Editeur d'automates: "AutomatesLab"

Dossier de conception

Référence	:	CG-8-0B
Fournisseur		
Date	:	11/01/2023
Version/Édition	:	0B
État	:	Préliminaire
Type de diffusion	:	Diffusion propriétaire
Autre référence	:	

Historique des révisions

Date	Description et justification de la modification	Auteur	Pages / Chapitre	Edition / Révision
11/01/2023	Création		Toutes	0A
20/01/2023	Rédaction du 5.1 et détails des classes		Toutes	0B

Table des matières

1. Introduction

1.1 Objet du document

La conception générale d'un logiciel est une démarche rationnelle, qui consiste à réduire la complexité initiale en la décomposant en constituants de niveau inférieur.

La conception permet de préciser *comment* vont être réalisées les spécifications.

La conception préliminaire permet d'élaborer une solution technique répondant aux spécifications. Elle précise l'architecture de cette solution (interfaces internes, constituants principaux, ...), ainsi que son comportement dynamique (logique d'enchaînements, diagrammes d'état, parallélisme, synchronisation, ...).

Il faut présenter succinctement la structure du document :

- description et analyse des ressources matérielles et logicielles,
- diagramme des catégories ou des classes principales,
- structure statique (interfaces externes, interfaces internes, ...),
- structure dynamique (synchronisation entre tâches, séquencement des traitements, passages entre les différents états ou mode de traitement, ...),
- implémentation sur le matériel cible (allocation des ressources CPU mémoire bande passante, adresse des périphériques, couche basse de protocoles spécifiques, ...).

La conception préliminaire doit permettre de réaliser le dossier d'intégration qui décrit les étapes permettant de vérifier que tous les composants identifiés au niveau de la conception préliminaire s'intègrent bien ensemble.

La revue de conception vérifie que tout le contenu des spécifications a bien été pris en compte par la conception préliminaire.

1.2 Outils utilisés

Les documents de conception sont rédigés avec Google Doc.

Les schémas UML sont créés avec le logiciel StarUML. Les schémas présentés respectent les conventions UML.

L'application sera développée en Java en utilisant l'IDE IntelliJ Idea, ainsi que Maven pour la compilation. Les interfaces graphiques seront implémentées à l'aide de JavaFX.

GitHub sera utilisé pour le versioning du projet.

2. Terminologie

2.1. Abréviations

UML	**U**nified **M**odeling **L**anguage
JRE	**J**ava **R**untime **E**nvironment
MVC	**M**odel **V**iew **C**ontroller

2.2. Définitions des termes employés

attribut	Un attribut est une information caractéristique mémorisée par un objet.
cas d'utilisation	Cas d'utilisation du système, par extension il représente également la technique de modélisation mise en œuvre dans UML (use case).
catégorie	Une catégorie consiste en un regroupement logique de classes à forte cohérence interne et faible couplage externe, associée au concept UML de package. Ce concept permet une présentation plus synthétique du diagramme des classes d'un système réel.
classe	Une classe définit un ensemble d'objets similaires potentiels. Elle fournit le modèle de la structure et les possibilités de chaque objet.
objet	Un objet est une instance d'une classe, c'est une entité informatique unique possédant ses propres attributs et opérations.
opération ou méthode	Une opération est un traitement spécifique qu'un objet est chargé de fournir.
tâche	Une tâche représente un élément manipulé par le système et est ordonnançable de manière individuelle. Cela peut représenter un processus d'un système Unix, une tâche d'un moniteur temps-réel, un thread d'une application.
threads	Codes exécutés de manière concurrente par le système d'exploitation mais partageant le même espace mémoire.
MVC	Architecture logicielle permettant une séparation entre l'interface utilisateur (View) et le cœur algorithmique de l'application (Model). Le Controller fait le lien entre les deux.

3. Description et analyse de l'environnement

3.1. Description des ressources matérielles

AutomatesLab est conçu pour un ordinateur ayant les spécifications minimales suivantes :

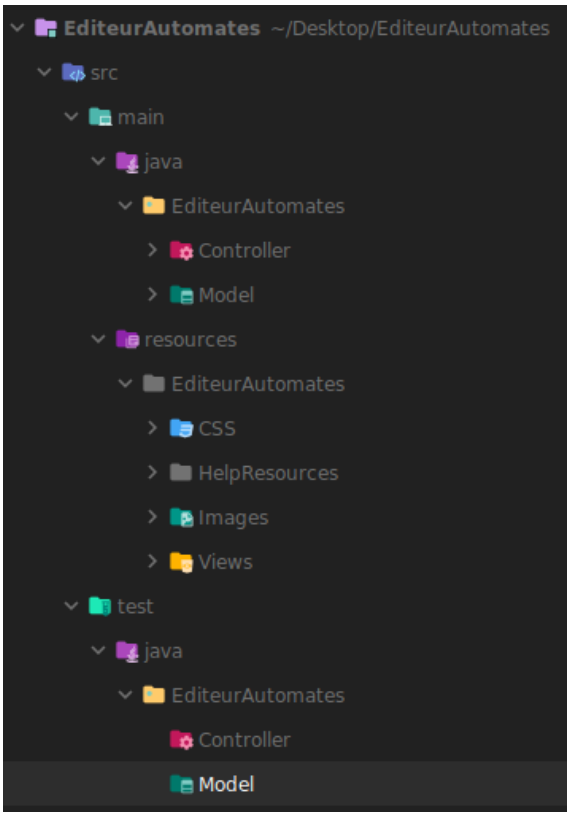
- CPU 4 coeurs / 8 threads @ 2.00 Ghz
- 8 Go de mémoire
- 30 Mo d'espace disque

3.2. Description des ressources logicielles

Pour fonctionner, AutomatesLab requiert uniquement qu'il y ait le JRE version 19 ou supérieure d'installée. La librairie graphique utilisée, JavaFX, est empaquetée directement dans l'exécutable. Ainsi, elle n'a pas besoin d'être spécifiquement installée sur la machine de l'utilisateur.

3.3. Organisation de l'espace de travail

Le projet est organisé en suivant une architecture MVC. Cela va nous permettre d'avoir une séparation entre l'interface utilisateur et la partie logique de l'application. JavaFX utilisant des fichiers XML pour l'affichage, ceux-ci sont dans le dossier resources plutôt que src directement.



