

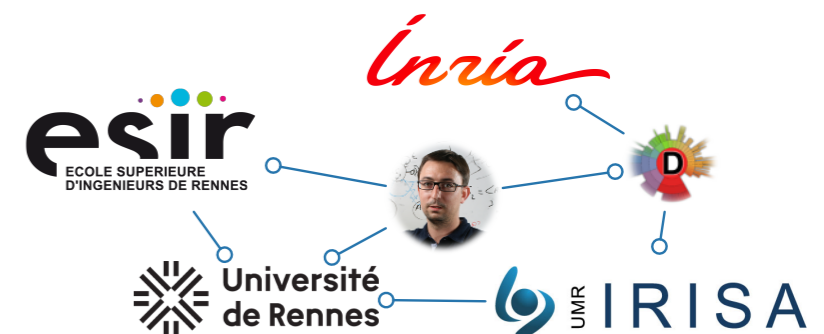
VALIDATION & VERIFICATION

INTEGRATION TESTING

UNIVERSITY OF RENNES, ESIR, 2023-2024

BENOIT COMBEMALE
FULL PROFESSOR, UNIVERSITY OF RENNES, FRANCE

[HTTP://COMBEMALE.FR](http://combemale.fr)
[BENOIT.COMBEMALE@IRISA.FR](mailto:benoit.combemale@irisa.fr)
[@BCOMBEMALE](https://twitter.com/bcombemale)



Plan

1. Introduction au test d'intégration
2. Mise en œuvre des mock avec EasyMock et Mockito

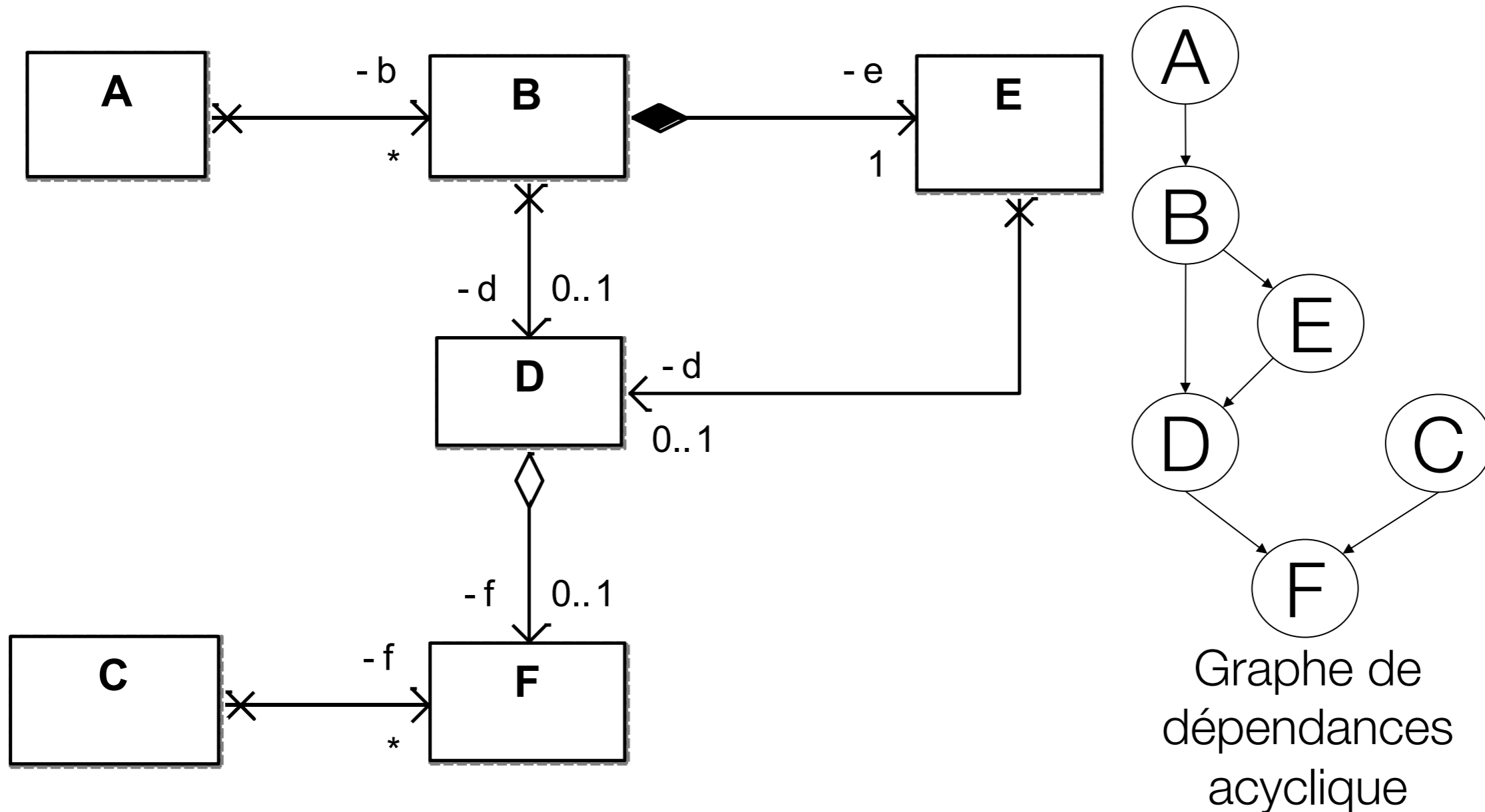
Plan

1. Introduction au test d'intégration
2. Mise en œuvre des mock avec EasyMock et Mockito

Intégration

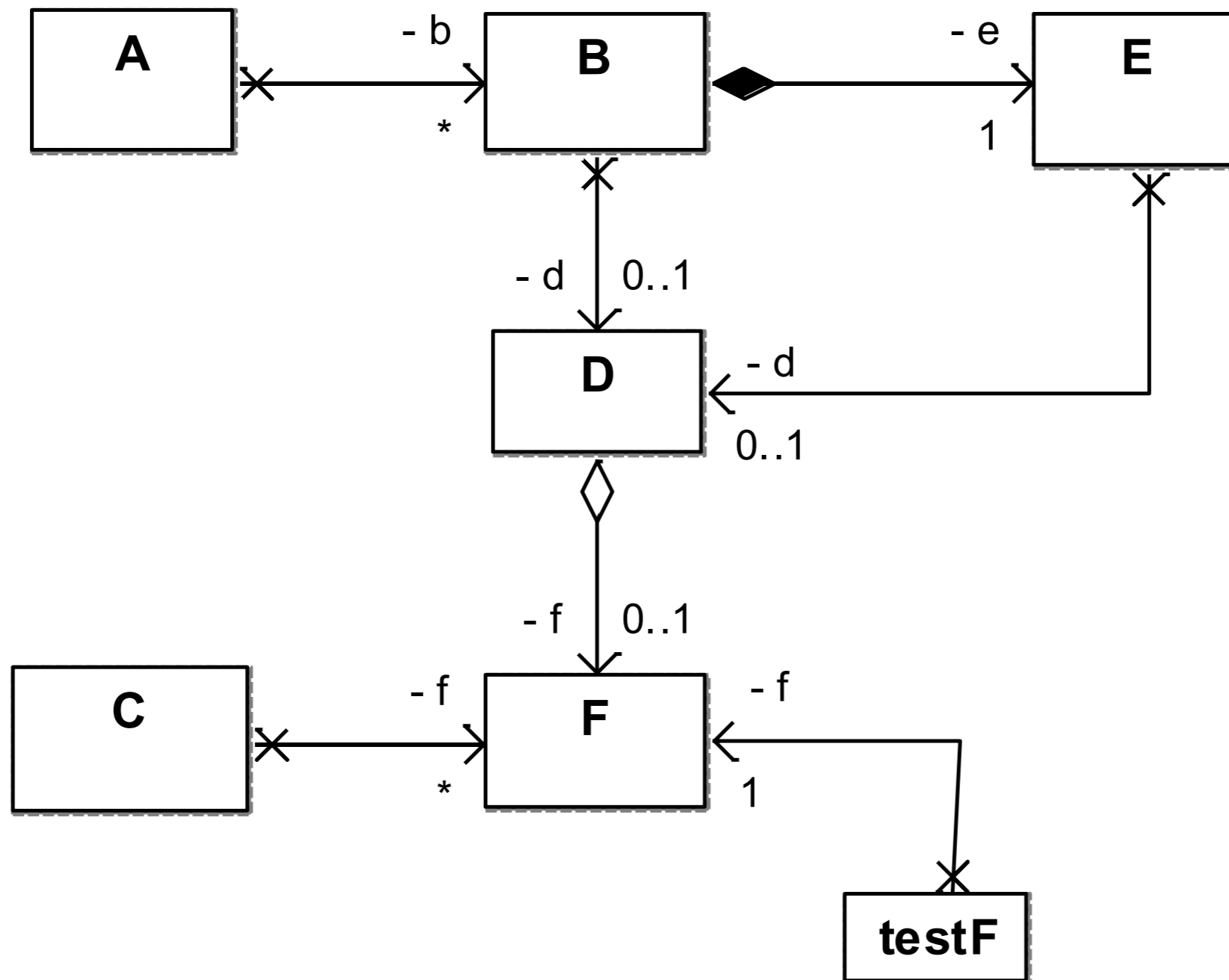
- But : tester les interactions entre classes
- Lien entre test d'intégration et unitaire:
 - il faut ordonner les classes pour le test
- Il faut identifier les dépendances entre classes
 - Problème dans le cas de cycles de dépendances

