

Intergiciels

Dr. Philippe Merle
 ADAM / INRIA Futurs – GOAL / LIFL - USTL
<http://www.lifl.fr/~merle>





Objectifs de ce cours

- **Introduire les principes de base de l'architecture des intergiciels ...**

- **... via une démarche systématique**
 - ◆ Identifier les principaux schémas d'architecture répondant aux problèmes des applications réparties
 - ◆ Mettre en évidence les patrons de conception (*design patterns*) et les canevas logiciels (*software frameworks*) utiles pour la construction de l'intergiciel
 - ◆ Illustrer l'usage de ces patrons et canevas sur des exemples concrets

© Novembre 2007, P. Merle Intergiciels 2




Architecture de l'intergiciel

- Schémas d'interaction
- Patrons élémentaires : *Proxy*, *Factory*, *Adapter*, *Interceptor*
- Schémas de décomposition

© 2003-2007, S. Krakowiak ICAR'06 3

Services et interfaces

- **Définition**
 - ◆ Un système est un ensemble de composants (au sens non technique du terme) qui interagissent
 - ◆ Un service est "un comportement défini par contrat, qui peut être implémenté et fourni par un composant pour être utilisé par un autre composant, sur la base exclusive du contrat" (*)
- **Mise en œuvre**
 - ◆ Un service est accessible via une ou plusieurs interfaces
 - ◆ Une interface décrit l'interaction entre client et fournisseur du service
 - ◆ Point de vue opérationnel : définition des opérations et structures de données qui permettent l'accès au service
 - ◆ Point de vue contractuel : définition du contrat entre client et fournisseur

(*) Bieber and Carpenter, *Introduction to Service-Oriented Programming*, <http://www.openwings.org>

© 2003-2007, S. Krakowiak ICAR'06 4

Définitions d'interfaces (1)

- ◆ La fourniture d'un service met en jeu deux interfaces
 - ◆ Interface requise (côté client)
 - ◆ Interface fournie (côté fournisseur)
- ◆ Le contrat spécifie la compatibilité (conformité) entre ces interfaces
 - ◆ Au delà de l'interface, chaque partie est une "boîte noire" pour l'autre (principe d'encapsulation)
 - ◆ Conséquence : client ou fournisseur peuvent être remplacés du moment que le composant remplaçant respecte le contrat (est conforme)

© 2003-2007, S. Krakowiak ICAR'06 5

Définitions d'interfaces (2)

- **Partie "opérationnelle"**
 - ◆ **Interface Definition Language (IDL)**
 - ◆ Pas de standard, mais s'appuie sur un langage existant
 - ▲ IDL CORBA sur C++
 - ▲ Java et C# définissent leur propre IDL
- **Partie "contractuelle"**
 - ◆ **Plusieurs niveaux de contrats**
 - ◆ Sur la forme : spécification de types -> conformité syntaxique
 - ◆ Sur le comportement (1 méthode) : assertions -> conformité sémantique
 - ◆ Sur les interactions entre méthodes : synchronisation
 - ◆ Sur les aspects non fonctionnels (performances, etc.) : contrats de QoS, ou SLA (*Service Level Agreement*)

© 2003-2007, S. Krakowiak ICAR'06 6

Langage de définition d'interfaces

- Divers langages de définition d'interfaces
 - ◆ Interface Definition Language (IDL)
 - ◆ Permet de décrire des interfaces
 - ◆ Opérations, paramètres, exceptions et types de données
 - ◆ Environnements hétérogènes
 - ◆ OMG Interface Definition Language (OMG IDL)
 - ◆ Microsoft IDL
 - ◆ Web Services Definition Language (WSDL)
 - ◆ ...
 - ◆ Environnements homogènes
 - ◆ Interfaces Java
 - Génération automatique de souches et de squelettes de communication

© 2005-2007, P. Merle Intergiciels 7

Séparation Interface – Implantation Souches et squelettes

The diagram illustrates the separation of interface and implementation. At the top, a **Client** uses an **Interface** (containing **Opération()**), which is implemented by a **Service**. Below this, a **Générateur** (Generator) takes the interface and produces a **Souche** (Stub) and a **Squelette** (Skeleton). The **Souche** and **Squelette** both implement the **Opération()** method. These stubs and skeletons are used by an **Intergiciel** (Interoperability layer), which then communicates over a **Réseau** (Network).

