ENSIMAG

ACVL

2e année alternance

Éditeur diagrammes états/transitions

Auteurs :
Arthur De Kimpe Maxime
Hagenbourger Nathanaël
Couret Thomas Bianchini

Enseignant :
Akram Idani

12 novembre 2015

Ce document est un rapport rendant compte de notre travail fourni sur le projet ACVL. Il contiendra notemment l'analyse et la conception de notre application. Cette application a pour vocation de créer et d'éditer des diagrammes états transitions en UML.

Table des matières

1 Analyse

L'analyse de l'application se découpe en deux parties. Tout d'abord l'analyse des cas d'utilisation puis les diagrammes de séquence des cas d'utilisation qui méritent à notre sens un éclaircissment sur l'utilisation de l'application.

1.1 Cas d'utilisation

La vision générale des cas d'utilisation est décrite ci-après :

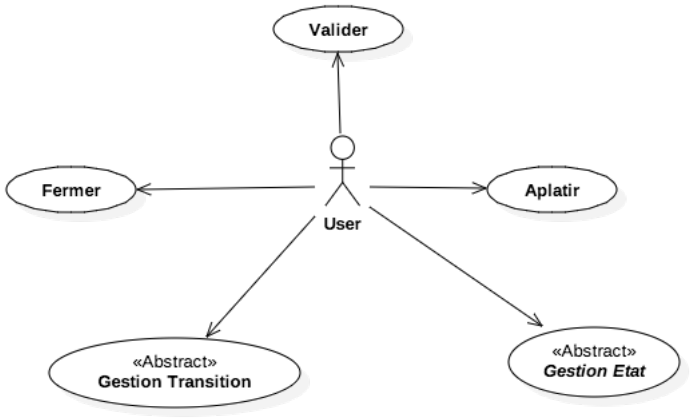

FIGURE 1 – Diagramme des cas d'utilisation factorisés de l'application

Dans un soucis de clarté nous avons découpé le cas “Gestion état ” ...

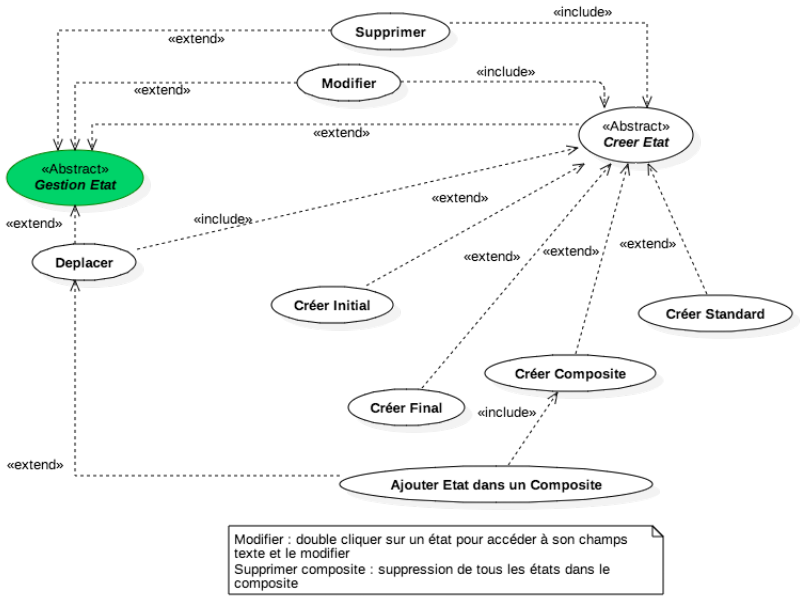

FIGURE 2 – Diagramme des cas d’utilisation détaillé pour Gestion État

... et le cas “Gestion des transtions”

