CENTRALE
L Y O N

# BE n°1-2

**Techniques de simulation à base de multitâche**

- Exemple de simulation de multiprocesseur
- TD de simulation de fonctionnement d'une architecture Client – Serveur par échange de messages

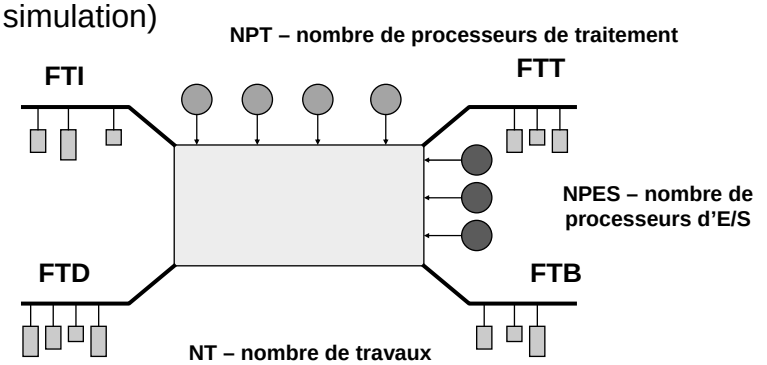
BTD/GL/BE 1

CENTRALE
L Y O N

Bertrand DAVID : Génie Logiciel

## BE n°1 : Simulateur d'un multiprocesseur

- Principes de simulation
- Expression du parallélisme en langage ADA : Tâches
- Etudier des architectures à base de tâches
- Etudier comment gérer le temps virtuel (de simulation)




BTD/GL/BE 2

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Démarche

- Présentation du problème : principes retenus pour la simulation du multiprocesseur
- Mettre en place un cas (choix du nombre de travaux NT, de processeurs de traitement NPT et de processeurs d'E/S NPES
- Faire à la main la simulation et bâtir des graphiques de bilan manuellement :
  - Processeurs / temps: occupation des processeurs
  - Travaux / temps: traitement des travaux
- Architectures du simulateur : représentation graphique
- Architectures du simulateur : description textuelle (en ADA)
- Prise en compte de la gestion du temps

BTD/GL/BE 3

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Architecture du multiprocesseur

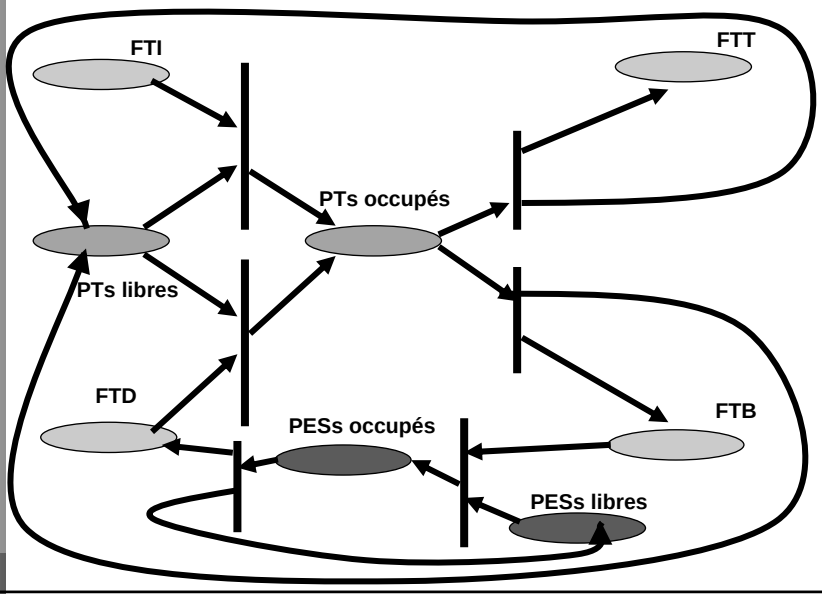
- Structure:
  - Mémoire commune partagée
  - NPT processeurs identiques partageant la mémoire
  - NPES processeurs spécialisés en Entrées-Sorties
  - 1 file de travaux initialisés FTI
  - 1 file de travaux terminés FTT
- Fonctionnement :
  - Un processeur traite un travail tant qu'il peut:
    PAS DE PREEMPTION
  - Les entrées-sorties sont à la charge du processeur d'E/S.
  - La suite du travail peut être effectuée par un autre processeur.
  - Le principe de fonctionnement peut être décrit par le réseau de Petri ci-joint.

