

Méthodes pour l'informatisation - compléments

Diagrammes de Composants,
Paquetages et Déploiements

Christine Plumejeaud

Doctorante Informatique UJF

CNAM - centre de Grenoble

Plan

1. Diagramme de composants
2. Diagramme de paquetages
3. Diagramme de déploiement

Diagramme de composants

- Il décrit l'architecture physique d'une application en termes de modules : fichiers sources, bibliothèques exécutables, etc.
- C'est une vue du développement, utile pour gérer les différentes couches de l'architecture du système.

Composant

- Un *composant* est un élément encapsulé, réutilisable et remplaçable du logiciel. Comme des briques de construction, ils se combinent pour construire le logiciel.
- Taille : variable (de classe à sous-système)
- Exemples
 - analyseur XML (*XMLparser*)
 - traducteur de langues (*ResourceBundle*)
 - gestionnaire de connexion sur une base de données (*JDBCdriver*).

Combinaison et encapsulation des composants

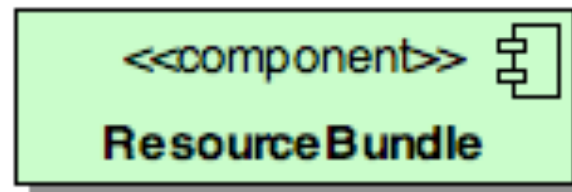
- Un composant peut contenir ou utiliser d'autres composants. Par exemple :

Le module d'acquisition de données d'un SIG peut inclure :

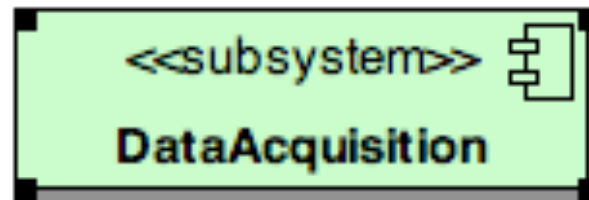
- le composant ExcelParser (pour analyser les fichiers fournis au format XLS)
- un gestionnaire de connexion sur la Base de Données (JDBCdriver) pour l'enregistrement des données.
- Ils doivent être *faiblement couplés* : leur accès se fait via des Interfaces (qui séparent le comportement de l'implémentation)

Notation du composant

1. Les composants simples



2. Les composants complexes



Interfaces du composants

Les composants interagissent à travers des interfaces, pour favoriser leur faible couplage.

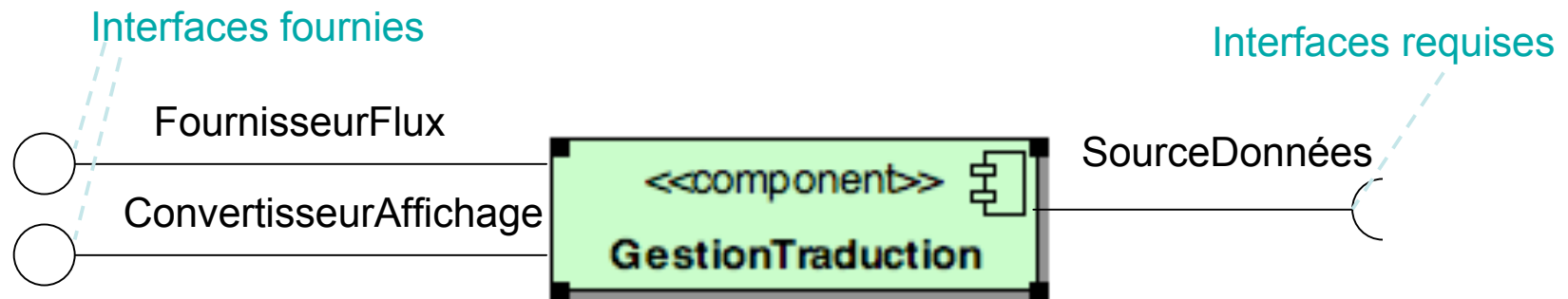
1. Les *interfaces fournies* : déclarent les services proposés (le composant réalise ces interfaces).
2. Les *interfaces requises* : déclarent les services dont le composant a besoin

Notation des interfaces 1/3

notation à rotule

Le composant *GestionTraduction* :

