

Méthodes de développement

Conception de l'architecture

1 - Rôle de la conception du logiciel	2
2 - L'architecture logiciel et sa description.....	2
2.1 L'architecture logiciel.....	2
2.2 Décomposition d'un logiciel.....	2
2.3 Dépendances et relations entre composants	3
2.4 Découpage en sous-systèmes	3
2.5 Interactions ou comportement dynamique d'un programme	3
3 - Démarche de conception.....	4
3.1 Une bonne solution d'architecture	4
3.2 La conception de l'architecture.....	4
3.3 Conception de l'architecture dans les méthodes itératives	5
3.4 Conception de l'architecture dans les méthodes agiles.....	5
4 - Méthodes de conception	6
4.1 Les grandes catégories de méthodes	6
4.2 Méthodes structurées.....	6
4.3 Méthodes orientées objet.....	7
4.4 Le problème de la réutilisation.....	9
4.5 Méthode du point milieu et méthodes mixtes	10
4.6 Choix d'une méthode.....	10
5 - La démarche de conception de l'architecture orientée objet	10
6 - Le modèle logique.....	11
6.1 Les principaux objets	11
6.2 Exemple d'identification des principaux objets.....	12
6.3 Examen des réutilisations.....	12
6.4 Interactions entre objets	13
6.5 Les relations entre classes et l'optimisation des classes	14
7 - Modèle des processus et modèle de déploiement	15
7.1 Programmes et processus	15
7.2 Découpage en programmes, processus et threads	16
7.3 Répartition en programmes et processus dans l'exemple	16
8 - Interfaces système.....	17
8.1 Données persistantes	17
8.2 Interfaces opérateur	18
9 - Décomposition en sous-systèmes	18
9.1 Critères de décomposition.....	18
9.2 Définition des sous-systèmes et de leurs interfaces	19
10 - Définition de la logique d'intégration	20
11 - Vérification de la couverture des besoins	20
11.1 Vérification de couverture fonctionnelle.....	20
11.2 Validation des performances et aspects critiques.....	21
12 - Fin de la conception de l'architecture.....	21
13 - Le document d'architecture logiciel (méthodes non agiles).....	21
13.1 Objectifs et plan type.....	21
13.2 Rédaction du document.....	22
14 - Le document de conception préliminaire (méthodes agiles)	23
14.1 Objectifs et plan type.....	23
14.2 Rédaction du document.....	24

1 - Rôle de la conception du logiciel

Le rôle de la conception du logiciel est de trouver une bonne solution d'architecture qui permette de remplir tous les besoins recensés dans la spécification technique de besoin, de décrire cette solution, puis de la détailler suffisamment pour permettre d'effectuer le codage.

Le résultat de la conception sera un découpage du logiciel en pièces élémentaires et en regroupements de pièces élémentaires pour le codage, la définition du rôle de ces pièces et de leurs interfaces, et la mise en évidence des relations et des interactions entre ces pièces.

On identifie aussi les pièces de logiciels existants qui peuvent être réutilisées.

La conception est conduite en deux étapes :

- la conception de l'architecture où on définit l'organisation du logiciel et on découpe en groupes de pièces (appelés sous-systèmes),
- la conception détaillée où dans chaque sous-système le logiciel à écrire est découpé en pièces élémentaires.

Nous allons d'abord préciser ce qu'est l'architecture du logiciel et comment la décrire. Nous aborderons ensuite la démarche de conception et ce qu'est une bonne solution, présenterons les méthodes de conception, et détaillerons la conception orientée objet.

La conception détaillée sera présentée dans la fabrication du logiciel.

2 - L'architecture logiciel et sa description

2.1 L'architecture logiciel

Les travaux d'architecture dans le bâtiment consistent dans le calcul des dimensionnements, les choix de matériaux, la réalisation des plans d'ensemble...

Par analogie, l'architecture du logiciel consistera à dresser les plans d'ensemble, c'est à dire définir les pièces élémentaires, leur organisation et leurs relations ou interactions.

L'architecture du logiciel recouvre en général 4 aspects : la décomposition du logiciel, les dépendances et relations, le découpage en sous-systèmes, les interactions (ou comportement dynamique).

2.2 Décomposition d'un logiciel

Un logiciel peut être composé d'un ou plusieurs programmes et de données.

Un programme regroupe des pièces élémentaires appelées "composants", la nature de ces composants dépend du langage de programmation : fonctions pour le langage C ou classes dans les langages objet.

Dans la conception de l'architecture on ne s'intéresse qu'à un nombre limité de composants jugés les plus importants (principaux composants), les autres seront identifiés dans la conception détaillée

2.3 Dépendances et relations entre composants

Le second aspect de l'architecture d'un logiciel est de montrer les relations entre les principaux composants.

Les types de relation dépendent des types de conception et des types de programmation, on pourra trouver :

- des relations de dépendance (ou association unidirectionnelle), lorsqu'un composant fait appel aux services d'un autre composant,
- des relations d'héritage (ou généralisation) en programmation objet,
- des relations d'agrégation...

2.4 Découpage en sous-systèmes

Les activités de codage et essais ne peuvent être réalisées efficacement que par de toutes petites équipes (1 ou 2 personnes). Il faut donc découper l'équipe de projet en petites équipes de réalisation (ou sous-groupes) et découper le logiciel en morceaux (les sous-systèmes), et confier la responsabilité de réalisation de chaque morceau (conception détaillée puis codage et essais) à une de ces petites équipes.

Un sous-système sera un ensemble de composants (classes ou fonctions).

Pour que les équipes soient productives, il faut de plus qu'elles puissent travailler de façon quasi indépendantes les unes des autres. Il faudra donc faire en sorte lors du découpage en sous-systèmes que ces sous-systèmes aient des interfaces limitées entre eux et que ces interfaces puissent être définies.

2.5 Interactions ou comportement dynamique d'un programme

Le dernier aspect de l'architecture est de montrer le comportement des programmes et composants lors de l'exécution.

Lors de l'exécution d'un programme, on cherchera à montrer la séquence des appels entre composants. Cependant cette séquence est extrêmement variable en fonction des conditions d'exécution, des données entrées... et il n'est pas possible de représenter toutes les séquences. On se limitera à quelques scénarios caractéristiques (définis par un état initial et une séquence d'événements) et on montrera pour chaque scénario les séquences des appels.

Si on travaille en UML, ces scénarios pourront être des cas d'utilisation, et la représentation pourra être faite par des diagrammes UML (diagrammes de collaboration d'objets ou diagrammes de séquence de messages).

Un deuxième aspect du comportement dynamique se situe au niveau des interactions entre les processus et les threads créés lors de l'exécution, lorsqu'il y a plusieurs processus ou threads l'exécution.

On pourra utiliser pour représenter ces interactions des diagrammes de type diagramme de séquence UML.

3 - Démarche de conception

3.1 Une bonne solution d'architecture

On peut trouver une infinité de solutions pour répondre au besoin exprimé, dans la mesure évidemment ou on s'est limité à exprimer des besoins sans définir des solutions.

Il semble difficile de trouver des critères pour définir la meilleure solution, et donc on ne prétendra pas trouver la meilleure solution, mais seulement trouver une solution qui soit jugée "bonne".

Une bonne solution est une solution qui :

- satisfasse le client, aussi bien pour les besoins exprimés que pour les besoins implicites ou non exprimés (facilité d'emploi, ergonomie, temps de réponse, évolutivité...),
- minimise les risques et le travail du réalisateur...

