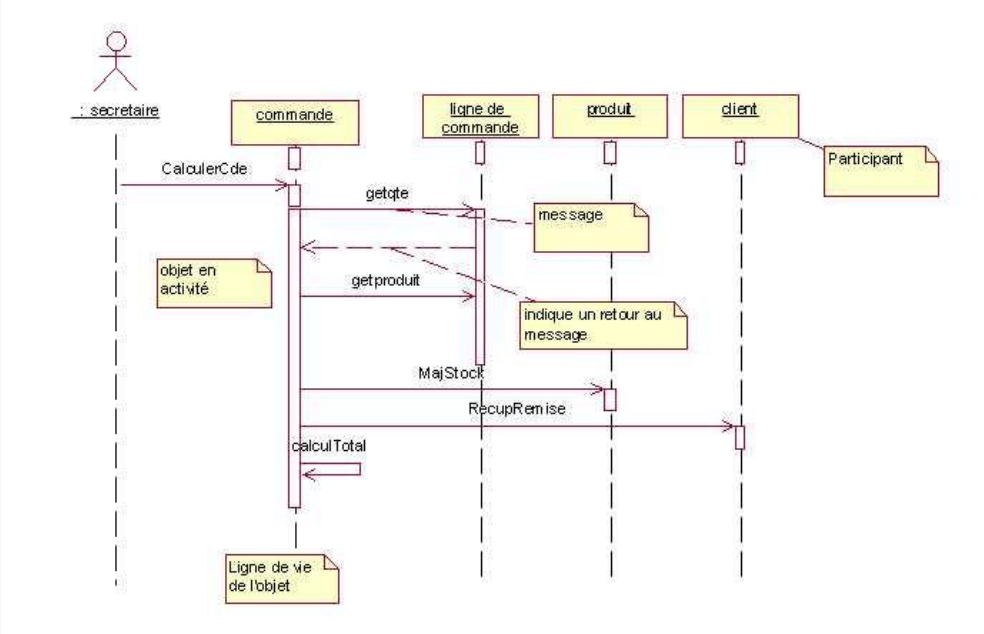
R2·01b

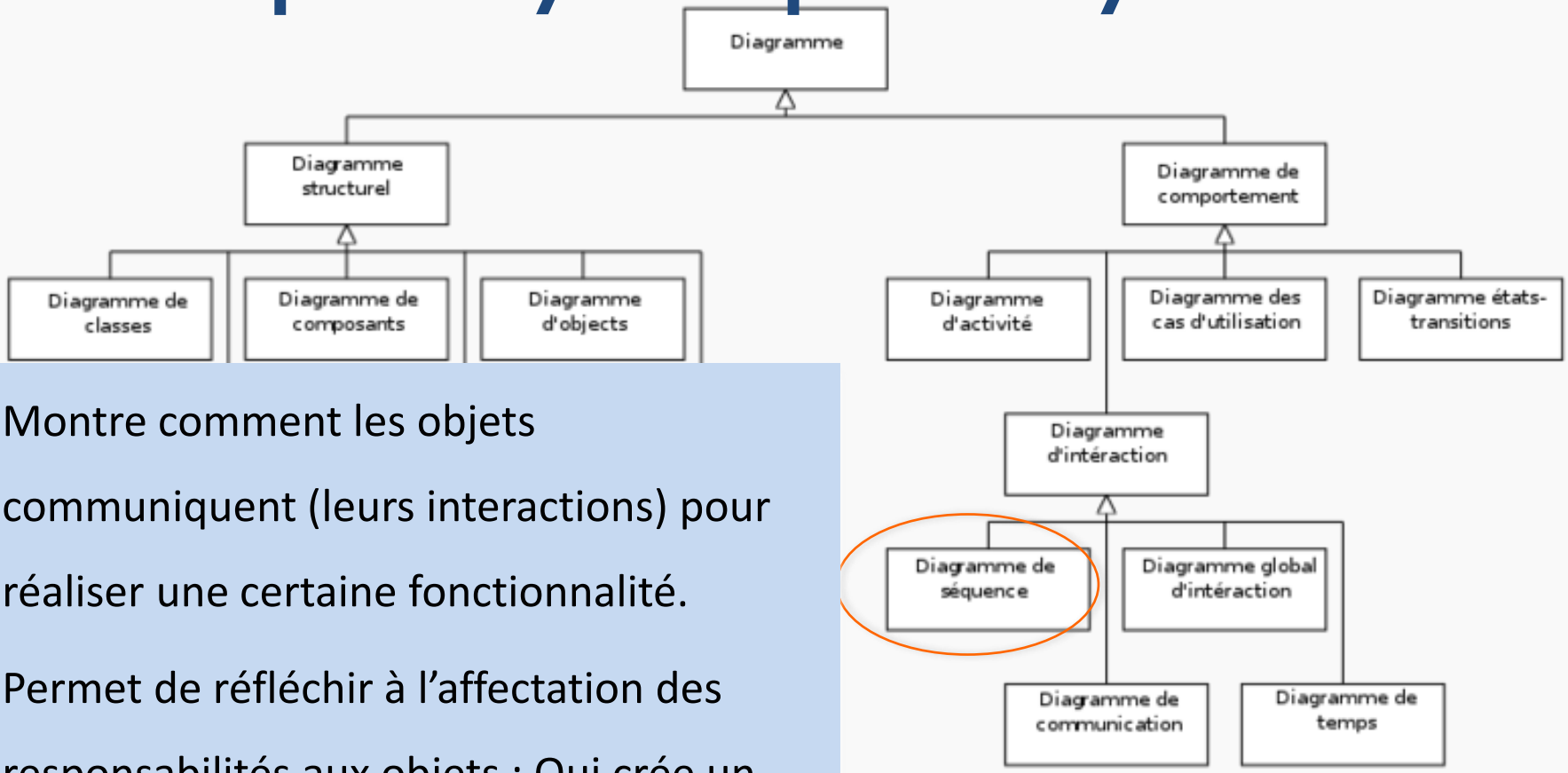
Bases de la conception orientée objet

Partie 5

UML / Diagramme de séquences



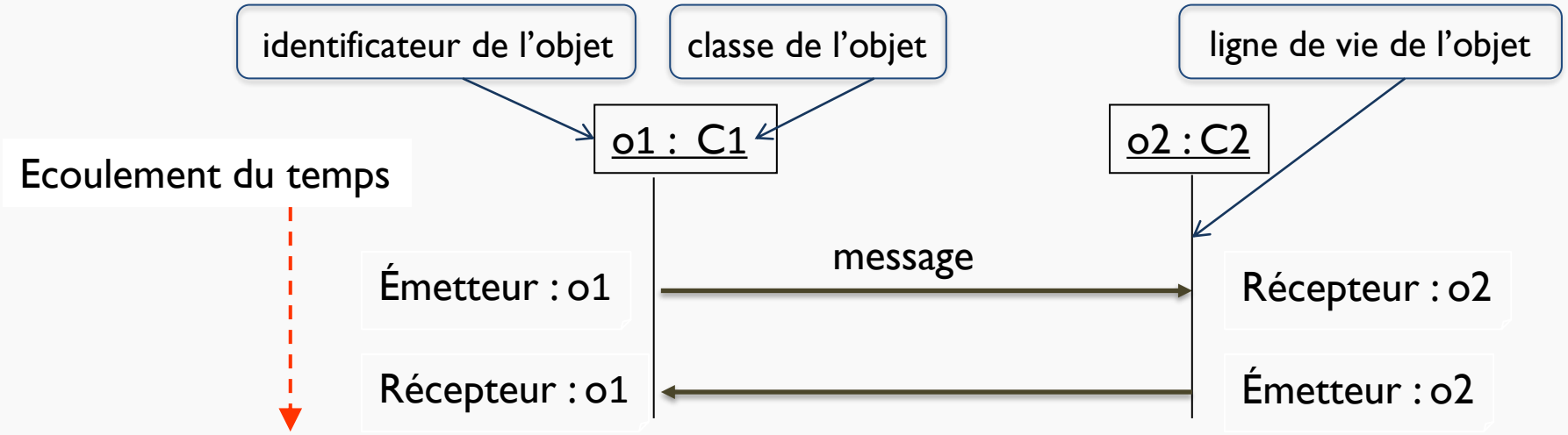
Aspects dynamiques du système



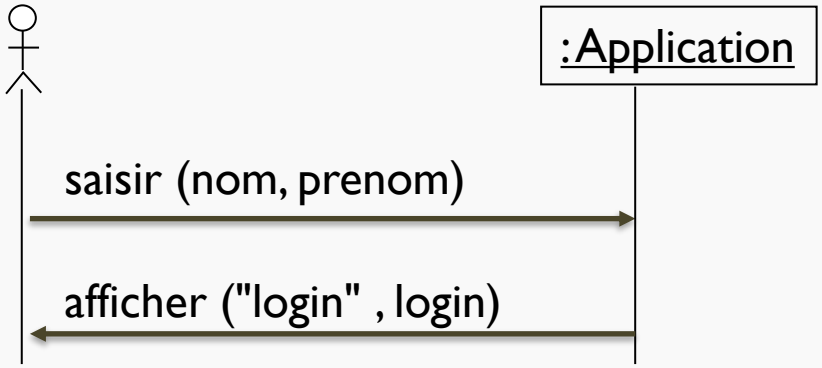
- Montre comment les objets communiquent (leurs interactions) pour réaliser une certaine fonctionnalité.
- Permet de réfléchir à l'affectation des responsabilités aux objets : Qui crée un objet ? Qui permet d'accéder à un objet ? Comment les objets se partagent-ils les responsabilités ?

Diagramme de séquences

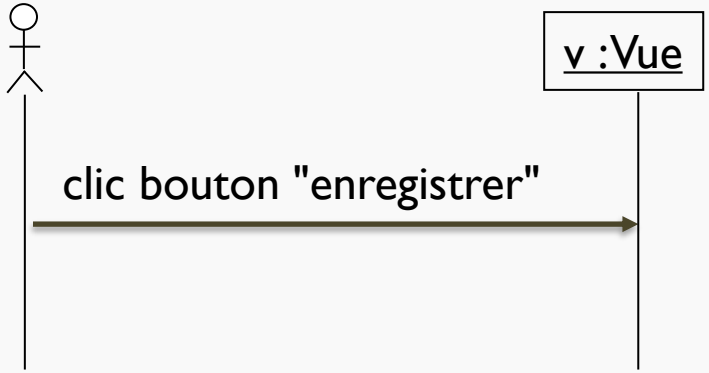
- Un diagramme de séquences permet de représenter les interactions entre les objets d'un point de vue temporel lors de l'exécution d'un scénario.
- **Scénario** : séquence particulière d'interactions lors de l'exécution d'une fonctionnalité du système.
- **Message** : forme de communication entre objets (flot de données, évènement, appel d'opération, signal, …)
- Un message a un objet émetteur et un objet récepteur.



