Service Locator et Injection de dépendances

Problème :

- Un service nécessite d'autres services pour assurer son contrat.
- On veut éviter un couplage fort entre le service client et ceux qu'il utilise.
- Pour optimiser l'utilisation et l'accès uniforme aux ressources les instances de services doivent être partagées entre plusieurs clients.

Orientation de la solution:

- Le client ne doit pas connaitre l'implémentation du service qu'il utilise, il ne doit donc pas le créer lui-même.
- L'ensemble des services disponibles en utilisation doivent être centralisés pour être mutualisés
- Il faut mettre à disposition un mécanisme permettant à un client d'obtenir un service.

Deux patterns permettent de lier les services entre eux à l'exécution :

- **Le service locator**
- **L'injection de dépendance**

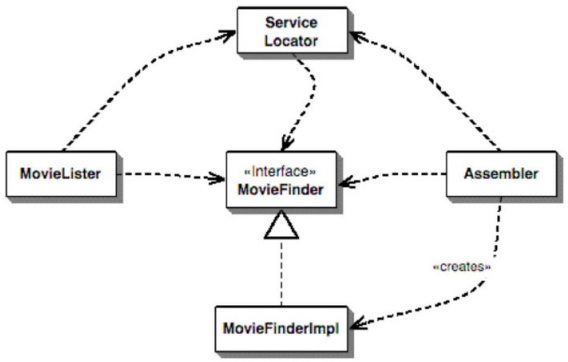
Le service Locator :

Idée principale :

- Le service locator est un objet qui connait et gère tous les services nécessaires à une application.
- Il met à disposition un mécanisme permettant aux clients d'obtenir les services dont ils ont besoin.

Implémentation :

- Le service locator enregistre et gère toutes les instances de services nécessaires à l'application (Registry)
- L'action de création et d'enregistrement de ces services auprès du locator doit être réalisée par un autre objet (« Assembler » ou « Initialiser ») pour éviter les dépendances entre le locator et les implémentations des services.
- Les clients doivent accéder simplement au service locator de l'application (Singleton).



Exemple pris de l'article de M. Fowler

Variantes et évolutions:

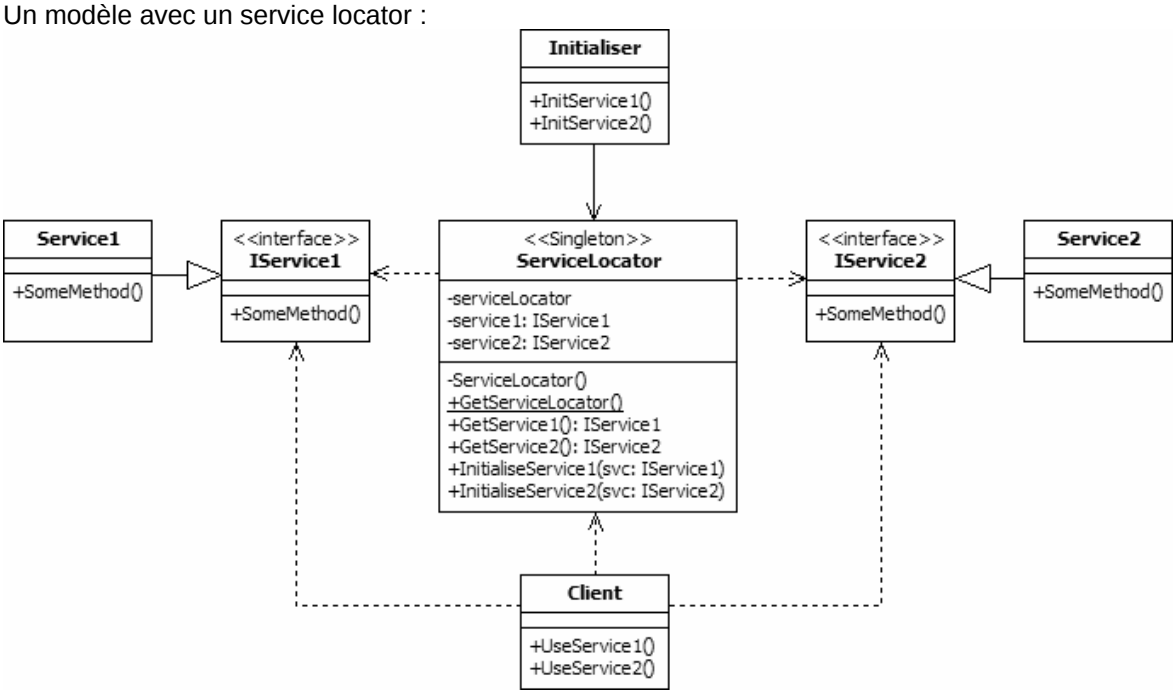
Segregated Interface :
La connaissance du mode d'accès au service locator par tous les clients induit une dépendance entre le client et le service locator, le service locator fournit un grand nombre de services dont tous les clients n'ont pas forcément besoin, pour réduire cette dépendance il est possible d'utiliser une « Segragated Inteface ».
Des interfaces réalisées par le service locator permettent de découper son contrat et aux clients de s'adresser au locator en tant que fournisseur d'un nombre réduit de services.

