Exemple de développement d'une application

Département Informatique

TELECOM SudParis — 1ère année

Dépt INF

2009/2010

Table des matières

2

1 Objectifs de l'étude de cet exemple

- Illustration de la démarche générale de développement
  - différentes étapes du développement
  - exemples de livrable pour chaque étape
- Préparation de la partie pratique sur les applications graphiques et web
  - connaissance de la structure générale de l'application exemple
  - différentes structures de données utilisées
  - interfaces des différents modules de l'application

Comm. Web

Seul l'aspect développement est pris en compte ici. L'aspect gestion de projet (planification, répartition du travail, suivi d'activité...) n'est pas retranscrit.

Dans le cadre des projets du module CSC3502, les rôles ne sont pas aussi rigoureusement définis qu'ils peuvent l'être dans des projets réels : l'enseignant responsable est en effet à la fois utilisateur final et maître d'ouvrage (i.e. donneur d'ordre et porteur du besoin fournissant les spécifications fonctionnelles) et l'étudiant chef de projet à la fois maître d'œuvre (responsable de la conception, du bon déroulement des travaux, de la coordination des divers prestataires et de la qualité technique) et développeur contribuant à la réalisation.

2 Énoncé du sujet

3

Il s'agit d'écrire une application de gestion des ouvrages d'une bibliothèque.

Les fonctionnalités qui devront être offertes par l'application sont :

- Ajouter un ouvrage
- Supprimer un ouvrage
- Rechercher un ouvrage
- Afficher l'ensemble des ouvrages

4

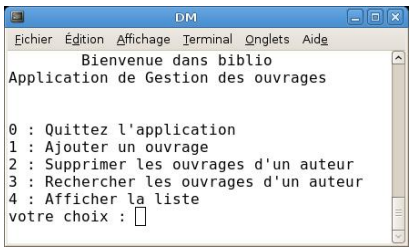
3 Cahier des charges

Les fonctionnalités décrites dans le sujet sont à implanter dans un premier temps avec une interface mode "terminal", puis le seront avec une interface graphique.

Le langage de développement choisi est le C.

Les fichiers source doivent être livrés avec un makefile.

L'interface de l'application doit être conforme aux maquettes fournies ci-dessous :



Comm. Web

Le cahier des charges est un document de référence contractuel qui formalise les besoins pour en assurer une compréhension homogène par tous les acteurs et cadrer la mission de ces derniers. Il peut comporter une partie technique énumérant les contraintes techniques à respecter. Il permet de garantir que les livrables fournis par le maître d'œuvre sont conformes à ce qui a été demandé par le maître d'ouvrage, et d'éviter des modifications de la demande en cours de projet, comme l'introduction de nouvelles fonctionnalités non prévues initialement, par exemple.

Concernant l'application proposée, une extension est envisagée dans un deuxième temps : une consultation en mode web à travers l'utilisation d'un navigateur (une application complète en ligne avec ajout et suppression demanderait la gestion de l'accès concurrent aux données de l'application côté serveur). La maquette de l'IHM (interface homme-machine) de l'extension envisagée est fournie ci-dessous. Cette extension devra être utilisable avec un client navigateur indépendant de l'OS. On utilisera un serveur Apache sous Linux.

5

4 Spécification

Comm. Web

L'objectif de cette phase est de décrire précisément et exhaustivement les fonctionnalités offertes par l'application (le QUOI ?). Les documents de spécification ont vocation à être validés par le client.

Pour une application de taille modeste, la spécification se limite en général à la spécification fonctionnelle qui décrit les fonctionnalités de l'application et les conditions d'utilisation de ces fonctionnalités (opérations à exécuter par l'utilisateur, interactions, règles...), souvent au travers de "cas d'utilisation".

Pour un développement plus conséquent, la spécification peut comporter une spécification d'architecture (ou étude technique) pour décrire les moyens techniques à mettre en oeuvre et l'organisation générale (par exemple en couches) de l'application et du système informatique dans lequel elle doit être intégrée.

4.1 Analyse

Une relecture détaillée du cahier des charges initial avec le "client" de l'application est souvent nécessaire pour :

- donner des réponses à toutes les questions,
- lever toutes les ambiguïtés.

 # 6

Par exemple, qu'est-ce qu'un ouvrage ?

Un ouvrage est en général composé des informations suivantes :

- le titre,
- l'auteur (principal),
- l'année d'édition,
- l'éditeur...

