

Génie Logiciel TD n° 9, 10

UML modèle d'interaction et modèle des états

Objectif du TD

Maîtriser les diagrammes de séquence, de collaboration, d'état-transitions en UML.

Exercice “ Téléphone ”

Dans cet exercice, on souhaite élaborer le modèle des états et d'interaction d'un appareil téléphonique.

- 1) Dessiner le diagramme de séquence correspondant au cas où un utilisateur (l'appelant) appelle un autre utilisateur (l'appelé). On supposera que le cas est normal (l'appelant fait un numéro valide et l'appelé n'est pas occupé et répond). On supposera que les entités du scénario sont l'appelant, l'appareil téléphonique de l'appelant, la ligne téléphonique, l'appareil téléphonique de l'appelé et enfin l'appelé. On supposera que l'appelant raccroche le premier.

(Cette question permet d'identifier les différents événements émis et reçus par un appareil téléphonique en provenance ou à destination de la ligne téléphonique et de l'utilisateur, que l'appareil soit appelé ou appelant.)

- 2) Compléter le scénario précédent avec le cas où l'appelé est occupé et où l'appelant effectue un faux numéro.
- 3) Dessiner un diagramme de collaboration autour de la classe 'AppareilTéléphonique' synthétisant tous les événements identifiés par les questions précédentes.
- 4) Dessiner le diagramme d'états-transitions de l'appareil téléphonique. Conseil : on pourra avoir le macro-état “ raccroché ” avec les sous-états “ silence ” et “ sonne ”, et le macro-état “ décroché ” avec les sous-états “ tonalitéBase ”, “ rechercheAutre ”, “ fauxNuméro ”, “ autreOccupé ”, “ autreSonne ”, “ autreRaccroché ” et “ communication ”.

Exercice “ Tamaguchi ”

On souhaite modéliser des “ Tamaguchi ” avec le modèle d'interaction et les modèles de états de UML.

Un Tamaguchi en état normal n'a pas faim pendant un certain temps (appelé temps d'autonomie). Au bout de ce temps, le Tamaguchi a faim et il pleure. Pour lui donner à manger, l'utilisateur du Tamaguchi le met à table et le Tamaguchi s'arrête de pleurer. Un Tamaguchi mange pendant un certain temps (appelé temps de restauration). Au bout de ce temps, il se remet à pleurer. Il pleure jusqu'à ce que l'utilisateur le sorte de table. Quand il sort de table, le Tamaguchi revient dans l'état normal... et ainsi de suite tant que le Tamaguchi ne meurt pas. Si le Tamaguchi pleure plus de 5 minutes d'affilée, il meurt.

On suppose que les événements émis par le Tamaguchi vers l'utilisateur sont : “ avoir faim ”, “ ne plus avoir faim ”, “ mourir ” et que les événements émis par l'utilisateur et reçus par le Tamaguchi sont “ être mis à table ” et “ sortir de table ”.

- 1) Dessiner un automate à 5 états modélisant le comportement du Tamaguchi. On utilisera les noms “ pas faim pleure pas ”, “ faim pleure ”, “ à table pleure pas ”, “ à table pleure ” et “ mort ” pour ces 5 états. On fera figurer les événements nommés à la question précédente sur les transitions de l'automate. Une flèche vers le haut (resp. bas) à côté d'un événement indiquera que l'événement est émis (resp. reçu) par le Tamaguchi.

2) Un Tamaguchi est une agrégation de matériels : 1 horloge, 1 bip, 2 boutons. Dessiner le diagramme d'agrégation du Tamaguchi.

3) Quand l'utilisateur appuie sur le bouton " a table " (resp. " sortie de table "), le Tamaguchi reçoit l'événement correspondant à sa mise à table (resp. à sa sortie de table). L'horloge se présente sous forme d'une instance de la classe Horloge avec une méthode lancerTempo(int durée). Si le Tamaguchi appelle la méthode lancerTempo(durée) à un instant t, il recevra un événement " tempo écoulée " à l'instant t+durée.

Préciser l'automate de la question I-1) en rajoutant les appels à la méthode lancerTempo(int t) et les événements " tempo écoulée " reçus par le Tamaguchi.