t.1 : Architecture des fichiers du projet

4. Structure statique

4.1. Décomposition générale

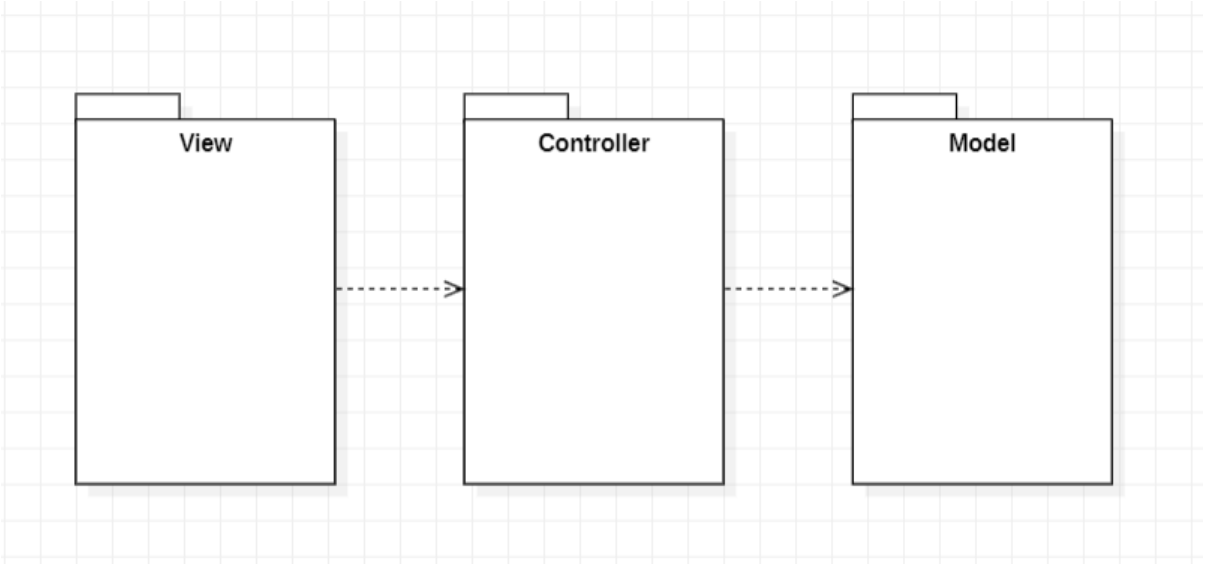


Fig. 2 : Diagramme de package

4.2. Allocations fonctionnelles et spécifications

Les fichiers XML du package View sont les affichages graphiques de l'application. Ils permettent ainsi à l'utilisateur d'interagir de manière graphique avec cette dernière.

Les classes présentes dans le package Model permettent de représenter logiquement les éléments présents dans l'application. Ainsi, la classe Automate permet de représenter un automate et ses transitions de manière générale. La classe Etat permet de représenter un État de cet automate. Enfin, il y a une classe utilitaire XMLParser s'occupant de la gestion des ressources XML, que ce soit charger un fichier ou vérifier qu'un XML est valide.

Les classes présentes dans le package Controller permettent de faire le lien entre l'affichage et le cœur logique de l'application. MainWindowController gère la fenêtre principale contenant l'ensemble de l'application : chaque fonction du modèle est associée à la vue correspondante. GraphicController gère la partie graphique de l'application (partie visuelle de la fenêtre de création d'automate). XMLController gère la partie codée en xml (partie de modélisation de l'automate rédigée au format xml). L'interface IAutomateView sert à définir le comportement qu'un contrôleur d'une vue doit avoir. Tout contrôleur doit impérativement implémenter les deux fonctions "updateModel" et "pullModel" pour s'assurer du bon fonctionnement du changement de vue.

4.3. Analyse statique

4.3.1. Package Model

Le package Model contient les classes représentant les objets nécessaires au bon fonctionnement du code. Ainsi, c'est le cœur logique de l'application. Ce sont les classes contenues dans le package Model qui s'occuperont d'interagir entre elles afin de modéliser un automate et de pouvoir interagir avec celui-ci.

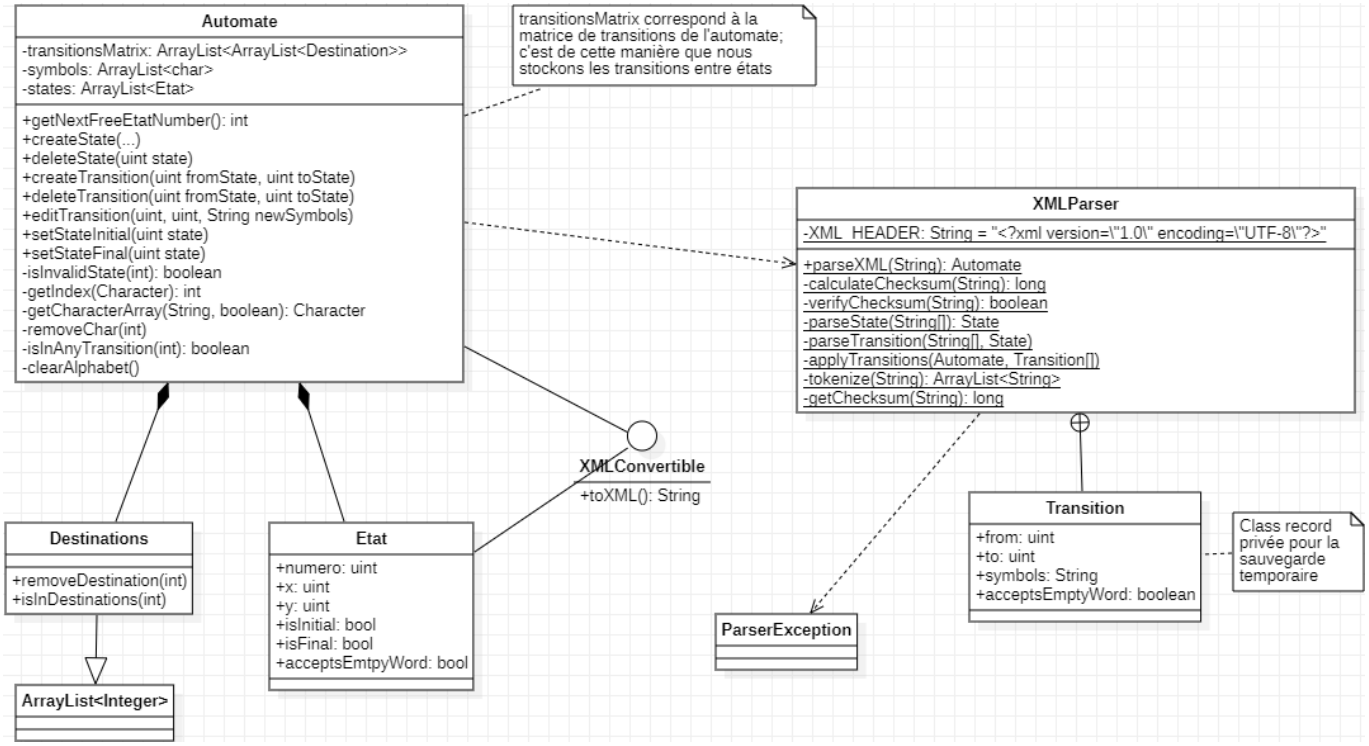


Fig. 3 : Diagramme de classes du package Model

4.3.1.1. Classe Automate

4.3.1.1.1. Définition

La classe Automate permet de représenter un automate et ses transitions.

4.3.1.1.2. Attributs

transitionMatrix :

- ArrayList<ArrayList<Destination>>
- Les transitions sont représentées par une matrice d'états cibles avec l'état d'origine sur l'axe i et le caractère nécessaire pour la transition sur l'état sur l'axe j. Cette matrice de transitions est une matrice à 3 dimensions. En effet, si il y a *n* états et *m* symboles dans l'alphabet, on aurait une matrice de dimensions *n*m* où chaque case de la matrice est une liste contenant les états atteignables à partir de cet état d'origine et avec ce caractère.

symbols :

- ArrayList<char>
- Alphabet utilisé par l'automate.

states :

- ArrayList<Etat>
- Etats composant l'automate.

Exemple :

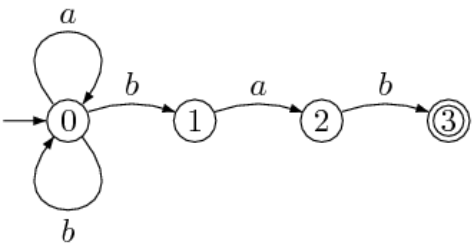


Fig. 4 : Représentation graphique d'un automate

symbols :

a	b

states :

0	1	2	3

[0]	[0,1]
[2]	[]
[]	[3]
[]	[]

4.3.1.1.3. Opérations

Fonction createState :

- Crée un état.
- Prend en paramètre un état.
- Ne renvoie rien.
- Engendre une vérification pour déterminer quel nombre attribuer au nouvel état créé, en fonction de sa position dans l'ordre de création.