Service locator dynamique :
A la place de proposer une méthode pour chaque service à fournir, le service locator enregistre les services en les associant à un nom symbolique et fournit les services demandés par les clients à partir de ce nom symbolique.
Les services sont stockés dans une map avec comme clé le nom symbolique et valeur l'instance du service associée à ce nom symbolique.
Le service locator dynamique se limite alors à deux méthodes :

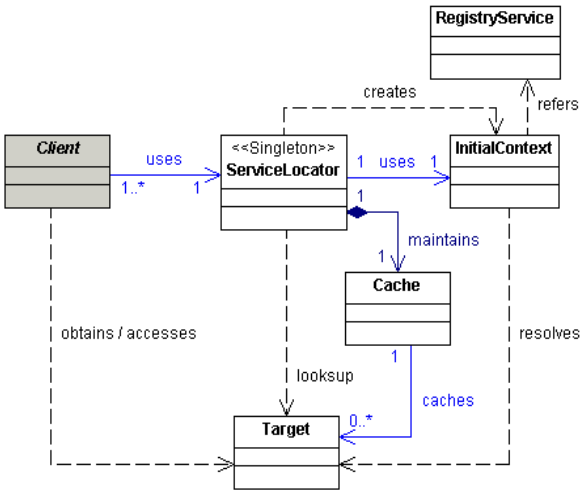
registerService(nomSymbolique, instance)
getService(nomSymbolique) : service

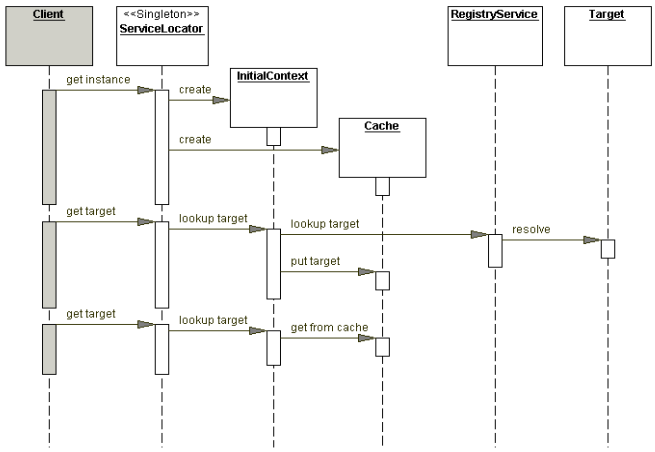
Service locator encore plus dynamique :
Les directives d'initialisation et le mapping des noms symboliques vers leur implémentation sont fournis par un fichier de configuration, un objet utilitaire se charge de lire ce fichier et de réaliser les enregistrements auprès du service locator.

Un modèle avec un service locator :



Le service locator (JEE) :





L'injection de dépendances :

Solution basée sur **IOC** (Inverse of Control)
principe d'Hollywood : "*Ne nous appelez pas, c'est nous qui vous appellerons*"
Framework ou du code non fonctionnel qui se charge de piloter l'exécution principale du programme.
Le concepteur de l'application utilise des API du framework pour déclarer les composants ou blocs de code devant être activés durant l'exécution. Il en résulte un découplage entre les composants qui ne sont liés qu'à l'exécution.

