Mouddene Hamza
Cazes Noa

Rapport Projet 2019 – 2020

Un arbre généalogique

Objectif et contenu de ce rapport

Ce rapport a pour objectif de présenter les jalons de réflexion (les réflexions menées sur l'architecture et les types de données adaptés, les problèmes rencontrés, les solutions trouvées, les idées abandonnées) rencontrés au cours de la construction d'un arbre généalogique en langage Ada, mais aussi les perspectives envisagées pour améliorer ou aller plus loin dans le projet.

Des captures d'écran permettront d'illustrer le cheminement effectué au cours du projet.

Introduction

L'objectif de ce projet est de construire un arbre généalogique et un registre afin de proposer diverses fonctionnalités telles que de créer un arbre constitué d'un unique nœud, d'ajouter un parent, d'obtenir l'ensemble des ancêtres situés à une certaine génération à partir d'un nœud donné …

Il s'en suit la nécessité d'ajouter dans l'arbre les conjoints des individus pour avoir accès aux demi-frères et demi-sœurs.

Plan

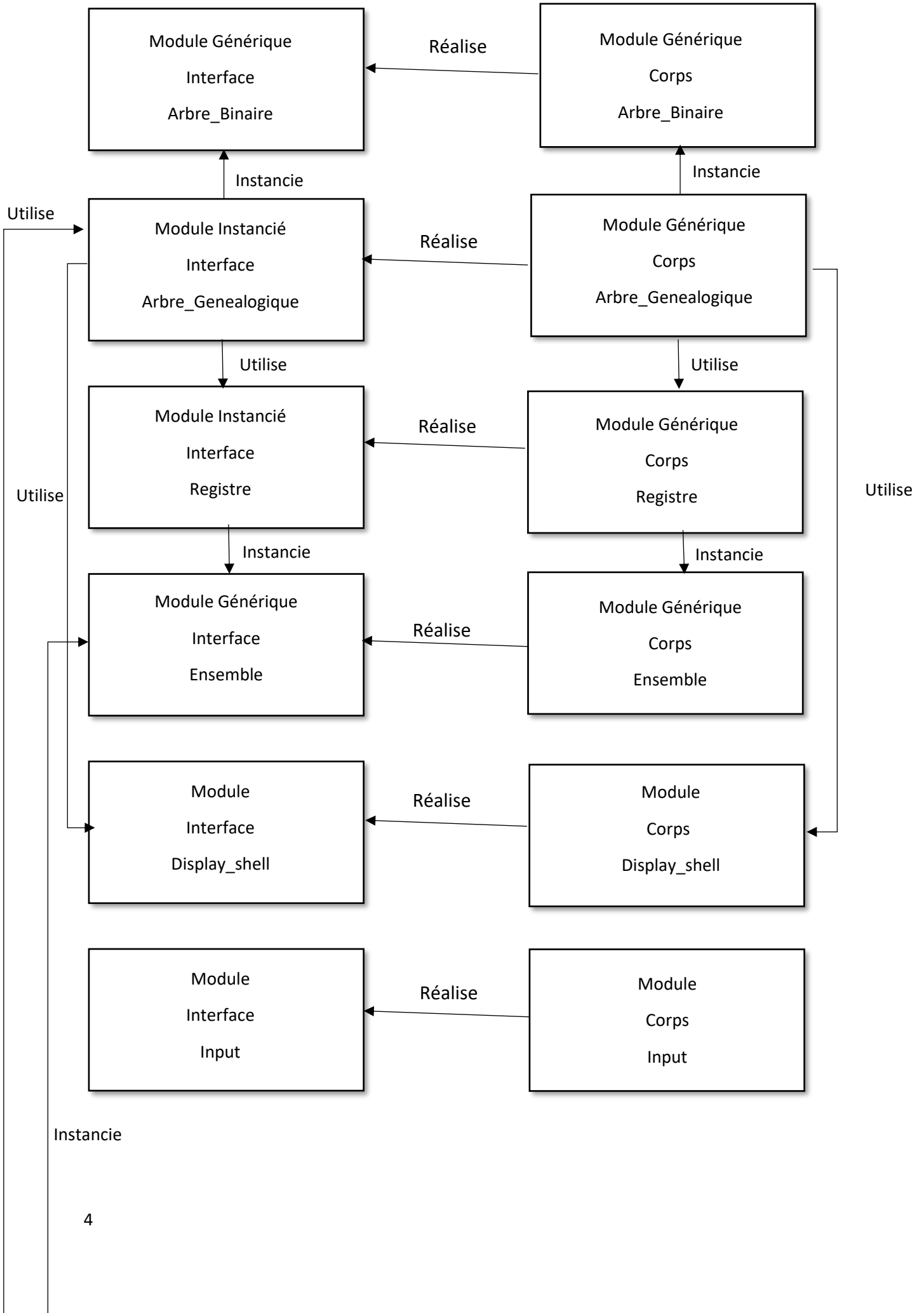
I. Présentation de l'architecture logicielle et des types de données choisis

