ETUDE DE CAS en UML : GESTION DES COMMANDES DE PIECES FABRIQUEES PAR LA SOCIETE C

La société C fabrique des pièces métalliques réalisées dans son atelier. La société C est en relation commerciale uniquement avec des clients réguliers. Les factures sont mensualisées : toutes les commandes d'un client dans le mois sont facturées à la fin du mois.

A la réception d'une commande, le secrétariat de la société C édite un bon de fabrication qu'elle transmet à l'atelier (un double de ce bon de fabrication est envoyé au client). Une fois la pièce fabriquée, l'atelier complète le bon de fabrication et le transmet au secrétariat qui enregistre la date de fin de fabrication et informe le client que sa pièce est disponible à l'atelier.

A la fin du mois, les factures sont éditées. Une facture est éditée pour chaque client ayant effectué une commande dans le mois.

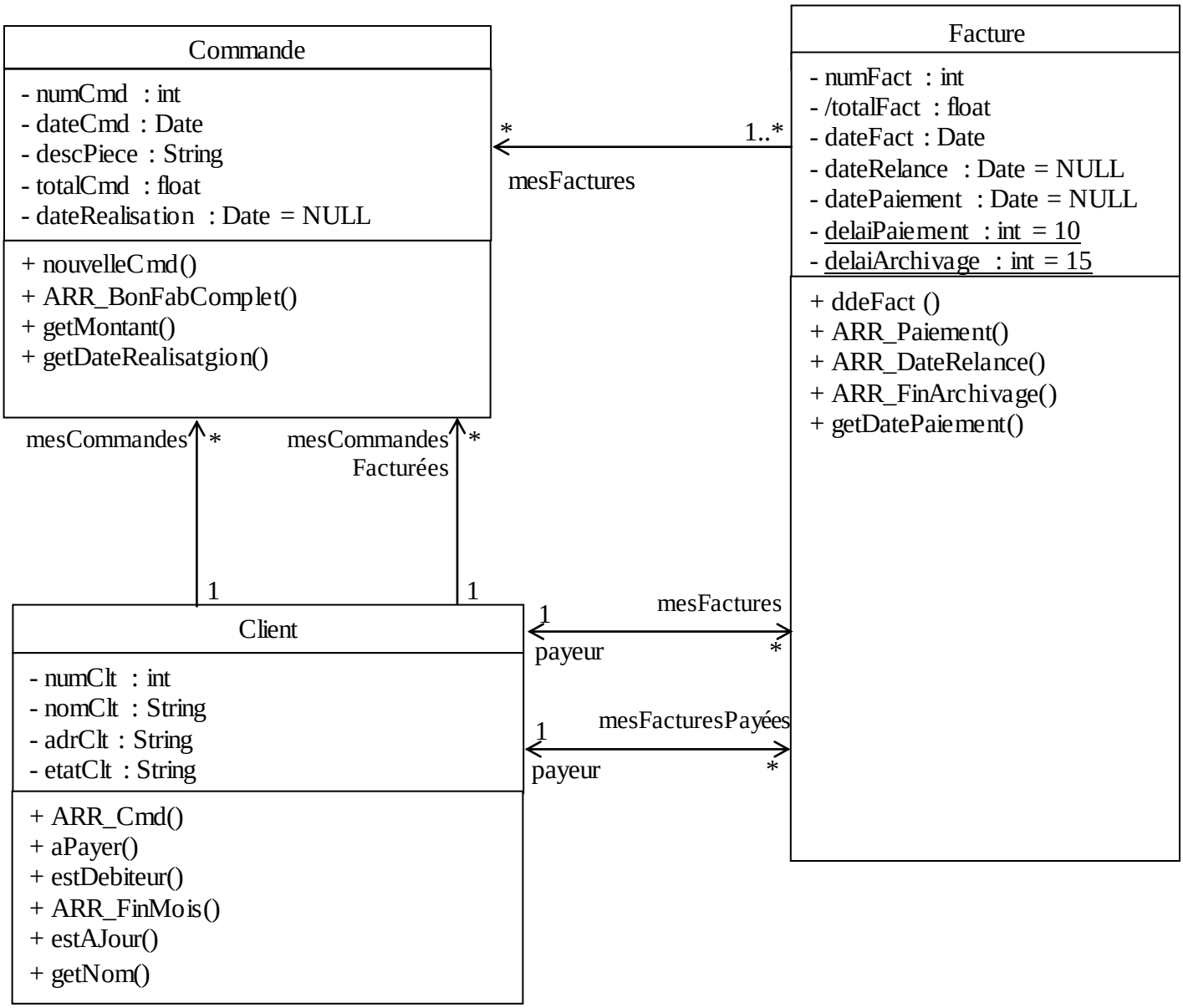
Si une facture n'est pas payée dans le délai de 10 jours ouvrables alors un courriel de relance est envoyé au client. Tant qu'il n'aura pas payé sa facture aucune de ses commandes ne sera acceptée : chaque fois qu'il effectuera une commande, un courriel de refus lui sera envoyé.

Une facture payée est archivée 15 ans et ensuite détruite.

Hypothèse : nous considérons que tous les paiements sont corrects : un client paye le total facturé ou il n'envoie pas de règlement.

Hors Contexte : l'enregistrement de nouveau client est hors contexte.

Le diagramme de classes



La signature des méthodes (en Java) :

- ARR_Cmd (descPiece : String, prixCmd : float) : void
- aPayer () : void
- estDebiteur () : void
- estAJour () : void
- ARR_FinMois () : void
- getNom() : String
- nouvelleCmd (clt :Client, descPiece : String, prixCmd : float): void
- ARR_BonFabComplet () : void
- getMontant () : float
- ddeFact (clt : Client, listCmd : ArrayList<Commande>) : void
- ARR_Paiement (sommePayee : float) : void
- ARR_DateRelance () : void
- ARR_FinArchivage () : void
- getDatePaiement() : Date

