IS1220 Object Oriented Software Design

Rapport de projet

SimErgy

Donatien CRIAUD
21/11/2017

Préface

Ce projet s'inscrit dans le cadre du cours IS1220 – Object Oriented Software Design dispensé à CentraleSupélec dans le cursus Ingénieur Centralien. Je suis actuellement en stage en entreprise à temps complet aussi, il a été convenu avec l'enseignant M. Ballarini que j'effectue ce projet seul. Compte tenu de cette situation particulière, je ne n'ai pas pu suivre les cours et les séances de travaux dirigés dispensés sur le campus de l'Ecole mais j'ai fait en sorte de pouvoir suivre ce cours à l'aide notamment des documents mis à disposition sur la plateforme Claroline. Mon organisation personnelle m'a conduit à avoir quasiment achevé la partie simulation de SimErgy au moment de la première soumission. Ma solution étant fonctionnelle, je n'ai pas utilisé le document concernant la modélisation d'événements mis à disposition sur Claroline. Ne pouvant bien évidemment pas décrire l'organisation de la réalisation du projet en binôme, je détaillerai tout de même dans ce rapport les détails de mon organisation personnelle pour réaliser ce projet. Je remercie tout lecteur de la compréhension qu'il pourra avoir de ma situation.

Les diagrammes de classe représentés dans ce documents sont présents dans le dossier *./model* et il est possible de les ouvrir en cliquant les images de ce dossier (chaque diagramme est linké à son document image). J'espère que cela permettra une plus grande lisibilité du rapport.

Table des matières

Introduction générale

Les centres hospitaliers sont actuellement sujets à la polémique. ARTE diffusait le 26 octobre 2017 un documentaire dans lequel un professionnel de santé en activité à l'hôpital Saint-Louis décrit « un système complètement fou ». Ce dernier parle d'une « pure gestion financière » des hôpitaux, délaissant l'humain. Dans un contexte où le gouvernement français affiche une volonté de redéfinir la politique d'attribution des subventions aux hôpitaux en favorisant l'efficacité des établissements, il paraît important d'étudier l'impact d'une telle politique et du fonctionnement d'un hôpital de manière générale. Un premier moyen d'analyse des performances d'un hôpital est la simulation, par laquelle il est possible de représenter un service et ses différents attributs. Outre ses médecins, un service comprend aussi une multitude de personnes et d'équipements médicaux. Simuler toutes ces parties prenantes pourrait permettre de prononcer des estimations de critères de performance, tout en cherchant à minimiser le coût du service et à alléger la charge de travail du personnel soignant. L'ingénieur a dans cette situation une position idéale pour répondre aux problématiques actuelles concernant les hôpitaux français.

Le projet *SimErgy* a pour prétention de développer un outil de simulation d'un service d'urgences. Ce service joue un rôle clé dans un hôpital dans le sens où il est le premier contact entre un patient et les services de santé dans le cas d'un accident. Ainsi, il doit être actif en permanence et apte à traiter tout patient arrivant de manière imprévue. L'outil *SimErgy* doit donc être apte à évaluer des critères de performance noyés dans un processus complexe. Pour cela, *SimErgy* doit représenter un service d'urgence et toutes les parties le composant à n'importe quel instant. L'interaction entre les patients, médecins, infirmiers, ressources etc… doit donc être implémentée.

Cette première partie de projet fut très intéressante d'un point de vue réflexion autour de l'architecture de ma solution. Il m'a été indispensable d'utiliser certains design patterns pour implémenter une solution qui soit aussi bien simple qu'efficace. En effet, j'ai gardé tout au long de ce projet la volonté d'encapsuler au maximum mes classes. Cela me fut notamment

utile lorsque j'ai dû rajouter une classe de ressource oubliée lors d'une première implémentation. J'ai porté une attention toute particulière à l'héritage et le polymorphisme de mes classes pour pouvoir par la suite designer le simulateur de la manière la plus générique possible. Cela m'a amené à modifier une première structure événementielle de *SimErgy* car je jugeais sa conception trop « ouverte » et insuffisamment encapsulée.

Architecture du noyau *SimErgy*

Dans cette partie je vais présenter de manière générale mes choix de conception concernant l'architecture de ma solution.

SimErgy possède un noyau composé de 4 principaux packages (voir Figure 1 ci-dessous).