Cas simple : un graphe acyclique

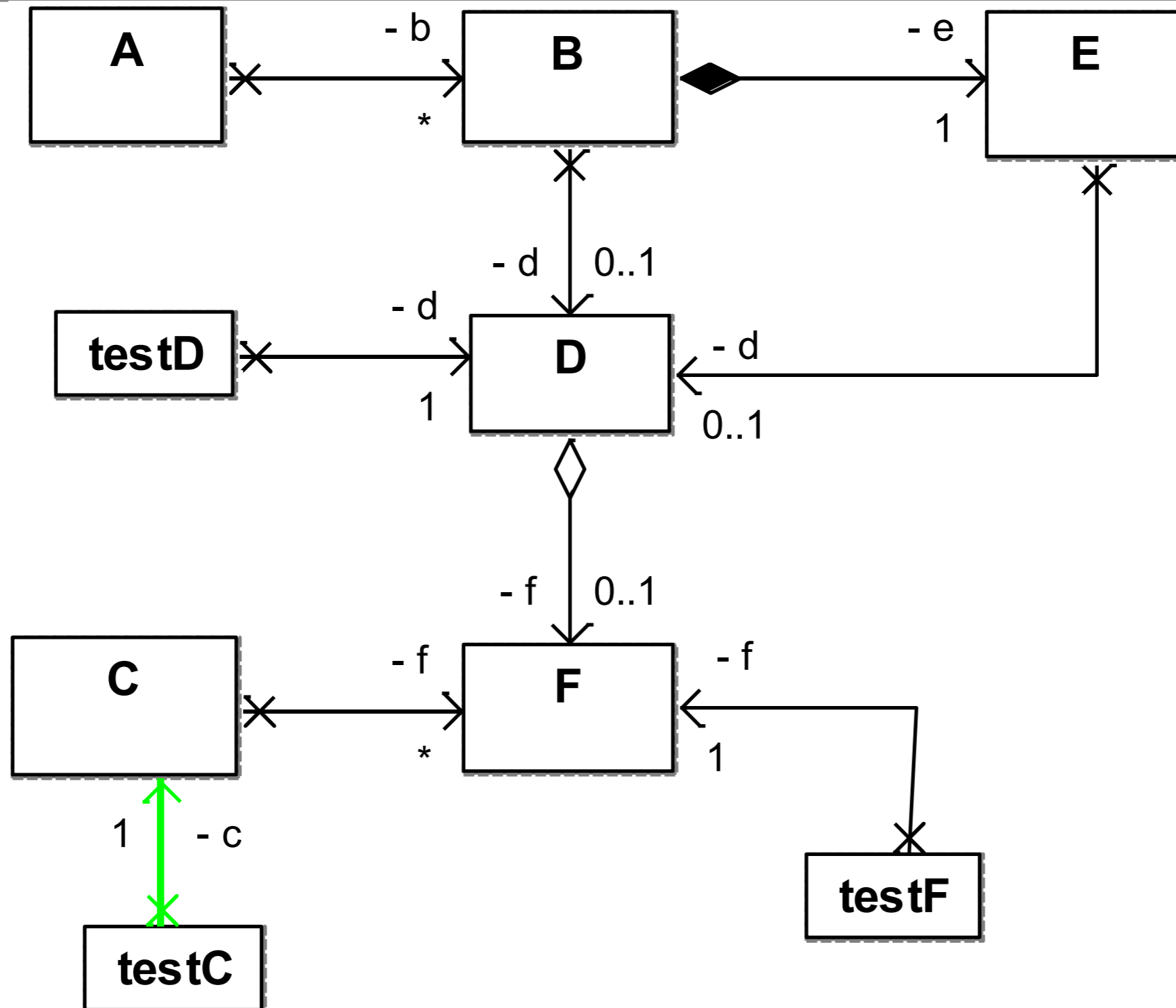


Ordre partiel pour le test: F, (C, D), E, B, A

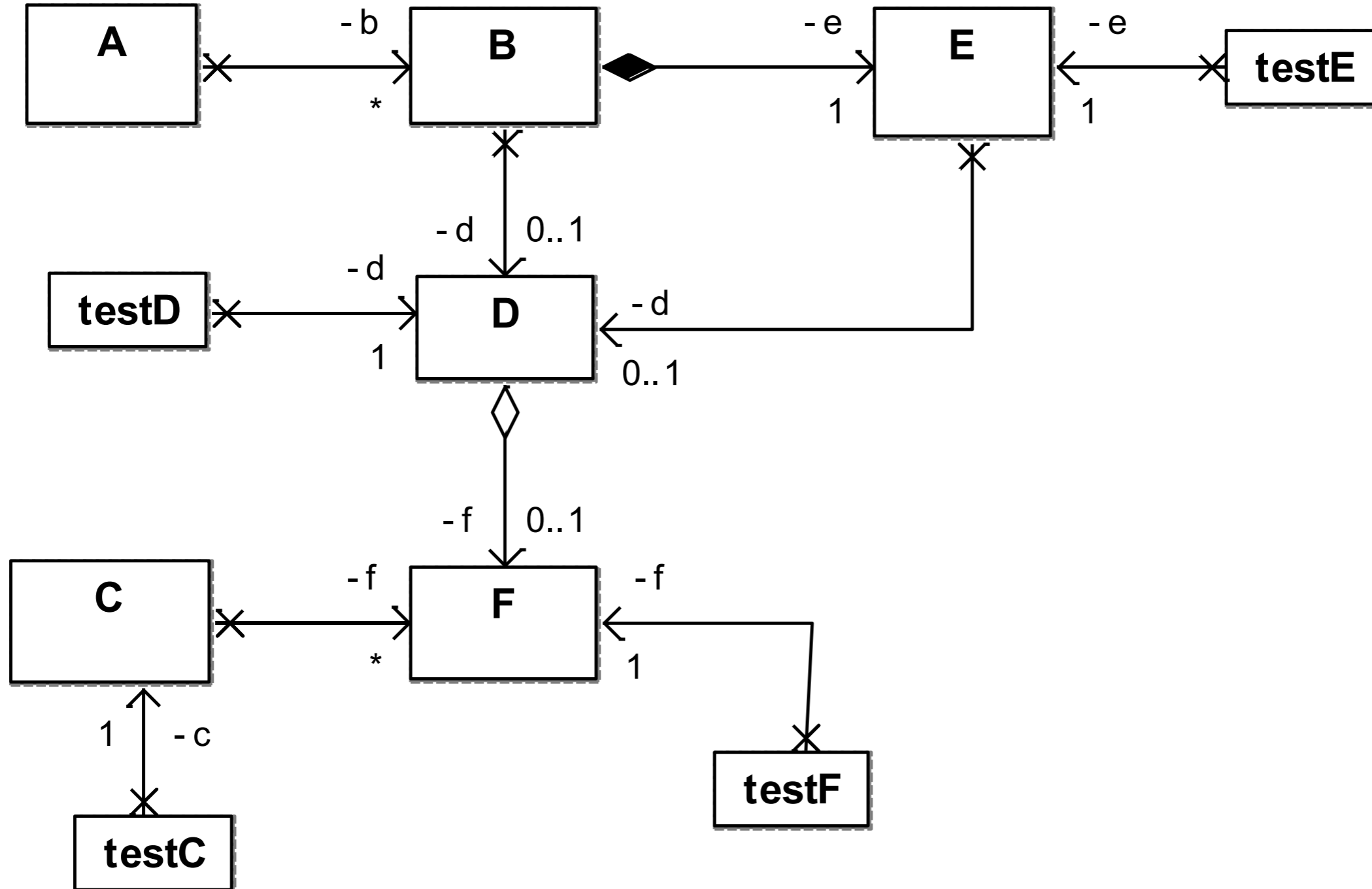
Étape 1



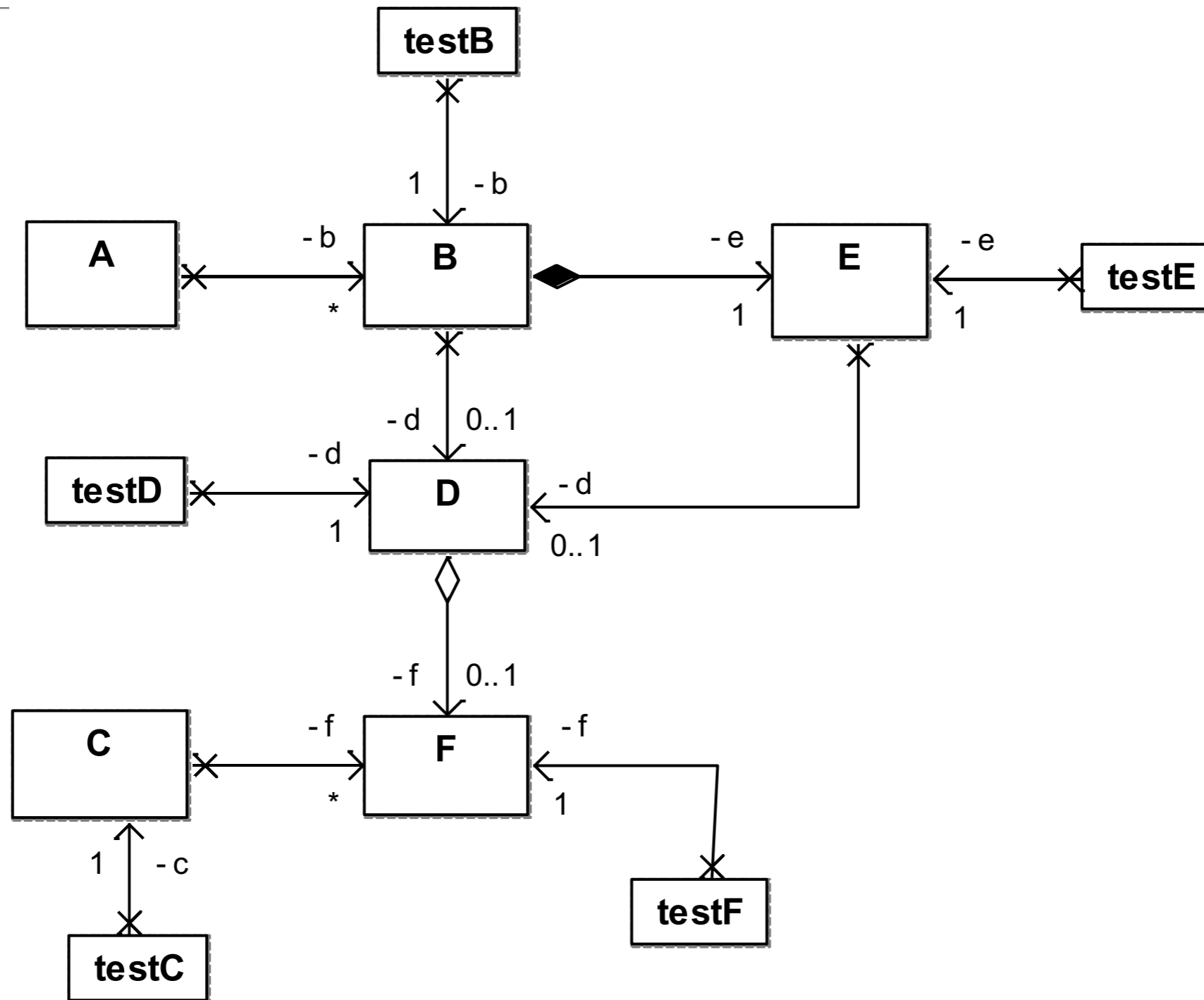
Étape 2



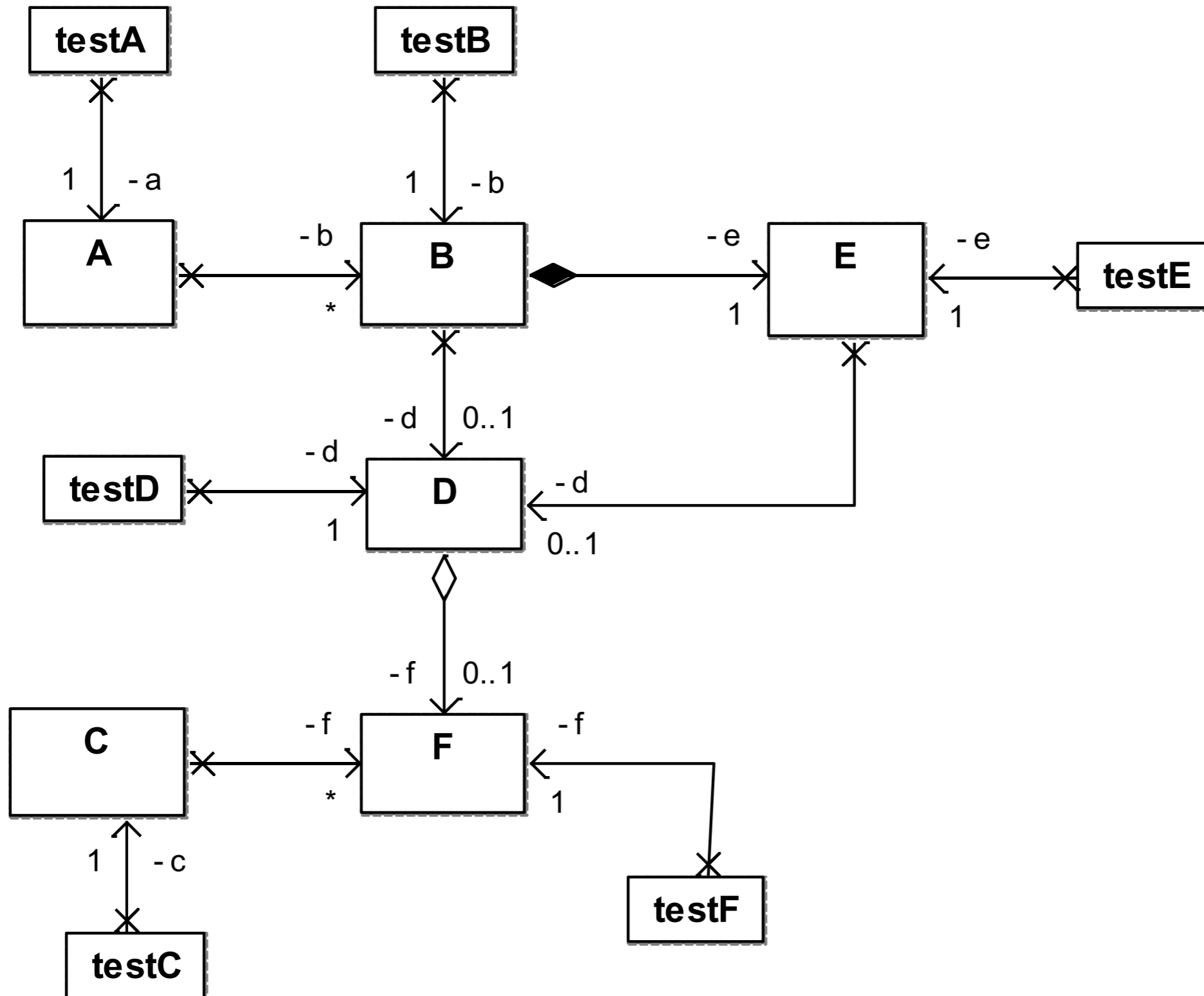
Étape 3



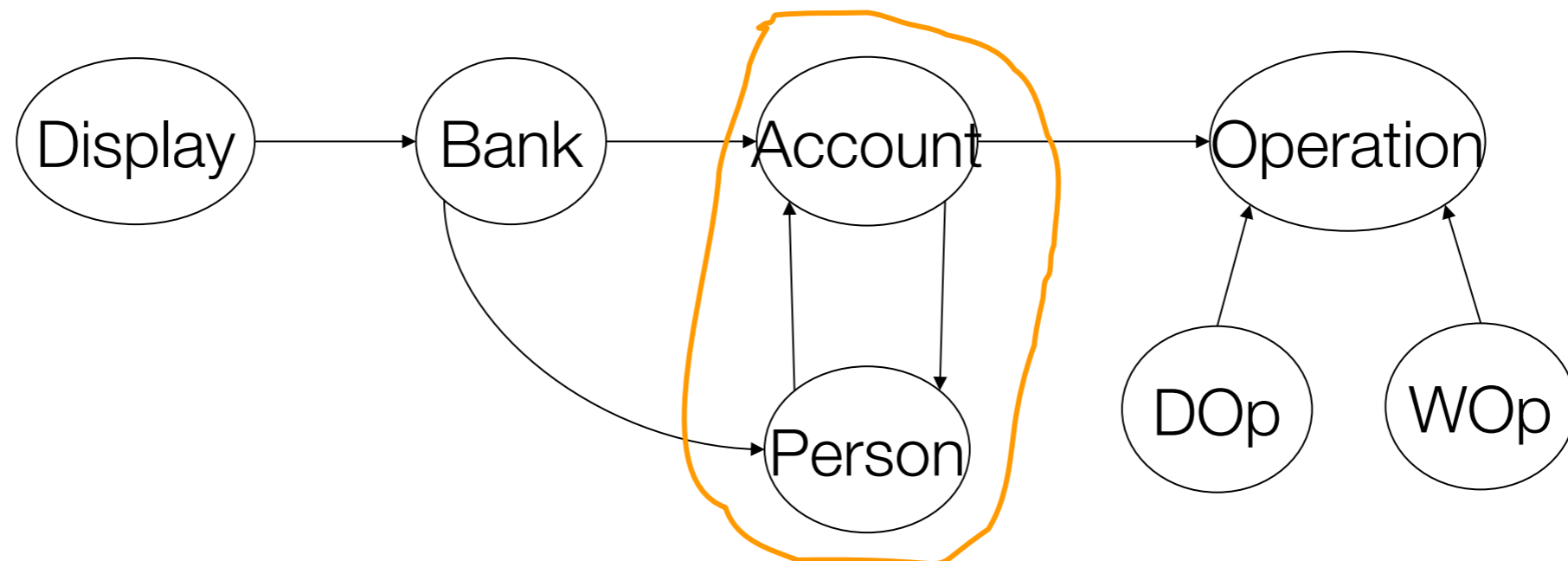
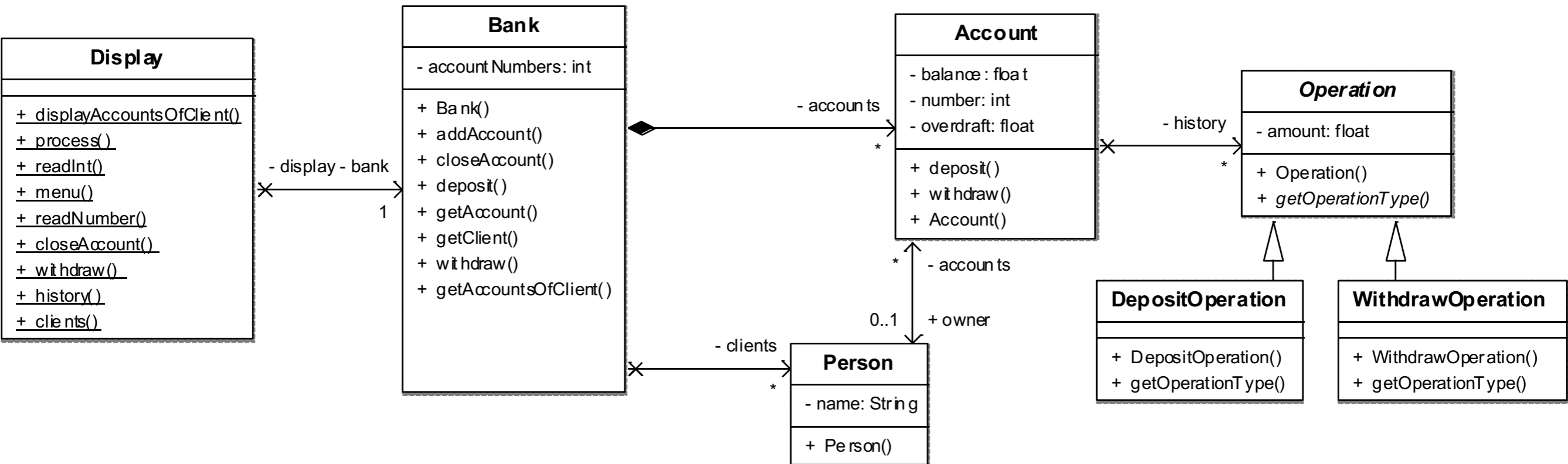
Étape 4



Étape 5



Cas moins simple: présence de cycles



Intégration avec cycles

- Il faut casser les cycles
 - développer des simulateurs de classes (« bouchon de test » ou « stub »)
 - un simulateur a la même interface que la classe simulée, mais a un comportement contrôlé
- Exemple

Exemples de stub

```
/**
 * Creates an account for the person named name
 * If no client has this name, a new client object is created and is added to the list of clients, then the account is created
 * If the client exists the account is created, added to the bank's and the client's list of accounts
 */
public int addAccount(String name, float amount, float overdraft) {
    this.accountNumbers++;
    Person p = getClient(name);
    //if a client named name already exists in the bank's set of clients
    if (p!=null){
        Account a = new Account(p, amount, overdraft, accountNumbers);
        p.addAccounts(a);
        this.addAccounts(a);
    }
    //if the client does not exist, add it tp the bank's list of clients and create account
    else{
        Person client = new Person(name);
        this.addClients(client);
        Account a = new Account(client, amount, overdraft, accountNumbers);
        client.addAccounts(a);
        this.addAccounts(a);
    }
    return accountNumbers;
}
```

Exemples de stub

Stub 1

```
/**
```

```
* Creates an account for the person named name
```

```
* If no client has this name, a new client object is created and is
```

```
* added to the list of clients, then the account is created
```

```
* If the client exists the account is created, added to the bank's and the client's list of accounts
```

```
*/
```

```
public int addAccount(String name, float amount, float overdraft) {
```

```
    return 1;
```

```
}
```

Stub 2

```
/**
```

```
* Creates an account for the person named name
```

```
* If no client has this name, a new client object is created and is
```

```
* added to the list of clients, then the account is created
```

```
* If the client exists the account is created, added to the bank's and the client's list of accounts
```

```
*/
```

```
public int addAccount(String name, float amount, float overdraft) {
```

```
    return 10000000;
```

```
}
```

Exemples de stub

```
/**
 * Looks for a person named name in the set of clients.
 * Returns the Person object corresponding to the client if it exists
 * Returns null if there is no client named name
 */
public Person getClient(String name) {
    Iterator it = this.clientsIterator();
    while (it.hasNext()){
        Person p = (Person)it.next();
        if(p.getName()==name){
            return p;
        }
    }
    return null;
}
```

Exemples de stub

Stub 1

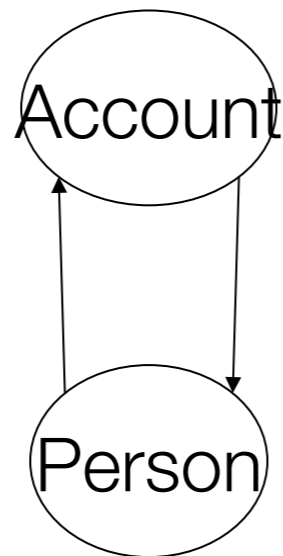
```
/**  
 * Looks for a person named name in the set of clients.  
 * Returns the Person object corresponding to the client if it exists  
 * Returns null if there is no client named name  
 */  
public Person getClient(String name) {  
    return null;  
}
```

Stub 2

```
/**  
 * Looks for a person named name in the set of clients.  
 * Returns the Person object corresponding to the client if it exists  
 * Returns null if there is no client named name  
 */  
public Person getClient(String name) {  
    return new Person("toto");  
}
```


Exemple Banque

- Exemple, pour tester en présence de ce cycle



Regarder quelles sont les méthodes de Person utilisées par Account

```
public class Person {  
    /*  
    * Initializes the name of the person with the param n  
    * Creates a new vector to initialize the accounts set  
    */  
    public Person(String n){  
        name = n;  
        accounts = new Vector(); }  
  
    public String getName(){return name;}  
}
```