4) Le bip se présente sous forme d'une instance de la classe Bip. Si le Tamaguchi appelle la méthode déclencherPleurs(), le bip commence à émettre des pleurs sans interruption. Si le Tamaguchi appelle la méthode arreterPleurs(), le bip s'arrête. Les méthodes lancerTempo(durée), déclencherPleurs() et arreterPleurs() s'exécutent instantanément. Au sens donné par UML, sont-elles des activités ou des actions ?

5) Préciser à nouveau l'automate en ajoutant les appels aux méthodes déclencherPleurs() et arreterPleurs() sous forme d'actions d'entrée ou de sortie.

Exercice " Thread "

En suivant le formalisme du modèle dynamique, dessiner un diagramme d'états-transition correspondant à la dynamique d'un " thread " définie de la manière suivante :

le thread est :

- " non démarré " au début,
- " en cours " lorsqu'il possède toutes ses ressources applicatives plus le processeur,
- " en attente " lorsqu'il lui manque une ressource applicative,
- " prêt " lorsqu'il a toutes ses ressources applicatives et pas le processeur,
- " terminé " lorsqu'il a terminé son exécution.

On supposera que les événements reçus par le thread sont :

" début ", " ressource attendue ", " ressource OK ", " processeur OK ", " fin ".

" début " correspond au démarrage du thread (start en java, execlv en Unix, ...). avant la réception de " début ", le thread est " non démarré ".

" ressource attendue " correspond à l'appel d'une réservation de ressource lorsque celle-ci n'est pas disponible.

" ressource OK " correspond à la libération d'une ressource par un autre thread et donc à la réservation effective de la ressource par le thread qui l'attendait.

" processeur OK " correspond à la libération du processeur par un autre thread et à l'utilisation effective du processeur par le thread qui l'attendait.

" fin " correspond soit à l'exécution de la dernière instruction du programme exécuté par le thread soit à l'envoi d'un événement pour tuer définitivement le thread. Sur réception de " fin ", le thread devient " terminé ".

On supposera qu'un thread n'envoie pas d'événement. Il ne fait que les recevoir.

Exercice “ Ascenseur ”

Le but de cet examen est de modéliser avec UML le fonctionnement d’un logiciel “Ascenseur” commandant un système d’ascenseur. Le système d’ascenseur est composé de matériels : l’ascenseur, les étages, les portes et les boutons. Chaque instance UML correspond à un objet matériel unique et permet de le commander. Les questions 1 et 2 sont préliminaires, elles demandent la construction du modèle des classes UML du logiciel “Ascenseur”. Les questions 3, 4 et 5 sont les plus intéressantes, elles demandent la construction du modèle des états UML du logiciel “Ascenseur”.

Question 1

Le constructeur d’ascenseur fait la description suivante:

“Un *ascenseur* <sert> tous les *étages* d’un immeuble. Grâce à des *boutons* (les *boutons d’étage* <situés> à l’étage et les *boutons d’ascenseur* <situés> dans l’ascenseur lui-même), certains étages <demandent> l’ascenseur. Un bouton <correspond à> un étage. Il y a deux types de *portes*: la *porte d’ascenseur* <située dans> l’ascenseur lui-même, et les *portes d’étage* <situées à> l’étage. Quand l’ascenseur <est arrêté> à un étage, la porte d’ascenseur est <en face de> la porte d’étage.”

En supposant que les classes correspondent aux mots *en italique* et les associations aux mots <entre crochets>, dessiner le **diagramme de généralisation UML** (avec une classe *Objet* générale) et le **diagramme de classes UML** avec les ordres de multiplicité convenables. On ne placera aucun attribut et aucune méthode sur ces diagrammes.

Question 2

On suppose que les classes suivent la description UML partielle donnée en annexe. Compléter la **description des classes UML avec attributs associatifs**, cela de manière cohérente avec le diagramme de classes de la question précédente.

