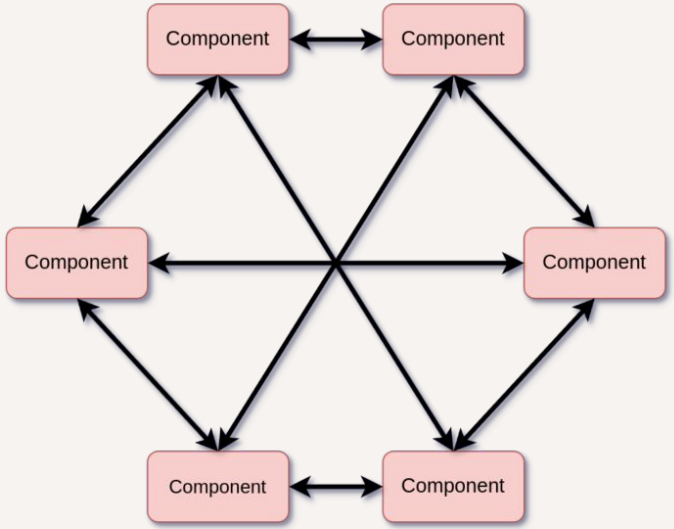
Mediator - Ant Colony Simulation

Quentin Surdez, Rachel Tranchida, Eva Ray

Table of Contents   

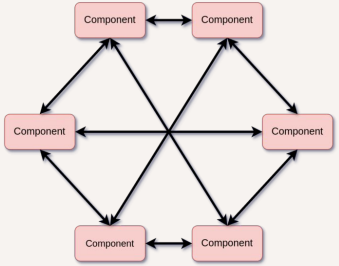
Rappel - Problématique

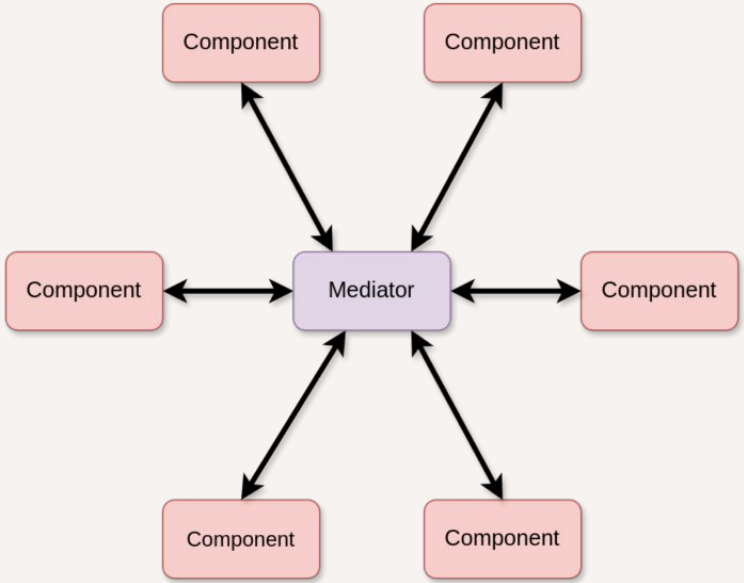
- Une prolifération d'interconnections peut **réduire la réutilisabilité** et rend un objet **dépendant** des autres.
- Il devient **difficile de changer le comportement** du système sans ajouter de nouvelles sous-classes.



Rappel - Solution

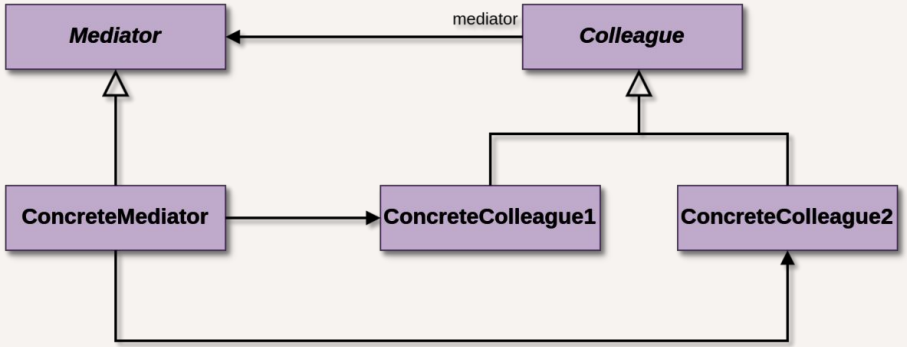
- Encapsulation du comportement dans un objet séparé appelé **Mediator**
- Le Mediator est responsable de **contrôler** et **coordonner les interactions** au sein d'un groupe d'objets
- Les objets n'ont connaissance que du Mediator, pas des autres objets -> **réduction des interconnections**