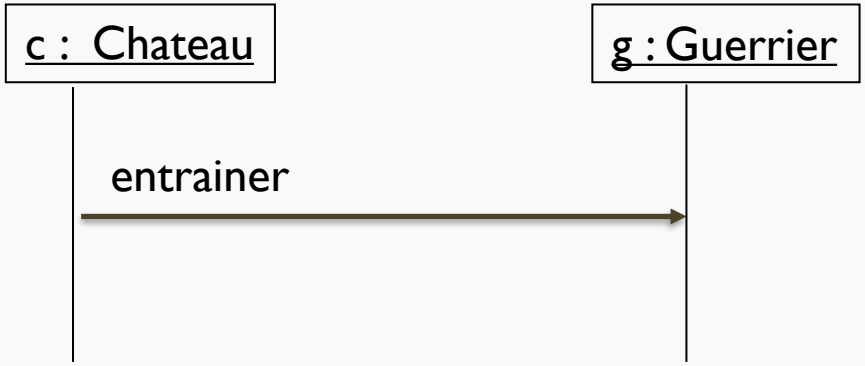
Exemples de messages



flot de données



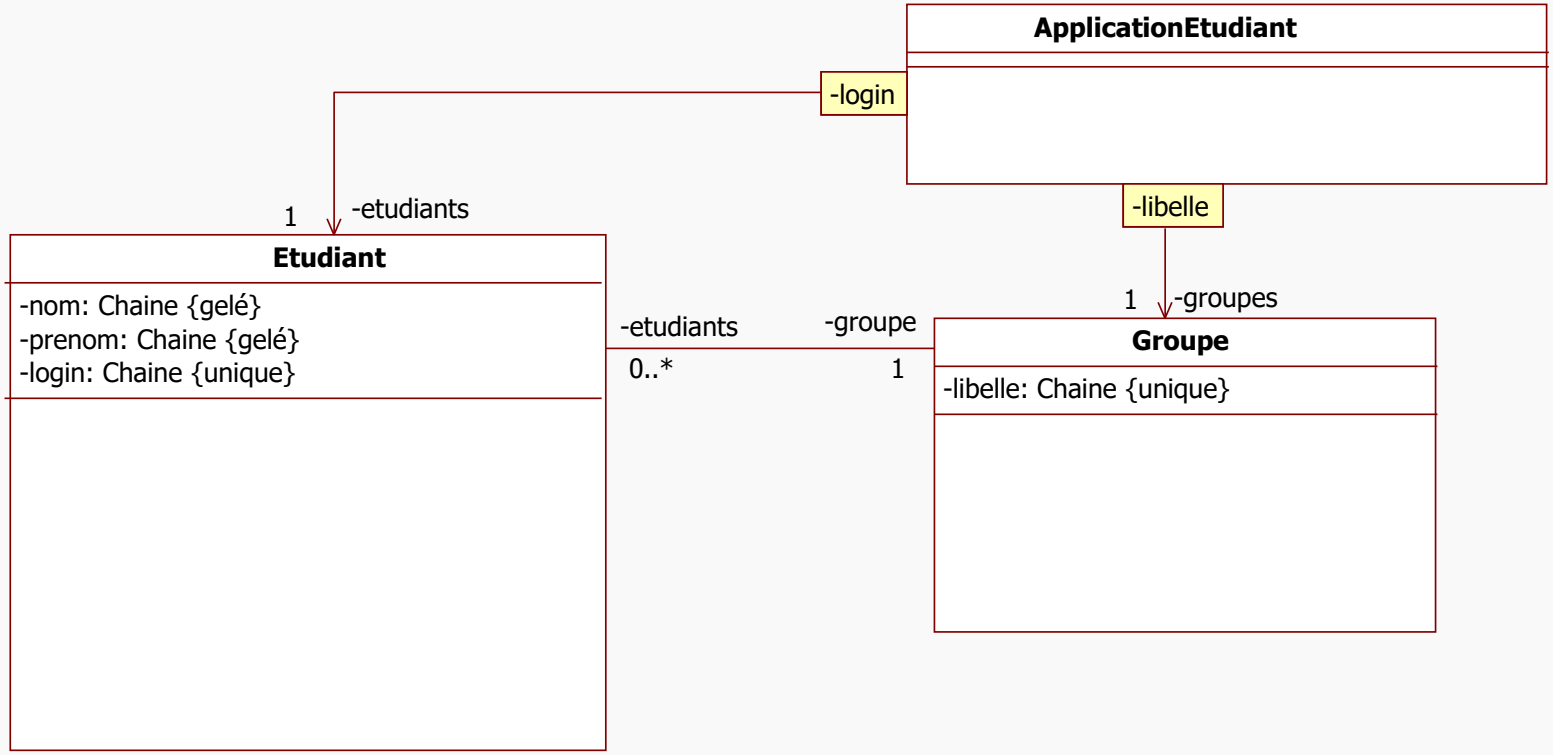
signal



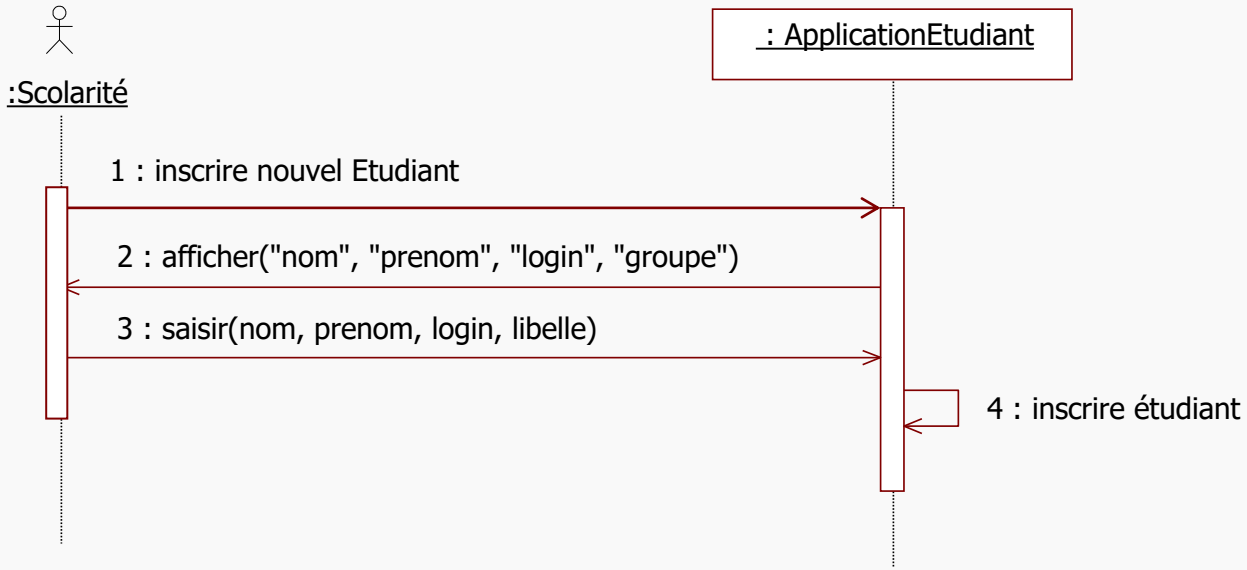
appel d'opération (conception intermédiaire)

appel d'opération (conception détaillée)