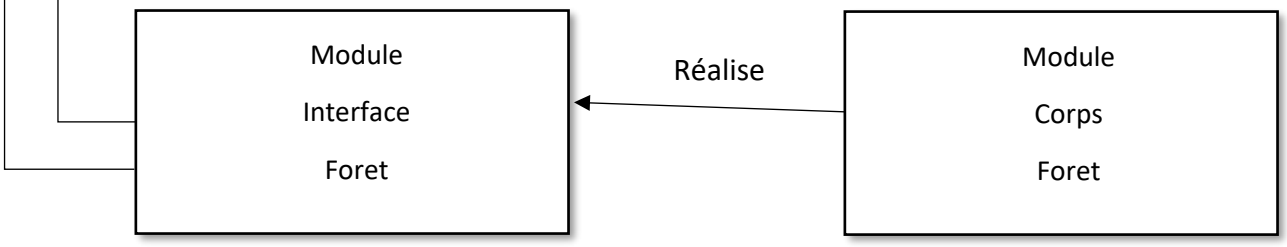
a) L'architecture logicielle

- **6 modules crées :**
  - **ensemble** : permet de créer et de manipuler un ensemble d'éléments (à définir)**.**
  - **arbre_binaire** : permet de créer la structure d'arbre binaire de recherche utilisée pour créer l'arbre généalogique.
  - **arbre_genealogique** : comprend toutes les fonctions te procédures nécessaires à la création et à la manipulation d'un arbre généalogique.
  - **registre** : permet la création et la manipulation d'un registre d'état civil.
  - **display_shell** : comprend toutes les procédures qui permettent d'afficher des données ou des éléments de présentation.
  - **input** : comprend toutes les procédures de saisie d'élément à partir du clavier.
  - **foret** : permet la construction de la forêt.

- **Modules utilisés :**
  - **alea :** permet de générer des identifiants (clés) aléatoirement.
  - **Ada.Text_I0**
  - **Ada.Strings.Unbounded**
  - **Ada.Text-I0.Unbounded_I0**
  - **Ada.Characters.Latin_1** : pour l'interface graphique.

- **Les liens entre les modules crées :**


Module Générique
Interface
Arbre_Binaire
Réalise
Module Générique
Corps
Arbre_Binaire
Instancie
Instancie
Utilise
Module Instancié
Interface
Arbre_Genealogique
Réalise
Module Générique
Corps
Arbre_Genealogique
Utilise
Utilise
Module Instancié
Interface
Registre
Réalise
Module Générique
Corps
Registre
Utilise
Utilise
Instancie
Instancie
Module Générique
Interface
Ensemble
Réalise
Module Générique
Corps
Ensemble
Module
Interface
Display_shell
Réalise
Module
Corps
Display_shell
Module
Interface
Input
Réalise
Module
Corps
Input
Instancie



b) Les types de données

- **Dans l'arbre généalogique :**
  Il a été construit sur un modèle *d'arbre binaire de recherche*.

```
private

    type T_Node; -- Un noeud d'un arbre

    type T_BT is access T_Node; -- Pointeur sur T_Node.

    type T_Node is record
        ID: T_ID;        -- ID du noeud.
        Left : T_BT;     -- Sous arbre gauche du noeud.
        Right : T_BT;    -- Sous arbre droit du noeud.
    end record;
```

