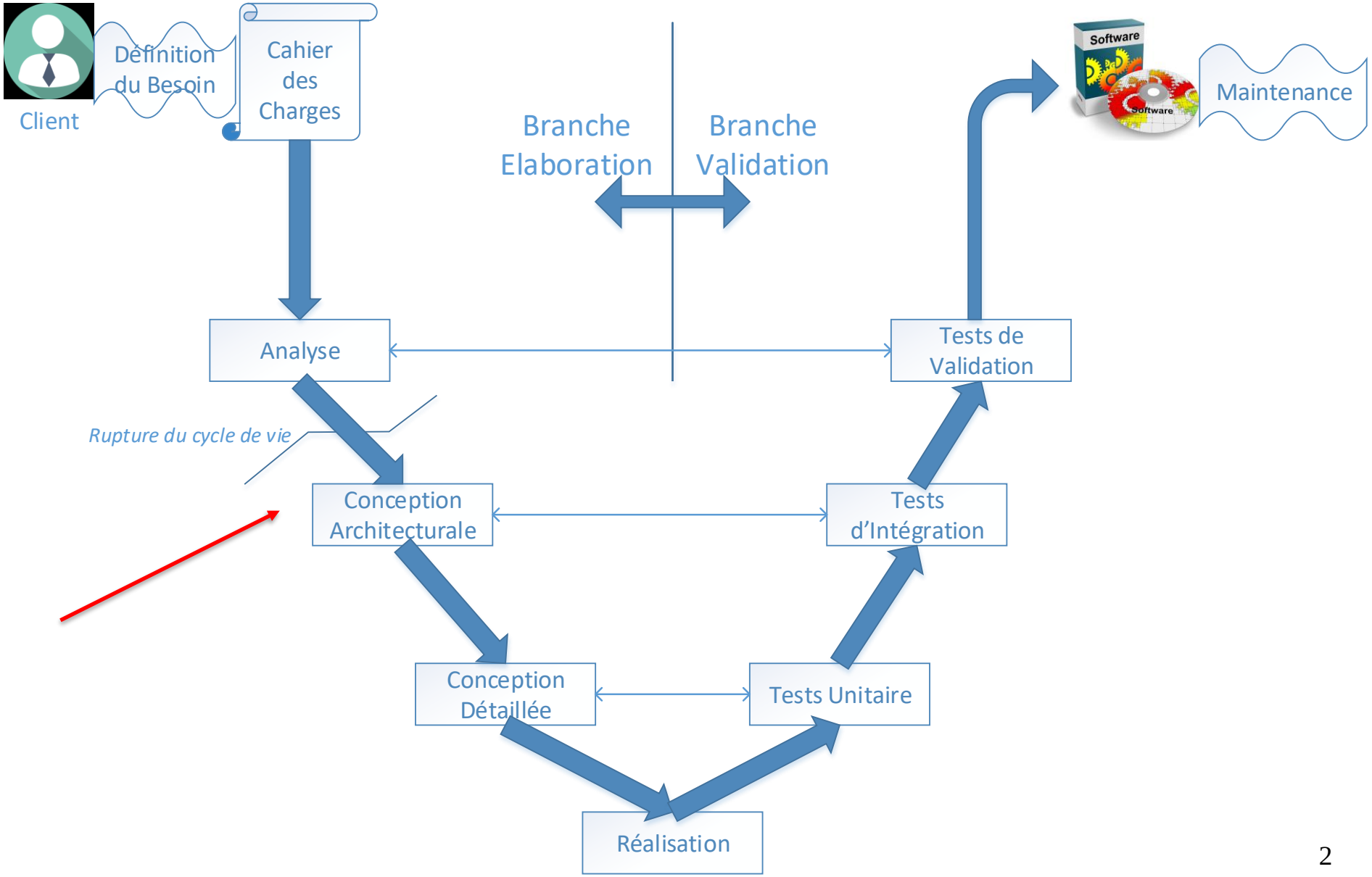
Ingénierie du Logiciel
Master 1 Informatique – 4I502

Cours 5 : Transition, Conception Architecturale

Yann Thierry-Mieg
Yann.Thierry-Mieg@lip6.fr

Rappel


Client
Définition du Besoin
Cahier des Charges
Branche Elaboration
Branche Validation
Software
Maintenance
Analyse
Tests de Validation
Rupture du cycle de vie
Conception Architecturale
Tests d'Intégration
Conception Détaillée
Tests Unitaire
Réalisation

Transition Analyse->Conception

Analyse : Bilan

- Spécifier le système tel que perçu de l'extérieur
- Spécifier les interactions et l'intégration de l'application dans son environnement métier
- Spécifier : ce que **fait** le système de façon **perceptible**
- Modélisation du **métier** du client
- Isolation de la description du comportement **observable** vis-à-vis de la complexité d'une *solution*

- E.g. Use case : Téléphoner

Conception : Comment ?

- Spécifier une solution particulière qui réalise les comportements observables
- Spécifier les agencements internes de la solution
  - une résolution basée sur la coopération de *composants* de l'application
- Spécifier *comment* le système *fait* pour réaliser les observables, et de façon plus générale ce dont il est capable
  - Conception plus générale que ce qu'exige l'analyse est OK

Conception Architecturale

- Modélisation d'artefacts logiciels ("notre" métier)
- Objectifs :
  - Solution générales :
    - *Maintenabilité*, Extensibilité, Portabilité ou Gestion de l'Hétérogénéité, …
  - Solution par morceaux ou *composants*
    - Réutilisabilité de composants
    - Diviser pour régner pour maitriser la complexité

« Rupture » du cycle de vie logiciel

Un lien de *raffinement* entre l'Analyse et la Conception ?

- « Break » : NON, les préoccupations sont fondamentalement différentes, même si la spéc d'analyse est essentielle, sa nature diffère trop de la conception pour parler de « raffinement ».
- « Seamlessness » : OUI, il doit exister un (ou des) raffinements progressif des exigences jusqu'à un niveau de précision suffisant pour que du code qui répondrait à ces exigences en découle directement.
  - E.g. Approche B.
- Dans l'UE : on ne tranche pas le débat, mais on va essayer de se resservir quand même des modèles d'analyse.

Modèles produits en Analyse

- Cas d'utilisation
  - Pilotage du développement transverse, need driven
- Séquences d'analyse => opérations du « système »
  - Ensemble de fonctions que le système doit réaliser
  - Réutilisable en conception
  - Remettre en cause tout ce qui est privé
- Classes métier
  - Structure des données stockées par le système
  - Réutilisable, mais pas tel quel, réorganiser les données pour organiser des traitements par morceaux

Décomposition du système

« Architecture »

« Gros grain » : considérer un assemblage de composants

- Composant : constituant du système sachant traiter des sous-problèmes
- Dépendances *Fonctionnelles* entre composants
  - Dépendre de services offerts, mais pas de la façon dont ils sont réalisés
- Diviser pour régner
  - *Maitriser* la complexité (sans la réduire)
  - Vision hiérarchique du problème
  - Frontières délimitées entre sous problèmes indépendants
  - Intégration de l'existant
  - Parallélisme du développement des composants

(Dépendance Fonctionnelle : DP Factory)

- Forme, Carré
- Evolution vers classe Rectangle
- Maitrise des répercussions chez les clients
- Centralisation de l'opération de construction dans un objet : Factory
- Effet :
  - Dépendances fonctionnelles pures au lieu de dépendances structurelles

Description d'Architecture Logicielle

- Architectural Description Langage (Medvidovic'00)
  - Décrire *séparément :*
  - Connecteur : Définition fonctionnelle à travers des signatures des services offerts par le composant
  - Composant : Unité de traitement, possède des connecteurs requis ou offerts
  - Topologie : Un assemblage ou instanciation des composants dans une configuration *particulière*, plusieurs instanciations des même composants sont possibles