© 2005-2007, P. Merle Intergiciels 8

Notion de séparation Interface - Implantation

- Interface = ensemble d'opérations publiques
- La séparation Interface – Implantation permet de
 - ◆ Caractériser le protocole d'accès à un composant logiciel indépendamment de son implantation
 - ◆ Réaliser plusieurs implantations de la même interface
 - ◆ Favoriser la substitution d'implantations

The diagram shows a **Client** using an **Interface** (with **Opération()**). This interface is implemented by three separate components: **ImplantationA**, **ImplantationB**, and **ImplantationC**, each containing its own **Opération()** method.

© 2005-2007, P. Merle Intergiciels 9

Réalisation des services

Dans un environnement réparti, le schéma abstrait cache une réalité complexe

- Le client et le fournisseur sont en général sur des sites différents
- Le client ne connaît pas au départ l'identité (ou la localisation) du fournisseur
- Le client et le fournisseur peuvent être mobiles
- Le fournisseur peut lui même être réalisé de manière répartie

Le contrat doit néanmoins être maintenu

© 2003-2007, S. Krakowiak ICAR'06 10

Schémas d'interaction (1)

Schémas asynchrones
Exécution parallèle de l'émetteur et du récepteur
Couplage faible

◆ Événement-réaction ◆ Messages asynchrones

événements et messages peuvent être ou non mémorisés

© 2003-2007, S. Krakowiak ICAR'06 11

Schémas d'interaction (2)

■ Appel synchrone

- ◆ L'émetteur (Client) est bloqué en attendant le retour
- ◆ Couplage fort

Avec appel en retour (*callback*)

© 2003-2007, S. Krakowiak ICAR'06 12

Schémas d'interaction (3)

Inversion du contrôle

- Situation où B "contrôle" A
- La requête de service pour A est déclenchée depuis l'extérieur

© 2003-2007, S. Krakowiak ICAR'06 13

Accès à un service

© 2003-2007, S. Krakowiak ICAR'06 14

Patrons de conception (1)

Définition [dépasse le cadre de la conception de logiciel]


- Ensemble de règles (définitions d'éléments, principes de composition, règles d'usage) permettant de répondre à une classe de besoins spécifiques dans un environnement donné.

Propriétés

- Un patron est élaboré à partir de l'expérience acquise au cours de la résolution d'une classe de problèmes apparentés; il capture des éléments de solution communs
- Un patron définit des principes de conception, non des implémentations spécifiques de ces principes.
- Un patron fournit une aide à la documentation, par ex. en définissant une terminologie, voire une description formelle ("langage de patrons")

E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995
 F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. *Pattern-Oriented Software Architecture - vol. 1*, Wiley 1996
 D. Schmidt, M. Stal, H. Rohnert, F. Buschmann. *Pattern-Oriented Software Architecture - vol. 2*, Wiley, 2000

© 2003-2007, S. Krakowiak ICAR'06 15




patrons de conception (2)




- Définition d'un patron**
 - ◆ Contexte : Situation qui donne lieu à un problème de conception ; doit être aussi générique que possible (mais éviter l'excès de généralité)
 - ◆ Problème : spécifications, propriétés souhaitées pour la solution; contraintes de l'environnement
 - ◆ Solution :
 - ✦ Aspects statiques : composants, relations entre composants; peut être décrit par diagrammes de classe ou de collaboration
 - ✦ Aspects dynamiques : comportement à l'exécution, cycle de vie (création, terminaison, etc.); peut être décrite par des diagrammes de séquence ou d'état
- Catégories de patrons**
 - ◆ Conception : petite échelle, structures usuelles récurrentes dans un contexte particulier
 - ◆ Architecture : grande échelle, organisation structurelle, définit des sous-systèmes et leurs relations mutuelles
 - ◆ Idiomatiques: constructions propres à un langage

Source: F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. *Pattern-Oriented Software Architecture* - vol. 1, Wiley 1996

