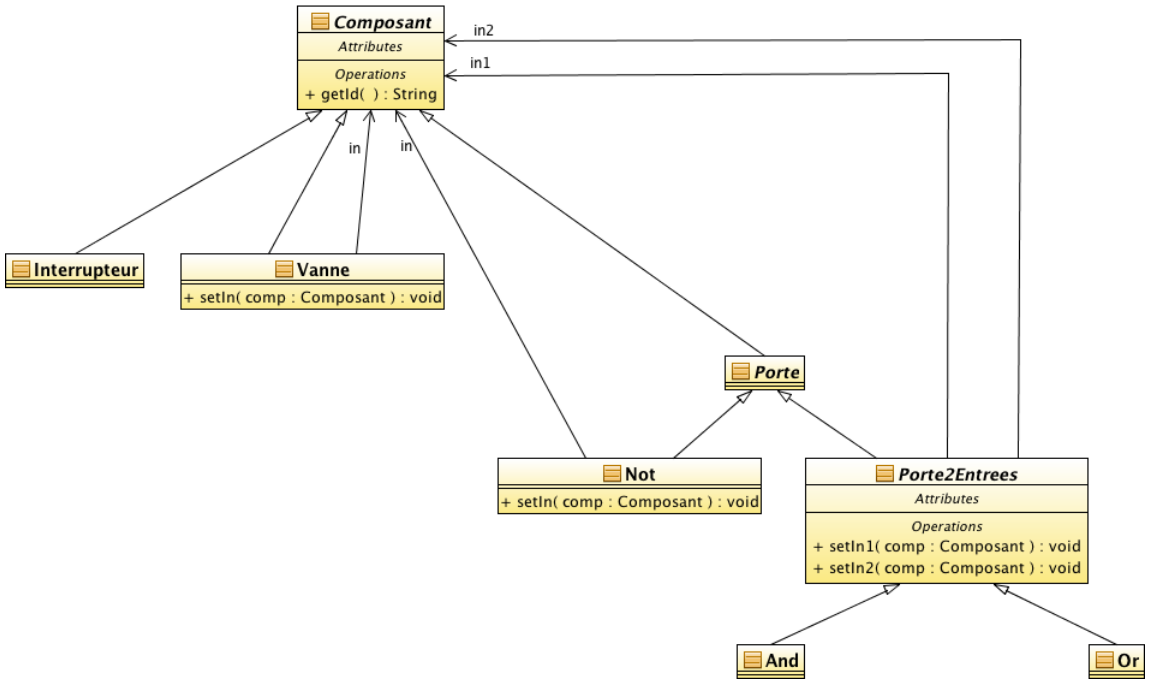
10 octobre 2021

On considère des circuits constitués de portes logiques *and*, *or* ou *not* connectées en entrée sur des interrupteurs et en sortie sur des appareils. Dans ce TP nous utiliserons cet exemple :



Composants

La hiérarchie de classes suivante permet de modéliser nos composants :



— `Composant` est la classe racine de la hiérarchie. Elle est abstraite et fournit une méthode `id`. Cette dernière retourne un identifiant unique calculé en renvoyant le hash du receveur : `^ self hash`.
— un `Interrupteur` n'a pas d'entrée
— une `Vanne` est connectée en entrée sur un composant (`in`)
— `Porte` est la classe abstraite racine des portes logiques
— un `Not` a une connection en entrée sur un composant (`in`)
— `Porte2Entrees` est abstraite et factorise les portes logiques à 2 entrées (And, Or, ...) connectées en entrée sur 2 composants (`in1` et `in2`)

1 Classes de portes

1. Développez le modèle de composants spécifié ci-dessus
2. Programmer en **TDD** une méthode `description` dans les classes de composants qui fournit une chaîne de caractères formée de :
   — Le nom de leur classe
   — leur identifiant (donné par `getId()`)
   — pour les composants disposant d'entrée(s), les identifiants (via la méthode `id`) des composants correspondants ou la chaîne de caractères `''non connecte''` si l'entrée n'est pas connectée.

   Par exemple pour un and (`And (48d6c16c)`) non connecté en entrée 1, connecté en entrée 2 sur un not (`Not (5abb7465)`) : `And (48d6c16c) in1: non connecté in2: Not (5abb7465)`

2 Simulation

L'état logique d'un composant est fourni par sa méthode `state` qui retourne un booléen. Un interrupteur peut être positionné à `true` ou `false` par ses méthodes respectives `on` et `off`. L'état des autres composants peut-être calculé en fonction de l'état des composants connectés sur ses entrées.