Exemple
Diagramme de classes

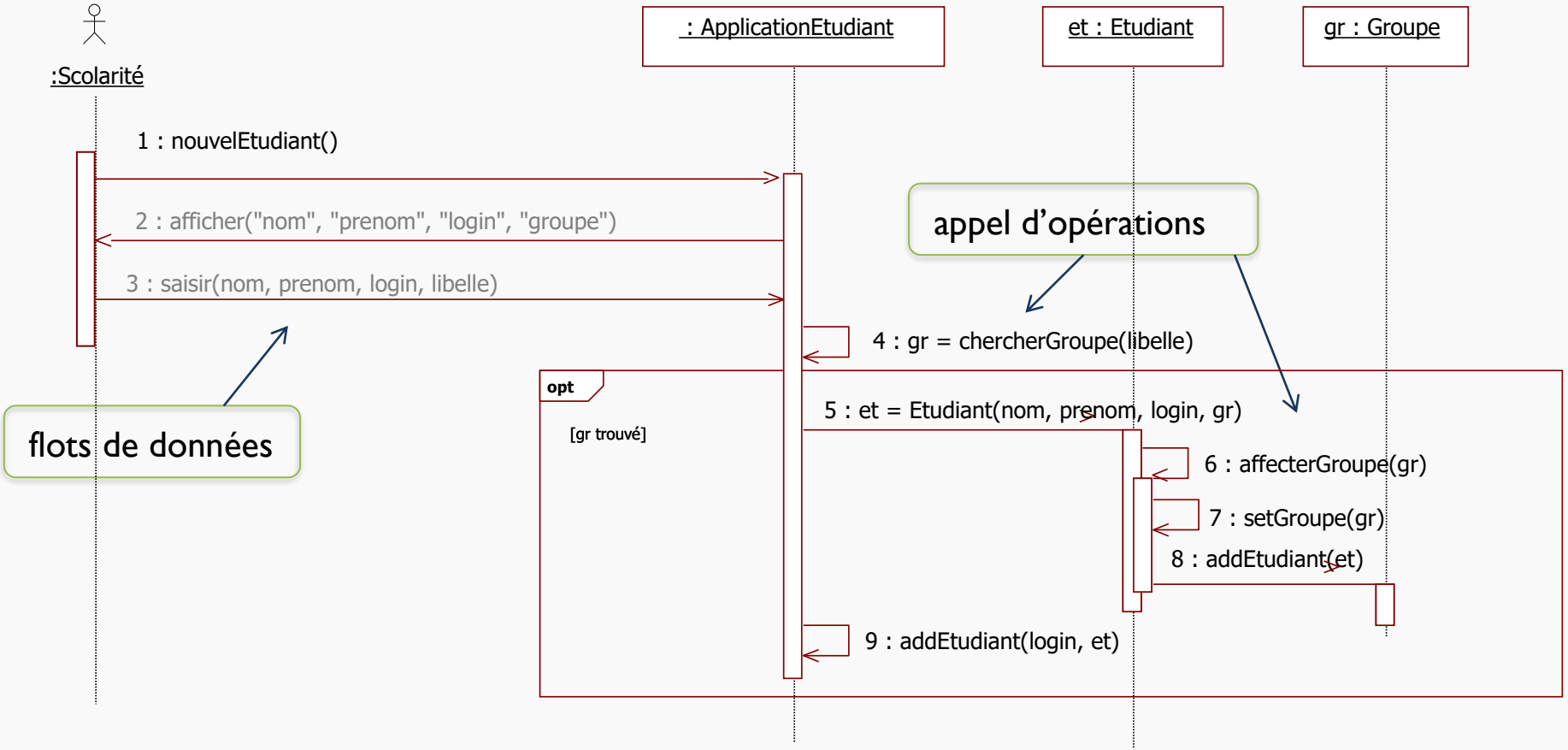


Exemple – Niveau Analyse
Diagramme de séquences haut niveau

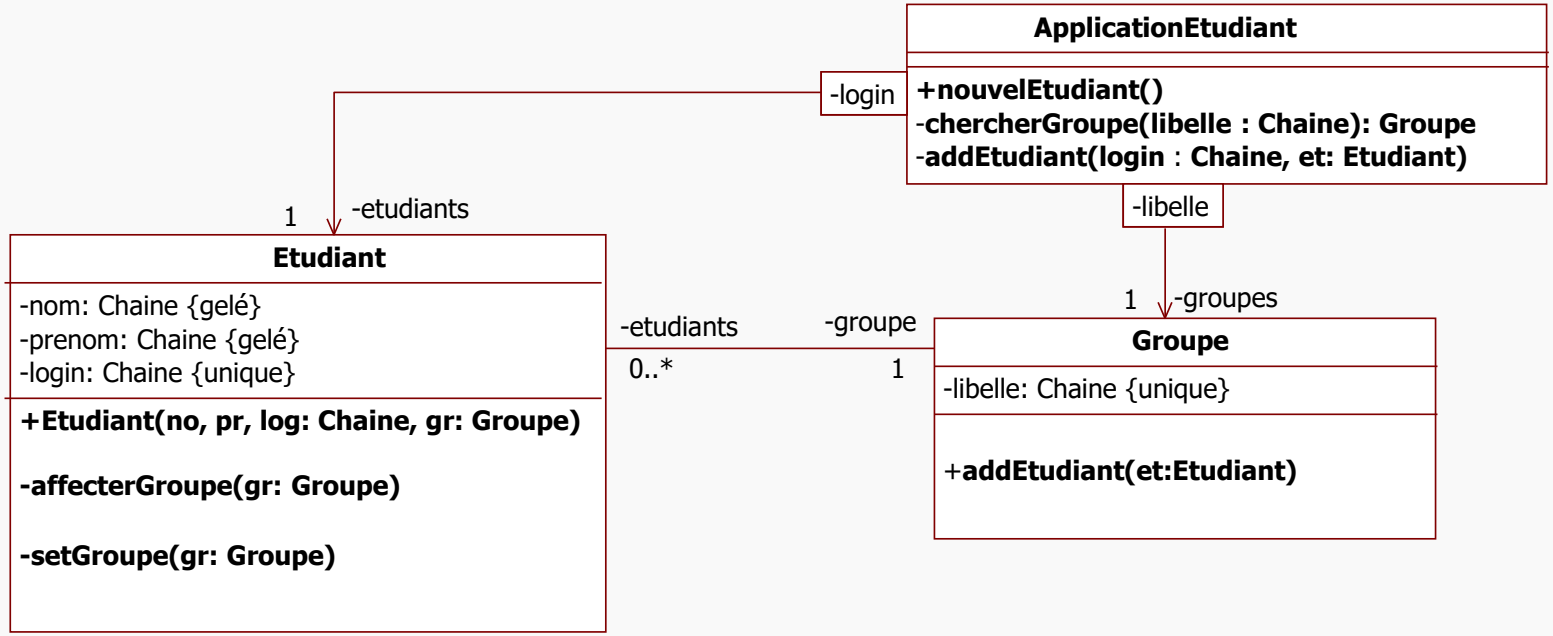


- Description d'un scénario de cas d'utilisation
- Description des interactions entre l'acteur et le système (phase analyse)
- Le système est vu comme une boite noire. Il sera détaillé en conception.
- Formalisme libre pour les messages.

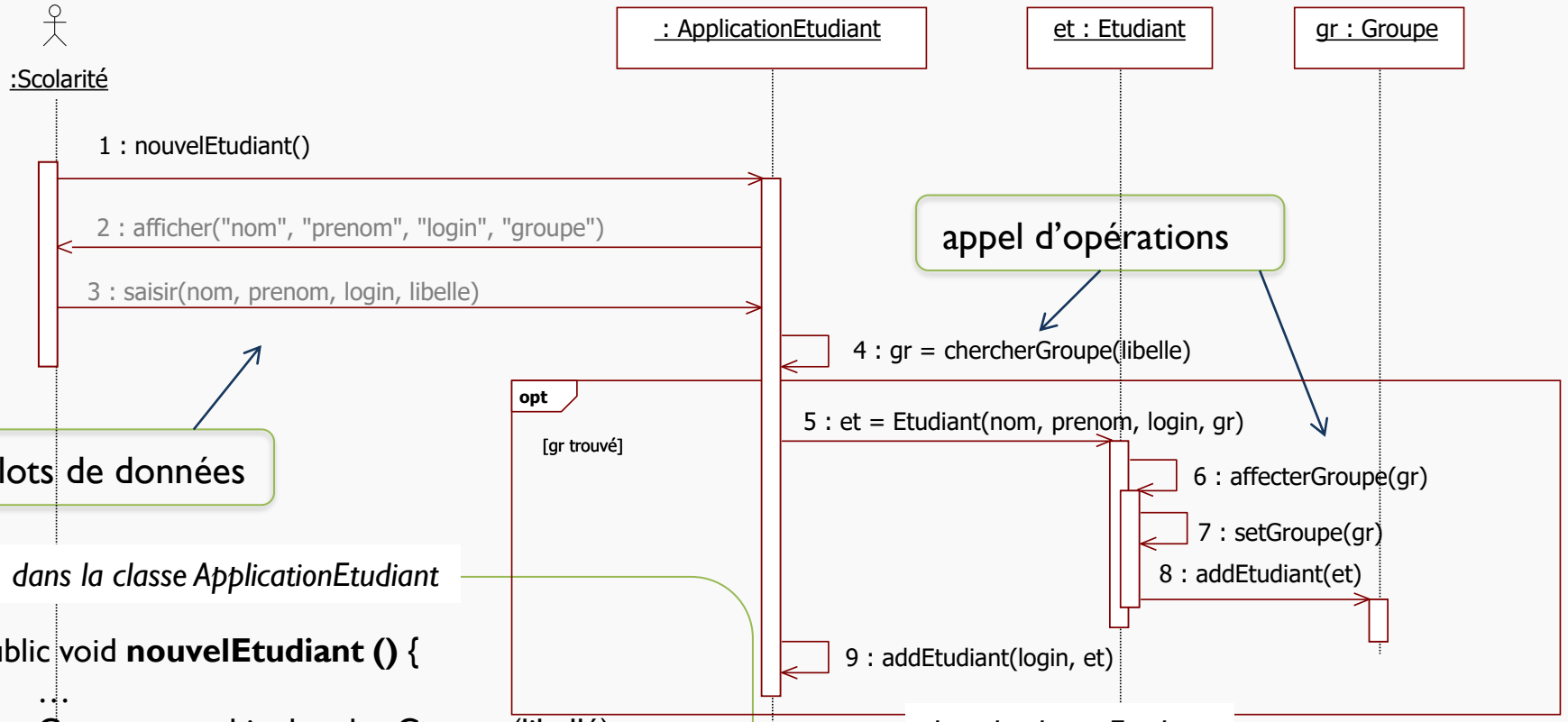
Exemple – Niveau conception
Diagramme de séquences détaillé



Exemple
Diagramme de classes complété



Exemple – Codage des Diagramme de séquences détaillés



dans la classe ApplicationEtudiant

```
public void nouvelEtudiant () {
    …
    Groupe gr = this.chercherGroupe (libellé);
    if (gr != null ) {
        Etudiant et = new Etudiant (nom, …, gr);
        this.addEtudiant (login, et);
    }
}
```

dans la classe Etudiant

```
private void affecterGroupe (Groupe gr) {
    this.setGroupe(gr);
    gr.addEtudiant(this);
}
```

Exemple Code Java

```
public class Etudiant {
private String nom ;  // prenom, login
private Groupe groupe ;
public Etudiant (String no, Groupe gr) {
        this.setNom(no);
        this.affecterGroupe(gr);
}
private void affecterGroupe (Groupe gr) {
        this.setGroupe(gr);
        gr.addEtudiant(this);
}
```

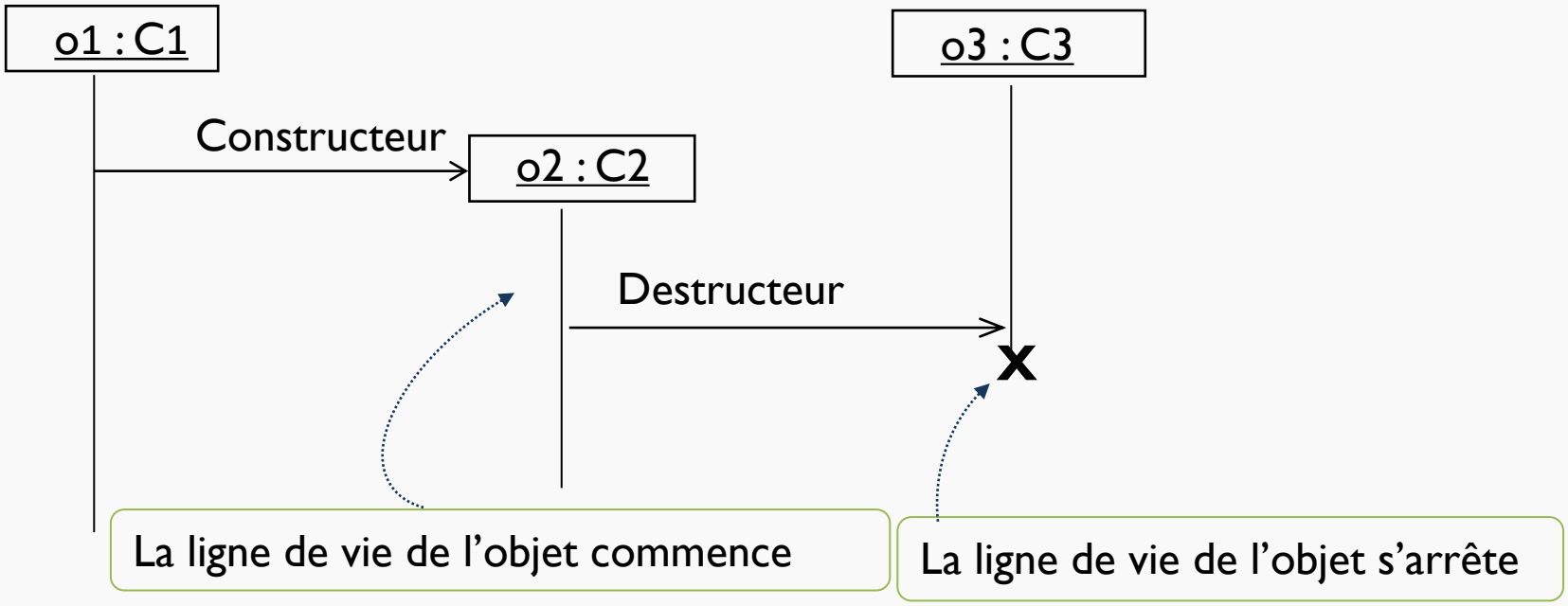
```
public class Groupe  {
private String libelle ;
private ArrayList< Etudiant > etudiants ;

public Groupe (String lib) {
        this.setLibelle(lib) ;
        etudiants = new ArrayList <>() ;
}
public void addEtudiant(Etudiant) {
        this.getEtudiants().add(et);
}
```

```
public class ApplicationEtudiant {
private HashMap<String, Groupe> groupes;
private HashMap<String, Etudiant> etudiants;
public void nouvelEtudiant () {
        Scanner sc = new Scanner(System.in);
        System.out.print("Nom : " );
        String nom = sc.nextLine(); …. ;
        System.out.print("Libellé du groupe : " );
        String libelle = sc.nextLine();
        Groupe gr = this.chercherGroupe (libellé);
        if (gr != null ) {
                Etudiant et = new Etudiant (nom, …, gr);
                this.addEtudiant (login, et);
                }
        }
        private Groupe chercherGroupe(String libelle) {
                return this.getGroupes(). get(libelle);
        }
        private addEtudiant(String log, Etudiant et) {
                this.getEtudiants().put (login, et);
        }
        }
```

Diagramme de séquences

- La représentation se concentre sur l'expression des interactions.
- L'accent est mis sur la chronologie des interactions.
- Un message est la représentation d'une communication au cours de laquelle des informations sont échangées.
- L'ordre des messages est donné par leur position sur la ligne de vie. Ils peuvent être numérotés.
- Message synchrone : l'émetteur reste bloqué le temps que dure l'invocation de l'opération.

