POLYTECH TOURS
64 avenue Jean Portalis
37200 TOURS, FRANCE
Tel +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Rapport projet “Simulation Hôpital”

2023-2024

Etudiants : Arthur Crochemore, Louis Soulier

Professeur encadrant : Yannick Kergosien

I) Périmètre

II) Spécifications

III) Conception

IV) Conclusion

V) Annexes

1 Périmètre du projet

1) Introduction du sujet

Le projet consistait à développer un moteur de simulation pour un hôpital. Nous avons donc décidé de représenter la répartition de patients dans les salles d'opérations d'un hôpital.

Nous avons décidé qu'il y ait 2 flux de patients distincts, un flux de patients prévus au début de la simulation et un flux "urgence", représentant les différents patients qui arrivent sont déclarés au cours de la simulation.

Pour améliorer la complexité de la simulation, nous avons décidé de définir des règles de gestion qui modifient le comportement des éléments de la simulation. Ainsi, l'application pourra permettre de comparer différentes organisations afin de l'optimiser.

2) Composants de la simulation

Durant les premières séances, nous nous sommes concentré sur l'analyse des différents composants qui feront partie de notre simulation. En effet, notre simulation est composée d'entités, de ressources, d'un planning, de règles de gestion et d'événements.

a) Les entités

Les entités sont des objets qui vont passer plusieurs états dans la simulation avant d'en sortir. Nous avons donc un type d'entité pour chaque flux de patients : les patients rendez-vous et les patients urgents.

Les patients ont tous une heure d'arrivée dans le bloc, un temps d'opération et une heure de sortie une fois leur passage terminé. Dans notre cas, ils passeront tous par les mêmes étapes et dans le même ordre. Ils vont donc changer d'état en fonction de l'étape à laquelle ils sont.

Liste des états d'un patient :

- pas encore arrivé
- en attente d'une salle
- a attendu une salle (= est affecté à une salle)
- en attente de préparation de la salle
- en préparation
- en attente d'un chirurgien
- en opération
- en attente de libération de la salle
- terminé

Les patients ont aussi une “gravité” c'est-à-dire qu'ils ne pourront pas être affectés à n'importe quelle salle, certains auront besoin d’une salle plus ou moins équipée.

Les patients “RDV” peuvent donc être affectés dans des salles “Peu équipées” , “Semi équipées” ou “Très équipées”.

Cependant, les patients “Urgent” ne peuvent être affectés qu’à des salles “Très équipées”. Une autre donnée supplémentaire que nous avons sur les patients “urgents” est celle de leur heure de déclaration.

b) Les ressources

Ensuite nous avons identifié 3 ressources: les infirmières, les chirurgiens et les salles, ce sont des objets qui alternent entre des états occupés et libres, dépendant de s’ils sont actifs ou non. Leur rôle est de permettre aux patients de passer certaines étapes, nécessitant leur présence.

Le cas des salles est un peu différent, il y a plusieurs types de salles en fonction du besoin de l’opération :

- Peu équipée (petite opération)
- Semi équipée (moyenne opération)
- Très équipées (opération critique)
- Réservées (salles Très équipées réservé pour les patients urgents)

De plus, les salles ont plus d'états possibles (permettant d’estimer le temps restant avant libération de la salle) :

- Libre
- En libération
- En attente de libération
- En opération
- En attente d’opération
- En préparation
- En attente de préparation

c) Les évènements

La simulation est constituée d'événements. Un événement représente un changement de l'état du système à un moment précis.

Liste des évènements :

- Déclaration d’un patient urgent (le moment où l'hôpital est prévenu de l’arrivé d’un patient urgent)
- Arrivée d’un patient (le moment où le patient arrive à l'hôpital)
- Arrivée dans la salle (le moment où le patient est affecté à une salle d’opération)
- Début de la préparation de la salle (commencement de la phase de préparation de la salle par une infirmière (préparer les machines, outils médicaux, faire entrer le patient dans la salle))
- Fin de préparation de la salle (fin de la préparation de la salle par l’infirmière)
- Début de l’opération (début de l’opération du patient par le chirurgien)
- Fin de l’opération (fin de l’opération par le chirurgien)

- Début de la libération de la salle (commencement de la phase de libération de la salle par une infirmière (sortir le patient, nettoyer les outils …))
- Fin de libération de la salle (fin de la libération de la salle par l'infirmière)
- Infirmière disponible (lorsqu' une infirmière devient disponible (après préparation ou libération))

d) Planning

La simulation comporte un planning qui gère l'affectation des patients dans les salles de sorte à éviter d'avoir des placements aléatoire qui pourrait faire perdre du temps. Ce planning est à la base représenté par les rendez-vous de la journée.

e) Règles de gestion

Une autre partie essentielle de la simulation est celle des règles de gestion qui permettent d'avoir une diversité de choix de gestion sur l'action des ressources. Nous en avons mis en place plusieurs sur chaque ressources. Tout d'abord, par rapport aux infirmières on peut vouloir jouer sur les règles suivantes :

- Priorité libération (Règle de gestion priorisant l'affectation des infirmières aux tâches de libérations des salles)

- Priorité premier en attente (Règle de gestion priorisant l'affectation des infirmières aux salles qui sont en attentes depuis le plus longtemps)

- Priorité préparation (Règle de gestion priorisant l'affectation des infirmières aux tâches de préparation des salles)

- Priorité Urgence (Règle de gestion priorisant l'affectation des infirmières aux salles où se trouve les patients urgents)

Nous avons fait de même pour les chirurgiens avec les règles suivantes :

- Priorité premier en attente (Règle de gestion priorisant l'affectation des chirurgiens aux salles qui sont en attentes depuis le plus longtemps)

- Priorité Urgence (Règle de gestion priorisant l'affectation des chirurgiens aux salles où se trouve les patients urgents)

Et de même pour la gestion des plannings, responsable de la génération des plannings, nous avons donc les règles suivantes :

- Priorité absolu urgence (Règle de gestion plaçant en priorité les patients urgents dans les salles de sorte à minimiser leur attente)

- Priorité priorité rendez-vous (Règle de gestion plaçant en priorité les patients RDV dans les salles de sorte à minimiser leur attente)

- Priorité première arrivée avec des salles réservées de façon dynamique (règle de gestion plaçant les patients dans l'ordre de leurs heures d'arrivée et gérant la réservation des salles de sorte à maintenir nbSalleReservée salles TE libres pour les patients urgents)

- Priorité première arrivée avec des salles réservées de façon statique (règle de gestion plaçant les patients dans l'ordre de leurs heures d'arrivée)

f) Les listes d'attente

Au sein de la simulation se trouve aussi différentes listes d'attente qui vont se remplir lorsque des ressources sont demandées mais que celles-ci ne sont pas disponibles à ce moment-là. Voici les listes d'attente que nous avons identifiés :

- Liste d'attente infirmière (liste d'attente comportant les salles en attente d'infirmière)
- Liste d'attente infirmière en préparation (utilisé pour la règle de gestion priorisant affectation des infirmières dans les salles en attente de préparation)
- Liste d'attente infirmière en libération (utilisé pour la règle de gestion priorisant affectation des infirmières dans les salles en attente de libération)
- Liste d'attente chirurgien (liste d'attente comportant les salles en attente de chirurgien)
- Liste d'attente infirmière urgent (utilisé pour la règle de gestion priorisant affectation des infirmiére dans les salles ayant un patient urgent en attente d'infirmière)
- Liste d'attente chirurgien urgent (utilisé pour la règle de gestion priorisant affectation des chirurgiens dans les salles ayant un patient urgent en attente de chirurgien)

g) Les constantes

Afin de pouvoir effectuer la simulation plusieurs informations sont nécessaires et commune à toutes les entités ce sont les constantes suivantes :