Stub de la classe Person

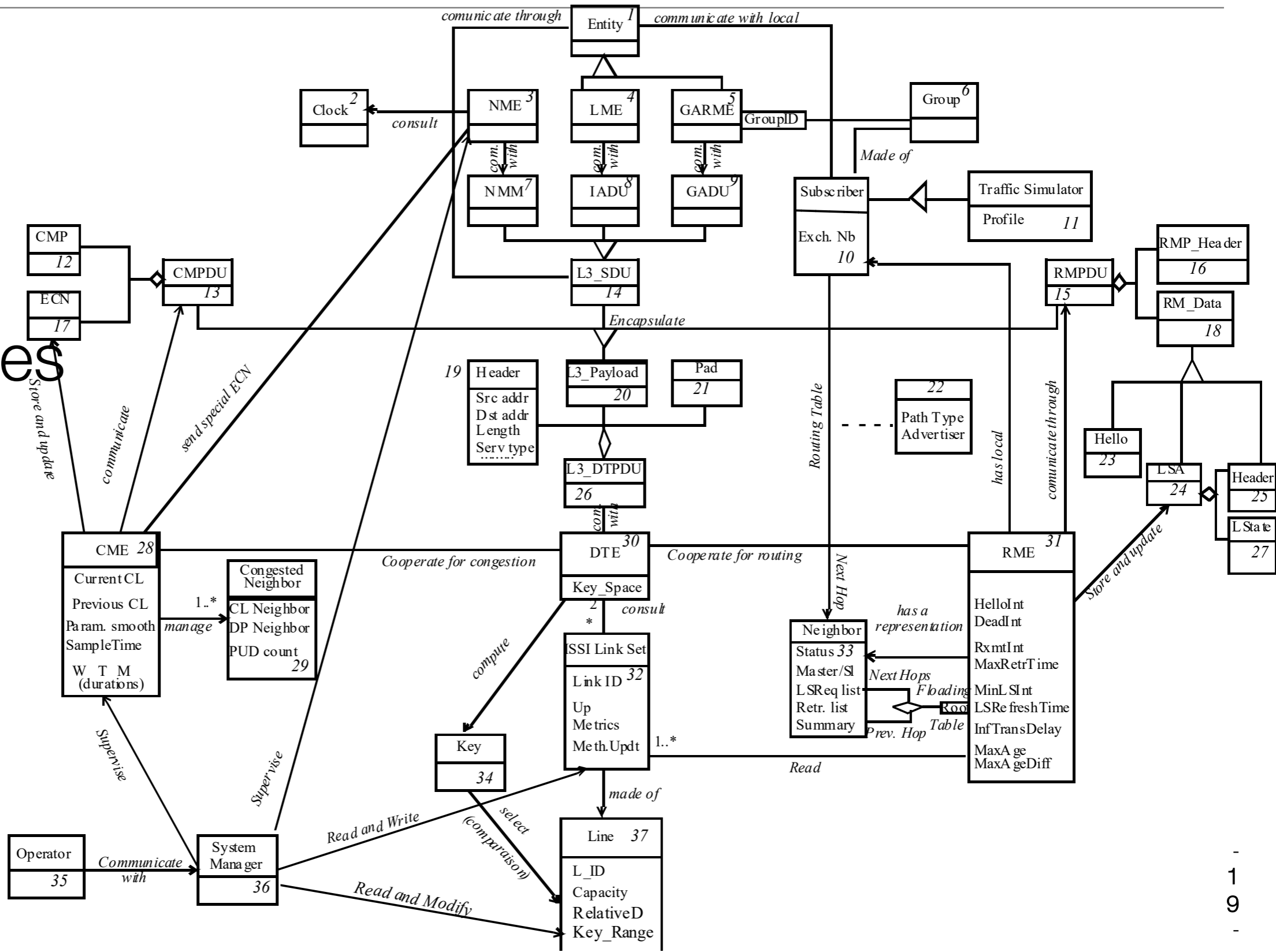
```
public class Person {  
    /*  
    * Initializes the name of the person with the param n  
    * Creates a new vector to initialize the accounts set  
    */  
    public Person(String n){ }  
  
    public String getName(){return ("toto");}  
}
```

Exemple Banque

- Etape 1
 - Tester la classe Account avec le stub de Person
- Etape 2
 - Tester la classe Person avec Account
- Etape 3
 - Retester la classe Account avec la vraie classe Person

Cas encore moins simple

37 classes



Cas encore moins simple

- Contraintes sur la conception
 - pas d'interdépendances
 - contrainte forte dans un cadre OO
- Sans contraintes sur la conception
 - on intègre tout d'un coup: stratégie « big bang »
 - heuristique pour prendre en compte les interdépendances au moment de l'intégration

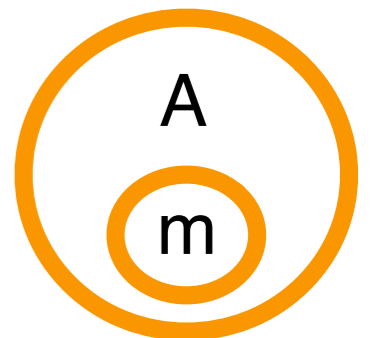
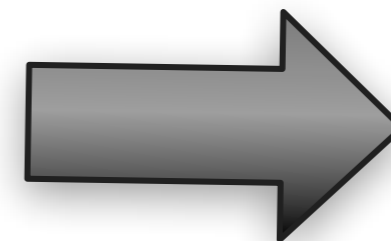
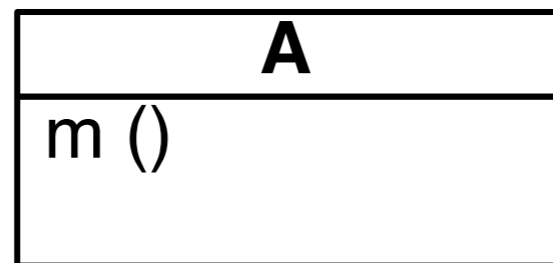
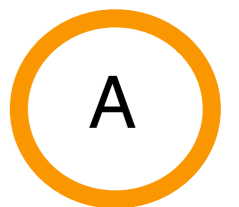
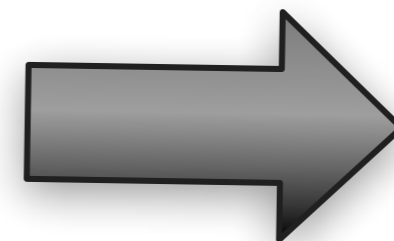
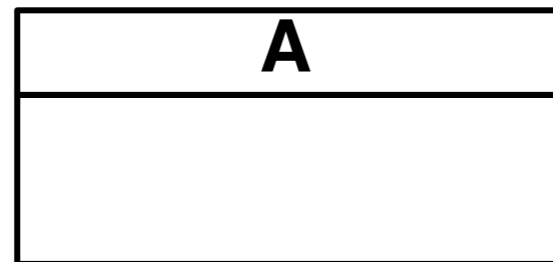
Une stratégie efficace pour l'ordre d'intégration

- Basée sur un modèle de graphe: graphe de dépendances de test (GDT)
- Deux types de dépendances
 - héritage
 - client/serveur
- Dépendances
 - classe – classe
 - méthode – classe

Transformation UML vers GDT

2 types of nodes

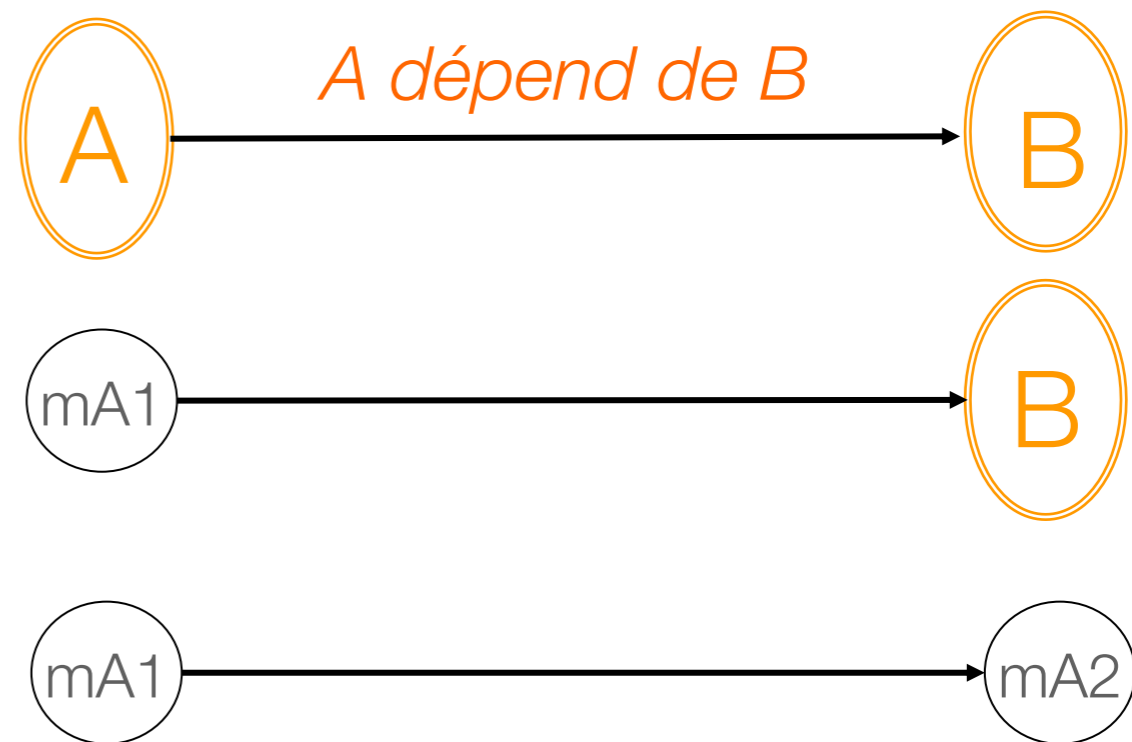
- class
- method



Transformation UML vers GDT

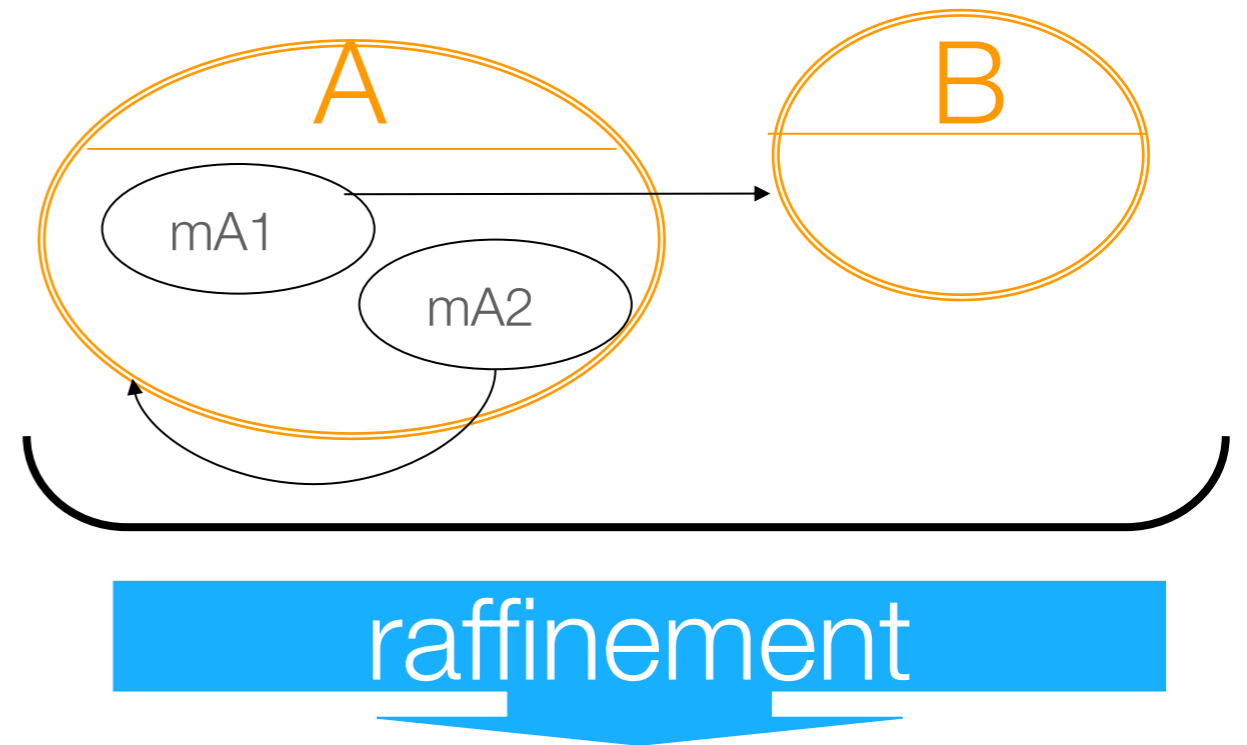
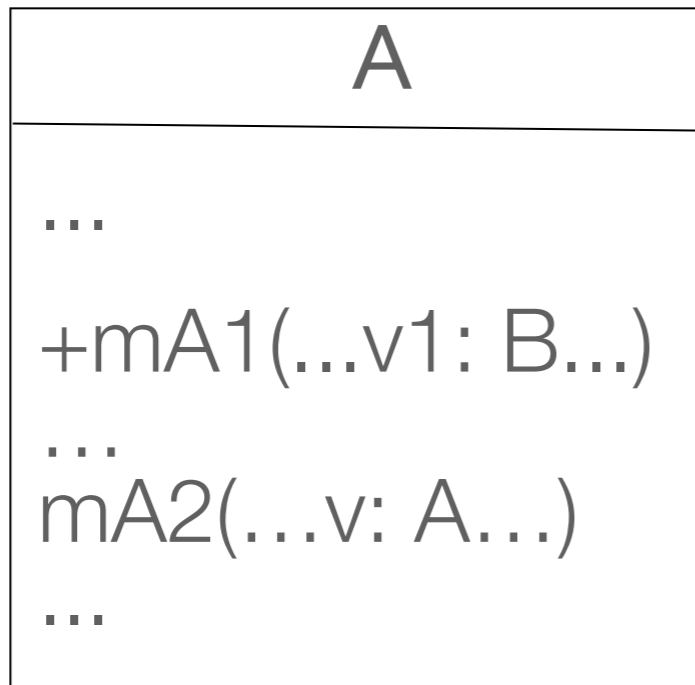
3 types d'arcs

- class_to_class
- method_to_class
- method_to_method

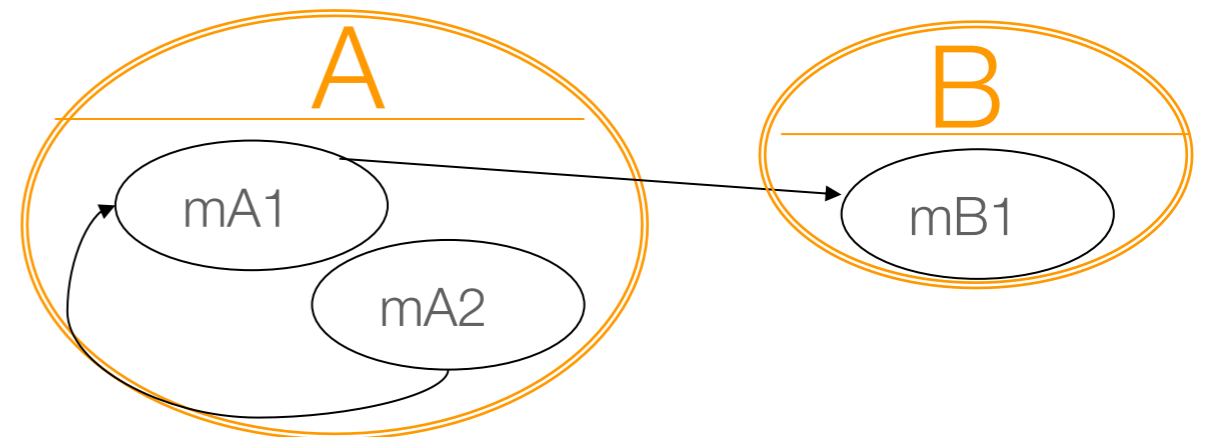
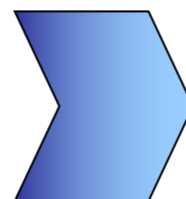
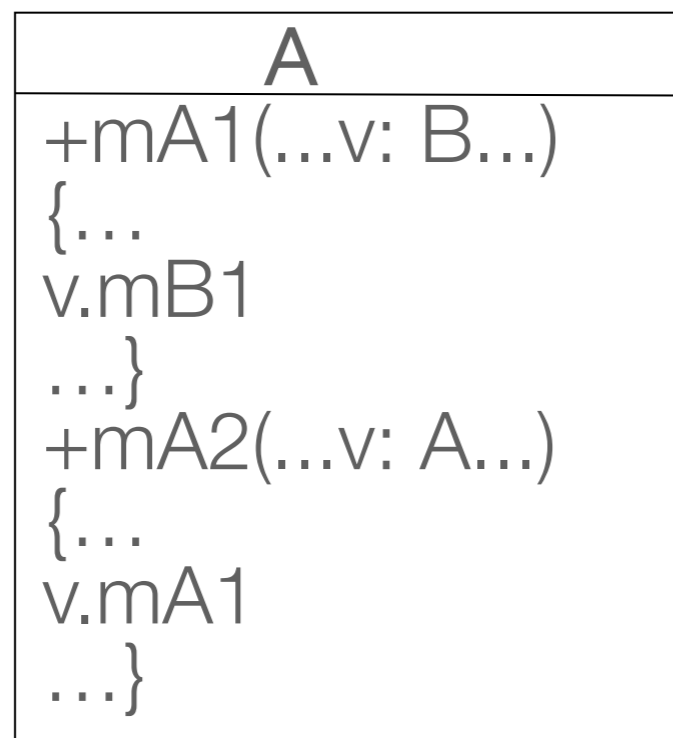


