Construction formelle de logiciels

La méthode B

J. Christian Attiogbé

Novembre 2008, maj 11/2009, 10/2010

Raffinement : technique de construction

- on va progressivement d'un modèle abstrait vers un modèle concret
- changement de niveau d'abstraction (ajout de détails)
- choix (conception)
- preuve de correction

Méthode B : Approche globale

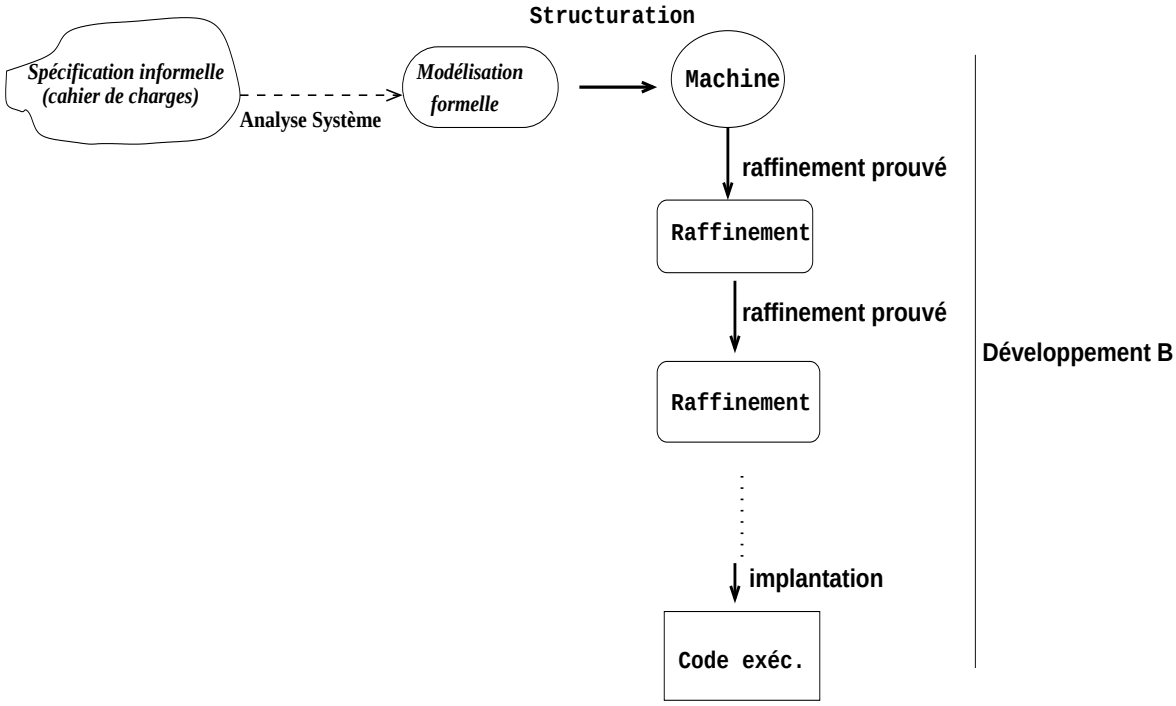


Figure: Analyse et développement B

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, exécutable = un code
  - pour obtenir ce code, on explicite le comment
    (on fait des choix d'objets informatiques)

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

Nous avions vu (en TD) la construction de la calculette par raffinement.