BTD/GL/BE 4

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Modélisation par réseau de PETRI




BTD/GL/BE

5

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Multiprocesseur

- Ressources critiques : files d'attente des travaux
  - FTI: file de travaux initialisés
  - FTB: file des travaux bloqués (sur E/S)
  - FTD: file des travaux débloqués
  - FTT: file des travaux terminés
- Simulation d'un multiprocesseur sur un mono ou multi-processseur :
  raisonner de façon abstraite en considérant d'avoir le nombre suffisant de processeurs

BTD/GL/BE

6

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Modélisation du travail

- Une suite alternée de traitements et d'E/S
- Un vecteur donne successivement les temps des séquences

- Travail : Item courant; NB d'Items

```
EXEC  10
E/S    5
EXEC  12
E/S    2
EXEC  20
E/S    4
EXEC   4
```

- Bilans à fournir :
  - Occupation des processeurs/ temps
  - Traitement des travaux par le processeurs/ temps

BTD/GL/BE

7

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Approche de modélisation par tâches ADA

- Rappels du cours :
  - tâches
  - Synchronisation : rendez-vous
  - protection de ressources sensibles : tampon actif
  - select et attente limitée

BTD/GL/BE

8

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Propositions de l'architecture du simulateur à base de tâches

- Processeur modélisé sous forme de tâche ADA ?
- Travail modélisé sous forme de tâche ADA ?
- File modélisée sous forme de tâche ADA ?

- Synchronisations:
  - Qui sollicite le rendez-vous ?
  - Qui propose et accepte le rendez-vous ?
  - Quelles ressources critiques à protéger ?
  - Comment protéger ?
  - Exclusivité d'accès

BTD/GL/BE 9

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Protection par le Maître

- **Métaphore du banquet :**

  **Comment protéger la nourriture et les boissons contre les bousculades**

BTD/GL/BE 10

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Explications du modèle

- Maître accède aux files
- Esclaves sollicitent le rendez-vous avec le maître pour
  - PRENDRE un travail à faire
  - RENDRE un travail fait
- Esclaves : processeurs de traitement et processeurs d'E/S

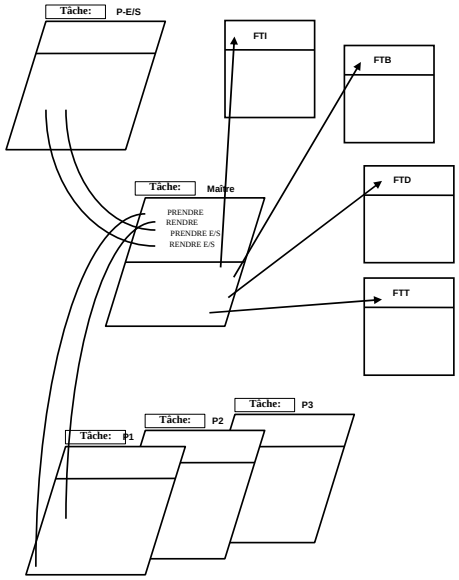
BTD/GL/BE

11

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Architecture n° 1 : approche centralisée maître-esclaves




BTD/GL/BE

12

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Approche distribuée

- **Comment éviter la fatigue du Maître ?**
- **Comment se protéger soi-même ?**

BTD/GL/BE

13

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Explications du modèle

- Chacun est maître à son tour
- Pour accéder aux files il faut être maître (mais un seul maître à la fois) :
- Tâches : processeurs et files
- Qui propose et accepte (les files)
- Qui sollicite (les processeurs)

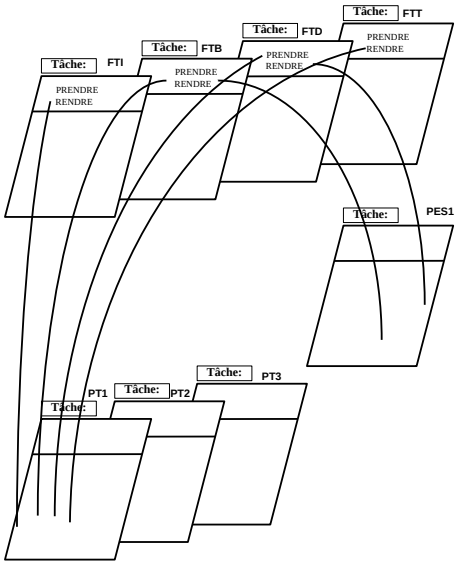
BTD/GL/BE

14

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Architecture n° 2 : approche décentralisée