Transformation UML vers GDT

Méthode vers classe

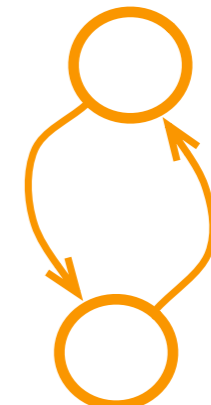
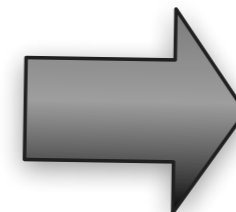
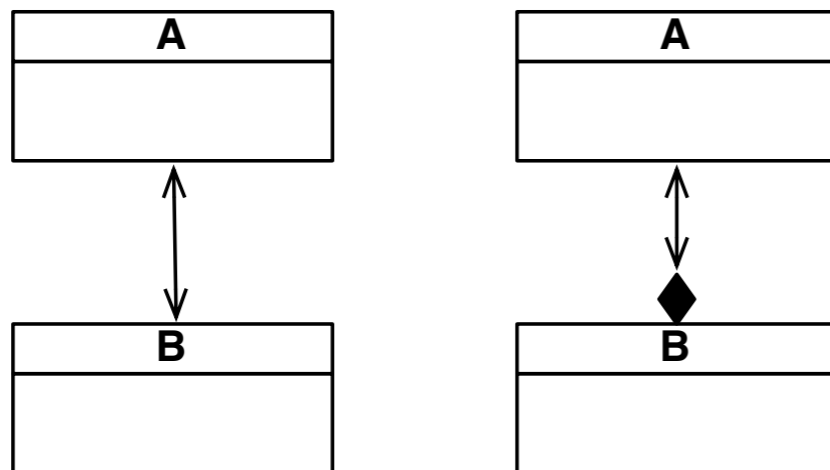
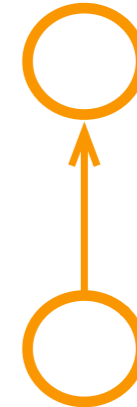
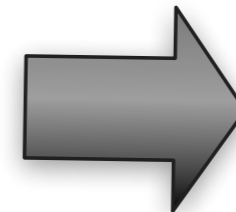
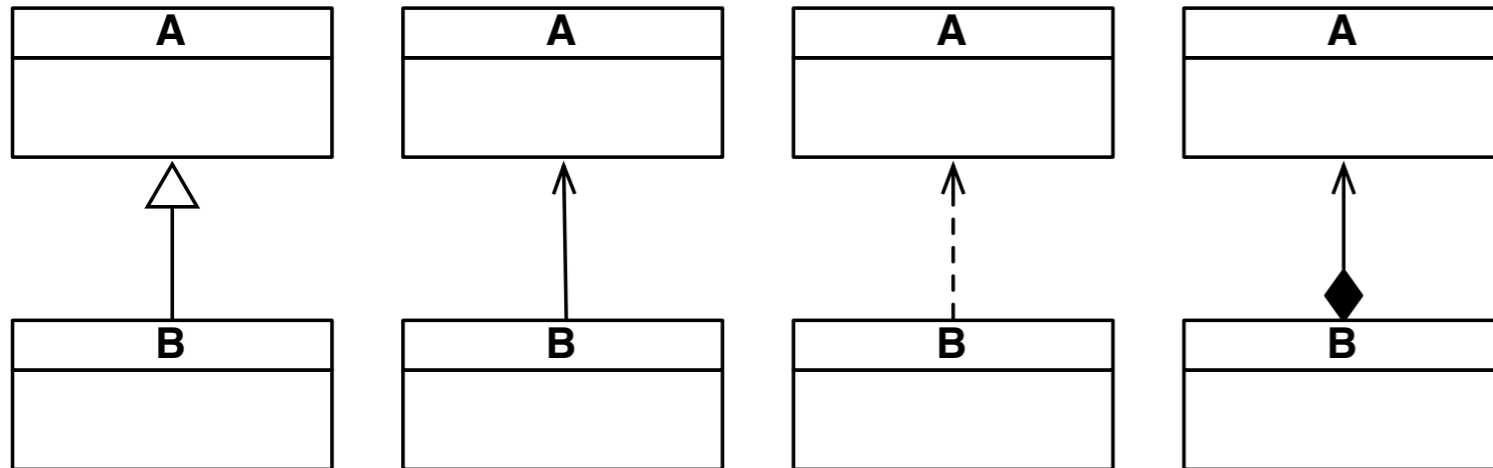
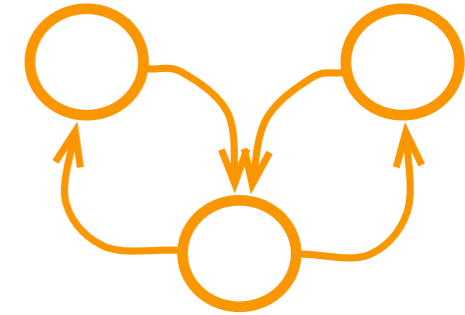
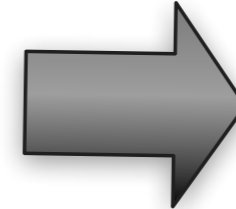
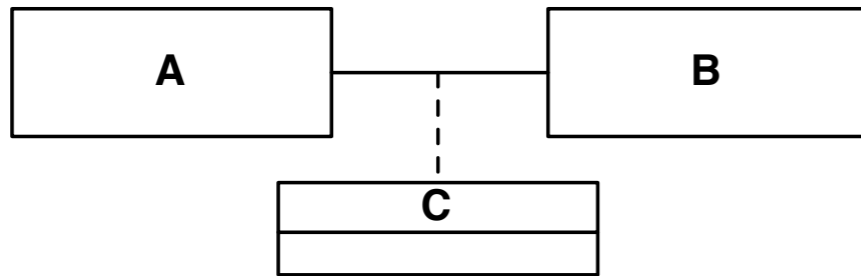


Méthode vers méthode

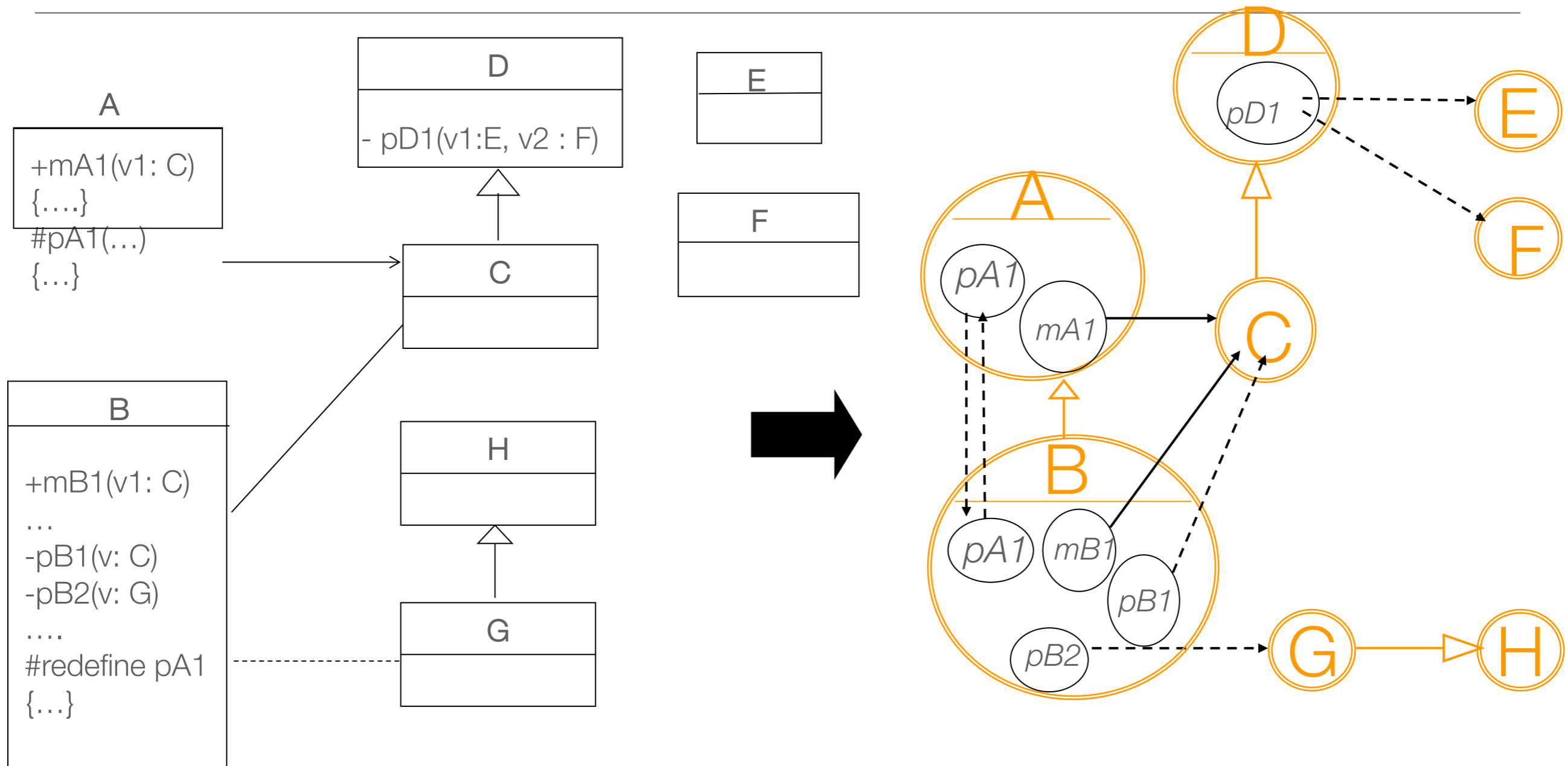


Langage d'action (AS / OCL ...)

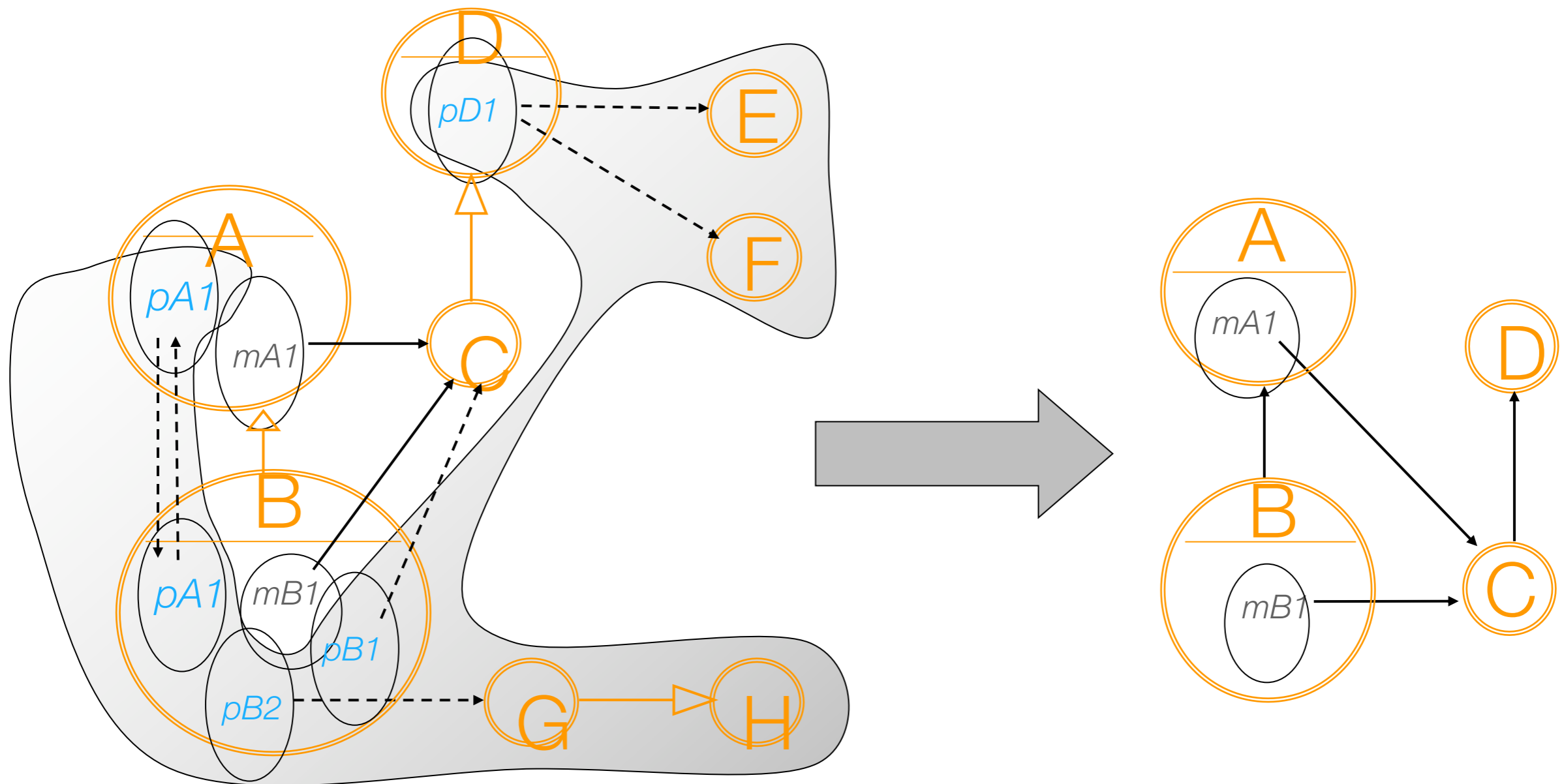
Transformation UML vers GDT



Transformation UML vers GDT



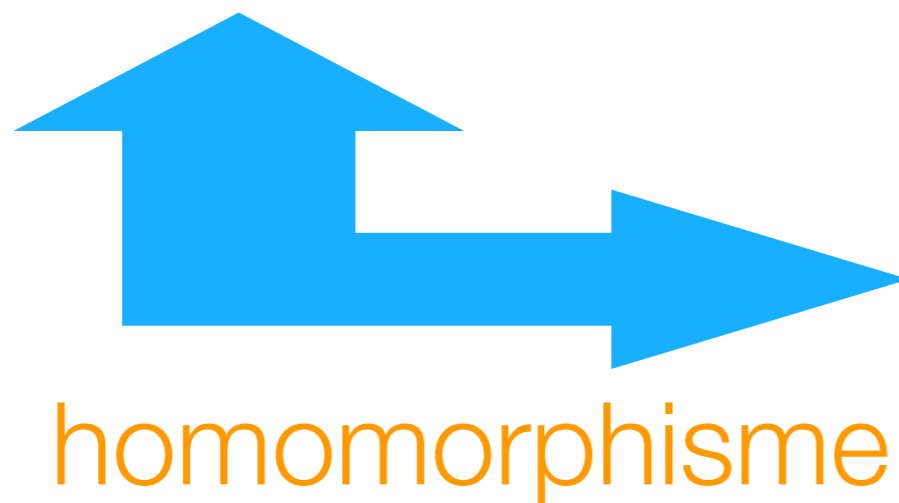
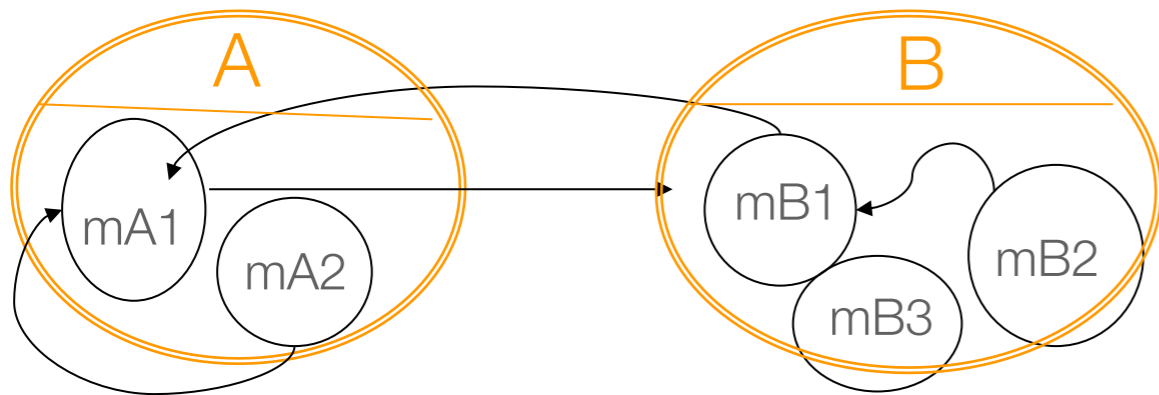
Transformation UML vers GDT