Plan de la suite

Exemple de raffinement

- 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

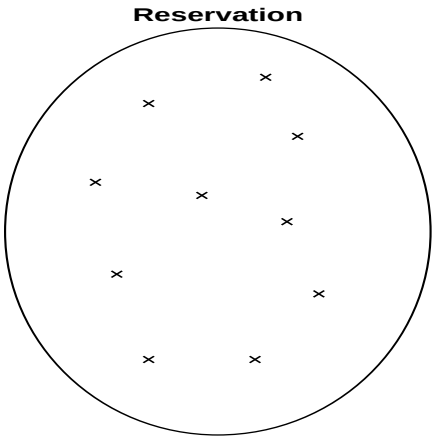
Exemple : Réservation de ressources

$n\_rsrc \in 0..100$

n_rsrc = cardinal de l'ensemble

réserver $\rightarrow$ - 1 élément

libérer $\rightarrow$ + 1 élément




```
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 de la machine abstraite a deux types d'obligations de preuve :
montrer que l'INITIALISATION établit l'invariant

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

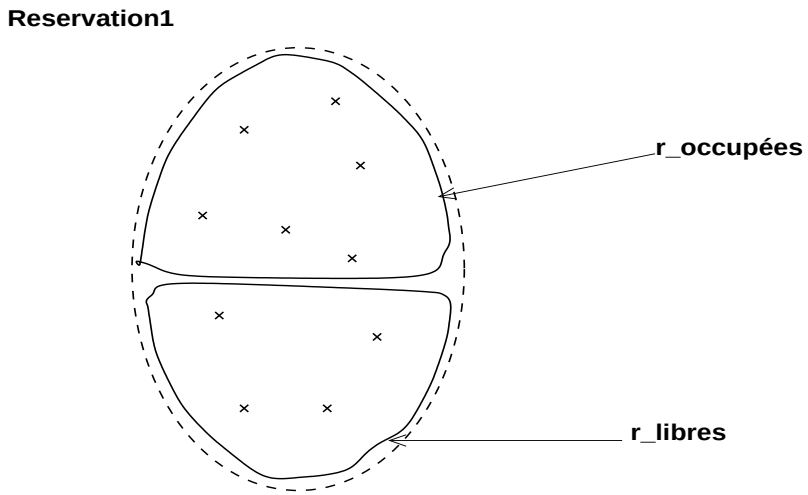
Il faut montrer que $100 \in 0..100$

Preuve de cohérence

On doit 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 :
  $n\_rsrc \in 0..100 \wedge (n\_rsrc > 0 \vee \neg (n\_rsrc > 0))$
  $\Rightarrow$
  $n\_rsrc \in 0..100$

Réservation de ressources (Raffinement)



réserver → trouver 1 élément libre

libérer → trouver 1 élément occupé


```
REFINEMENT
  Reservation_R1
REFINES
  Reservation
VARIABLES
   r_libres, r_occupees // n_rscrc est incluse
// nouvelles variables moins abstraites
INVARIANT
   r_libres : POW(INTEGER)
&  r_occupees : POW(INTEGER)
&  r_libres /\ r_occupees = {}
 & n_rsrc = card(r_libres) // invariant de liaison
INITIALISATION
 r_libres, r_occupees, n_rsrc := 1..100, {}, 100
```

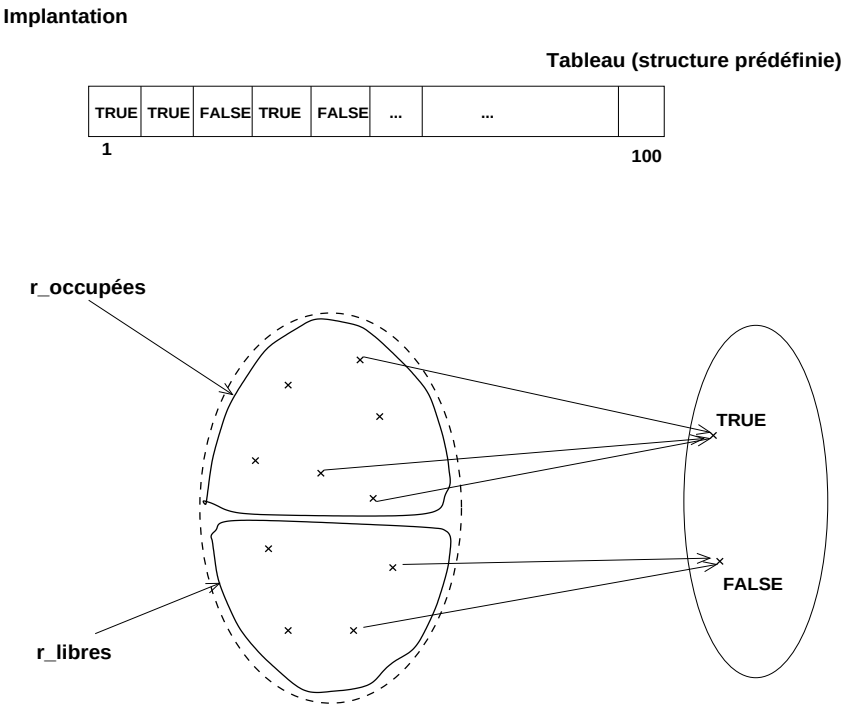
```
 OPERATIONS
reserver =     // reecrite avec les nouvelles variables
   ANY ss WHERE
      ss : r_libres  // de façon non déterministe
   THEN
       r_libres := r_libres - {ss}
   || r_occupees := r_occupees \/  {ss}
   || n_rsrc := n_rsrc - 1
   END
;
```