BTD/GL/BE

15

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Gestion du temps

**Trouver le modèle du temps le plus juste : attention on a**

- **le temps réel** d'exécution sur le simulateur qui nous n'intéresse pas particulièrement,
- **le temps virtuel de chaque processeur** qui nous intéresse

Comment faire:

- approche non tâche: une boucle sur les processeurs faisant avancer le temps virtuel d'une unité de temps à chaque tour.
- approche tâche :
  Rendez-vous avec une tâche Horloge

- Travail : Item courant; NB d'Items

EXEC 10
E/S 5
EXEC 12
E/S 2
EXEC 20

| Item courant | Nb d'Items | | |
|---|---|---|---|
| Nature | Durée | N°Processeur | Temps Début |
| **EXE** | **10** | | |
| **E/S** | **5** | | |
| **EXE** | **12** | | |
| **E/S** | **2** | | |
| **EXE** | **20** | | |

BTD/GL/BE

16

CENTRALE
L Y O N

Bertrand DAVID : Génie Logiciel

## Deux solutions

==> **modèle à progression indépendante de chaque processeur :** le processeur avance directement du temps demandé par le travail.
Approche très séduisante, mais dans ce cas on trouve dans les files
(FTB, FTD, FTT) des travaux dans le désordre. On peut essayer d'ordonner, mais cela ne marche pas.

==> **modèle avec l'horloge virtuelle** : chaque processeur doit recevoir un top d'horloge pour pouvoir progresser dans son exécution. Il faut donc une tâche HORLOGE et choisir un bon modèle de rendez-vous: qui sollicite, qui propose et accepte. Attention à la famine et aux blocages.

- Etudier la bonne terminaison de la simulation (sur nombre de travaux terminés).

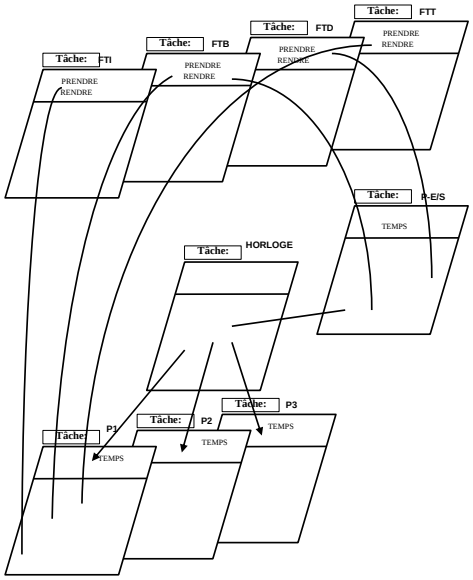
BTD/GL/BE

17

CENTRALE
L Y O N

Bertrand DAVID : Génie Logiciel

## Architecture n° 3: approche décentralisée avec gestion du temps




BTD/GL/BE

18

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Ecriture "pseudo-ADA" des tâches HORLOGE et PROCESSEUR

```
task HORLOGE
    task body is
    LOOP
      FOR I in 1..N DO
                  P(I).Temps
      END FOR

      PES.Temps
    END LOOP
    end horloge

task type PROCESSEUR
    entry Temps
    end
```

```
task body is
LOOP        SELECT
                        FTD.Prendre (T)
                        Etat:= Travail
            ELSE
                        Etat:= FTDVide
            END SELECT
IF Etat= FTDVide then
            SELECT
                        FTI.Prendre (T)
                        Etat:= Travail
            ELSE
                        Etat:=FTIVide
            END SELECT
            END IF;

IF (ETAT= FTIVide)      -- AND (ETAT=FTDVide)
                        THEN        Accept.Temps
            END IF
IF Etat = Travail THEN
                                    FOR I in 1..UT DO
                                    Accept.Temps
                                    END FOR
IF Etat = Travail-fini THEN         FTT.Rendre
                        ELSE        FTB.Rendre
                        END IF
            END IF
END LOOP
end processeur
```

BTD/GL/BE

19

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## BE à faire

- **Simulation d'une application coopérative fonctionnant sur l'architecture Client-Serveur à base de messages :**
  - **Quoi simuler ?**
    - **Choisir des simplifications**
    - **Trouver des éléments significatifs**
  - **Comment simuler ?**
    - **A l'aide de threads Java**
    - **Quelle(s) architecture(s) ?**
    - **Comment montrer les résultats ?**