On suppose que les événements émis et reçus par les classes du modèle d’interaction UML sont:

- “U appuie B” (reçu par le bouton sur lequel l’utilisateur appuie),
- “B étage demandé A” (envoyé par le bouton vers l’ascenseur),
- “A étage satisfait E” (envoyé par l’ascenseur à l’étage lorsque l’ascenseur est arrivé à l’étage),
- “E étage satisfait B” (envoyé par l’étage à ses boutons),
- “A arrivé arrêté P” (envoyé par l’ascenseur à la porte lorsque l’ascenseur est arrivé à l’étage),
- “A es-tu bien fermée ? P” (envoyé par l’ascenseur à la porte lorsque l’ascenseur veut repartir),
- “P bien fermée A” (envoyé par la porte à l’ascenseur lorsque sa fermeture est terminée),
- “H délai écoulé P” (envoyé par l’heure à la porte lorsque le délai de porte ouverte est écoulé),
- “A sens montée A” (envoyé par Ascenseur.determinerSens(), cf annexe),
- “A sens descente A” (idem),
- “E passage A” (envoyé par un étage à l’ascenseur lorsque celui-ci passe à l’étage).

“X e Y” signifie que l’événement “e” est envoyé par la classe X vers la classe Y.

Question 3

Bouton.état peut prendre 2 valeurs (allumé, éteint). Dessiner le **diagramme d’états-transitions UML** de l’attribut Bouton.état.

Indications: on placera les appels des méthodes allumer() et éteindre() et un envoi adéquat d’événement en action d’entrée. On placera les deux événements adéquats sur les deux transitions du diagramme.

Question 4

Porte.état peut prendre 4 valeurs (fermée, ouverture, ouverte, fermeture). Une porte ouverte démarre sa fermeture au bout d'un certain délai, identique pour toutes les portes. Dessiner le **diagramme d'états-transitions UML** de l'attribut Porte.état.

Indications : on placera l'appel de lancerDélaiOuverte() et l'envoi de "P bien fermée A" en actions d'entrée des états adéquats. ouvrir() et fermer() seront placées en activités se terminant par des transitions automatiques. On placera les événements reçus sur les autres transitions.

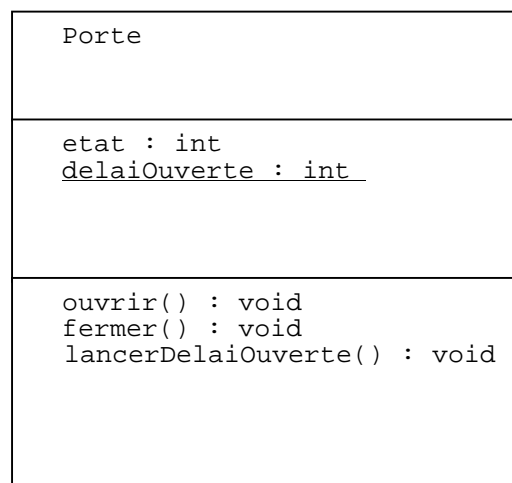
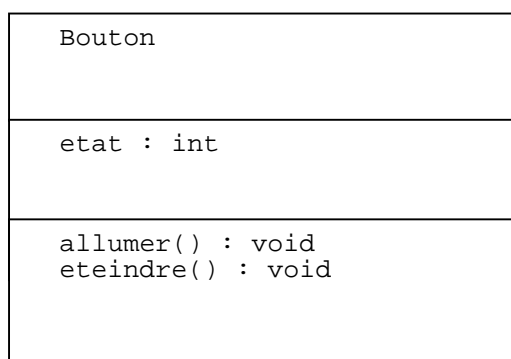
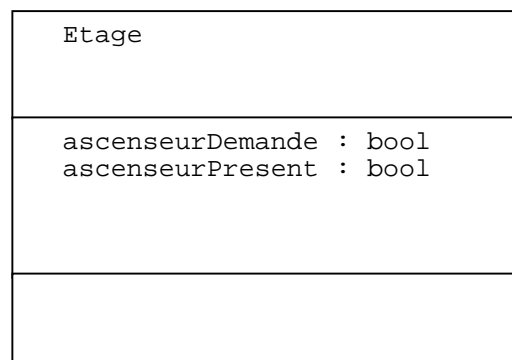
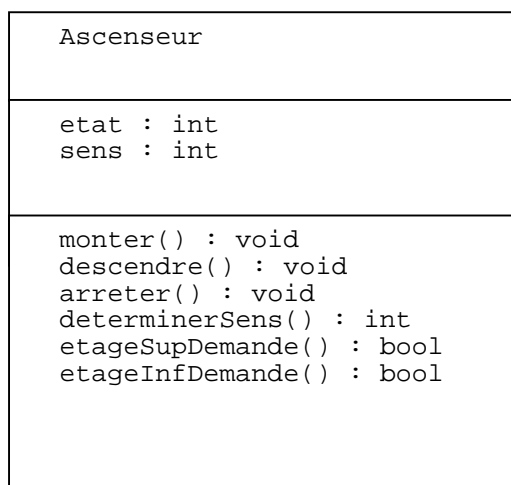
Question 5

