Institut
Mines-Télécom

Bases de la programmation pour le chef de projet digital

Télécom ParisTech
Jean-Claude Moissinac – Octobre 2014

Mastère CPD

Objectifs

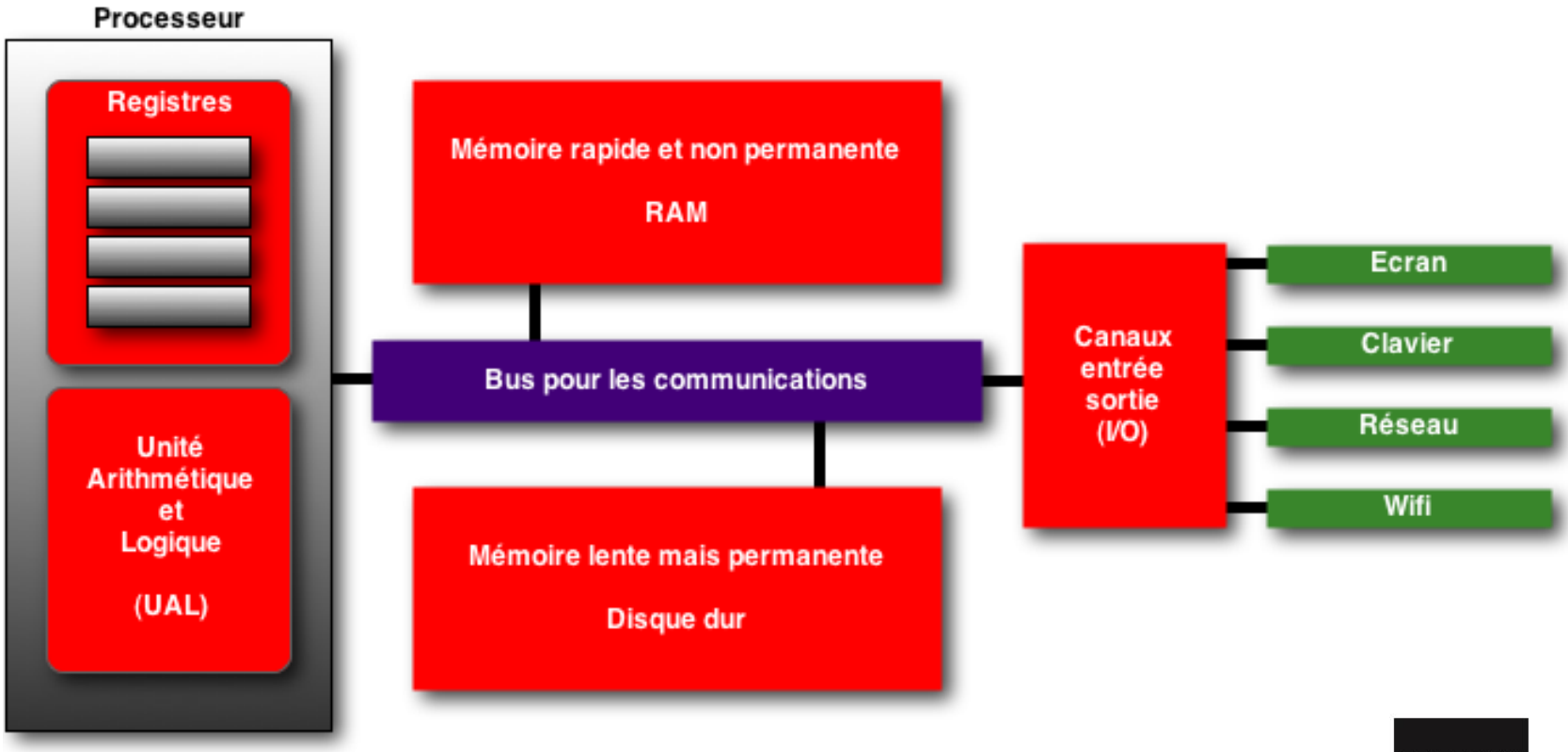
- **Montrer ce qu'est un langage de programmation**
  - Utilité
  - Contraintes
- **Présenter quelques langages**
- **Présenter quelques principes**
- **Présenter quelques outils**

Objectifs (suite)

- **Donner une idée des bonnes pratiques de programmation et les savoir-faire permettant de travailler dans un contexte de développement logiciel professionnel.**
- **Vous donner les bases nécessaires pour forger un modèle du fonctionnement d'un ordinateur et des façons de lui faire faire des choses**
- **Vous donner des bases imaginer la structure d'un programme et des structures de données associées**
- **Vous donner une vision des fondements théoriques de l'informatique.**

L’ordinateur

- C'est le mathématicien **John Von Neumann** en 1940 qui a inventé l'ordinateur actuel aussi appelé Machine de Von Neumann.



Programme

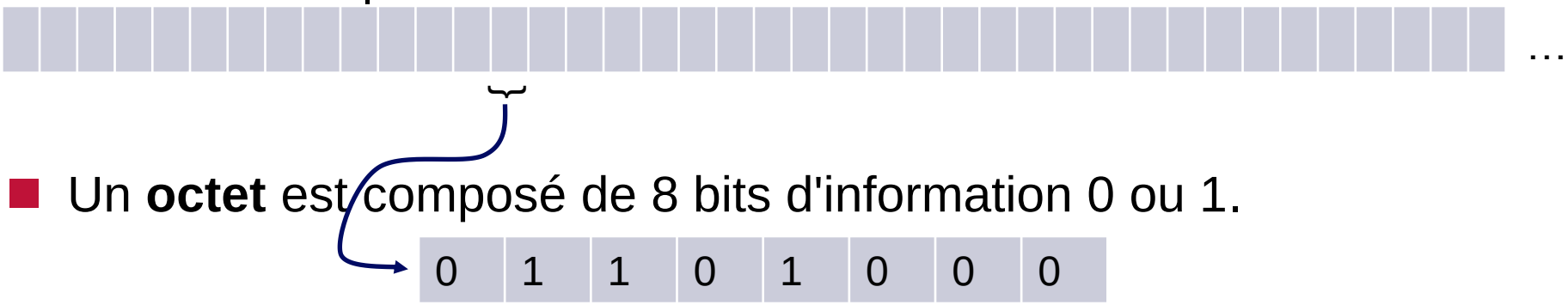
- **Algorithme + Structure de données=Programme**

(Recette + ingrédients = programme)

- **Lors de l'exécution, le programme est stocké en mémoire à accès rapide (RAM) et exécuté par le processeur**

Mémoire RAM

- La mémoire RAM est composée d'une suite d'octets numérotés à partir de zéro.



- Un **octet** est composé de 8 bits d'information 0 ou 1.
- Un octet peut représenter 256 informations différentes en fonction des différentes configurations possibles de 0 et de 1
- La mémoire est ensuite découpée en octets consécutifs correspondant à un même ensemble de données

Adresse RAM

- L'octet est la plus petite partie **adressable** de la mémoire: lorsque l'on veut lire des données dans la mémoire ou lorsque l'on veut écrire des données dans la mémoire, on le fait octet par octet.

- L'**adresse** d'une donnée en mémoire est le numéro du premier octet de cette donnée.

