TP2, INF5153, UQAM

Document de conception détaillée

Version <1.0>
Date

Auteurs:

Nom et prénom	Code permanent	Responsabilité
Maxime Lafond	LAFM31059007	Gestionnaire Projet
Nicolas Rivet	RIVN13098901	Contrôle Qualité
Pascal Féo	FEOP09088802	Responsable Implémentation

Historique des révisions

Date	Version	Description	Auteur
2015-04-16	1.0	Commencement document	maxime lafond
2015-04-25	1.1	Correction diagramme séquence	maxime lafond

Table des matières

Conception détaillée

1. Introduction

1.1 But du document

Ce document est un support explicatif pour le logiciel Circuit editor qu'on construit présentement pour notre client. Il explique en détail la structure interne du logiciel via différents diagrammes de conception.

1.2 Public visé

En ayant eu comme mandat de créer un logiciel de modélisation de circuits logiques, ce document est fort utile au projet pour transposer nos idées à notre client. En effet, ce document regroupe beaucoup d'information sur la structure globale du logiciel. Le public visé est notre client.

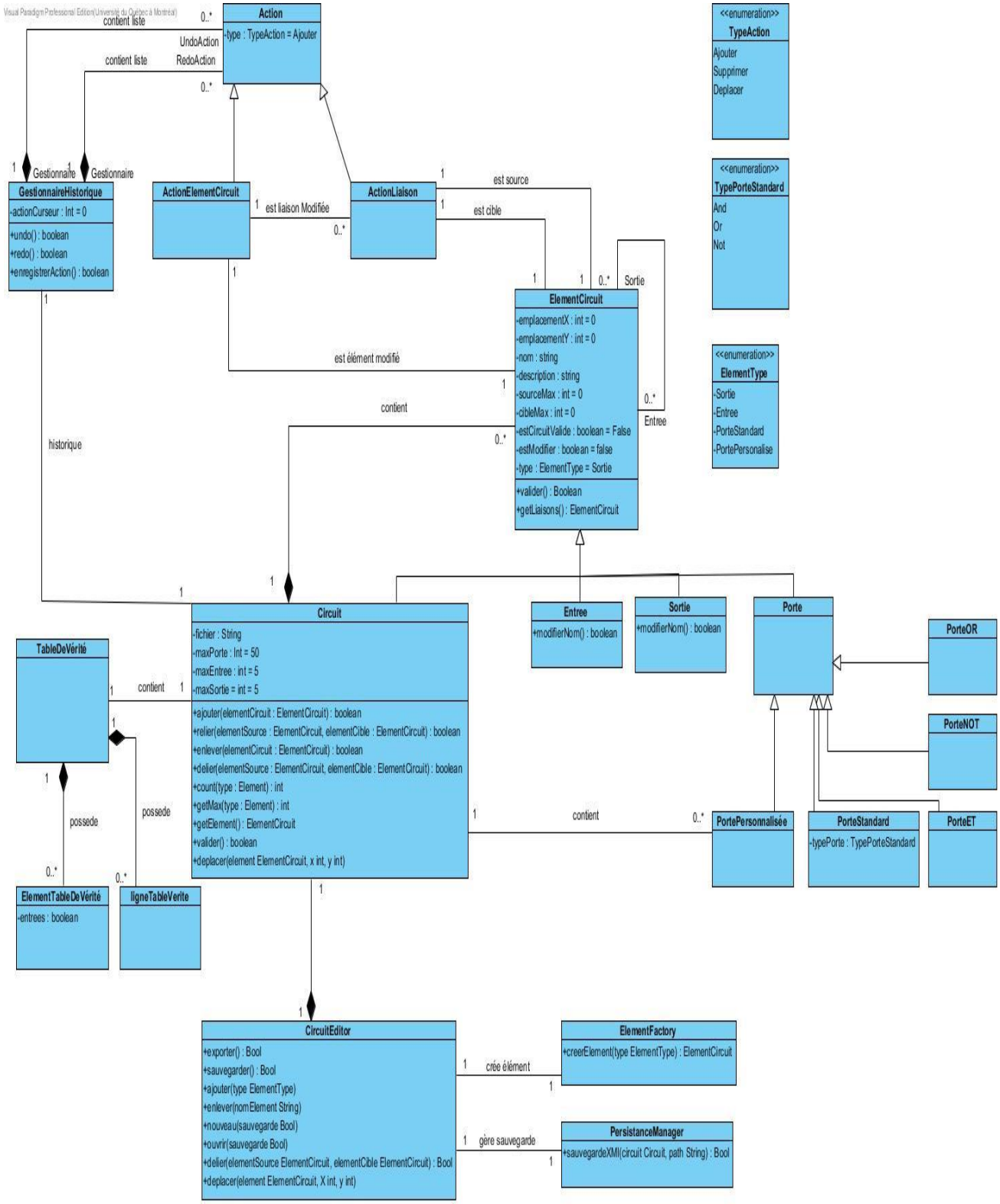
1.3 Définitions, acronymes et abréviations

Les sections qui suivent regrouperont les différents diagrammes conceptuels du projet ainsi que l'explication des patrons utilisés. Il aura pour débuter le diagramme de classe, ensuite l'explication des patrons utilisés et s'ensuivra d'une multitude de diagrammes de séquence pour l'ensemble des cas d'utilisation du projet.

2. Description sommaire du système

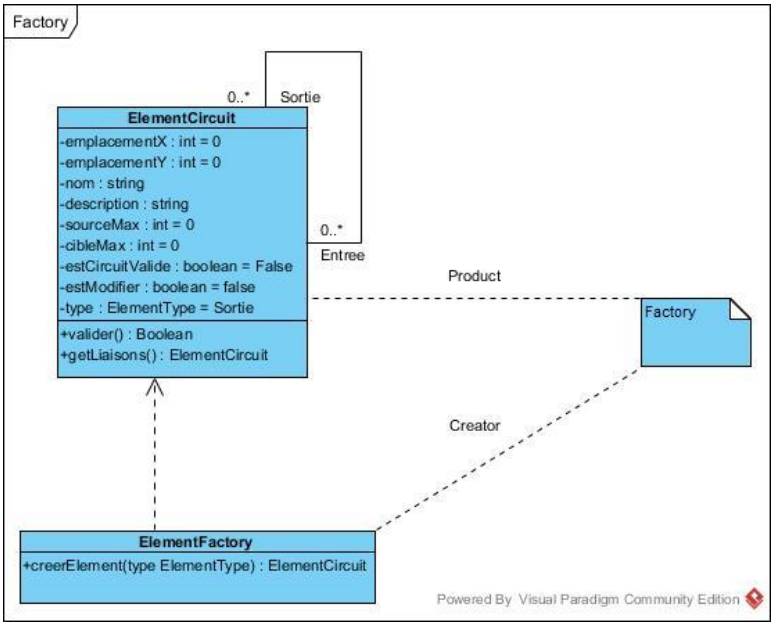
Ce projet de session vise à réaliser la conception et l'implémentation d'un logiciel de modélisation de circuits logiques. Un circuit logique est un ensemble de portes logiques reliées entre elles à travers leurs entrées et leurs sorties. Principalement utilisé en électronique, un circuit logique permet d'utiliser la logique booléenne pour concevoir des circuits intégrés; les portes logiques étant remplacées par des diodes ou des transistors.