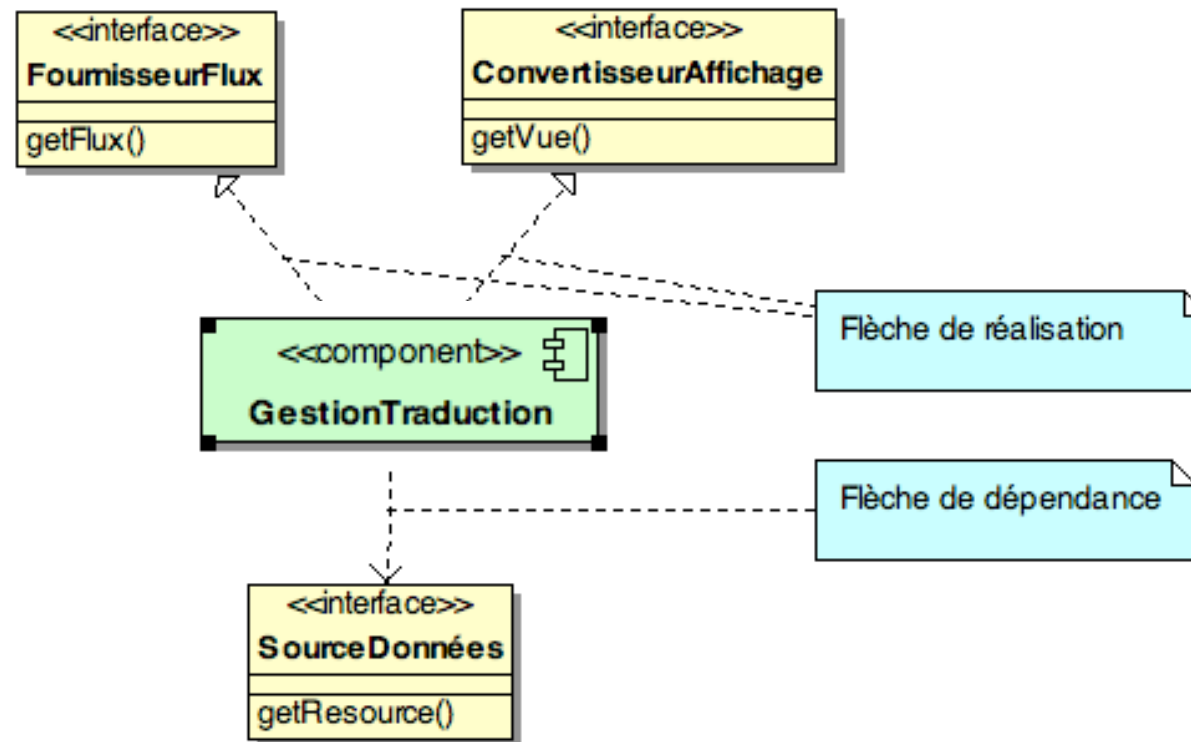
- fournit les interfaces *FournisseurFlux*, et *ConvertisseurAffichage*
- requiert l'interface *SourceTraduction*.



Notation des interfaces 2/3

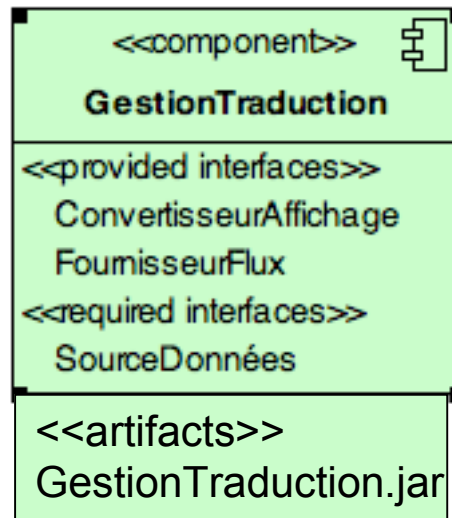
notation de stéréotype

Utile pour montrer les opérations des interfaces



Notation des interfaces 3/3

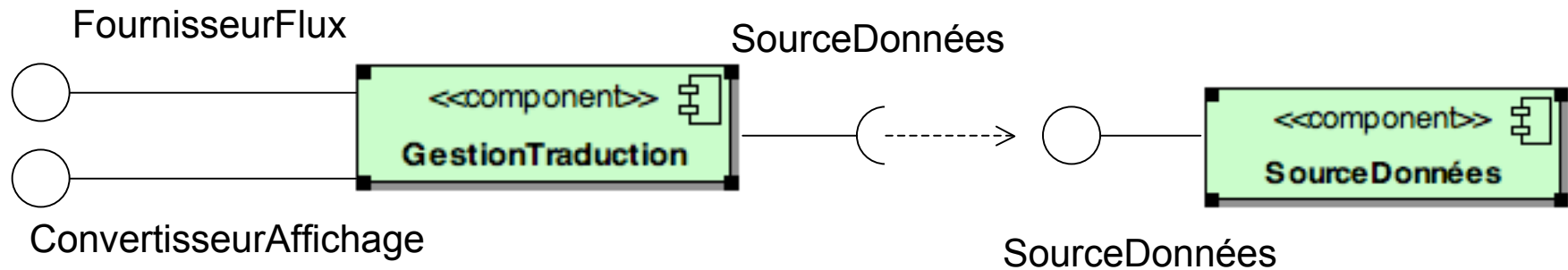
énumération des interfaces



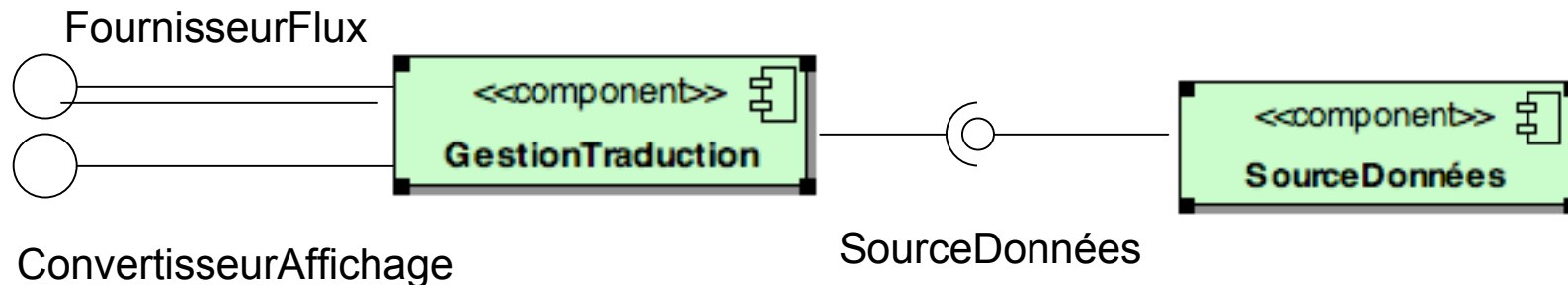
`<< artifacts >>` liste les fichiers physiques implémentant le composant.

