
# Le projet **DELIRE**

# **D**éveloppement par **E**quipe de **L**ivrables **I**nformatiques et **R**éalisation **E**ncadrée

# Rôles et Livrables



© 2003-2017 **CARAPACE**

L'**Architecture Logique** décrit d'une manière symbolique et schématique les différents éléments de votre système informatique, leurs interrelations et leurs interactions. Contrairement à l'Architecture Fonctionnelle, l'A.L. ne décrit pas ce que doit réaliser le système mais plutôt comment il doit être conçu pour répondre à ses spécifications.

Pourquoi l'Architecture Logique doit être une contrainte préalable ? Parce que sa qualité détermine l'intégrité conceptuelle du système, qui a son tour détermine la qualité finale de celui-ci.

- Une architecture bien pensée fournit la structure nécessaire pour maintenir l'intégrité conceptuelle s'un système depuis es plus hauts niveaux jusqu'à la base. Elle fournit une ligne de conduite aux programmeurs, à un niveau de détail adapté aux techniques de ceux-ci et au travail de base.
- Elle partage le travail afin que plusieurs développeurs, en plusieurs équipes de développement, puissent travailler de manière indépendante.

Une bonne architecture facilite la construction, alors qu'une architecture ratée peut rendre la construction pratiquement impossible.

- Les modifications architecturales coutent cher si elles sont réalisées en phase de Réalisation. Elle coutent encore plus cher en phase de convergence, et sont dramatiques en phase de maintenance, une fois le produit déployé en production.
- Une modification architecturale traduit généralement une erreur de conception
  - Une erreur de conception est facile à trouver mais difficile à corriger
  - Une erreur de codage est difficile à trouver mais facile à corriger
- Une modification architecturale, qui semble mineure au départ, peut entrainer par propagation des modifications profondes du système, entrainant des coûts de réarchitecture, de redéveloppement, de reconvergence et de restabilisation du produit en clientèle non anticipés.

La conception générique consiste à développer le squelette technique d'un projet, garant de son intégrité conceptuelle future. Elle conduit à identifier des composants qui sont connus par leur interfaces

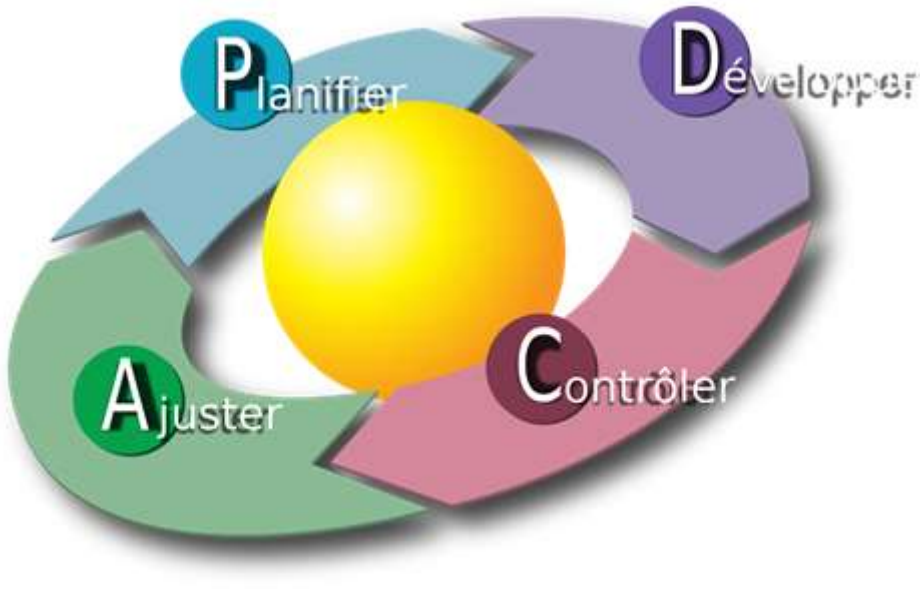
En phase de Structuration, ces composants seront détaillés par les développeurs en charge de chaque composant, en s'appuyant sur

- La ou les finalités du composant
- Les interfaces à fournir
- Les interfaces fournies par les autres composants sur lesquels il est autorisé à tirer
- Les technologies disponibles

.


**© 2003-2017 CARAPACE**

**Le rôle de l'Architecte**
Le travail de l'Architecte, comme celui du Designer, est un travail chronophage et difficile. Et l'angoisse de la page blanche peut paralyser
Comme toujours dans l'industrie, la méthodologie à suivre est une roue de Deming : on fait, on analyse le résultat, on identifie els problème, on définit comment améliorer




Votre rôle est de faire un bon produit. Bon ?

La finalité d'un produit est de répondre aux besoins du client. Il pourrait donc être tentant de concevoir l'Architecture Logique du produit en la calquant sur l'Architecture Fonctionnelle définie par le Designer : grosso modo, un composant par commande majeure de l'Architecture Fonctionnelle et hop, on peut passer au coding : c'est une approche court terme.

L'Architecte doit certes définir une architecture qui supporte l'Architecture Fonctionnelle telle que définie par le Designer, mais il doit prendre en compte de nombreuses autres dimensions dans son travail de conception.

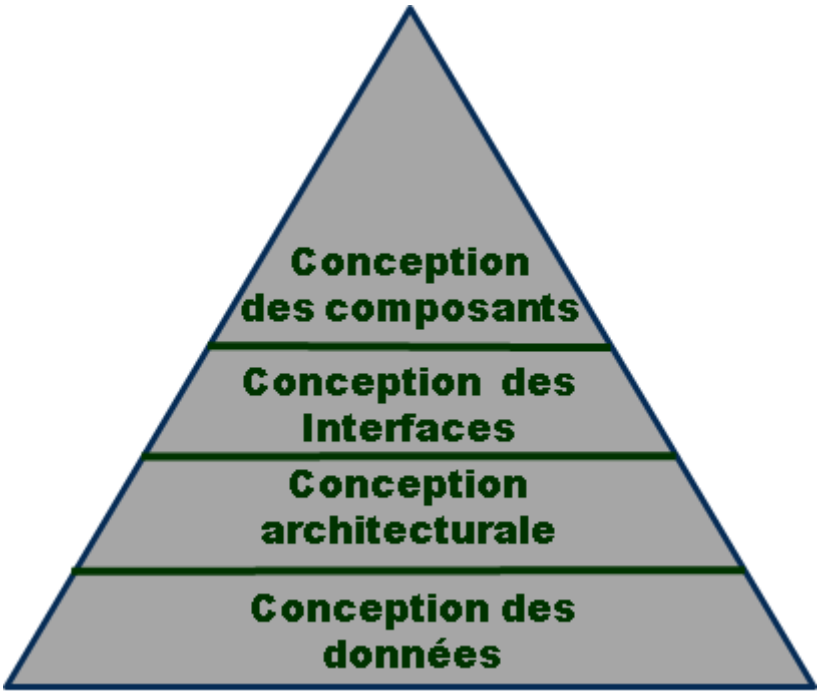
Le rôle de l'architecte est de

- Définir une A.L, Architecture Logique (ou Logicielle), qui satisfasse les exigences du client
- Définir une A.L. qui satisfasse les contraintes du système
- Défendre l'intégrité conceptuelle de son architecture logique
- Critiquer le résultat de son Architecture, la raffiner
- Communiquer et promouvoir son architecture logique

Le travail d'Architecture Logique se décompose en 4 phases

- La conception des données
- La conception architecturale
- La conception des Interfaces
- La conception des corposants


© 2003-2017 **CARAPACE**




En d'autres termes, la conception des composants du système, qui est la finalité de l'Architecture Logique, n'en est également que l'étape ultime. Dur, dur…


© 2003-2017 **CARAPACE**

**Définir une A.L, Architecture Logique (ou Logicielle), qui satisfasse les exigences du client**

Les exigences fonctionnelles du Client sont exprimées dans le Cahier des Charges, qui explicite de façon plus ou moins détaillé le processus du Client, et se traduisent par des commandes qui vont permettre, au travers d'un Workflow, de supporter ledit processus. C'est le travail du Designer, et on peut lui donner quitus sur le fait que c'est son savoir-faire et que le résultat de son Architecture Fonctionnelle, c'est-à-dire l'arborescence de ses commandes, supportent le Workflow visé.