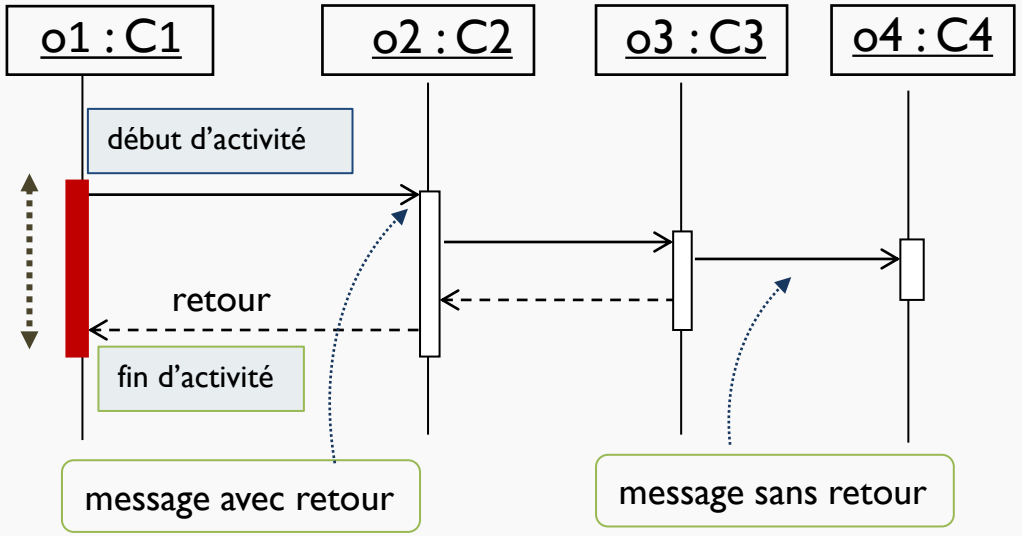
Ligne de vie



En Java, un objet qui n'est plus référencé est détruit automatiquement.

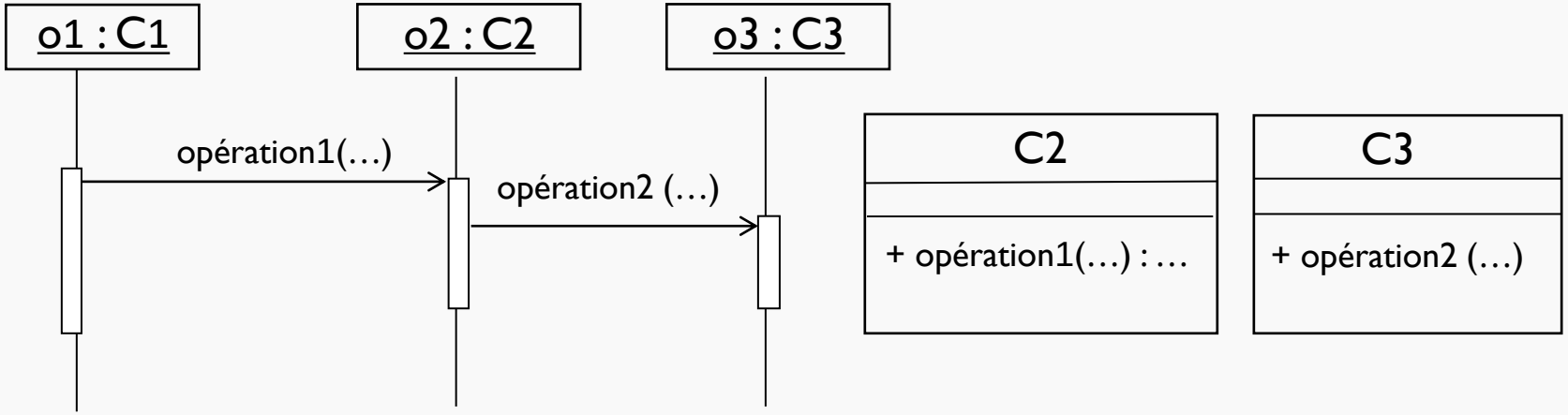
Activation d'un objet

Période d'activation :
temps pendant lequel l'objet exécute une action soit directement soit par l'intermédiaire d'un autre objet.
Il reste actif s'il attend le retour d'un envoi de message.



La réception d'un message provoque une période d'activité (rectangle vertical sur la ligne de vie) marquant le traitement du message (spécification d'exécution dans le cas d'un appel d'opération).
Lorsqu'il y a des appels d'opérations en cascade, chaque traitement emboîté doit se terminer pour que le traitement englobant reprenne le contrôle.

Message et diagramme de classes



L'envoi d'un message à un objet est la sollicitation d'un service qui est offert par la classe de cet objet.

➡ **appel d'une opération de la classe réceptrice du message**.

o2.opération1 (…)

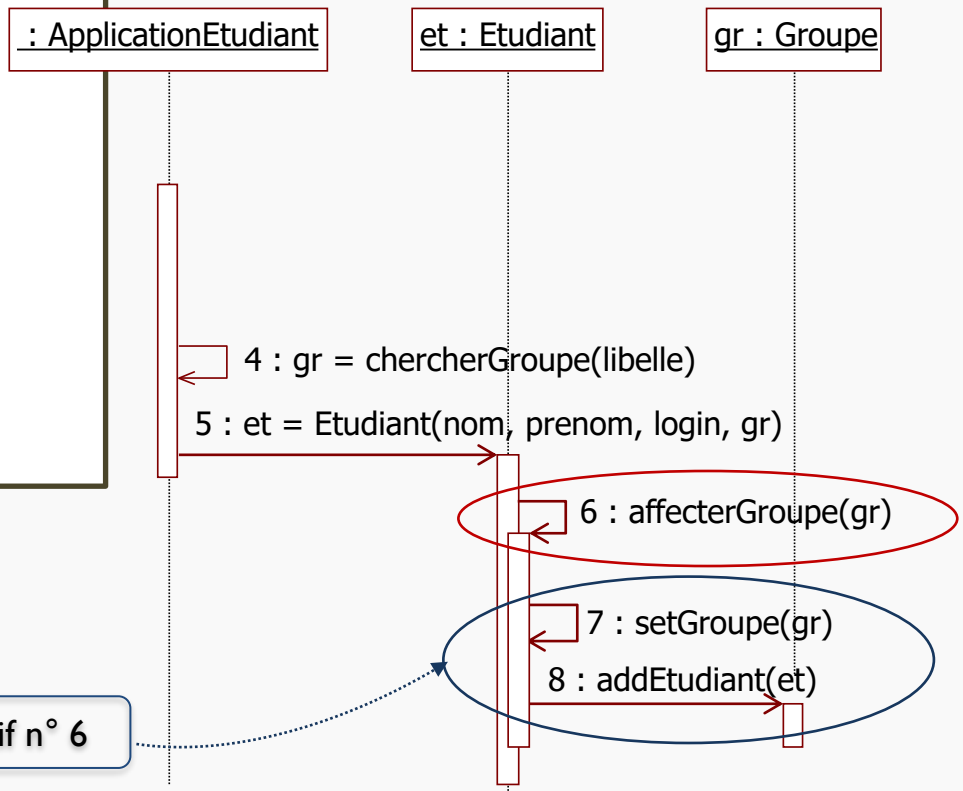
o3. opération2 (…)

Message réflexif

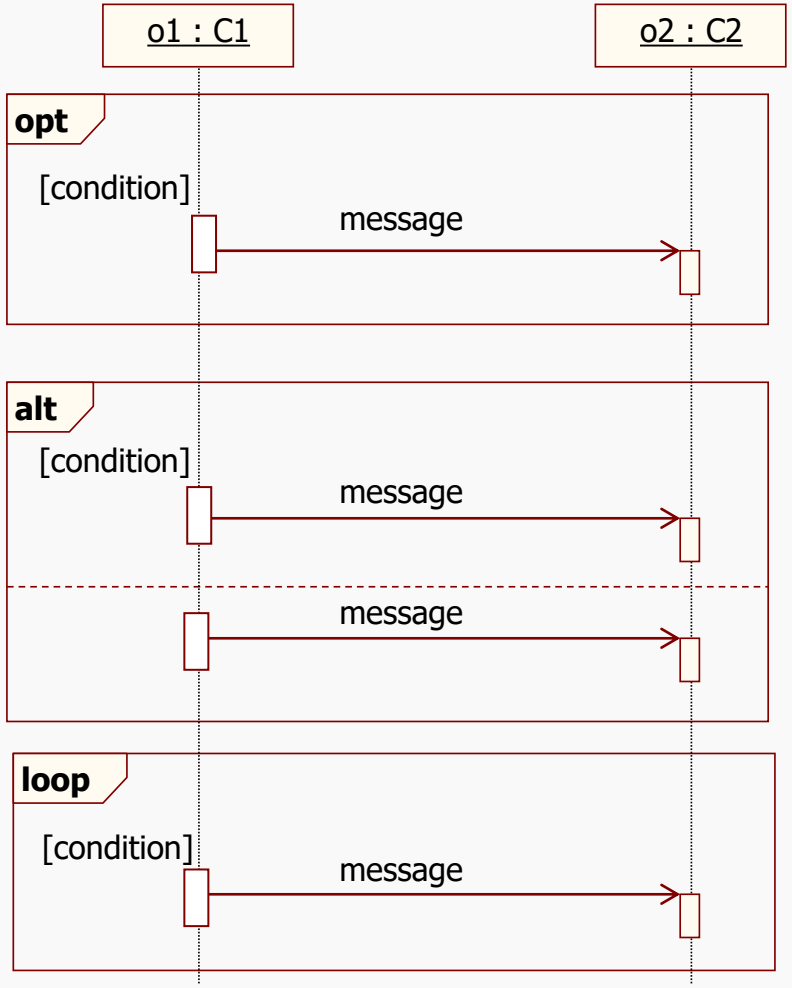
Un objet peut s'envoyer un message à lui-même :

appel d'une opération de la classe de l'objet

```
public class Etudiant {
private String nom ;  // prenom, login
private Groupe groupe ;
public Etudiant (String no, Groupe gr) {
        this.setNom(no);
        this.affecterGroupe(gr);
}
private void affecterGroupe (Groupe gr) {
        this.setGroupe(gr);
        gr.addEtudiant(this);
}
```