La coopération des composants (1/2)

- Première notation

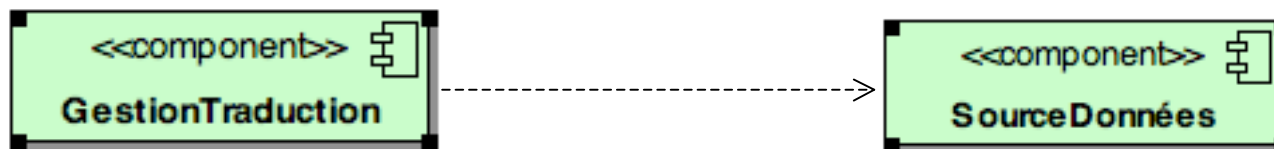


- Connecteurs d'assemblage



La coopération des composants (2/2)

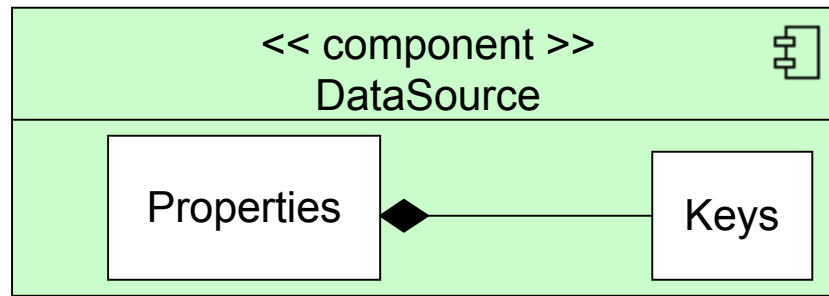
- Seconde notation



- Cette notation plus compacte autorise des vues de haut niveau des dépendances entre les composants. Utile pour le déploiement.

Classes réalisant un composant

1. Première notation



2. Seconde notation

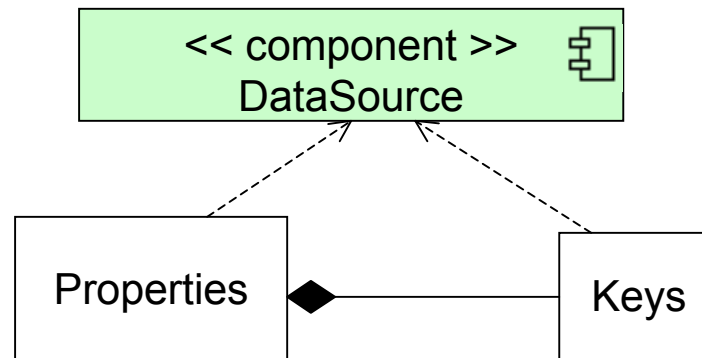


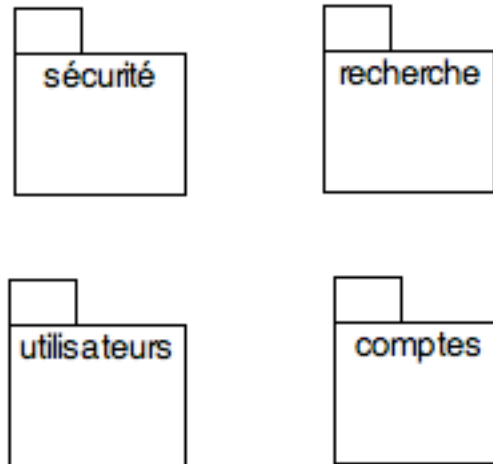
Diagramme de paquetages

- Un paquetage permet de regrouper des objets (classes ou diagrammes) de façon logique, et permet d'organiser l'ensemble des fichiers sources.
- Les diagramme de paquetages servent à illustrer les dépendances entre groupes d'objets (ou paquetage). La compréhension de ces dépendances est cruciale pour la stabilité du logiciel.