Le diagramme de contexte statique

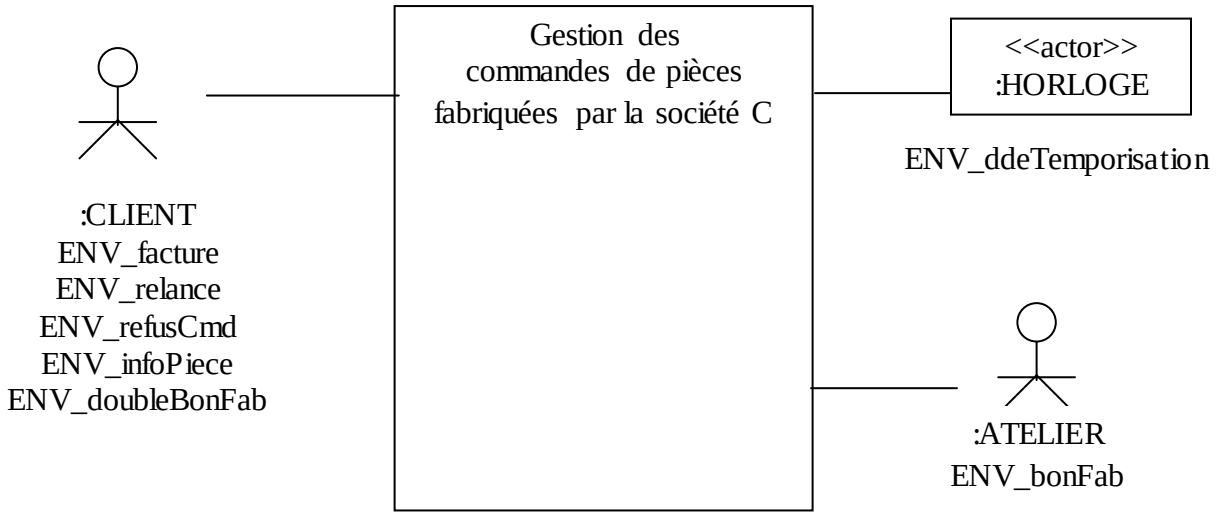
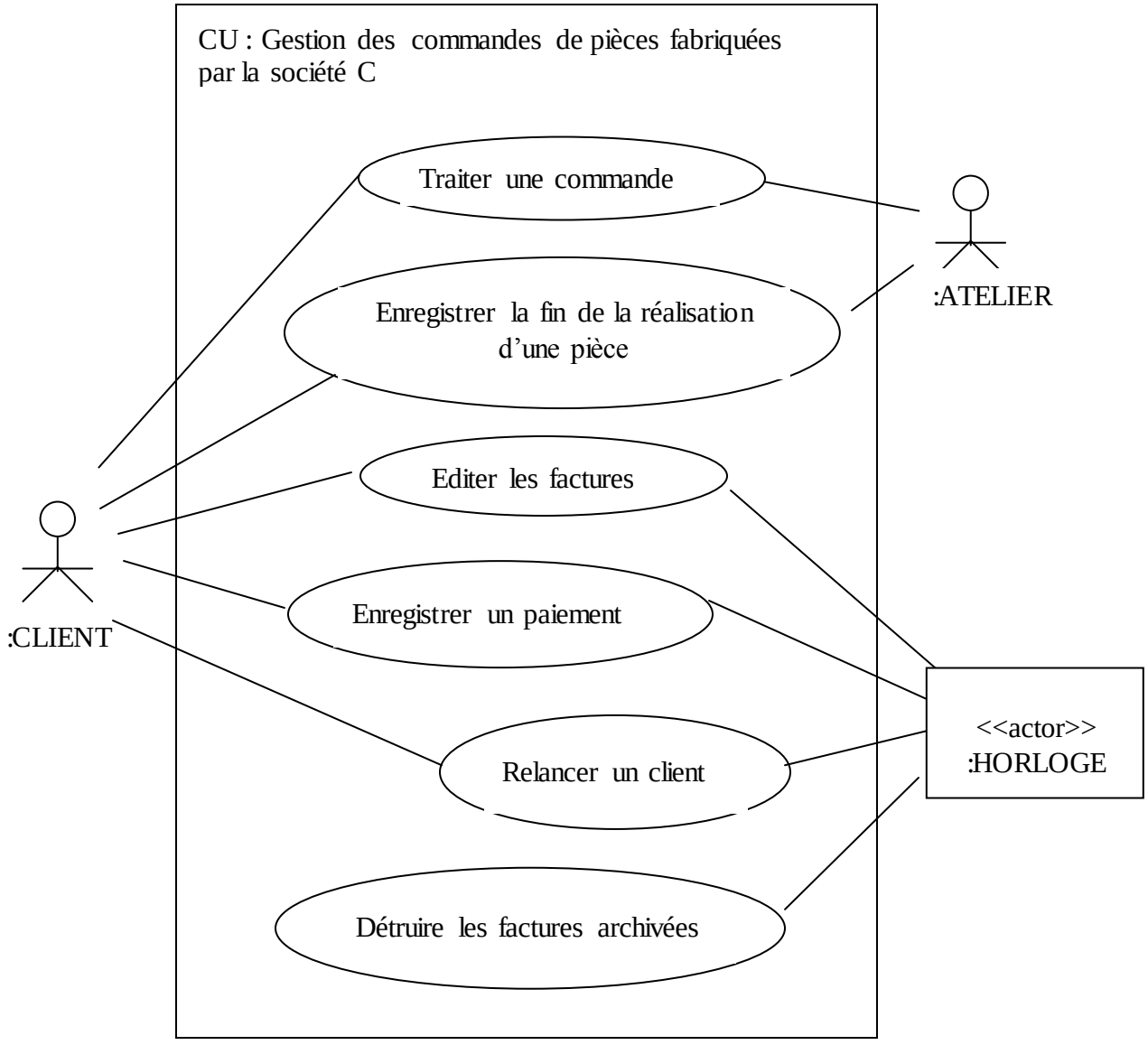
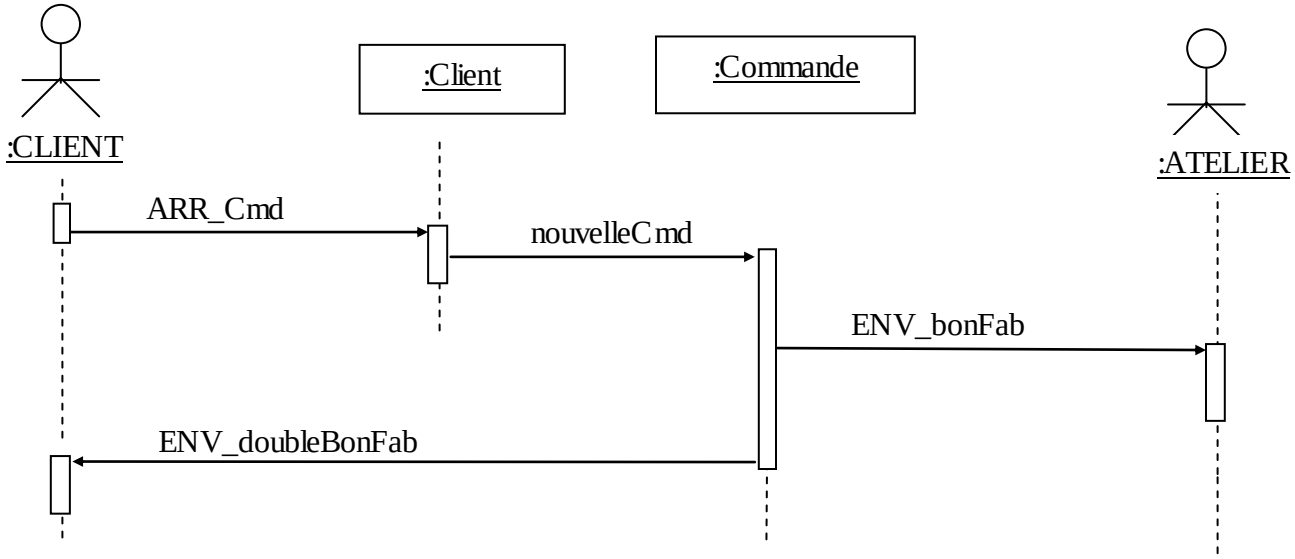