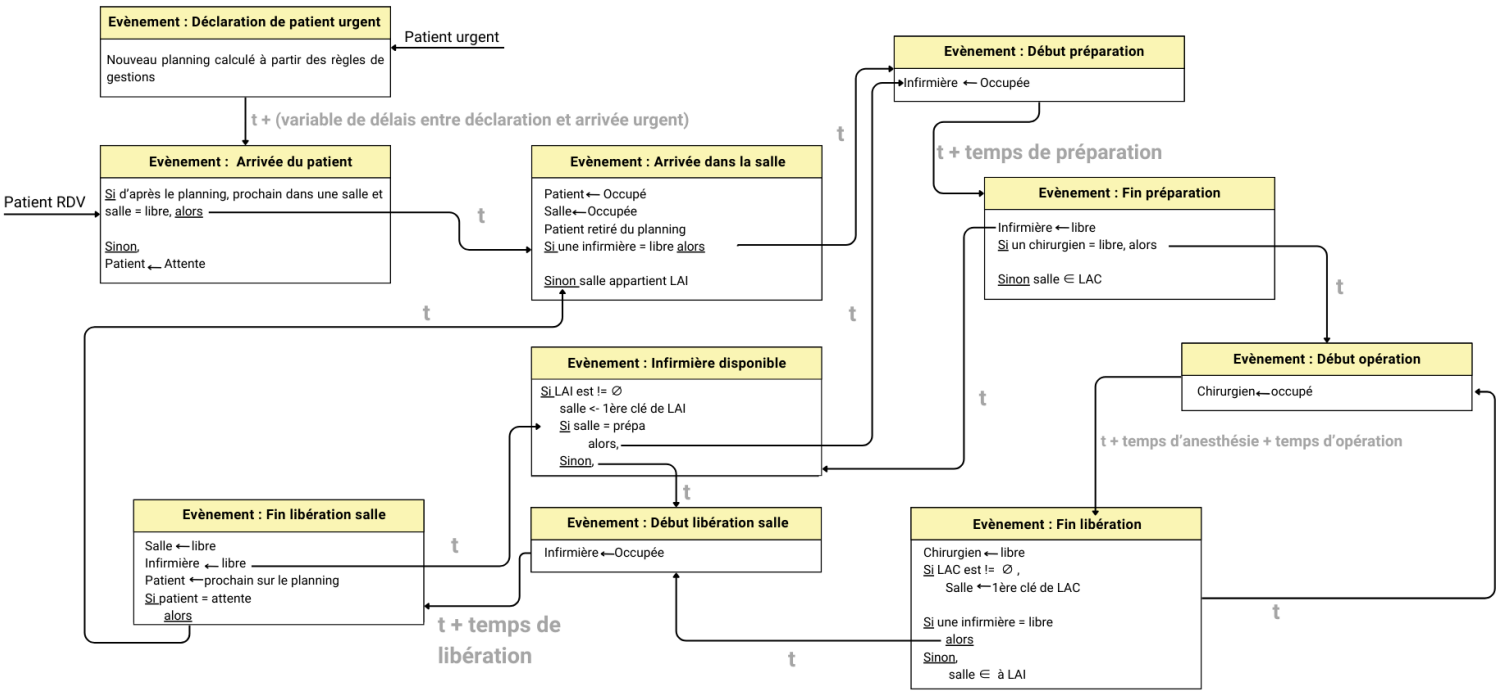
- Le temps moyen d'opération
- Le temps d'anesthésie
- Le temps de libération
- Le temps de préparation
- Le nombre de salles réservées

Finalement, grâce à l'identification des composants de la simulation nous avons pu commencé à finaliser la carte d'évènement.

2 Spécifications

Dans un second temps, nous avons élaboré des documents servant en quelque sorte de spécifications. En effet, nous avons commencé par faire une carte d'événement qui illustre comment les différents évènements s'enchaînent sous quelles conditions par rapport aux ressources et aux entités. Puis, nous nous sommes penchés sur la conception du diagramme de classe.

1) Carte d'événements



La conception d'un tel document ne se fait pas en une fois et nous avons dû en discuter plusieurs fois avec notre encadrant avant d'établir une carte d'événement abouti et finale que voici :

Nous pouvons voir que selon si l'entité entrant dans la simulation est un patient "Urgent" ou "RDV" le cheminement est quasiment le même sauf au tout début. En effet, un patient "RDV" arrive dans la simulation via l'évènement "Arrivée patient" tandis qu'un patient "Urgent" passe d'abord par l'évènement "Déclaration patient urgent" avant de pouvoir rejoindre la suite du déroulement des évènements "classique" suivante :

Événement “Déclaration patient urgent” :

Dans le cas où l'entité est un patient urgent, le planning est actualisé en fonction des règles de gestion choisies. Puis, déclenchement de l'événement “Arrivé patient” à heure de déclaration + heure d’arrivée prévue de l’urgence.

Événement “Arrivée Patient” :

Le patient arrive dans l’hôpital à l’heure t. S’il est le prochain affecté à une salle et que cette salle est libre, alors il est affecté à la salle et se déclenche l'événement “Arrivée dans la salle”. Sinon il entre en attente.

Événement “Arrivée dans la salle” :

Le patient arrive dans l'événement à l’heure “t”(heure de début de l'événement précédent + le temps d’attente). Le patient et la salle passe dans l’état occupé. Puis, le patient est retiré du planning. Si une infirmière est libre alors le patient arrive dans l’évènement “Début préparation”. Sinon, la salle est ajoutée à la liste d’attente d’infirmière.

Événement “Début préparation” :

Le patient arrive dans l'événement à l’heure “t”(heure de début de l'événement précédent + le temps d’attente). L’infirmière passe dans l’état “occupé”. Puis, déclenchement de l'événement “Fin préparation” à t + la constante de temps de préparation.

Événement “Fin préparation” :

Le patient arrive dans l'événement à l’heure “t”(heure de début de l'événement précédent + le temps de préparation). L’infirmière passe dans l’état “libre” ce qui déclenche l’évènement “Infirmière disponible”. Si un chirurgien est libre alors déclenchement de l’évènement “Début opération”. Sinon, la salle est placée dans la liste d’attente de chirurgien.

Événement “Début opération” :

Le patient arrive dans l'événement à l’heure “t”(heure de début de l'événement précédent + le temps d’attente). Le chirurgien passe dans l’état “occupé” puis, déclenchement de l'événement ”Fin opération” à t + la constante de temps d' opération + la constante de temps d’anesthésie.

Événement “Fin opération” :

Le patient arrive dans l'événement à l’heure “t”(heure de début de l'événement précédent + le temps d’opération + la constante de temps d’anesthésie). Le chirurgien passe dans l’état “libre”. Si la liste d’attente de chirurgien n’est pas vide alors le chirurgien est affecté à la première salle de cette liste et donc déclenchement de l'évènement ‘Début opération”. Si une infirmière est “libre” alors déclenchement de l’évènement “Début libération”. Sinon la salle est ajoutée à la liste d’attente d’infirmière.

Événement “Début libération” :

Le patient arrive dans l'événement à l’heure “t”(heure de début de l'événement précédent + le temps d’attente).L’infirmière passe dans l’état “occupé”. Puis, déclenchement de l'événement “Fin libération” à t + la constante de temps de libération.

Événement “Fin libération” :

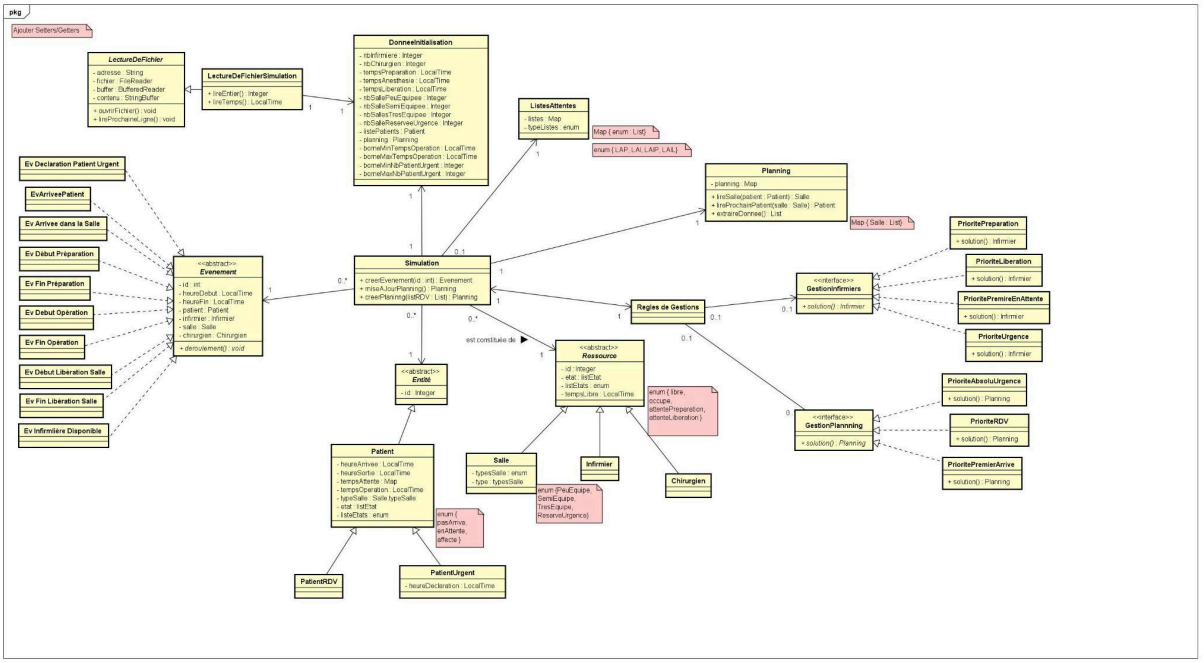
Le patient arrive dans l'événement à l’heure “t”(heure de début de l'événement précédent + le temps de libération). L’infirmière passe dans l’état “libre” ce qui déclenche l’évènement “Infirmière disponible”. La salle et l’infirmière passe dans l’état “libre”. Le patient devient le prochain patient sur le planning. Si il est en état d’attente alors déclenchement de l'événement “Arrivée dans la salle”.

