ISIMA

Patrons de services

(PARTIE VII - Patrons de conception)

Bruno Bachelet

Christophe Duhamel

Luc Touraille

Patrons de service *(1/2)*

- Accéder et configurer des services
- Quelque soit le mécanisme de communication entre composants
- Application *«standalone»*
  - Simple espace d'adressage
    - Fonctions avec paramètres
    - Variables globales
- Application en réseau
  - Communication interprocessus (IPC)
    - Mémoire partagée, tubes, sockets (TCP, UDP)…
  - Protocoles de communication
    - Telnet, FTP, SSH, HTTP…
  - Opérations distantes via services
    - CORBA, COM+…

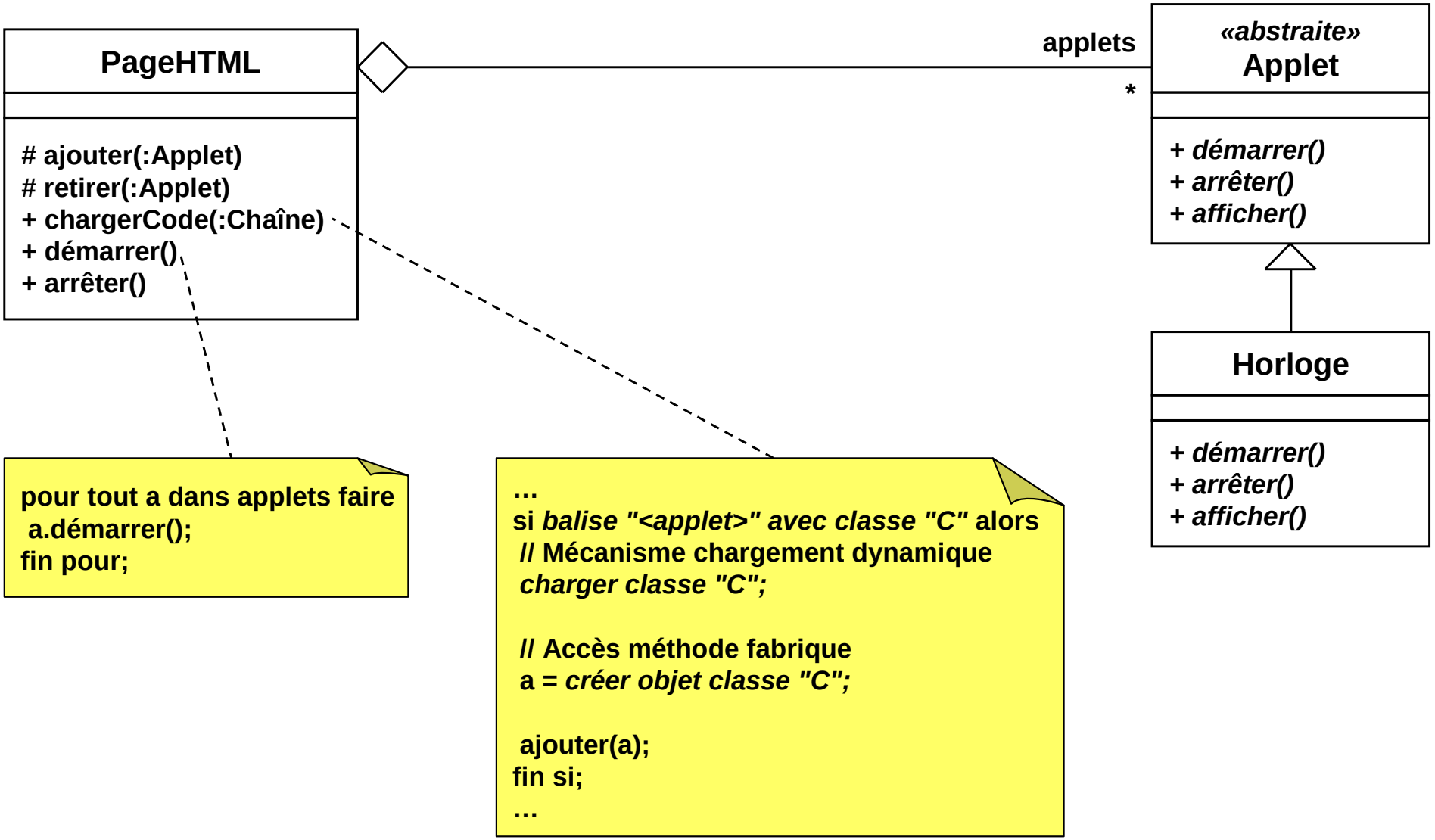
Patrons de service *(2/2)*

- Façade d'adaptation / *Wrapper Facade*
  - Fournir une interface objet pour une API non objet existante
- Configurateur (de composant) / (*Component) Configurator*
  - Lier dynamiquement des implémentations (sans recompilation)
- Intercepteur / *Interceptor*
  - Ajouter de manière transparente des services à un *framework*
- Interface d'extension / *Extension Interface*
  - Permettre à un composant d'exporter plusieurs interfaces

Configurateur / *Configurator (1/4)*