Par contre, ce qui n'est pas du ressort du travail du Designer est de définir le modèle de données de l'application, qui va permettre de :

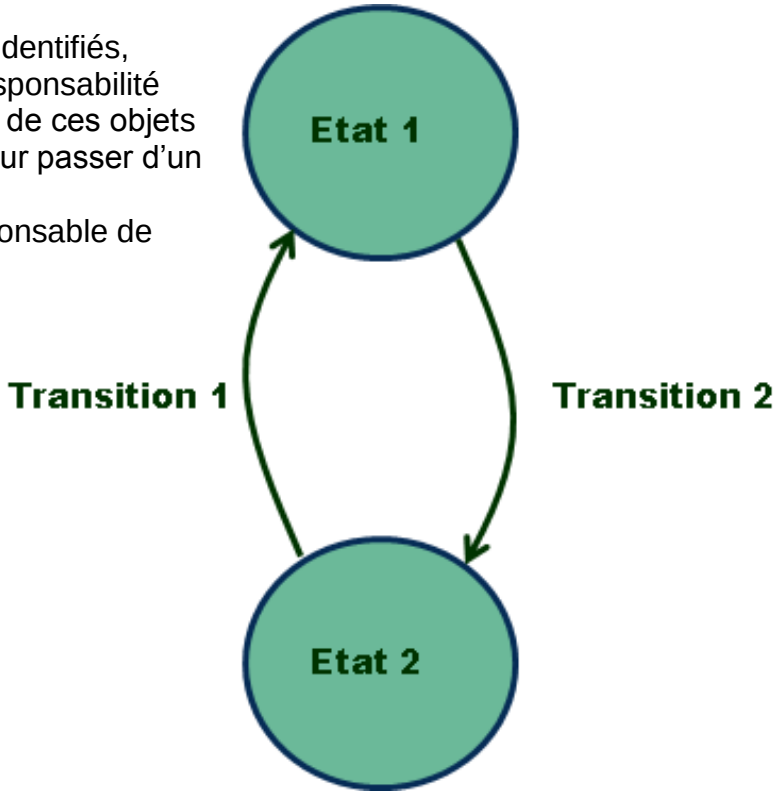
- Présenter les informations
- Modifier ces mêmes informations
- Et stocker le résultat de la session dans la base de données.

Le piège traditionnel, dans la conception d'un modèle de données, est de se précipiter sur une liste d'attributs. La démarche est un peu plus complexe.

Dans le modèle d'analyse, qui va conduire aux Spécifications du Système, on distingue

- La spécification des données : le diagramme Entité – Relations
- La Spécification du contrôle : le diagramme Etats – Transitions
- La Spécification des Processus : le diagramme des Flux de Données.

Une fois les objets identifiés, ce sera de votre responsabilité d'identifier les états de ces objets et les transitions pour passer d'un état à un autre.
Je vous laisse responsable de ce travail




C'est en général dans la phase d'élaboration du diagramme Etats – Transitions que se logera l'identification des attributs de l'objet.


© 2003-2017 **CARAPACE**

Pour l'instant contentons-nous d'identifier les objets du système.
En première approche, il y a 3 typologies d'objets principales dans notre produit

- L'arborescence de l'hôpital
- Le personnel de l'Hôpital
- Le patient

L'arborescence de l'AP-HP

- Des hôpitaux, rattachés à l'AP-HP
- Des pôles, rattachés à l'hôpital
- Des services, rattachés a des pôles
- Des unités hospitalières, rattachées à des services
- Des unités de soins, rattachées à des unités hospitalières

Mais pour chaque hôpital, il faudra également créer (dans une version future) l'arborescence des services administratifs

- Direction financière
- Direction juridique
- Direction des ressources humaines…

Le personnel de l'hôpital

- Les médecins
- Les infirmiers
- Les secrétaires
- Les personnels de laboratoires
- Les administratifs (qui gèrent l'hôpital : direction financière, direction des ressources humaines, direction juridique……)
- Le data administrateur, qui gère la base de données….

En règle générale, le personnel va avoir

- Un rôle: médecin, infirmier, secrétaire, laborantin, administratif, data administrateur…
- Potentiellement une spécialité :
    - Neurologie cardiologie, traumatologie… pour le personnel soignant
    - Ressources humaines, droit, finance… pour le personnel administratif

Le patient (en fait le Dossier Médical Personnalisé)
Il est composé

- Des informations socio démographiques du patient
- De l'ensemble des documents rattachés à ce patient

Un document c'est :

- Un fichier (word pour un diagnostic ou une posologie, avi pour un IRM, jpeg pour une radio, ….) stocké dans le Vaut attaché à la base de données
- Un certain nombre d'informations, extraites dudit document, qui synthétisent les informations du fichier :
    - Ce sont ces informations qui seront accédées lors de la consultation du DMP au travers d'un Smartphone


© 2003-2017 **CARAPACE**

- Ces informations sont obtenues
  - Par saisie manuelle
  - Automatiquement à partir du contenu du fichier.

Une fois les principaux objets identifiés, il faut établir les relations entre ces objets :

Les liens dans l'arborescence de l'AP-HP

- AP-HP – Hôpital
- Hôpital – Rôle
- Rôle – Service
- Service Unité Hospitalière
- Unité Hospitalière – Unité de soin

Un personnel est rattaché à un nœud de l'arborescence de l'hôpital

- Mais bien entendu, on ne peut pas rattacher quelqu'un ayant une spécialité de neurologie à un service de cardiologie.
- Ceci signifie qu'à chaque nœud de l'arborescence de l'hôpital est rattachée une spécialité
- Ma recommandation est donc que la spécialité soit un objet dans le système
  - Chaque nœud de l'arborescence est rattaché à une spécialité
  - Chaque personnel de l'hôpital est rattaché à une ou plusieurs spécialités
  - Pout être rattaché à un nœud de l'arborescence de l'optimal pointant sur une spécialité S, il faut que la personne possède S dans sa liste de spécialité

Un dossier médical est rattaché à un nœud de l'arborescence de l'hôpital.

Et chaque nœud de l'arborescence de l'hôpital doit avoir un responsable

Identifier les relations c'est bien, identifier le sens des relations c'est mieux.
Prenons l'exemple du lien Pôle – Service.

- Si c'est le pôle qui pointe vers le service,
  - Cela nécessite de gérer une liste de services dans les sonnées du pôle.
  - Rien n'interdit à deux pôles de pointer vers le même service
  - Rien n'interdit à un service de ne pas être rattaché à un pôle. o
- Si c'est le service qui pointe vers le pole
  - Cela évite d'avoir à gérer une liste supplémentaire
  - Cela permet de garantir que chaque service sera rattaché à un pôle
  - Cela permet de garantir que chaque service ne sera rattaché qu'à un seul pôle