Structures de contrôle



message ou séquence de messages optionnels

[condition] : expression booléenne

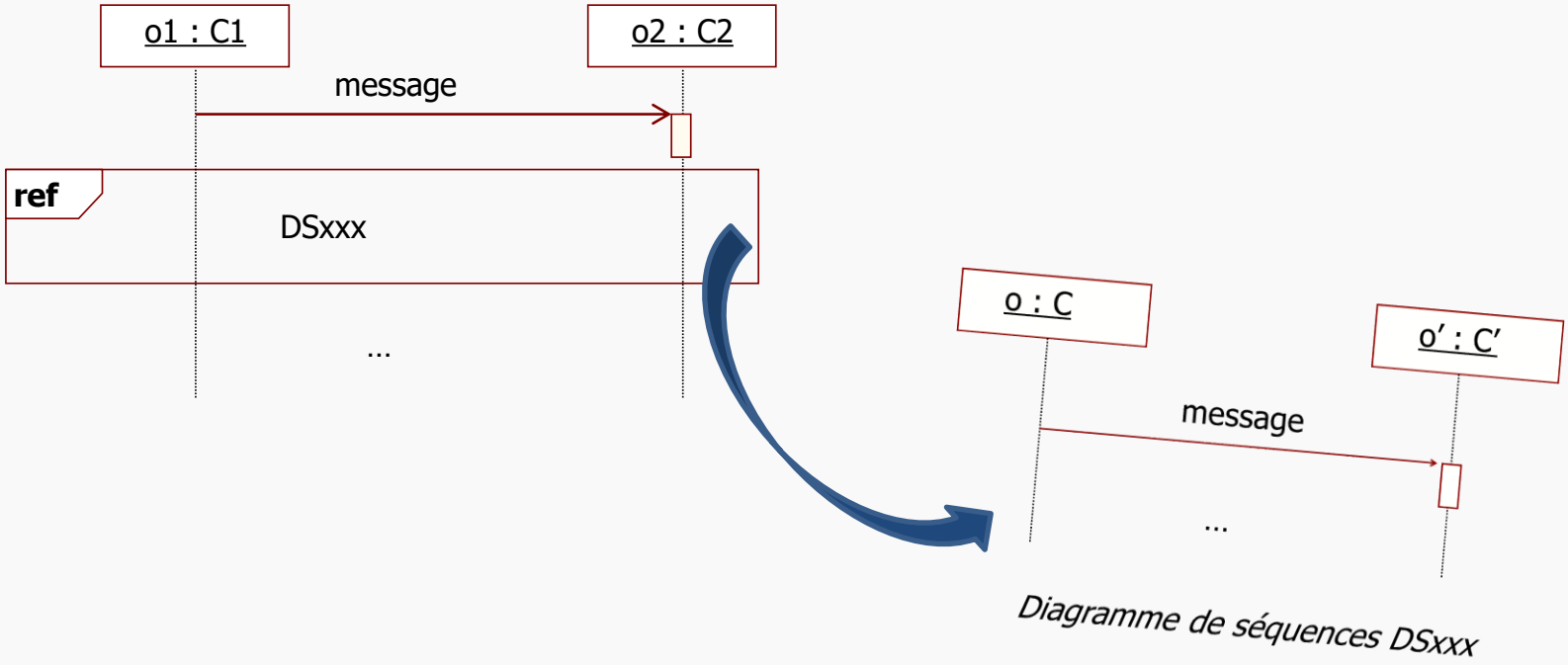
message ou séquence de messages alternatifs

si [condition] …. alors …..sinon ….

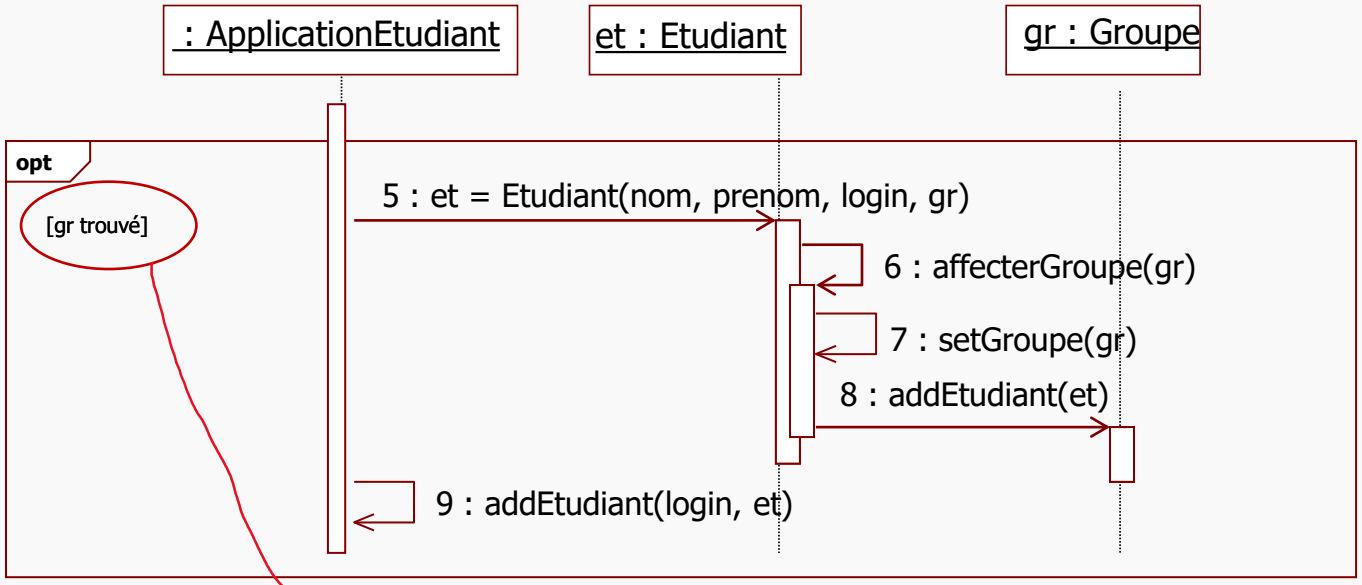
itération d'un message ou d'une séquence de messages
[condition] : condition d'itération

Structures de contrôle

- **ref** est utilisé pour indiquer une référence vers un autre diagramme de séquences.
- Son rôle est de factoriser des parties de comportement communs à plusieurs scénarios et ou de donner une vision globale d'un DS.



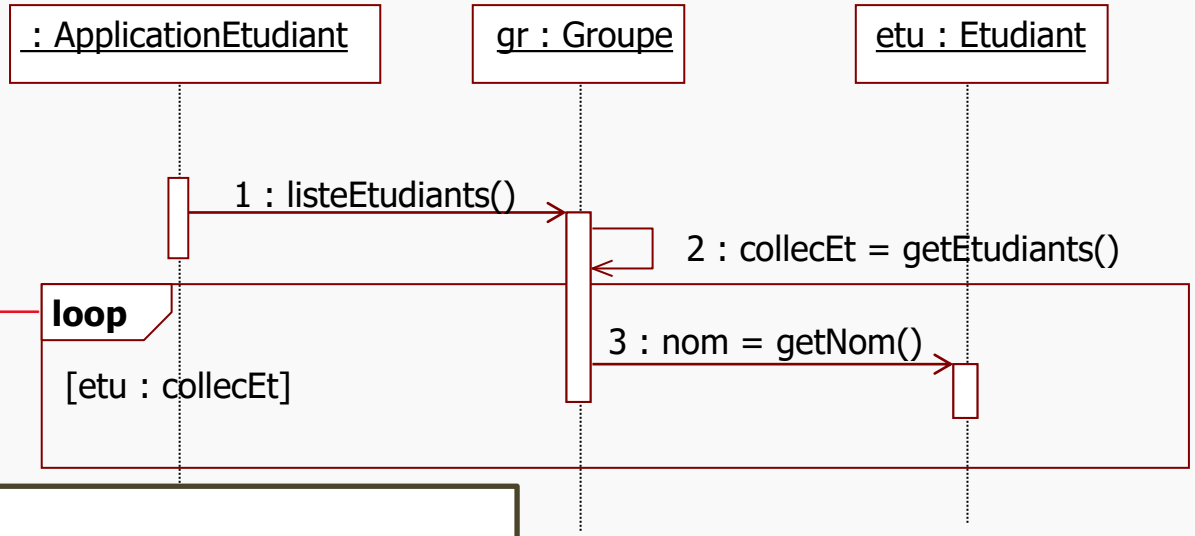
Exemple messages conditionnés




```
public class ApplicationEtudiant {
private HashMap<Groupe> groupes;
public void nouvelEtudiant () {
        Scanner sc = new Scanner(System.in);
        System.out.print("Nom : " );
        String nom = sc.nextLine();
        …
        System.out.print("Libellé du groupe : " );
        String libelle = sc.nextLine();
        Groupe gr = this.chercherGroupe (libellé);
        if (gr != null ) {
        Etudiant et = new Etudiant (nom, …, gr);
        …
        }
}
private Groupe chercherGroupe(String libelle) {
        return this.getGroupes(). get(libelle);
        }
}
```