Comm. Web

On ne considère pas ici le cas d'ouvrages avec plusieurs auteurs.
De nombreuses questions doivent recevoir des réponses :

- Est ce que tout le monde aura accès à toutes les fonctionnalités ? Sinon, quels sont les types d'utilisateurs et leurs droits respectifs, et comment s'authentifie-t-on ?
  On considère ici que tous les utilisateurs ont le même privilège (c'est-à-dire qu'il n'y a pas de notion d'administrateur) et qu'ils n'ont pas à s'authentifier.
- Doit-on gérer les accès simultanés ?
  On considère qu'à un moment donné, il n'y a qu'un seul utilisateur de l'application.
- Connaît-on le nombre maximum d'ouvrages ?
  La collection d'ouvrages est de taille quelconque.
- Faut-il créer un seul fichier pour stocker les ouvrages ou plusieurs ?
  On considère qu'il faut créer un seul fichier, c'est-à-dire s'assurer a minima de la persistance des données.
- Le fichier doit-il être localisé sur le compte de l'utilisateur ou sur un compte spécifique ?
  Sur le compte de l'utilisateur.
- La recherche a-t-elle lieu dans le fichier ou en mémoire ?
  Afin de limiter les accès fichier, la recherche sera effectuée sur la représentation interne de la collection d'ouvrages en mémoire et non pas dans le fichier.

7

4.2 Description des fonctionnalités

Les fonctionnalités offertes par l'application sont les suivantes :

- Ajouter un ouvrage
  → doit permettre à l'utilisateur de saisir les informations d'un ouvrage.
- Supprimer un ouvrage
  → doit permettre à l'utilisateur de saisir les informations utiles pour la suppression (selon un ou plusieurs critères d'un ou plusieurs ouvrages)
- Rechercher un ouvrage
  → doit permettre à l'utilisateur de saisir les informations utiles pour la recherche (selon un ou plusieurs critères d'un ou plusieurs ouvrages)
- Afficher l'ensemble des ouvrages
  → doit afficher tous les ouvrages.

Comm. Web

Il convient de préciser les conditions d'utilisation des fonctionnalités :

- Ajouter un ouvrage : on vérifiera au préalable que l'ouvrage n'existe pas déjà pour ne pas créer de doublon, et on refusera la création d'un ouvrage lorsque les informations sont incomplètes (cette spécification n'est pas prise en compte dans l'implantation proposée).
- Supprimer un ouvrage : on affichera tous les ouvrages trouvés. Pour simplifier, on considère qu'on supprime tous les ouvrages trouvés selon tous les critères proposés, et pour simplifier l'implantation, un seul critère sera utilisé dans la suite, le nom de l'auteur.
- Rechercher un ouvrage : on affichera tous les ouvrages trouvés. Pour simplifier l'implantation, la recherche ne portera dans la suite que sur le critère nom de l'auteur (et, comme prévu lors de l'analyse, elle sera effectuée sur la représentation interne de la collection d'ouvrages en mémoire).

4.3 Plan de tests

Dès la spécification, on doit prévoir les tests de validation.

8

Exemple de tests à prévoir pour la fonctionnalité "Ajouter un ouvrage" :

- Donnée : un ouvrage déjà existant à ajouter
  Résultat : message d'erreur : "L'ouvrage existe déjà".

- Donnée : un ouvrage incomplet à ajouter
  Résultat : message d'erreur : " Veuillez renseigner toutes les informations demandées"

Comm. Web

On rappelle que l'implantation proposée ne vérifie pas la contrainte du premier de ces deux tests.
Par ailleurs, il est clair que si l'on souhaitait améliorer le confort d'utilisation de l'application, on pourrait par exemple :

- demander une confirmation à l'utilisateur lors des suppressions,
- ajouter une liste de mots-clés pour aider à la recherche,
- effectuer un tri sur l'auteur ou le titre pour l'affichage...

9

5 Conception préliminaire

Comm. Web

Cette première étape de la conception vise à décrire à haut niveau et d'un point de vue "informatique" (le COMMENT) :

- les structures de données qui seront utilisées dans l'application,
- le stockage de l'information,
- la structuration de l'application en modules,
- toutes les fonctions nécessaires au fonctionnement de l'application (prototypes et description sommaire),
- les tests d'intégration à prévoir.

10

5.1 Les structures de données

Représentation d'un ouvrage :

```
structure ouvrage
    titre         : Chaîne de caractères
    nomAuteur     : Chaîne de caractères
    anneeEdition  : Entier
fstruct
```

On peut également introduire une structure bibliothèque :

```
structure bibliotheque
    nomFichier     : Chaîne de caractères
    listeOuvrages  : ListeOuvrages
fstruct
```

Comm. Web

Un ouvrage est caractérisé par un ensemble de propriétés. Pour simplifier, on ne considère dans la suite que les trois premiers champs : le titre de l'ouvrage, son auteur, l'année d'édition (voir la partie analyse).

