Les Design Patterns et après . . .

L'évolution de la méthodologie orientée objet :

- Les **design patterns** comme illustration du dogme.
- Les **langages de patterns** : l'instantiation du modèle à une application définie.
- La suite de l'histoire . . .

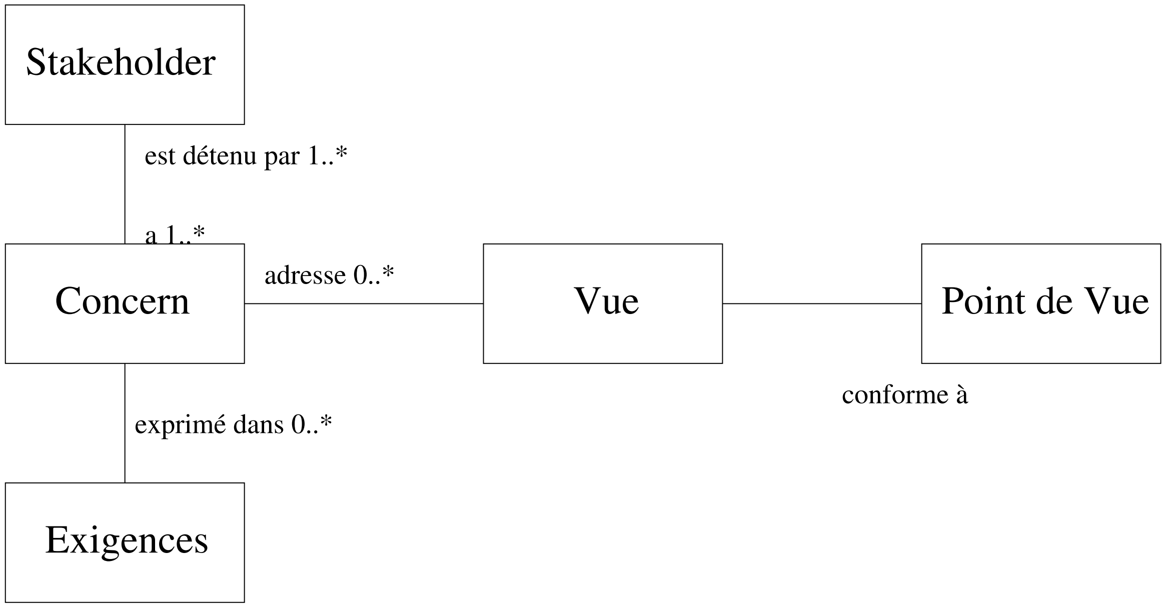
Separation of Concerns

- Objectifs : réduire le *“couplage”* entre les modules et accroître leur cohésion.
- Qu’est-ce qu’un *concern* : exprime un intérêt spécifique sur un point pertinent d’un domaine (système) d’*intérêt* particulier.
- Ce concept s’articule autour de plusieurs questions : Pertinence du système ? Quelle tâche le système accomplit ? Comment le système est-il déployé ?

Concerns

- En relation avec les pôles d'intérêt humain et pas avec l'abstraction
- Les *system stakeholders* : clients, utilisateurs, développeurs et mainteneurs . . .
- Apparaissent à tous les stades du cycle de vie : analyse des besoins, conception, développement, évolution . . .

Un ensemble de conception



Vues et Point de vue(1)

- Une *vue* est un modèle du système. Par exemple dans le passé, le *point de vue* **fonctionnel** et le *point de vue* **orienté données**
- Une *vue* désigne un ensemble de *concerns*
- Les travaux les plus récents sur la description des architectures logicielles, préconisent une déclaration explicite du *point de vue* avant l'utilisation de *vues* pour définir le modèle.
- Les *points de vue* peuvent être assimilés à des entités de premières classes auxquelles sont associés des attributs et des opérations.

Vues et Point de vue (2)

- Une *vue* est l’instance, un *point de vue* est une information réutilisable qui définit la règle pour créer et utiliser une *vue*.
- La construction d’une *vue* du système adresse un ou plusieurs *concerns* relatifs à ce système, dans le contexte courant : conception, fiabilité . . .
- Cette méthode contraint un *vue* à couvrir tout le système pour un *point de vue* donné.

Encore un mot : Aspect ?

Qu'est-ce qu'un Aspect ? :

- Comme une *vue*, il *"adresse"* un *"concern"* particulier mais il n'est pas obligé de couvrir tout le système, il est plus *"léger"*.
- C'est une nouvelle unité de la modularité des logiciels.
- Il gère les points de *"cross-cutting"*.
- Comme les objets, ils se rapportent à la fois à l'étape de *design* et d'*implémentation*.
- Les langages de programmation OA permettent de programmer directement en termes de *design aspects* (comme les LOO).

Quel rôle pour les Aspects ?

- Capture la trace et le support de réalisation d'un système complexe, plutôt que des fragments multiples de code au travers des classes.
- Localise l'implémentation de quelques *design patterns* plutôt que de disperser les champs et les méthodes de ces patterns au travers des classes.
- Factorise la gestion de protocoles d'erreurs.
- Factorise les ressources partagées par des algos concernant plusieurs classes mais dans un seul *"aspect"*.

Motivation

- Exprimer clairement et formellement les points importants mis en avant par la conception.
- Certains de ces points ne sont bien capturés ni par les méthodes OO ni par la conception procédurale.

Quelques principes

- Relations mutuelles entre le processus de conception et les langages de programmation.
  - Conception $\Rightarrow$ éclatement du système en modules de plus en plus petits.
  - Langage de POO $\Rightarrow$ mécanismes qui permettent au programmeur d'exprimer des abstractions de sous-unités du système et de les composer de différentes façons afin de produire le système en entier.
- Une association productive si le langage de POO fournit des mécanismes d'abstraction et de composition qui supportent *"proprement"* les types d'unités (les concepts) qui ont été isolés (cassés) par l'étape de conception.