Rappel - Design Pattern Mediator

- Classe abstraite `**Mediator**` avec laquelle les objets vont dialoguer
- Implémentation d'un `**Concrete Mediator**` afin d'avoir les différentes communications implémentées
- Chaque `**Colleague**` aura une référence sur le `**Mediator**`
- `**Concrete Mediator**` possède des références sur les `**Concrete Colleague**`



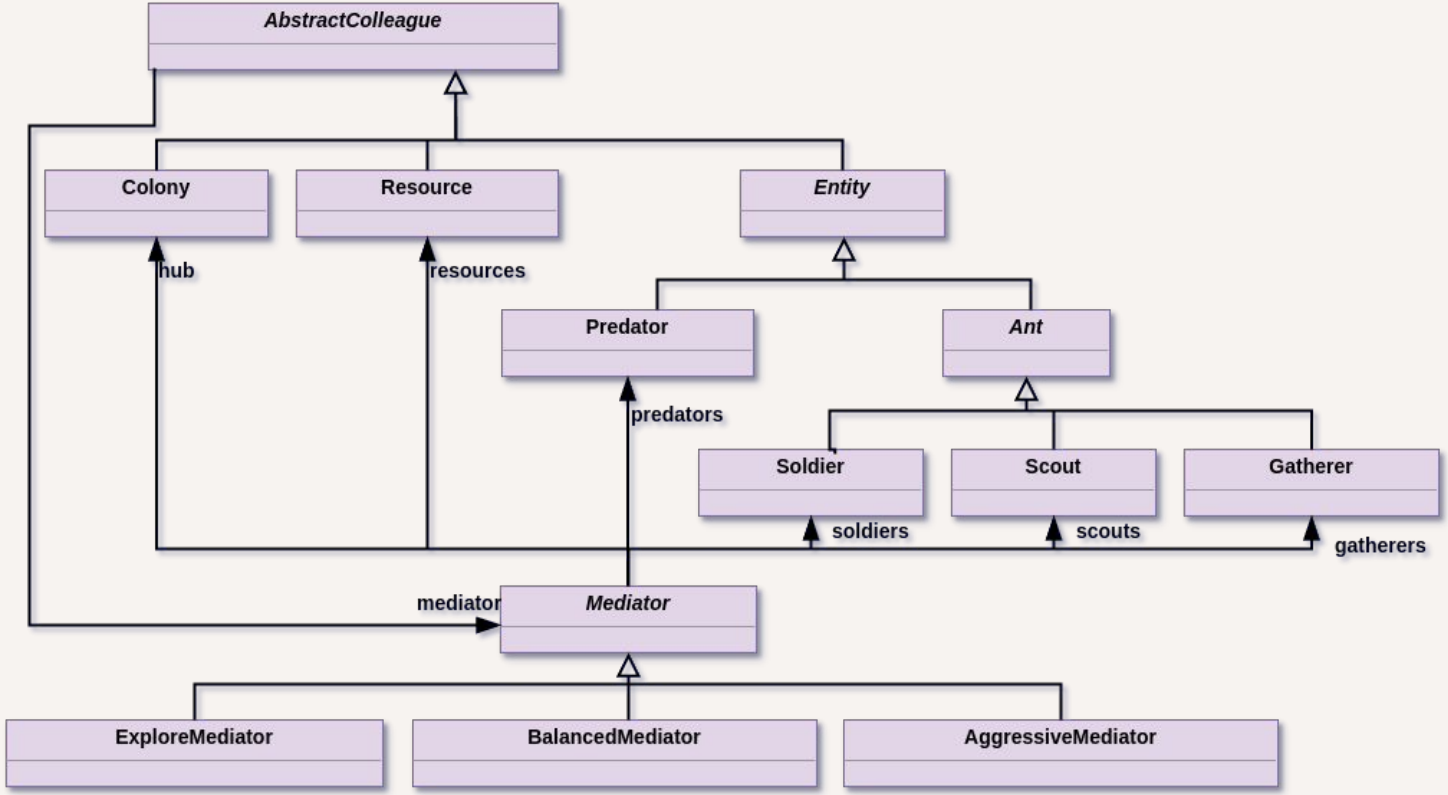
Description du Projet

- Simulation d’une **colonie de fourmis**
- Plusieurs types de **fourmis** avec des **aptitudes** différentes
- Des **prédateurs** peuvent **attaquer** les fourmis
- Les fourmis peuvent **explorer** leur entourage pour découvrir des ressources ou des prédateurs
- En fonction de leur découvertes elle peuvent aller **chercher des ressources**, **attaquer** des prédateurs ou **fuir**
- Un **médiateur** coordonne les interactions entre les différentes entités et leur réactions

Changement Dynamique du Mediator

- Le côté dynamique du Mediator fait référence à la capacité de **changer le médiateur en cours de route**
- Permet de modifier la logique de communication entre les objets **sans** avoir à **modifier le code du Mediator** ni **relancer le programme**
- Au niveau de l'implémentation, nous avons décidé de remonter les Concreate Colleagues dans le **Abstract Mediator** pour éviter d'avoir les mêmes attributs dans plusieurs classes.

Extrait du Diagramme de Classes



Concrete Colleagues

- **Fourmis**:
  - **Scouts**: ont un grand champ de vision
  - **Gatherers**: a peu de force et peu de PVs
  - **Soldiers**: a beaucoup de force et de PVs
- **Predator**: a beaucoup de force et énormément de PVs
- **Resource**: a un nombre de ressources disponibles
- **Colony**: stocke les ressources rapportées

Concrete Mediators

- **Balanced Mediator**: Coordonne les interactions entre collègues dans une situation équilibrée
  - Les scouts cherchent des ressources et des prédateurs
  - Les soldiers se battent