3. Diagramme(s) de classes



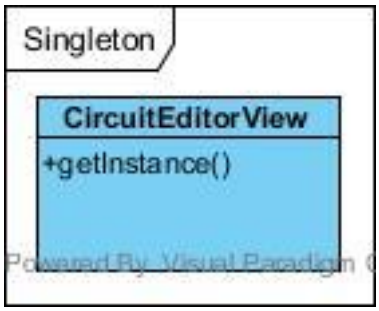
3.1 Patrons de conception

3.1.1 Patron : Fabrique



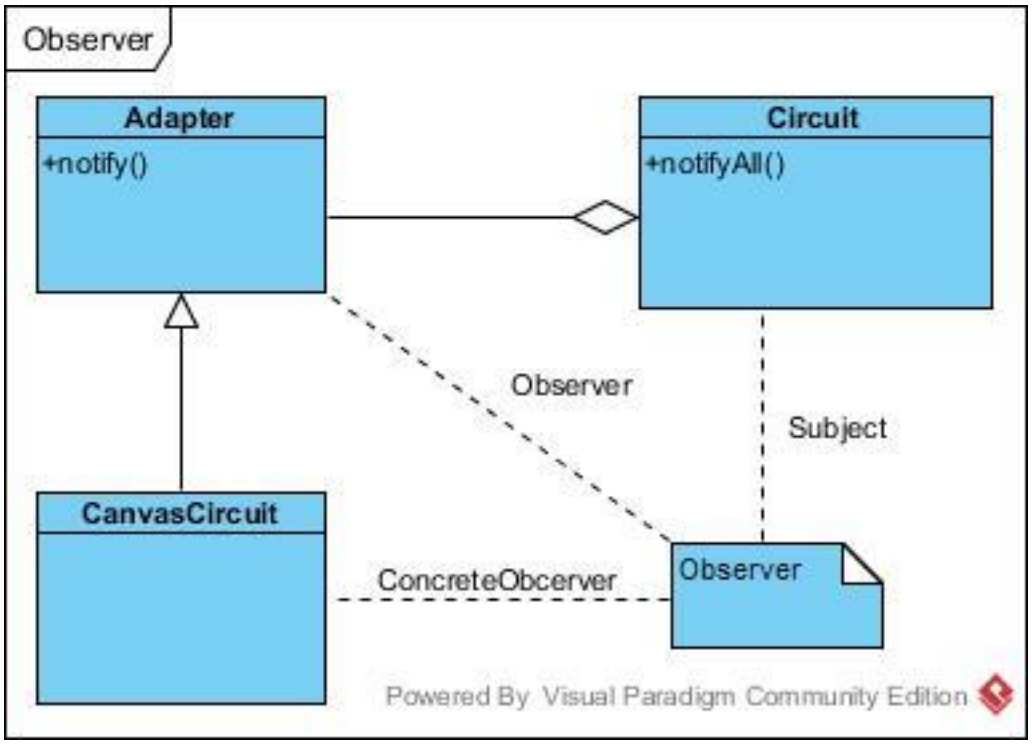
Circuit Editor implémente le style architectural MVC en utilisant EMF. Pour pouvoir définir et ajouter des éléments au modèle sans s'attacher directement au Framework EMF, la classe ElementFactory a été créé pour gérer la création des ElementCircuit. Le factorie générer par EMF aurait pu suffire pour cette responsabilité mais depuis que l'application devait créer des objets spécifique, par exemple un circuit de porte AND standard avec les noms des éléments déjà spécifié, le contrôleur devait avoir cette responsabilité afin d'augmenter la cohésion de l'application en séparant les préférences et les fonctions fondamentaux du modèle d'affaire. Pour ce dernier, la factory CircuitEditorFactoryImpl à été utilisé pour factoriser les créations d'objet et limiter le couplage.

3.1.2 Singleton



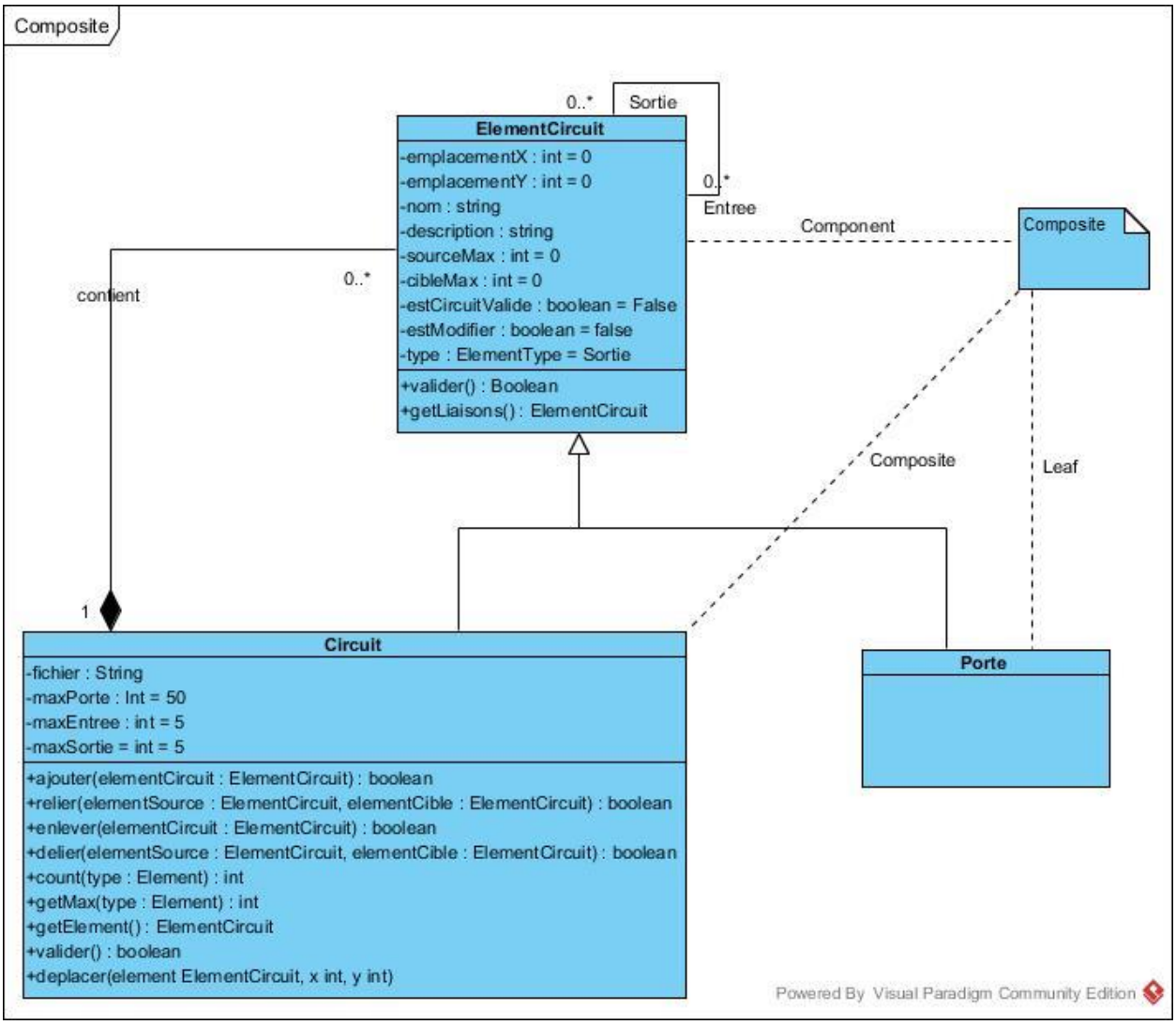
Le patron Singleton à été utilisé pour s'assurer qu'une seule instance du GUI était active à la fois. Ceci est très utile depuis que CircuitEditorView est l'expert en information pour l'interface graphique. Il est donc important d'avoir une référence unique pour tous les composants du GUI et accès au modèle qu'il observe. De plus, lorsqu'un évènement est déclenché par un bouton, le gestionnaire des évènements peut facilement accéder aux composants nécessaires sans qu'ils soient couplés directement.

