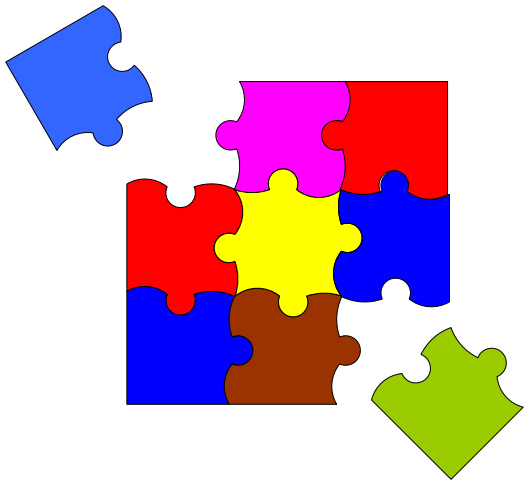
Contribution à une démarche de vérification formelle d'architectures logicielles



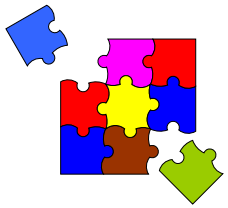
Soutenance de thèse
présentée par

Mohamed GRAIET

dirigée par

Jean-Pierre GIRAUDIN, Abdelmajid Ben HAMADOU & Mohamed Tahar BHIRI

25 octobre 2007



Plan

-Problématique

-La démarche DVFAL

-L'ADL Wright comme langage pivot

-Traduction d'une architecture logicielle Wright vers Ada

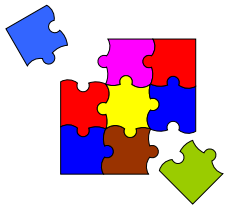
-Vérification formelle d'architectures logicielles à base UML

-Le profil W-UML

-Vérification formelle d'architectures logicielles Symphony

-Outils développés

-Conclusion & perspectives



Problématique

-Architecture Logicielle : AL

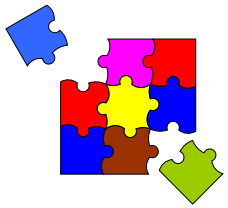
-Pluralité des formalismes de description des AL

-Pluralité des classes de propriétés

-Propriétés standards / propriétés spécifiques

-Pluralité des techniques de vérification

- Le sujet



Problématique

-Fournit une description de haut niveau de la structure d'un système

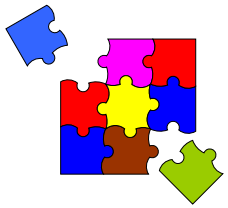
-Est définie par des composants, des connecteurs et des configurations (Garlan,1993)

-Favorise le passage de l'étape de conception à l'étape d'implémentation

-Facilite la maintenance corrective et évolutive

-Favorise une approche incrémentale de développement

Mais, ceci suppose que l'architecture logicielle possède des propriétés bien définies et prouvées formellement par des outils appropriés.



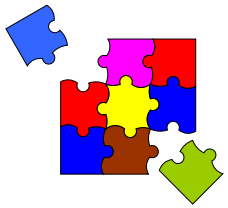
Problématique

Pluralité des formalismes de description des AL

-Langages de description d’architectures : (ADL)

- UML et ses extensions

- Formalismes orientés composants



Problématique

-Un ADL est un langage qui modélise l'architecture du système sous forme graphique ou textuelle

-Trois concepts partagés par la plupart des ADL : Composant, Connecteur et Configuration (Garlan,1993)

- Rares sont les ADL qui permettent de raisonner formellement sur les architectures logicielles tels que Wright et Rapide

Problématique

UML et ses extensions -sous forme des profils- sont moins formels et moins rigoureux que les ADL

Avantages

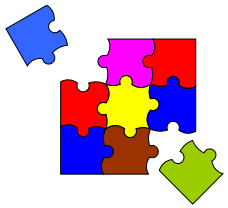
-Permettre à plusieurs intervenants qui ont peu de connaissances dans le domaine des spécifications formelles de comprendre et manipuler une description architecturale

-Tirer profit des outils supportant le standard UML

Inconvénient

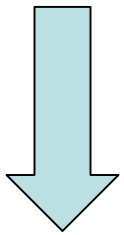
Une description architecturale en UML ne peut pas être analysée d'une façon rigoureuse et formelle dans le cadre UML

Solution potentielle: **ouverture d'UML sur certains ADL formels**

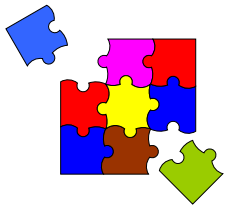


Problématique

Ces formalismes (Symphony, Ugatze, …) comme UML ne favorisent pas une analyse rigoureuse d'une architecture logicielle



Des ouvertures de ces formalismes sur certains ADL formels s'imposent



Problématique

Pluralité des classes de propriétés (Schnoebelen 1995)

-Les propriétés d'atteignabilité

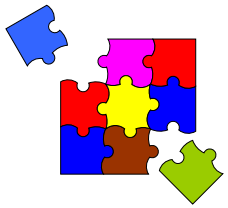
-Les propriétés de sûreté

-Les propriétés de vivacité

-L'absence de blocage

-Les propriétés d'équité

→ **Applicables sur les AL**



Problématique

Propriétés standards / propriétés spécifiques

Les propriétés **standards** :

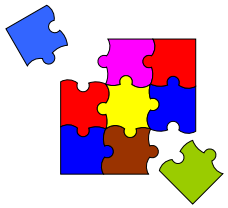
-Sont communes à toutes les architectures logicielles

-Sont attachées aux trois principaux concepts architecturaux : composant, connecteur et configuration

-Sont liées à la **cohérence** de chaque concept architectural

-Dépendent du formalisme utilisé

-Sont plus ou moins automatisables



Problématique

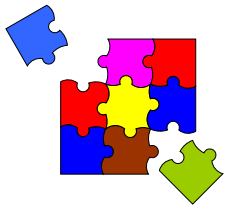
Propriétés standards / propriétés spécifiques

Les propriétés **spécifiques** :

-Dépendent du système étudié

-Conditionnent tout raffinement ultérieur de l'architecture du système étudié

-Peuvent être hiérarchisées : domaine et type d'architecture



Problématique

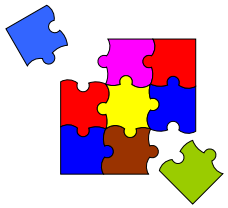
Pluralité des techniques de vérification

Les techniques de vérification de **modèles** :

-La simulation

-La preuve de théorèmes

-Le model checking



Problématique

Pluralité des techniques de vérification

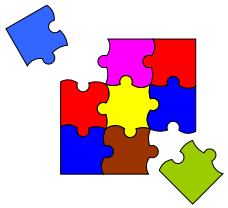
Les techniques de vérification des **programmes** :

-Le test

-L’analyse statique par interprétation abstraite

-La preuve

-Le model checking



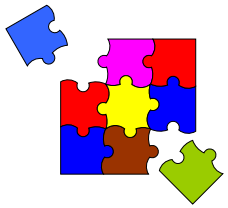
Le sujet

comment peut-on vérifier la correction d'une Architecture Logicielle (AL) décrite dans n'importe quel formalisme adéquat ?

- Un formalisme adéquat est un formalisme ayant des aptitudes pour décrire les aspects structuraux et comportementaux d'une AL

-Une AL est jugée correcte si et seulement si elle possède des propriétés bien définies

Nous apportons une contribution à une démarche de Vérification Formelle d'Architectures Logicielles : DVFAL



Plan

-Problématique

-La démarche DVFAL

-L'ADL Wright comme langage pivot

-Traduction d'une architecture logicielle Wright vers Ada

-Vérification formelle d'architectures logicielles à base UML

-Le profil W-UML

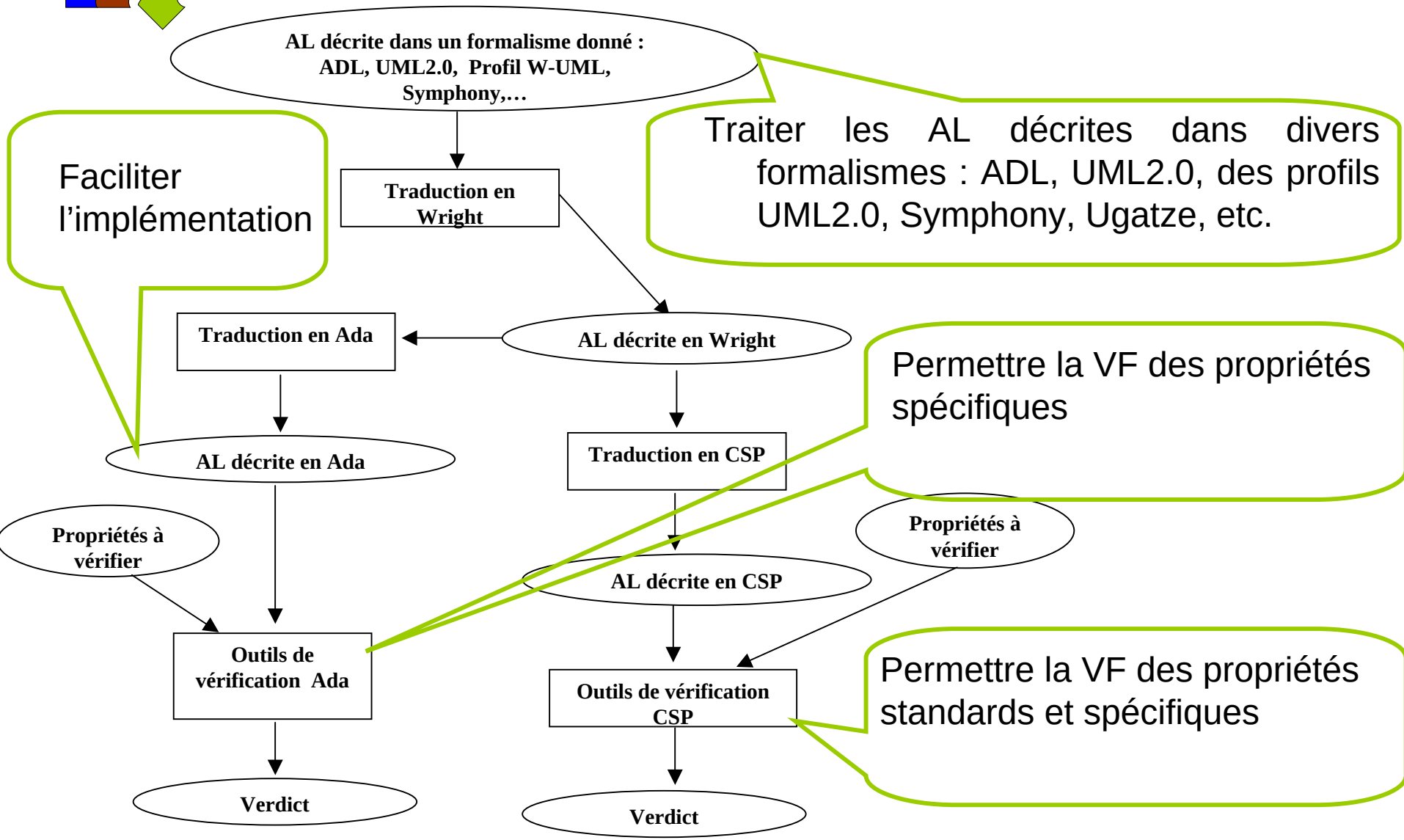
-Vérification formelle d'architectures logicielles Symphony

-Outils développés

-Conclusion & perspectives

La démarche DVFAL

Vue d'ensemble

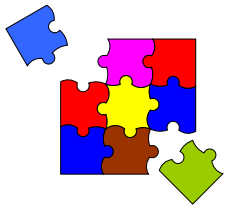


Faciliter l'implémentation

Traiter les AL décrites dans divers formalismes : ADL, UML2.0, des profils UML2.0, Symphony, Ugatze, etc.

