INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

Encadrants

Gildas AVOINE

Barbara KORDY

Glasir

Application au réseau STAR

Rapport de conception logicielle

Étudiants

Pierre-Marie AIRIAU

Valentin ESMIEU

Maud LERAY

Résumé

Glasir est le logiciel d'analyse d'arbres d'attaque et de défense que nous développons. Pour ce faire, nous allons améliorer ADTool, un outil d'édition d'arbres, avant de l'intégrer à notre logiciel.

Ce rapport décrit l'architecture interne de Glasir, ainsi que sa modélisation UML par le biais de diagrammes de séquence, de classes et de cas d'utilisation. Il présente également l'interfaçage entre les différents modules du logiciel.

12 février 2015

Table des matières

1 Introduction

Le projet décrit dans ce rapport consiste à développer un logiciel d'analyse informatique des AD-Trees [1]. Ce logiciel, baptisé Glasir et dont le logo est visible sur la FIGURE 1, intègrera ADTool [4], un outil d'édition d'ADTrees déjà existant. ADTool sera utilisé dans Glasir pour l'affichage, la création et l'édition des ADTrees, et subira quelques modifications destinées à améliorer son ergonomie. Glasir, pour sa part, fournira les fonctionnalités d'analyse suivantes :

- l'**Éditeur de fonctions**, qui permettra d'étendre le nombre de paramètres d'analyse qu'offre ADTool ;
- le **Filtre**, qui aidera l'utilisateur à ôter de l'ADTree les nœuds aux valuations hors d'un intervalle défini ;
- l'**Optimiseur**, qui sélectionnera le(s) meilleur(s) chemin(s) dans l'ADTree selon une certaine valuation.

Trois rapports ont déjà été rédigés dans le cadre de ce projet : celui de pré-étude [3], celui de spécifications fonctionnelles [2] et celui de planification [1]. Ces derniers ont permis de mieux définir les objectifs du logiciel ainsi que l'organisation de son implémentation. En décembre, nous avons rendu compte de l'état de notre travail par le biais de deux soutenances : l'une consacrée à la gestion du projet et à sa planification, et l'autre décrivant le projet de façon générale.

Le présent rapport de conception logicielle a pour but de détailler l'architecture de Glasir. Ce rapport commencera donc par illustrer grossièrement cette architecture à l'aide d'un prototype d'interface et d'un diagramme de cas d'utilisation, afin de montrer les actions accessibles à l'utilisateur. Puis un diagramme de classes et des diagrammes de séquence permettront de détailler beaucoup plus précisément l'agencement interne du logiciel. Une partie sera ensuite consacrée aux améliorations prévues pour ADTool, puis les limites d'utilisation de Glasir seront évoquées. Enfin, un bref résumé du planning et de l'organisation prévue sera effectué.

FIGURE 1 – Logo du logiciel Glasir.

1. Abréviation d'« Attack-Defense Trees », ou « Arbres d'Attaque et de Défense » en français.

2 Architecture globale

Afin de mieux fixer les idées quant aux fonctionnalités de Glasir, voici une présentation de son utilisation et de son interface.

2.1 Diagramme de cas d'utilisation

L'utilisateur-type de notre outil d'analyse serait un expert en sécurité connaissant déjà le formalisme des ADTrees. C'est donc l'acteur qui a été choisi pour le diagramme de cas d'utilisation de la FIGURE 2, qui représente l'ensemble des actions possibles à partir de Glasir. Afin de mieux comprendre ce diagramme, un petit rappel sur le formalisme UML s'impose. En effet, si l'on prend deux évènements A et B, les termes « étendre » et « inclure » associés aux flèches ont une signification bien particulière :

$A \overset{\text{« inclure »}}{\longrightarrow} B$ signifie que si A est réalisé, B l'est **forcément** ;
$B \overset{\text{« étendre »}}{\longrightarrow} A$ signifie que si A est réalisé, B **peut** l'être mais pas obligatoirement.

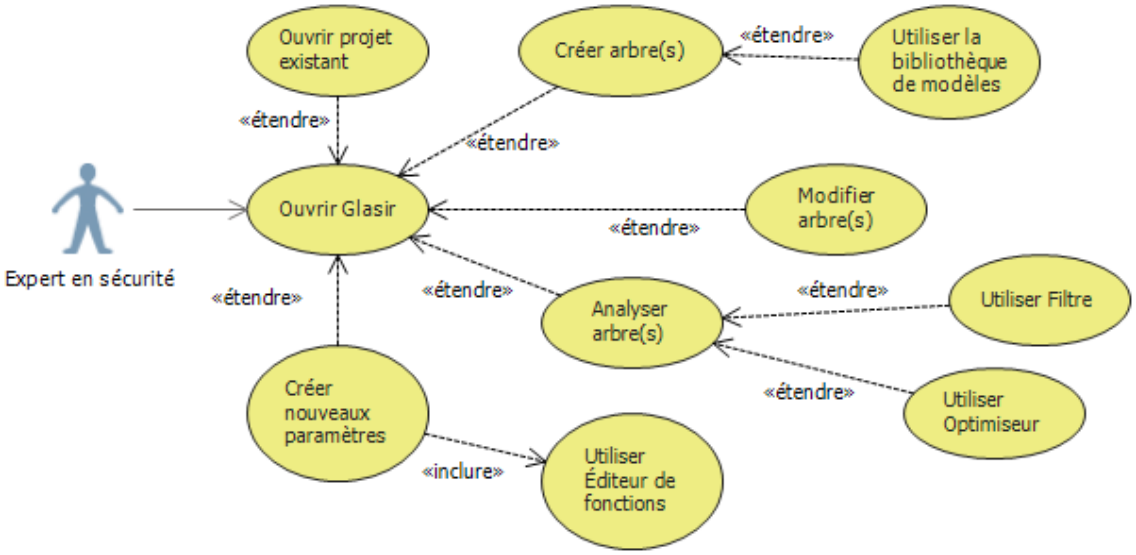

FIGURE 2 – Diagramme de cas d'utilisation de Glasir.