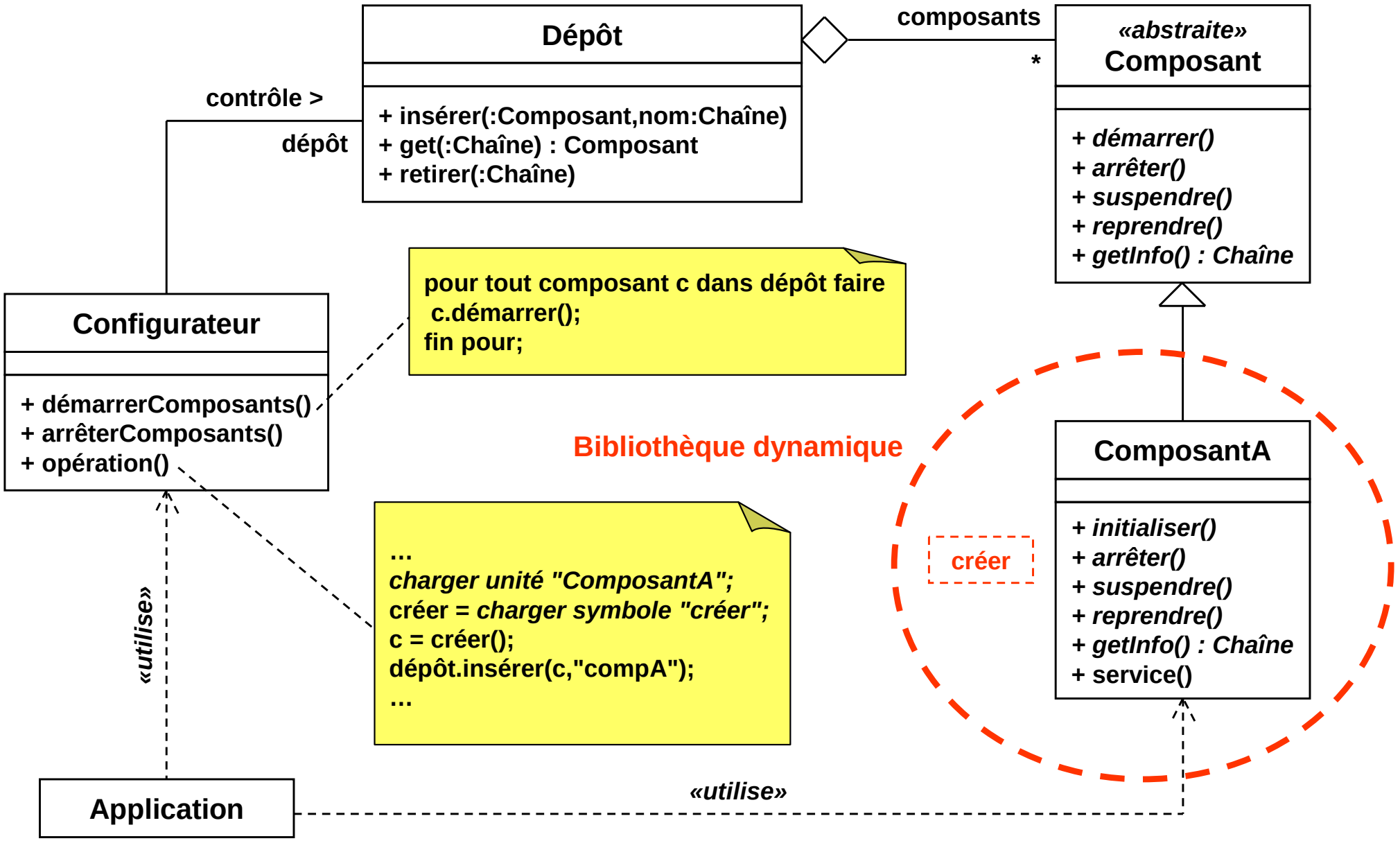
- Objectif
  - Lier et délier l'implémentation de composants à l'exécution
  - Sans recompiler, ni éditer les liens
- Principe
  - Une classe abstraite représente les composants
    - Nouvelle implémentation = définition d'une sous-classe
    - Sous-classe stockée dans une unité chargeable dynamiquement
      - Exemples: bibliothèque dynamique (DLL), classe Java
  - Un «configurateur» manipule les composants
    - S'occupe de la création, du démarrage et de l'arrêt des composants
- Motivation
  - Page HTML où les applets sont chargées dynamiquement