  - Les gatherers vont chercher les ressources et les ramènent à la colonie
  - Les prédateurs attaquent les fourmis
  - Les ressources apparaissent au hasard sur la map
  - La colonie est l'endroit où les fourmis sont à la maison

Concrete Mediators

- **Aggressive Mediator**: Utilisé pour créer un mode agressif de la simulation où les gatherers et les soldats vont se battre contre les prédateurs découverts. Les prédateurs spawn fréquemment et on ne s'occupe pas de la récolte des ressources.
- **Explore Mediator**: Utilisé pour représenter un mode de simulation abondant où les ressources spawn très fréquemment et toutes les fourmis scout. De nouvelles fourmis sont créées lorsque suffisamment de ressources ont été ramenées à la fourmilière. Le type de ces fourmis dépend de la situation.

Stratégies

- Abstraction pour représenter le comportement qui doit être appliqué à une certaine entité
- **Structure** :
  - Interface fonctionnelle qui fournit la méthode execute
  - Implémentation de l'interface selon le comportement désiré dans une classe
  - Une entité en est l'unique attribut
- **Mediator** sera l'objet décidant de l'application d'une certaine stratégie à une entité
- Appel de la méthode execute depuis **Mediator**

Extrait de Code

Dans la classe Gatherer:

```java
@Override
public void update() {
    getMediator().handleUpdateGatherer(this);
}
```

Dans la classe BalancedMediator:

```java
@Override
public void handleUpdateGatherer(Gatherer gatherer) {
    if (gatherer.isFleeing()) {
        strategies.put(gatherer, new FleeingStrategy(gatherer));
    } else if (objectives.get(gatherer) != null) {
        if (objectives.get(gatherer) == hub) {
            strategies.put(gatherer, new ObjectiveHomeStrategy(gatherer));
        } else {
            strategies.put(gatherer, new ObjectiveResourceStrategy(gatherer));
        }
    } else {
        strategies.put(gatherer, new IdleStrategy());
    }
    strategies.get(gatherer).executeStrategy();
}
```