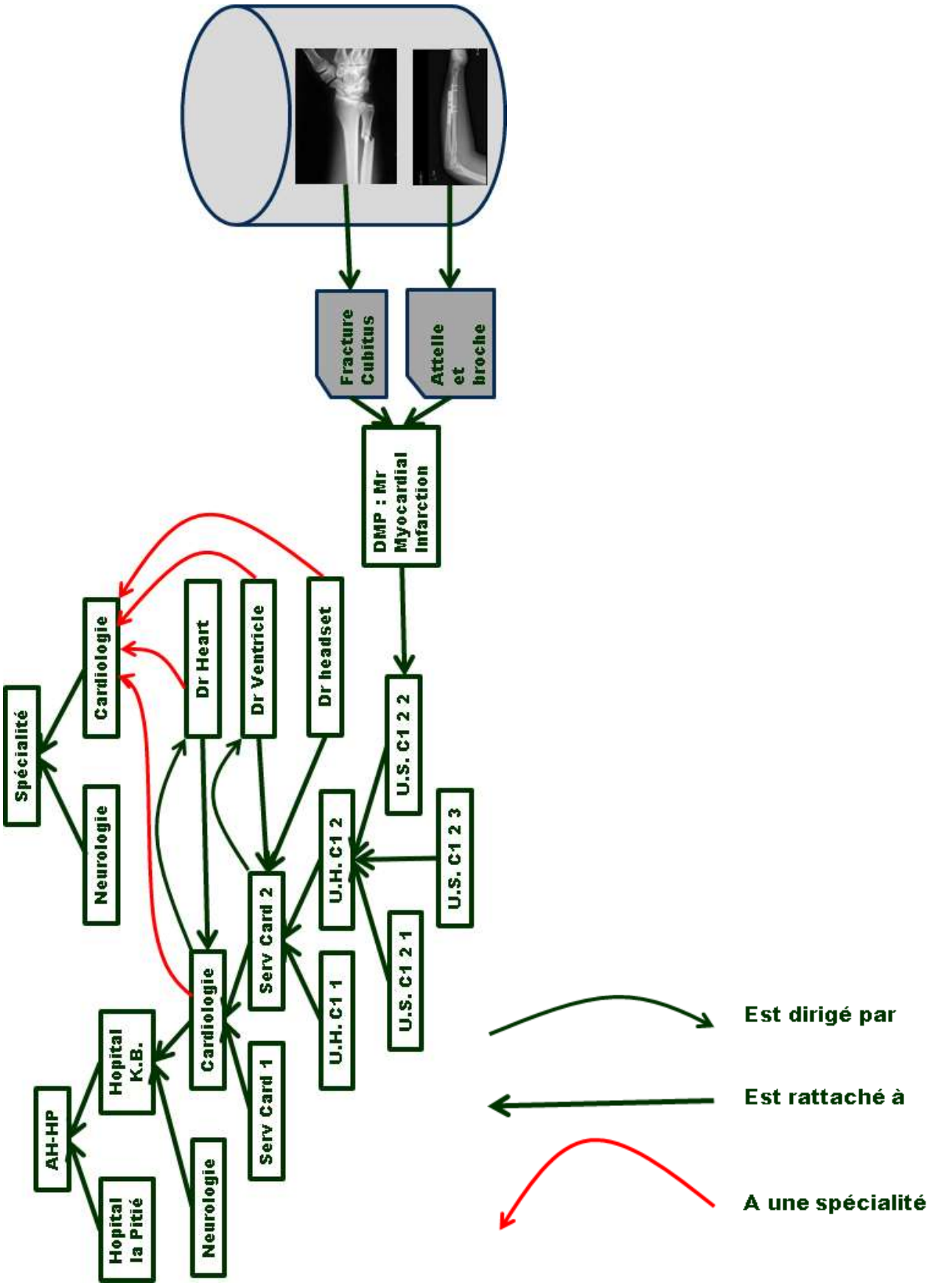
Une démarche similaire conduit à ce que

- Ce soit le personnel qui pointe vers un nœud de l'arborescence et non l'inverse
- Ce soit le DMP qui pointe vers un nœud de l'arborescence et non l'inverse


**© 2003-2017 CARAPACE**

Une fois tout ceci réalisé, on peut commencer à cartographier les objets :

.





© 2003-2017 **CARAPACE**

Tout ce travail, de définition du Data Model est ensuite formalisé dans ce qu'on appelle le dictionnaire de données.
Ce dictionnaire de données est généralement décrit en XML

C'est le référentiel de l'application.
Il permet :

- De générer les structures de donnes en base de données
- De générer les structures de données en mémoire
- De définir les méthodes et les règles attachées à chaque objet
- De définir des outils de validation, des outils de print, des outils de navigation…r
- De pouvoir définir des mécanismes de présentation
- C'est à partir de ce XML et du XML d'un autre hôpital qu'on construira un outil de conversion de données : par exemple un échange entre ce format, qui est celui de votre produit, et le format de l'hôpital d'Innsbruck, pour transférer les données du petit Karl Marx.

La probabilité pour que vous atteigniez ce niveau de généricité dans le projet DELIRE est assez faible.


© 2003-2017 **CARAPACE**

**Définir une A.L. qui satisfasse les contraintes du système**
C'est la seconde question que vous devez vous poser : quelle sera l'architecture de votre système à terminaison.
Et ce pour une raison simple : pour minimiser le travail de redesign et de recodage entre chaque version de votre produit.
Le travail de l'Architecte est un travail difficile et chronophage. Tout ce qui peut être évité est bon à prendre

Reprenons notre problème et posons quelques hypothèses

- Un DMP, c'est 20 fichiers de 5 méga-octets chacun, soit 100 méga-octets
- Il y a 20 millions de patients qui sont géré par l'AP-HP, soit 2 téra-octets de données à conserver
- Il y a 100,000 membres du personnel à l'AP-HP mais 80% de l'activité se fait de jour
- Simultanément i y a 5000 personnes en cours de traitement dans un hôpital donné.

2 téra-octets ce n'est pas grand-chose, inutile d'archiver cela sur des cassettes ou autre.
Par contre corrompre la base de données serait dramatique. Il est donc nécessaire de répliquer la base de données

- Pour redémarrer à chaud si la machine de gestion de données se casse la figure
- Pour redémarrer à froid si le Datacenter où est stockée la base de données subit un problème (incendie, inondation, attentat…)

J'ai bien dit LA machine de gestion de données. Ce qui est sous-entendu est le fait qu'il y a une seule base de données pour toute l'AP-HP. Ce qui signifie que cette machine est accessible depuis l'un quelconque des hôpitaux, et donc n'est pas sur le réseau local.

J'ai 100,000 personnes qui peuvent être connecté sur mon application mais je peux penser que raisonnablement 20,000 personnes utiliseront celle-ci simultanément. Et ce chiffre tombe à moins de 5,000 durant la nuit.

- Je peux certes dimensionner ma machine d'application pour traiter simultanément 20,000 utilisateurs
- Mais durant la nuit je suis surdimensionné
- Et je suis un peu niaiseux si, pour une raison quelconque, plus de 20,000 membres de l'AP-HP se connectent simultanément.

J'ai donc le sentiment que je peux avoir plusieurs machines d'application qui tournent simultanément

- Si j'ai un sous charge, la nuit, je peux diminuer mon nombre de machines d'application
- Si j'ai une surcharge, je peux au contraire augmenter diminuer mon nombre de machines d'application

Les personnes qui se connectent sur l'application doivent pouvoir le faire depuis leur PC dans leur bureau, depuis une tablette s'ils sont dans les chambres d'hôpital ou dans une unité de soin, depuis leur Smartphone s'ils sont à l'extérieur.


**© 2003-2017 CARAPACE**


Dans ce cadre, une application Web me semble pertinente.

- Les requêtes depuis le client vers le serveur seront faites en http
- Les réponses du serveur seront des fichiers html.
- Il n'y a pas de code installé '(en dehors du browser) ni de données persistante sur les machines clientes (PC, Tablette, Smartphone)

Et dans ce cadre, mes machines d'application n'ont pas vocation à être localisées dans les différents hôpitaux. Il me semble plus pertinent de les faire fonctionner dans le même Datacenter que celui où se trouve ma machine de gestion de données : les échanges entre les machines d'application et la machine de gestion de données se feront sur un LAN

L'application est la même pour tous les hôpitaux de l'AP-HP : les machines d'application peuvent donc servir indifféremment un membre de KB (Kremlin Bicêtre), de Necker ou de la Pitié-Salpêtrière.

Je fais le choix de faire tourner des machines pouvant servir simultanément 1,000 utilisateurs.

- Dès que l'on approche de 800 (puis 1800, 2800, …) utilisateurs, je démarre une nouvelle machine d'application
- Dès que l'on redescend en deça de 3500 (puis 2500, 1500 …) utilisateurs, j'interromps une machine d'application
- Le nombre minimum de machine d'application tournant simultanément est de 2 : cela me permet d'avoir au moins une machine qui fonctionne si l'autre se casse la figure, et je n'ai donc pas d'interruption de service

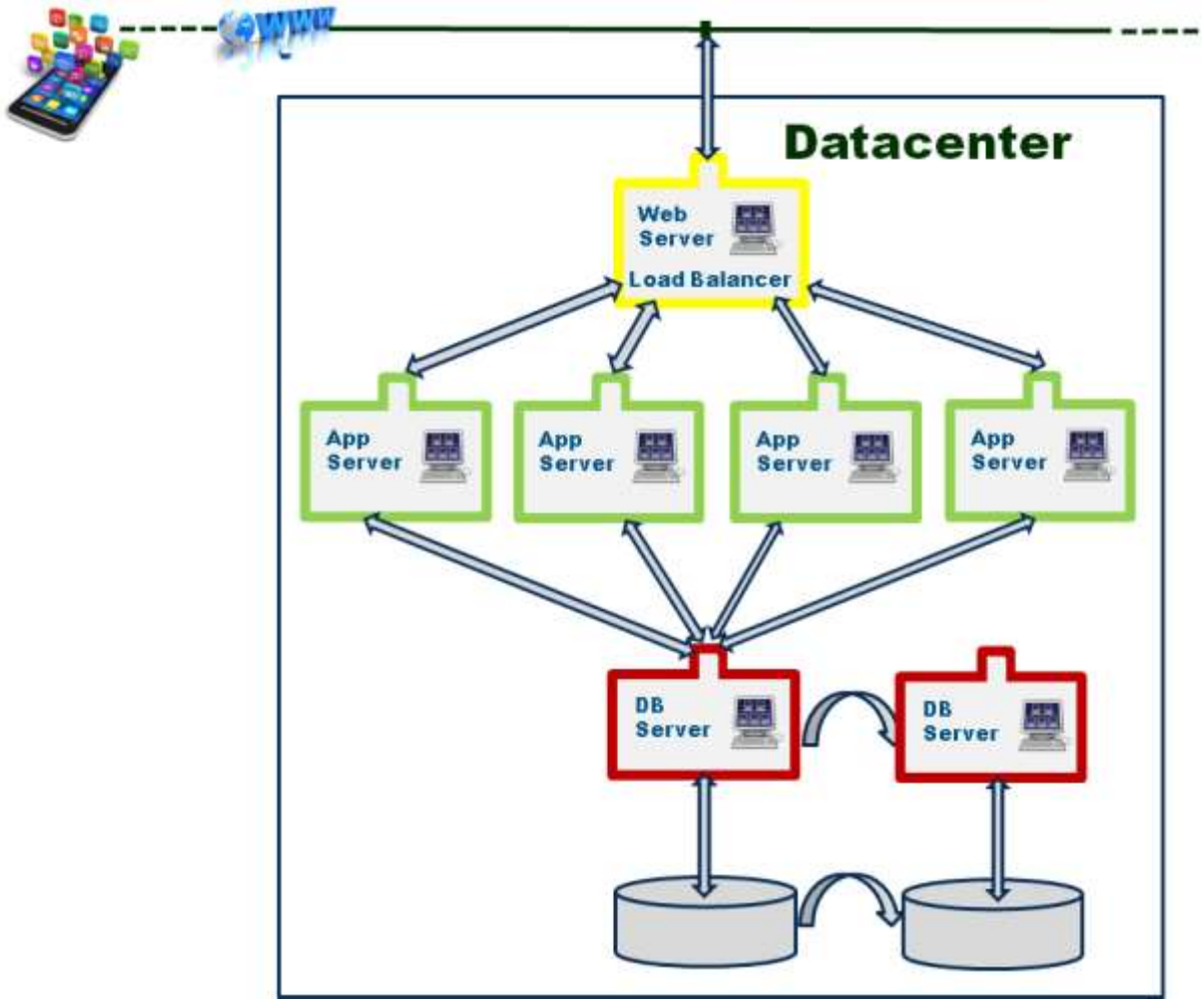
Ce qui signifie que je dois avoir un Load Balancer pour gérer de façon pertinente la charge de mes machines.

