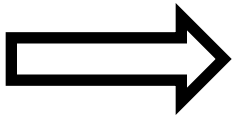
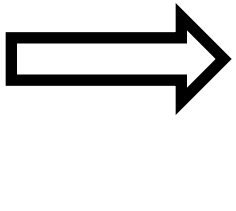
Les Lignes de Produits Logiciels (Software Product Lines)

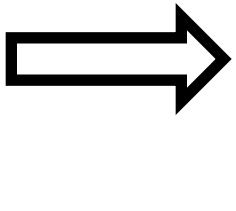
Tewfik Ziadi
UPMC/LIP6
tewfik.ziadi@lip6.fr


android



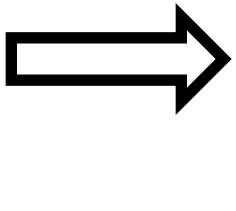


1. Plusieurs modèles de téléphones.
2. Des utilisateurs à travers le monde



Plusieurs **versions** de la même application

1. Plusieurs modèles de téléphones.
2. Des utilisateurs à travers le monde



Plusieurs **versions** de la même application

1. Plusieurs modèles de téléphones.
2. Des utilisateurs à travers le monde

Comment peut-on gérer cette **variabilité logicielle**?

Pourquoi cette variabilité?

- Les facteurs de variabilité :
  - **Economique**
    - une version complète du logiciel, une version gratuite,..
  - **Culturel**
    - Ex. Langue
  - **Technique** :
    - Lié au matériel utilisé

Dans ce cours

- Présenter la notion de Ligne de Produits (LdP)
  - Motivations
  - Définitions et Principes
- L'ingénierie des lignes de produits
  - Ingénierie du Domaine
  - Ingénierie d'Application
  - Démo: l'outil FeatureIDE
- Vers une construction automatique de LdP
  - Des premiers résultats de recherche

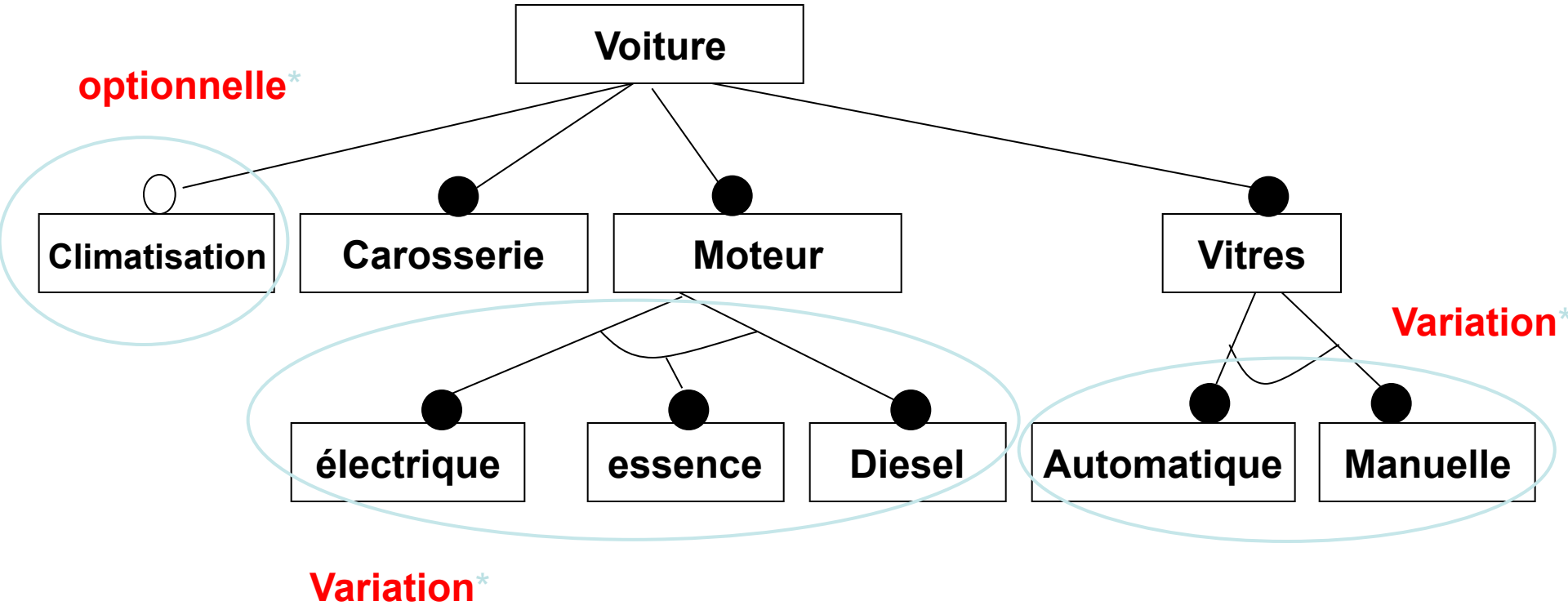
Ligne de Produits Logiciels : définition

*“a set of software- intensive systems that share **a common**, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way”* [Clements et al., 2001