Les fonctionnalités de Glasir apparaissent donc maintenant clairement. Grâce à cet outil, l'utilisateur pourra effectuer les actions suivantes :

- charger un projet sauvegardé ;
- créer de nouveaux ADTrees, en utilisant ou non la bibliothèque de modèles ;
- modifier des ADTrees déjà existants ;
- analyser les ADTrees du projet en cours, à l'aide du Filtre ou de l'Optimiseur ;
- créer de nouveaux paramètres pour ADTool par le biais de l'Éditeur de fonctions.

Toutes ces possibilités seront accessibles facilement grâce à une interface utilisateur claire et fonctionnelle, dont un prototype est présenté dans la SECTION 2.2.

2.2 Prototype d'interface

La FIGURE 3 représente un prototype d'interface pour Glasir. Ce prototype n'est pas définitif, mais permet de donner un avant-goût de l'organisation générale du programme.

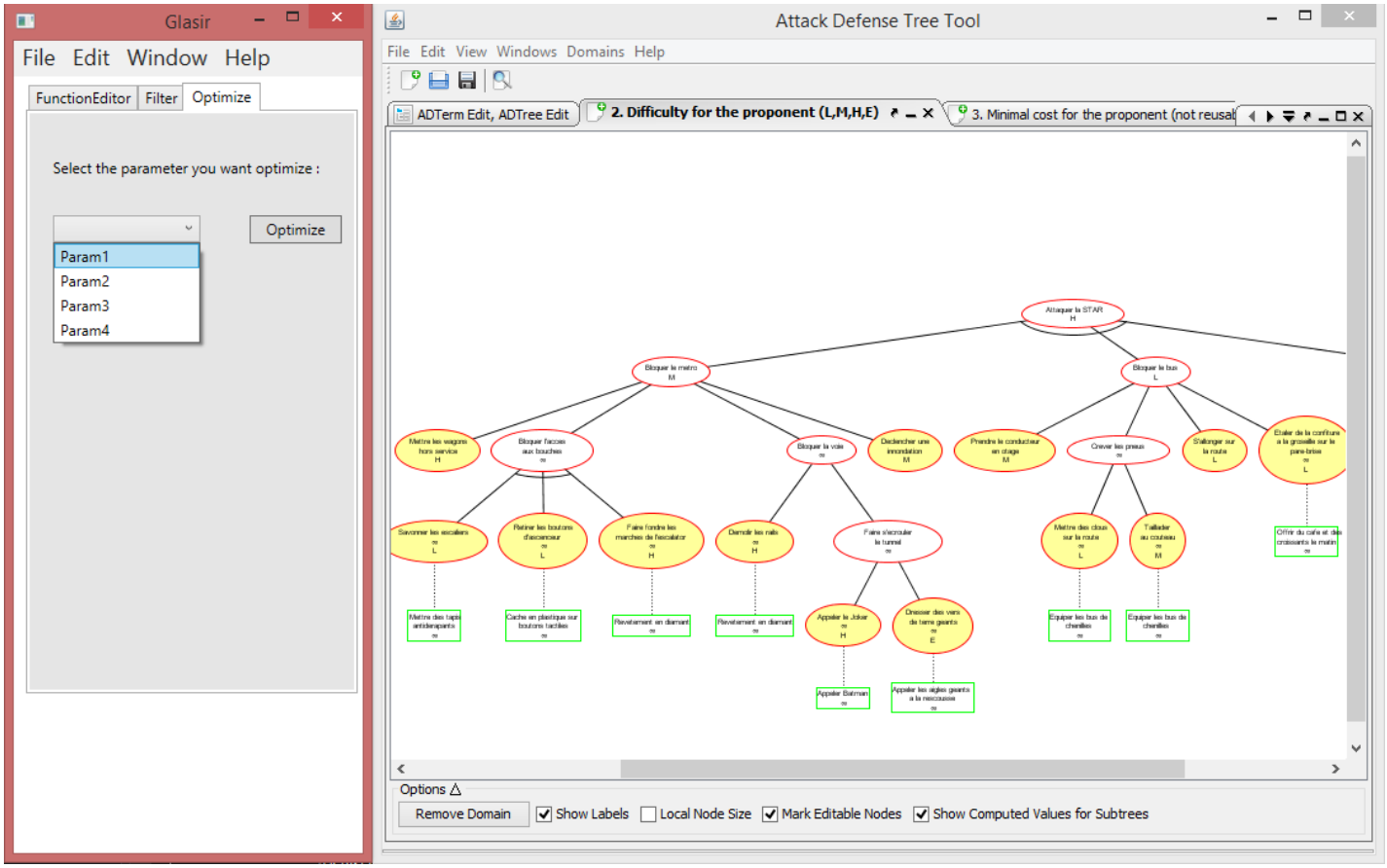

FIGURE 3 – Prototype de l'interface de Glasir.

L'interface de Glasir est scindée en deux parties distinctes : la fenêtre de Glasir à proprement parler (située à gauche sur le prototype de la FIGURE 3), et les fenêtres des instances d'ADTool (situées à droite). Au sein de la fenêtre de Glasir, les fonctionnalités du Filtre, de l'Optimiseur et de l'Éditeur de fonctions sont accessibles facilement. La barre de menu (située en haut à gauche sur le prototype de la FIGURE 3) permet de créer un nouveau projet, d'en charger un déjà existant, ou encore de sauvegarder le projet courant.

La FIGURE 4 illustre également un prototype des panneaux qui permettent d'accéder aux fonctionnalités principales de Glasir. Ces fonctionnalités sont détaillées dans les SECTIONS 3 et 4.

