**Université de Monastir**
**Institut Supérieur d'Informatique et de Mathématiques**

# Cours: Approche à base de composants et concurrence

**Filière: MR-GL2**

# Chapitre 3: Le modèle de composants Wright

Réalisé par:

Dr. Sakka Rouis Taoufik

# I. Introduction

- L'ADL Wright (Allen, 1997) offre des concepts structuraux et comportementaux permettant de décrire **les architectures logicielles abstraites**.
- Il s'agit d'un ADL formel basé sur CSP (Communicating Sequential Processes) (Hoare, 1985).
- l'ADL Wright, joue un rôle déterminant dans la vérification des propriétés comportementales (de concurrence) relatives aux interactions inter et intra-composants.

# II. Concepts structuraux

- Les concepts structuraux de Wright sont assez similaires à ceux d'Acme.

## 1) Composant

- Un composant en Wright est une unité de traitement abstraite localisée et indépendante.

- La description d'un composant contient deux parties :
  - **La partie interface** composée d'un ensemble de ports (Port) qui fournissent les points d'interaction entre le composant et son environnement.
  - **La partie calcul (computation)**, quant à elle, consiste à décrire le comportement du composant en indiquant comment celui-ci utilise les ports.

# II. Concepts structuraux

- Chaque port est équipé d'une description formelle en CSP (Hoare, 1985) spécifiant son comportement par rapport à l'environnement ➔ Un port permet de décrit **le comportements partiel d'un composant.**

- La partie computation est équipée d'une description formelle en CSP qui décrit **le comportement global en** composant concerné.

**Exemple**

**Component** Filter
**Port** Input = //CSP Expression
**Port** Output = //CSP Expression
**Computation** = //CSP Expression

# II. Concepts structuraux

## 2) Connecteur

- Un connecteur en Wright représente une interaction explicite et abstraite entre une collection de composants.

- Le connecteur contient deux parties importantes :
  - une interface constituée de points d'interactions appelés rôles (Role)
  - une partie qui représente la spécification d'assemblages (Glue).
  - Le rôle indique comment se comporte un composant qui participe à l'interaction. La glue spécifie les règles d'assemblage entre un ensemble de composants pour former une interaction. Les comportements des rôles et de la glue d'un connecteur Wright sont décrits en CSP.

# II. Concepts structuraux

- Le rôle indique comment se comporte un composant qui participe à l'interaction.
- La glue spécifie les règles d'assemblage entre un ensemble de composants pour former une interaction.
- Les comportements des rôles et de la glue d'un connecteur Wright sont décrits en CSP.

**Exemple**

**Connector** Pipe
**Role** Source = //CSP Expression
**Role** Sink = //CSP Expression
**Glue** = //CSP Expression

# II. Concepts structuraux

## 3) Configuration

- La configuration Wright permet de décrire l'architecture d'un système en regroupant des instances de composant et des instances de connecteur.

- La description d'une configuration est composée de trois clauses:
  - la déclaration des composants et des connecteurs utilisés dans l'architecture,
  - la déclaration des instances de composants et de connecteurs,
  - la description des liens reliant les instances de composants avec les instances de connecteurs.

# II. Concepts structuraux

**Exemple**

**Configuration** System
**Component** Filter
**Port** Input = //CSP Expression
**Port** Output =//CSP Expression
**Computation** =//CSP Expression
**Connector** Pipe
**Pole** Source = //CSP Expression
**Role** Sink = //CSP Expression
**Glue** = //CSP Expression
**Instances**
Split, Merger: Filter
P1, P2: Pipe
**Attachments**
Split.Input **as** P1. Source
Merger.Output **as** P1. Sink
Merger.Input **as** P2. Source
Split.Output **as** P2. Sink
**End Configuration**

# II. Concepts structuraux

## 4) Composition et Style

- L'ADL Wright supporte la composition hiérarchique. Ainsi, un composant peut être lui-même composé d'un ensemble de composants. Il en va de même pour un connecteur.

- A l'instar d'Acme, Wright permet de décrire des styles architecturaux. Les contraintes associées aux styles sont décrites à l'aide d'un langage de contraintes simple spécifique à Wright.