On pourra en déduire les critères suivants pour qu'une solution soit jugée bonne :

- la solution doit satisfaire complètement le document d'exigences, c'est à dire remplir toutes les fonctions demandées, respecter les performances spécifiées et respecter les contraintes,
- le logiciel résultant doit être facile à utiliser, présenter une bonne ergonomie, répondre rapidement aux requêtes utilisateur...
- le logiciel résultant doit être facile à modifier, et continuer à fonctionner après mise en place de matériels informatiques plus modernes ou de nouvelles versions de logiciel de base,
- la solution doit réutiliser le maximum de composants existants,
- la solution doit minimiser les interfaces entre composants, et les interactions à l'exécution du logiciel,
- le nombre de composants à coder doit rester à l'échelle humaine,
- la réalisation des sous-systèmes doit pouvoir être répartie équitablement entre les différents membres de l'équipe en tenant compte de leur compétence,
- la réalisation des sous-systèmes doit permettre d'obtenir les composants codés et testés à temps pour faire une intégration progressive dans un ordre logique.

3.2 La conception de l'architecture

Le document d'entrée est le document d'exigences (spécification technique de besoin ou spécification générale de besoins) qui définit les fonctions et performances demandées, et les contraintes relatives à la réalisation.

La conception de l'architecture est une démarche itérative :

- à partir du problème physique posé par le document d'exigences, on identifie les principaux composants (classes ou fonctions),
- on identifie les principaux composants ajoutés par les nouvelles fonctions induites par la solution logicielle (communications, interfaces opérateurs...) et on recherche tous les logiciels qui peuvent être réutilisés,
- on examine les interactions entre tous ces composants et on complète ou aménage les principaux composants et les fonctionnalités,
- on examine la répartition entre processeurs et les parallélismes d'exécution,
- on examine les interfaces du système, et en particulier les données dites persistantes (fichiers ou bases de données),
- on répartit les principaux composants et les fonctionnalités entre des sous-systèmes pouvant faire l'objet d'une réalisation en parallèle,

- on définit la logique d'intégration des sous-systèmes,
- on vérifie que la solution esquissée permet de couvrir toutes les fonctions et performances demandées dans le document d'exigences.

On itère jusqu'à converger vers une bonne solution.

Cependant ce type de démarche ne s'applique bien que si une partie importante du logiciel doit faire l'objet d'un développement. On trouve maintenant de nombreux logiciels qui peuvent être réalisés en grande partie en utilisant des logiciels existants, ce qui conduit à modifier le début de la démarche :

- d'abord identifier les logiciels à réutiliser, les fonctions qu'ils peuvent remplir et leurs interfaces,
- identifier ensuite les fonctions ou tables de données à réaliser pour enchaîner ces logiciels, traiter les interfaces et compléter les fonctionnalités, et définir les principaux composants à réaliser.

On pourra ensuite examiner les interactions, les programmes et processus, la répartition en sous-systèmes...

En sortie de la conception de l'architecture on disposera :

- du document d'architecture logicielle, qui décrit l'architecture du logiciel,
- du plan d'intégration, qui définit la logique d'intégration (ou les logiques d'intégration de chaque itération dans une méthode itérative).

3.3 Conception de l'architecture dans les méthodes itératives

La conception de l'architecture est faite en principe de façon incrémentale. On pourra ainsi à la première itération définir l'organisation générale du programme et l'architecture de la structure d'accueil, et compléter aux itérations suivantes avec les fonctions applicatives. Le document d'architecture correspondant sera alors rédigé de façon incrémentale.

Il sera nécessaire de prendre quelques précautions au niveau de l'architecture. Il faudra commencer par définir une structure d'accueil ou base applicative sur laquelle on pourra ajouter progressivement les fonctions d'application. Cette structure d'accueil ou base applicative comprendra par exemple l'affichage d'un écran de base, la prise en compte des actions opérateurs, les échanges avec le réseau, la gestion de la base de données...

Une solution alternative est de consacrer l'essentiel de la première itération à l'analyse des besoins et de n'y faire qu'une ébauche d'architecture, et de consacrer la deuxième itération à l'architecture et la réalisation d'une structure d'accueil.

3.4 Conception de l'architecture dans les méthodes agiles

Le principe dans les méthodes agiles est de ne surtout pas chercher à optimiser, et limiter les travaux d'architecture pour ne pas anticiper sur ce qui sera réalisé ou non réalisé (inutile de faire du travail pour ce qui ne sera peut être jamais réalisé...).

De même lorsqu'on réalise une fonction, on ne prévoit pas les futures fonctions à implémenter, c'est lorsqu'on implémentera ces fonctions qu'on modifiera si besoin les fonctions déjà implémentées.

Cependant il faut avoir une architecture qui permette l'ajout progressif des fonctions et permette de modifier facilement les fonctions.

Pour cela il est fortement recommandé d'utiliser une conception objet et des schémas d'architecture type Modèle Vue Contrôleur (MVC).

Il est fortement souhaitable avant de commencer à coder des fonctions de faire une étude générale d'architecture au niveau des principaux objets et du déploiement.

Par contre on ne regardera pas le détail du contenu des classes et des interfaces comme recommandé dans les méthodes classiques. Ce détail sera examiné lors de la réalisation et commenté dans le code.

Il semble aussi inutile d'aborder les aspects tels que les interfaces opérateurs, la décomposition en sous-systèmes et la logique d'intégration.

Il n'y aura pas lieu de rédiger un Document d'Architecture Logiciel (DAL) détaillé, mais il sera tout de même utile de consigner les résultats de l'étude générale d'architecture (souvent appelée conception préliminaire) dans un Document de Conception Préliminaire (DCP) qui sera présenté plus loin.

4 - Méthodes de conception

4.1 Les grandes catégories de méthodes

Dans les temps héroïques de l'informatique, la conception était considérée comme un travail d'artiste, et le concepteur pouvait laisser libre cours à son imagination débridée... Le résultat dépendait de l'esprit plus ou moins brouillon du concepteur.

C'est pourquoi des méthodes ont été définies pour canaliser le concepteur et faciliter la mise au point puis la maintenance des logiciels.

Les principales méthodes de conception peuvent être regroupées en deux catégories :

- les méthodes de conception structurée,
- les méthodes de conception orientée objet.

Dans les méthodes de conception structurée, le découpage du logiciel s'appuie sur l'arbre de décomposition fonctionnelle structurée, avec des aménagements pour permettre la réutilisation de l'existant ou l'optimisation de la production de code.

Dans les méthodes objet, à partir des concepts physiques, on découpe en objets encapsulant les structures de données et les traitements relatifs à ces structures ; il n'y a pas de structuration.

Ces méthodes doivent être considérées comme des guides pouvant être adaptés en fonction de la situation, et il reste encore une grande liberté de conception.

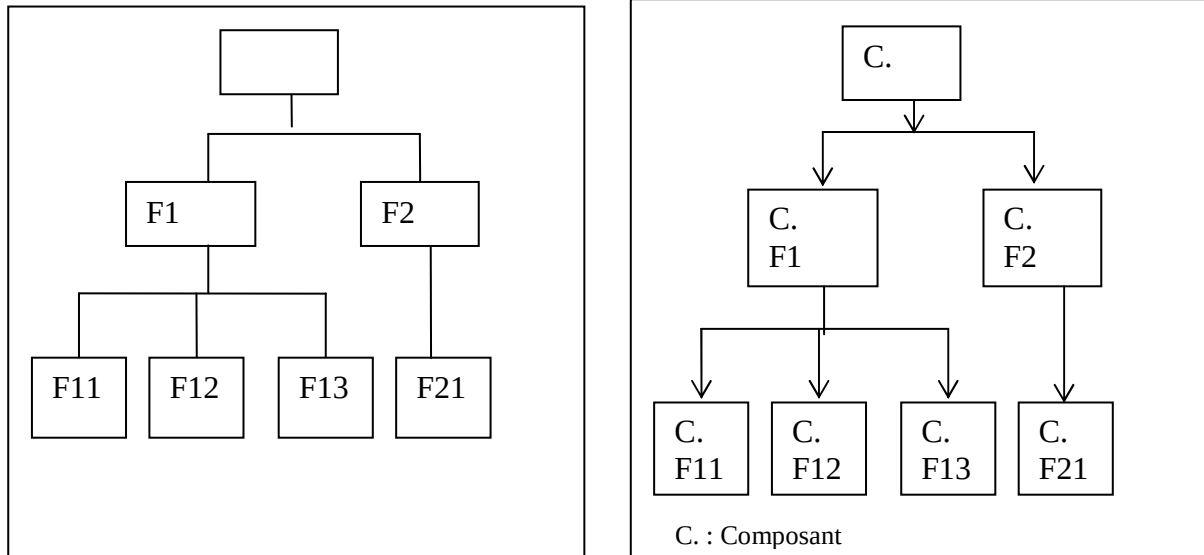
4.2 Méthodes structurées

Dans les méthodes structurées, la décomposition du logiciel en composants (fonctions ou classes) s'appuie sur la décomposition structurée des fonctions du logiciel.