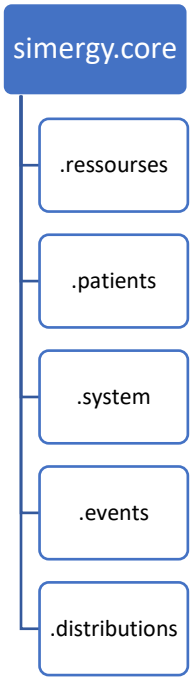

FIGURE 1 : ORGANISATION DE SIMERGY.CORE

Ressources

Le package simergy.core.ressources contient les classes relatives à la représentation de toute ressource présente dans un service d'urgences. J'ai considéré comme une ressource toute entité intervenant dans le service et qui ne soit pas un patient. Pour créer ce package, j'ai choisi d'utiliser au maximum la généricité. En effet, comme une ressource est un type très générique qui peut être étendu en divers types (humains et non humains) et que chacun de ceux-ci possède des propriétés différentes, il apparait logique d'utiliser un tel design. Cela m'a notamment permis par la suite de gérer les événements du service d'urgence de manière indépendante du type de ressource nécessaire pour la réalisation de ceux-ci. Nous verrons

plus tard que la classe représentant le service d'urgences gère ses ressources à la manière d'un design factory pattern.

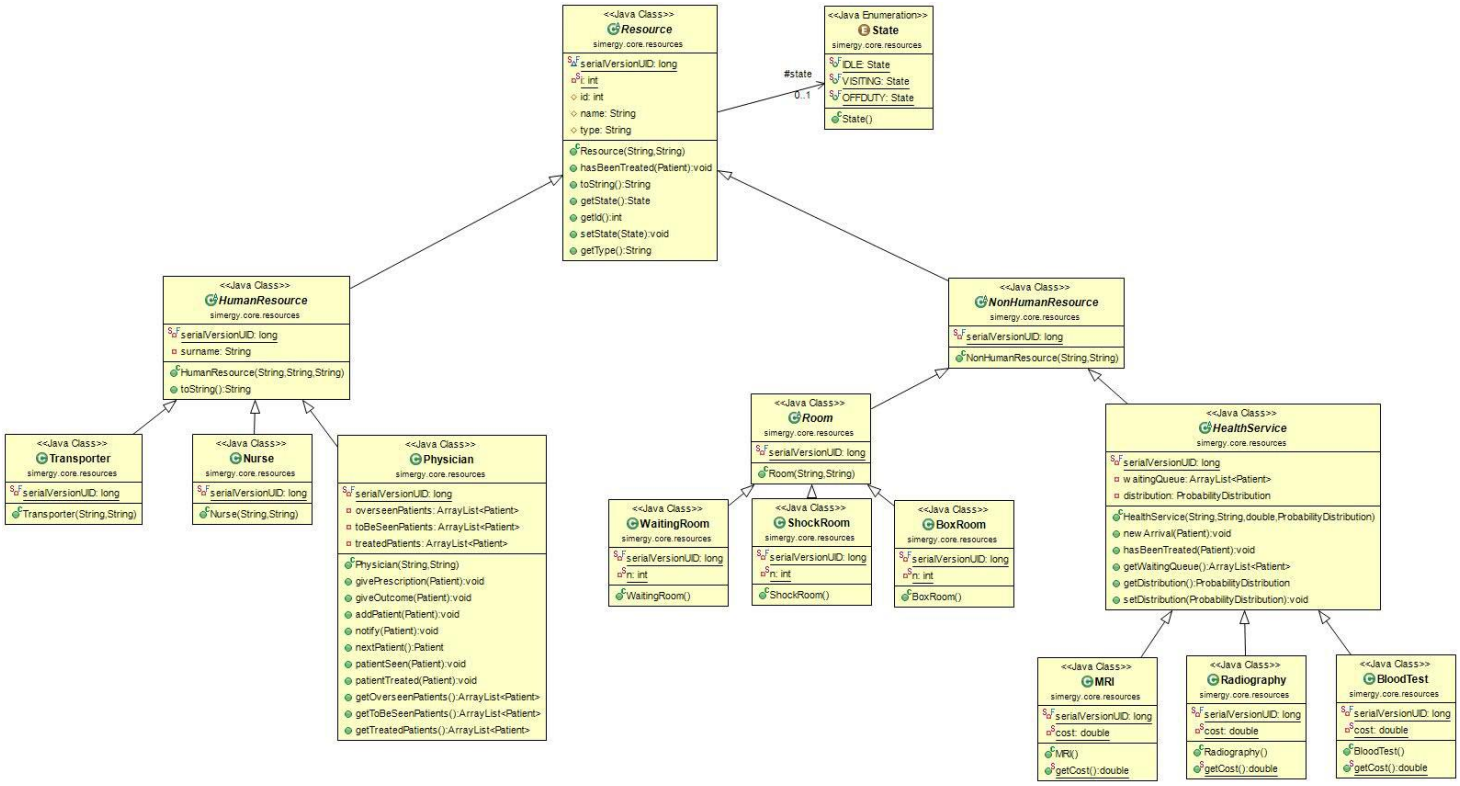

FIGURE 2 : GENERICITE DU PACKAGE SIMERGY.CORE.RESSOURCES

Ci-dessus, la représentation du package contenant l'architecture des classes ressources. Pour être plus précis sur la classe *Physician*, je pense qu'elle aurait pu étendre la classe *HealthService* dans le sens où la classe *Physician* possède un attribut *waitingqueue* comme les classes étendant la classe *HealthService*. Cependant, j'ai jugé plus approprié de laisser l'architecture comme telle dans le cas où d'autres paramètres relatifs à l'humain seraient à implémenter dans *Simergy*.

Patients

Le package simergy.core.patients contient un faible nombre de classes. Il est utilisé principalement pour créer les patients au moment de l'arrivée d'un patient dans le service.

C'est pourquoi ce package ne contient qu'une classe. Le reste est constitué d'enum utilisées pour décrire plusieurs attributs du patient (voir Figure 3 ci-dessous). Une particularité de ma solution est que je n'ai pas choisi de stocker les informations relatives à un patient dans cette classe mais plutôt dans la classe *Workflow*. Je reviendrait plus en détail sur le package simergy.event par la suite. Il faut juste retenir qu'un patient possède un attribut workflow qui stocke toutes les informations utiles durant le séjour dans le service.

