Le projet **DELIRE**

Développement par **E**quipe de **L**ivrables **I**nformatiques et **R**éalisation **E**ncadrée

PSp7 – Architecture Logique

ARCHITECTURE LOGIQUE

Qui commence par un rappel sur l'Orienté Objet

A) La crise du logiciel

Dans les années 70, devant l'impossibilité avérée de traiter des problèmes de plus en plus complexes, on a parlé de crise du logiciel.

La problématique était simple :
1) La population des programmeurs augmentait de 4% par an
2) La productivité augmentait de 4% par an
3) Mais la demande de logiciel progressait de 12% par an.

En parallèle le cout de la maintenance devenait prohibitif

La réponse fut « simple » :
1) Il faut mettre en place une notion de réutilisabilité
2) Ce qui conduira aux concepts de la Conception et de la Programmation Orientée Objets.

B) Les paradigmes de la COO et la POO
(J'appelle ici paradigmes l'ensemble des idées et concepts qui permettent de construire cette démarche Orientée Objets)

La Conception et la Programmation Orientée Objets reposent sur quelques paradigmes
1. La modularité
2. Encapsulation et méthodes d'accès
3. La notion de classe abstraite
4. La généricité
5. L'héritage
6. Le polymorphisme
7. La surcharge
8. L'objet
9. Composition

Je n'ai pas de doute sur le fait que ces concepts vous sont plus que familiers, mais néanmoins, je souhaite revenir sur deux notions : la modularité d'une part, l'encapsulation et les méthodes d'accès d'autre part.

Modularité
L'architecture, c'est le choix des composants (ou modules) que l'on va spécifier et réaliser, pour construire un système.
Et pour commencer à penser cette architecture, il faut avoir comme objectif le fait qu'un module est fortement cohérent (grande unité de ce qui est fait à l'intérieur du module) et faiblement couplé (il n'a que peu de dépendances avec d'autres modules).

Lorsqu'on construit l'architecture de son système, on doit prendre en compte 5 critères

1. Décomposabilité modulaire: la décomposition doit permettre de rendre la construction de chaque module indépendante
2. Composabilité modulaire: la conception d'un système par composition de module est facilitée
3. Compréhensibilité modulaire: le but et le fonctionnement de chaque module est aisée
4. Continuité modulaire: un changement mineur n'affecte qu'un seul module
5. Protection modulaire: le traitement d'une condition anormale d'utilisation d'un module est confinée au dit module

Et dans les choix de structuration on doit respecter 5 règles

1. Correspondance directe: adéquation entre la structure modulaire de modélisation et la structure modulaire de construction
2. Peu d'interfaces: chaque module correspond avec un minimum d'autres modules
3. Petites interfaces: échange de peu d'informations
4. Interfaces explicites: les relations entre 2 modules doivent être explicites dans la déclaration des modules
5. Rétention d'information: le choix des informations publiques accessibles par les autres modules

<u>Encapsulation, méthodes d'accès à l'information</u>

L'encapsulation, dans la gestion de classes abstraites, c'est cacher la structure (les attributs) de l'objet et n'accéder à l'information que par des fonctions
Imaginons que nous ayons créé une classe abstraite Individu, et qu'une des fonction soit de donner l'âge de l'individu.
L'implémentation va être :

1. Dans la première version, lecture d'une valeur dans la structure de donnée (attribut « âge »). Problème, il faut changer la valeur de l'attribut à chaque date anniversaire
2. En seconde version, soustraction de la date courante moins la date de naissance. Cette version, plus robuste, nécessite que la classe abstraite Individu accède à un module qui lui rende la date courante.

Donner la visibilité à la structure de données aurait fait que la modification de cette structure (suppression de l'attribut « âge », ajout de l'attribut « date de naissance ») aurait occasionné un travail de réécriture important. Le fait d'accéder à l'âge par une fonction rend la modification locale à la classe abstraite Individu.