© 2003-2007, S. Krakowiak ICAR'06 16



Quelques patrons de base




- Proxy**
 - ◆ Patron de conception : représentant pour accès à distance
- Factory (+ Pool)**
 - ◆ Patron de conception : création d'objet
- Wrapper [Adapter]**
 - ◆ Patron de conception : transformation d'interface
- Interceptor**
 - ◆ Patron d'architecture : adaptation de service


Ces patrons sont d'un usage courant dans la construction d'int logiciel

Nombreux exemples dans toute la suite

© 2003-2007, S. Krakowiak ICAR'06 17

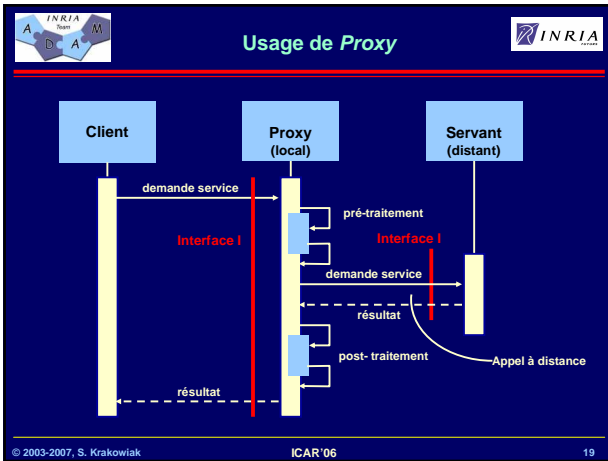


Proxy (Mandataire)

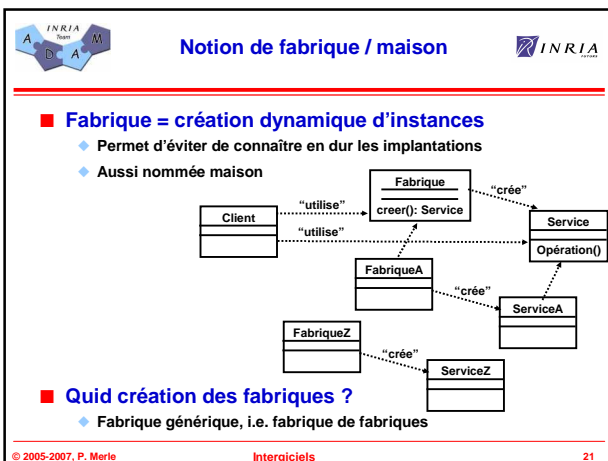


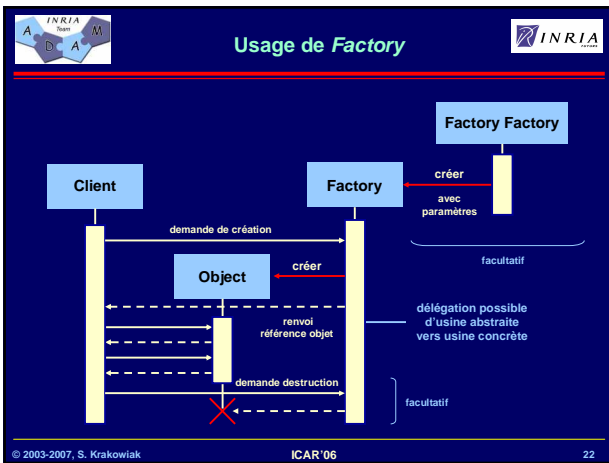
- Contexte**
 - ◆ Applications constituées d'un ensemble d'objets répartis ; un client accède à des services fournis par un objet pouvant être distant (le "servant")
- Problème**
 - ◆ Définir un mécanisme d'accès qui évite au client
 - ✦ Le codage "en dur" de l'emplacement du servant dans son code
 - ✦ Une connaissance détaillée des protocoles de communication
 - ◆ Propriétés souhaitables
 - ✦ Accès efficace et sûr
 - ✦ Programmation simple pour le client ; accès "transparent"
 - ◆ Contraintes
 - ✦ Environnement réparti (pas d'espace unique d'adressage)
- Solutions**
 - ◆ Utiliser un représentant local du servant sur le site client (isole le client du servant et du système de communication)
 - ◆ Garder la même interface pour le représentant et le servant
 - ◆ Définir une structure uniforme de représentant pour faciliter sa génération automatique

