Intégration de services au sein d'un serveur d'EJB

Présentateur : Audrey Occello
Equipe RAINBOW
Laboratoire I3S - Université de Nice
Soutenu par le projet RNTL ARCAD

Pourquoi intégrer des propriétés non fonctionnelles?

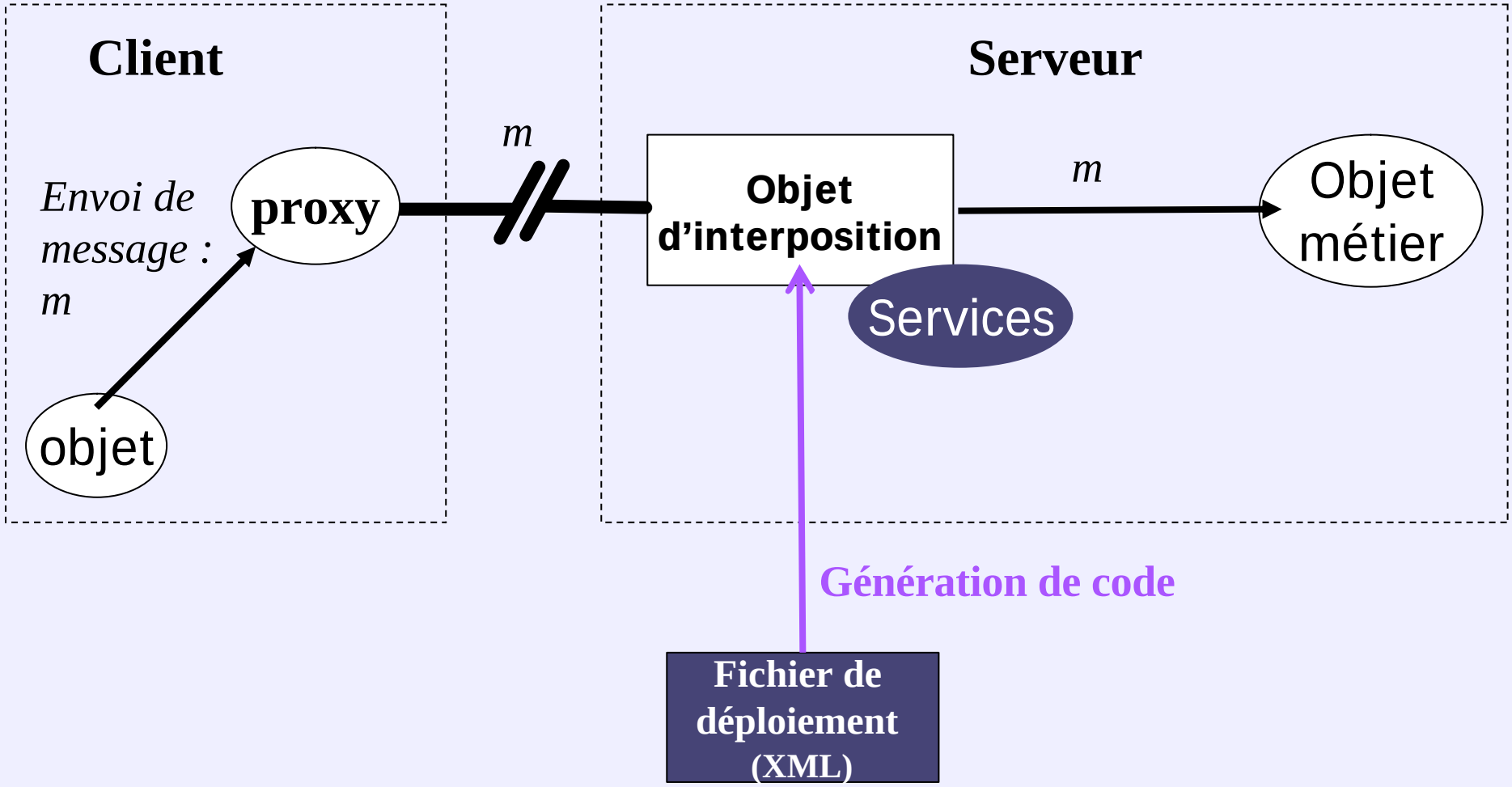
Besoin d'adaptabilité et d’évolutivité des applications

- dissocier code fonctionnel / non fonctionnel
- changer de propriétés fonctionnelles
- ajouter de nouvelles propriétés

Nombre de propriétés fonctionnelles grandissant

⇨ Réponse des modèles à composants
Faciliter la mise en œuvre de services standard

EJB : Point de vue «acquisition» de services



Difficultés d’intégration actuelles

Intégrer ou modifier une intégration de service implique :

⇨ Extension du fichier de déploiement
(i.e : DTD + analyseur syntaxique)

⇨ Génération étendue de l’objet d’interposition
(i.e : utilisation des fonctionnalités du service)

Or la tâche est :

• difficile, répétitive, source d’erreurs

Intégration d'un service … On aimerait ...

DTD pour la part d'intégration du service
exemple : <ejb-name> Account
<ordonnancement> ListeDesMéthodes ... **1**

Composantes du services
exemple: **MailBox**, ... **2**

Expression des Points de contrôle pour l'intégration du service : **méta-modèle** des Plates-formes
exemple: o.**addVariable**(**MailBox** mailBox)
o.**receive**(M) -> o.mailBox.**put**(M)
o.**send**(M) -> o.**send**(o.mailBox.**nextMessage**()) **3**

Méta-modèle commun des Plates-formes ?