Configurateur / *Configurator (2/4)*



Exemple code HTML: `<applet code="Horloge.class">...</applet>`

Configurateur / *Configurator (3/4)*



Configurateur / *Configurator (4/4)*

- Appelé aussi «*service configurator*»

- Intérêts
  - Uniformité des composants
    - Tous les composants respectent la même interface
  - Administration centralisée
    - Facilite le démarrage/arrêt de tous les composants
  - Remplacement de composants «à chaud»
    - Chargement/déchargement dynamique de composants
  - Permet une adaptation dynamique
    - Mécanisme d'analyse / Apprentissage

    ⇒ Réglage des paramètres du composant

    ⇒ Chargement d'un composant mieux adapté

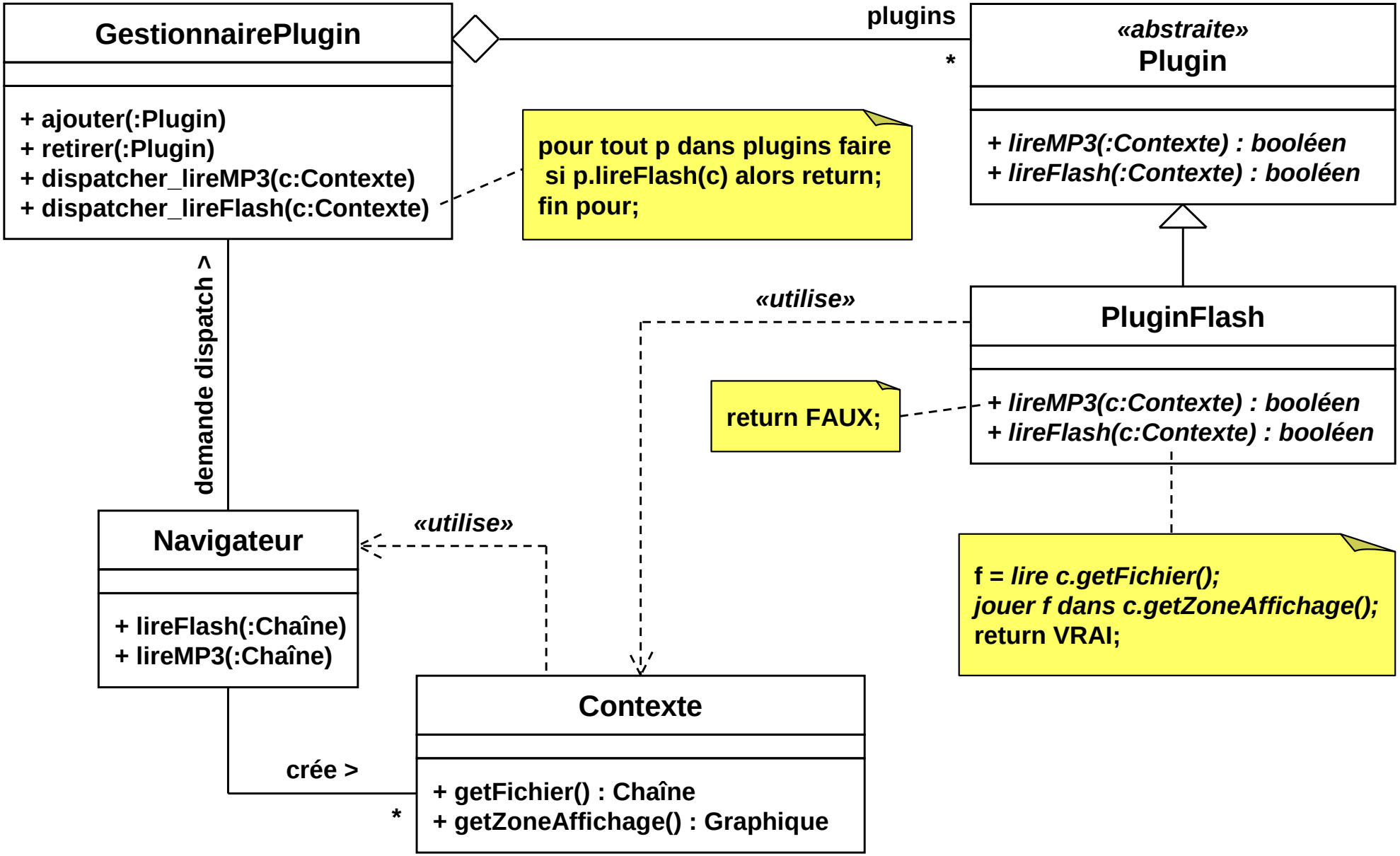
Intercepteur / *Interceptor (1/4)*

- Objectif
  - Ajouter de manière transparente des services à un *framework*
  - Les activer automatiquement suite à certains événements