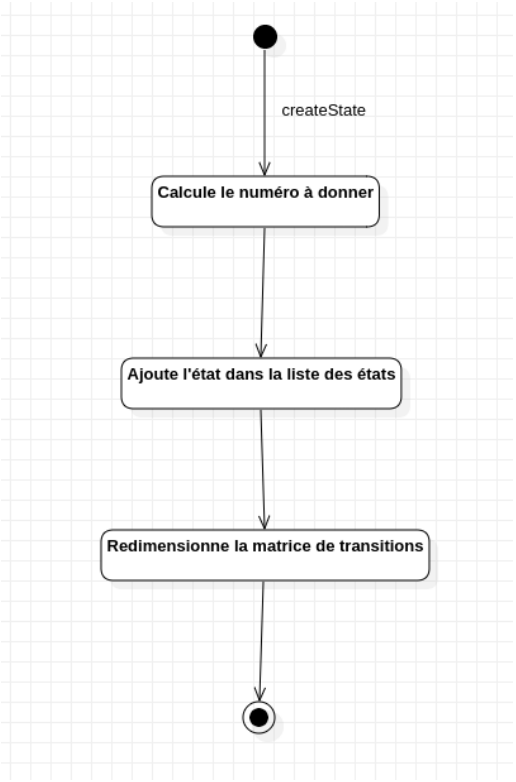


Fig. 5 : Diagramme d'activité de createState()

Fonction deleteState :

- Supprime l'état sélectionné.
- Prend en paramètre l'identifiant d'un état.
- Ne renvoie rien.
- Supprime les transitions associées à l'état, redimensionne la matrice de transitions et l'ArrayList des états.

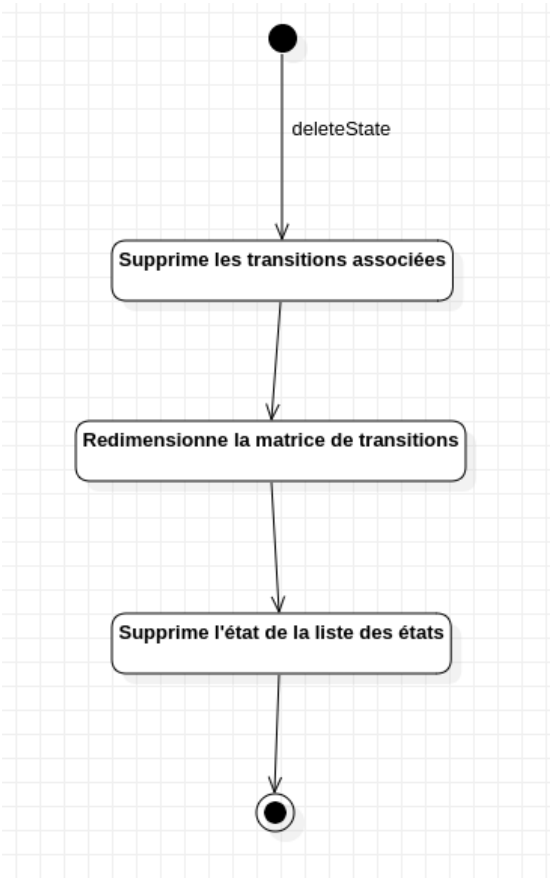


Fig. 6 : Diagramme d'activité de deleteState()

Fonction createTransition :

- Crée une transition entre les deux états.
- Prend en paramètre le numéro de l'état de départ, le numéro de l'état d'arrivée et une chaîne de caractères pour les caractères de la transition.
- Ne renvoie rien.
- En cas de nouveaux caractères, ajoute ces derniers dans l'alphabet et redimensionne la matrice de transitions. Dans tous les cas, ajoute les références vers le l'état d'arrivée dans le ou les bonnes cases de la matrice de transitions.

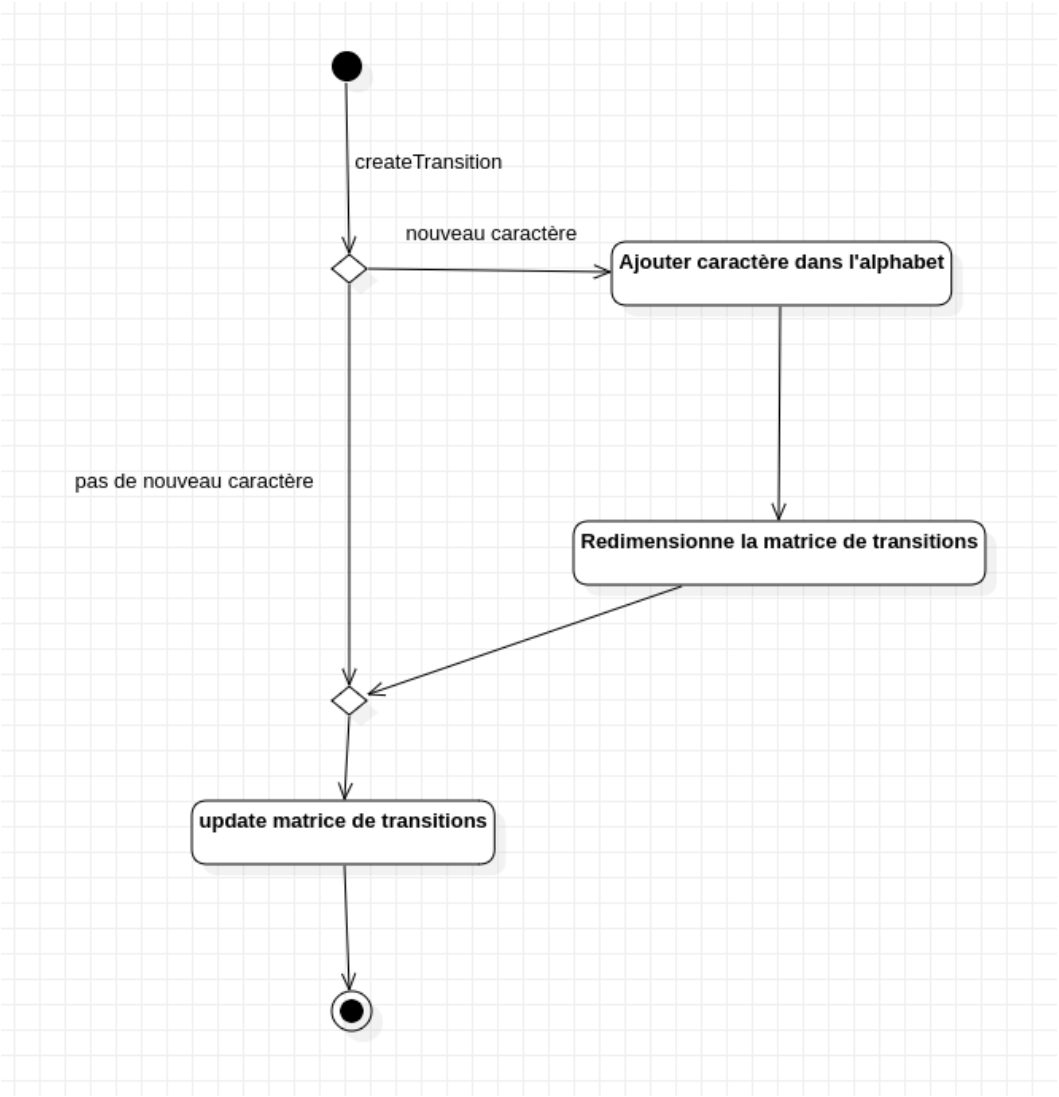


Fig. 7 : Diagramme d'activité de createTransition()

Fonction deleteTransition :

- Supprime une transition entre deux états.
- Prend en paramètre l'état de départ ainsi que l'état d'arrivée.
- Ne renvoie rien.
- Supprime la ou les références de l'état d'arrivée, vérifie si un des caractères de la transition n'est plus utilisé, le cas échéant il est supprimé de l'alphabet.