Jonas

JBoss

Automatiser l’intégration de services ?

⇨ A partir de la « définition » d’une intégration de services

- code plus sûr
- intégration de service par le programmeur de plate-forme plus facile
- portable

⇨ Difficulté de trouver le méta-modèle

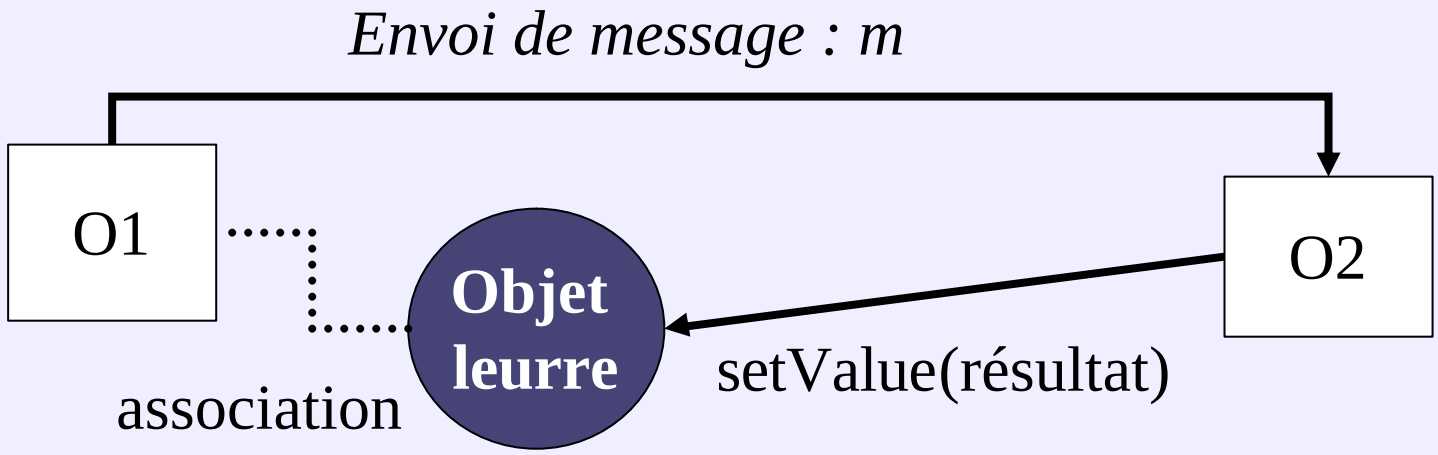
- Définition des points de contrôle
- Expression de la composition de ces points
- Projection vers les plates-formes

Démarche : une approche pragmatique

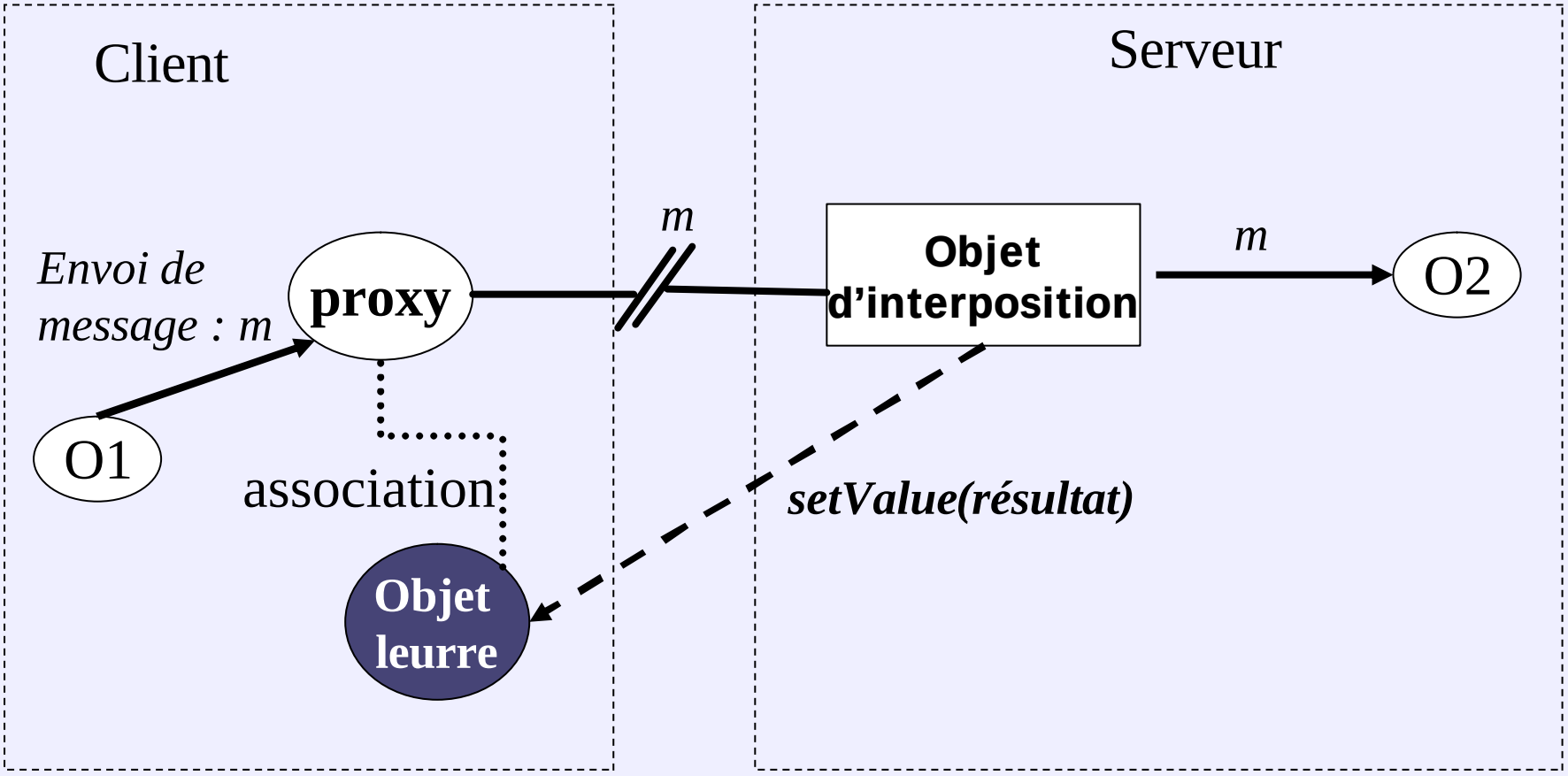
⇨ Introduire 3 services supplémentaires en utilisant des implémentations existantes :

