Les Lignes de Produits Logiciels (Software Product Lines)

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

L'exemple de Notepad

- Nous avons le code source d'une application implémentant l'éditeur « Notepad ».
  - Fonctionnalités classiques:
    - Edition
    - Ouvrir
    - Copier/Coller.Couper (Cut)
    - Recherche (Finder)
    - Annuler (Undo)

L'exemple de Notepad

- Nous avons le code source d'une application implémentant l'éditeur « Notepad ».
  - Fonctionnalités classiques:
    - Edition
    - Ouvrir
    - Copier/Coller.Couper (Cut)
    - Recherche (Finder)
    - Annuler (Undo)
- **Question** : comment modifier le code source de cette application pour produire plusieurs versions :
  - V 1 : Notepad sans Copier/Coller sans Recherche sans Annuler
  - V 2 : Notepad avec Copier/Coller avec Recherche sans Annuler
  - V 3 : Notepad sans Copier/Coller avec Recherche avec Annuler

Une idée?

Une idée?

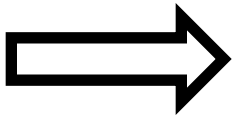
- Prendre la version originale et enlever “manuellement” le fragment de code concernant les deux fonctionnalités Copier/Coller, Recherche, Annuler.
- et ça marche..

Des idées…?

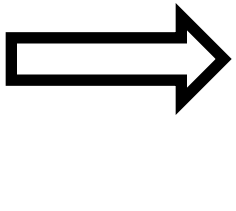
- Prendre la version originale et enlever “manuellement” le fragment de code concernant les deux fonctionnalités Copier/Coller, Recherche, Annuler.
- et ça marche..

Mais quels sont les problèmes?


android

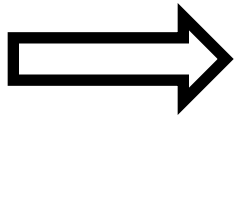


1. Plusieurs modèles de téléphones.
2. Des utilisateurs à travers le monde



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 de Domaine
  - Ingénierie d'Application
  - Démo: l'outil FeatureIDE
- Vers une construction automatique de LdP
  - Des premiers résultats de recherche

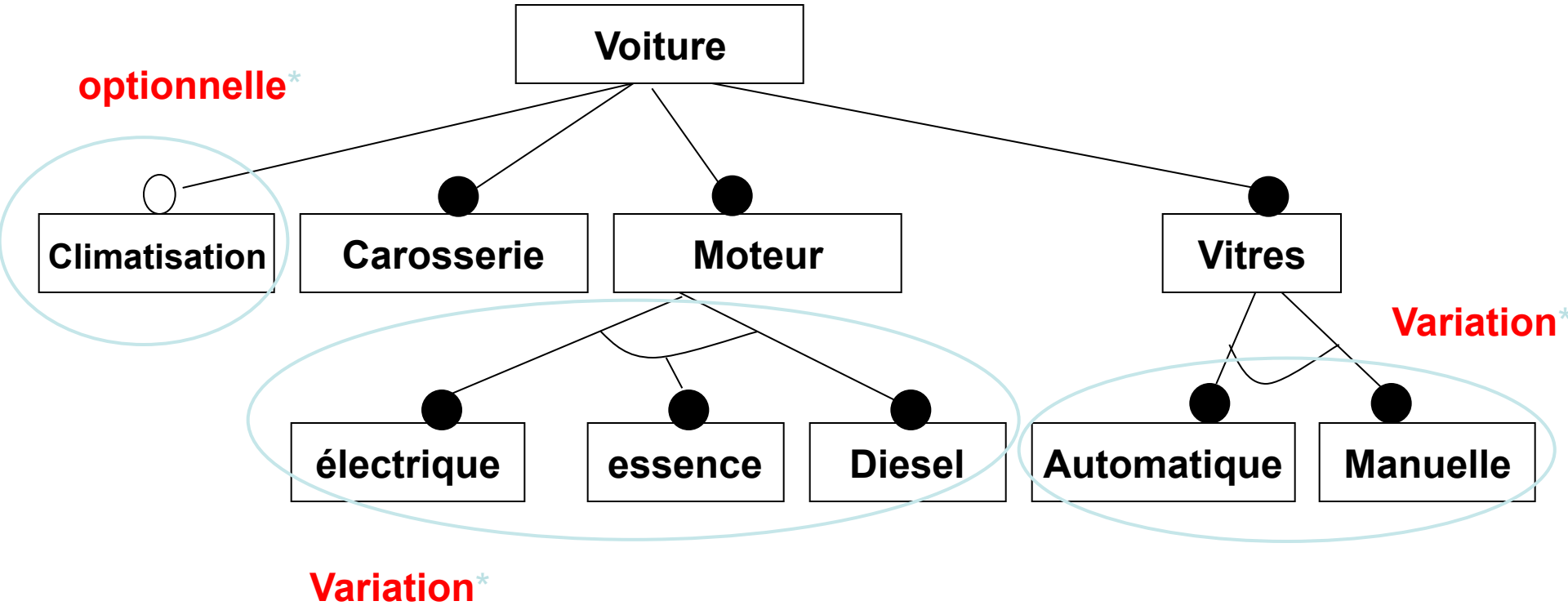
Ligne de Produits Logiciels : définition

*“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

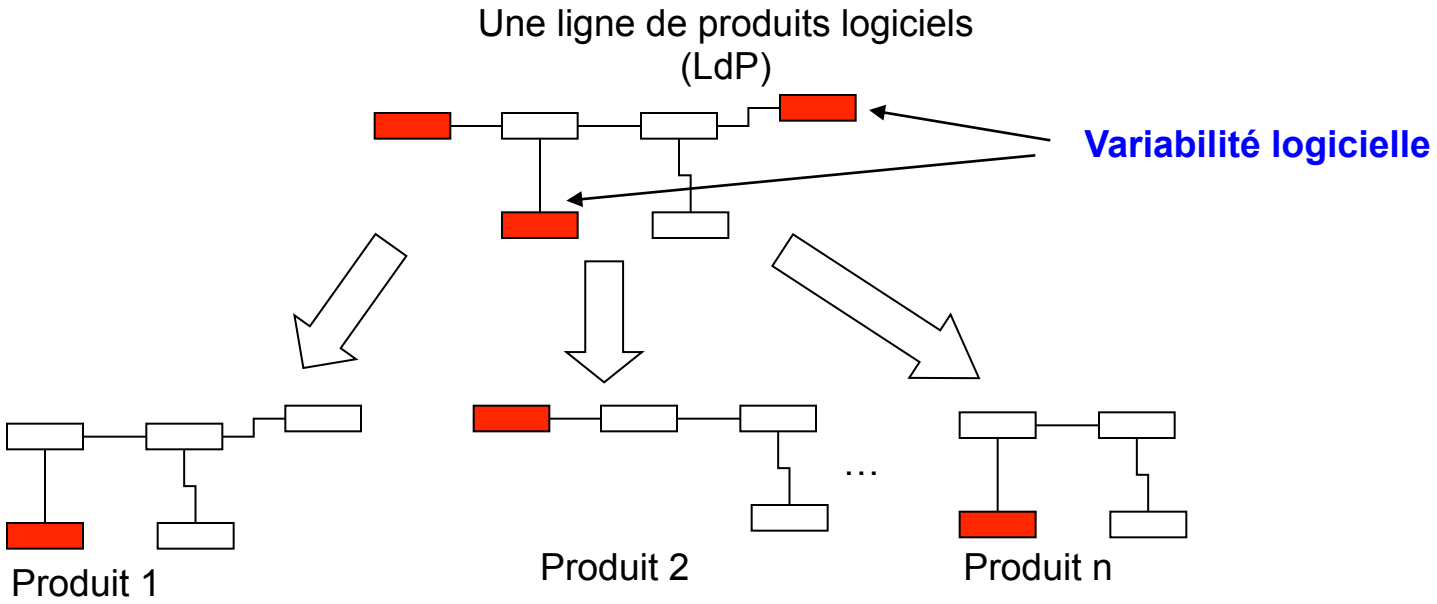
➔ Une transposition des chaines de production industrielles au monde logiciel.

Motivations



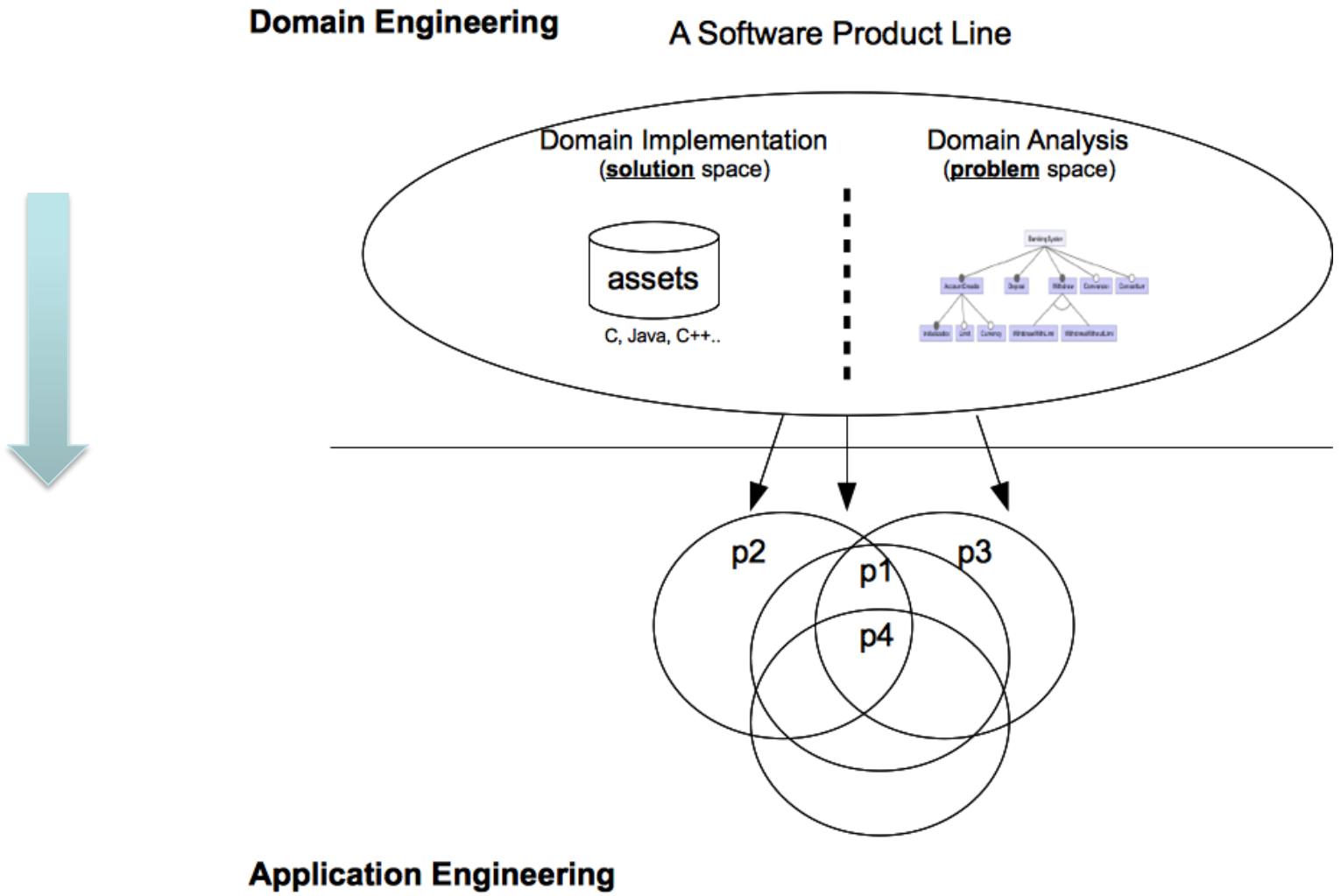
* Notations FODA

Lignes de Produits Logiciels : Vue « Top-Down »