# II. Concepts structuraux

**Exemple**

**Style** PipeFilter
**Component** Filter
  **Port Input** = //CSP Expression
  **Port Output** = //CSP Expression
  **Computation** = //CSP Expression
**Connector** Pipe
 **Role** Source = //CSP Expression
 **Role** Sink = //CSP Expression
 **Glue** = //CSP Expression

**Constraints**
 ∃ **c : Connectors; r : Roles(c) • Type(r) = Sink**
 ∀ **c : Components • Type (c) = Filter**
**End Style**

**Configuration** System **Style** PipeFilter
 **Component** Filter2
  **Port** Input = //CSP Expression
  **Port** left =//CSP Expression
  **Port** right =//CSP Expression
  **Computation** =//CSP Expression
 **Instances**
  Split, Merger: Filter2
  Upper: Filter
  P1, P2, P3, P4: Pipe
 **Attachments**
**…**
**End Configuration**

# III. Concepts comportementaux

- Contrairement à Acme, Wright permet de décrire les aspects **comportementaux** d'une architecture logicielle **abstraite**.
- Pour y parvenir, Wright utilise un sous-ensemble de CSP de Hoare.
- Sachant que CSP est un modèle mathématique qui a pour but de formaliser la conception et le comportement des systèmes.
- CSP est basé sur de solides fondements mathématiques qui permettent une analyse rigoureuse.

# III. Concepts comportementaux

## 1) Les événements

- Dans le modèle CSP, tout est représenté par des événements. Un événement correspond à un moment ou une action qui présente un intérêt.
- CSP ne fait pas la distinction entre les événements initialisés et observés. Mais, CSP pour Wright le fait :
  - Un événement initialisé s'écrit sous la forme (ē ou _e).
  - Un événement observé est noté (e).
- De plus, les événements peuvent transmettre des données : (e?x) et (e!x), représentent respectivement les données d'entrée et de sortie.
- CSP définit un événement particulier noté (√), qui indique la terminaison de l'exécution avec succès.

# III. Concepts comportementaux

## 1) Les procéssus

- Pour définir un comportement, il faut pouvoir combiner les événements.
- Un processus correspond à la modélisation du comportement d'un objet par la combinaison d'événements avec d'autres processus simples.
- Les principaux opérateurs fournis par CSP sont :

- L'opérateur de préfixage noté ( ->) : Le séquencement ou le préfixage est la façon la plus simple de combiner des événements. Un processus qui s'engage dans un événement e, puis se comporte comme le processus P, est noté (e->P ).

# III. Concepts comportementaux

- La récursion : en utilisant la possibilité de nommer un processus, il devient possible de décrire les comportements répétitifs très facilement. Nous décrivons par exemple le processus qui ne s'engage que dans l'événement e et qui ne s'arrête jamais par : P=e->P.

- L'alphabet : L'alphabet fait référence à un processus P et est noté (αP). L'alphabet d'un processus est l'ensemble des événements sur lequel le processus a une influence.

# III. Concepts comportementaux

- L'opérateur de choix externe ou déterministe noté ( □ ou [ ]) : Si nous avons le processus e -> P [ ] u ->Q et que l'environnement s'engage dans l'événement u, alors le processus s'engagera dans cet événement et se comportera comme le processus Q.

- L'opérateur de choix interne ou non déterministe noté ( Π ou |~|): A l'inverse du choix déterministe, c'est le processus qui choisit de façon non déterministe le comportement à choisir parmi plusieurs.

  Cette fois le processus ē ->P |~| ū ->Q va choisir entre initialiser l'événement e et continuer comme P ou initialiser u et continuer comme Q. **Il décide lui-même** de ce choix sans se préoccuper de l'environnement.

# III. Concepts comportementaux

- L'opérateur de composition parallèle noté ( | | ): Les processus parallèles peuvent s'engager d'une manière synchrone dans des événements appartenant à l'intersection de leurs alphabets.

- **Remarque :** En complément de la notation standard de CSP, Wright introduit une nouvelle notation notée (TICK ou §). Cette notation désigne le processus de terminaison avec succès, ce qui veut dire que le processus s'est engagé dans un événement succès √ et s'est arrêté. Formellement, § = √ -> STOP (En CSP il est généralement noté «SKIP»).

