Workshop OCM
2 juin 2003

Une démarche outillée pour spécifier formellement des patrons de conception réutilisables

Sandrine BLAZY, Frédéric GERVAIS, **Régine LALEAU**

Centre d'Étude et de Recherche
en Informatique du CNAM

http://cedric.cnam.fr

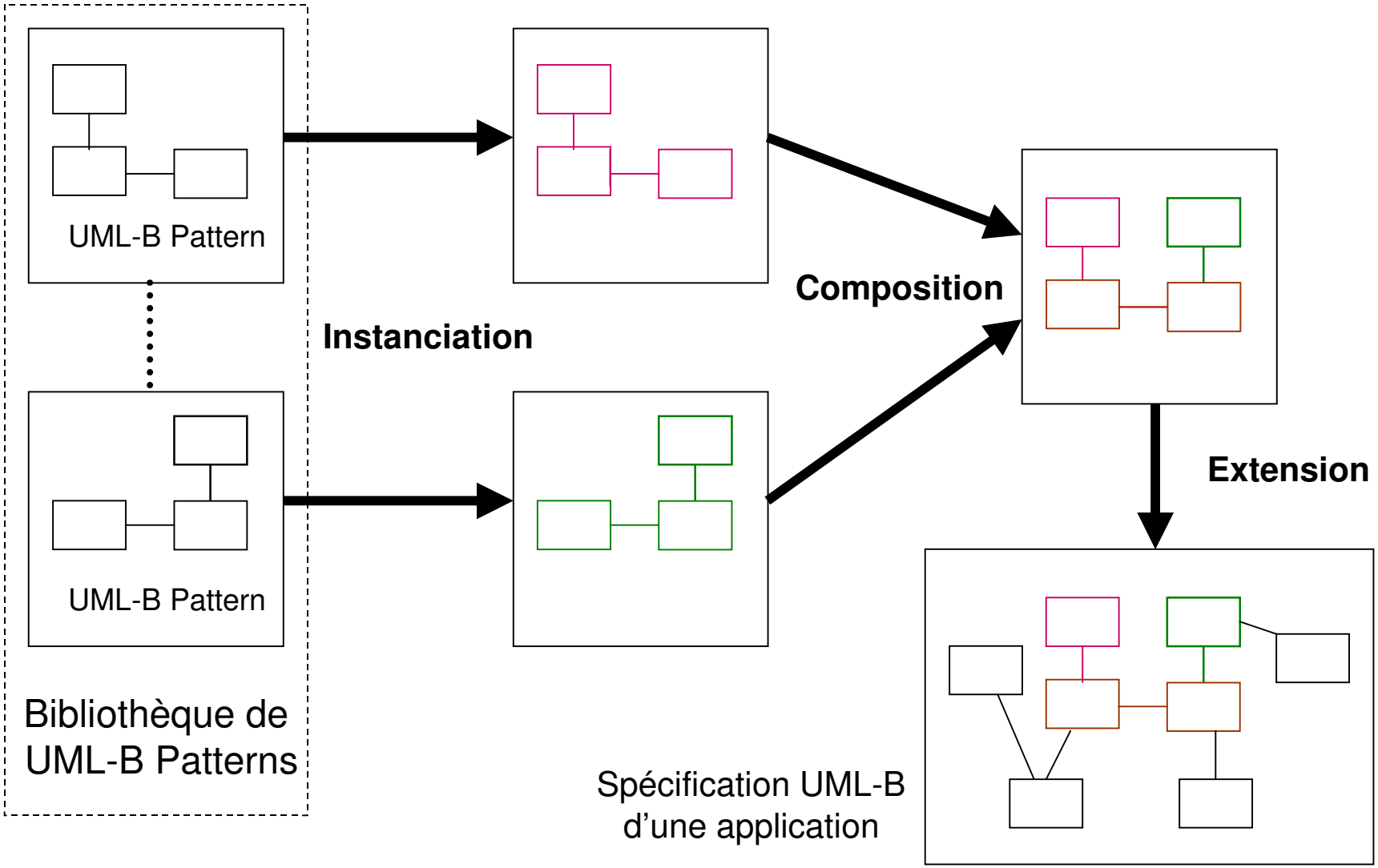
Spécification par réutilisation

- Pattern : une solution à un problème de conception récurrent (ex : patterns Composite, Resource Allocation)