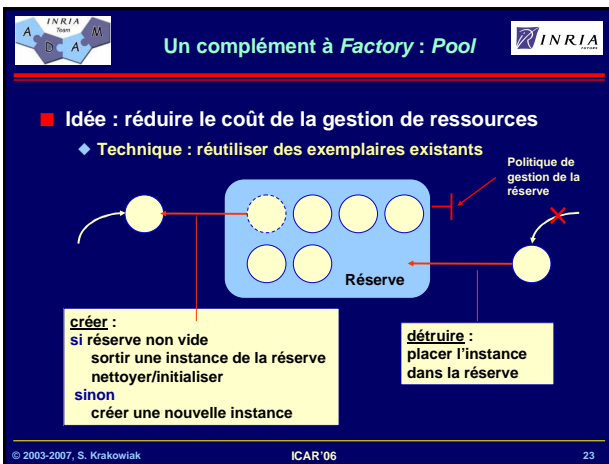
© 2003-2007, S. Krakowiak ICAR'06 18











- ### Utilisation de Pool
- Gestion de la mémoire
 - ◆ Réserve (pool) de zones (plusieurs tailles possibles)
 - ◆ Évite le coût du ramasse-miettes
 - ◆ Évite les copies inutiles (chaînage de zones)
 - Gestion des activités
 - ◆ Réserve de threads
 - ◆ Évite le coût de la création
 - Gestion de la communication
 - ◆ Réserve de connexions
 - Gestion des composants
 - ◆ Réserve de composants
 - ◆ Mise en place dans conteneurs
- © 2003-2007, S. Krakowiak ICAR'06 24

Wrapper (ou Adapter)

- **Contexte**
 - ◆ Des clients demandent des services ; des servants fournissent des services ; les services sont définis par des interfaces
- **Problème**
 - ◆ Réutiliser un servant existant en modifiant son interface et/ou certaines de ses fonctions pour satisfaire les besoins d'une classe de clients
 - ◆ Propriétés souhaitables : doit être efficace ; doit être adaptable car les besoins peuvent changer de façon imprévisible ; doit être réutilisable (générique)
 - ◆ Contraintes :
- **Solutions**
 - ◆ Le *Wrapper* isole le *servant* en interceptant les appels de méthodes vers l'interface de celui-ci. Chaque appel est précédé par un prologue et suivi par un épilogue dans le *Wrapper*
 - ◆ Les paramètres et résultats peuvent être convertis

© 2003-2007, S. Krakowiak ICAR'06 25

Notion d'adaptateur

- **Adaptateur = entité gérant l'impédance entre interfaces du client et celles du service**
 - ◆ Conversion de paramètres
 - ◆ 1 appel client -> plusieurs appels du service

```

classDiagram
    class Client
    class InterfaceA {
        Opération()
    }
    class Adapter
    class InterfaceB {
        Méthode()
    }
    class Service
    Client ..> InterfaceA : "utilise"
    Adapter ..> InterfaceA
    Adapter ..> InterfaceB : "utilise"
    InterfaceB ..> Service : "implante"
  
```

© 2005-2007, P. Merle Intergiciels 26

Usage de Wrapper

```

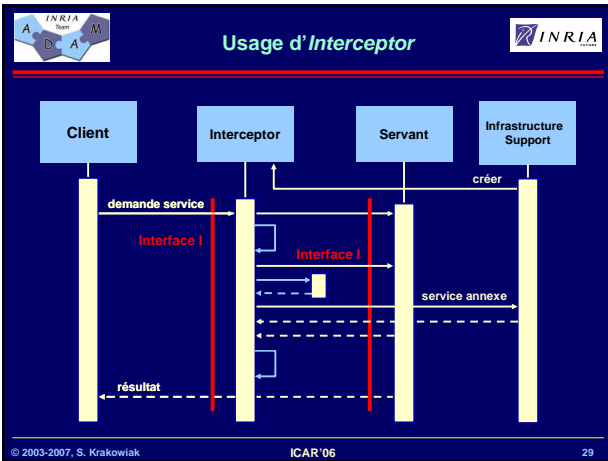
sequenceDiagram
    participant Client
    participant Wrapper
    participant Servant
    Client->>Wrapper: demande service (Interface I2)
    Wrapper->>Wrapper: pré-traitement
    Wrapper->>Servant: demande service (Interface I1)
    Servant-->>Wrapper: résultat
    Wrapper->>Wrapper: post-traitement
    Wrapper-->>Client: résultat
  