- Asynchronisme
- Ordonnancement
- Interactions

Service d’asynchronisme (1)

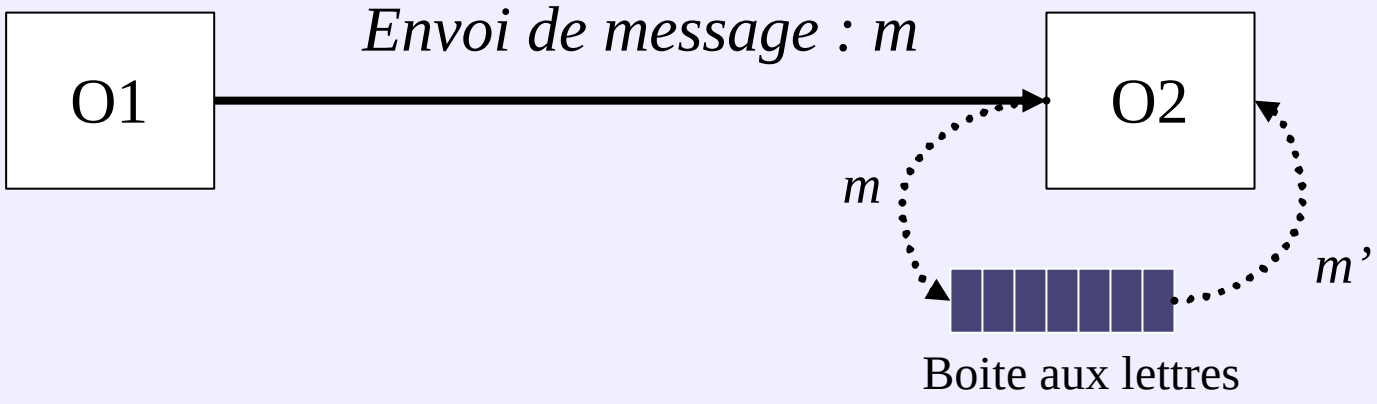


Service d'asynchronisme (2)