  souvent exprimé en UML

  -----> pas de définition précise

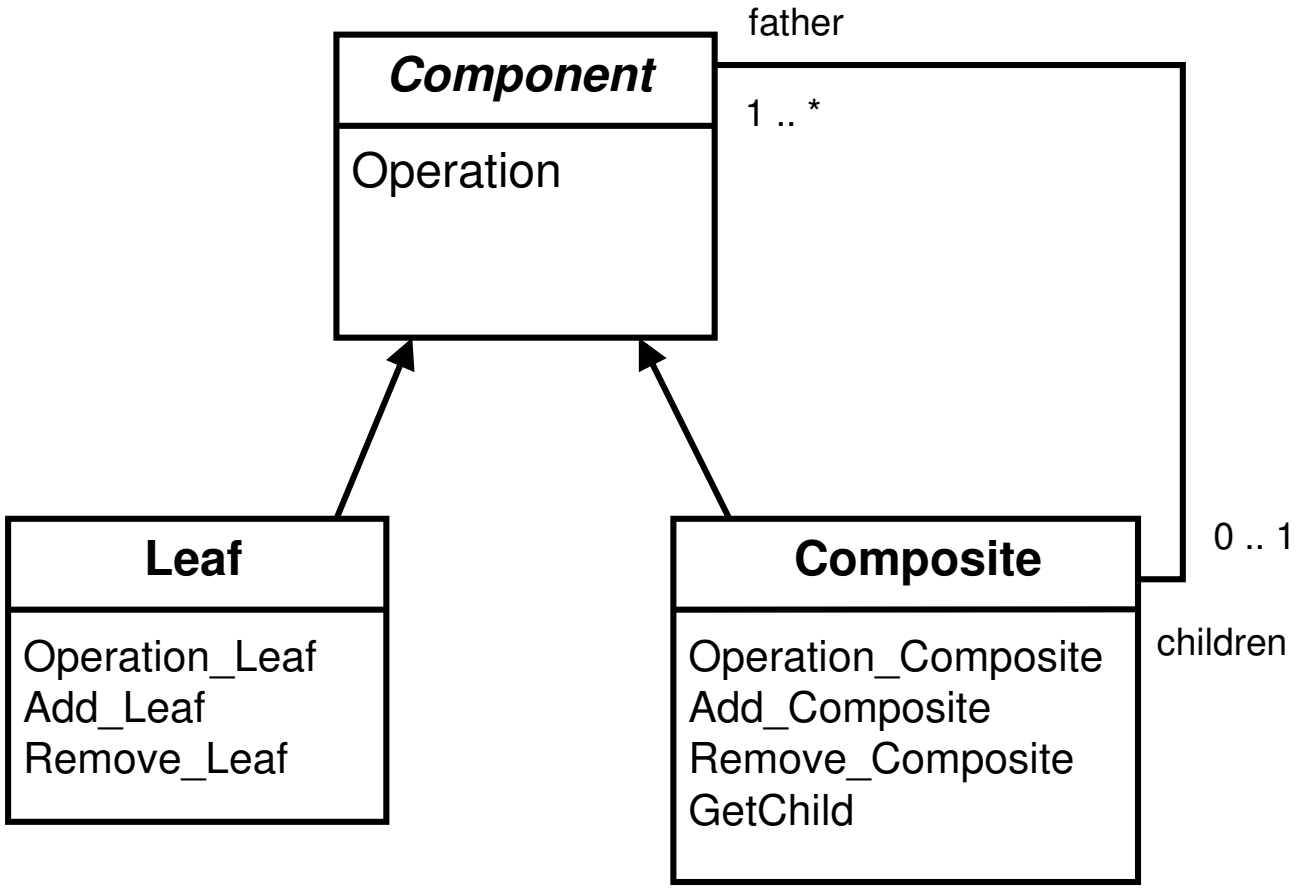
- Pas de concept de pattern en B


UML-B Pattern
UML-B Pattern
Bibliothèque de
UML-B Patterns
Instanciation
Composition
Extension
Spécification UML-B
d'une application