L'encapsulation d'un module informatique permet que les clients d'un module puissent être capables d'utiliser tout service fourni par le module, indépendamment de la façon dont le service est implémenté.
Autrement dit, le choix des services qui vont être exposés dans la partie publique du module est le travail d'architecture le plus délicat en phase de conception :

1. Publier tous les services qui vont être nécessaires pour les autres modules. Ne pas publier ce qui devrait manifestement être public aurait deux conséquences :
    a. Duplication des services, induisant une consommation mémoire inutile et un effort supplémentaire de maintenance

   b. Réduction de la cohérence des modules qui ré-implémentent un service hors scope.
2. Ne pas publier ce qui n'est pas nécessaire.
   a. Beaucoup de service peuvent être internes au module et non publics, du fait de la forte cohérence et du faible couplage du module.
   b. Ne l'oubliez pas, publier un service vous impose une contrainte de grande stabilité : un service public est un contrat qu'il faudra respecter dans le temps.

Concept d'architecture pour une maison

Si un jour vous êtes appelés à faire construire une maison, vous aurez à travailler avec un architecte.

Le premier travail de l'architecte est de vous demander de formaliser :

1) Vos besoins
   a. Certains besoins sont évidents : salon, cuisine, chambres, salle de bain…
   b. D'autres sont plus délicats :
      i. Si Madame a l'intention de travailler à domicile, une pièce bureau s'impose
      ii. Si Monsieur fait de la gymnastique à la maison, un local s'impose également
   c. Vos besoins doivent s'inscrire dans la durée.
      i. Oui certes, les 3 petits souhaitent dormir actuellement dans la même chambre, mais qu'en sera-t-il lorsqu'ils auront 13 ans.
      ii. Oui, une seule salle de bains permettrait de faire de sérieuses économie, mais là aussi, lorsque les enfants seront adolescents, c'est fou le temps qu'ils passeront, garçon ou fille, à se pomponner ; et ce jour-là vous regretterez de devoir vous raser dans l'évier de la cuisine
      iii. Avez-vous l'intention de garder la maison une fois les enfants partis ?
2) Votre délai
3) Votre budget (tendu, je vous rassure)

Puis votre architecte va vous demander de lui expliquer comment vous souhaitez vivre

1) Est-il important que cuisine et salle à manger soit séparées, ou bien souhaitez-vous que celui qui prépare le repas soit totalement intégré à la conversation
2) Souhaitez-vous une cave à vin (critère très important)
3) Les toilettes dans la ou les salles de bain, ou séparées ?
4) Si vous ne souhaitez pas que vos enfants aient à terminaison une TV et un ordinateur dans leur chambre, c'est maintenant qu'il faut le prévoir.
5) Avez-vous l'intention de recevoir souvent votre belle-mère?
6) Avez-vous également l'intention de recevoir souvent les amis des enfants ?

L'ensemble de ces discussions a pour objectif de faire ressortir les demandes non exprimées dans les besoins, mais également de mettre en exergue les contraintes que vont induire ces modes de vie sur les choix d'architecture.

Pour la petite histoire, il arrive souvent que ces discussions mettent en évidence le fait que Madame et Monsieur n'ont pas du tout la même approche. Such is live.

Une fois ces données collectées, commence la véritable activité d'architecture.
Le travail de l'architecte consiste a définir le nombre de pièces, leur surface, et les communication entre elles, qui permettent de couvrir la majeure partie de vos besoins et de vos demandes, et qui entre néanmoins dans le délai et le budget proposés.
Plusieurs propositions sont susceptibles de répondre à toutes ces contraintes

1) Faire une immense salle salon + salle à manger + cuisine a d'immenses qualités, en particulier pour les jours où vos enfants organiseront une teuf de 70 personnes. Par contre, il faut que vous acceptiez le fait que cette salle ne sera jamais parfaitement rangée.
2) Prévoir un coin bureau dans votre chambre répond au besoin de Madame, mais suppose que Monsieur puisse dormir avec la lumière allumée et le bruit de l'imprimante.

D'autres problème peuvent se gérer dans le temps.