```

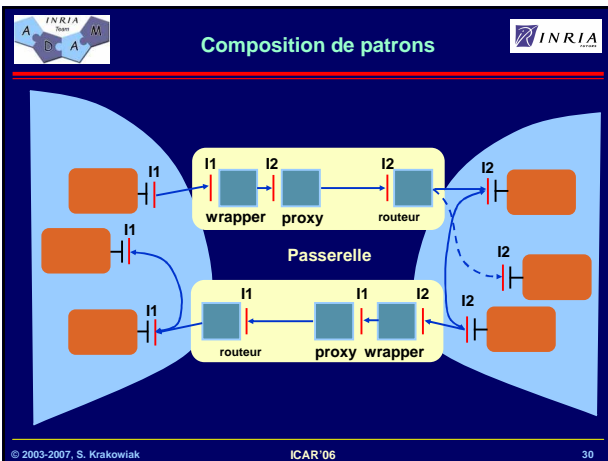
© 2003-2007, S. Krakowiak ICAR'06 27

Interceptor (Intercepteur)

- Contexte**
 - ◆ **Fourniture de services (cadre général)**
 - ◆ Client-serveur, pair à pair, hiérarchique
 - ◆ Uni- ou bi-directionnel, synchrone ou asynchrone
- Problème**
 - ◆ **Transformer le service (ajouter de nouvelles fonctions), par différents moyens**
 - ◆ Interposer une nouvelle couche de traitement (cf. *Wrapper*)
 - ◆ Changer (conditionnellement) la destination de l'appel
 - ◆ **Contraintes**
 - ◆ Les programmes client et serveur ne doivent pas être modifiés
 - ◆ Les services peuvent être ajoutés ou supprimés dynamiquement
- Solutions**
 - ◆ **Créer des objets d'interposition (statiquement ou dynamiquement). Ces objets**
 - ◆ interceptent les appels (et/ou les retours) et insèrent des traitements spécifiques, éventuellement fondés sur une analyse du contenu
 - ◆ peuvent rediriger l'appel vers une cible différente
 - ◆ peuvent utiliser des appels en retour

© 2003-2007, S. Krakowiak ICAR'06 28





Comparaison des patrons de base

- **Wrapper vs. Proxy**
 - ◆ **Wrapper et Proxy ont une structure similaire**
 - ◇ Proxy préserve l'interface ; Wrapper transforme l'interface
 - ◇ Proxy utilise (pas toujours) l'accès à distance; Wrapper est en général local
- **Wrapper vs. Interceptor**
 - ◆ **Wrapper et Interceptor ont une fonction similaire**
 - ◇ Wrapper transforme l'interface
 - ◇ Interceptor transforme la fonction (peut même complètement détourner l'appel de la cible initiale)
- **Proxy vs. Interceptor**
 - ◆ **Proxy est une forme simplifiée d'Interceptor**
 - ◇ on peut rajouter un intercepteur à un mandataire (*smart proxy*)

© 2003-2007, S. Krakowiak ICAR'06 31

Mise en œuvre des patrons de base

- **Génération automatique**
 - ◆ À partir d'une description déclarative

```

graph LR
  IDL --> Box1[ ]
  Box1 --> proxy
  IDL1 --> Box2[ ]
  IDL2 --> Box2
  Box2 --> wrapper
  
```

- **Optimisation**
 - ◆ **Éliminer les indirections, source d'inefficacité à l'exécution**
 - ◇ Court-circuit des chaînes d'indirection
 - ◇ Injection de code (insertion du code engendré dans le code de l'application)
 - ◇ Génération de code de bas niveau (ex. bytecode Java)
 - ◇ Techniques réversibles (pour adaptation)