Qualités d'une Architecture Logicielle

- Non redondance
  - Eviter que deux composants aient les même responsabilités
- Existence de composants bout de chaine
  - Donc plus facilement réutilisables
- Equilibre entre les responsabilités des composants
- Maîtrise des éventuels cycles de dépendances
  - DP Factory, observer, …
- Elégance vs utilisabilité pratique
- Définir de bonnes API pour des sous-systèmes

Diagrammes de Composant UML

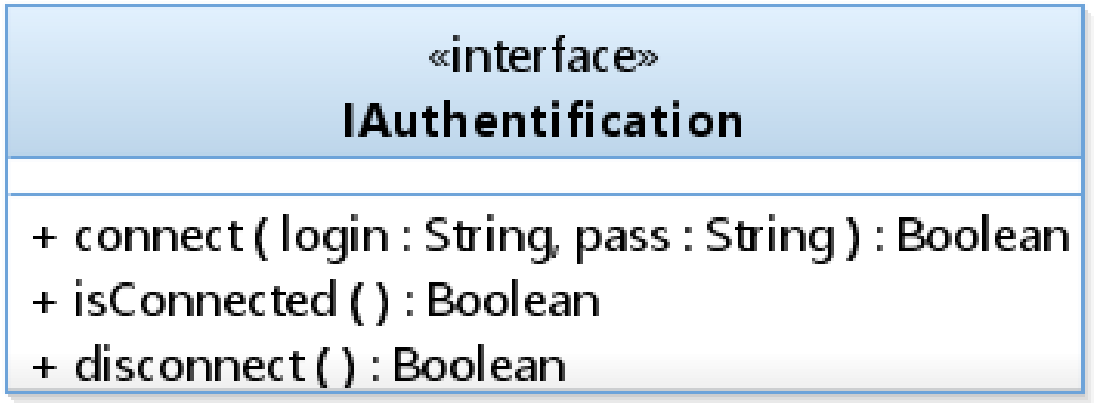
Deux diagrammes complémentaires

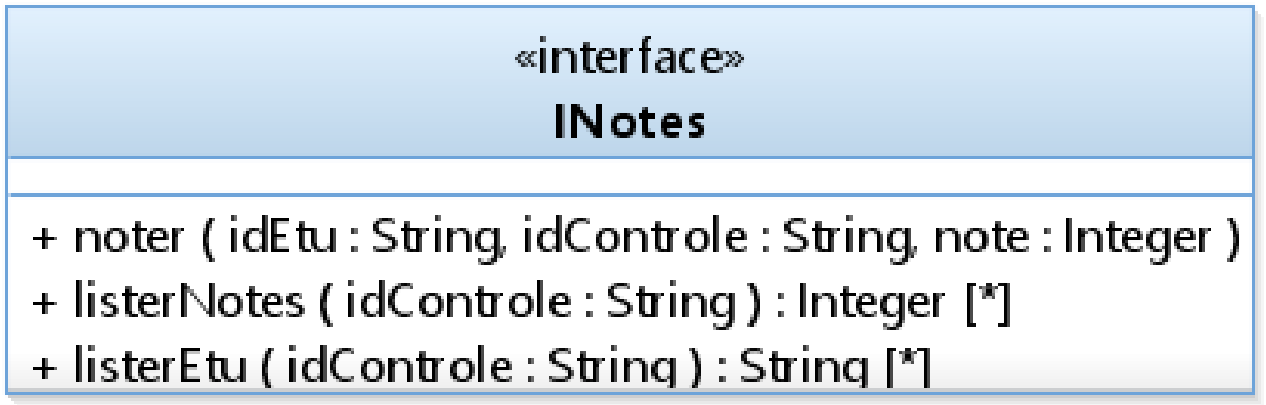
- Diagrammes de composant :
  - Définition des connecteurs :
    - des interfaces UML ! Fonctionnel pur.
    - Sens UML et OO (Java) identique
  - Définition des composants :
    - offrent et/ou utilisent des interfaces
- Diagramme de structure interne
  - Est au diagramme de composant ce que le diagramme des objets est pour les classes
  - Représente des *instances* de composants (*Part*)
  - Configurés ou connectés d'une manière particulière (*Connecteur*)

Diagramme de Composants

- Deux notions /entités :
  - composant, interface
- Deux liens possibles entre un composant et une interface
  - Utilise : invoque au moins une méthode déclarée dans l'interface
  - Réalise : offre les service déclarés dans l'interface
- Pas de lien direct entre composants