Il y a donc tout intérêt dans une telle méthode à rédiger une STB présentant les fonctions de façon structurée (et non une STB UML).

On commence par associer un composant à chaque fonction identifiée. L'arbre des dépendances correspond alors à l'arbre de décomposition en fonctions.

Décomposition fonctionnelle et arbre de dépendances



On rajoute ensuite dans l'arbre des composants pour les principales fonctionnalités qui résultent des besoins de réalisation (par exemple ouvrir et fermer les fichiers, générer les écrans...) ou des séquences de traitement qui n'avaient pas été identifiées comme fonction dans la décomposition.

On cherche ensuite à optimiser l'arbre en regroupant les traitements qui apparaissent plusieurs fois dans l'arbre, de façon identique ou voisine.

On regarde enfin comment raccorder à l'arbre les logiciels réutilisés, et on modifie l'arbre en conséquence.

On a ainsi défini les principaux composants et leur arbre de dépendances qui ne sera pas identique à l'arbre de décomposition fonctionnelle, mais restera voisin et gardera la même décomposition en niveaux et la correspondance avec les fonctions de haut niveau.

On pourra ensuite passer aux autres travaux de conception (interactions entre composants, répartition entre processeurs, examen des interfaces système, répartition en sous-systèmes, logique d'intégration, vérification de la couverture de la STB) et itérer jusqu'à ce que la solution soit bonne.

4.3 Méthodes orientées objet

4.3.1 Principes

Dans une approche objet, on commence à identifier des objets (et les classes associées) à partir du problème physique, des données et des interfaces externes, et on complète

progressivement ce modèle jusqu'à ce que toutes les fonctionnalités du document d'exigences soient couvertes.

Les intérêts d'une approche objet sont :

- faciliter les évolutions du logiciel,
- faciliter la mise au point du logiciel,
- réaliser des objets réutilisables (objets métier).

On notera que pour que ces intérêts soient réels il faut bien respecter les deux grands principes de la programmation objet : principe d'encapsulation, et principe de limitation de visibilité.

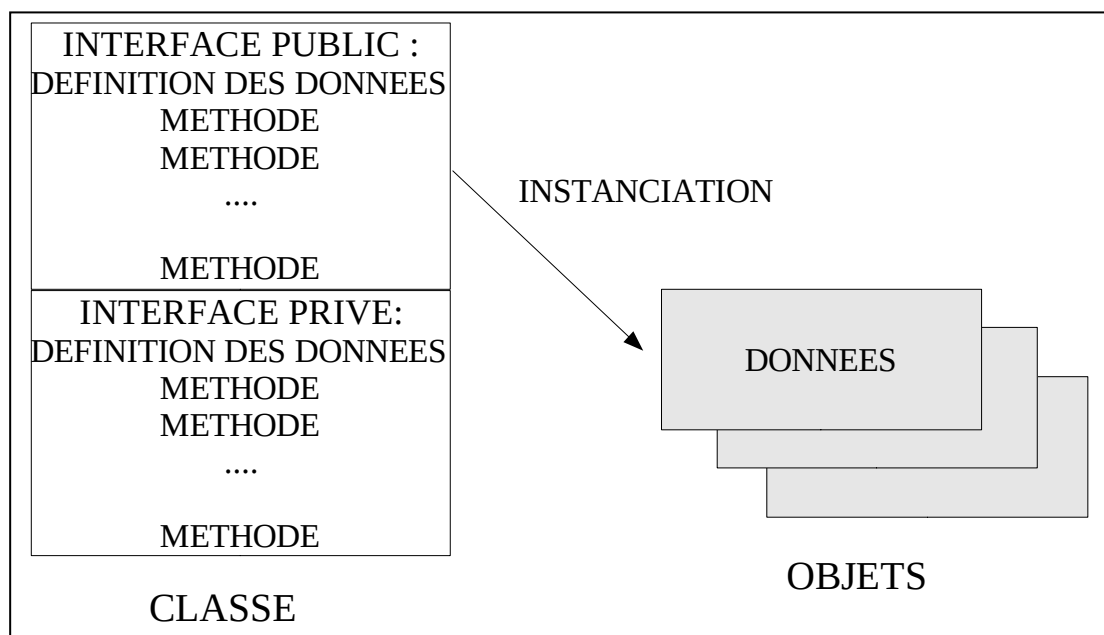
4.3.2 Classes et objets

Dans une approche objet, les composants issus de la décomposition du programme correspondent aux classes (pour la partie instructions) et aux objets (pour la partie données).

Chaque classe regroupe la définition des structures de données et les traitements qui sont propres à ces structures ou dépendent de la représentation des données (méthodes de la classe). C'est le principe d'encapsulation.

Seules les données qui doivent être accédées de l'extérieur de la classe et les méthodes qui peuvent être appelées de l'extérieur de la classe sont définies dans l'interface publique de la classe. C'est le principe de limitation de visibilité.

La classe contient les instructions du sous-programme. Lors de l'exécution du programme, il sera créé un ou plusieurs objets sur la classe ("instanciation"). L'objet contient toutes les données variables qui caractérisent l'exécution et sera implanté dans la zone de données du processus. On considère que ce sont les objets qui s'exécutent en faisant dérouler les instructions de leur classe.



4.3.3 Méthodes orientées objet

Les méthodes orientées objet sont jeunes (moins de 10 ans) et pas très mures, elles présentent plutôt des concepts qu'une démarche guidée

Deux méthodes ont émergé :

- OOD de Grady Booch
- OMT de James Rumbaugh's

Les représentations de ces deux méthodes ont ensuite été unifiées dans UML (Uniform Method Language)

La validation des concepts a été faite sur de petits systèmes, l'extension à de moyens et grands systèmes n'est pas évidente.

Il est donc rarement possible pour un problème donné de se limiter au choix d'une seule méthode existante, il faut aussi définir la démarche.

Je vais donc vous proposer une démarche de conception adaptée à un moyen projet et au travail en équipe, inspirée des méthodes OOD, OMT et DOOS (pour la décomposition).

La démarche est itérative et sera présentée par la suite.

4.4 Le problème de la réutilisation

Les fonctions demandées au logiciel sont de plus en plus complexes. Si l'on veut maîtriser les coûts des réalisations de logiciel, il faut essayer de réutiliser un maximum de logiciel existant. Cela réduira d'autant les travaux de conception détaillée, codage et tests unitaires.

De plus, si l'on pense aux projets futurs on essaiera d'écrire le logiciel de façon à ce que ses sous-programmes soient facilement réutilisables.

Une activité majeure aujourd'hui au niveau de la conception est de rechercher les logiciels existants pouvant répondre à une partie du besoin qui peuvent être réutilisés.

Le choix est très large : bibliothèques associées aux principaux langages, bibliothèques scientifiques, gestion de base de données, accès réseau, outils Internet...

Le logiciel à réaliser devra pouvoir intégrer ces logiciels réutilisés, ou s'interfacer avec. Cela oriente vers le choix d'un langage permettant une utilisation facile des logiciels réutilisés, et une méthode de conception adaptée à ce langage et à l'architecture des logiciels à réutiliser.

Par exemple si on a à faire des calculs de structures, la quasi totalité des logiciels réutilisables est écrite en langage Fortran. On aura intérêt à réaliser le projet en Fortran en dépit de tous les défauts qu'on trouve à ce langage, et adopter une méthode de conception structurée (le Fortran n'est pas un langage objet et ne permet pratiquement pas d'en faire).

