Vers un développement formel non incrémental

Thi-Kim-Dung Pham (CNAM-Cedric)
Catherine Dubois (ENSIIE-Samovar)
Nicole Levy (CNAM-Cedric))

Contexte : Lignes de produits logicielles

Décliner les programmes en une kyrielle de variantes pour répondre spécifiquement à certaines exigences des clients

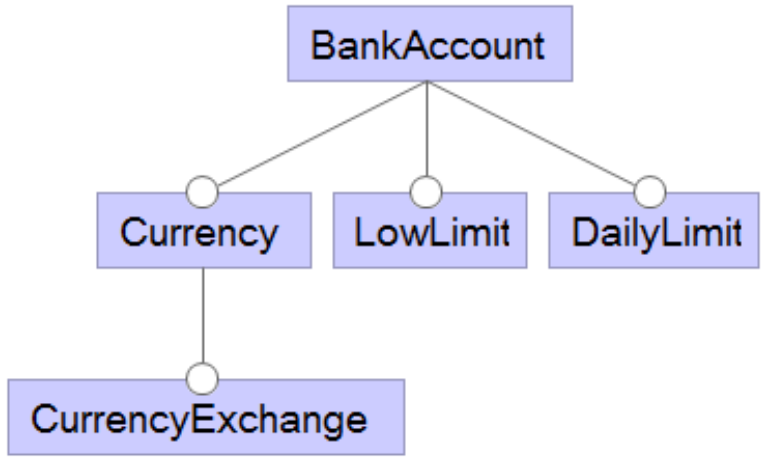
a set of software-intensive systems that share a common managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [Clements et al., 2001]

$\longrightarrow$ Construire des variantes correctes par construction

Feature / diagramme de features

- Feature (définition) : capacité d'un produit à couvrir un certain cas d'utilisation ou un certain besoin de l'utilisateur
- Une variante (produit de la famille) est obtenu à partir d'une configuration (ens. de features à combiner)
- Certaines combinaisons n'ont pas de sens : certaines features requièrent/excluent d'autres features.
- Modèle de features définit les combinations légales représentation graphique : arbre/diagramme + relations (optional, mandatory, cardinality, etc)
- A chaque nœud de l'arbre est associé un *asset*, i.e. artefact logiciel nécessaire au développement de produits (code, modèles, tests etc)
- Une variante est obtenue par assemblage des assets associés à sa configuration.

Exemple de ligne de produits



Bankaccount (BA) : fonctionnement de base d'un compte : calculer le solde d'un compte, débiter et créditer un compte.
Le débit n'est autorisé que si le compte reste créditeur d'une certaine somme.

(Daily)Limit (DL) : limitation supérieure pour un retrait

LowLimit (LL) : limitation inférieure pour un retrait

Currency (CU) et CurrencyExchange (CE) : gestion de devises, conversion vers différentes devises

D'une feature mère à une feature fille, on peut :

- ajouter une nouvelle fonctionnalité
- redéfinir une fonctionnalité :
  - à spécification constante ou non,
  - à interface constante ou non
- supprimer une fonctionnalité

$\longrightarrow$ Tout ce qui était vrai avant n'est plus nécessairement vrai après !

Notre objectif : produire des variantes *correctes par construction* en utilisant le langage FoCaLiZe (`http://focalize.inria.fr/`)

Pourquoi FoCaLiZe ? un unique langage pour spécifier, implanter et prouver (asset = ensemble d'espèces FoCaLiZe) + mécanismes pour faciliter la réutilisation (héritage, redéfinition, liaison retardée)

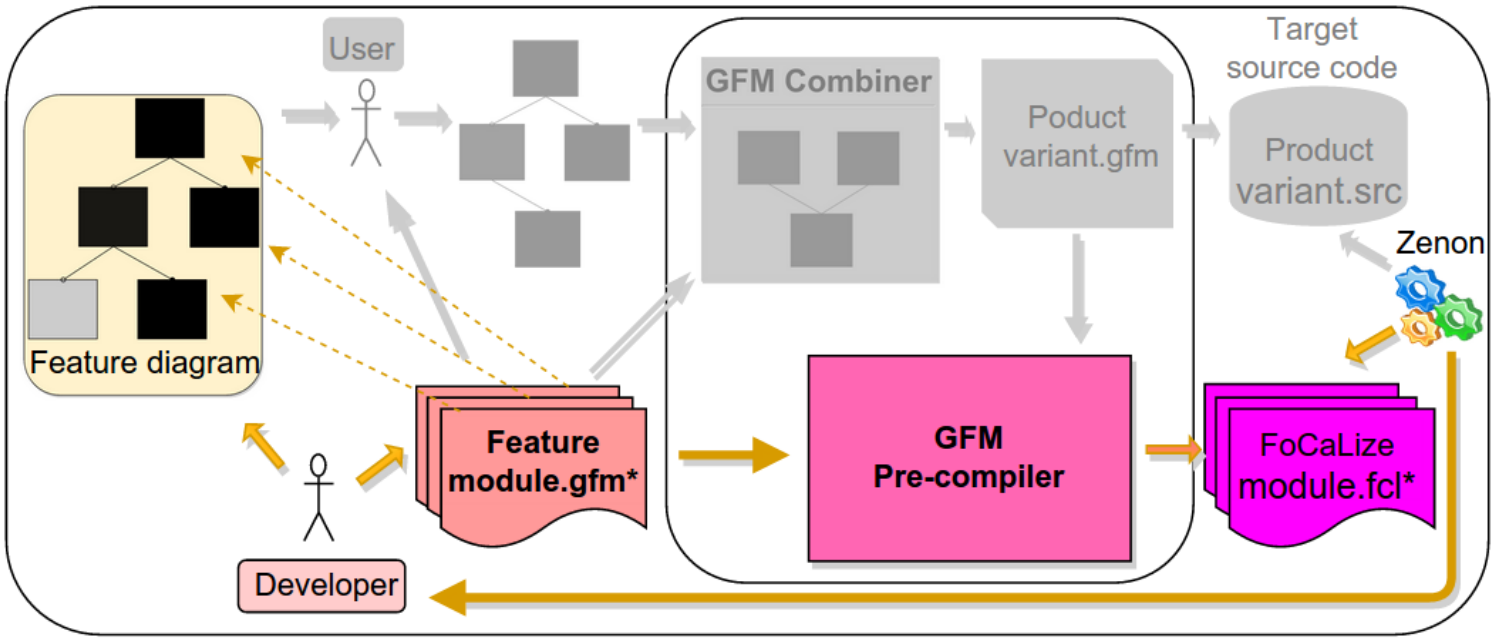
Mais ☹ compliqué : FoCaLiZe pas conçu pour exprimer les changements