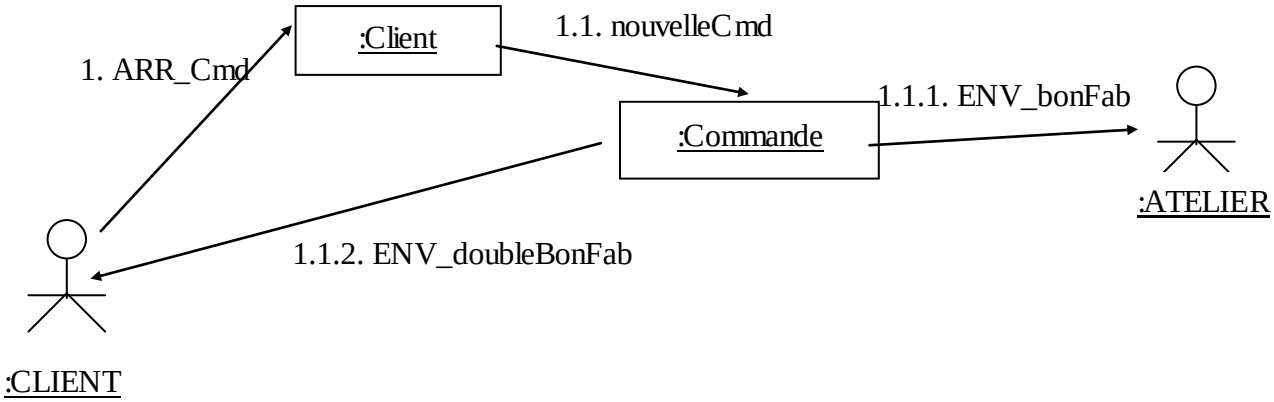
Diagramme des cas d'utilisation :



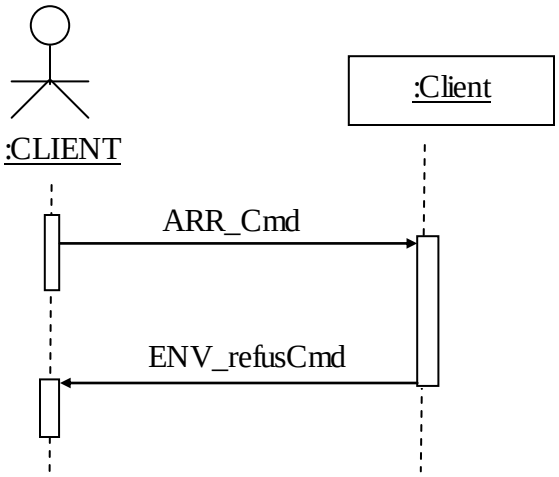
Le diagramme de séquence « Enregistrer une commande acceptée» :



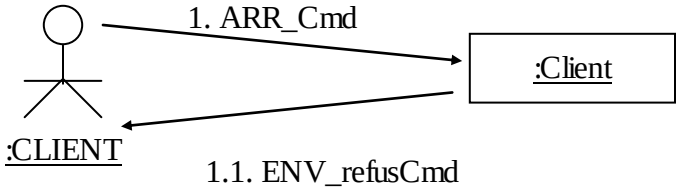
Le diagramme de communication « Enregistrer une commande acceptée» :



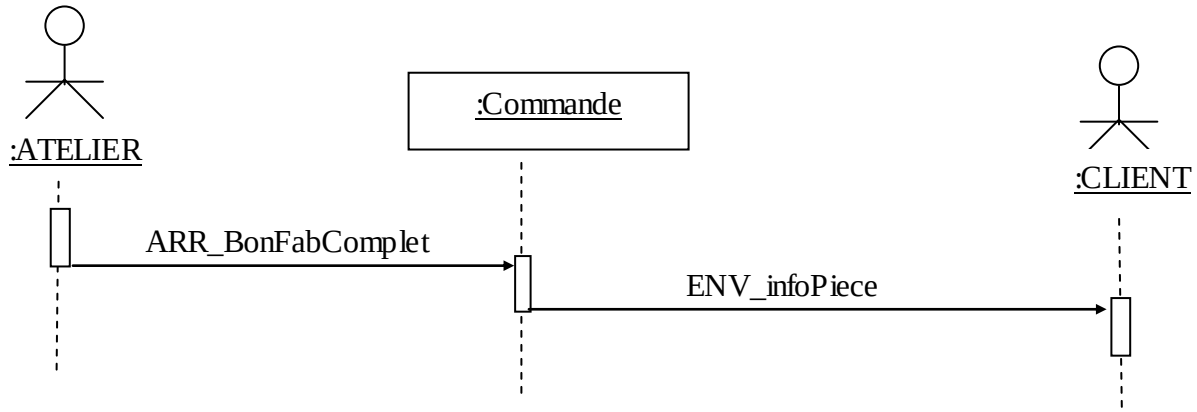
Le diagramme de séquence « Refuser une commande» :



Le diagramme de communication « Refuser une commande» :

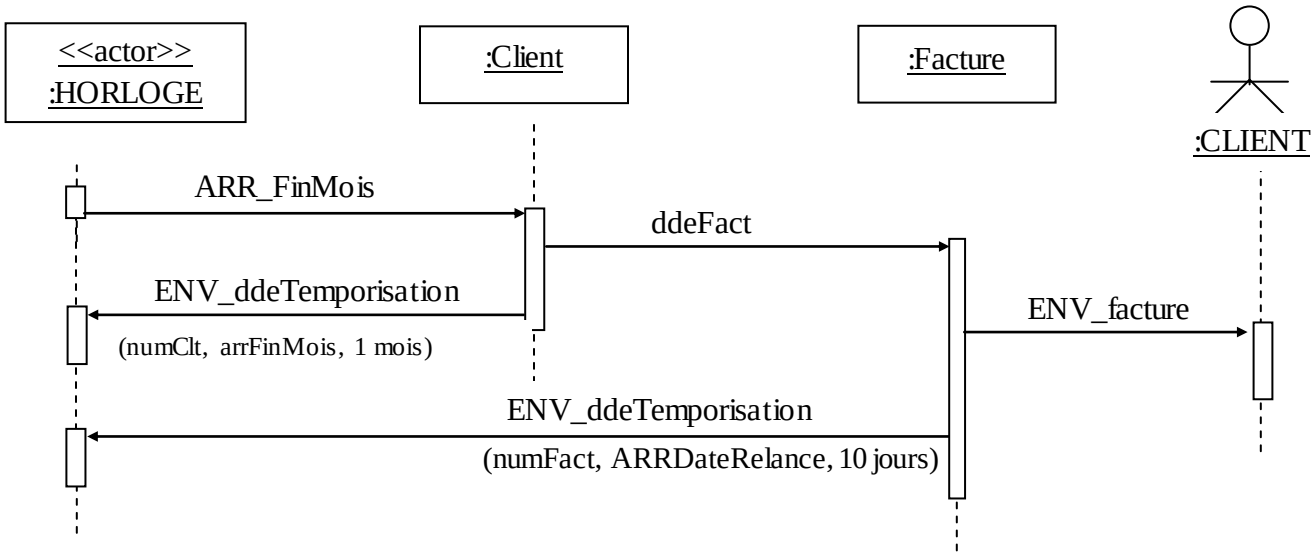


Le diagramme de séquence «Enregistrer la fin de la réalisation d'une pièce» :