Ascenseur.état décrit l'état instantané de l'ascenseur. Il peut prendre 3 valeurs (arrêté, montée, descente).

Dessiner le **diagramme d'états-transitions UML** de l'attribut Ascenseur.état.

Indications : on placera les événements "A sens montée A" et "A sens descente A" sur des transitions normales et l'événement "E passage A" sur des transitions avec condition à déterminer. Dans l'état arrêt, on placera des actions d'entrée et des actions internes liées aux événements "P bien fermée A" et "B étage demandé A". On placera les actions déterminerSens() et arrêter() et les activités monter() et descendre().

Description partielle des classes



Description partielle des méthodes

actions UML

- La méthode `Ascenseur.determinerSens()` est appelée lorsque, l'ascenseur étant arrêté, il reçoit l'événement "P bien fermée A". Son objectif est de calculer l'attribut `Ascenseur.sens` et d'envoyer les événements "A sens montée A" ou "A sens descente A". `Ascenseur.sens` décrit le sens global de l'ascenseur. Par exemple, si l'ascenseur est arrêté au deuxième étage en venant du 1er étage, alors `sens` vaut montée. Si l'ascenseur est arrêté au deuxième étage depuis longtemps sans aucune demande d'étage, alors `sens` vaut rien. `Ascenseur.sens` peut prendre 3 valeurs (montée, descente, rien). Le pseudo-code ci-dessous est le corps de `Ascenseur.determinerSens()`.

```

if ((sens == montée) or (sens == rien)) {
    if (ilExisteUnEtageSuperieurDemande() == true) {
        sens = montée ; envoyer "A sens montée A" ; return sens;
    }
    if (ilExisteUnEtageInferieurDemande() == true) {
        sens = descente ; envoyer "A sens descente A" ; return sens;
    }
    sens = rien ; return sens;
}
if (sens == descente) {
    if (ilExisteUnEtageInferieurDemande() == true) {
        sens = descente ; envoyer "A sens descente A" ; return sens;
    }
    if (ilExisteUnEtageSuperieurDemande() == true) {
        sens = montée ; envoyer "A sens montée A" ; return sens;
    }
    sens = rien ; return sens;
}

```

- La méthode `Porte.lancerDelaiOuverte()` lance un délai de longueur `Porte.delaiOuverte` au bout duquel l'événement "P délai porte ouverte écoulé P" sera envoyé à la porte elle-même.
- La méthode `Bouton.allumer()` permet d'allumer la lumière associée au bouton.
- La méthode `Bouton.eteindre()` permet d'éteindre la lumière associée au bouton.
- La méthode `Ascenseur.arreter()` arrête l'ascenseur à l'étage.
- La méthode `Ascenseur.etageSupDemande()` retourne `true` si un étage strictement inférieur est demandé.
- La méthode `Ascenseur.etageInfDemande()` retourne `true` si un étage strictement supérieur est demandé.

activités UML

- La méthode `Ascenseur.monter()` fait monter l'ascenseur.
- La méthode `Ascenseur.descendre()` fait descendre l'ascenseur.
- La méthode `Porte.ouvrir()` ouvre la porte et se termine par une transition automatique.
- La méthode `Porte.fermer()` ferme la porte et se termine par une transition automatique.