Événement “Infirmière disponible” :

Si la liste d’attente d’infirmière n’est pas vide, l'infirmière va dans la première salle de la liste. Si c’est une salle en attente de préparation, déclenchement de l’évènement “Début préparation” sinon c’est une salle en attente de libération et déclenchement de l’évènement “Début libération”.

2) Diagramme de classe (pattern, explications des rôles des classes)

Notre premier diagramme de classe ressemblait à :



Nous essayons de séparer chaque fonctionnalité de manière indépendante de sorte à améliorer la lisibilité et la compréhension tout en gardant un objet Simulation au centre qui lie l’ensemble des éléments.

a) Ressource et entités

Nous avons représenté chaques éléments de la simulation par une classe en les mettant classes filles de respectivement Entité et Ressource, gérant chacuns les ids. En plus de ça, nous avons créé une classe Patient pour regrouper les attributs communs à PatientRDV et PatientUrgent.

Pour chacuns nous avons défini un attribut enum listEtat ainsi qu'un attribut état pour gérer les état des ressource/entité. Nous avons fait de même avec typeSalle et type dans Salle pour représenter le type de salle.

b) Planning

Nous avons représenté le planning via le classe Planning qui contient un attribut planning représentant la liste d'affectation des patients aux salles pour chaque salle ("Map<Salle, List<Patient>>"). Nous avons ajouté à ça 2 méthodes qui permettent de lire le planning : une première qui pour une salle lit le prochain patient, et pour un patient lit la salle à laquelle il est affecté.

Nous avons aussi mis une méthode extractionDonnee qui renvoie la liste des patients stockés dans le planning, appelés lorsqu'un nouveau planning va être créé de sorte à ne considérer que les patients connus au moment du changement de planning dans le nouveau planning (sans les patients urgents pas encore connus et les patients déjà affectés).

c) Initialisation

Nous avons aussi défini une fonctionnalité d'initialisation depuis un fichier txt. Pour se faire, nous avons découpé en 3 classes :

Une première classe LectureDeFichier qui contient les attributs de base et les méthodes pour pouvoir lire les lignes une à une.

Une seconde classe LectureDeFichierSimulation fille de LectureDeFichier qui gère la récupération des données depuis le fichier txt. Elle contient donc 2 méthodes qui lisent respectivement un entier ou un LocalTime, et la méthode initialiserSimulation qui gère la configuration pour la création de la simulation.

Une troisième classe DonneeInitialisation qui contient les éléments nécessaires à la création de Simulation. Elle est créée dans la méthode initialiserSimulation de LectureDeFichierSimulation. Cette classe DonneeInitialisation contient aussi tous les accesseurs nécessaires à sa configuration et à sa lecture pour Simulation, ainsi qu'une méthode creerSimultation, qui comme son nom l'indique, crée l'objet Simulation en appelant le constructeur de confort de ce dernier.

d) Liste d'attentes

Pour les listes d'attente (hors planning), nous les avons stockées dans un objet ListesAttentes. Pour faciliter leur stockage, nous avons décidé de créer un objet enum typeListes qui contient les intitulés des différentes listes, ainsi les listes sont représentées par un attribut listes Map<typeListes , List<Patient>> qui lit à chaque intitulé la liste associée.

e) Evénements

Pour les événements, nous avons utilisé un patron de conception Stratégies, puisque à chaque événement est associé un déroulement propre à lui-même. Ainsi, chaque événement est fils d'une même classe Evenement.

De plus, puisque chaque événement peut nécessiter un patient et/ou un infirmier et/ou un chirurgien et/ou une salle, nous avons mis un attribut de chaque événement un élément de chaque. Nous avons donc spécifié ça directement dans Evenement, en plus des

attributs heureDebut et heureFin, caractérisant la période de temps lié au déroulement de l'événement.

Ensuite, nous avons décidé que dans la méthode déroulement de chaque événement (correspondant à sa définition dans la carte d'événements) soit fait la création des événements qui suivent.

Par exemple : dans un événement arrivé patient, si le patient est affecté à une salle , alors un événement arrivé dans la salle est créé.

f) Règles de gestions

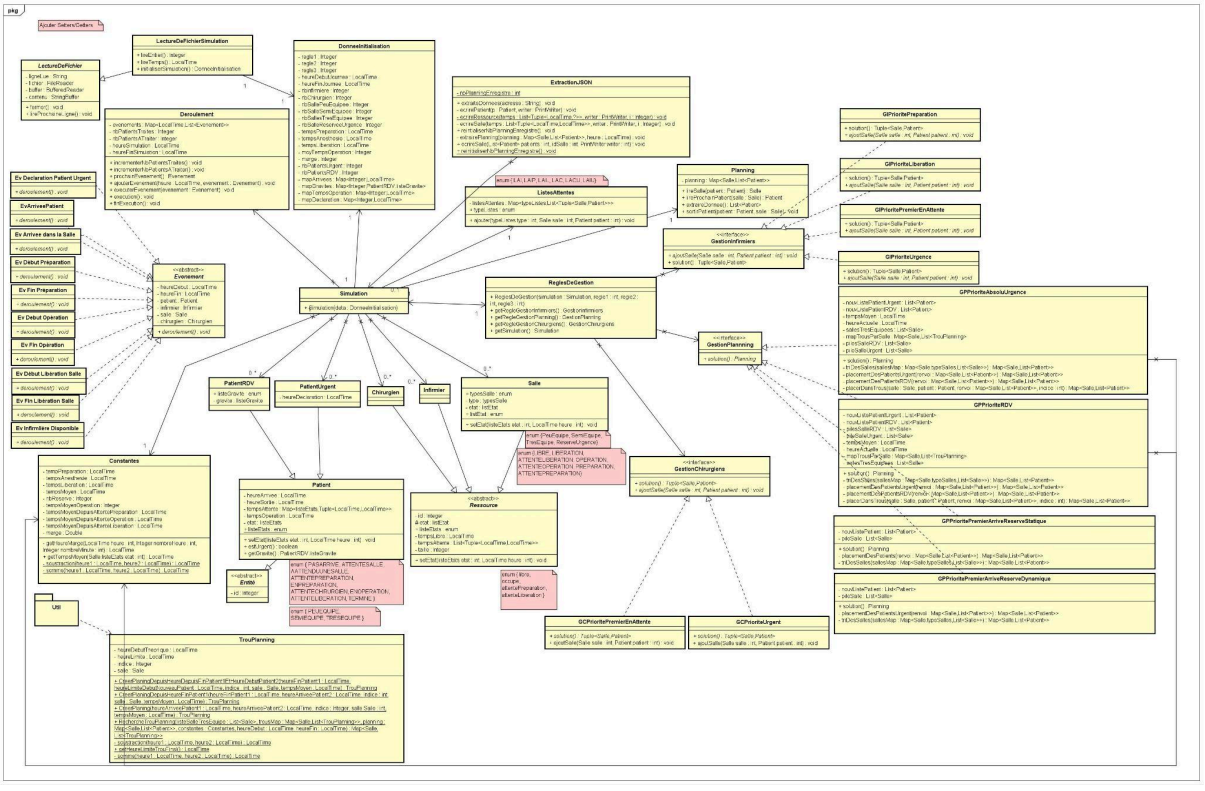
La dernière partie de la simulation est la partie concernant les règles de gestion. Pour cette partie nous avons décidé d'utiliser de nouveau un patron de conception Stratégie, puisque chaque version d'une règle de gestion fait la même chose mais de manière différentes. Nous avons utilisé un nommage pour ses différentes versions pour comprendre facilement à quelle règle de gestion cela fait réference.

Nous avons donc une interface pour chaque règles de gestion reliées à ses différentes versions, et chacune de ces interfaces est stockée dans un objet Règles de Gestions.

