La conception

La conception vise à définir une architecture modulaire du logiciel qui facilitera la maintenance et permettra le développement parallèle par plusieurs personnes.
But : enrichir la description du logiciel de détails d'implémentation afin d'aboutir à une description très proche d'un programme (décrire le *comment*).
On distingue deux phases de conception dans un cycle de développement :

- Conception préliminaire (architecturale) : ou conception globale a pour but de décomposer le logiciel en composants plus simples, définis par leurs interfaces et leurs fonctions (les services qu'ils rendent). Elle comporte deux aspects, l'un fonctionnel et l'autre organisationnel. L'aspect fonctionnel concerne la description des services que le système devra fournir (quoi). Quant à l'aspect organisationnel, l'intérêt est porté sur comment ces services vont être réalisés.

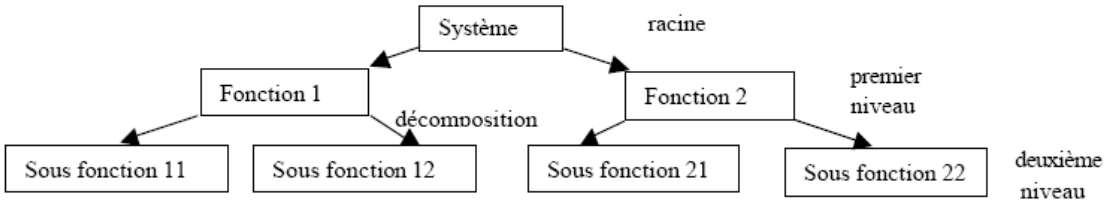
- Conception détaillée: IL s'agit d'affiner (détailler) les aspects fonctionnel et organisationnel du système. où la définition des modules et leurs structurations constitue la principale activité. La conception détaillée fournit pour chaque composant une description de la manière dont les fonctions ou les services sont réalisés : algorithmes, représentation des données.

I- La modularité :

Un module est un composant d'une application, contenant des définitions de données et/ou de types de données et/ou de fonctions et constituant un tout cohérent et homogène (un logiciel ou un assemblage de modules).
On peut distinguer deux démarches de conception :

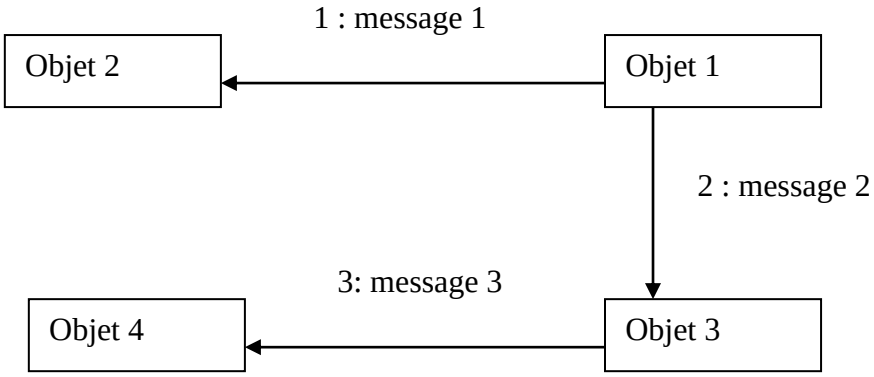
- L'approche fonctionnelle,
- l'approche à objets.

Dans l'approche fonctionnelle, un module peut être vu comme un bloc de code pouvant être appelé tel q'une procédure, une fonction ou encore comme un include de déclaration C/C++.
Toute partition de code entre deux éléments servant l'encadrement (Begin, end, {….},…) et pouvant être nommé.
Dans l'approche fonctionnelle, un module est un sous-système du système global. La relation de base est la relation de décomposition, qui constitue une hiérarchie.



Décomposition fonctionnelle

Dans l'approche par objet, un module correspond à un objet concret ou abstrait du monde de l'application. Les objets se collaborent entre eux pour réaliser le système global.
Par exemple, dans la bibliothèque on peut trouver les objets (modules) Livre, Bibliothécaire, Emprunt, Réservation, Catalogue, etc. Les objets regroupent données et traitements. Ce sont des entités autonomes qui collaborent pour réaliser le système global.