Permettre la VF des propriétés spécifiques

Permettre la VF des propriétés standards et spécifiques



La démarche DVFAL

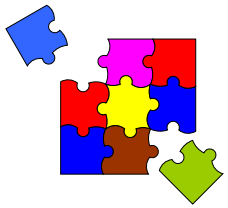
Les traducteurs de la démarche DFVAL

La démarche propose trois types de traducteurs :

- Traducteurs de formalismes utilisés vers Wright

- Traducteur de Wright vers CSP

- Traducteur de Wright vers Ada



Plan

-Problématique

-La démarche DVFAL

-L'ADL Wright comme langage pivot

-Traduction d'une architecture logicielle Wright vers Ada

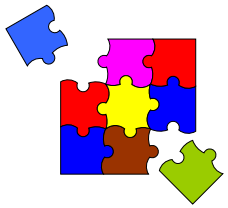
-Vérification formelle d'architectures logicielles à base UML

-Le profil W-UML

-Vérification formelle d'architectures logicielles Symphony

-Outils développés

-Conclusion & perspectives



L'ADL Wright comme langage pivot

Les points forts

✓ Aspects **structuraux**:

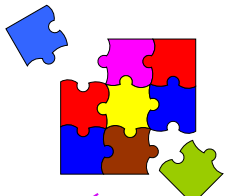
-Composant

-Connecteur
- -permet de définir des **patterns** d'interaction
- -permet de concevoir des connecteurs réutilisables

-Configuration

-Style → permet de définir les caractéristiques communes à plusieurs configurations

✓ Aspects **comportementaux: CSP** → Ouverture sur les outils de vérification formelle supportant CSP tels que FDR et Satchecker



L'ADL Wright comme langage pivot

Les points forts

✓ Langage de contraintes : permet la définition des propriétés architecturales orientées état (Propriétés invariantes attachées aux styles)

✓ Des propriétés standards :

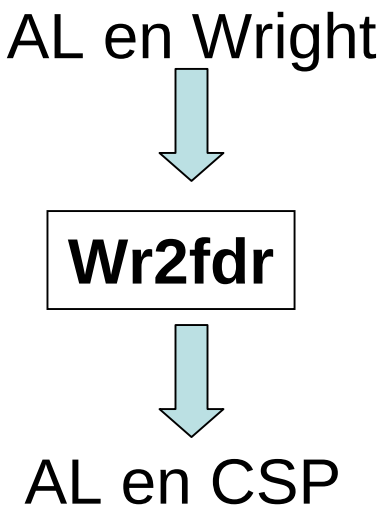
1 **Cohérence des ports avec le Calcul (composant)**
2 **Absence d'interblocage sur les connecteurs (connecteur)**
3 **Absence d'interblocage sur les rôles (rôle)**
4 **Initialiseur unique (connecteur)**
5 **Engagement de l'initialiseur (n'importe quel processus)**
6 **Substitution de paramètres (instance)**
7 **Bornes d'un intervalle (instance)**
8 **Compatibilité port / rôle (lien)**
9 **Contraintes pour les styles (configuration)**
10 **Cohérence de style (style)**
11 **Complétude des liens (configuration)**

L'ADL Wright comme langage pivot

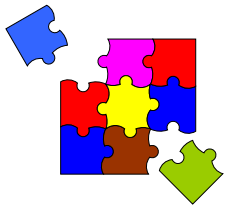
Les points forts

✓**Vérification formelle:**

- Wr2fdr : automatisation de certaines propriétés (4/11) à l'aide du raffinement CSP



- vérification avec l'outil FDR



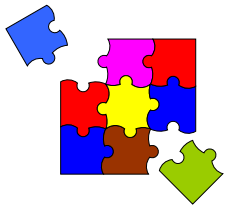
L’ADL Wright comme langage pivot

Les points faibles

-Insuffisances liées à l’outil **Wr2fdr :** absence de distinction entre événements initialisés et observés, pas d’interface, sémantique statique de l’ADL Wright non traitée, erreurs liées à la génération du code CSP

- Automatisation des autres propriétés (7/11)

- Absence des propriétés standards liées aux configurations par exemple l’absence d’interblocage



L'ADL Wright comme langage pivot

Les points faibles

Les propriétés spécifiques vérifiables dans le cadre de CSP ne couvrent pas toutes les classes de propriétés

⇩

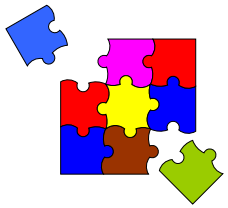
La technique de model checking appliquée sur des programmes (programmes concurrents)

⇩

vérification des propriétés spécifiques plus ou moins diversifiées

Mais, ceci suppose le passage d'une représentation architecturale en Wright vers une représentation architecturale sous forme d'un programme concurrent

→ pour faciliter un tel passage un bon candidat est Ada



Plan

-Problématique

-La démarche DVFAL

-L'ADL Wright comme langage pivot

-Traduction d'une architecture logicielle Wright vers Ada

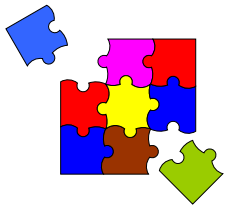
-Vérification formelle d'architectures logicielles à base UML

-Le profil W-UML

-Vérification formelle d'architectures logicielles Symphony

-Outils développés

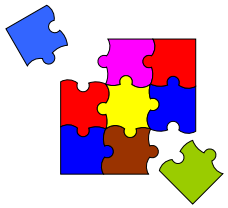
-Conclusion & perspectives



Traduction d'une architecture logicielle Wright vers Ada

-Rapprochement sémantique entre CSP et Ada : concept du rendez-vous

-Outils d'analyse statique supportant Ada : SPIN,SMV, INCA et FLAVERS → Propriétés orientées état et propriétés orientées chemin → propriétés de sûreté, de vivacité et d'équité



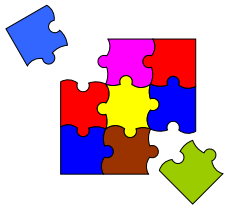
Traduction d'une architecture logicielle Wright vers Ada

Une configuration Wright est traduite en Ada par un programme concurrent dans lequel :

- Chaque instance de type «Component » est traduite par une tâche Ada

- Chaque instance de type «Connector» est traduite également par une tâche Ada

- Les tâches de même type ne communiquent pas entre elles



Traduction d'une architecture logicielle Wright vers Ada

Traduction des événements observés

Un événement observé est traduit par une entrée (**entry**) et par une acceptation de rendez-vous (instruction **accept**)

Traduction des événements initialisés

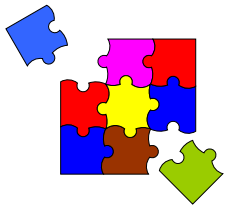
Un événement initialisé de la forme $\overline{e}!x$ est traduit par une demande de rendez-vous sur l'entrée e exportée par une tâche de type différent à identifier

Traduction des interfaces des composants

L'interface d'un composant Wright est traduite par une interface d'une tâche Ada

Traduction des interfaces des connecteurs

L'interface d'un connecteur Wright est traduite par une interface d'une tâche Ada



Traduction d'une architecture logicielle Wright vers Ada

Cas 1 : a[?x] -> P

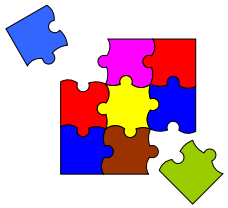
La traduction en Ada est :

```
accept a(…) do
        null ;
end a ;
traiter P
```

Cas 2 : a[!x] -> P

La traduction en Ada est :

```
nom_ tache.a(…);
traiter P
```



Traduction d'une architecture logicielle Wright vers Ada

Cas 1 : P= a[?x] -> Q -> P

La traduction en Ada est basée sur une boucle infinie :

```
loop
        accept a(…) do
                null ;
        end a;
        traiter Q
end loop;
```

Cas 2 : P= a[!x] -> Q -> P

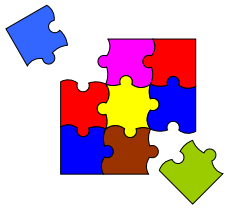
La traduction en Ada est également basée sur une boucle infinie :

```
loop
        nom_ tache.a(…);
        traiter Q
end loop;
```

Cas 1 & 2 : (a[?x] -> P) ∏ (a ou b [?x] -> Q)

La traduction en Ada est basée sur la construction non déterministe select :

```
select
        accept a(…) do
                null ;
        end a;
        traiter P;
        or
        accept a ou b(…) do
                null ;
        end a;
        traiter Q;
end select;
```



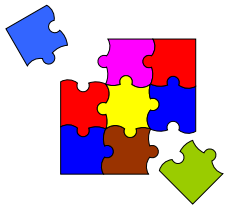
Traduction d'une architecture logicielle Wright vers Ada

Traduction de la partie calcul des composants

La partie «Computation» d'un composant Wright est traduite en un corps (partie implémentation) d'une tâche Ada conformément aux règles de traduction des opérateurs CSP

Traduction de la partie Glu des connecteurs

La partie «Glue» d'un connecteur Wright est traduite en un corps (partie implémentation) d'une tâche Ada conformément aux règles de traduction des opérateurs CSP