Paquetage

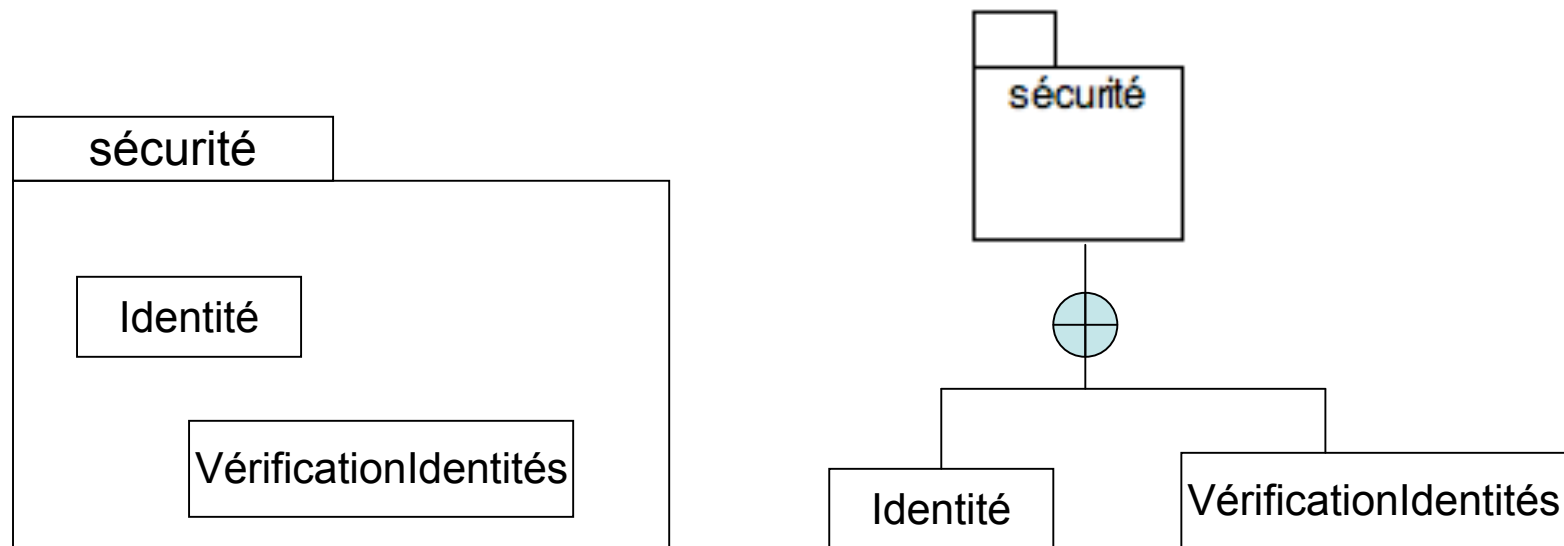
Un paquetage est symbolisé par un rectangle avec un onglet, et son nom est écrit dans le rectangle.

Exemple : Pour un Content Management System (CMS), on peut regrouper les classes en 4 paquetages :



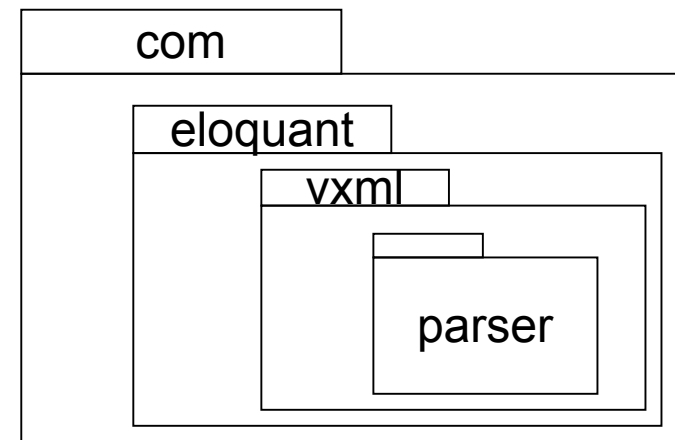
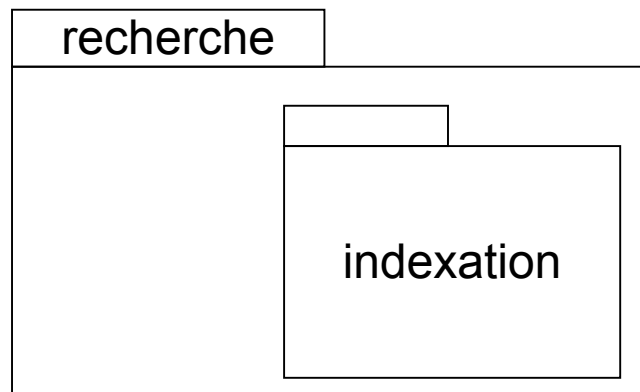
Contenu d'un paquetage

Un paquetage regroupe des éléments UML comme des classes.



Composition de paquetages

Les paquetages peuvent en contenir d'autres : *indexation* est un sous-paquetage de *recherche*



Notation: **nom du paquetage :: nom de classe.**

Exemple : si la classe *VxmlParser* appartient au paquetage *parser*, *com::eloquant::vxml::VxmlParser*

Espace de nommage

1. *L'espace de nommage* d'une classe est son paquetage d'appartenance. Les classes du même paquetage peuvent faire référence les unes aux autres sans employer les noms complets.
2. Deux paquetages différents peuvent contenir une classe de même nom.

sécurité::Identité

utilitaires::Identité

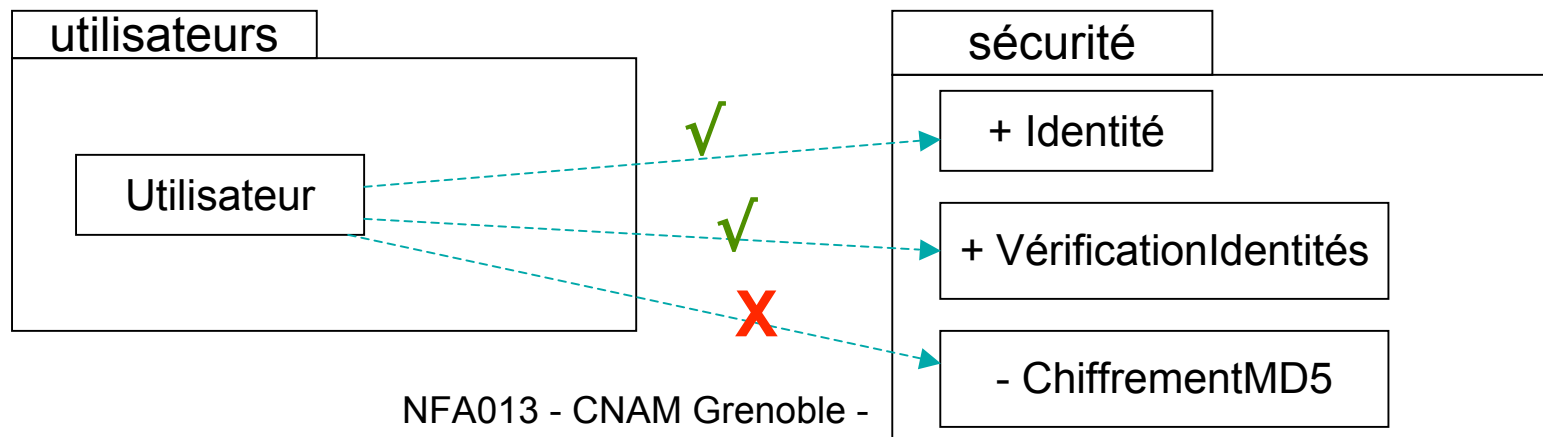
3. Une classe utilisant deux classes de même nom doit les référencer par leur nom complet.