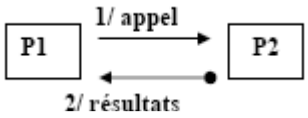
Collaboration d'objets

La modularité doit répondre aux règles suivantes pour obtenir un logiciel facilement maintenable :

1. Correspondent aux besoins.
2. Assurer une forte cohésion : Les éléments d'un module remplissent des tâches très complémentaires. Chaque élément est essentiel pour que ce module fonctionne correctement.
3. Assurer un couplage faible : les modules à couplage faible sont indépendant ou presque.

***Le couplage de données :**

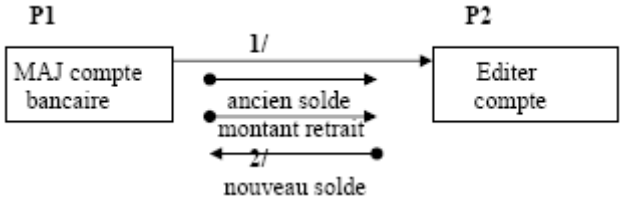
Quand on a un passage de données atomiques entre module appelant et module appelé.



***Le couplage de contrôle**

Quand un module transmet à un autre module une information destinée à contrôler sa logique interne.

Exemple:



L'avantage du système à forte cohésion et faible couplage est qu'il est possible de remplacer un module quelconque par un module équivalent avec peut ou pas de changement dans les autres modules du système.

1. Les critères de modularité :

a- La décomposition modulaire :

Permet de décomposer un problème complexe en problèmes plus petits et solubles isolements (conception descendante).

b- La composabilité modulaire :

Favorise la production indépendante du module pouvant être combiné dans des contextes différents de ceux dans lesquelles ils ont été initialement développés. On utilise les éléments existant pour faire un nouveau système (la réutilisation des composant logiciel).

c- La compréhensibilité modulaire :

Très utile à la maintenance, la compréhension d'un module n'exige pas celles des autres.

d- La continuité :

Est assurée si des petites modifications de spécification n'amènent pas à se voir l'ensemble de l'architecture. Les modifications ne doivent affecter que quelques modules (l'extensibilité).

e- La protection modulaire :

Est assuré si une condition anormale se produisant pendant l'exécution d'un module reste localisée dans ce module ou bien ne se propage qu'à dans quelques modules voisins.

2. Les principes de la modularité :

a- Unité modulaire linguistique :

Les modules doivent correspondre à des unités syntaxiques du langage de conception ou de spécification. Les modules doivent être compilables séparément (la décomposabilité, la composabilité, la protection).
Exemple : classes, fichiers d'implémentation,…

b- Minimisations des interfaces :

Tout module doit communiquer avec aussi que d'autres modules que possible (protection, continuité).

c- Simplification des interfaces :

Les modules qui communiquent doivent échanger aussi peu de données que possible.

d- Interfaces explicites :

Lorsque deux modules communiquent, ceci doit se voir clairement dans le texte de l'un et/ou de l'autre (compréhensibilité).

Contre exemple :

Utilisation de variables globales ou des paramètres optionnel pour communiquer entre modules.

e- Masquage d'information (encapsulation) :

Toute information d'un module doit être privée sauf l'interface : le contenu d'un module peut changer sans que ces appels soient inchangés (continuité).

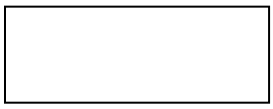
II- La conception structurée (approche fonctionnelle):

Il s'agit de structurer l'application aux modules fonctionnels, généralement sous une hiérarchie de fonctions.
Le système peut d'abord être décomposé en modules de haut niveau ou sous systèmes. Puis chaque sous système est à son tour décomposé indépendamment et ceux à plusieurs niveau de raffinement, par exemple : jusqu'à ce que les modules obtenus soient réalisables par une seule personne (approche descendante).
On construit tout d'abord des modules de base en essayant de réaliser des modules assemblés des modules de base (l'approche ascendante).

III- Les diagrammes de structure :

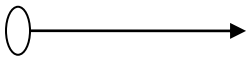
Ils reflètent une organisation hiérarchique et fonctionnelle des systèmes.