- **Dans le registre :**
  Il a été construit sur un modèle de *liste chaînée simple* dans laquelle les éléments sont insérés par ordre croissant d'identifiants (clés).

```
private

    type T_Cellule;

    type T_Donnee is record
        Nom : Unbounded_String;         -- le nom
        Prenom : Unbounded_String;      -- le prénom
        Date_N : Integer;               -- la date de naissance, de la forme JJMMAAAA
        Lieu_N : Unbounded_String;      -- le lieu de naissance
        Date_D : Integer;               -- la date de décès, de la forme JJMMAAAA. Si Id est toujours vivant, inscrire 00000
        Lieu_D : Unbounded_String;      -- le lieu de décès. Si Id est toujours vivant, inscrire "Vivant".
        Sexe : Character;               -- le sexe, 'F' pour femme, 'H' pour Homme, 'U' pour non-genré
        Email : Unbounded_String;       -- l'email
        Tel : Integer;                  -- le numéro de téléphone portable, de la forme XXXXXXXXXX
        Concubain : Integer;            -- l'identifiant du concubain. Si Id n'a pas de conjoint, inscrire -1.
        Age : Integer;                  -- l'âge, sous la forme XXX
    end record;

    type T_Cellule is record
        Id : Integer;                -- l'identifiant. Enier naturel.
        Donnee : T_Donnee;           -- les données concernant l'identifiant
        Suivant : T_Registre;        -- pointeur vers l'identidiant suivant (forcément plus grand que Id)
    end record;

    type T_Registre is access T_Cellule;
```

II. Présentation des principaux algorithmes et des tests des programmes

a) Les principaux algorithmes

- **Les principales procédures :**

Nom : Inserer_R
Sémantique : Insérer une nouvelle donnée dans le registre pour un identifiant donné à sa place (en termes de croissance des identifiants).

Nom : Ancetres_N_Generation
Sémantique : Obtenir l'ensemble des ancêtres situés à une certaine génération d'un nœud donné.

Nom : Ancetres_Sur_N_Generation
Sémantique : Obtenir l'ensemble des ancêtres situés sur un nombre de génération donné, d'un nœud donné.

Nom : Add_To_Forest
Sémantique : Ajouter un arbre à la forêt.

Nom : Find_Tree
Sémantique : Trouver un identifiant dans la forêt.

Nom : Update_Forest
Sémantique : Mettre à jour la forêt.

Nom : Remove_Tree
Sémantique : Supprimer un arbre de la forêt.

Nom : Add_Cohabitant
Sémantique : Ajouter un concubain à un individu donné.

Nom : Add_Half_Brother
Sémantique : Ajouter un demi-frère ou une demi-sœur.

Nom : Get_Set_Half_Brother
Sémantique : Obtenir l'ensemble des demi-frères et demi-sœurs d'un individu donné.

Nom : Destruct_Forest
Sémantique : Détruire la forêt.

- **Les principales fonctions :**

Nom : Nombre_Ancetres

Sémantique : Obtenir le nombre d'ancêtres connus (lui compris) d'un individu donné

Nom : Ancetres_Homonymes
Sémantique : Vérifier que deux individus n et m ont un ou plusieurs ancêtres homonymes.

b) Les tests de programmes

- Pour tester les programmes du registre, la démarche adoptée a été de créer un registre, soit avec un élément, soit avec trois éléments, avec des données déjà saisies. Ensuite des sous-programmes Tester_Nom_Du_Sous_Programme ont été réalisés en utilisant ces registres de tests.
  Un message de réussite s'affiche à chaque test réussi.
  De même pour les tests des sous-programmes du module ensemble.
- Pour les tests des sous-programmes du module arbre_genealogique, à chaque procédure de test, un arbre minimal est créé auquel on ajoute des éléments en fonction des sous-programmes à tester.
  De même pour les tests des sous-programmes du module arbre_binaire.