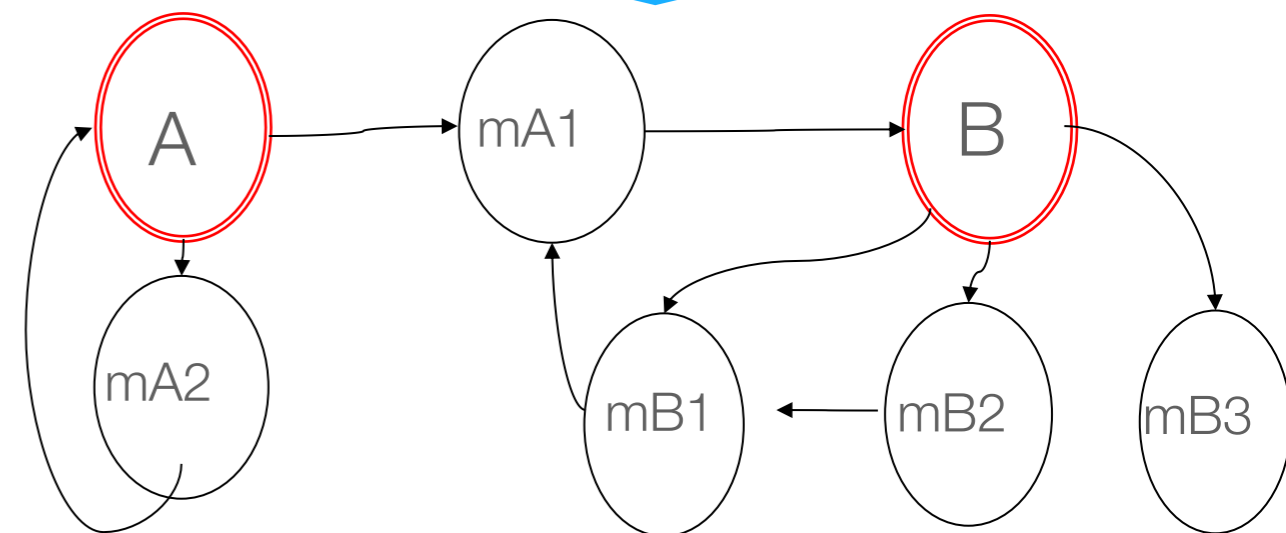
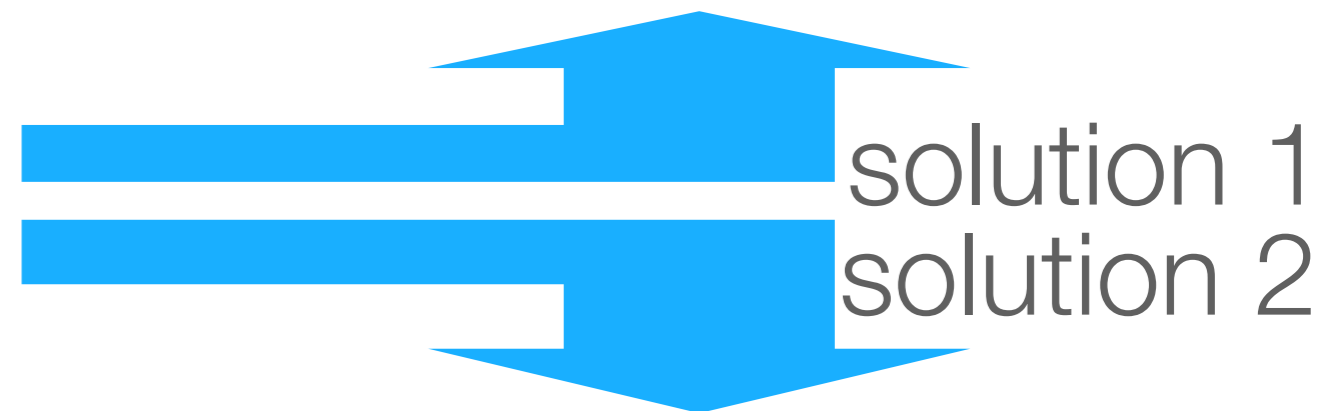
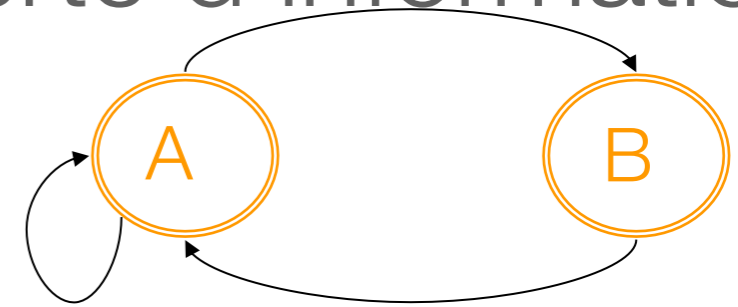
Supprimer les dépendances spécifiques à l'implantation

Transformation UML vers GDT

Pas un graphe classique



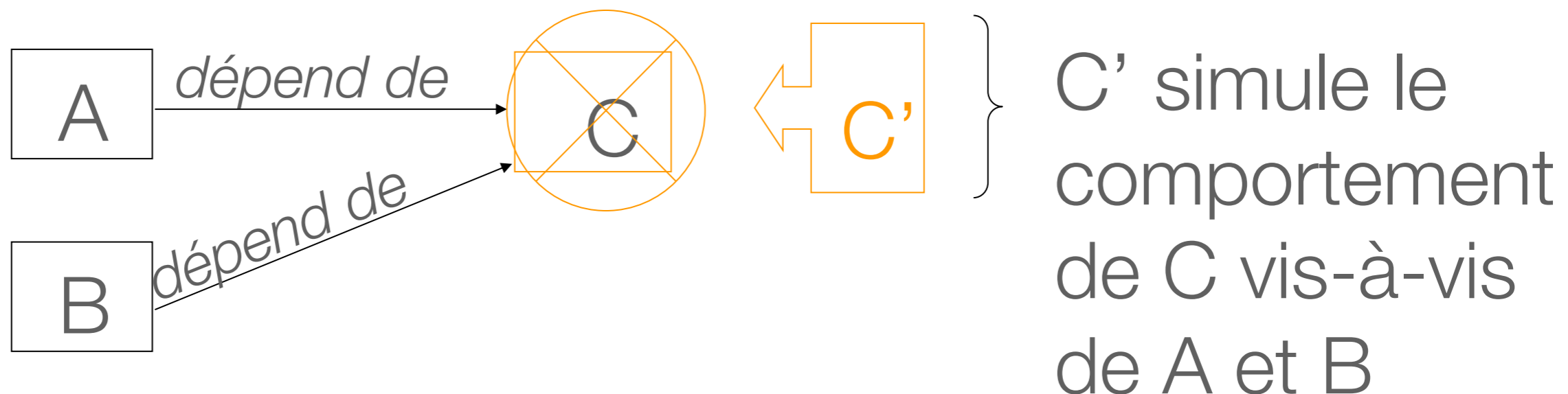
Perte d'information



Pas de perte d'information

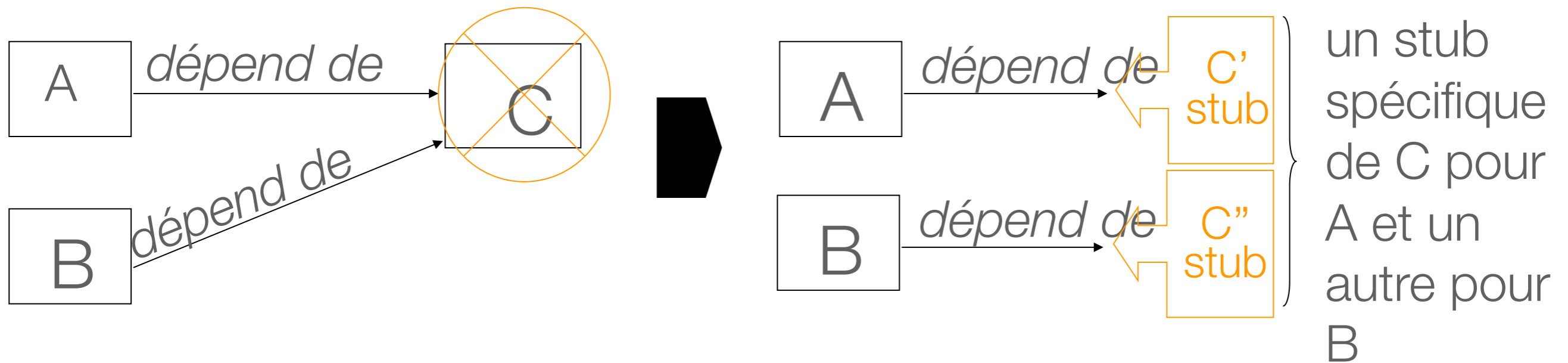
Une stratégie efficace pour l'ordre d'intégration

- Comment choisir un ordre d'intégration à partir du GDT?
 - Minimiser le nombre de stubs à écrire
 - stub réaliste => simule tous les comportements (réutiliser une ancienne version du composant)

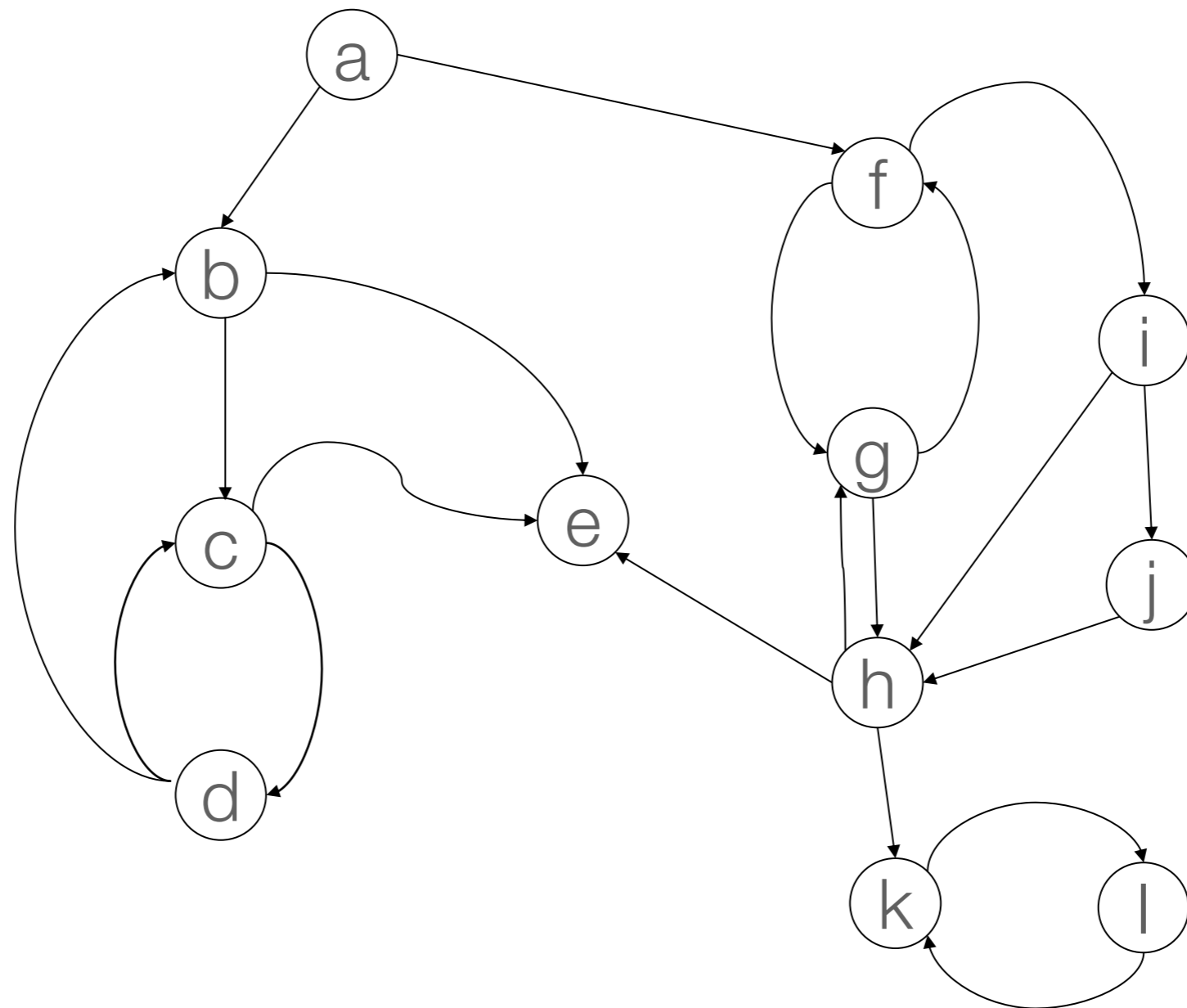


Une stratégie efficace pour l'ordre d'intégration

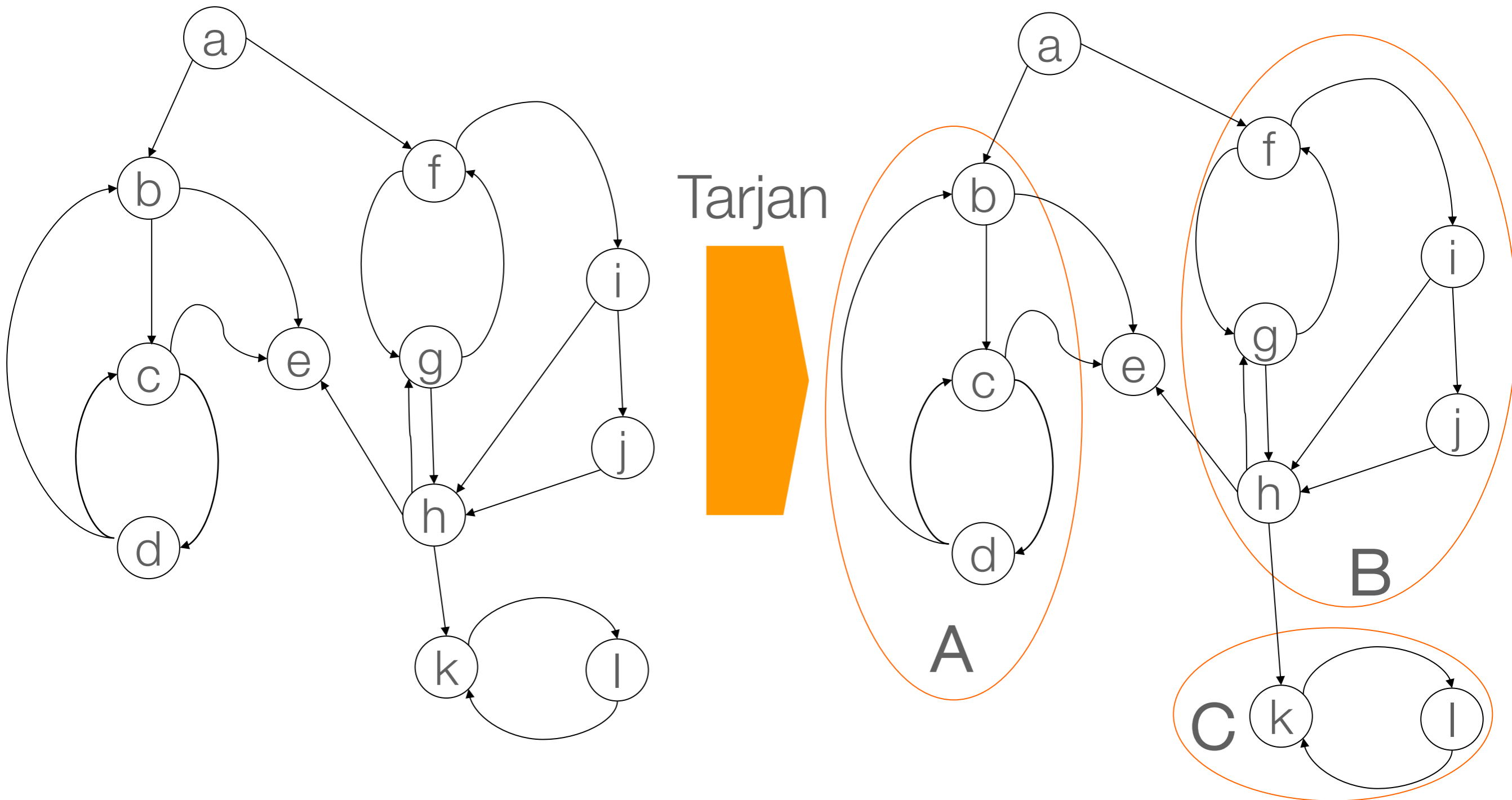
- Stub spécifique => ne simule que les comportements utilisés par le client



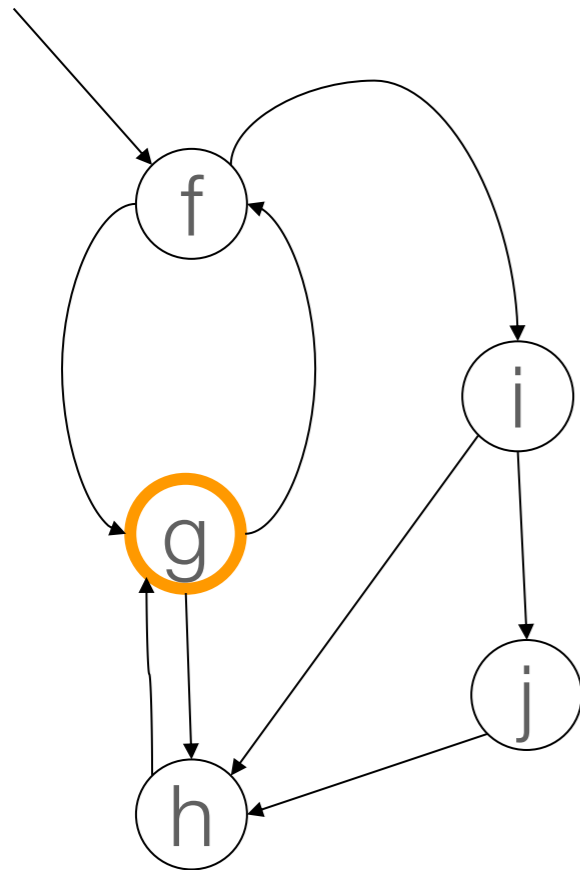
Une stratégie efficace pour l'ordre d'intégration



