Université du Québec à Montréal

Projet de session

Phase 2

Par

Jonathan Aubut
Mélanie Boisvert-Langlois
Julien Champagne
Ussel Sabbat

Travail remis à Imen Benzarti

Dans le cadre du cours INF5153

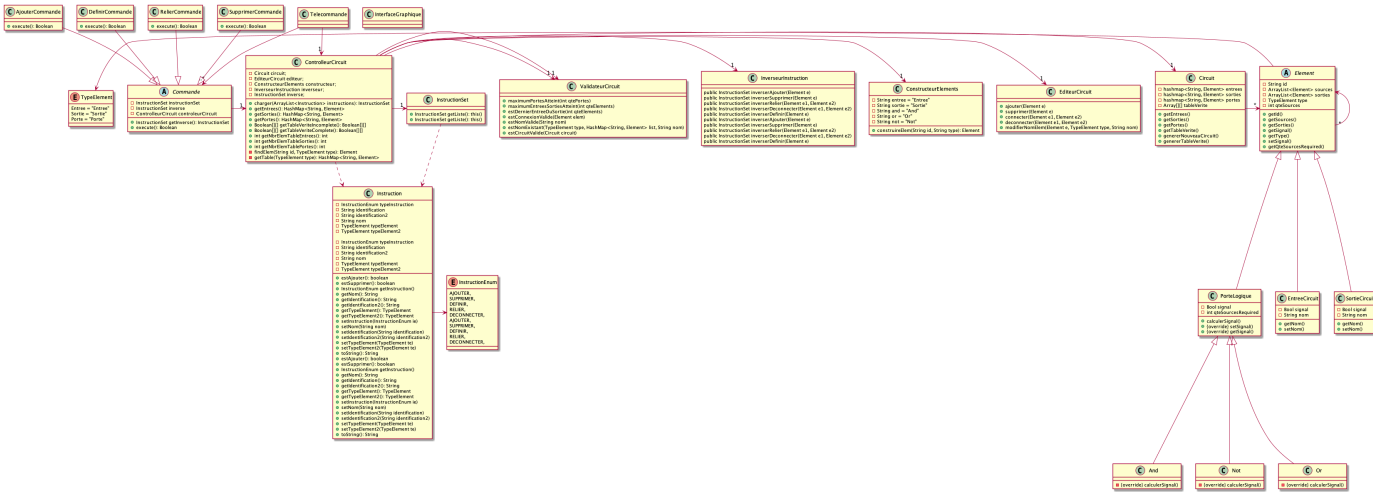
Remis le 12 décembre 2021

Table des matières

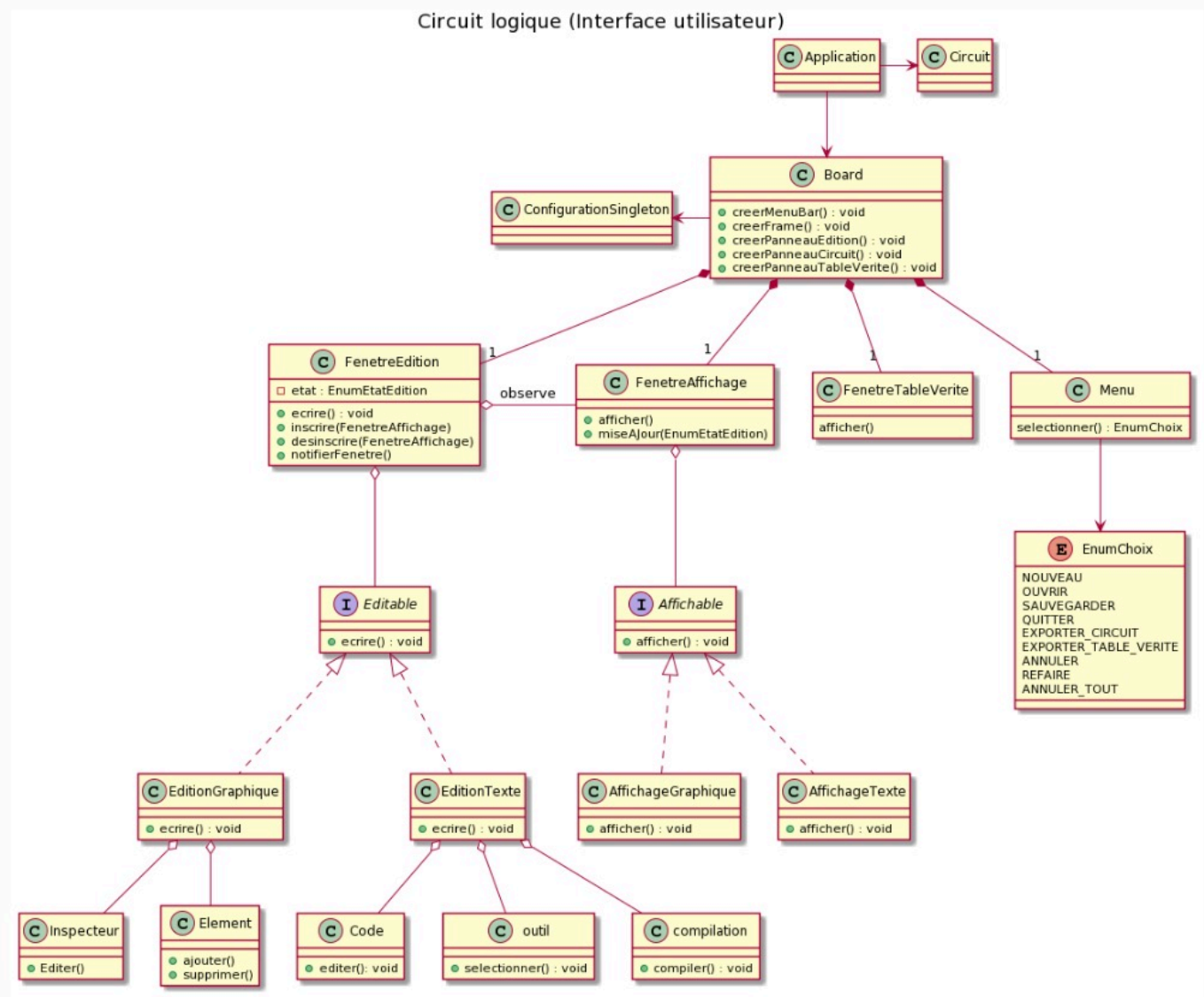
1. Modèles de conceptions