Enfin si l'on constate que la plus grande partie du logiciel peut être faite à partir de logiciels réutilisés, ni la démarche structurée ni la démarche objet ne s'appliqueront, et il faudra définir une démarche particulière adaptée au problème.

4.5 Méthode du point milieu et méthodes mixtes

La méthode du point milieu est une variante des méthodes structurée qui prend mieux en compte la réutilisation de logiciels

On commence la conception par les deux bouts : les couches hautes de la décomposition fonctionnelle, et les couches basses.

Pour les couches hautes, on suit la décomposition arborescente des fonctions.

Dans les couches basses, on commence par placer tous les logiciels réutilisés, puis on complète par des sous-programmes couvrant les autres fonctionnalités de base nécessaires pour réaliser les fonctions. On définit ensuite une couche au dessus de cette couche de base, et on progresse jusqu'à se raccorder à l'arbre des couches hautes.

L'approche pour les couches basses est voisine d'une approche objet, et on peut en dériver une méthode mixte où on adopte une approche objet pour les couches basses.

4.6 Choix d'une méthode

Si l'on a bien respecté les règles de base de la conception objet, et que les graphes de dépendance ne sont pas trop complexes, les méthodes orientées objet ont de grands avantages :

- simplification des travaux de mise au point et d'intégration,
- évolutions plus faciles.

Cependant, il n'y a pas de hiérarchie et de structuration entre les classes, ce qui n'est pas facile pour traiter les gros problèmes.

Les méthodes structurées présentent l'avantage d'une organisation structurée par niveaux bien adaptée aux esprits cartésiens. La compréhension est plus facile, et l'approche par niveaux successifs est bien adaptée au traitement de gros problèmes.

Une méthode mixte peut être un bon compromis en traitant en objets la partie du logiciel où cette technique apporte un maximum de bénéfices.

Dans la pratique, le critère principal sera la réutilisation, et après avoir identifié les logiciels pouvant être réutilisés, on cherchera pour ce qu'il restera à développer le langage et la méthode de conception qui semblent le mieux s'adapter aux logiciels réutilisés.

5 - La démarche de conception de l'architecture orientée objet

La démarche proposée pour la conception de l'architecture comprend plusieurs étapes. Chaque étape peut modifier les résultats d'une des étapes précédentes, et il faudra donc faire des rebouclages jusqu'à converger vers une bonne solution.

Les étapes sont :

- l'établissement du modèle logique avec
 - l'identification des principaux objets,
 - l'examen des réutilisations,
 - l'examen des interactions entre objets,
 - l'examen des relations entre classes et l'optimisation des classes,
- l'établissement du modèle des processus et du modèle de déploiement,

- l'examen des interfaces du système,
- le découpage en sous-systèmes,
- la définition de la logique d'intégration,
- la vérification de la couverture des fonctions du document d'exigences.

Lorsque le processus a convergé, on peut finaliser le document d'architecture logiciel et répartir la réalisation des sous-systèmes entre les sous-groupes de réalisation qui pourront chacun faire la conception détaillée de leur sous-système, puis le codage et les tests unitaires.

Dans le cas de méthodes itératives ou agiles, l'étude d'architecture est complétée au cours des itérations. On pourra donc se limiter au départ à une étude globale de l'architecture comprenant :

- l'établissement du modèle logique,
- l'établissement du modèle des processus et du modèle de déploiement.

6 - Le modèle logique

Le document d'exigences, si il a été rédigé avec la technique UML, définit le besoin par des cas d'utilisation correspondant à des séquences d'événements. Pour réaliser le logiciel correspondant, il va falloir trouver des objets qui font dérouler ces séquences d'événements.

On commence donc par identifier des objets (principaux objets) qui vont permettre de remplir les principales fonctionnalités du document d'exigences.

On étudiera ensuite les possibilités de réutilisation et les interactions entre objets en vue de modifier ou compléter ce modèle.

On fera correspondre une classe à chacun de ces objets, on examinera les relations entre ces classes puis on cherchera à optimiser les classes en définissant des classes abstraites ou en revoyant le contenu de certaines classes et objets correspondants.

6.1 Les principaux objets

L'homme ne peut raisonner que sur un nombre limité d'éléments (fonctions, objets...)

Dans une méthode hiérarchique (conception structurée), on fait une décomposition arborescente en niveaux afin d'avoir un nombre limité d'éléments à chaque niveau.

Les méthodes objet ne sont pas hiérarchiques, tous les objets sont sur le même niveau, et il peut y en avoir des centaines ou des milliers...

Lorsque le système n'est pas tout petit, il n'est pas possible à l'homme de raisonner sur tous les objets à la fois, les graphes deviendraient inextricables.

D'autre part un découpage a priori du logiciel n'est pas en général une bonne solution...

Il est donc proposé une autre méthode de décomposition :

- raisonner d'abord sur les principaux objets, ou objets significatifs pour l'architecture, liés au problème (monde réel, fonctions demandées...) indépendants de la manière d'implémenter la solution,

- ensuite, découper en sous-systèmes puis introduire les objets d'implémentation, objets liés à l'implémentation de la solution (par exemple crayon et pinceau de Visual C++).

Dans un premier temps, on peut retenir comme principaux objets :

- les objets du monde réel apparaissant dans le document d'exigences,
- les objets gérants les interfaces (réseaux, périphériques, fichiers...),
- les objets gérants les IHM,
- les objets gérants les principales structures de données,
- les objets remplissant des fonctions importantes du document d'exigences non attribuées à un des objets précédents.

Cette liste est provisoire au début, elle pourra être complétée ou modifiée au cours des étapes suivantes.

Chacun des principaux objets identifiés fera l'objet d'une définition sommaire pour préciser son contour :

- rôle de l'objet : fonctions assurées, données gérées, interfaces avec le monde extérieur ...
- principales méthodes et leurs rôles, objets "serveurs" appelés,
- objets "clients" auxquels il rend service.

6.2 Exemple d'identification des principaux objets

Prenons l'exemple de navettes d'aéroport décrit dans les guides de rédaction, on peut identifier comme objets principaux :

Monde réel	Ligne Station Navette Borne
Interfaces	Réseau Base_de_données
IHM	
Structures de données	Tarif Cumul_ventes
Autres fonctions importantes	Gestion_bornes Gestion_centre_de_contrôle

Il ne s'agit que d'une liste provisoire.

6.3 Examen des réutilisations

Dans la réalisation d'un logiciel aujourd'hui, il faut viser à réutiliser un maximum de logiciel existant.

On recherchera tous les logiciels qui peuvent être réutilisés pour remplir une partie des besoins (bibliothèques graphiques, bibliothèques d'accès réseau, gestion de base de données, programmes utilitaires...), puis la manière dont ces logiciels peuvent être utilisés (appels de bibliothèque, insertion de classes dans le programmes, programmes indépendants...

On pourra aménager ou compléter la liste des principaux objets pour prendre en compte l'interface avec ces logiciels réutilisés.

Par exemple sur l'exemple précédent, on peut identifier la réutilisation des logiciels suivants :

- la bibliothèque des fonctions graphiques du compilateur pour les IHM (par exemple la bibliothèque QT de C++),
- une bibliothèque d'accès à un système de gestion de base de données,
- les utilitaires du SGBD (création de la base de données, sauvegarde et restauration, consultation du contenu de la base de données),
- une bibliothèque d'accès réseau.

Le choix des logiciels retenus peut demander un travail conséquent de recherche documentaire et d'essais, et pourra être fait un peu plus tard.

Ces logiciels réutilisés conduisent à modifier la liste des principaux objets :

- ajouter pour IHM
- ajouter un objet "Accès BD" pour faire l'interface avec la bibliothèque d'accès à la base de données.