Nous avons aussi défini un constructeur à Règles de Gestions qui gère le choix des règles de gestion à créer en fonction d'entiers passés en paramètres.

g) Dernier planning

Au cours du codage, nous avons identifié plusieurs problèmes à ce premier diagramme. Cela nous a donc amené, après plusieurs rectifications, à ce diagramme :



Nous pouvons voir plusieurs changements :

- Ajout de la GestionPlanning, permettant de gérer la création des planning en fonction de la règle de gestion choisie

- Modification des références aux entités et ressources dans Simulation de sorte à ce qu'il ne soit plus considéré comme des Entite ou des Ressource (nous ne pouvions rien en faire, leur méthode respective étant inatteignable)
- Ajout de l'objet Déroulement pour gérer le déroulement des événements. Ainsi cet objet contient un map <LocalTime, List<Evenement>> de sorte à lier à chaque heure la liste des événements qui doivent avoir lieu. Ainsi, l'attribut heureSimulation est initialisé à l'heure de début, puis, tant qu'il ne dépasse pas l'heure de fin :
    - le premier événement stockés à heureSimulation est exécutés
    - s'il n'y en a pas, heureSimulation est augmenté de 1 minute

  Ainsi, tous les événements sont déroulés 1 à 1 dans un ordre logique
- Ajout de l'objet Constante qui contient l'ensemble des constantes liés à la simulation. Aussi, spécifications des temps moyens estimés passés dans une salle d'opération calculée à partir des constantes de temps, d'un temps moyen d'opération et d'une marge. Ces temps moyens étant nécessaire pour les règles de gestion planning.
- Passage des attributs enum en publique pour qu'il soit accessible de l'extérieur (vu qu'il n'y a pas de risque qu'ils soient modifié)
- Passage de l'attribut état de Ressource en protégé pour que Infirmier et Chirurgien y est accès tandis que Salle puisse avoir une surcharge de cet attribut, lui permettant d'état plus précis
- Nous avons ajouté listeGravite et gravite dans PatientRDV pour représenter le type de salle dans lesquels le patient peut être opéré

3) Plan de conception des fonctionnalités

Dans le cadre de la réalisation de ce projet, nous avons fait un tri au niveau des fonctionnalités afin de les concevoir en suivant leur ordre d'importance. Tout d'abord, le modèle puis, le moteur de simulation avec le déroulement des évènements et enfin, les règles de gestion.

Le projet avait pour but de développer une simulation cependant, il nous tenait à cœur de le compléter avec d'autres fonctionnalités supplémentaires, nous avons eu ainsi plusieurs idées.

Tout d'abord, une interface graphique permettant la saisie des informations relatives à la simulation, facilitant la création du fichier txt à la main.

De plus, un outil pour extraire les données de la simulation afin de pouvoir constater les résultats.

Aussi, un outil pour traiter ses données afin d'obtenir des graphiques.

Finalement, un outil pour extraire les différents planning permettant d'observer plus en détail le fonctionnement de la simulation.

3 Conceptions

1) Moteur de simulation

a) Modèle

La première étape de la conception a été de créer les différentes classes modèles. Nous avons donc réparti chacune des classes dans différents modules.

Tout d'abord, un module Ressource correspondant aux objets représentant les ressources de la simulation. Nous avons donc coder Ressource, Chirurgien, Infirmier et Salle. Ces classes correspondent à leur définition du diagramme de classe. Ils sont donc seulement composé de simples accesseurs et de constructeurs de conforts.

Ensuite, un module Entite correspondant aux objets représentant les entités de la simulation. Nous avons donc codé cette fois-ci Entite, Patient, PatientRDV et PatientUrgent. Comme pour le module ressource, ces classes sont elles aussi composées de simples accesseurs et de constructeurs de conforts.

Enfin, nous avons créé l'objet Simulation, comprenant l'ensemble des éléments de la simulation. Elle est donc composée de simples assesseurs et d'un constructeur de confort qui initialise tous les objets de la simulation.

b) Initialisation

Pour pouvoir lancer la simulation, il nous fallait de nombreuses données en paramètre c'est pourquoi nous avons créé un module initialisation comportant 3 classes présentes sur le diagramme de classe (LectureFichier, LectureFichierSimulation et DonneesInitialisation) et qui permettent un paramétrage de la simulation grâce à la lecture d'un fichier .txt.

c) Déroulement

Le déroulement des événements est une pièce maîtresse dans le moteur de la simulation, c'est lui qui détermine comment s'enchaîne les événements et dans quel ordre.
Ainsi la classe contient un map <LocalTime, List<Evenement>> de sorte à lier à chaque heure la liste des événements qui sont programmés. De cette manière, l'attribut heureSimulation est initialisé à l'heure de début, puis, tant qu'il ne dépasse pas l'heure de fin de la simulation :
Le premier événement stockés à heureSimulation est exécutée, s'il n'y en a pas, heureSimulation est augmentée de 1 minute. Par conséquent, tous les événements sont déroulés 1 à 1 dans un ordre logique.

d) Règles de gestion

Dans le but d'intégrer des règles de gestion, nous avons appliqué des patrons de conception factory. En effet, nous avons créé 3 interfaces GestionChirurgiens, GestionInfirmières et GestionPlanning pour représenter chaque règles de gestion. Ainsi chaque sous-classe implémente leur version de la méthode solution() .

Pour les gestions des infirmiers et des chirurgiens, le code a été relativement simple. Cependant, les règles de gestion planning ont été bien plus complexes, du fait qu'elles intègrent chacune la génération d'un Planning.

Dans le cas des deux autres règles de gestion **GPPrioritePremierArriveeReserveDynamique** et **GPPrioritePremierArriveeReserveStatique** nous avons dû trier les patients dans leur ordre d'arrivé puis les salles dans l'ordre de leur heure de libération puis placer les patients un par un dans la première salle qui est cohérente pour le patient, dépiler la salle, et la mettre à la fin (elle devient la dernière salle libérée).

Dans le cas de **GPPrioritePremierArriveeReserveDynamique**, il a aussi fallu gérer le changement de salle réservées quand une salle réservée est affectée. Nous avons fait cela en stockant le nombre de salle qui passe de réservé à non-réservé (salle affecté à un patient urgent) et, lorsqu'une salle TE non-réservée doit être affecté à un patient non-urgent, on diminue le nombre de 1 et on réserve la salle (la salle n'est donc pas affectée).

Notamment, **GPPrioriteAbsoluUrgence** qui elle initialise de nouvelles listes de patients RDV et urgents, trie ces listes en fonction de leur heure d'arrivée, puis procède à la planification en affectant les patients urgents aux salles appropriées, puis la méthode gère l'affectation des patients RDV en les plaçant entre les patients urgents déjà planifiés.

La classe utilise diverses listes, des piles, et des maps pour organiser les patients, les salles, et les tranches horaires disponibles. Elle prend en compte la disponibilité des salles en fonction de leur type (réservées, peu équipées, semi-équipées, très équipées) et exploite les trous dans le planning pour placer stratégiquement les patients RDV.

La classe **GPPrioriteRDV** est une implémentation de gestion de planning privilégiant d'abord les patients RDV puis les patients urgents. Elle présente la même structure que **GPPrioriteAbsoluUrgence** mais commence par les patients RDV puis s'occupe des patients urgents.

Pour nous aider, nous avons dû créer des structures supplémentaire, comme la classe TrouPlanning qui identifie et représente les "trous" dans l'affectation des patients aux salles, notamment dans les salles de type "Très Équipé" (TE), où les patients urgents et les patients RDV peuvent être placés. Ces "trous" sont des intervalles de temps estimés suffisamment longs entre les patients déjà affectés à une salle afin d'en placer un autre. La classe offre des méthodes pour rechercher, créer et mettre à jour ces "trous".

e) Traitement des données

Pour pouvoir analyser une simulation, nous avions besoin de stocker des données liées au déroulement quelque part. Pour ce faire, nous avons ajouté des attributs tempsAttente (de type Map<listeEtats, Tuple<LocalTime, LocalTime>>) à toutes les entités.

Ainsi, les entités stockent leurs temps d'attente à chaque étape de leur passage dans l'hôpital.

Du côté des ressources , nous avons aussi ajouté un attribut tempsAttente (de type List<Tuple<LocalTime, LocalTime>>) permettant de stocker l'ensemble des intervalles pendant lesquels une ressource n'est pas active.

Enfin, pour que ces tempsAttente se mettent à jour au fur et à mesure de la simulation, nous avons modifié tous leurs accesseurs setEtat en ajoutant un LocalTime en paramètre, permettant de stocker le moment à partir desquels la ressource ou l'entité change d'état. Nous avons aussi rajouté du code pour que les temps d'attente soient correctement mis à jour.

f) Extraction des données

Une première fonctionnalité qui s'ajoute au moteur de la simulation est l'extraction des données d'attente créées juste au-dessus. Pour ce faire, nous avons créé un objet ExtractionJSON dans le bût est d'extraire ces données dans un fichier extraction.json.

Nous y avons donc créé une méthode extraiteDonnees qui gère l'écriture en respectant le format Json.