Plan

-Problématique

-La démarche DVFAL

-L'ADL Wright comme langage pivot

-Traduction d'une architecture logicielle Wright vers Ada

-Vérification formelle d'architectures logicielles à base UML

-Le profil W-UML

-Vérification formelle d'architectures logicielles Symphony

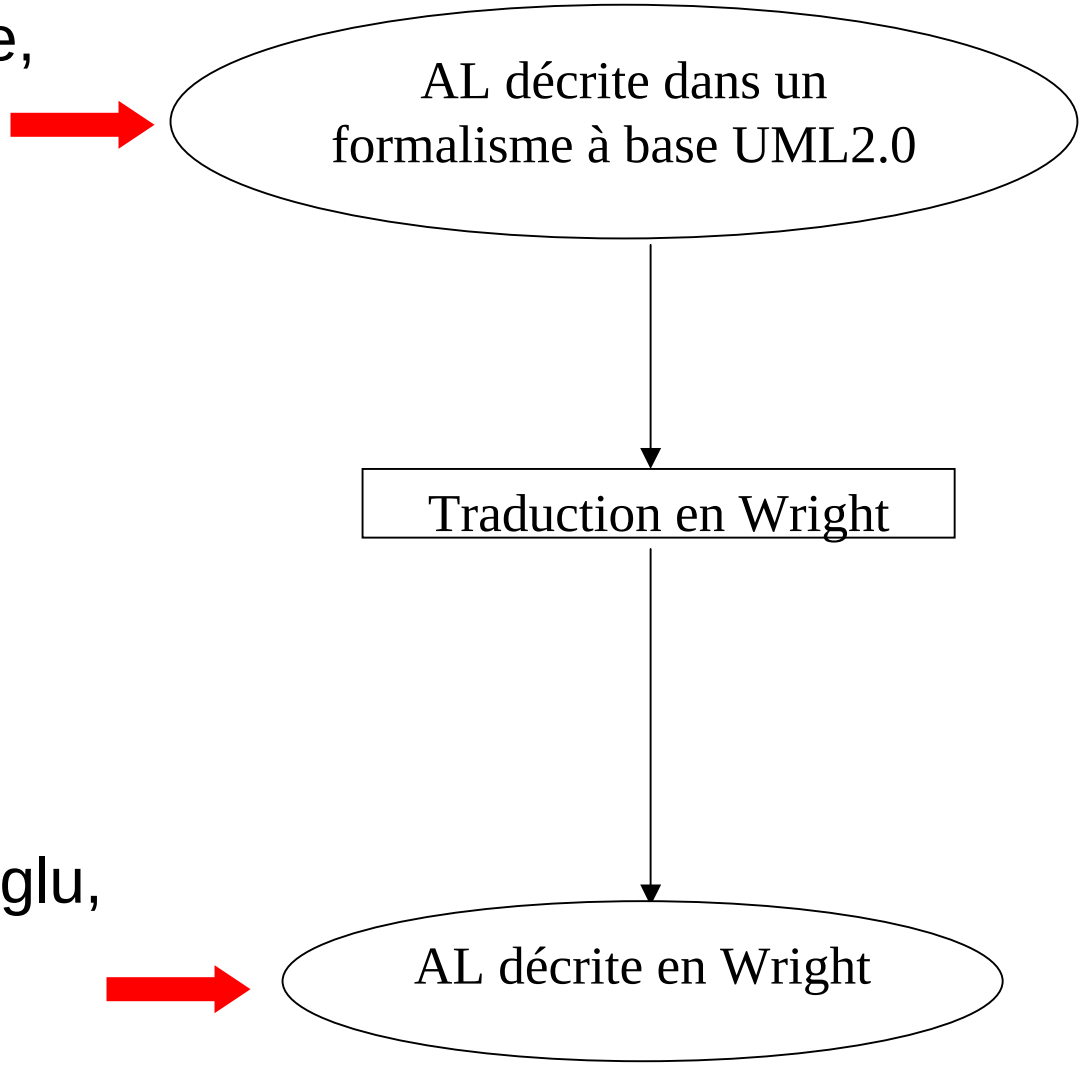
-Outils développés

-Conclusion & perspectives

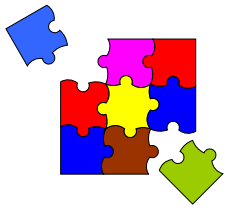
Vérification formelle d'architectures logicielles à base UML

Langage intermédiaire dans la démarche DVFAL

composant, port, interface, connecteur, machine de description de protocoles

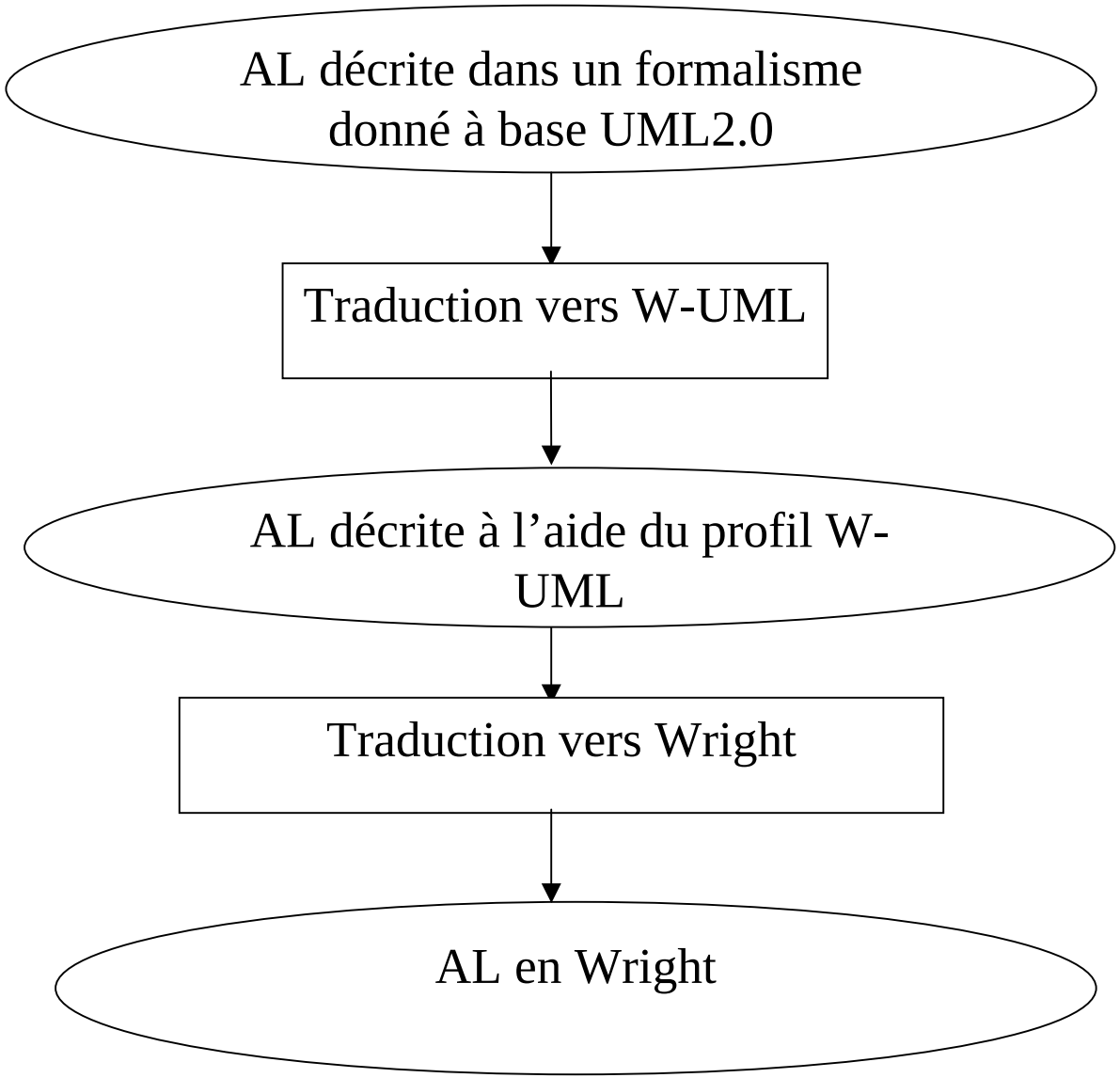


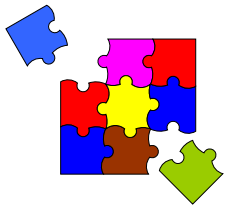
composant, connecteur, glu, rôle, port, configuration, calcul, CSP



Vérification formelle d'architectures logicielles à base UML

Langage intermédiaire dans la démarche DVFAL





Plan

-Problématique

-La démarche DVFAL

-L'ADL Wright comme langage pivot

-Traduction d'une architecture logicielle Wright vers Ada

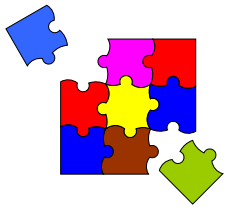
-Vérification formelle d'architectures logicielles à base UML

-Le profil W-UML

-Vérification formelle d'architectures logicielles Symphony

-Outils développés

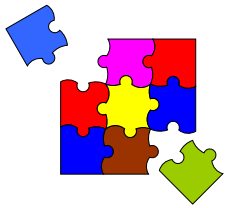
-Conclusion & perspectives



Le profil W-UML

- Traiter les aspects **structuraux** de Wright :
  component, connector et configuration.
- Traiter les aspects **comportementaux** de Wright :
  le langage CSP pour Wright.
- Relier les deux aspects structuraux et comportementaux de Wright pour obtenir le métamodèle voulu.

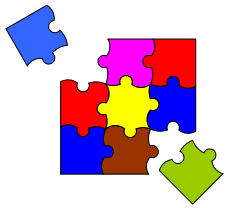
cerner d'une façon semi-formelle les concepts Wright coordonnés avec UML2.0.