6.4 Interactions entre objets

Pour affiner l'identification des objets et de leurs rôles, on construira des diagrammes d'interaction entre objets indiquant les relations de type client - serveur.

On identifiera pour cela dans le document d'exigences des "scénarios" correspondant à des situations typiques des états du système ou de son utilisation, et on construira pour chaque "scénario" un diagramme des objets concernés et des appels entre objets.

Avec une spécification UML, les scénarios correspondront à des cas d'utilisation ou des parties de cas d'utilisation, par exemple :

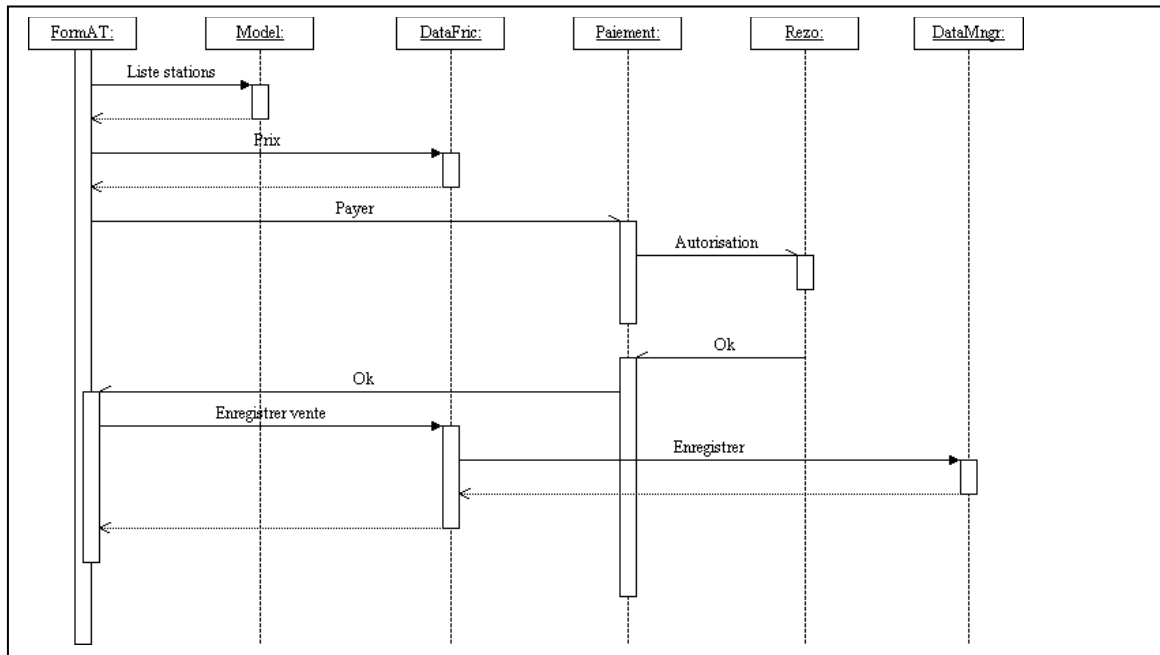
- initialisation du système,
- traitement d'une catégorie de commandes utilisateur,
- arrivée d'un événement.

Les diagrammes pourront être des diagrammes de collaboration montrant les échanges de messages entre objets ou des diagrammes de séquence de messages montrant les séquences d'échanges de messages entre objets (on représentera chaque appel entre objets comme un échange de message).

La réalisation de ces diagrammes doit permettre :

- de préciser le rôle des objets, compléter les méthodes des classes correspondantes, compléter le recensement des objets clients et serveurs,
- détecter des manques d'objet, et définir de nouveaux objets ou modifier des objets pour combler ces manques,
- détecter les objets orphelins, pour les supprimer ou les rendre non orphelins.

Dans l'exemple ci-dessus, le diagramme de séquence suivant montre une vente de billet :



6.5 Les relations entre classes et l'optimisation des classes

Dans une première approche, on associe une classe à chacun des principaux objets, et on examine les relations entre classes (associations, agrégations).

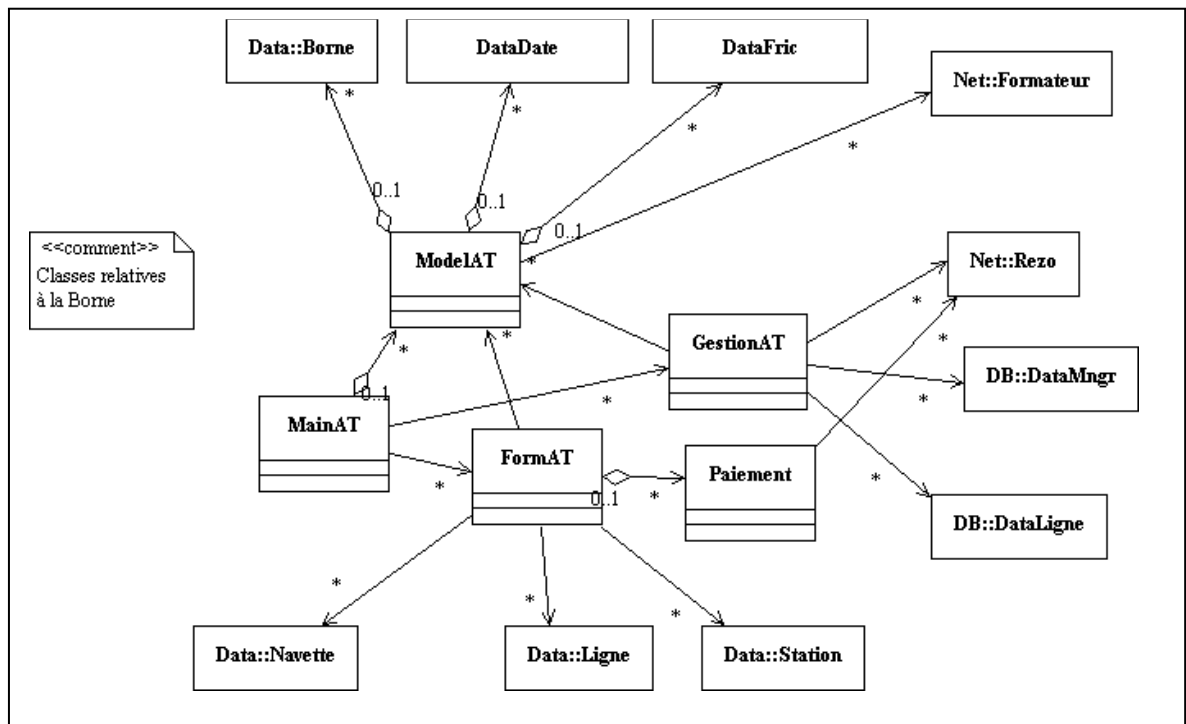
On peut représenter ces relations sur des diagrammes de classe.

On cherchera ensuite à optimiser le modèle des classes qui vient d'être défini :

- en examinant si les classes peuvent hériter de classes existantes (classes de bibliothèque ou classes créées antérieurement),
- en recherchant les propriétés communes à plusieurs classes pour créer des classes abstraites et des relations de généralisation (héritage),
- En cherchant à exploiter les autres possibilités des modèles de classes (spécialisation, agrégation, classes amies, héritage multiple...).

L'objectif est de minimiser la quantité de code à écrire pour implémenter le codage sans pour autant augmenter la complexité du codage...