3.1.3 Observer



La vue du style architectural MVC doit observer le modèle et se mettre à jour lorsqu'y change. Le Framework EMF facilite la gestion des changements en déclarant tous les éléments du model en tant qu'observable. Chaque élément contient un liste d'Adapter qui peuvent être notifié lorsqu'il change. Pour CircuitEditor, la classe CanvasCircuit est responsable d'écouter au changement du model afin de mettre à jour la représentation graphique du Circuit actuellement charger. La classe Adapter fait référence a EMF.

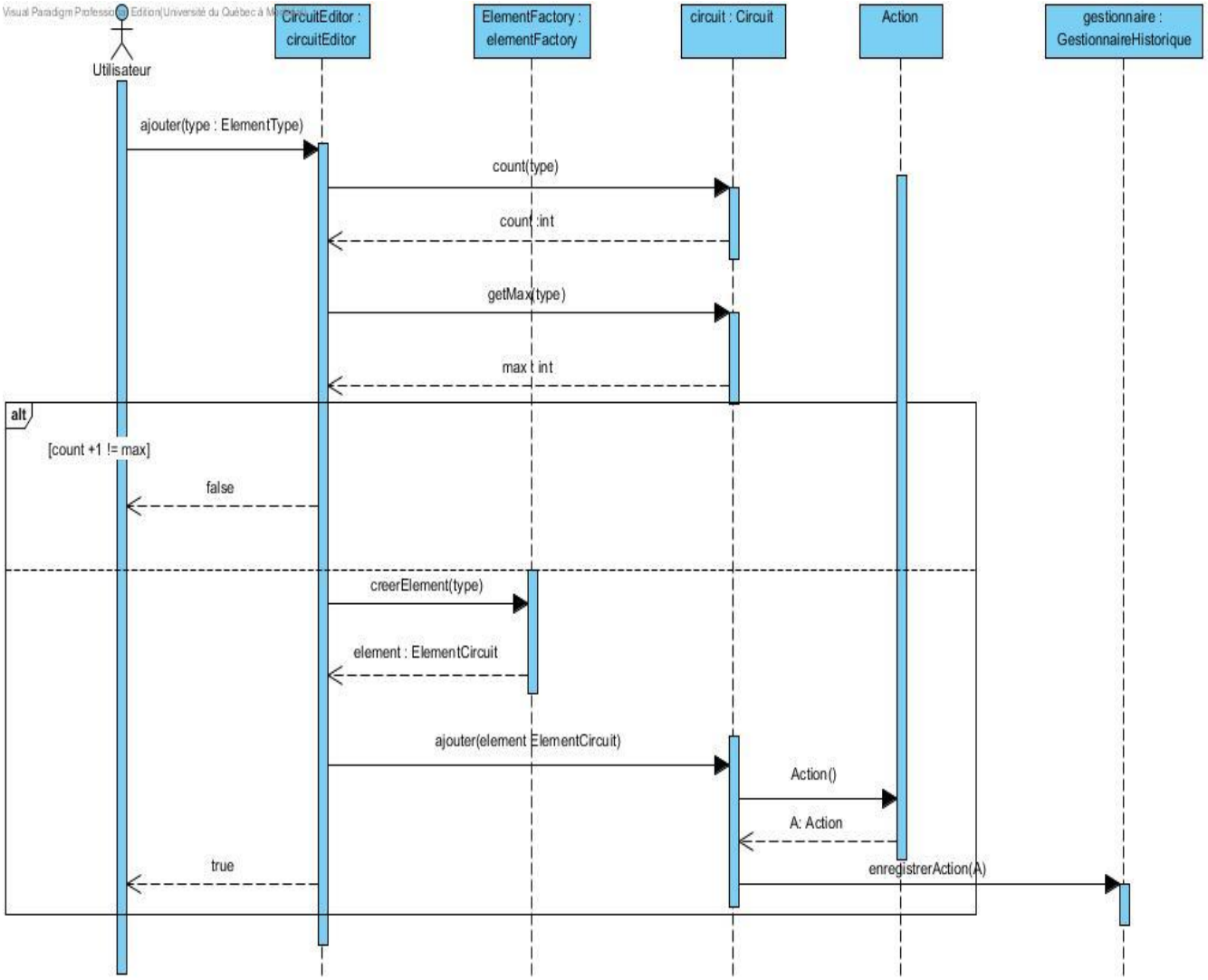
3.1.4 Composite



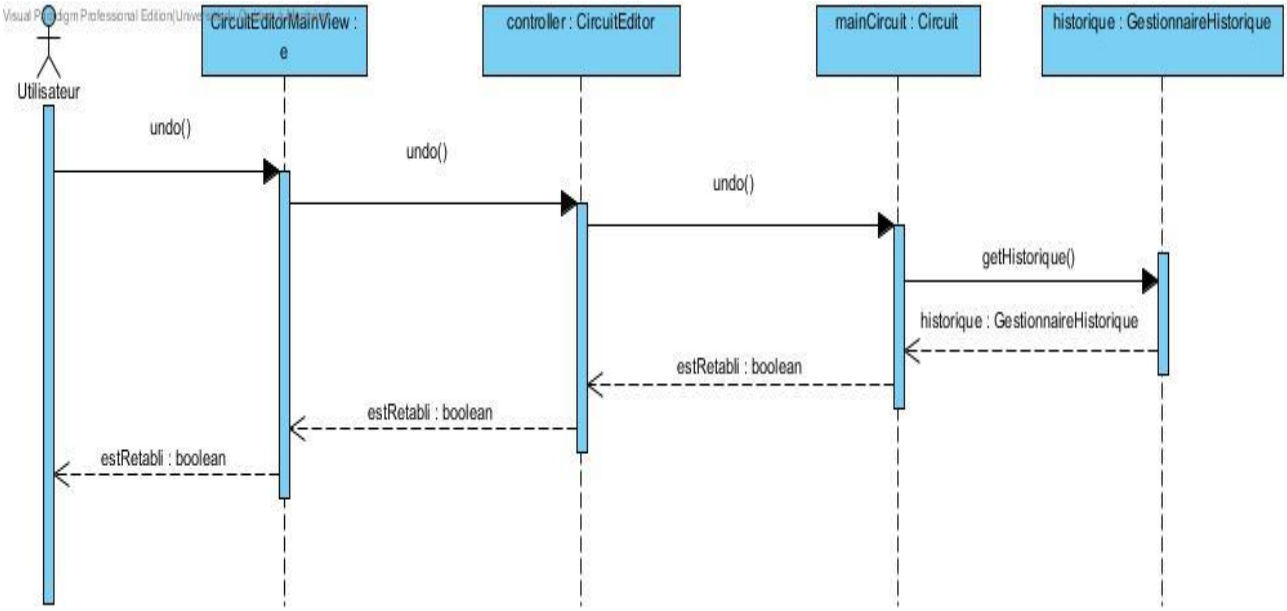
Le patron Composite facilite la modélisation d'un circuit depuis qu'il a une définition récursive. Un Circuit et une porte sont composés d'entrée/sortie mais un Circuit est composé de plusieurs portes au lieu d'une seule. Ce patron permet de modéliser une porte étant un cas particulier de circuit et augmenter la cohésion entre ces éléments. Les opérations