Interface

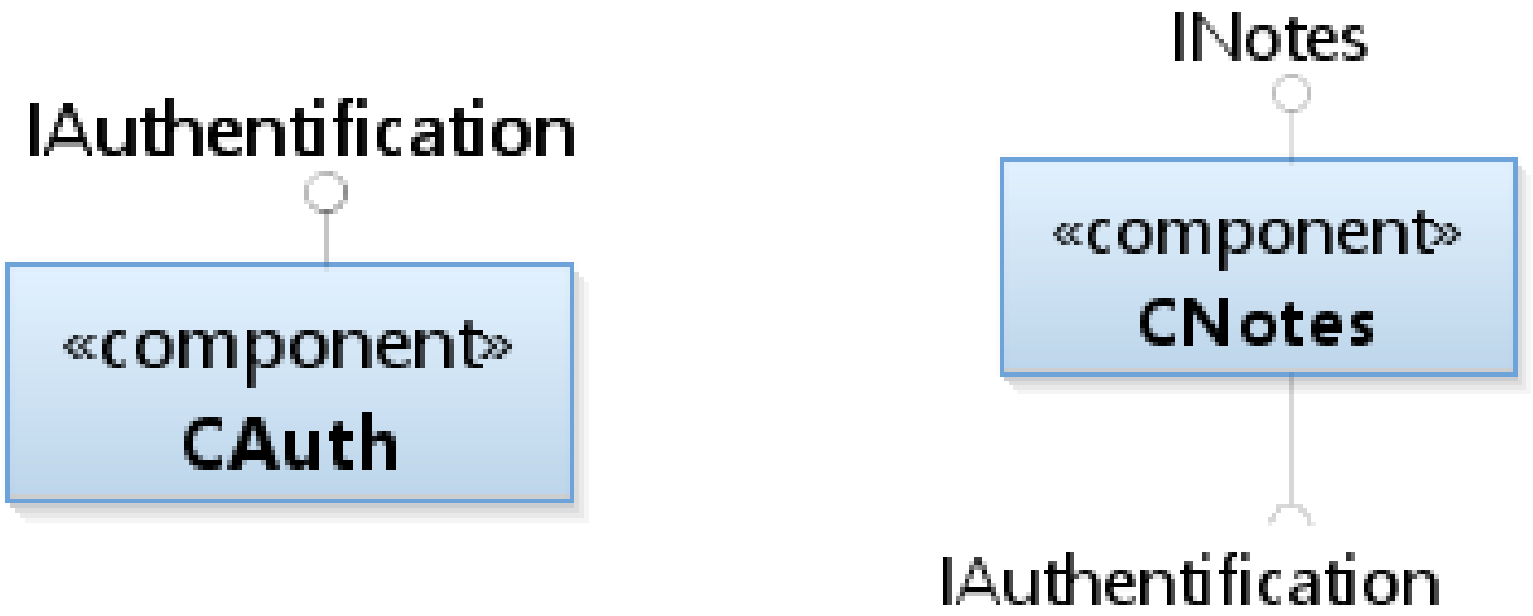




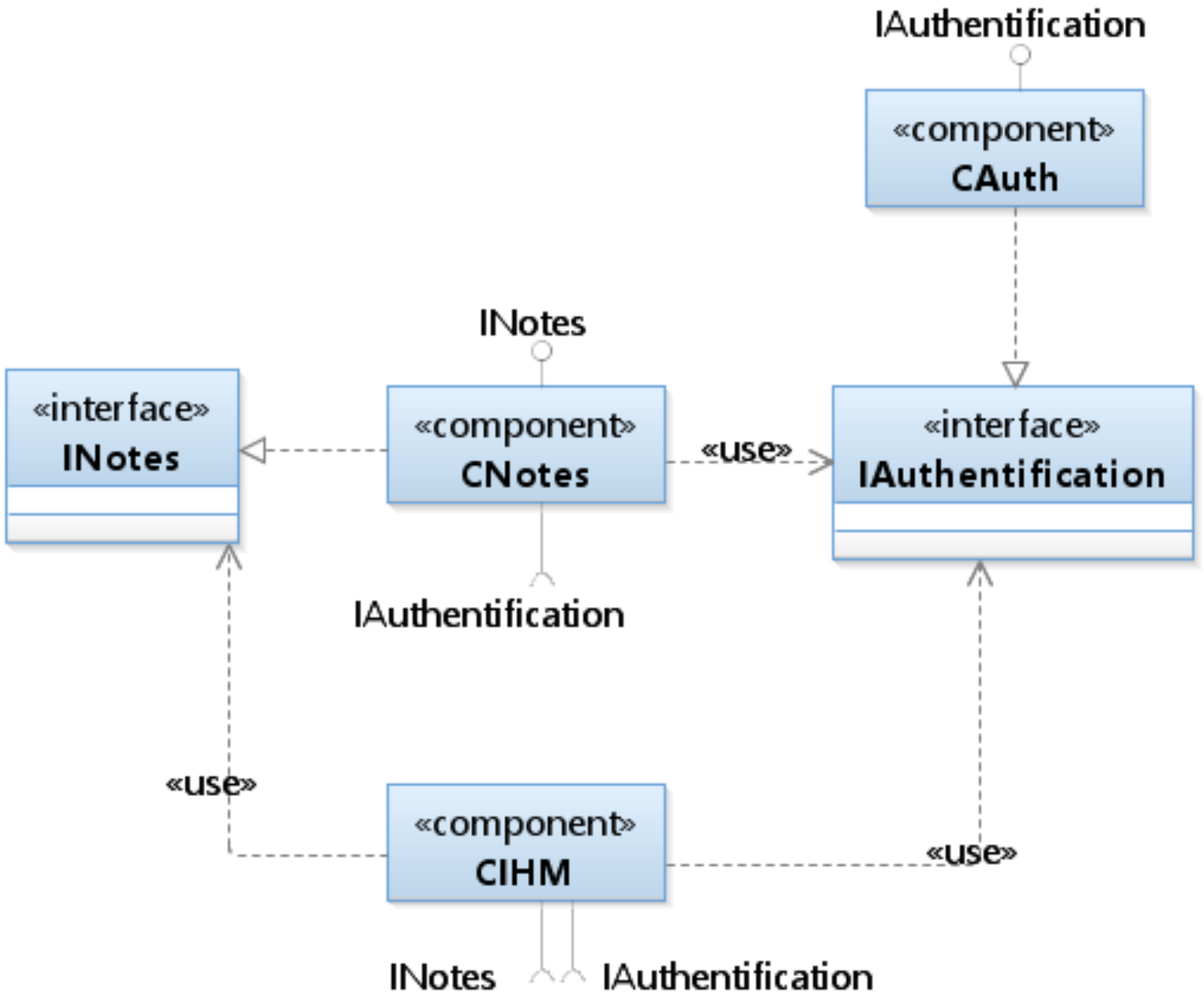
- Interfaces UML, sens idem Java
  - Que des opérations, abstraites

Composant

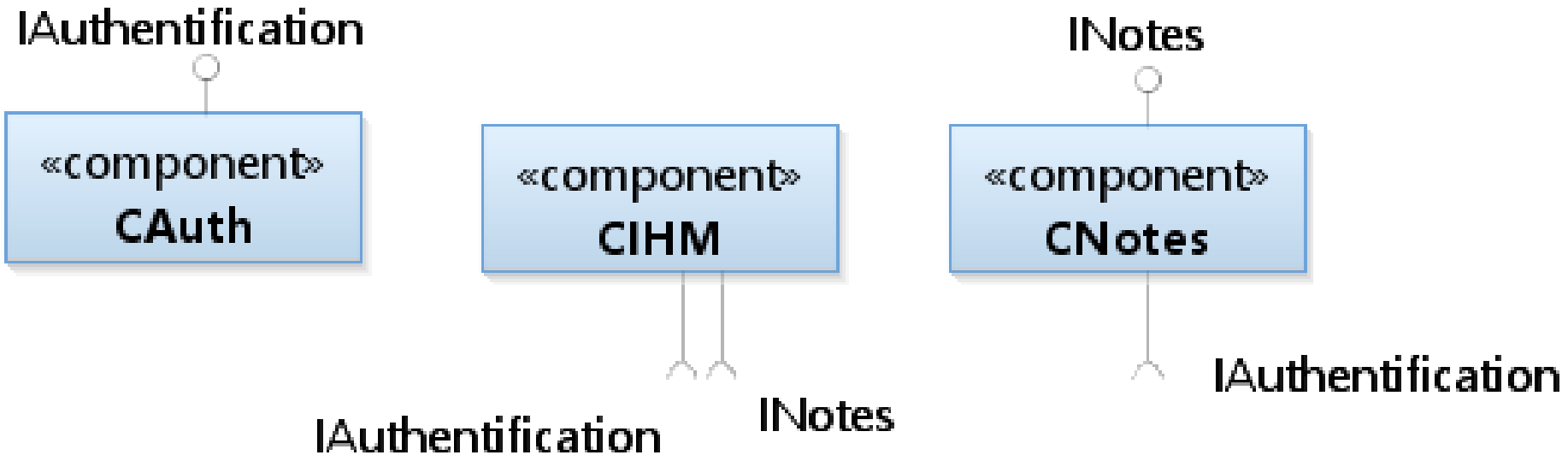
- Boite noire :
  - On ne voit que les connecteurs
  - Réalisations fonctionnelles (lollipop)
  - Dépendances fonctionnelles (socket)
  - Connecteurs typés, mais pas chiffrés (cardinalités ?)