Visibilité des classes entre paquetsages

On peut restreindre la visibilité des classes d'un paquetage :

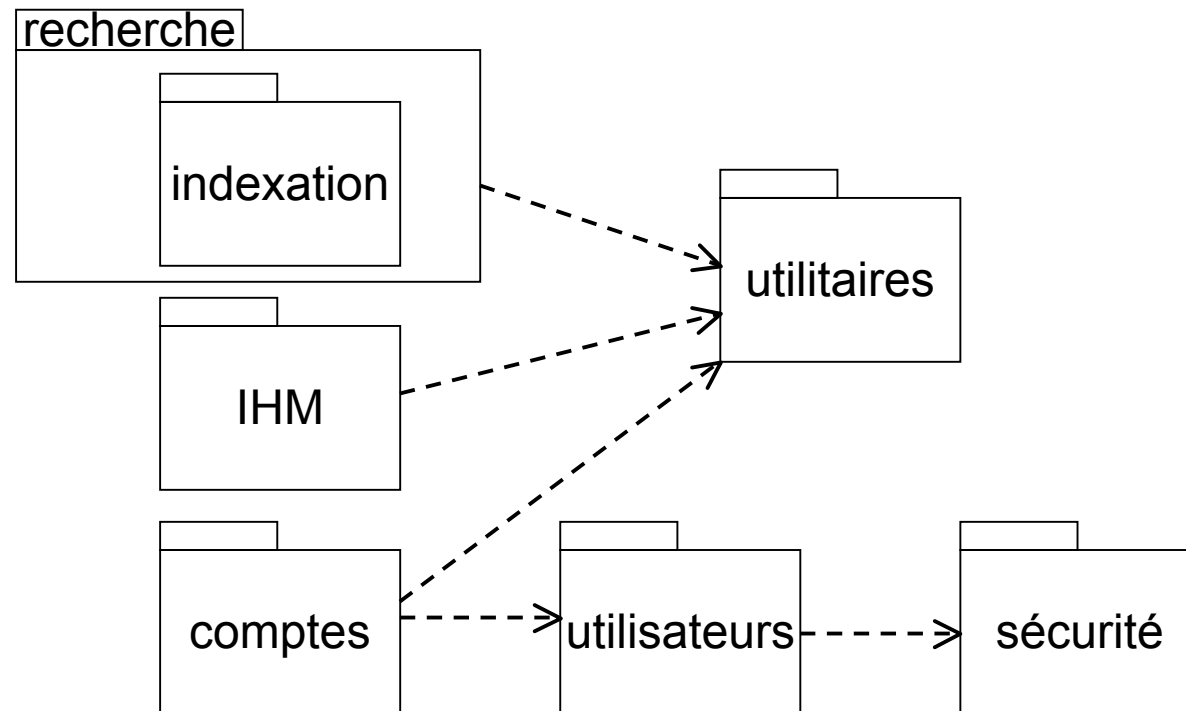
Public (+) : les classes des paquetsages extérieurs peuvent y accéder

Private (-): les classes ne sont disponibles que par les classes du même paquetage.



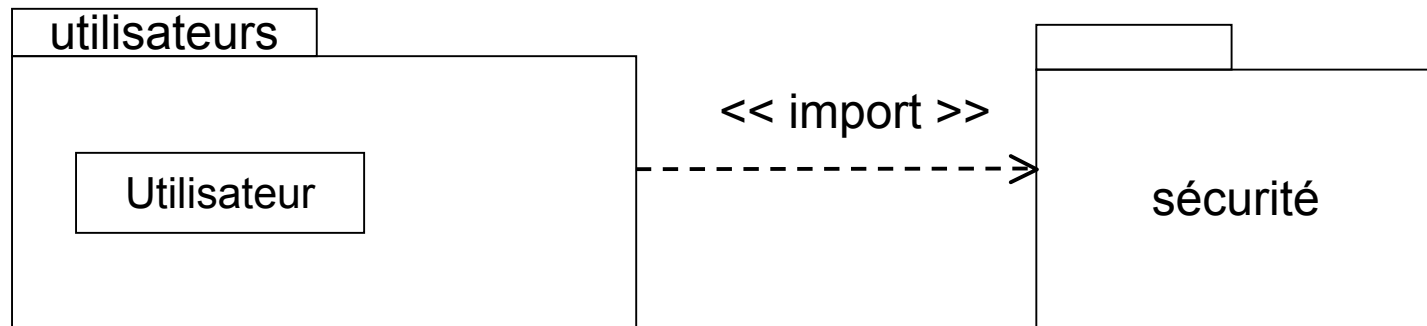
Dépendance entre les paquetages

Les flèches en pointillés marquent la dépendance d'un paquetage vers un autre.