Motivations

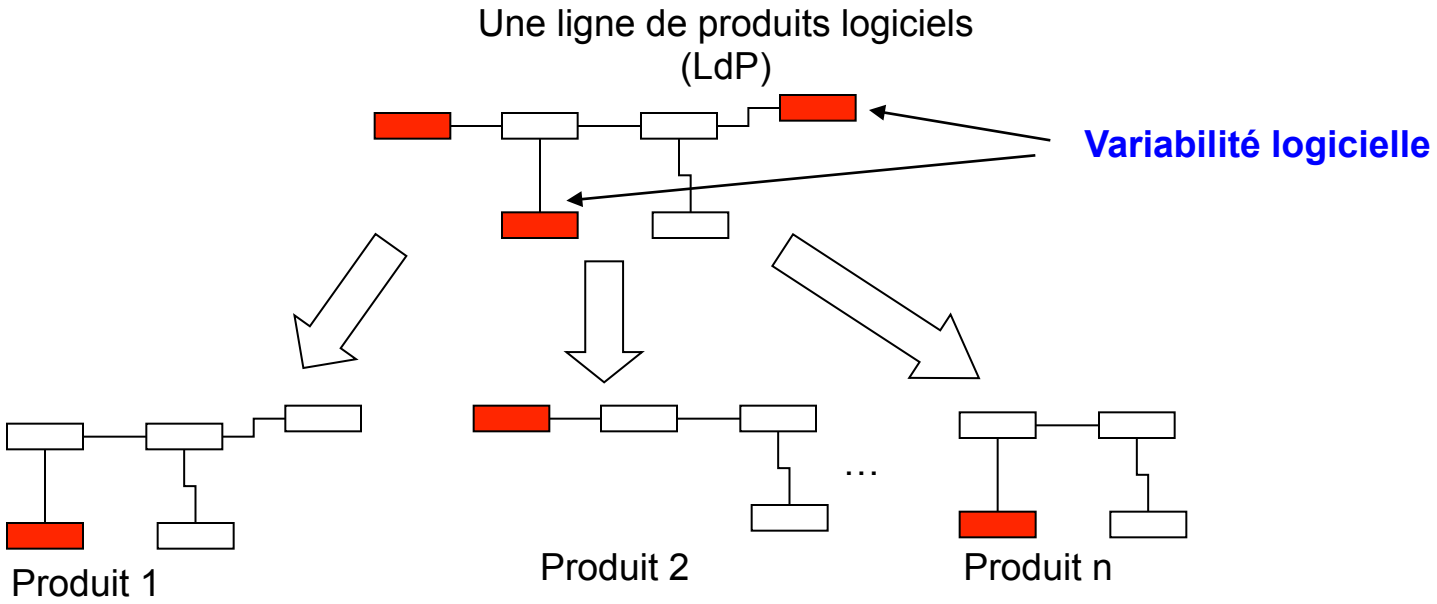
➔ Une transposition des chaines de production industrielles au monde logiciel.

Motivations



* Notations FODA

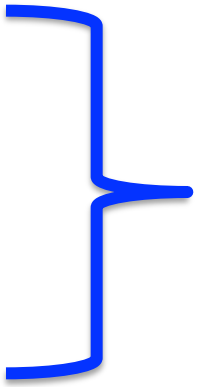
Lignes de Produits Logiciels : Vue « Top-Down »



- ✓ **<u>Dimension 1</u> : Modélisation de la variabilité dans des LdP.**
- ✓ **<u>Dimension 2</u> : Dérivation automatique des produits.**

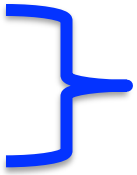
Ingénierie des LdP

- Étape 1:Ingénierie du domaine
  - Analyse du domaine
  - Implémentation du domaine



Dev. « for reuse »

- Étape 2 : Ingénierie d'application
  - Dérivation de produits



Dev. « by reuse »

Étape 1:Ingénierie du domaine (Analyse du domaine)

- Objectifs
  - Étudier le domaine pour identifier les caractéristiques communes et variables pour la famille (*features*).
  - Elle nécessite l'intervention des experts du domaine.
- Comment?
  - La définition du « feature model »

Features

- Les membres de la LdP diffèrent par un ensemble de caractéristiques : « **features** »

- Feature : « *…une caractéristique d'un logiciel définie par les experts de domaine comme importante pour distinguer les différents produits..* »

Features

- Exemple de features : le domaine de véhicules
  - Carrosserie
  - Moteur Essence,
  - Moteur électrique
  - Moteur diesel
  - Vitres manuelle
  - Vitres automatique
  - Climatisation

Spécification de la variabilité

- Feature Modeling Techniques
  - L'origine : FODA (Feature Oriented Domain Analysis)[Chan 90]

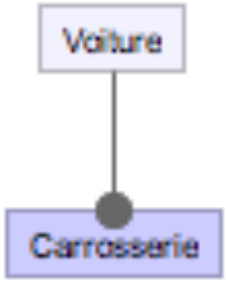
- Décrire la variabilité sous forme d'un diagramme features(**feature model**)

  - **Feature Model** : une notation standard pour décrire la variabilité dans les lignes de produits.

Feature Model (FM)

- Features obligatoires.
  - Les caractéristiques communes à tous les produits.

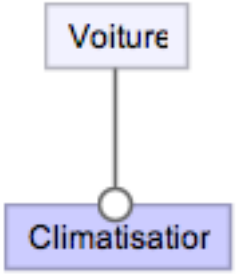
- Notation :



Feature Model (suite)

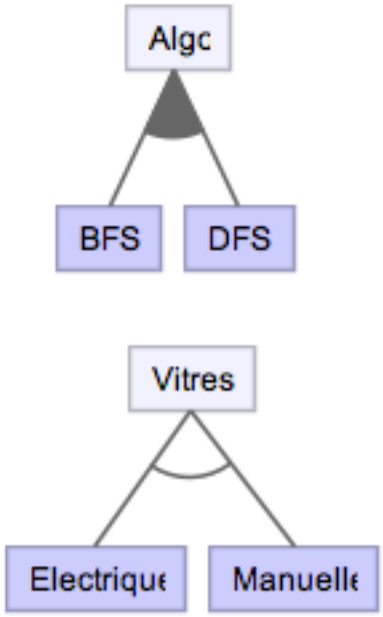
- Feature Optionnelle. Une caractéristique présente seulement dans **certains** produits.

- Notation :



