Méthodologie de construction du logiciel avec B

Modélisation et Construction des logiciels - M3301-2)

J. Christian Attiogbé

UNIVERSITÉ DE NANTES

IUT NANTES Université de Nantes

Méthode B : construction par raffinements

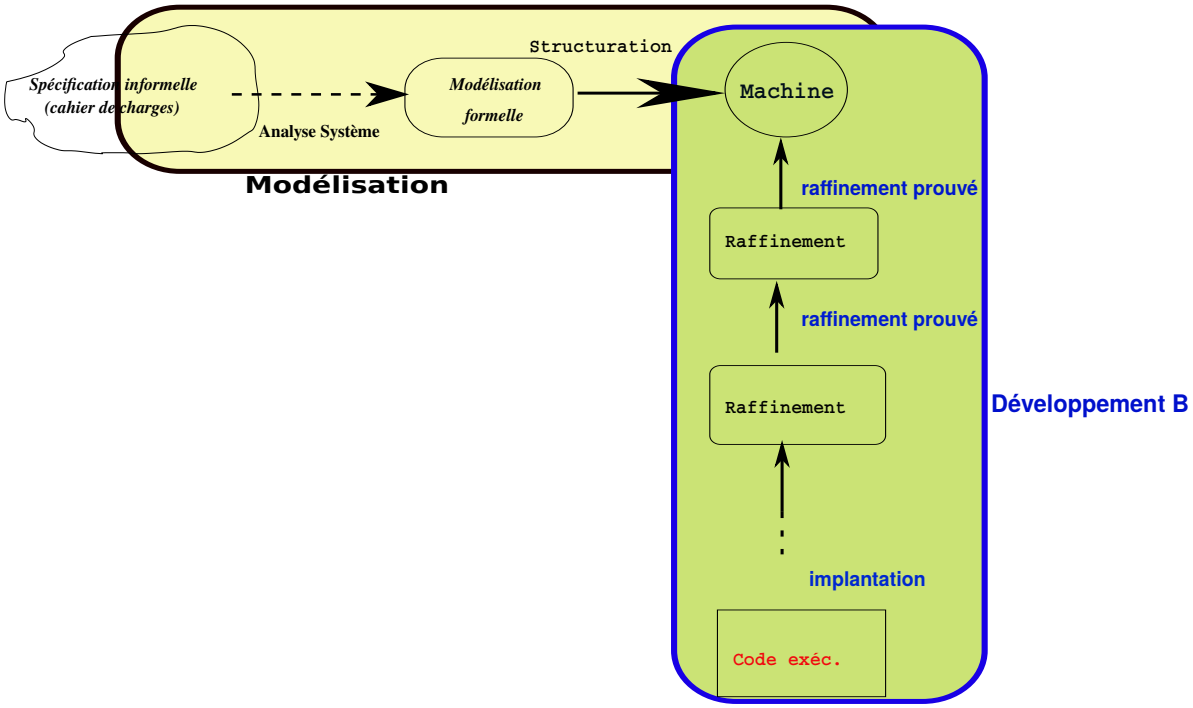


Figure: Analyse et développement B

Raffinement (= passage de l'abstrait au concret)

Raffinement : technique de construction
- on va progressivement d'un modèle abstrait vers un modèle concret
- changement de niveau d'abstraction (en ajoutant des détails)
- choix de structures - données et contrôle (c'est la conception)
- preuves de correction !

Intuition : Raffinement, technique de développement

- On part d'une machine abstraite (modèle mathématique : le quoi),
- on raffine ce modèle pour obtenir un modèle concret :
  - un modèle concret est exécutable = un code
  - pour obtenir ce code, on explicite le comment
    (on fait des choix d'objets/structures informatiques)

= Phases de conception et implantation dans le cycle de vie du logiciel.

Exemple : construction d'une calculette par raffinements successifs.