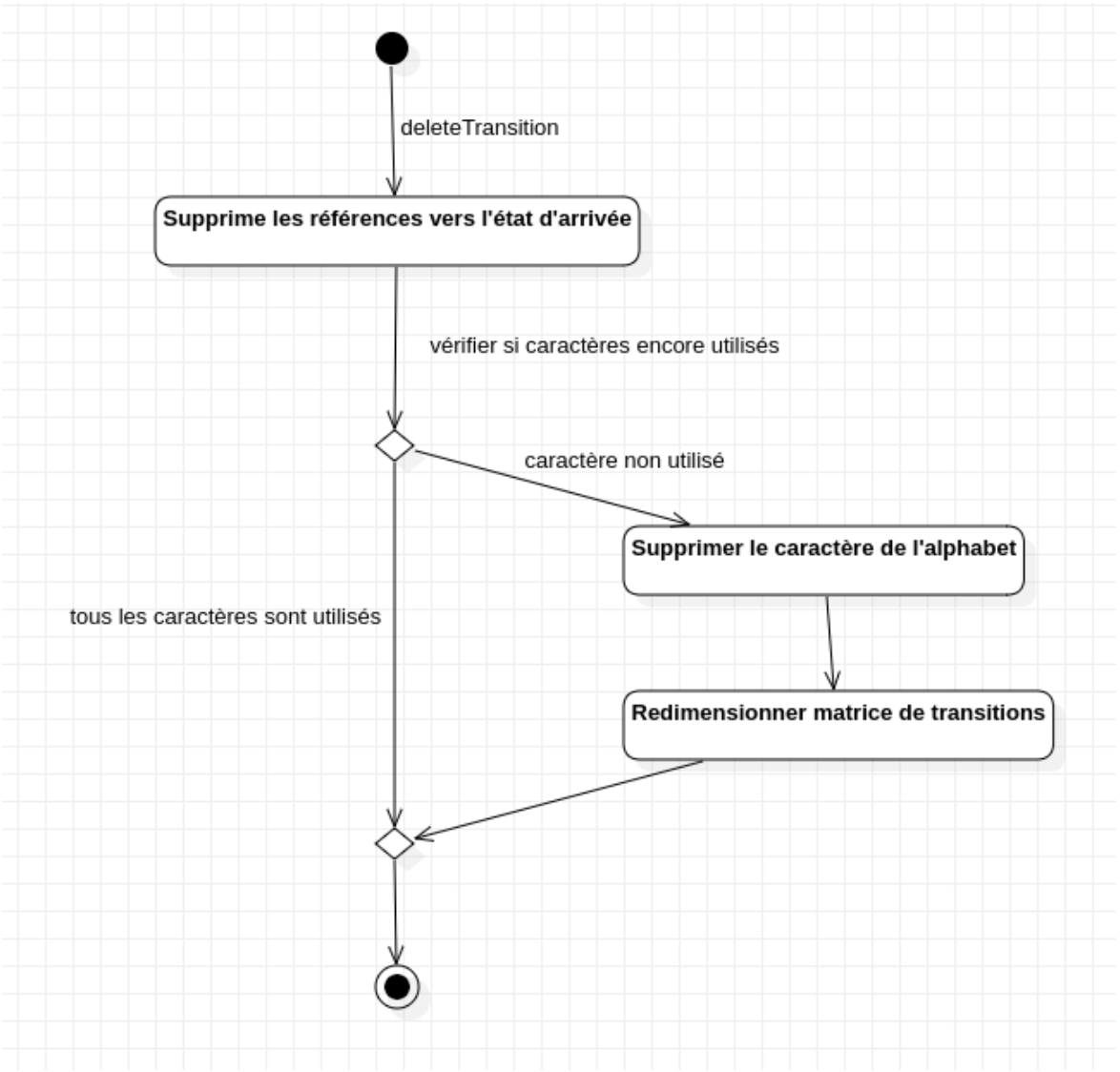


Fig. 8 : Diagramme d'activité de deleteTransition()

Fonction editTransition :

- Permet de modifier le ou les caractères associés à une transition.
- Prend en paramètre le numéro de l'état de départ, le numéro de l'état d'arrivée, et une chaîne de caractères correspondant aux nouveaux caractères de la transition.
- Ne renvoie rien
- Dans le cas où un nouveau caractère est ajouté, l'ajoute dans l'alphabet et redimensionne la matrice de transition. En cas de suppression d'un caractère, vérifie s'il n'est plus utilisé dans l'automate et, le cas échéant, le supprime de l'alphabet et redimensionne la matrice.