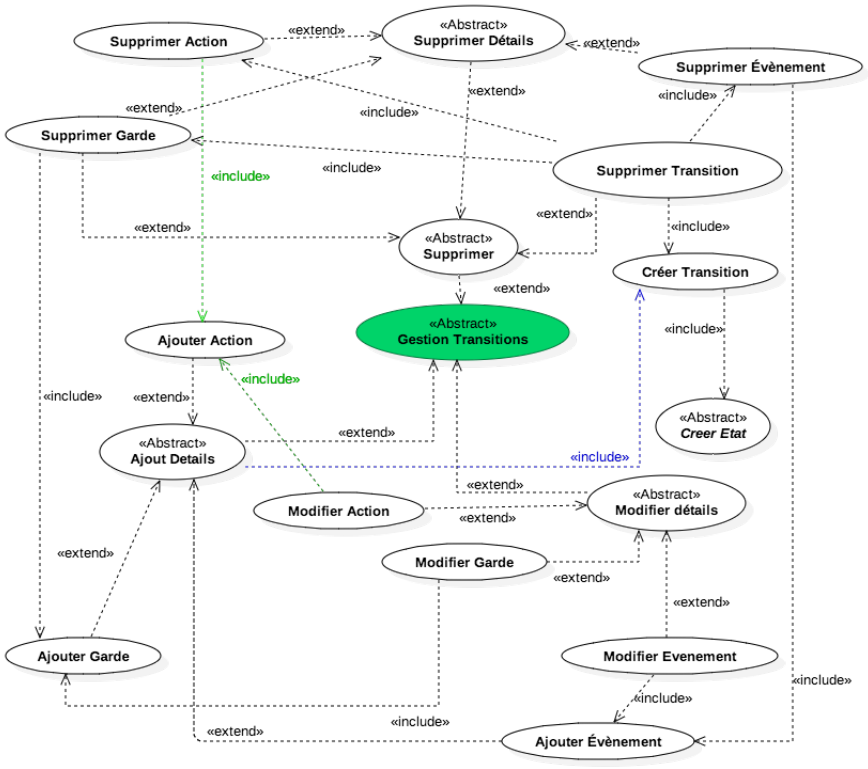

FIGURE 3 – Diagramme des cas d’utilisation détaillé pour Gestion Transition

1.2 Diagrammes de séquence "pertinents"

Nous avons identifié les cas d'utilisation qu'il semble important de décomposer en diagrammes de séquence :
— ajouter un état dans un composite
— créer une transition entre deux états
— déplacer un état
— modifier un état