Données en mémoire

- Les données d'un programme sont **représentées** en mémoire par des suites d'octets. Exemples:
  - Ainsi une valeur numérique entre 0 et 255 peut être représentée en mémoire par un seul octet en utilisant les 8 bits (0 ou 1).
  - Un nombre plus grand pourra être représenté par plusieurs octets (par exemple, 2 octets pour un entier sur 16 bits)
  - Il y a moins de 256 caractères (ponctuation comprise) dans la langue française usuelle, on peut donc décider d'attribuer un numéro entre 0 et 255 à chaque caractère et représenter celui-ci par un octet.
- Lorsqu'il utilise un langage de programmation de haut niveau comme javascript, le programmeur a rarement besoin de savoir comment les données sont représentées en mémoire.
  - Mais le fait d'en avoir une petite idée aide quelquefois à comprendre ce qui se passe ou ce qu'on fait

Algorithme

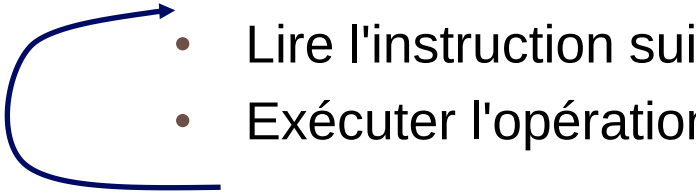
- **Une recette pour atteindre un objectif**
  - Ex: méthode (algorithme) de tri
- **La recette est indépendante de la langue utilisée pour la décrire**
- **La recette définit**
  - les éléments sur lesquels on travaille (les ‘ingrédients’)
  - Une série d’actions à faire dans un certain ordre
  - Peut faire référence à une autre recette
- **Le chef de projet peut concevoir des algorithmes**
  - … sans être programmeur

Unité Arithmétique et Logique

- L'UAL, Unité Arithmétique et Logique, est le moteur de l'ordinateur.
- L'UAL sait exécuter des opérations simples:
  - recopie d'un mot de la mémoire dans un registre ;
  - recopie d'un registre dans un mot de la mémoire ;
  - opérations sur les registres ;
  - saut : continuer l'exécution à une autre adresse de la mémoire ;
  - saut conditionnel : effectuer un saut si une condition est vérifiée ;
  - mise à zéro d'une zone de la mémoire ;
  - etc.

Exécution d'un programme

- Chaque instruction que le processeur (UAL) sait réaliser possède un code
- Lors de leur exécution, les programmes sont des suites d'instructions
- L'UAL sait lire ces codes en mémoire et effectuer les opérations décrites
- L'ordinateur répète en boucle le travail suivant :
  - Lire l'instruction suivante en mémoire.
  - Exécuter l'opération décrite.



Langage de programmation

En 1958, John Backus (IBM) invente le premier **langage de programmation** nommé FORTRAN.

- Sur les premiers ordinateurs (1940-1960), on écrivait directement les codes pour programmer les ordinateurs.
- Un langage de programmation permet de décrire des programmes de manière « lisible ».
- Un programme particulier –compilateur, interpréteur- permet de transformer le programme « lisible » en une suite de codes que l'UAL sait exécuter.
- Il y a de nombreux langages de programmation
  - Java, Javascript, C, C++, Actionscript, PHP…
  - Utilisés dans des contextes différents

Langage de programmation et projet multimédia

- **Pourquoi programmer?**
  - Définir la réaction de l'ordinateur à des évènements:
    - actions de l'utilisateur,
    - arrivée d'un message…
  - Ajouter de la dynamicité à des pages HTML
    - Afficher la date courante
    - Tenir compte de l'utilisateur connecté…
  - Ajouter des possibilités à un programme existant
    - … si celui-ci le permet

Cuire un œuf coque

- **Les éléments avec lesquels on travaille**
  - Casserole, œuf, source d'eau, plaque de cuisson
- **Des fonctions de base** (qui dépendent de l'exécutant)
  - Remplir, deplacer
- **Une séquence**
  - Deplacer(casserole, source d'eau)
  - MettreDans(casserole, source d'eau)
  - MettreDans(casserole, œuf)
  - Deplacer(casserole, plaque de cuisson)
  - Allumer(plaque de cuisson)
  - FixerValeur(attente, 180000)
  - Attendre(attente)
  - Eteindre(plaque de cuisson)

Opérations de base

- **Opérations fournies par le langage**
  - Accès à des fonctions du système (fonctions de base d'entrée/sortie, fonctions de calcul, lecture de l'horloge…)
- **Opérations fournies par un travail préalable de programmation**
  - Déjà fait dans le cadre du programme
  - Déjà fait par un tiers (librairies)

Librairies

- **Ensemble de fonctions qui permettent de rendre des services dans un domaine**
  - Définissent le type de données sur lesquelles elles travaillent
  - Définissent un ensemble de traitements qu'elles assurent sur des données de ce type
- **Exemples**
  - Librairie de traitement d'image
  - Librairie d'accès au réseau
  - Librairie d'opérations sur des textes

Définir un programme, deux approches

- **De bas en haut**
  - Définir des fonctions élémentaires sur lesquelles on veut batir
  - Elaborer des fonctions plus complexes par combinaison de fonctions élémentaires jusqu'à l'élaboration d'une fonction qui réponde à l'objectif
- **De haut en bas**
  - Décomposer l'objectif en sous-objectifs
  - Décomposer chaque sous-objectif en sous-objectifs
  - Continuer jusqu'à ce que chaque sous-objectif soit réalisable par combinaison de fonctions de base du dispositif d'exécution retenu
- **Mixte/Objet**
  - Définir les opérations sur les 'objets' que l'on va manipuler
  - Construire avec ces objets

Approche ‘Objet’

- **Définir des objectifs par les catégories d’’objets’ qu’on veut manipuler et les méthodes qu’on sait leur appliquer**

- **Ex: ‘objet’ qui définit un organisateur**
  - Changer/Demander le nom de son organisation
  - Changer/Demander son numéro de téléphone
  - Associer un nouvel évènement à cet organisateur
  - Demander la liste d’évènements dont s’occupe un organisateur
  - …

Les variables

- **Un nom donné à un emplacement pour ranger quelque chose**
  - *Tiroir-du-haut*
  - *Passe au broyeur le (contenu du tiroir-du-haut)*
  - *Boite-archive-impots*
  - *Boite-archive-factures*

  Je peux définir une action (fonction) traiter le contenu de la boite-archive-factures sans savoir ce qu'elle contient précisément et ce contenu peut changer d'une fois à l'autre
  - *Par exemple*

    Variable: nombreDeCheveux

    nombreDeCheveux ← nombreDeCheveux – 1

    — Noté

    nombreDeCheveux = nombreDeCheveux – 1

    — Dans de nombreux langages

Types de variable

- **Une variable est donc un nom donné pour désigner une portion de la mémoire de l'ordinateur utilisée pour stocker quelque chose**

- **Pour savoir la place que cette chose prend, on doit dire de quel type est cette chose**
  - Des types de base
    - Un entier, un nombre réel, un caractère, une chaîne de caractère, un booléen (vrai/faux)
    - Un tableau d'objets
  - Des types composés à partir des types de base