Pour améliorer la lisibilité de cette méthode et éviter la répétition du code, nous avons créé 3 méthodes:

- ecrirePatient, qui gère l'écriture des temps d'attentes d'un patient appelé pour chaque patient RDV et pour chaque patient urgent.
- ecrireRessource, qui gère l'écriture des intervalles de temps d'un infirmier ou d'un chirurgien ou celui-ci ne travaille pas, appelé pour chaque infirmier et chirurgien
- ecrireSalle, qui gère l'écriture des intervalles de temps d'une salle où celle-ci n'est pas occupé, appelé pour chaque salles de chaque type

Nous avons ainsi fait en sorte que extraiteDonnees soit appelée à la fin de la simulation (lorsque Déroulement atteint l'heure de fin). Ainsi, le déroulement de la simulation peut être facilement analysé en regardant ce fichier. De plus, les données peuvent être facilement récupérées puisque le format json est respecté.

2) Fonctionnalités supplémentaires

Puisque nous avancions bien et que le projet était assez libre, nous avons décidé d'implémenter plusieurs fonctionnalités supplémentaires qui nous semblaient utiles pour compléter le projet.

a) Interface

La première fonctionnalité à laquelle nous avons pensé et que nous avons implémenté fut la création d'une interface graphique. Après réflexion, nous avons décidé qu'une interface Swing ferait l'affaire, nous avons donc créé une interface à l'aide de NetBeans.

Cette interface aurait donc pour bût de faciliter la création du fichier d'initialisation au travers d'un formulaire de saisie correspondant à chaque valeur du fichier d'initialisation et qui permettrait de générer ce fichier d'initialisation.

Nous avons donc simplement représenter chaques champs soit par un comboBox, lorsqu'il s'agissait d'un élément qui devait être choisir entre plusieurs (comme pour les

règles de gestion), par un timePicker dans le cas de LocalTime ou par un textField dans le cas où une valeur était attendu.

Le seul élément que nous ne pouvions pas représenter ainsi était la liste des patients à créer. Pour ce faire, nous avons créé une deuxième fenêtre qui permet la saisie d'un patient un à un, et qui liste tous les patients déjà créés dans un tableau.

Pour implémenter la dynamique d'aléatoire dont nous avions parlé plus tôt vis à vis des temps d'opérations et de l'arrivée des patients urgents, nous avons :

- Ajouter un champs qui rempli le champs de saisie du temps d'opération en générant un entier entre 2 bornes en suivant une loi de poisson
- Ajouter un bouton qui génère automatiquement nombre de patients urgents donné en générant :
    - une heure d'arrivée à partir d'une loi de poisson entre l'heure de début de la simulation et l'heure de fin
    - une heure de déclaration qui correspond à l'heure d'arrivée moins un nombre de minutes générée en suivant une loi de poisson entre 5 et 45
    - un temps d'opération générer comme vu précédemment

b) Graphiques

Nous avons ensuite décidé d'ajouter à ça une seconde fonctionnalité liée à l'interface. Cette fois il s'agissait de traiter les données enregistrées dans le fichier extraction.json de sorte à les représenter sous forme de graphe.

Pour se faire, nous avons dû dans un premier temps lire les données et les stocker dans des map. Nous avons donc ajouté une méthode lireDonnees à partir d'un package "com.orsoncharts.util.json".

Une fois fait, nous avons dû utiliser un second package JFreeChart pour les graphes (org.jfree.chart). Pour ce faire, nous avons créé un graphique de type *LineChart* sur la fenêtre à partir d'une méthode initChart appelée juste après la méthode lireDonnees. Nous avons aussi ajouté un jPanel à la fenêtre qui permet de changer les données du graphes.

Ainsi, il y a 4 graphes :

- Un premier qui affiche le nombre moyens d'infirmiers non occupé à chaque intervalle de 15min depuis l'heure de début de la simulation à l'heure de fin
- Un second qui affiche le nombre moyens de chirurgiens non occupé à chaque intervalle de 15min depuis l'heure de début de la simulation à l'heure de fin
- Un troisième qui affiche le nombre moyens de salles non occupé à chaque intervalle de 15min depuis l'heure de début de la simulation à l'heure de fin, pour chaque type de salles
- Un dernier qui affiche le nombre moyens de patients qui attendent à chaque étape de la simulation pour chaque intervalle de 15min depuis l'heure de début de la simulation à l'heure de fin, pouvant être observé pour les patients RDV et pour les patients urgents

c) Sauvegardes des plannings

La dernière fonctionnalité à laquelle nous avons pensé est un enregistrement des différents planning de la simulation chaque fois qu'un nouveau est généré.

Ainsi nous avons ajouté une méthode extrairePlanning dans ExtractionJSON qui gère l'écriture des listes d'attentes de chaque salle, pour un planning donné, dans un fichier txt qui est stocké dans un dossier "historique_plannings". Nous avons aussi ajouté diverses informations utiles à la lisibilité ainsi que des listes des patients à chaques étapes :

- pas encore arrivé : qui représente les patients qui sont affecté à une salle mais ne sont pas encore là, expliquant ainsi pourquoi certaines salles sont libres alors que des patients y sont affecté
- en attente de salle : qui représente ceux qui sont là et qui attendent leur tour
- en opérations : les patients qui sont actuellement dans une salle, expliquant pourquoi certain patient n'apparaissent pas dans les listes d'attentes bien qu'il ne soit pas encore sortie de l'hôpital
- déjà traités : les patients sortit de l'hôpital (ou plutôt de la simulation)

4 Conclusion

1. Résultats

Finalement, nous avons réussi à développer un moteur de simulation capable de simuler la répartition de patients dans les salles d'opérations d'un hôpital. De plus, nous avons ajouté une interface graphique pour l'utilisateur et nous avons implémenté des outils d'extraction et de traitement des données afin de permettre à l'utilisateur de visualiser les résultats sous différents angles.

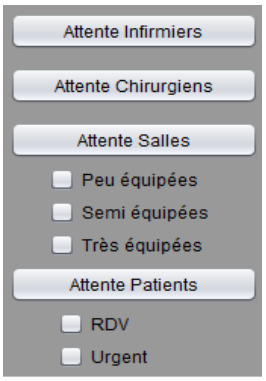
a) Les interfaces :

Captures d'écran des interfaces : ***Annexe 1 : Interface de saisie des données*** ***Annexe 2 : Interface de saisie des patients***

b) Les graphiques :

Une fois la simulation lancé nous avons un menu qui apparaît en haut à gauche de l'interface que voici :

Ce menu permet de choisir les différents graphiques que vous pouvez afficher.



En cliquant sur le bouton “Attente Infirmiers” vous obtiendrez le graphique du nombre d'infirmiers libre à chaque instant de la simulation.
(Voir ***Annexe 3 : Graphique Attente Infirmier***)

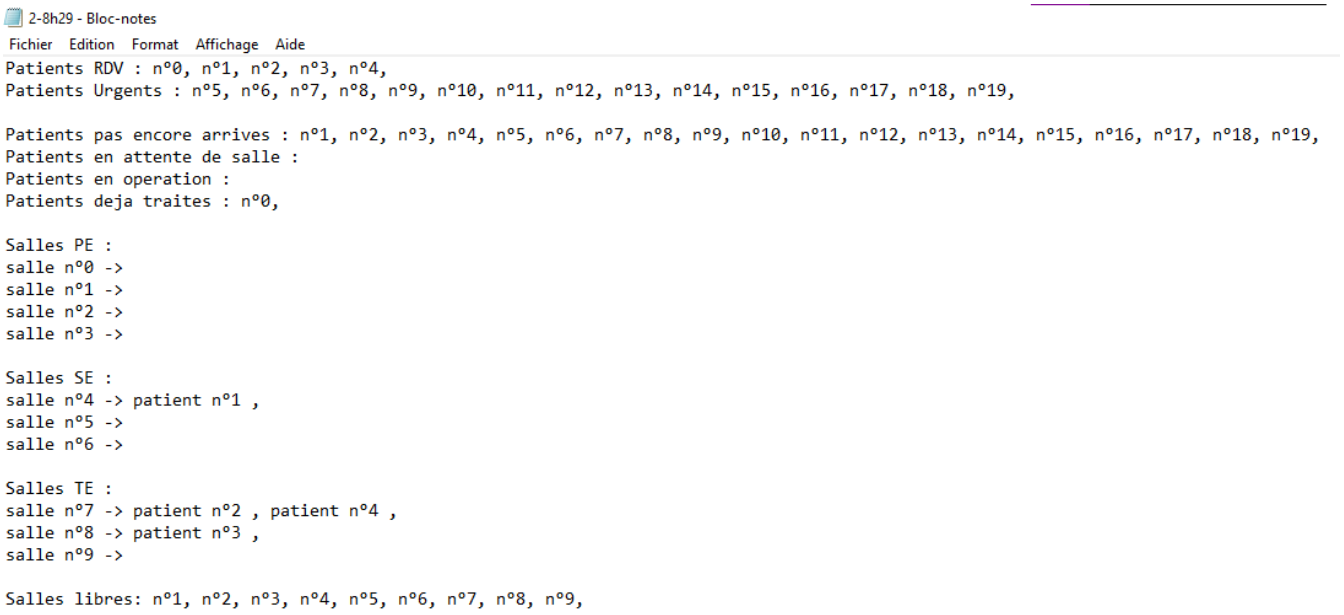
En cliquant sur le bouton “Attente chirurgien” vous obtiendrez le graphique du nombre de chirurgiens libres à chaque instant de la simulation.
(Voir ***Annexe 4: Graphique Attente de chirurgiens***)