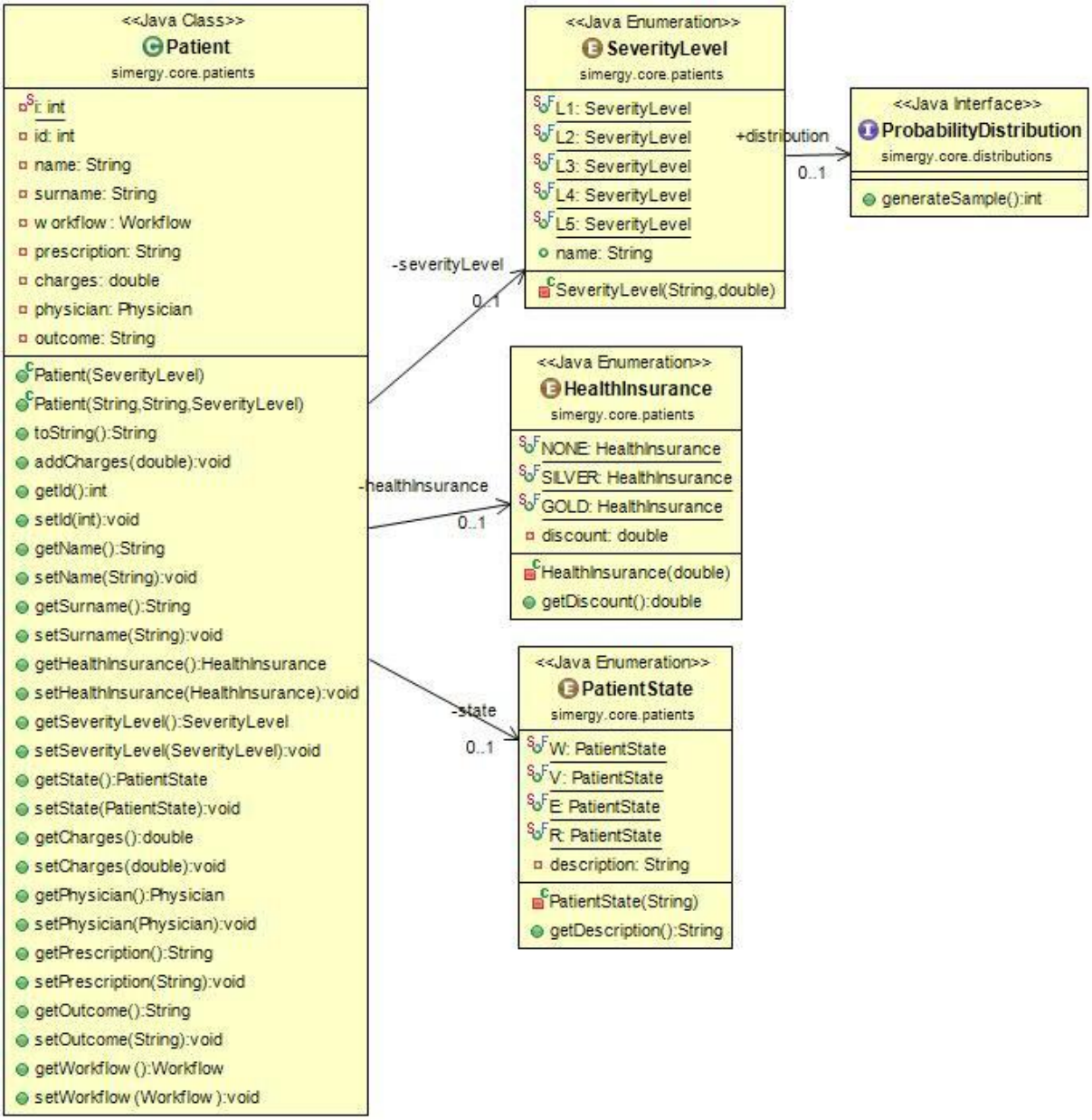

FIGURE 3 : ARCHITECTURE DU PACKAGE SIMERGY.CORE.PATIENTS

Le package simergy.core.distributions contient les outils utilisés lors de la génération aléatoire de durées d'événements ou d'arrivée de nouveaux patients dans le service. Je ne le détaillerai donc pas plus.

Système

Le dernier package du noyau *SimErgy* est celui du système. Il contient la classe *EmergencyDept* qui représente le service dans sa totalité. J'invite le lecteur à ouvrir le document suivant contenant le diagramme des classes général de ma solution : model\OverallClassDiagram.png. Pour revenir à la gestion des ressources, la classe EmergencyDept gère l'attribution de ressources à différents événements en utilisant un design pattern factory. En effet, la méthode *generateResources* ci-dessous ne fait appel qu'au type de la ressource (au format String) et n'a pas à utiliser de constructeur. Cette méthode sert à générer un HashMap destiné à contenir les ressources :

```
	/**
	 * Generates the HashMap containing all the resources.
	 * The index is made of Strings representing a canonical type of the
resource.
	 *
	 * @return the hash map containing all the ED's resources.
	 */
	public static HashMap<String,ArrayList<Resource>> generateResources(){
		HashMap<String,ArrayList<Resource>> resources = new
HashMap<String,ArrayList<Resource>>();
		resources.put("PHYSICIAN",new ArrayList<Resource>());
		resources.put("NURSE",new ArrayList<Resource>());
		resources.put("TRANSPORTER",new ArrayList<Resource>());
		resources.put("BOXROOM",new ArrayList<Resource>());
		resources.put("SHOCKROOM",new ArrayList<Resource>());
		resources.put("WAITINGROOM",new ArrayList<Resource>());
		resources.put("STRETCHER",new ArrayList<Resource>());