Déclaration de variable

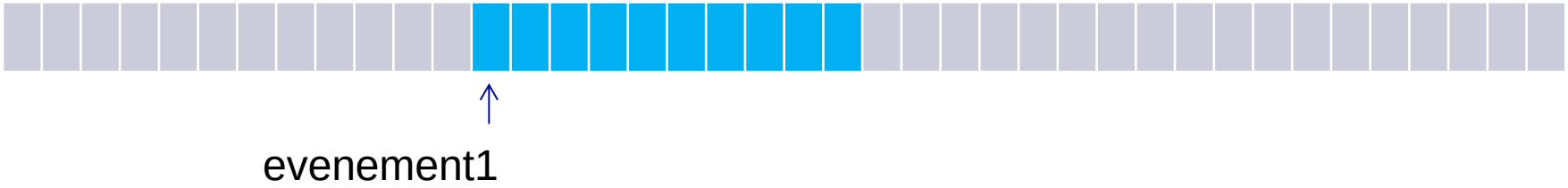
- **On déclare à l'ordinateur qu'on va utiliser une variable en donnant (explicitement ou implicitement):**
  - Le type de la variable
  - Le nom qu'on veut lui donner
  - Exemples
    - Int i;
      - Déclare la variable nommée i de type entier
    - Int notes[10];
      - Déclare la variable nommée notes de type tableau de 10 entiers
  - *Dans certains langages, la déclaration peut être implicite à la première utilisation, mais cette pratique n'est pas conseillée*

Exemple de variable composée

var evenement1; // une variable de type évènement

Qui peut réunir par exemple réunir les informations suivantes

Numéro de l'organisateur; nom de l'évènement; date de l'évènement…



Syntaxe

- **Chaque langage a une façon d'écrire les choses**
- **Le respect de la syntaxe est nécessaire *(même si certains langages supportent un peu de laisser-aller)***
  - *Nombrededoigts est différent de NombreDeDoigts*
- ***Déclaration d'une variable entière nommée nombreDeCheveux***
- ***En javascript***
  - *var nombreDeCheveux = 1234567;*
- ***En Actionscript***
  - *var nombreDeCheveux:int;*
- ***En PHP***
  - *int $nombreDeCheveux;*

Les instructions

- **Des instructions pour manipuler des variables**
  - Donner une valeur à une variable

    i =10;

  Comparer des variables
- **Des instructions pour décider de ce qui va être exécuté**
  - Si … alors … sinon …
  - Tant que ?condition? Alors …
  - Pour i allant de 0 à 10 alors …

Exécution conditionelle ‘if’

- On teste une condition
- Si elle est vraie, exécuter un bloc d’instructions
- Sinon, éventuellement exécuter un autre bloc

```
If (age>=18)
{
permettreLeVote();…
}
Else
{
expliquerQueCelaNeVaPasEtrePossible();…
}
```

Boucle ‘for’

- On a un compteur de boucle
- On répète un bloc d’instructions d’une valeur initiale du compteur de boucle à une valeur finale

```
For (numeroOeuf=1; numeroOeuf<5; numeroOeuf++)
{
        « casser_oeuf numeroOeuf »;
        « mettre le blanc dans bol1 »;
        « mettre le jaune dans bol2 »;
}
```

Boucle ‘while’

- Peut remplacer le ‘for’ et faire plus, par exemple si on ne sait pas à l’avance combien de fois on veut boucler

```
Time = 1;
While (time<200)
{
        « faire ceci »
        time = uneFonctionQuiDonneTime();
        …
}
```

Langages ‘objet’

- Associe un type de données avec les fonctions qui permettent les traitements élémentaires sur ce type de données
- Exemple: un type String

```
var s1 = « ma chaine de caractère »;
var longueur = s1.length;
var s2 = s1.substring(2,3);
```

Outils

- **Editeurs**
  - Exemple: PSPad, coloration syntaxique
- **Visualisation**
  - Exemple: navigateur
- **Debugger**
  - Exemple: navigateur
- **Dépôt de code**
- **Trace des souhaits (Issues Tracker)**

Langages

Langages

- Java
- C, C#, C++
- PHP
- Javascript
- Python
- …

PHP

- **PHP: Hypertext Processor**
- **Langage de script avec syntaxe perl/C**
- **Crée en 1994, actuellement en version 5**
  - PHP3: totalement interpreté
  - PHP4: utilise un moteur de script (ZEND) pour améliorer les performances
- **PHP vient avec une énorme librairie de fonctions**

PHP

- **Le code PHP se trouve dans un fichier côté serveur**
- **Le serveur interprète le code PHP et envoie la page générée au client**
- **Le serveur détecte les fichiers PHP grâce a leur extension**
- **Intégration dans un fichier:**
  - <? *code php* ?>
  - <?php *code php* ?>
  - <?PHP *code php* ?>
  - <SCRIPT LANGUAGE="php"> *code php* </SCRIPT>

  - *Vous verrez PHP plus tard*

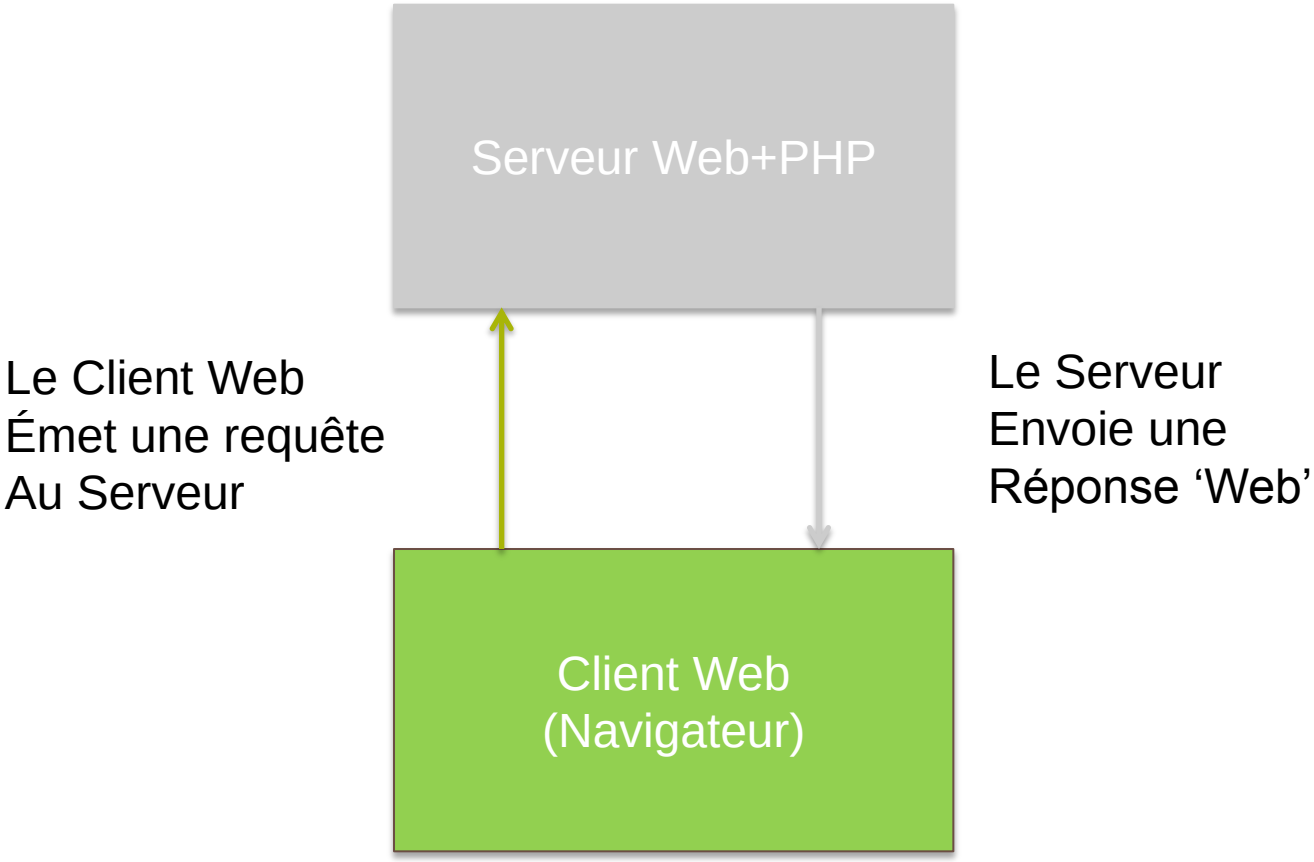
PHP

```
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
     <?php echo '<p>Hello World</p>'; ?>
</body>
</html>
```