1) Il est nécessaire d'avoir à terminaison 4 salles de bain. Prévoyons les emplacements, les arrivées d'eau et les cloisons pour ce faire, mais n'en aménageons qu'une seule actuellement. Ceci permettra de gagner en délai et en budget pour la livraison de la maison
2) Le garage peut attendre

Et puis un jour, votre architecte arrive avec des plans. Avant de signer, il vous est conseillé de jouer des scénarios de vie, pour vous assurer que vous n'aurez pas de problème pénible une fois la maison construite ; en particulier à 90 ans avec les escaliers. L'architecture est ce qui va contraindre la suite des évolutions.

Les critères d'architecture

On ne définit pas une architecture que dans le cas du logiciel. L'architecture est la base de conception pour tout projet, et en particulier pour les systèmes complexes, dont la gestion est souvent dénommée gestion de programme.

Les méthodes de management de programme (MMP) structurent le développement de programme autour de l'architecture du système développé

1. Systèmes
2. Sous systèmes
3. Equipements
4. Composants

Dans un projet de type avion de combat, les MMP recommandent de définir les composants de l'architecture système (système, sous systèmes…) selon 6 critères. '

1. Critère fonctionnel
    - Regroupement de fonctions homogènes et cohérentes entre elles, pour simplifier les interfaces fonctionnelles
    - Faire coïncider l'Arborescence Produit et l'Arborescence Fonction
2. Critère organique
    - Chaque constituant est sous la responsabilité d'un acteur du réseau d'acquisition

- Faire coïncider l'Arborescence Produit et Structure d'organisation du réseau d'acquisition

3. Critère de soutenabilité
   - Chaque constituant correspond à un domaine de responsabilité d'un acteur du réseau d'utilisation chargé de la maintenance, du ravitaillement et du suivi technique
   - Faire coïncider l'Arborescence Produit et Structure d'organisation du réseau de soutien
4. Critère de normalisation
   - Des fonctions voisines sont assurées par des constituants identiques de l'Arborescence Produit, choisis dans des listes d'articles standards.
5. Critère de confidentialité
   - Des contraintes de confidentialité conduisent à regrouper des fonctions sensibles ou des technologies avancées
6. Critère de criticité et de flexibilité
   - La mise en évidence de risques importants peut conduire à isoler dans l'Arborescence Produit des constituants critiques pour préparer des solutions alternatives, étudiées dès la phase de conception

<u>L'architecture en informatique</u>

L'architecture, c'est le choix des modules que l'on va spécifier et réaliser, pour construire un système.
Les règles pour l'élaboration d'une architecture sont regroupées dans le concept de génie logiciel, au même titre que le savoir faire en conception de systèmes mécanique est regroupé dans la notion de génie mécanique.
Génie logiciel qui vise la production de logiciels de qualité, c'est-à-dire des logiciels fonctionnels, rapides et performants, fiables, faciles à utiliser, simples à lire et à maintenir, et évolutifs.
On distingue :

1. Des facteurs externes : fonctionnels, rapides et performants, fiables, faciles à utiliser, parce que perceptibles par l'utilisateur
2. Des facteurs internes : faciles à lire et à maintenir, évolutifs, parce que seulement perceptibles par l'informaticien professionnel

A priori, l'utilisateur se moque des facteurs internes, mais la réalité montre que la possibilité d'atteindre les facteurs externes résident dans les facteurs internes : un code lisible, maintenable et évolutif.

Pour votre information, lorsque Grady Booch, un des fondateur d'ADA, et un des pères du langage UML, avait conçu sa méthode d'ingénierie du logiciel, il avait proposé que les modules puissent s'apparenter à 4 types majeurs pour concevoir aisément en O.O.

1. Collections nommées de déclarations
   - Exportent des objets et des types
   - N'exportent pas d'autres unités de programme
2. Groupement d'unités de programmes reliés
   - N'exportent pas d'objets ni de types
   - Exportent d'autres unités de programmes
3. Types de données abstraits
   - Exportent des objets et des types
   - Exportent d'autres unités de programmes

- Ne conservent aucune information d'état dans le corps

4. Machines à états abstraits
    - Exportent des objets et des types
    - Exportent d'autres unités de programme
    - Conservent de l'information d'état dans le corps
    - Chacune d'elles est unique dans le système.