Une stratégie efficace pour l'ordre d'intégration

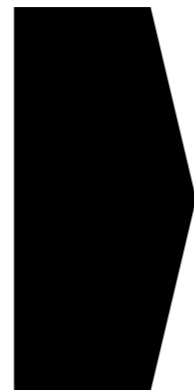


Une stratégie efficace pour l'ordre d'intégration

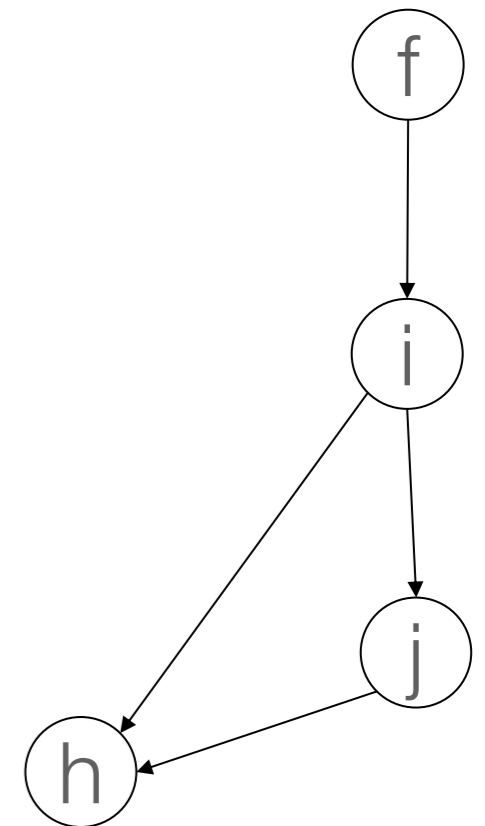


B

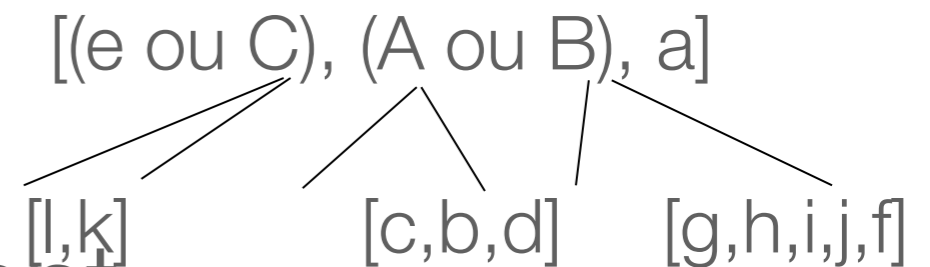
Algorithme de Bourdoncle



Noeud candidat =
max(fronds)

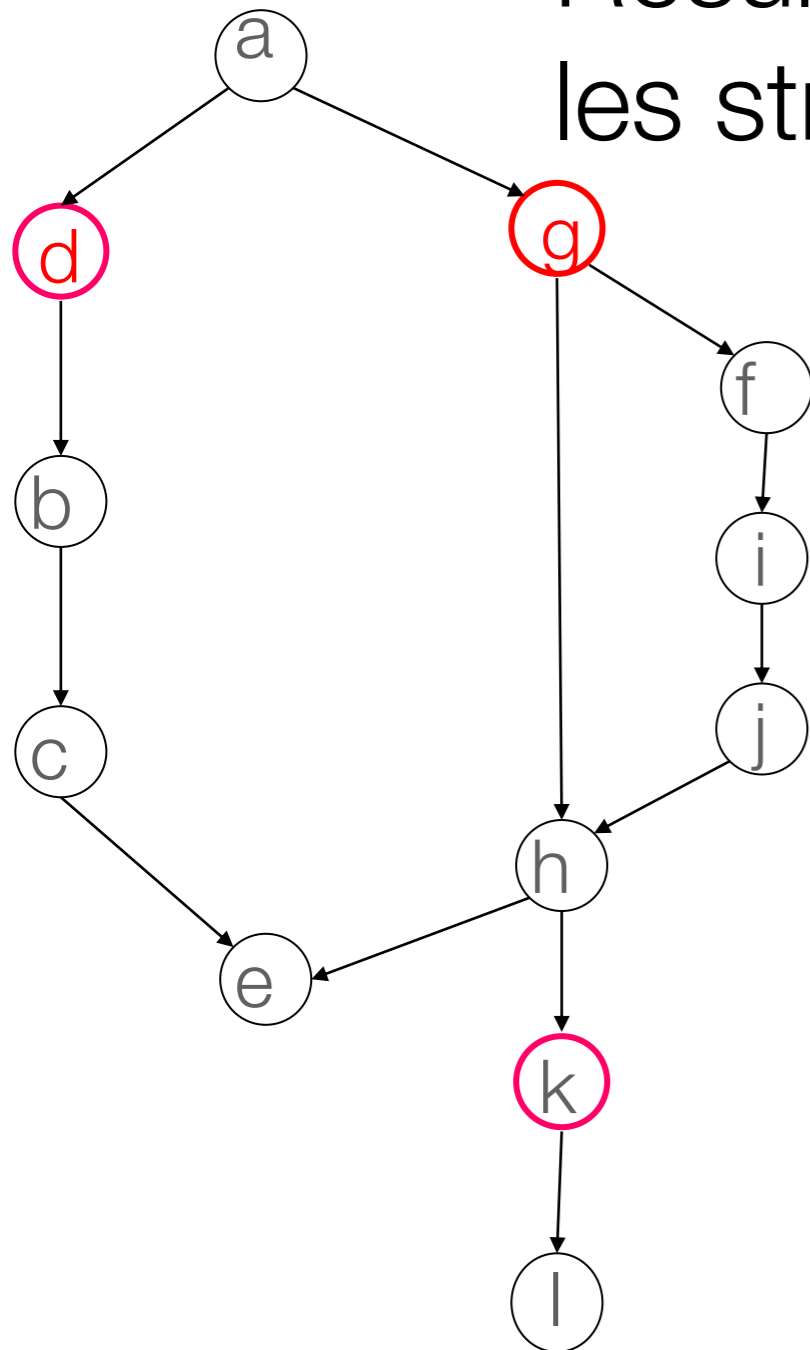


Casse les CFCs
réapplique Tarjan éventuellement



Une stratégie efficace pour l'ordre d'intégration

Résultat = un ordre partiel de toutes les stratégies possibles



Algorithme optimisé

#stubs spécifiques = 4

#stubs réalistes = 3

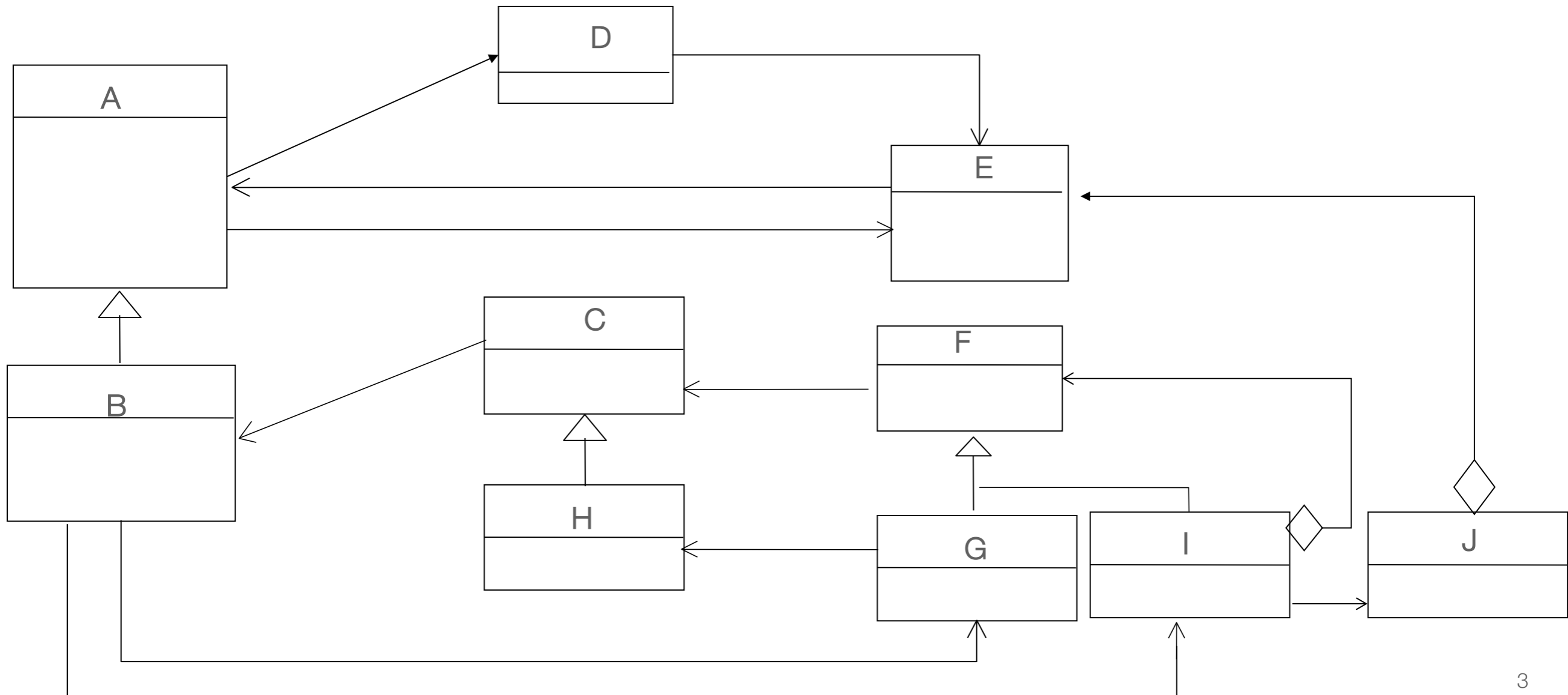
Génération aléatoire (moy.)

#stubs spécifiques = 9.9

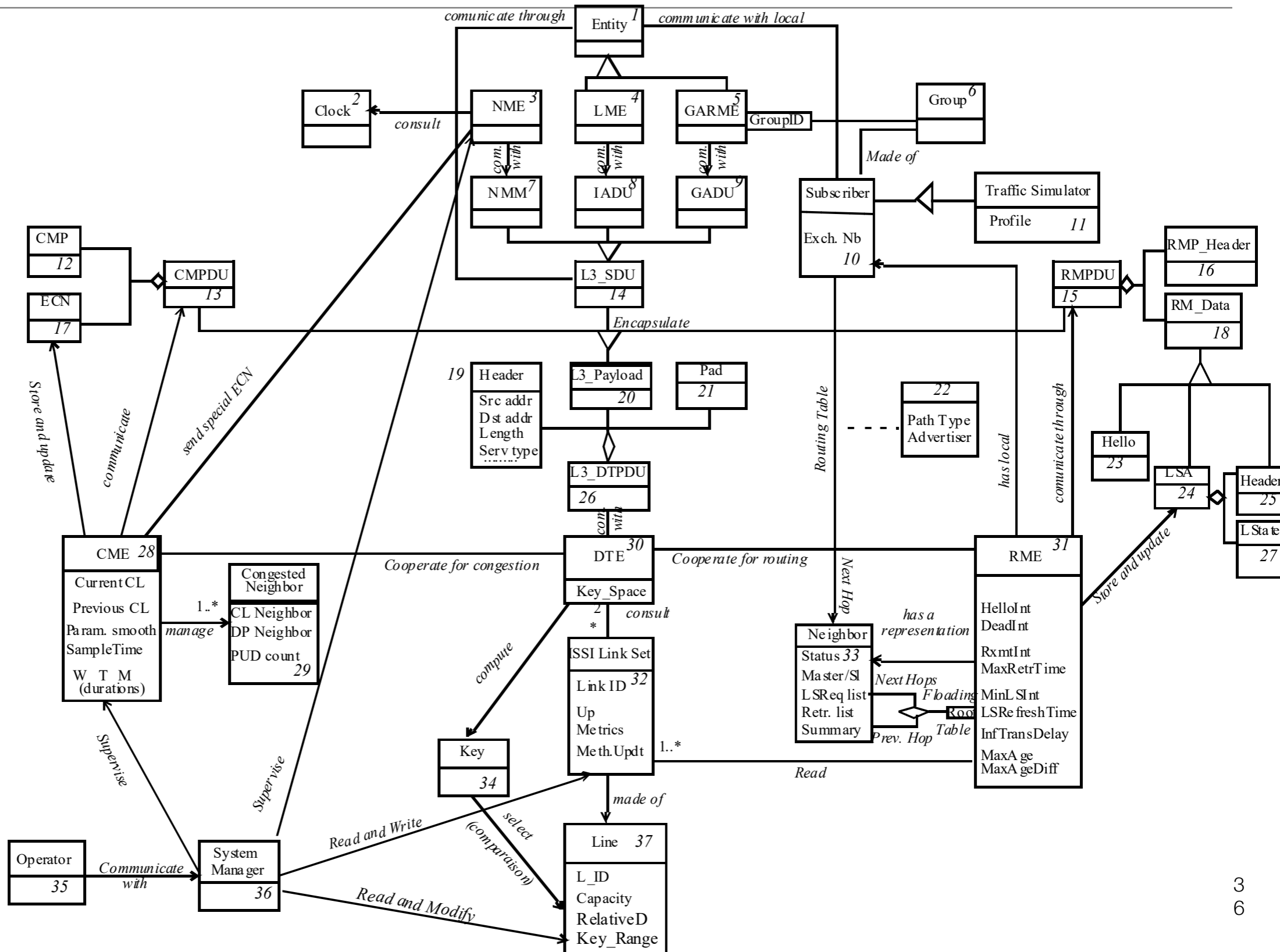
#stubs réalistes = 5

Exo

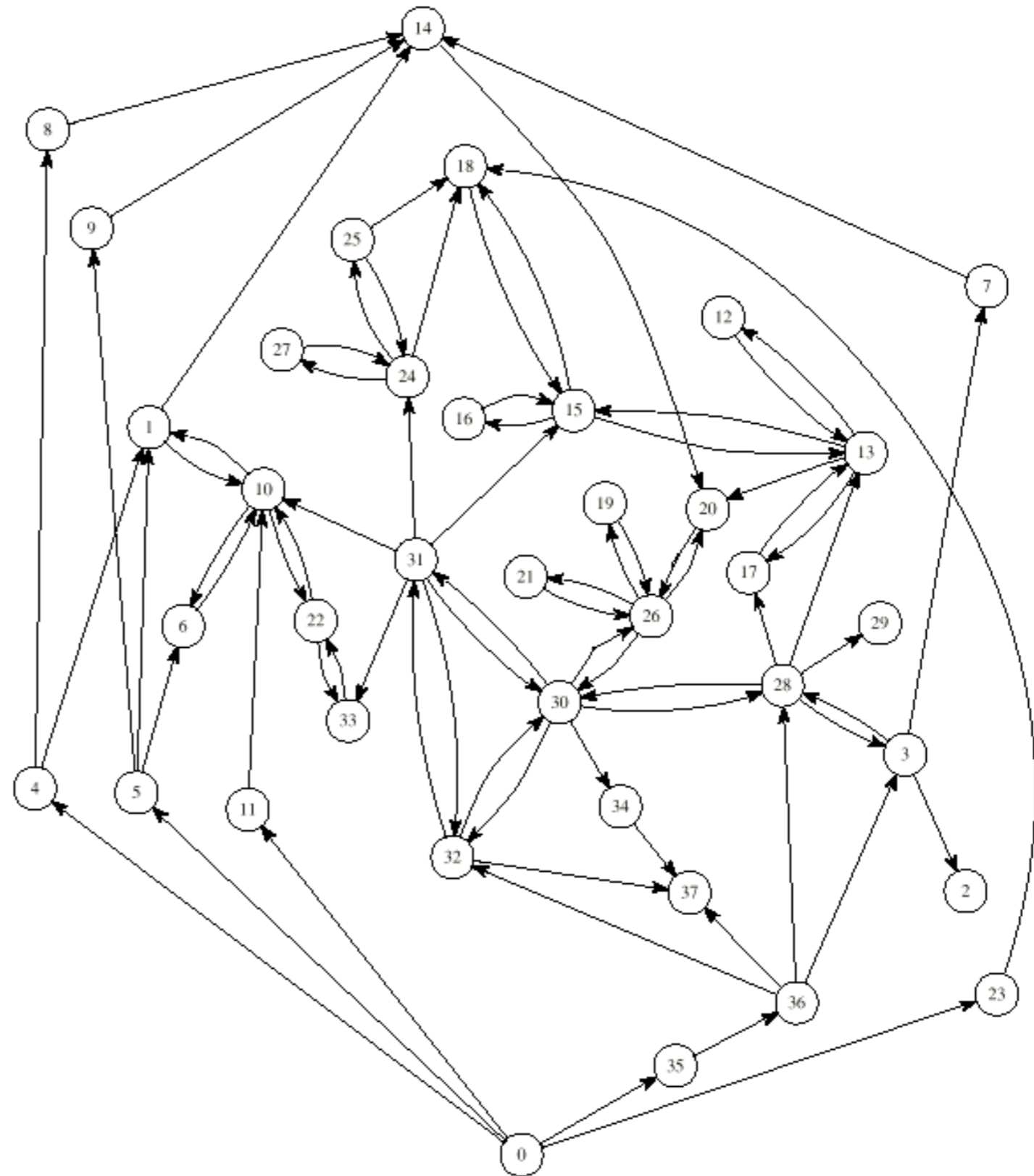
- Plan de test d'intégration pour :