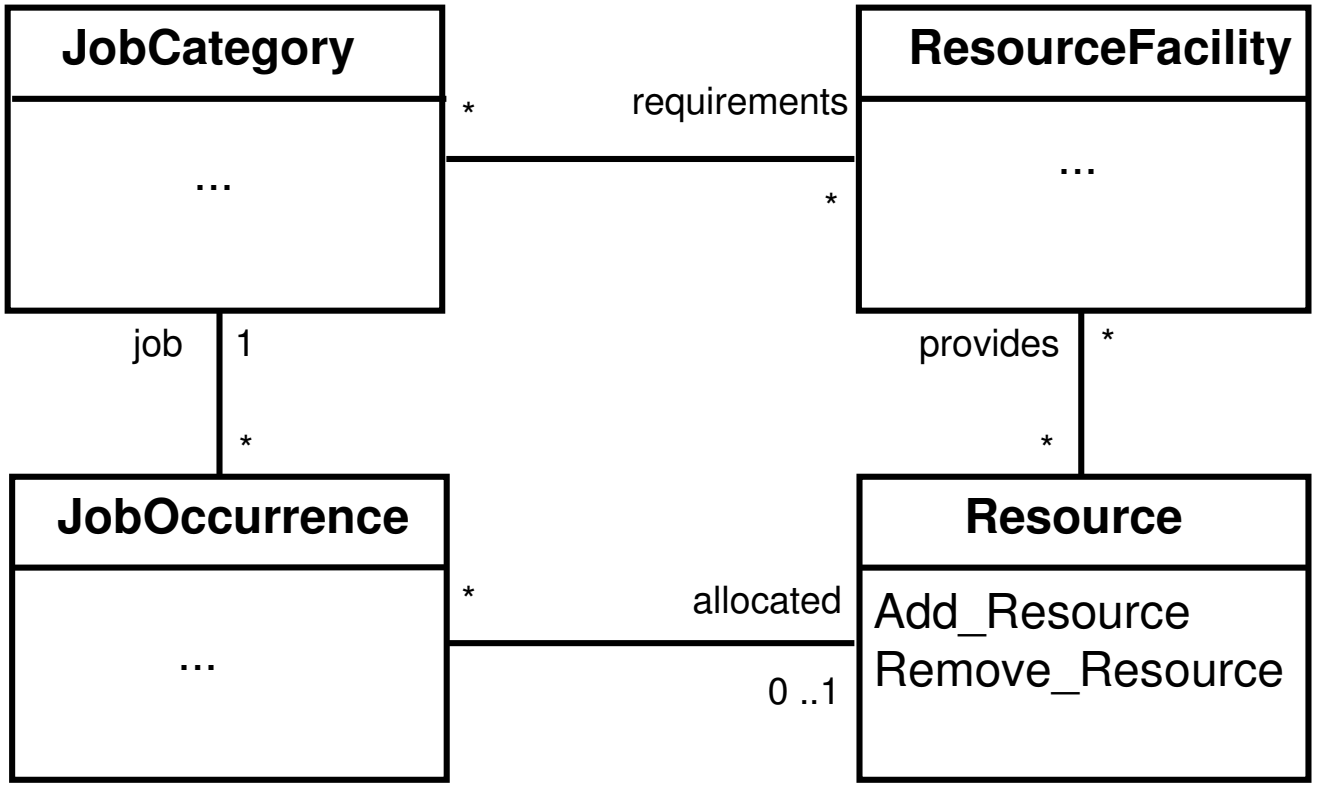
Plan

- Pattern :
  - Exemples : Composite, Resource Allocation
  - Exemple de réutilisation en UML
- Notre approche de réutilisation de patterns de spécification avec la méthode B :
  - Définition d'un pattern de spécification en B
  - Mécanismes de réutilisation en B :
    - instanciation
    - composition
    - extension
- Conclusion et perspectives