Le profil W-UML

Correspondances entre les concepts

Métaclasses du métamodèle Wright	Métaclasses cibles du métamodèle UML2.0
Component	Component
Connector	Class
Process	StateMachine
Attachment	Connector

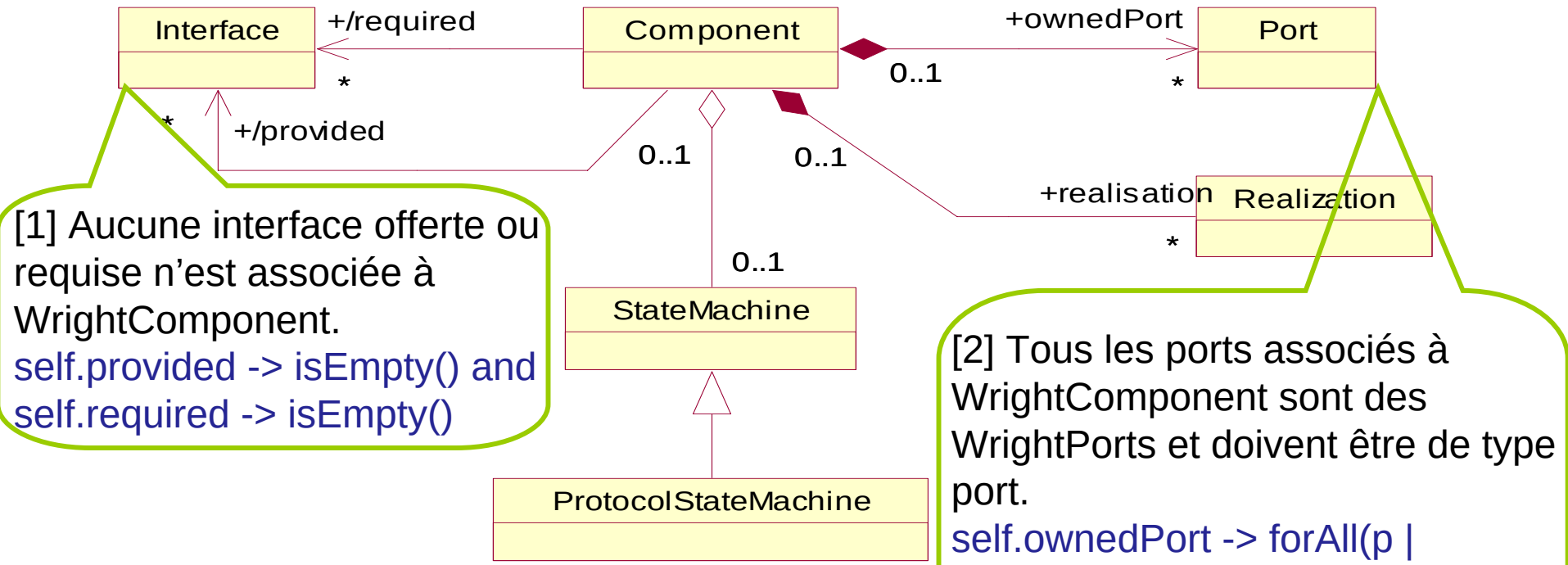


Le profil W-UML

Stéréotype	Métaclasse cible d'UML2.0
«WrightOperation»	Operation
«WrightProtocolTransition»	ProtocolTransition
«WrightVertex»	Vertex
«WrightRegion»	Region
«WrightProtocolStateMachine»	StateMachine
«WrightInterface»	Interface
«WrightPort»	Port
«WrightComponent»	Component
«WrightConnector»	Class
«WrightAttachement»	Connector

Le profil W-UML

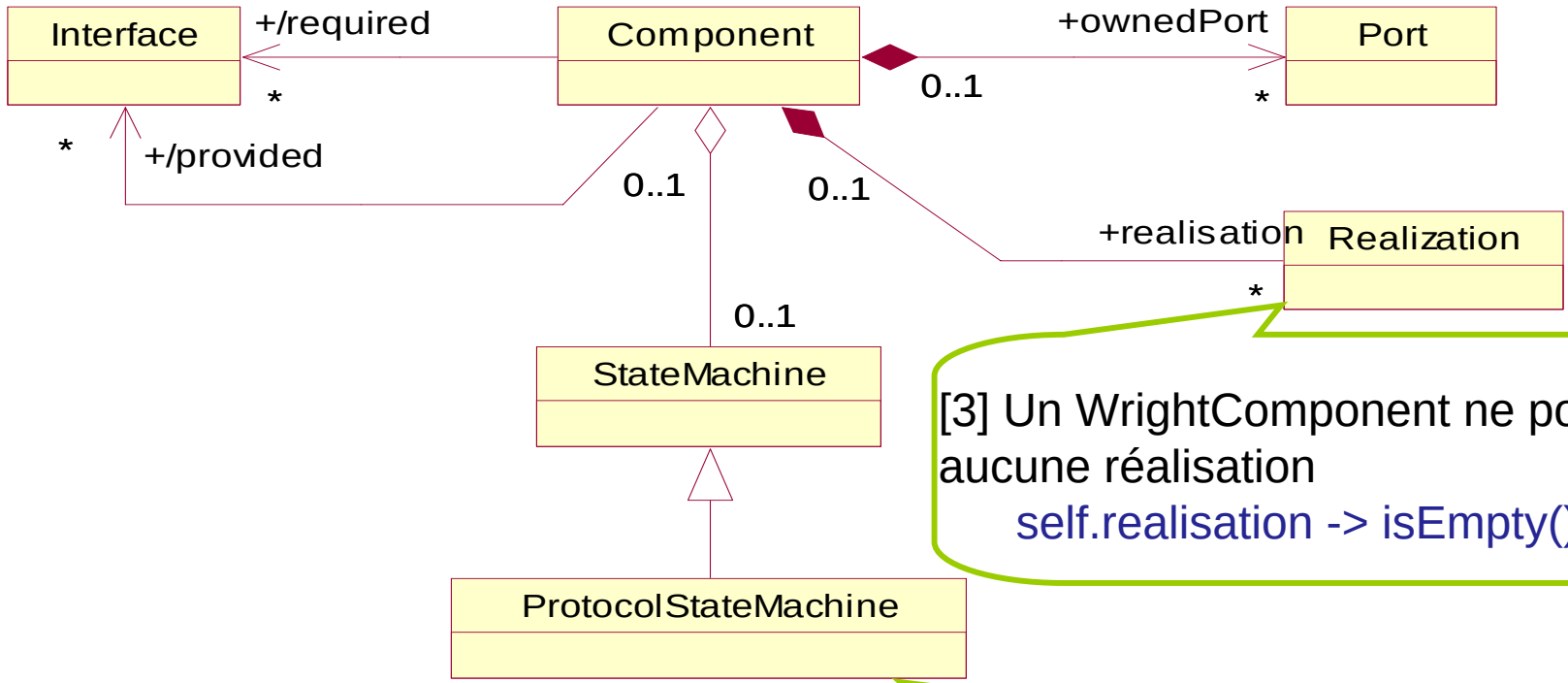
Stéréotype <<WrightComponent>>



La métaclasse Component dans le métamodèle UML2.0

Le profil W-UML

Stéréotype <<WrightComponent>>

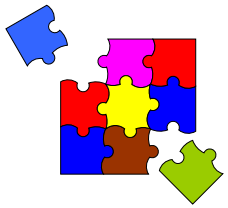


[3] Un WrightComponent ne possède aucune réalisation

self.realisation -> isEmpty()

[4] Un WrightProtocolStateMachine est associé au WrightComponent traduisant le protocole CSP de computation attaché à un composant Wright.

self.stateMachine -> size() = 1 and self.stateMachine. oclAsType (ProtocolStateMachine).stereotype = WrightProtocolStateMachine



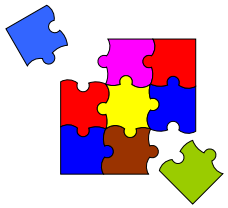
Exemple

Démarche

- Spécification formelle en Wright
- Vérification des propriétés standards

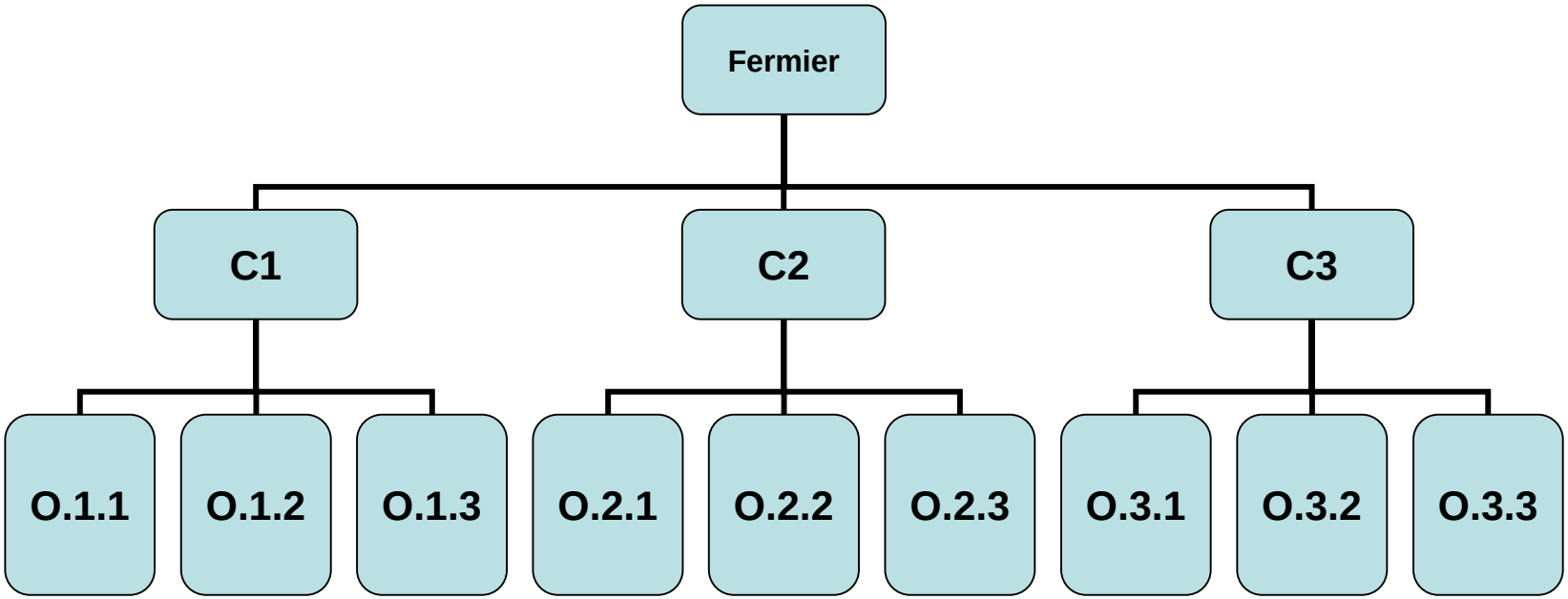
Fonctionnement d'une Ferme

- Un fermier emploie **n** contremaîtres
- Chaque contremaître est responsable de **m** ouvriers
- Le fermier, les contremaîtres et les ouvriers communiquent via des canaux



