Projet d'application *InterfaceBioInfo*

—

Rapport final

Sommaire

1. Introduction

1. présentation de PINT

Au cours des dernières années, l'équipe MeForBio (Méthodes Formelles pour la Bio-Informatique) de l'IRCCyN a développé une modélisation novatrice pour représenter et analyser les réseaux de régulation génétique de grande ampleur : les Frappes de Processus. Des algorithmes efficaces d'analyse de ce modèle ont vu le jour, et ont été implémentés dans l'outil pint.

Les frappes de processus visent ainsi la simplicité et l'efficacité afin de permettre la vérification formelle. La biologie des systèmes est le principale champ d'application, avec notamment la modélisation des réseaux biologiques de régulation (BRN). Des travaux récents portent sur la vérification formelle des BRN concernant plus de 100 composants.

Les principales caractéristiques et possibilités des frappes de processus sont les suivantes:

- les sortes regroupent un ensemble fini de processus;
- à un moment donné, un et un seul processus de chaque sorte est présent;
- les actions sont de la forme "processus de a_k frappe le processus b_i pour le faire passer à b_j" où a et b sont des sortes.
- listing des états stables (ou points fixes)
- analyse d'accessibilité
- Paramètres temporels et stochastique pour les actions.

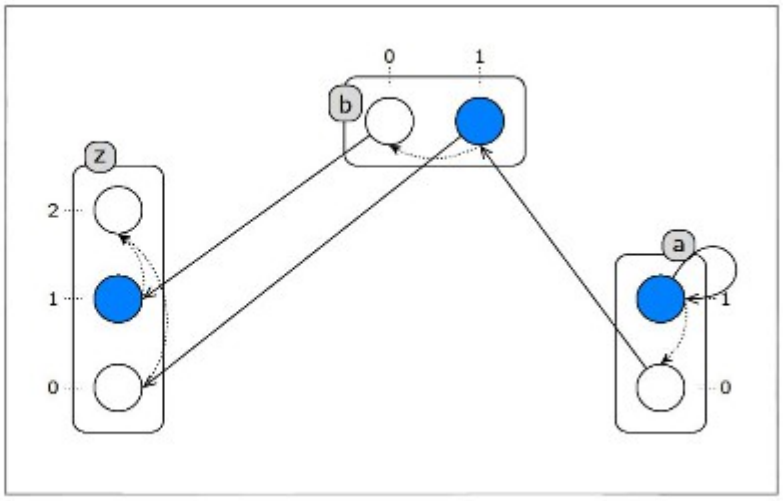
Pint rassemble divers programmes manipulant des modèles de processus de frappe (simulation stochastique, recherche des états stables, exportation / importation pour divers formats, etc.)

2. présentation du projet

L'outil pint ne prend en entrée, pour le moment, que des fichiers textes au format .ph . L'absence d'interface graphique le réserve donc à des informaticiens et des bio-informaticiens. Afin de favoriser la diffusion de l'outil et de rendre son utilisation plus facile pour des biologistes, l'équipe MeForBio souhaite doter l'outil d'une interface graphique.

La lecture d'un fichier .ph est en effet bien moins intuitive qu'une représentation graphique, comme en atteste l'exemple ci-dessous (la représentation graphique est extraite des slides de la présentation de Loïc Paulevé au LIP6/MoVe seminar).

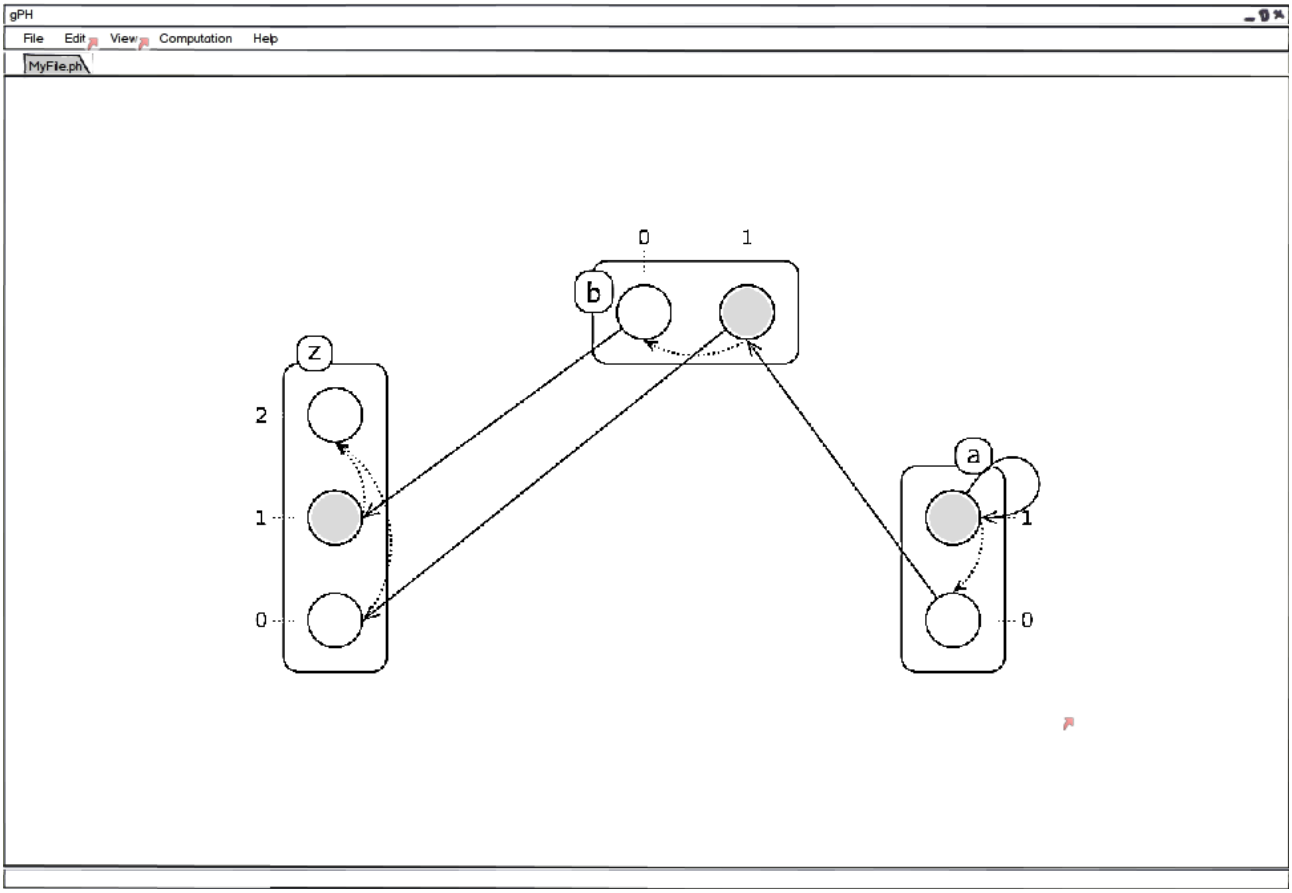
```
process a 1
process b 1
process z 2

action a 0 -> b 1 0
action a 1 -> a 1 0
action b 0 -> z 1 2
action b 1 -> z 0 2

initial_state a 1, b 1, z 1