Ma machine de gestion de données, et la base de données qui lui est attachée, sera dupliquée de façon synchrone sur le même Datacenter. La base de données sera dupliquée de façon asynchrone sur un autre Datacenter.

Mais Antoine, pourquoi faire tout ce travail, nous n'allons tout de même pas faire tourner notre application chez AWS 'Amazon Web Services) ?
Non certes, mais en faisant tout ce travail, vos définissez un cadre

- Pour identifier les savoir-faire à mobiliser pour construire votre application
- Pour structurer vos composants, de façon à ce que la migration vers cette architecture, en version 2 ou e version 3, se passe sans heurt et sans coût excessif.


**© 2003-2017 CARAPACE**




J'ai dit précédemment que la taille d'un DMP pouvait être d'un ordre de grandeur de quelques centaines de méga-octets.

- Si j'ai une liaison de 3 Giga Bits entre mon hôpital et mon Datacenter
- Si j'ai 600 personnes connectées simultanément sur l'application au sein de mon hôpital
- Il me faudra 3 minutes pour décharger le DMP complet d'un patient.

Dans la réalité

- Les médecins de l'hôpital consultent les DMP des patients hospitalisés dans leur hôpital
- On sait quels sont les patients hospitalisés
- Les fichiers s DMP ne sont pas modifiables

On peut donc avoir pré-recopié dans un Vault répliquant sur le LAN de l'hôpital une copie des fichiers du DMP des patients en cours d'hospitalisation.

.


© 2003-2017 **CARAPACE**

**Défendre l'intégrité conceptuelle de son architecture logique**
L'intégrité, conceptuelle c'est la capacité à protéger la robustesse tant de l'Architecture Logique que du modèle de données.
La méthode la plus efficace pour ce faire est l'encapsulation

- Encapsulation des méthodes :
  - En d'autres termes les méthodes qui n'ont pas la nécessité à être publiques sont internes au composant et inaccessibles depuis l'extérieur
  - Une méthode est publique si elle est appelée par une commande
  - Ou si elle est appelée par un autre composant
- Encapsulation des données :
  - Un attribut de l'objet est modifié par l'appel d'une méthode
  - Aucune donnée n'est publique
  - La modification de la structure de données d'un objet n'impacte que le composant qui encapsule l'objet

Au travers de ce qui vient d'être dit, on voit clairement des dégager deux objectifs

- Identifier les Interfaces, c'est à dire les méthodes publiques (défini dans le paragraphe « Rôle de l'Architecte » comme l'étape 3 de l'Architecture Logique)
- Définir les composants de la solution, qui encapsule les méthodes et les données (défini dans le paragraphe « Rôle de l'Architecte » comme l'étape 4 de l'Architecture Logique)

Comme je le répète souvent, la première qualité d'un ingénieur c'est la réutilisation.

- Certes, la probabilité de trouver l'architecture de votre système déjà définie est assez faible
- Mais par contre, vous pouvez sans vergogne vous inspirer des styles architecturaux reconnus.

Un style architectural définit quels sont les composants, les connecteurs et les contraintes de l'architecture du système.
Les plus connus (ou reconnus) sont :

- Pipe / Filtre
- Client / Serveur
- 3 Tires / N Tiers
- Multi Couches
- MVC
- Cloud Computing

Identifier les interfaces reste d'une complexité raisonnable.
En déduire les composants pertinents est nettement plus problématique

Au-delà des difficultés intrinsèques du composant

- Fortement cohérent et faiblement couplé
- Permettant de distribuer le travail
- Permettant d'encapsuler les méthodes et les données
- Cohérent avec les contraintes du système

S'adjoint une autre difficulté : à quel composant attribuer certaines méthodes ?


**© 2003-2017 CARAPACE**


Prenons un exemple : quels DMPs je peux consulter et modifier ?

En première approche j'aurais tendance à rattacher cette méthode au composât quii gère le médecin.
Mais on s'aperçoit que la question se pose également à l'infirmier.
Ok, dois-je rattacher l'interface à l'objet « Personnel soignant » dont dérivent le médecin et l'infirmier ?
Pas sûr, le personnel de laboratoire en hériterait

En fait la réponse est intrinsèquement lié au modèle de données

- Si nous avons défini une liste des patients à suivre pour un médecin donné, l'Interface aurait sans doute due être attaché au composant qui gère le médecin.
- Mais dans notre modèle le médecin peut consulter et modifier tous les DMPs
    - Rattachés au nœud auquel il (le médecin) est lui-même rattaché
    - Rattachés aux nœuds fils (ou descendant) de ce nœud auquel il est rattaché

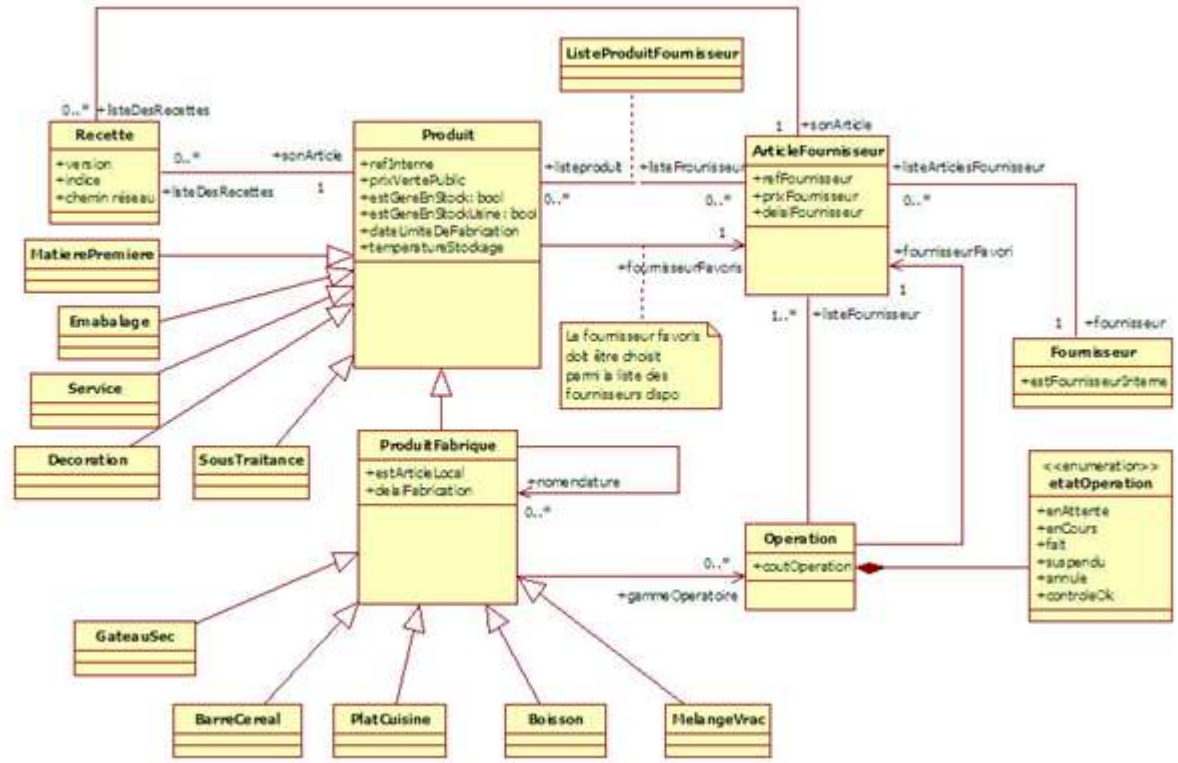
La solution la plus pertinente va alors être de rattacher une interface « quels DMPs sont accessibles à mon niveau » à un composant gérant un objet de type « Nœud de l'arborescence de l'hôpital ».
La réponse est définie par