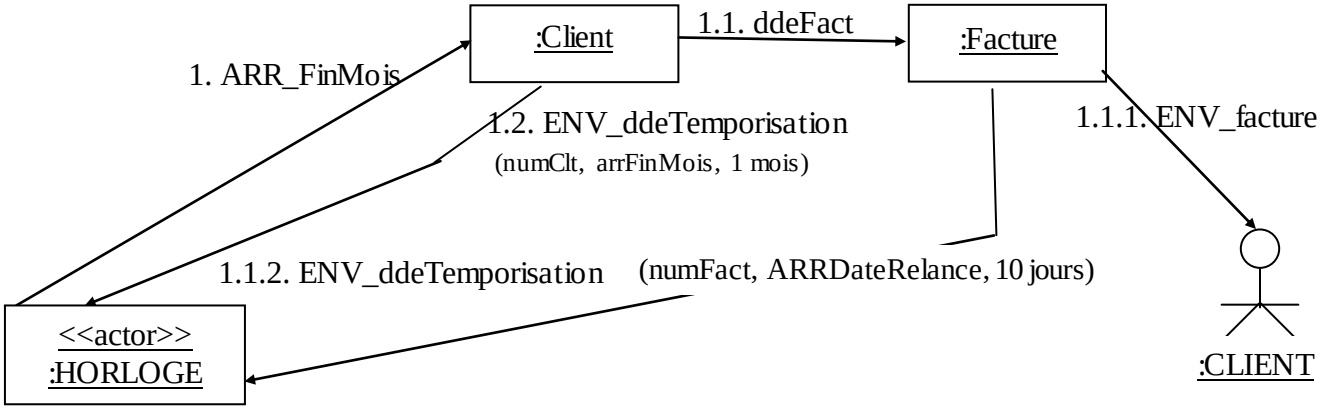


Le diagramme de communication «Enregistrer la fin de la réalisation d'une pièce» :

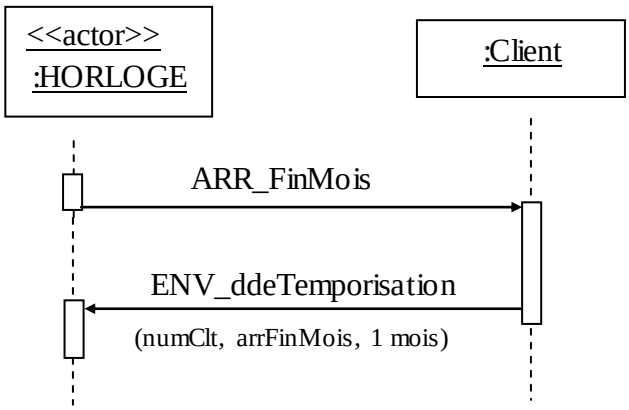
Le diagramme de séquence «Edition d'une facture» :



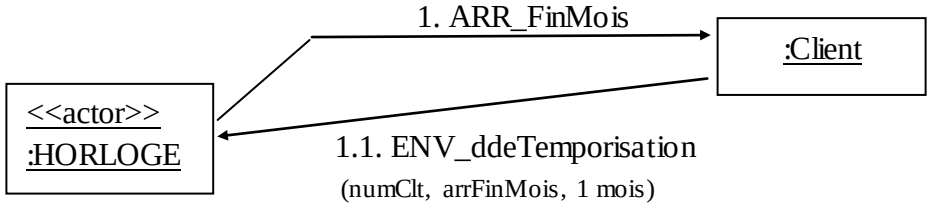
Le diagramme de communication «Edition d'une facture» :



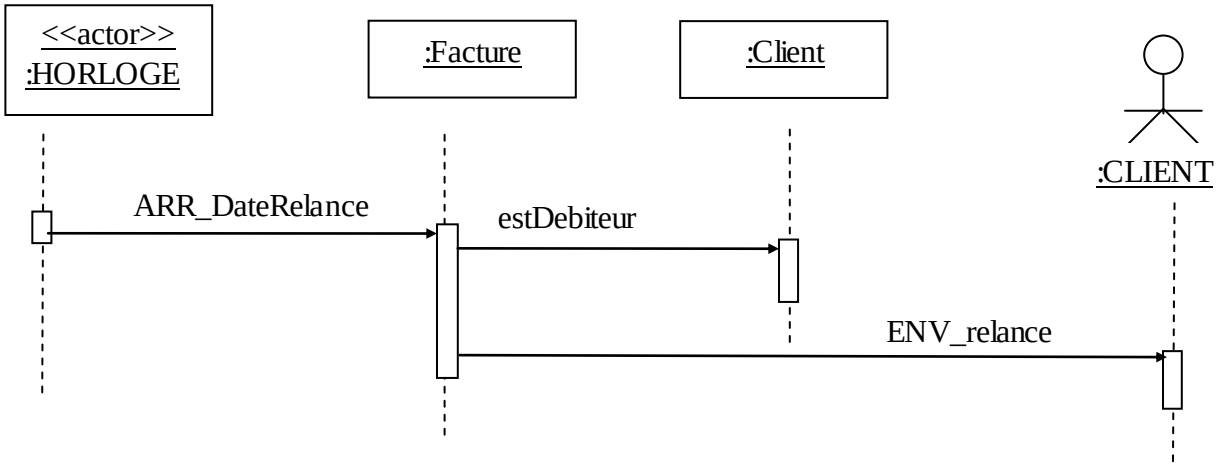
Le diagramme de séquence «Vérifier qu'un client n'a pas fait de commande durant le précédent mois» :



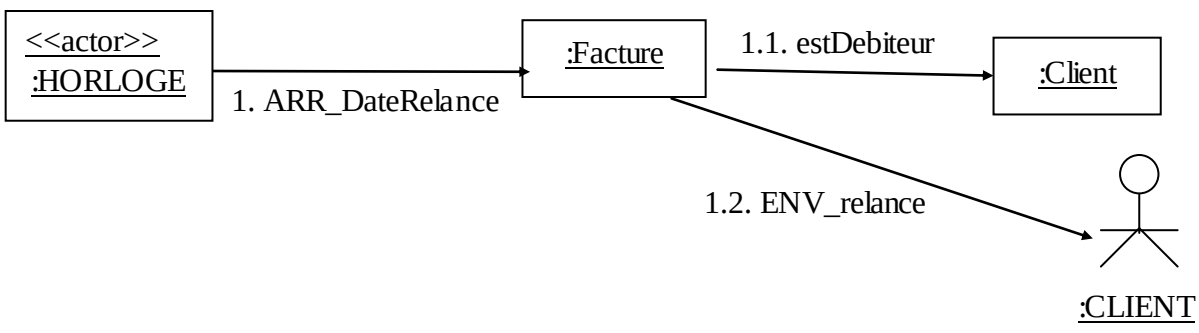
Le diagramme de communication «Vérifier qu'un client n'a pas fait de commande durant le précédent mois» :