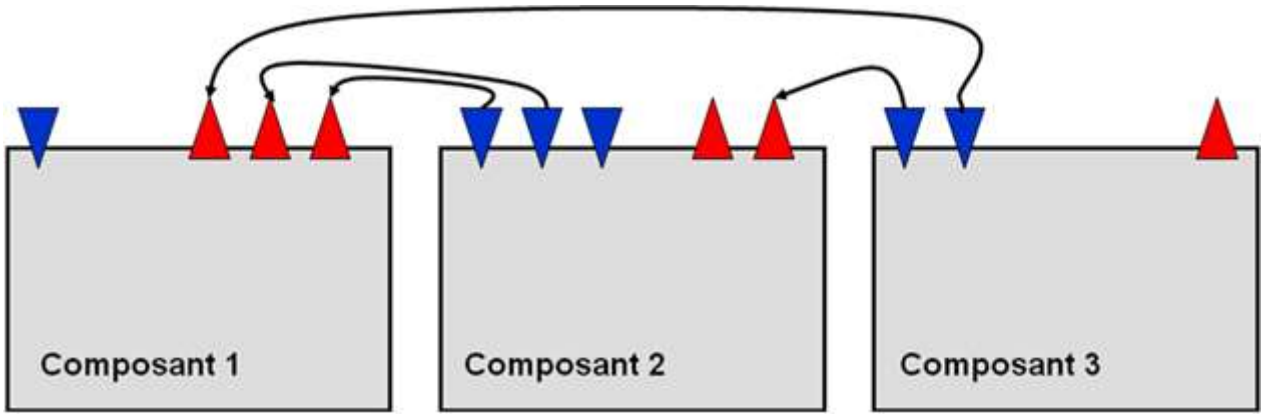
La démarche d'architecture en informatique, va être rigoureusement similaire à l'élaboration d'une architecture dans un quelconque domaine.

1. A partir du cahier de charge, définir les fonction de service (ou d'usage) qui doivent être implémentées
2. Répartir ces fonctions en ensembles cohérents : chacun de ces ensemble définira un premier module
3. Pour chacun de ce module, identifier les services qu'il sera nécessaire d'appeler pour réaliser les fonctions
4. Pour chacun de ces services :
    a. Est-ce un service qui ne sera nécessaire qu'à mon module (faible couplage)
    b. Si oui, est-ce un service qui est de la même nature que les services que j'ai déjà identifiés pour mon module (forte cohérence)
5. Si vous répondez non à une de ces deux questions, à l'évidence ce service doit être fourni par un autre module, en particulier pour des raisons triviales de réutilisation et de minimisation de code à maintenir.
6. Vous vous retrouvez dès lors avec un ensemble de services, ou fonctions, qu'il est nécessaire d'implémenter
7. Et vous pouvez réitérer la démarche.

En résumé l'architecture c'est

1. L'identification des modules, et des services exposés

2. L'identification des liens entre les modules.



Au moment où vous définirez les services nécessaires pour un module, faites preuve de la qualité fondamentale de tout ingénieur : la paresse. Pour être plus explicit, modifiez si besoin très légèrement les services dont vous avez besoin si ceci peut vous permettre d'utiliser un module disponible soit au sein de votre entreprise soit sur le net. Pourquoi faire demain ce qu'un autre a fait pour vous hier. Et souvenez vous qu'acheter un composant pour 1000 € est toujours moins chez que le développer pendant 2 mois, et le maintenir ensuite.