Pattern *Composite*



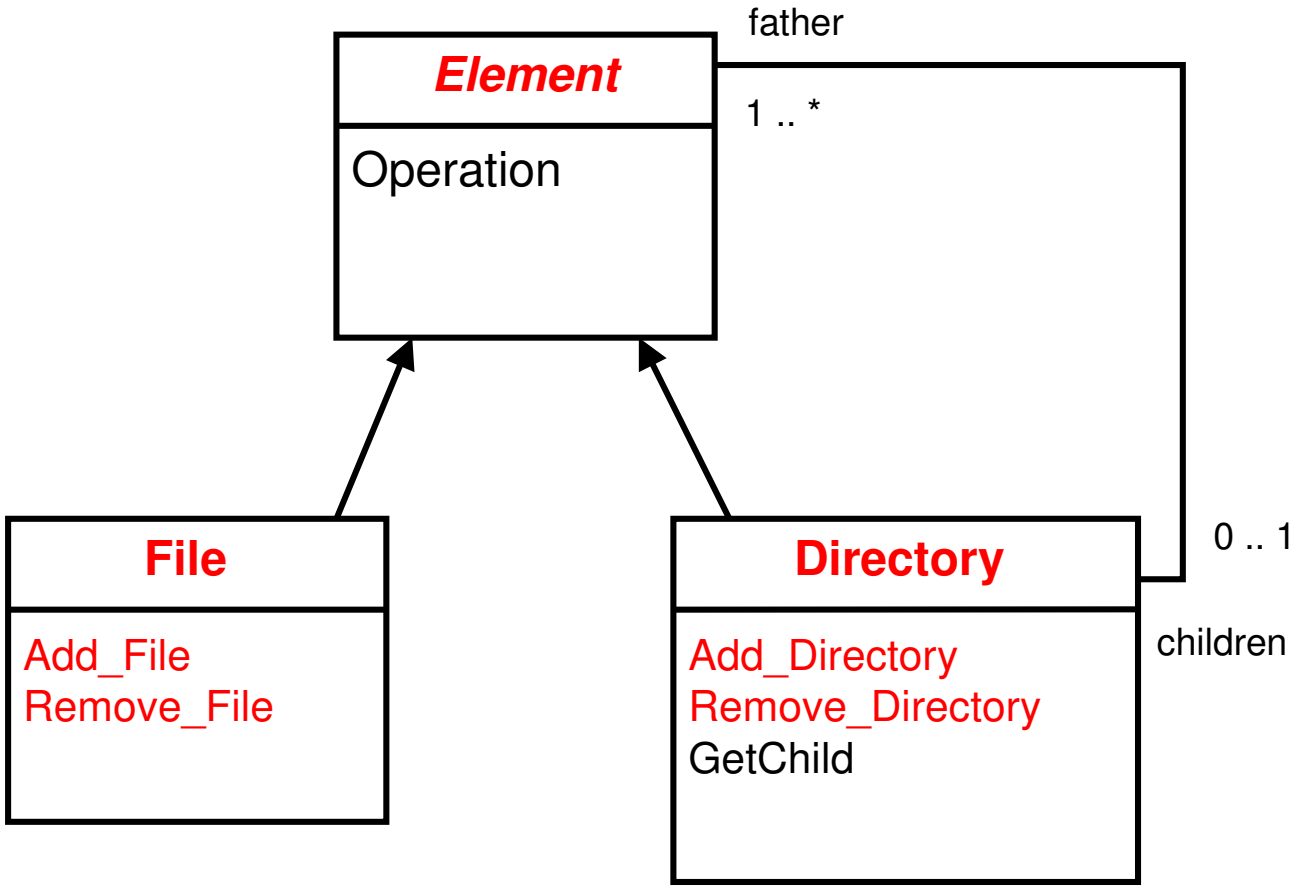
Pattern *Resource Allocation*



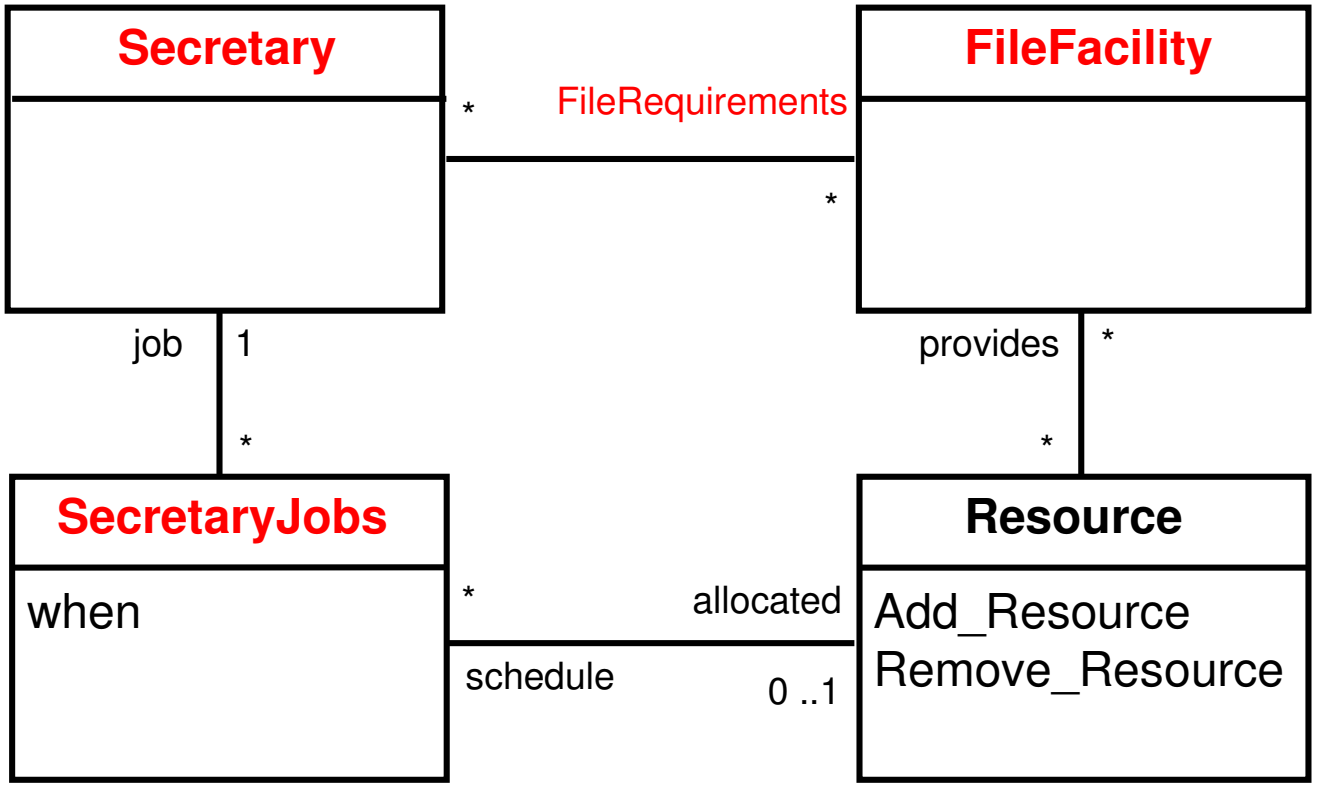
Exemple de réutilisation

- Allocation de dossiers à des secrétaires
- Un dossier : un objet composite
- Un problème d'allocation de ressources
- Réutilisation des patterns Composite et Resource Allocation