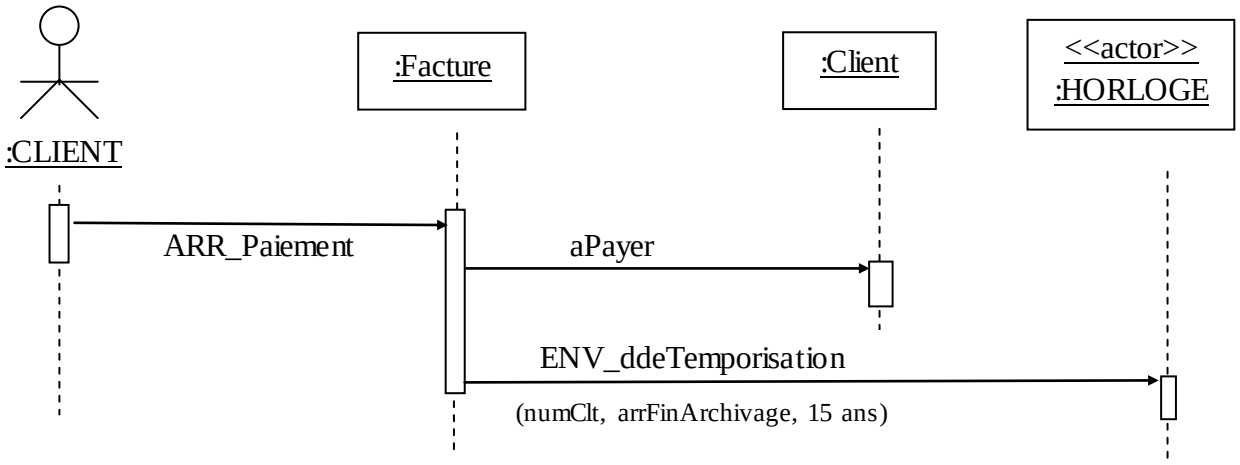
Le diagramme de séquence «Relancer un client qui n'a pas payé une facture» :



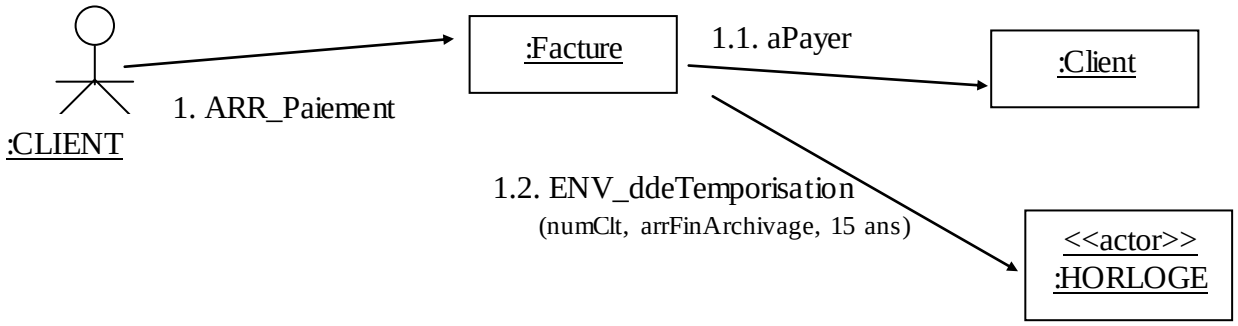
Le diagramme de communication «Relancer un client qui n'a pas payé une facture» :



Le diagramme de séquence «Enregistrer un paiement d'un client non débiteur» :

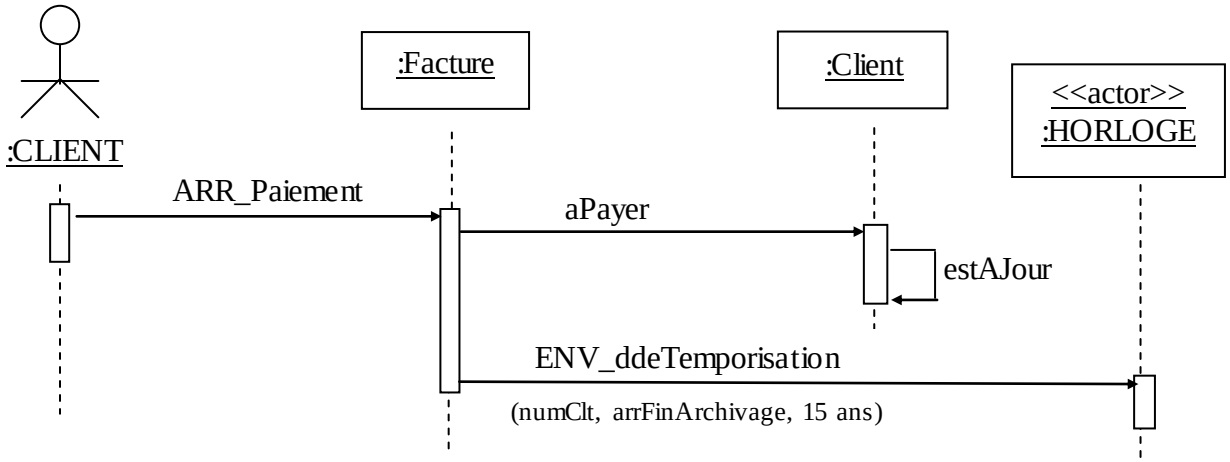


Le diagramme de communication «Enregistrer un paiement d'un client non débiteur» :

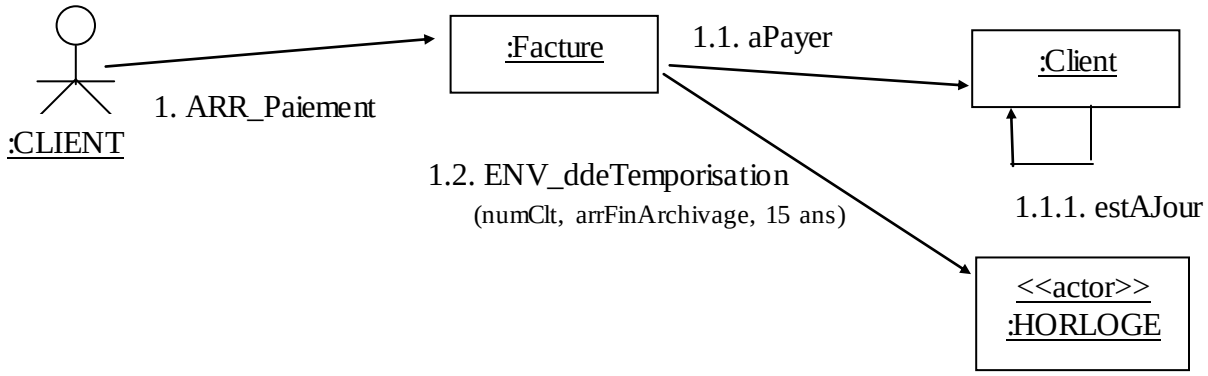


Le diagramme de séquence «Enregistrer du paiement d'un client débiteur qui reste débiteur malgré ce paiement» est identique au diagramme de séquence «Enregistrer un paiement d'un client non débiteur» .
De la même manière, le diagramme de communication «Enregistrer du paiement d'un client débiteur qui reste débiteur malgré ce paiement» est identique au diagramme de communication «Enregistrer un paiement d'un client non débiteur» .

