Projet Qualité Logicielle : Modèles des opérations

Équipe H4314 – 9 février 2014

Chef de projet
Jean-Marie COMETS

Membres de l'équipe
Franck MPEMBA BONI: responsable qualité
Pierre TURPIN
Samuel CARENSAC
Grégoire CATTAN
Van PHAN HAU
Iler VIRARAGAVANE

Table des matières

1 Système Balance

1.1 Opération : $SystemeBalance :: demandeMesure()$

Acteur du MdE : CapteurBalance
Cas d'utilisation : Obtenir Mesure
Messages : * CapteurBalance : :mesure Pré-conditions :

Post-conditions :

```
if self.assign != None then
self.mesure = 0.8 * self.mesure + 0.2 * mesure
if self.mesure < self.seuilMin then
        self.SystemeWeb^alarm(self.quantite)
wait(1 minute)
self.demandeMesure()
```

1.2 Opération : $SystemeBalance :: scan()$

Acteur du MdE : Système externe
Cas d'utilisation : Installation balance
Messages : * SystèmeExterne : :scan(clé) Pré-conditions :

Post-conditions :

```
if self.cleClient = cle then
sender^confirmScan
self.synchro = True
self.SystemeWeb = sender
```

1.3 Opération : $SystemeBalance :: assignBal()$

Acteur du MdE : Système web
Cas d'utilisation : Assigner un produit à une balance
Messages : * SystèmeWeb : :assignBal(Prod, Seuil, Quantité) Pré-conditions :

```
self.synchro = True
```

Post-conditions :

```
sender = self.SystemeWeb
if Seuil >= 0 and Quantite > 0 then
self.assign = Prod
self.seuilMin = Seuil
self.quantite = Quantite and
self.mesure = 0
sender^confirmAssign
else
        sender^errorAssign
```

1.4 Opération : $SystemeBalance :: configBal()$

Acteur du MdE : Système web
Cas d'utilisation : Modifier un paramètre d'une balance
Messages : * SystèmeWeb : :configBal(Seuil, Quantité) Pré-conditions :

```
self.synchro = True and self.produit != None
```

Post-conditions :

```
if Seuil >= 0 and Quantite > 0 then
self.seuilMin = Seuil
and self.quantite = Quantite and
self.mesure = 0
sender^confirmConfig
else
        sender^errorConfig
```

2 Receptacle

2.1 Opération : $Receptacle :: rentrerCode(n : number)$

Acteur du MdE : PaveNum
Cas d'utilisation : Réception d'une commande
Messages : * porteLaterale : :debloquerPorte * panneauAffichage : :msgModifierCode, msgCodeIncorrect * diode : :diodeVerte, diodeRouge Pré-conditions :

Post-conditions :

```
if n = self.codeAdmin then
self.mode = EnumMode::Admin
self.porteLaterale^debloquerPorte()
self.panneauAffichage^msgModifierCode(true)
self.diode^diodeVerte()
else if n = self.codeUser then
self.mode = EnumMode::User
self.porteLaterale^debloquerPorte()
self.panneauAffichage^msgModifierCode(true)
self.diode^diodeVerte()
else
self.panneauAffichage^msgCodeIncorrect(true)
self.diode^diodeRouge()
```

2.2 Opération : $Receptacle :: modifierCode(n : number)$

Acteur du MdE : PaveNum
Cas d'utilisation : Réception d'une commande
Messages : * panneauAffichage : :msgModifiercode Pré-conditions :

```
not(self.porteLaterale.closedState)
```

Post-conditions :

```
        if self.mode == EnumMode::Admin then
                self.codeAdmin = n
self.panneauAffichage^msgModifierCode(false)
        if self.mode == EnumMode::User then
                self.codeUser = n
self.panneauAffichage^msgModifierCode(false)
```

2.3 Opération : $Receptacle :: verificationBadge(b : badge)$

Acteur du MdE : capteurBadge
Cas d'utilisation : Réception d'une commande
Messages : * porteLaterale : :debloquerPorte * panneauAffichage : :msgModifierCode, msgBadgeIncorrect * diode : :diodeVerte, diodeRouge Pré-conditions :