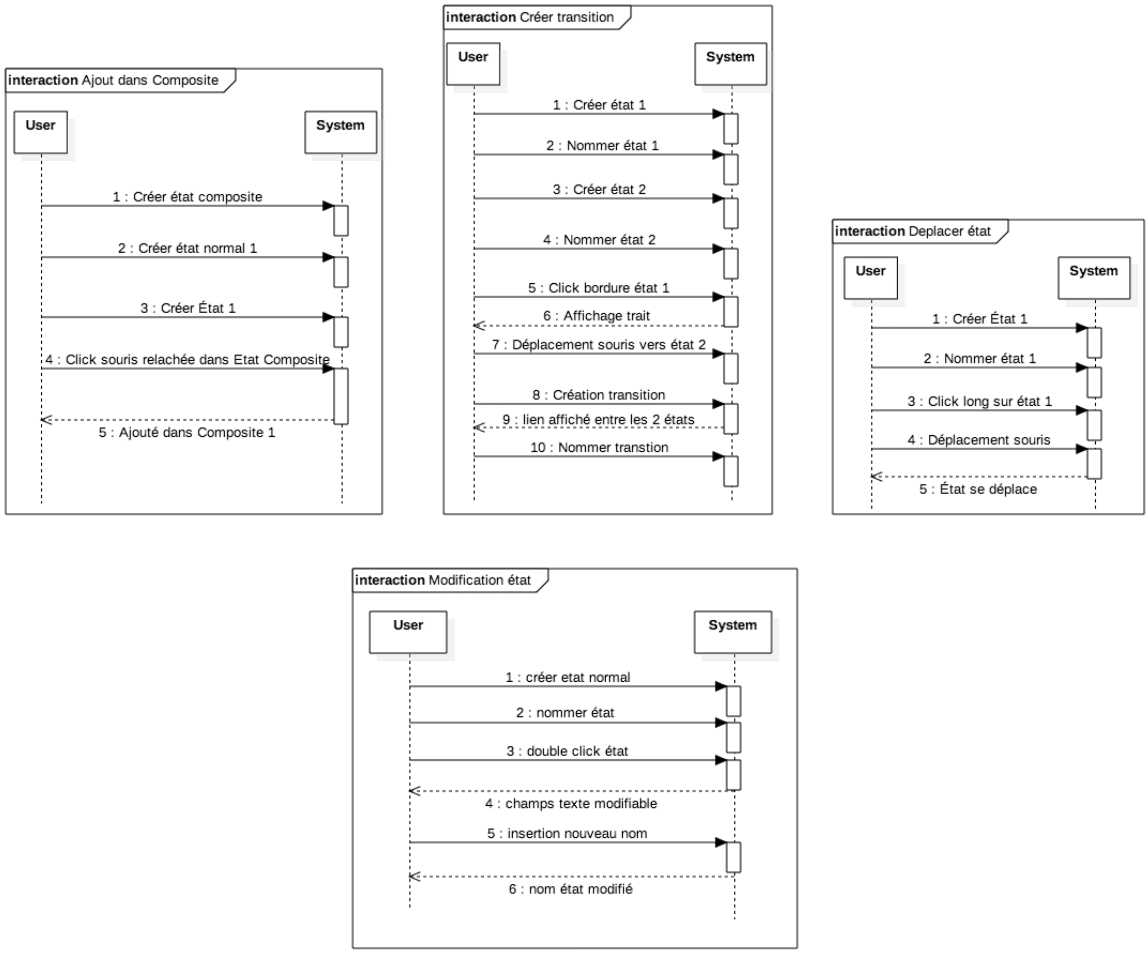

FIGURE 4 – Diagrammes de séquence

1.3 Diagramme de classes d'analyse

Dans la figure 5 ci-dessous, nous considérons la classe Diagram comme étant une classe contrôleur. L'appellation Direct sons indique que le Diagram est composée d'une liste d'état de premier niveau, c'est-à-dire qu'il n'est pas composé des états contenus dans les états composites dont il est composé. Il en va de même pour les états composites.