```
liberer =  // reecrite avec les nouv var
 ANY ss WHERE
    ss : r_occupees
 THEN
     r_libres := r_libres \/ {ss}
 || r_occupees := r_occupees - {ss}
 || n_rsrc := n_rsrc + 1
 END
;
```

```
bb <-- disponibilite =
   IF 0 < n_rsrc
   THEN
      bb := TRUE
   ELSE
      bb := FALSE
   END
END
```

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
 ... // elles sont maintenant écrites avec des susbtitutions de
raffinement
```


et d'implantation (à voir dans la suite du cours)

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 mathématiques)
  - pour obtenir un modèle équivalent sur le plan des fonctionnalités mais plus concret.
    (on y utilise des objets informatiques)

Raffinement : technique de développement

- La finalité du raffinement est l'obtention du code exécutable.
- Il faut garantir que le raffinement est correct :
  (la preuve 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.
  Introduire des **substitutions de raffinement**
  (jusqu'à atteindre des substitutions de programmation).

Démarche de raffinement : Comment raffiner ?

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

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

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

- Raffinement de l'espace d'états :
  - introduire de nouvelles variables (concrètes),
  - choix de structures moins abstraites,
  - liaisons entre les variables concrètes et abstraites par un invariant de liaision

Démarche de raffinement : Comment raffiner ?

- Raffinement des opérations :
  - Les interfaces ne peuvent pas être modifiées.
  - Réécrire les opérations abstraites avec les nouvelles variables et les substitutions appropriés (introduction des séquences, boucles, var locales).
  - Introduire des substitutions de raffinements.
  - Lever le non-déterminisme
  - Affaiblir dans la machine raffinée (concrète), les préconditions qu'avaient les opérations dans la machine abstraite, jusqu'à les faire disparaître.

$\Rightarrow$ compléter le langage de substitutions.

Exemples de raffinement

Exemples déjà vus :

- Réservation de ressources
- Division euclidienne

Substitutions de raffinement

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

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

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

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)

Raffinements successifs

- Plusieurs niveaux de raffinement 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 du développement d'une application autour d'un annuaire