© 2003-2007, S. Krakowiak ICAR'06 32

Canevas logiciels (Frameworks)

- **Définition**
 - ◆ Un canevas est un "squelette" de programme qui peut être réutilisé (et adapté) pour une famille d'applications
 - ◆ Il met en œuvre un modèle (pas toujours explicite)
 - ◆ Dans les langages à objets : un canevas comprend
 - ◇ Un ensemble de classes (souvent abstraites) devant être adaptées (par ex. par surcharge) à des environnements et contraintes spécifiques
 - ◇ Un ensemble de règles d'usage pour ces classes
- **Patrons et canevas**
 - ◆ Ce sont deux techniques de réutilisation
 - ◆ Les patrons réutilisent un schéma de conception ; les canevas réutilisent du code
 - ◆ Un canevas implémente en général plusieurs patrons

© 2003-2007, S. Krakowiak ICAR'06 33

Schémas de décomposition

- Objectifs
 - ◆ Faciliter la construction
 - ◇ La structure reflète la démarche de conception
 - ◇ Les interfaces et les dépendances sont mises en évidence
 - ◆ Faciliter l'évolution
 - ◇ Principe d'encapsulation
 - ◇ Échange standard
- Exemples
 - ◆ Structures multi-niveaux
 - ◇ Décomposition "verticale" ou "horizontale"
 - ◆ Canevas pour insertion de composants

© 2003-2007, S. Krakowiak ICAR'06 34

Décomposition en couches

- Hiérarchie de "machines abstraites"
 - ◆ La réalisation des niveaux < i est invisible au niveau i
 - ◆ Exemple : machines virtuelles (OS multiples, JVM, etc.)

Protocoles de communication

appel ascendant (upcall)

© 2003-2007, S. Krakowiak ICAR'06 35

Décomposition "horizontale"

- Exemple : évolution du schéma client-serveur

(a) application gestion de données

(b) interface utilisateur application gestion de données

(c) : 3-tier

© 2003-2007, S. Krakowiak ICAR'06 36

Exemple de canevas global (1)

Architecture de micro-noyau

© 2003-2007, S. Krakowiak ICAR'06 37

Exemple de canevas global (2)

Architecture d'un canevas pour composants (middle tier)

© 2003-2007, S. Krakowiak ICAR'06 38

Canevas de base et personnalités

Motivation : réutilisation de mécanismes génériques

- ◆ Un canevas de base réalise les entités définies par un modèle abstrait
 - ◆ Critères : générique, modulaire, composable, adaptable
- ◆ Des "personnalités" utilisent les APIs du canevas de base (y compris appels en retour) pour réaliser des mises en œuvres concrètes du modèle
- ◆ Avantages : réutilisation, unité conceptuelle, facilité de (re)configuration
- ◆ Difficulté : efficacité

Exemples

Unix	Autre OS	Java RMI	CORBA	EJB	CCM
micronoyau		ORB générique		Noyau à composants	

© 2003-2007, S. Krakowiak ICAR'06 39



Adaptation des systèmes et applications



- **Qu'est ce que l'adaptation ?**
 - ◆ Changement de la structure et/ou des fonctions d'une application
 - ◆ Adaptation dynamique : réalisée sans arrêt de l'application
- **Pourquoi l'adaptation ?**
 - ◆ **Pour répondre à l'évolution**
 - ◆ Des besoins : nouvelles fonctions, nouvelles qualités
 - ◆ De l'environnement d'exécution (capacités du matériel, mobilité, conditions de communications, perturbations et défaillances, etc.)
- **Comment?**
 - ◆ Principe : système réflexif (fournit une représentation de son propre fonctionnement, pour l'inspecter ou le modifier)
- **Techniques**
 - ◆ Techniques ad hoc (intercepteurs)
 - ◆ Protocoles à méta-objets (MOP)
 - ◆ Programmation par aspects (AOP)

Ces techniques permettent de réaliser des actionneurs dans un système réactif. cf intergiciel Jade