On peut ainsi arriver à un diagramme du type suivant :



7 - Modèle des processus et modèle de déploiement

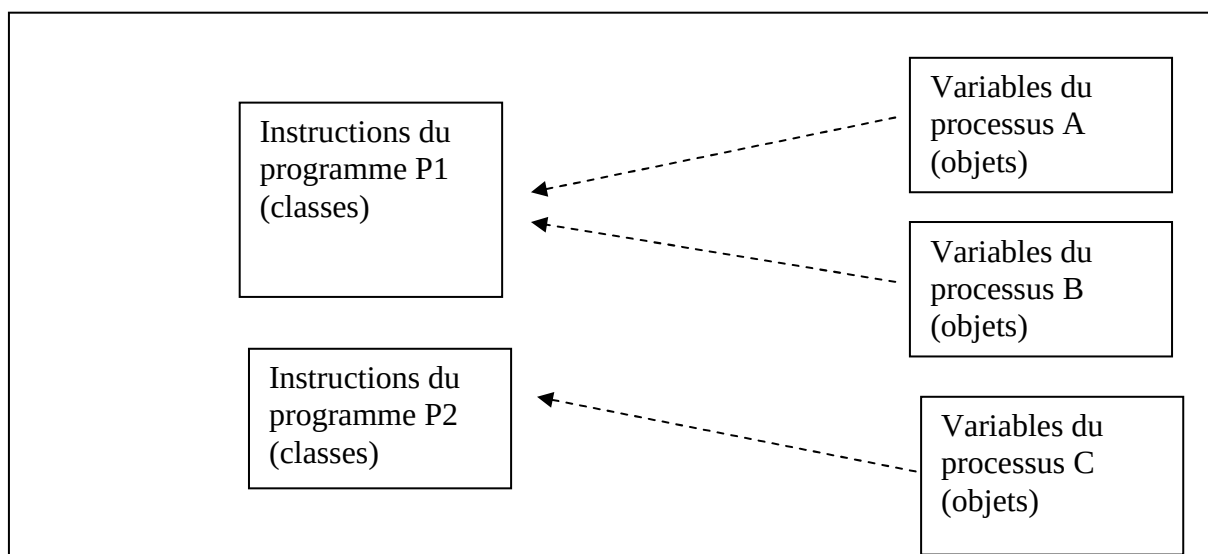
7.1 Programmes et processus

Un programme regroupe les instructions à exécuter, il contient les instructions des méthodes des classes qui le composent.

Un processus est créé sur un programme pour une exécution, plusieurs processus peuvent être créés sur un même programme.

Un processus travaille avec une zone de variables qui lui est propre, il ne peut accéder aux variables des autres processus ni appeler les méthodes des objets des autres processus.

On peut représenter ainsi les programmes et processus dans la mémoire d'un ordinateur :



L'échange d'informations entre processus ne peut se faire que :

- par un passage de paramètres si un processus crée un autre processus,
- par l'intermédiaire de fichiers ou bases de données,
- en faisant appel à des mécanismes de transferts de messages ou signaux fournis par le système d'exploitation.

Cependant les systèmes d'exploitation permettent aussi en général de créer des threads à partir d'un processus. Les threads s'exécutent en parallèle avec le processus et peuvent accéder aux données du processus, ce qui facilite les échanges d'information.

7.2 Découpage en programmes, processus et threads

Les critères à prendre en compte pour le découpage sont :

- le nombre de processeurs (au moins un processus par processeur),
- les contraintes liées aux systèmes d'exploitation (traitement des entrées sorties) et les contraintes temps réel,
- l'indépendance lors de l'exécution des fonctions remplies,
- la différence de nature des fonctions remplies,
- la difficulté de mise au point.

Il faut noter que prévoir plusieurs processus et des threads sur un processeur peut être intéressant pour séparer des fonctions, et peut conduire à un code plus simple à écrire, mais que la mise au point de processus et threads travaillant en parallèle sur un même processeur n'est pas facile.

La répartition en programmes sera définie, chaque classe sera attribuée à un ou plusieurs programmes, et les processus seront précisés (1 ou plusieurs par programme) ainsi que les threads et les processeurs sur lesquels ils s'exécuteront.

Les processeurs et processus pourront être représentés sur un diagramme de déploiement.

Si des objets de même classe sont instanciés dans des processus différents, on vérifiera qu'ils ne travaillent pas sur les mêmes données, sinon il faudra modifier le modèle objet.

Si un objet d'un processus appelait un objet d'un autre processus, il faudra modifier le modèle et créer une interface de communication entre les processus.

On examinera aussi les interactions entre processus ou threads à l'aide de diagrammes d'interaction entre objets, chaque processus ou thread étant représenté sous forme d'un objet.

Les résultats attendus sont :

- besoins en initialisations et lancements de processus, objets ou modifications d'objets pour couvrir ces besoins,
- modifications d'objets induites par le découpage en processus et les contraintes qui en découlent, et les parallélismes d'exécution.

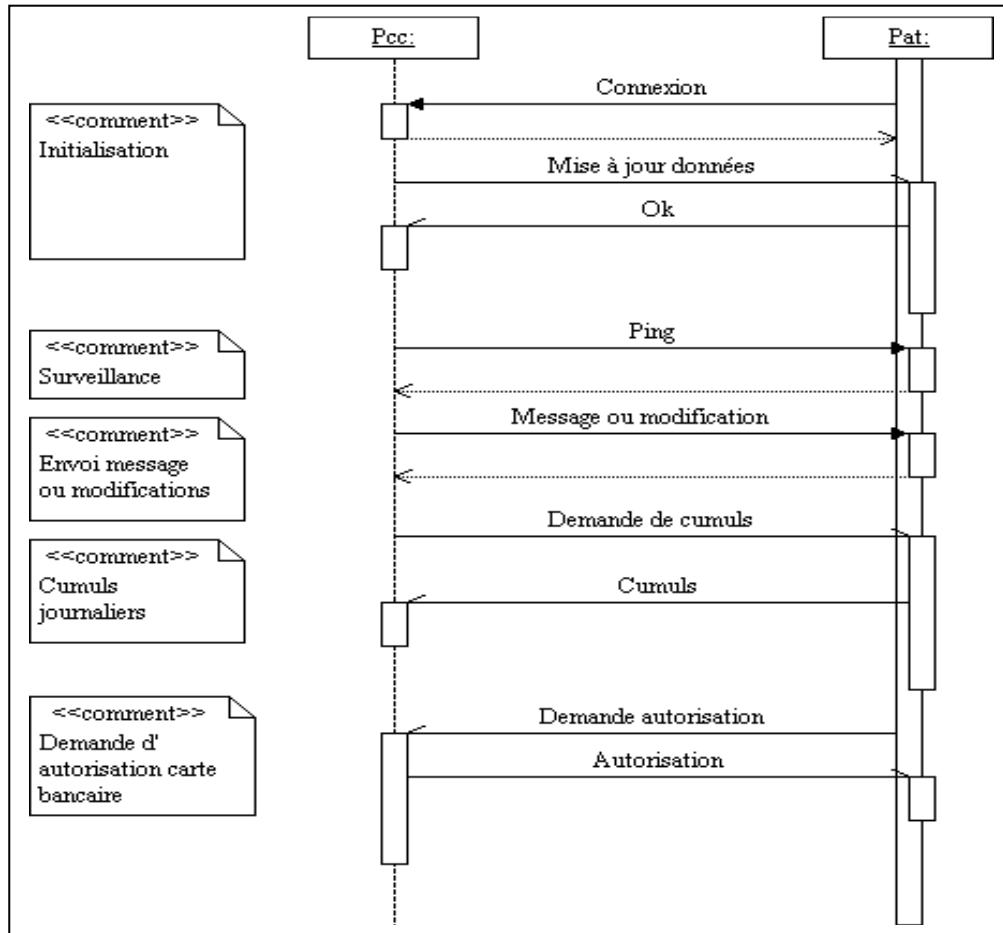
7.3 Répartition en programmes et processus dans l'exemple

Programmes et processus

Processeur	Programme	Processus	Rôle
Borne	PBORNE	1 processus Pborne	Gestion d'une borne

Centre de contrôle	PCC	1 processus Pcc	Gestion du centre de contrôle
--------------------	-----	-----------------	-------------------------------

On peut représenter les interactions entre processus par le diagramme de séquence de messages suivant :



8 - Interfaces système

Les principaux interfaces système à examiner sont en général les données persistantes et l'interface opérateur

8.1 Données persistantes

Les données persistantes sont les données qui sont conservées entre les exécutions des programmes. Ce sont généralement des fichiers disques et des bases de données.