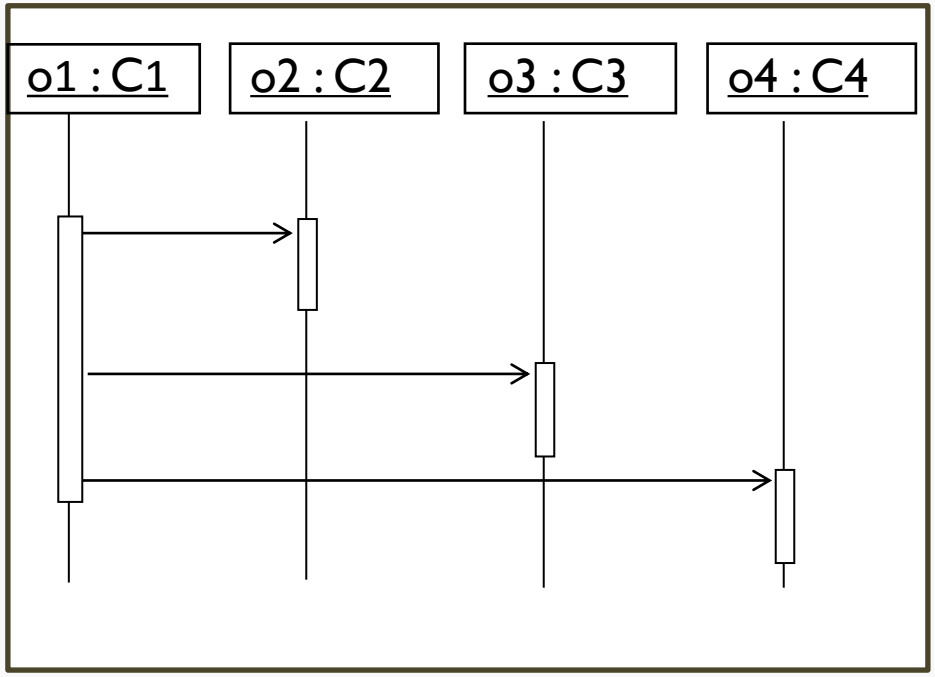
Exemple itération de messages

- Liste des étudiants d'un groupe



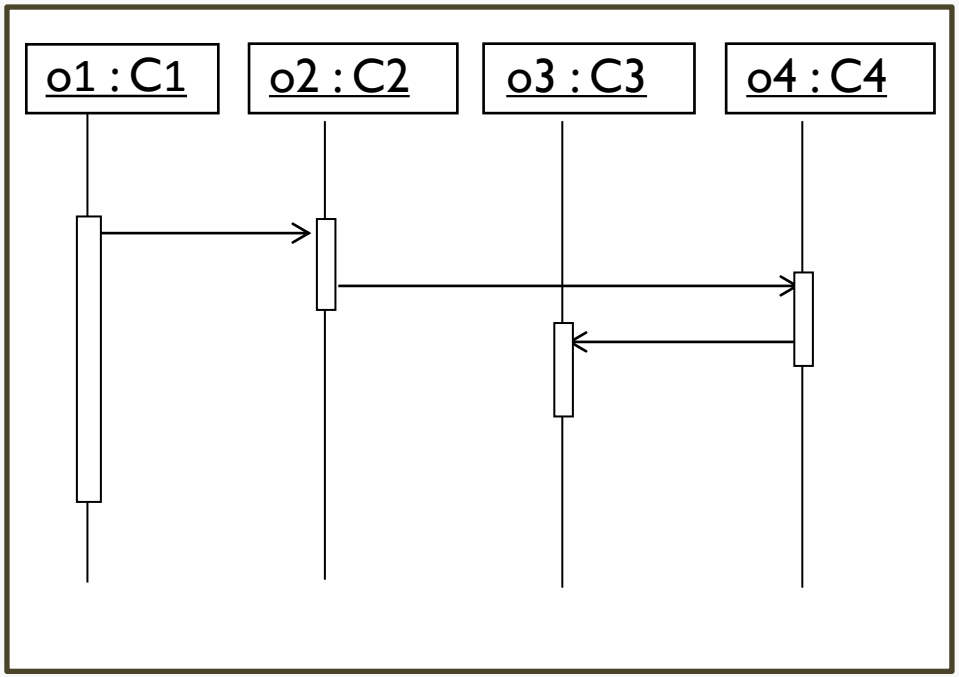

```
public class Groupe  {
private  ArrayList< Etudiant > etudiants ;

public void listeEtudiants() {
      for (Etudiant etu :  this.getEtudiants()) {
            String nom = etu.getNom();
      }
}
private  ArrayList< Etudiant > getEtudiants() {
      return etudiants;
}
```

Forme de contrôle



Contrôle centralisé

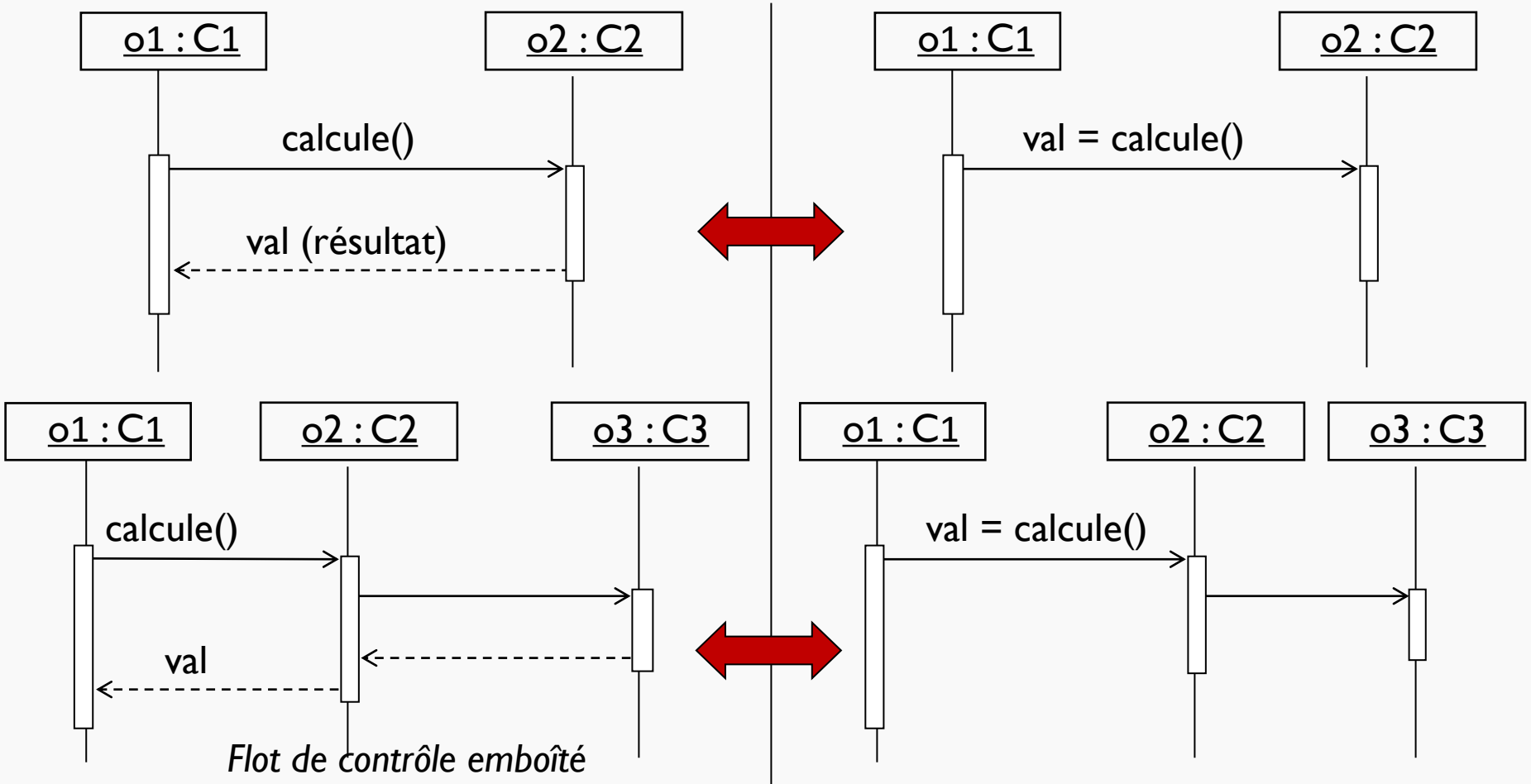
o1 active successivement les autres objets
o1 est le chef d'orchestre



Contrôle décentralisé

o1 ne domine pas mais délègue des responsabilités à d'autres objets

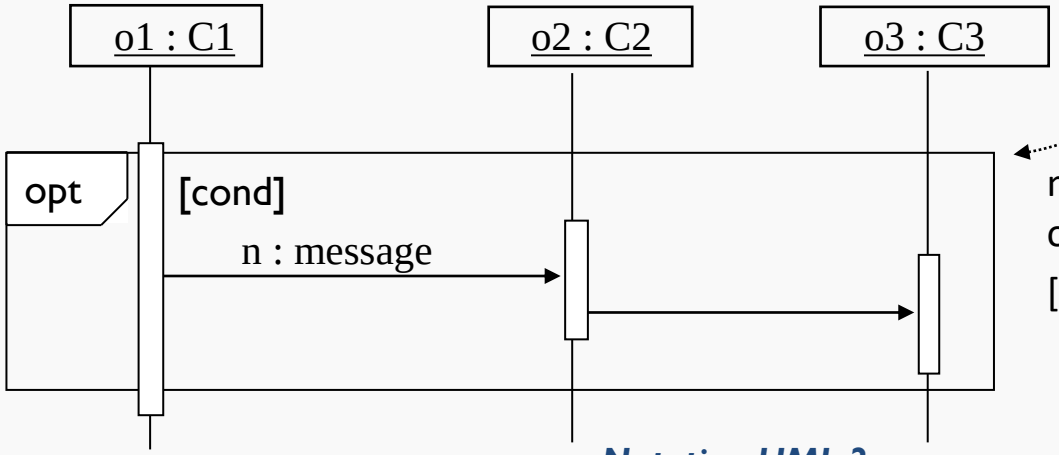
Retour de message
Notation simplifiée



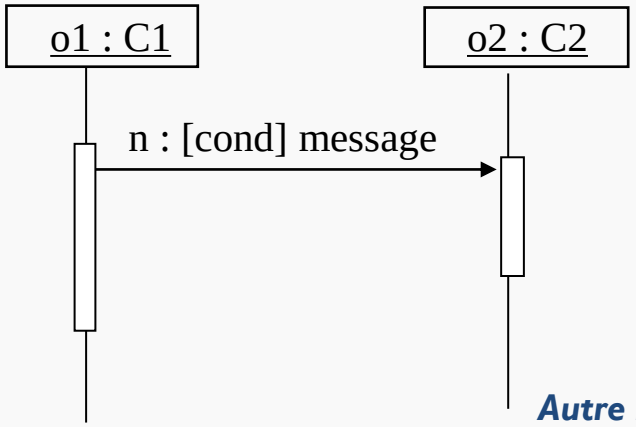
Flot de contrôle emboîté

La séquence emboîtée doit se terminer pour que la séquence englobante reprenne le contrôle.

Structures de contrôle
Notation simplifiée



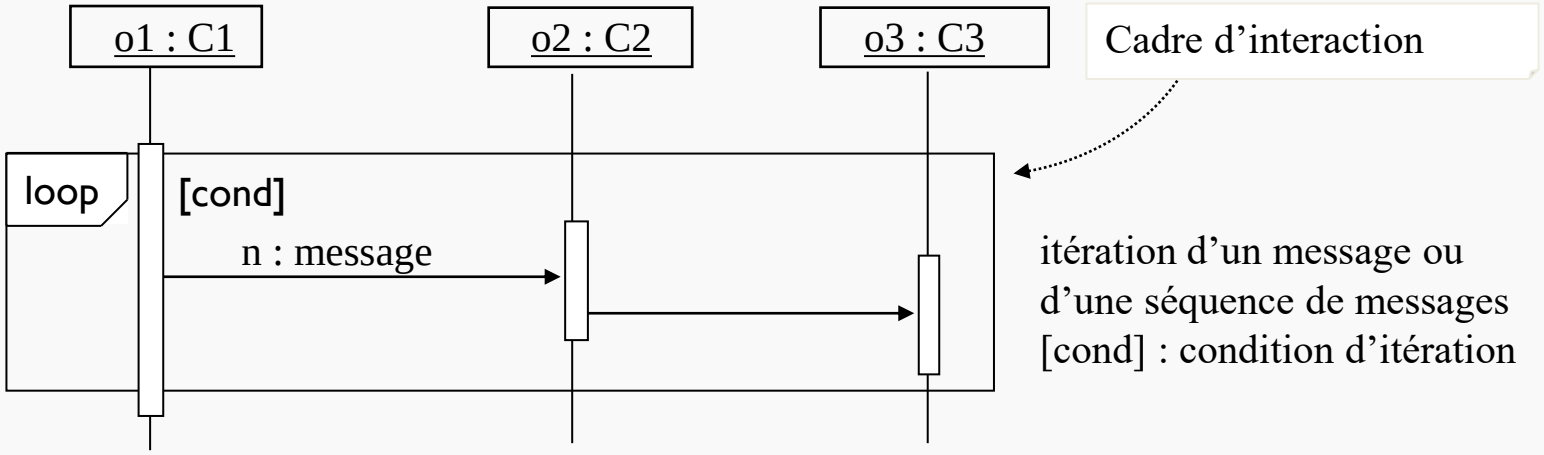
Notation UML 2



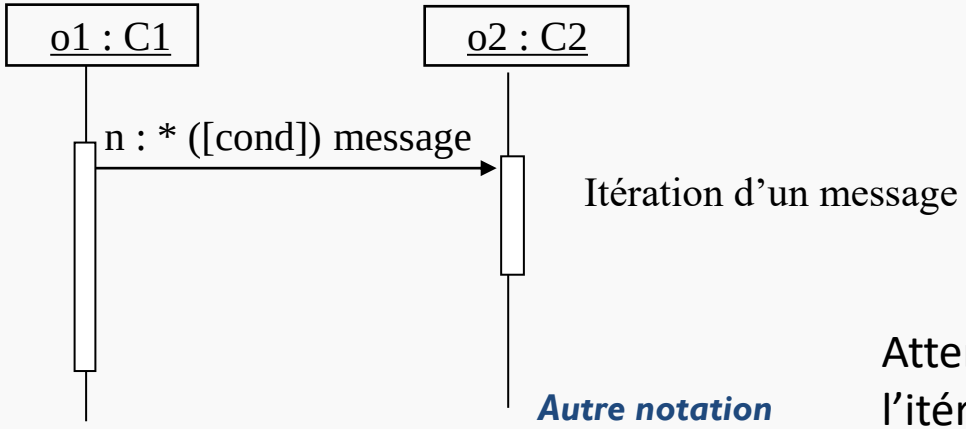
Autre notation

Attention : ne peut être utilisé que si le non respect de la condition entraine un arrêt de la fonctionnalité