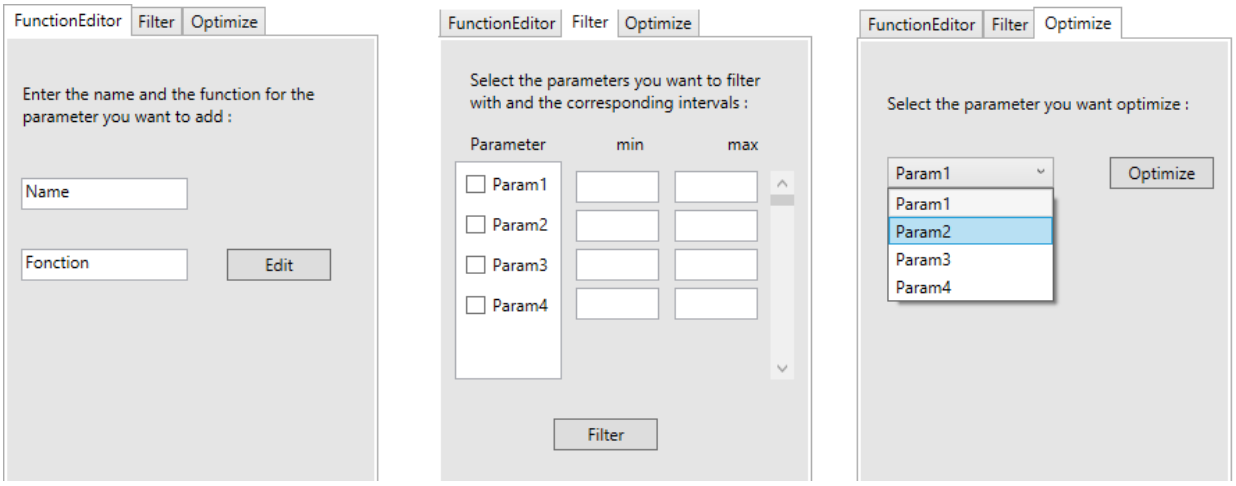

FIGURE 4 – Prototype des panneaux permettant d'utiliser les fonctionnalités de Glasir.

3 Diagramme de classes

Le diagramme de classes détaillé dans cette section permet de mieux comprendre l'architecture interne de Glasir. Afin de bien saisir ce diagramme, il est nécessaire d'indiquer qu'un ADTree est enregistré dans un fichier au format XML [2].

3.1 Classe centrale

La classe centrale du diagramme est *BigGlasir*, illustrée sur la FIGURE 5. C'est autour d'elle que s'articule l'ensemble des fonctionnalités de Glasir. *BigGlasir* dispose de méthodes permettant de charger un ADTree depuis la bibliothèque de modèles, de lancer des instances d'ADTool afin d'afficher un ADTree, mais également de créer ou de charger un projet. Cette classe possède également des modules, décrits dans la SECTION 3.3, une bibliothèque de modèles, décrite dans la SECTION 3.4, et une liste des instances d'ADTool, décrite dans la SECTION 3.2.

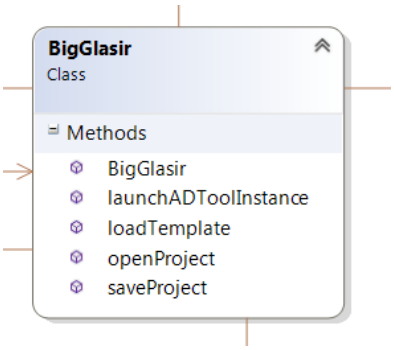

FIGURE 5 – Classe centrale gérant l'ensemble des autres fonctionnalités.

2. ADTool permet, de base, d'enregistrer un ADTree dans plusieurs formats, dont le XML : nous avons donc choisi ce format pour pouvoir travailler sur des ADTrees dans Glasir.

3.2 Instances d'ADTool

Comme mentionné dans le rapport de spécification, Glasir gère plusieurs instances d'ADTool pour permettre à l'utilisateur de travailler sur différents ADTrees en même temps [3]. La classe *ADToolInstance* contient donc le processus d'une instance d'ADTool, ainsi que le fichier de l'ADTree sur lequel l'instance porte. Ce fichier est représenté par la classe *XMLFile*, qui contient un attribut pour représenter le nom du fichier, et des méthodes permettant d'opérer sur ce fichier, comme illustré sur la FIGURE 6.

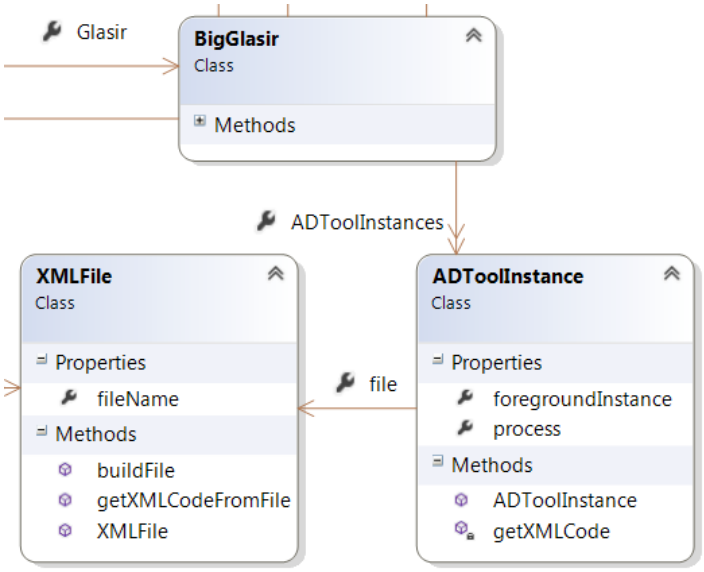

FIGURE 6 – Diagramme de classes des instances d'ADTool.

3.3 Modules

Les trois modules principaux de Glasir, c'est-à-dire le Filtre, l'Optimiseur et l'Éditeur de fonction, ont chacun leur propre classe *Filter*, *Optimizer* et *FunctionEditor*. Ces trois classes héritent de *Module*, une classe abstaite contenant les méthodes communes aux trois modules, comme par exemple la méthode permettant d'ouvrir un fichier décrivant un ADTree. De plus, *Module* contient des méthodes abstraites devant être implémentées dans chaque classe héritière, comme la méthode permettant par exemple d'obtenir le fichier résultant d'une opération appliquée par le module. La FIGURE 7 illustre l'agencement de ces classes.

3. Pour rappel, ADTool ne peut gérer qu'un seul ADTree à la fois, il n'est par exemple pas possible d'ouvrir deux ADTrees en même temps pour les comparer.