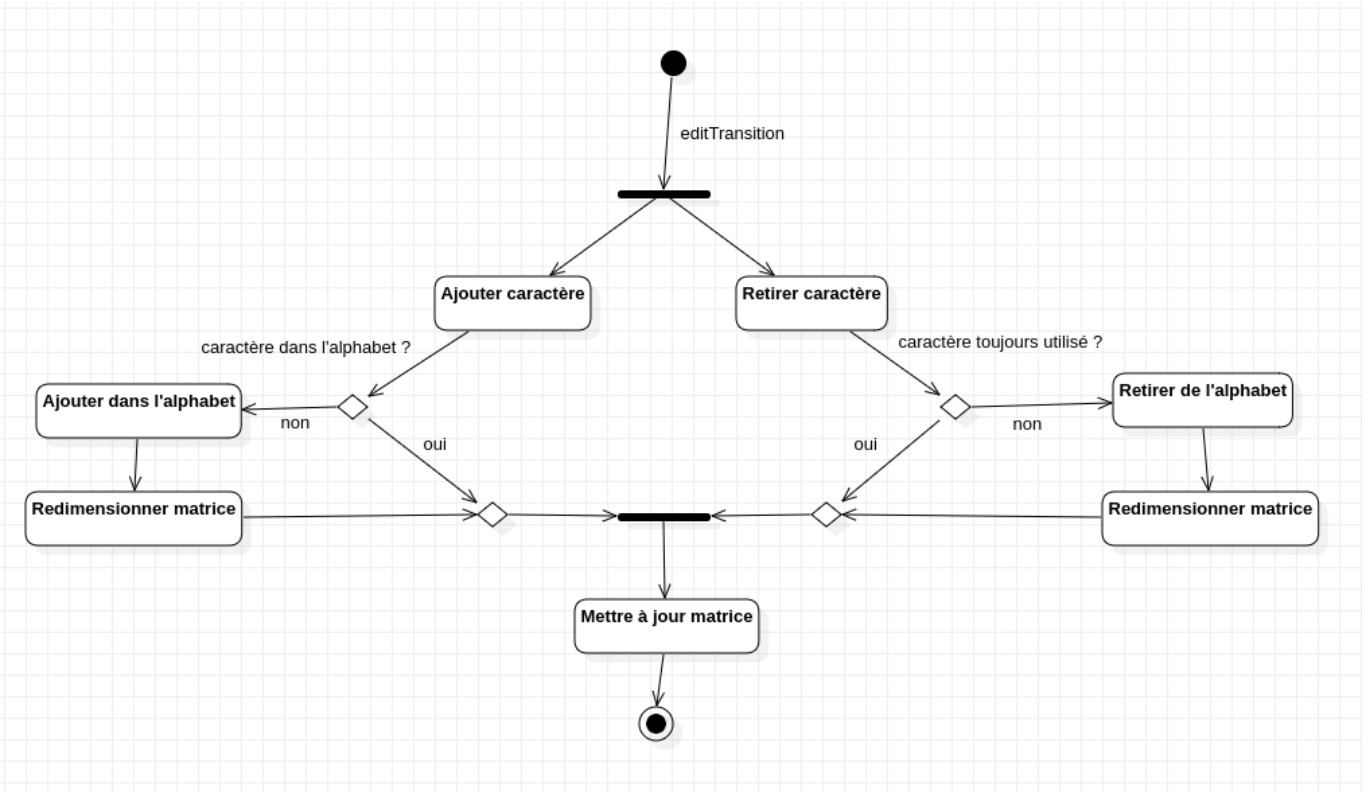


Fig. 9 : Diagramme d'activité de editTransition()

Fonction setStateInitial :

- Permet de définir si un état est initial.
- Prend en paramètre le numéro de l'état.
- Ne renvoie rien.

Fonction setStateFinal :

- Permet de définir si l'état est final.
- Prend en paramètre le numéro de l'état.
- Ne renvoie rien.

4.3.1.2. Classe Destination

4.3.1.2.1. Définition

La classe Destination est une classe privée de la classe Automate. Elle sert à faciliter la gestion de la matrice de transition. Ainsi, elle extends de ArrayList<Integer>. Ce qui fait que la classe Destination se comporte comme une liste d'entiers. Ces entiers représentent les états destinations pour toutes les transitions présentes dans l'automate.

4.3.1.2.2. Opérations

Fonction add :

- Ajoute un entier à la liste s'il n'est pas déjà présent
- Prend en paramètre l'entier à ajouter
- Renvoie un boolean, true si l'ajout à fonctionné, false sinon
- Override la fonction add de la classe ArrayList

Fonction removeDestination :

- Retire un entier de la liste
- Prend en paramètre l'entier à enlever
- Ne renvoie rien

Fonction isInDestination :

- Teste si un entier est présent dans la liste
- Prend en paramètre l'entier à tester
- Renvoie True s'il est présent, false sinon

4.3.1.3. Classe Etat

4.3.1.3.1. Définition

La classe Etat permet de représenter un état de l'automate et ses attributs. Ces derniers sont publics car la classe sert uniquement à agréger les attributs d'un État. Il n'y a pas de méthodes associées.

4.3.1.3.2. Attributs

Numéro :

- Entier
- ID de l'état, sera affiché sur l'interface et sert à référencer l'état.

X / Y :

- Entier
- Coordonnées en pixels de l'état, sert à le placer sur l'interface au moment de l'affichage.

isFinal :

- Booléen
- Indique si l'état est final.

isInitial :

- Booléen
- Indique si l'état est initial.

acceptsEmptyWord :

- Booléen
- Indique si l'état accepte le mot vide.

4.3.1.4. Classe XMLParser

4.3.1.4.1. Définition

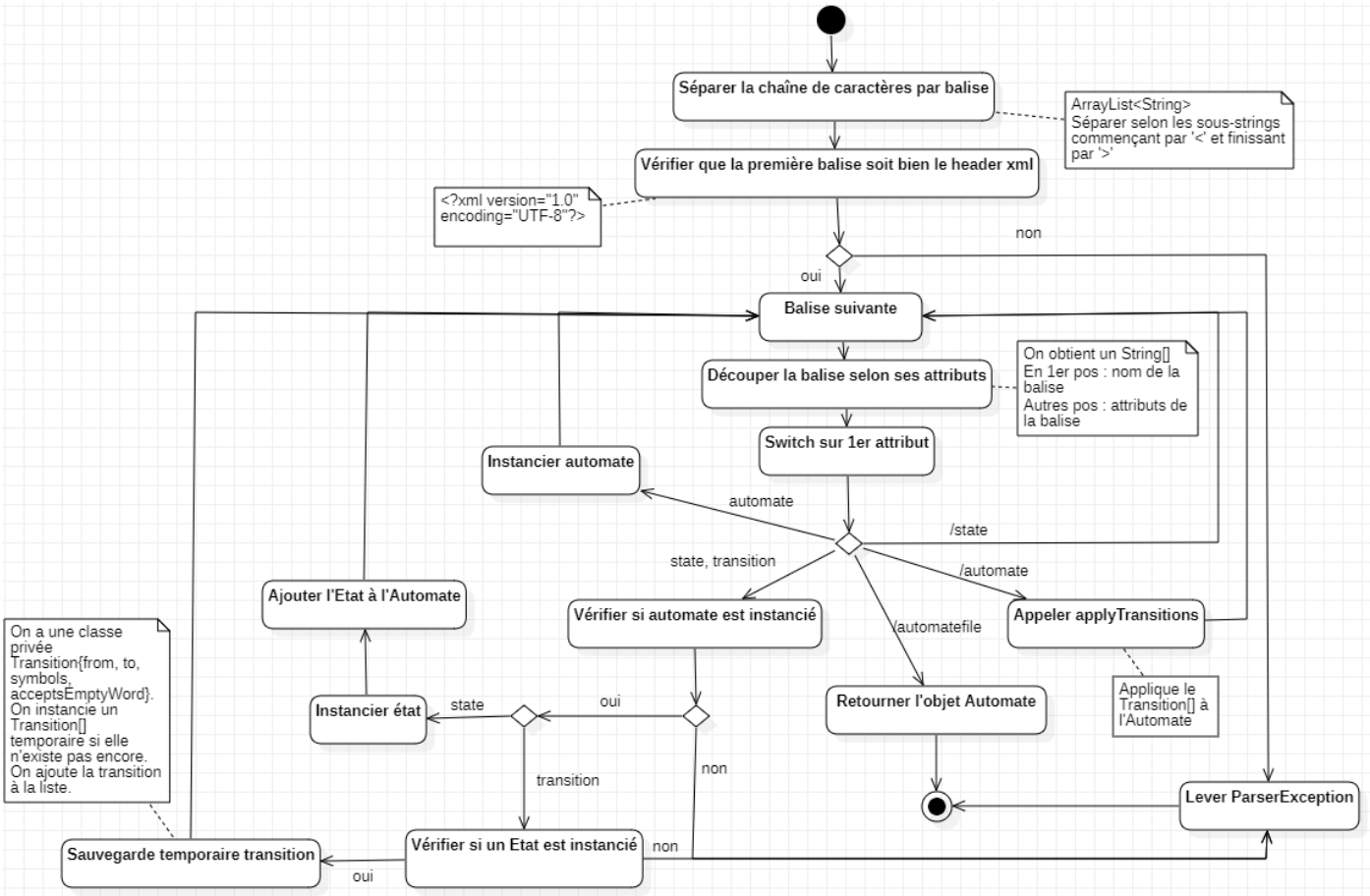
La classe XMLParser est une classe utilitaire mettant à disposition des fonctions statiques utiles permettant de parser un fichier ou une chaîne de caractère formatée dans notre standard XML et de convertir sa représentation vers nos classes.