		resources.put("MRI",new ArrayList<Resource>());
		resources.put("BLOODTEST",new ArrayList<Resource>());
		resources.put("RADIOGRAPHY",new ArrayList<Resource>());
		return resources;
	}
```

Cette méthode statique créer une structure de type HashMap qui est initialisée avec un String contenant le type générique de la ressource et un ArrayList qui contiendra après ajout les ressources correspondantes. J'ai choisi de laisser la généricité `ArrayList<Resource>`

plutôt que d'écrire par exemple : `ArrayList<Physician>`. En effet, cela me permet par la suite d'utiliser la généricité du factory pattern pour attribuer les ressources :

```
/**
 * Gives a resource and set its state to VISITING.
 *
 * @param resourceType the resource type
 * @return the resource
 * @throws ResourceNotAvailableException if the resource is not available
 */
public Resource giveResource(String resourceType) throws
ResourceNotAvailableException{
	/*
	 * Check si la ressource demandée est disponible ou non
	 * Si oui, change son état et la renvoie
	 * Sinon, leve une exception
	 */
	ArrayList<Resource> askedRessource = resources.get(resourceType);
	for(Resource ressource : askedRessource){
		if(ressource.getState() == State.IDLE){
			ressource.setState(State.VISITING);
			return ressource;
		}
	}
	throw new ResourceNotAvailableException(resourceType);
}
```

Ici le principe d'encapsulation est utilisé afin de passer outre le type de chaque ressource. La méthode reçoit un String correspondant à la ressource demandée et renvoie cette ressource sans jamais considérer le type d'instance manipulée.

Un autre attribut important de la classe *EmergencyDept* est l'ArrayList workflows. Il contient les workflows se déroulant dans le service. Ces derniers représentent l'état d'un patient.