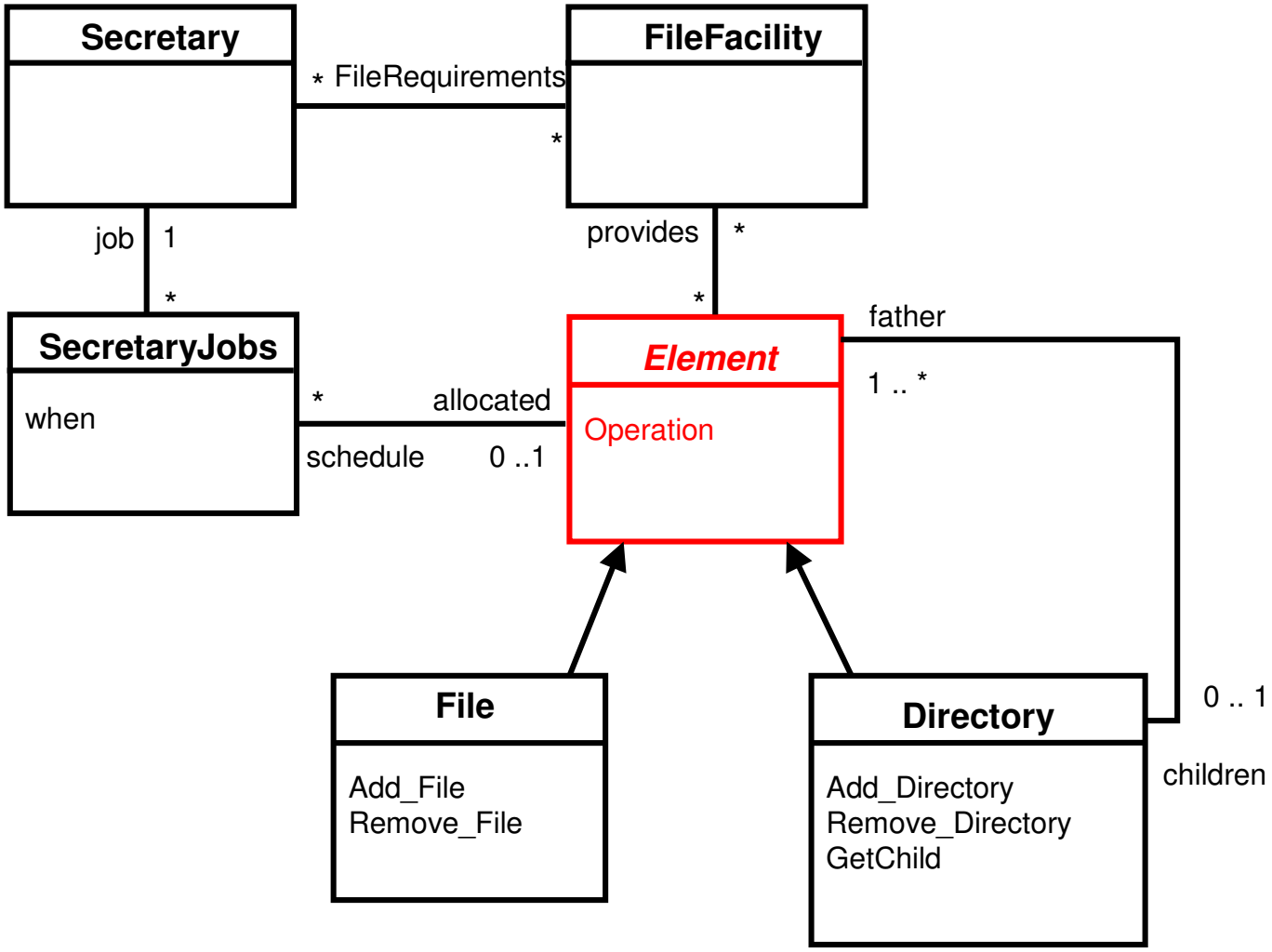
Exemple de réutilisation : instanciation de *Composite*



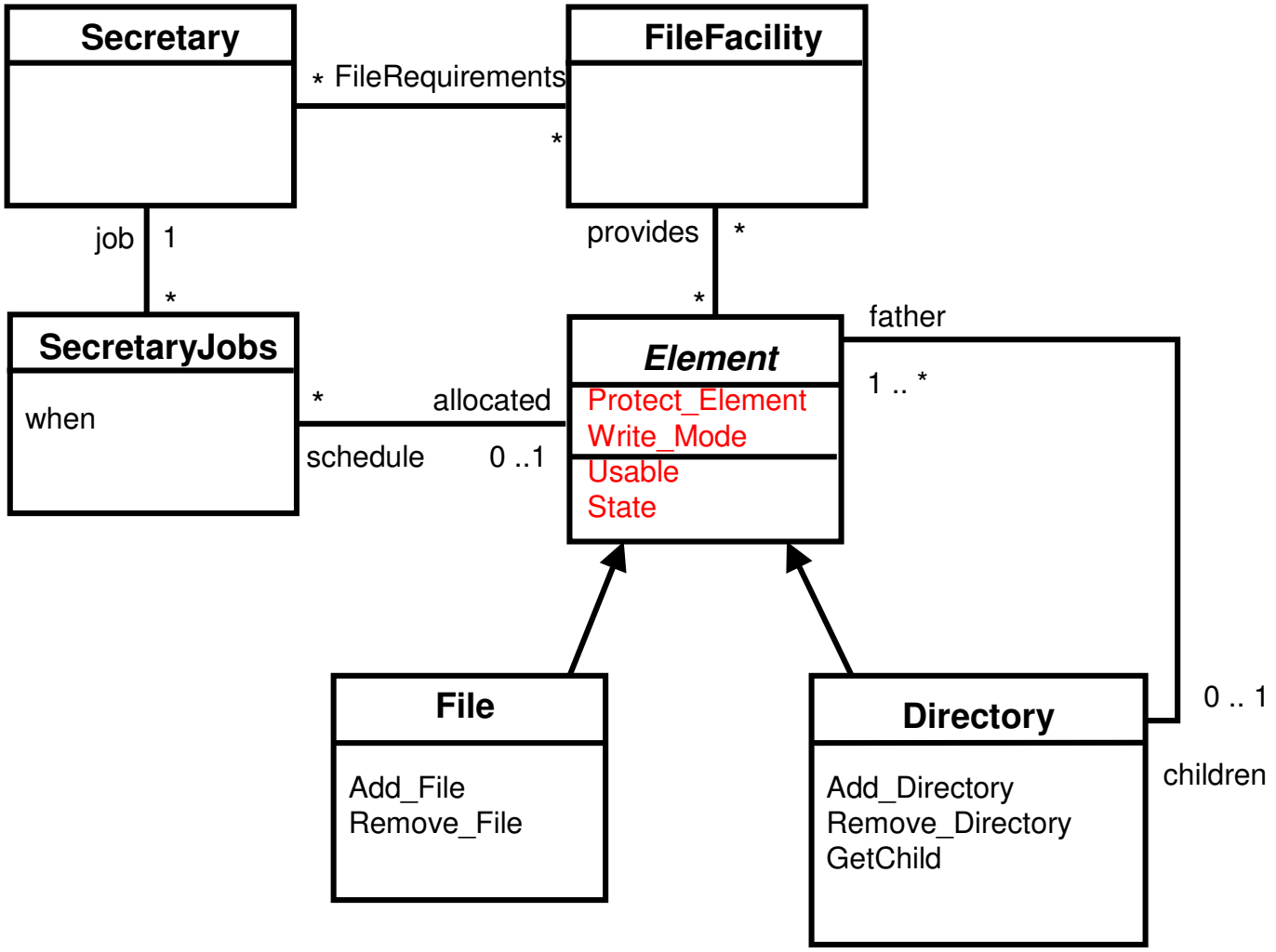
Exemple de réutilisation : instanciation de *Resource Allocation*