4.3.1.4.2. Opérations

parseXML :

- Parse une chaîne de caractère XML
- Prend en paramètre la chaîne de caractères à parser
- Renvoie le modèle associé à l'automate (l'objet Automate correspondant). Émet une exception si la chaîne ne correspond pas à un Automate, ou est un XML invalide.

L'implémentation suit le schéma suivant:



calculateChecksum :

- Calcule la checksum associée à l'automate
- Prend en paramètre un automate
- Renvoie la checksum sous forme de long
- Va servir pour vérifier que le fichier n'a pas été altéré au moment de le charger

tokenize :

- Découpe le fichier XML par balise
- Prend en paramètre la chaîne de caractères XML
- Renvoie un tableau de chaînes de caractères

parseState :

- Instancie un état à partir des informations du XML
- Renvoie un objet Etat
- Prend en paramètre un String[] correspondant aux paramètres de l'état

parseTransition :

- Instancie une transition à partir des informations du XML
- Renvoie un objet Transition
- Prend en paramètres un String[] correspondant aux paramètres et un entier correspondant à l'état d'origine

applyTransition :

- Applique les transition sauvegardées temporairement (cf parseXML)
- Ne prend pas de paramètre
- Ne renvoie rien
- Est appelée quand la balise </Automate> est lue

4.3.1.5. Interface XMLConvertible

4.3.1.5.1. Définition

L'interface XMLConvertible met à disposition la méthode *toXML* à redéfinir. Cette fonction décrit la manière de convertir l'objet implémentant l'interface en XML.

4.3.2. Package Controller

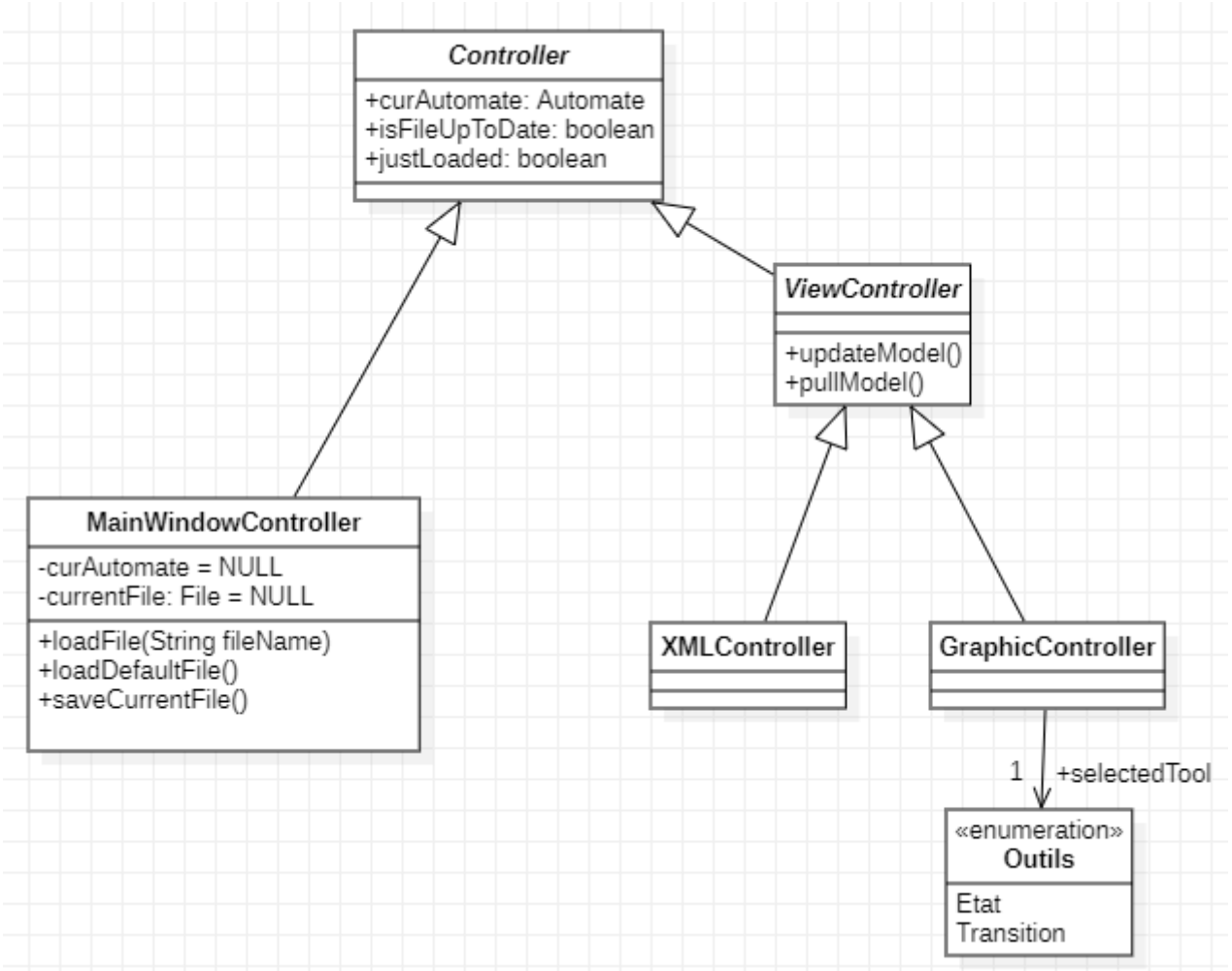


Fig. 10 : Diagramme de classes du package Controller

4.3.2.1. Classe abstraite Controller

4.3.2.1.1. Définition

La classe abstraite Controller doit être étendue par tous les contrôleurs concrets. Elle permet de partager une instance de l'automate courant.

4.3.2.1.2. Attribut

curAutomate :

- Référence statique du modèle de l'automate courant

4.3.2.2. Classe MainWindowController

4.3.2.2.1. Définition

La classe MainWindowController sert à faire la liaison entre l'interface de la fenêtre principale et les différentes classes du modèle.

4.3.2.2.2. Attributs

currentFile :

- Référence du fichier courant
- Sert pour la sauvegarde du fichier, si l'attribut est NULL alors on est sur un nouveau fichier.

4.3.2.2.3. Opérations

loadFile :

- Charge le fichier passé en paramètre
- Prend en paramètre le chemin vers le fichier à charger
- Ne renvoie rien
- Change la référence du fichier et de l'automate courant. Appelle la fonction parseXML de la classe XMLParser. Doit gérer les possibles exceptions levées par parseXML.

saveCurrentFile :

- Sauvegarde le fichier courant sur le disque
- Ne prend pas de paramètre
- Ne renvoie rien

loadDefaultFile :

- Charge le fichier par défaut en cas de création de nouvel automate
- Ne prend pas de paramètre
- Ne renvoie rien

- Appelle loadFile avec le chemin du fichier par défaut mais ne change pas la référence vers le fichier courant.

4.3.2.3. Classe abstraite ViewController

4.3.2.3.1. Définition

La classe abstraite ViewController doit être étendue par toutes les classes controller d'une vue. En effet, elle définit les deux méthodes essentielles au bon fonctionnement des transitions entre les types de vue.

4.3.2.3.2. Opération

updateModel :
- Met à jour le modèle de l'automate à partir des changements effectués sur la vue
- Ne prend pas de paramètre
- Ne renvoie rien
- Il est appelé juste avant de sortir de la vue en cas de changement ou de fermeture de l'application.

pullModel :
- Affiche la vue en fonction du modèle dans le MainWindowController
- Ne prend pas de paramètre
- Ne renvoie rien
- Est appelé en premier lieu au changement vers la vue en question. La fonction s'occupe de convertir le modèle de l'automate dans le format requis par la vue demandée.