Il faut aussi pouvoir gérer un ensemble d'ouvrages : pour cela, on pourra considérer la structure de tableau ou de liste chaînée. Indépendamment de la représentation interne choisie, nous parlerons dans la suite de liste d'ouvrages, sous la forme du type abstrait ListeOuvrages.

La structure bibliothèque regroupe des informations sur le stockage fichier de la collection d'ouvrages (le nom du fichier, information de type Chaîne de caractères), et sur sa représentation en mémoire (information de type ListeOuvrages).

5.2 Stockage de l'information

Stockage des informations de l'ensemble des ouvrages dans un fichier texte *data.txt*.

⇒ modifiable dans un éditeur de texte ⇒ facilite les tests

Structure du fichier volontairement simplifiée :

- une information par ligne,
- une ligne vide entre deux ouvrages successifs.

11

```
Introduction au génie logiciel
J-L. Raffy
2005

Mon Système Linux
C. Schüller
2002
```

Comm. Web

La structure du fichier pourrait bien entendu être plus sophistiquée ; par exemple :

- une première ligne avec un commentaire précisant les significations des informations.
- les lignes suivantes : 1 ouvrage par ligne, avec les champs séparés par un ';' (ou encore format CSV, i.e. Comma Separated Values).

5.3 Les modules

On choisit ici de décomposer l'application en cinq modules :

- *main* : correspond à la fonction principale, avec le point d'entrée de l'application (initialisation, affichage d'un menu),
- *menu* : dédié à l'interaction utilisateur (affichage/saisie d'informations, boucle d'interaction avec l'utilisateur),

12

- *ouvrage* : dédié à la gestion des ouvrages,
- *bibliotheque* : dédié à la gestion des listes d'ouvrages,
- *util* : fonctionnalités de plus bas niveau liées au stockage de l'information (sur disque et en mémoire).

Le module *menu* sera clairement impacté par la déclinaison selon les différentes modalités d'utilisation (terminal, graphique) ; c'est aussi le cas de *ouvrage* et *bibliotheque*

Comm. Web

Différentes décompositions en modules sont possibles. Celle que nous proposons ici tente de concilier d'une part la contrainte des différentes modalités de fonctionnement (terminal, graphique) prévues par le cahier des charges et d'autre part une factorisation maximale des fonctionnalités communes aux différentes versions.

Ce premier découpage va s'avérer insuffisant lors de l'étape de conception détaillée, mais il permet néanmoins de recenser les fonctionnalités indispensables et de préciser les interactions entre modules.

Dans un deuxième temps, l'implantation des modules *menu*, *ouvrage* et *bibliotheque* nous amènera à distinguer les opérations spécifiques à chaque modalité d'utilisation.

13

5.4 Interfaces des modules

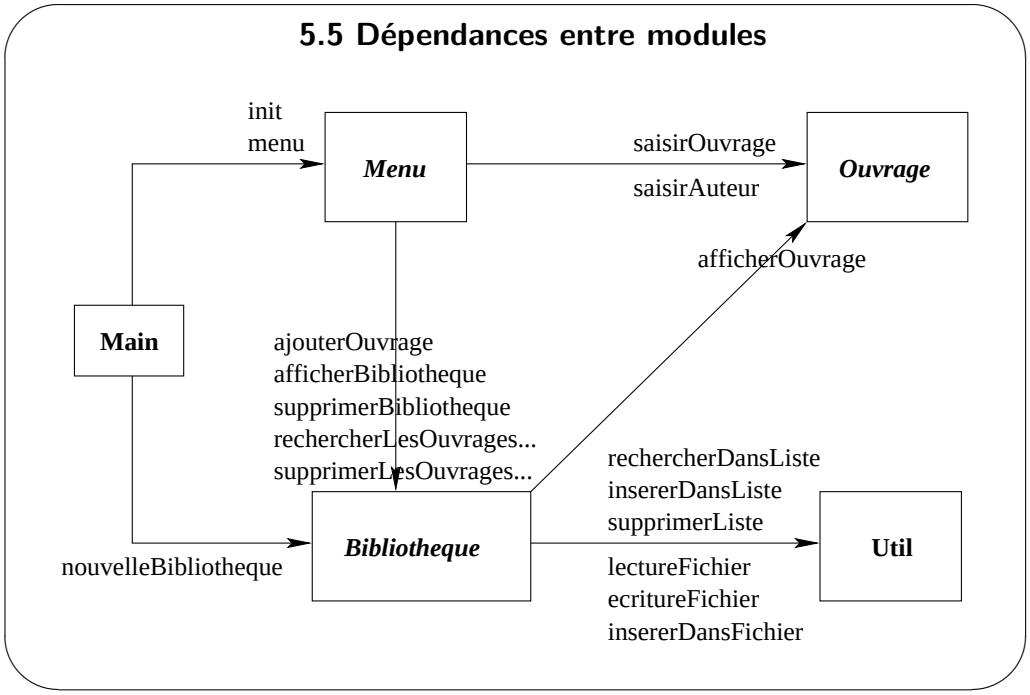
On dresse ensuite la liste des fonctions assurant l'interface des différents modules