```



L'outil développé se doit d'être multi-plateforme (Windows, MacOSX, systèmes Linux) afin d'être utilisable par le plus grand nombre. De plus, les langages et bibliothèques utilisés pour le développement de l'outil devront avoir des licences suffisamment permissives pour ne pas entraver sa diffusion.

3. solution envisagée

Notre solution se présente sous la forme d'une fenêtre. Celle-ci possède une barre de menu permettant d'accéder aux différentes fonctions.



Mockup de la fenêtre

L'utilisateur, en passant par les différents menus peuvent réaliser des actions de bases sur les fichiers ph (qui contiennent les frappes de processus): les ouvrir, les enregistrer...

Une fois ouvert, une représentation graphique est déployée dans un onglet.

L'utilisateur peur alors utiliser les fonctions de pint ainsi que modifier le fichier ph ou sa représentation graphique.

2. Outils

1. Frameworks et librairies utilisés

1. boost

Boost est un très vaste ensemble de bibliothèques C++ qui étendent les fonctionnalités du langage. Nous avions prévu d'utiliser 5 des bibliothèques de boost, à savoir boost::filesystem, boost::exception, boost::spirit, boost::smartptr et boost::units.

boost::smartptr permet d'utiliser en C++ des pointeurs avec compteur de référence et évite les fuites mémoires dues à d'éventuelles erreurs de programmation. La prise en main était modérément aisée mais la documentation sur le sujet est très abondante. Au final, nous avons utilisé les smart pointers à de très nombreuses reprises et en sommes très contents.

Prise en main (Antoine): environ deux heures

boost::filesystem nous a pleinement donné satisfaction. La prise en main était aisée et la documentation très accessible. Nous n'avions toutefois que des besoins très basiques en ce qui concerne l'accès au système de fichiers et il est plutôt rassurant que nous n'ayons pas rencontré de problèmes significatifs de ce côté.

Prise en main (Antoine): moins d'une heure

boost::exception permet d'étendre les fonctionnalités de C++ vis à vis des exceptions, notamment pour faciliter l'ajout d'informations complémentaires à un objet excpetion après son lancement (via le mot-clé throw). Notre application utilise assez peu d'exceptions et il semble que l'on aurait pu faire aussi bien sans cette bibliothèque. La prise en main était toutefois assez rapide.

Prise en main (Antoine): environ une heure

boost::units est une bibliothèque qui propose un framework de tests unitaires en C++. C'est d'autant plus important que cette fonctionnalité est complètement absente du langage. La prise en main de boost::units n'est pas aisée en raison de la documentation un peu légère et parfois périmée. Le

fonctionnement de la bibliothèque fondé sur des appels de macro préprocesseurs n'arrange rien. Nous avons fini par abandonner boost::units au profit de QTest, le framework de test fournit avec Qt qui est bien mieux documenté.

Prise en main (Antoine): environ deux heures

boost::spirit est un générateur de parsers qui se base sur la combinaison de parsers simples. La bibliothèque fait un usage avancé des surcharges opérateurs et de la programmation générique, de telle sorte qu'une grammaire décrite en C++ en utilisant boost::spirit ressemble à la même grammaire sous forme EBNF (pratique et facile à lire donc). Le problème est que boost::spirit pour être utilisée de manière complète nécessite l'utilisation de plusieurs autres bibliothèques de boost elles aussi très complexes (boost::lamba, boost::bind et surtout boost::phoenix). Enfin, l'usage extrême de la programmation générique dans le code de la bibliothèque rend la correction d'erreurs de programmation détectées à la compilation TRES difficile. En effet, la moindre erreur de compilation prend alors plusieurs dizaines d'écran de haut en raison des noms de types génériques extrêmement longs impliqués.