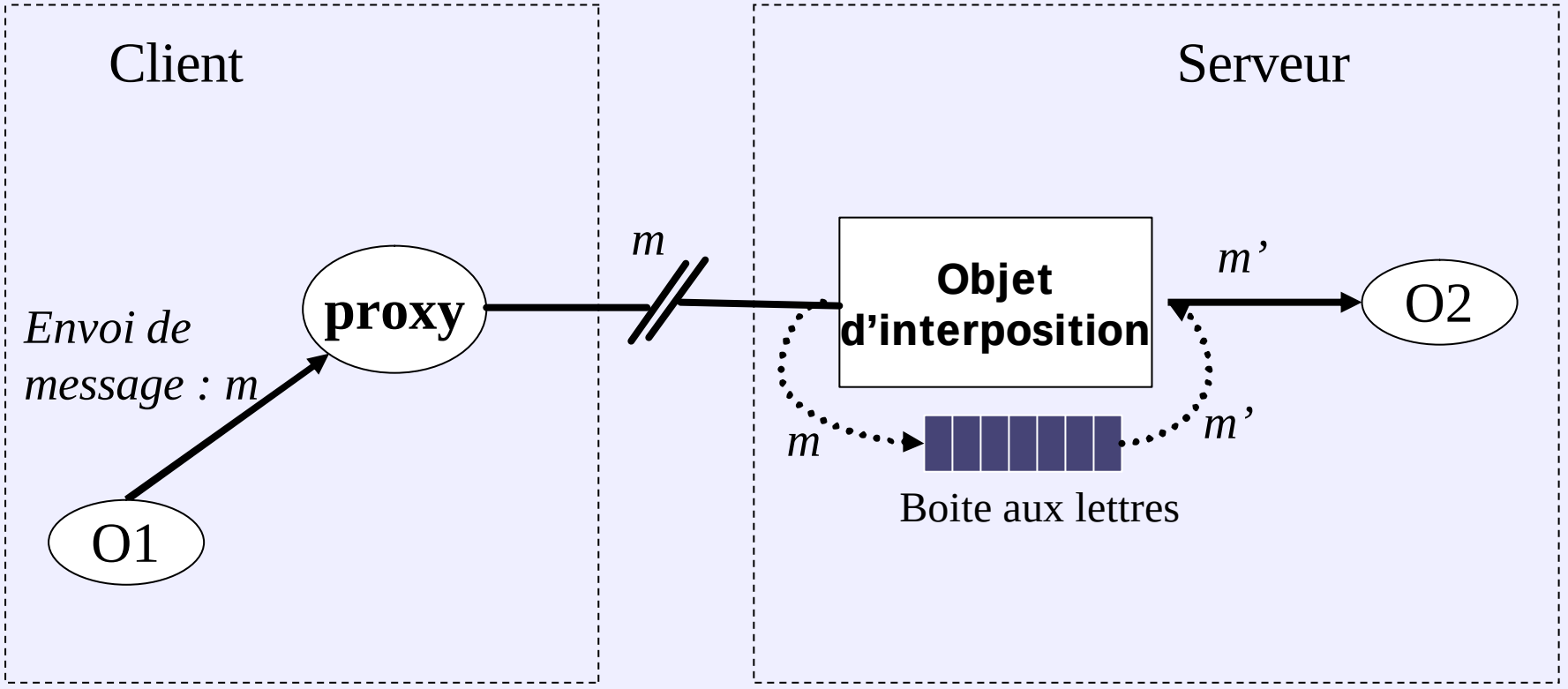


Particularité du service :
nécessite une gestion du leurre à l'envoi de message (coté client)

Service d'ordonnancement (1)



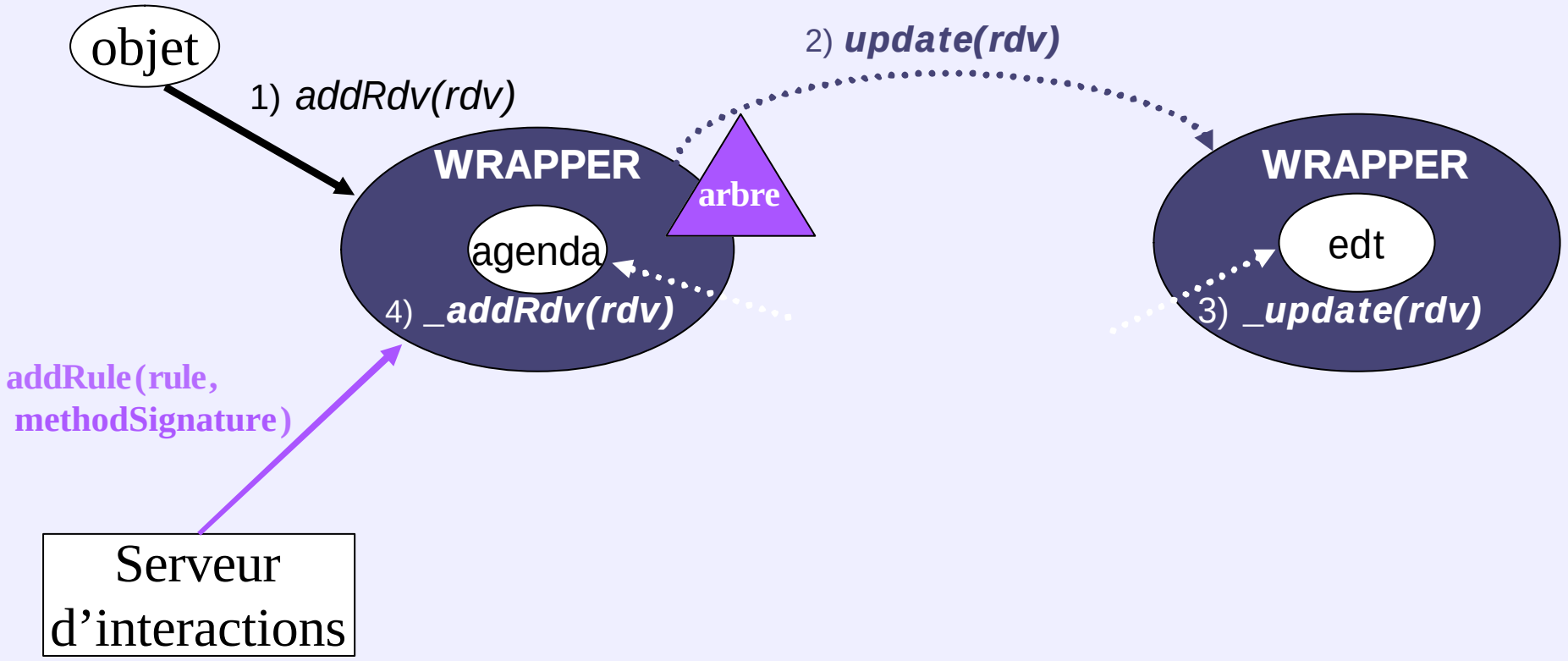
Service d'ordonnancement (2)