communes peuvent être implémenter différemment si nécessaire, par exemple valider() vérifie s'il une Porte à ces entrée et sortie connecté mais pour Circuit itère sur tous les éléments du circuit et les valide, individuellement. Ce patron aide aussi à différencier les entrées/sortie globales du circuit, qui sont représenté par des icone bleu/rouge dans la vue, et les entrées d'une porte qui sont d'autres portes.

4. Diagrammes d'interaction

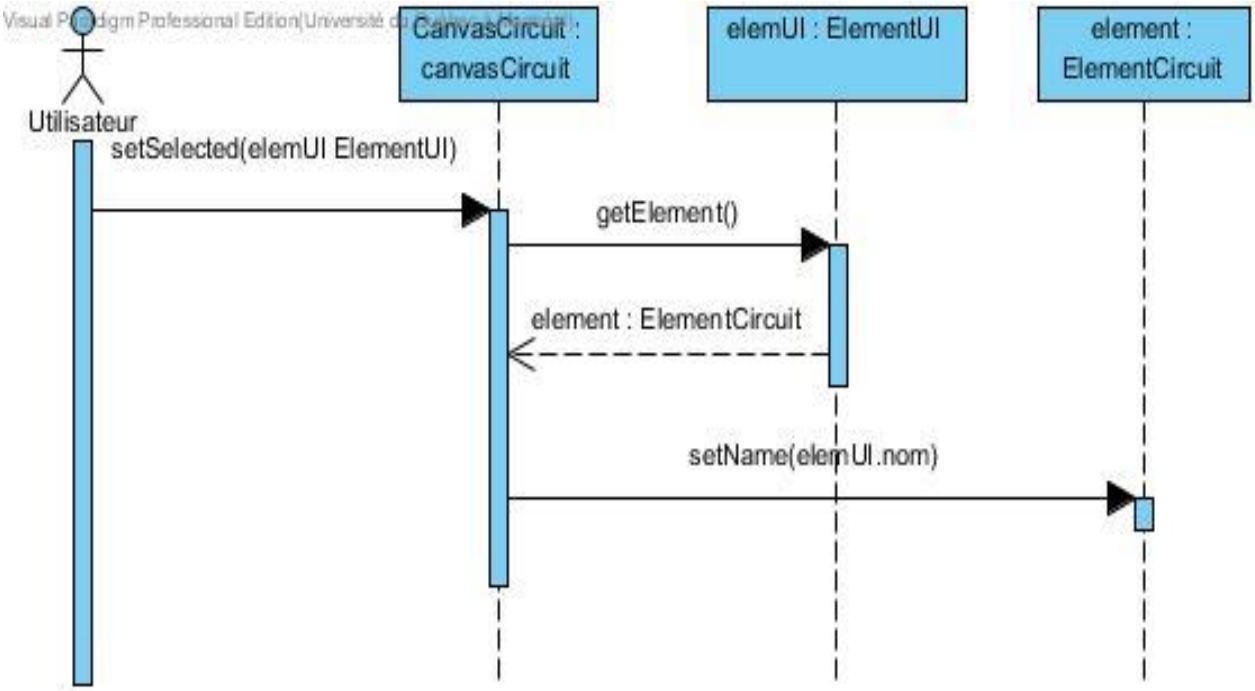
Ce diagramme de séquence fait référence à l'action ajouté un élément au circuit. Avant d'ajouter un élément, on voit qu'il doit vérifier à ne pas dépasser le maximum permis. Après l'ajout, on voit également qu'il doit le mentionner à l'historique.



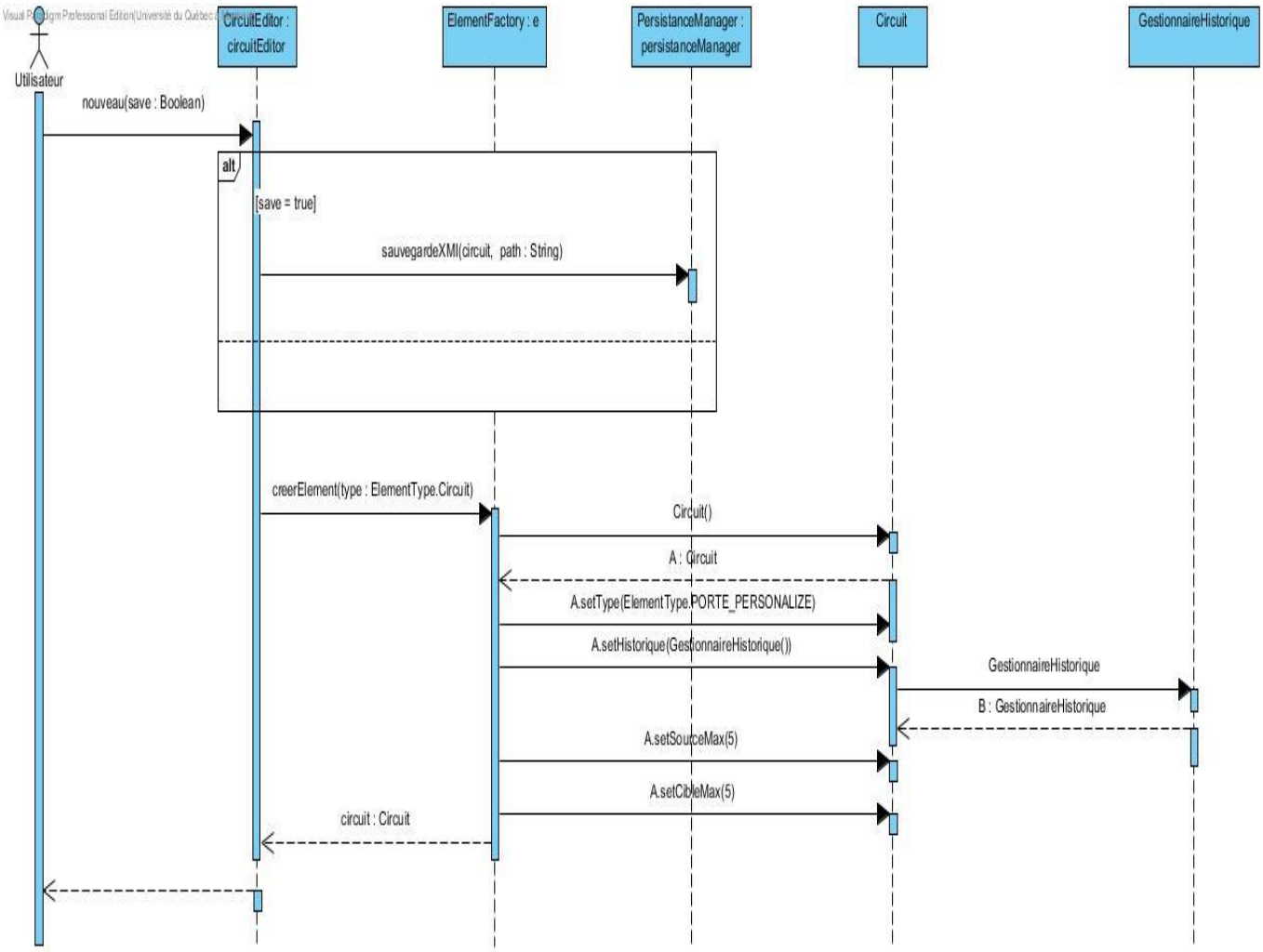
Ce diagramme de séquence fait référence à l'action d'annuler la dernière opération fait sur le circuit. L'historique va chercher la dernière action dans sa liste pour effectuer un retour en arrière du circuit.