Prise en main (Antoine): environ quinze heures (avant abandon et remplacement par axe)

2. graphviz

Graphviz est une bibliothèque écrite en C qui permet de calculer des représentations graphiques de graphes (au sens mathématique du terme) selon différents algorithmes de positionnement tels que dot et neato. La bibliothèque est une référence dans le domaine et nous sommes satisfaits de ses fonctionnalités. Cependant, elle est complexe à maîtriser (en raison de sa polyvalence) et surtout très peu pratique lorsqu'on l'utilise comme bibliothèque (et non en ligne de commande). L'API n'existe qu'en C (pas en C++) et est **extrêmement mal documentée** : le guide de référence est très incomplet et la documentation générée automatiquement ne donne aucune information sur la signification et l'utilisation des structures de données manipulées par la bibliothèque. Le tout devient encore plus pratique quand les structures de données en question ont des champs aux noms aussi évocateurs que « u » ou « bb ».

Prise en main (Antoine): environ 8 heures

3. axe

Axe est un générateur de parser écrit en C++11 dont les fonctionnalités ressemblent fortement à celle de boost::spirit décrit précédemment. Sa conception est cependant plus adaptée à un usage réel (et non à un exercice de style technique). Nous sommes très contents de ce choix que nous avons d'ailleurs recommandé à un autre groupe de PAPPL qui en semble lui aussi très content.

Prise en main (Antoine): environ 1 heure (en bénéficiant de l'expérience acquise avec boost::spirit)

4. qt

Qt est un framework multi-plateforme qui est largement utilisé pour développer des applications avec une interface utilisateur graphique (GUI).

Ce framework est développé par un projet open source, le projet Qt, impliquant à la fois les développeurs individuels ainsi que les développeurs de Nokia, Digia, et d'autres entreprises intéressées par le développement de Qt.

Qt respecte le standard C++, mais fait un usage intensif d'un générateur de code spécifique (appelé Meta Object Compiler, ou moc) en collaboration avec plusieurs macros pour enrichir le langage. Ceci explique la création de "fichiers moc" lors de la compilation.

Distribué sous les termes de la GNU Lesser General Public License, Qt est un logiciel libre et open source. Toutes les éditions supportent une grande diversité de compilateurs, donc le compilateur GCC.

Ainsi, Qt est une bibliothèque C++ qui permet de développer des interfaces graphiques indépendamment de la plate-forme ciblée. Nous l'avons choisie (plutôt que son concurrent GTK) pour sa documentation plus abondante et aussi car GTK demande un serveur X11 pour fonctionner sous Mac OSX (qui fait partie des plate-formes ciblées).

Prise en main (Karlotcha) : 10 heures

2. Outils de développement

1. SVN

SVN est un système de gestions de versions. Il permet de centraliser et archiver les différentes versions du code et des documents en rapport avec le projet. C'est un outil que nous sommes habitués à utiliser et dont nous sommes toujours aussi satisfaits.

Prise en main : déjà connu

2. Virtual box

Virtualbox est un logiciel qui permet de créer et utiliser des machines virtuelles, ce qui nous a permis de développer notre projet d'application dans des environnements Linux (à savoir Ubuntu 11/10 et Arch Linux). Il est évidemment possible de se passer de cet outil pour peu que l'on dispose d'une machine capable de booter sous linux.
Prise en main : 3 heures chacun (en comptant l'installation de l'OS)

3. QtCreator

QtCreator est un IDE (integrated development environment).

Il se compose d'un éditeur de texte destiné à développer en C++/Qt . Il offre ainsi des outils comme la coloration syntaxique, l'auto completion, l'aide contextuel.

Il peut également intégrer également un debugger et un compilateur, mais finalement toute la partie compilation s'effectuait sur Virtualbox dans un environnement Linux.

Prise en main (Karlotcha) : 3 heures (avant de passer sur Virtualbox)