En cliquant sur le bouton “Attente salles” vous pouvez cocher une ou plusieurs cases afin d’afficher en superposition les différentes courbes en fonction des salles pour voir le nombre de salles libres à chaque instant de la simulation.
(Voir ***Annexe 5 : Graphique d’attente des salles***)

En cliquant sur le bouton “Attente patients” et cochant “RDV” ou “urgent” vous pouvez afficher le nombre de patients de chaque type qui sont libres à chaque instant de la simulation. (Voir ***Annexe 6 : Graphique d’attente des patients***)

c) L’historique de planning

Capture d’écran d’un historique de planning :



2. Retours d'Expérience

Ce projet a été une expérience extrêmement enrichissante à divers égards. Tout d'abord, l'utilisation d'outils tels que GitHub pour la gestion collaborative du code, Astah pour la modélisation (diagramme de classe), Apache NetBeans comme environnement de développement intégré. De plus, nous avons pu découvrir ce qu’était une simulation d'événement discret, comment faire une carte d'événement et surtout un moteur de simulation.

A cela s'ajoute l'opportunité d'appliquer les concepts de génie logiciel vu en cours et d'en ressentir les résultats directement sur notre travail comme par exemple l'importance d'avoir des spécifications claires et précises, des commentaires, une convention de nommage compréhensible par son équipe que ce soit pour les variables où même pour les noms de classes surtout dans notre projet où elles sont nombreuses.

3. Perspectives d'amélioration ou d'évolution

Un premier aspect qui aurait pu être amélioré est l'utilisation de JUnit pour implémenter des tests unitaires, chose que nous n'avons pas faite et qui nous a fait perdre du temps.

En effet, le fait d'avoir implémenté l'historique des plannings nous a fait nous rendre compte que certaines règles de gestion ne fonctionnaient pas bien, ce que nous aurions pu nous rendre directement si nous avions implémenté des tests.

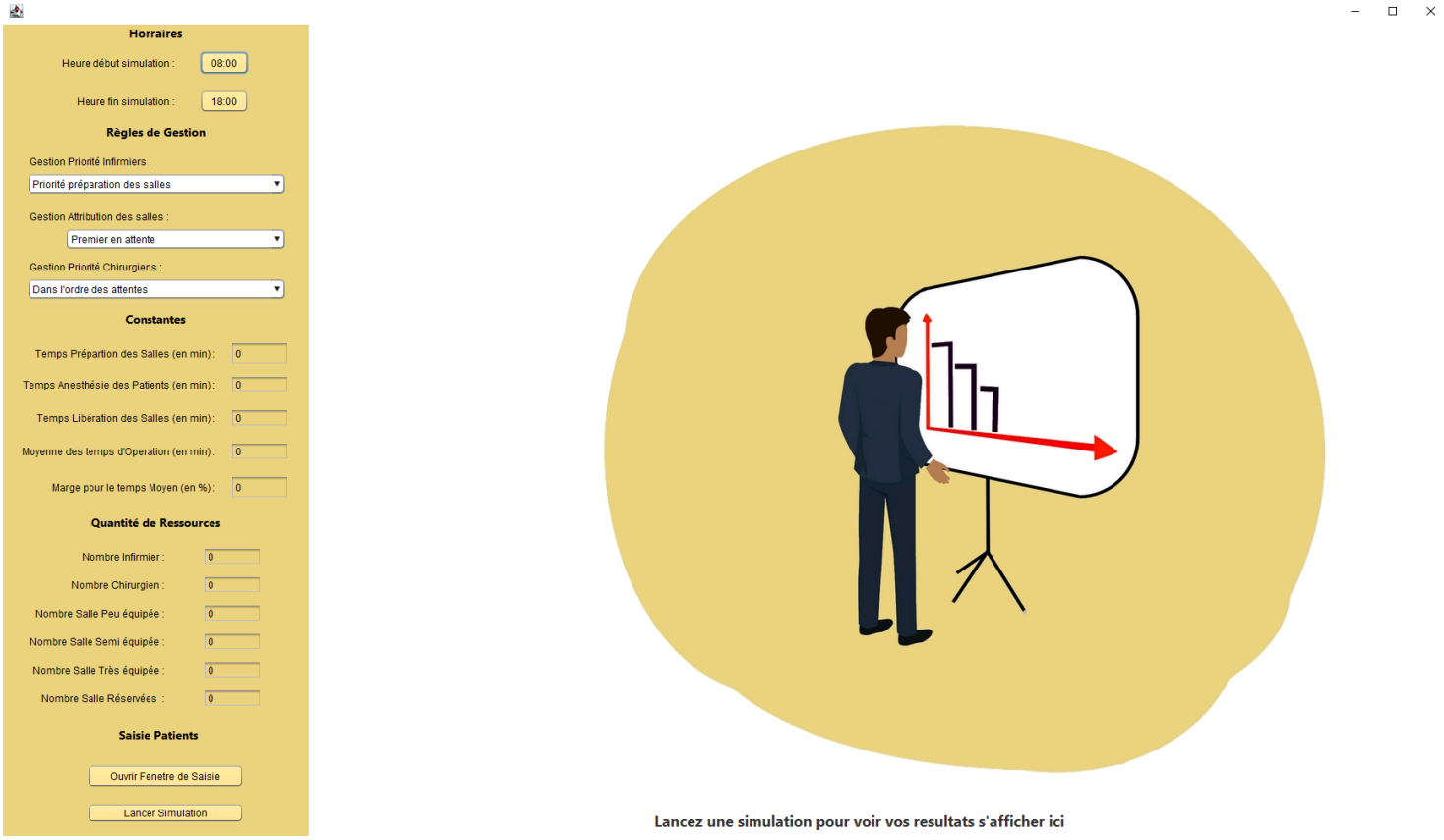
Un autre axe d'amélioration est d'améliorer l'extraction des données. En effet, si nous avions configuré Maven, nous aurions pu utiliser un package comme jackson pour l'extraction des données ce qui aurait permis un résultat plus propre. De plus, avec Maven, l'interface graphique aurait été peut être plus simple à faire en utilisant ProjectBuilder par exemple.

De plus, NetBeans nous a généré une énorme classe dont la lecture est compliquée. Nous aurions pu séparer cette classe en plusieurs sous classes pour améliorer la lisibilité.

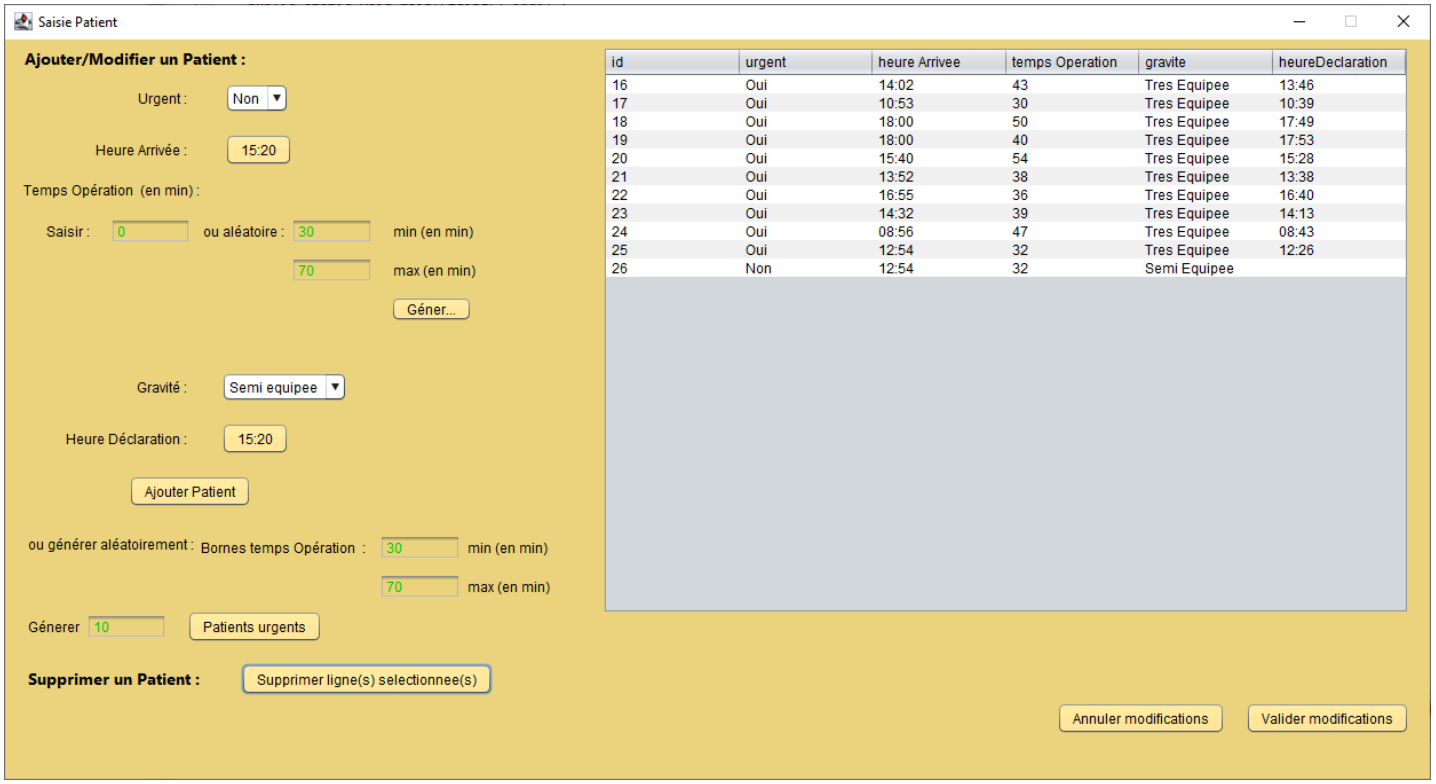
Sur le même aspect, nous pourrions améliorer nos graphes, qui sont pas très lisibles et qui suivent tous le même type (lineChart) en utilisant un meilleur package.