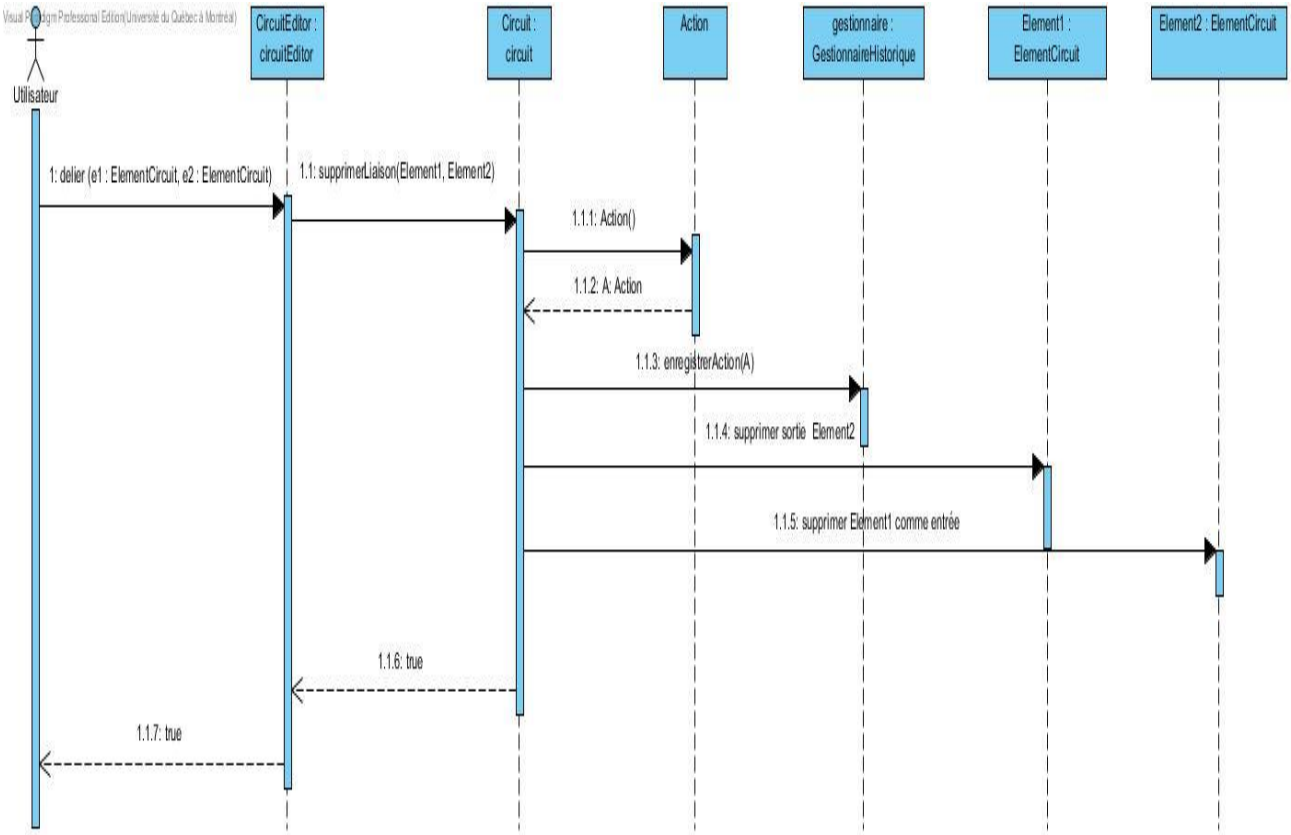
Ce diagramme de séquence fait référence à l'action de définir un nom pour une entrée ou une sortie.