Notre approche

$\longrightarrow$ Développement de GFML langage formel pour spécifier, implanter, prouver les features

$\longrightarrow$ Langage inspiré de FoCaLiZe (logique des prédicats, code fonctionnel)

$\longrightarrow$ Compilation en FoCaLiZe, preuves faites sur le code FoCaLiZe avec Zenon (prouveur du 1er ordre)



(processus haut : composition des features - hors sujet ici)

BA en GFML

```
fmodule BA
(* spécification*)
signature update : Self -> int -> Self;
signature balance : Self -> int;
signature over : int;

invariant property ba_over : all  b: Self,  balance(b) >= over;

property ba_update : all b:Self, all a: int,
  balance(b) + a >= over ->
    balance(update(b, a)) = balance(b) + a;

(* implantation *)
representation = int;
let balance(b) = b;
let update(b,a) = if b + a >= over then b + a else b;

(* preuves de correction *)
proof of ba_update =
  {* focalizeproof = by definition of update, balance ;*}
proof of ba_over = assumed ;
end ;;
```

DL en GFML

```
fmodule DL from BA
(* spécification*)
signature limit_withdraw : nat;
property ba_update_ref1 refines BA!ba_update
extends premise (a >= 0);
property ba_update_ref2 refines BA!ba_update
extends a < 0 /\ -a <= limit_withdraw;

(* implantation *)
let update(b,a) = if a < 0
  then if -a <= limit_withdraw then BA!update (b, a) else b
  else BA!update(b, a);

(* preuves de correction *)
proof of ba_update_ref1 =
{* focalizeproof =  by definition of update, balance, over
     property BA!ba_update, int_ge_le_eq  ; *}
proof of ba_update_ref2 =
{* focalizeproof =  by definition of update, balance, over
     property BA!ba_update ; *}
end ;;
```

Traduction vers FoCaLiZe

- BA : 3 espèces : 2 pour la spécification (`BA_spec0` et `BA_spec1`) et 1 pour l'implantation et les preuves (`BA_impl`).

```
species BA_spec0 =
signature update : Self -> int -> Self;
signature balance : Self -> int;
signature over : int;
property ba_over : all  b: Self,  balance(b) >= over;
end ;;

species BA_spec =
inherit BA_spec0;
...
end ;;

species  BA_impl =
inherit BA_spec;
...
end ;;
```

- DL : 3 espèces : 2 pour la spécification (`DL_spec0` et `DL_spec1`) et 1 pour l'implantation et les preuves (`DL_impl`) + paramétrisation.

```
species DL_spec0 =
inherit BA_spec;
signature limit_withdraw : int;
end;;

species DL_spec =
inherit DL_spec0;
property ba_update_ref1 : all b:Self, all a: int,
a >= 0-> balance(b) + a >= over -> balance( update(b, a)) =
balance(b) + a;

property ba_update_ref2 : all b:Self, all a: int,
a < 0 -> (- a) <= limit_withdraw -> ...
end;;
```

```
species DL_impl (BA is BA_impl) =
inherit DL_spec;
representation = BA;
let balance(b) = BA!balance (b) ;
let over = BA!over ;
proof of ba_over = by definition of balance, over property BA!ba_over ;

let update(b,a) =
if a < 0 then if (0 - a) <= limit_withdraw then BA!update (b, a)
              else b
else BA!update(b, a);

proof of ba_update_ref1 =
  by definition of update, balance, over
     property BA!ba_update, int_ge_le_eq  ;

proof of ba_update_ref2 = ...
end;;
```

Conclusion

Approche de la construction non incrémentale de logiciels

- à l'aide d'un langage formel, GFML, qui introduit des opérateurs proches de la variabilité mise en œuvre dans les lignes de produits
- permettant de spécifier, implanter et prouver les étapes de développement
- compilé vers FoCaLiZe

Perspectives (ou en cours de réalisation) :

- composition de features pour obtenir des variantes correctes par construction
- extension de GFML pour prendre en compte d'autres constructions : ajout ou modification d'un paramètre d'une fonction