Exemple

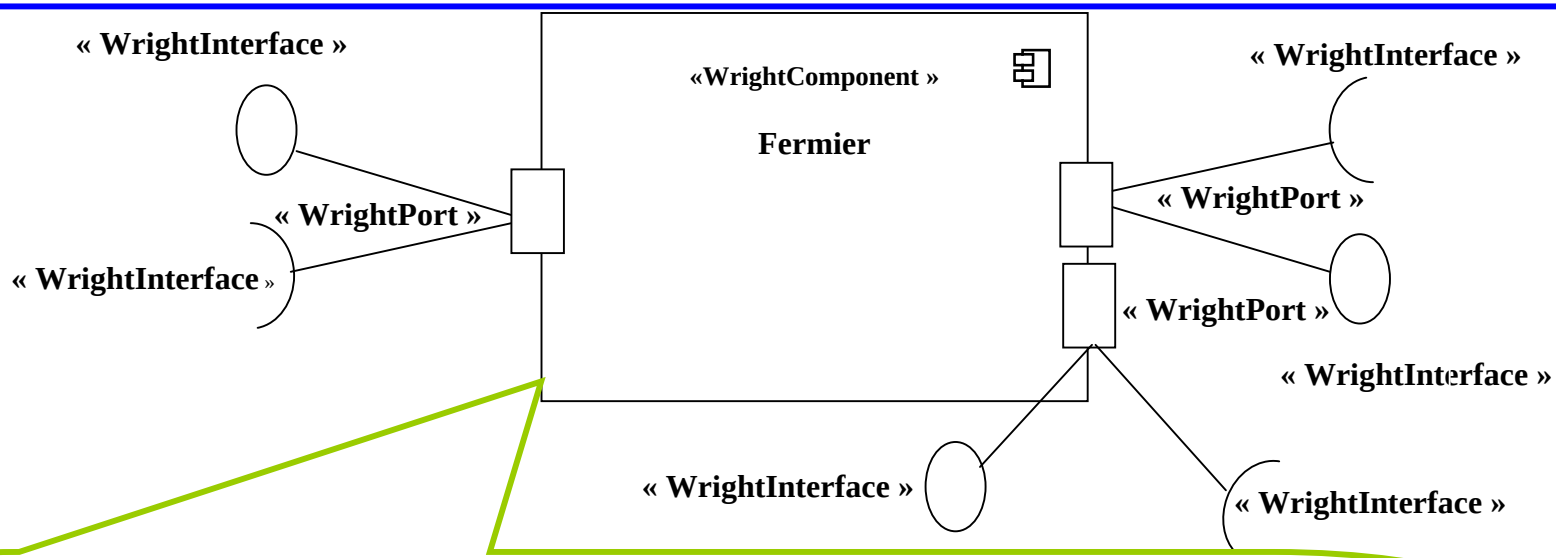
Instance n=3 et m=3



Exemple

Le composant Fermier

Description en W-UML



Component Fermier

Port Contremaitre1 = c?t -> $\overline{d!v}$ -> Contremaitre1

Port Contremaitre2 = c?t -> $\overline{d!v}$ -> Contremaitre2

Port Contremaitre3 = c?t -> $\overline{d!v}$ -> Contremaitre3

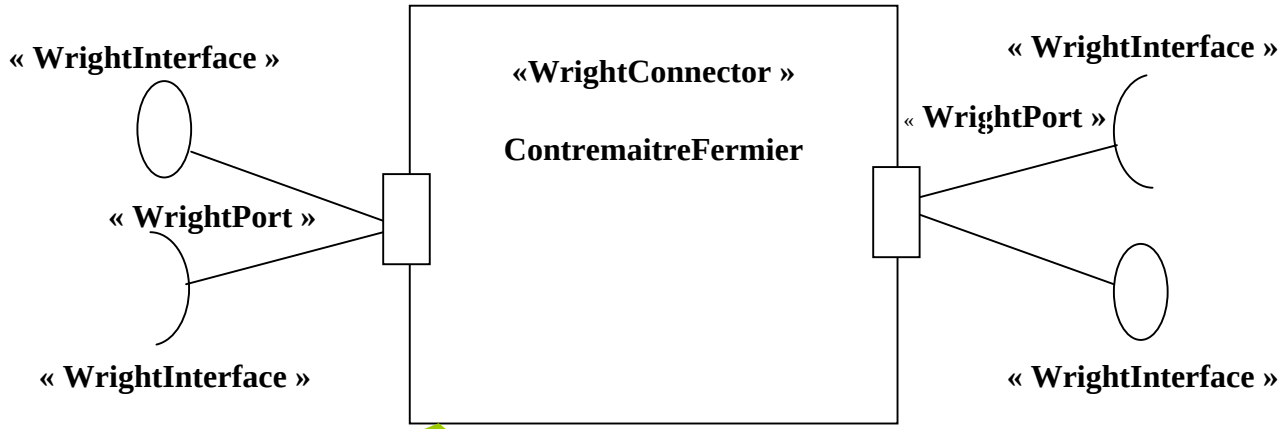
Computation = Contremaitre1.c?t -> $\overline{\text{Contremaitre1.d!v}}$ -> Computation

[] Contremaitre2.c?t -> $\overline{\text{Contremaitre2.d!v}}$ -> Computation

[] Contremaitre3.c?t -> $\overline{\text{Contremaitre3.d!v}}$ -> Computation

Exemple

Description en W-UML



Connector ContremaitreFermier

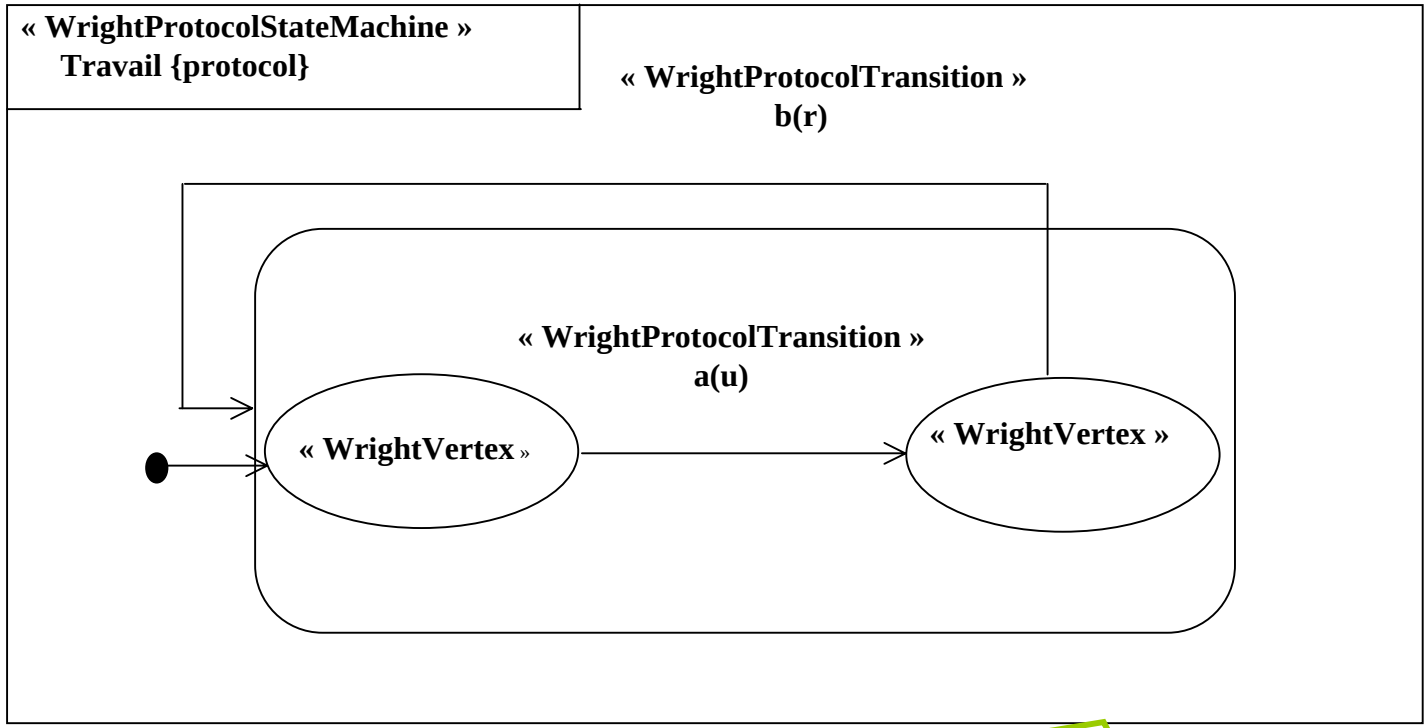
Role Contremaitre = $\overline{c!t}$ -> d?v -> Contremaitre

Role Fermier = c?t -> $\overline{d!v}$ -> Fermier

Glue = Contremaitre.c?t -> $\overline{\text{Fermier.c!t}}$ -> Fermier.d?v -> $\overline{\text{Contremaitre.d!v}}$->
Glue

Exemple

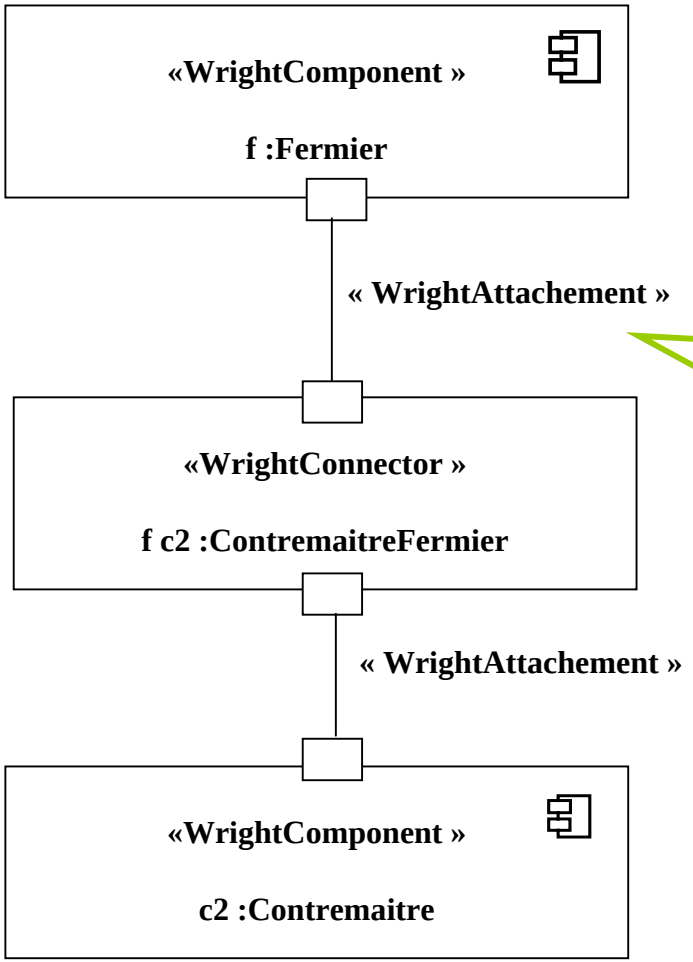
L'expression CSP associée au port Travail du composant Ouvrier