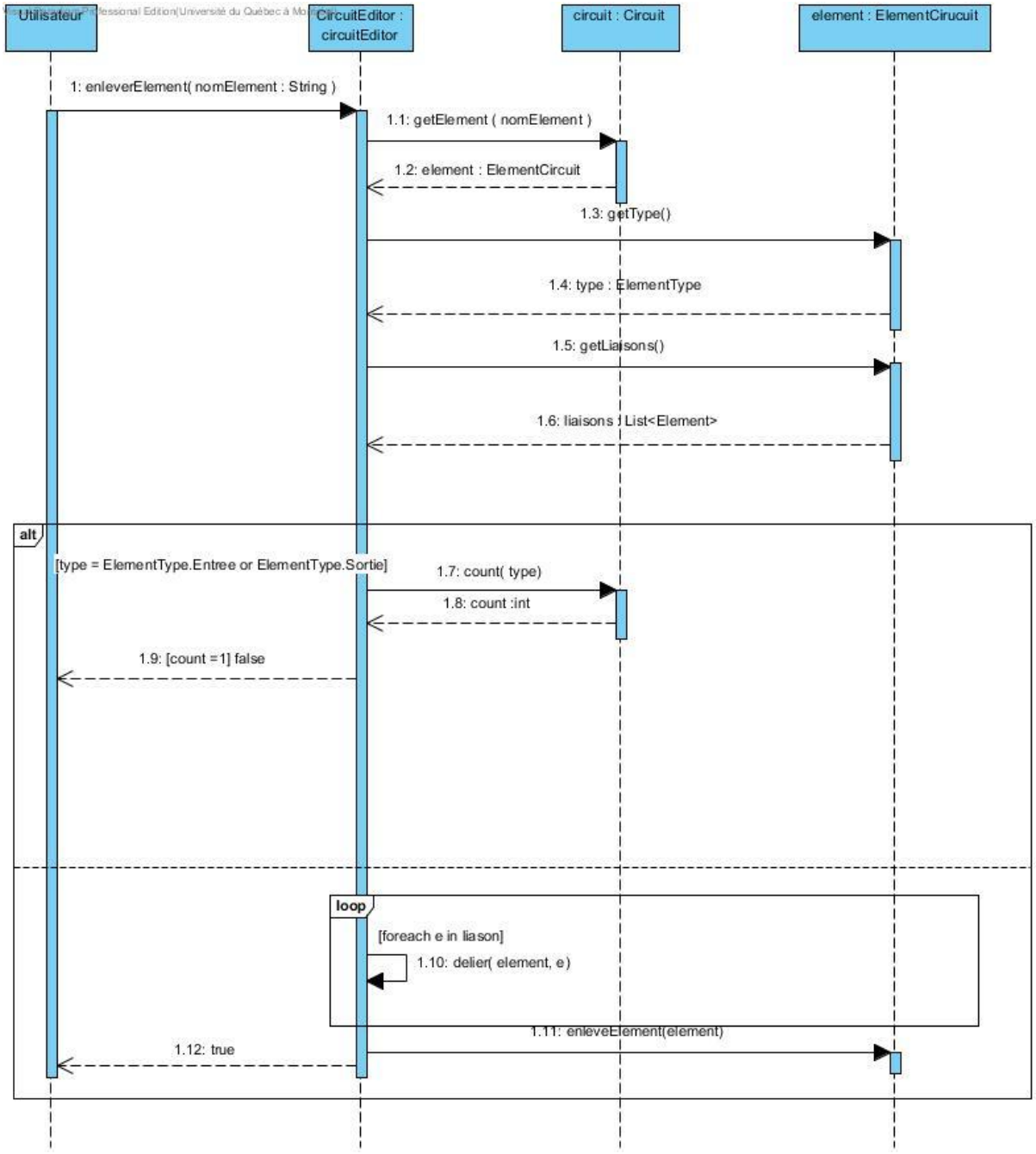
Ce diagramme de séquence fait référence à l'action de créer un nouveau circuit. Le logiciel sauvegarde avant tout l'ancien circuit si l'utilisateur le demande via sauvegardeXMI() de persistanceManager. Ensuite, il crée le circuit avec ElementFactory.



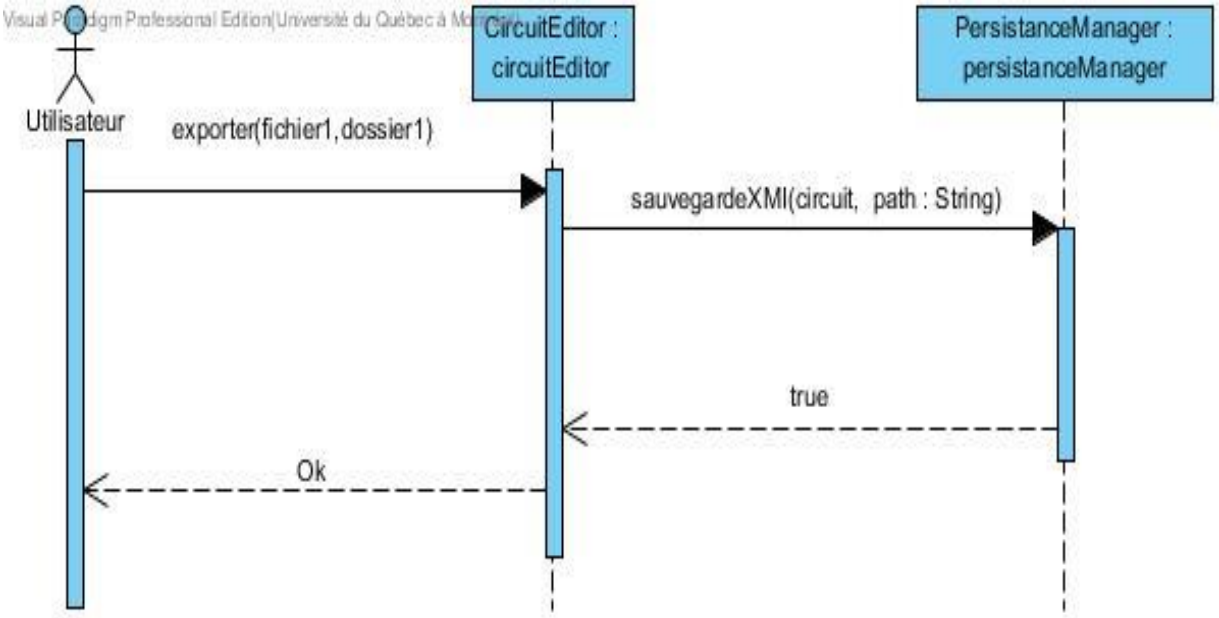
Ce diagramme de séquence fait référence à l'action de délier deux éléments du circuit. On voit qu'il est important par la suite de créer une action et de l'envoyer à l'historique au cas entre autres d'un undo ou d'un redo par la suite.



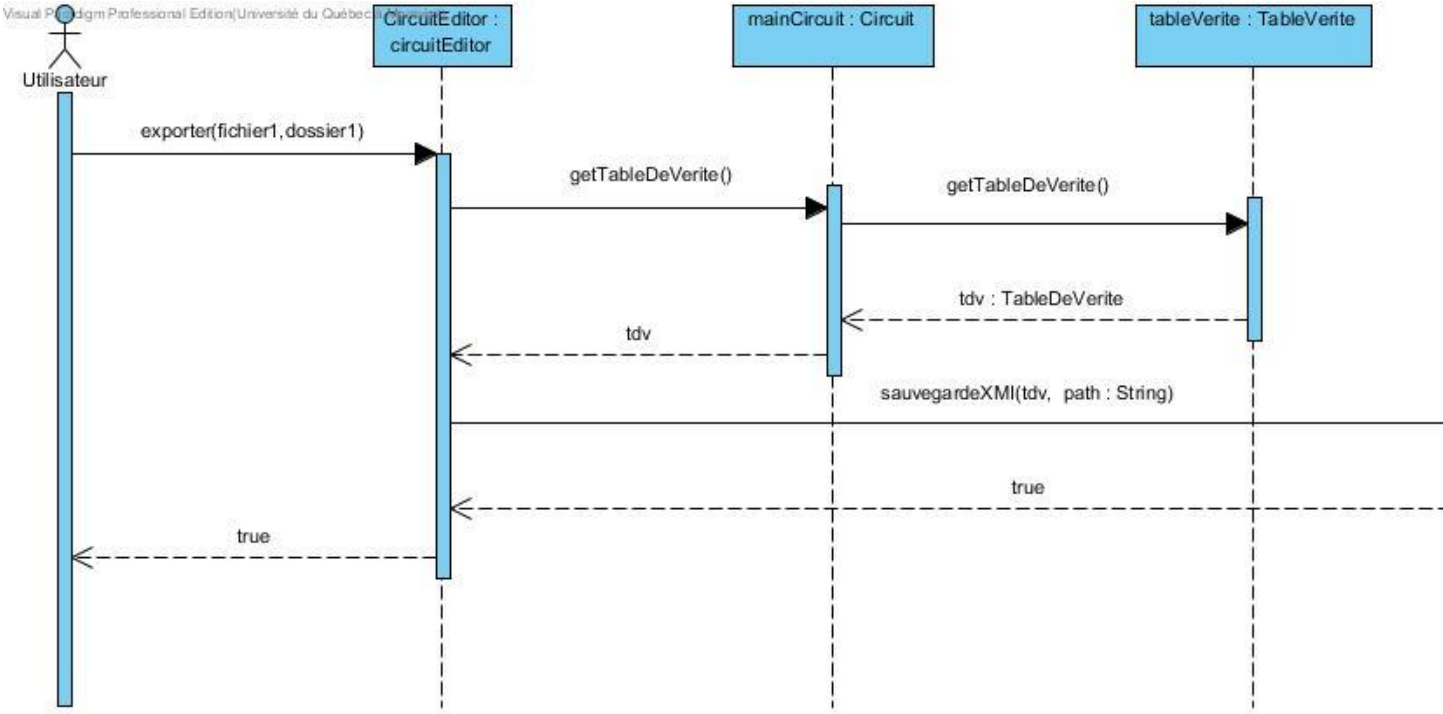
Ce diagramme de séquence fait référence à l'action d'enlever un élément au circuit. Il faut entre autres délier tous les éléments étant rattacher à cette élément et vérifier qu'on respecte toujours les contraintes.