Post-conditions :

```
if b == self.idBadgeAdmin then
self.mode = EnumMode::Admin
self.porteLaterale^debloquerPorte()
self.panneauAffichage^msgModifierCode(true)
self.diode^diodeVerte()
else if b == self.idBadgeUser then
self.mode = EnumMode::User
self.porteLaterale^debloquerPorte()
self.panneauAffichage^msgModifierCode(true)
self.diode^diodeVerte()
else
self.panneauAffichage^msgBadgeIncorrect(true)
self.diode^diodeRouge()
```

2.4 Opération : $Receptacle :: verificationColis(idDestinataire : IdRFID)$

Acteur du MdE : systemeDrone
Cas d'utilisation : Livrer une commande
Messages : * systemeDrone : :colisCorrect Pré-conditions :

Post-conditions :

```
if self.id == idDestinataire then
        self.systemeDrone^colisCorrect(true)
else
        self.systemeDrone^colisCorrect(false)
```

2.5 Opération : $Receptacle :: ouverturePanneau()$

Acteur du MdE : systemeDrone
Cas d'utilisation : Livrer une commande
Messages : * panneauSuperieur : :ouvrirPanneau * systemeDrone : :sendConfirmation Pré-conditions :

```
not(self.panneauSuperieur.lockedState)
```

Post-conditions :

```
if self.panneauSuperieur^ouvrirPanneau() then
        self.panneauSuperieur.closedState = false
        self.systemeDrone^sendConfirmation(ConfirmMsg::PanneauOuvert)
```

2.6 Opération : $Receptacle :: fermeturePanneau()$

Acteur du MdE : systemeDrone
Cas d'utilisation : Livrer une commande
Messages : * panneauSuperieur : :fermerPanneau * systemeDrone : :sendConfirmation Pré-conditions :

```
not(self.panneauSuperieur.lockedState)
```

Post-conditions :

```
if self.panneauSuperieur^fermerPanneau() then
        self.panneauSuperieur.closedState = true
        self.systemeDrone^sendConfirmation(ConfirmMsg::PanneauFerme)
```

2.7 Opération : $Receptacle :: timeoutVerification()$

Acteur du MdE : systemeDrone
Cas d'utilisation : Livrer une commande
Messages : * PanneauAffichagemsgHorsService Pré-conditions :

Post-conditions :

```
self.panneauAffichage^msgHorsService(true)
```

2.8 Opération : $Receptacle :: dronePresent()$

Acteur du MdE : supportDrone
Cas d'utilisation : Livrer une commande
Messages : * supportDrone : :bloquerDrone Pré-conditions :

```
not(self.supportDrone.lockedState)
```

Post-conditions :

```
self.supportDrone^bloquerDrone(true)
self.supportDrone.lockedState = true
```

2.9 Opération : $Receptacle :: porteFerme()$

Acteur du MdE : PorteLaterale
Cas d'utilisation : Livrer une commande
Messages : * porteLaterale : :bloquerPorte Pré-conditions :

Post-conditions :

```
self.porteLaterale^bloquerporte()
self.porteLaterale.lockedState = true
```

2.10 Opération : $Receptacle :: panneauFerme()$

Acteur du MdE : PanneauSuperieur
Cas d'utilisation : Livrer une commande
Messages : * panneauSuperieur : :bloquerPanneau Pré-conditions :

Post-conditions :

```
self.panneauSuperieur^bloquerPanneau()
self.panneauSuperieur.lockedState = true
```

2.11 Opération : $Receptacle :: colisPresent()$

Acteur du MdE : capteurRFID
Cas d'utilisation : Livrer une commande
Messages : * systemeClient : :signalerReception Pré-conditions :

Post-conditions :

```
Self.systemeClient^signalerReception()
```

3 Système Ronde

3.1 Opération : $SystemeRonde :: cdeVisualiserDrone()$

Acteur du MdE : Superviser Ronde
Cas d'utilisation :
Messages : ecranSupervision : :visualiserDrones Pré-conditions :

Post-conditions :

```
self.ecranSupervision^visualiserDrones()
```

3.2 Opération : $SystemeRonde :: cdeVisualiserTournes()$

Acteur du MdE : Superviser Ronde
Cas d'utilisation :
Messages : ecranSupervision : :visualiserTournees Pré-conditions :