Vue liaisons



Vue Externe

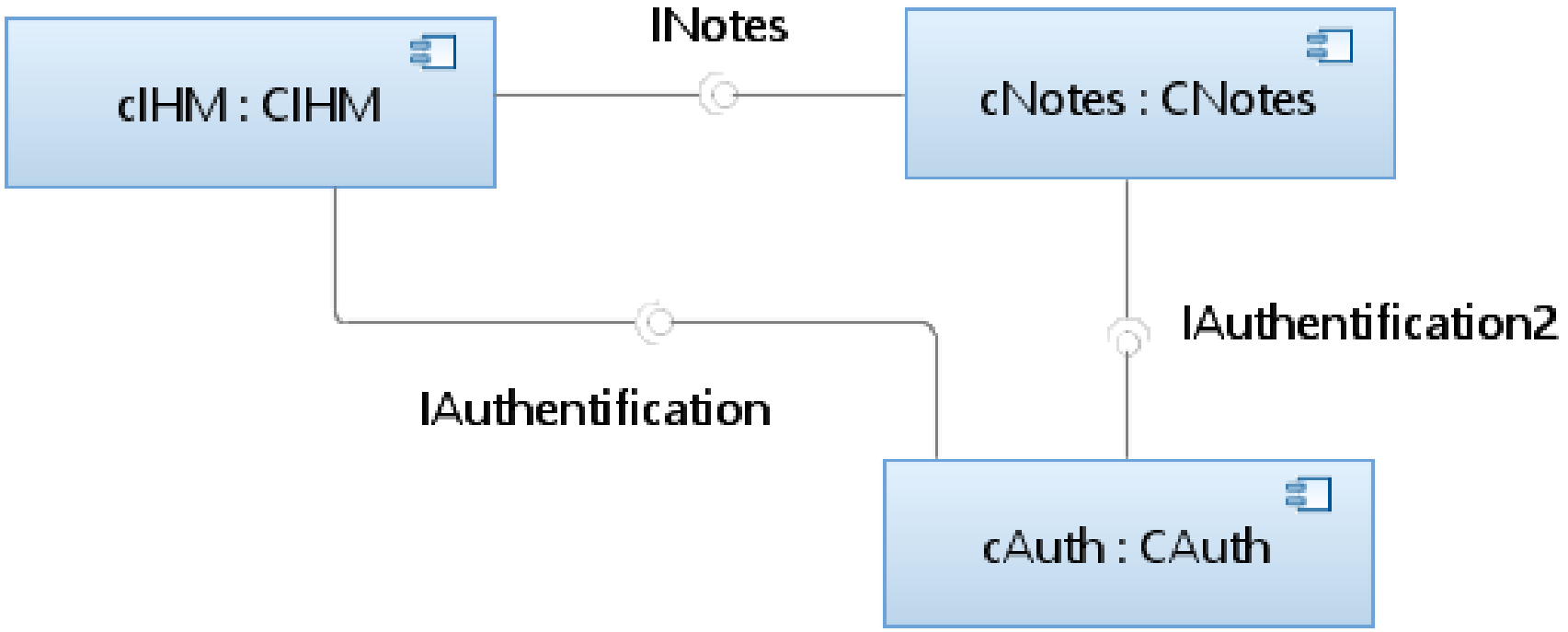


- Strictement équivalent au précédent
- Collections de composants sur étagère
  - Bout de chaine => Réutilisable ++
  - Certains composants n'ont pas d'interface offerte
    - Au niveau logiciel du moins…

Diagramme de structure interne

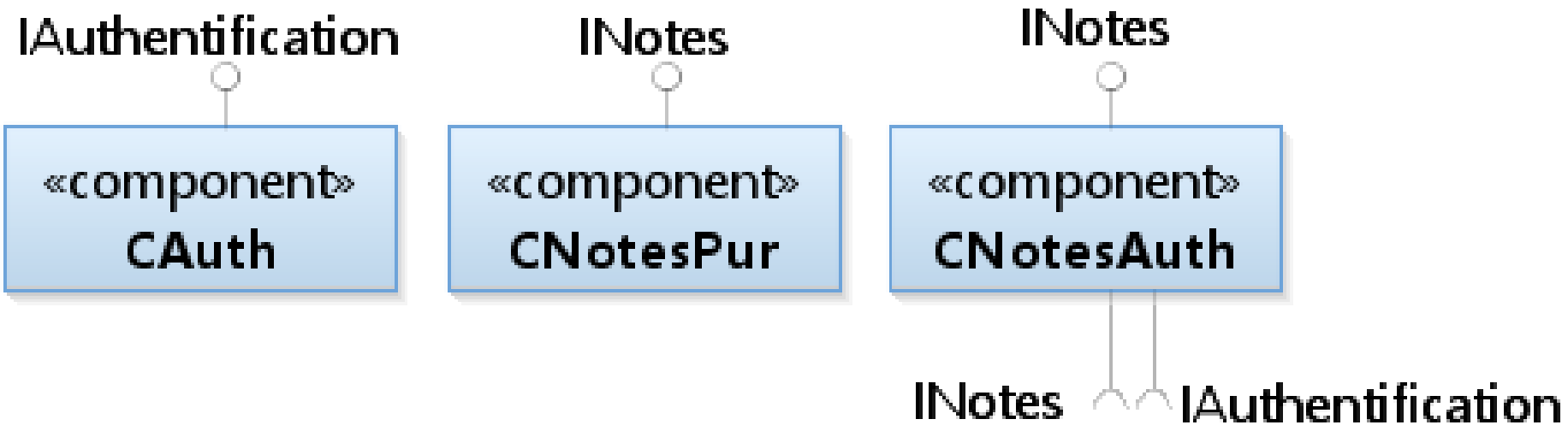
- Instanciation des composants dans une configuration particulière
  - Plus stable au cours de l'exécution que les diagrammes d'objet
- Part:
  - instance d'un composant
  - Notation : nomInstance **:** TypeComposant
- Connecteur :
  - Lien instancié entre deux composants (liaison)
  - Lien dirigé : A->B <=> A utilise ou invoque B
  - Doit être compatible avec les interfaces offertes requises

Instanciation



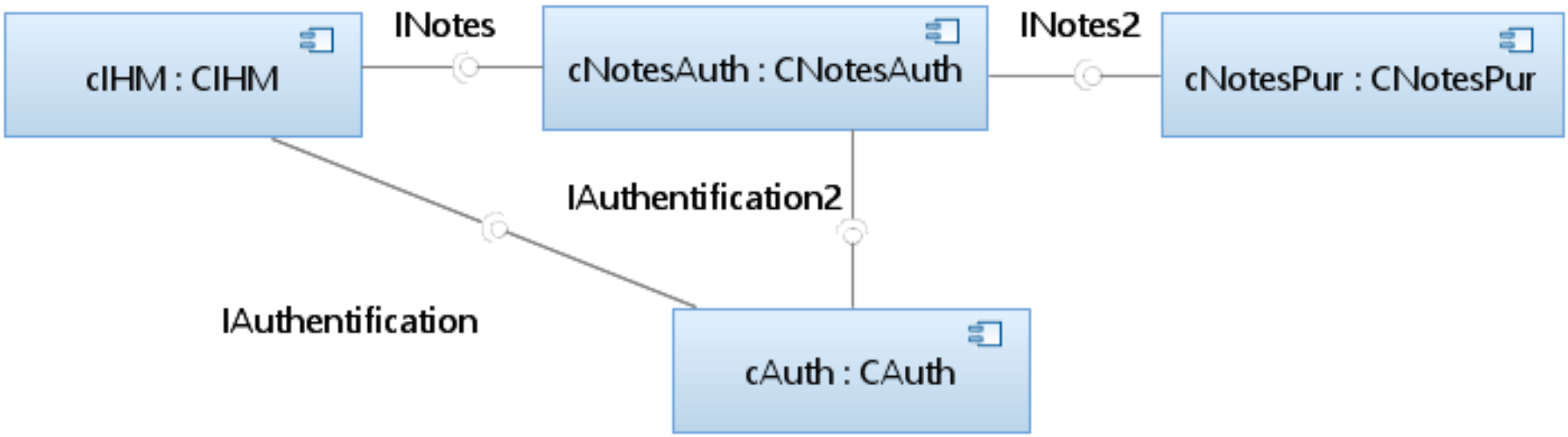
- Double utilisation de IAuthentification

Reconfiguration ?