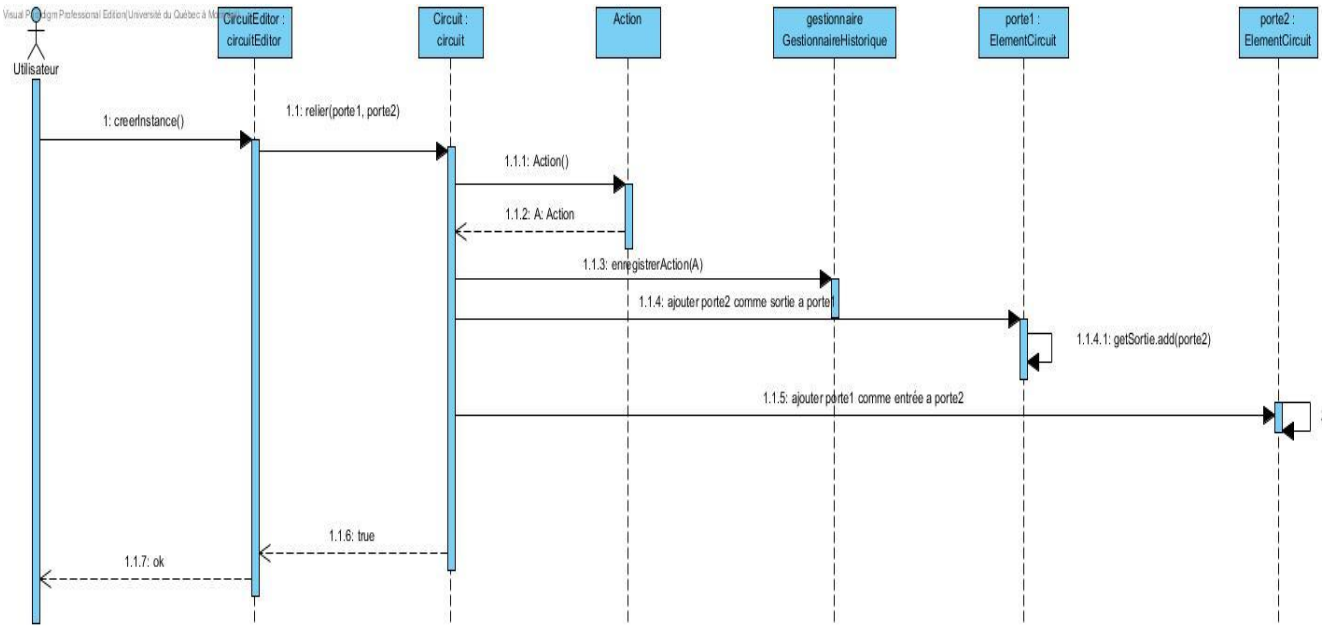
Ce diagramme de séquence fait référence à l'action d'exporter un circuit. C'est la classe circuit qui gère la transaction avec le disque via la méthode exporter().



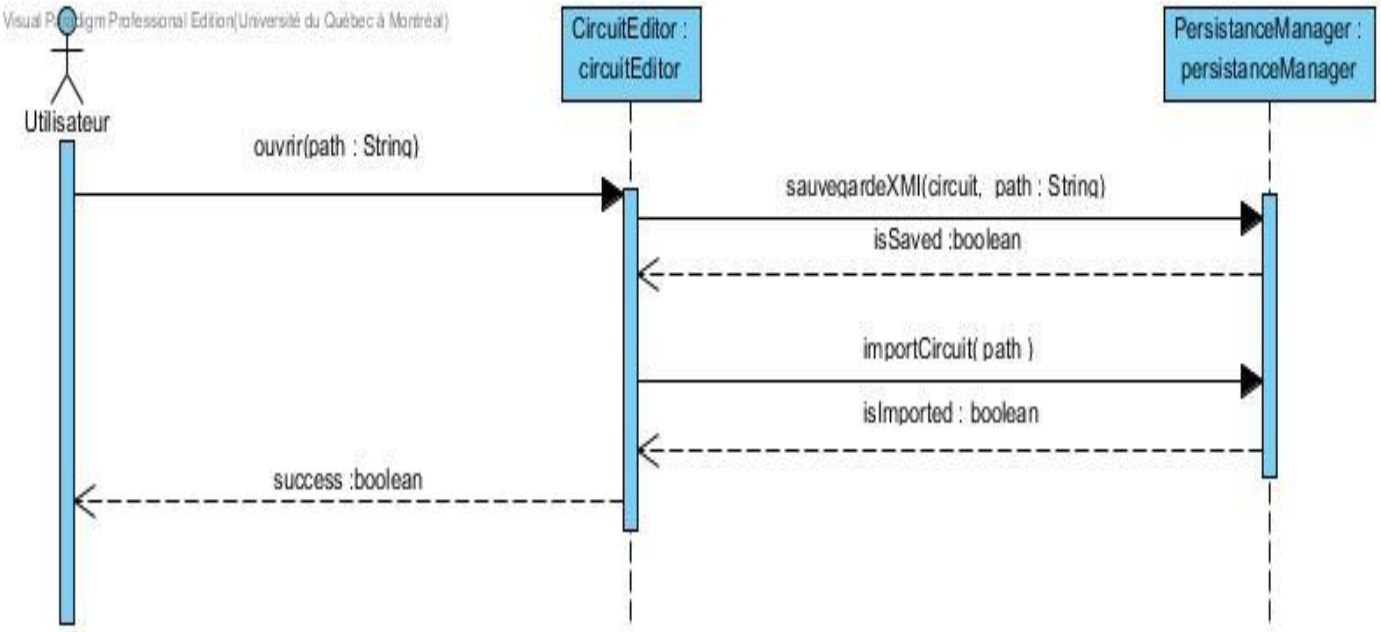
Ce diagramme de séquence fait référence à l'action d'exporter une table de vérité.