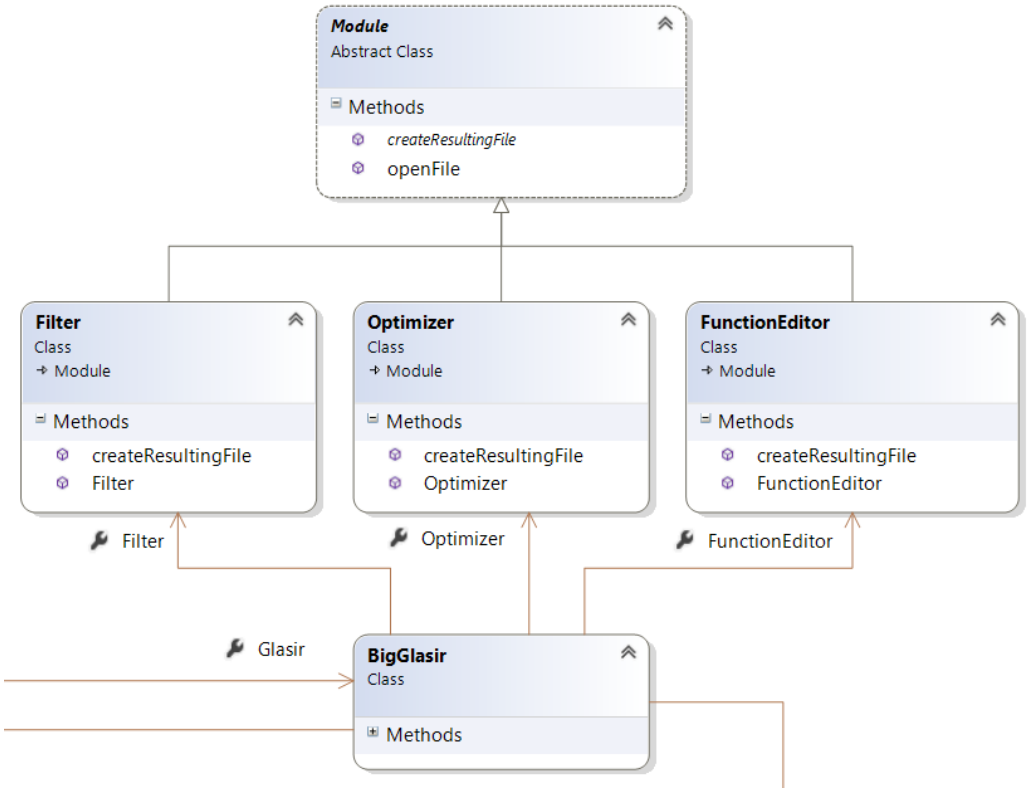

FIGURE 7 – Diagramme de classes des trois modules de Glasir.

3.4 Bibliothèque de modèles

La bibliothèque de modèles est constituée de la classe *TemplateLibrary* qui contient une liste des fichiers modèles. Des méthodes permettent d'ajouter ou de supprimer un ADTree de la bibliothèque, ou encore de charger un ADTree dans le projet courant. Les fichiers de cette bibliothèque sont là-aussi représentés par la classe *XMLFile*, comme illustré sur la FIGURE 8.

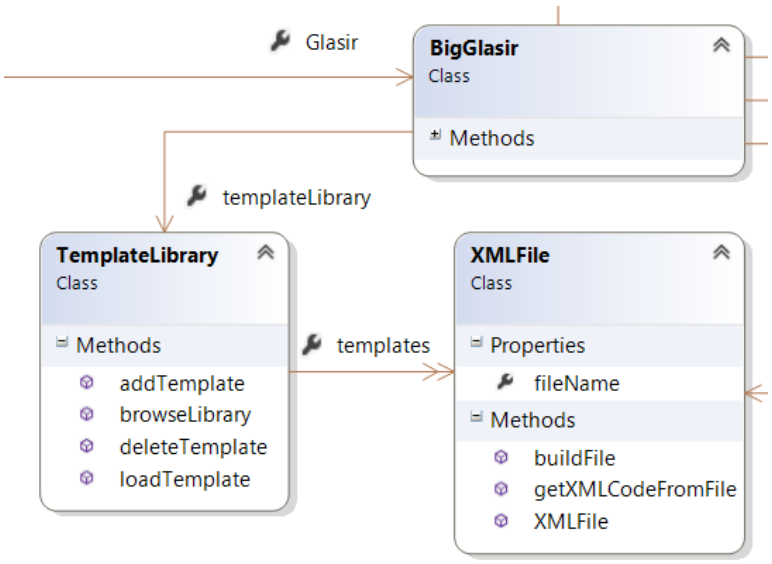

FIGURE 8 – Diagramme de classes de la bibliothèque de modèles.

4 Description séquentielle des modules

Le diagramme de séquence présenté dans cette section a pour objectif d'illustrer le fonctionnement des trois principales fonctionnalités de Glasir : le Filtre, l'Optimiseur et l'Editeur de fonction. Pour rappel, la lecture d'un diagramme de séquence se fait de haut en bas suivant un ordre chronologique, et en suivant les flèches. Les flèches correpondent aux méthodes des différentes classes présentées en tête du diagramme, l'éxecution d'une fonction est représentée par une barre verticale sur la colonne de la classe correspondante. Une fonction est appelée et rend une valeur en retour, et peut appeler d'autres fonctions lors de son éxecution.

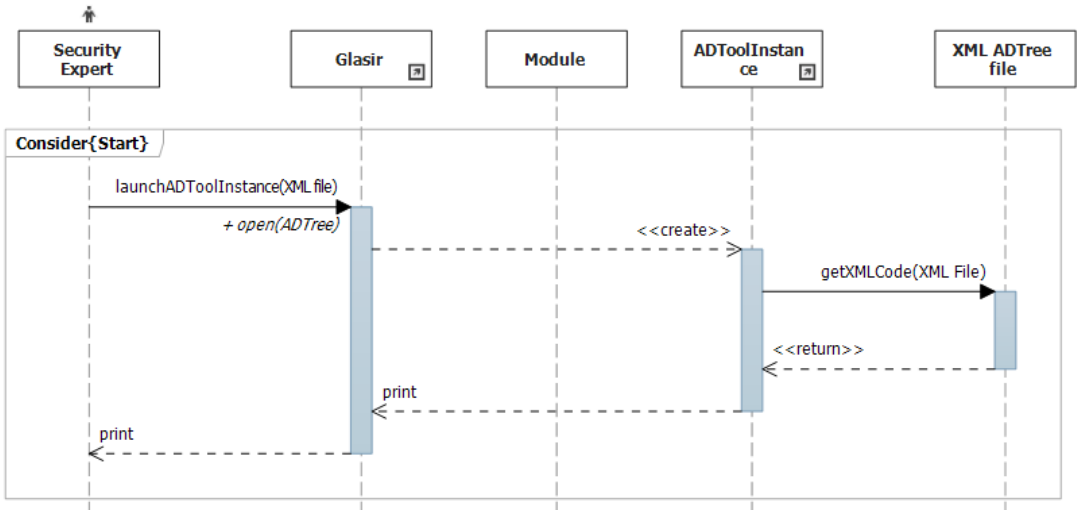

FIGURE 9 – Diagramme de séquence du démarrage d'une instance d'ADTool dans Glasir.