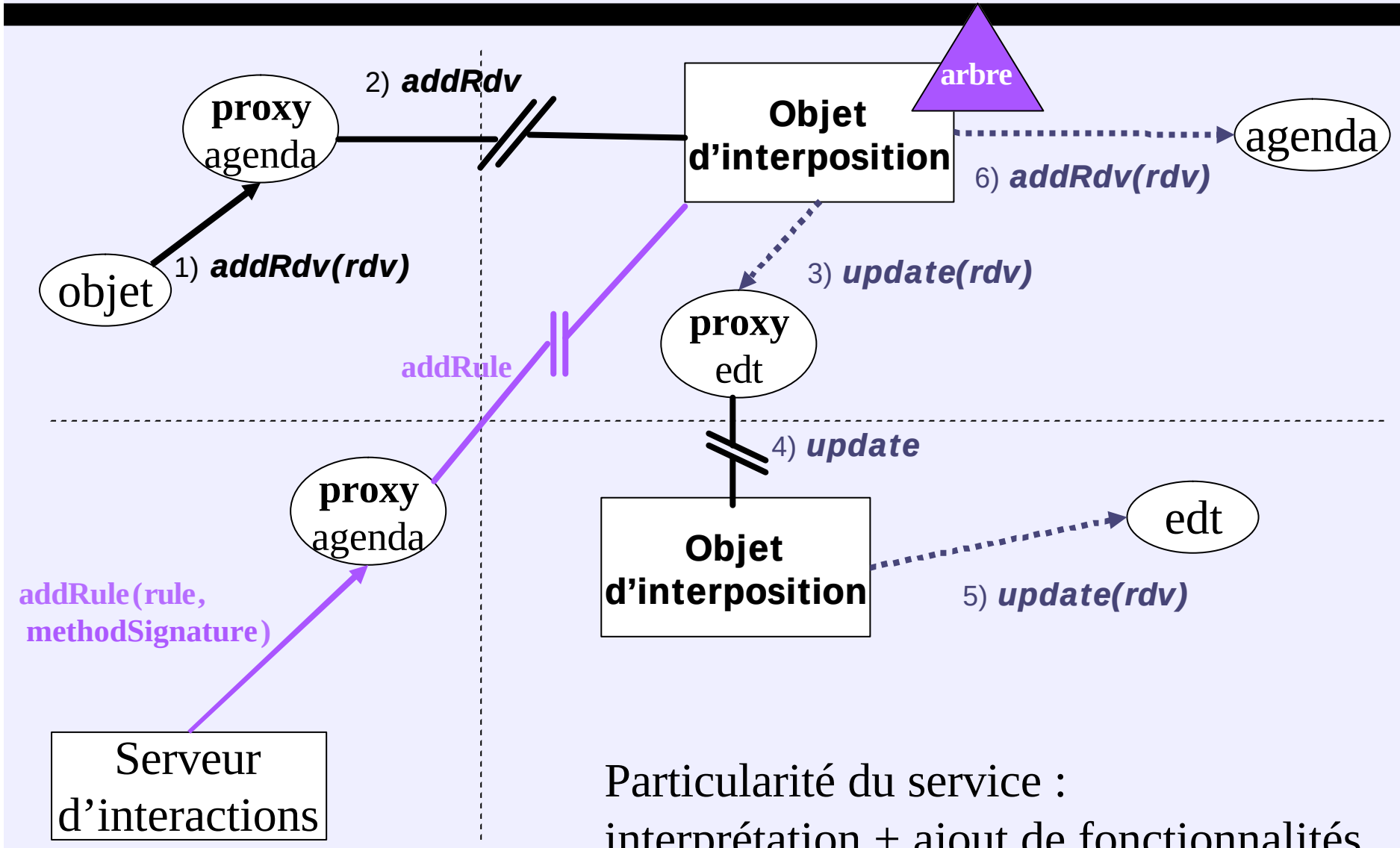
Particularité du service :
ordre d'exécution différent de l'ordre d'envoi des messages

Service d'interactions (1)

Règle : agenda.addRdv(rdv) -> edt.update(rdv); agenda.addRdv(rdv)



Service d'interactions (2)