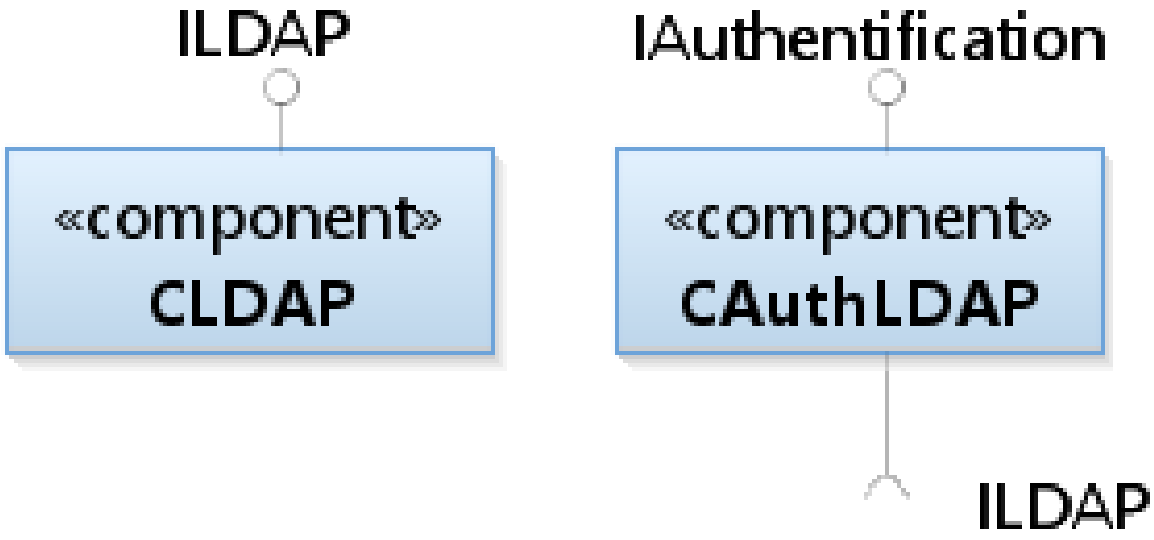
- Rôles dans Cnotes :
  - Contrôle authentification + stockage des notes
- Nouveaux composants
  - CNotesPur : composant bout de chaine

Assemblage v2



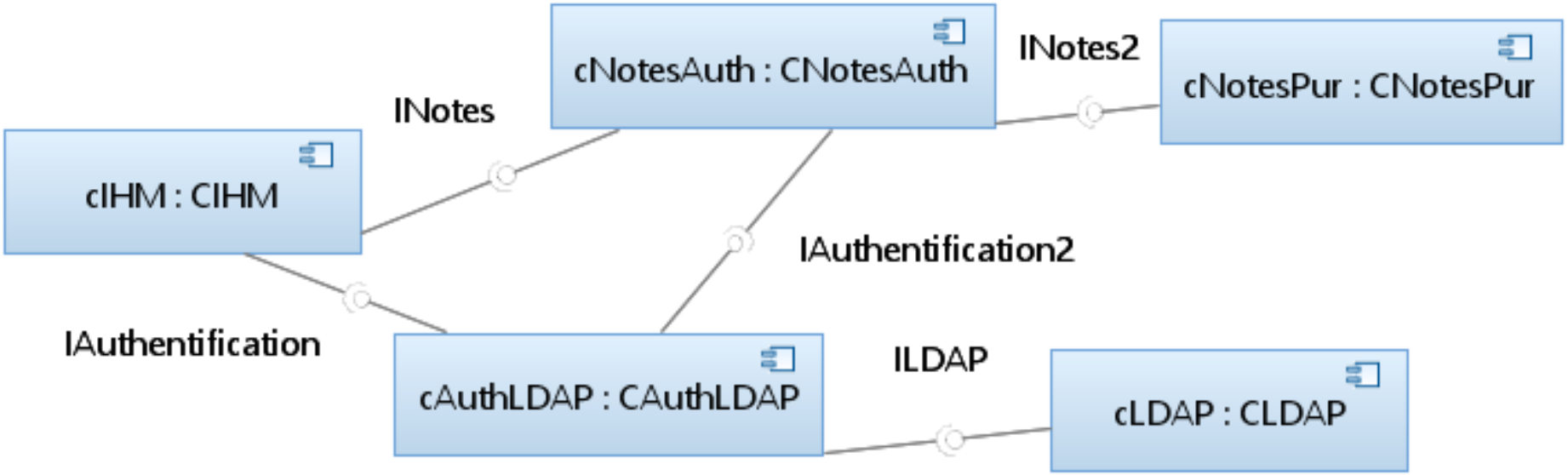
- Reconfiguration
  - Réutilisation de certains composants
  - Transparent pour l’ IHM

Authentification via CAS



- Une implantation plus riche de IAuthentification
  - DP Adapter

Configuration v3



- Encore une configuration
  - L'architecture logicielle = raisonner sur ces reconfigurations

Approche Proposée

Aborder la Conception Architecturale

- /!\ étape un peu difficile, beaucoup de choix possibles
- On souhaite des composants pour l'application :
  - Proposer un premier découpage « naturel » en briques: les composants
  - Chaque composant de base doit avoir une interface offerte
- Découpe structurelle : affecter la responsabilités de certaines données métier au composant.
- Découpe fonctionnelle ; affecter les opérations du « système » de l'analyse sur des interfaces