Evènements

Le dernier package du noyau *SimErgy* est celui contenant les événements. Quelques détails seront expliqués dans la partie destinée à la simulation. Il faut retenir que la classe workflow se place entre la classe EmergencyDept et la classe Event. Autrement dit, cette classe sert à stocker les informations relatives à un patient (notamment l'historique des

occurrences d'événements pour ce patient, les différentes informations utiles pour évaluer les critères de performances etc…). Le workflow n'intervient pas dans la demande d'attribution de ressource par un événement ni dans l'attribution de celles-ci par la classe EmergencyDept.

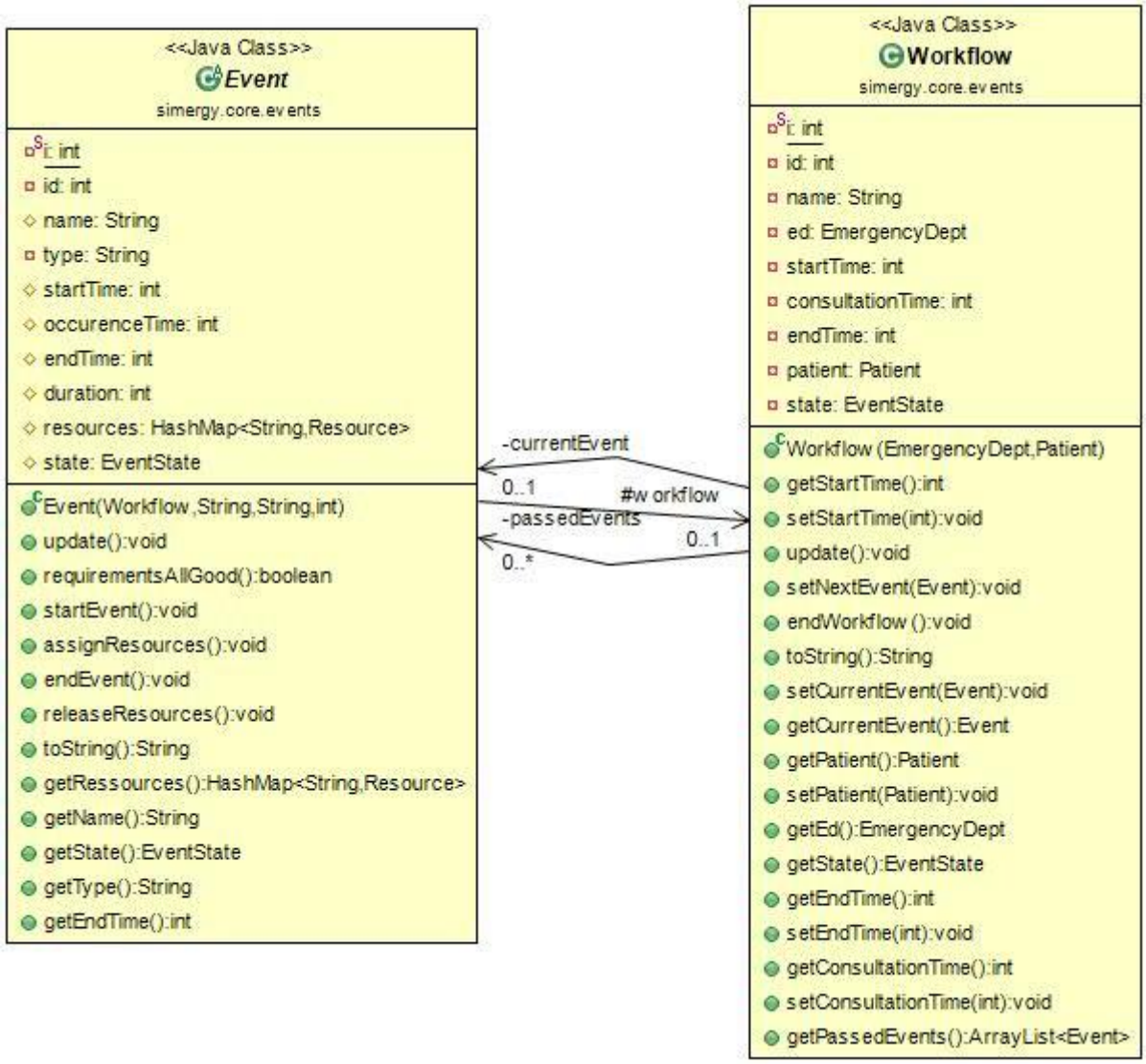

FIGURE 4 : RELATIONS ENTRE LES CLASSES WORKFLOW ET EVENT