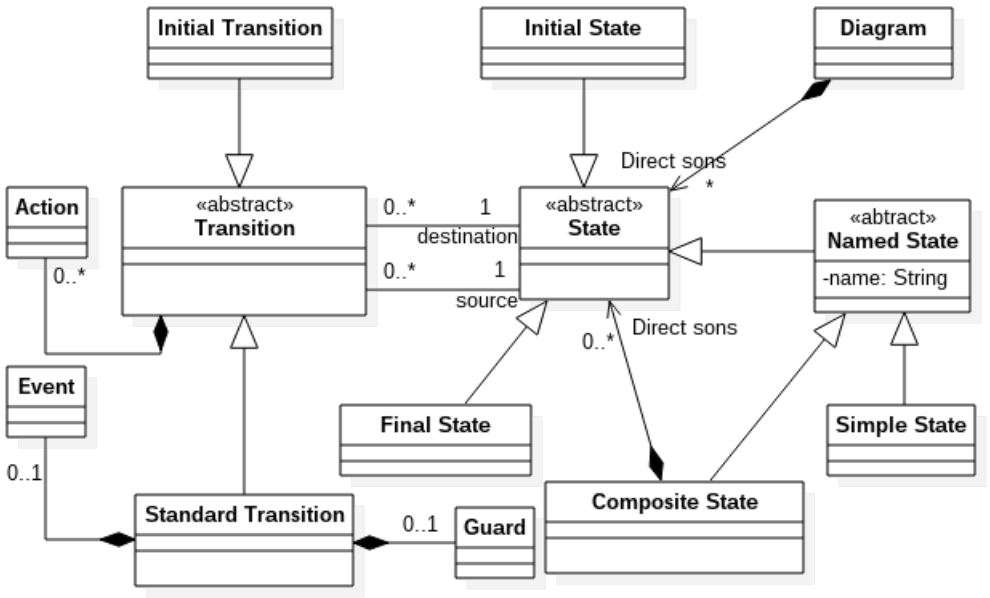

FIGURE 5 – Diagramme de classe d'analyse

2 Conception

2.1 Diagrammes

Ci-après, nous avons découpé le diagramme de classes logicielles en deux partie par rapport à la classe Diagram dans les figures 6 et 7, aucune association n'a été enlevée dans le processus.