Plan de la suite

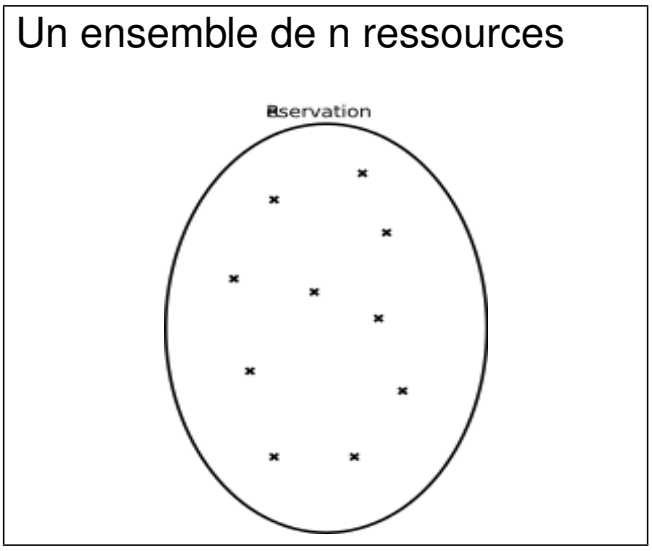
Exemple de raffinement

Exercice :

- Modélisation et développement d'un système de réservation de ressources
- Il y a N ressources à réserver/libérer
- On réserve (allocation) en fonction de la disponibilité des ressources
- les ressources réservées sont par la suite libérées

Exemple : Réservation de ressources

Un ensemble de n ressources



Abstractions:

$n\_rsrc \in 0..100$

n_rsrc = card. de l'ensemble

réserver $\rightarrow$ - 1 élément

libérer $\rightarrow$ + 1 élément

Structuration comme une machine abstraite

```
MACHINE
     Reservation

VARIABLES
     n_rsrc

INVARIANT
     n_rsrc : 0..100

INITIALISATION
     n_rsrc := 100
```

```
 OPERATIONS
reserver =
    PRE  n_rsrc > 0
    THEN
         n_rsrc := n_rsrc - 1
    END
;
liberer =
    PRE  n_rsrc < 100
    THEN
         n_rsrc := n_rsrc + 1
    END
;
bb <-- disponibilite =
     bb :: BOOL
// ou  bb := bool(0 < n_rsrc)
 END
```

Preuve de cohérence

Le développeur d'une machine a deux types d'obligations de preuve :
1) montrer que l'INITIALISATION établit l'invariant :

$$[n\_rsrc := 100](n\_rsrc \in 0..100)$$

Il faut montrer que $100 \in 0..100$

Preuve de cohérence

2) prouver que chaque opération, lorsqu'elle est appelée sous sa PREcondition, préserve l'invariant.

- Pour l'opération ***reserver*** il faut prouver :
  $n\_rsrc \in 0..100 \wedge 0 < n\_rsrc \Rightarrow n\_rsrc - 1 \in 0..100$
- Pour l'opération ***disponibilite*** il faut prouver :
  $n\_rsrc \in 0..100 \wedge (n\_rsrc > 0 \vee \neg (n\_rsrc > 0))$
  $\Rightarrow$
  $n\_rsrc \in 0..100$

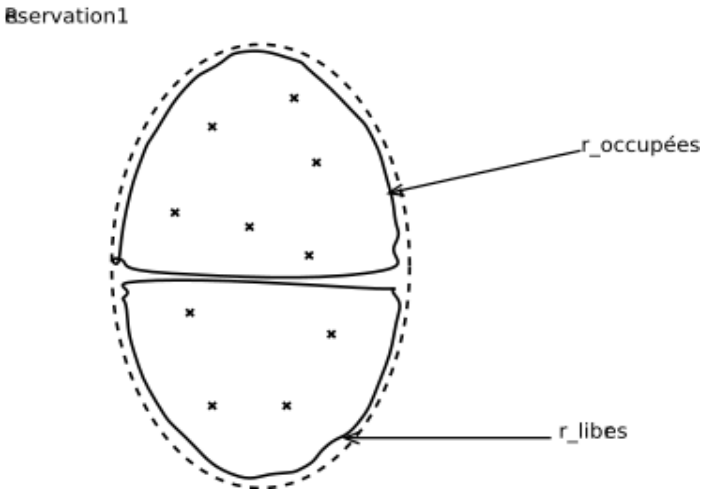
☞ Une fois la machine prouvée cohérente, on la raffine ! allons-y.