Exemple de réutilisation : unification de *Resource* et *Element*



Exemple de réutilisation: Extension par ajout de caractéristiques dans *Element*



Réutilisation de patterns de spécification : les problèmes

Quatre problèmes à résoudre en B :

- **1. Définition des patterns**
- Mécanismes de réutilisation :
  - **2. Instanciation** : renommer les éléments d'un pattern
  - **3. Composition** : combiner plusieurs patterns
  - **4. Extension** : ajouter de nouvelles spécifications

Notre proposition

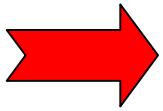
- Pourquoi B ? Pourquoi pas un nouveau langage ou une extension de B ?
  - Utilisation des mécanismes de B (inclusion, raffinement)
  - Utilisation des outils de B (Atelier B)
- Une approche pragmatique (on montre la faisabilité)

Définition de patterns

Pattern de spécification : une machine unique

- Raison technique : besoin pour certains mécanismes comme le raffinement
- Première limite de notre approche

Ensembles abstraits spécifiés comme paramètres de la machine



Pour implémenter la composition

MACHINE Composite_Pattern(COMPONENT)

VARIABLES Component, Composite, Leaf, Father

INVARIANT $Component \subseteq COMPONENT \wedge Composite \subseteq Component \wedge Leaf \subseteq Component \wedge Father \in Component \nrightarrow Composite \wedge Leaf \cup Composite = Component \wedge Leaf \cap Composite = \varnothing$

…

OPERATIONS

Remove_Leaf(leaf) =

pre $leaf \in Leaf \wedge leaf \in DOM(Father)$

then …

end

MACHINE Resource_Allocation(JOBS, CATEGORY, FACILITY, RESOURCE)
SETS DATE
VARIABLES JobOccurrence, Resource, Provides, Allocated, …
INVARIANT JobOccurrence $\subseteq$ JOBS $\wedge$ Resource $\subseteq$ RESOURCE $\wedge$
Provides $\in$ Resource $\leftrightarrow$ ResourceFacility $\wedge$
Allocated $\in$ JobOccurrence $+\!\!\rightarrow$ Resource $\wedge$ …
OPERATIONS
Remove_Resource(res) =
pre res $\in$ Resource
then
    Resource := Resource – {res} ||
    Provides := {res} $\lhd$ Provides ||
    Allocated := Allocated $\rhd$ {res}
end;
…

Instanciation

Mécanisme d'instanciation : inclusion de machine

Renommage :

- des ensembles : instanciation des paramètres
- des variables : clause **DEFINITION**
- des opérations : 2 cas
  - Pas de renommage : clause **PROMOTES**
  - Renommage : spécification dans **OPERATIONS**

MACHINE Composite_Pattern(COMPONENT)

VARIABLES Component, Composite, Leaf, Father

INVARIANT $Component \subseteq COMPONENT \wedge Composite \subseteq Component \wedge$
$Leaf \subseteq Component \wedge Father \in Component \twoheadrightarrow Composite \wedge$
$Leaf \cup Composite = Component \wedge Leaf \cap Composite = \varnothing$

…

OPERATIONS

Remove_Leaf(leaf) =

pre $leaf \in Leaf \wedge leaf \in DOM(Father)$

then …

end

MACHINE Directory_Renaming
SETS ELEMENT
INCLUDES Composite_Pattern(ELEMENT)
DEFINITIONS Directory == Composite,
    File == Leaf
…
OPERATIONS
Remove_File(file) =
pre file ∈ File ∧ file ∈ DOM(Father)
then Remove_Leaf(file)
end;
…