Structures de contrôle
Notation simplifiée



Notation UML 2



Autre notation

Attention : ne peut être utilisé que si l'itération ne porte que sur un seul message

Dans un projet : fixer des notations communes

- Primitives d'entrées-sorties :
  - saisir (*n1, n2*, …) *utiliser les noms d'attribut du diagramme de classes*
  - afficher (… , …) *invites de saisie (ex. "nom") ou valeurs*

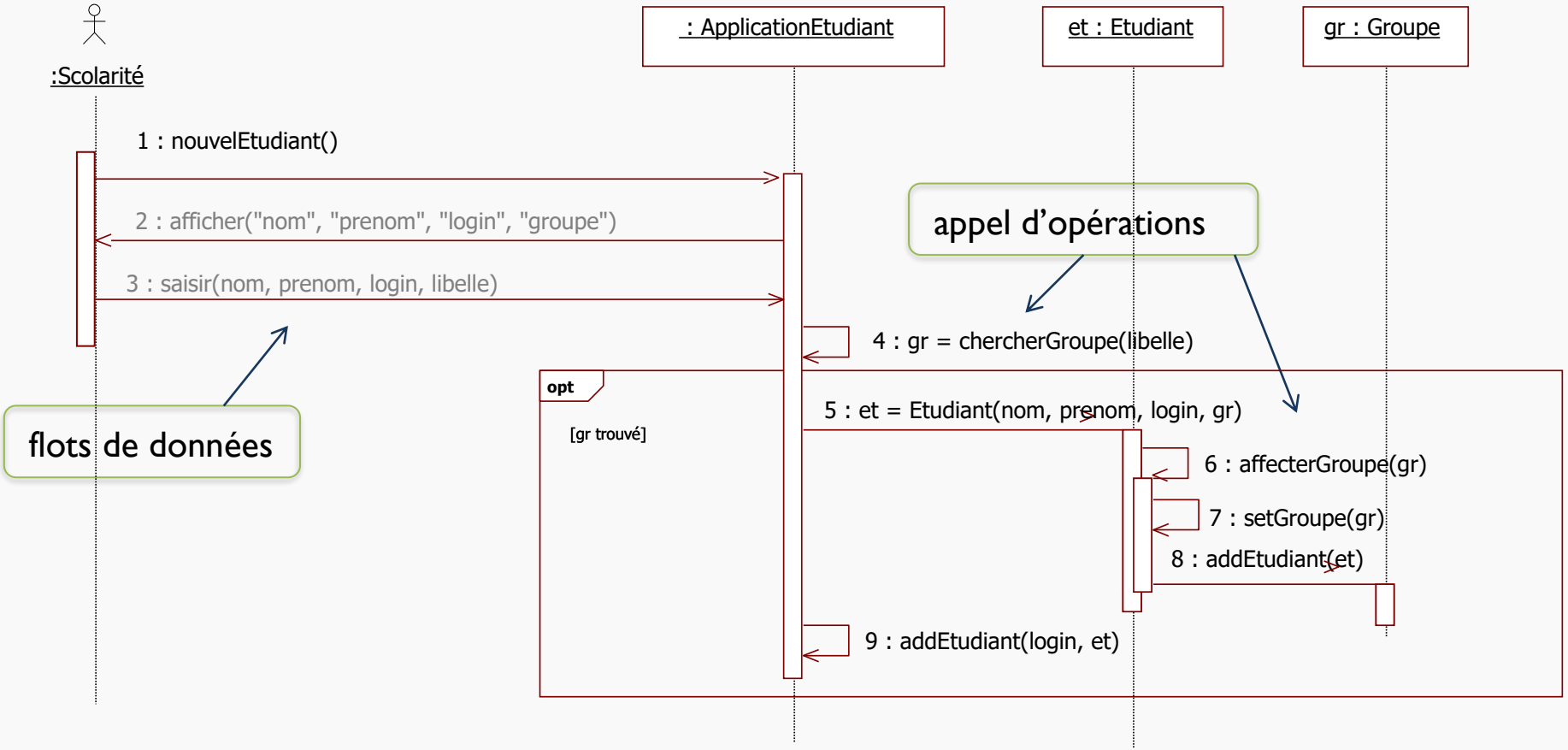
- Paramètres et résultat de l'invocation d'une opération
  - Soyez cohérents en particulier avec les noms des objets en interaction dans le diagramme de séquences (lignes de vie)
  - Pour les collections, utilisez collect*Nom*, *nom* étant le nom d'un objet en interaction dans le diagramme

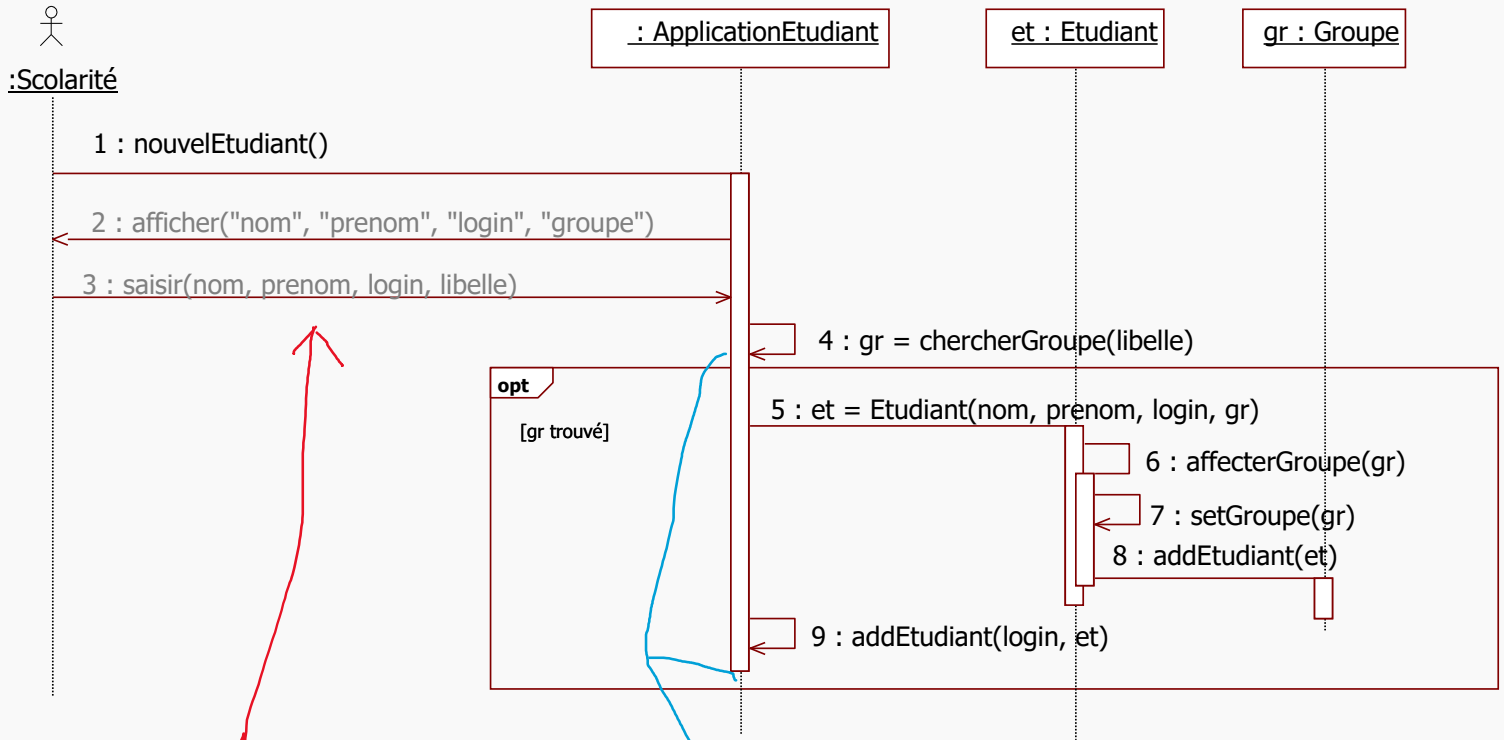
- Mise à jour de rôle d'association
  - setNom(…) *pour affecter ou modifier l'objet destination d'un rôle monovalué,*
  - addNom (…) *pour ajouter un élément à l'objet destination d'un rôle multivalué (2 paramètres dans le cas d'une association qualifiée)*

- Abréviation pour le retour des valeurs
  - au lieu de : no = getNom() pr = getPrenom() adr = getAdresse() …

  abrégez en : (no, pr, adr) = getValeurs()