Dans un but de synthèse, nous avons représenté le diagramme de séquence du fonctionnement d'un module, pouvant représenter n'importe laquelle des fonctionnalités de Glasir. Les différences entre chaque module sont détaillées plus loin dans cette section. Pour mieux saisir le fonctionnement de l'un de ces modules, la FIGURE 9 illustre l'ouverture d'un ADTree dans Glasir : l'expert en sécurité demande à ouvrir un fichier XML contenant un ADTree dans Glasir. Ce dernier crée alors une instance d'ADTool, qui va chercher le code XML du fichier afin d'afficher l'ADTree.

La FIGURE 10 illustre le fonctionnement générique d'un module. L'expert en sécurité spécifie d'abord les paramètres de l'opération que va effectuer le module. Puis, une fois lancé, le module va effectuer des opérations sur le fichier de l'ADTree cible. Un fichier résultat au format XML est alors généré puis ouvert dans une nouvelle instance d'ADTool, afin d'être affiché à l'écran.

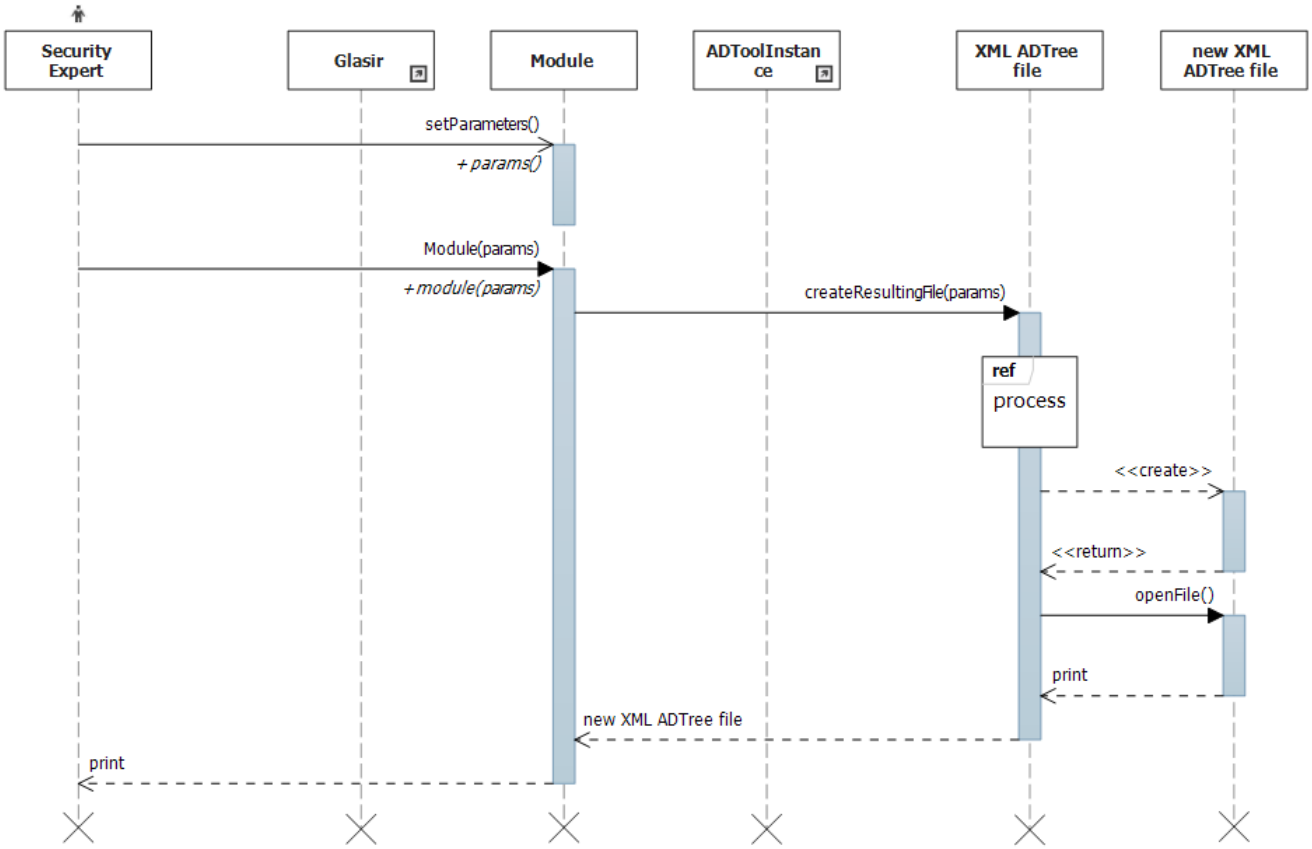

FIGURE 10 – Diagramme de séquence générique d'un module.

Le bloc nommé « process » illustre la partie différente de chaque module, tout le reste (ouverture du module avec les paramètres , et création et ouverture du fichier résultat) étant commun à tous les modules.

Dans le cas du Filtre, les paramètres passés à l'appel sont les intervalles [4] de filtrage sur les valuations de l'ADTree. Le bloc process réalise l'algorithme de filtrage, qui a déjà été présenté dans

4. ADTool propose des paramètres de valuation tels que le coût d'une attaque (un nombre réel) ou sa faisabilité (un booléen). Bien que certains de ces paramètres soient définis sur un ensemble discret, il est toujours possible de définir une relation d'ordre sur cet ensemble, et donc d'y définir des intervalles.

le rapport de spécifications fonctionnelles [2], appliqué au contenu du fichier XML de l'ADTreer.