Importation de paquetage

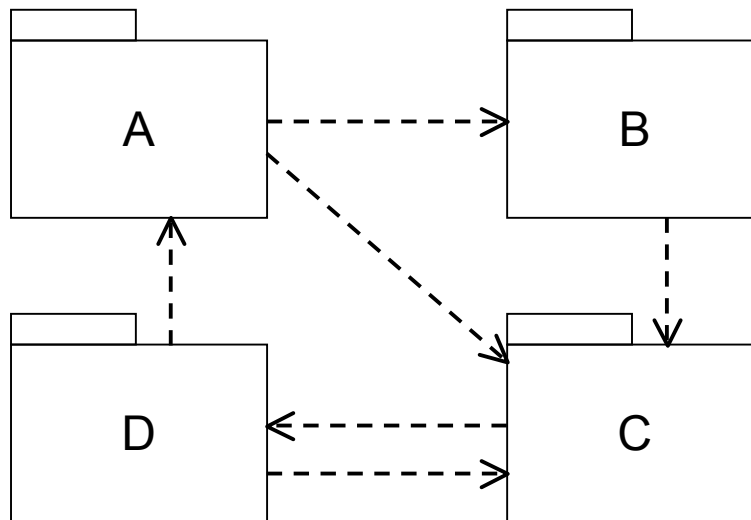
L'importation de paquetage permet d'utiliser les classes (publiques seulement) d'un paquetage A dans un autre paquetage B, sans référencer par leur nom complet les classe de B.



Gérer les dépendances entre paquetages

Les dépendances complexes entre paquetages peuvent fragiliser le logiciel :
il faut éviter les cycles.

Exemple de ce qu'il NE FAUT PAS faire :



Pour la stabilité :
Les paquetages ne doivent dépendre que de peu de paquetages, eux-mêmes stables

Diagrammes de déploiement

- Le diagramme de déploiement décrit la disposition physique des matériels et la répartition des composants sur ces matériels.

Avec le développement de systèmes répartis complexes, (banques, télécoms, etc.), comprenant par exemple des modules embarqués dans des plateformes mobiles (GPS) ou s'exécutant sur des gros serveurs de données(TeraBytes), **la compréhension, et l'automatisation du déploiement des applications est cruciale.**

Déployer un système simple

Le logiciel est livré sous la forme d'un seul fichier exécutable qui s'exécute sur un ordinateur.

- L'ordinateur est représenté par un nœud. Il est marqué par << device >>
- Le fichier logiciel physique est modélisé par un artefact, noté << artifact >>