Réservation de ressources (Raffinement)

réserver → trouver 1 élément **libre**

libérer → trouver 1 élément **occupé=alloué**

donc il faut distinguer les éléments de l'ensemble, par exemple:




```
REFINEMENT
  Reservation_R1
REFINES
  Reservation
VARIABLES
   r_libres, r_occupees // on prend de nouvelles variables
// nouvelles variables moins abstraites
// la variable n_rscrc est incluse
INVARIANT
   r_libres : POW(INTEGER) // un ens. d'entiers
&  r_occupees : POW(INTEGER)
&  r_libres /\ r_occupees = {} // ils sont disjoints
 & n_rsrc = card(r_libres) // invariant de liaison
INITIALISATION
  r_libres, r_occupees, n_rsrc := 1..100, {}, 100
```

```
OPERATIONS
  reserver =     // reecrite avec les nouvelles variables
  ANY ss WHERE
     ss : r_libres  // de façon non déterministe
  THEN
      r_libres := r_libres - {ss}
  || r_occupees := r_occupees \/  {ss}
  || n_rsrc := n_rsrc - 1
  END
;
```



```
liberer =  // reecrite avec les nouv var
ANY ss WHERE                          bb <-- disponibilite =
   ss : r_occupees                       IF 0 < n_rsrc
THEN                                     THEN
    r_libres := r_libres \/ {ss}            bb := TRUE
|| r_occupees := r_occupees - {ss}       ELSE
|| n_rsrc := n_rsrc + 1                    bb := FALSE
END                                      END
;

END // du Refinement
```

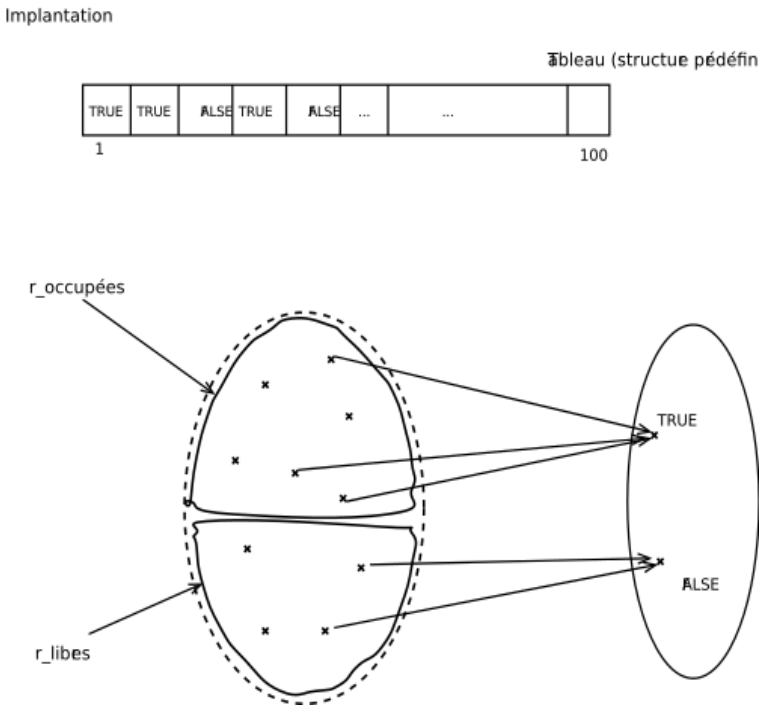
☞ On prouve le raffinement et on continue de raffiner ! allons-y.

Réservation de ressources (Implantation)

Un **dernier rafinement (= implantation)** avant la génération de code.

- un ensemble abstrait est implanté par une plage de valeurs entières (ensemble de valeurs distinctes).
  - Il faut savoir dans la plage quelle valeur est utilisée comme élément.
  - prendre/remettre un élément de la plage comme élément de l'ensemble
- un tableau (indices - valeurs) est une fonction.