1- Notation :



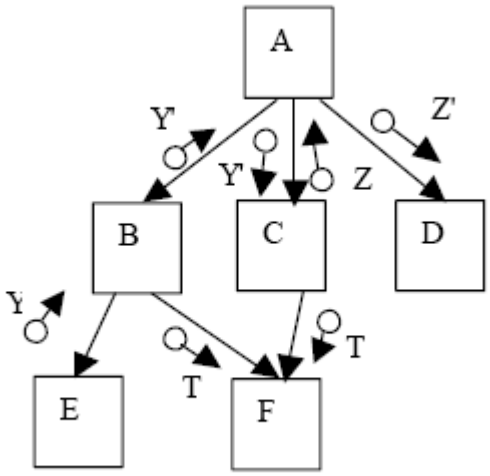
: Module

: Relation d'appel entre modules



: Flux de données (paramètres, données retournées).

2- Exemple:



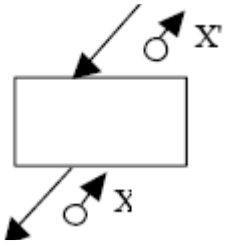
Les modules A appelle les modules B, C et D.
Les modules B appelle les modules E, F.
Le module C appelle le module F.
La donnée Y trouve son origine dans le module E, et transforme en Y' par le module B et est transmise au module C. Le module B transmis également la donnée T au module F. Le module A transmit Y' à C et C transmit T à F. C retourne Z à A qui transmit Z à D.

Remarque :

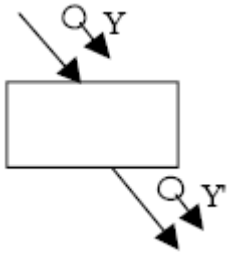
- Un module de niveau i peut être utilisé par plusieurs modules de niveau i-1, mais un module de niveau i ne peut utiliser un module de même niveau.
- Le fait que les modules B, C et D apparaissent de gauche à droite n'implique pas qu'il soit appelé dans cet ordre.

3- Classification des modules:

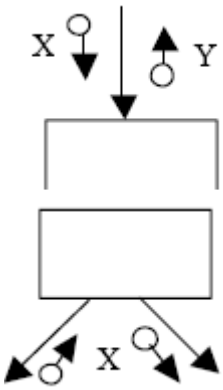
Les modules peuvent être classées en types entrée, sortie, transformation et coordination.



Entrée: le module est chargé d'accepter les données d'un module situé à un niveau inférieur de la hiérarchie et de transmettre ces données à une unité de niveau plus élevé sous une forme module.



Sortie: Le module est chargé d'accepter des données provenant d'un module situé à un niveau plus élevé et de les transmettre à un module de niveau inférieur.



Transformation: Le module est chargé d'accepter des données provenant d'un module de plus haut niveau, transforme ces données et les retournes à ce même module.