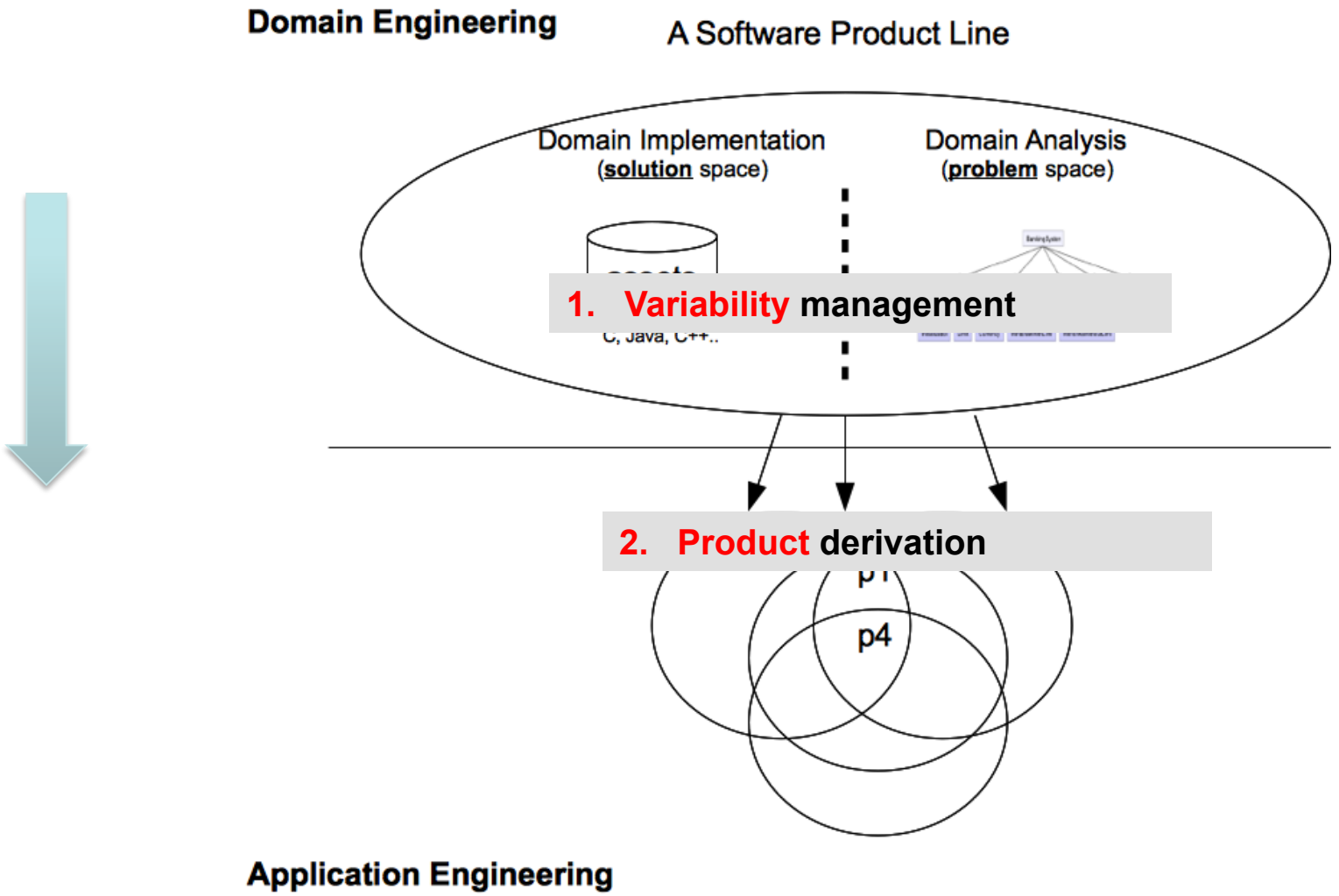


- ✓ **<u>Dimension 1</u> : Modélisation de la variabilité dans des LdP.**
- ✓ **<u>Dimension 2</u> : Dérivation automatique des produits.**

Software Product Line Engineering (SPLE)

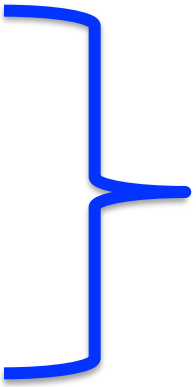


Software Product Line Engineering (SPLE)



Ingénierie des LdP

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



Dev. « for reuse »

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

Dev. « by reuse »

Étape 1:Ingénierie de domaine (Analyse de 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]

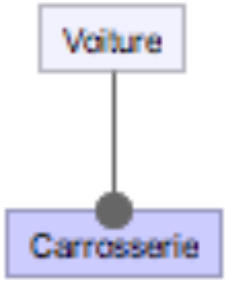
- Décrire la variabilité sous forme d'un diagramme features(**feature model**)

  - **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.

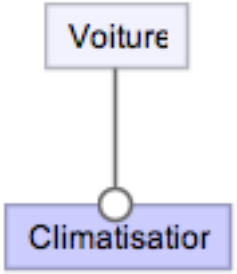
- Notation :



Feature Model (suite)

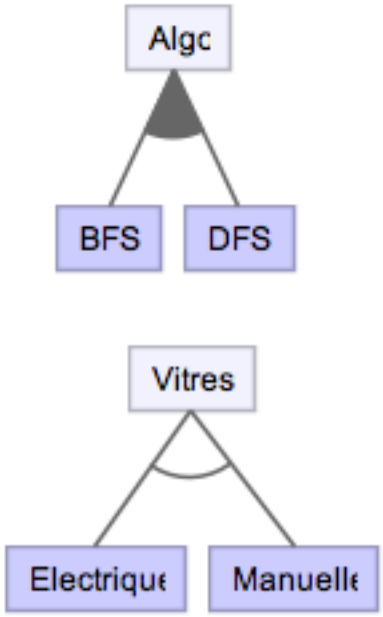
- Feature Optionnelle. Une caractéristique présente seulement dans **certains** produits.

- Notation :



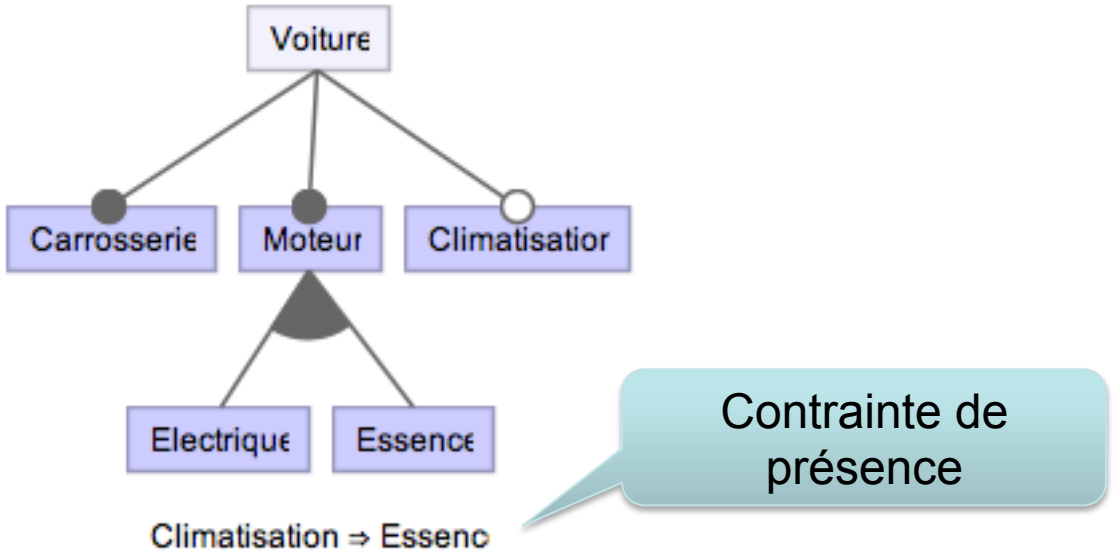
Feature Model (suite)

- 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



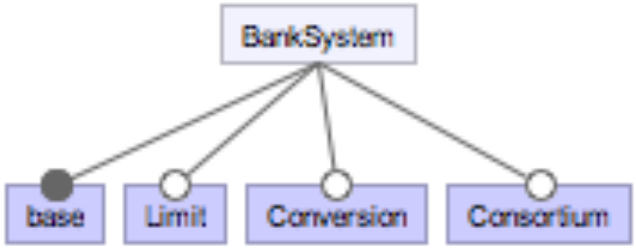
Exemple : logiciels de téléphones mobiles

- 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 de domaine (Implémentation de domaine)

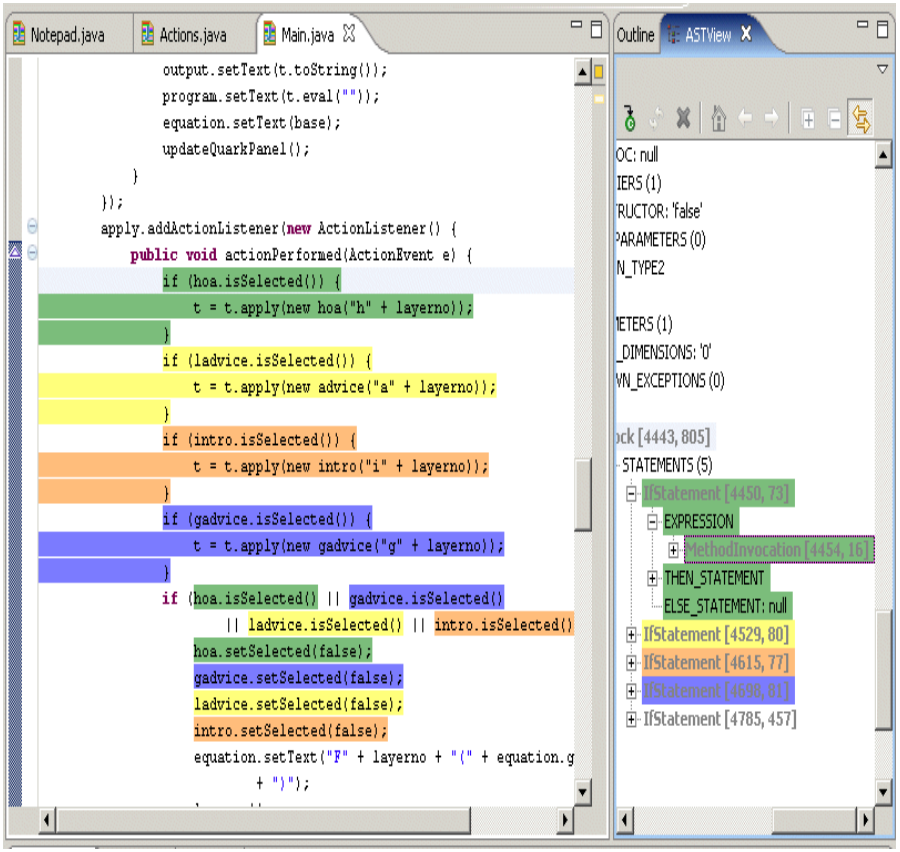
- Il s'agit principalement de construire les assets.
- Un « asset » : un artefact logiciel nécessaire au développement de produits.
  - Des fichiers de code source
  - Des bibliothèques
  - Des modèles..ect

Implémentation

- Comment implémenté les artefacts de la ligne de produits?
  - Comment associé les artefacts logiciels aux features?
    - eg.; Le code source, modèles (architecture).