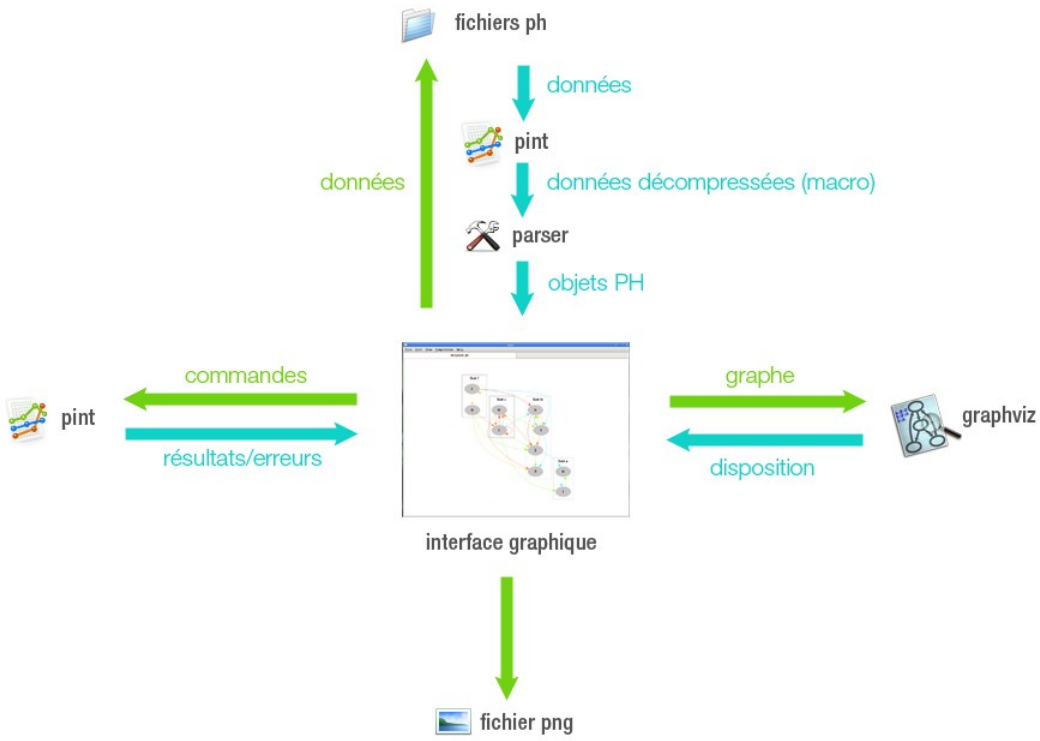
4. Trac

Trac est un système de wiki et de gestionnaire de bugs couplés à SVN. L'outil s'est avéré assez peu utile au projet puisqu'il a simplement servi à maintenir à jour quelques pages sur le processus de compilation et de test de notre projet. Le gestionnaire de bugs a uniquement servi à rapporter à Loïc Paulevé des coquilles dans la page dédiée web dédiée à Pint. La raison à cela est que l'équipe

était suffisament petite (et en contact direct) pour ne pas ressentir le besoin de formaliser nos demandes/remarques via des tickets.

3. Travail effectué

1. architecture



Le schéma ci-dessus récapitule les différents échanges entre modules et bibliothèques qui se produisent lors de l'utilisation de l'application. Le code source de l'application est réparti en différents dossiers qui correspondent à ces fonctionnalités :

- Le package « ph » contient les classes qui servent à construire la représentation en mémoire d'un fichier PH (les objets PH dans le diagramme ci-dessus). Il contient les classes PH, Sort, Process et Action.

- Le package « io » qui contient les classes concernant l'entrée et la sortie de fichiers. La classe IO gère les entrées/sorties au niveau le plus bas tandis que PHIO contient le parser de fichiers PH et la fonction d'export en .png

- Le package « ui » contient les classes relatives à l'interface graphique de l'application, à savoir MainWindow (la fenêtre principale et ses menus) et MyArea (le contenu d'un onglet ouvert)

- Le package « gfx » contient les classes qui permettent de fabriquer la représentation graphique d'un Process Hitting qui est synthétisée dans la classe PHScene. Les classes GSort et GProcess aggrègent un objet Sort/Process (respectivement) et les données

d'affichage correspondantes

- Le package « gviz » contient des structures de données qui permettent de manipuler la structure de graphes utilisée par graphviz selon un modèle orienté objet (graphviz étant écrit en C, il ne le permet pas). Ce package ne part pas du principe que l'on va représenter des process hitting et a une valeur très générale.

- Le package « test » contient les classes relatives aux tests unitaires du programme.

2. parser de fichier ph

Le parser généré par la bibliothèque axe se trouve dans la classe PHIO qui ne contient que des méthodes statiques. La méthode parseFile() se charge d'exécuter l'utilitaire phc de pint pour calculer un dump (explications ci-dessous) sur lequel agit ensuite le parser de la méthode parse(). La valeur de retour de parseFile() est un smart pointer vers un objet de la classe PH qui représente le Process Hitting ainsi chargé (avec ses sorts, ses actions, etc.).

Le parser ne travaille pas directement sur les données du fichier .ph afin d'éviter autant que possible afin d'éviter de répliquer du travail déjà existant dans pint qui dispose de son propre parser de fichiers ph (en OCaml). En effet, la syntaxe des fichiers ph décrite sur le site de pint (processhitting.wordpress.com/doc/language/) inclut plusieurs commandes qui sont en faites des macros dont le résultat peut être exprimé en utilisant les expressions les plus simples du langage. Les « dumps » livrés par l'utilitaire phc sont en fait des fichiers ph valides et équivalents aux fichiers d'entrée après exécution des macros. En travaillant sur de tels dumps, nous avons pu limiter notre parser à une grammaire plus simple sans pour autant avoir perdu en fonctionnalité.

La déclaration du parser présente peu de subtilités. Il est construit comme le veut la bibliothèque axe en combinant des parsers simples entre eux jusqu'à aboutir à un parser qui reconnaît le langage souhaité. Il est intéressant de remarquer que du point de vue de l'utilisateur, axe ne sépare pas les phases d'analyse lexicale, syntaxique et sémantique - ce qui est particulièrement pratique pour assigner des actions sémantiques qui correspondent à la création de l'objet PH au fur et à mesure que le dump est parsé.