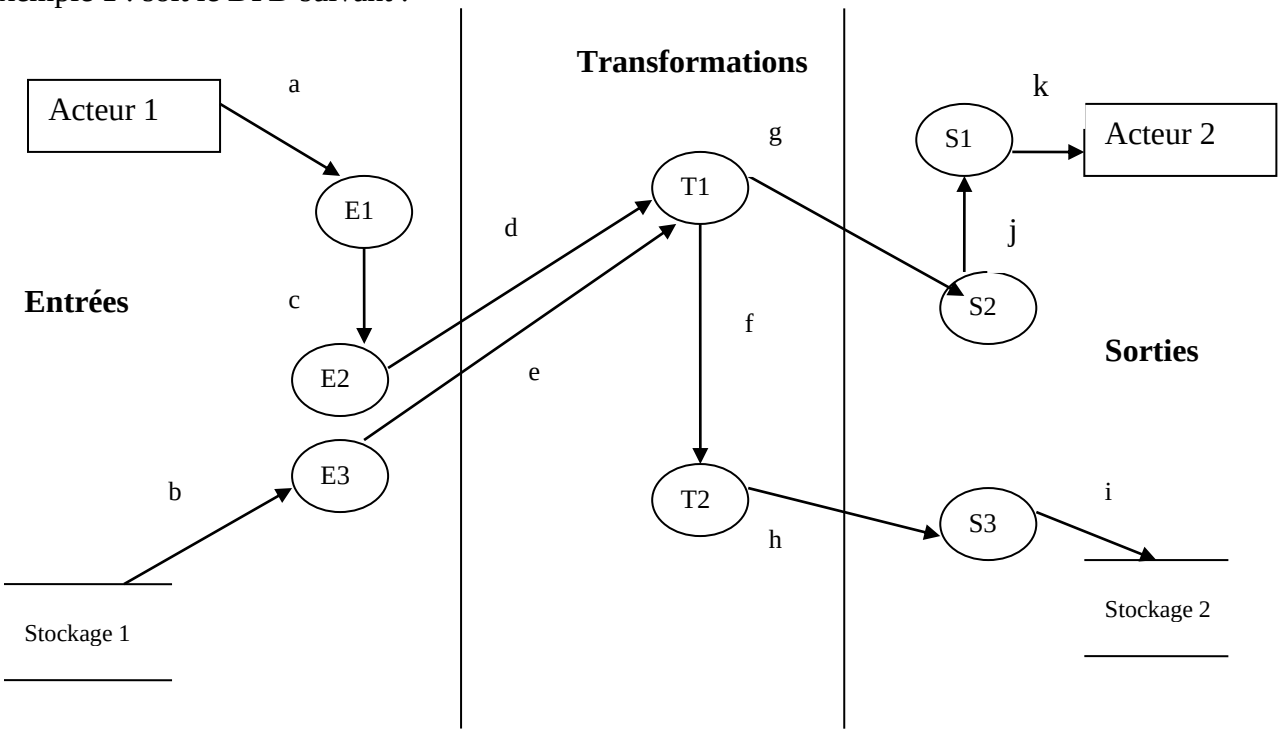
Coordination: Un tel module est responsable de la gestion des autres modules.

4- Elément de démarche :

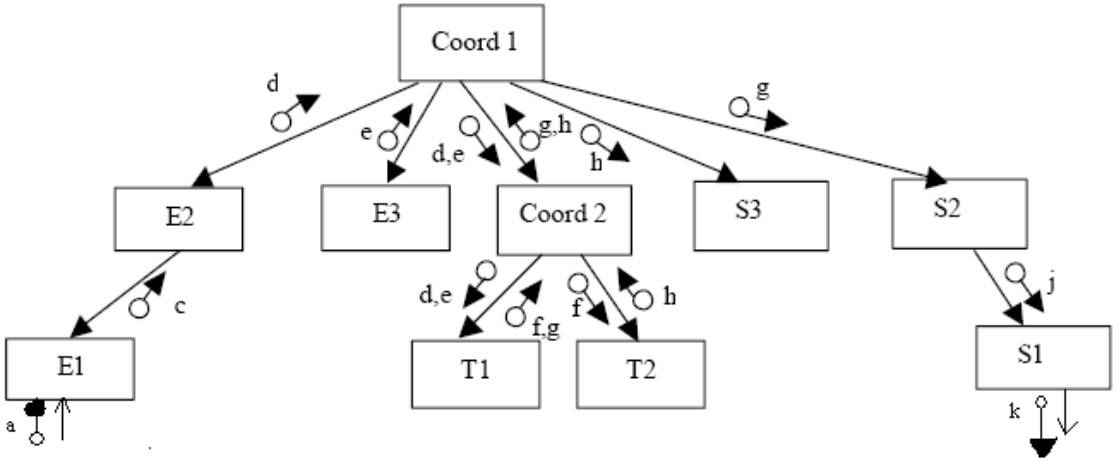
A partir d'un diagramme de flot de données :

- Classer les fonctions en entrée, transformation et sortie.
- Introduire des fonctions de coordination pour dispatcher (distribuer) les flots, vers les relevant d'un même sous système.

Exemple 1 : soit le DFD suivant :



Donne :



Exemple:

Soit un vérificateur d'orthographe qui cherche chaque mot d'un document dans un dictionnaire. Les mots trouvés sont considérés comme corrects. Les mots non trouvés apparaissent à l'écran et l'utilisateur prend une décision : soit ils sont corrects et sont ajoutés au dictionnaire, soit ils sont incorrects et ajoutés avec la correction à un fichier des mots à corriger.