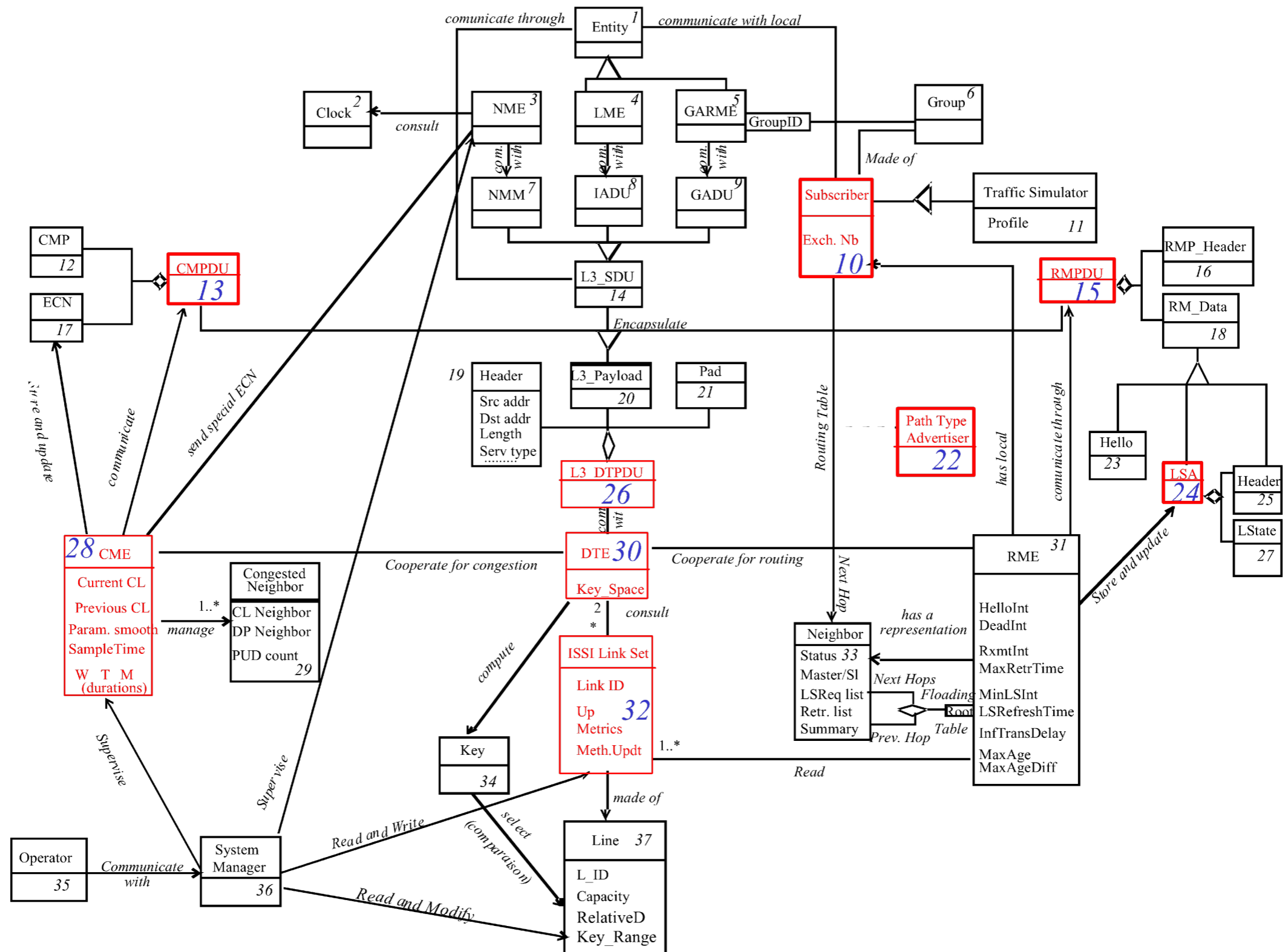
Cas encore moins simple



Graphe de dépendances



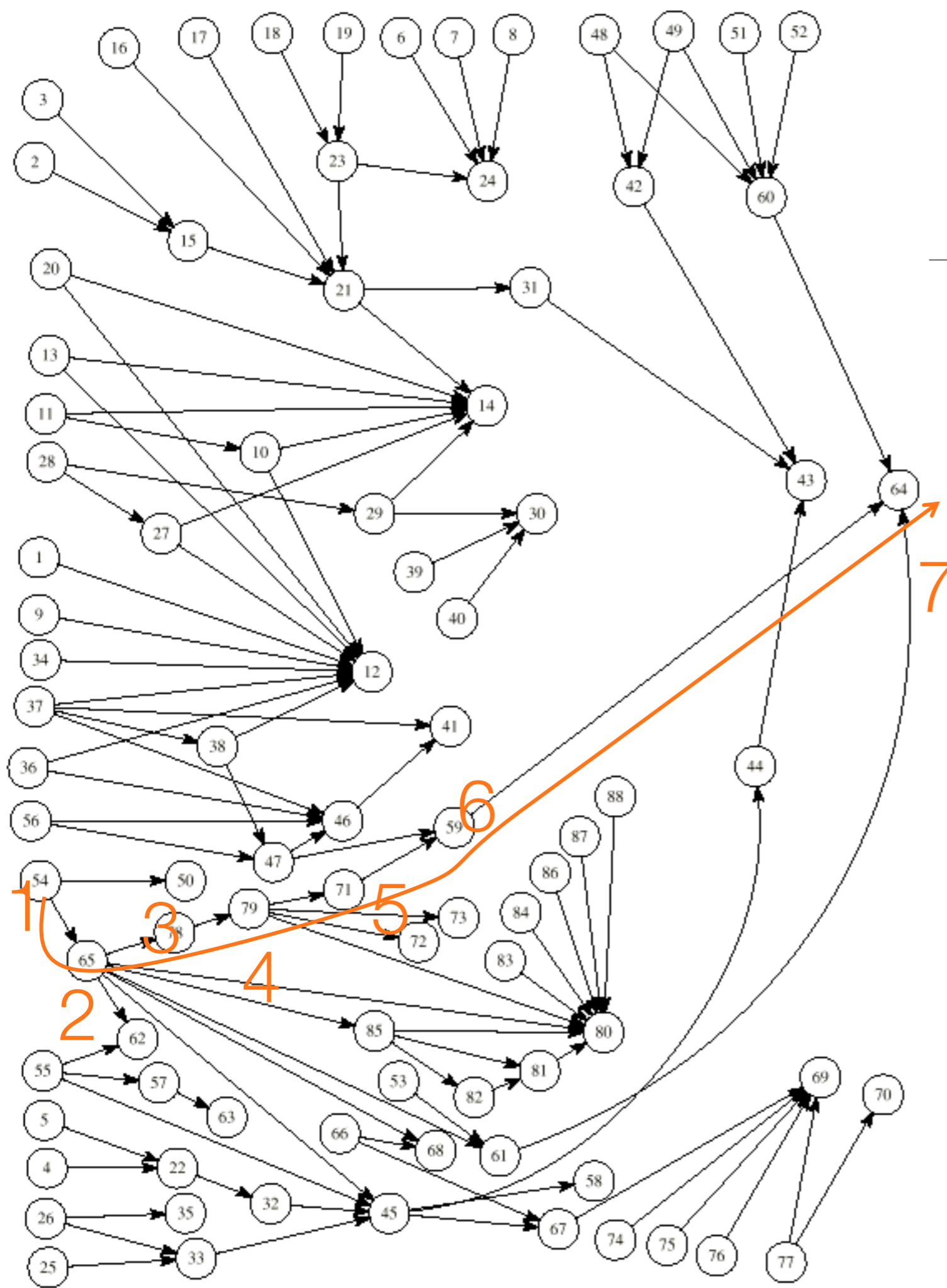
Bouchons de test



Une stratégie efficace pour l'ordre d'intégration

- Quand un ordre partiel est disponible, on peut paralléliser les tâches
 - en fonction d'un nombre fixe de testeurs
 - pour un délai minimum

Répartition des testeurs

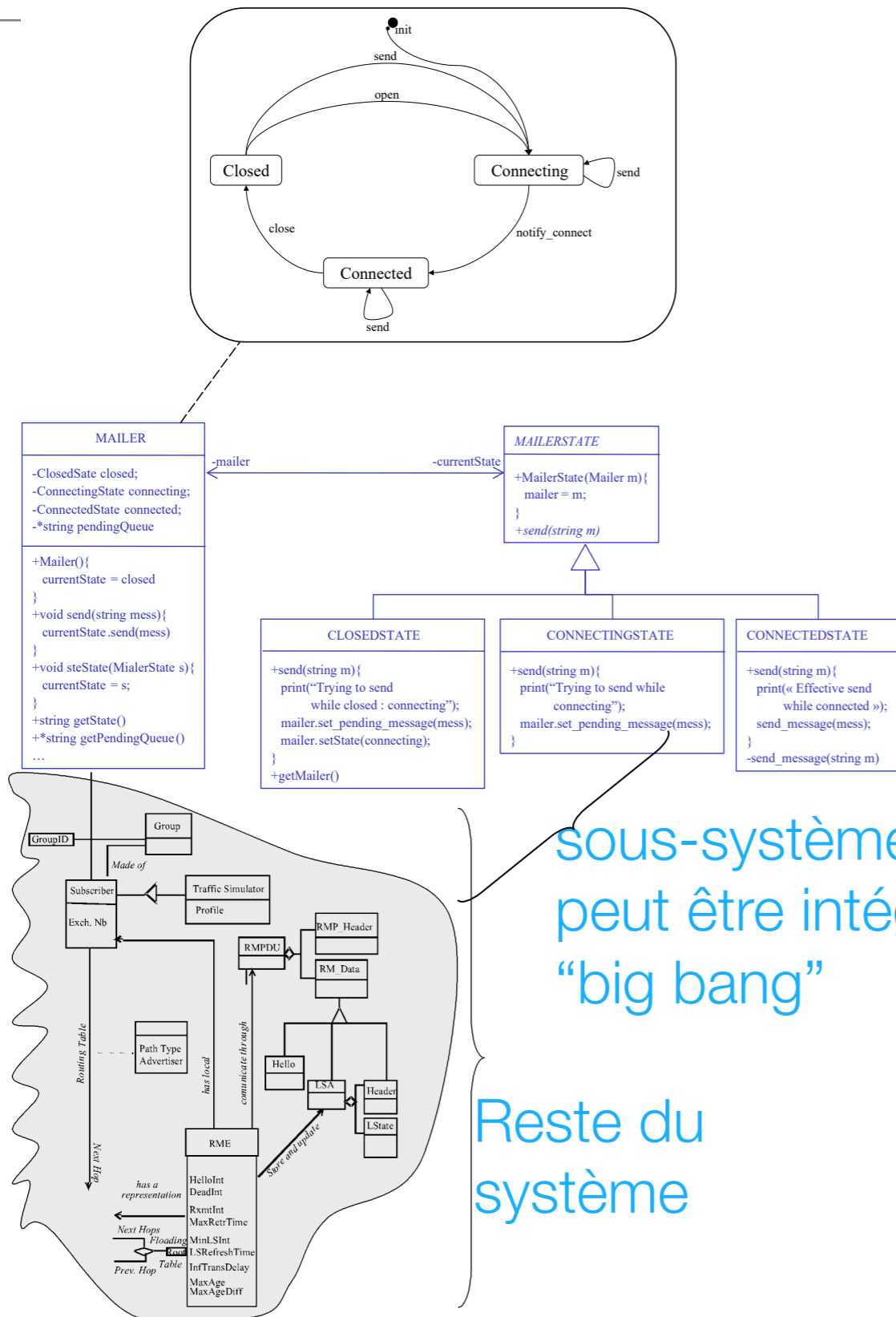


- Exemple de GNU-Eiffel
- Nombre de testeurs nécessaire pour intégrer en un minimum de temps
 - $88 \text{ div } 7 + 1 = 13$ testers
- temps minimum : 7 étapes

Une stratégie efficace pour l'ordre d'intégration

- Stratégie efficace
 - casse les cycles de dépendances avec un minimum de stubs
- Autre stratégie
 - prend en compte les pratique de conception OO
 - certains cycles sont très cohérents du point de vue fonctionnel (Ex: design patterns)
 - ça peut être intéressant d'intégrer cette interdépendance d'un coup

Stratégie mixte



sous-système qui peut être intégré en "big bang"

Reste du système

- Minimise encore le nombre de stubs
- Maintient un niveau de cohérence dans l'intégration
- Pas complètement automatisable
 - pattern matching

Plan

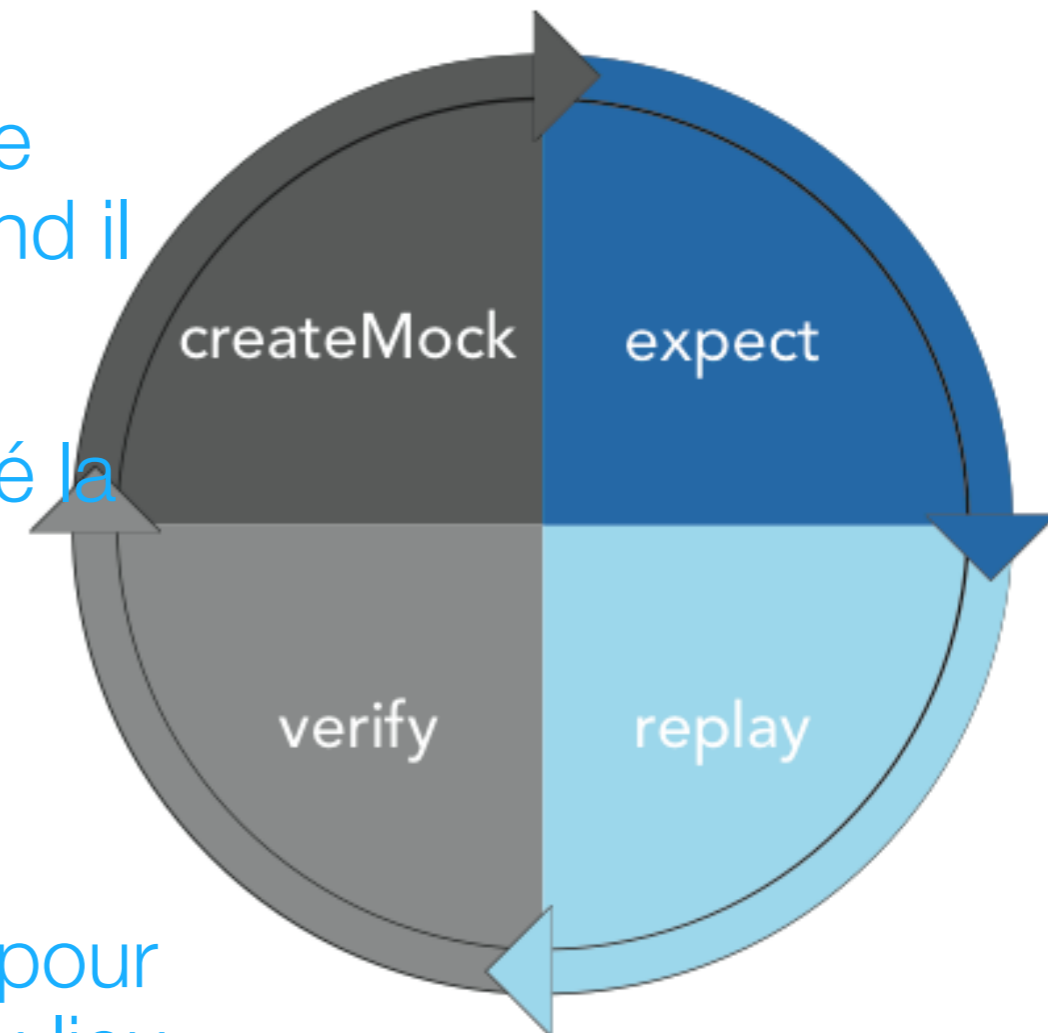
1. Introduction au test d'intégration
2. Mise en œuvre des mock avec EasyMock et Mockito

Mocks

- Les mocks sont un type de stub
- Simulent les échanges de message entre une classe et ses dépendances
- Permet de tester
 - une classe en isolation
 - Les interactions avec l'environnement

Easymock

- Étapes:
 1. Création du mock
 2. Configuration du mock pour lui dire comment il doit se comporter quand il est appelé.
 3. Activation des mocks (C'est appelé la fonction "replay" dans la documentation d'EasyMock)
 4. *Exécuter les tests*
 5. Après l'exécution, vérifier le mock pour savoir si les appels attendus ont eu lieu



Mockito

- Mockito is used by calling static methods defined in the `org.mockito.Mockito` class.