Exercice “ Windows-Linux ”

Dessiner un diagramme de séquence et diagramme d'états-transitions correspondant à un étudiant de l'Ufr de Mathématiques et Informatique souhaitant travailler en Windows sur un ordinateur connecté sous Linux. L'étudiant doit donc redémarrer l'ordinateur, attendre le menu de choix NT ou Linux, taper NT avant que la temporisation de 30 secondes ne soit écoulée, taper CTRL-ALT-SUP pour ouvrir une session, son login, son mot de passe qui doit être vérifié, puis travailler. Ensuite l'étudiant souhaite revenir travailler en Linux. Dessiner un diagramme de séquence et un diagramme d'états-transitions correspondant.

Exercice “ Loge Démon ”

Cet exercice sur le modèle dynamique est la suite de l'exercice 4 “ DEMON LOGE ” sur le placement des informations dont il faut avoir fait le début pour être capable de commencer l'exercice présent. Dans l'exercice présent, on veut modéliser la dynamique du processus d'occupation d'une “ loge ” par les “ démons ” avec un diagramme d'états-transitions UML.

Les attributs d'une loge

Une loge possède un attribut ‘état’. Pendant le processus d'occupation, la loge est dans l'état ‘I’, c'est-à-dire indéterminée. L'état ‘I’ est un regroupement des sous-états ‘IV’(indéterminée et vide), ‘IB’ (indéterminée mais avec des bosons occupants), ‘IF’ (indéterminée mais avec un fermion occupant). A la fin du processus, la loge devient soit ‘V’ (vide), soit ‘B’ (remplie de bosons), soit ‘F’ (occupée par un fermion).

Une loge possède aussi un degré d'occupation, nommé ‘do’, exprimé en %. ‘do’ est remis à 0 à chaque fois que l'occupant de la loge est délogé.

Les événements temporels

Une loge possède un temps d'indétermination ‘TI’ égal à 100 secondes. Au bout de ce temps ‘TI’, un événement “ TI ” est reçu par la loge et l'état de la loge devient ‘B’, ‘F’ ou ‘V’ selon qu'il était ‘IB’, ‘IF’ ou ‘IV’ juste avant la réception de “ TI ”. Chaque seconde, un événement “ t ” est reçu par la loge. ‘do’ est incrémenté de 1 sur réception de l'événement “ t ” lorsque la loge n'est pas vide. Si ‘do’ est supérieur à 100 lorsqu'un événement “ t ” est reçu, l'état de la loge devient ‘B’ ou ‘F’ selon qu'il était ‘IB’ ou ‘IF’ juste avant.

L'énergie d'occupation et de délogement

Un ensemble de bosons ou un fermion possèdent une énergie. Quand un ensemble de bosons ou un fermion est dans une loge, cette énergie est l'énergie d'occupation de la loge ‘eo’. Quand un ensemble de bosons ou un fermion n'est pas dans une loge, cette énergie est l'énergie de délogement ‘ed’.

Les événements démoniaques

Quand un fermion veut rentrer dans une loge, la loge reçoit un événement “ f ”. Quand un ensemble de bosons veut rentrer dans une loge, la loge reçoit un événement “ b ”. Quand un fermion ou un ensemble de bosons veulent rentrer dans une loge vide, ils y rentrent sans problème. Quand un ensemble de bosons veut rentrer dans une loge contenant déjà des bosons, ils y rentrent aussi sans problème. Quand un fermion veut déloger un fermion ou un ensemble de bosons ou quand un ensemble de bosons veut déloger un fermion, l'énergie d'occupation de la loge est comparée à l'énergie de délogement du délogeur. Si cette dernière est supérieure, l'occupant est délogé et la loge devient occupée par le délogeur et ‘do’ et remis à 0. Si elle est égale, l'occupant est délogé et la loge devient vide. Si elle est inférieure, l'occupant reste dans la loge et le degré d'occupation est doublé.

Dessiner un diagramme d'états-transitions UML qui représente le processus d'occupation d'une loge décrit ci-dessus. On fera apparaître explicitement, les états, les sous-états, les événements, les actions d'entrée ou de sortie et les conditions de transitions.