Réservation de ressources (Implantation)



Structure de l'implantation

```
IMPLEMENTATION
  Reservation_I1
REFINES
  Reservation_R1
IMPORTS
 ... // On importe des machines prédéfinies
VARIABLES
 ... // nouvelles variables concretes
INVARIANT
 ...
INITIALISATION
 ...
OPERATIONS
 ... // maintenant écrites avec des susbtitutions de raffinement
     et d'implantation (dans la suite)
END
```


Raffinement : technique de développement

L'idée du raffinement :

- On part d'une machine abstraite définissant un modèle mathématique abstrait,
- on raffine ce modèle pour obtenir un modèle concret :
  - le modèle abstrait n'est pas exécutable.
    Pourquoi ? (on y utilise des objets/structures mathématiques)
  - pour obtenir un modèle équivalent sur le plan des fonctionnalités mais plus concret.
    (on y utilise des objets/structures informatiques)

Raffinement : technique de développement

- La finalité du raffinement est l'obtention du code exécutable.
- Il faut garantir que le raffinement est correct :
  (prouver la correction des étapes du raffinement).

$\Rightarrow$ obligations de preuve de raffinement

Démarche de raffinement

Quoi raffiner dans le modèle ?

- Les **variables et l'invariant**
  Partie statique - espace d'états
  Changement de variables
- Les **opérations**
  Partie dynamique - substitutions généralisées
  raffinement des substitutions généralisées.
  Introduire des **substitutions de raffinement**
  (jusqu'à atteindre des substitutions de programmation).
  IF-THEN-ELSE..., WHILE..., VAR..., $\cdots$

Démarche de raffinement : Comment raffiner ?

Introduction de structures de données et
représentation de structures par d'autres structures plus concrètes.

- Utiliser la clause REFINES pour faire le lien entre la machine abstraite et son raffinement

```
REFINEMENT
        MM_R1
REFINES
        MM
...
END
```


Démarche de raffinement : Comment raffiner ?

- Raffinement de l'espace d'états :
  - choix de structures moins abstraites,
  - invariant de liaision : liaisons entre variables concrètes et abstraites
    - introduire de nouvelles variables concrètes, et
    - les relier aux abstraites
  - raffiner les ensembles (disjoints) par des plages de valeurs (disjointes)
  - raffiner les relations et fonctions par des tableaux(= fonction de indices vers valeurs)

Exemples de raffinement

Exemples (certains déjà vus) :

- Calcul du PGCD de deux entiers
- Réservation de ressources
- Division euclidienne
- Dictionnaire de mots

Développement du PGCD : machine abstraite

```
MACHINE
    pgcd1 /* pour le PGCD de deux entiers, JCA, U. Nantes */
            /* pgcd(x,y) is d | x mod d = 0 ∧ y mod d = 0
            ∧ ∀ other divisors dx d > dx
            ∧ ∀ other divisors dy d > dy */
OPERATIONS
        rr <-- pgcd(xx,yy) = /* SORTIE : rr ; ENTREE xx, yy */
            ...
END
```

Développement du PGCD : machine abstraite

```
OPERATIONS
rr <-- pgcd(xx,yy) = /* spécification du pgcd */
PRE
    xx : INT & xx >= 1 & xx < MAXINT
& yy : INT & yy >= 1 & yy < MAXINT
THEN
    ANY dd WHERE
    dd : INT
    & (xx - (xx/dd)*dd) = 0 /* d is a divisor of x */
    & (yy - (yy/dd)*dd) = 0 /* d is a divisor of y */
        /* and the other common divisors od are < d */
    & !od.((od : INT & od < MAXINT
        & ((xx- (xx/od)*od)= 0) & ((yy-(yy/od)*od)=0) ) => od < dd)
    THEN rr := dd
    END
END
```


Développement du PGCD : raffinement

```
REFINEMENT /* raffinement de ...*/
    pgcd1_R1
REFINES pgcd1 /* la machine précédente */
OPERATIONS
rr <---- pgcd (xx, yy) = /* l'interface ne change pas */
    BEGIN
        ... Corps de l'opération raffinée
    END
END
```

Développement du PGCD : raffinement

```
rr <-- pgcd (xx, yy) = /* opération raffinée */
     BEGIN
          VAR cd, rx, ry, cr IN
               cd := 1
               ; WHILE ( cd < xx & cd < yy) DO
                    ; rx := xx - (xx/cd)*cd ; ry := yy - (yy/cd)*cd
                    IF (rx = 0 & ry = 0)
                    THEN /* cd divise x et y, possible GCD */
                         cr := cd /* possible rr */
                    END
                    ; cd := cd + 1 ; /* on tente plus grand */
               INVARIANT
                    xx : INT & yy : INT & rx : INT & rx < MAXINT
                    & ry : INT & ry < MAXINT & cd < MAXINT
                    & xx = cr*(xx/cr) + rx & yy = cr*(y/cr) + ry
               VARIANT
                    xx - cd
               END
               ; rr := cd
          END      END
```