4.3.2.4. Classe GraphicController

4.3.2.4.1. Définition

La classe GraphicController permet de faire la liaison entre la vue graphique dans l'interface et les classes du modèle. Étant le contrôleur d'une vue, elle extends ViewController.

4.3.2.4.2. Attribut

selectedTool :
- Outil courant sélectionné

4.3.2.4.3. Opérations

pullModel :
- Convertit l'automate courant en FXML affichable par JavaFX. La fonction accède au modèle de l'automate courant et place tous les états dans la zone d'affichage à partir de leurs classes contenues dans le modèle. Ensuite, elle lit la matrice de transitions pour dessiner les flèches correspondant aux transitions.

createState() :

- Permet de créer un nouvel état
- Ne prend pas de paramètre
- Ne renvoie rien
- Appelle la fonction createState de l'automate courant. Cela va permettre d'ajouter l'état au niveau du model. Ensuite, appelle la fonction pullModel pour mettre à jour l'affichage à partir du modèle.

4.3.2.5. Enumération Outils

4.3.2.5.1. Définition

L'énumération Outils sert à définir les outils mis à dispositions à l'utilisateur sur l'affichage graphique. Elle contient deux éléments :

- Etat
- Transition

4.3.3. Package View

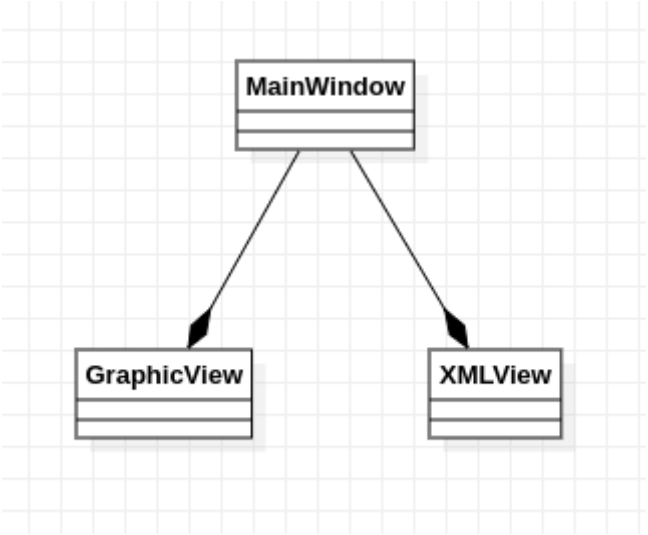


Fig. 11 : Diagramme de classes du package View

Le package View contient uniquement les fichiers permettant d'afficher l'interface. Ainsi, ils ne contiennent ni méthodes, ni attributs. Ils sont écrits en FXML comme requis par JavaFX.

4.3.3.1. Fichier MainWindow

4.3.3.1.1. Définition

Le fichier MainWindow correspond à la fenêtre principale de l'application. Il va contenir la barre de menu ainsi que le widget permettant d'avoir des onglets pour changer de vue. Ces dernières sont définies dans des fichiers FXML à part et possèdent leur propre contrôleur. Le code FXML est ensuite injecté dans un panneau dans le fichier MainWindow.

4.3.3.2. Fichier GraphicView

4.3.3.2.1. Définition

Le fichier GraphicView s'occupe d'afficher les éléments nécessaires pour la vue graphique des automates. Ainsi, il va contenir une barre latérale contenant la boîte à outils et une zone d'affichage graphique où l'automate sera rendu.

4.3.3.3. Fichier XMLView

4.3.3.3.1. Définition

Le fichier XMLView affiche les éléments nécessaires pour la vue XML des automates. De ce fait, il est très simple et comporte uniquement une zone de texte éditable. Cette zone affiche le code XML de l'automate, formaté dans notre standard qui est défini dans Représentation XML.