3. visionneuse de fichiers ph

La partie « représentation graphique » d'un fichier ph est au cœur des objectifs de l'application et c'est également le sujet le plus complexe que nous avons abordé lors de ce projet. Représenter un fichier ph de manière semblable à l'illustration présente dans nos maquettes et l'introduction de ce rapport demande de calculer une disposition qui assure une bonne lisibilité des sorts et des actions (éviter les croisements etc.). Ce sujet étant bien trop complexe pour être traité au cours de notre projet, nous nous en sommes remis à la bibliothèque graphviz dont le rôle est de calculer de telles dispositions pour toute structure assimilable à un graphe mathématique.

L'interaction de notre application avec graphviz se déroule en 3 étapes qui sont déclenchées par un appel à la méthode render() de la classe PH :

1. On représente notre process hitting par un graphe mathématique. Concrètement, la méthode toGVGraph() de la classe PH crée un objet de la classe GVGraph (qui est, rappelons le, notre surcouche objet par dessus le modèle de graphe utilisé par graphviz). Le graphe que l'on définit a pour nœuds les process (du Process Hitting que l'on veut représenter) et pour arcs les actions et les sauts. Les process d'une même sort sont regroupés au sein d'un sous graphe que graphviz appelle "cluster" afin de se retrouver côte à côte dans la disposition finale.

2. On demande à graphviz de faire son travail. La méthode toGVGraph() se termine par un appel à la méthode applyLayout() de la classe GVGraph qui fait elle-même appel aux routines de graphviz. Le graphe que l'on a fournit a graphviz est alors modifié par celui-ci et se voit attribué un grand nombre de propriétés mal documentées qui décrivent la disposition calculée. Les méthodes nodes(), clusters() et edges() de la classe GVGraph extraient ces informations vers des structs simples qui ne contiennent que les données dont on aura besoin pour dessiner le process hitting (tailles, positions), non sans avoir appliqué le changement de résolution (dpi) entre graphviz et Qt et en ayant converti les coordonnées calculées vers un repère orienté dans le bon sens pour Qt (axe y vers le bas de l'écran).

3. Enfin, la classe PH est en mesure de renvoyer un objet de la classe PHScene (via la méthode getGraphicsScene) qui est un Widget affichable par Qt. Ce widget est dessiné à partir des informations contenues dans les structs que l'on a construits à l'étape 2.

De plus, la classe PHScene met en relation les structs en question avec les objets du Process Hitting qu'ils représentent via des structures de données appropriés du package gfx (GProcess, GSort, etc.). C'est très important car c'est ce qui permettra dans les futurs développements de l'application de savoir avec quelles données du Process Hitting (quelle sort, quel process, etc.) l'utilisateur tente d'intéragir avec sa souris.

4. création de la fenêtre principale

Qt a permis de réaliser la fenêtre principale (dans le code, il s'agit de la classe MainWindow).

Cette fenêtre est très simple dans sa composition.

Tout d'abord, une barre de menu unique en haut permet d'accéder aux différentes fonctionnalités. Le menu File est classique. Il permet par exemple d'ouvrir un fichier, de le sauvegarder ou encore de l'exporter.

Le menu Edit regroupe les fonctions d'annulation de la dernière tache effectuée ou de répétition cette tache (non implémentées).

Le menu View permet de modifier l'affichage du fichier ph (fonctions non implémentées).

Le menu Computation permet d'appeler les fonctions de pint (fonctions implémentées : find fixpoints, run stochastic simulation, statistics / fonctions non-implémentées : check model type, compute reachability)

Le menu Help permet d'accéder à l'aide (non implémentée).

Ensuite, la zone principale de la fenêtre permet l'affichage des fichiers ph au sein d'onglets.

Les fonctions dans le menu liées à un fichier ph ne sont activées que lorsque au moins un onglet est ouvert et actif.

Au niveau du code, la classe MainWindow contient en attribut des pointeurs vers la zone centrale et les différentes parties de son menu.

Concernant les méthodes :

- getAllPaths permet d'obtenir les chemins vers tous les fichiers actuellement ouvert

Elle permet en particulier de savoir si un fichier est déjà ouvert lorsqu'on tente d'en ouvrir un.

- compute permet d'appeler un fonction extérieur (de pint en l'occurrence) et gère l'affichage des réponses.

- enableMenu permet d'activer les menus concernés lorsqu'un onglet est ouvert

Concernant les slots (qui sont des méthodes un peu particulières de Qt) :

- openTab permet d'ouvrir un nouvel onglet

-save permet de sauvegarder

-exportPng permet d'exporter un fichier ph en png

- findFixpoints, computeReachability, runStochasticSimulation, checkModelType (non implémentée) et statistics (non fonctionnelle) permettent d'appeler compute avec les bons arguments

-disableMenu permet de désactiver les menus concernés lorqu'aucun onglet n'est ouvert.

La classe MyArea est la classe appelée lors de la création d'un nouvel onglet (l'onglet contient un instance de MyArea).

Cette classe hérite de QgraphicsView permettant ainsi l'affichage d'une scène. Elle contient également quelques informations utiles : myPHPtr qui pointe vers un PH et path qui contient le chemin vers le fichier associé.