- Deux familles d'approches :
  - Approches basées sur **des annotations**
  - Approches **compositionnelles**

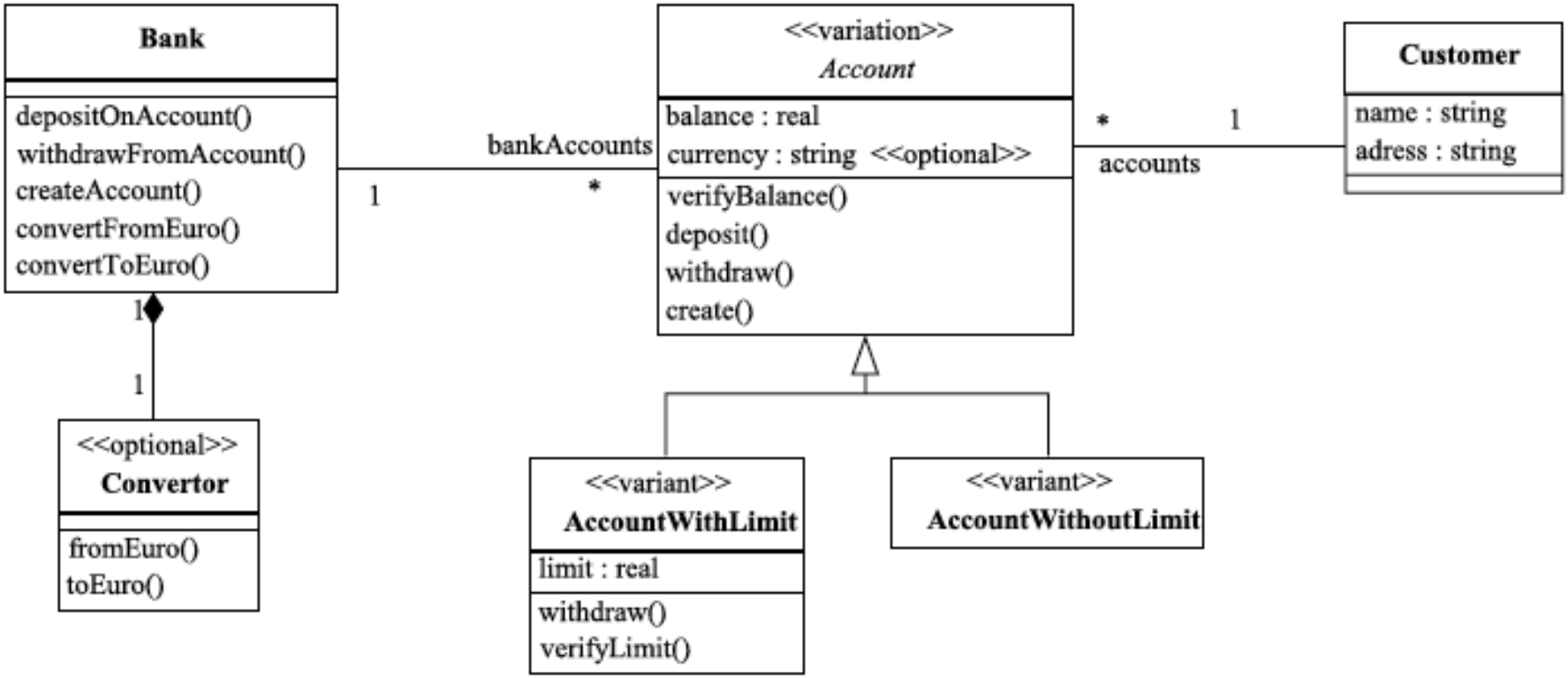
Approches basées sur **des annotations**



Source: [Cide, 10]

- **Gestion de la variabilité**
  - Un produit **MAXIMAL**.
  - Des annotations pour spécifier les fragments qui sont associés à chaque feature.
- **Dérivation de produits**
  - Un produit particulier est dérivé par la suppression des fragments associés aux features qui sont désactivées.

Annotative approaches: Model



Source: Ziadi et al.2006

Approches **compositionnelles**

- **Gestion de la variabilité**
  - Les features de la ligne de produits sont implémentés comme des fragments séparés..
- **Dérivation de produits**
  - Un **compositeur** qui compose les fragments qui correspondent aux features sélectionnées.

États de l'art

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

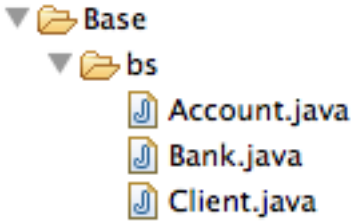
Exemple : **FeatureHouse**

- **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

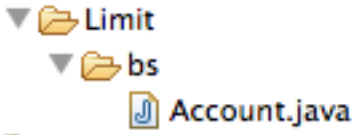
- Une LdP selon FeatureHouse:
  - Un ensemble de fichiers de code source commun (Base)
  - Chaque feature ajoute un raffinement :
    - Ajoutant des nouveaux fichiers de code source
    - Raffine les fichiers communs de base:
      - Ajouter des attributs
      - Ajouter des méthodes/fonctions
     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

Feature : **Base**



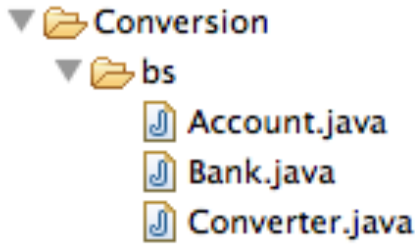
```
package bs;

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..!!");
        }
    }
}
```

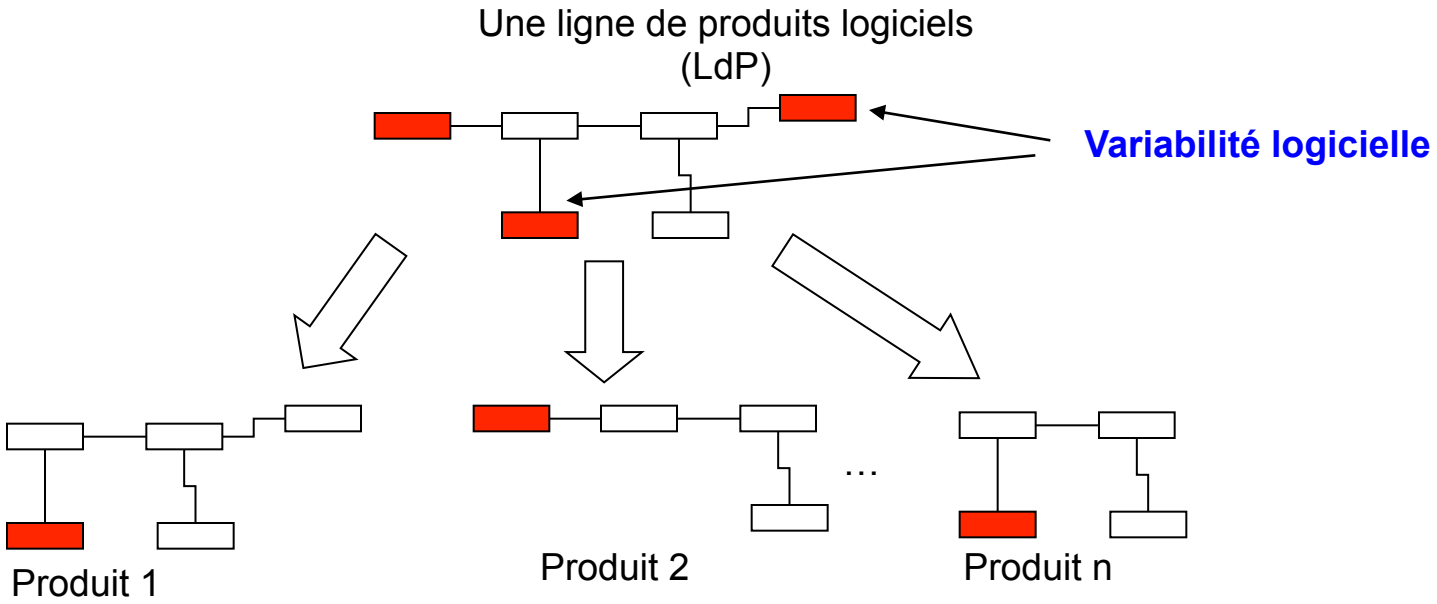
Feature : **Conversion**




```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.**

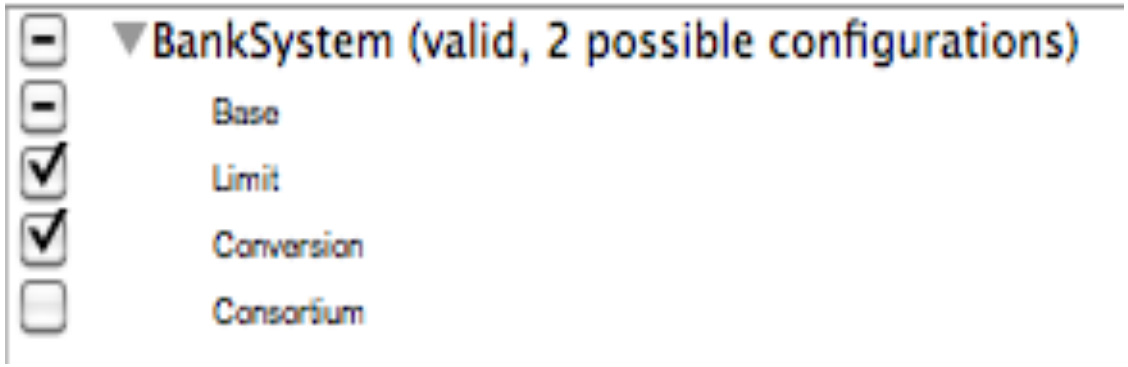
Étape 2: Ingénierie d'application (dérivation de produits)

- 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;
```

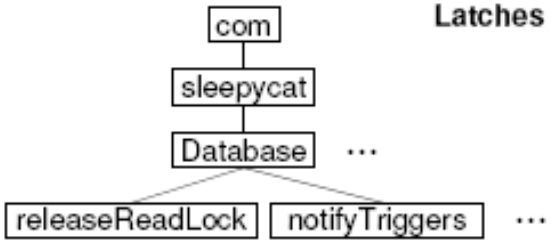
Feature : Base

```
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

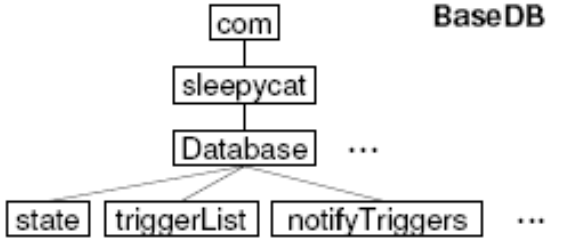
```
package bs;
import java.util.ArrayList;

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

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

    }
}
```

```
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  }
```



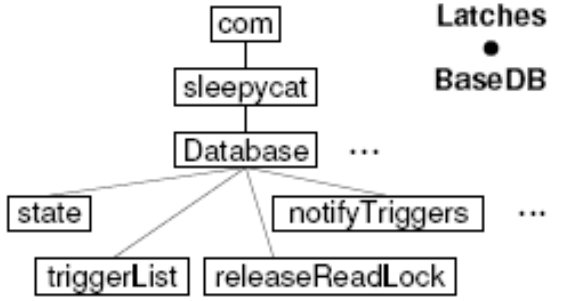
●