Component Ouvrier
Port Travail = $\overline{a!u}$ -> b?r -> Travail

Exemple

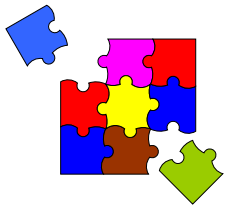
Attachement entre le Fermier et un seul Contremaître



Attachments

f.Contremaitre2 as fc2. Fermier

c2.Fermier as fc2. Contremaitre



Plan

-Problématique

-La démarche DVFAL

-L'ADL Wright comme langage pivot

-Traduction d'une architecture logicielle Wright vers Ada

-Vérification formelle d'architectures logicielles à base UML

-Le profil W-UML

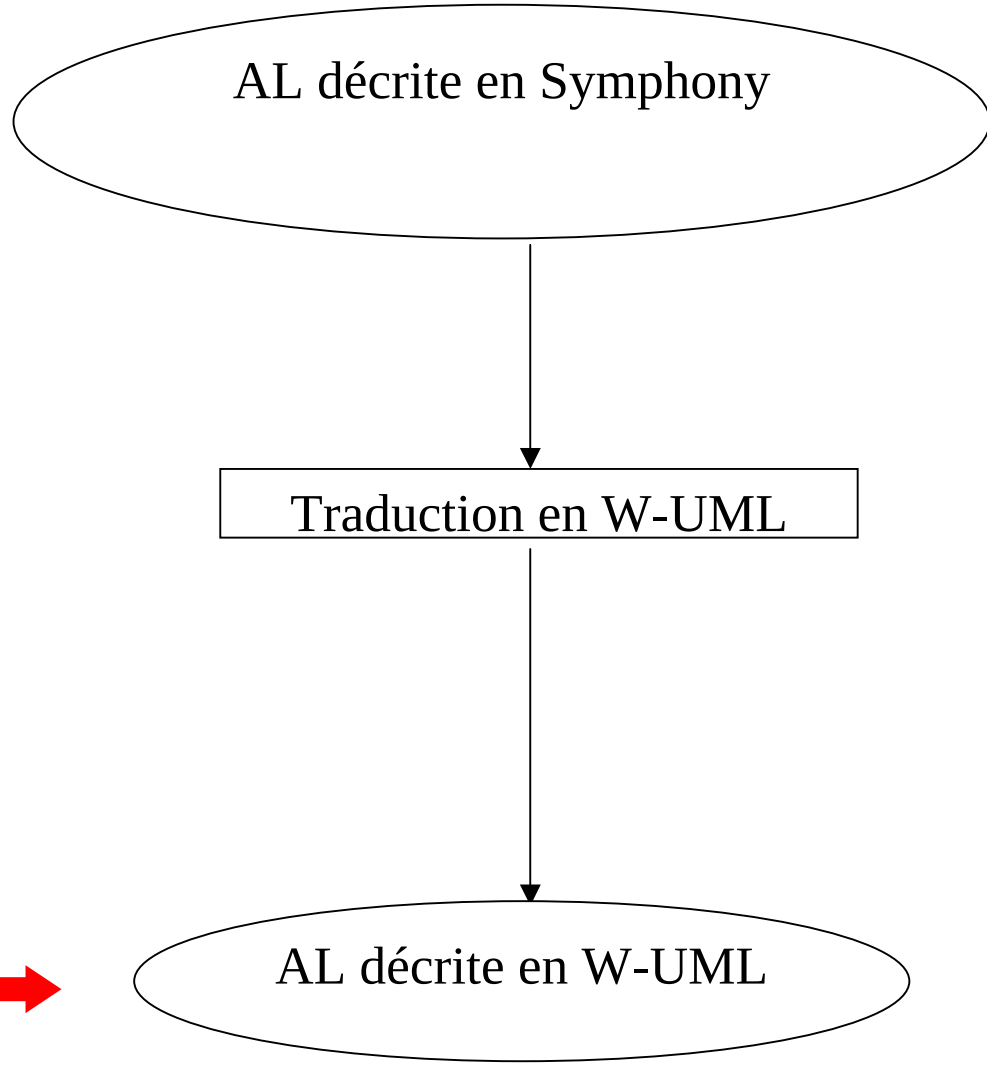
-Vérification formelle d'architectures logicielles Symphony

-Outils développés

-Conclusion & perspectives

Vérification formelle d'architectures logicielles Symphony

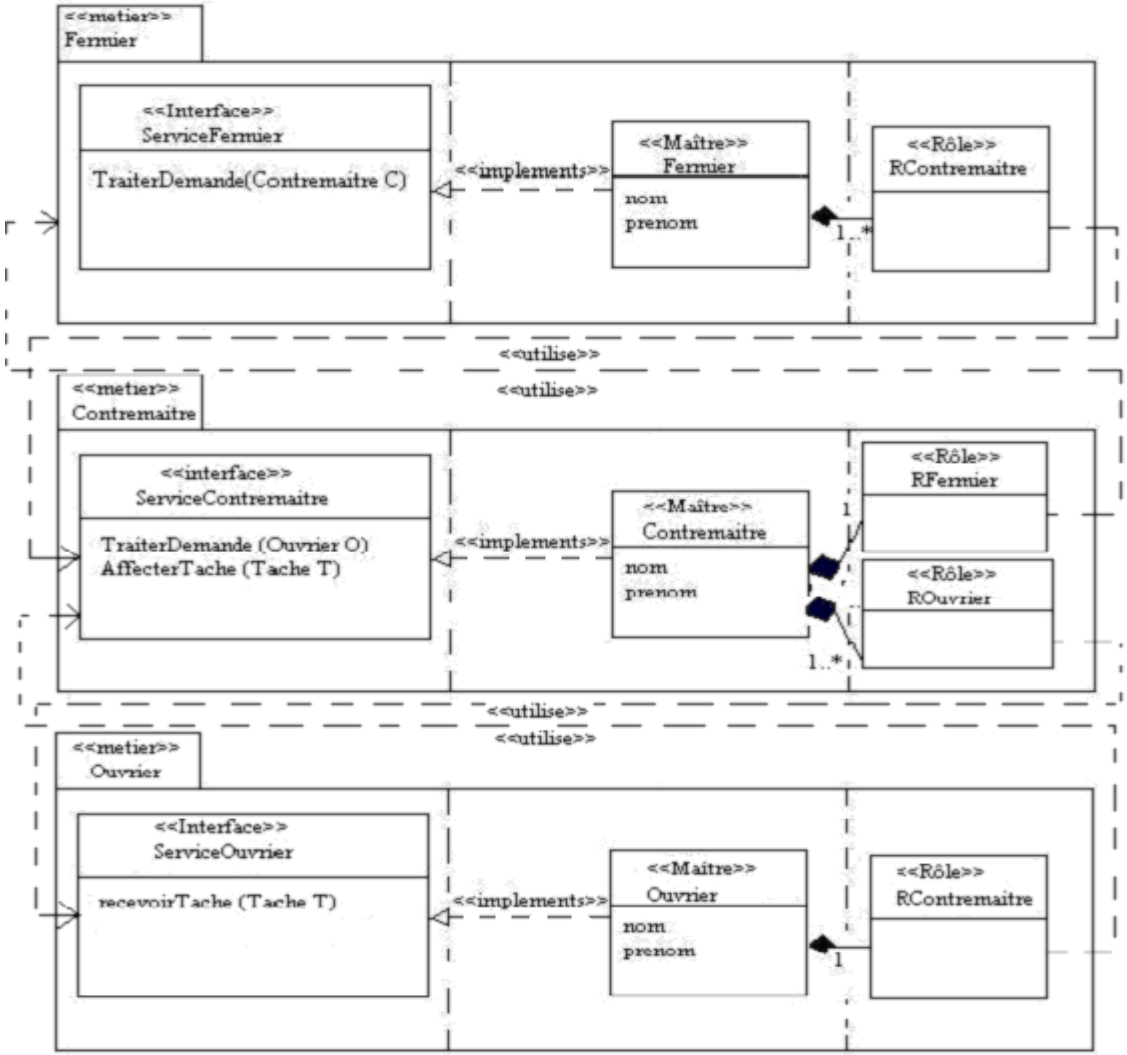
Composant métier : classe Maître, classe rôle, classe interface



<<WrightComponent>>,
<<WrightConnector>>,
<<WrightPort>>,
<<WrightAttachement>>,
<<WrightPort>>, …

Vérification formelle d'architectures logicielles Symphony

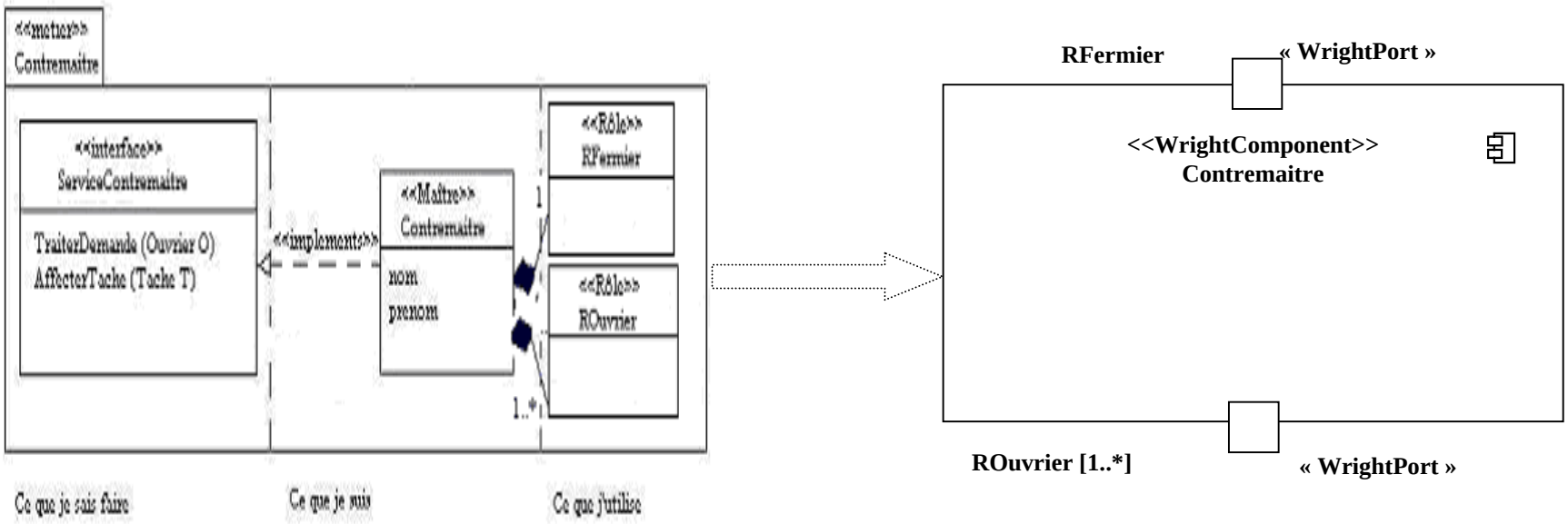
Modélisation de l'application «Ferme» en Symphony