# IV. Propriétés standard de Wright

- Les auteurs de l'ADL Wright proposent onze propriétés standard liées à la cohérence et complétude des architectures logicielles décrites en Wright (Allen, 1977). Ces propriétés sont présentées informellement comme suit :

**Propriété 1: Cohérence d'un composant**

- La spécification de chaque port doit être une projection de la partie calcul (computation).
  - Un Port est une projection d'un Composant si ce dernier agit de la même manière que le Port quand nous ignorons tous les événements n'appartenant pas à l'alphabet de ce Port.
  - Intuitivement, la propriété 1 indique que le composant ne se soucie pas des événements non couverts par les ports.

# IV. Propriétés standard de Wright

**Exemple**

**Component** Double
  **Port** In = read -> In [ ] close -> TICK [ ] fail -> TICK
  **Port** Out = _write -> Out |~| _close -> TICK
  **Computation** = In.read -> _Out.write -> Computation [ ]
   In.close -> _Out.close -> TICK [ ] In.fail-> TICK

- Si nous ignorons tous les événements qui n'appartiennent pas à l'alphabet du Port *Input*, nous obtenons :

***Computation = In.read -> Computation [ ] In.close -> TICK [] In.fail -> TICK***

➔ Le port *In* est bien une projection du computation.

- De la même manière, il est facile de montrer que le port *Out* *n'est* pas une projection de la partie computation.

# IV. Propriétés standard de Wright

## Propriété 2: Connecteur sans interblocage

- La glue d'un connecteur interagissant avec les rôles doit être **sans interblocage**.

- Une autre catégorie d'incohérence est détectable comme une situation d'interblocage, lorsque la spécification d'un rôle est elle-même incohérente.

- La description du connecteur doit vérifier que la coordination des rôles par la glue est cohérente avec le comportement attendu des composants.

# IV. Propriétés standard de Wright

## Propriété 3: Rôle sans interblocage

- Chaque rôle d'un connecteur doit être **sans interblocage**.
- Une autre catégorie d'incohérence est détectable comme une situation de blocage, lorsque la spécification d'un rôle est elle-même incohérente.
- Dans une spécification d'un rôle complexe, il peut y avoir des erreurs qui conduisent à une situation dans laquelle aucun événement n'est possible pour le participant, même si la glu était prête à prendre n'importe quel événement.

# IV. Propriétés standard de Wright

## Propriété 4: Un seul initialiseur

- Dans une spécification de connecteur, tout événement doit être initialisé par un rôle ou une glue.

- Tous les autres processus doivent soit l'observer, soit l'oublier (grâce à leur alphabet).

## Propriété 5: L'engagement d'initialisation

- Si un processus initialise un événement, il doit s'engager à cet événement sans être influencé par l'environnement.

- Cette propriété garantit que les concepts d'initialisation et d'observation des événements sont utilisés correctement

# IV. Propriétés standard de Wright

## Propriété 6: Paramètres de substitution

- Une déclaration d'instance définissant un type doit résulter de ce type de validation après avoir remplacé tous les paramètres formels manquants.
- Pour le paramètre numérique, nous devons nous assurer que les paramètres donnés entrent dans les limites dans la description du type.

## Propriété 7: Test sur les valeurs d'intervalle

- Un paramètre numérique ne doit pas être inférieur à la limite inférieure (si elle est déclarée) et ne doit pas être supérieur à la limite supérieure (si elle est déclarée).

# IV. Propriétés standard de Wright

## Propriété 8: Compatibilité port / rôle

- Tout port attaché à un rôle doit toujours poursuivre son protocole dans une direction que le rôle peut avoir. Au niveau des liens, la question qui se pose est: Quels ports peuvent être utilisés pour ce rôle?

## Propriété 9: Contraintes de style

- Les prédicats de style doivent être vrais pour une configuration déclarée dans ce style.

➔ Les contraintes de style doivent être cohérentes.