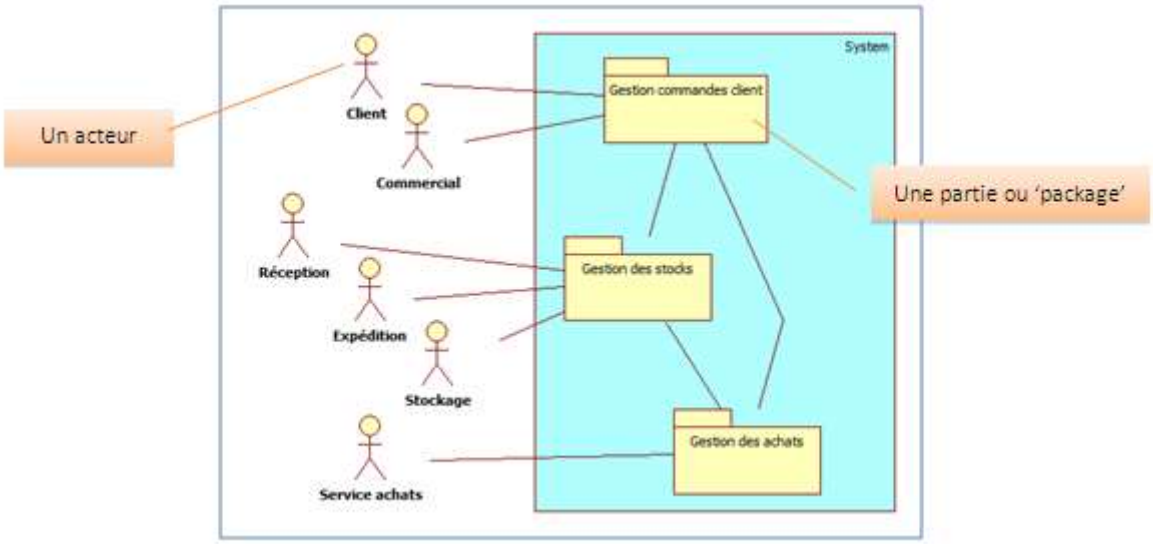
- La liste des DMPs qui me sont directement rattachés
- La liste des DMPs rattachés à un de mes fils

La méthode « Liste des DMPs rattachés à un nœud » doit être réentrante

Vous avez vos composants, les interfaces qu'ils proposent, et la liste des composants sur lesquels ils tirent, il reste encore a matérialiser de façon simple votre architecture
Là aussi, un peu de réutilisation ne peut nuire. Il existe un formalise désormais utilisé par p


**© 2003-2017 CARAPACE**




A la fin de cette étape, on a en général une bonne idée des attributs de chacun des objets.
Ils se composent

- Des attributs de base
- Des attributs calculés

Ainsi, pour un cercle

- Centre et Rayon sont des attributs de base
- Diamètre et aire sont des attributs calculés

Il reste à rajouter ces attributs dans le dictionnaire de données s'il existe.

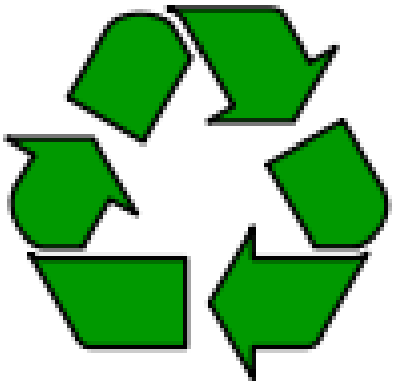

**© 2003-2017 CARAPACE**

**Critiquer le résultat de son Architecture, la raffiner**

Le travail de l'Architecte est un travail difficile et chronophage. Tout ce qui peut être évité est bon à prendre

Comme je le répète souvent, la première qualité de l'ingénieur c'est la réutilisation.

Pourquoi faire demain ce qu'un autre a faut pour vous hier ?



Dans un projet comme DELIRE, vous pouvez identifier 3 types de réutilisation

- La factorisation de composants
- La réutilisation d'une commande au sein du produit
- La capacité à rendre générique un composant pour le rendre réutilisable dans d'autres projets

<u>La factorisation de composants</u>

Cette capacité est souvent facilitée par le travail du Designer, qui identifie des tâches communes ou similaires pour des rôles différents. En ce cas, si on conçoit des composants s'inspirant fortement des rôles identifiés, on conçoit en parallèle le ou les composants communs qui seront appelés par les différents composants orientés rôles.

C'est le cas dans notre produit de la tâche « Mettre à jour mon profil », qui est disponible pour chaque type d'utilisateur, quels que soient son rôle et ses spécialités.

<u>La réutilisation d'une commande au sein du produit</u>

Ne pensez aps que ce soit une révolution : c'est une des base de l'Architecture Orientée Objet

Nous allons l'illustrer sur un exemple

Dès la première version de votre produit, il vous faudra offrir la capacité à définir l'arborescence de l'hôpital : c'est une nécessité pour la cohérence de votre projet.

Problème, cela signifie développer une interface utilisateur pour le seul data administrateur

Et il va falloir développer la capacité à créer

- Un nouvel hôpital au sein de l'AP-HP
- Un nouveau pôle au sein de l'hôpital
- Un nouveau service au sein du pôle…

Bref, beaucoup de travail pour un seul utilisateur

En général, les promotions précédentes ont élégamment contourné le problème en générant un script pour populer la base de données préalablement à la démonstration finale. Et cette structure était figée dans le cadre de la démo.

Hors, il n'y a pas de différence fondamental entre un hôpital, un pôle, un service, une unité hospitalière ou une unité de soin. Et on peut même pousser le bouchon en considérant que l'AP-HP est également similaire à tout cela


**© 2003-2017 CARAPACE**

Le nom est certes différent, et chaque objet a une caractéristique

- L'hôpital ne peut être rattaché qu'à l'AP-HP
- Un pôle ne peut être rattaché qu'à un hôpital
- Un service ne peut être rattaché qu'à un pôle
- Une unité hospitalière ne peut être rattachée qu'à un service
- Une unité de soin ne peut être rattachée qu'à une unité hospitalière

Posons qu'il existe un et un seul objet générique, « nœud d'arborescence AP-HP », qui a les caractéristiques suivantes :