Substitutions de raffinement

- **Substitutions séquentielles**
  Soient S et T deux substitutions,
  la substitution séquentielle est notée : $S \ ; \ T$
  Sa définition sémantique s'exprime par :

$$\begin{aligned} [S;T]R &\equiv [S][T]R \\ &\equiv [S]([T]R) \end{aligned}$$

S établit $[T]R$

Substitutions de raffinement

- **Substitution de boucles**
  La substitution de boucle à la forme suivante :

  while P do
  S
  invariant
  I
  variant
  V
  end

Sémantique de la substitution de boucle

sur le plan sémantique, c'est

$$
\begin{array}{l}
I \ \wedge \\
\qquad \text{/* le variant est un entier */} \\
\forall\, x.(I \ \Rightarrow \ V \ \in \ NATURAL) \ \wedge \\
\qquad \text{/* le variant décroit à chaque pas */} \\
\forall\, (x,n).(I \ \wedge \ P \ \Rightarrow \ [n \ := \ V][S](V \ < \ n)) \ \wedge \\
\qquad \text{/* continuation de la boucle */} \\
\forall\, x.(I \ \wedge \ P \ \Rightarrow \ [S]I) \ | \\
\qquad @x'.([x \ := \ x'](I \ \wedge \ \neg \ P) \ \Rightarrow \ x \ := \ x'))
\end{array}
$$

Substitution VAR ... IN

- **Bloc avec variables locales**
  La notation est :

  var x in // introduction de variables locales
  S
  end

Raffinement des opérations : synthèse

Les substitution simultanées ||	raffinées par des substitutions séquentielles ;
PRE...THEN Subst END	BEGIN Raffinement(Subst) END
ANY ... WHERE ... END	WHILE ... DO ... END
CHOICE ...OR ... END	...IF ... THEN ... ELSE

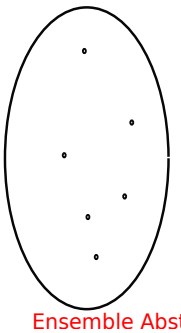
Obligations de preuve de raffinement

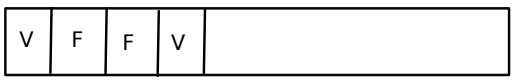
Intuition : la machine concrète ne fait pas ce qu'elle ne devrait pas faire (elle ne contredit pas l'abstraite)
Obligation de preuve:

$$INV_{abs} \wedge INV_{conc} \Rightarrow [Subst_{conc}]\neg\ ([Subst_{abs}]\neg\ (INV_{abs}))$$

Une opération concrète ne fait ce qu'elle ne devrait pas faire (ie : contredire que l'abstraite préserve l'invariant)

Raffinement de données : ensembles abstraits

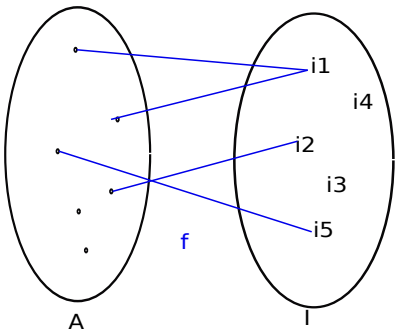

Ensemble Abstrait

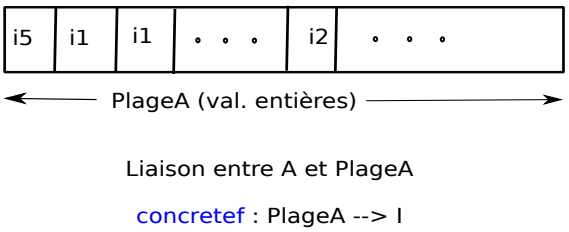


Plage de valeurs (entières)