Bien entendu, la démarche que je viens d'exposer est un tantinet réductrice.
Vous serez probablement amenés à retravailler votre architecture en fonction de divers critères :

1. Taille de chargement : si quelques services rarement appelés sont responsables de l'essentiel de la taille de votre module, cela peut valoir le coup de splitter votre module en deux
2. Gestion des droits d'accès : si votre produit peut être utilisé par divers types d'utilisateurs, dont certain n'auront des droits qu'en lecture et d'autre en écriture, il peut être intéressant de splitter votre module en une partie lecture et une partie écriture et modification
3. Si, dans la liste des service que vous exportez, l'un d'eux est beaucoup plus importé que les autres services par les autres modules, il peut valoir le coup de splitter votre module en deux : vous réduirez votre couplage
4. Si l'analyse des savoir faire nécessaires pour développer votre module fait apparaitre deux famille de savoir faire, il faut se poser la question de savoir si là aussi cela ne vaudrait pas le coup de le splitter en deux.
5. Certains modules peuvent avoir un comportement qui est dépendant du système, de la machine, de la base de données ou de la carte graphique.
    a. Il est donc important d'avoir N modules qui exposent les même services, et le bon module sera chargé en fonction de la configuration informatique de l'utilisateur.
    b. Mais le code de ces modules doit être minimum, et tout ce qui est indépendant de la configuration doit être remonté dans un module général.
    c. Ceci est comparable à la notion de périphérique en informatique, qui peuvent être interchangés grâce à la normalisation des interfaces.

Vous avez maintenant, pour chacun des modules de votre architecture :

1. La liste des services qu'il publie, soit comme fonction d'usage pour votre utilisateur, soit comme service pour un (ou des) autre module
2. La liste des services qu'il doit implémenter en interne
3. La liste des services qu'il va importer depuis d'autres modules.

Vous avez encore la possibilité de modifier les noms et les arguments de vos services, voire d'en splitter un en deux, ou au contraire en agréger deux en un. Mais la priorité n°1 est de stabiliser et figer la liste des fonctions que chaque module publie.
Une fois que ce travail est finalisé, vous pouvez ensuite retravailler l'architecture interne de votre module. Ce travail d'architecture interne n'impacte

1. Ni la vue publique des modules (ce que je publie, ce que j'importe)
2. Ni l'architecture des liens entre les modules

Vous avez alors la possibilité de tout ré-architecturer au sein de votre module : le travail d'architecture de vos modules vous a permis d'être isolé du reste du monde.

L'architecture et la distribution des responsabilité

Un des objectifs que permettent des modules bien architecturés (forte cohérence, faible couplage) est de pouvoir distribuer l'architecture interne et la réalisation des composants.
Cette distribution est favorisée par le fait que la grande cohérence cible un savoir faire bien défini.
Cette distribution est également favorisée par le fait que le faible couplage permet à chacun un grand espace de liberté dans un univers globalement clos.
La distribution et le parallélisme des tâches est la condition sine qua non pour maximiser la charge sur le projet.

L'architecture dans votre carrière

Et plus tard
Vous le constaterez vite, la plupart des projets informatique ne partent pas d'une feuille blanche, mais doivent s'inscrire dans un système et une architecture déjà existante.
Ceci signifie que, lorsqu'une demande d'évolution va être formulée, vous pouvez vous retrouver confrontés à plusieurs situations :

1. L'évolution est mineure, ne touche qu'un seul composant. C'est un cas idéal.
2. C'est une évolution importante, mais qui peut se décomposer en un ensemble de modifications mineures sur un certain nombre de composants. Situation simple, pour peu que la gestion de projet soit bien menée.
3. C'est une évolution importante qui suppose une modification en profondeur de plusieurs modules, avec des évolutions couplées et des risques majeurs. C'est à vous de savoir si le jeu en vaut la chandelle.
4. L'évolution impose de devoir ré-architecturer tout le système. En général le coût du chantier est bien supérieur à ce qu'on peut tirer du projet, et l'affaire se terminera par une verrue infâme dans le code. Sans savoir que cette lâcheté par rapport au chantier de ré-architecture risque dans les faits d'entrainer à terminaison des dépenses de maintenance regrettable.