Traduction du Composant métier Symphony

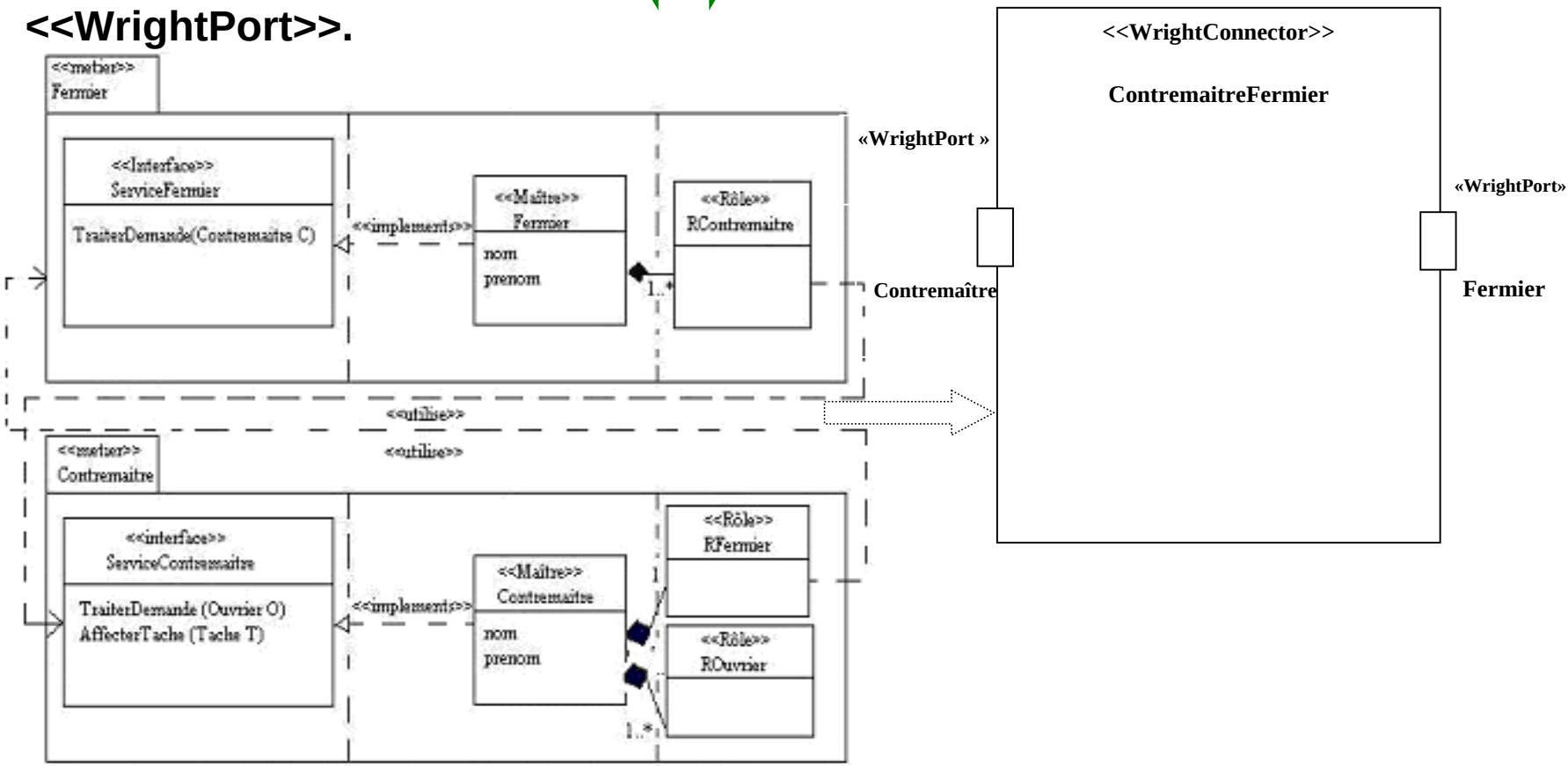
Composant métier Symphony ⟺ <<WrightComponent>>

Rôle du composant métier ⟺ <<WrightPort>>



Traduction de la relation Client-Fournisseur

relation Client-Fournisseur ⟺ <<WrightConnector>> ayant deux <<WrightPort>>.



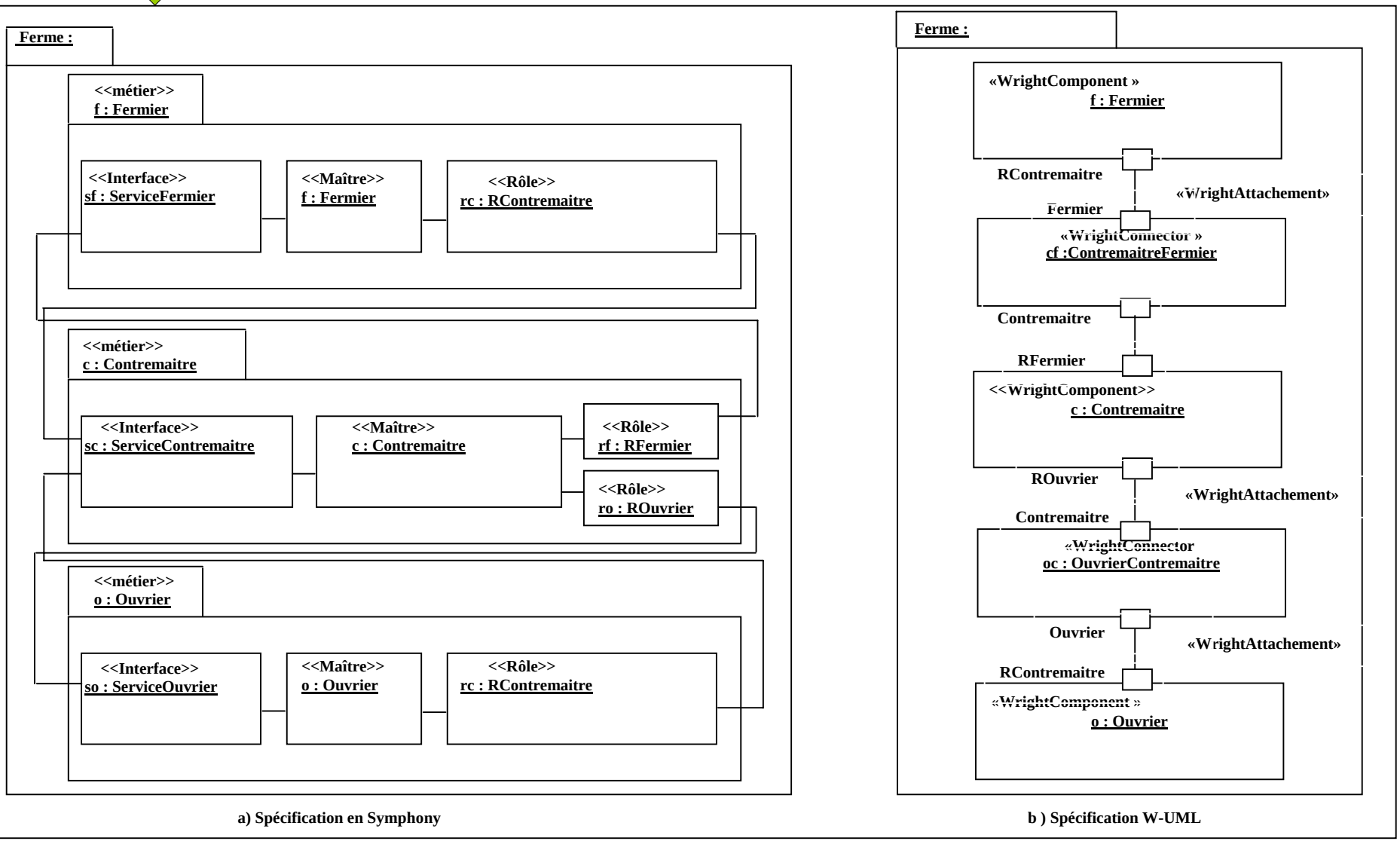
Traduction d'un diagramme d'instances

Un diagramme d'instances	⟺	configuration Wright
la classe <<Maître>>	⟺	<<WrightComponent>>
relation Client-Fournisseur	⟺	<<WrightConnector>>

De plus deux liens de type <<WrightAttachement>> sont créés entre une instance de type <<WrightComponent>> et une instance de type <<WrightConnector>>.

Vérification formelle d'architectures logicielles Symphony

Traduction Symphony vers W-UML



a) Spécification en Symphony

b) Spécification W-UML

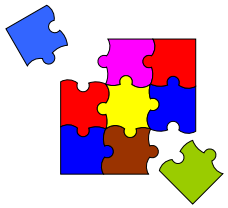
Traduction de l'aspect comportemental

Se heurte aux deux problèmes majeurs suivants :

- L'interface d'un composant W-UML combine via le concept port les services offerts et requis. Par contre, un composant métier Symphony les sépare nettement : classe <<Interface>> et classe <<Rôle>>.
- Contrairement au profil W-UML, la méthode Symphony ne propose pas un modèle de connecteur explicite.