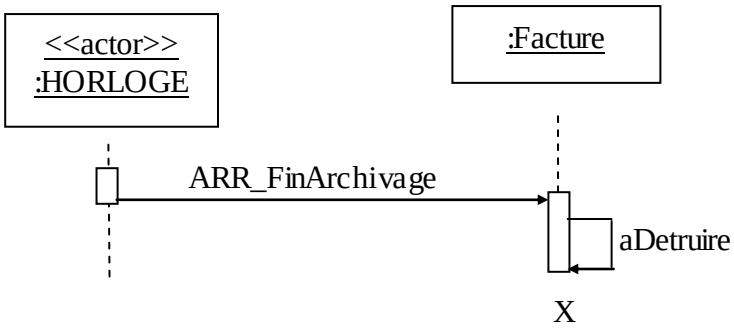
Le diagramme de séquence «Enregistrer du paiement d'un client débiteur qui suite à ce paiement est à jour» :



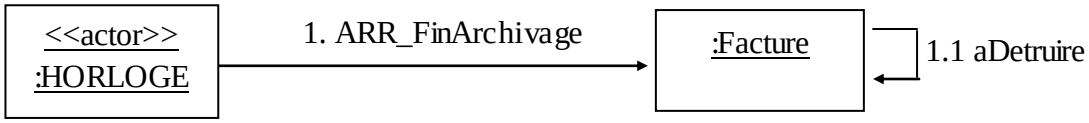
Le diagramme de communication «Enregistrer du paiement d'un client débiteur qui suite à ce paiement est à jour» :



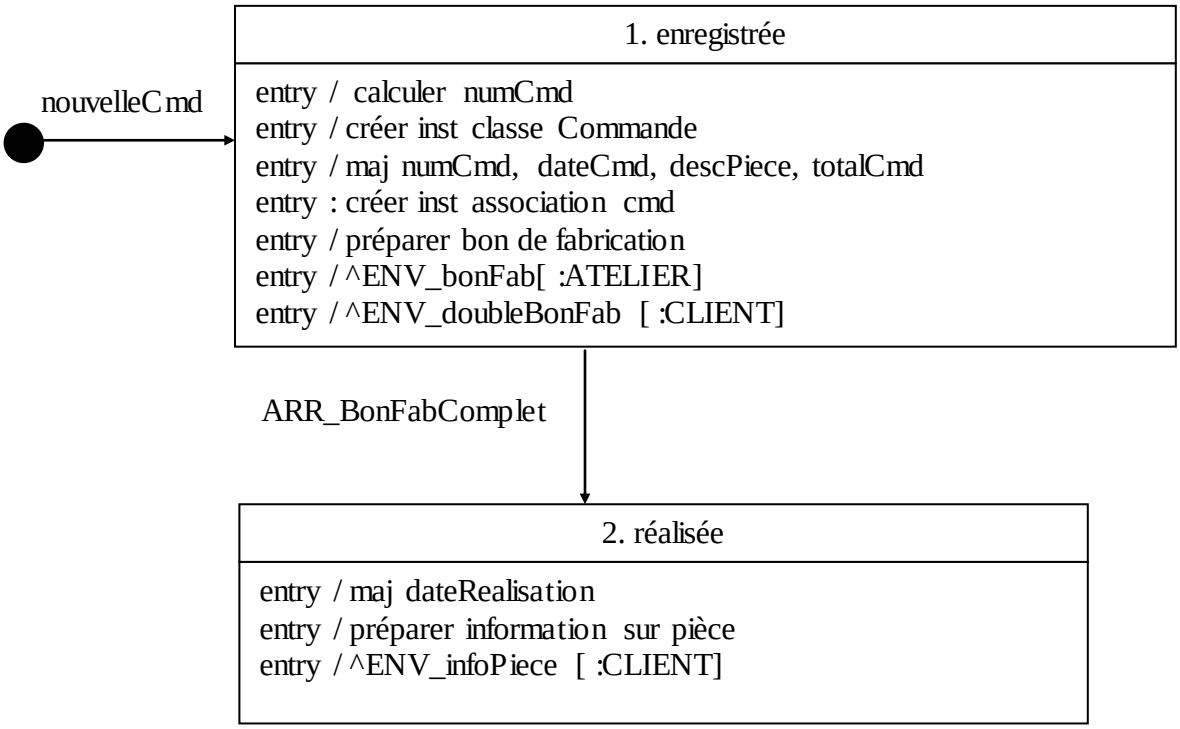
Le diagramme de séquence «Détruire une facture» :



Le diagramme de communication «Détruire une facture» :



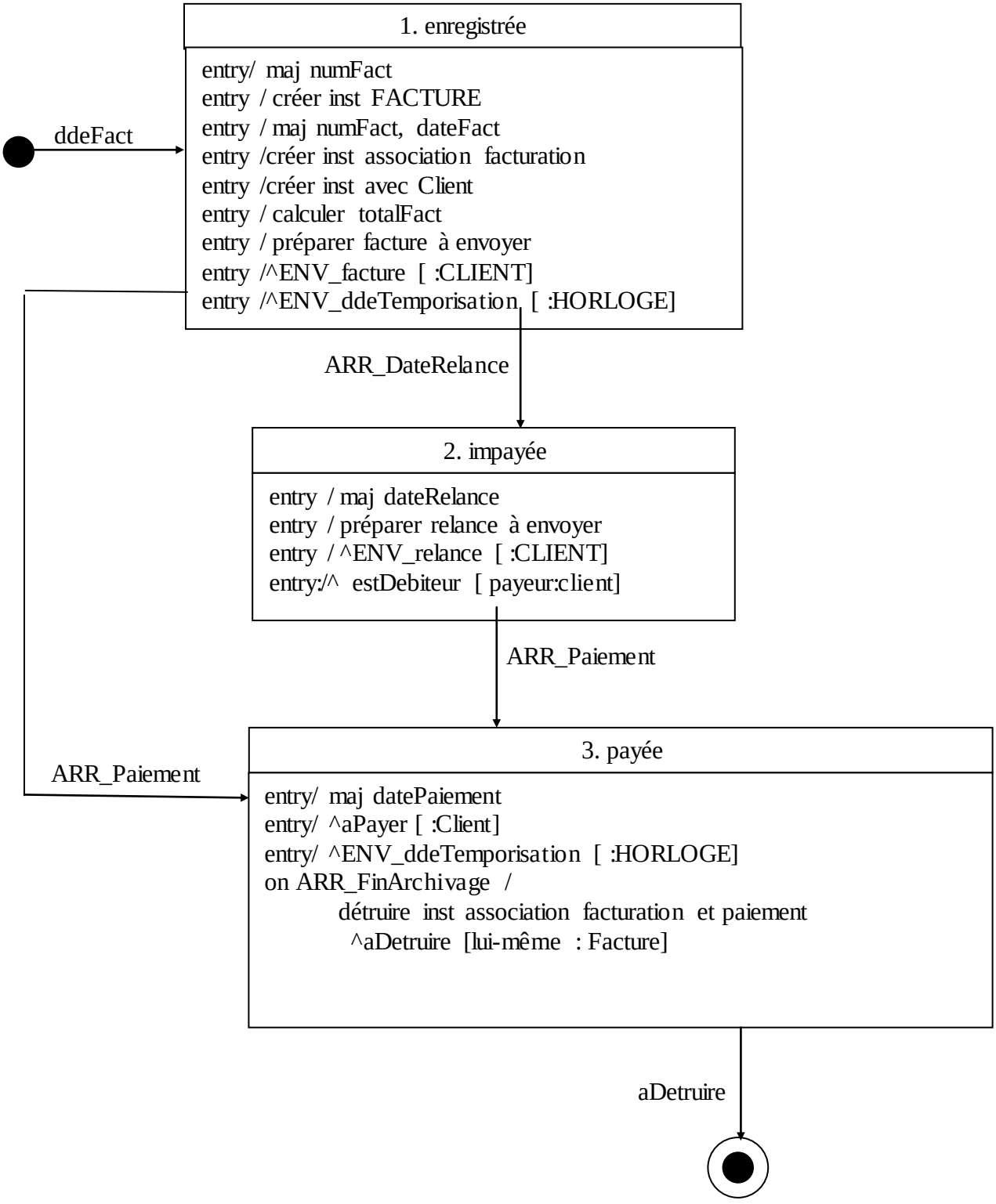
Le diagramme Etat/Transition de la classe « Commande » :