Un autre point d'amélioration aurait été de passer plus de temps sur la phase de spécification, ce qui aurait permis de repérer en amont certaines problématiques que nous avons identifiées en codant, comme à quel point les règles de gestion planning allaient être complexes.

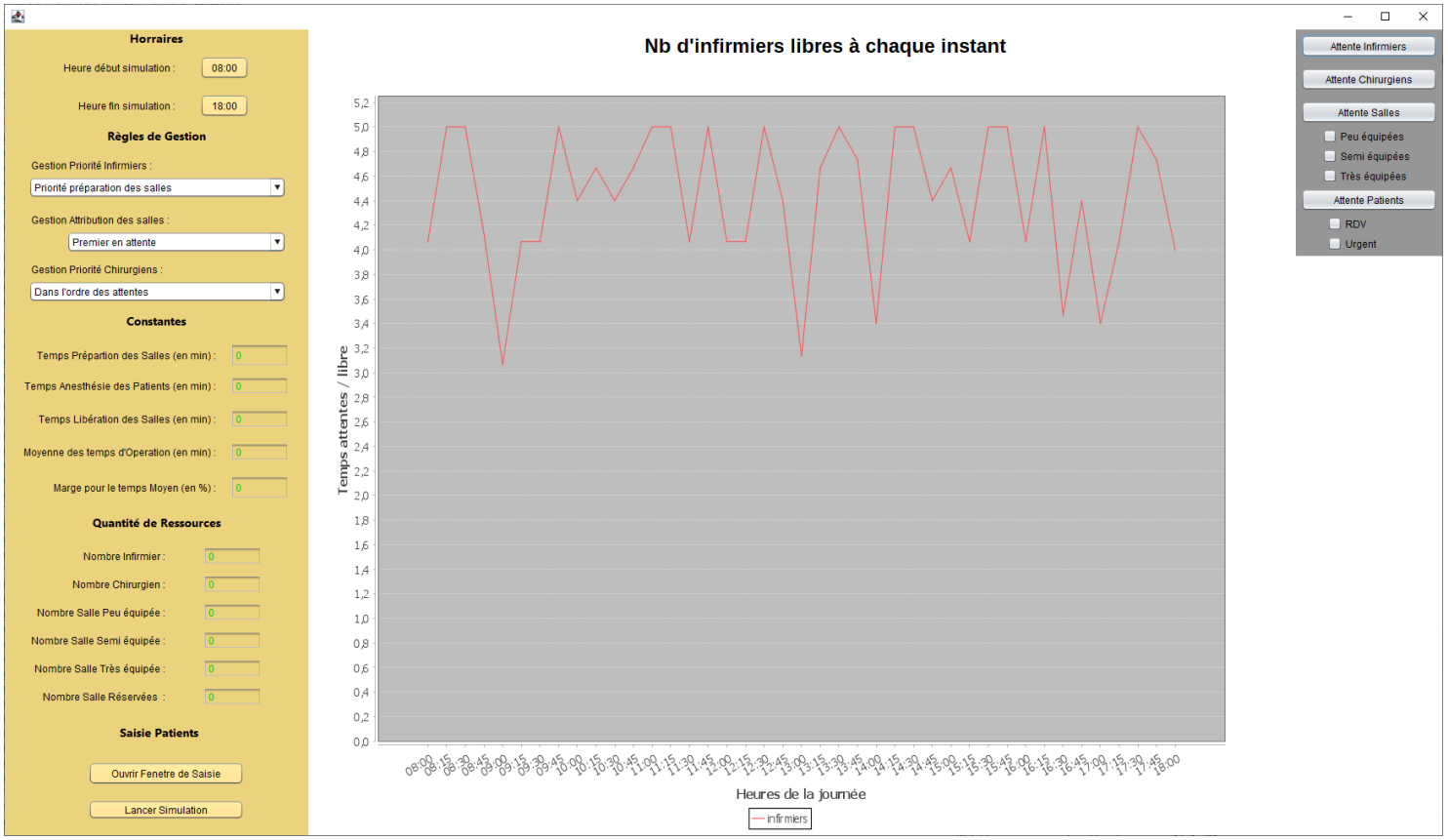
5 Annexes



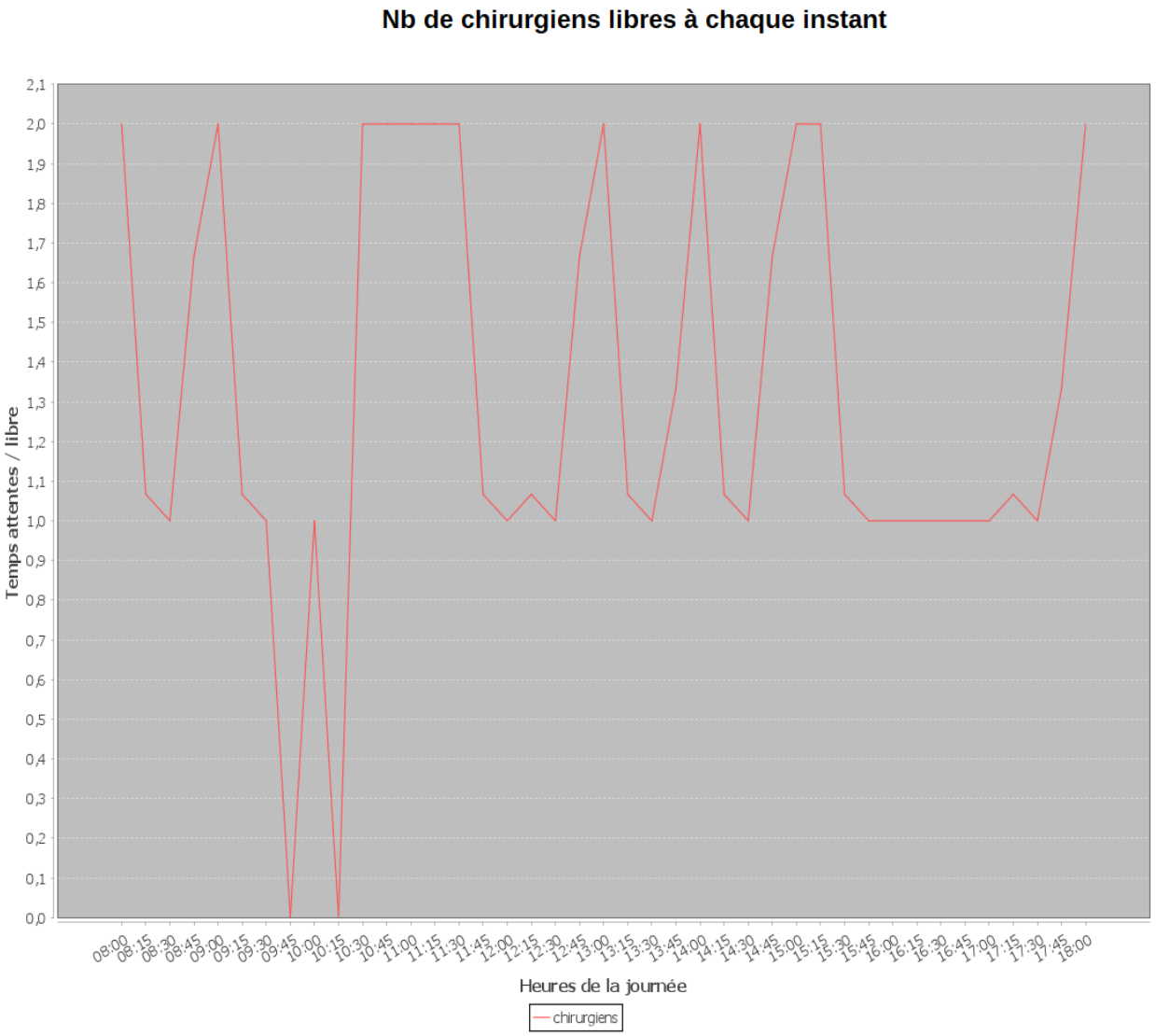
Annexe 1 : Interface de saisie des données