Vous avez peut-être vécu cette situation où on laisse une petite fuite d'eau s'écouler en se jurant qu'il faut traiter le problème, mais pas aujourd'hui, vraiment pas le temps. Et de découvrir un jour que toute une partie du mur est imbibé et à reprendre. Bien sûr, vous seriez scandalisés si, lorsqu'une crique apparaît dans un avion, on se contentait de masquer le problème avec un morceau de scotch. Et pourtant c'est bien souvent ce que l'on fait en informatique, qui par bien des aspects n'est pas une industrie majeure.

Vous le constaterez vite par vous-mêmes, plus l'architecture initiale a été bien conçue, c'est-à-dire :

1. Des modules bien structurés : un objectif et un domaine clairs, une grande cohérence interne, un faible couplage avec les autres modules
2. Un code fait pour pouvoir facilement évoluer
3. Un code bien documenté et lisible, car le code bien structuré est sa première documentation et donne les clés pour le faire évoluer.

plus les intégrations d'évolutions seront faciles et rapides

Ceci dit, ne rêvez pas : faire évoluer un code induit toujours des distorsions dans l'architecture du système, et même si elle a été bien pensée au début, l'architecture finira toujours par avoir piètre mine. Les demandes auront souvent pour conséquence de faire tourner un composant dans des conditions non prévues à la base, composant dont la cohérence, la fiabilité, et la maintenabilité iront en se dégradant.
Simplement, en inscrivant judicieusement les évolutions dans l'architecture existante, vous retardez le moment où il faudra prendre la décision, toujours délicate et couteuse, de devoir totalement réécrire le système qui est devenu in-maintenable.

<u>Retour sur la comparaison avec notre architecte d'immeuble</u>

Dans le chapitre plus en amont, j'avais décrit pour notre architecte en bâtiment une situation idyllique : construire une nouvelle maison.

Dans la réalité, de nombreux chantier ont lieu dans des maisons anciennes à rénover, et dans ce cadre les contraintes sont imposées par les murs porteurs : c'est donc le choix de vie qui doit s'adapter à la maison, et non l'inverse. En général les architectes sont très créatifs, mais dans un cadre qui impose néanmoins ses contraintes.

Si vous vivez cette situation de chantier de rénovation, vous pourrez toujours demander à ce qu'on modifie la position d'un mur porteur, qu'on surélève la maison où qu'on creuse en sous sol. Dans le bâtiment tout est possible, mais vous entrez alors dans une zone de chantiers très onéreux. Et ne vous étonnez pas de voire apparaître des fissures dans vos murs pendant quelques années.
Dans le bâtiment comme en informatique, modifier une architecture existante est une opération complexe, coûteuse, et déstabilisante.

Et pour information, dans l'industrie le partenaire qui est à la source d'une modification d'architecture (une fois la phase de preliminary design achevée) est celui qui finance le coût de modification : chez lui et chez les autres partenaires.

L'architecture dans DELIRE

J'ai globalement conscience que la notion d'architecture vous est assez peu familière.

Bien entendu, je ne peux pas vous proposer une architecture générique qui fonctionne pour chacun des projets, mais j'ai tendance à penser qu'une analyse pourrait permettre de mettre en évidence quelques composants que je vous livre ::

1. Un ou des composants de dialogue et de présentation
2. Une bibliothèque de services élémentaires
3. Une bibliothèque des widgets de base, des standards de présentation (police de caractère et couleur, positionnement des fenêtres…), des logos.
4. Un gestionnaire de cohérence des informations saisies par l'utilisateur
5. Un composant de messages, pour ceux qui veulent supporter plusieurs langues pour leur produit
6. Un composant de gestion du dictionnaire
7. Un composant de sécurité
8. Un ou des composants de gestion de base de données
9. Un buffer de données, pour éviter de ré exécuter systématiquement les mêmes requêtes à la base…

La construction de cette architecture devrait vous permettre de distribuer le travail en fonction des appétences de chacun, de construire des plans de test de vos composants. Bref de faire du concurrent engineering.