Caractérisation des états de la Classe Commande

Commande enregistrée	Commande réalisée
dateRéalisation = NULL	dateRéalisation ≠ NULL

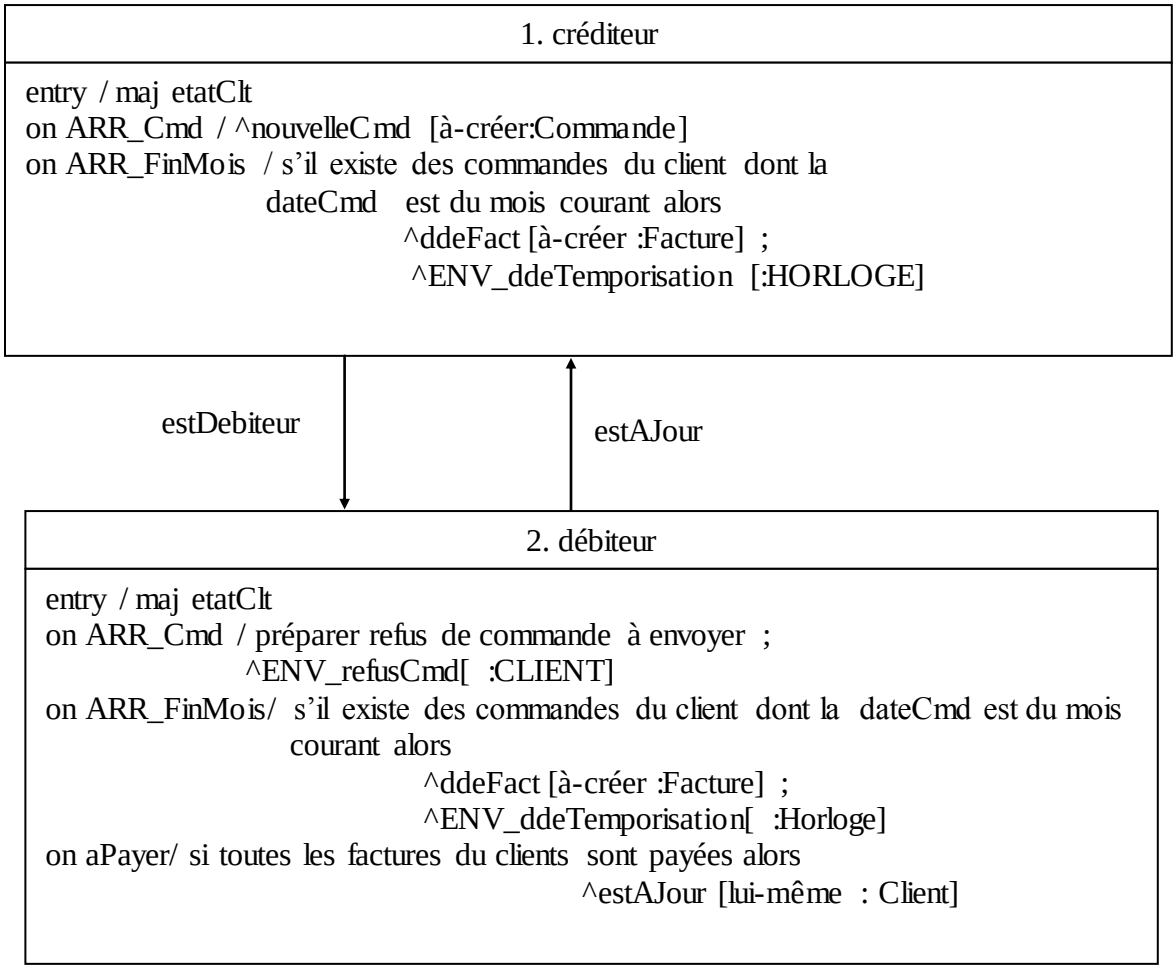
Le diagramme Etat/Transition de la classe Facture :



Caractérisation des états de la Classe Facture

Facture enregistrée	Facture payée	Facture impayée
datePaiement = NULL dateRelance = NULL	datePaiement ≠ NULL	datePaiement = NULL dateRelance ≠ NULL

Le diagramme Etat/Transition de la classe Client :