Post-conditions :

```
self.ecranSupervision^visualiserTournees()
```

3.3 Opération : $SystemeRonde :: envoyerCommande(cde : Commande)$

Acteur du MdE : Livrer une tournée
Cas d'utilisation :
Messages : BD : :ajouterCommande,calculerTournées Pré-conditions :

Post-conditions :

```
self.bd^ajouterCommande(cde)
self.bd^calculerTournees()
```

3.4 Opération : $SystemeRonde :: annulerCommade(lst : ListeCommande)$

Acteur du MdE : Livrer une tournée
Cas d'utilisation :
Messages : BD : :supprimerCommande,calculerTournees, Pré-conditions :

Post-conditions :

```
self.bd^supprimerCommande(lst)
self.bd^calculerTournees()
```

3.5 Opération : $SystemeRonde :: notifierExpiration(t : Tournee)$

Acteur du MdE : Livrer une tournée
Cas d'utilisation :
Messages : drone : :envoyerDrone BD : :supprimerCommande boiteMail : :envoyerMessage Pré-conditions :

Post-conditions :

```
self.bd^supprimerCommande(sender.tournee.lstCommande)
sender.tournee.drone^envoyerDrone()
self.boiteMail^envoyerMessage(sender.tournee.drone.id,DEPART_DRONE)
sender.tournee.drone.oclInState(Livraison)
```

3.6 Opération : $SystemeRonde :: signalerPosition(p : Position)$

Acteur du MdE : Livrer une tournée
Cas d'utilisation :
Messages : ecranSupervision : :visualiserDrone BD : :majPositionDrone Pré-conditions :

Post-conditions :

```
self.bd^majPositionDrone(sender.drone.id, p)
self.ecranSupervision^visualiserDrone()
```

3.7 Opération : $SystemeRonde :: signalerLivraison(cde : Commande)$

Acteur du MdE : Livrer une tournée
Cas d'utilisation :
Messages : BD : :supprimerCommande Pré-conditions :

Post-conditions :

```
self.bd^supprimerCommande(cde)
```

3.8 Opération : $SystemeRonde :: signalerAnormalie(a : Anormalie)$

Acteur du MdE : Livrer une tournée
Cas d'utilisation :
Messages : BD : :majEtatDrone boiteMail : :envoyerMessage Pré-conditions :

Post-conditions :

```
sender.drone.oclInState(Erreur)
self.bd.majEtatDrone(sender.drone.id, a)
self.boiteMail^envoyerMessage(sender.drone.id, a)
```

3.9 Opération : $SystemeRonde :: signalerRetour()$

Acteur du MdE : Livrer une tournée
Cas d'utilisation :
Messages : BD : :majEtatDrone boiteMail : :envoyerMessage Pré-conditions :

Post-conditions :

```
self.bd.majEtatDrone(sender.drone.id,PRET)
self.boiteMail^envoyerMessage(sender.drone.id,RETOUR)
sender.drone.oclInState(Inactif)
```

3.10 Opération : $SystemeRonde :: estOperationnel()$

Acteur du MdE : Livrer une tournée
Cas d'utilisation :
Messages : BD : :majEtatDrone DronePhysique : :envoyerDrone Pré-conditions :

Post-conditions :

```
self.bd.majEtatDrone(idDrone,BON_ETATt)
if sender.drone.Tournee.lstCommandes is null
        sender.drone.oclInState(Inactif)
        self.bd.majEtatDrone(idDrone,Pret)
else
        sender.drone.oclInState(Livraison)
        self.sender^envoyerDrone()
```

4 Drone

4.1 Opération : $Drone :: verificationColisActuel(idColis : IdRFID)$

Acteur du MdE : listeColis
Cas d'utilisation : Livrer une commande
Messages : * systeReceptacle : :verificationColis * ListeTimers : :enclanchementTimer Pré-conditions :

```
self.receptacleConnecte <> 0
```

Post-conditions :

```
self.systeReceptacle^verificationColis(self.listeColis[idColis].idDestintaire)
timeInfo.command := EnumCommand::verificationColis
self.ListeTimers^enclanchementTimer(timerInfo)
```