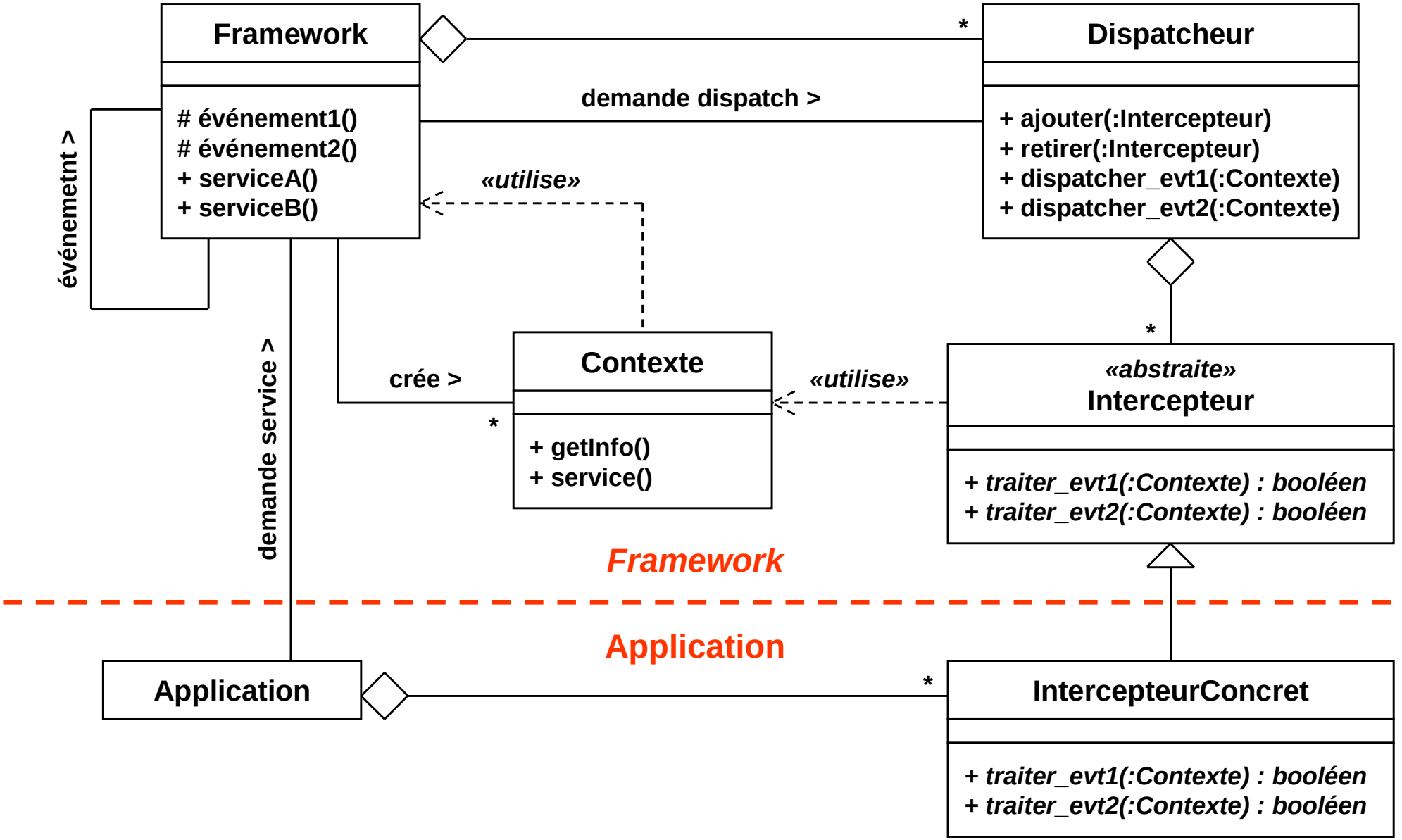
- Principe
  - Un «dispatcheur» est chargé de diffuser les événements
  - Des «intercepteurs» sont capables de recevoir les événements
    - Classe abstraite possédant une méthode par événement
    - Héritage $\Rightarrow$ Proposition de nouveaux services
  - Les intercepteurs n'ont pas un accès direct au *framework*
    - Communication par l'intermédiaire d'un «contexte»
    - Le contexte accède aux données et services du *framework*

- Motivation
  - Mécanisme de *plugins* dans un navigateur Web

Intercepteur / *Interceptor (2/4)*



Intercepteur / *Interceptor (3/4)*



Intercepteur / *Interceptor (4/4)*

- Intérêts
  - Extensibilité du *framework*
    - Services liés aux intercepteurs intégrés au *framework*
    - Nouveaux services ⇒ Héritage de «`Intercepteur`»
      - Aucun impact sur le *framework*
  - Séparation des intérêts
    - Infrastructure du *framework* d'un côté
    - Les services de l'autre
      - Inutile de connaître toute l'infrastructure pour coder un service
  - Mécanisme de contrôle et de surveillance
    - Intercepteur + contexte ⇒ Moyen de tracer l'application