- *menu.h* : interaction utilisateur
  init(), menu(), afficher()
- *ouvrage.h* : gestion des ouvrages
  afficherOuvrage(), afficherAuteur(), saisirOuvrage(), saisirAuteur()
- *bibliotheque.h* : gestion des listes d'ouvrages
  nouvelleBibliotheque(), supprimerBibliotheque(), ajouterOuvrage(), supprimerLesOuvragesDeAuteur(), rechercherLesOuvragesDeAuteur(), afficherListeOuvrages(), afficherBibliotheque()
- *util.h* : gestion du stockage
  insererDansListe(), rechercherDansListe(), supprimerListe(), insererDansFichier(), lectureFichier(), ecritureFichier()

Comm. Web

L'interface d'un module est la partie publique du module, qui sera décrite en C dans un fichier d'entête (.h). Elle expose les prototypes des fonctions fournies par le module (services disponibles) et que les fonctions des autres modules peuvent appeler.
La partie privée du module (.c) fournit quant à elle l'implantation (le corps) de ces fonctions et définit toute fonction utile à cette implantation. Par exemple dans le cas du module *menu*, une approche descendante de la conception de la fonction menu() amènera vraisemblablement à définir des fonctions choixAjouter(), choixRechercher(), choixSupprimer() et choixAfficher(). Ces fonctions resteront privées et n'apparaissent donc pas au niveau de l'interface.

14



Comm. Web

Pour chacun des modules *menu*, *ouvrage* et *bibliotheque*, les fonctions seront réparties dans deux fichiers sources lors du codage : l'un pour la partie générique, constituée des fonctions du module dont l'implantation est indépendante des modalités d'interaction avec l'utilisateur, l'autre pour la partie spécifique à chaque modalité. Ce découpage peut être adopté pendant la conception.

Pour avoir un exemple du fonctionnement de ces dépendances, on peut considérer le cas d'utilisation où l'utilisateur fait le choix d'ajouter un ouvrage, ce qui devrait a priori produire dans la fonction menu() un appel d'une fonction choixAjouter() du module *menu*. Cette fonction appelle alors saisirOuvrage() du module *ouvrage*, puis ajouterOuvrage() du module *bibliotheque* qui elle-même appelle insereDansListe() et insererDansFichier() du module *util*. A noter que la fonction saisirOuvrage() fait partie de la partie spécifique du module *ouvrage*, ce qui n'est pas le cas de la fonction ajouterOuvrage() pour le module *bibliotheque*.

5.6 Description des fonctions

La description sommaire des fonctions de l'application peut alors suivre une méthode descendante (top-down), en commençant par la fonction principale :

 # 15

```
fonction principale() : CodeRetour
   bibliotheque : Bibliotheque
   bibliotheque = nouvelleBibliotheque(nomFichier)
   init()   // Initialisations diverses (affichage...)
   menu(bibliotheque)
   retourner OK
ffct
```

Comm. Web

L'approche descendante amène de proche en proche à décrire toutes les fonctions et procédures nécessaires. par exemple :

```
procédure menu (donnée-résultat bibliotheque : Bibliotheque)
    afficherMenu()
    tq (choix ← menu()) <> quitter faire
        cas où choix vaut
            ajout :
                choixAjouter(bibliotheque)
            supprimer :
                choixSupprimer(bibliotheque)
            rechercher :
                choixRechercher(bibliotheque)
            afficher :
                choixAfficher(bibliotheque)
            autre :
                afficher("Erreur dans le choix du menu")
        fcas
        afficherMenu()
    ftq
fproc
```

À titre d'exemple, nous considérons ci-dessous la fonctionnalité d'ajout d'un ouvrage, qui doit permettre à l'utilisateur de saisir les informations d'un ouvrage. Ces informations saisies doivent être persistantes (sauvegardées dans le fichier) :

```
procédure choixAjouter (donnée-résultat bibliotheque : Bibliotheque)
    Ouvrage ouvrage
    ouvrage ← saisirOuvrage()
    ajouterOuvrage( bibliotheque, ouvrage )
fproc
```