De toute façon, l'architecture de votre produit finira bien par émerger. Mais globalement au bout d'un effort plus long, en étant souvent provoquée par les attentes de tout un chacun. Et le résultat de ce travail ne respectera peut être pas les critères de base d'une bonne modularisation : forte cohérence et faible couplage des modules. Bref, un coût supplémentaire pour votre projet à court terme dont vous vous seriez bien passé

Dans l'hypothèse fictionnelle où vous commercialiseriez votre produit, devriez ensuite le maintenir et le faire évoluer, les problèmes d'architecture et l'absence possibles de documentation (en particulier des services exposés) vous pénaliseraient également à plus long terme.

Ok, ok Antoine, mais physiquement là qu'est-ce qu'on fait ?

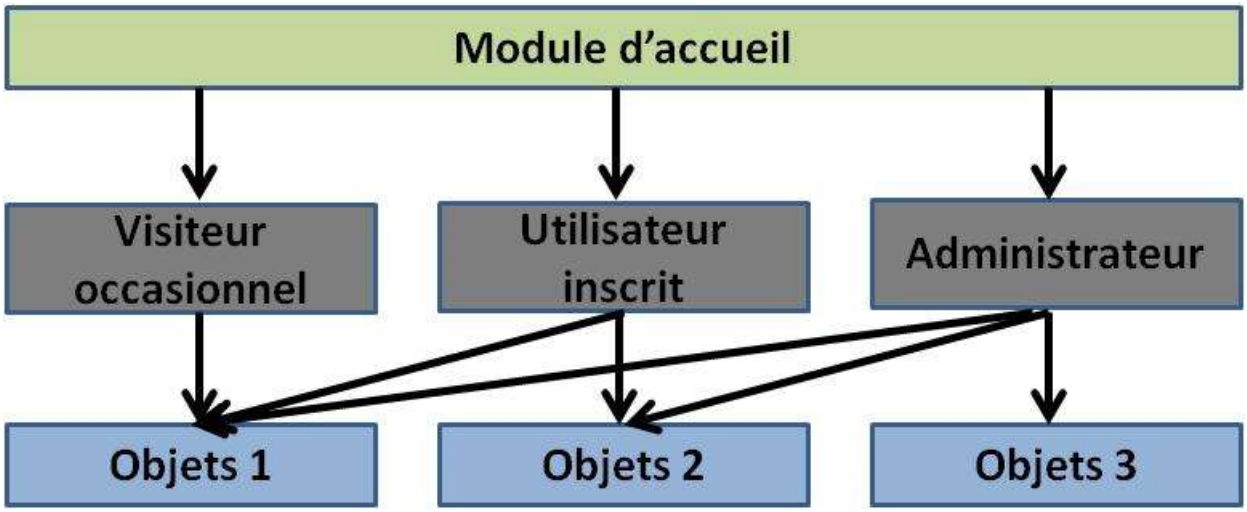
Ma recommandation est de partir du haut, par des modules liés aux rôles des utilisateurs. On peut par exemple en imaginer 3 :

1. Visiteur occasionnel
2. Utilisateur inscrit sur le site
3. Administrateur du site.

On peut faire l'hypothèse qu'il existe un mode d'accueil et de pilotage de dialogue qui va vous orienter vers le bon module en fonction du rôle de la personne connectée. On peut également faire l'hypothèse que les fonctions disponible pour un visiteur occasionnel sont également disponibles pour un utilisateur inscrit sur le site ou un administrateur du site.

Ma recommandation est également de partir du bas, et plus exactement du modèle logique de données. Ayant identifié les grands objets, définissez le ou les modules qui vont les gérer. On va imaginer que ce découpage vous conduit à 3 modules principaux. On peut par exemple imaginer que le premier module est accessible à tous, le second uniquement à un utilisateur inscrit sur le site ou un administrateur du site et le troisième uniquement à un administrateur. .

Vous obtenez alors une architecture de ce type



,

Le travail va consister à bien identifier les services délivrés par les modules Objets. Vous pouvez pour ce faire proposer des services de très bas niveau (quasiment accès à l'attribut de l'objet, ou au contraire viser des services de haut niveau : service d'authentification, service de connexion, liste des objets des type données dans la base….