VI. Les difficultés rencontrées et les solutions apportées

- Difficulté rencontrée lors du choix de la structure du registre (hésitation entre liste chaînée simple et arbre binaire de recherche)
  - ➔ Nous avons pris la décision de le construire selon une structure de liste chaînée simple, avec éléments rangés par ordre croissant des identifiants, afin de pouvoir accéder efficacement aux éléments du registre.
- Difficulté rencontrée avec les types Strings et Unbounded.Strings
  - ➔ Utilisation du type Unbounded.Strings et étude des modules Ada.Strings.Unbounded et Ada.Text_I0.Unbounded_I0 pour avoir accès aux procédures nous permettant de résoudre les problèmes rencontrés
- Beaucoup de répétitions au niveau des saisies et de l'affichage des données
  - ➔ Création des modules input et display_shell

VII. L'état d'avancement et les perspectives d'amélioration

a) L'état d'avancement

Toute l'implantation des sous-programmes de tous les modules sont terminés, et le programme principal fonctionne, ainsi que les programmes de test.

Le module forêt a été spécifié et implanté.

Le menu et les messages d'erreur sont affichés en couleur afin de créer une interface plus agréable pour l'utilisateur.

b) Les perspectives d'amélioration

On pourrait ajouter une base de données qui contient les arbres, ainsi que faire une interface graphique pour faciliter l'utilisation du programme.

VIII. Les bilans personnels

a) L'organisation du groupe

On se répartissait les tâches au fur et à mesure que les jalons passaient, tout en travaillant ensemble.

Au début du projet, on a donc discuté de la structure globale du projet : Quelle structure choisir pour l'arbre généalogique ? Pour le registre ?

Hamza avait en charge les modules arbre_binaire et arbre_genealogique et les tests associés. J'avais en charge la partie registre, les tests associés et l'architecture logicielle.

Ensuite, Hamza a eu l'idée de réunir tous les sous-programmes permettant de saisir les données du clavier dans un nouveau module, et de faire de même pour les sous-programmes qui permettent l'affichage des données, et a spécifié et implanté de nouveaux sous-programmes dans ces mêmes modules.

Le programme principal a été réalisé ensemble.

La spécification de certains modules a été refaite à deux.

Je réalise le rapport et le guide utilisateur, tandis qu'Hamza a réalisé la forêt.

b) Bilan de Hamza Mouddene

Dans le cadre de l'UE programmation impérative, un projet en ADA a été réalisé, et concernait des types abstraits de données allant de liste chainée, arbre binaire jusqu'à une collection d'arbres que l'on l'appellera dans le projet une forêt.
Lors de la réalisation de ce projet, on a appris beaucoup de choses, comme notamment la familiarisation avec le langage ADA et ses spécificités, en traitant des concepts plus au moins complexes comme l'encapsulation, l'instanciation et la généricité, ainsi que l'optimisation algorithmique.

J'ai passé 10h sur la conception et environ 30h sur la programmation.

Tout le long de projet, des problèmes croissants en difficultés ont été rencontrés, ce qui nous a forgé et nous appris beaucoup de choses comme : la rigueur, avoir un code générique qui s'adapte à toutes les situations, par exemple.

Ce projet nous a permis d'acquérir un niveau plus élevé en programmation, et nous a insufflé la volonté de toujours découvrir de nouvelles choses.

c) Bilan de Noa Cazes

En tant que premier gros projet d'informatique, ce projet a été très formateur dans le sens où il a permis de revoir les listes chaînées, les modules, la généricité, les arbres binaires, l'allocation dynamique de mémoire. Et c'est aussi un projet que j'ai beaucoup apprécié faire car il nous a permis de construire un réel projet informatique, et de nous apprendre à prendre des initiatives.

Le temps passé à la conception est de 4h, celui passé à l'implantation d'environ 15/18 h, celui passé sur le rapport est de 4h, celui passé sur le manuel utilisateur est de 3h.

De ce projet, on a appris à créer une architecture logicielle, à subdiviser les étapes en sous-programmes, et à manier plus facilement la généricité.