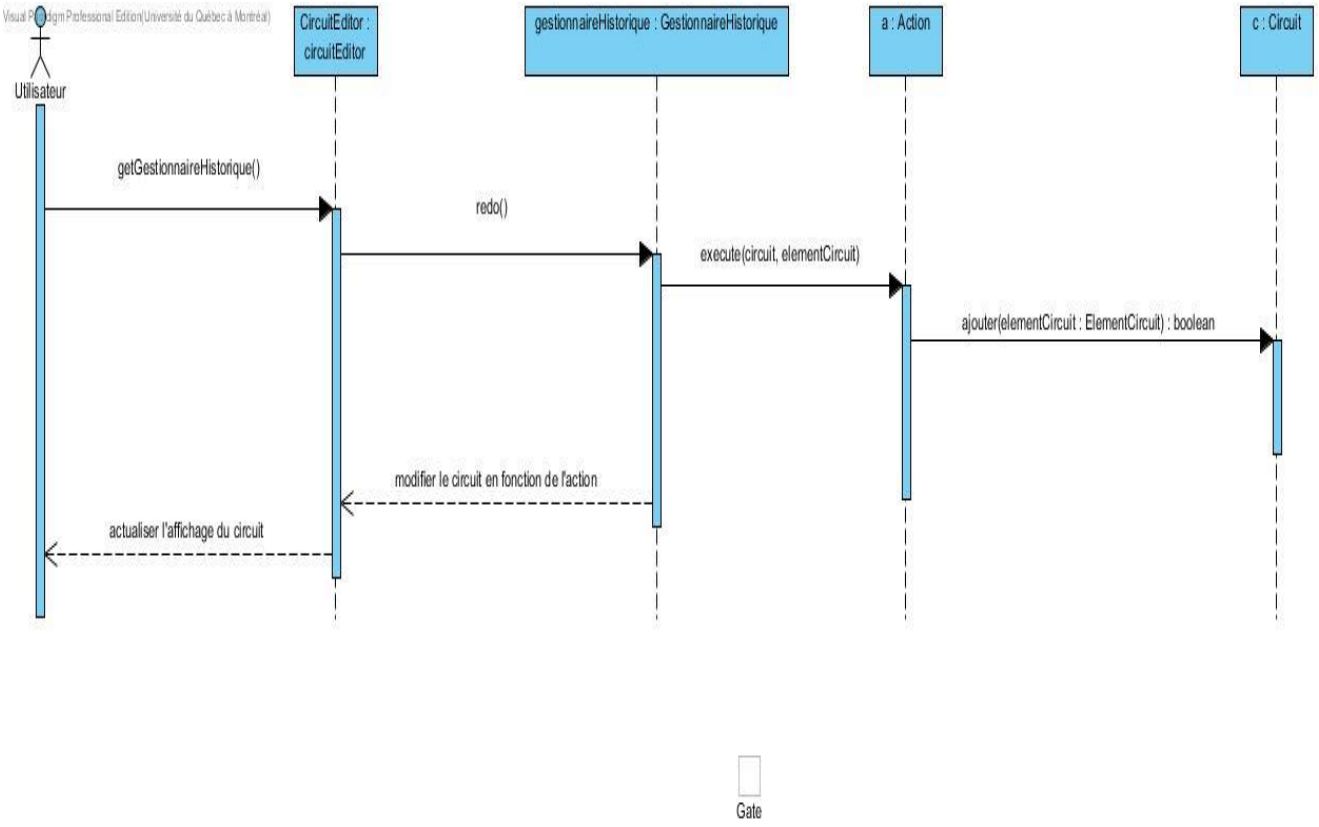
Ce diagramme de séquence fait référence à l'action de lier deux éléments du circuit. On crée l'action sur les deux éléments et on l'enregistre dans l'historique.



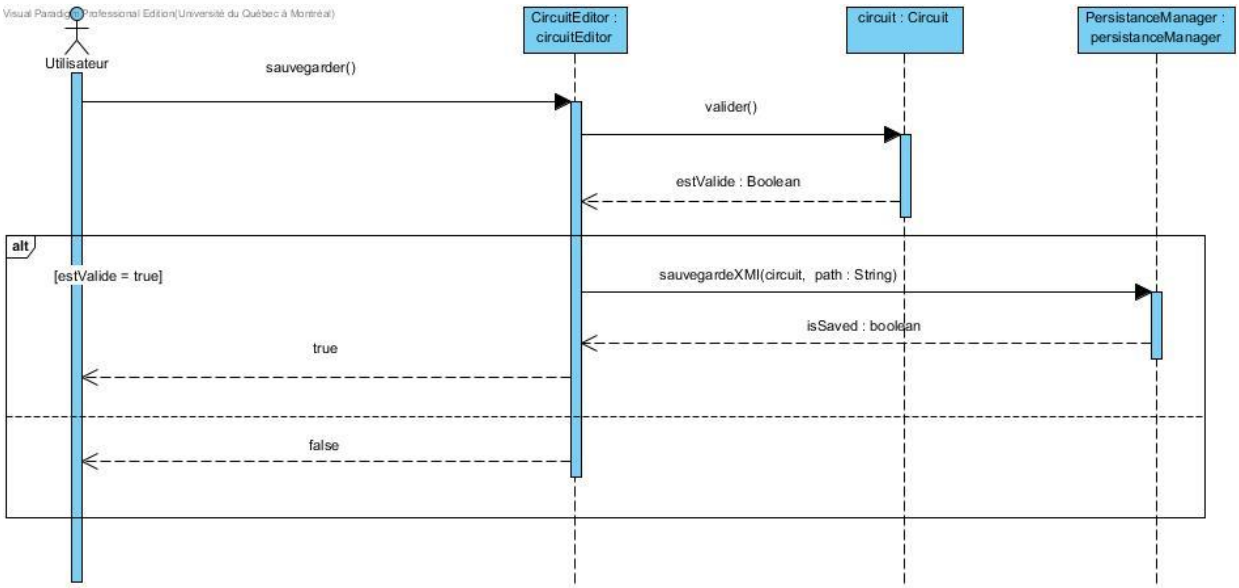
Ce diagramme de séquence fait référence à l'ouverture d'un circuit. On sauvegarde l'ancien circuit si nécessaire et on importe le nouveau circuit.



Ce diagramme de séquence fait référence à l'action de refaire la dernière opération. On met à jour le circuit en allant chercher la dernière action dans la liste du gestionnaireHistorique.



Ce diagramme de séquence fait référence à la sauvegarde du circuit. On commence par le valider et après on le sauvegarde via persistanceManager.



Ce diagramme de séquence fait référence à l'action de valider le circuit. On doit vérifier que chaque élément rencontre les normes exigées.