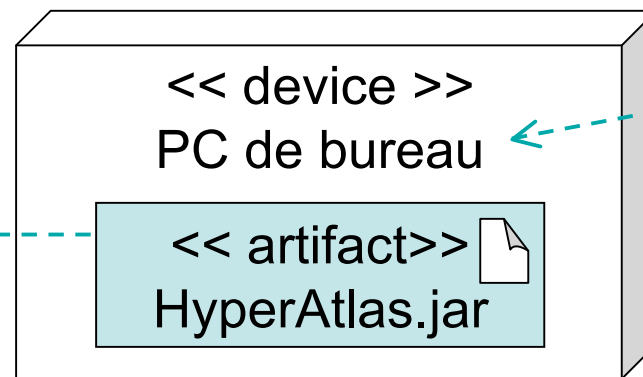
Exécutable : .exe, .jar

Bibliothèque : .dll, .jar

Source : .cpp, .java

Configuration :
.properties, .xml, .txt

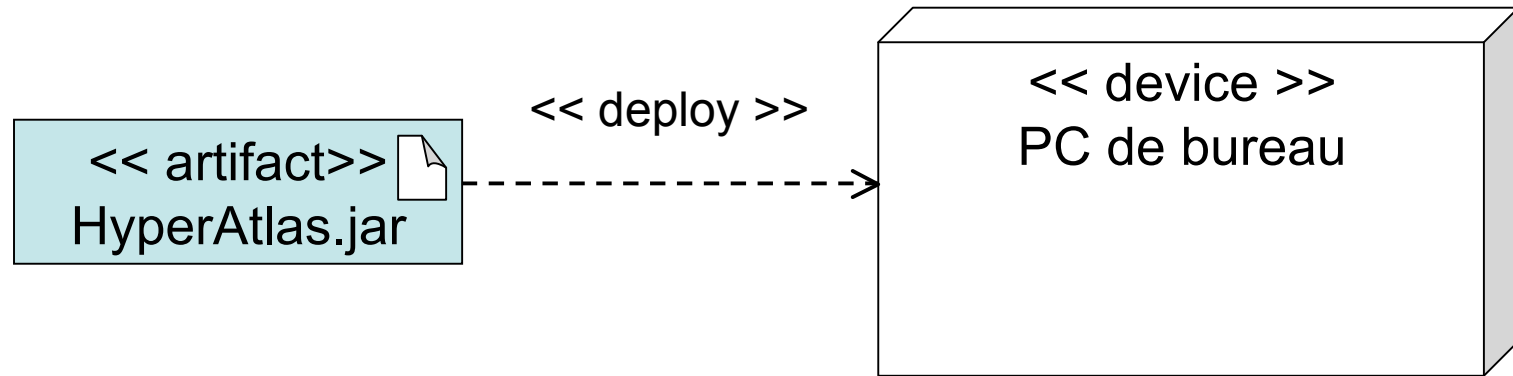
2008



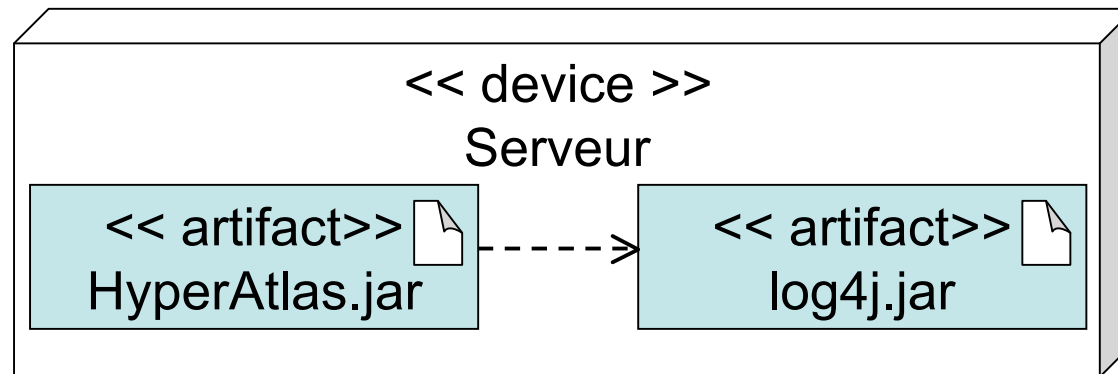
Cette description
peut être plus ou
moins précise :

cela dépend du
public à qui
s'adresse le
diagramme

Autre notations du déploiement

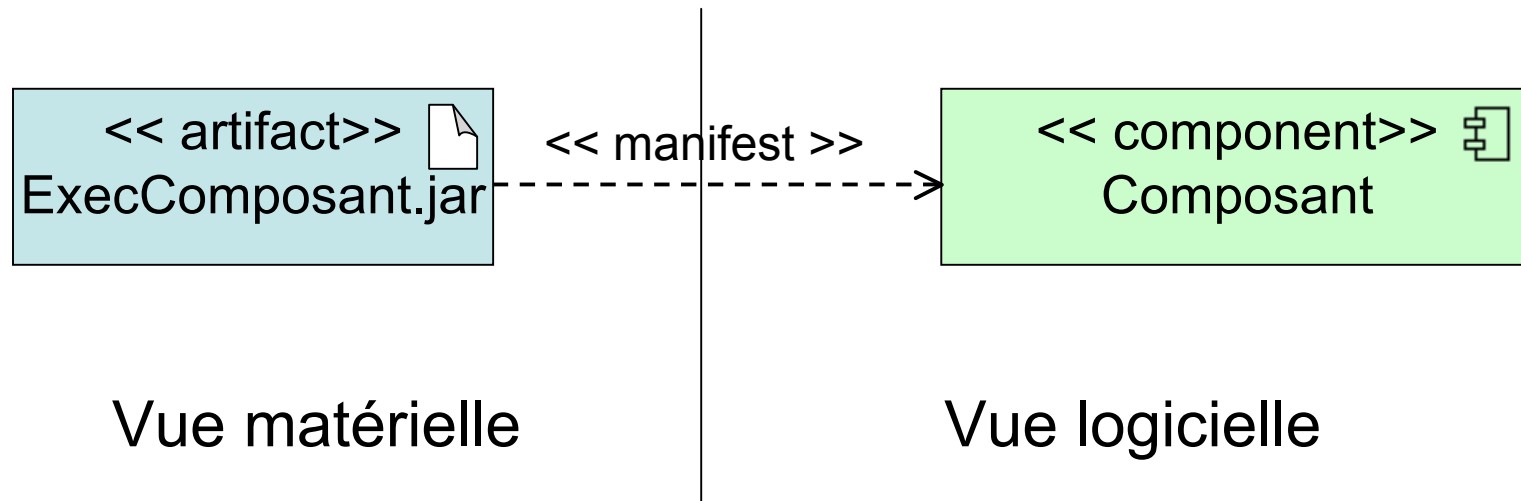


Montrer les dépendances entre artefacts



Composant et artefact

Avant UML2, le composant était assimilé à l'artefact. Dans UML2, l'artefact constitue le manifeste d'un composant.