En ce qui concerne l'Editeur de fonctions, les paramètres passés à l'appel sont la formule de calcul du paramètre à définir, et son nom. Le bloc process contient la série de calculs nécessaires à l'attribution des valeurs du nouveaux paramètres à chacun des noeuds de l'ADTree.

Enfin, le paramètre passé lors de l'appel de l'Optimiseur désigne le paramètre de valuation de l'ADTree selon lequel l'expert veut réaliser l'optimisation. Le bloc process réalise l'algorithme d'optimisation sur le contenu du fichier XML de l'ADTree, algorithme qui a déjà été détaillé dans le rapport de spécifications fonctionnelles [2].

5 Fonctionnalités supplémentaires d'ADTool

Cette section décrit la conception des fonctionnalités qui vont être ajoutées à ADTool afin de le rendre plus ergonomique.

5.1 Annulation des dernières actions

La FIGURE 11 illustre l'agencement des classes permettant de réaliser la fonctionnalité d'annulation d'ADTool. Tout d'abord, chaque action annulable possède sa propre classe définissant l'action à exécuter, ainsi que l'action opposée permettant de l'annuler. Par exemple, les actions permettant d'ajouter un nœud-fils, de changer le label d'un nœud ou encore d'ajouter un paramètre possèdent leurs classes respectives *AddChildEdit*, *ChangeLabelEdit* et *AddDomainEdit*. Ces dernières travaillent toutes sur un ADTree, représenté ici par une classe *ADTree* pour simplifier le diagramme. Ces classes implémentent également *UndoableEdit*, l'interface générique d'une action annulable. Enfin, un gestionnaire d'actions *HistoryManager* stocke les actions effectuées dans un attribut, tout en disposant de méthodes permettant d'annuler ces actions dans l'ordre inverse de celui de leur réalisation.

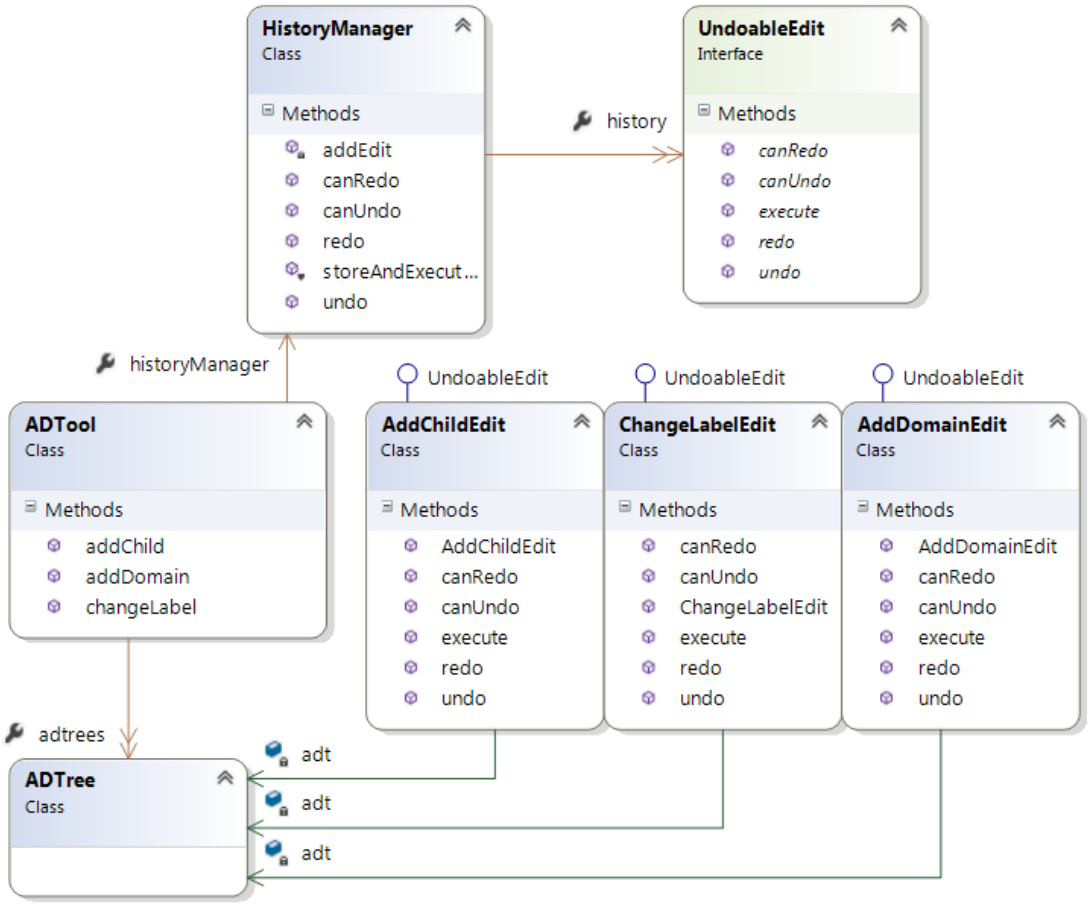

FIGURE 11 – Diagramme de classes de l'annulation des dernières actions.

5.2 Couper-copier-coller

La FIGURE 12 illustre le diagramme de classes permettant d'implémenter le couper-copier-coller dans ADTool. Les ADTrees, représentés par la classe *ADTree*, implémentent *Transferable*, l'interface permettant à un élément d'être coupé, copié ou collé. ADTool, schématisé par la classe *ADTool*, se voit doté d'un *ADTreeTransfertHandler*, une classe permettant de réaliser les actions couper-copier-coller en elles-mêmes. *ADTreeTransfertHandler* implémente *TransfertHandler*, l'interface qui permet de gérer un « presse-papier ». Ce dernier est représenté par la classe *Clipboard*, et contient l'élément coupé ou copié, c'est-à-dire un ADTree.