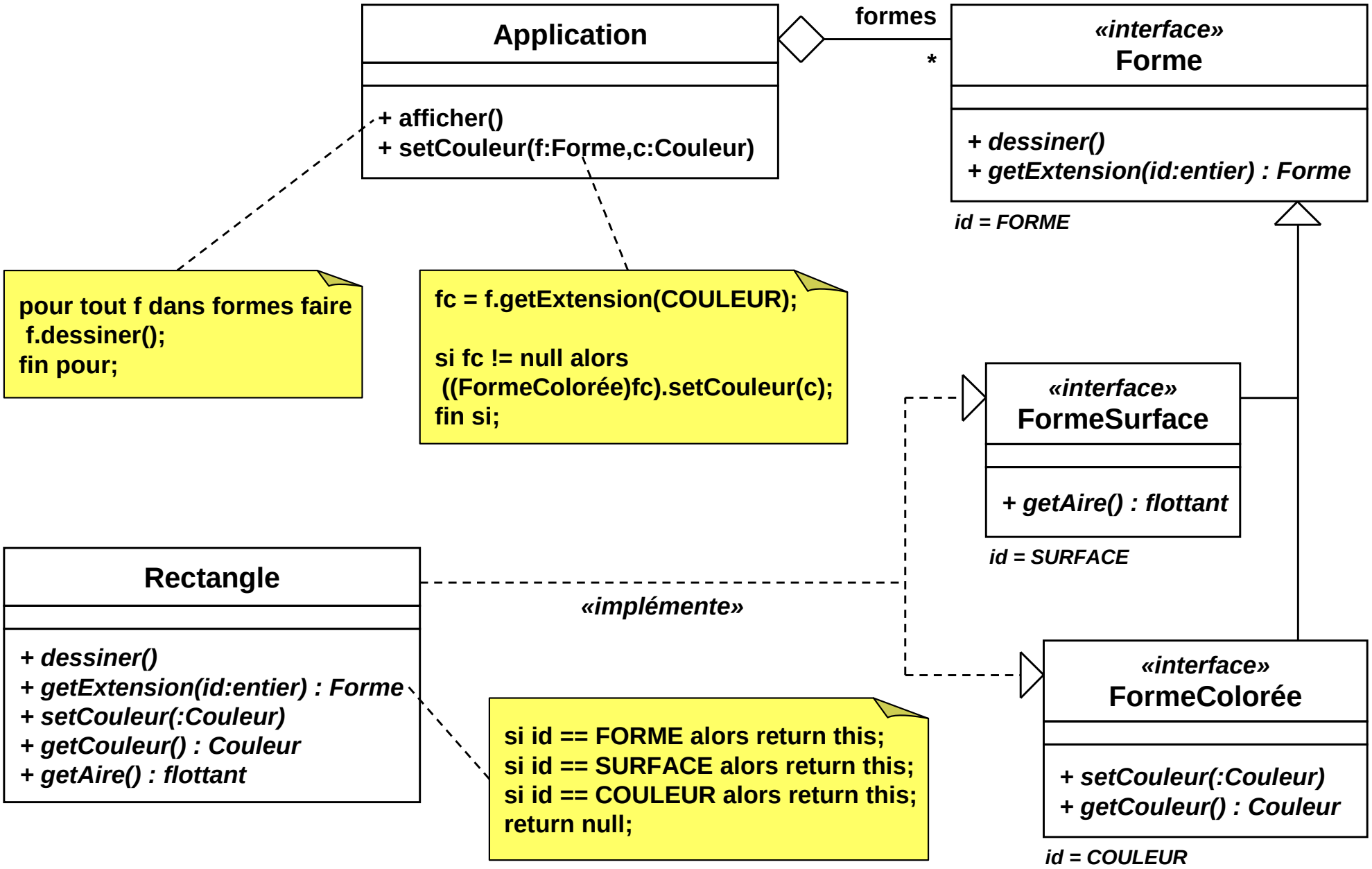
Interface d'extension / *Extension Interface (1/4)*

- Objectif
  - Permettre à un composant d'exporter plusieurs interfaces
  - Eviter le «gonflement» de son interface
    - Lors de l'ajout de nouvelles fonctionnalités
- Principe
  - Décomposer l'interface par intérêt
    - Une interface mère + des interfaces filles (une par intérêt)
  - Un composant agrège plusieurs interfaces
    - Plusieurs possibilités