⇩

La résolution de ces deux problèmes exige des connaissances spécifiques liées à l'application



Plan

-Problématique

-La démarche DVFAL

-L'ADL Wright comme langage pivot

-Traduction d'une architecture logicielle Wright vers Ada

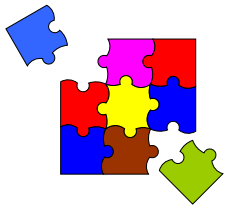
-Vérification formelle d'architectures logicielles à base UML

-Le profil W-UML

-Vérification formelle d'architectures logicielles Symphony

-Outils développés

-Conclusion & perspectives



Outils développés

- Vers un outil de transformation d'expressions CSP en machines à états de description de protocoles UML → W-UML

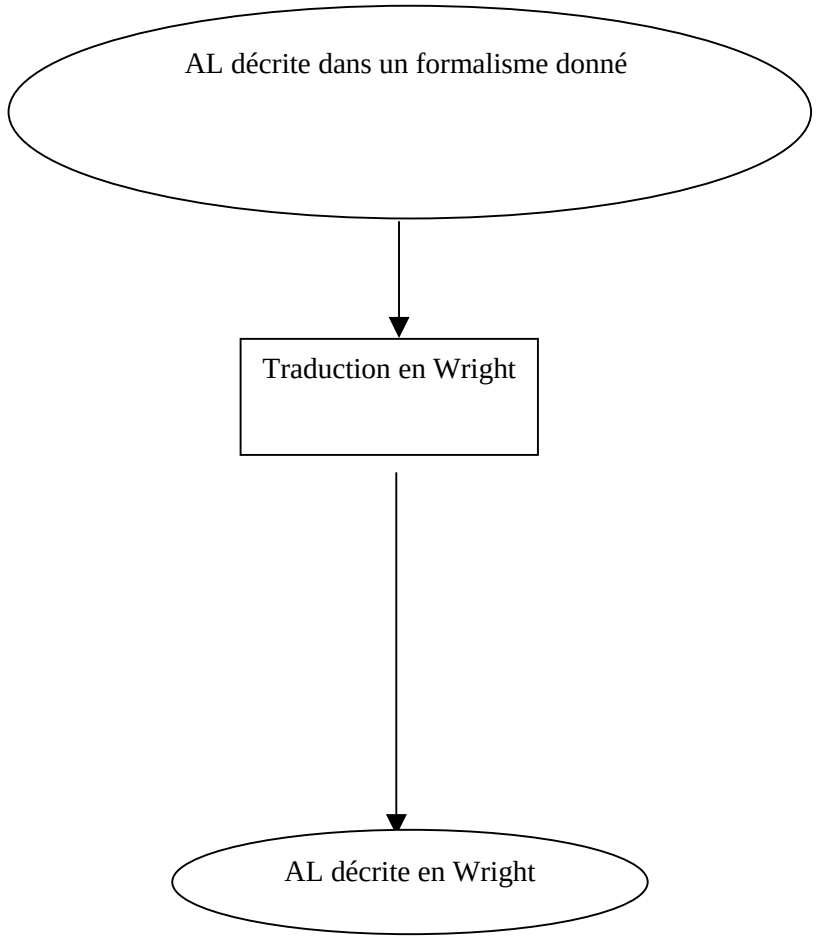
- Vers une implémentation du langage de contraintes de l'ADL Wright

Conclusion

Un cadre méthodologique permettant la vérification formelle de la correction d'architectures logicielles

La démarche **DVFAL supporte** Les formalismes de description d'architectures logicielles :

- Les ADL
- UML
- Symphony

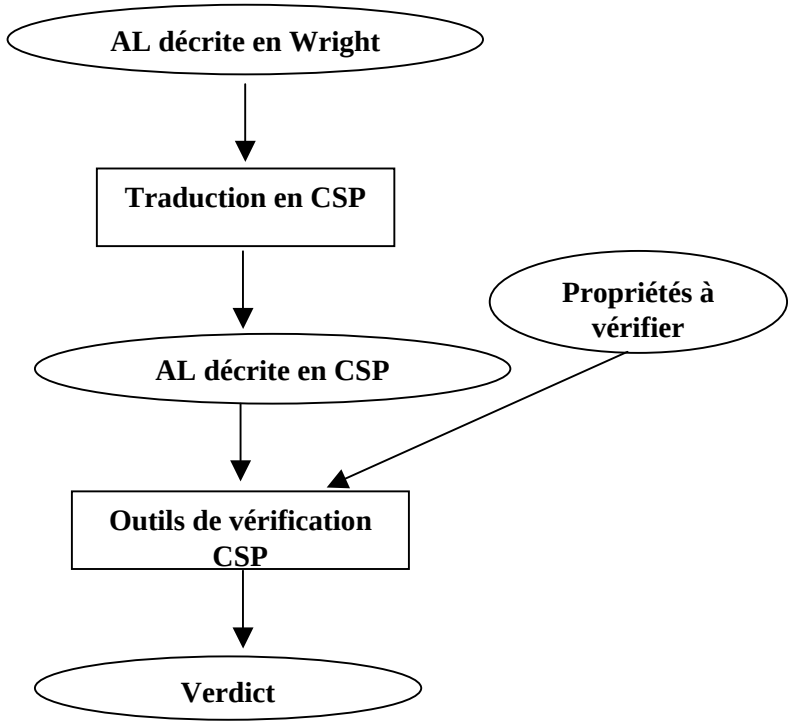


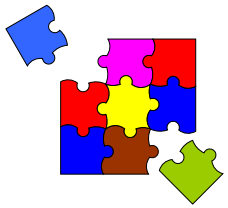
Conclusion

La démarche DVFAL réutilise :

- Wr2fdr
- Les outils supportant CSP tel que FDR

Propriétés standards et spécifiques dans le cadre CSP



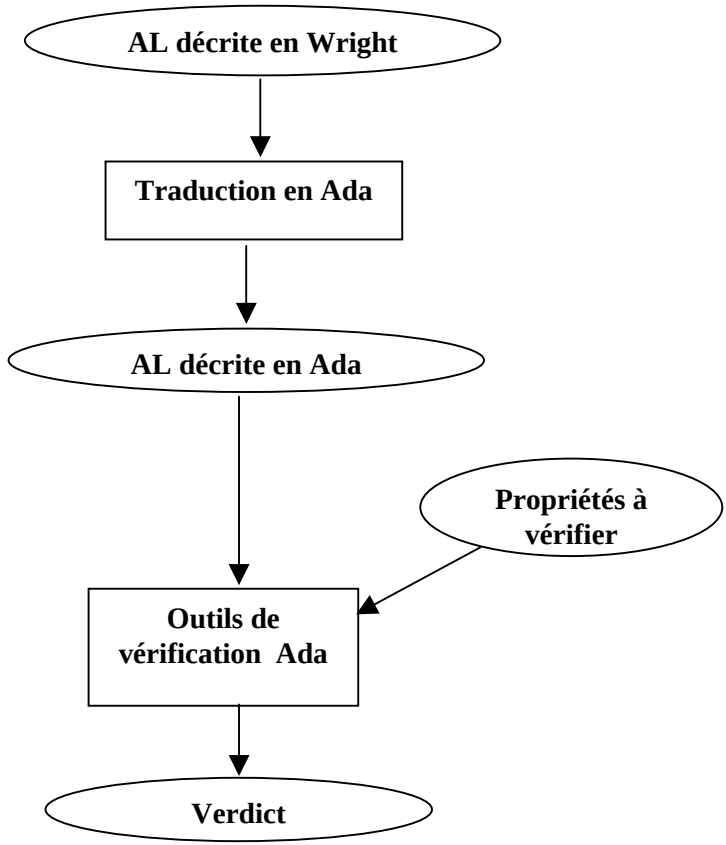


Conclusion

La démarche DVFAL propose et réutilise :

- Un traducteur de Wright vers Ada
- Les outils supportant Ada tels que FLAVERS, SPIN, SMV et INCA

Propriétés spécifiques

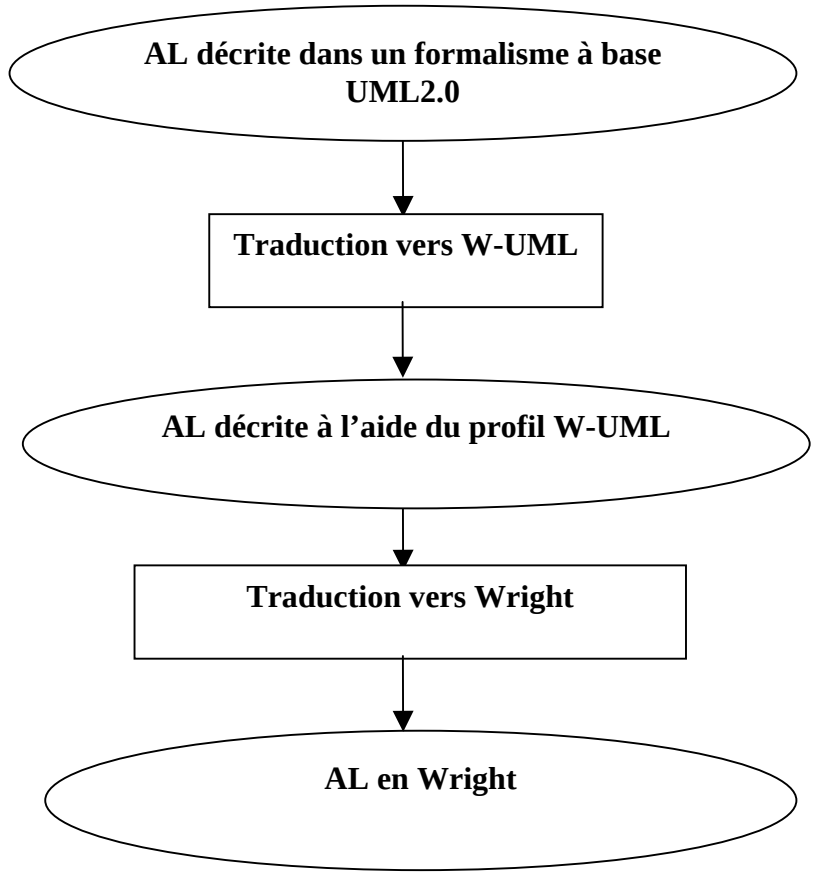


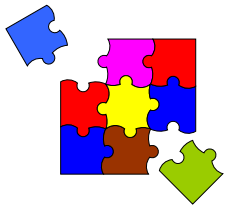
Conclusion

La démarche DVFAL propose :

– un langage intermédiaire W-UML

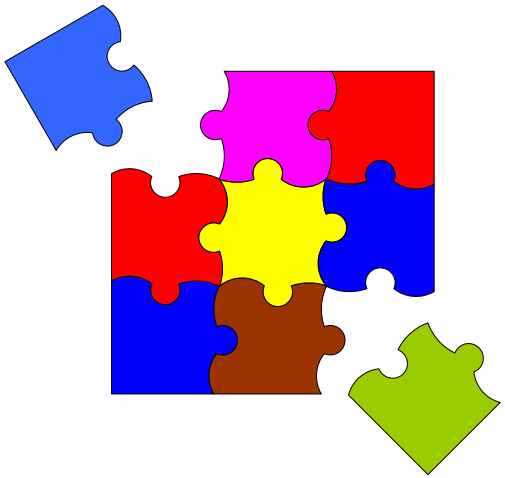
Modèle pivot pour des spécifications semi-formelles





Perspectives

- **Améliorations significatives à l'outil Wr2fdr**
  - Sémantique statique de Wright
  - Prise en compte des événements initialisés et observés
  - Prise en compte de la notion d'interface
- **Outillage de la démarche DVFAL**
  - Automatisation de l'approche de traduction de Wright vers Ada
    ➡ Approche classique
  - Automatisation des schémas de transformation du Profil W-UML vers Wright ➡ IDM
  - Automatisation des schémas de transformation de Symphony vers le profil W-UML ➡ IDM
- **Ouverture de la démarche DVFAL**
  - Sur la méthode formelle avec preuve B ➡ propriétés orientées état et Cohérence de style (style)



Merci