```
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

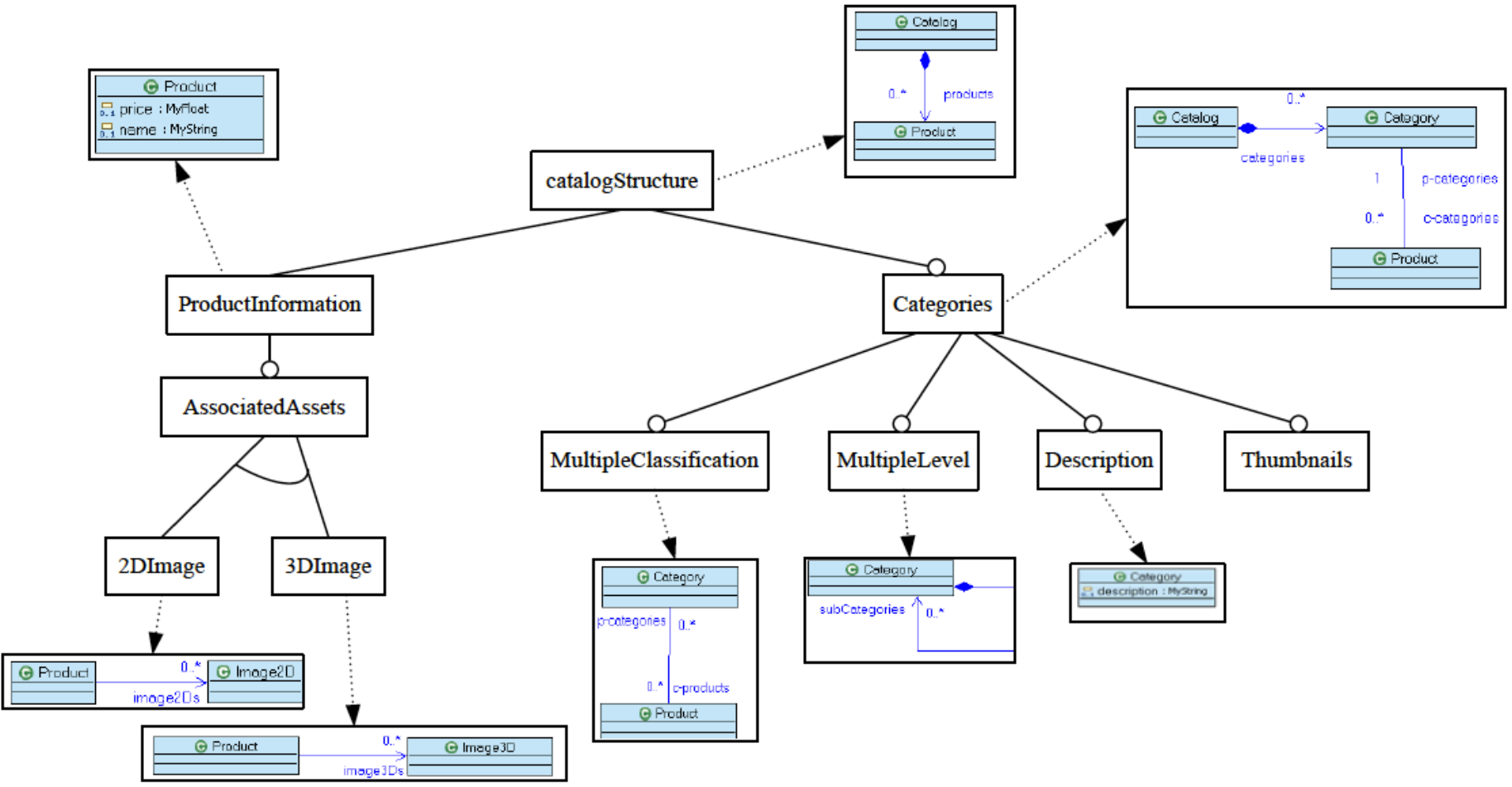
- Éditeur des feature modèles

- Des outils pour les étapes 2 et 3
- La possibilité d'intégrer un mécanisme de composition.

Dans le contexte de l'ingénierie dirigée par les modèles

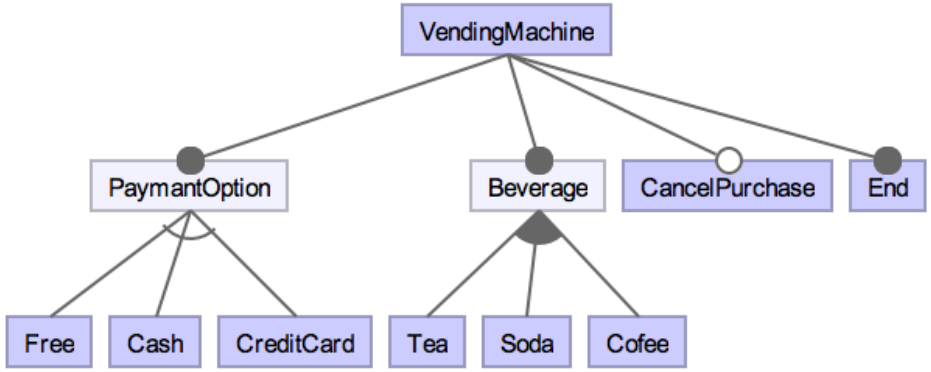
- Principe
  - Associer à chaque feature un modèle ou un fragment
  - Dérivation de modèles de produits = composition de modèles

Exemple de l'approche Kompose [INRIA-Rennes]

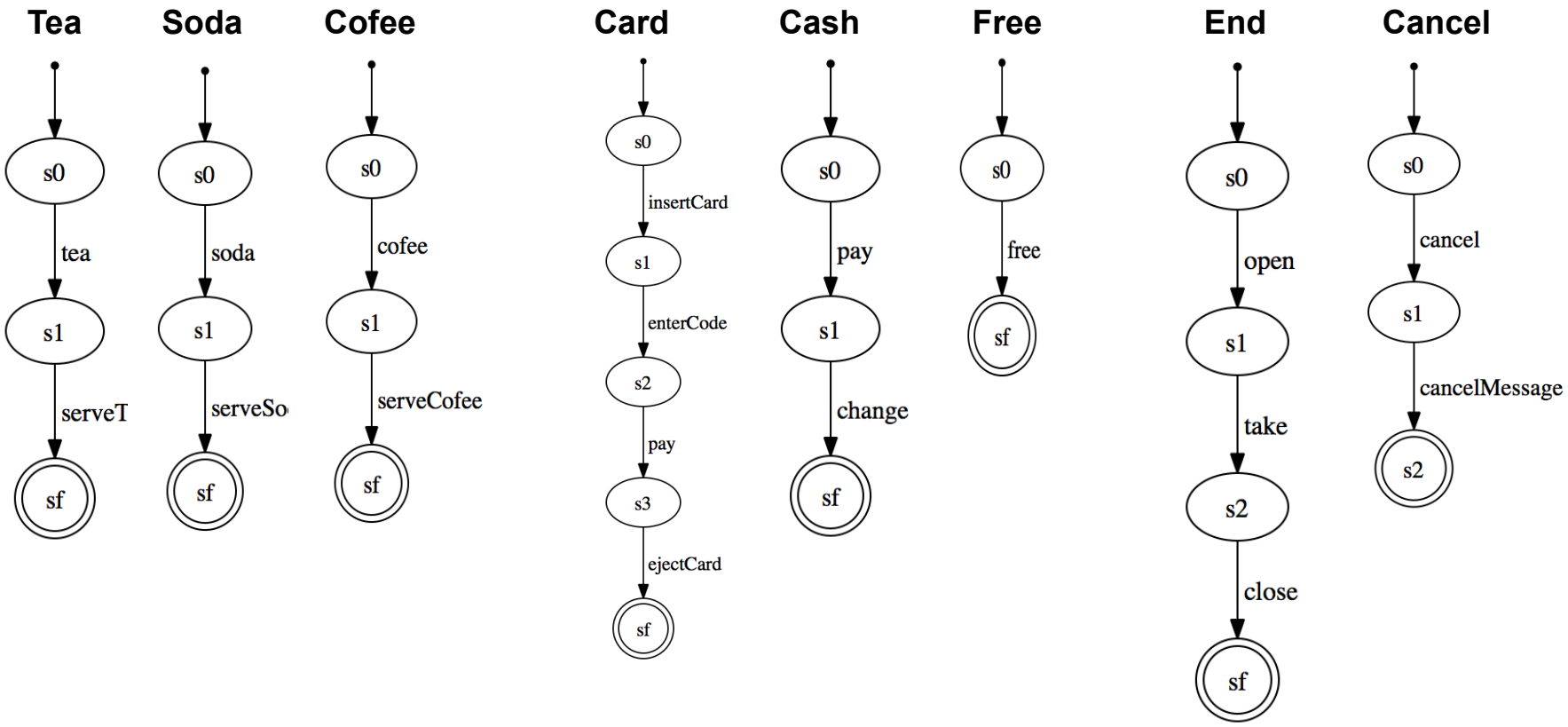


Composition de automates

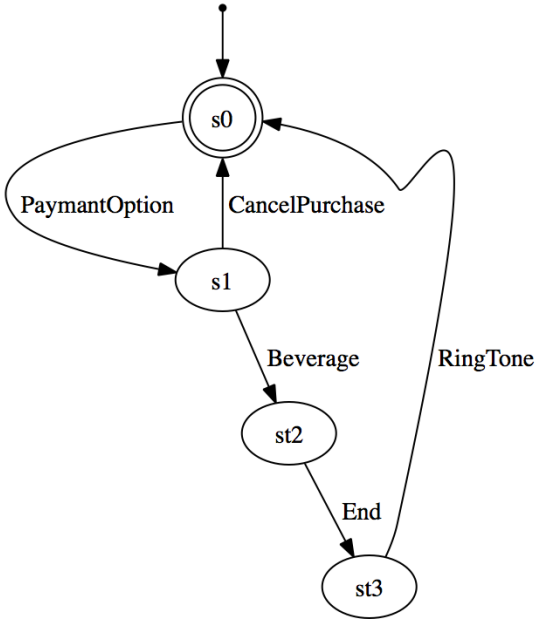
- Un premier travail de recherche réalisé au LIP6.
- **Exemple** : Une famille de distributeurq de boisson
  - **Objectif** : dériver un automate de fonctionnement en fonction de type du distributeur.



Composition des automates



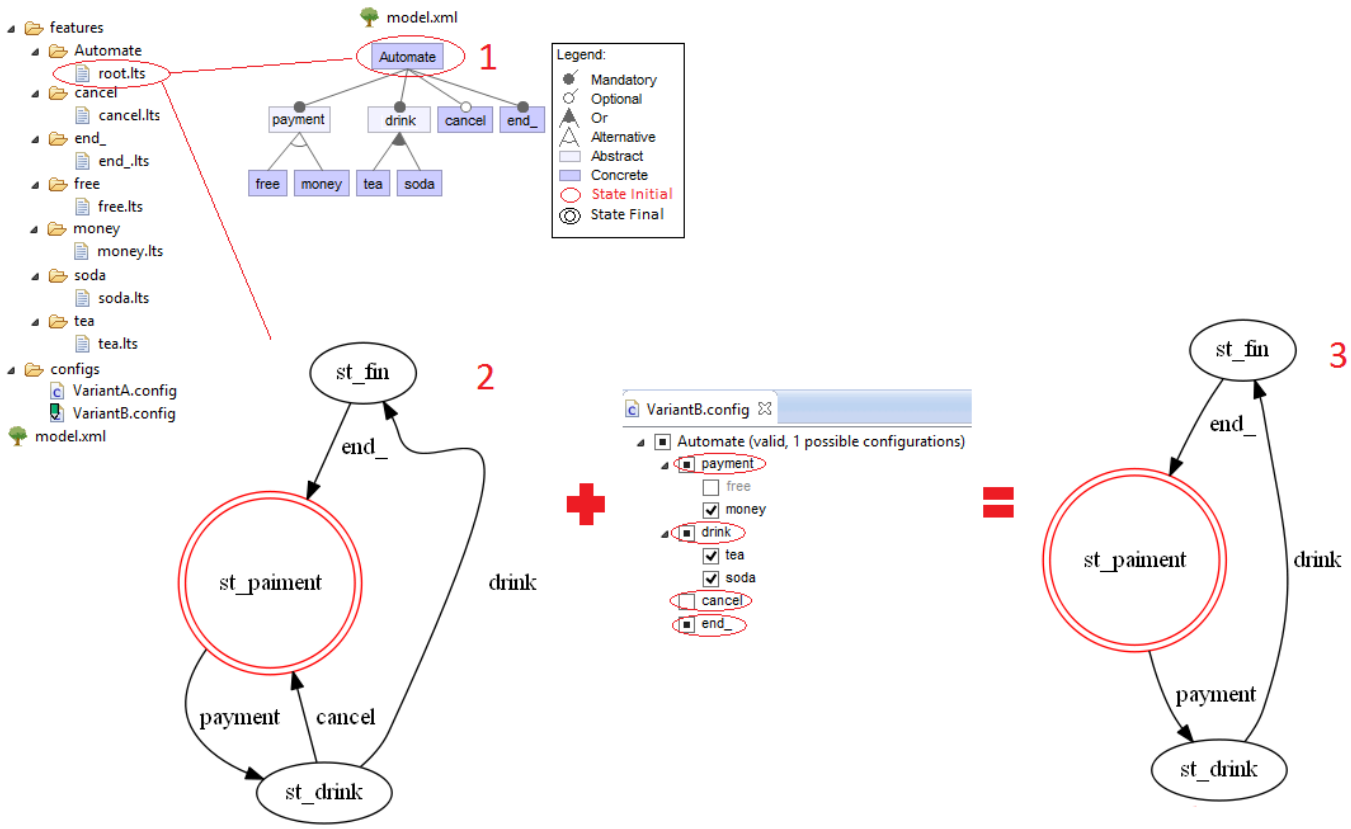
Compositeur des automates



- Algorithme de composition basé sur le rafinement de l'automate globale.