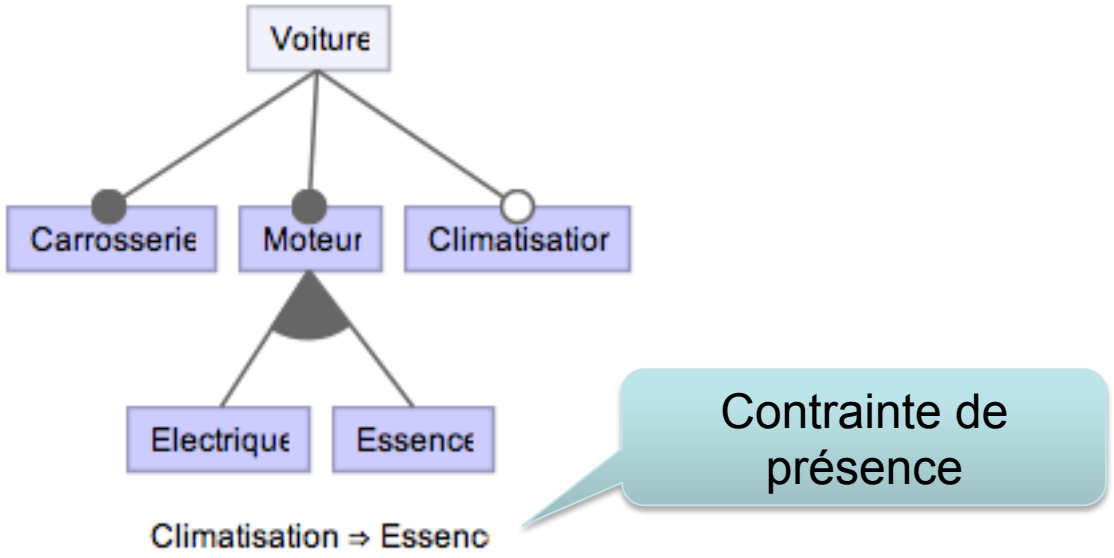
Feature Model (suite)

- Composition de features :
  - OR. Un choix
  - XOR. Un choix exclusive



Feature Model (suite)

- Contraintes de cohérence
  - Des dépendances de présence (ou d'exclusion) entre features



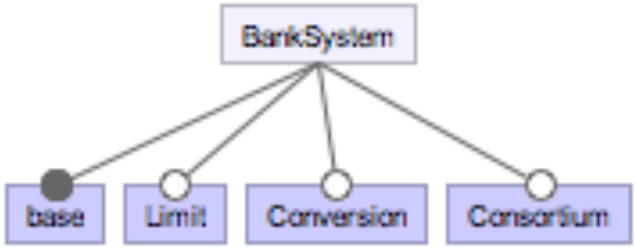
Exemple : logiciels de téléphones mobiles

- Quelles sont les caractéristiques permettant de différentier les produits?
- Définissez le feature model pour cette ligne de produits.

À vous de jouer (TP ☺)

Exemple d'illustration : la LdP banque

- Un famille de systèmes du domaine bancaire.
- Variabilité :
  - La possibilité de découvert sur les compte est optionnelle.
  - L'opération de conversion de devise est optionnelle.
  - La connexion au consortium est optionnelle.



Étape 1: Ingénierie du domaine (Implémentation du domaine)

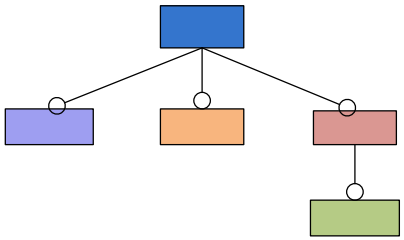
- Il s'agit principalement de construire les assets.
- Un « asset » : un artefact logiciel nécessaire au développement de produits.
  - Des fichiers de code source
  - Des bibliothèques
  - Des modèles..ect

- Étape 2 : Associer à chaque « feature » les artefacts logiciels permettant de l'implémenter.
  ➔ tout un domaine de recherche (très active ces dernières années)

Ingénierie du domaine

Analyse du domaine

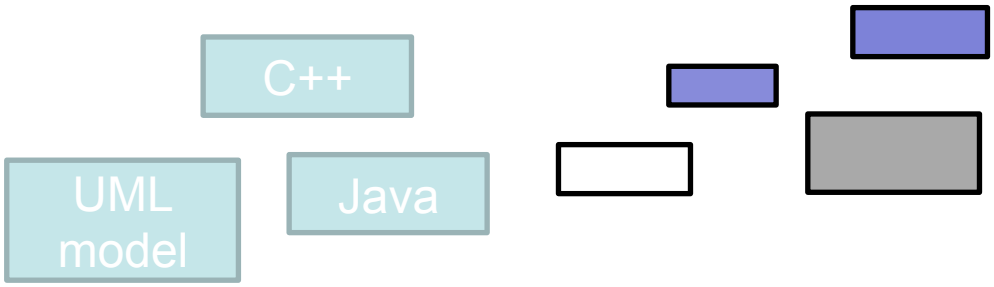
(problem)



Feature Model

Implementation du domaine

(solution)



assets

États de l'art

- **Au niveau code** :
  - FeatureHouse [Apel et al. 09]. Une feature ➔ des fichiers de code
  - AspectJ. Une feature ➔ un aspect (Aspect Oriented Programing))
- **Au niveau modèles**:
  - Feature Oriented Modeling [Czarnecki et al.]
  - Kompose [INRIA]
  - CVL(Common Variable Language) : Un standard de l'OMG
  - LIP6 (Modèles UML)

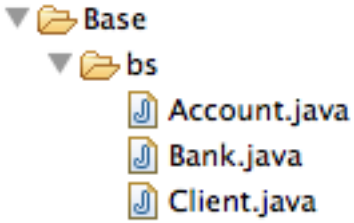
Exemple : **FeatureHouse**

- **FeatureHouse**:
  - Une approche proposée par le groupe SPL de l'université Passau, Allemagne.
  - Une approche supportant plusieurs langages de programmation :
    - C, C++, Java, Csharp,..
  - Une approche intégrée dans l'outil FeatureIDE (cf. TP)

FeatureHouse

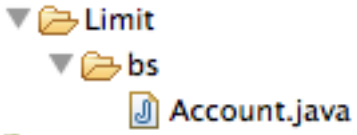
- Une LdP selon FeatureHouse:
  - Un ensemble de fichiers de code source commun (Base)
  - Chaque feature ajoute un raffinement :
    - Ajoutant des nouveaux fichiers de code source
    - Raffine les fichiers communs de base:
      - Ajouter des attributs
      - Ajouter des méthodes/fonctions
      - Modifier le code des méthode existantes (surcharge)

Feature : **Base**



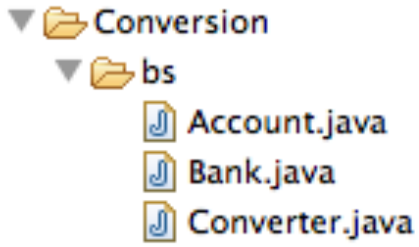
```
package bs;

public class Account {
    private String id;
    private double balance;

   public Account(String i, double m){

       id=i;
       balance=m;
   }
    public void deposit(double amount) {
        this.balance += amount;
    }

    public double getAmount() {
        return balance;
    }
    public void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
        } else {
            System.out.println("Insuffisent balance..!!");
        }
    }

}
```