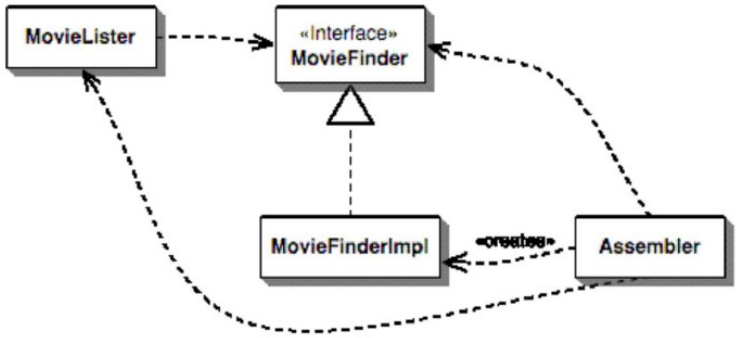
L'injection de dépendances (Dependency Injection) est un mécanisme qui consiste à créer dynamiquement (injecter) les dépendances entre les différentes classes en s'appuyant généralement sur une description ou des directives associant un type de service à son implémentation. Ainsi les dépendances entre composants logiciels ne sont plus exprimées dans le code de manière statique mais déterminées dynamiquement à l'exécution.

Idée principale :

- Un objet indépendant « Assembler » est chargé d'initialiser toutes les dépendances (références à des services) en fournissant l'implémentation correspondante aux clients.
- Pour que l'assembler réalise l'injection il lui faut connaitre les éléments à injecter et à qui. (configuration et/ou marquage par convention).
- Pour réaliser l'injection il faut que le client fournisse à l'« Assembler » un moyen de lui affecter ses dépendances.

Il existe 3 manières de réaliser cette injection de dépendances :

- **L'Injection par constructeur (Constructor injection)**
- **L'injection par setter (Setter injection)**
- **L'injection par interface (Interface injection)**



Exemple pris de l'article de M. Fowler

Injection par constructeur :

- Chaque client déclare un constructeur avec en paramètre les dépendances (interfaces de services) qui lui sont nécessaires.
- Chaque client et implémentation de service devant être injectée (pouvant être les mêmes) s'enregistre auprès d'un container en déclarant le type de service qu'il est en mesure de fournir (interface)
- Le container réalise l'injection de dépendances en construisant les instances de services nécessaires et en les fournissant à leurs clients par l'intermédiaire de leur constructeur.
- L'application utilisatrice demande au container un service, l'instance du service est retournée avec toutes ses dépendances initialisées.

Framework PicoContainer. http://picocontainer.org/ , GoogleGuice

Injection par setter :

- Chaque client déclare des setter sur les dépendances (interfaces de services) qui lui sont nécessaires
- Chaque client et implémentation de service devant être injectée s'enregistre auprès d'un container en s'associant au type de la dépendance (ou dans un fichier de configuration)
- Le container réalise l'injection de dépendances en utilisant des setters pour fournir les dépendances aux clients.
- L'application utilisatrice demande au container un service, l'instance du service est retournée avec toutes ses dépendances initialisées.

Framework Spring http://www.springsource.org/ , GoogleGuice

Injection par interface :

- Des interfaces d'injection sont déclarées pour chaque service à injecter
- Chaque client réalise les interfaces d'injection correspondantes aux dépendances (interfaces de services) qui lui sont nécessaires.
- Chaque client et implémentation de service devant être injectée s'enregistre auprès d'un container en s'associant au type de la dépendance (ou dans un fichier de configuration)
- Pour chaque interface d'injection un injector réalisant l'interface est créé.
- Le container réalise l'injection de dépendances en utilisant les interfaces d'injection pour fournir les dépendances aux clients.
- L'application utilisatrice demande au container un service, l'instance du service est retournée avec toutes ses dépendances initialisées.

Remarques :

- Service Locator et Injection de dépendances assurent un découplage entre clients et services (couplage lâche).
- Le service locator impose au client de demander le service (mode tiré).
- L'injection de dépendances fournit le service au client (mode poussé).
- Les framework réalisant l'injection de dépendances « travaillent dans l'ombre » et leur cycle de vie peut être complexe à appréhender.
- En utilisant le service locator, chaque client est en dépendance avec le locator.
- Injection par dépendance est plus souple que service locator.
- Avec le service locator, il existe un risque de dépendance non résolue à l'exécution (au moment de la demande par le client) et un crash de l'application.
- L'injection par interface est très intrusive et complexe.
- L'injection par constructeur permet d'exposer clairement les dépendances et de rendre certains objets non mutables.
- Pour assurer la souplesse d'utilisation de ces systèmes, la configuration doit être clairement séparée de l'utilisation.
- La configuration par fichier procure une souplesse (pas de recompilation nécessaire sur changement).
- La configuration par code permet d'éviter l'écriture de fichiers de configuration complexes.