Exemple – Niveau conception
Diagramme de séquences détaillé

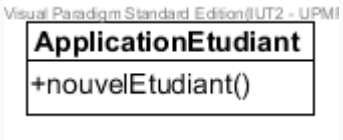




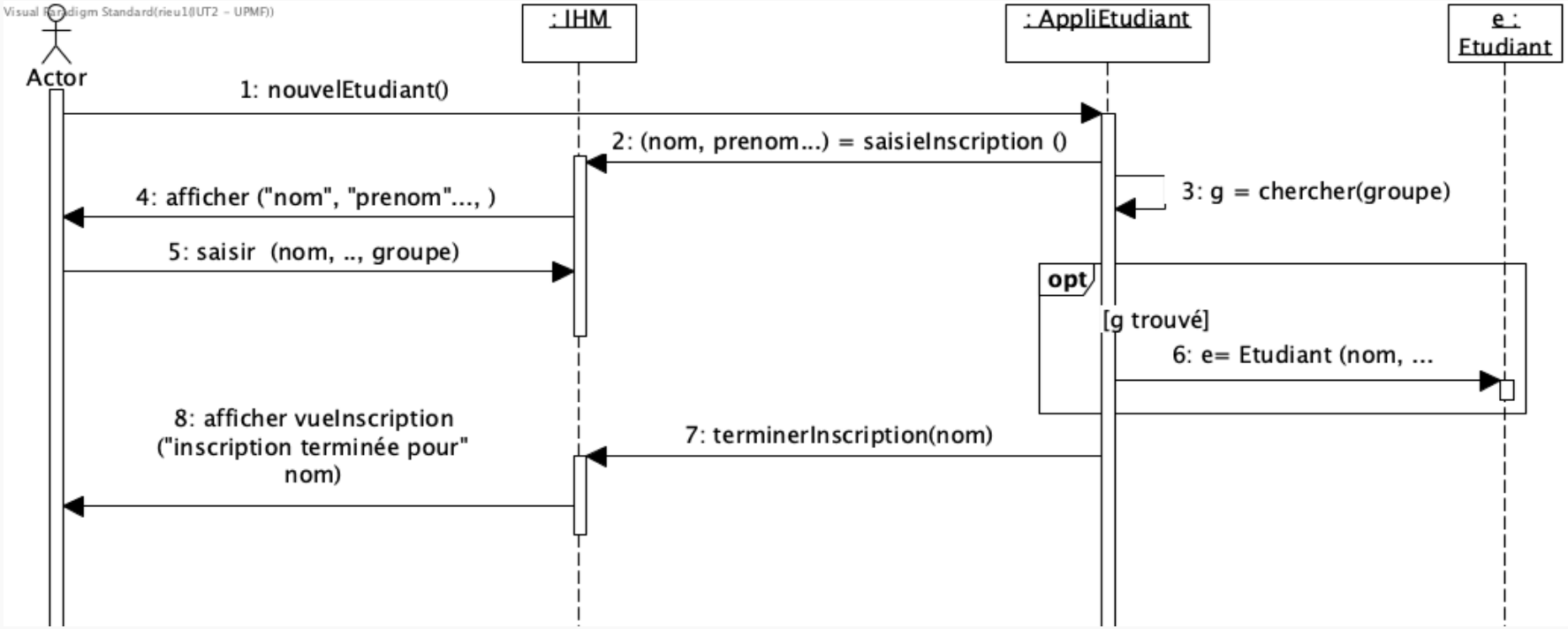
Dans nouvelEtudiant : interactions HM et code fonctionnel

```
public class ApplicationEtudiant {
private HashMap<String, Groupe> groupes;
private HashMap<String, Etudiant> etudiants;
public void nouvelEtudiant () {
    Scanner sc = new Scanner(System.in);
    System.out.print("Nom : " );
    String nom = sc.nextLine(); …. ;
    System.out.print("Libellé du groupe : " );
    String libelle = sc.nextLine();
    Groupe gr = this.chercherGroupe (libellé);
    if (gr != null ) {
        Etudiant et = new Etudiant (nom, …, gr);
        this.addEtudiant (login, et);
        }
}
private Groupe chercherGroupe(String libelle) {
        return this.getGroupes(). get(libelle);
}
private addEtudiant(String log, Etudiant et) {
        this.getEtudiants().put (login, et);
}
}
```

DS V2 : IHM – Classe application

- Classe Application 

détient la logique applicative et réalise les dialogues.
- Possibilité de regrouper dans une classe (IHM) les dialogues



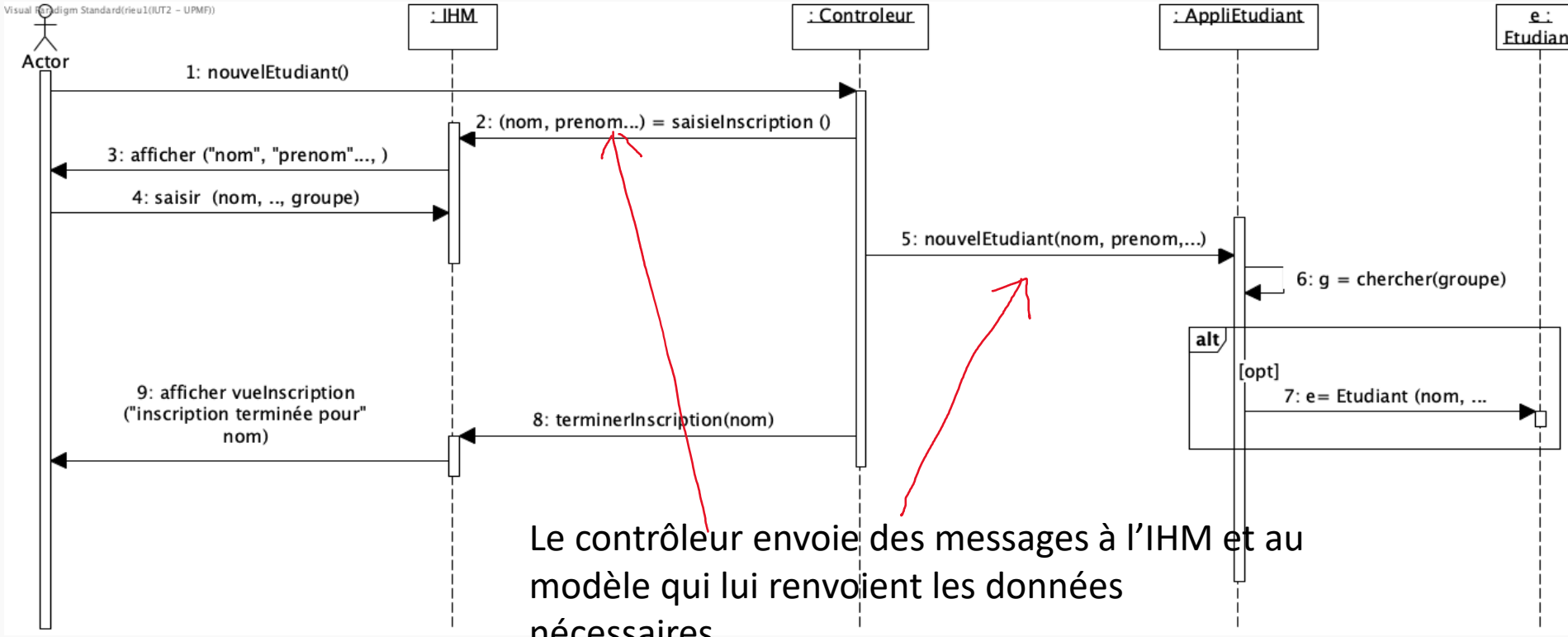
DS V3 : IHM-Contrôleur-Classe application

- *Approche MVC (Modèle, Vue, Contrôleur) : séparation claire entre la logique applicative et la gestion de l'interaction*

- Modèle : gestion de la logique applicative

- IHM : gestion de l'interaction

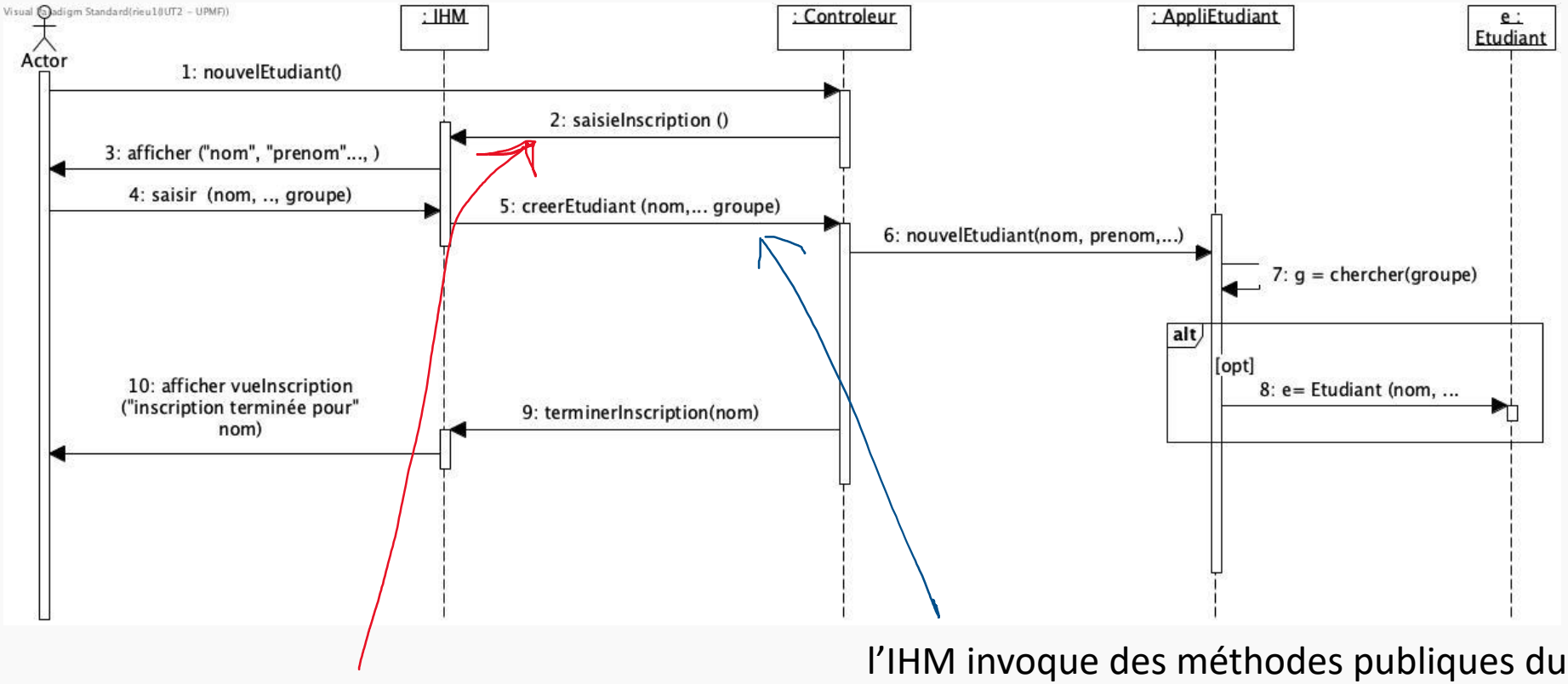
- Contrôleur : observateur du Modèle et de l'IHM, assure la traduction entre les objets métier et les objets IHM



Le contrôleur envoie des messages à l'IHM et au modèle qui lui renvoient les données nécessaires

DS V3' : IHM-Contrôleur-Classe application

- *Variante de l'approche précédente*
- *Avantage : interaction entre IHM et Contrôleur indépendante du type de l'interface (CLI : Command-Line Interface ou GUI : Graphical User Interface)*
- *Inconvénient : le contrôleur ne garde pas le contrôle global de l'exécution du cas d'utilisation*



Le contrôleur envoie des messages à l'IHM et au modèle

l'IHM invoque des méthodes publiques du contrôleur

Dans un projet : fixer des notations communes

- Primitives d'entrées-sorties :
  - ouvrir/ activer vueX
  - ouvrir / désactiver vueX
  - fermer vueX
  - saisir vueX (nom,…)
  - afficher vueX (« hhhhh », nom,…)
  - clic Nomboutton