Feature : **Limit**




```
package bs;

public class Account {

    private double limit;

    public double getLimit() {
        return limit;
    }

    public void withdraw(double amount) {
        if (amount <= balance + limit) {
            balance -= amount;
        } else {
            System.out.println("Insuffisent balance..!!");
        }
    }
}
```

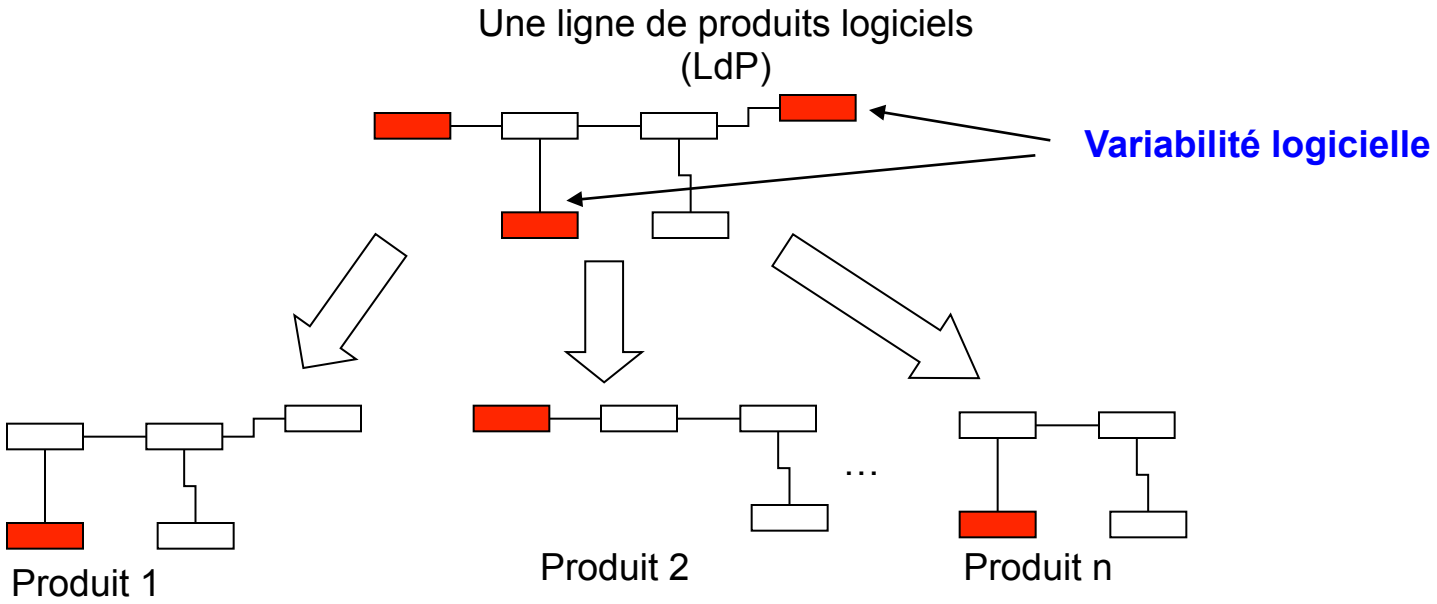
Feature : Conversion




```java
package bs;

public class Account {

    private int currency;

    public Account(String i, double m, double c){

            id=i;
            balance=m;
            currency=c;
       }

    public int getCurrency(){

        return currency;
    }
}
```

```java
package bs;
import java.util.ArrayList;

public class Bank {

    private Converter converter;

    public Bank(Converter c){

        this.converter=c;
    }
    public double convert(int curSource, int curTarget, double amount) {
        return converter.conv(curSource, curTarget, amount);
    }
}
```

Lignes de Produits Logiciels : Vue « Top-Down »



- ✓ **<u>Dimension 1</u> : Modélisation de la variabilité dans des LdP.**
- ✓ **<u>Dimension 2</u> : Dérivation automatique des produits.**

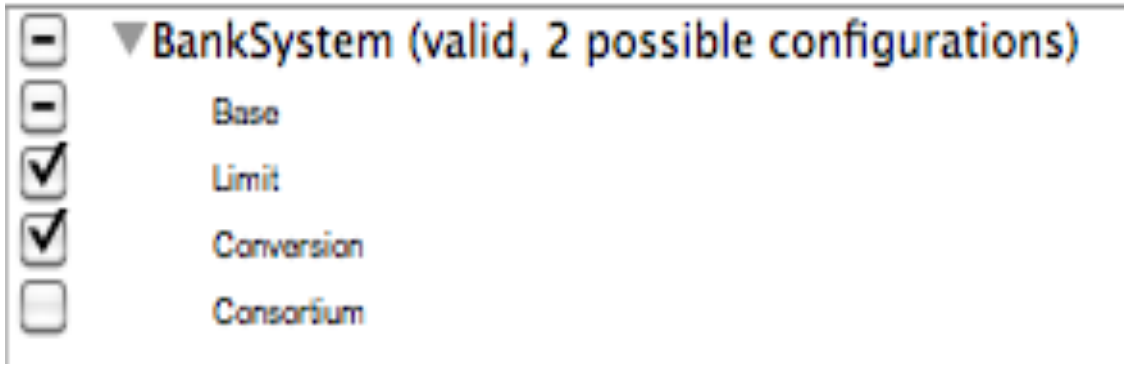
Étape 2: Ingénierie d'application (dérivation de produits)

- Comment générer (dériver) un produit spécifique à partir de la ligne de produits?

- Le besoin de faire des choix de features.
  - On parle de *configurations*

Étape 3: dérivation de produits (suite)

- Une « configuration » : une instantiation de feature modèle.
  - Choix des features optionelles, alternatives
  - Mais des choix qui respectent les contraintes de cohérence.



Dérivation de produits

- La formalisation de ce processus dépend de la façon dont les assets sont définis.