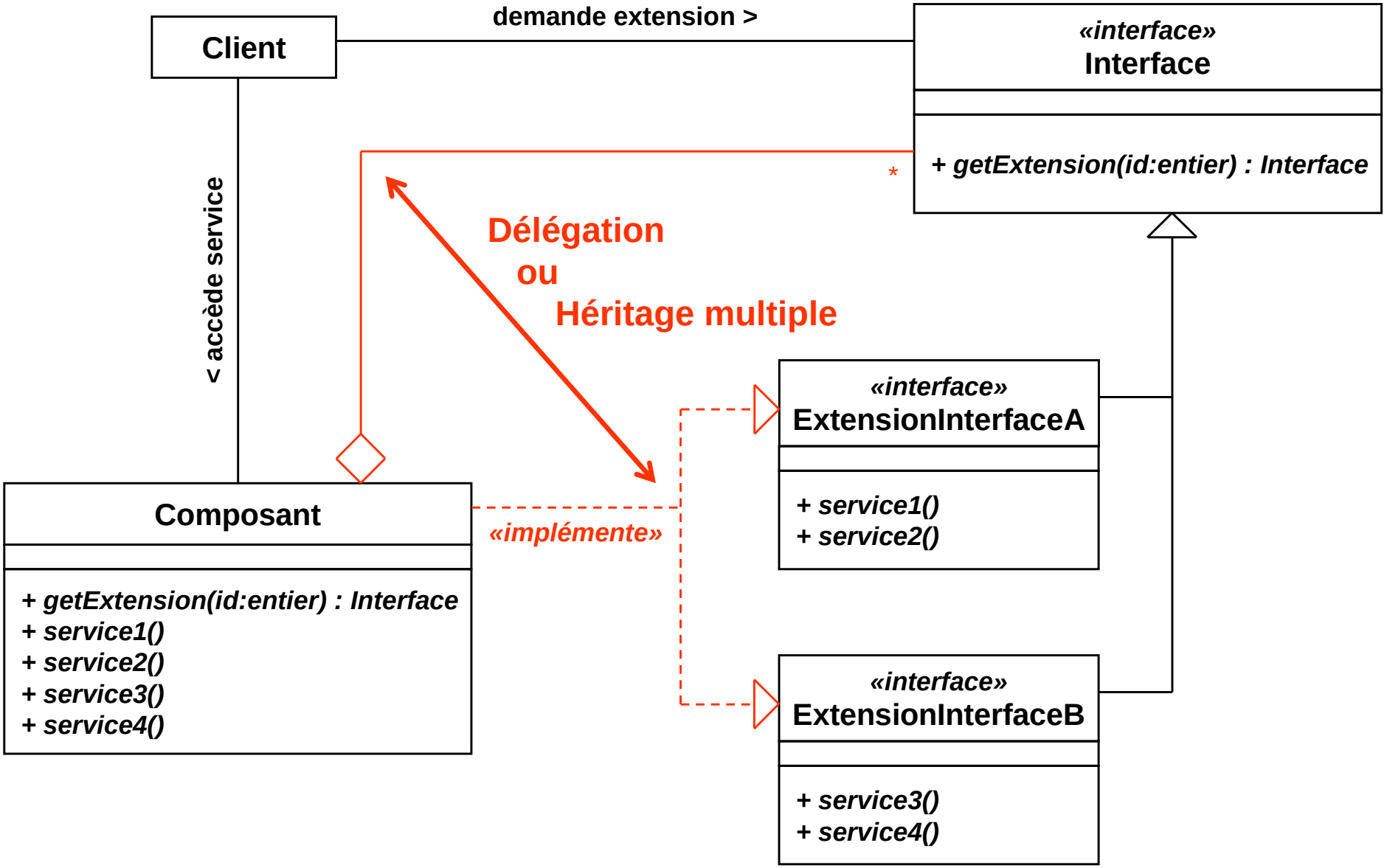
    ⇒ Implémentation multiple des interfaces

    ⇒ Agrégation d'interfaces et délégation
- Motivation
  - Proposer de nombreuses fonctionnalités sur des composants
  - Mais en évitant d'alourdir l'interface de tous les composants

Interface d'extension / *Extension Interface (2/4)*



Interface d'extension / *Extension Interface (3/4)*



Interface d'extension / *Extension Interface (4/4)*

- Intérêts
  - Extensibilité
    - Ajout de nouvelles fonctionnalités ⇒ Nouvelle interface
  - Séparation des intérêts
    - Une interface par thème
  - Attention au surcoût
    - Accès indirect au composant
- Relations avec d'autres patrons
  - Pont
    - Implémentation de la composition d'interfaces
  - Fabrique abstraite
    - Peut être utilisée pour la création de composants