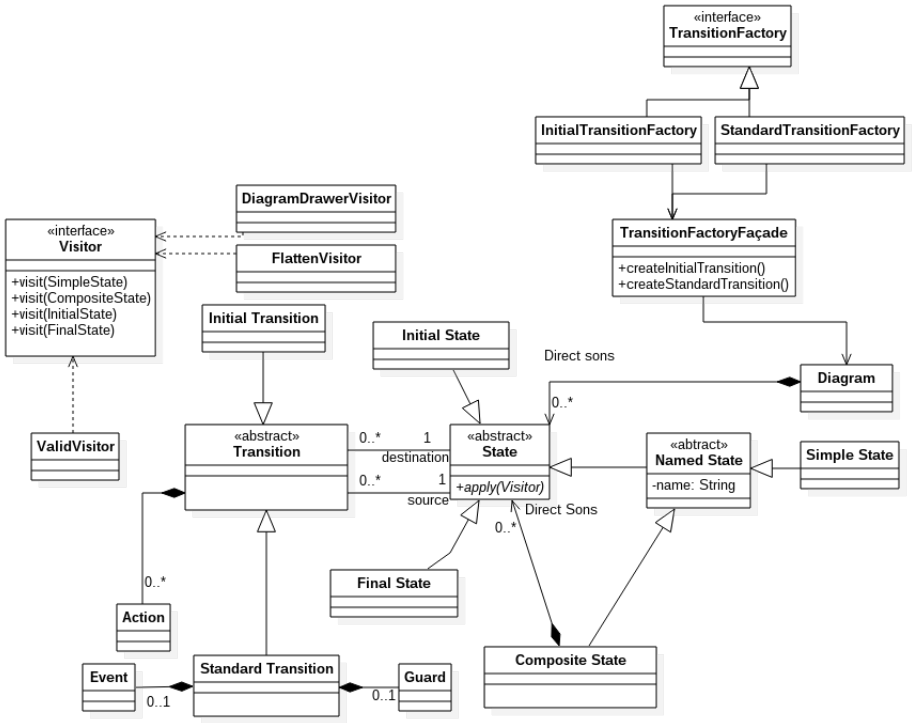

FIGURE 6 – Diagramme de classes logicielles partie 1/2

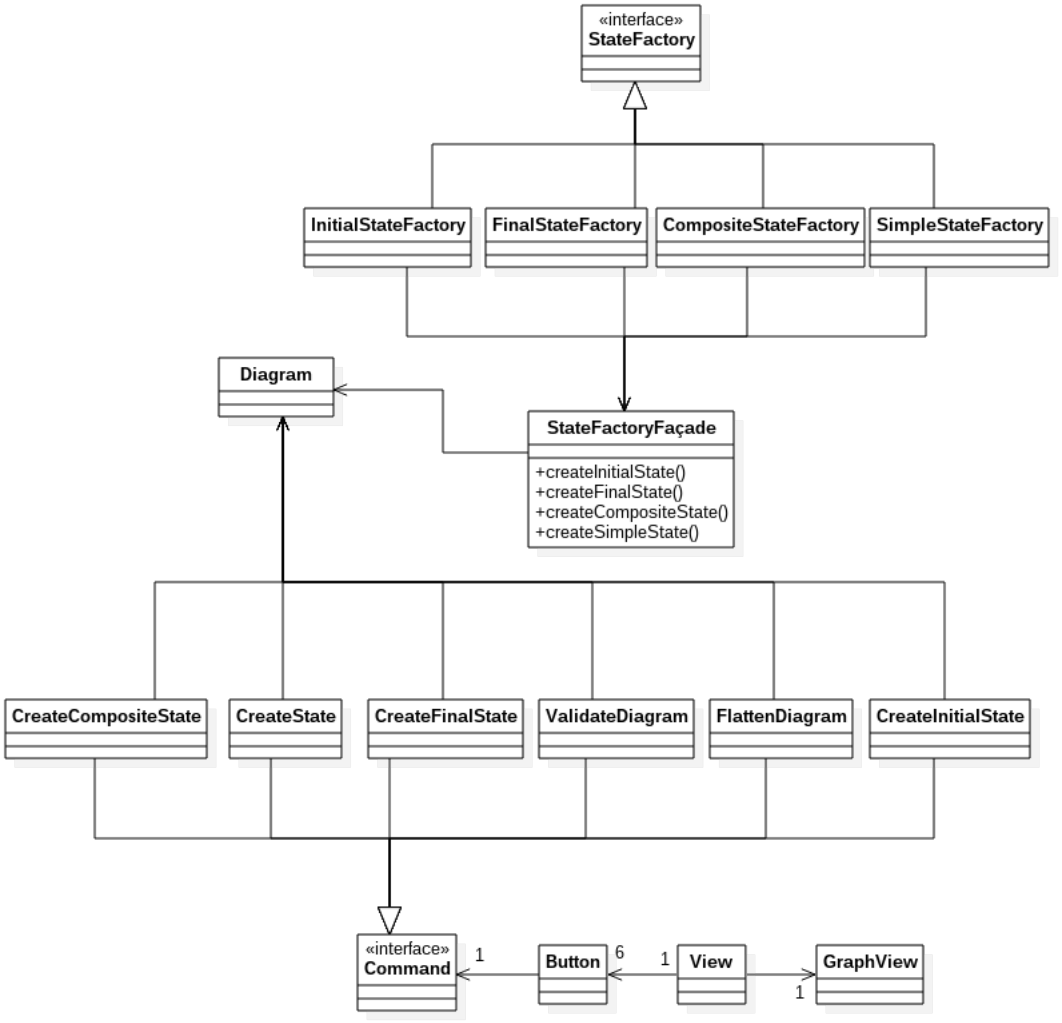

FIGURE 7 – Diagramme de classes logicielles partie 2/2

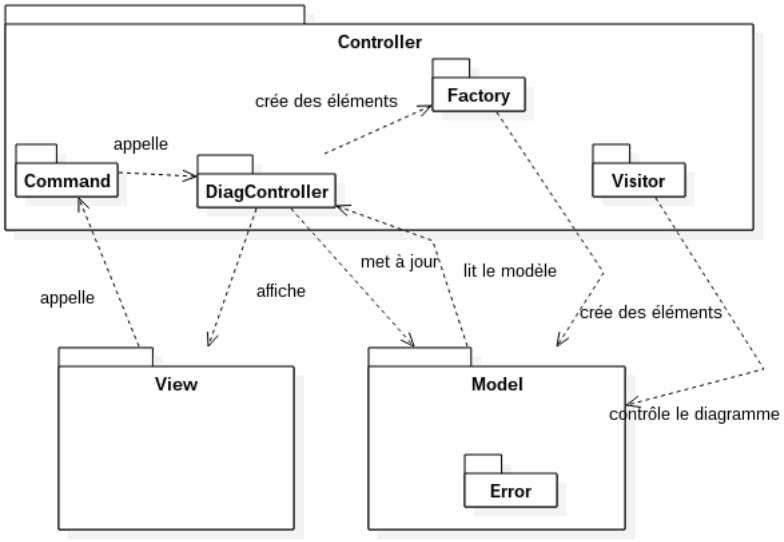

FIGURE 8 – Diagramme de packages

2.2 Justification Patrons de conception

— Commande : notre interface grpahique comprend un ensemble de boutons comme la création d'un état initial, final, composite, standard mais aussi la possibilité d'aplanir et de valider le diagramme. Chaque bouton ayant un comportement propre et travaillant sur une IHM, le patron Commande semblait tout indiqué pour décrire l'ensemble des comportements.
— Singleton sur le contrôleur : un diagramme doit être unique et notre contrôleur est notre diagramme. Le patron singleton nous permet donc de nous assurer cette unicité.
— Visiteur : le visiteur est utilisé pour la validation et l'aplanissement. Ces deux parties sont traitées après.
— Factory : Nous créons souvent des états et/ou des transitions avec des propriétés communes nécessaires. Cela permet de factoriser le code car nous n'appelons plus que la factory lorsque l'on souhaite créer les objets de notre modèle. De plus, la méthode abstraite publique délégue la construction des objets à ses sous classes qui déterminent le comportement de création souhaité.
— Facade : Encapsulation des factory pour servir ses méthodes dans un contexte statique.

2.3 Validation d'un diagramme état transition

La validation d'un diagramme dépend évidemment du modèle. Ainsi nous considérons que la validation est un contrôleur. La classe DiagramValidator contient donc un ensemble de méthode qui vérifient l'ensemble des conditions identifiées par l'équipe. Nous vérifions donc : tous les états sont atteignables, un état n'est pas un puit (on ne peut pas en sortir sauf pour les états finaux), un diagramme contient forcément un état initial et au moins un état final, un diagramme doit être déterministe selon les transitions (2 transitions identiques sortants d'un même état est interdit)...