Extrait de Code

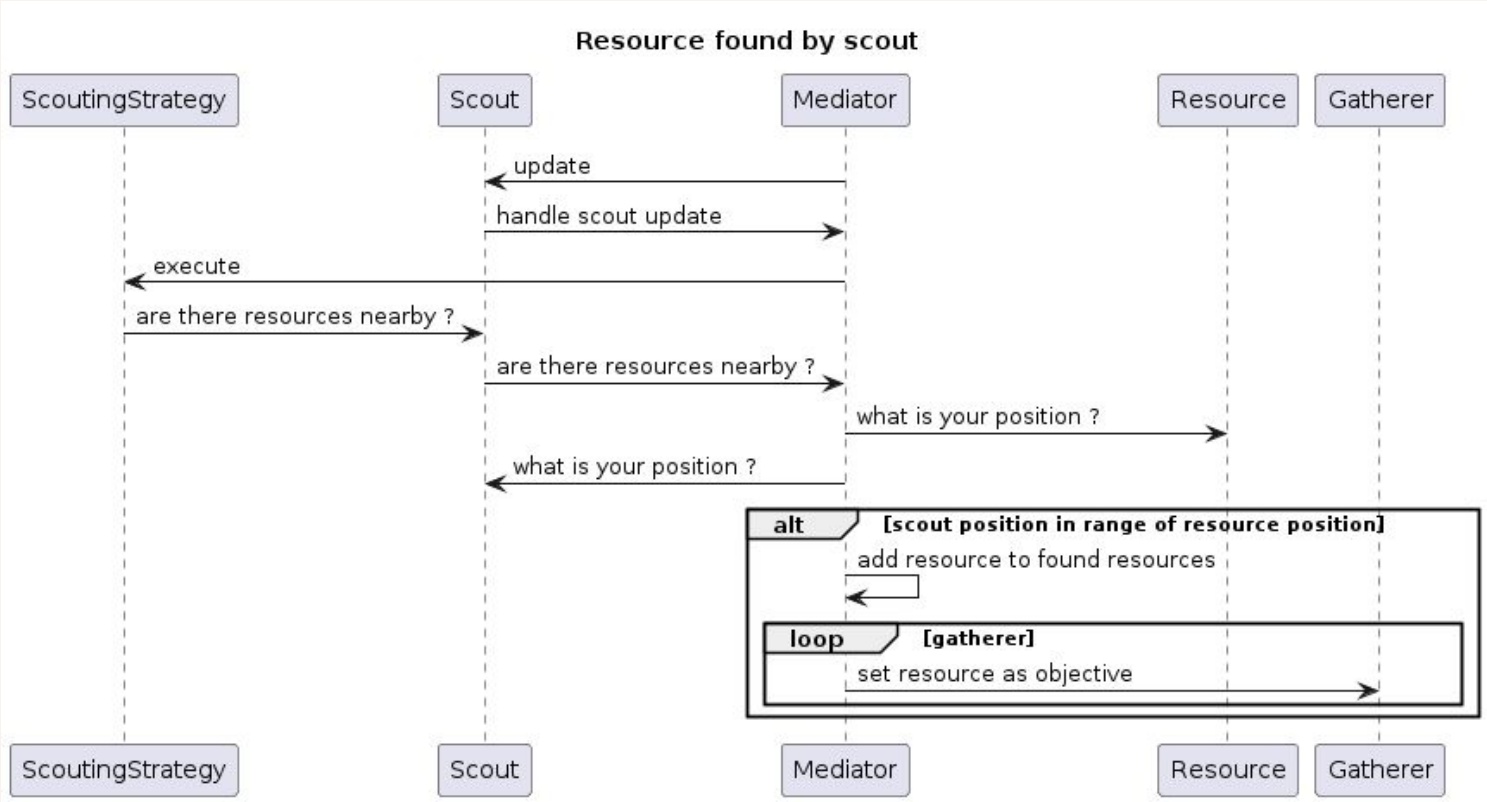
Dans la classe Ant:

```
public void areThereResourcesNearby() {
    getMediator().checkForNearbyResources( scout: this);
}
```

Dans la classe Mediator:

```
public void checkForNearbyResources(Ant scout) {
    Vector2 scoutPosition = scout.getPosition();
    for (Resource resource : resources) {
        double distance = scoutPosition.dist(resource.getPosition());
        if (distance <= getRadiusResource()
                + ScoutRenderer.getInstance().getSize() / 2.0 + scout.getVisionRange()) {
            scout.hasFoundResource(resource);
        }
    }
}
```

Exemple de Communication



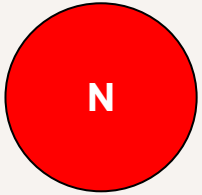
Demo

Fourmi Scout (S)

Fourmi Gatherer (G)

Fourmi Soldier (F)

Ressource
N : nb ressources



Colonie
N : nb ressources

Prédateur
N : nb PVs