- FeatureHouse: Raffinement de code
- AspectJ : Tissage d'aspects
- Modèles: Transformations de modèles.

Dérivation de produits avec FeatureHouse

1. **Un mécanisme de dérivation automatique de code basé sur le raffinement de code.**

Feature : Base

```java
package bs;

public class Account {
    private String id;
    private double balance;

   public Account(String i, double m){

       id=i;
       balance=m;
   }
    public void deposit(double amount) {
        this.balance += amount;
    }

    public double getAmount() {
        return balance;
    }
    public void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
        } else {
            System.out.println("Insuffisent balance..!!");
        }
    }

}
```

Feature : Limit

```java
package bs;

public class Account {

    private double limit;

    public double getLimit() {
        return limit;
    }

    public void withdraw(double amount) {
        if (amount <= balance + limit) {
            balance -= amount;
        } else {
            System.out.println("Insuffisent balance..!!");
        }
    }
}
```

Composition(Base, Limit)?

```java
package bs;

public   class  Account {

    private String id;

    private double balance;

   public Account(String i, double m){

       id=i;
       balance=m;
   }

    public void deposit(double amount) {
        this.balance += amount;
    }

    public double getAmount() {
        return balance;
    }

    public void withdraw  (double amount) {
        if (amount <= balance + limit) {
            balance -= amount;
        } else {
            System.out.println("Insuffisent balance..!!");
        }
    }

    private double limit;

    public double getLimit() {
        return limit;
```

Feature : Base

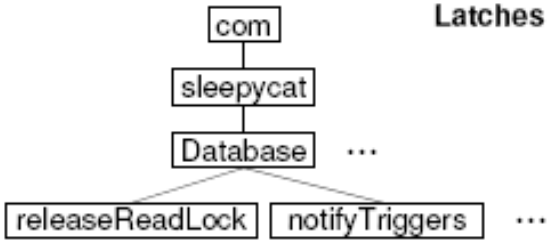
```
package bs;
import java.util.ArrayList;

public class Bank {
    private java.util.HashMap<java.lang.String,bs.Account> accounts;
    public void depisitOnAccount(String id, double amount) {

    public void withdrawfromAccount(String id, double amount) {

    public void display(){
        System.out.println("Bank");
    }
}
```

Feature : Consortium

```
package bs;
import java.util.ArrayList;

public class Bank {
    private Consortium cons;
    public Bank(Consortium c){
        this.cons=c;
    }

    public void display(){
        original();
        System.out.println("Consortium");

    }
}
```

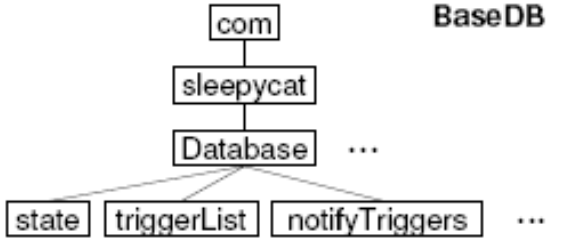
```java
package bs;
import java.util.ArrayList;

public   class  Bank {

    private java.util.HashMap<java.lang.String,bs.Account> accounts;
    public void depisitOnAccount(String id, double amount) {

    public void withdrawfromAccount(String id, double amount) {
    private void  display__wrappee__Base  (){
        System.out.println("Bank");
    }


    public void display(){
        display__wrappee__Base();
        System.out.println("Consortium");

    }


    private Consortium cons;


    public Bank(Consortium c){


}
```

```
1  package com.sleepycat;
2  public class Database {
3    private void releaseReadLock() throws DatabaseException { ... }
4    protected void notifyTriggers(Locker locker, DatabaseEntry priKey,
5      DatabaseEntry oldData, DatabaseEntry newData) throws DatabaseException {
6      original(locker,priKey,oldData,newData);
7      releaseReadLock();
8    } // 50 further lines of code...
9  }
```



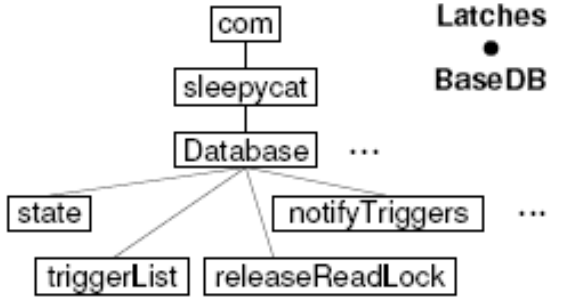
●

```
1  package com.sleepycat;
2  public class Database {
3    private DbState state;
4    private List triggerList;
5    protected void notifyTriggers(Locker locker, DatabaseEntry priKey,
6      DatabaseEntry oldData, DatabaseEntry newData) throws DatabaseException {
7        acquireReadLock();
8        for (int i=0; i < triggerList.size(); i+=1) {
9          DatabaseTrigger trigger=(DatabaseTrigger)triggerList.get(i);
10         trigger.databaseUpdated(this,locker,priKey,oldData,newData);
11       }
12   } // over 650 further lines of code...
13 }
```



=

```
1  package com.sleepycat;
2  public class Database {
3    private DbState state;
4    private List triggerList;
5    protected void notifyTriggers(Locker locker, DatabaseEntry priKey,
6      DatabaseEntry oldData, DatabaseEntry newData) throws DatabaseException {
7        acquireReadLock();
8        for (int i=0; i < triggerList.size(); i+=1) {
9          DatabaseTrigger trigger=(DatabaseTrigger)triggerList.get(i);
10         trigger.databaseUpdated(this,locker,priKey,oldData,newData);
11       }
12       releaseReadLock();
13   }
14   private void releaseReadLock() throws DatabaseException { ... }
15   // over 700 further lines of code...
16 }
```



Démo : la suite FeatureIDE

- Un plugin Eclipse pour l'ingéniérie des LdP.