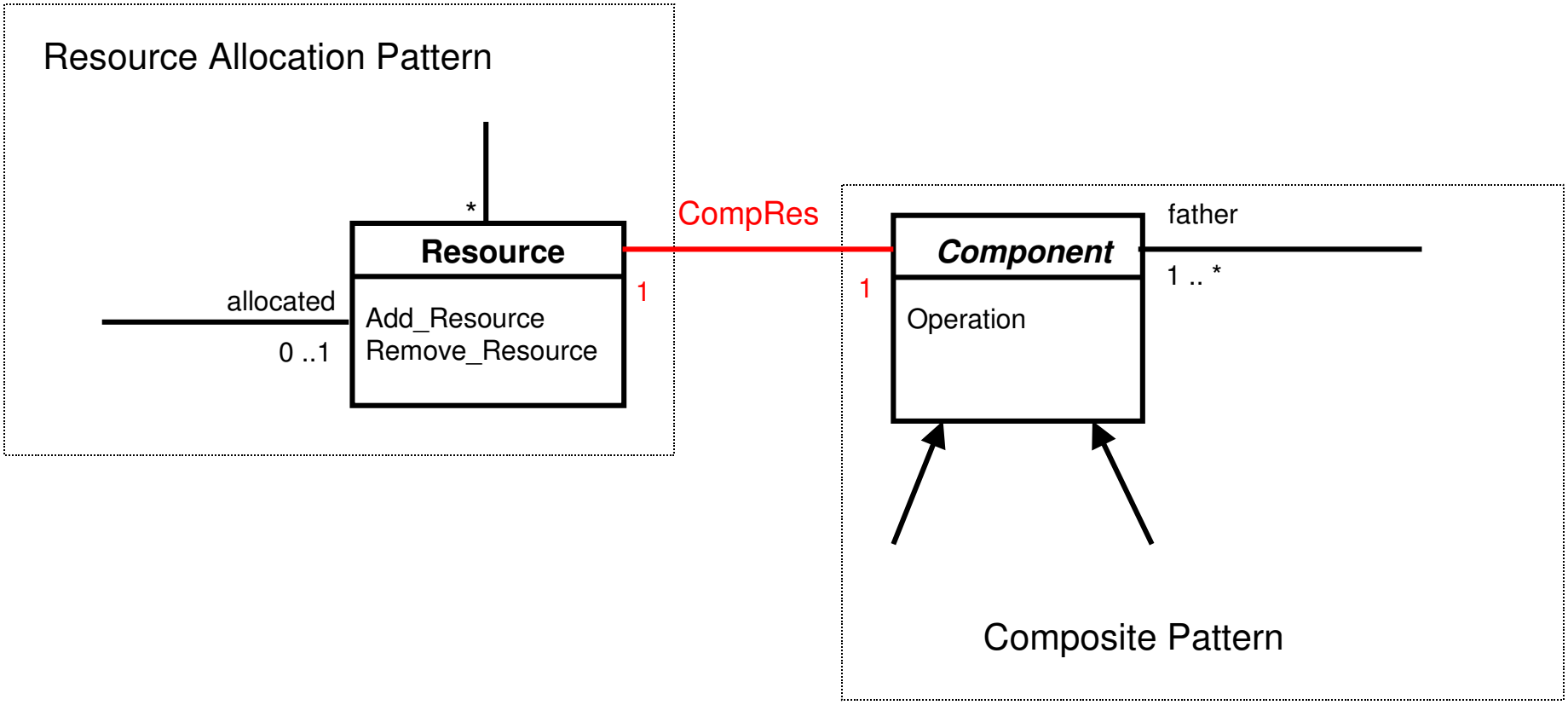
Composition

Mécanisme de composition : inclusion de machines

Trois niveaux de composition selon les liens entre les patterns composés :

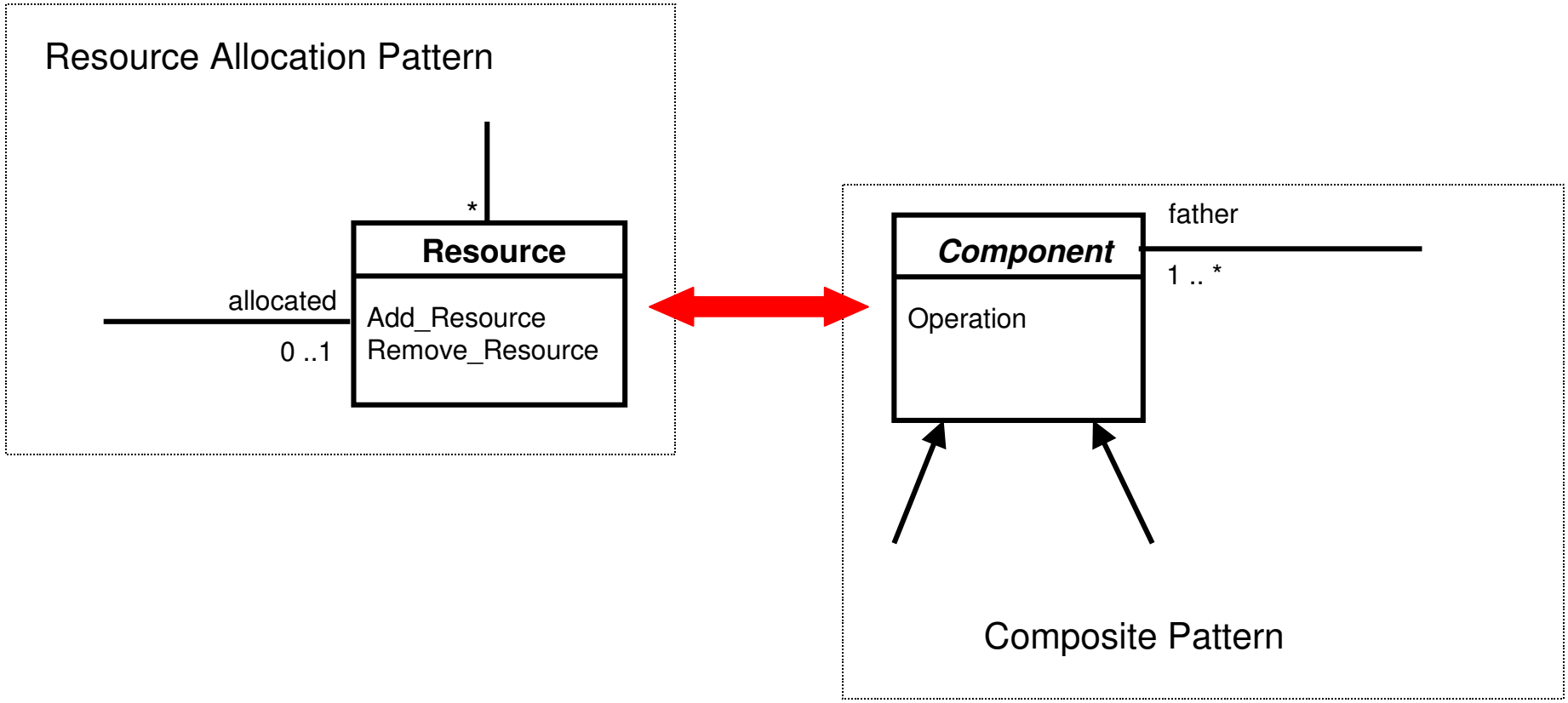
- **Juxtaposition** : pas de lien entre patterns
- Composition avec des **liens inter-patterns**
- **Unification** de certains éléments