Particularité du service :
interprétation + ajout de fonctionnalités

De l’étude pragmatique vers un méta-modèle

⇨ Identification de points de contrôle

↙ à l’émission et à la réception du message

⇨ Combinaison des actions à effectuer pour un point de contrôle donné

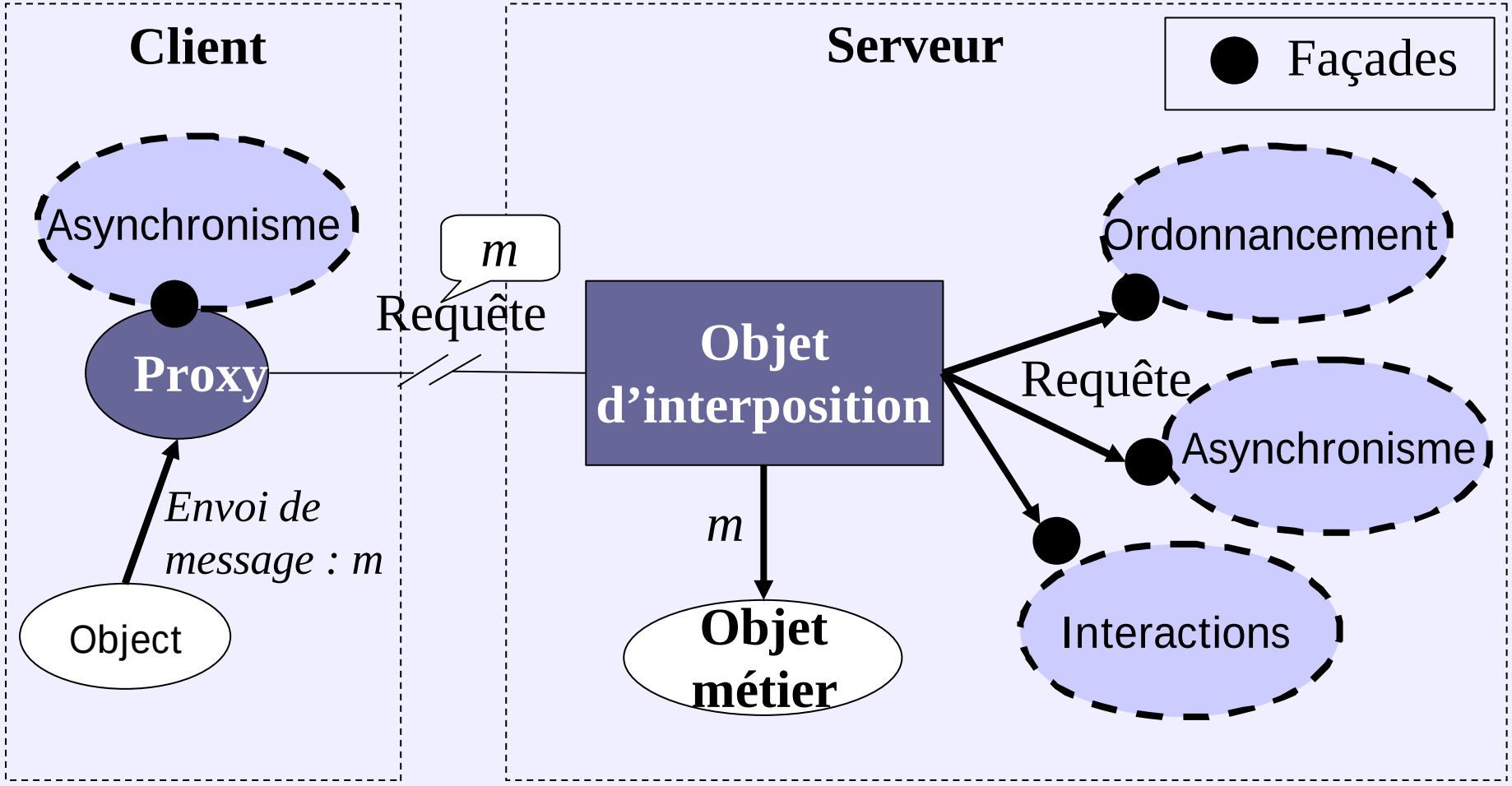
⇨ Ajout de fonctionnalités

↙ coté serveur (objet d’interposition)

⇨ Projections

↙ contrôle coté client / coté serveur

Architecture proposée



Le proxy : un contrôleur coté Client

Transparence de l'envoi de messages en exhibant

- l'interface métier de l'objet métier *(Interface Agenda)*

Mais aussi

- les fonctionnalités des services destinées au Client *(changePriority, getRules, ..)*

Réification de requêtes

↙ Ajout d'information pour les services

Contrôle de l'émission de messages (gestion du leurre)

Les composants techniques

Un service = un composant technique contenant :

- des « objets » d'implémentation
- une façade

Façade = interface commune à tous les services

- permet de s'adresser à tous les services de manière identique (*receive(requête)*)
- est extensible

⇨Facilite la communication avec les services

L’objet d’interposition : un contrôleur côté serveur

Fournisseur de services

- Facettes multiples « à la CORBA »
  - Interfaces pour les clients
  - Interfaces pour les administrateurs