Idée : ajouter un automate globale montrant la composition des features.

Raffinement de l'automate root



Suppression de la transition Cancel

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

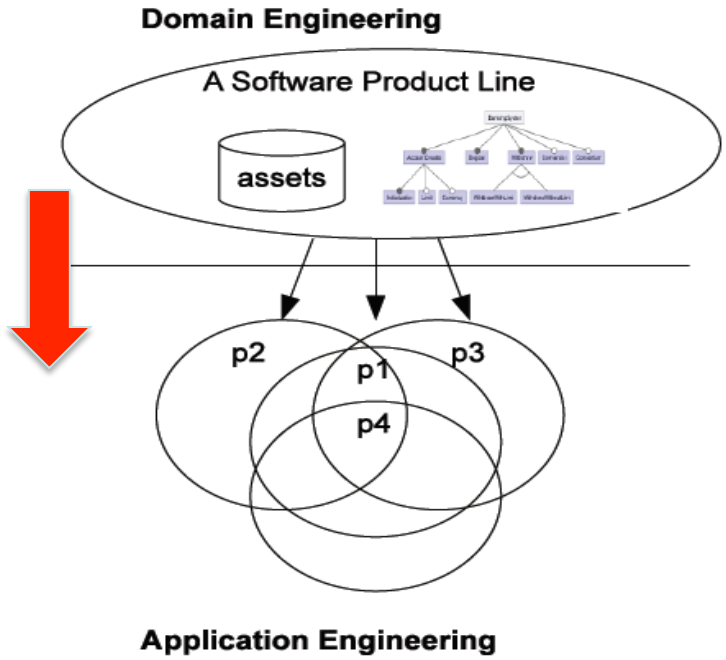
- Un domaine de recherche très active et qui intéresse les industriels
  - Un vrai besoin de gestion de variabilité

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 FeatureHouse et l'outil FeatureIDE
  - Démo: l'outil FeatureIDE
- **Vers une construction automatique de LdP**
  - **Nos premiers résultats de recherche**

SPLE: a top-down process

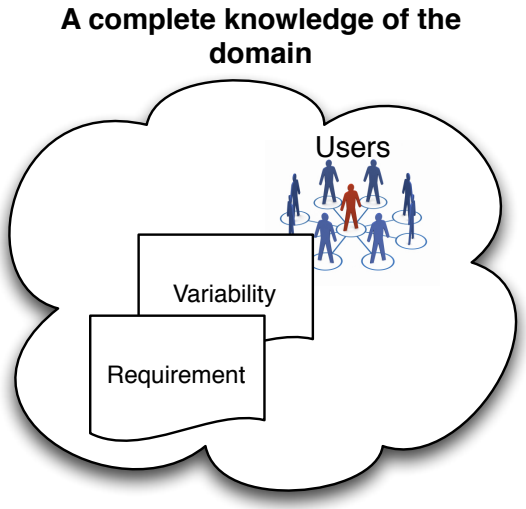
- We implement the SPL and then we derive product variants.



SPLE: a top-down process

- We implement the SPL and then we derive product variants.

« ..**Only 35%** of industrial practitioners reported the use of the top-down process [1].. »

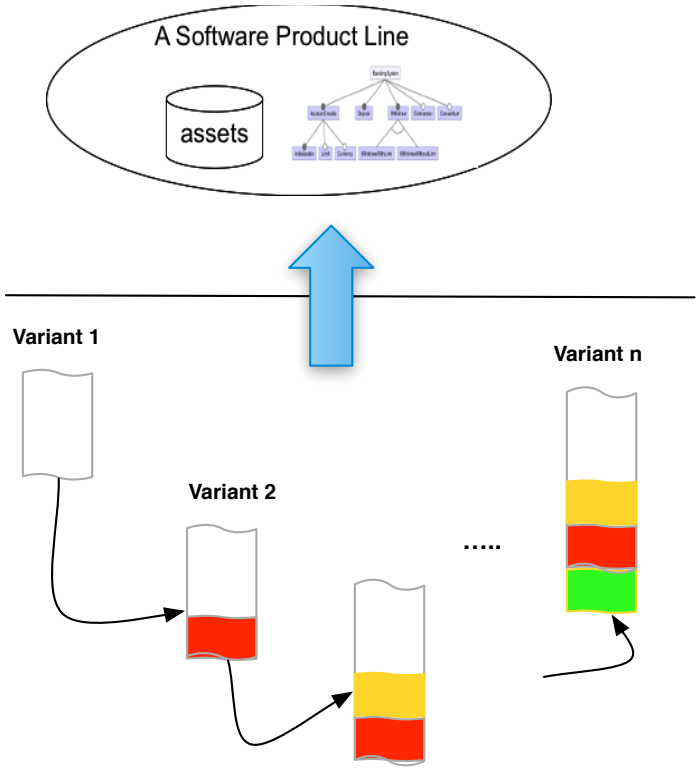


[1] Czarnecki et al. A Survey of Variability Modeling in Industrial Practice, VaMoS2013.

Extractive Approach

- More than **50%** of industrial practitioners implement the SPL after the implementation of several similar **product variants** using ad-hoc reuse techniques [1].

- However this migration is often performed manually [1].



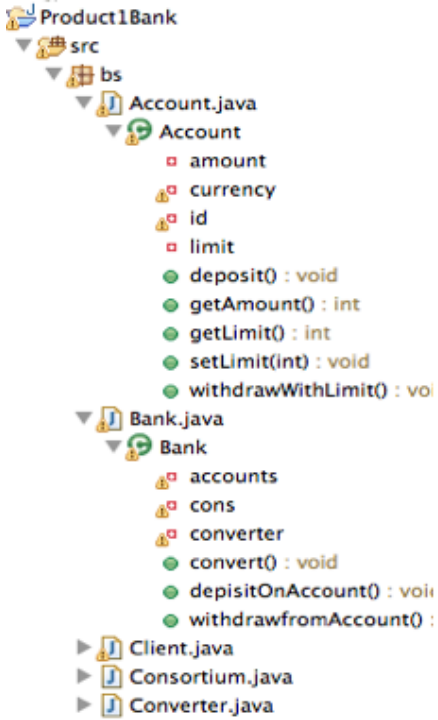
Research Questions

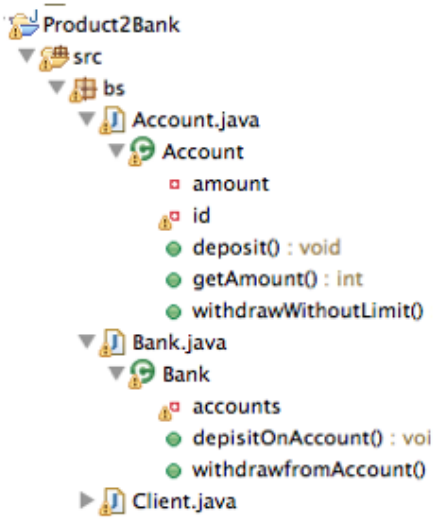
- How can product variants can be **migrated** into a product line?
  - Can we **extract the feature model** ?
  - Can we **extract assets**?

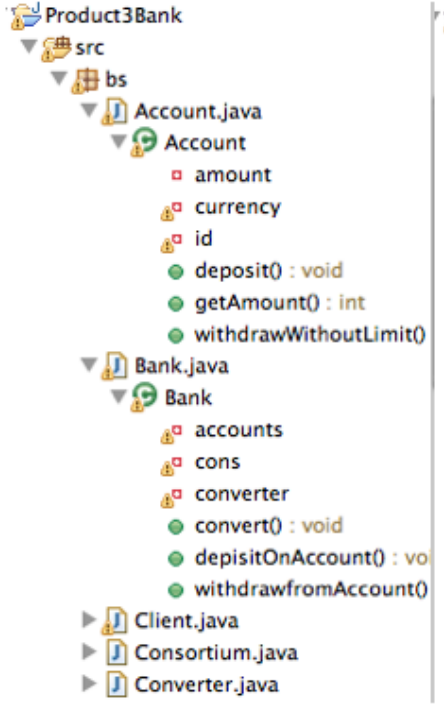
Motivations

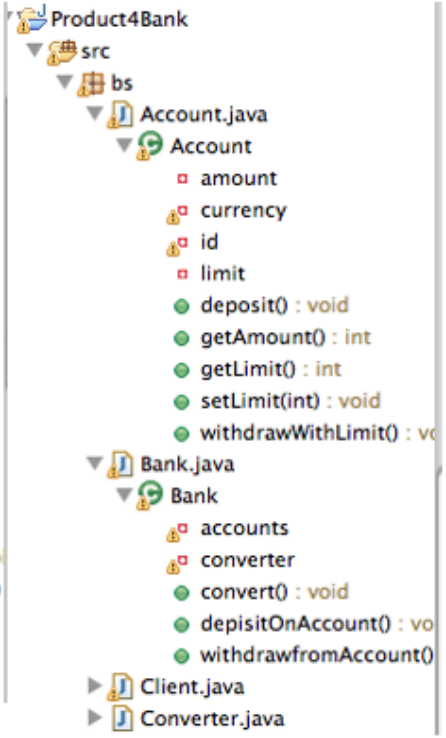
- How can product variants can be **migrated** into a product line?
-
  - Can we **extract the feature model** ?
  - Can we **extract assets**?

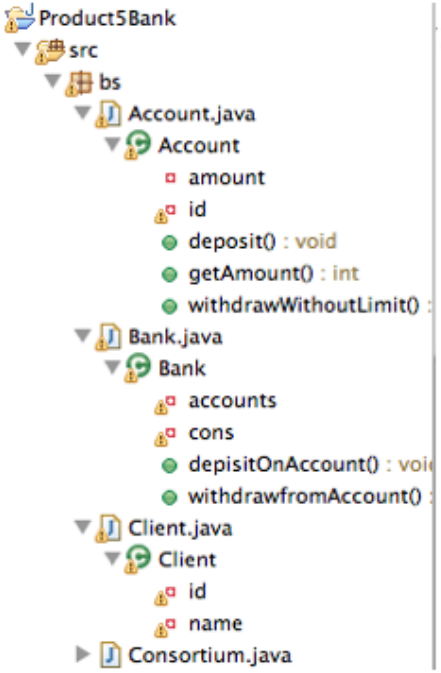
(from the source code)

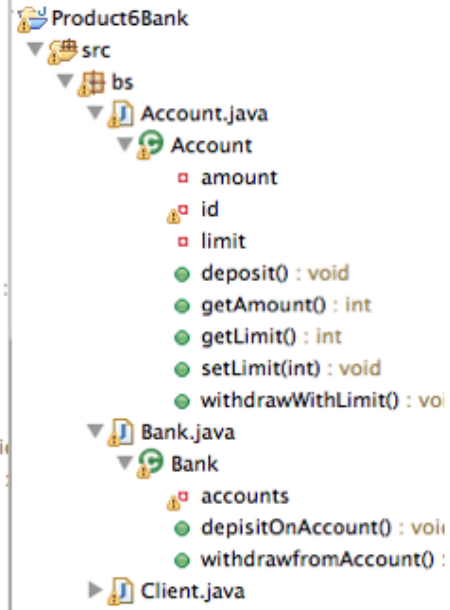

Product1Bank
src
bs
Account.java
Account
amount
currency
id
limit
deposit() : void
getAmount() : int
getLimit() : int
setLimit(int) : void
withdrawWithLimit() : voi
Bank.java
Bank
accounts
cons
converter
convert() : void
depisitOnAccount() : voi
withdrawfromAccount() :
Client.java
Consortium.java
Converter.java