5. Aspects dynamiques

5.1. Échanges entre éléments

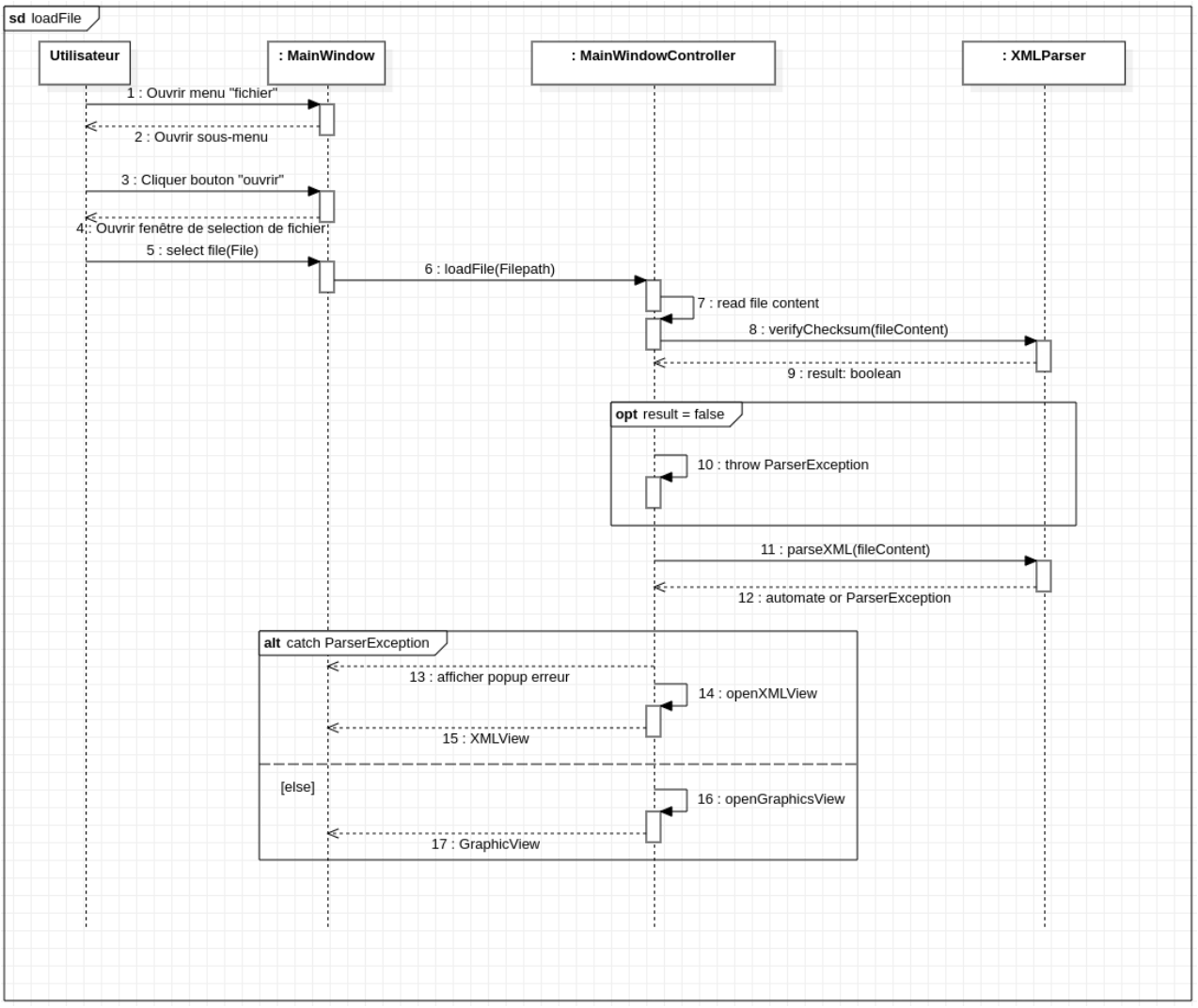


Fig 12: Diagramme de séquence de PRJ-002

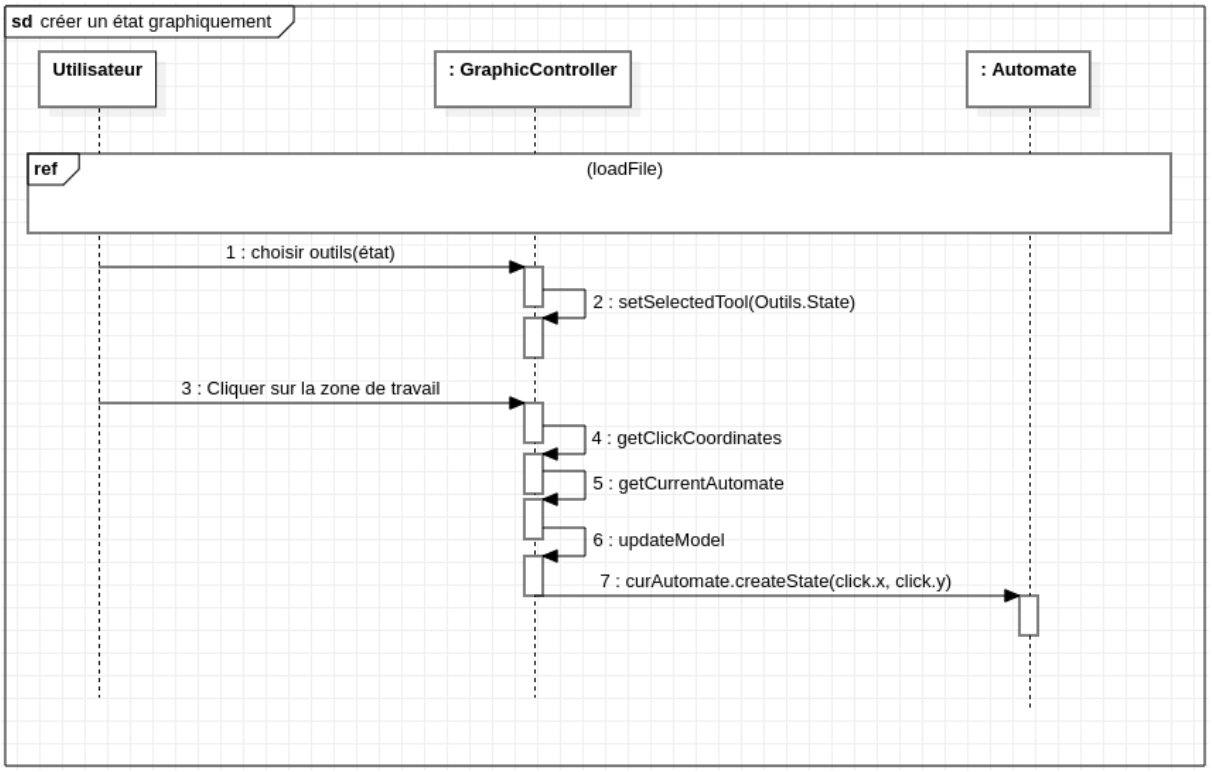


Fig 13: Diagramme de séquence de PRJ-004-1

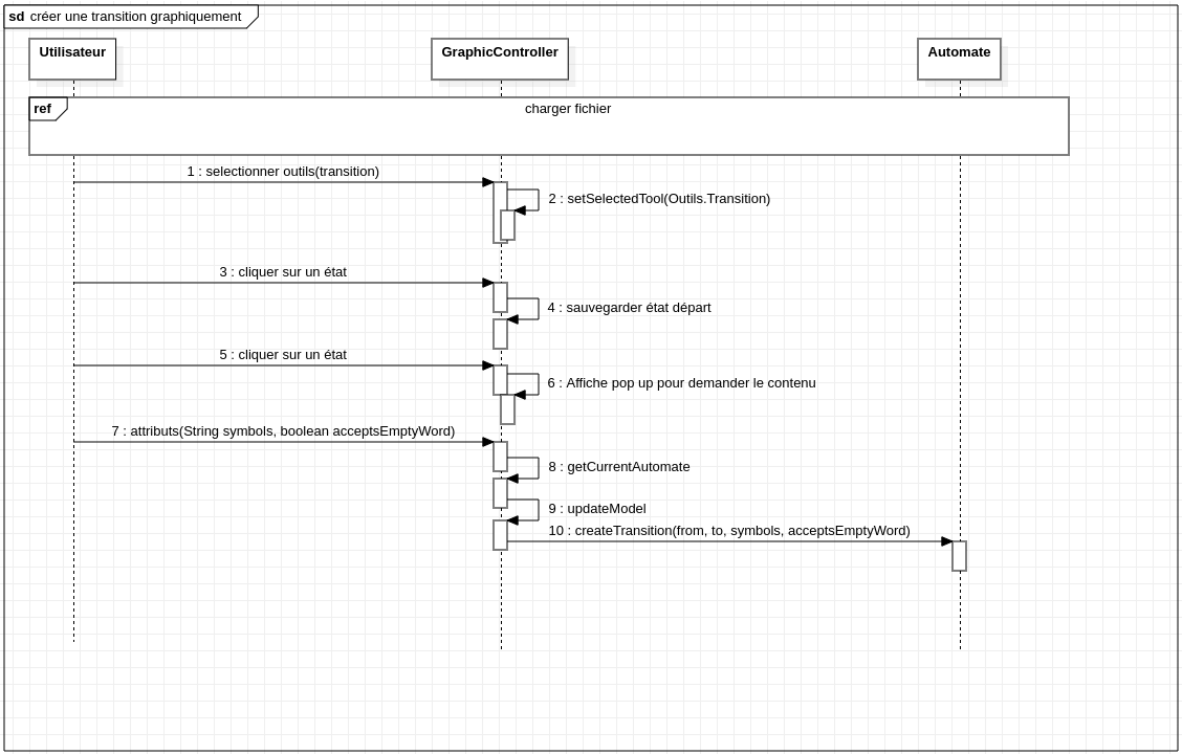


Fig. 14 : Diagramme de séquence de PRJ-004-3

5.2. Comportement du logiciel en erreur

5.2.1. Fichier corrompu

Si le fichier à charger est corrompu, une fenêtre pop-up d'erreur indiquant une corruption apparaît et l'utilisateur est renvoyé sur la page d'accueil.

5.2.2. Fichier altéré

Si le fichier a été altéré et que la checksum calculée ne correspond pas avec celle du fichier, alors il sera toujours lisible, mais une fenêtre pop-up apparaîtra, proposant d'ouvrir le fichier dans la vue XML.

5.2.3. Fichier non XML

La fonction parseXML de la classe XMLParser peut émettre une exception si la chaîne de caractères n'est pas du XML. Elle peut arriver si l'on essaye de charger un fichier n'étant du XML, le cas échéant, une pop-up indiquant l'erreur apparaîtra et l'utilisateur sera redirigé vers la page d'accueil. Elle peut également arriver lors de la sortie du mode XML, alors une pop-up apparaîtra et l'utilisateur restera sur la vue XML pour corriger son erreur.