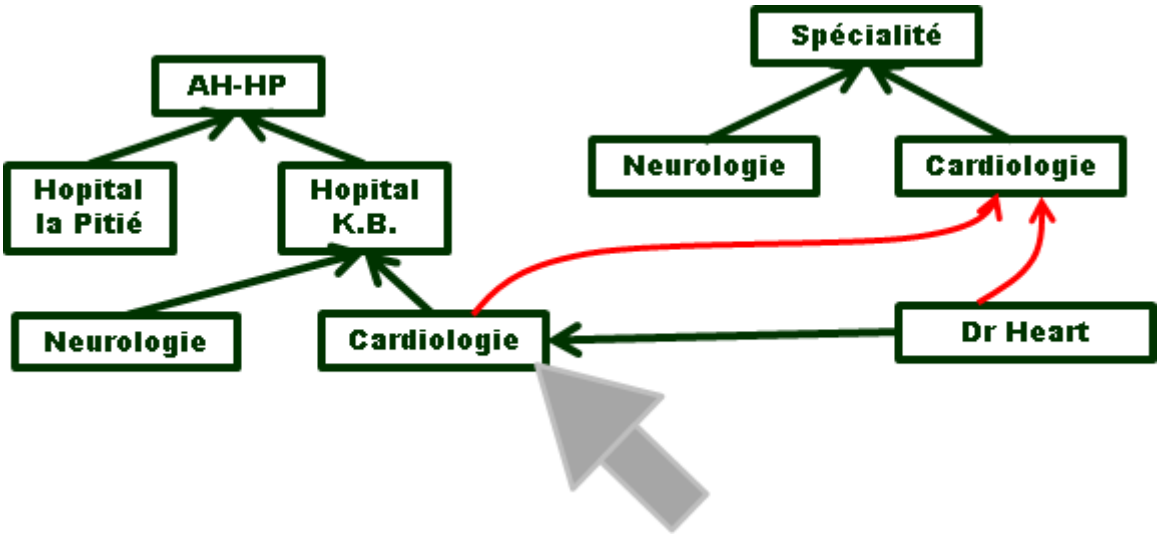
- Chaque instance pointe vers un et un seul membre de l'hôpital, considéré comme le responsable du nœud : ce membre doit lui-même nécessairement être rattaché au dit nœud
- Un objet a nécessairement un rang, qui peut prendre les valeurs entières de 0 à 5
- Un objet a un Display Name
    - Le Display Name d'un objet de rang 0 est AP-HP : il n'y a qu'un seul objet de rang 0 et il est créé à l'initialisation de la base de données
    - Le Display Name d'un objet de rang 1 est Hôpital
    - Le Display Name d'un objet de rang 2 est Pôle
    - Le Display Name d'un objet de rang 3 est Service
    - Le Display Name d'un objet de rang 4 est Unité Hospitalière
    - Le Display Name d'un objet de rang 5 est Unité de Soin
- Un objet a un compteur, valué à 0 à sa création
    - Il pointe vers une spécialité s'il est de rang 1 à 5
    - La spécialité doit être choisie lors de la cération d'un objet de ra,dg 1
    - La spécialité d'un objet de rang N compris entre 2 et 5 est égale à la spécialité du nœud de rang N-1 auquel il est rattaché
- Il a un identifiant
    - Soit une chaîne de caractère définie à sa création
    - Soit un nom automatique constitué par l'accolement du Display Name et de la spécialité, complété par un numéro qui est égal au compteur de l'objet sous lequel il a été créé et qui vient d'être incrémenté d'une unité.
        - Ainsi, sous le pôle Cardiologie, on pourra créer un premier service qui sera automagiquement nommé Service Cardiologie 1
        - Puis un second service qui sera automagiquement nommé Service Cardiologie 2
- Chaque objet de rang N est rattaché à un objet de rang N-1
- Sous chaque objet de rang N compris entre 0 et 4, on peut créer des objets de type N+1
- Un objet de type « nœud d'arborescence AP-HP » ne peut être détruit que si
    - Aucun objet de type « nœud d'arborescence AP-HP » ne lui est rattaché
    - Aucun membre de l'hôpital ne lui est rattaché
    - Son pointeur vers un responsable n'est pas ou n'est plus valué


**© 2003-2017 CARAPACE**

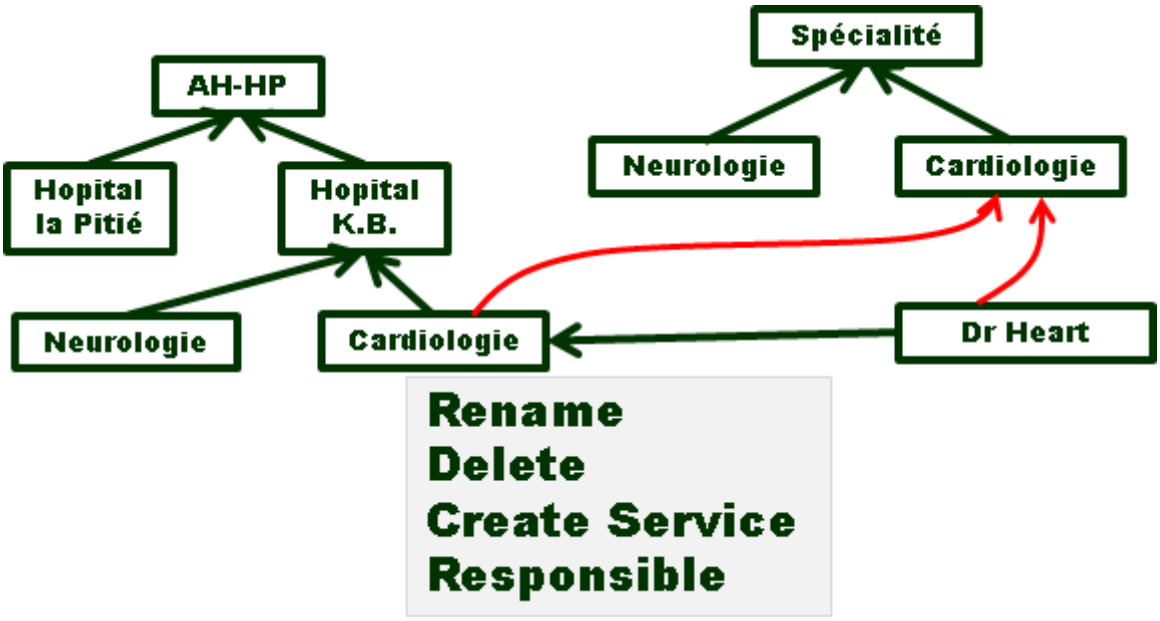

Dès lors, si j'ai le rôle de data Administrateur, et si je clique sur un « nœud d'arborescence AP-HP », alors apparait un menu contextuel

- Une commande **Rename**
- S'il respecte toutes les conditions pré fixées, une commande **Delete**
- Si c'est un objet de rand N 0 à 4, une commande « **Create Display Name (N+1)** »
- Si au moins un membre de l'hôpital est rattaché à ce nœud et si le pointeur Responsable n'est pas valué, une commande **Responsible**
- Si le pointeur Responsable est valué, une commande **Reinit Responsible**