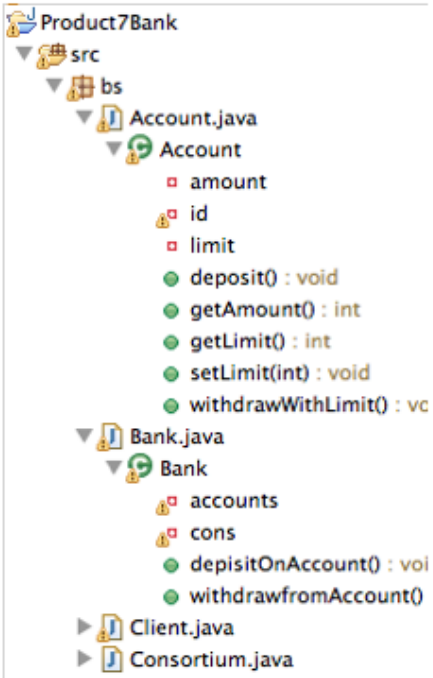

Product2Bank
src
bs
Account.java
Account
amount
id
deposit() : void
getAmount() : int
withdrawWithoutLimit()
Bank.java
Bank
accounts
depisitOnAccount() : voi
withdrawfromAccount()
Client.java

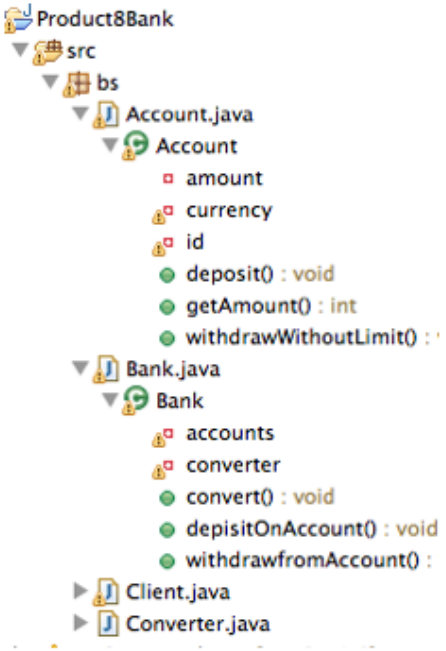

Product3Bank
src
bs
Account.java
Account
amount
currency
id
deposit() : void
getAmount() : int
withdrawWithoutLimit()
Bank.java
Bank
accounts
cons
converter
convert() : void
depisitOnAccount() : vo
withdrawfromAccount()
Client.java
Consortium.java
Converter.java


Product4Bank
src
bs
Account.java
Account
amount
currency
id
limit
deposit() : void
getAmount() : int
getLimit() : int
setLimit(int) : void
withdrawWithLimit() : vo
Bank.java
Bank
accounts
converter
convert() : void
depisitOnAccount() : vo
withdrawfromAccount()
Client.java
Converter.java


Product5Bank
src
bs
Account.java
Account
amount
id
deposit() : void
getAmount() : int
withdrawWithoutLimit() :
Bank.java
Bank
accounts
cons
depisitOnAccount() : voi
withdrawfromAccount() :
Client.java
Client
id
name
Consortium.java

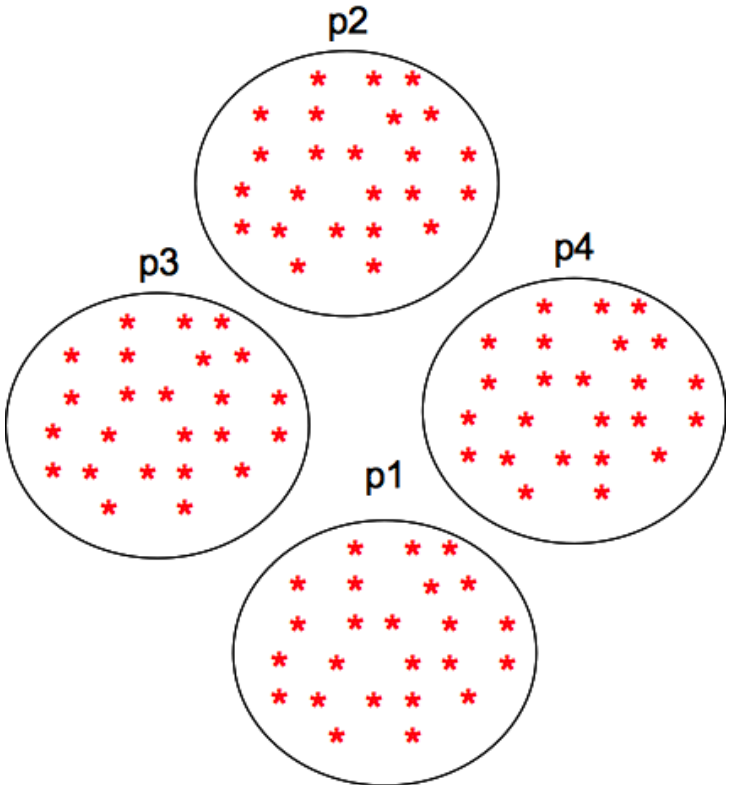

Product6Bank
src
bs
Account.java
Account
amount
id
limit
deposit() : void
getAmount() : int
getLimit() : int
setLimit(int) : void
withdrawWithLimit() : voi
Bank.java
Bank
accounts
depisitOnAccount() : voi
withdrawfromAccount() :
Client.java


Product7Bank
src
bs
Account.java
Account
amount
id
limit
deposit() : void
getAmount() : int
getLimit() : int
setLimit(int) : void
withdrawWithLimit() : vc
Bank.java
Bank
accounts
cons
depisitOnAccount() : voi
withdrawfromAccount()
Client.java
Consortium.java


Product8Bank
src
bs
Account.java
Account
amount
currency
id
deposit() : void
getAmount() : int
withdrawWithoutLimit() :
Bank.java
Bank
accounts
converter
convert() : void
depisitOnAccount() : void
withdrawfromAccount() :
Client.java
Converter.java

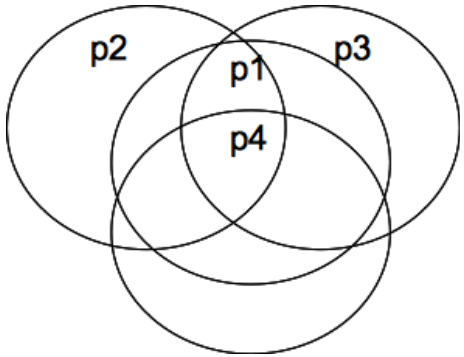
Our claim

- Product variants are represented as sets of *atomic pieces.*
- Equivalence between these atomic pieces.



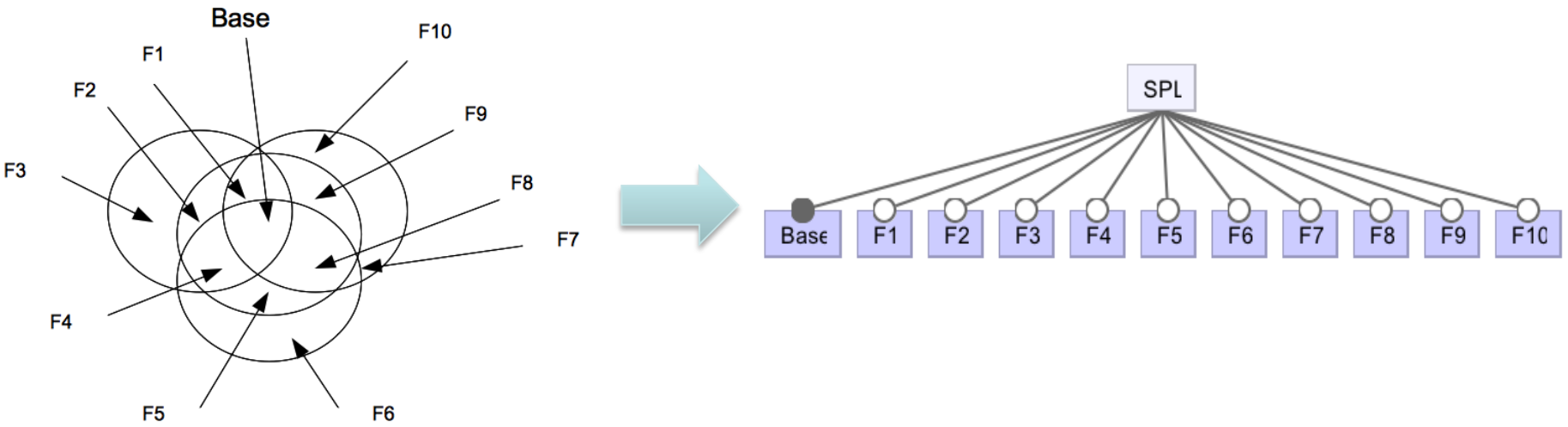
Our claim

- Product variants are represented as sets of *atomic pieces*
- Equivalence between atomic pieces.



Our claim

- Product variants are represented as sets of *atomic pieces.*
- Equivalence between atomic pieces.



Requirements

- How to define the atomic pieces?
- How to define the equivalence relationship?

  - CSMR12 paper:
    - **Construction Primitives** to abstract products variants

Tewfik Ziadi and al. Feature Identification from the Source Code of Product Variants. The European Conference on Software Maintenance and Reengineering, IEEE/CSMR 2012

  - Recent results:
    - The use of **Feature Set Trees (FST)**

CSMR12 paper: Our approach(1/2)

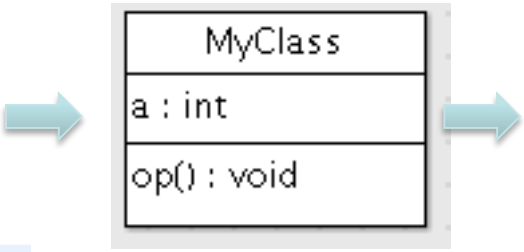
Step 1: Product Abstraction

Input: the source code product variants

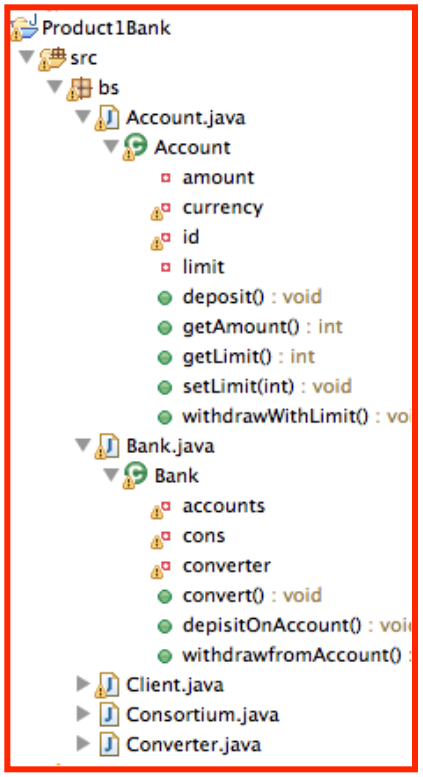
Output: Set of Construction Primitives (CSs)-one per product variant.

Idea: Extract a class diagram and represent it as a set of CPs.