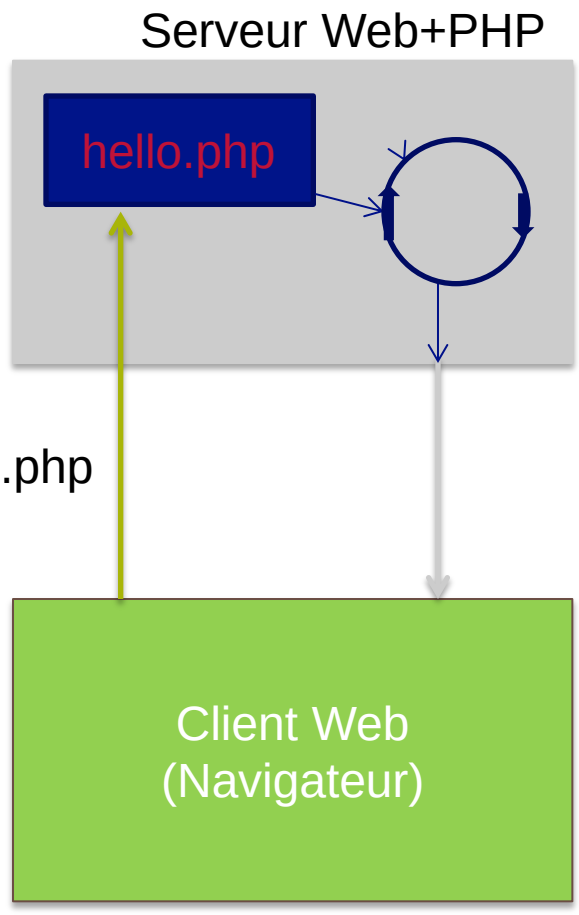
Exécution par le serveur

```
<html>
<head>
<title>PHP Test</title>
</head>
<body>
      <p>Hello World</p>
</body>
</html>
```

Client/Serveur



PHP



http://shadok.enst.fr/cpm/hello.php

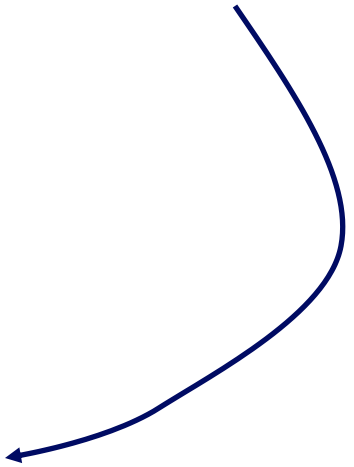
Le Serveur
Envoie une
Réponse ‘Web’

Hello.php

```
<html>
        <head>
                <title>PHP Test</title>
        </head>
        <body>
                <h1><?php echo '<p>Hello </p>'; ?></h1>
        </body>
</html>
```

HelloP.php

http://monsite.com/HelloP.php?nom=Moi&prenom=Meme



```
<html>
        <head>
                <title>PHP Test</title>
        </head>
        <body>
                <h1>
                <?php
                $nom = $_GET[‘nom'];
                $prenom = $_GET[‘prenom'];
                $jour =  date("d/m");
                echo '<p>Hello’ . $prenom . ‘ ‘. $nom. ‘ ‘ . $jour .‘</p>';
                ?>
                </h1>
        </body>
</html>
```

HelloP.php

http://monsite.com/HelloP.php?nom=Moi&prenom=Meme

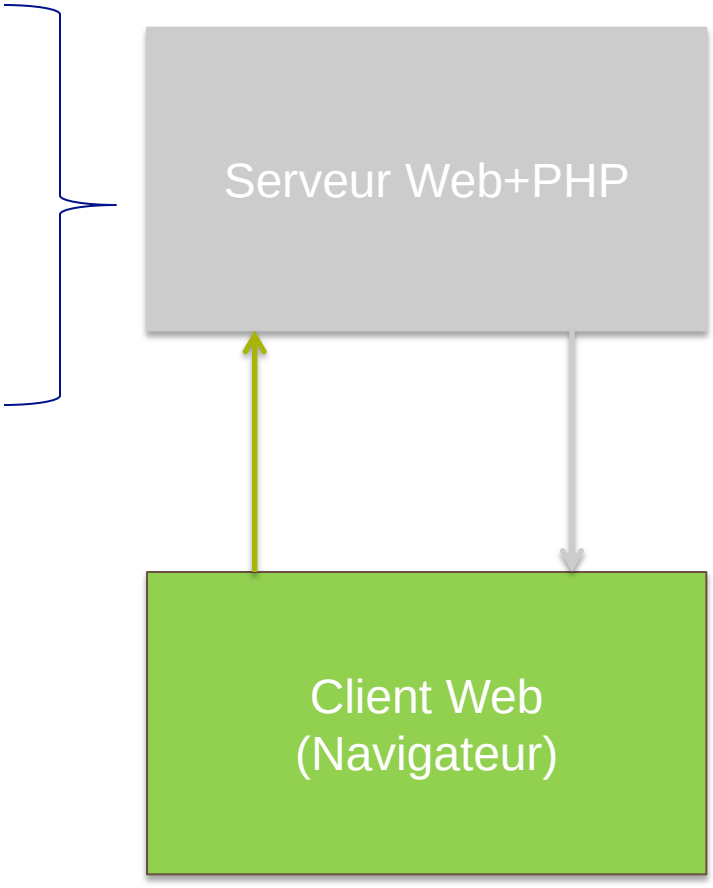
```
<html>
        <head>
                <title>PHP Test</title>
        </head>
        <body>
                <h1>
                <p>Hello Moi Meme 13/10/2014</p>
                </h1>
        </body>
</html>
```

Serveur WAMP/MAMP/LAMP

WAMP

MAMP

LAMP



Windows
Ou
Macintosh
Ou
Linux
et
Apache
MySql
PHP

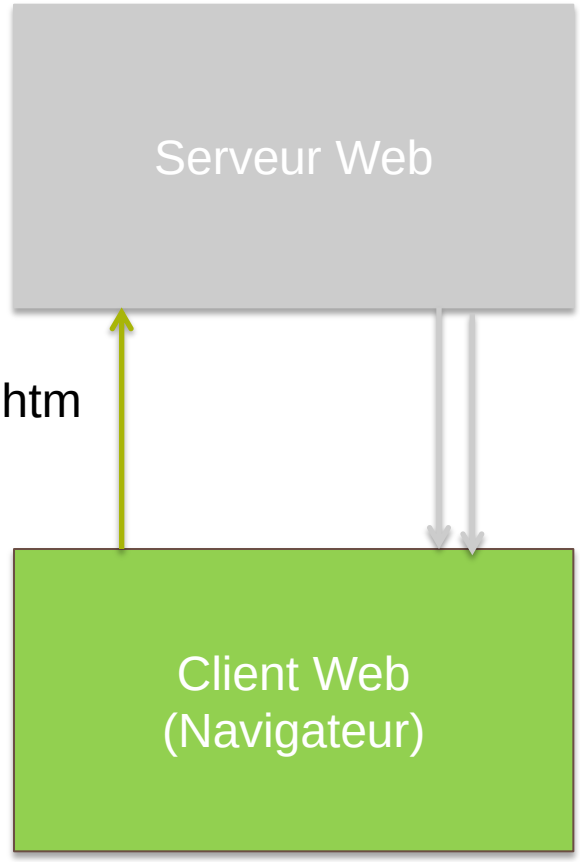
Javascript

Javascript

- **HISTORIQUE**
- Créé en quelques semaines par Brendan Eich, Netscape, en 94 au plus fort de la bataille avec Microsoft ("browsers war")
- Mocha, LiveScript, ECMA-262, ECMAScript…
- Appelé JavaScript pour apaiser une dispute avec Sun
- Inspiré des langages Scheme et Self (et Java pour la syntaxe)
- Très puissant, mais avec des erreurs de jeunesse dignes d'une béta
- 3rd edition
- 5th edition 2011 (ES5)