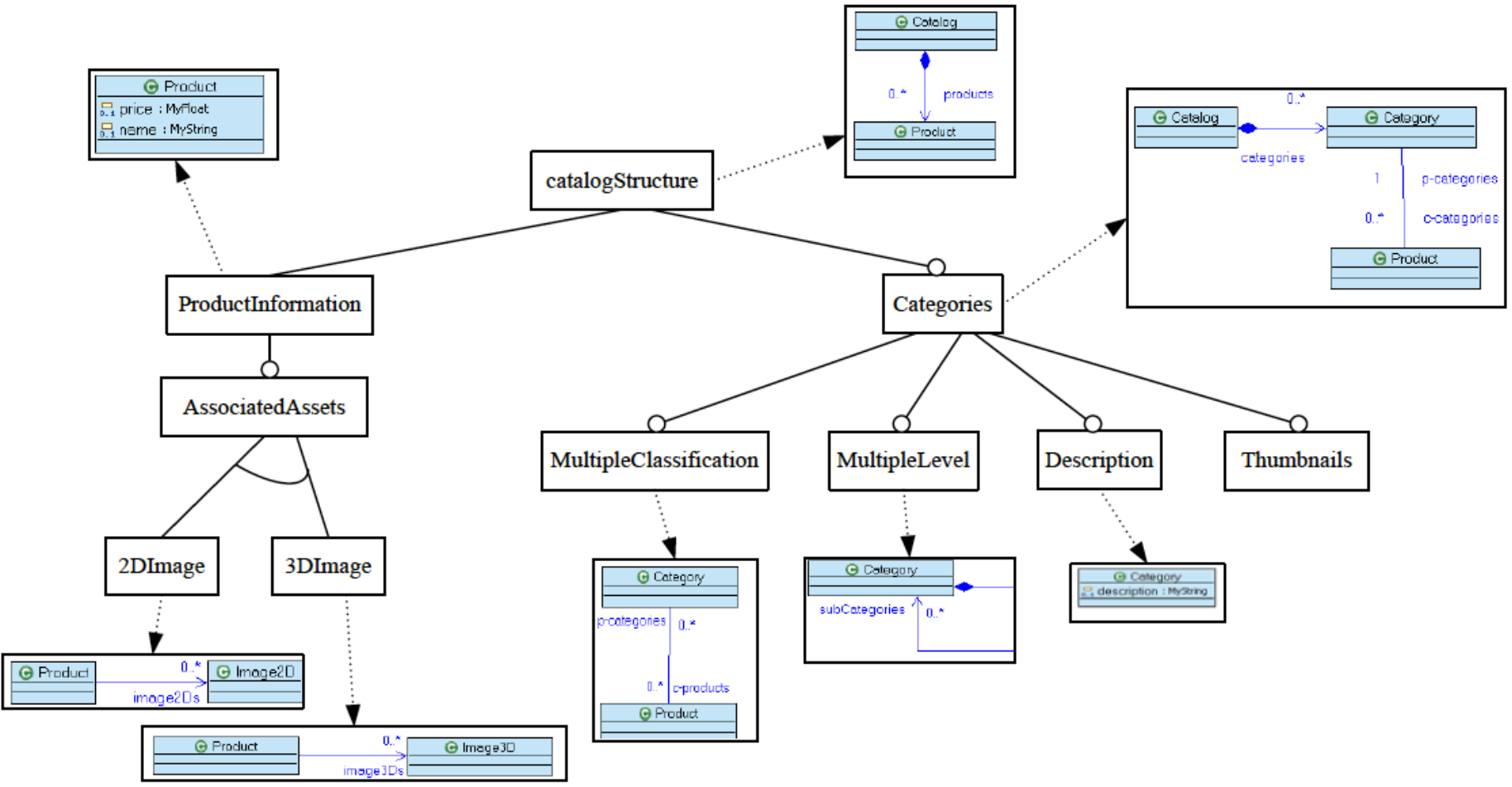
http://wwwiti.cs.unimagdeburg.de/iti_db/research/featureide/

- Open source
  - Université Magdeburg (Ger), Texas (USA)..ect
- Éditeur des feature modèles
- Des outils pour les étapes 2 et 3

Dans le contexte de l'ingénierie dirigée par les modèles

- Principe
  - Associer à chaque feature un modèle ou un fragment
  - Dérivation de modèles de produits = composition de modèles

Exemple de l'approche Kompose [INRIA-Rennes]



LdP « Top down » : Bilan

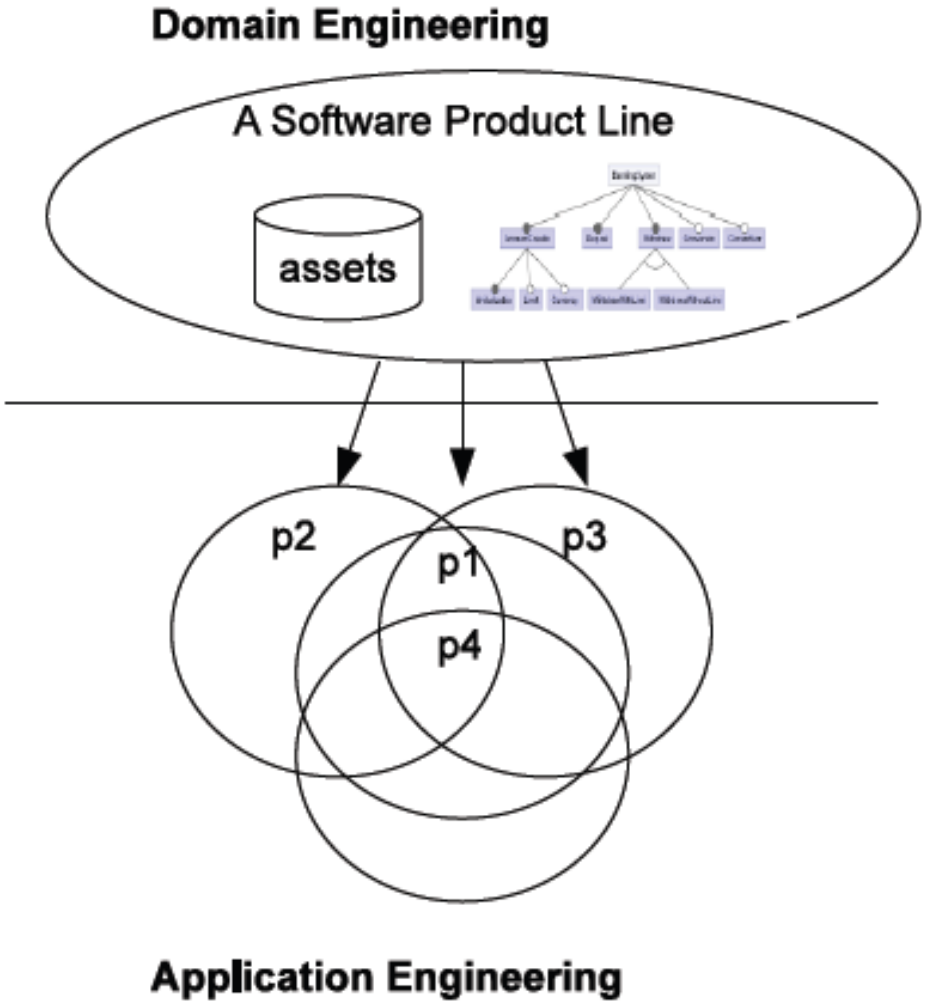
- Lignes de Produits Logiciels (Software Product Lines)
  - Un nouveau paradigme du génie logiciel
  - Une approche en 2 étapes
  - Feature Model : élément central