V : dans l'ensemble
F : pas dans l'ensemble

Raffinement de données : fonctions





Raffinement des données : synthèse

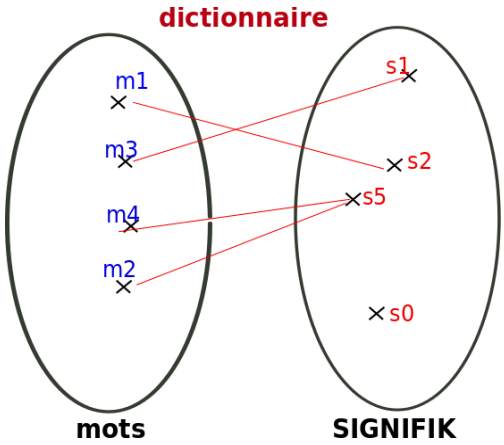
les types de base	inchangé
les ensembles prédéfinis	inchangé
les ensembles énumérés	inchangés car ils sont concrets
un ensemble abstrait A	raffiné par une plage de valeurs choisies $PlageA$ avec un invariant de liaison
les sous-ensembles	raffinés par des plages de valeurs choisies
les relations $A \leftrightarrow B$	raffinées par des fonctions totales $PlageA \times PlageB \rightarrow \{0,1\}$ et un invariant de liaison
les fonctions partielles $A \nrightarrow B$	raffinées par des fonctions totales $PlageA \rightarrow PlageB$ et un invariant de liaison

Implantation avec AB V4

- Voir le mémento complet (en ligne, site du cours)
- Dans la version 4 de AtelierB, il n'y a plus de bibliothèques de composants réutilisables (comme dans la V3.7).
- Le générateur de code C utilisé est ComenC.

Raffinement/Implantation avec AB V4

Application : gestion d'un dictionnaire de mots
Fonctionnalités : ajouter mot+signification ; recherche mot, etc
Après analyse (c'est quoi un dictionnaire ?) :



Implantation avec AB V4

Un exemple de machine abstraite (dicoMot.mch) :

```
MACHINE
    dicoMot
SETS
     MOT /* un ensemble abstrait de mots */
    ; SIGNIFIK = {s0,s1,s2,s5}/* un ensemble abstrait, des significations
CONSTANTS
     maxMots  /* borne */
PROPERTIES
    maxMots : 1..MAXINT
VARIABLES
    mots /* sous ensemble de mots */
,    dico /* représente le dictionnaire */
INVARIANT
    mots <: MOT /* sous ens de mots utilisés, borné */
    & card(mots) <= maxMots
    &  dico : mots --> SIGNIFIK
```


Implantation avec AB V4 : suite machine dico

(suite ...)

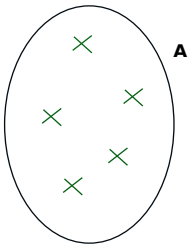
```
INITIALISATION
    mots := {}
||    dico := {}
OPERATIONS
    ajoutMot(mm, signif) =
    PRE mm : MOT & mm /: mots
        & signif : SIGNIFIK
        & (mm,signif) /: dico
        & card(mots) < maxMots
    THEN
        mots := mots \/ {mm}
        || dico(mm) := signif
    END
END
```

Implantation avec AB V4

Rappel :

- On implante un ensemble abstrait d'éléments par un ensemble d'éléments choisis dans une plage de valeurs.
- Une plage de valeurs est par exemple un intervalle de valeurs entières (on en prend certaines et pas d'autres).
- Connaître celles des valeurs qu'on a prises (OK), celle qui ne sont pas prises (KO).

Implantation avec AB V4



PLAGE_A == 10..200

Ens_A : PLAGE_A --> {OK, KO}

OK	OK	KO	OK	KO	. . . .	OK	KO
10	11	12	13	14	. . . .		200

ff : PLAGE_A --> PLAGE_B

liaison

eB	eB	eB	eB	eB	. . . .	eB	eB
10	11	12	13	14	. . . .		200
OK	OK	KO	OK	KO	. . . .	OK	KO

liaison ff

Implantation avec AB V4

Implantation (dicoMot_i.imp) :

```
IMPLEMENTATION    dicoMot_i
REFINES    dicoMot
 DEFINITIONS
    PLAGE_MOT == 0..20  /* une plage pour implanter l'ens des mots */
SETS
    OKKO = {ok, ko} /* pour indiquer quelle valeur est utilisee comme mot
 VALUES
     MOT = PLAGE_MOT  /* implantation de l'abstrait MOT */
    ; maxMots = 22   /* une valeur quelconque ici */
CONCRETE_VARIABLES
    c_mots,    /* nouvelles variables */
    c_dico
INVARIANT
    c_mots : PLAGE_MOT --> OKKO   /* mots utilisés ou non */
    & mots = c_mots~[{ok}]  /* liaison abstrait concret */
    &  c_dico : PLAGE_MOT --> SIGNIFIK
    &  dico = (mots <| c_dico) /* liaison abstrait concret */
```


