Modularité en méthode B

Notion de modularité

2

- Différentes relations entre machines, raffinements et implantations permettent le développement modulaire d'un projet B.
- INCLUDES/IMPORTS : permet une imbrication des machines / implantations (hiérarchie verticale)
- PROMOTES/EXTENDS : permet l'utilisation des opérations des machines incluses / importées
- USES/SEES : permet une visibilité « horizontale » entre machines dans une hiérarchie

Clause INCLUDES

- La clause INCLUDES permet d'inclure dans une machine ou un raffinement une instance de la machine déclarée comme incluse.
- Les paramètres formels de la machine incluse s'ils existent doivent être fournis par la machine qui inclut (génère une PO visant à montrer que ces paramètres respectent les CONSTRAINTS de la machine incluse).
- L'utilisation de INCLUDES est une facilité pour de grands projets. On pourrait faire une machine « à plat » (sans INCLUDES) équivalente.

Clause INCLUDES : contraintes

4 La machine qui inclut peut :

- Faire référence aux SETS et CONSTANTS de la machine incluse.
- Faire référence aux VARIABLES de la machine incluse dans son INVARIANT (genre d'invariant de liaison) et les consulter dans ses OPERATIONS, mais pas les modifier (référence à gauche d'un := interdite)
- Faire appel aux opérations de la machine incluse, mais sans les mettre en parallèle, sinon on pourrait violer l'invariant de machine incluse, même en respectant les préconditions (exemple ci-après).
- La clause INCLUDES est transitive (les machines incluses dans l'incluse sont automatiquement incluses, on peut référencer leurs variables etc.)

Clause INCLUDES : contraintes


```
MACHINE M1
VARIABLES xx, yy
INVARIANT xx : NATURAL & yy : NATURAL & xx<=yy
INITIALISATION xx, yy := 0, 10
OPERATIONS
increment = PRE xx<yy THEN xx:=xx+1 END;
decrement = PRE xx<yy THEN yy:=yy-1 END
END

MACHINE M2
INCLUDES M1
OPERATIONS
bug = PRE xx<yy THEN increment || decrement END
END
```

- Violation possible d'invariant de M1 donc || interdit ici

Clause INCLUDES

Il est possible d'inclure plusieurs instances d'une même machine, qu'il faut par conséquent distinguer par le nom (renommage par préfixe)

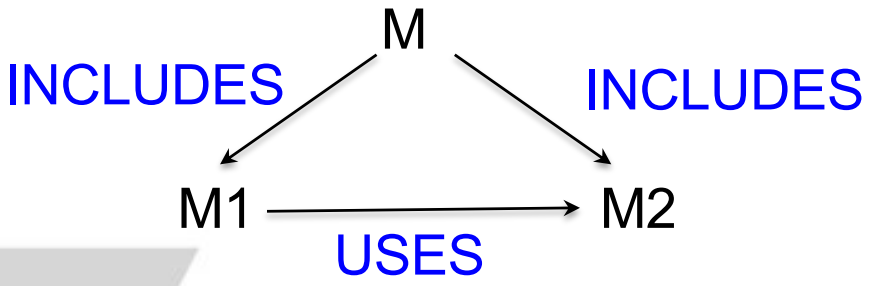
```
MACHINE Unevariable(DOM)
VARIABLES var
INVARIANT var:DOM
INITIALISATION var::DOM
OPERATIONS
ch(newval) = PRE newval:DOM
THEN var:=newval END
END
```

```
MACHINE Deuxvariables(DOM)
INCLUDES xx.Unevariable(DOM),
yy.Unevariable(DOM)
OPERATIONS
swap = BEGIN xx.ch(yy.var) ||
yy.ch(xx.var) END
END
```

Clause USES

7 Dans certains cas où plusieurs machines sont incluses dans une même machine, une certaine visibilité d'une machine incluse sur une autre peut être souhaitable : c'est l'objet de la clause USES.

- Exemple : M1 et M2 : deux instances de machines incluses dans M, M1 USES M2
- M1 : voit paramètres, constantes et variables (mais pas opérations) de M2 et peut énoncer des prédicats les concernant dans son invariant. Les preuves correspondantes sont faites au niveau de M.
- USES est limité aux machines (ne se raffine pas) et n'est pas transitive.