BTD/GL/BE

20

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

# Client - Serveur

**Définitions :**

- Un service est un ensemble de fonctions mises à la disposition des programmes d'application et généralement partageable entre ces programmes.
- Très souvent, un service logiciel gère des ressources.
- La notion de service n'est pas nouvelle, ce qui est nouveau c'est la possibilité de dissocier la localisation de l'application appelante et la localisation d'une partie ou de la totalité des fonctions de service appelé.
- On dira que l'on est en mode client / serveur lorsque le service appelé incorpore un dispositif logiciel qui lui permet d'être exécuté localement ou à distance et de façon transparente pour l'application qui a recours à ce service.
- On observe que le service comporte en fait deux parties : une partie proche de l'application appelante nommée demandeur et une partie éloignée chargée de gérer une ressource appelée gestionnaire de ressource.
- Par extension, on appelle client la machine qui contient l'application appelante et la partie demandeur, la machine qui contient la portion déportée du service est nommée serveur.

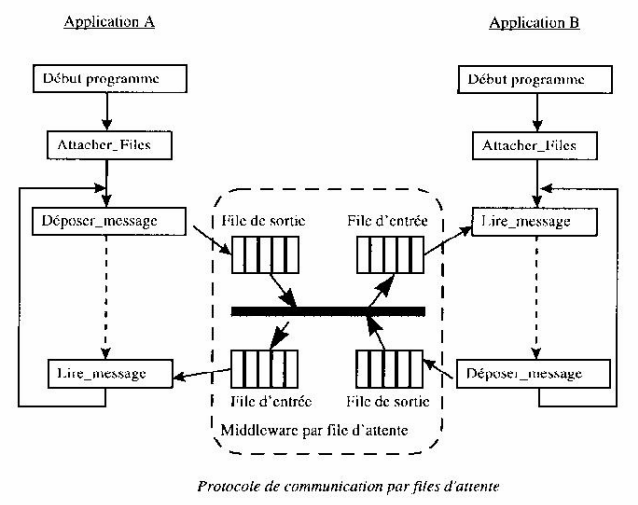
BTD/GL/BE

21

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

# Client - Serveur




*Protocole de communication par files d'attente*

BTD/GL/BE

22

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Client – Serveur : Configurations logicielles

**Trois schémas sont envisageables :**

- le cas simple dans lequel une application fait appel à un seul serveur (base de données par exemple),
- le cas où l'application fait appel à plusieurs services qui peuvent être localisés ou non sur le même serveur (accès aux fichiers et à la base de données). On parle de configuration logique en parallèle,
- le cas se services imbriqués où un service est lui-même client d'un autre service (accès à la base de données fait appel à des services d'accès aux fichiers). On parle de configuration logique en série.

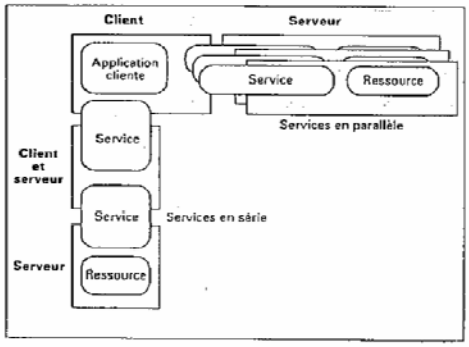
BTD/GL/BE

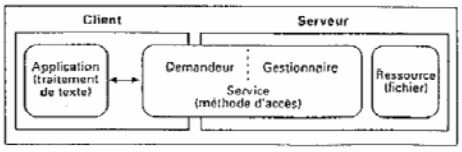
23

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Client - Serveur




BTD/GL/BE

24

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Différentes configurations possibles

| Cas | Clients | Middleware | Serveur |
|---|---|---|---|
| N-1-1 | N | 1 | 1 |
| N-L-K | N | Plusieurs identiques | Plusieurs identiques |
| N-3(Mi-Pi) | N | Différents en amont de différents types de serveurs spécialisés | Différents types spécialisés : Calcul, Impression, Données |