4. Résultats

1. représentation graphique d'un fichier ph

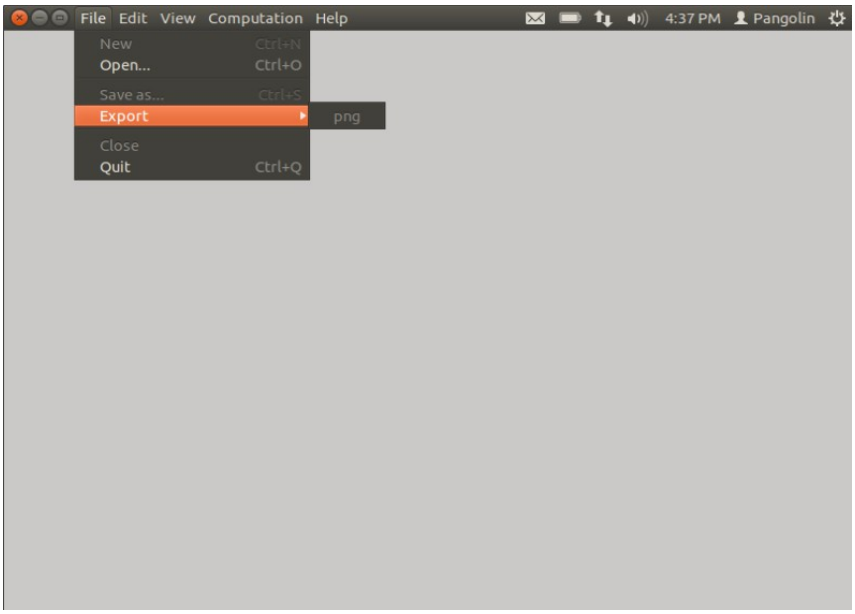
La représentation graphique des fichiers ph proposée par notre visionneuse est fonctionnelle et valide les outils choisis le processus que nous avons établi pour nous rendre du fichier ph au visuel. La visionneuse donne de bons résultats sur des fichier simples comme metazoan.ph (un fichier exemple de pint) mais plusieurs problèmes limitent son utilité immédiate :

- Tout d'abord, le calcul d'une disposition est un calcul très lourd dès lors que le fichier PH contient plus de quelques sortes (ce qui arrive très vite dès que l'on utilise la macro appelée COOPERATIVITY). Il en résulte un délai de plusieurs secondes à l'ouverture d'un fichier. Ce délai risque plus tard de se répercuter sur la manipulation du modèle qui ne manquera pas de provoquer le calcul d'une nouvelle disposition et qui risque de rendre l'application pénible à utiliser. Fort heureusement, il semble possible d'accélérer les calculs de graphviz en fixant arbitrairement (par exemple d'après la disposition précédente) la position de noeuds du graphe.

- Plus grave, le concept même de représentation de fichiers PH pose des problèmes d'ergonomie qui sont apparus dès les premiers tests grandeur nature. Le très grand nombre de sorts et process à représenter conduit au calcul de graphes gigantesques et qui sont au final encore moins explicites que le format texte. A titre d'exemple, une représentation de ERBB_G1-S.ph (un des fichiers d'exemples de pint) avec toutes les macro cooperativity dépliées mesure environ 31 000 pixels de large. Inutile de dire que l'ensemble n'est pas très facile à appréhender, ce qui met d'autant plus en évidence la nécessité de prévoir une fonctionnalité permettant de masquer les sorts issues de macro et/ou de représenter les COOPERATIVITY par des hyperarcs (des « double flèches »).

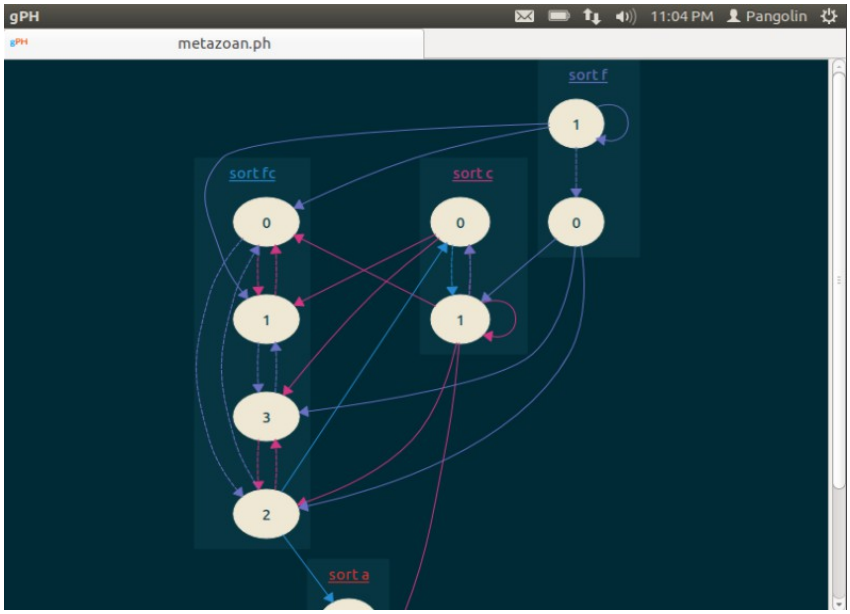
2. présentation de la fenêtre principale