Cette procédure choixAjouter() utilise une fonction et une procédure que nous explicitons sommairement :

fonction **saisirOuvrage** () : Ouvrage

```
    // Il faut saisir successivement les informations
    // caractéristiques de l'ouvrage
    // en utilisant des modalités spécifiques pour chaque version
ffct
```

Ainsi, au cours de la descente dans l'écriture, nous finissons par rencontrer la nécessité de tenir compte des modalités d'interaction avec l'utilisateur. Dans la suite, nous nous intéressons exclusivement à la version "terminal" de l'application, et l'implantation de chacun des trois modules *menu*, *ouvrage* et *bibliotheque* comportera une partie générique et une partie propre à la version "terminal".

```
procédure ajouterOuvrage (donnée ouvrage : Ouvrage, donnée-résultat bibliotheque : Bibliotheque)
    // Garder l'information dans la liste
    insererDansListe(ouvrage, bibliotheque -> listeOuvrages)
    // Garder l'information dans le fichier
    insererDansFichier(ouvrage, bibliotheque->nomFichier)
fproc
```

Nous devons bien entendu expliciter les deux procédures utilisées par ajouterOuvrage() :

```
procédure insererDansListe ( donnée ouvrage : Ouvrage, donnée-résultat liste : ListeOuvrages)
    // L'insertion d'un élément dans une
    // liste est un problème connu
fproc
```

```
procédure insererDansFichier ( donnée ouvrage : Ouvrage, donnée-résultat fichier : Fichier)
    // Ecrire un item dans un fichier est connu
    // Il faudra cependant étudier le format
fproc
```

5.7 Exemple de tests d'intégration

16

Pour la fonction "ajouterOuvrage()" :

- Donnée : un ouvrage déjà existant à ajouter
  Résultat : liste et fichier inchangés
- Donnée : un ouvrage incomplet à ajouter
  Résultat : liste et fichier inchangés
- Donnée : un ouvrage saisi correctement, mais liste et fichier inexistants
  Résultat : liste créée avec l'ouvrage saisi, fichier créé contenant la liste

Comm. Web

Un ou plusieurs de ces tests ne seront peut-être pas réalisables. Cela va dépendre de la conception détaillée des fonctions utilisées. Par contre le fait de décrire les tests fait réfléchir au meilleur moyen d'implanter les fonctions concernées.
On peut convenir soit que l'application ne peut être lancée s'il n'existe pas de fichier, soit de créer le fichier s'il n'existe pas (cette dernière solution est retenue dans l'implantation proposée).

 # 17

6 Conception détaillée de la version "terminal"

Cette étape de la conception va fournir :

- le contenu détaillé des différentes structures de données
- le corps de toutes les fonctions et procédures de l'application
  - en distinguant, pour les modules *menu*, *ouvrage* et *bibliotheque*, partie générique de l'interface et partie spécifique à la version "terminal"

Exemple du module bibliotheque :

- partie générique (→ bibliotheque.c) : nouvelleBibliotheque, ajouterOuvrage...
- partie spécifique (→ bibliotheque-term.c) : afficherListeOuvrages

Comm. Web

Lors du développement de la version graphique de l'application, les parties génériques des modules *menu*, *ouvrage* et *bibliotheque* seront ré-utilisées sans aucune modification.
Les parties spécifiques devront être ré-écrites ; la nouvelle implantation des fonctions correspondantes de l'interface pour cette version graphique s'appuient sur l'API de la bibliothèque graphique choisie.

18

7 Codage de la version "terminal" de l'application

Le codage de l'application consiste à traduire en langage C l'ensemble des informations produites lors de la conception détaillée :

- module *main* : main.c,
- module *menu* : menu.h, menu.c, menu-term.c
- module *ouvrage* : ouvrage.h, ouvrage.c, ouvrage-term.c
- module *bibliotheque* : bibliotheque.h, bibliotheque.c, bibliotheque-term.c
- module *util* : util.h, util.c.

Étudier la documentation :

- le fichier makefile
- le graphe d'appel de la fonction main
- la documentation doxygen complète

Tester le code à l'aide de l'archive en ligne :

- Archive codeTerminal.tgz

Comm. Web

Pour appréhender les sources, il est conseillé de commencer par l'étude des fichiers d'entête (les .h), puis d'utiliser la documentation doxygen pour naviguer dans les sources à partir de la fonction principale, en s'appuyant sur le graphe d'appel.