Caractérisation des états de la Classe Client

Client créditeur	Client débiteur
etatClt = "créditeur"	etatClt = "débiteur"

```
public class Commande {
	private int numCmd;
	private static int nombreCmd=0;
	private String descriptionPiece;
	private float totalCmd;
	private Calendar dateCmd;
	private Calendar dateRealisation=null;

	public Commande(String description, float cout) {
		numCmd= nombreCmd++;
		descriptionPiece = description; totalCmd = cout;
		dateCmd = Calendar.getInstance();};

	public float getMontant() {return totalCmd;}
	public Calendar getDateRealisation() {return dateRealisation;}
	public String toString() { return(" "+descriptionPiece+" au cout : "+totalCmd);}
	public void arrBonFabricationComplet(){
		dateRealisation = Calendar.getInstance();
		System.out.println("ENV information sur realisation "+descriptionPiece);}
}

public class Facture {
  private int numFacture;
  private static int nombreFactures = 0;
  private float totalFacture;
  private Calendar dateFacture, datePaiement=null, dateRelance=null;
  private static int delaiRelance = 10;
  private static int delaiArchivage = 15;
  private Collection<Commande> mesCommandes;
  private Client monPayeur;

  public Calendar getDatePaiement() { return datePaiement;}
  public float  getMontant() { return totalFacture;}

  public void arrPaiement(float somme) {
	if (somme == totalFacture) {
		datePaiement = Calendar.getInstance(); monPayeur.aPayer() ;  }  }

  public void arrDateRelance() {
	if (datePaiement == null) {monPayeur.estDebiteur();
	System.out.print("ENV relance de la "); this.affichage();} }

  public void arrDateFinArchivage() { monPayeur.factureADetruire(this, mesCommandes);}

  public Facture(Client payeur, Collection<Commande> mesCmds) {
	numFacture = nombreFactures++; monPayeur = payeur;
	mesCommandes = mesCmds; dateFacture = Calendar.getInstance();
	totalFacture = 0;
	for (Commande cmd : mesCommandes)  totalFacture =+ cmd.getMontant();
	System.out.print("ENV facture ");  this.affichage(); }

private void affichage() {
	System.out.print("facture à "+monPayeur.getNom());
	System.out.println(" d'un montant  "+totalFacture);
	for (Commande cmd : mesCommandes) System.out.println("  "+cmd); }

}
```

```
public class Client {
	private int numClt;
	private static int nombreClients=0;
	private Collection<Commande> mesCommandes;
	private Collection<Facture> mesFactures;
	private Collection<Facture> mesFacturesPayees;
	private Collection<Commande> mesCommandesFacturees;

	private String nom;
	private String prenom;
	private String adresse;
	private int etatClient; // = 1 si il est débiteur

	public String getNom() {return nom;}

	public Client(String nom) {
		this.nom = nom;
		 mesCommandes = new ArrayList<Commande>();
		 mesFactures = new ArrayList<Facture>();
		 mesFacturesPayees = new ArrayList<Facture>();
		 mesCommandesFacturees = new ArrayList<Commande>() ; }

	public void aPayer(){
		ArrayList<Facture> factures = new ArrayList<Facture>();
		for (Facture facture : mesFactures)
			if (facture.getDatePaiement() != null)  {factures.add(facture);  }
		mesFactures.removeAll(factures); mesFacturesPayees.addAll(factures);
		if (mesFactures.isEmpty()) this.estAJour();  }

	public void estDebiteur() {etatClient = 1;}
	public void estAJour() {etatClient = 0; }

	public void factureADetruire(Facture facture, Collection<Commande> mesCmds) {
	  mesFacturesPayées.remove(facture); mesCommandesFacturees.removeAll(mesCmds);}

	public Commande arrCommande(String description, float cout){
		if (etatClient == 1){
			System.out.println("ENV refus Commande à "+nom); return  null;}
		Commande cmd = new Commande(description, cout);
		mesCommandes.add(cmd);
		System.out.println("ENV Bon fabrication de "+description);
		System.out.println(" payé par "+nom);
		return cmd;  }

	public Facture arrFinMois() {
		ArrayList<Commande> commandesAPayees = new ArrayList<Commande>();
		if (mesCommandes.isEmpty()) return null;
		for (Commande cmd : mesCommandes)
			if (cmd.getDateRealisation()!= null)   commandesAPayees.add(cmd);
		if (commandesAPayees.isEmpty()) return null;
		mesCommandes.removeAll(commandesAPayees);
		mesCommandesFacturees.addAll(commandesAPayees);
		Facture fact= new Facture(this,commandesAPayees);
		mesFactures.add(fact);  return fact;}
}
```

```
public class ClientTest {
	Client client = new Client("Duval");
	Facture facture, factureB;
	Commande cmd1, cmd2, cmd3;

	@Test
	public void testEtudiantNom() {
		assertEquals("nom client enregistre", client.getNom(),"Duval");}

	@Test
	public void fonction() {
	  // est A Jour
		client.estAJour();
		cmd2 = client.arrCommande("desc-piece2",(float)25.5);
		assertNotNull("accepte commande",cmd2);
	// Facturation seulement les commandes realisées
		facture = client.arrFinMois();
		assertNull("facturation",facture);
		cmd2.arrBonFabricationComplet();
	    factureB = client.arrFinMois();
		assertNotNull("facturation",factureB);
		assertEquals(factureB.getMontant(),cmd2.getMontant(),0);
	// conséquence d'une relance d'une facture
		factureB.arrDateRelance();
		Commande cmd0 = client.arrCommande("desc-piece0",(float)5.5);
		assertNull("refus commande",cmd0);
	// conséquence du paiement des factures
		factureB.arrPaiement((float)25.5);
		facture =client.arrFinMois();
		assertNull("facturation mensuelle non utile",facture);
		 cmd3 =  client.arrCommande("desc-piece3",(float)35.5);
		assertNotNull("accepte commande",cmd3);
		}
}
```

Déclaration des classes en C++ (Date.h, Client.h, Commande.h et Facture.h)

==========

```
#ifndef DATE_H
#define DATE_H
#include <string>
#include <vector>
using namespace std;

class Date {
 private :
    int  _jour, _mois, _annee;

 public :
    Date(int jour, int mois, int annee) ;
    int getJour();
    int getMois();
    int getAnnee();
};

#endif
```

```
#ifndef CLIENT_H
#define CLIENT_H
#include <Date.h>
class Facture ;
class Commande ;

class Client {
 private :
  int  _numClt;
  string _nomClt, _adrClt, _etatClt;
  vector<Facture> _mesFactures;
  vector<Commande> _mesCommandes;

 public :
  Client(string nom, string adresse);
  void arrCommande(string descPiece, float totalCmd) ;
  void aPayer();
  void estDebiteur();
  void estAJour();
  void arrFinMois();
};

#endif
```

```
#ifndef COMMANDE_H
#define COMMANDE_H
#include <Client.h>

class Commande {
 private :
  int  _numCmd;
  string _descPiece;
  float _totalCmd ;
  Date _dateRealisation;
  Date _dateCmd;

 public :
  Commande() ;
  Commande(Client clt, string descPiece, float totalCmd) ;
  void arrBonFabComplet(Date dateRealisation);
};

#endif
```

```
#ifndef FACTURE_H
#define FACTURE_H
#include <Commande.h>

class Facture {
 private :
  static const int _delaiPaiement = 10; // 10 jours
  static const int _delaiArchivage =  5475; // 15 ans = 5475 jours
  int  _numFact;
  float _totalfact;
  Date  _dateFact, _dateRelance, _datePaiement;
  Client _payeur;
  vector<Commande> _mesCommandes;

 public :
  Facture (Client clt, vector<Commande> mesCommandes);
  void arrPaiement(float somme);
  void arrDateRelance();
  void arrFinArchivage();
};

#endif
```

```
#include <Facture.h>
int main() {
  Client  cl("Johnen", "IUT Bordeaux 1");
  Commande c(cl, "pieceAColette", 13.3);
  vector<Commande> listeCmd(10);
  listeCmd[0] = c;
  Facture f(cl, listeCmd);
  return 0;  }
```