Javascript, usage classique

http://shadok.enst.fr/cpm/jshello.htm

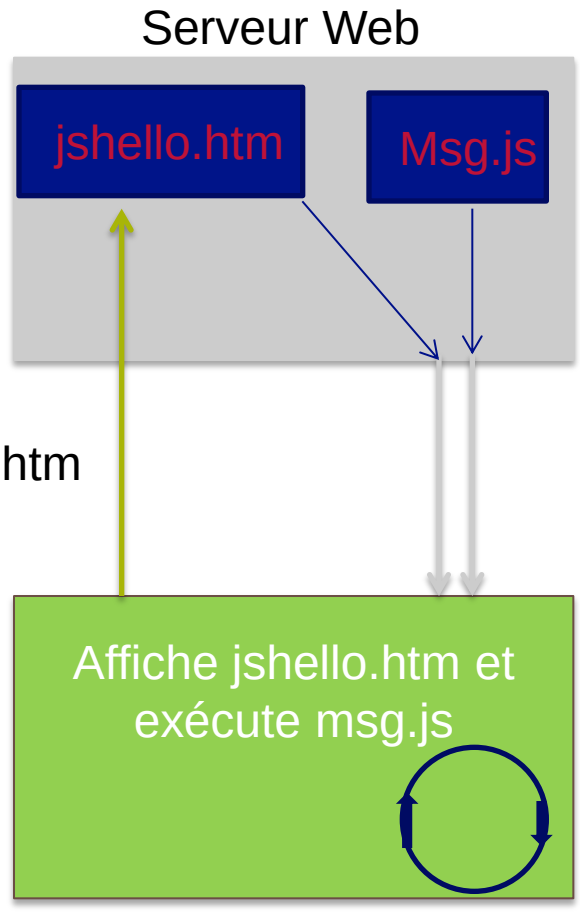


Le Serveur
Envoie une
Réponse HTML+Javascript

jshello.htm

```
<html>
        <head>
                <title>JS Test</title>
            <script type="text/javascript" src="msg.js“/>
        </head>
        <body onload="msgcoucou(); ">
                <h1>
                        <p>Hello</p>
                </h1>
        </body>
</html>
```

Javascript



http://shadok.enst.fr/cpm/jshello.htm

Le Serveur Envoie une Réponse 'Web'

Navigateur

Script msg.js

```
function msgcoucou()
{
        i = 100*56;
        alert('coucou '+i);
}
```

Demo outils dans Chrome

- Voir la page
- Voir le script
- Mettre un point d’arrêt
- Détecter une faute

Interfaces graphiques et programmation

Les interfaces graphiques

- Tout ce qui apparaît sur l'écran d'un ordinateur fait partie des interfaces graphiques : fenêtres, menus, boutons, champs d'entrée, zone de dessin, etc.
- L'utilisateur, par ses actions avec le clavier et la souris, déclenche des actions: création d'une nouvelle fenêtre, action associée à un bouton, etc. Certains changements sont automatiques: les animations.
- Et il est difficile d'imaginer un programme centralisé gérant cette interface graphique.

Interfaces graphiques et ‘objets’

- Dans l'approche ‘objet’ des interfaces graphiques, chaque élément apparaissant sur l'écran est un **objet**
- Chacun de ces objets a des **attributs** : ses coordonnées sur l'écran, ses dimensions, sa profondeur (qui est devant qui ?), etc.
- Chacun de ces objets a des **méthodes** qui lui permettent de répondre aux messages qu'il reçoit.
- Les attributs et les méthodes d'un objet dépendent de son type: fenêtre, bouton, etc.
- Chaque type d'objet est décrit par une **classe** contenant les déclarations d'attributs et de méthodes.

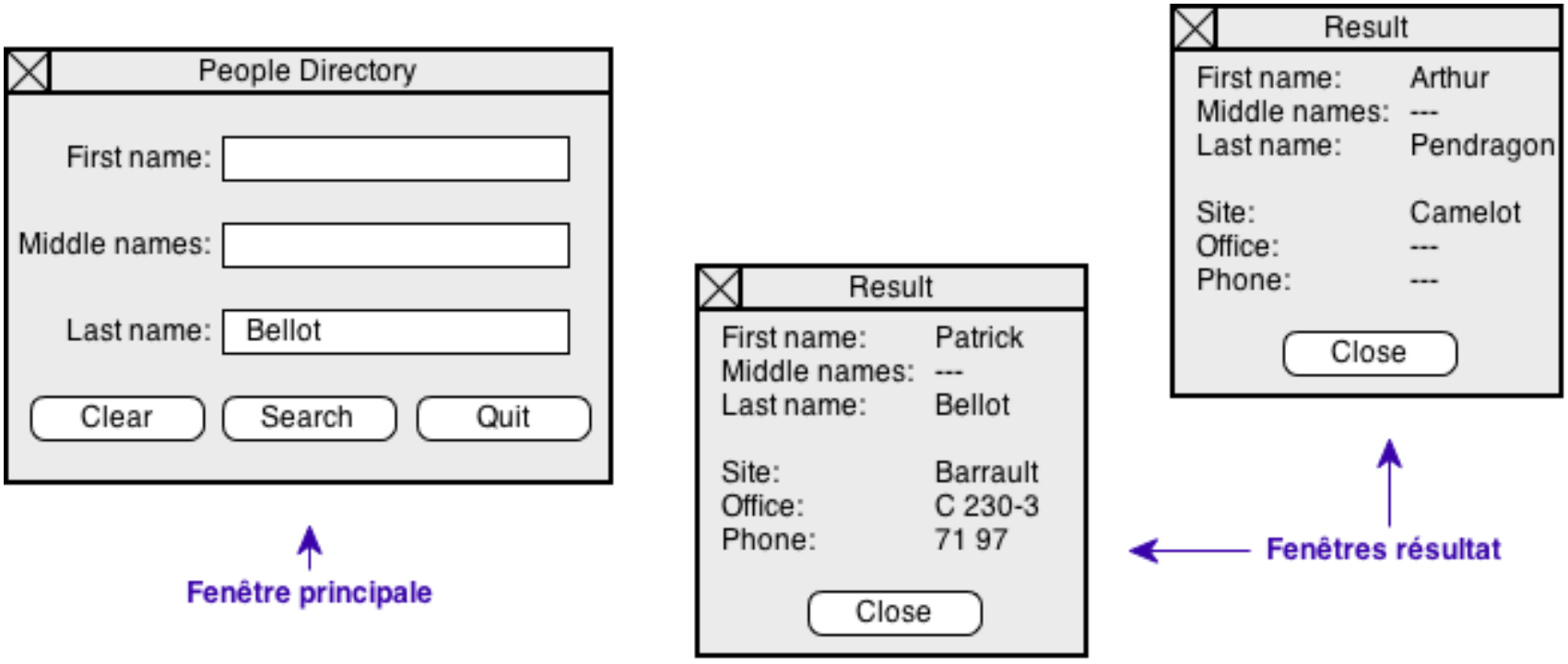
Les messages dans les interfaces

- Le principal émetteur de messages est l'utilisateur de l'ordinateur :
  - un clic de souris avec l'un des boutons ;
  - un déplacement de la souris ;
  - la frappe d'une touche du clavier ;
  - etc.
- En réponse à une action de l'utilisateur sur l'un des objets de l'interface graphique, cet objet peut émettre des messages à destination d'autres objets de l'interface graphique.
- Voyons cela sur un exemple...