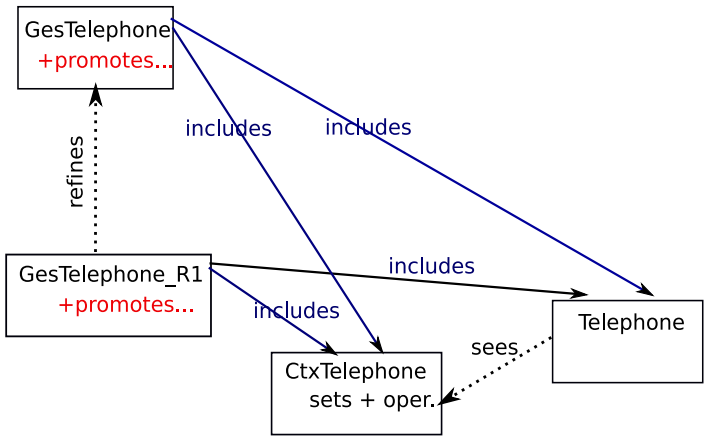


Figure: Architecture type de développement en B

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

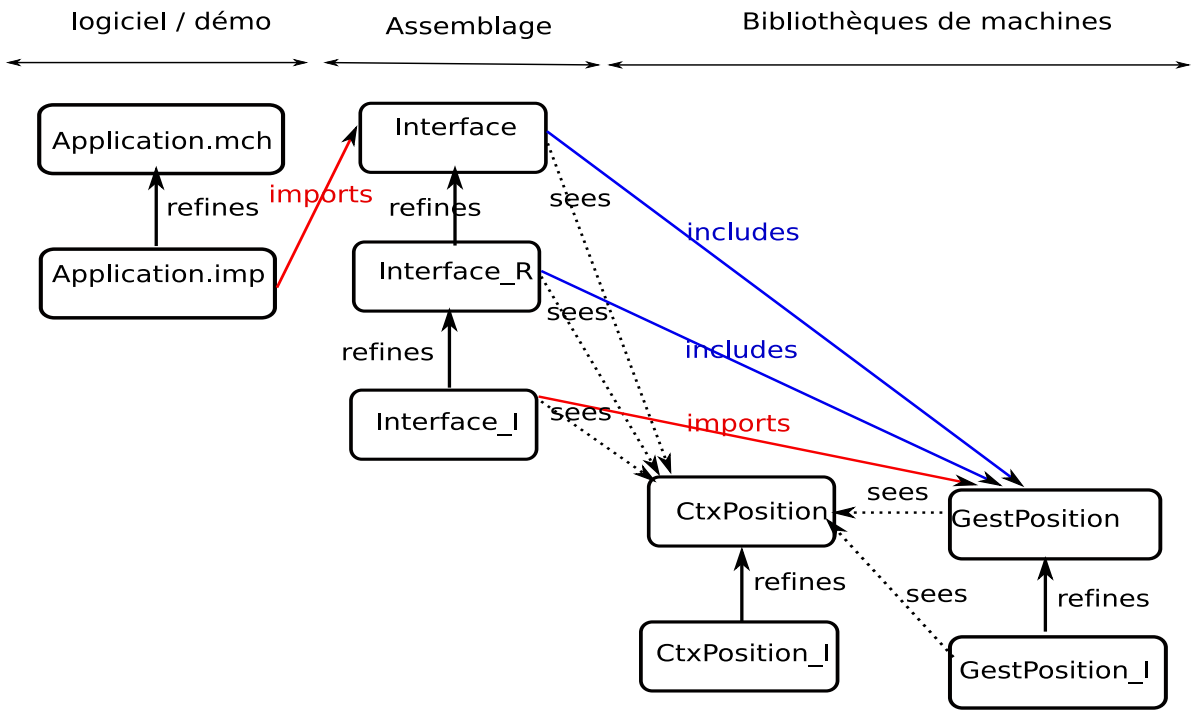


Figure: Architecture type de développement en B

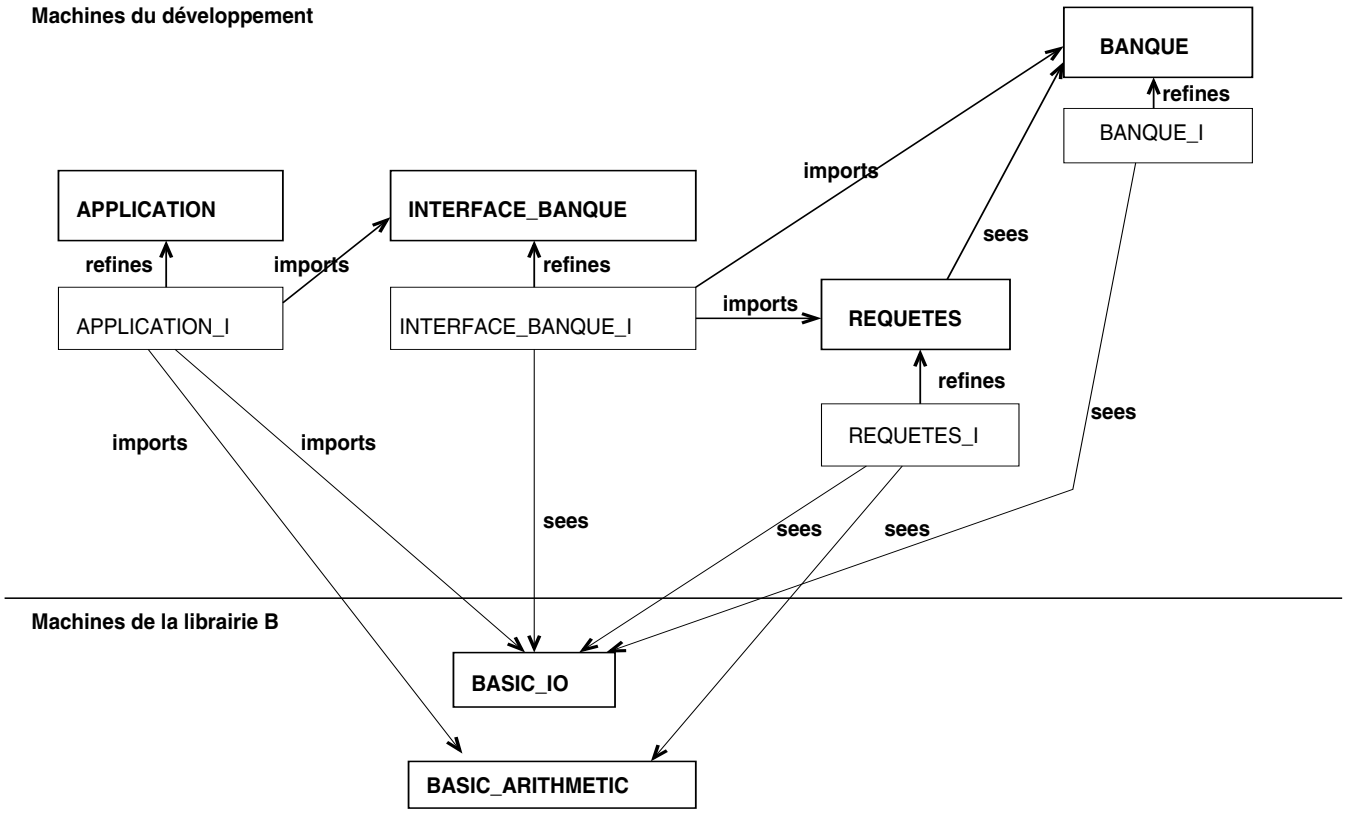
Architecture de grands systèmes

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