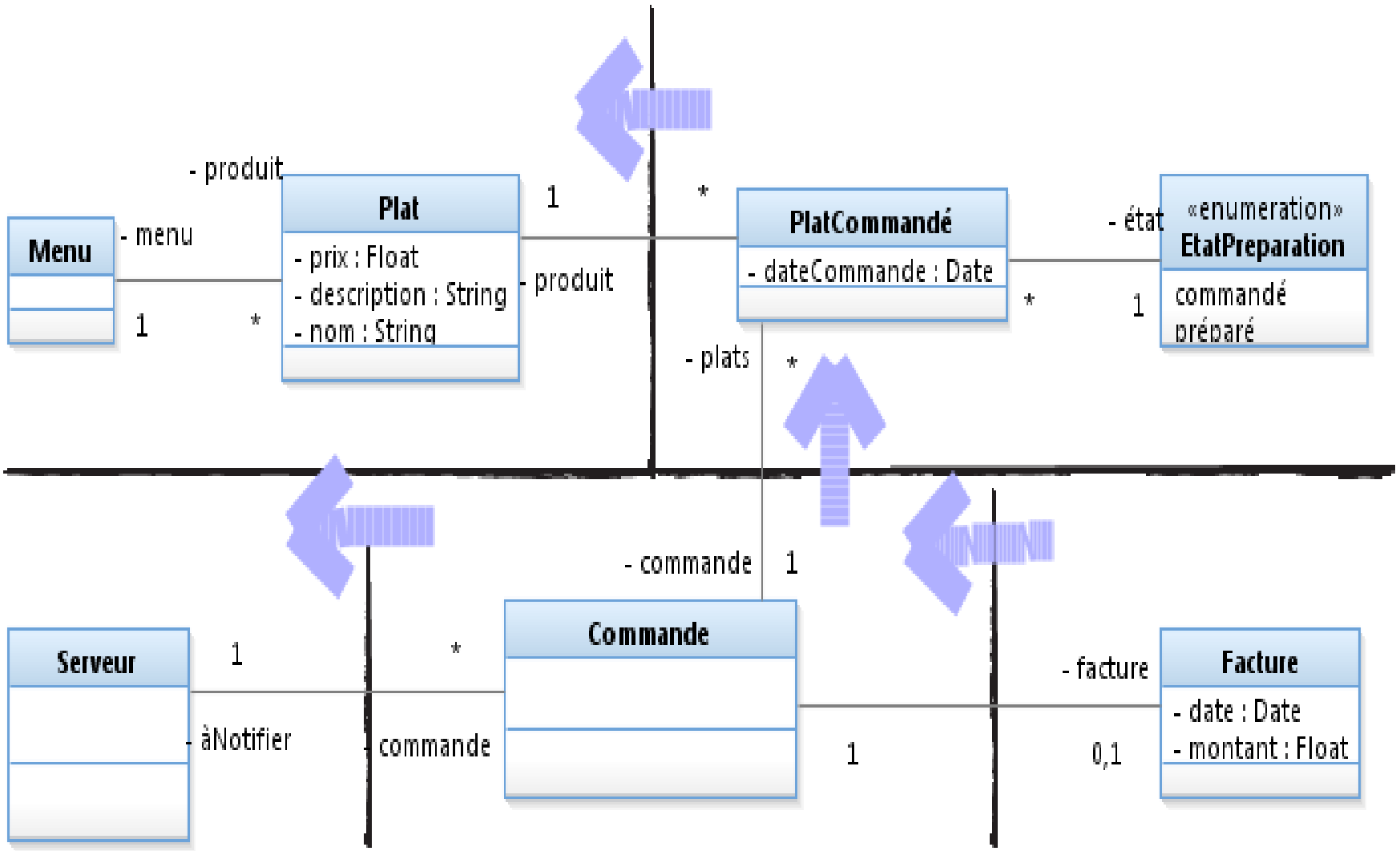
Affiner via la dynamique

- Valider le découpage proposé à l'aide de diagrammes de séquence
  - On reprend et on raffine les séquences d'analyse
  - La ligne de vie du système => plusieurs lignes de vie, une par composant
  - Les interactions externes (acteurs) sont préservées
    - L'acteur et son IHM sont fusionnés
  - On découvre de nouvelles opérations sur les interfaces

Découpe Structurelle

- Découper le diagramme de classes métier
  - Affecter la responsabilité de parties des données aux composants
  - Un composant responsable d'une donnée la stocke et permet d'y accéder (e.g. API CRUD Create/Replace/Update/Delete)
  - Ne pas hésiter à restructurer ! On va vers une solution, les objectifs sont différents de l'analyse
- Découpage « Brut »
  - Frontières « logiques » entre des données moins liées
  - Eviter de couper trop d'associations
- Associations « coupées »
  - Choisir une orientation pour ces associations
  - Ajouter des identifiants dans la cible de l'association
  - Représenter l'association par des identifiants stockés dans la source

Découpe : exemple (Jan'18)



Découpe Fonctionnelle

- En première approximation, chaque composant offre une interface
  - Sauf CIHM
- *Partition* de tous les services offerts du système sur ces interfaces
  - Les services du système sont réalisés par des composants
- Le ou les composants d'IHM
  - Jouent le rôle de vues, très peu de traitements, pas de données stockées
  - Les affichages induisent des API en lecture sur les données des autres composants
  - En principe l'IHM doit invoquer des opérations sur le reste du système dont la signature correspond aux services « public » de l'analyse

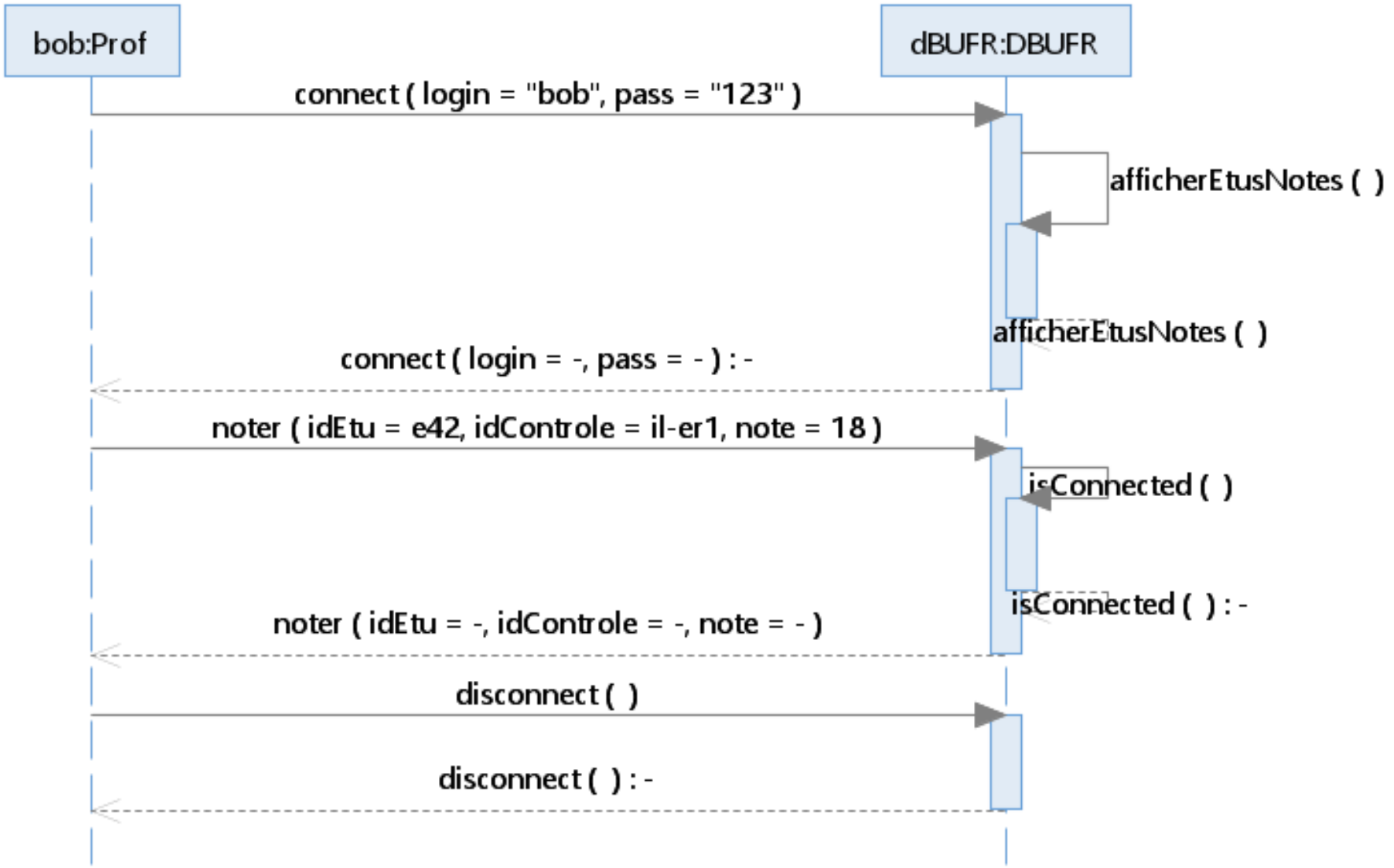
Remise en cause des séquences d'analyse

- Les opérations « public » doivent être retrouvées
  - => Opérations de saisie des acteurs
  - A priori invocation de l'IHM vers le reste
- Les opérations du système :
  - Vérifie que… : typiquement une interaction entre composants pour réaliser le contrôle ou le calcul
  - Enregistre …: mise à jour des données stockées, API en écriture dans les composants qui la stockent
  - Affiche… : engendre une API en lecture sur les données pour remplir l'IHM