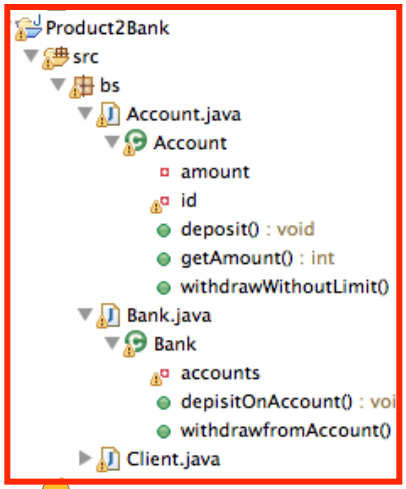
```
public class MyClass {

    private int a1;
    public void op(){

    }
}
```



SoCPs1= {
CreateClass(MyClass,(default package))
CreateAttribute(a, MyClass),
CreateOperation(op,MyClass)
}



P_1^{Bank} = {
CreatePackage(bs),
CreateClass(Converter),
CreateOperation(conv,Converter),
CreateClass(Bank),
CreateAtribute(accounts,Bank),
CreateAtribute(converter,Bank),
CreateAtribute(cons,Bank),
CreateOperation(depisitOnAccount,Bank),
CreateOperation(withdrawfromAccount,Bank),
CreateOperation(convert,Bank),
CreateClass(Consortium),
CreateClass(Client),
CreateAtribute(name,Client),
CreateAtribute(id,Client),
CreateClass(Account),
CreateAtribute(id,Account),
CreateAtribute(currency,Account),
CreateAtribute(limit,Account),
CreateAtribute(amount,Account),
CreateOperation(deposit,Account),
CreateOperation(withdrawWithLimit,Account),
CreateOperation(getAmount,Account),
CreateOperation(getLimit,Account),
CreateOperation(setLimit,Account),
}



P_2^{Bank} = {
CreatePackage(bs),
CreateClass(Bank),
CreateAtribute(accounts,Bank),
CreateOperation(depisitOnAccount,Bank),
CreateOperation(withdrawfromAccount,Bank),
CreateClass(Account),
CreateAtribute(id,Account),
CreateAtribute(amount,Account),
CreateOperation(deposit,Account),
CreateOperation(withdrawWithLimit,Account),
CreateOperation(getAmount,Account),
CreateClass(Client),
CreateAtribute(name,Client),
CreateAtribute(id,Client),
}

Our approach(2/2)

Step 2: Feature identification

Input: set of CPs

Output: a set if features.

Idea: A Feature is an **equivalence class** of interdependent sets of CPs.

➔ «Two *CPs are Interdependent iff:*

they belong to exactly the same product variants »

➔ Matching names to define equivalence between CPs

➔ A feature is a set of CPs.

Inputs

AllP = {P1, P2, P3}

P1 = {a,b,c,d,e,k}

P2 = {a,b,c,d,e,g,h}

P3 = {a,b,k,g,h}

Initialization

R ={ (1,a),(1,b),(1,c),(1,d),(1,e),(1,k),
(2,a),(2,b),(2,c),(2,d),(2,e), (2,g),(2,h),
(3,a),(3,b),(3,k),(3,g),(3,h) }

Iteration 1

mfcp = a

products = {P1, P2, P3}

f = {a,b}

R = { (1,c),(1,d),(1,e),(1,k),
(2,c),(2,d),(2,e),(2,g),(2,h),
(3,k),(3,g),(3,h) }

iteration 2

mfcp = c

products = {P1, P2}

f = {c,d,e}

R = {(1,k),
(2,g),(2,h),
(3,k),(3,g),(3,h) }

Iteration 3

mcfp = k

products = {P1, P3}

f = {k}

R = { (2,g),(2,h),
(3,g),(3,h) }

teration 4

mfcp = g

products = {P2, P3}

f = {g,h}

R = {}

Result

F = {{a,b}, {c,d,e}, {k},{g,h} }

Evaluation: Banking SPL

```
Base = {
  CreatePackage(bs),
  CreateClass(Bank,bs),
  CreateAttribute(accounts,Bank),
  CreateOperation(depisitOnAccount,Bank),
  CreateOperation(withdrawfromAccount,Bank),
  CreateClass(Account,bs),
  CreateAttribute(id,Account),
  CreateAttribute(amount,Account),
  CreateOperation(deposit,Account),
  CreateOperation(getAmount,Account),
  CreateClass(Client,bs),
  CreateAttribute(name,Client),
  CreateAttribute(id,Client)
 }

F1 ={
  CreateAttribute(currency,Account),
  CreateClass(Converter,bs),
  CreateOperation(conv,Converter),
  CreateAttribute(converter,Bank),
  CreateOperation(convert,Bank)
  }
F2 = {
  CreateAttribute(limit,Account),
  CreateOperation(withdrawWithLimit,Account),
  CreateOperation(getLimit,Account),
  CreateOperation(setLimit,Account),
  }
F3 = {
  CreateOperation(withdrawWithoutLimit,Account)
  }
F4 ={
  CreateClass(Consortium,bs)
  CreateAttribute(cons,Bank)
}
```

Limit

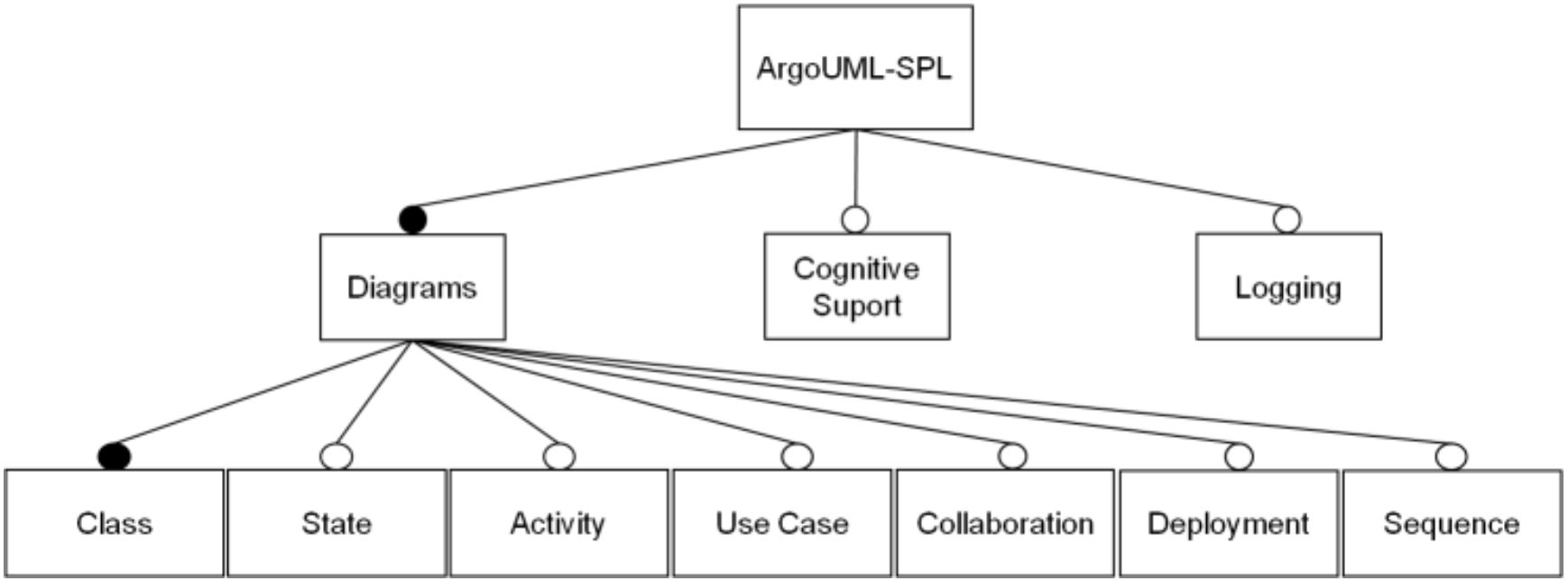
Withdrawwithoutlimit

	Base	F1	F2	F3	F4
P1	X	X	X	X	
P2	X				
P3	X	X	X		X
P4	X	X		X	
P5	X		X		X
P6	X			X	
P7	X		X	X	
P8	X	X		X	X

Mandatory

Optional

Evaluation: ArgoUML-SPL



Source : Marcus Vinícius Couto; Marco Túlio Valente; Eduardo Figueiredo. Extracting Software Product Lines: A Case Study Using Conditional Compilation. CSMR, p. 191-200, 2011.

Evaluation: ArgoUML-SPL

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

Evaluation: ArgoUML-SPL (contd.)

Feature	Name	# Cons. Primitiv	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
1	Base	11223	X	X	X	X	X	X	X	X	X	X
2	Diagram	151	X		X	X	X	X	X	X	X	X
3	Logging	175	X		X	X	X		X	X	X	X
4	Cognitive	1663	X			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		X
7	Collab	110	X		X			X	X	X	X	X
8	State	479	X		X	X		X	X	X	X	X
9	Deploy	219	X			X			X	X	X	X
10	Activity	243			X	X			X		X	X
11	JunctionCognitiveLogging	37	X		X	X		X	X		X	X
12	JunctionSequenceLogging	12	X			X		X	X	X	X	X
13	JunctionCollabSequence	1	X					X	X		X	X
14	JunctionCognitiveDeploy	67	X		X	X			X	X	X	X
15	JunctionCognitiveSequence	5	X		X	X			X	X	X	X
16	JunctionCollabLogging	2	X		X				X	X	X	X
17	JunctionDeployLogging	5	X		X	X			X	X	X	X
18	JunctionStateLogging	1	X		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	X
21	JunctionActivityState	9			X	X	X	X	X	X		X

Evaluation: ArgoUML-SPL (contd.)

```
F3: Cognitive
  CreateClass(Goal,cognitive)
  CreateAttribute(name,Goal)
  CreateAttribute(priority,Goal)
  CreateClass(ToDoItem,cognitive)
  CreateAttribute(INTERRUPTIVE_PRIORITY,ToDoItem)
  CreateAttribute(HIGH_PRIORITY,ToDoItem)
  CreateAttribute(MED_PRIORITY,ToDoItem)
  CreateAttribute(LOW_PRIORITY,ToDoItem)
  ...

F8: State
  ...
  CreateClass(InitStateDiagram,ui)
  CreateOperation(getDetailsTabs,InitStateDiagram)
  CreateOperation(getProjectSettingsTabs,InitStateDiagram)
  CreateOperation(getSettingsTabs,InitStateDiagram)
  CreateOperation(init,InitStateDiagram)
  ...
```

Evaluation: ArgoUML-SPL (contd.)

Feature	Name	# Cons. Primitiv	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
1	Base	11223	X	X	X	X	X	X	X	X	X	X
2	Diagram	151	X		X	X	X	X	X	X	X	X
3	Logging	175	X		X	X	X		X	X	X	X
4	Cognitive	1663	X			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		X
7	Collab	110	X		X			X	X	X	X	X
8	State	479	X		X	X		X	X	X	X	X
9	Deploy	219	X			X			X	X	X	X
10	Activity	243			X	X			X		X	X
11	JunctionCognitiveLogging	37	X		X	X		X	X		X	X
12	JunctionSequenceLogging	12	X			X		X	X	X	X	X
13	JunctionCollabSequence	1	X					X	X		X	X
14	JunctionCognitiveDeploy	67	X		X	X			X	X	X	X
15	JunctionCognitiveSequer											
16	JunctionCollabLogging											
17	JunctionDeployLogging											
18	JunctionStateLogging											
19	JunctionUseCaseLoggin											
20	JunctionActivityLogging											
21	JunctionActivityState											

F20: JunctionActivityLogging

{

CreateAttribute(LOG,UMLActivityDiagram)

}

Recent directions

- Our construction primitives can be only used for object-oriented systems.
- CPs do not allow the extraction of assets (the complete source code of assets)

- A compositional implementation of the SPL

Recent directions

- Our construction primitives can be only used for object-oriented systems.
- CPs do not allow the extraction of assets (the complete source code of assets)

The need of a new formalism to specify atomic pieces

- An abstract representation of the source code of product variants
- With all informations to generate assets.