Implantation avec AB V4

(suite ...)

```
INITIALISATION
   c_mots := (PLAGE_MOT)*{ko}; /* aucun n'est encore utilisé */
   c_dico := (PLAGE_MOT)*{s0}  /* qui est vide */
OPERATIONS
   ajoutMot ( mm , signif ) =
   BEGIN
       c_mots(mm) := ok ; /* mots := mots \/ { mm } */
       c_dico (mm) := signif
   END
END
```

Raffinements successifs

- Plusieurs niveaux de raffinements sont souvent nécessaires.
- Une chaîne verticale allant de la machine abstraite au code (refines ...).
- Introduction de données détaillées.
- Choix de structure de contrôle.
- Réutilisation de machines existantes:
  chaîne horizontale (includes..., sees..., imports ...)

Structure type de construction en B

MACHINE Appli

/* un scenario
pour tester
les machines
élémentaires */

où on appelle une
opération

MACHINE Interface

où on teste les
préconditions
avant d'appeler
les opérations
voulues

MACHINE MD
Où on définit
les opérations
élémentaires
issues du cahier
de charges

Exemple d'une application : gestion de positions d'un pointeur (x,y)

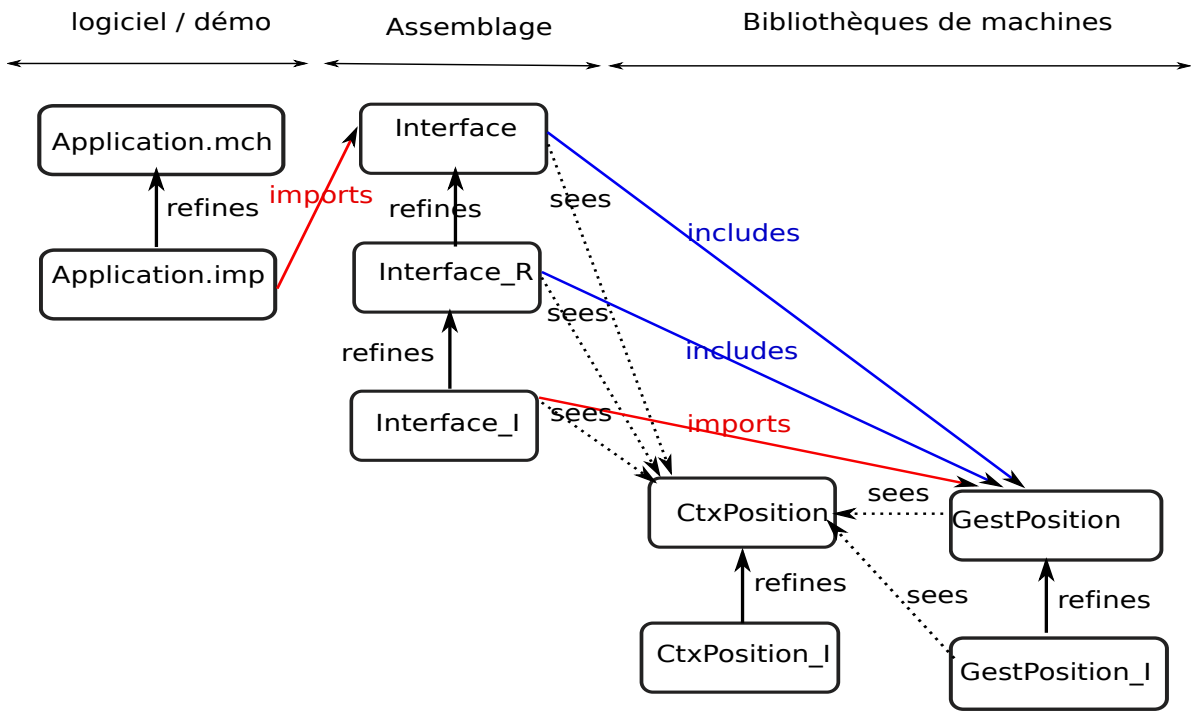


Figure: Architecture type de développement en B

Architecture de grands systèmes

Composition de machines → machines de grande taille.