Un logiciel d'annuaire

- Dans ce logiciel, on entre des données sur la personne recherchée dans la fenêtre principale, puis on clique sur le bouton Search, une fenêtre est créée avec le résultat de la recherche.
- Un bouton Clear permet de réinitialiser les champs d'entrée.
- Un bouton Quit permet de terminer l'application.

Les fenêtres de l'interface



Architecture des objets

- Pour qu'un objet puisse envoyer un message à un autre objet, il faut qu'il connaisse cet autre objet.
- Un objet connaît un autre objet s'il possède une **référence** sur cet autre objet.
- Une **référence** sur un objet est un attribut qui identifie l'objet référencé.
- Le programmeur doit déterminer les liens entre les objets : quel objet connaît quel objet ?

Echanges entre les éléments (1)

- La fenêtre principale contient 3 champs d'entrée. Comme elle doit y accéder, elle doit connaître ces champs.
- La fenêtre principale doit donc avoir trois attributs qui sont des **références** sur les champs d'entrée.
- La fenêtre principale crée les fenêtres résultats. Lorsque l'application se terminera, elle devra faire disparaître ces fenêtres.
- La fenêtre principale doit donc avoir un attribut qui est une liste de **références** sur les fenêtres résultats.

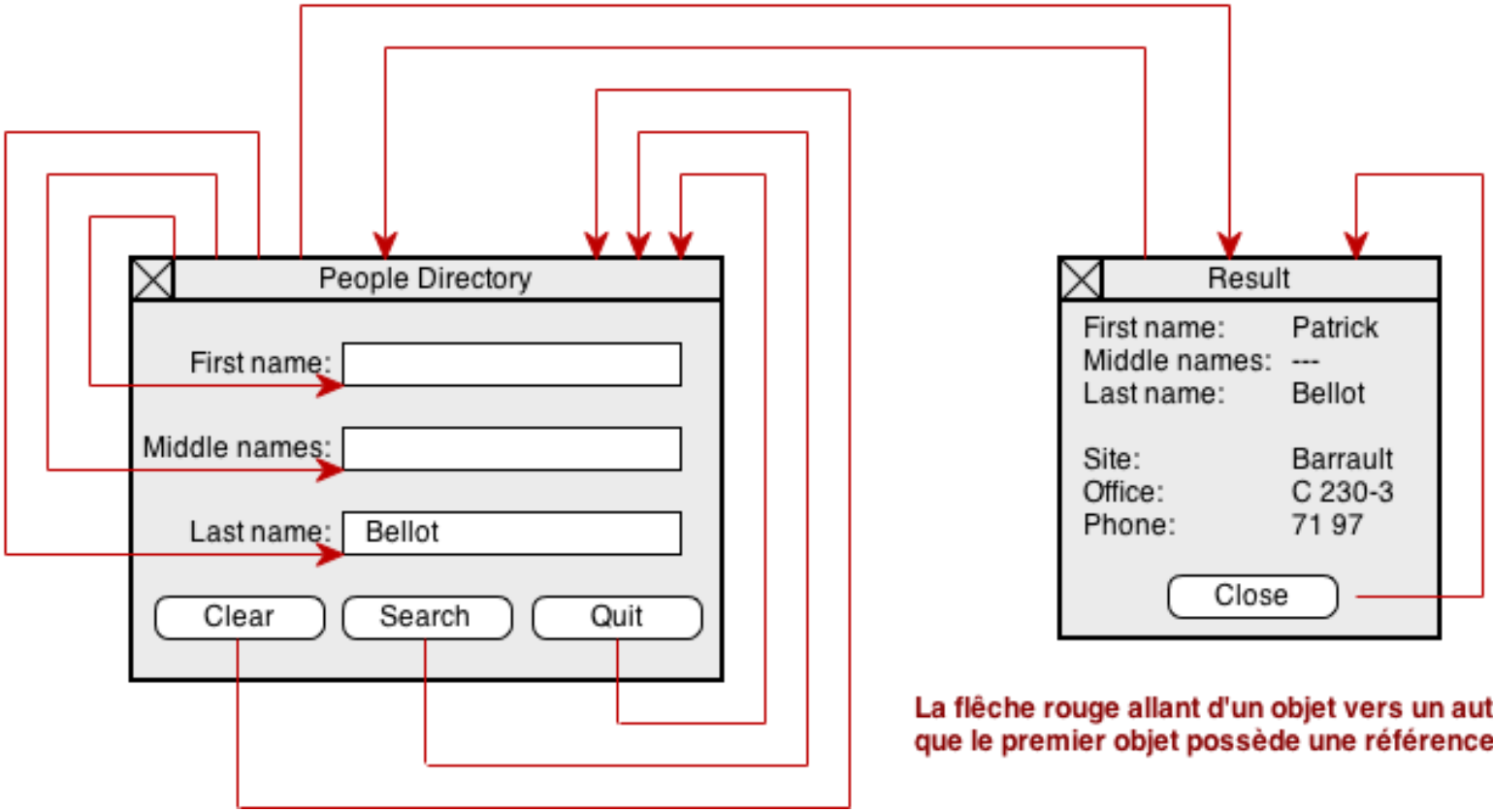
Echanges entre les éléments (2)

- Une fenêtre résultat peut être fermée en cliquant sur son bouton **Close**.
- Mais il ne faut pas oublier que la fenêtre principale maintient une liste des fenêtres résultats. Si une fenêtre résultat est fermée par son bouton **Close**, il faudra qu'elle prévienne la fenêtre principale.
- Chaque fenêtre résultat doit donc avoir un attribut qui est une **référence** sur la fenêtre principale.

Echanges entre les éléments (3)

- Tous les boutons déclenchent des actions: **Clear**, **Search** et **Quit** dans la fenêtre principale, **Close** dans les fenêtres résultat.
- Lorsque l'utilisateur cliquera sur l'un de ces boutons, le plus simple est que le bouton demande à sa fenêtre de faire le travail.
- Donc, chaque bouton doit avoir un attribut qui est une **référence** sur la fenêtre qui le contient.

Echanges entre les éléments (4)



La flèche rouge allant d'un objet vers un autre signifie que le premier objet possède une référence sur le second.

Cascade de messages

- Que se passe-t-il quand l'utilisateur clique sur le bouton **Clear** ? Il faut vider les trois champs d'entrée.
- Lorsque l'utilisateur clique sur le bouton **Clear**, le bouton reçoit un message click().
- Le bouton se contente alors d'envoyer le message clearFields() à la fenêtre qu'il connaît.
- Et la fenêtre envoie le message clear() à chacun des champs d'entrée.
- A la réception du message clear(), chaque champ d'entrée efface le texte qu'il contient. Et il renvoie un message ok à sa fenêtre.
- Quand la fenêtre a reçu ses trois messages ok(), elle renvoie un message ok() au bouton.