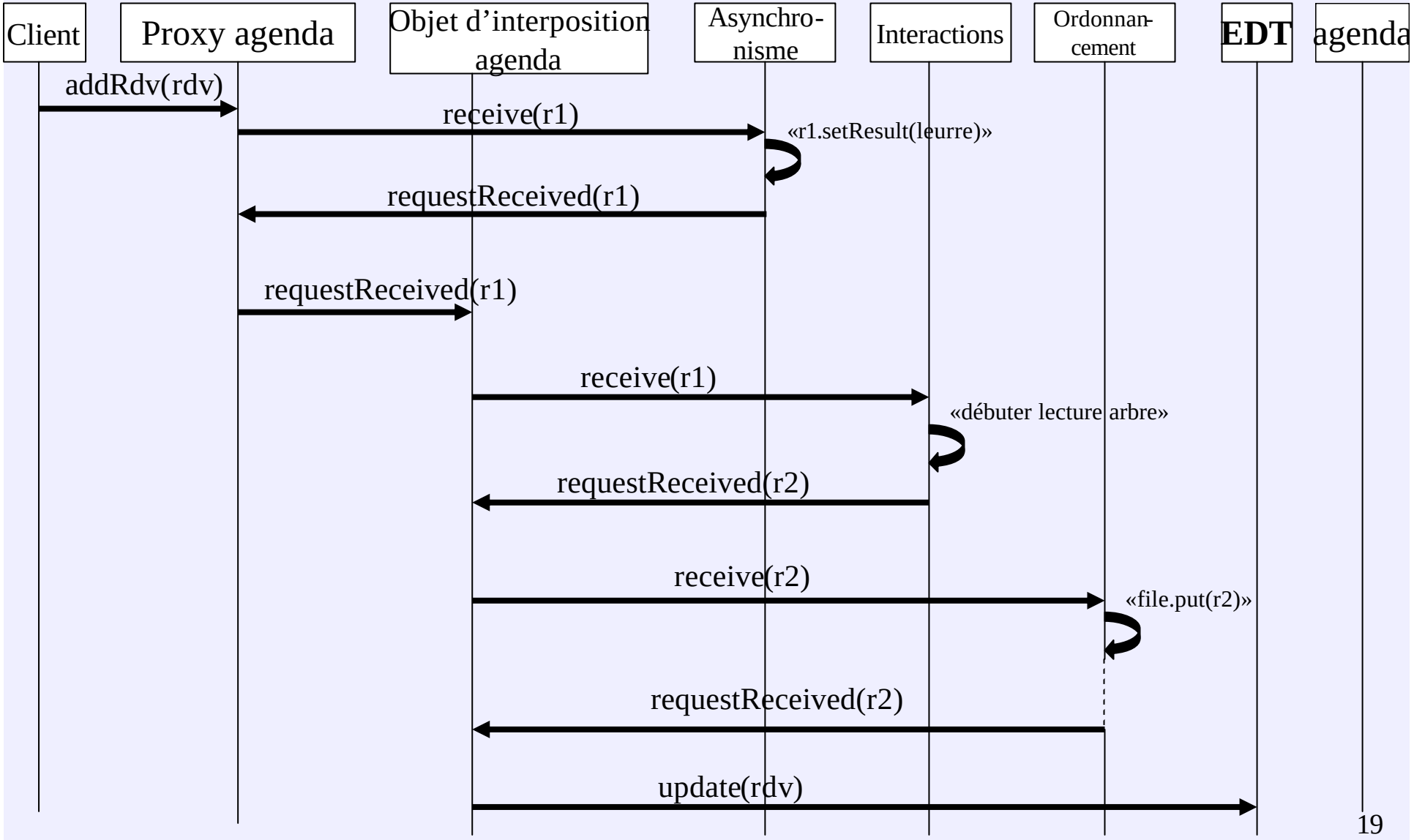
Intercepteur de Requêtes

- Ajout de comportement lors de la réception des requêtes (*requestReceived(requete)*)

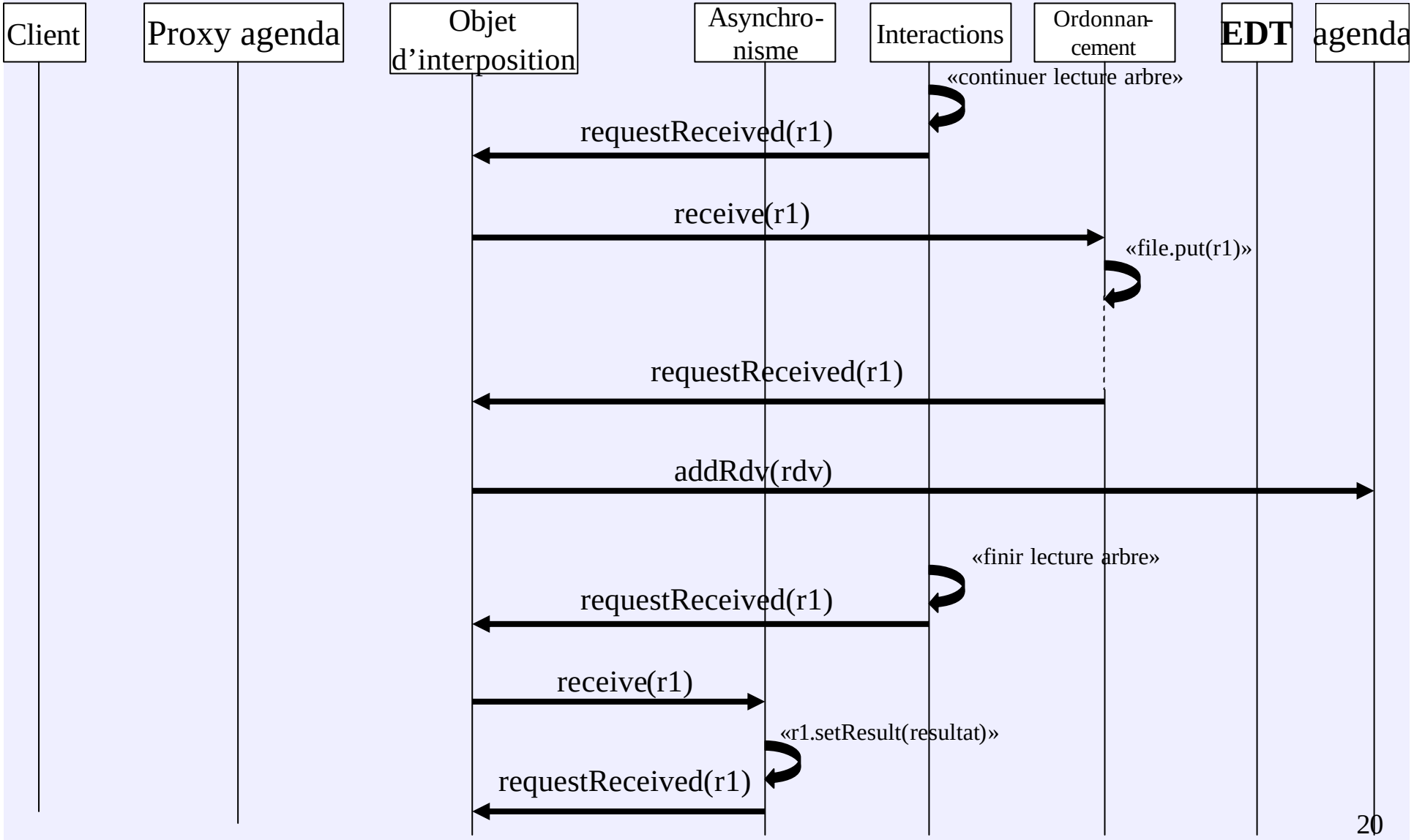
Routeur des requêtes

- Coordination entre les services (automate)
- Routage éventuel vers l’objet métier