Construction du DFD:

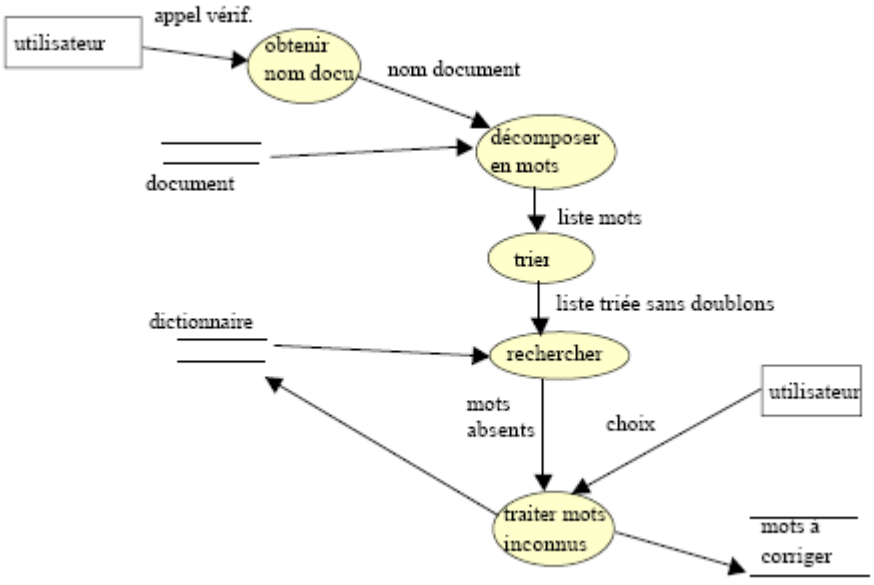
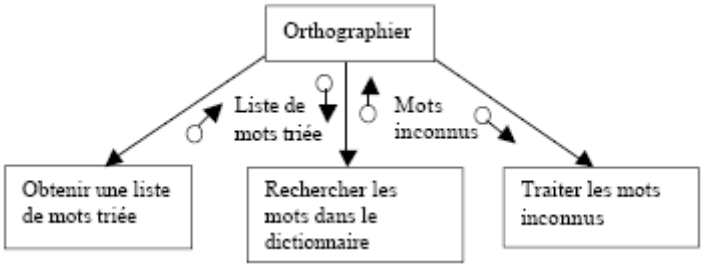
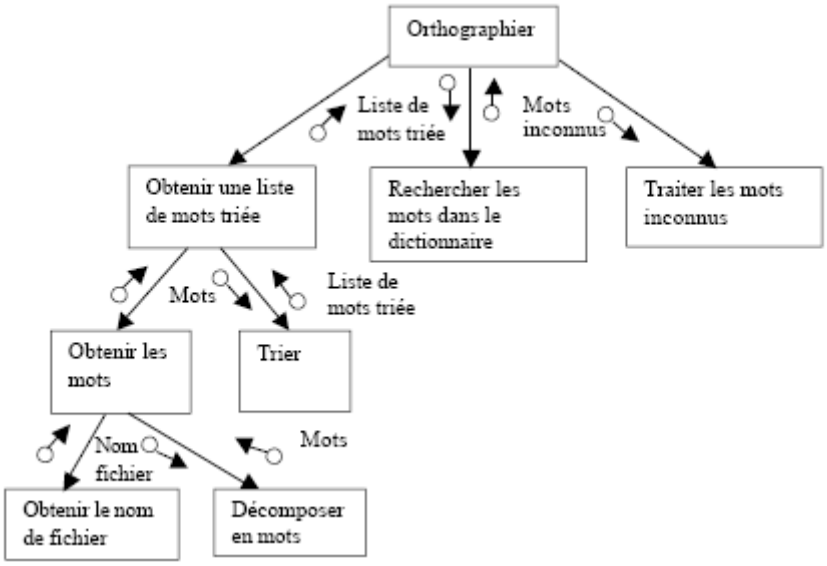


Diagramme de structure:

La transformation centrale est 'rechercher dans le dictionnaire'. C'est là ou les données deviennent des résultats. D'où la première décomposition :



L'entrée 'obtenir une liste de mots triée' peut se décomposer à deux niveaux pour obtenir :