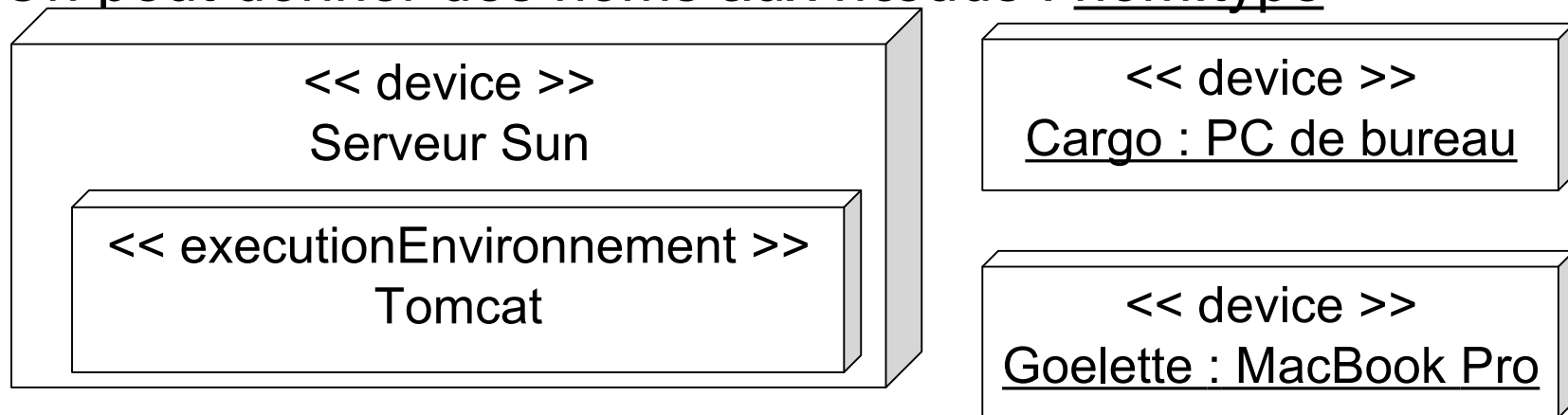


Les noeuds

Un nœud est un élément matériel ou logiciel offrant un environnement d'exécution aux artefacts

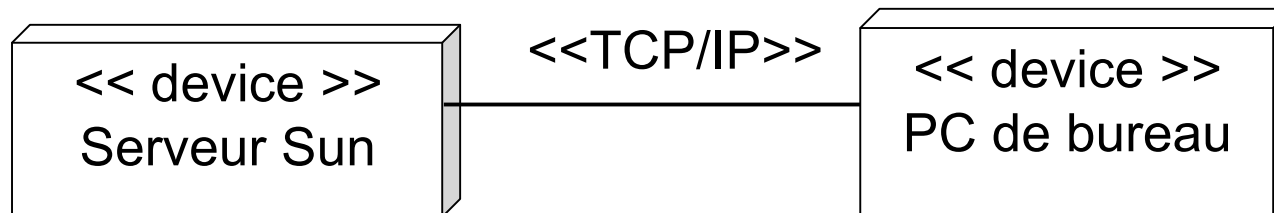
- Nœud matériel : serveur, PC de bureau, disque dur, mobile
- Nœud d'environnement d'exécution : système d'exploitation (Windows, Linux, Unix, mac OS), serveur Web (apache, tomcat), container J2EE, serveur d'application.

On peut donner des noms aux nœuds : nom:type



Communication entre les noeuds

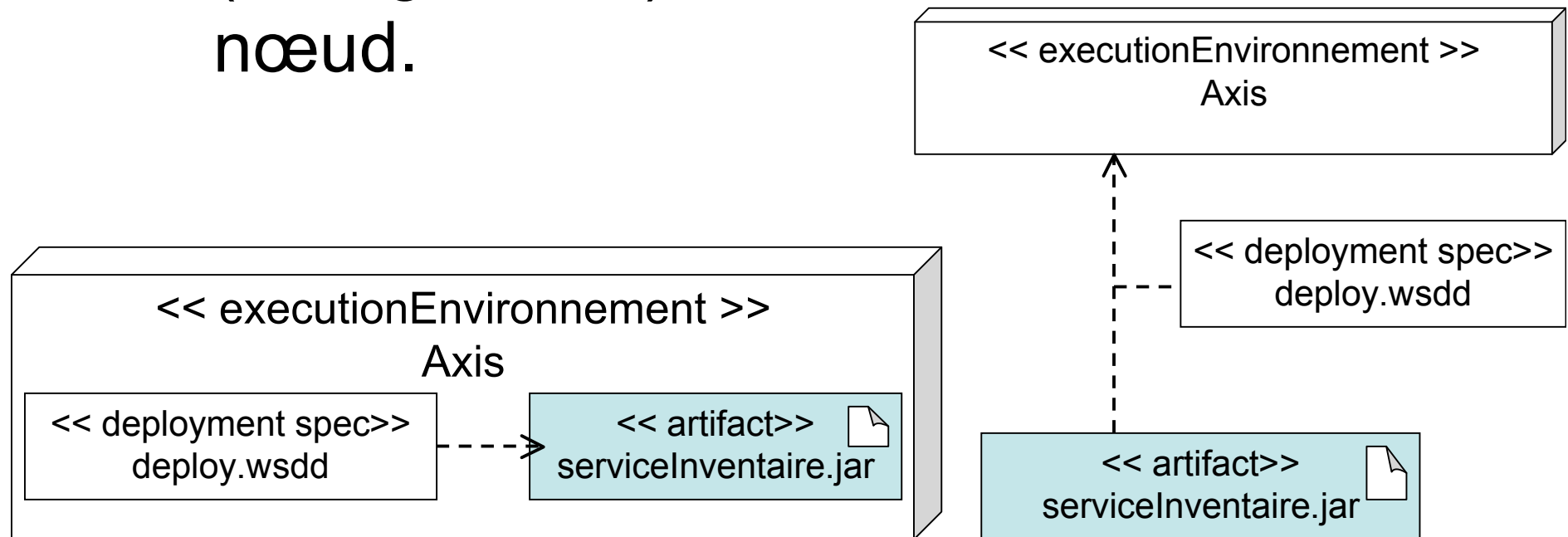
Les modules d'une application distribuée sur différents nœuds ont besoin de communiquer entre eux. On utilise un **chemin de communication** (ligne pleine) et on note par un stéréotype le **type de communication**.



Le stéréotype de communication doit avoir le niveau de communication le plus élevé possible : RMI ou TCP/IP ? SOAP ou HTTP ?

Spécifications de déploiement

Une *spécification de déploiement* est un artefact spécial décrivant le déploiement (configuration) d'un autre artefact sur un nœud.



Exemple de diagramme pour un SI