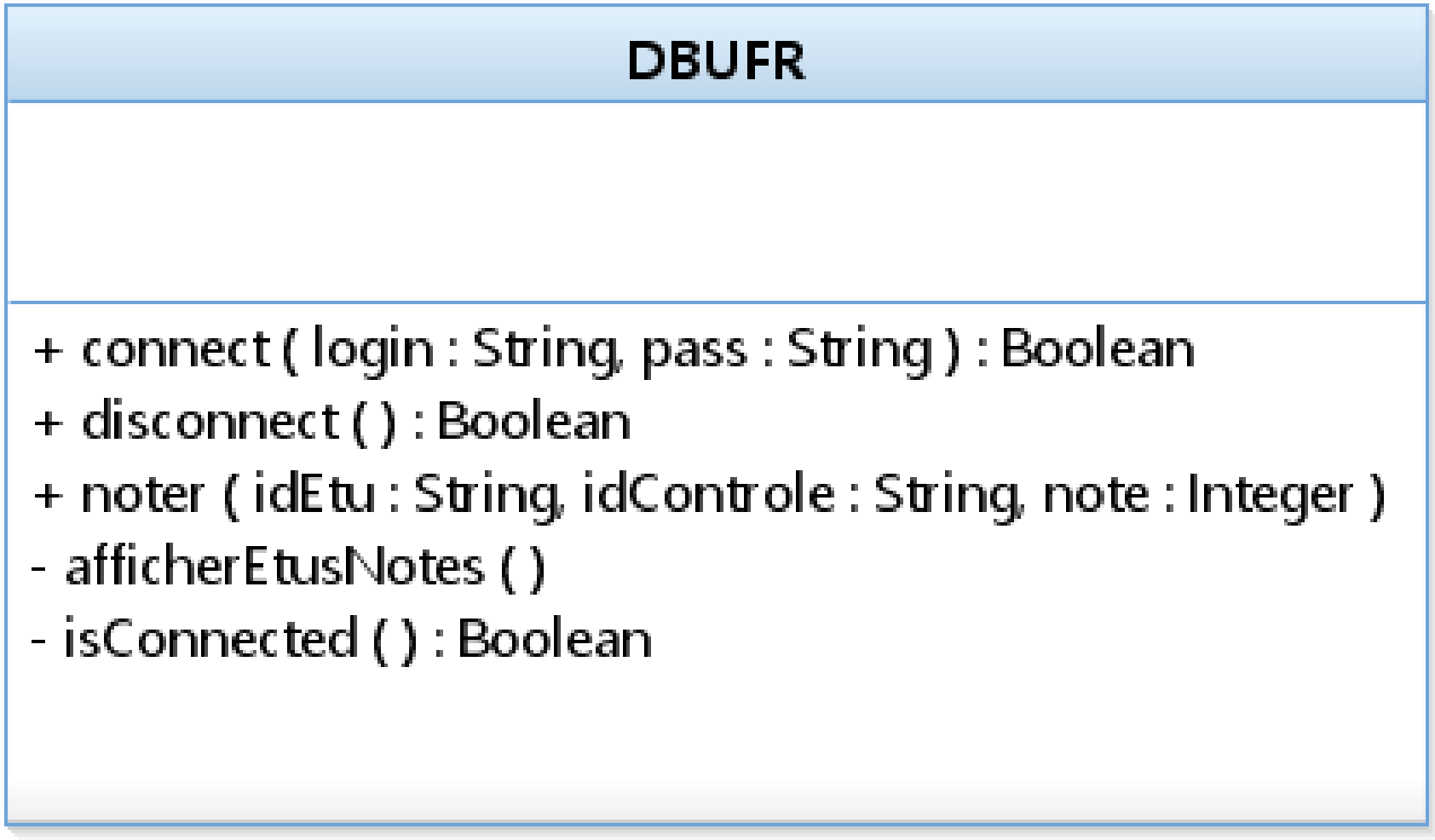
Séquences d'Intégration

- Grain utilisé en conception architecturale :
  - Le système est vu comme un assemblage de composants
    - Services offerts et requis
  - Les composants sont opaques à cette étape
  - Une ligne de vie par instance de composant (lié au diagramme de structure interne)
- La réalisation du diagramme de séquence doit permettre :
  - S'assurer de la faisabilité des traitements
  - Détecter les interactions entre composants nécessaires
    - => Induit des interfaces requises
    - CA invoque foo() sur CB => CA requiert IB, et IB contient foo()
  - ✓ Remettre en cause, affiner
    - Affectation des données (échanges trop complexes ?)
    - Signatures des traitements
    - Affectation des traitements
-