Institut
Mines-Télécom

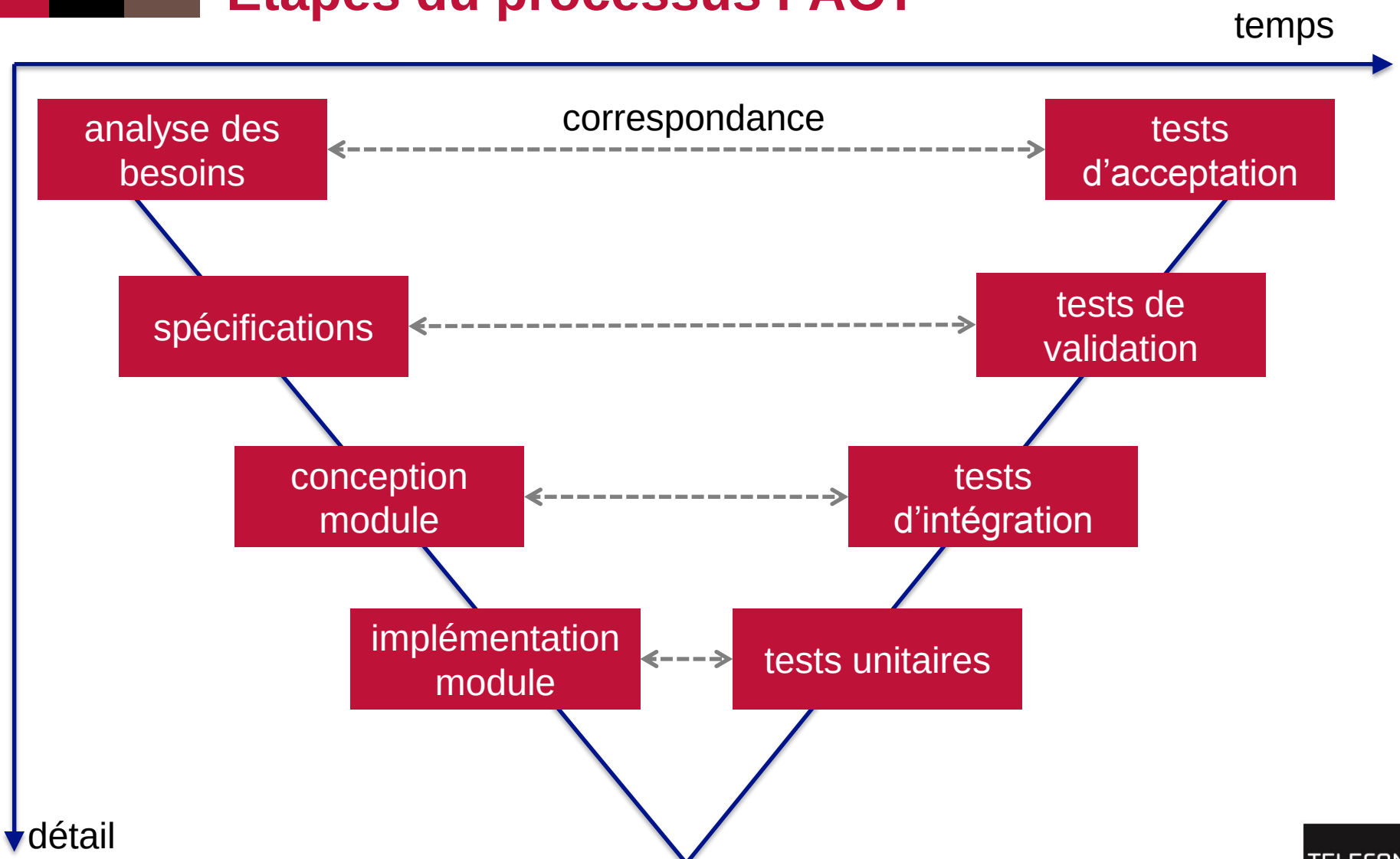
Génie Logiciel

But

- **Organiser l'activité de développement**
- **Réduire les risques :**
  - Technologiques.
    - Ex: tel matériel ne fonctionne pas comme prévu
  - Retards.
    - Ex: telle fonction a pris plus de temps à être codée
  - Inadéquation entre l'attendu et le produit développé.
    - Ex: les performances ne sont pas temps-réel
- **Utilisation d'une méthode de développement**
  - Nombreuses méthodes existantes (Agile, Scrum, …)
  - Une méthode choisie pour CPD

    → une séquence d'étapes ordonnées.