4.2 Opération : $Drone :: receptaclePlein()$

Acteur du MdE : SystemeReceptacle
Cas d'utilisation : Livrer une commande
Messages : * systèmeROnde : :SignalementErreur Pré-conditions :

```
self.receptacleConnecte <> 0
```

Post-conditions :

```
        self.systemeRonde^SignalementErreur("ReceptaclePlein")
self.etat := EnCirculation
```

4.3 Opération : $Drone :: colisCorrect(e : bool)$

Acteur du MdE : SystemeReceptacle
Cas d'utilisation : Livrer une commande
Messages : * systemeReceptacle : :ouverturePanneau Pré-conditions :

```
self.receptacleConnecte <> 0
```

Post-conditions :

```
self.systemeReceptacle^ouverturePanneau()
```

4.4 Opération : $Drone :: signalReceptacle(idReceptacle : IdRFID)$

Acteur du MdE : SystemeReceptacle
Cas d'utilisation : Livrer une commande
Messages : - Pré-conditions :

```
self.receptacleConnecte == 0
```

Post-conditions :

```
self.receptacleConnecte == idReceptacle
self.etat := LivraisonEnCours
```

4.5 Opération : $Drone :: sendConfirmation(msg : ConfirmMsg)$

Acteur du MdE : SystemeReceptacle
Cas d'utilisation : Livrer une commande
Messages : * ListeTimers : :desactiverTimer Pré-conditions :

```
self.receptacleConnecte <> 0
```

Post-conditions :

```
self.ListeTimers^desactiverTimer(msg)
```

4.6 Opération : $Drone :: expirationTimer(idTimer : IdTimer)$

Acteur du MdE : ListeTimer
Cas d'utilisation : Livrer une commande
Messages : * trappe : :activerSystUrgence * systemeRonde : :SygnalementErreur Pré-conditions :

Post-conditions :

```
if ListeTimers[idTimer].command = FermetureTrappe then
        self.trappe^activerSystUgence()
fi
self.systemeRonde^ SignalementErreur( ListeTimers[idTimer].command )
```

4.7 Opération : $Drone :: ColisPret()$

Acteur du MdE : tapisRoulant
Cas d'utilisation : Livrer une commande
Messages : * ListeColis : :getCurColis Pré-conditions :

Post-conditions :

```
self.ListeColis^getCurColis()
```

4.8 Opération : $Drone :: EnvoyerDrone(itineraire : ItineraireLivraison)$

Acteur du MdE : SystemeRonde
Cas d'utilisation : Livrer une commande
Messages : - Pré-conditions :

Post-conditions :

```
self.etat := EnCirculation
```

5 Système Web

5.1 Opération : $SystemeWeb :: voirCommande()$

Acteur du MdE : BtnVoirCommande
Cas d'utilisation : Valider une commande
Messages : * PanneauInformation : :afficherCommande, afficherFacture, produitNonDispo Pré-conditions :

Post-conditions :

```
if not(self.commande.estCommandeVide) then
        self.panneauInformation^afficherCommande(self.commande)
        self.panneauInformation^afficherFacture(self.commande, self.commande.pri
        self.commande.estCommandeValidee = false
        if not(self.commande.article.estDisponible) then
                self.panneauInformation^produitNonDispo( self.commande.article ,
        else
                self.panneauInformation^produitNonDispo( self.commande.article ,
```

5.2 Opération : $SystemeWeb :: alarm(art : Article, qte : Quantite)$

Acteur du MdE : SystèmeBalance
Cas d'utilisation : MAJ d'une commande
Messages : * PanneauInformation : :produitNonDispo * ListeCommande : :majCommande Pré-conditions :

Post-conditions :

```
        if art.estDisponible then
                if self.commande.estCommandeVide then
                        self.commande.estCommandeVide = false
self.commande.article = art
self.commande.prixTotal = art.prixArticle * qte
                else
                        if art == self.commande.article then
                                self.commande.article.estDisponible = true
                                self.panneauInformation^produitNonDispo(
self.commande.article, false)
                                self.commande.article.qteArticle =
self.commande.article.qteArticle@pre + qte
self.commande.prixTotal = self.commande.prixTotal@pre +
art.prixArticle * qte
                        self.commande.estCommandeVide = false
                self.commande^majCommande()
        else
                if self.commande.estCommandeVide then
                        self.panneauInformation^produitNonDispo(art, true)
                else
if art == self.article then
        self.commande.article.estDisponible = false
                                self.panneauInformation^produitNonDispo(
self.commande.article, true)
                self.commande^majCommande()
```