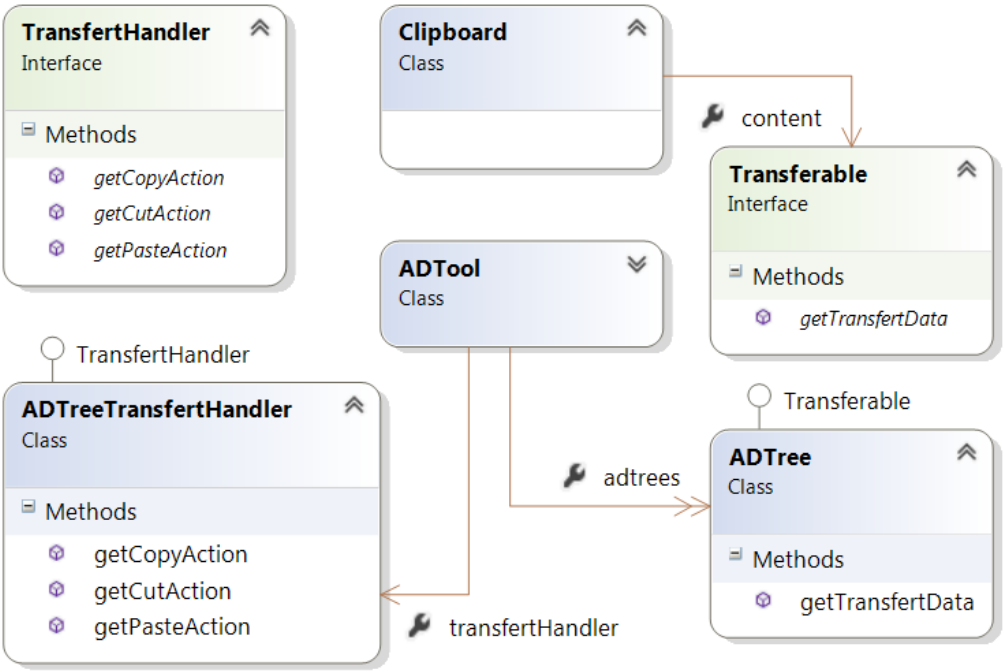

FIGURE 12 – Diagramme de classes du couper-copier-coller.

5.3 Vue globale des paramètres

La vue globale permet de présenter un condensé de l'ensemble des paramètres présents sur un ADTree. En effet, actuellement, chaque paramètre est affiché dans un onglet, ce qui ne permet pas d'avoir une vision de toutes les valuations en un seul coup d'œil. Il est donc nécessaire d'appliquer quelques changement d'affichage pour offrir à l'utilisateur un aperçu simple et condensé de toutes les valuations d'un nœud.

L'affichage d'un ADTree sous ADTool se fait au moyen des fonctions de la classe *ADTreeCanvas*. Ces fonctions utilisent une autre classe de ADTool, *DomainCanvas*, qui gère l'affichage des paramètres. Comme toutes les valuations de chacun des noeuds d'un ADTree sont présentes dans son fichier XML, nous allons gèrer l'affichage simultané de plusieurs paramètres en détectant avec les fonctions de *ADTreeCanvas* le nombre de paramètres de l'ADTree à afficher, et en appelant autant de *DomainCanvas* lors de l'affichage.

5.4 Amélioration de la représentation textuelle

Pour rappel, ADTool dispose d'une zone d'édition dans laquelle l'ADTree courant est montré sous forme textuelle. La grammaire employée dans cette fenêtre n'est pas tout à fait complète, et mérite donc d'être améliorée. Pour illustrer cela, nous allons utiliser comme exemple l'ADTree de la FIGURE 13.

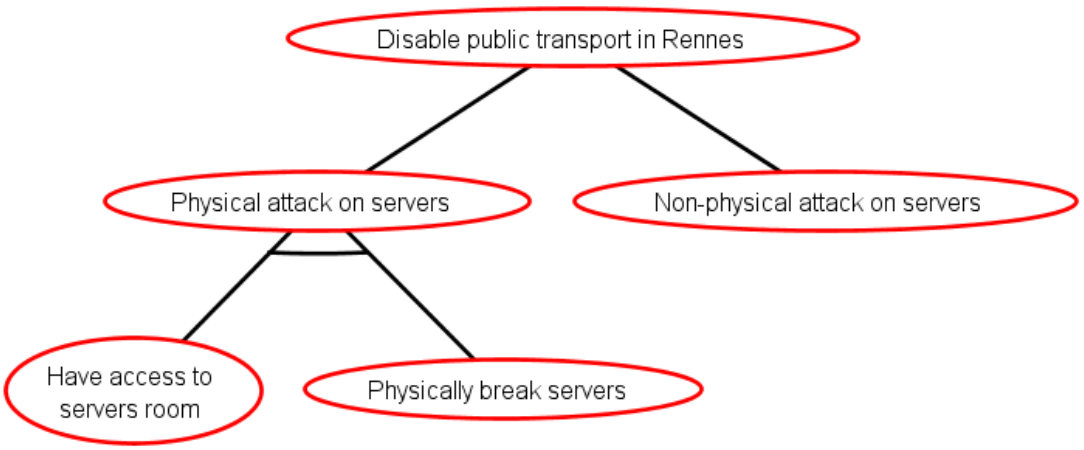

FIGURE 13 – ADTree servant d'exemple pour illustrer l'amélioration de la zone textuelle.

L'ADTree de la FIGURE 13 est ainsi actuellement représenté textuellement dans ADTool par le texte de gauche sur la FIGURE 14. On constate l'absence des labels des nœuds pères des différentes opérations (conjonction ou disjonction). Or, ces labels sont primordiaux pour avoir une bonne compréhension de l'ADTree, et surtout pour avoir la totalité des informations qu'il contient. C'est pourquoi nous souhaiterions les rajouter, d'une façon semblable à celle présentée sur la partie droite de la FIGURE 14.

```
op(
      ap(
            Have access to servers room,
            Physically break servers
      ),
      Non-physical attack on servers
)
```

```
op{Disable public transport in Rennes}
(
      ap{Physical attack on servers}
      (
            Have access to servers room,
            Physically break servers
      ),
      Non-physical attack on servers
)
```

FIGURE 14 – La grammaire actuelle (à gauche) et la grammaire améliorée (à droite).