La fenêtre principale se présente ainsi :



aucun onglet ouvert

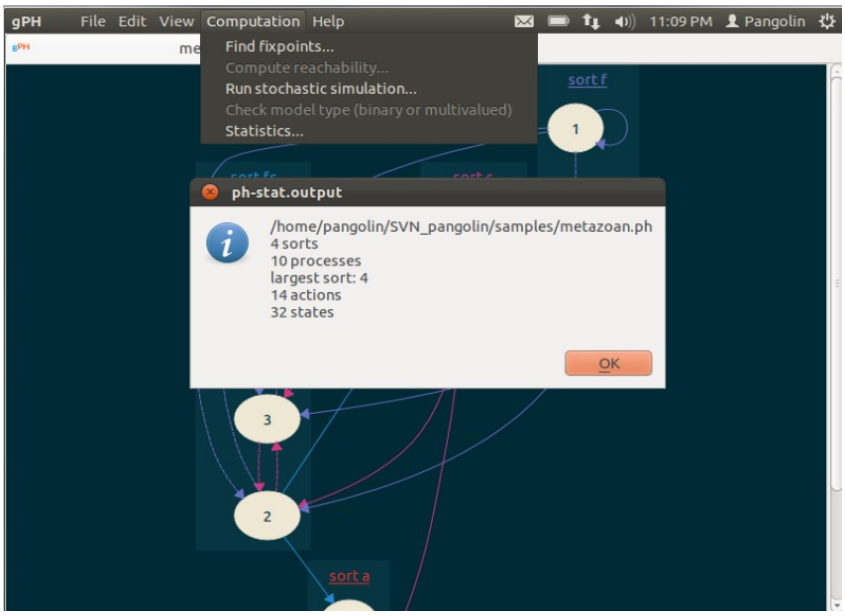
Peu de choix sont alors disponible dans les menus. Mais si on ouvre un fichier ph :



un fichier ph a été ouvert

On observe alors dans l'onglet une représentation graphique du ph.

On peut alors utiliser certaines fonctions de Pint :



résultat d'un traitement

3. tests

Il est possible de compiler l'application en mode test (voir documentation pour la marche à suivre). Après une telle compilation, l'exécution du programme lance les tests unitaires disponibles plutôt que l'interface graphique. Ces tests unitaires assurent du bon fonctionnement du parser de fichiers PH en le soumettant à 9 exemples élémentaires qui illustrent les expressions simples du langage qui les décrit ainsi qu'à 3 des fichiers d'exemples disponibles sur le site de pint (metazoan.ph, ERBB_G1-S.ph et tcrsig40.ph). Ces tests ont été écrits avant le développement du parser et ont permis de mesurer la progression de ce dernier de 0% à 100% correct.

Le reste du programme ne dispose à l'heure actuelle pas de tests unitaires car programmes basés sur des interaction graphiques s'y prêtent mal. Les fonctions disponibles ont cependant été testées manuellement après leur implémentation.

Si le développement de cette application vient à continuer, il est dorénavant très facile d'ajouter de nouvelles séries de tests unitaires en prenant exemple sur le fichier PHIOTest.cpp en en modifiant le fichier TestRunner.cpp.

Au niveau de la fenêtre, plusieurs cas ont été envisagés afin d'éviter des erreurs.

Un système d'activation et de désactivation des menus permet d'empêcher l'appel de certaines fonctions lorsqu'aucun onglet n'est ouvert (les fonctions qui en ont besoin en l'occurrence). On a ainsi les fonctions de sauvegarde et d'export, les fonctions de pint et le fonction de fermeture d'un onglet.

Au cas où ce système serait défaillant, chacune de ses fonctions prévoit un message d'erreur en cas d'appel malvenu.

Pour le cas de l'ouverture d'un fichier, plusieurs situations ont également été prévues. Ainsi, si le fichier est déjà ouvert ou si le fichier choisit n'est pas reconnu par le parser,

un message d'erreur s'affiche (le test s'effectue au niveau du parser, ainsi on peut ouvrir un fichier même s'il ne possède pas la bonne extension, laissant l'utilisateur le choix d'utiliser des fichiers avec ou sans extension).

Au niveau de l'appel des fonctions de pint, on récupère l'erreur si erreur il y a.

---

5. Bilan

1. comparaison cahier des charges et travail effectué

Créer / ouvrir / fermer des fichiers .ph	
Enregistrer dans les formats supportés par l'utilitaire phc de pint	
Éditer le contenu d'un fichier ph	
Proposer des outils de visualisation du contenu d'un fichier ph	
Positionner les sorts par un système de Drag 'n drop et imposer ces positions	
Définir des groupes de sorts (par une entrée texte, ou en sélectionnant avec la souris)	
Afficher / masquer des sorts ou des groupes de sorts et des actions	
Zoomer/dézoomer	
Mettre en valeur les sorts/actions/process activés par un scénario	
Colorer une sort/un groupe de sorts	
Exporter au format PNG	
Effectuer une recherche textuelles sur les sorts	
Obtenir des statistiques sur un fichier ph (ph-stat)	
Trouver les états stables d'un BRN (ph-stable) et les afficher en texte	
Afficher les états stables en graphique	
Déterminer si un état est atteignable (ph-reach)	
Faire une simulation (ph-exec) du PH et pouvoir l'enregistrer	
Pouvoir la réaliser en cliquant directement sur les frappes	
Vérifier qu'un BRN est binaire (et non multivalué)	
Définir un BRN à l'aide d'un modèle de Thomas et le convertir en fichier .ph	
Pouvoir ajouter des traitements sans avoir à recompiler (via un fichier de configuration) avec prise en compte des arguments et du format de sortie	
Rechercher la liste des états possibles	