Lien inter-patterns : exemple



MACHINE Composition_By_InterPatterns_Link
SETS COMPONENT; JOBS; CATEGORY; FACILITY; RESOURCE
INCLUDES
    Composite_Pattern(COMPONENT),
    Resource_Allocation(JOBS, CATEGORY, FACILITY, RESOURCE)
VARIABLES CompRes
INVARIANT CompRes $\in$ Component $\rightarrowtail\!\!\!\!\!\rightarrow$ Resource
OPERATIONS
Remove_Thing(thing) =
pre thing $\in$ Component $\wedge$ thing $\in$ DOM(Father)
then
    Remove_Leaf(thing) ||
    Remove_Resource(CompRes(thing)) ||
    CompRes := {thing} $\lhd$ CompRes
end;
…

Unification



Unification

MACHINE Composition_By_Unification
SETS ELEMENT; JOBS; CATEGORY; FACILITY
INCLUDES
    Composite_Pattern(ELEMENT),
    Resource_Allocation(JOBS, CATEGORY, FACILITY, ELEMENT)
INVARIANT Component = Resource
…

Extension

Extension: ajouter de nouveaux ensembles, de nouvelles variables et de nouvelles opérations

(pas de suppression)

Extension : 2 solutions

- Inclusion
- Raffinement

Notre choix : raffinement

Extension des opérations

- **Opérations existantes** : spécification des nouvelles substitutions en utilisant les nouvelles variables

- **Nouvelles opérations** : n'impliquent que de nouvelles variables et des appels d'opérations de la machine raffinée

Extension d'opérations existantes

Remove_File(file) =
pre file $\in$ File $\wedge$ file $\in$ DOM(Father)
then
    Remove_Leaf(file) ||
    Remove_Resource(file)
end

Extension d'opérations existantes

Remove_File(file) =
pre file $\in$ File $\wedge$ file $\in$ DOM(Father)
then
    Remove_Leaf(file) ||
    Remove_Resource(file) ||
    Usable := Usable – {file} ||
    State := {file} $\lhd$ State
end

Nouvelles opérations

Dans la machine à étendre :

Protect_Element(el) =

pre el $\in$ Element

then SKIP

end

Nouvelles opérations

Dans la machine à étendre :

Protect_Element(el) =

pre el $\in$ Element

then SKIP

end

Dans le raffinement:

Protect_Element(el) =

pre el $\in$ Element

then State(el) := protected || Usable := Usable $\cup$ {el}

end

Activité de preuve

- Toutes les machines ont été prouvées avec Atelier B
- Instanciation : pas de nouvelle PO (les PO générées sont automatiquement prouvées)
- Composition : pas de nouvelle PO par construction
- Extension : nouvelles PO
  - Nouvelles substitutions
  - Raffinement

Quatre PO liées au raffinement sont réutilisables pour d'autres extensions de la même composition de patterns

Conclusion

- Une approche de réutilisation de patterns de spécification avec la méthode B
- Utilisation des mécanismes de B exclusivement
- Deux principaux défauts :
  - Définition d'un pattern : une machine unique
  - Besoin de définir les nouvelles opérations dans la machine à étendre
- Introduction de la notion de réutilisation de preuve

Perspectives

- Composition de plusieurs instances du même pattern
- Règles formelles pour définir correctement la composition des opérations
- Formalisation des mécanismes de réutilisation de patterns
- Développement d'un outil d'assistance
- Généralisation de la réutilisation de preuves
- Création d'une bibliothèque de patterns de spécification