Nous avons identifié que chaque état a des contraintes de validité qui lui sont propres. Nous avons donc choisi d'utiliser le patron visiteur afin de créer une méthode de validation par type d'état. Ainsi la condition "un état initial doit avoir seulement une transition sortante" n'est vérifiée que dans la méthode du visitor qui prend en paramètre un InitialState par exemple. Le code source du ValidVisitor présente l'ensemble des conditions vérifiées pour chaque type d'état.
De plus, nous souhaitons que l'utilisateur soit au courant des erreurs présentes dans son diagramme. Pour cela nous avons créer une classe DiagramError qui permet notamment de d'avoir une liste d'erreurs dans l'objet DiagramValidator. À chaque erreur détectée, on ajoute un objet DiagramError dans la liste de l'objet DiagramValidator. Une fois le traitement de toutes les conditions accomplies, la liste des erreurs est récupérée par le contrôleur liée à la vue. Ce dernier se charge alors de transmettre cette liste à la vue qui regarde simplement si la liste est vide (pas d'erreur) ou pas. Dans ce dernier cas, elle affiche dans une popup l'ensemble des erreurs.

2.4 Aplanissement d'un diagramme état transition

L'aplanissement d'un diagramme est intrinsèquement lié au modèle, c'est-à-dire qu'il utilise la hiérarchie de classe que nous avons conçu.

Il est basé sur le fait que la structure de données représentant le graphe en mémoire doit être valide.

L'utilisation du patron de conception Visiteur est justifiée car la gestion de l'aplanissement est différent pour chaque état. Dans les faits, seule la gestion de l'état composite est nécessaire. Cependant, dans le cas où nous voudrions rajouter des diagrammes parallélisés, il suffirait de rajouter une méthode prenant en charge ce type d'état pour l'ajouter à notre traitement. Ce choix est donc fait dans le souhait de garder une application modulaire.