**Exemple:**

$\forall c : Component; p : Ports(c) \bullet Type(p) = DataOutput$

$\exists c : Component; p : Ports(c) \bullet Type(p) = DataInput$

- Ces deux contraintes sont incohérentes, donc le style contenant ces deux contraintes est incohérent.

# IV. Propriétés standard de Wright

## Propriété 10: Cohérence de style

- Au moins une configuration doit satisfaire les contraintes de style.

**Exemple** La configuration du Diapo 10 ne satisfait pas les contraintes spécifiées au niveau style.

# IV. Propriétés standard de Wright

## Propriété 11: Exhaustivité des liens

- Afin d'assurée la complétude, on doit assurer que chaque port et chaque rôle non attaché dans la configuration doit être compatible avec le processus de terminaison avec succès (noté §).
  - Au niveau des liens, si un lien est omis alors un composant va dépendre des évènements qui ne vont jamais avoir lieu, ou une interaction va échouer car il manque un participant,
  - D'autre part, il existe des ports de composants qui n'ont pas besoin d'être attachés et il y a des interactions qui peuvent continuer même si un participant manque.

➔Pour ces deux raisons, il n'est pas suffisant de contrôler que tous les ports et rôles soient bien attachés.

# V. Les outils Wr2fdr et FDR

- Les auteurs de Wright proposent un outil appelé Wr2fdr (ABLE, 2005) permettant d'automatiser les quatre propriétés (1, 2, 3 et 8) décrites précédemment.

- Pour y parvenir, l'outil Wr2fdr traduit une spécification Wright en une spécification CSP dotée des relations de raffinement à vérifier.

- La spécification CSP générée par l'outil Wr2fdr est soumise à l'outil FDR (Failure-Divergence Refinement) (FDR2, 2012).

# V. Les outils Wr2fdr et FDR

## 1) L'outil Wr2fdr

- Wr2fdr est un outil développé par l'université de Carnegie Mellon (ABLE, 2005).

- Il permet de traduire une spécification Wright vers une description CSP acceptable par le model-checker FDR2 (FDR2, 2012).

- L'outil Wr2fdr est censé implémenter les propriétés 1, 2, 3 et 8 présentées précédemment.

# V. Les outils Wr2fdr et FDR

## 2) L'outil FDR

- FDR permet de vérifier de nombreuses propriétés sur des systèmes d'états finis.
- FDR s'appuie sur la technique du model-checking (Schnoebelen, 1999).
- Celle-ci effectue la vérification d'un modèle d'un système par rapport aux propriétés qui sont attendues sur ce modèle.
- Cette vérification est entièrement automatisée et consiste à explorer tout l'espace d'états.

# V. Exercices d'application

## Exercice 1: Corriger la configuration Wright suivante:

**Configuration** Client_Serveur
**Component** Client
  **Port** port_Client= requete -> reponse -> port_Client |~| **TICK**
  **Computation**= _port_Client.requete ->  port_Client.reponse -> computation

**Component** Serveur
  **Port** port_Serveur = requete -> _reponse -> port_Serveur| ~| **TICK**
  **Computation**= _traitement_interne-> port_Serveur.requete -> _ port_Serveur.reponse -> computation
[] **TICK**

**Connector** Lien_CS
  **Role** Appelant=  requete -> reponse -> Appelant |~| **TICK**
  **Role** Appele= requete -> reponse -> Appele [] **TICK**
  **Glue** = Appelant.requete -> _Appele.requete -> glue  [] Appele.reponse -> _Appelant.reponse -> glue
 **Instances**
  client1: **Component** Client
  serveur1: **Component** Serveur
  appel_cs: **Connector** Lien_CS
 **Attachments**
  appel_cs-Appelant **As**  client1-port_Client
  serveur1-port_Serveur **As** appel_cs-Appelant
**End Configuration**

# V. Exercices d'application

**Exercice 2:**

Proposer une description Wright/CSP pour le système ATM présenté dans le Diapositive 26 du chapitre 2.

**Exercice 3:**

Proposer une description Wright/CSP pour le système Compression de proxy présenté dans le Diapositive 32 du chapitre 2.

**Exercice 4:**

Déduire les règles de traduction d'UML2.0/PoSM vers Wright/CSP