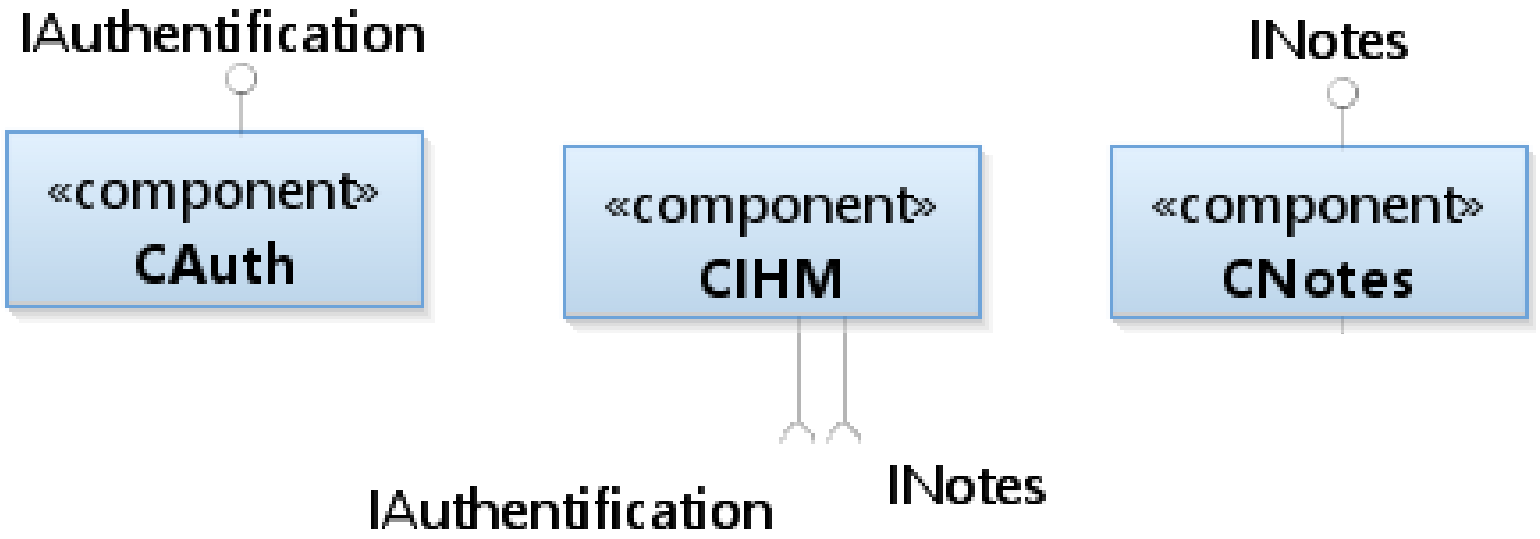
Exemple DBUFR



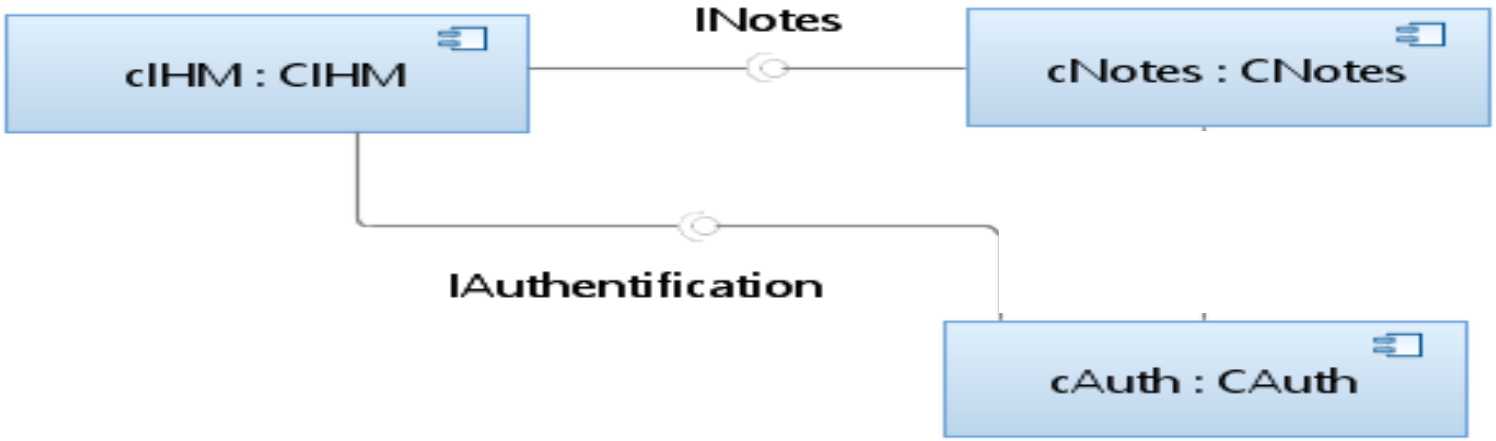
Classe système engendrée



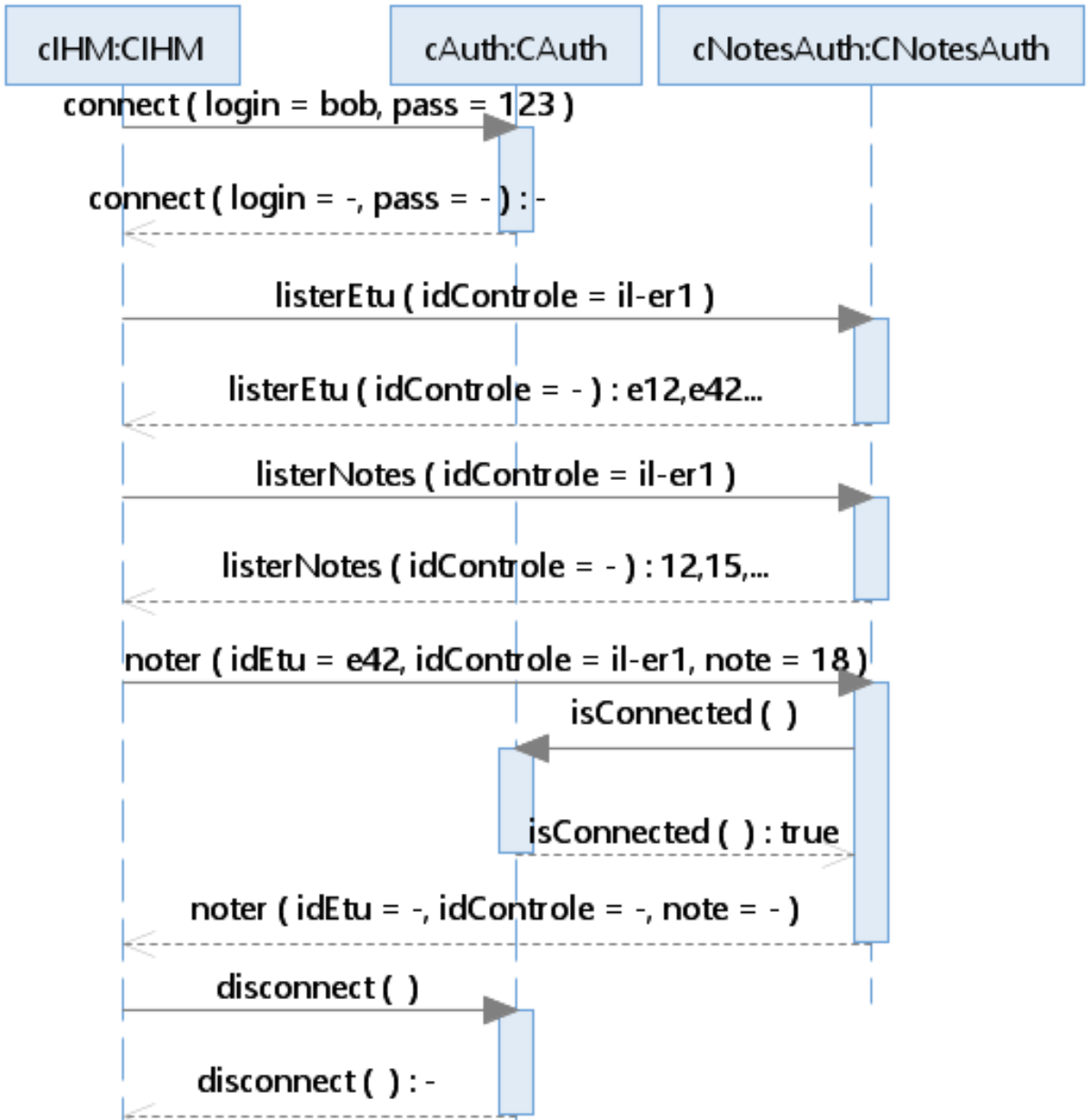
Première approximation



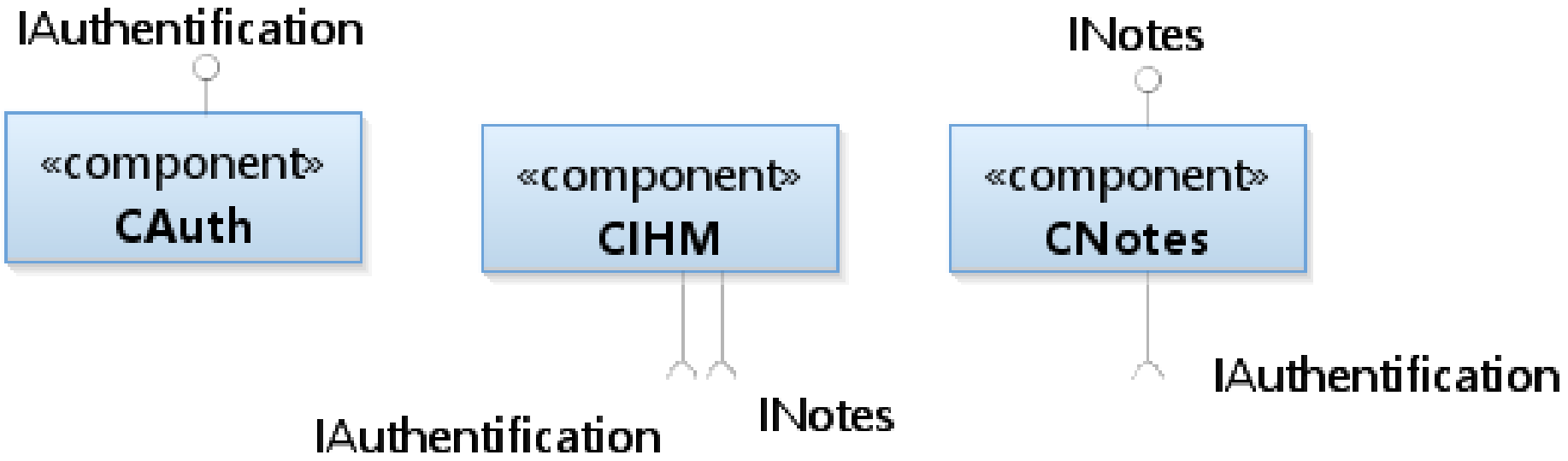
Chaque composant offre une interface sauf IHM



Séquence raffinée



Déduire les dépendances



- CNotes invoque CAuth !
  - Révèle une interface requise dans CNotes
  - Deuxième essai avec un découplage : CNotesAuth
    - DP Décorateur

Séquence raffinée

La ligne de vie du système devient :