Lorsque je clique sur le pôle Cardiologie



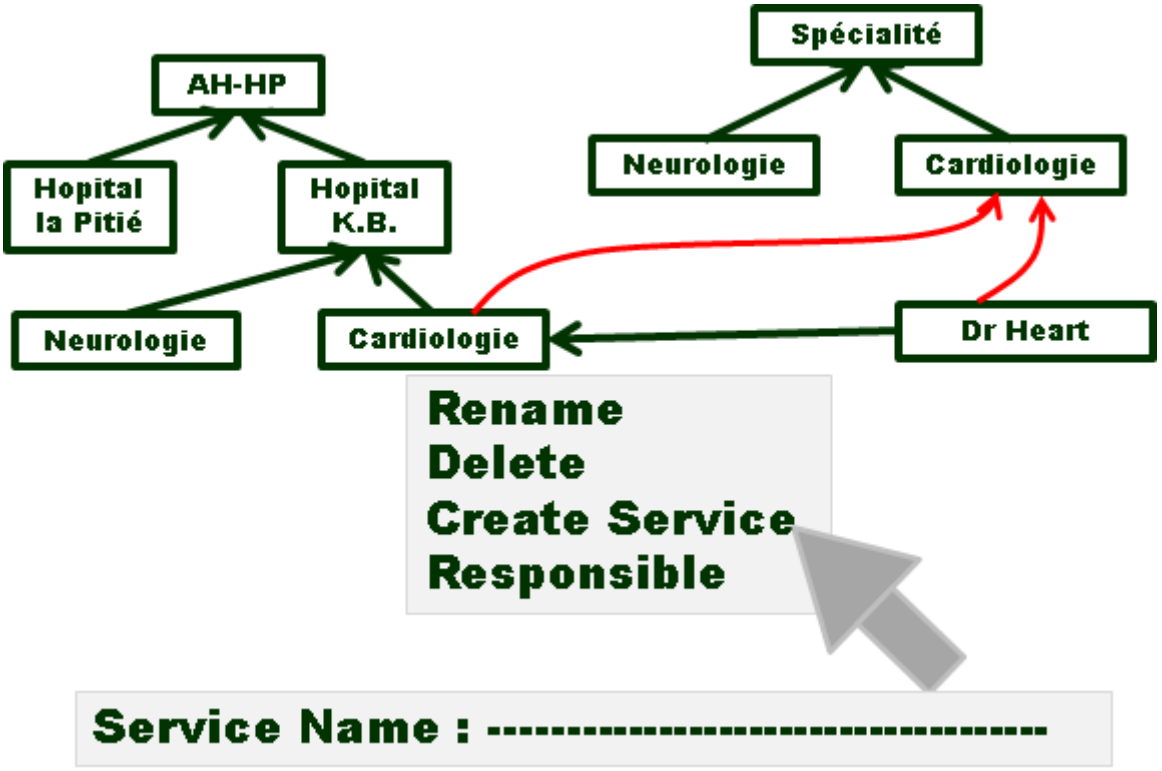

Voici ce qui apparait



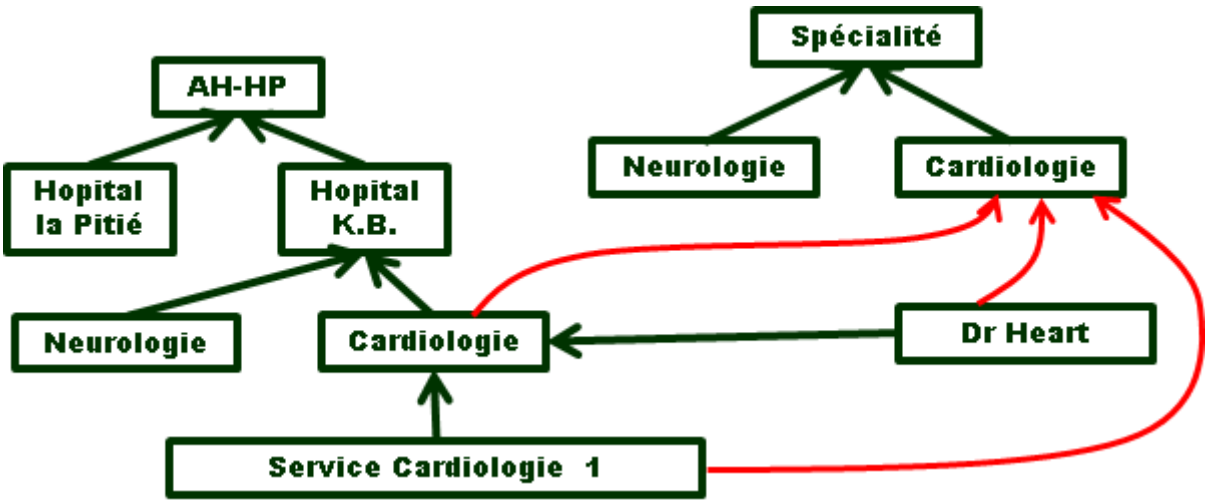


© 2003-2017 CARAPACE

Et si je clique sur le sous menu **Create Service**




Si je fais **Enter** sans avoir valué le champ Service Name, voici ce qui s'affichera





© 2003-2017 **CARAPACE**

La capacité à rendre générique un composant pour le rendre réutilisable dans d'autres projets

L'objectif de votre tout jeune entreprise est de développer des projets à façon pour répondre à des clients


- Mais alors que, dans projet « concret », on ne peut accumuler que du savoir-faire, dans un produit virtuel comme le logiciel on peut réutiliser des composants existants.
- Et c'est une bonne nouvelle : vous pourrez refacturer à un client B un composant développé dans le cadre d'un projet pour le client A.

Pour vous c'est une garantie de

- Gain de temps
- Ggain d'argent
- Gain de Qualité (pas besoin de faire converger le composant)

Une promo comme ça, ça ne se refuse pas.

La réalité est un peu plus complexe, sinon, cela ferait longtemps que tous les composants logiciels seraient génériques et réutilisables, le développement logiciel serait un jeu d'enfant consistant à assembler des composants sur étagère

- La première difficulté est qu'on ne découvre qu'une fois le composant terminé qu'il aurait pu être conçu de façon à être générique.
- La seconde difficulté est que le coût d'un composant générique est généralement évalué à deux fois le coût d'un composant spécifique, et dans un projet le temps nous est souvent cruellement compté.
- La troisième difficulté est le fait qu'un composant générique est plus complexe à concevoir qu'un composant spécifique ; il faudra lui passer en arguments les paramètres d'utilisation nécessaire pour l'utilisation en cours.

Une des méthodes les plus puissantes pour obtenir des composants génériques est d'utiliser un dictionnaire des données.

Cela permet d'avoir

- Une base de données générique
- Une gestion des objets en mémoire générique
- Des outils de browse, d'édition, de print, de check génériques

Cela permet également de faciliter l'évolutivité du projet

Cela permet enfin de figer le plus tard possible le modèle de données du projet

Bref que du bonheur. Et même le dictionnaire de données peut lui même être un composant générique.

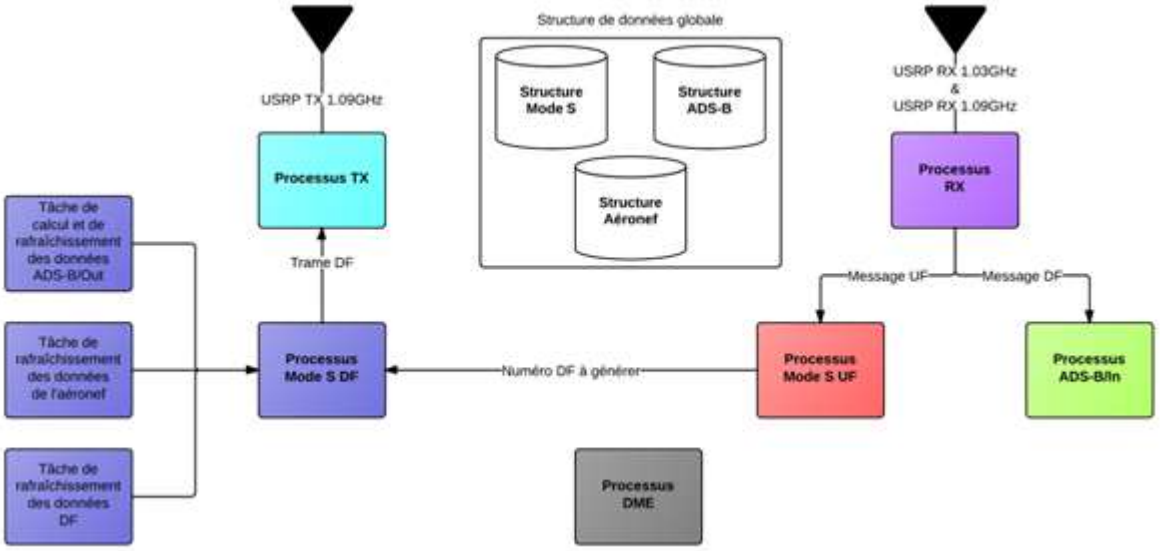
Je me doute néanmoins que vous ne déploierez pas ce type de solution dans le projet DELIRE

Mais de façon plus pragmatique, le fait de concevoir des dialogues et des panels génériques, et de stocker les messages et les libellés dans des fichiers par langue vous permettra de pouvoir offrir un produit en français, en anglais ou en espagnol dès la première version.


**© 2003-2017 CARAPACE**

**Communiquer et promouvoir son architecture logique**
A force d'itération, de raffinement et de modification, vous avez enfin votre architecture.

- Dans la réalité on ne sait jamais si on va mettre une semaine ou un an pour avoir une bonne architecture.
- On sait simplement qu'on a convergé quand une analyse de l'architecture obtenue vous satisfait et que vous ne souhaitez plus y apporter de retouche.




Votre travail d'architecte est-il fini ? Pas tout à fait : il va désormais falloir en faire la promotion.

L'Architecture Logique est un outil précieux utilisé par de nombreux acteurs du projet

Le Développeur, qui comprendra aisément la finalité de son composant, la justification des choix faits par l'Architecte. Il peut ensuite définir les Spécifications Techniques Détaillées de son composant en phase de Structuration, et le développer ensuite en phase de Réalisation

Le Testeur qui, en fonction de l'incident identifié en phase de Convergence, saura quelles traces mettre en place ou quelle jauge surveiller pour identifier le composant défectueux

Le Chef de Projet qui saura quel profil affecté au développement de chaque composant, et qui aura une meilleure visibilité de la difficulté et du coût de chacun de ces composants

La Maintenance, qui aura à cœur, lors d'une correction nécessaire suit à un incident troué en clientèle, de trouver une solution corrective qui respecte l'intégrité conceptuelle de l'architecture

Le Designer qui pourra identifier des points d'amélioration de son design présent qui permettent de converger vers une solution plus simple, plus robuste et plus facile à faire adopter par les futurs utilisateurs, de la même façon que le travail du designer


**© 2003-2017 CARAPACE**

donne un cadre à l'Architecte pour identifier des composants génériques. L'architecture permettra également de fixer un cadre pour les évolutions fonctionnelles futures du produit. Dans votre cas, les versions de 2019 et de 2020.

Le Client, qui percevra mieux les exigences exprimés par le fournisseur en termes de hardware, de software prérequisites, et de mise en place de mécanisme de sécurité, et qui comprendra la justification des temps de réponses constatés

L'Utilisateur qui comprendra mieux la logique intrinsèque du produit, ses possibilités, et le pourquoi de l'organisation de certaines commandes.