2. décompte des heures

Activités d'Antoine	Durée
Prise en main des outils et bibliothèques	33h
Prise en main de Pint	2h
Développement du parser de fichiers PH	8h
Écriture de fichiers PH	1h
Développement de la visionneuse	20h
Cahier des charges	5h
Réunions (avec Karlo et/ou MeForBio)	6h
Soutenance et préparation	2h
Écriture des rapports d'avancement	1h30
Écriture du rapport final	5h
Documentation	5h
Total	**88h30**

Activités de Karlotcha	Durée
Prise en main des outils et bibliothèques	16h
Prise en main de Pint	2h
Développement de la fenêtre principale	30h
Cahier des charges	4h
Réunions (avec Antoine et/ou MeForBio)	6h
Soutenance et préparation	3h
Écriture des rapports d'avancement	0h30
Écriture du rapport final	8h
Documentation	1h
Total	**70h30**

3. suite à donner à ce projet

Le projet étant loin de l'avancement espéré au moment de la rédaction du cahier des

charges, il convient de se poser la question de la direction à prendre pour poursuivre son développement. Dans l'hypothèse que le projet soit repris par exemple en projet de groupe ou d'application l'année prochaine, voici une « roadmap » qui liste dans un ordre logique les prochaines étapes que nous envisagerions pour mener à bien le projet. Les numéros indiquent la difficulté relative des taches (de (1) pour quelques heures à (3) pour plusieurs jours de travail).

- (2) Déléguer l'exécution des calculs de disposition (par graphviz) à un thread dédié pour éviter que l'application ne freeze à l'ouverture d'un fichier complexe.

- (1) Ajouter les fonctions d'export vers les autres formats pris en charge par Pint

- (1) Ajouter l'opération qui vérifie qu'un fichier PH chargé est binaire et non multivalué

- (3) Développer les fonctionnalités d'édition de fichier PH en exploitant les events à la souris qui ont lieu sur les éléments de la représentation graphique du Process Hitting.

- (2) Ajouter le menu qui permet de paramétrer de nouveaux traitements que l'on souhaite lancer sur un fichier PH

- (2) Ajouter le panneau qui permet d'éditer un fichier PH en mode texte

- (2) Ajouter les fonctions de recherche textuelles dans un PH (exemple : sélectionner toutes les sortes dont le nom contient une chaîne)

- (3) Faire un affichage dynamique du résultat de l'exécution du traitement de ph-exec (simulation stochastique)

- (3) Ajouter les fonctionnalités d'édition via un modèle de Thomas

Il nous semble important de noter qu'au regard de notre expérience sur ce projet, le travail restant représente toujours un volume horaire très conséquent. Il nous parait déraisonnable d'espérer développer toutes les fonctionnalités envisagées au départ dans le cadre d'un seul projet d'application dans les mêmes conditions que le notre.

4. difficultés rencontrées

Au niveau technique, nous pouvons souligner la difficulté de maniement de plusieurs bibliothèques : boost::spirit et graphviz (dont la documentation n'est pas non plus d'un grand secours). Le langage utilisé (le C++) n'est pas non plus une source de facilité (sans être un réel problème).

Sur le plan de la gestion du projet, les rapports d'avancement ne répondaient pas tout

à fait aux attentes. Nous avons réalisé des efforts afin de les rendre plus conséquents et espérons avoir réussi à atteindre à la fin de ce projet un meilleur niveau de communication.

Enfin, le résultat final en comparaison du cahier des charges peut surprendre. Cette divergence n'est pas seulement à mettre sur le compte de notre sous-estimation de la durée de développement des fonctionnalités clés du projet. Il s'agit également d'un manque de compréhension de notre part vis-à-vis des modalités du projet d'application. Nous n'avons pas pensé (à tort) être en droit de refuser fermement l'ajout de fonctionnalités au cahier des charges. Nous l'avons plus considéré comme un sujet scolaire qu'un document contractuel. Cela a conduit à la rédaction d'un cahier des charges et d'un planning irréalistes dans le cadre de ce projet.

5. apport du projet de PAPPL

Outre ces aspects négatifs, le projet de PAPPL présente tout de même un certain nombre de points positifs. Ainsi, nous avons tous deux acquis des compétences techniques réutilisables (utilisation des bibliothèques, progrès en C++) et avons également acquis une expérience supplémentaire de gestion de projets.

Nous avons également pu voir d'un peu plus près ce à quoi pouvait ressembler la bio-informatique et les problèmes que l'on peut rencontrer (comme l'extrême complexité des réseaux biologiques, même simplifiés).

De plus, même si le projet est loin des attentes initiales, il est stable et utilisable.

Enfin ce projet est resté convivial malgré les difficultés et nous avons pris plaisir à y participer.

6. conclusion

Comme nous l'avons vu, tout ne s'est pas déroulé sans accroc. Nous considérons cependant ce projet comme une expérience très positive. En effet, chaque difficulté a permis de progresser (sur le plan technique autant que de gestion de projet).

Nous nous sommes également efforcés de rendre notre travail facile à reprendre en main dans l'espoir que d'autres étudiants puissent nous succéder.