- To create a mock:

```
MyClass someMock = mock(MyClass.class);
```

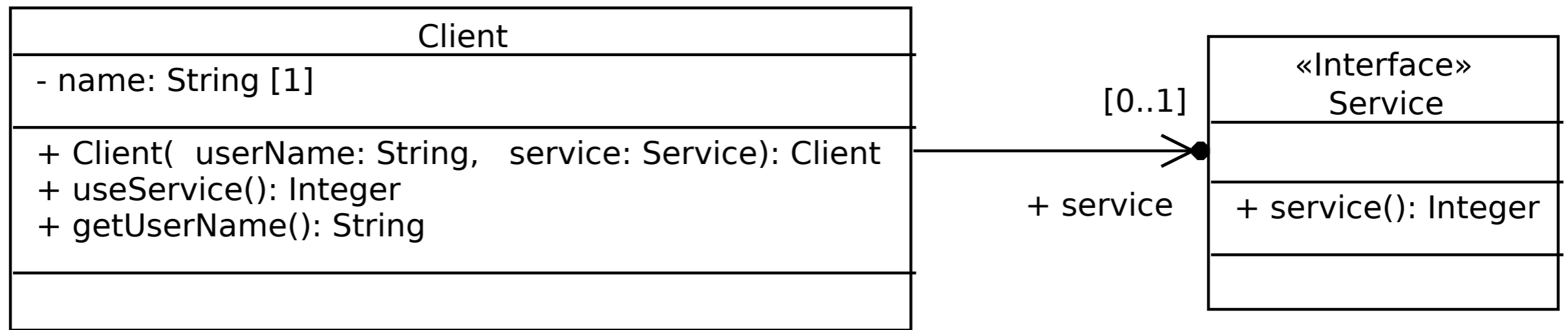
- To configure a mock:

```
when(someMock.someOp()).thenReturn(« someResult »);
```

- Tip

- **import static** org.mockito.Mockito.*; to write mock() instead of Mockito.mock(...)

Unit test Client



useService calls "service"
and returns the obtained
value increased by 10

Stub Service class

```
import org.junit.Test;
import static org.mockito.Mockito.*;
import static org.junit.Assert.*;
...

public class ClientTestWithStub {

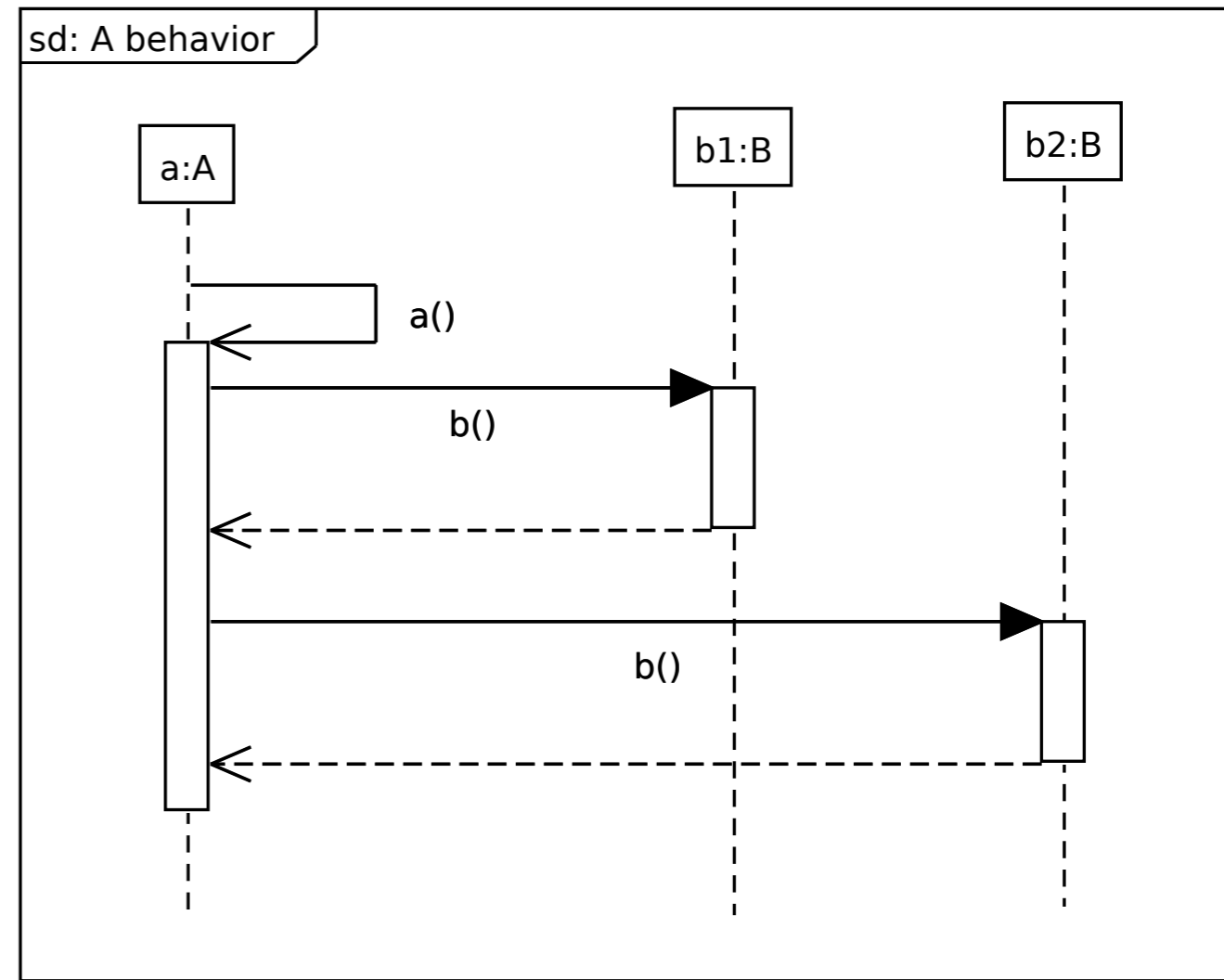
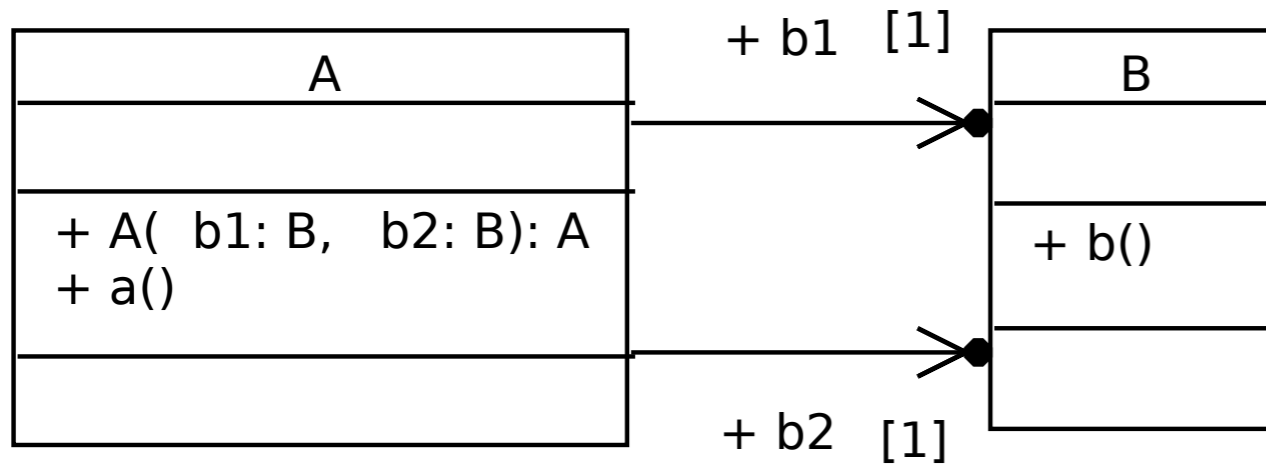
    @Test
    public void testUseService() {

        // Preparing the context -- we stub the Service instance
        Service mockService = mock(Service.class);
        when(mockService.service()).thenReturn(1327);

        // Regular test case
        Client client = new Client("John", mockService);
        int result = client.useService();

        // Oracle
        assertEquals(result, 1337);
    }
}
```


Checking calls on methods



Checking calls on methods

- It is also possible to check that methods are called on a mock

```
// Context
```

```
B mockB1 = mock(B.class);
```

```
B mockB2 = mock(B.class);
```

```
A someA = new A(mockB1, mockB2);
```

```
// Calling the tested operation
```

```
a.a();
```

```
// Checking if b1.b() was called during a()
```

```
// to see if b1 was used correctly by a
```

```
verify(b1).b();
```

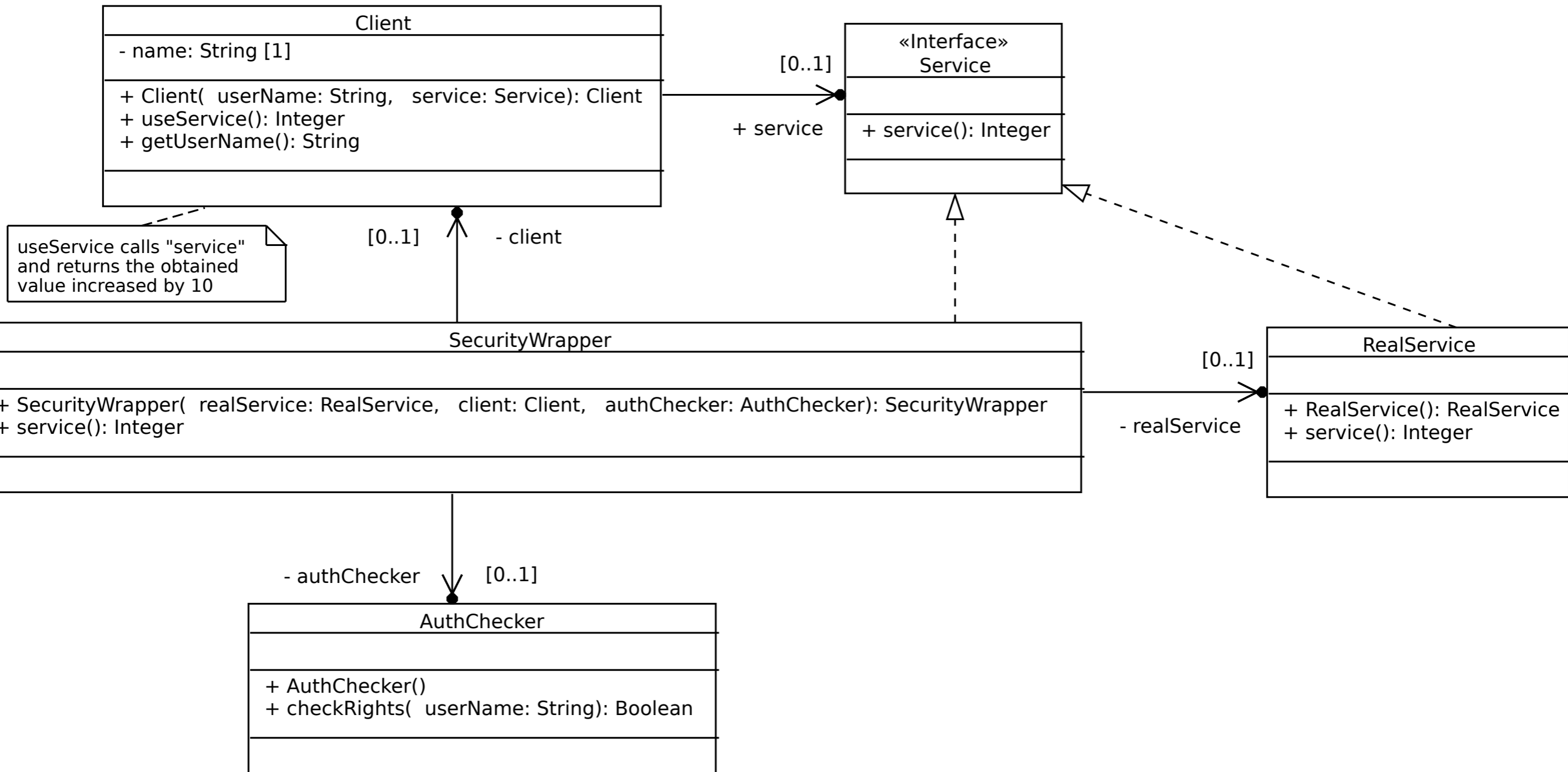
- A different form of oracle

Checking calls on methods

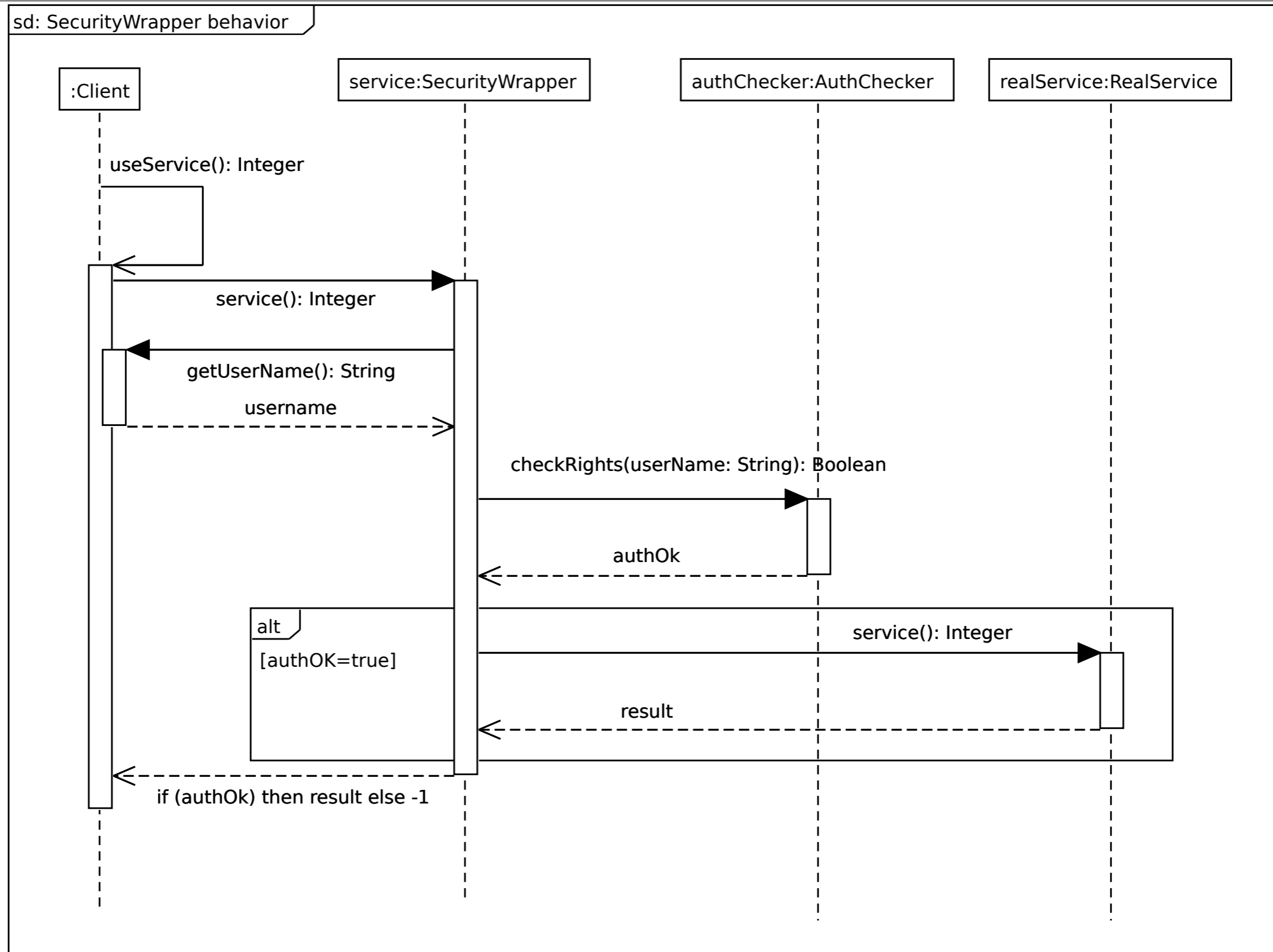
- It is also possible to check the order of method calls

```
// Checking if b1.b() was called before b2.b()  
InOrder mocksWithOrder = inOrder(b1, b2);  
mocksWithOrder.verify(b1).b();  
mocksWithOrder.verify(b2).b();
```

Test the behavior of a wrapper for Service



Test the behavior of a wrapper for Service



```
@Test
public void testSecurityWrapperBehavior() {

    // Preparing the context: mocks and their behaviors + tested object
    RealService mockRealService = mock(RealService.class);
    Client mockClient = mock(Client.class);
    AuthChecker mockAuthChecker = mock(AuthChecker.class);
    when(mockClient.getUserName()).thenReturn("John");
    when(mockAuthChecker.checkRights("John")).thenReturn(true);
    SecurityWrapper wrapper = new SecurityWrapper(mockRealService,
        mockClient, mockAuthChecker);

    // Calling the tested operation
    wrapper.service();

    // Oracle
    InOrder mocksWithOrder = inOrder(mockAuthChecker, mockRealService);
    mocksWithOrder.verify(mockAuthChecker).checkRights("John");
    mocksWithOrder.verify(mockRealService).service();
}
```

Summary

- Integration testing
 - looks for interaction between units of code
 - requires stubs to break cycles between units
- Mocks
 - Specific stubs to simulate a class (from the public interface)
 - Specific oracle to check the interactions between classes
- Integration testing is (usually) part of unit testing