Recent directions

- Our construction primitives can be only used for object-oriented systems.
- CPs do not allow the extraction of assets (the complete source code of assets)

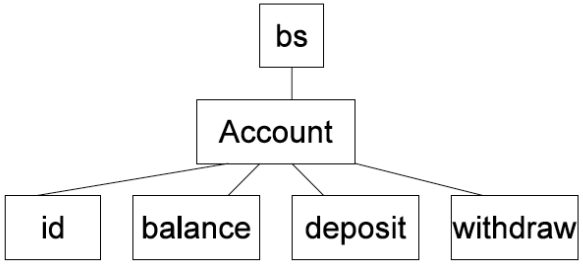
The need of a new formalism to specify atomic pieces

- An abstract representation of the source code of product variants
- With all information to generate assets.

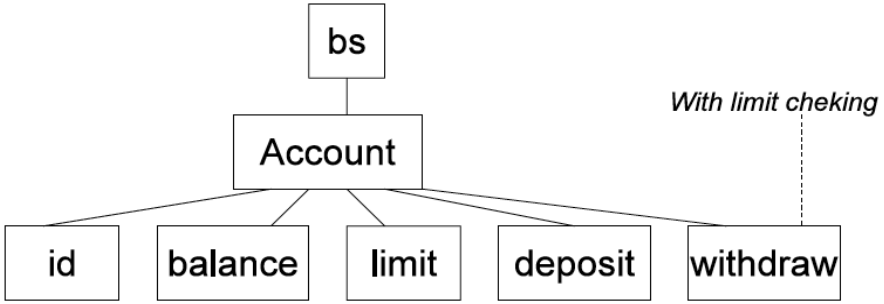
➔FeatureHouse: Feature Set Trees (FST) [1]

[1] Sven Apel, and al. Language-Independent and Automated Software Composition: The FEATUREHOUSE Experience, IEEE-TSE, 2012

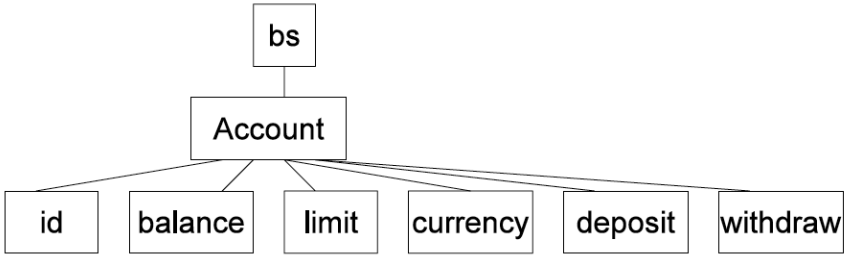
```java
package bs;
public   class   Account {
      private String id;
      private double balance;
      public void deposit(double
amount) {
            this.balance +=
amount;
      }
      public void withdraw(double
amount) {
            if (amount <= balance)
                  balance -=
amount;
      }
}
```




```java
package bs;
public     class   Account {
      private String id;
      private double balance;
      private double limit;
      public void deposit(double
amount)    {
            this.balance +=
amount;
      }
      public void withdraw
(double amount) {
      if (amount <= balance +
limit)
            balance -= amount;
      }
      public double getLimit() {
            return limit;
      }
}
```



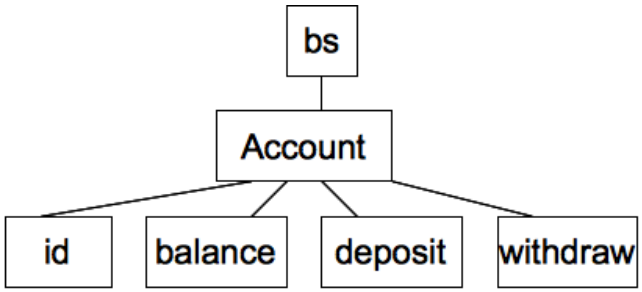

```java
package bs;
public     class   Account {
      private String id;
      private double balance;
      private double limit;
      private double currency;
      public void deposit(double
amount)    {
            this.balance +=
amount;
      }
      public void withdraw
(double amount) {
      if (amount <= balance +
limit)
            balance -= amount;
      }
      public double getLimit() {
            return limit;
      }
}
```



T.Ziadi

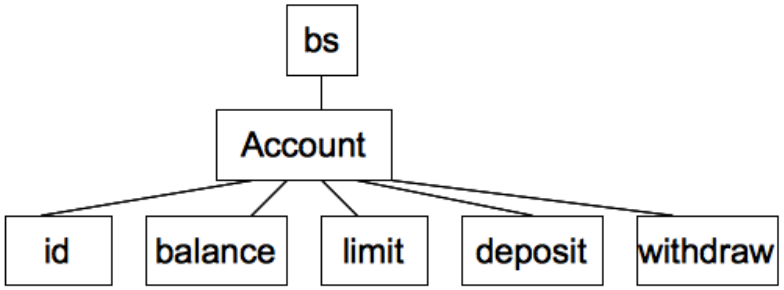
UPMC PARIS UNIVERSITAS

cnrs dépasser les frontières



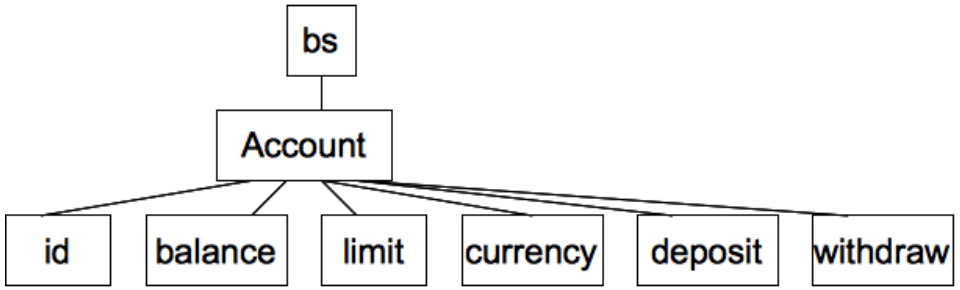
a) FST of Bank1Product

```
P3 Bank= {
    CreateNonTerminal(bs, package, (Account)),
    CreateNonTerminal(Account, Class, bs,
      (id, balance, deposit, withdraw)),
    CreateTerminal(id,filed, Account),
    CreateTerminal(balance,field, Account),
    CreateTerminal(deposit,method, Account),
    CreateTerminal(withdraw,method, Account),
}
```



b) FST of Bank2Product

```
P3 Bank= {
    CreateNonTerminal(bs, package, (Account)),
    CreateNonTerminal(Account, Class, bs,
      (id, balance, deposit, withdraw)),
    CreateTerminal(id,filed, Account),
    CreateTerminal(balance,field, Account),
    CreateTerminal(limit,,filed, Account),
    CreateTerminal(deposit,method, Account),
    CreateTerminal(withdraw,method, Account),
    CreateTerminal(getLimitw,method, Account),
}
```



c) FST of Bank3Product

```
P3 Bank= {
    CreateNonTerminal(bs, package, (Account)),
    CreateNonTerminal(Account, Class, bs,
      (id, balance, deposit, withdraw)),
    CreateTerminal(id,filed, Account),
    CreateTerminal(balance,field, Account),
    CreateTerminal(limit,,filed, Account),
    CreateTerminal(currency,,filed, Account),
    CreateTerminal(deposit,method, Account),
    CreateTerminal(withdraw,method, Account),
    CreateTerminal(getLimitw,method, Account),
}
```

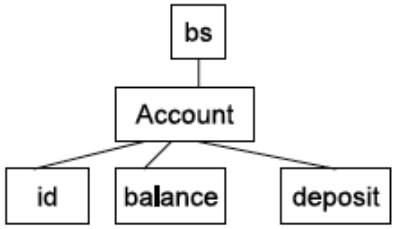
Features as FSTs

```
Base= {
    CreateNonTerminal(bs, package, (Account)),
    CreateNonTerminal(Account, Class, bs,
      (id, balance, deposit, withdraw)),
    CreateTerminal(id,filed, Account),
    CreateTerminal(balance,field, Account),
    CreateTerminal(deposit,method, Account),

}
```

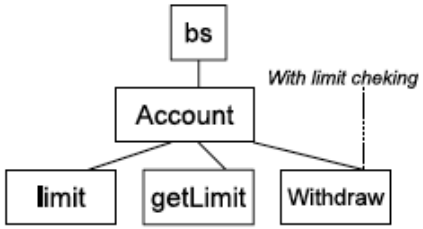
```
F1= {
    CreateTerminal(limit,,filed, Account),
    CreateTerminal(withdraw,method, Account),
    CreateTerminal(getLimitw,method, Account)
}
```

```
F2= {

    CreateTerminal(currency,,filed, Account)

}
```

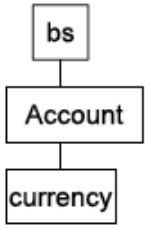
```
F3= {

    CreateTerminal(withdraw,method, Account)

}
```



a) FST of the Base feature

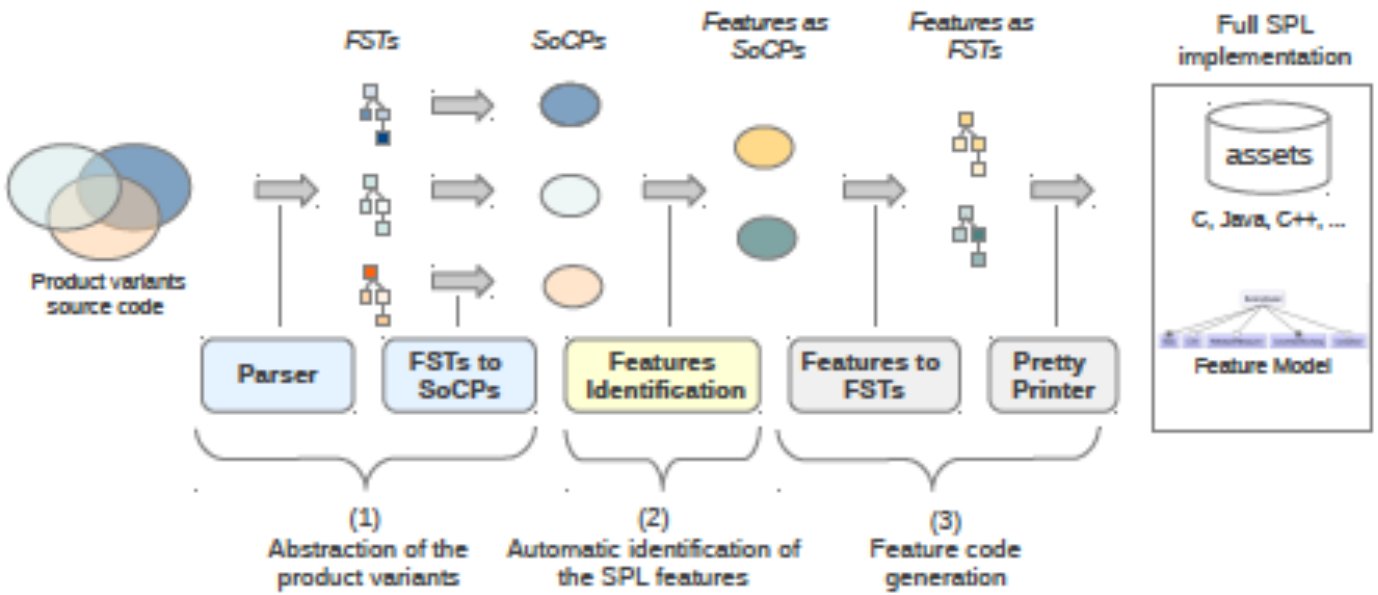


b) FST of the F1 feature



c) FST of the F2 eature

The approach



Discussion and Evaluation

- ExtractorPL supporting Java, C, C#.
- Evaluation
  - ArgoUML
  - Examples of existing SPL(FeatureIDE)
    - GPL (Java), Notepad(Java), BarkeleyDB (Java), Mail Systems (C)
- Favorable case studies?

Conclusions and Future work

- **Summary**
  - A bottom-up approach to reverse-engineering of SPL: **first results.**
  - **Only for product variants created using copy-paste-modify**
- **Perspectives**
  - Considering the behavior aspect (body of operations)
  - Forks in software repositories as input product variants?
  - Releases of a software as a product poruduct variants?