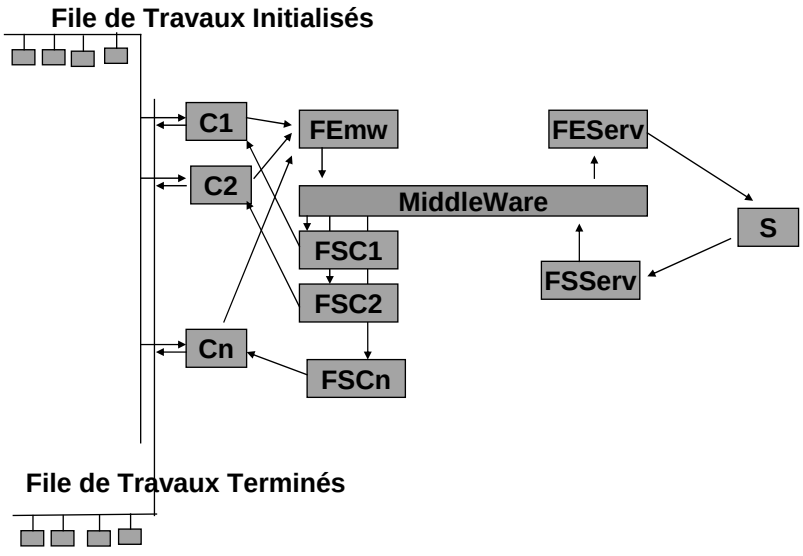
BTD/GL/BE

25

CENTRALE LYON

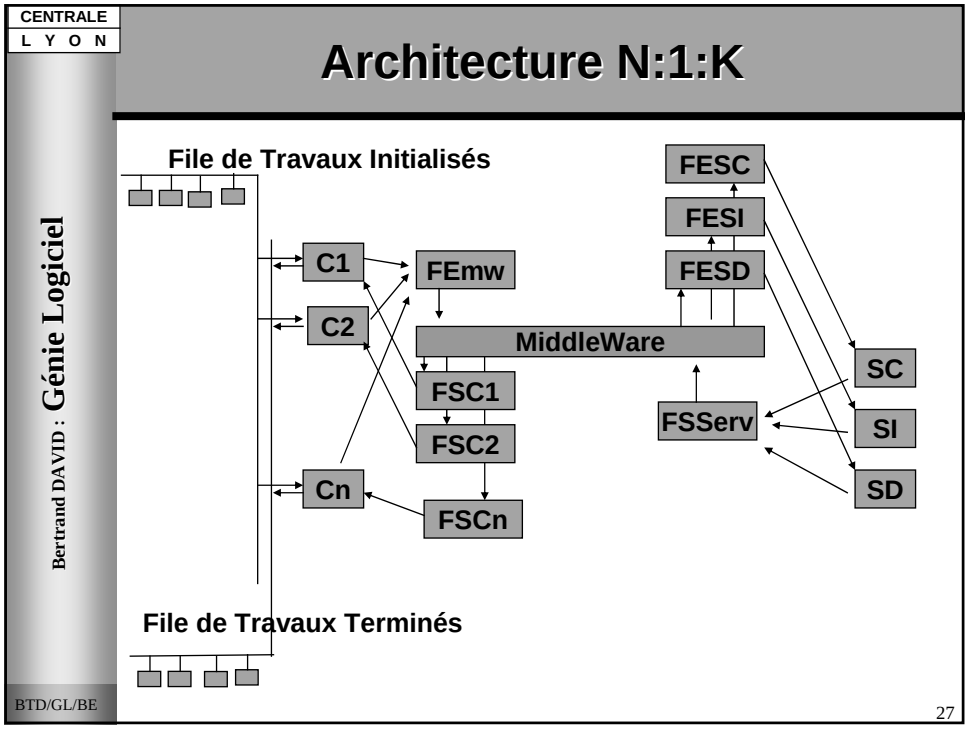
Bertrand DAVID : Génie Logiciel

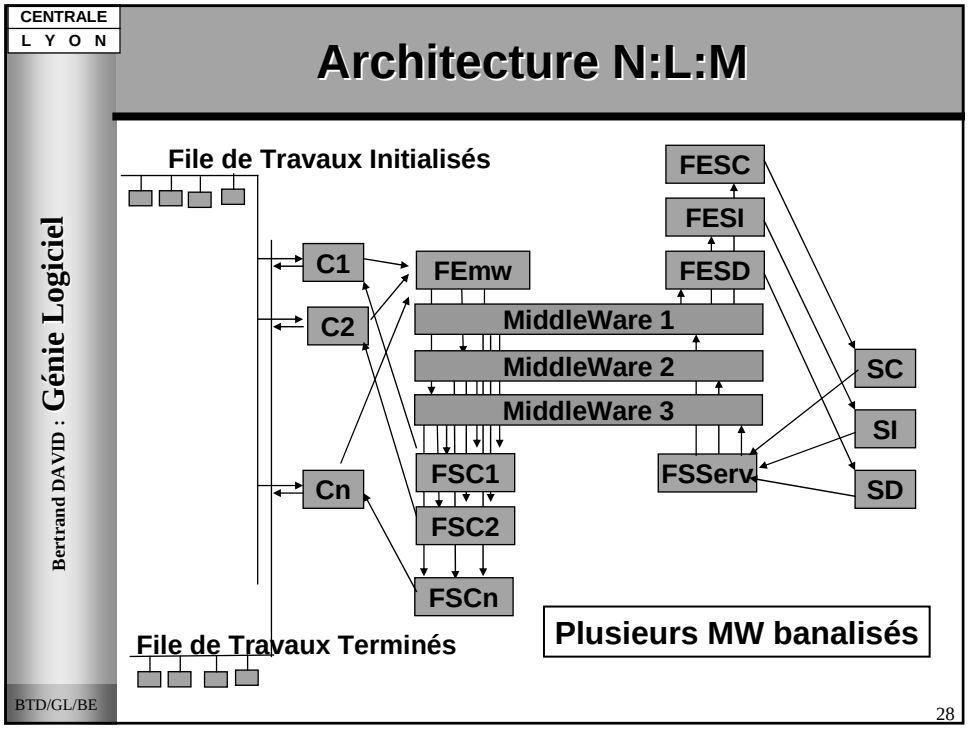
## Architecture N:1:1




BTD/GL/BE

26

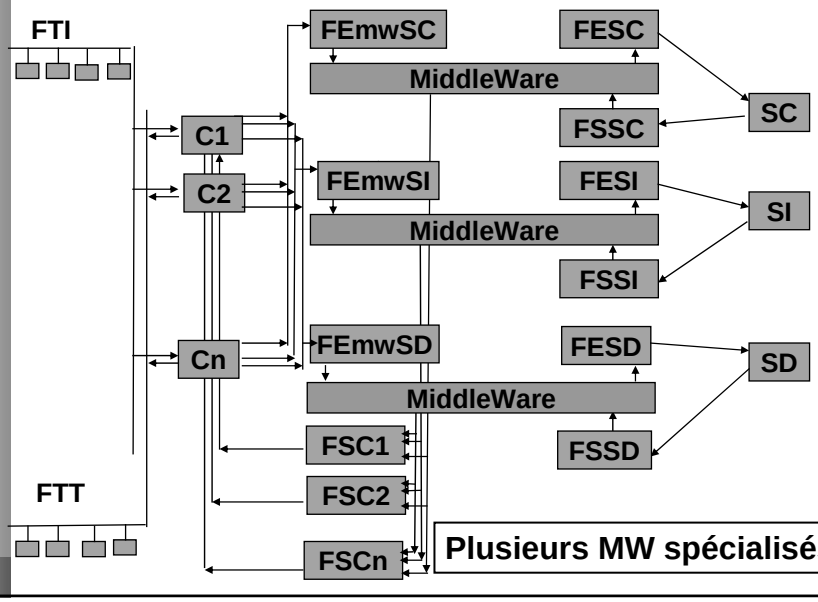

CENTRALE
L Y O N
Bertrand DAVID : Génie Logiciel
Architecture N:1:K
File de Travaux Initialisés
C1
C2
Cn
FEmw
MiddleWare
FSC1
FSC2
FSCn
FESC
FESI
FESD
FSServ
SC
SI
SD
File de Travaux Terminés
BTD/GL/BE
27


CENTRALE
L Y O N
Bertrand DAVID : Génie Logiciel
Architecture N:L:M
File de Travaux Initialisés
C1
C2
Cn
FEmw
MiddleWare 1
MiddleWare 2
MiddleWare 3
FSC1
FSC2
FSCn
FESC
FESI
FESD
FSServ
SC
SI
SD
Plusieurs MW banalisés
File de Travaux Terminés
BTD/GL/BE
28

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Architecture N-3(Mi-Pi)




BTD/GL/BE

29

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

## Dossier à rendre

- L'approche : principes, éléments significatifs, modélisation des travaux, comment ça fonctionne
- Simulation à la main du cas simple M travaux, N clients, 1 middleware, 1 serveur
  - Avec M = 5, N = 2 ou 3, MW = 1, S =1
- Elaborations d'architectures (composants et relations, notamment de type Rendez-vous ou appels de services)
  - Architecture centralisée (non nécessaire car peu utile)
  - Architecture distribuée sans gestion du temps
  - Architecture distribuée avec gestion du temps
- Codes des tâches ADA (corps)
- Généralisations :
  - M K L (avec K middlewares banalisés et L serveurs banalisés)
  - M 3(Ki et Li) avec i= C, I, D pour Calcul, Impression, Données