1.1 Diagramme de classe

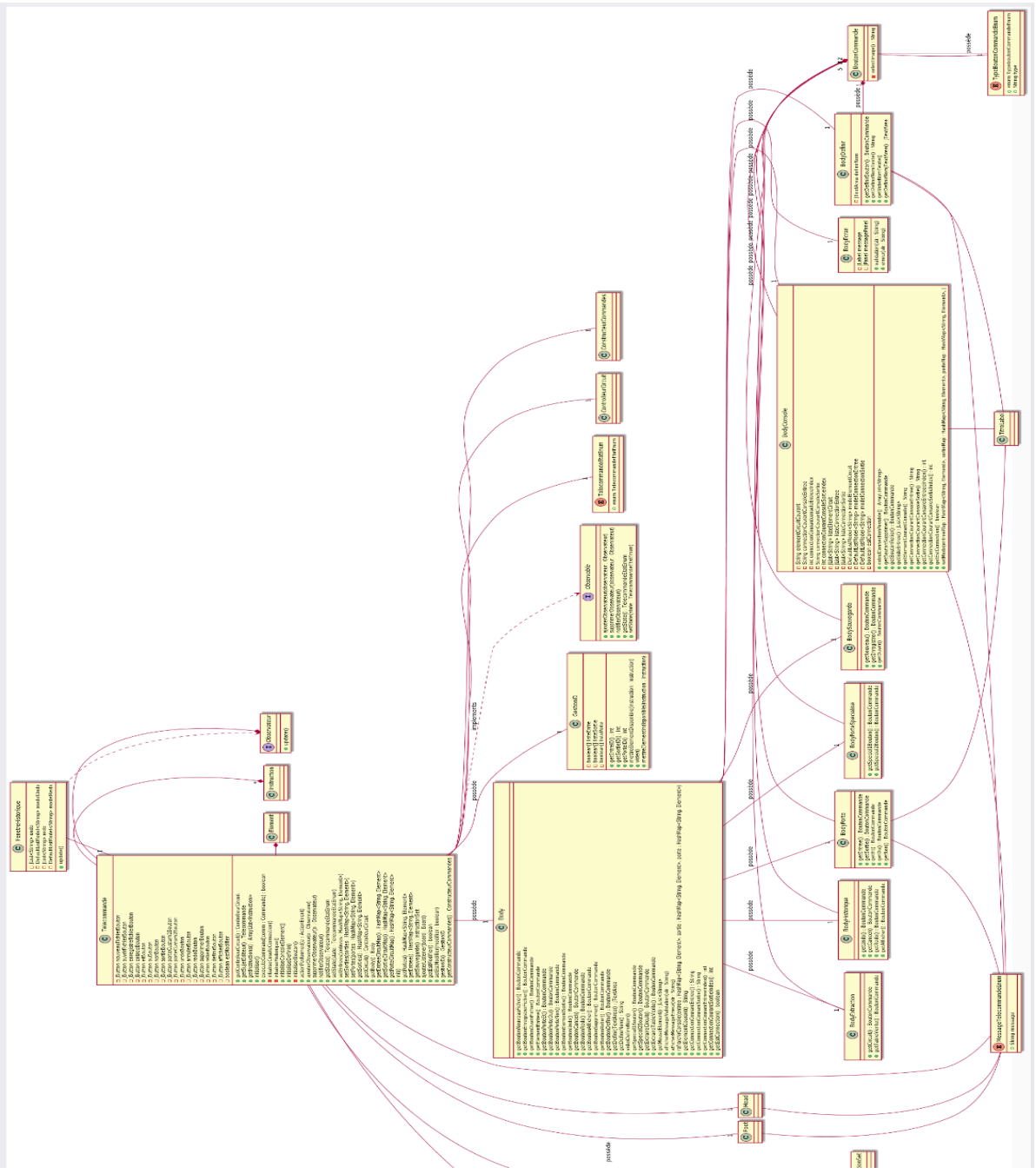
1.1.1 Circuit logique



1.1.2 Interface utilisateur

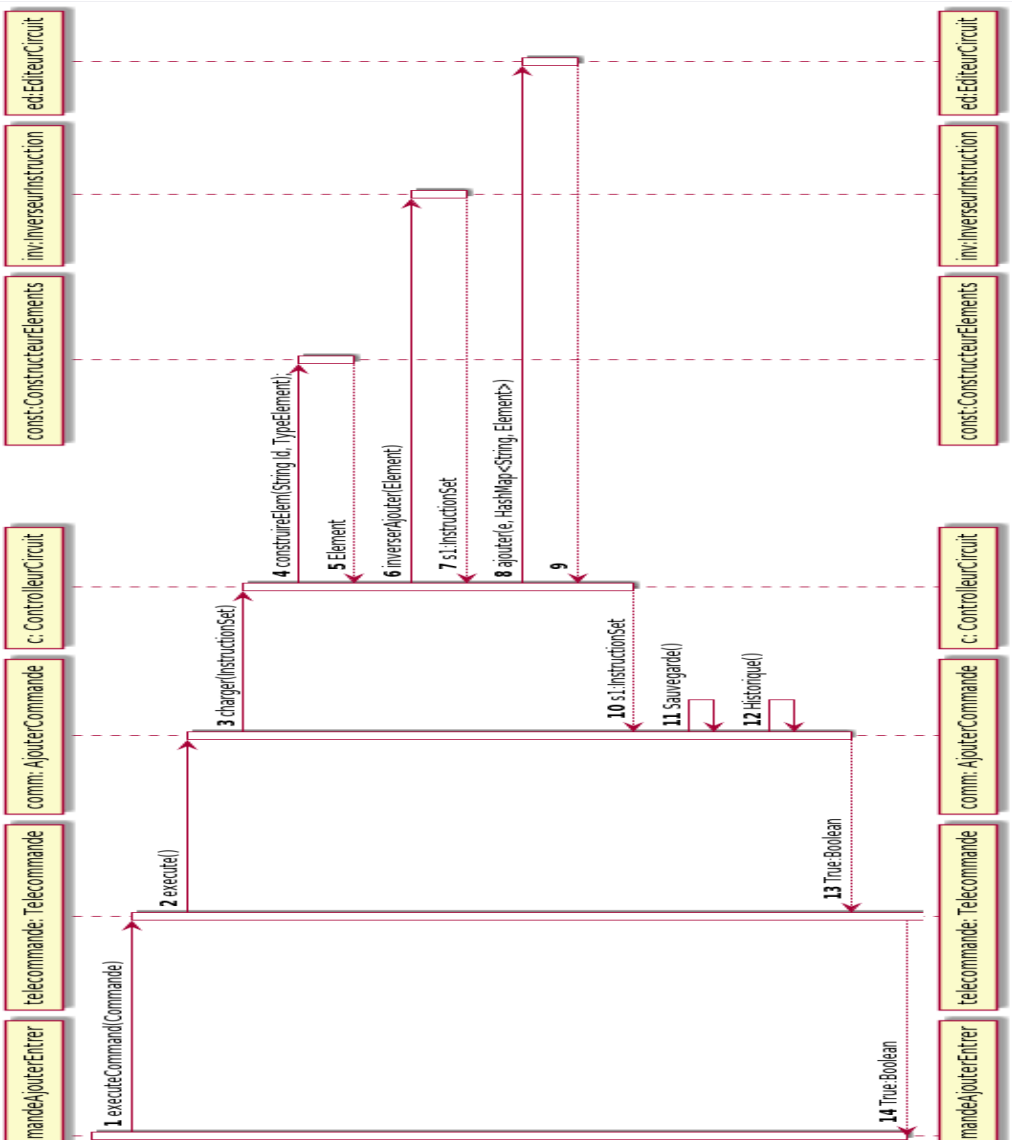


1.1.3 Télécommande

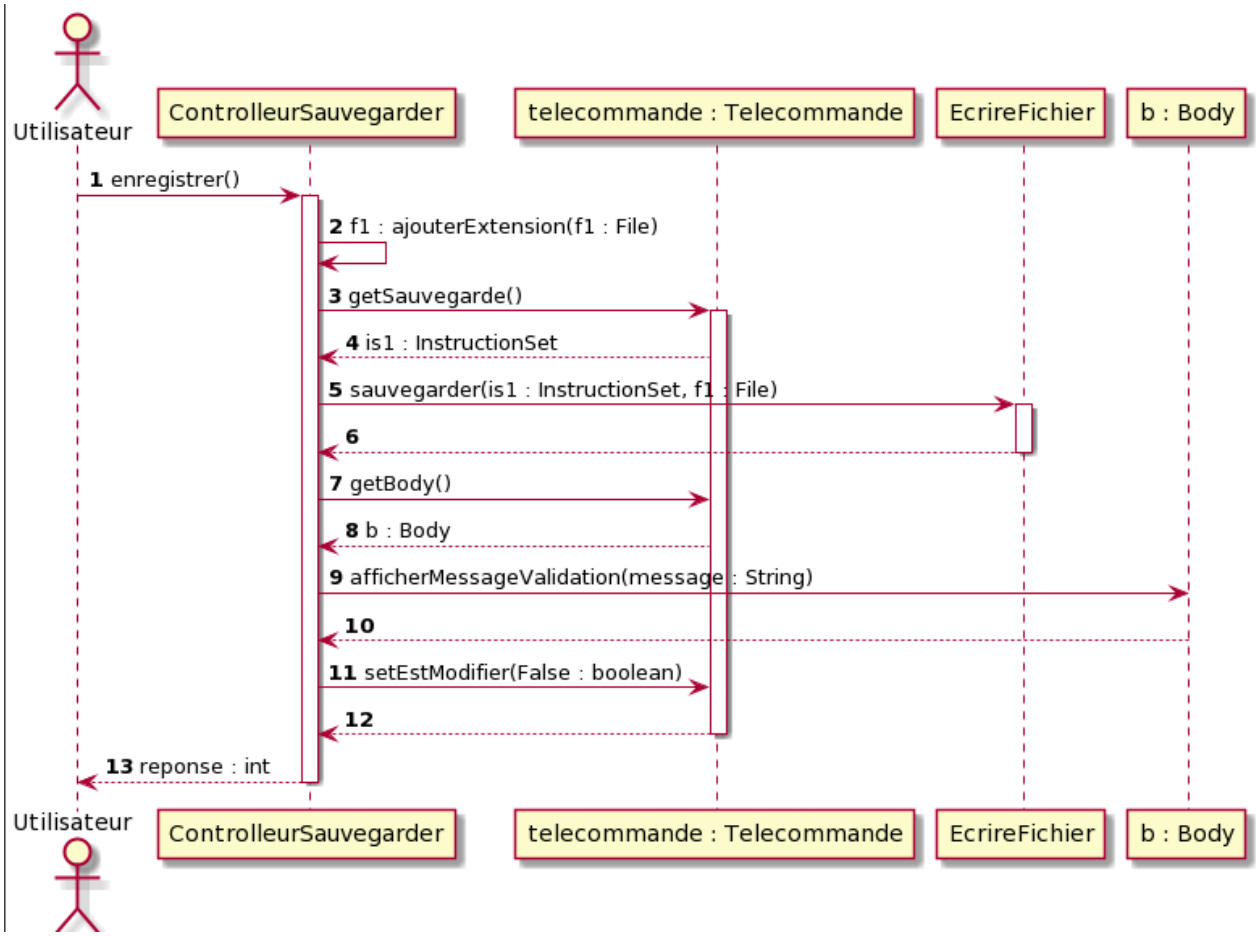


1.2 Diagramme de séquence

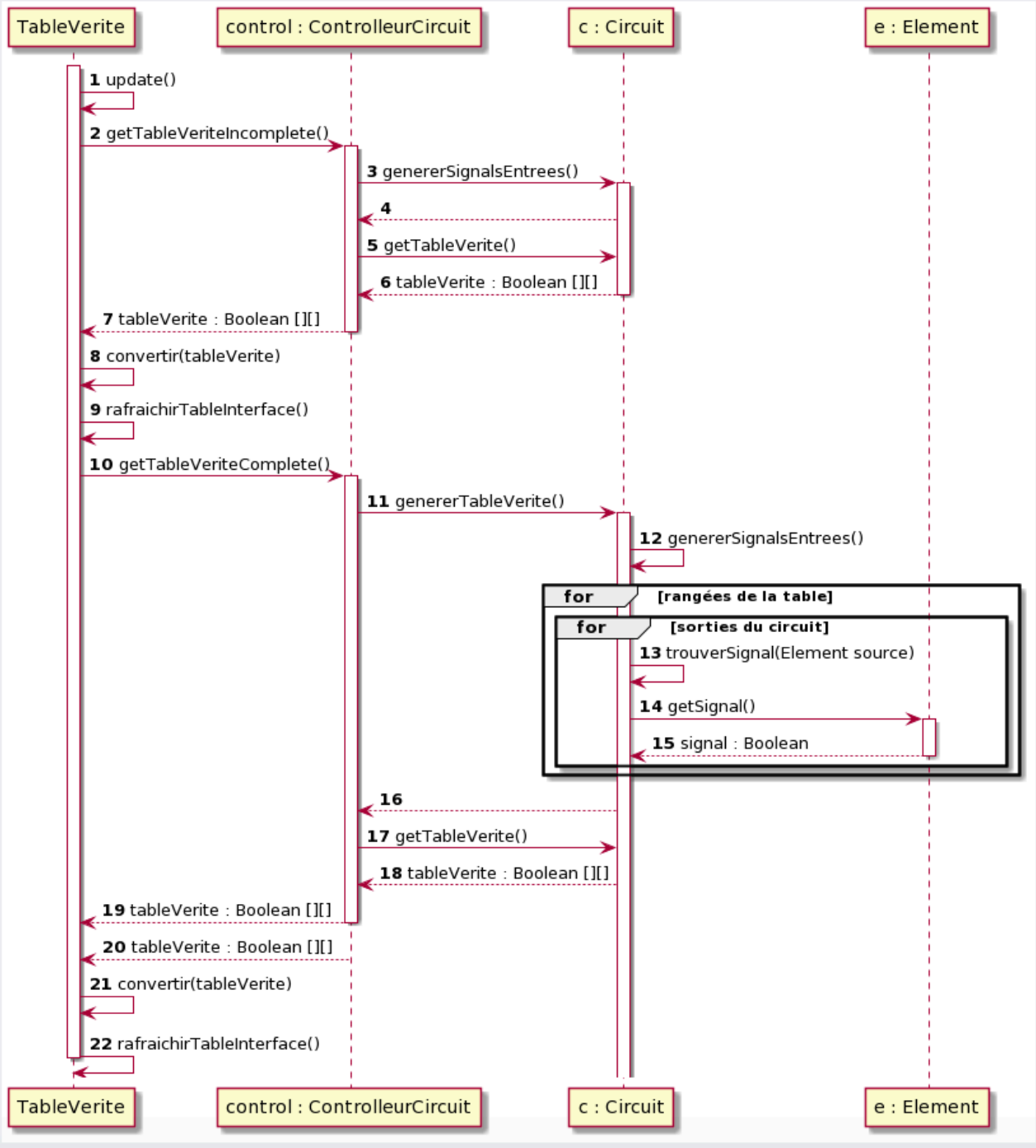
1.2.1 Ajouter commande



1.2.2 Sauvegarder



1.2.3 Table de vérité



2. Analyse critique

2.1 Forces

Une des plus grandes forces du projet a été la capacité d'adaptation de l'équipe tout au long du processus de création du logiciel. Les différentes compétences que chaque membre apportait démontrent aussi un trait marqué du projet. À chaque étape de la conception du produit, nous faisions face à divers défis et nous arrivions à y faire face. Après plusieurs itérations de l'interface graphique, nous sommes parvenu à présenter un logiciel dont l'apparence est conviviale et simple d'utilisation. Et, l'interaction entre l'interface et le circuit venait à répondre parfaitement aux exigences du client.

De plus, la conception de notre circuit exploite le polymorphisme à son plein potentiel. Les aspects communs à toutes les portes du circuit sont encapsulés dans la classe abstraite Element. Ceci permet de généraliser toutes les opérations du circuit (ses modifications comme la déconnection de deux éléments ainsi que la génération de la table de vérité). Par conséquent, notre projet est doté d'une grande flexibilité tant qu'à l'implémentation de nouveaux types de portes logiques.

Enfin, la majorité des difficultés ont été résolues grâce à nos choix de patrons de conception. Conséquemment, plusieurs forces de notre logiciel découlent de ces patrons et sont expliquées dans les sections plus bas.