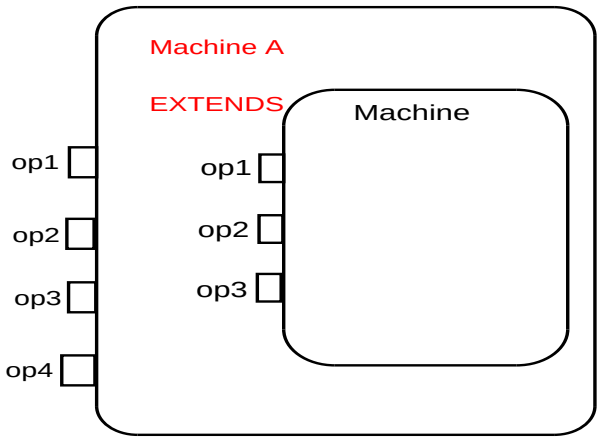
INCLUDES permet d'inclure une machine dans une autre
+ promotion de certaines opérations PROMOTES

```
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

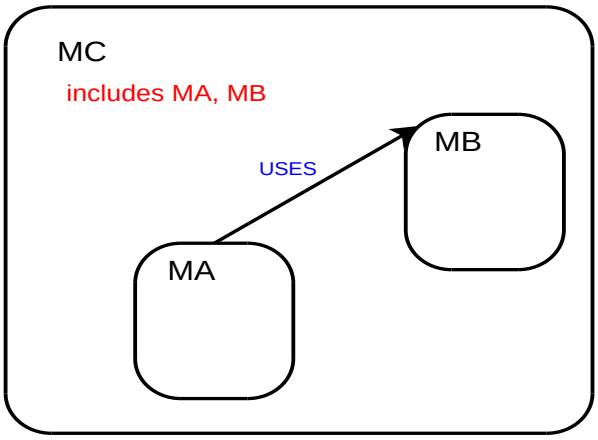
SEES fait le partage de type lecture seule

```
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.