Montrer l'évolution des architectures et des codes

BTD/GL/BE

30

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

# Threads - exemple

```
package Essai;

/**
 * Ce type a été créé dans VisualAge.
 */
class Afficheur extends Thread {
/**
 * Commentaire relatif au constructeur Afficheur.
 */
public Afficheur() {
        super();
}
  /** constructeur permettant de nommer le
        processus */
  public Afficheur(String s) {
        super(s);
  }
  /** affiche son nom puis passe le controle au
        suivant */
  public void run() {
        while (true) {
          System.out.println("je suis le processus
        "+getName());
          Thread.yield();    // passe le controle
        }
  }
}
```

```
package Essai;

/**
 * Ce type a été créé dans VisualAge.
 */

        class TestThread {
/**
 * Commentaire relatif au constructeur
        TestThread.
 */
public TestThread() {
        super();
}
  public static void main(String args[]) {
        Afficheur thread1 = new Afficheur("1");
        Afficheur thread2 = new Afficheur("2");
        thread1.start();
        thread2.start();
        while (true) {
          System.out.println("je suis la tache
        principale !");
          Thread.yield();
        }
  }
}
```

BTD/GL/BE

31

CENTRALE LYON

Bertrand DAVID : Génie Logiciel

# Producteur – Consommateur (1/2)

```
package Essai2;

/**
 * Ce type a été créé dans VisualAge.
 */
class Consommateur extends Thread {
  private TamponCirc tampon;
/**
 * Commentaire relatif au constructeur Consommateur.
 */
public Consommateur() {
        super();
}
  public Consommateur(TamponCirc tampon) {
        this.tampon = tampon;
  }
  public void run() {
        while (true) {
          System.out.println("je preleve " +
        ((Integer)tampon.preleve()).toString());
          try {
                  Thread.sleep((int)(Math.random()*200));
                  // attente de max 200 ms
          } catch (InterruptedException e) {}
        }
  }
}
```

```
package Essai2;

/**
 * Ce type a été créé dans VisualAge.
 */
class Producteur extends Thread {
  private TamponCirc tampon;
  private int val = 0;
/**
 * Commentaire relatif au constructeur Producteur.
 */
public Producteur() {
        super();
}
  public Producteur(TamponCirc tampon) {
        this.tampon = tampon;
  }
  public void run() {
        while (true) {
          System.out.println("je depose "+val);
          tampon.depose(new Integer(val++));
          try {
                  Thread.sleep((int)(Math.random()*100));
                  // attente de max 100 ms
          } catch (InterruptedException e) {}
        }
  }
}
```

BTD/GL/BE

32

CENTRALE L Y O N

Bertrand DAVID : Génie Logiciel

# Producteur – Consommateur (2/2)

```
package Essai2;
/** * Ce type a été créé dans VisualAge. */

class TamponCirc {
  private Object tampon[];
  private int taille;
  private int en, hors, nMess;

/** * Commentaire relatif au constructeur TamponCirc. */
public TamponCirc() {
        super();
}
  /** constructeur. crée un tampon de taille éléments */
  public TamponCirc (int taille) {
        tampon = new Object[taille];
        this.taille = taille;
        en = 0;
        hors = 0;
        nMess = 0;
  }
  public synchronized void depose(Object obj) {
        while (nMess == taille) {    // si plein
          try {
                wait();             // attends non-plein
          } catch (InterruptedException e) {}
        }
        tampon[en] = obj;
        nMess++;
        en = (en + 1) % taille;
        notify();               // envoie un signal non-vide
  }
  public synchronized Object preleve() {
        while (nMess == 0) {    // si vide
          try {
                wait();         // attend non-vide
          } catch (InterruptedException e) {}
        }
        Object obj = tampon[hors];
        tampon[hors] = null;    // supprime la ref a l'objet
        nMess--;
        hors = (hors + 1) % taille;
        notify();               // envoie un signal non-plein
        return obj;
  }
}
```

```
package Essai2;

/**
 * Ce type a été créé dans VisualAge.
 */
class UtiliseTampon {
/**
 * Commentaire relatif au constructeur UtiliseTampon.
 */
public UtiliseTampon() {
        super();
}
  public static void main(String args[]) {
        TamponCirc tampon = new TamponCirc(5);
        Producteur prod = new Producteur(tampon);
        Consommateur cons = new Consommateur(tampon);

        prod.start();
        cons.start();
        try {
          Thread.sleep(30000); // s'execute pendant 30 secondes
        } catch (InterruptedException e) {}
  }
/**
 * Cette méthode a été créée dans VisualAge.
 * @return
 */
public Object run() {
        return null;
}
}
```