Le Marketing qui pourra valoriser la robustesse de la solution, ses points clé différentiateurs et qui pourra, lors de discussion futures avec le Client, poser un cadre pour de futures évolutions du produit,

Bien entendu, on ne communique pas la même granularité de conception au Développeur et à l'Utilisateur.
L'architecture représente souvent un des fondements du savoir-faire de l'entreprise : sa connaissance par les architectes de la concurrence peut leur permettre d'anticiper les axes de stratégies de votre entreprise, mais également es faiblesses potentielles de votre solution, qui permettront à leur Marketing de vous enfoncer

Le fait d'avoir communiqué et promu son architecture Logique va permettre à l'Architecte :

- De justifier le coût de développement de la version en cours et des suivantes
- De favoriser le travail du Responsable Planning et Budget pour identifier dans quel ordre développer les composants de la version courante: cela permettra, en cas de dérapage du planning, de pouvoir couper dans les fonctionnalités délivrées sans avoir besoin de mettre à la poubelle un tas de développement commencés de façon prématurée.
- De justifier pourquoi certaines propositions d'évolutions ne sont pas les bienvenues

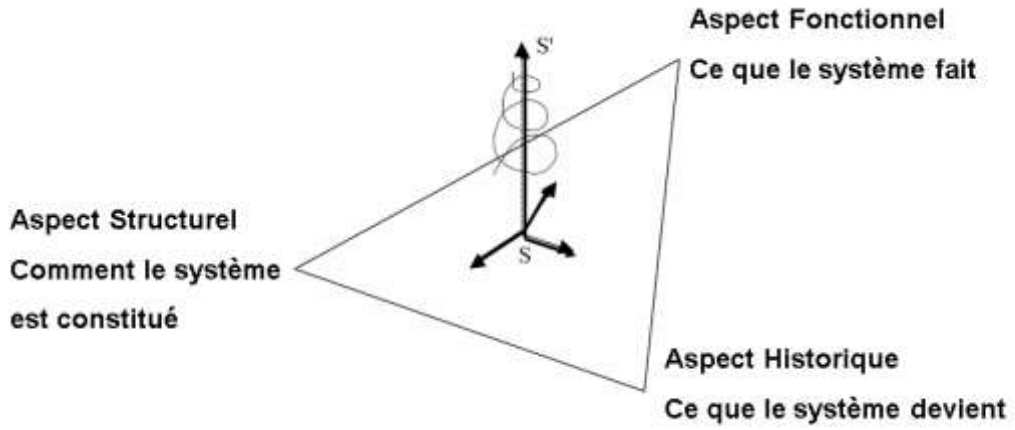
Un produit est amené à se détériorer au cours de son cycle de vie, du fait des opérations de maintenance et des évolutions qui fragilisent son architecture et en dégradent l'intégrité conceptuelle.
A termes, il faudra probablement renoncer à faire évoluer le produit, c'est-à-dire qu'il faudra recommencer une nouvelle architecture complète de la solution, prenant en compte les nouvelles technologies ayant émergé depuis la naissance de la version actuelle du produit, les évolutions antérieures du produit au cours de cette version, et les évolutions futures anticipées. Réachitecturer un produit est une opération coûteuse, et qui peut mette en péril la survie de l'entreprise.

Le fait d'avoir bien architecturé dès le départ votre solution et d'en défendre l'intégrité conceptuelle au cours de son évolution vous permettra de faire durer le plus longtemps possible la solution initiale, les prises en compte de nouvelles technologies, les corrections et les évolutions trouvant naturellement leur place dans des composants dont le rôle est clairement identifié, et dont la conception tant des


**© 2003-2017 CARAPACE**

données que des fonctions est encapsulé, permettant de minimiser les impacts de chaque modification.

Mais cela suppose que l'Architecte ait non seulement défini l'architecture de la version en cours, mais qu'il ait une bonne visibilité des versions futures




En d'autres termes l'Architecte doit avoir une bonne visibilité

- De ce que fait le produit : l'aspect fonctionnel, le travail du Designer pour répondre aux exigences du Client
- De la façon optimale dont il doit être conçu pour répondre aux besoins du client tout en répondant aux contraintes du système : l'aspect structurel, l'Architecture Logique, son propre travail
- De la façon dont il devra évoluer, l'aspect historique.

Voilà pourquoi le travail de l'Architecte est si difficile, si chronophage et si important pour la réussite du projet.
Trouver un bon architecte est difficile : c'est un métier qui demande des qualités de rigueur, de bon sens, d'empathie et de communication, et dont la compétence se développe par l'expérience plus que par l'apprentissage de techniques.
Dans les grands produits logiciel, l'Architecte est souvent, sinon le plus haut, du moins un des plus hauts salaires de l'entreprise.


**© 2003-2017 CARAPACE**

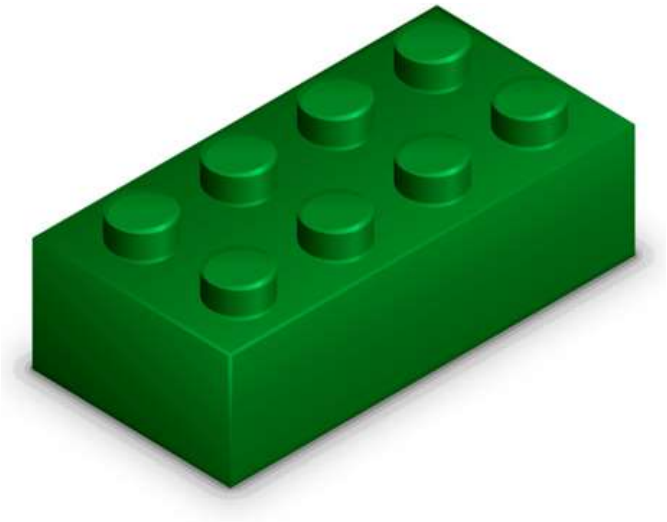
**L'importance de l'Architecture Logique dans le projet**

Si votre Architecture Logique est de qualité, c'est-à-dire quu'elle est

- Aboutie : elle couvre mes besoins du produit actues et à venir
- Complète : elle définit, pour chaque composant
  - Sa finalité
  - Les interfaces à publier
  - Les composants sur lesquels il peut tirer
  - Le type de savoir-faire nécessaire pour développer le composant
- Cohérente : le développeur n'identifiera pas de manque pour définir ses Specification Techniques Détaillées,
- Robuste : chaque composant est fortement cohérent et faiblement couplé

Alors les chances de votre projet de pouvoir converger ont fortement augmenté

- Les spécifications Techniques Détaillées devraient pouvoir être réalisées sans problème majeure en phase de Structuration
- La réalisation du composant, sa capacité à pouvoir être testé de façon unitaire (c'est-à-dire sans la présence des autres composants qui seront simplement simulés) est favorisé par la disponibilité des STD d'une part, par la forte cohérence et le faible couplage du composants d'autre part, en phase de Réalisation

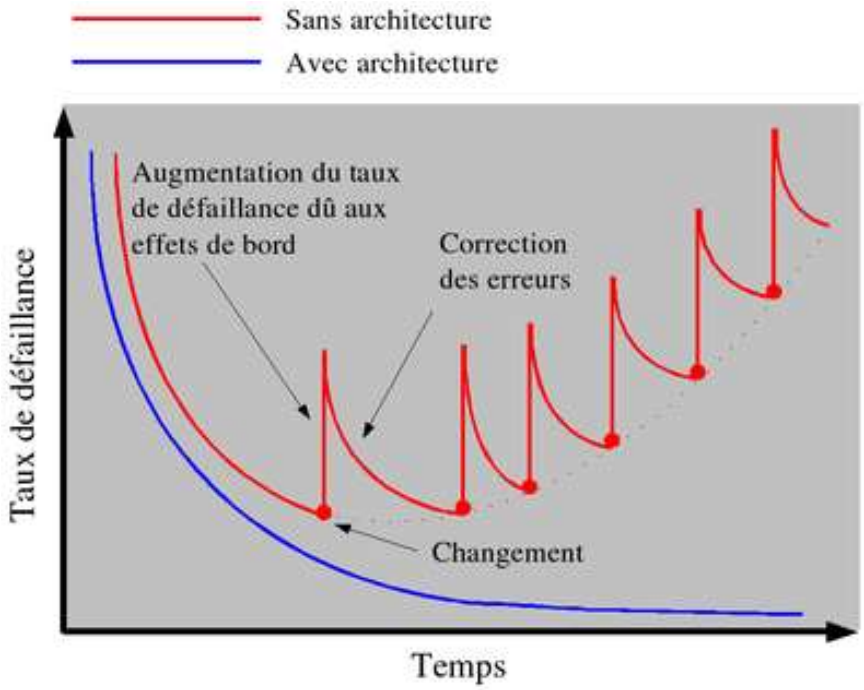


- Et lors de la phase de Convergence, une fois le tas de composants disponibles , l'intégration de ceux-ci pour obtenir le produit visé devrait ne pas révéler de surprise majeure, les interfaces ayant été bien spécifiées et le schéma global de l'architecture ayant permis d'éviter les problème de cyclicité ;


**© 2003-2017 CARAPACE**

Lors de tests d'intégration, des tests fonctionnels et des stress tests, l'intégrité conceptuelle de votre architecture devrait permettre d'avoir des correctifs localisé au sein d'un seul composant.

Certes L'Architecture Logique ne suffit pas pour garantir le succès du projet
Par contre l'absence d'Architecture logique de qualité suffit pour garantir l'échec du projet.





© 2003-2017 **CARAPACE**