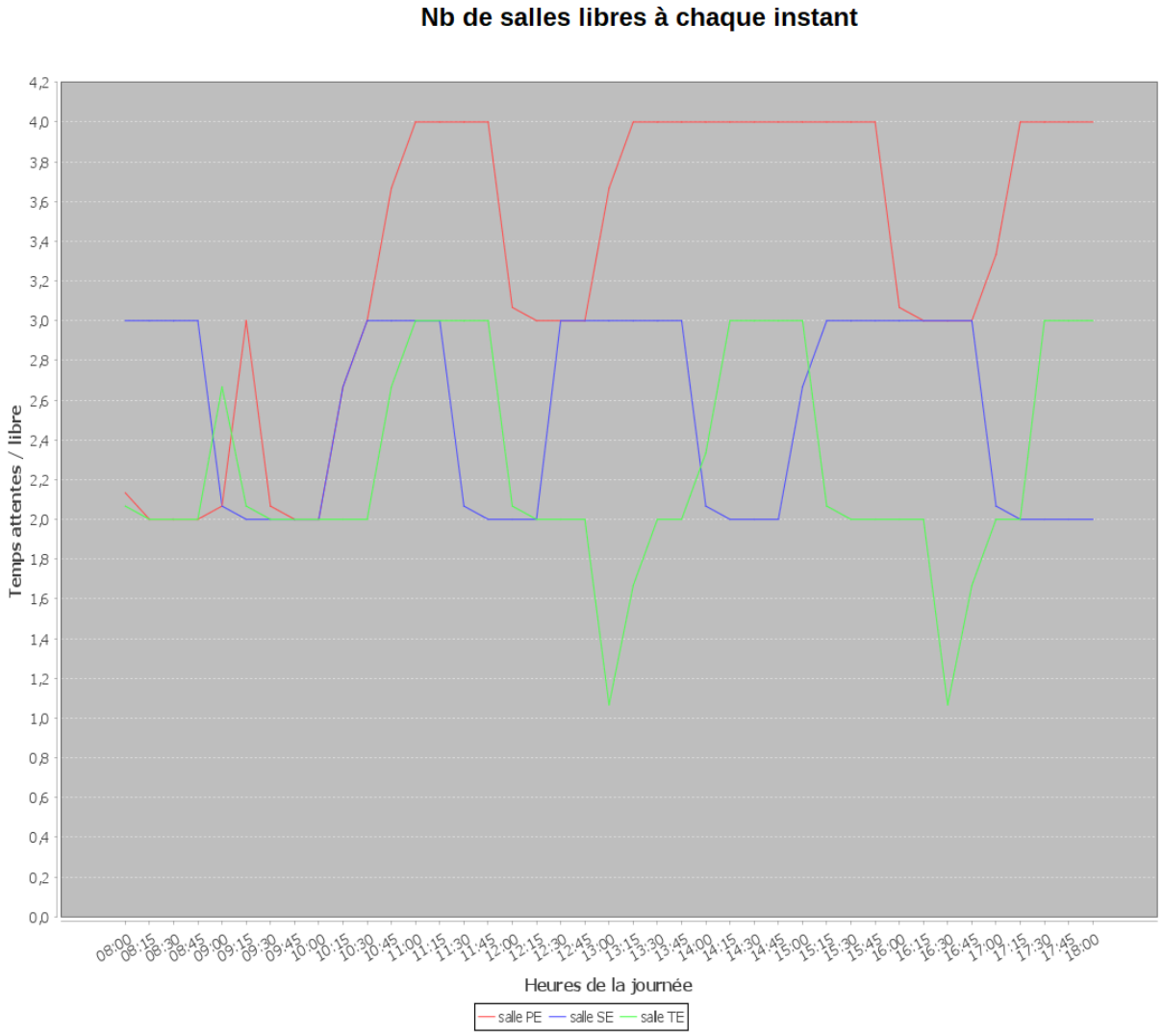
Annexe 2 : Interface de saisie des patients



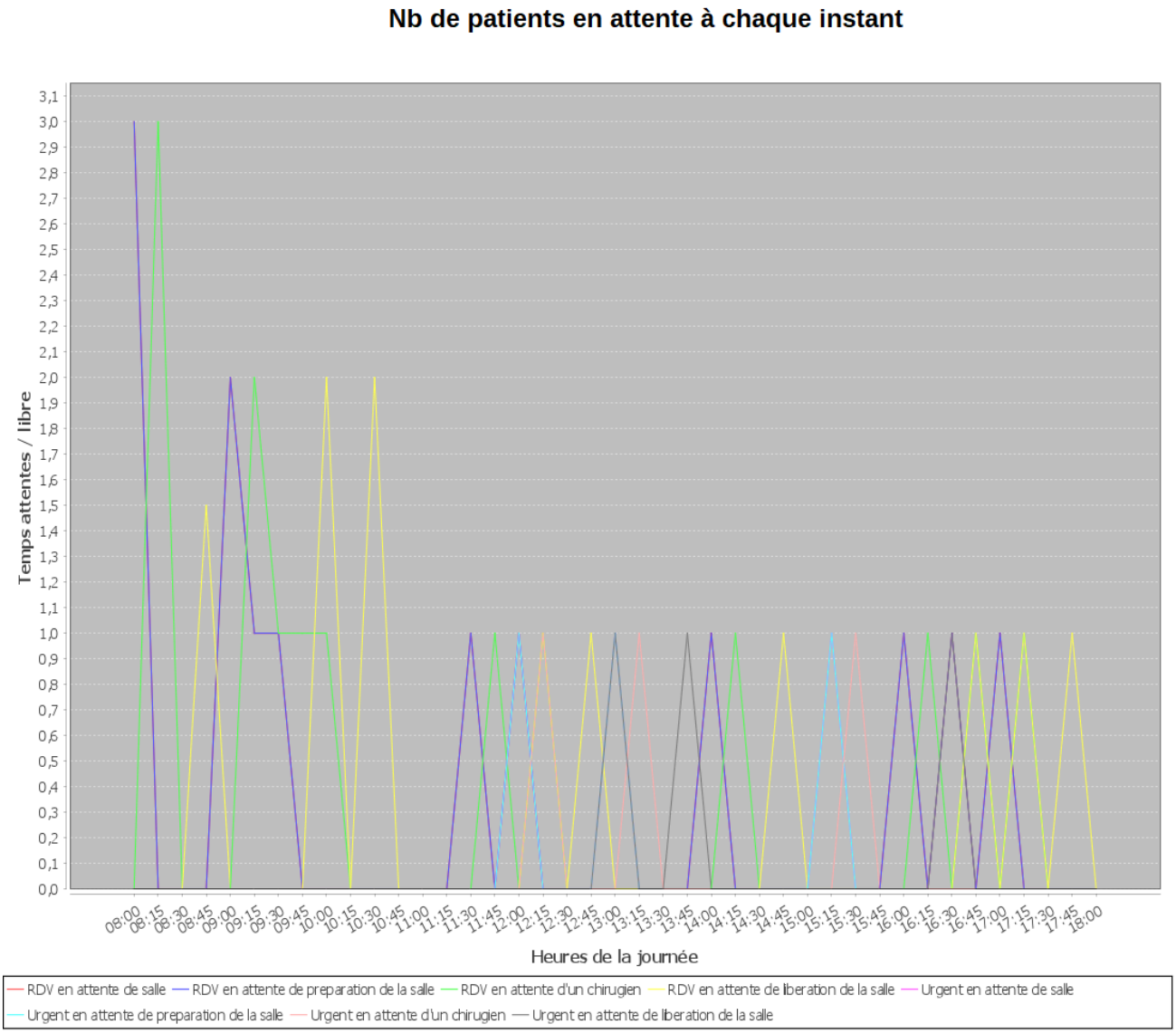
Annexe 3 : Graphique Attente Infirmiers



Annexe 4 : Graphique Attente Chirurgiens



Annexe 5 : Graphique Attente salles



Annexe 6 : Graphique Attente patients