BTD/GL/BE

33

CENTRALE L Y O N

Bertrand DAVID : Génie Logiciel

# Horloge – Processeur (1/2)

```
package Essai3a;

/**
 * Ce type a été créé dans VisualAge.
 */
public class Horloge extends Thread {
        Processeur[] listeProcesseurs;


/**
 * Commentaire relatif au constructeur Horloge.
 */
public Horloge(Processeur[] liste) {
        super();
        listeProcesseurs = liste;

}
/**
 * Cette méthode a été créée dans VisualAge.
 */
public synchronized void run() {
        for (int i = 1;i <= 10;i++)
          {
                  System.out.println(i);
            for(int j = 0 ; j<3 ;j++)
                  listeProcesseurs[j].top(i);
          }
}
}
```

```
package Essai3a;

/**
 * Ce type a été créé dans VisualAge.
 */
public class MonHorloge {
        public String npro;

/**
 * Commentaire relatif au constructeur MonHorloge.
 */
public MonHorloge() {
        super();
}
        public static void main(String args[]) {
        Processeur[] liste = new Processeur[3];
        Processeur p1 = new Processeur("1"); liste[0]=p1;
        Processeur p2 = new Processeur("2"); liste[1]=p2;
        Processeur p3 = new Processeur("3"); liste[2]=p3;
        Horloge horloge = new Horloge(liste);
        System.out.println("init OK");
        horloge.start();
        p1.start();
        p2.start();
        p3.start();
        try {
          Thread.sleep(7000); // s'execute pendant 30 secondes
        } catch (InterruptedException e) {};
        System.out.println("on tue les threads");
        p1.stop();
        p2.stop();
        p3.stop();
        System.out.println("fini");
  }
}
```

BTD/GL/BE

34

CENTRALE L Y O N

Bertrand DAVID : Génie Logiciel

# Horloge – Processeur (2/2)

```
ackage Essai3a;

/**
 * Ce type a été créé dans VisualAge.
 */
public class Processeur extends Thread {
	String npro;
	int top;
	boolean isAlive = true;
/**
 * Commentaire relatif au constructeur Processeur.
 */
public Processeur() {
	super();
}
/**
 * Cette méthode a été créée dans VisualAge.
 * @param num java.lang.String
 */
public Processeur(java.lang.String  num) {
	npro=num;

}
/**
 * Cette méthode a été créée dans VisualAge.
 */
public synchronized void run() {
	while (isAlive) {
	  try {
			wait();		// attends top
	   } catch (Exception e) {/*System.out.println("l'erreur est là: top = "+ top + " process = "+ npro);*/}
	   System.out.println("reveil du processeur "+ npro);
	   yield();
	  }
}
/**
 * Cette méthode a été créée dans VisualAge.
 */
public synchronized void top(int num) {
	top = num;
	System.out.println(top +" top pour le processeur "+ npro);
	notify();
}
}
```

BTD/GL/BE

35

CENTRALE L Y O N

Bertrand DAVID : Génie Logiciel

# Comment exporter et importer avec VisualAge :

**Pour exporter :**

- Il faut versionner le projet puis exporter le Référentiel en se rappelant les noms de projet et de package.

**Pour importer :**

- Il faut créer un nouveau projet vide. Ce nouveau projet apparaît comme un dossier vide.
- Sélectionner ce projet, cliquer sur le bouton droit et choisir importer...
- Choisir Référentiel, puis Projets et aller chercher le fichier XXX.dat à l'aide du bouton Parcourir...
- Une fois le fichier XXX.dat choisi, cliquer sur Détails et cocher le projet YYY. Vérifiez que la version Z.Z est également cochée.
- Valider 2 fois : VisualAge charge le projet.
- A ce stade, rien n'a changé dans l'interface graphique de VisualAge : le programme a en fait chargé le projet dans une base de données. Il faut donc charger le projet définitivement.
- Sélectionner de nouveau le projet dernièrement créé. Cliquez sur la 4ème icône de la barre de boutons pour créer un nouveau package dans le projet.
- Choisir Ajouter un package à partir d'un référentiel. Rechercher le package correspondant. Valider.
- Normalement, c'est fini et ça marche

BTD/GL/BE

36