Implantation avec AB V4

Dans la version 4, il n'y a plus de bibliothèques de composants réutilisables (comme dans V3.7).
Le générateur de code utilisé est ComenC.
Voir le mémento prévu à cet effet.

On peut voir dans la suite le principe d'utilisation de bibliothèques de composants prédéfinis (comme mis en œuvre dans AB V3.7)

Eléments pour l'implantation des machines(AtelierB V3.7)

L'Atelier B propose un ensemble de composants prédéfinis.
Ils sont importés via la clause IMPORTS.
Référence :
Atelier B, *Composants réutilisables, manuel de référence*, Steria

Implantation des ensembles différés

Les ensembles différés (clause SETS) peuvent être implantés par des ensembles concrets énumérés ou définis par un intervalle.
La clause VALUES est utilisée.

```
MACHINE                          IMPLEMENTATION  M1_I
       M1                        REFINES   M1
SETS                              VALUES
       ETAT                         ETAT = {actif, pret, bloque}
;      PROCESSUS                                    /* énuméré    */
                                 ;  PROCESSUS = 0..100
VARIABLES                                           /* intervalle */
       ...                       ...
END                              END
```

Implantation d'ensembles et fonctions

Implantation des ensembles par des ensembles prédéfinis
Le composant prédéfini L_SET réalise un ensemble par une variable nommée **set_vrb**(qui est une séquence injective).
Le codomaine de cette séquence représente les images de l'ensemble à implanter.
Plusieurs opérations sont fournies par ce composant:

- le cardinal de l'ensemble (CARD_SET),
- recherche d'un élément de l'ensemble (FIND_SET),
- test si l'ensemble est plein(IS_FULL_SET) ,
- retrait d'un élément de l'ensemble, etc.

Implantation d'ensembles et fonctions

```
MACHINE Proc
SETS
  PROCESSUS = {p1, p2, p3, p4, p5}
VARIABLES
  prets
INVARIANT
  prets <: PROCESSUS
  /* ensemble de processus prets */

INITIALISATION
  prets := {p2, p4}
...
END
```

```
IMPLEMENTATION Proc_I
REFINES
     Proc
 IMPORTS
     L_SET(5, PROCESSUS)

INVARIANT
     prets = ran(set_vrb)

INITIALISATION
      INS_SET(p2);INS_SET(p4);
...
END
```

Implantation des ensembles et fonctions par des tableaux

Les composants prédéfinis L_ARRAY1, L_ARRAY3, L_ARRAY5 sont des réalisations de tableau à une dimension.
Ils peuvent être utilisés pour implanter des ensembles (domaine ou codomaine de fonctions).
L_ARRAY3 réalise un tableau à une dimension nommé **arr_vrb** et fournit plusieurs opérations intéressantes, telles que :
initialisation d'une portion du tableau avec une même valeur,
recherche d'une valeur dans une portion du tableau.

Exemple

```
MACHINE ProcE
SETS
      PROCESSUS = 0..100
;     ETAT = {pret, actif, bloque}
VARIABLES
      etat
INVARIANT
      etat : PROCESSUS --> ETAT     /* etat des processus */
INITIALISATION
      etat := PROCESSUS x {pret}
OPERATIONS
id, bl <-- chercherProc(pret) =     /* recherche d'un processus pret */
       PRE     pret : ran(etat)
       THEN
               id :: etat~[{pret}]     ||  bl :: BOOL
       END
END
```