3 Manuel utilisateur

3.1 Gestion des états

Pour ajouter des états normaux, composites, initiaux ou finaux il suffit de cliquer sur le bouton correspondant dans la barre de menu affichée en haut de l'application.

Les états sont déplaçables en les glissant/déposant quand le curseur de la souris indique que cela est possible.

Pour inclure un état dans un état composite il suffit de le déplacer au dessus de celui-là, une indication graphique qu'en le relâchant au dessus de l'état il sera ensuite ajouté.

À leur création, les états sont dotés d'un nom par défaut unique. Le double clic sur un état normal ou composite permet d'éditer son nom, si le nom existe déjà alors il est suffixé par un numéro pour rester unique.

3.2 Gestion des transitions

Pour tracer une transition vers un autre état, il faut placer la souris au centre d'un état, celui-ci passe en surbrillance et à partir de ce moment il est possible de maintenir le clic gauche et tracer une transition vers un autre état.

Par défaut, pour l'ensemble des transitions, une transition "Default transition", décrivant l'action, sera créée. Pour renommer une transition, il suffit de double cliquer sur le nom de la transition. Pour les transitions entre états, on peut préciser un évènement et/ou une garde. Les transitions auront donc le format suivant :

- Action
- Action / (Évènement)
- Action / [Garde]
- Action / (Évènement) / [Garde]

Si l'utilisateur ne précise pas le type de ce qu'il cherche à décrire (garde ou évènement), par défaut, la première partie décrira systématiquement l'action, la deuxième partie l'évènement (et ajoutera automatiquement des parenthèses) et la troisième partie la garde (et ajoutera automatiquement des crochets).
Si l'utilisateur veut ajouter uniquement une garde en plus de l'action, il devra donc bien ajouter les crochets pour pas que ce ne soit confondu avec un évènement.

Pour supprimer état ou transition, il faut sélectionner la cible et presser la touche Suppr. (ou sur Mac : fn + backspace).

3.3 Validation du graphe

Pour savoir si un graphe est valide, il convient de cliquer sur le bouton Validate qui informe l'utilisateur de la validité du graphe en indiquant.

3.4 Aplanissement du graphe

L'aplatissage d'un graphe se fait en pressant le bouton Flatten. Si le graphe est invalide alors l'opération n'est pas effectuée.

Dans le coup où une transition est supprimée par l'opération (i.e. elle est liée à un état composite ou est liée à l'état initial d'un état composite) alors les informations la concernant ne sont pas propagées sur les nouvelles crées pour faciliter le traitement. Il conviendra donc de les rentrer à nouveau.

4 Bilan sur les outils de modélisations

Afin de modéliser nos diagrammes nous avons réalisé une étude des logiciels de modélisation UML existants. Très peu de logiciels sont compatibles sur l'ensemble des systèmes d'exploitation et sont open sources. Le seul outil utilisable a été StarUML car simple à installer, compatible multi-plateformes et assez ergonomique.
Cependant StarUML réserve quelques comportements surprenants comme le copié collé des éléments qui crée des liens entre les objets ou encore la gestion des nombres d'opérations dans les diagrammes de séquences.
Néanmoins StarUML permet de créer un projet contenant tous les diagrammes possibles(use cases, séquences, classes ...). Les projets StarUML permettent donc de gérer correctement l'ensemble des diagrammes et se partager les informations "proprement".

5 Annexe

5.1 Librairie graphique utilisée pour les graphes

Pour faciliter la mise en place d'un éditeur graphique permettant la manipulation de graphe, nous avons utilisés la libraire JGraphX.

Nativement, elle permet de gérer toutes les actions glisser/deposer, l'assignation des touches comme Suppr. vers un évènement donné (par exemple la suppression), le traçage simplifié de transitions, l'édition du nom des états etc.

Pour plus d'informations, il est possible de se rendre sur la page Github du projet :
`https://github.com/jgraph/jgraphx/`

5.2 Table des figures

Table des figures