- Un domaine de recherche très active et qui intéresse les industriels
  - Un vrai besoin de gestion de variabilité

Dans ce cours

- Présenter la notion de Ligne de Produits (LdP)
  - Motivations
  - Définitions et Principes
- L'ingénierie des lignes de produits
  - L'approche générale
  - Exemple de l'approche AHEAD et l'outil FeatureIDE
  - Démo: l'outil FeatureIDE
- **Vers une construction automatique de LdP**
  - **Nos premiers résultats de recherche**

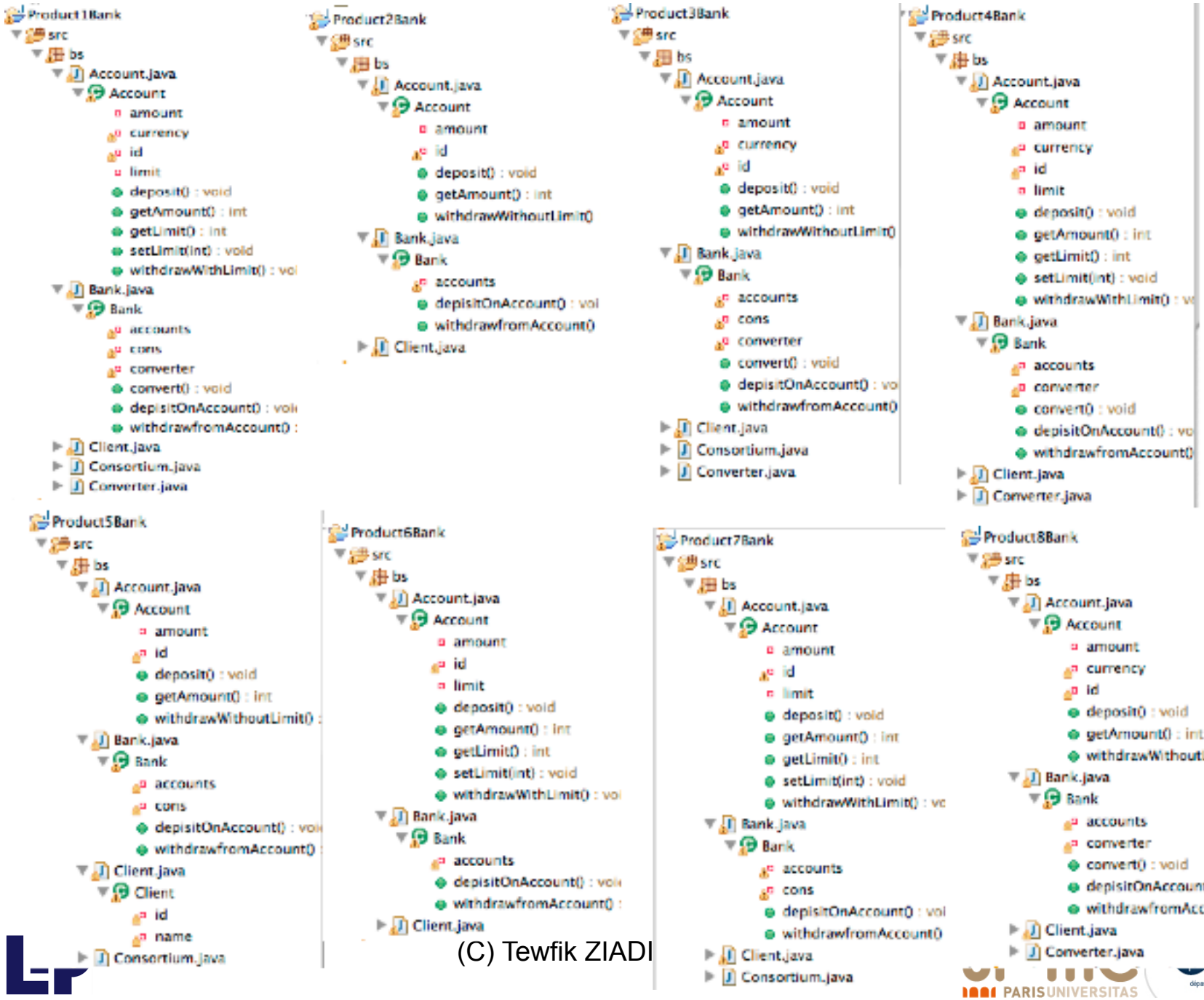
Constats : approche **top-down**



Constats

- L'approche top-down est très intéressante si l'approche LdP a été adopté dès le départ (avant de commencer le dev. de produits)

- **Question** : que peut-on faire lorsqu'on a pas adopté cette approche de LdP mais on a plusieurs produits qui sont similaires?