Etapes du processus PACT

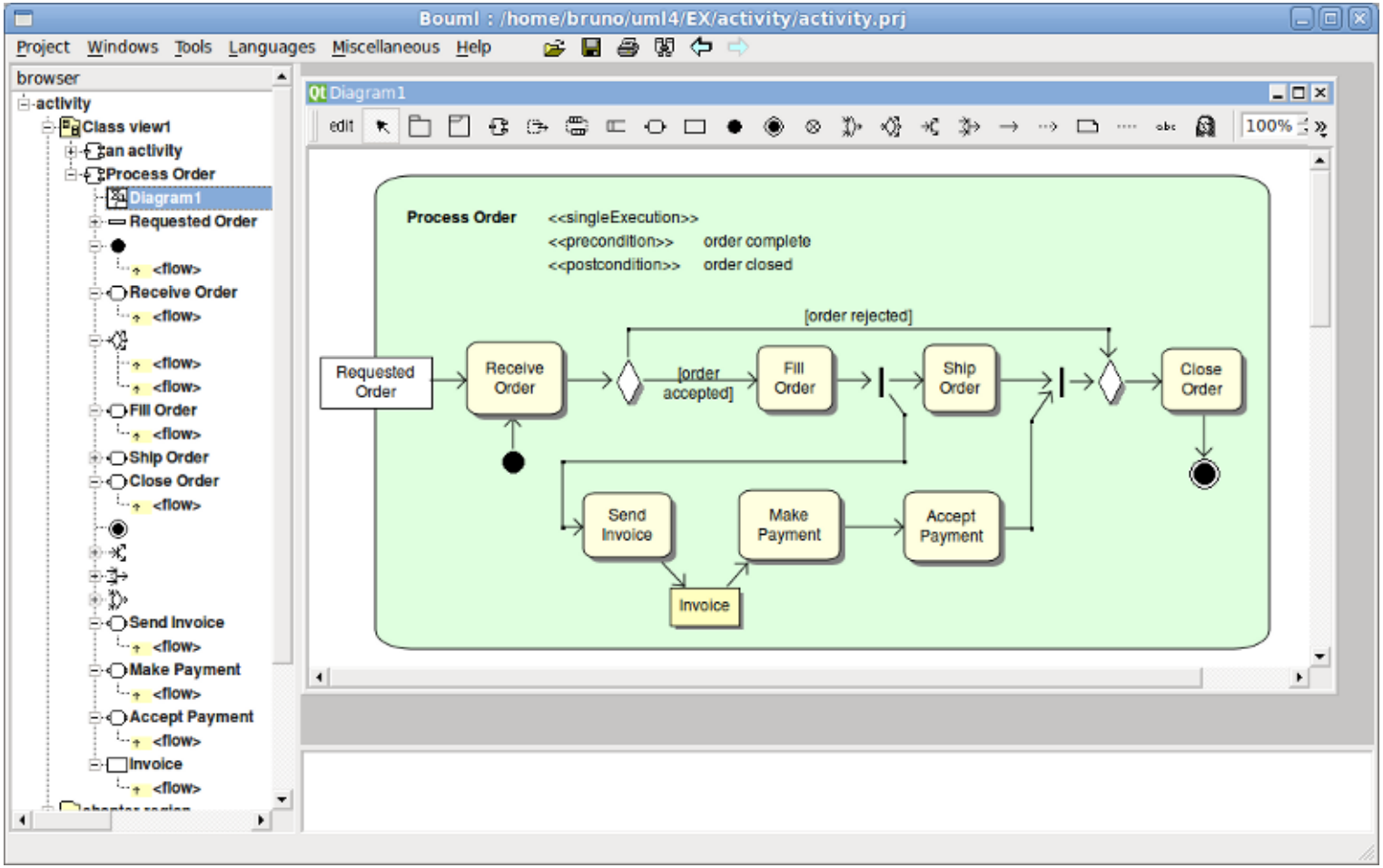


Analyses des besoins

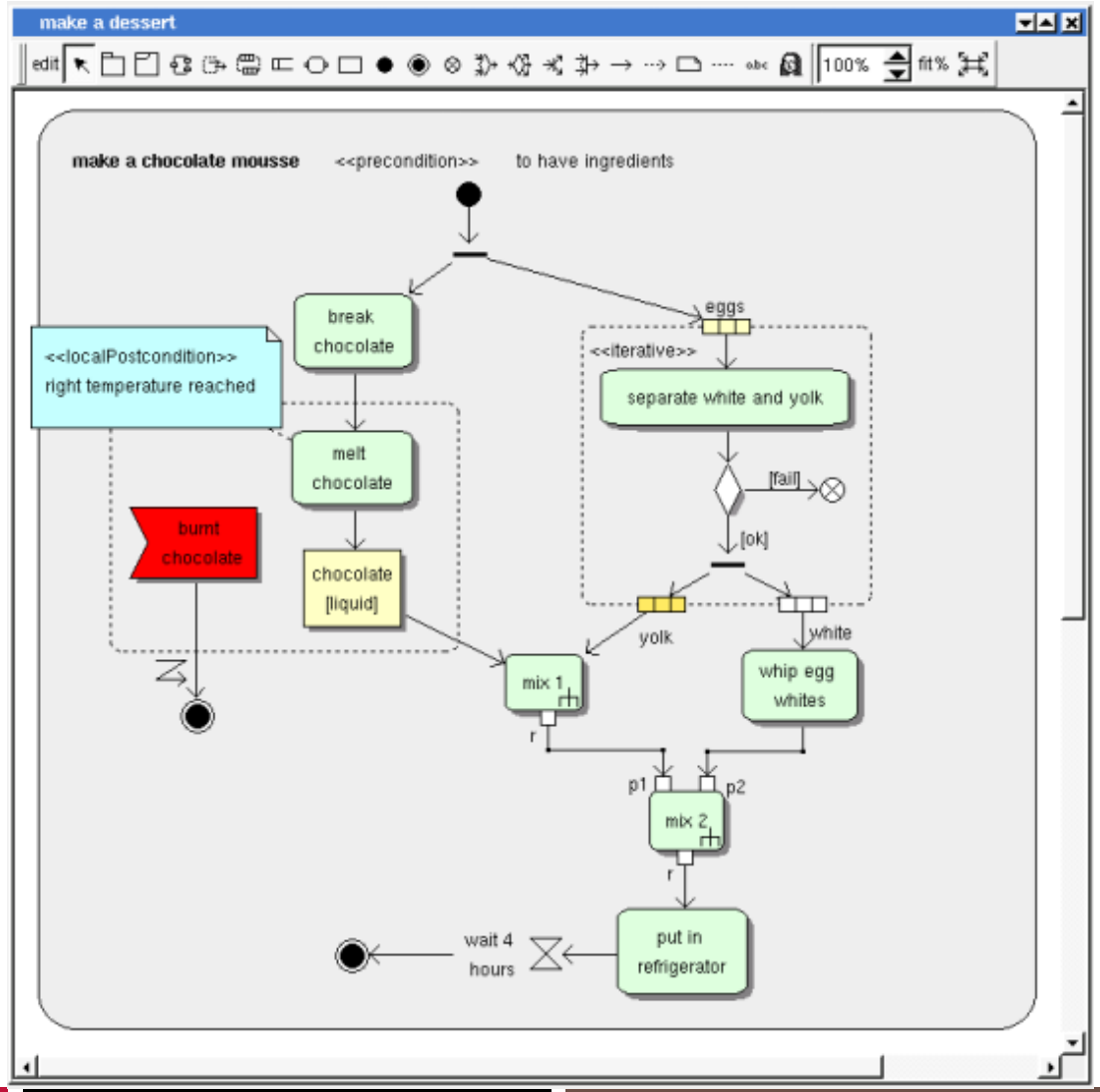
- **Travail à faire**
  - Trouver les scénarios d'utilisation du système.
  - Recenser les acteurs (personne ou automate) agissant sur le système.
  - Recenser les actions sur le système et ses réactions ⇔ les fonctionnalités du système.

- **Produire notamment**
  - Scénarios
  - Description textuelle et/ou schéma décrivant l'architecture du système (cf partie « Architecture de ce cours »)
    - formalisme libre mais cohérent et légendé
  - Diagramme d'activité (temporel)
    - Faisant apparaître les acteurs et actions

Exemple de diagramme d'activités (1)



Exemple de diagramme d'activités (2)



Spécifications

A partir de l'analyse des besoins, formaliser les caractéristiques macroscopiques du système à développer :

- **Fonctions à réaliser**
  - Ex : calcul du rythme d'une musique, reconnaissance de formes, etc.
- **Éléments techniques nécessaires**
  - Ex : un serveur Web, module pour une base de données, couche de communication, etc.
  - Ex d'éléments matériels : liaison Wifi, mobile Android, Kinect, etc.

Spécifications

- **A rendre**
  - Diagramme de boîtes noires avec entrées/sorties:
    - de chaque fonction
    - De chaque élément technique
  - Description textuelle de la nature (voire du format) des données échangées entre les boîtes noires.

Implémentation d'un module

- **Objectif:**
  - Coder les fonctions du modules.
- **Implémenter les méthodes des objets associés au module**
  - Les méthodes non implémentées mais nécessaires à l'exécution du système seront simulées !
- **Simulations possibles :**
  - Fantôme : méthode vide utilisant par exemple juste une trace sur l'affichage pour indiquer qu'elle est appelée.
  - Bouchon : méthode renvoyant toujours le même résultat.
  - Substitut : implémentation de la méthode très simplifiée.

Tests

A chaque type de test correspond une phase :

- **Tests unitaires : vérification du codage au niveau des méthodes des modules.**
  - À faire avec le module
- **Tests d'intégration : vérification de la conception des modules : les modules doivent s'assembler correctement.**
  - A faire pendant l'intégration
- **Tests de validations : vérification du respect des spécifications au niveau du module.**
  - A faire pendant l'intégration
- **Tests d'acceptation : vérification de l'adéquation entre le besoin et le système développé (analyse).**

Intégration

L'intégration est une tâche très chronophages et source de nombreux correctifs aux modules

Ex : des modules défaillants (bugs, cas non prévus, etc.) ou modules non interopérables (mauvaise conception des interfaces).

- **Nécessité de commencer l'intégration tôt**
  - Notamment pour détecter des faiblesses de la communication conçue entre modules

Les méthodes agiles tendent à faire de l'intégration en continu avec des cycles courts d'implémentation