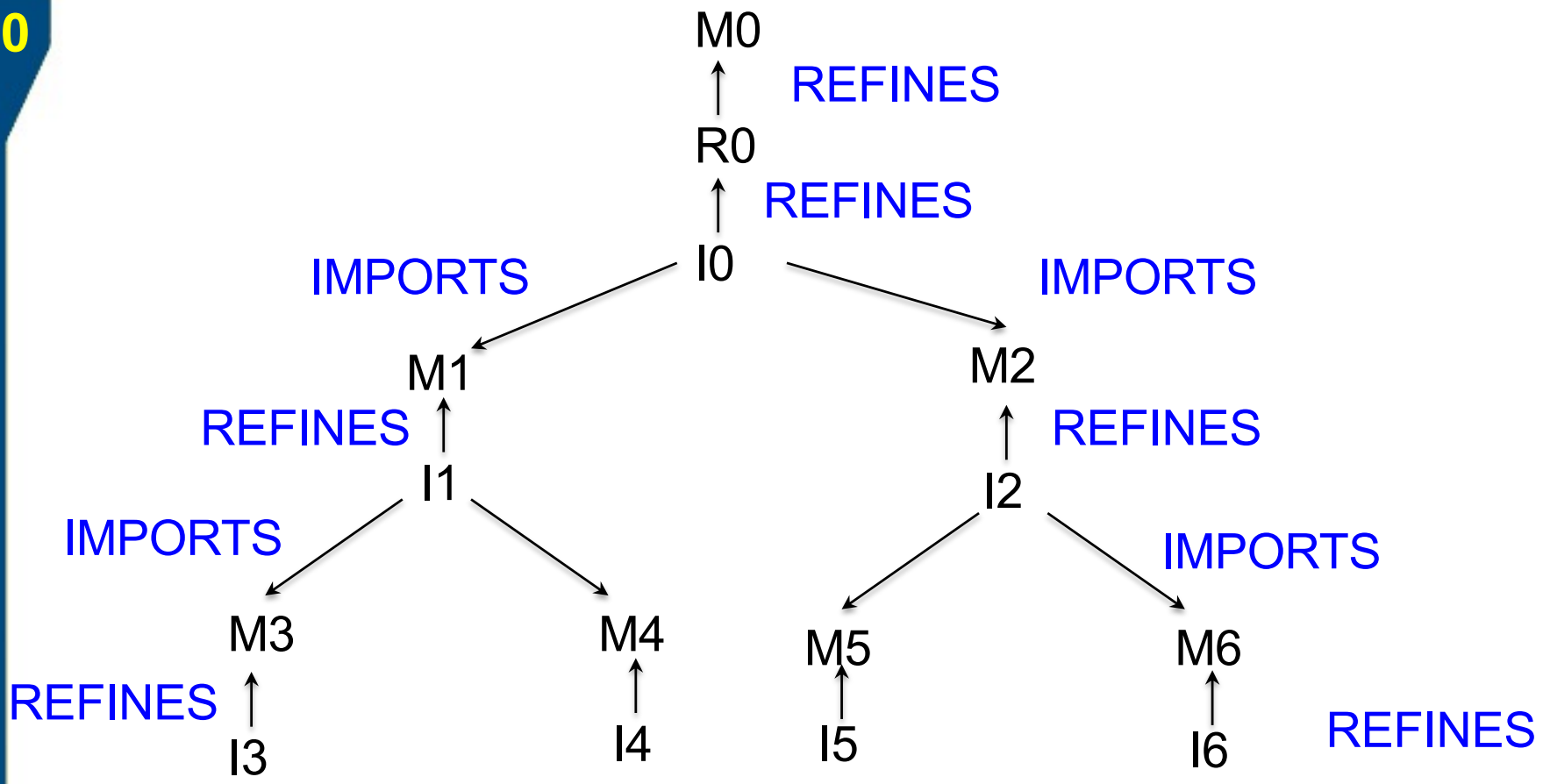
Clauses PROMOTES et EXTENDS

- Dans une machine qui en inclut une autre, la clause PROMOTES suivie d'une liste d'opérations de la machine incluse en fait des opérations à part entière de la machine qui inclut.
- La clause EXTENDS est l'équivalent de la clause INCLUDES et du PROMOTES de toutes les opérations de la machine incluse.

Clause IMPORTS

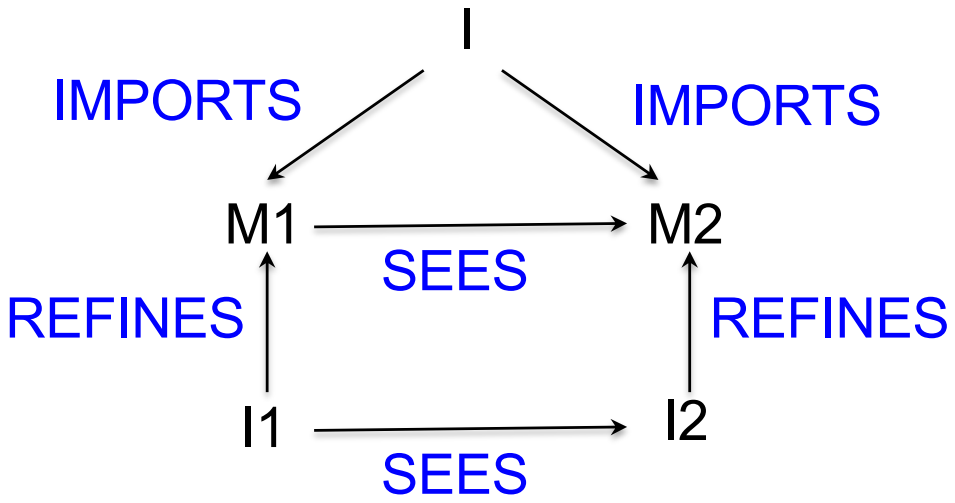
- La clause IMPORTS est l'équivalent du INCLUDES pour les implantations.
- Il est ainsi fréquent qu'une implantation importe des machines, qui sont à leur tour raffinées et implantées, leur implantation important à leur tour des machines etc.
- Une telle architecture de développement très fréquente (et conseillée) porte le nom de développement « par couches ».
- Une différence majeure entre le INCLUDES et le IMPORTS est que les variables abstraites des machines importées (qui pourront être raffinées par la suite) sont donc invisibles de l'implantation qui les importe (« vue encapsulée »).

Développement « par couches »



Clause SEES

- Le développement en couches introduit une indépendance entre modules d'une même couche qui peut être jugée trop rigide. C'est l'objet de la clause SEES.
- Contrairement à USES, SEES a pour objectif de se propager jusqu'à l'implantation (se raffine) les règles de visibilité sont donc totalement différentes.



Clause SEES

- M1 peut voir les variables de M2 mais seulement dans ses opérations (pas dans son invariant) : en effet les variables et opérations pourront être raffinées.
- M1 peut voir les opérations de M2 mais seulement celles de consultation.
- L'architecture choisie (strict respect ou non de l'architecture « en couches ») dépend du contexte applicatif.