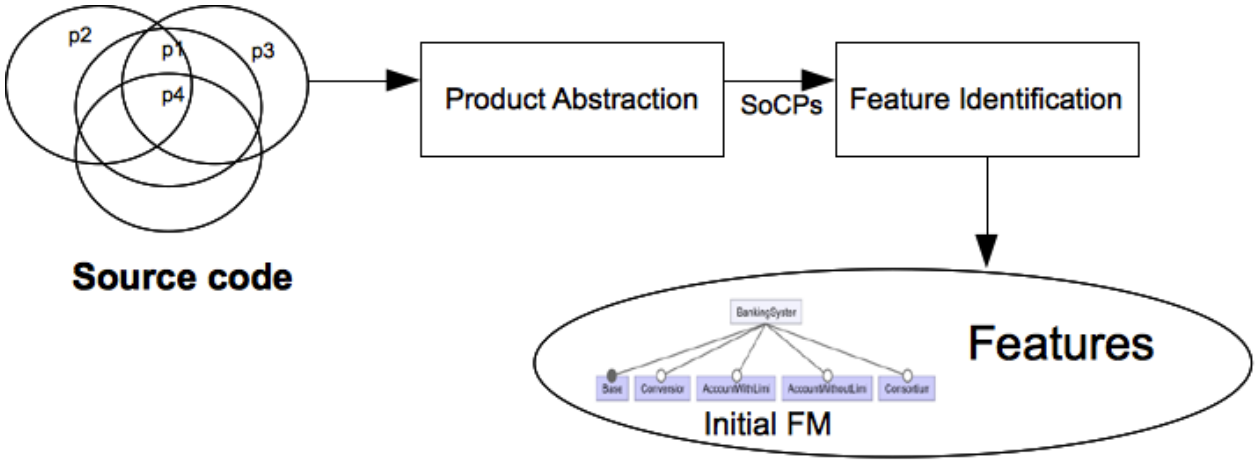


Problématique

- Construction automatique des lignes de produits à partir de produits similaires
  - Identification automatique des features?
  - Construction de feature modèle?
  - Construction des assets ?
- **Pourquoi?**
  - Factoriser les connaissances dispersées au niveaux de différent produits.
  - Faciliter la gestion de l'évolution
  - Découvrir des nouveaux produits

Notre approche (un article publié accepté à CSMR 2012)

- Identification automatique de features à partir de code source de plusieurs produits similaires

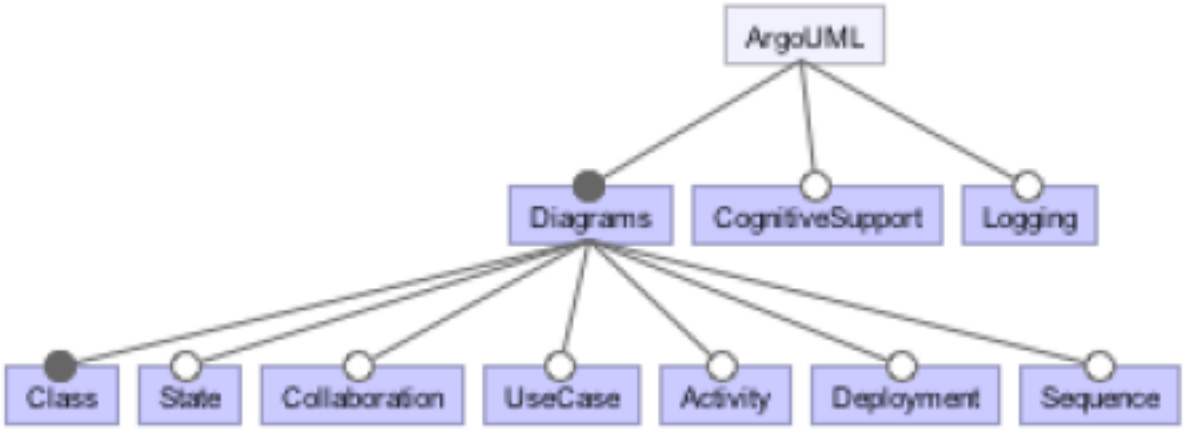


Product Variant Abstraction

- **Questions**:
  - Comment représenter les produits (le code source) pour pouvoir les comparer?
  - Comparer le code source?
  - Quelle granularité?
- **Idée**
  - Parser le code source de chaque produit et le représenter sous forme d'un ensemble de AST (Abstract Syntax Tree) –cf. FeatureHouse
  - Chaque produit est décrit sous la forme d'un ensemble atomique ou chaque élément est un nœud dans l'AST.
  - Des algorithmes :
    - Identification de features et construction de FM.
    - Génération de code des features.

Validation sur un exemple réel

- Argo-UML : un outil UML open source (Java)
- Nous disposons de 10 produits



Validation sur un exemple réel (suite)

Product	Description	LOC
P1	Only Activity disabled	118,066
P2	All features disabled	82,924
P3	Only Cognitive Support disabled	104,029
P4	Only Collaboration disabled	118,769
P5	Only Deployment disabled	117,201
P6	Only Logging disabled	118,189
P7	All features enabled	120,348
P8	Only Sequence disabled	114,969
P9	Only State disabled	116,431
P10	Only Use Case disabled	117,636

Résultats

ture	Name	# Cons. Primitiv	P1	P2	P3	P4	P5	P6	P7	P8	P9	P
1	Base	11223	X	X	X	X	X	X	X	X	X	
2	Diagram	151	X		X	X	X	X	X	X	X	
3	Logging	175	X		X	X	X		X	X	X	
4	Cognitive	1663	X			X	X	X	X	X	X	
5	Usecase	270	X		X	X	X	X	X	X	X	
6	Sequence	423	X		X	X		X	X	X		
7	Collab	110	X		X			X	X	X	X	
8	State	479	X		X	X		X	X	X	X	
9	Deploy	219	X			X			X	X	X	
10	Activity	243			X	X			X		X	
11	JunctionCognitiveLogging	37	X		X	X		X	X		X	
12	JunctionSequenceLogging	12	X			X		X	X	X	X	
13	JunctionCollabSequence	1	X					X	X		X	
14	JunctionCognitiveDeploy	67	X		X	X			X	X	X	
15	JunctionCognitiveSequence	5	X		X	X			X	X	X	
16	JunctionCollabLogging	2	X		X				X	X	X	
17	JunctionDeployLogging	5	X		X	X			X	X	X	
18	JunctionStateLogging	1	X		X	X	X		X	X		
19	JunctionUseCaseLogging	4	X		X	X	X		X	X	X	
20	JunctionActivityLogging	1			X	X	X		X	X	X	
21	JunctionActivityState	9			X	X	X	X	X	X		