Là c'est le critère forte cohérence faible couplage (et pour ce faire liste de services réduite) qui doit vous guider.

Je vais faire l'hypothèse que vous avez finalisé la définition de

Il ya a probablement des services communs entre

D'autre part, pour ne pas le dupliquer, il serait bon que le service de connexion soit localisé dans le module d'accueil, ou à tout le moins être factorisé.

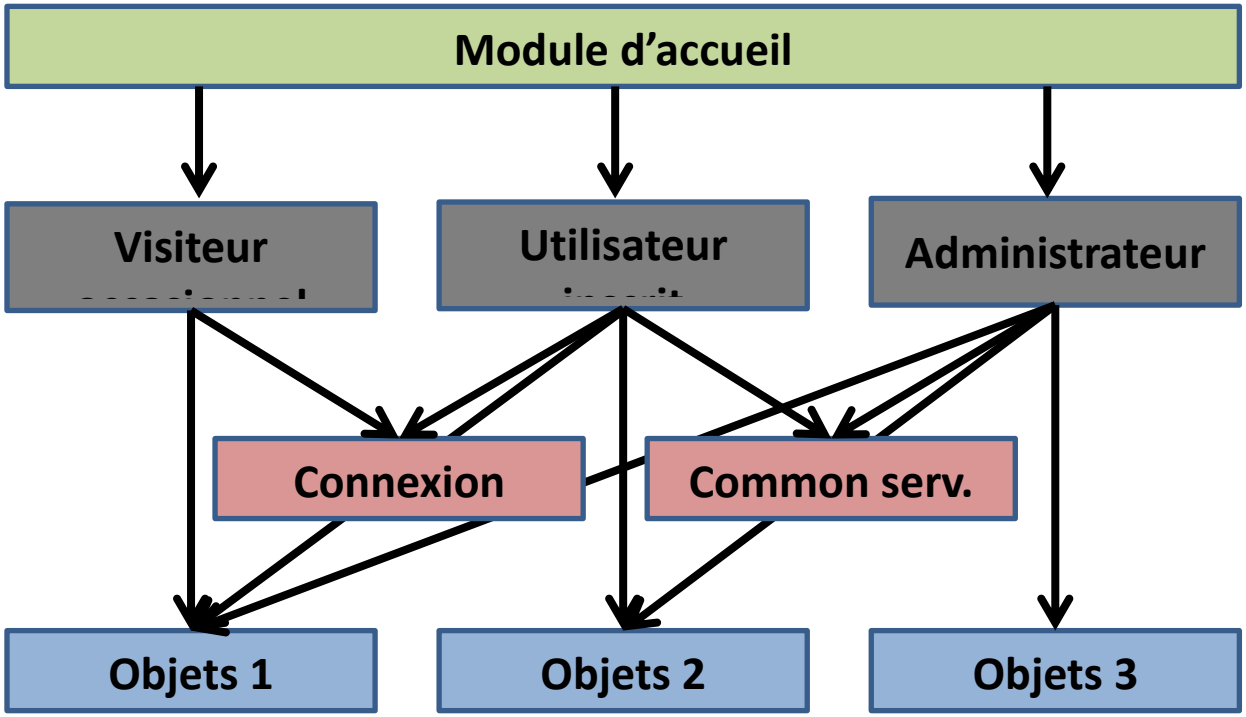
Celui-ci devra donc pouvoir accéder à 

. De facto on identifie de nouveaux liens.

En balayant à nouveau les fonctions nécessaires pour exécuter

en vous appuyant sur les scénarios fonctionnels, vous allez trouver encore des services à rendre commun…

D'où une nouvelle architecture



Les questions que vous devez vous poser ensuite sont :

1. Les modules sont ils fortement cohérents (un seul savoir faire nécessaire pour les prendre en charge)
2. Les modules sont ils de taille raisonnable

Si besoin, il faut continuer à utiliser la paire de ciseaux à module.

Souvenez vous que ce qui vient d'être défini est juste l'illustration d'une démarche, pas l'architecture que vous devez produire. Bon courage.