Un examen de ces données est nécessaire pour identifier les logiciels nécessaires pour accéder à ces données, les initialiser, les sauvegarder et les restaurer en cas de problèmes...

Il faut aussi vérifier que le volume de ces données est compatible avec la taille des disques prévus.

Pour cela, on recensera ces données, on recherchera la meilleure solution pour les conserver (type de fichier ou de base de données), on définira les besoins correspondants en logiciels (autant que possible des logiciels réutilisés !) et on estimera le volume.

8.2 Interfaces opérateur

Pour prendre en compte les actions opérateur et générer les écrans, il faudra avoir recours à des fonctions du système et utiliser une bibliothèque. Cela va conduire à choisir une bibliothèque et un mode de programmation répondant bien à son besoin et à introduire dans le modèle les objets principaux qui en découlent.

Par exemple si on retient une solution du type MVC (modèle - vue - contrôleur), on introduira des objets de type Modèle, Vue, Contrôleur.

9 - Décomposition en sous-systèmes

9.1 Critères de décomposition

Pour la conception détaillée et le codage, le logiciel doit être découpé en sous-systèmes pouvant être confiés chacun à un sous-groupe et réalisés en parallèle de façon quasi indépendante.

Le découpage sera fait

- en répartissant les classes abstraites et les classes des principaux objets déjà identifiés,
- en définissant en complément les "contrats" demandés à chaque sous-systèmes (services attendus, fonctions identifiées dans le document d'exigences à assurer...).

Les critères à prendre en compte pour le découpage sont :

- définir des sous-systèmes dont on est capable de définir les "contrats" (ou services attendus) et les interfaces
- pouvoir équilibrer les charges de travail entre sous-groupes de réalisation (un sous-groupe réalisera un ou plusieurs sous-systèmes,
- préparer et faciliter les étapes de tests unitaires et intégration en minimisant les interfaces, les découpages de programmes...,
- permettre une intégration progressive dans un ordre logique en ajoutant les sous-systèmes un par un en fonction de leurs dépendances,
- éventuellement regroupement selon la spécialisation des réalisateurs (IHM, SGBD...)

On pourrait faire le découpage suivant dans l'exemple :

Nom du sous-système	Principales classes	Rôle	Programmes concernés
RESEAU	Rezo Serveur Formateur	Regroupe les fonctions d'accès réseau	PAT PCC
DONNEES	Ligne Station Navette Borne	Regroupe la définition des données et leur accès	PAT PCC
BASE DE DONNEES	DataMngr DataLigne DataFric DataDate DataUser	Accès à la base de données	PAT PCC
GESTION BORNE	MainAT FormAT GestionAT ModelAT Paieement	Base du programme borne	PAT
GESTION CC	MainCC FormCC GestionCC ModelCC Login Autorisation	Base du programme centre de contrôle	PCC

9.2 Définition des sous-systèmes et de leurs interfaces

Pour pouvoir mener de façon quasi indépendante la réalisation des sous-systèmes, chaque sous-système devra faire l'objet d'une définition précise de son contenu et de ses interfaces.

Le contenu du sous-système sera défini par des "contrats" :

- en terme de services, en précisant les services rendus par les principaux objets inclus dans le sous-système,
- en terme de fonctions en précisant les fonctions du document d'exigences couvertes par le sous-système.

Les interfaces du sous-système comprendront :

- les interfaces des classes des objets appelables de l'extérieur du sous-système (dites classes principales ou classes publiques),
- les interfaces des classes dont héritent d'autres sous-systèmes,
- les appels système, les messages échangés, les événements ou interruptions déclenchées, et les interfaces avec les entrées sorties, s'il y en a,
- la définition des fichiers ou base de données accédés, s'il y en a.

Pour toutes les classes dites principales ou publiques, on définira dans le détail l'interface public : paramètres d'appel des méthodes publiques et données publiques (par exemple en écrivant l'interface public en C++).

Pour les fichiers et base de données, il faudra préciser leur structure.

10 - Définition de la logique d'intégration

Lorsque les classes des sous-systèmes auront été codées et testées unitairement, il faudra les intégrer par étapes pour fabriquer le programme ou les programmes composant le logiciel.

Il faut définir dans quel ordre seront faites ces opérations et avec quel calendrier compte tenu du calendrier prévu pour la réalisation des sous-systèmes. La définition de l'ordre prendra en compte les dépendances entre sous-systèmes. Ces aspects seront précisés dans le cours sur la réalisation.

Si le calendrier ou l'ordre n'apparaît pas satisfaisant, on essaiera de revoir le découpage en sous-systèmes ou la répartition de leur réalisation.

Dans une méthode itérative, une logique d'intégration sera définie pour chaque itération.

11 - Vérification de la couverture des besoins

L'objectif est de s'assurer que le modèle et les sous-systèmes définis permettent de couvrir toutes les exigences du document d'exigences.

On pourra procéder à :

- la vérification de couverture fonctionnelle,
- la validation des performances et aspects critiques.

11.1 Vérification de couverture fonctionnelle

Il faut s'assurer que l'ensemble des objets et méthodes définies ou des "contrats" demandés aux sous-systèmes permet de couvrir toutes les exigences du document d'exigences.

On pourra pour cet examen construire une matrice de traçabilité indiquant pour chaque exigence quelles sont les objets et méthodes ou "contrats" concernés et s'ils permettent de satisfaire l'exigence.

La matrice peut avoir la forme suivante :

N°	Exigence	Sous-système	Objets/méthodes ou contrats	Remarques

Les différents cas d'utilisation du document d'exigences seront découpés en exigences élémentaires repérées par une numérotation dans le document d'exigences (par exemple N° du cas d'utilisation suivi du N° dans la séquence d'événements) et un libellé. Un exemple de découpage en exigences élémentaires est donné dans le chapitre sur l'analyse du besoin. En face de chaque exigence on indiquera dans quel sous-système elle est prise en compte, et par quel objet et méthode, ou quel contrat demandé au sous-système. On notera dans les remarques si l'exigence est entièrement satisfaite ou doit faire l'objet d'une validation comme performance ou aspect critique.

Pour chaque exigence non entièrement satisfaite, on cherchera une solution telle qu'ajout de méthode, modification d'objet ou de méthode, nouvel objet, nouveau contrat...et on mettra à

jour le modèle, la définition des sous-systèmes et la matrice de traçabilité jusqu'à qu'il ne reste plus de problème identifié.

11.2 Validation des performances et aspects critiques

Il faut s'assurer que les performances et autres aspects critiques (par exemples contraintes de délais) sont bien tenus par la solution qu'on vient de définir.

On pourra selon le cas :

- faire des analyses "papier",
- définir des scénarios et étudier les interactions entre objets ou processus sur des diagrammes,
- réaliser des maquettes d'évaluation, ou utiliser les premiers prototypes dans une méthode itérative pour des évaluations.

Le type de diagramme devra être adapté à la nature et à l'étendue du problème étudié, selon le cas, on pourra utiliser :

- des diagrammes de collaboration (objets messages),
- des diagrammes de sequences de messages,
- des diagrammes états transitions (système de type automate).

12 - Fin de la conception de l'architecture

On pourra considérer que la conception de l'architecture est terminée lorsque toutes les étapes auront été parcourues, que l'on aura fait tous les rebouclages nécessaires pour mettre en cohérence les différents aspects du modèle, et que la solution obtenue sera jugée suffisamment "bonne".

Dans une méthode de développement itérative ou agile, l'architecture sera étudiée de façon incrémentale sur les premières itérations en analysant de nouvelles fonctions à chaque itération. Il faut éviter lors d'une itération d'avoir à remettre en cause l'architecture définie à l'itération précédente.

Pour cela, à la première itération on analysera de façon grossière l'ensemble des fonctions, et on cherchera une organisation du logiciel qui se prête à une réalisation incrémentale (par exemple une organisation basée sur une structure d'accueil des applications).