Une implantation utilisant L_ARRAY3 est présentée ci-dessous.

```
 IMPLEMENTATION ProcE_I
REFINES        ProcE
 IMPORTS
        etats.L_ARRAY3(ETAT, 100)  /* importat. d'1 exemplaire (nommé eta
                                      de L_ARRAY3
,       BASIC_ARITHMETIC          /* pour disposer des opérations arithmét.
,       BASIC_BOOL                /* et booléennes
INVARIANT
        etats.arr_vrb = etat      /*  liaison de la variable abstraite eta
                                      la variable concrete arr_vrb
INITIALISATION
       etats. SET_ARRAY(0,100, pret) /*  tous les elts à pret  */
OPERATIONS
       id, bl <-- chercherProc(pret) =
           id, bl <-- etats. SEARCH_MAX_EQL_ARRAY(0,100,pret)
END
```


Implantation des fonctions

Le composant L_PFNC permet d'implanter les fonctions partielles.
Plusieurs autres composants sont disponibles pour implanter les fonctions.

Exemple : BASIC_ARRAY_RGE .

BASIC_ARRAY_RGE est une machine de base qui réalise un tableau à deux dimensions.

Ce composant offre une réalisation à travers une variable **arr_vrb** de la forme :

arr_vrb : LIGNES -> (COLONNES -> VALEURS).

Implantation des fonctions (suite)

`arr_vrb : LIGNES -> (COLONNES -> VALEURS).`
`LIGNES` représente l'intervalle des valeurs d'indice des lignes du tableau,
`COLONNES` représente l'intervalle des valeurs d'indice des colonnes du tableau, et
`VALEUR` représente l'intervalle des valeurs des éléments du tableau.

Implantation des fonctions (suite)

```
MACHINE
   ProcTrace
SETS
   PROCESSUS = p1, p2, p3
;  DATE = 1..5
;  ETAT = pret, actif, bloque
VARIABLES
   trace5
INVARIANT
   trace5 : PROCESSUS --> (DATE +-> ETAT)
            /* On modelise la trace des 5 derniers etats des processus */
INITIALISATION
  trace5 :=
(p1,{1 |-> pret, 2 |-> pret,3 |-> pret,4 |-> pret, 5 |-> pret}),
 (p2,{1 |-> pret, 2 |-> pret,3 |-> pret,4 |-> pret, 5 |-> pret}),
 (p3,{1 |-> pret, 2 |-> pret,3 |-> pret,4 |-> pret, 5 |-> pret})

END
```

Implantation des fonctions (suite)

Voici une implantation utilisant BASIC_ARRAY_RGE pour réaliser la machine ProcTrace.

```
 IMPLEMENTATION  ProcTrace_I
REFINES          ProcTrace
 IMPORTS
                BASIC_ARRAY_RGE(DATE, ETAT, PROCESSUS)
INVARIANT
       arr_vrb = trace5  /* la liaison entre la variable abstraite (trace
                            et la concrete (arr_vrb)
INITIALISATION
       STR_ARR_RGE(p1, 1, pret);
      STR_ARR_RGE(p1, 2, pret);
      STR_ARR_RGE(p1, 3, pret);
      ...
END
```


Remarque : Les fonctions dom et ran restent applicables aux variables concretes.
Par exemple dom(ran(arr_vrb)) est l'ensemble DATE.
De cette façon, l'invariant de liaison peut permettre de lier les variables de fonctions n'ayant pas strictement la même signature que arr_vrb.
Exemple

```
MACHINE ProcTracep
...
INVARIANT       /* de la machine abstraite */
       trace5prime : PROCESSUS x DATE +-> ETAT
...
```

on peut réaliser l'implantation de la façon suivante :

```
IMPLEMENTATION ProcTracep_I
...
 IMPORTS
              BASIC_ARRAY_RGE(DATE, ETAT, PROCESSUS)

INVARIANT    /* dans l'implantation */
     ran(ran(arr_vrb)) = ran(trace5prime)          /* ETAT      */
     dom(ran(arr_vrb)) = ran(dom(trace5prime))     /* DATE      */
     dom(arr_vrb) = dom(dom(trace5prime))          /* PROCESSUS */
...
```