5.3 Opération : $SystemeWeb :: produitNonStock(art : Article)$

Acteur du MdE : InterfaceCatalogue
Cas d'utilisation : MAJ d'une commande
Messages : * PanneauInformation : :produitNonDispo Pré-conditions :

Post-conditions :

```
        if self.commande.article == art then
                self.commande.article.estDisponible = false
                self.panneauInformation^produitNonDispo(self.commande.article, true)
```

5.4 Opération : $SystemeWeb :: validerCommande()$

Acteur du MdE : BtnValiderCommande
Cas d'utilisation : Valider une commande
Messages : * PanneauInformation : :afficherCommande, afficherFacture, commandeValidée * ListeCommande : :majCommande Pré-conditions :

Post-conditions :

```
if not(self.commande.estCommandeVide) then
        self.panneauInformation^afficherCommande(self.commande)
self.panneauInformation^afficherFacture(self.commande,
self.commande.prixTotal)
                self.commande.estCommandeValidee = true
                self.panneauInformation^commandeValidee(true)
                self.commande^majCommande()
        else
                self.panneauInformation^commandeValidee(false)
```

5.5 Opération : $SystemeWeb :: changerQuantit(art : Article, qte : Quantit)$

Acteur du MdE : ListeCommande
Cas d'utilisation : Valider une commande
Messages : * PanneauInformation : :afficherCommande, commandeValidée * ListeCommande : :majCommande
Pré-conditions :

Post-conditions :

```
if not(self.commande.estCommandeVide) then
                self.panneauInformation^afficherCommande(self.commande)
                self.panneauInformation^commandeValidee(false)
                if art==self.commande.article then
                        self.commande.prixTotal = self.commande.prixTotal@pre + (qte -
self.commande.article.qteArticle) * art.prixArticle
                        self.commande.article.qteArticle =qte
                self.commande^majCommande()
```

5.6 Opération : $SystemeWeb :: estArticleCoch(art : Article)$

Acteur du MdE : NavigateurWeb
Cas d'utilisation : Valider une commande
Messages : * PanneauInformation : :afficherCommande, commandeValidée * ListeCommande : :majCommande
Pré-conditions :

Post-conditions :

```
if not(self.commande.estCommandeVide) then
                self.panneauInformation^afficherCommande(self.commande)
                self.panneauInformation^commandeValidee(false)
                if art==self.commande.article then
                        if self.commande.article.estCoche then
                                self.commande.article.estCoche = false
                                self.commande.prixTotal = self.commande.prixTotal@pre -
self.commande.article.qteArticle * art.prixArticle
                        else
                                self.commande.article.estCoche = true
                                self.commande.prixTotal = self.commande.prixTotal@pre +
self.commande.article.qteArticle * art.prixArticle
                        self.commande^majCommande()
```

5.7 Opération : $SystemeWeb :: configBal(bal : Balance, seuil : Seuil, qte : Quantite)$

Acteur du MdE : BtnConfigBalance
Cas d'utilisation : Configurer les balances
Messages : * PanneauInformation : :confirmConfigBal, erreurConfigBal Pré-conditions :

Post-conditions :

```
        if bal==self.balance then
                if seuil <= 0 then
                        if qte <=0 then
                                self.panneau.information^erreurConfigBal()
        else
                self.panneauInformation^confirmConfigBal()
```

5.8 Opération : $SystemeWeb :: assignBal(bal : Balance, art : Article, seuil : Seuil, qte : Quantit)$

Acteur du MdE : BtnAssignBalance
Cas d'utilisation : Configurer les balances
Messages : * PanneauInformation : :afficherBalance, confirmAssignBal, erreurAssignBal Pré-conditions :

Post-conditions :

```
self.panneauInformation^afficherBalance(bal)
if bal==Null then
        if art == self.commande.article then
                if seuil <= 0 then
                        if qte <=0 then
                                self.panneau.information^erreurAssignBal()
else
        self.panneauInformation^confirmAssignBal()
```