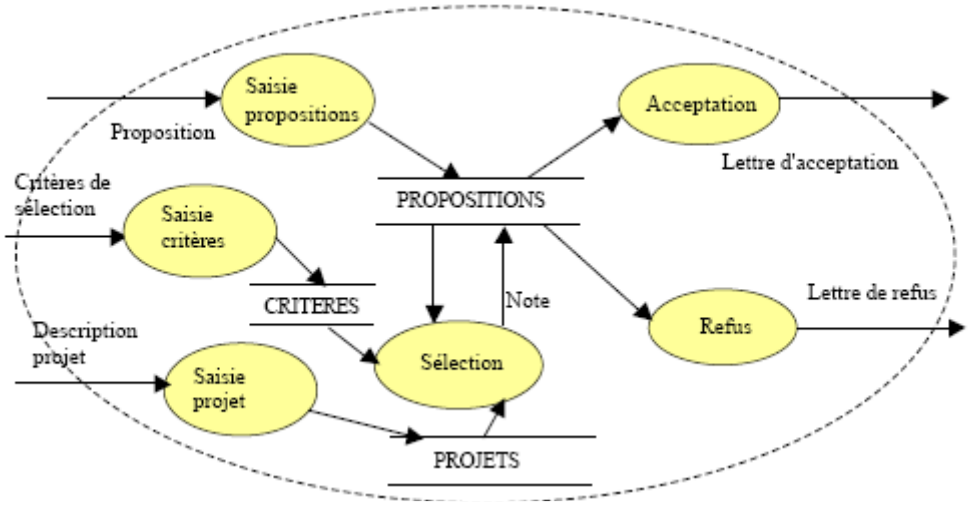
Une fois l'architecture globale est décrite, les données des différents modules doivent être précisées afin que les différents programmeurs qui coderont les modules soient bien d'accord sur le type et l'ordre sur le type et l'ordre des paramètres à échanger.

Exemple:

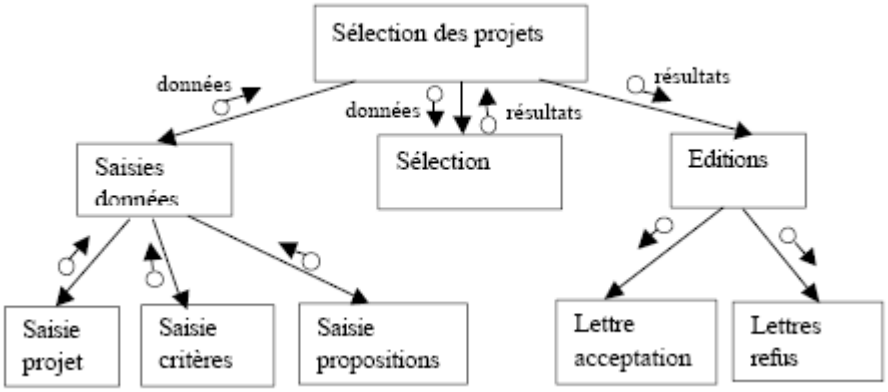
```
Procédure      orthographier
MotsTries, MotsInconnus, ListeDeMots
Procédure      ObtenirMotsTries(
        MotsTries: OUT listeDeMots);
Procedure RechercheDansDictionnaire(
        MotsTries: IN ListeDeMots
        MotsInconnus: OUT ListeDeMots);
Procédure TraiterMotsInconnus(
        MotsInconnus: IN ListeDeMots);
Begin
ObtenirMotsTries(MotsTries);
RechercheDansDictionnaire(MotsTries, MotsInconnus);
TraiterMotsInconnus(MotsInconnus);
END
END orthographier.
```

Exercice:

A partir du DfD construit au chapitre précédent pour la sélection d'une proposition suite à un appel d'offres, proposer un diagramme de structure.


Saisie propositions
Proposition
Acceptation
Lettre d'acceptation
Critères de sélection
Saisie critères
PROPOSITIONS
CRITERES
Note
Refus
Lettre de refus
Description projet
Saisie projet
Sélection
PROJETS

Solution:


Sélection des projets
données
données
résultats
résultats
Saisies données
Sélection
Editions
Saisie projet
Saisie critères
Saisie propositions
Lettre acceptation
Lettres refus