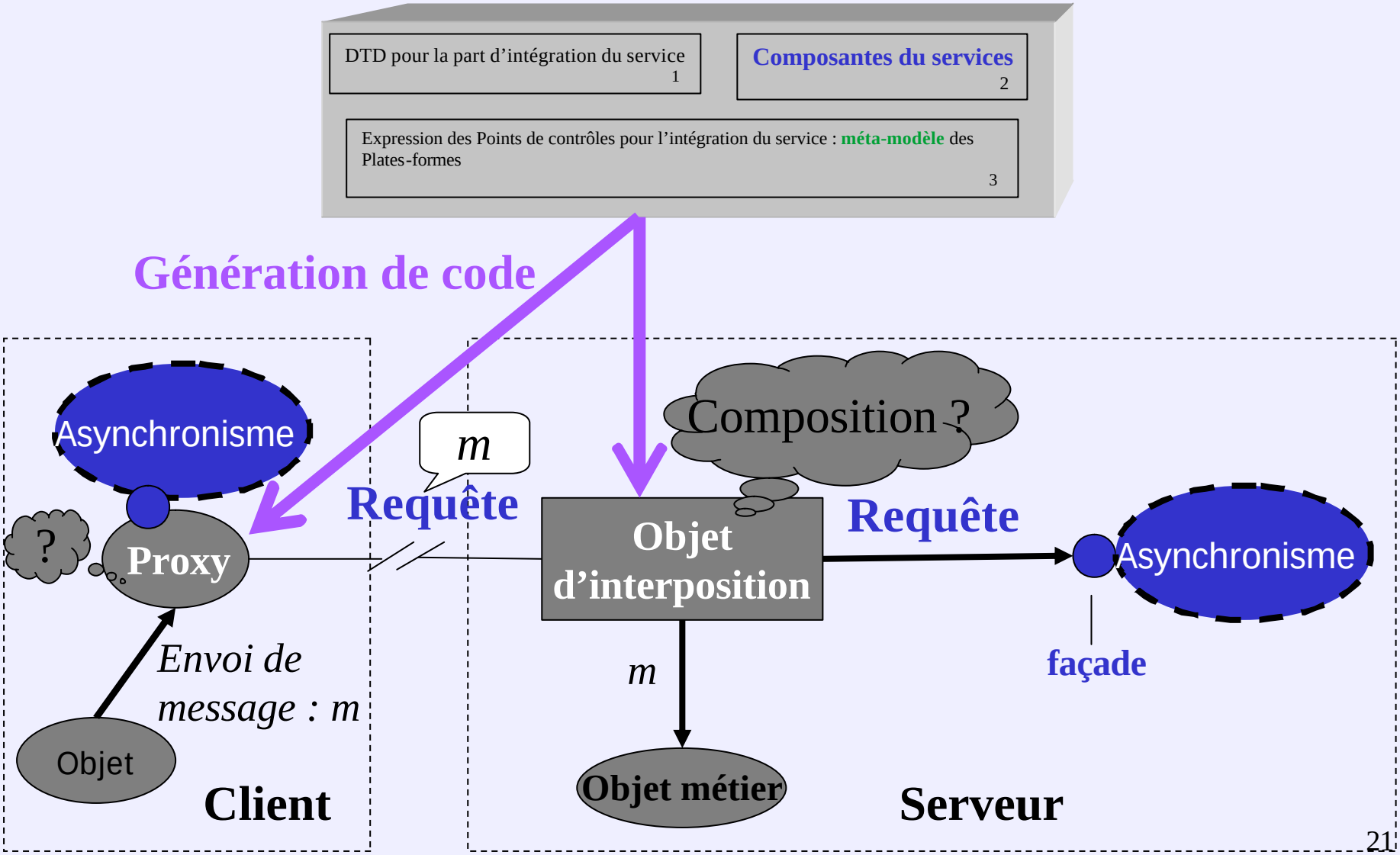
Routage (1) : agenda.addRdv(rdv) -> edt.update(rdv); agenda.addRdv(rdv)



Routage (2) : agenda.addRdv(rdv) -> edt.update(rdv); agenda.addRdv(rdv)



Intégration d'un service … exemple



Réflexions autour de l’architecture

Diminution du coût de communication avec l’objet métier :

⇨ fusionner l’objet d’interposition avec l’objet métier (wrapper) ?

Augmenter la capacité d'extensibilité :

⇨ découpler les fonctions d’interception et de routage (container ouvert) ?

Réflexivité de l’architecture :

⇨ L’objet d’interposition est lui-même une entité qui doit pouvoir bénéficier de services

⇨ Faut-il lui ré-appliquer l’architecture … tour infinie?

Conclusion (1)

L’architecture proposée a permis

- d’intégrer 3 services
- d'identifier des points de contrôle
- d’appréhender les besoins liés à la composition

Constat issu de l’expérimentation :

- ↙ Raffiner les points de contrôle
- ↙ Expliciter les « interactions » introduites entre composants métiers et/ou techniques

Conclusion (2)

Approche statique

Début de généralisation (3 services bien choisis)

⇨ Pouvoir automatiser le plus possible la génération de code (automates de combinaison)

⇨ Pouvoir le faire dynamiquement

⇨ Aller plus loin dans l'expression déclarative de l'intégration de services

Integration d'un service … perspectives

DTD pour la part d'intégration du service
exemple : <ejb-name> Account
<ordonnancement> ListeDesMéthodes ... **1**

Composantes du services
exemple: **MailBox**, ... **2**

Expression des Points de contrôle pour l'intégration du service : **méta-modèle** des Plates-formes
exemple: o.**addVariable**(**MailBox** mailBox)
o.**receive**(M) -> o.mailBox.**put**(M)
o.**send**(M) -> o.**send**(o.mailBox.**nextMessage**()) **3**

Méta-modèle commun des Plates-formes

actuellement : contrôleurs, façades, communication par requête

Jonas

JBoss

?

CCM