1. Programmez les méthodes `on` et `off` de la classe interrupteur (toujours en TDD)
2. Si ce n'est déjà fait, instanciez et reliez entre eux les composants nécessaires à la modélisation du circuit décrit au début du sujet
3. Écrivez des tests qui vérifie l'état final de la connexion du circuit (true ou false) en fonction de l'état des deux interrupteurs en début de circuit.

3 Modèle de circuit

Jusqu'à maintenant, les circuits n'ont pas d'existence propre et sont manipulés comme des composants individuels. Nous allons modéliser les circuits sous forme d'objets définis par la classe Circuit :

3.1 La classe Circuit

Programmer la classe Circuit suivant les spécifications ci-dessous :
- Un Circuit est composé d'instances de Composant au travers du rôle composants. Les composants seront stockés dans une collection.
- Un circuit comporte, en sus de sa description UML, une collection d'interrupteurs et une collection de sorties de circuit (*e.g.*, des vannes).
- Une méthode de classe de Circuit permet d'instancier un circuit en lui passant une collection de composants connectés (formé comme précédemment.) :
  - Vous devez ajouter les composants dans la liste composants du nouveau circuit,
  - trier les composants par identifiant,
  - trier les interrupteurs et les sorties et les ajouter respectivement aux collections d'interrupteurs et de vannes.

3.2 Instanciation et test du modèle de circuit

Dans une classe de test :
- Instancier le circuit en lui fournissant un nom et le tableau de composants connectés précédemment créé.
- Testez :
  - La liste des interrupteurs du circuit,
  - la liste de vannes.

4 Modélisation des signaux logiques

Nous allons maintenant simuler des signaux logiques en entrée de notre circuit pour démarrer la vanne. Pour cela, nous allons modéliser deux types de signaux : SignalHaut et SignalBas. Les instances de la classe SignalHaut correspondent à la valeur logique 1 (booléen true). Les instances de la classe SignalBas correspondent à la valeur logique 0 (booléen false).

Les objets représentant les signaux se propagent au travers du circuit par retro-évaluation. Pour décider de son statut, l'objet en fin de circuit (la vanne) évalue son état en évaluant ses composants en entrée. Chacun de ces composants évaluent alors leurs propres composants en entrée, et ainsi de suite jusqu'à atteindre les interrupteurs. Lors de leur évaluation, les interrupteurs renvoient alors des objets *signaux* qui se propagent en chemin inverse par retour de composant jusqu'à la vanne.

Lors de leur passage par un composant, les signaux peuvent être modifiés en fonction de l'opérateur logique appliqué par le composant. Par exemple, lorsqu'un composant Not récupère une instance de SignalHaut, le composant renvoit une instance de SignalBas. Désormais, nous allons manipuler des objets qui représentent les signaux logiques. Nous allons donc leur déléguer l'application des opérations logiques : ce sont les objets qui savent comment ils doivent être transformés par une opération logique.

4.1 Les classes SignalHaut et SignalBas

Les classes de signaux logiques sont indépendantes du projet de modélisation de circuit. Elles doivent être utilisables comme une librairie dans n'importe quel autre projet. Pour parler des classes de signaux de manière générique, nous faisons référence dans le texte à une classe SignalLogique.

Les objets signaux savent répondre au protocole suivant :
- **value** : renvoie la valeur logique du signal (true ou false).
- **not** : renvoie le signal "inverse".
  Exemple : SignalHaut new not → renvoie une instance de *signal bas*.
- **and : aSignalLogique** : opérateur logique "*and*".
  Exemple : new SignalHaut new and : SignalBas new → renvoie une instance de *signal bas*.
- **or : aSignalLogique** : opérateur logique "*or*".
  Exemple : SignalHaut new or : SignalBas new → renvoie une instance de *signal haut*.

Vous validerez votre modèle de signaux logiques via des tests unitaires.

4.2 Transit des signaux par les différents composants

Les composants doivent maintenant pouvoir évaluer un signal d'entrée et renvoyer un signal en sortie. Les composants et leur circuit implémentent tous une méthode evaluate. Cette méthode suit les spécification suivantes :

La méthode evaluate des circuits considère la liste des composants en sortie, leur envoie le message evaluate puis traite le résultat.

La méthode evaluate des interrupteurs renvoie un objet signal qui est l'état de l'interrupteur.

Pour tout autre composant, la méthode evaluate :

— récupère le résultat de l'évaluation de ses composants en entrée (un ou plusieurs signaux),

— applique une opération logique sur ce résultat,

— renvoie le signal produit par l'opération.

Les opérations logiques sont appliquées en appelant les méthodes présentes dans le protocole des objets signaux. Par exemple, prenons un composant Not et sa méthode evaluate. L'opération exécutée par cette methode sera ^ in evaluate not.

Testez l'évaluation de votre circuit, en appelant sa méthode evaluate.