Les résultats de la conception de l'architecture sont consignés un document d'architecture.

13 - Le document d'architecture logiciel (méthodes non agiles)

13.1 Objectifs et plan type

Le document d'architecture logiciel a plusieurs objectifs :

- il constitue le document d'entrée pour les sous-groupes qui feront la conception détaillée des différents sous-systèmes, puis le codage et essais des sous-programmes,
- il permet de présenter la solution au client, et aux participants à la revue de conception préliminaire si elle est prévue,
- il servira aux équipes qui feront ultérieurement les évolutions du logiciel pour comprendre comment est organisé le logiciel.

Le contenu de ce document sera plus ou moins développé selon l'importance et la criticité du projet. Les sujets abordés dépendront aussi de la méthode de conception choisie.

On ne peut donc fournir un plan universel. Nous nous limiterons à fournir un plan pour un dossier réduit dans une méthode de conception objet. Un guide de rédaction de ce type de document est aussi fourni. On pourra adapter ce plan dans les autres situations.

Le plan proposé pour un document d'architecture réduit dans une méthode objet est le suivant :

1	Introduction 1.1 Objet 1.2 Terminologie et sigles utilisés 1.3 Documents de référence
2	Présentation générale de l'architecture
3	Les principaux objets et leurs classes 3.1 Liste des principales classes 3.2 Relations entre classes 3.3 Rôles et interfaces des principales classes
4	Scénarios d'interaction entre objets
5	Les processus et le déploiement 5.1 Déploiement, programmes, processus et threads 5.2 Principes de fonctionnement des processus 5.3 Interactions entre processus ou threads
6	Interfaces externes du système
7	Découpage de réalisation (sous-systèmes) 7.1 Liste des sous-systèmes et leurs relations 7.2 Fonctions et interfaces externes des sous-systèmes
8	Dimensionnements et performances

Ce plan type est adapté à un petit système qui ne présente pas de caractère de sûreté ou de criticité, dans le cas contraire un document plus étoffé serait probablement demandé, et pourraient être à rajouter :

- les objectifs de l'architecture et les contraintes,
- le rappel des cas d'utilisation avec les séquences d'événements significatives,
- des diagrammes montrant la réalisation des cas d'utilisation,
- des objectifs de qualité,
- une matrice de traçabilité entre les exigences de la STB et les sous-systèmes et classes,
- des justifications de choix d'architecture, à moins que la méthode de développement ait prévu un dossier justificatif d'architecture pour expliciter ces choix.

13.2 Rédaction du document

L'introduction précise l'objet du document (description de l'architecture), le système concerné, la terminologie, et les références des documents ayant servi de référence pour l'architecture (la STB principalement) ou pouvant aider à comprendre le problème posé.

La présentation générale de l'architecture présente rapidement l'organisation du logiciel en indiquant les principaux éléments de l'architecture et les principaux choix faits, par exemple :

- découpage en programmes, répartition du logiciel entre les calculateurs,
- processus et threads éventuels,
- architecture réseau et accès réseau,

- accès aux bases de données (par serveur base de données, base réseau...),
- conception objet,
- mode de programmation (événementiel ou séquentiel...),
- langages et systèmes d'exploitation,
- principaux logiciels réutilisés...

Les principaux objets et leurs classes comprennent :

- la liste des principales classes avec leur objet principal,
- les diagrammes de relations entre classes,
- pour chaque classe principale, le rôle détaillé et les principales données et méthodes, et la définition de l'interface.

Les scénarios d'interaction montrent quelques scénarios typiques sélectionnés à l'aide de diagrammes de collaboration entre objets ou de séquence de messages. Ces scénarios pourront montrer le déroulement de cas d'utilisation si la spécification en comprend.

Les processus et le déploiement présentent :

- le découpage du logiciel en programmes et les processus attachés aux programmes,
- le déploiement sous forme de schéma montrant les principaux matériels, les réseaux et les interfaces externes, avec indication des processus exécutés sur chaque calculateur,
- les principes de fonctionnement des processus,
- les interactions entre processus ou threads (diagrammes de séquence de messages).

Les interfaces externes du système présentent les caractéristiques des différents interfaces du système, par exemple : interface opérateur, interface avec une base de données ou des fichiers, des interfaces de communication, interfaces avec des matériels spécifiques, interfaces avec des outils logiciels...

Le découpage de réalisation présente le découpage en sous-systèmes qui a été fait pour la conception détaillée et le codage. On fournira la liste des sous-systèmes et un diagramme montrant les dépendances entre sous-systèmes. Pour chaque sous-système on précisera les fonctions (principales classes du sous-système et autres fonctions) et les interfaces externes (principales classes servant d'interface, événements transmis ou reçus d'autres sous-systèmes, interfaces avec le réseau...).

Les dimensionnements et performances précisent les dimensionnements ou contraintes de performances significatifs pour l'architecture, par exemple :

- nombre maximum d'utilisateurs ou nombre d'utilisateurs simultanés,
- tailles maxima de tables,
- temps de réponse
- délais à respecter...

14 - Le document de conception préliminaire (méthodes agiles)

14.1 Objectifs et plan type

Le document de conception préliminaire a plusieurs objectifs :

- il constitue un document d'entrée pour la définition du contenu des itérations et les programmeurs qui feront la conception détaillée des fonctions à implémenter, puis le codage et essais des sous-programmes,

- il servira aux équipes qui feront ultérieurement les évolutions du logiciel pour comprendre comment est organisé le logiciel.

On ne peut donc fournir un plan universel. Nous nous limiterons à fournir un plan pour un dossier réduit dans une méthode de conception objet. Un guide de rédaction de ce type de document est aussi fourni. On pourra adapter ce plan dans les autres situations.

Le plan proposé pour ce document de conception préliminaire est le suivant :

1	Introduction 1.1 Objet 1.2 Terminologie et sigles utilisés 1.3 Documents de référence
2	Présentation générale de l'architecture
3	Les principaux objets et leurs classes 3.1 Liste des principales classes et leur rôle 3.2 Relations entre classes
4	Scénarios d'interaction entre objets
5	Les processus et le déploiement
6	Base de données
7	Dimensionnements et performances

14.2 Rédaction du document

L'introduction précise l'objet du document (description de l'architecture), le système concerné, la terminologie, et les références des documents ayant servi de référence pour l'architecture (la SGB principalement) ou pouvant aider à comprendre le problème posé.

La présentation générale de l'architecture présente rapidement l'organisation du logiciel en indiquant les principaux éléments de l'architecture et les principaux choix faits, par exemple :

- découpage en programmes, répartition du logiciel entre les calculateurs,
- processus et threads éventuels,
- architecture réseau et accès réseau,
- accès aux bases de données (par serveur base de données, base réseau...),
- conception objet,
- mode de programmation (événementiel ou séquentiel...),
- langages et systèmes d'exploitation,
- principaux logiciels réutilisés...

Les principaux objets et leurs classes comprennent :

- la liste des principales classes avec leur objet principal,
- les diagrammes de relations entre classes.

Les scénarios d'interaction montrent quelques scénarios typiques sélectionnés à l'aide de diagrammes de collaboration entre objets ou de séquence de messages.

Les processus et le déploiement présentent :

- le découpage du logiciel en programmes et les processus attachés aux programmes,
- le déploiement sous forme de schéma montrant les principaux matériels, les réseaux et les interfaces externes, avec indication des processus exécutés sur chaque calculateur.

On pourra compléter cette partie si cela s'avère utile par les principes de fonctionnement des processus (sous forme de diagrammes) et les interactions entre processus ou threads (diagrammes de séquence de messages).

Lorsque le logiciel utilise une base de données, existante ou à définir, on pourra montrer les schémas de cette base

Les dimensionnements et performances précisent les dimensionnements ou contraintes de performances significatifs pour l'architecture, par exemple :

- nombre maximum d'utilisateurs ou nombre d'utilisateurs simultanés,
- tailles maxima de tables,
- temps de réponse
- délais à respecter...