- Modules - Composition - Architecture en couches
- Modularité

Composition de machines

- Hiérarchisation
  Les clauses INCLUDES, EXTENDS, PROMOTES
- Partage
  Les clauses SEES, USES

Hiérarchisation

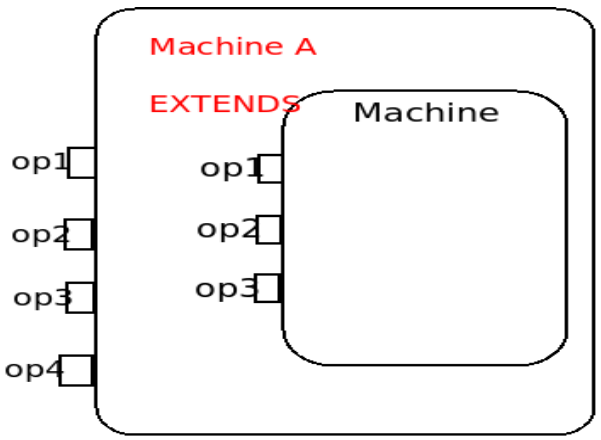
INCLUDES permet d'inclure une machine dans une autre
+ promotion de certaines opérations PROMOTES

```
MACHINE
        MA
 INCLUDES
        MB                  /* acces par Opmb aux varB */
 PROMOTES
        Opmb1, Opmb3        /* deviennent des Op. de MA */
...
END
```


Hiérarchisation

EXTENDS, inclusion mais pas besoin de promotes

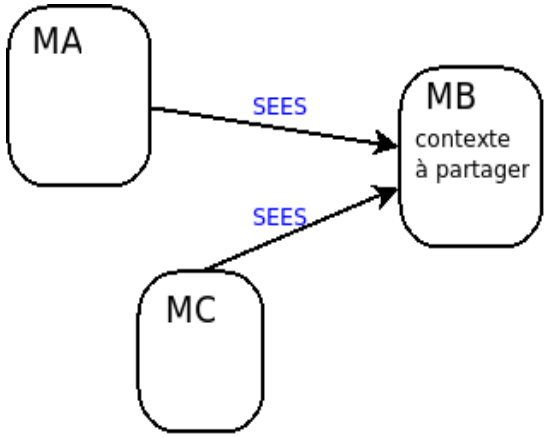
```
MACHINE
        MA
 EXTENDS
        MB
...
END
```



Partage

SEES fait le partage de type lecture seule

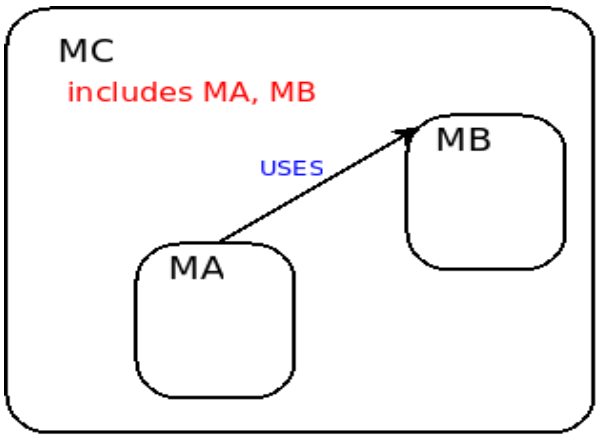
```
MACHINE
        MA
 SEES
        MB
...
END
```



Partage

USES fait le partage de type lecture/écriture

```
MACHINE
        MA
 USES
        MB
...
END
```



MA et MB devront être incluses dans une autre machines.

Références

- J-R. Abrial, The B-Book, Cambridge University Press, 1996
- J. Wordsworth, Software Engineering with B, Addison-Wesley, 1996
- J-R. Abrial, Modeling in Event-B: System and Software Engineering, Cambridge University Press
- H. Habrias, Spécification formelle avec B, Hermès - Lavoisier, 2001