2.1.1 Patron de conception observateur

L'édition graphique d'un circuit amène l'exécution d'une multitude de commande. Or, ces commandes peuvent affecter certaines zones de l'application par diverses transformations. La surveillance de ces changements réclame beaucoup de temps du processeur. Ce désavantage est accentué par le fait que le processeur est accablé même si l'utilisateur ne fait aucune action avec le logiciel. Nous avons utilisé le patron de conception observateur afin de corriger ce problème. Grâce à ce patron, le processeur n'est plus monopolisé en devant surveiller constamment les demandes de l'utilisateur.

2.1.2 Patron de conception singleton

Afin de permettre une meilleure utilisation de l'application, nous avons créé la classe Configuration pour paramétrer certaines informations et rendre le logiciel plus adaptable pour l'utilisateur. Comme il ne peut pas y avoir plusieurs instances de notre objet de configuration, nous utilisons le patron de conception singleton. Ici nous partons du principe que nous n'avons pas un environnement multiutilisateur où chaque joueur pourrait enregistrer sa propre configuration. Un patron singleton a également été utilisé pour le gestionnaire de l'historique afin d'implanter le patron Momento.

2.1.3 Patron de conception Commande

Le patron de conception commande est utilisé pour acheminer les instructions au circuit de la télécommande. L'idée était de simplifier le lien entre les demandes de l'utilisateur et les opérations effectuées sur le circuit. Une commande, peut-importe laquelle, peut aisément être effectuée grâce à la méthode execute() que les membres de cette classe partagent. Ceci permet l'ajout et l'exécution de nouvelles commandes ou boutons avec peu de modification.

2.1.4 Patron de conception MVC

Plusieurs contrôleurs sont implémentés dans notre projet. Ceux-ci permettent de factoriser et de simplifier l'envoi de messages entres nos diverses classes. Un exemple est ControlleurCircuit qui s'occupe de recevoir des instructions provenant de l'interface, de les décoder et de les envoyer aux classes permettant d'effectuer les demandes de l'utilisateur (ex : editeurCircuit pour l'ajout d'une porte).

2.1.5 Patron de conception Factory

Des classes de type Fabrique existent et ont principalement la tâche de faciliter le travail de nos contrôleurs. En effet, celles-ci s'occupent de la création de certains objets qui découlent des instructions venant de l'utilisateur. Un exemple et l'ajout d'une nouvelle porte : ControlleurCircuit demande à ConstructeurElements d'instancier une nouvelle porte. Le type de cette dernière est automatiquement déterminée grâce au polymorphisme.

2.2 Faiblesses

Au début, une faiblesse de notre projet était l'approche de l'interface graphique pour l'édition du circuit. Nous nous étions inspirés d'une conception équivalente à « Plantuml » qui offrait des possibilités multiples au niveau de la portabilité de l'interface et ne limitait pas l'usager à de simples cliques. Malheureusement, cette approche nous empêchait de respecter certaines des exigences demandées pour le logiciel par le client. C'est pourquoi nous avons dû changer de cap et avons implémenté des boutons.

Cependant, le problème persistait! Les instructions de l'utilisateur étaient toujours envoyées en groupe, ce qui empêchait l'implémentation de l'historique. Comme solution, nous nous sommes tournés vers le patron de conception commande. Ceci nous a permis une meilleure gestion entre l'interface utilisateur et le circuit. Maintenant, il devenait facile d'y intégrer l'historique.

Le fait que nous avons dû passer par trois conceptions différentes d'interface pour l'édition nous a laissé très peu de temps pour l'implémentation additionnelle de patrons de conception. Ce qui nous a contraint à faire des choix d'implémentation lors de l'affichage graphique, car le temps venait à manquer et plusieurs difficultés de conception se présentaient.

Enfin, la sauvegarde souffre d'une faiblesse de conception. Ce sont les instructions, en totalités, qui sont enregistrées. Conséquemment, notre programme peut souffrir d'une perte en performance si les instructions d'un circuit sont nombreuses et qu'il est enregistré ou chargé.