Pour réaliser ces changements et améliorer la lisibilité de la zone textuelle, il va falloir modifier deux fichiers d'ADTool :

- *ADTNode.java*, la classe représentant un nœud au sein d'un ADTree ;
- *adtparser.jit*, le fichier générant la zone textuelle en elle-même.

À travers ces deux fichiers, il est possible de modifier la grammaire régissant la représentation textuelle des ADTrees.

6 Rétrospective

Une ébauche du cahier des charges de la conception de Glasir a été fournie en novembre, dans le rapport de pré-étude [3]. Cette première version du projet a quelque peu évolué suite aux réunions de groupe qui ont suivi, et plus encore lors de la rédaction du présent rapport.

Le cahier des charges initial prévoyait l'implémentation d'un « Guide » dans Glasir, afin d'aider un éventuel utilisateur n'étant pas familier avec les ADTrees. En effet, une limitation non-négligeable de Glasir est son accessibilité limitée : un expert en sécurité ne pourra l'utiliser que s'il dispose de connaissances de base sur la théorie des ADTrees. Il doit comprendre leur représentation, et leurs valuations, afin de pouvoir ensuite les interpréter et les analyser. Prévu sous forme de journal de quêtes, le guide aurait permis d'expliquer chaque étape d'utilisation de Glasir, de la création d'un ADTree jusqu'à son analyse. Cependant, après discussion de cette idée avec nos encadrants, il est apparu que ceci relevait plus du domaine de la recherche que de celui d'un projet étudiant, et que le travail engendré serait trop conséquent. Le guide a donc été retiré de la planification.

Il était également indiqué dans le rapport de pré-étude que les technologies utilisées seraient le C++ pour Glasir, et Qt pour son interface graphique. Finalement, vu le temps nécessaire à un apprentissage poussé de ces technologies, il a été décidé que Glasir serait codé en C# et l'interface graphique en WPF, le tout sous Visual Studio. Toutefois, ADTool restera bien en Java, et sera amélioré dans ce langage. Le choix de ces technologies pose plus de problèmes de compatibilité multi-plateformes, mais nous préférons donner la priorité au bon fonctionnement du logiciel sous Windows plutôt que de fournir une version hasardeuse qui fonctionnerait sur toutes les plateformes. Par conséquent, nous développerons l'application pour Windows, sur lequel les tests seront effectués ; d'autres tests sur d'autres systèmes d'exploitation tels que GNU/Linux ou Mac OS ne sont pas prévus.

7 Organisation et planning

L'organisation de la réalisation de Glasir a déjà été détaillée lors du rapport de planification [1]. Il y était précisé que la méthode de gestion de projet suivie serait celle du SCRUM, qui s'apparentait le plus à notre façon de travailler. Il était aussi indiqué que trois versions de Glasir allaient être livrées : deux intermédiaires (0.1 et 0.2) et une finale (1.0), dont le développement a été partitionné en tâches unitaires. Ces deux faits sont toujours d'actualité, et constitueront le fil conducteur du projet à partir de maintenant.

Cependant, le début de l'implémentation de la version 0.1 ne devait commencer qu'à la fin de la rédaction du présent rapport de conception. En réalité, nous avons pu en parallèle du rapport commencer à coder Glasir, entre autres en créant un prototype d'interface (Figure 3) sous Visual Studio. Nous nous sommes également intéressés au code source d'ADTool, et avons pu rencontrer son développeur Piotr Kordy afin de lui poser nos questions. La prochaine étape est donc, comme prévu, de livrer la version 0.1 avec son Éditeur de fonctions ainsi que les modifications prévues pour ADTool (vue globale des paramètres et amélioration de la représentation textuelle des ADTrees).

8 Conclusion

Ce rapport a débuté par un aperçu global des possibilités offertes par Glasir suivi d'une présentation d'un prototype d'interface. Cette première partie a permis de mieux comprendre la conception générale du logiciel. Les trois parties suivantes, contenant différents diagrammes de classes et de séquence, ont donné plus de détails sur la structure de Glasir et sur les savoir-faire utilisés. Ainsi, l'implémentation des fonctionnalités principales de Glasir mais aussi les modifications apportées à ADTool ont été explicitées. Nous avons ensuite fait le point sur la progression du projet par le biais

d'une rétrospective, qui a été l'occasion de justifier quelques petits changements de programme. Ces derniers ont alors été pris en compte dans notre organisation, ce qui nous permet donc de continuer l'implémentation de Glasir que nous avons amorcée lors de la rédaction de ce rapport. La livraison de la version intermédiaire 0.1 est en effet prévue pour le 20 mars 2015.

Références

[1] Pierre-Marie Airiau, Valentin Esmieu, Hoel Kervadec, Maud Leray, Florent Mallard, and Corentin Nicole. Glasir - Rapport de planification, décembre 2014. Rapport de spécification fonctionnelle, projet de quatrième année, INSA de Rennes.

[2] Pierre-Marie Airiau, Valentin Esmieu, Hoel Kervadec, Maud Leray, Florent Mallard, and Corentin Nicole. Glasir - Rapport de spécifications fonctionnelles, novembre 2014. Rapport de spécifications fonctionnelles, projet de quatrième année, INSA de Rennes.

[3] Pierre-Marie Airiau, Valentin Esmieu, Hoel Kervadec, Maud Leray, Florent Mallard, and Corentin Nicole. Rapport de pré-étude, octobre 2014. Rapport de pré-étude, projet de quatrième année, INSA de Rennes.

[4] Barbara Kordy, Piotr Kordy, Sjouke Mauw, and Patrick Schweitzer. ADTool : Security Analysis with Attack–Defense Trees. In Kaustubh R. Joshi, Markus Siegle, Mariëlle Stoelinga, and Pedro R. D'Argenio, editors, *QEST*, volume 8054 of *LNCS*, pages 173–176. Springer, 2013.

INSA Rennes
20 Avenue des Buttes de Coësmes
CS 70839
35708 Rennes Cedex 7

Tél. +33 (0) 2 23 23 82 00
Fax +33 (0) 2 23 23 83 96

www.insa-rennes.fr

INSA

UNIVERSITÉ EUROPÉENNE DE BRETAGNE

Commission des Titres d'Ingénieur

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE