# Développement de portails et outils de travail collaboratifs

Mehdi Louizi

# Plan

# Plan

- Introduction
- Développement de portails
  - Définitions
  - Exemples
  - XML
  - Protocoles d'intégration
  - SSO
- Outils collaboratifs
  - Définitions
  - Exemples
  - Plateformes / Logiciels / Outils

# Introduction

# Contexte

- Les portails sont nés du constat que l'information sur l'Internet et les intranets est diffuse et pas du tout structurée.
- L'internaute est débordé par le nombre d'outils en tout genre.
- souhait d'une approche plus simple et unifiée.

# Contexte

- Visent à regrouper sous un accès unique
  - Un espace informationnel de recherche
  - Un espace communautaire de partage
  - Un espace personnalisé de services
- Caractéristiques :
  - Point d'accès unique
  - Organisation des informations accessibles et des applications disponibles
  - Personnalisation des services offerts, individuelle ou par groupe
  - Contrôle d'accès centralisé et gestion des utilisateurs

# Définition générale

- **Un portail, qu'est ce que c'est ?**
  - Un portail est un point d'entrée sous forme d'application Web regroupant un certains nombres de services et de contenus à un ensemble de clients.

# Les Portails

Visent à remplacer le bureau.

**Avantages :**

- Ils sont accessibles de n'importe quel poste
- Ils sont personnalisés
- Les services augmentent : calendrier, agenda, répertoire en ligne, traitement de texte, un tableur…

Des obstacles restent à surmonter:

- les temps de chargement
- les coûts de connexion
- la largeur de la bande passante

→ les bureaux représenteraient l'évolution naturelle des portails.

# Les principaux portails web

- Les moteurs ou annuaires de recherche
  - Google, Yahoo!,,…
- Site des fournisseurs d'accès
  - gnet.tn, wanadoo.fr, …
- Site de logiciel de navigation
  - msn.com (Microsoft) , skype.com (Skype –> Microsoft)
- Les éditeurs de contenu
  - Hachette.fr, go.com (Disney), …
- Portail spécialisé ou privé
  - Portail d'entreprise, site de communauté (FB, Twitter)

# Différents type de portails Web

- Portail généraliste ou horizontal
  - 1er type de portail apparu
  - Regroupe le maximum d'informations de tous les thèmes sur un seul site.
  - Point d'entrée sur le Web
  - Yahoo, Google, Site Web des providers (Aol, Wanadoo, Free, Tunet, Gnet…).

# Différents type de portails Web

- Portail spécialisé ou vertical
  - Portail spécialisé dans un thème donné
  - Portail d'entreprise ou inter-entreprise
  - Il regroupe et partage des ressources pour un groupe de client ayant un intérêt commun.
    - Keejob, Tunisie-annonces, Pages jaunes, Sogefoires…

# Les portails horizontaux

- But
  - Attirer un maximum de client sur le serveur
  - Vendre de l'espace aux annonceurs (Estimation 2011 : 50 Milliards de dollars).
  - Prendre des commissions sur des transactions
  - Vendre des services améliorés aux clients (boite aux lettres plus grandes avec plus de fonctionnalités).
- Moyens mis en œuvre
  - Fournir et ordonner du contenu et des services pour attirer un maximum de clients

# Les portails horizontaux. L'exemple Google

- Au début un annuaire de sites classés par thèmes.
- Dans le but de garder les clients après la consultation des annuaires, création de services comme les calendriers, les carnets d'adresses, l'agenda, la messagerie instantanée…

# Les portails horizontaux. L'exemple Google

- Maintenant les principales activités de Google sont l'agrégation de contenu :
  - Le service de news de Google regroupe les informations publiés par d'autres site et les classes
  - Le service de mail
  - Gtalk
  - Partage de documents
  - Google Apps
  - Page personnalisée à un utilisateur (iGoogle)
  - Boutique
  - Services de plus en plus évolués comme Google +, Google Voice, Google Maps, Google Streets…

# Les portails verticaux

- Regroupent les sites de communautés et les portails d'entreprises.
- Spécialisés dans un domaine unique.

# Sites de communautés

- Le but est proche des portails horizontaux. Ils cherchent à créer un annuaire de personnes intéressées par un même thème (permet des annonces beaucoup plus ciblées).
  - Facebook, Twitter, Trombi

# Portails d'entreprise

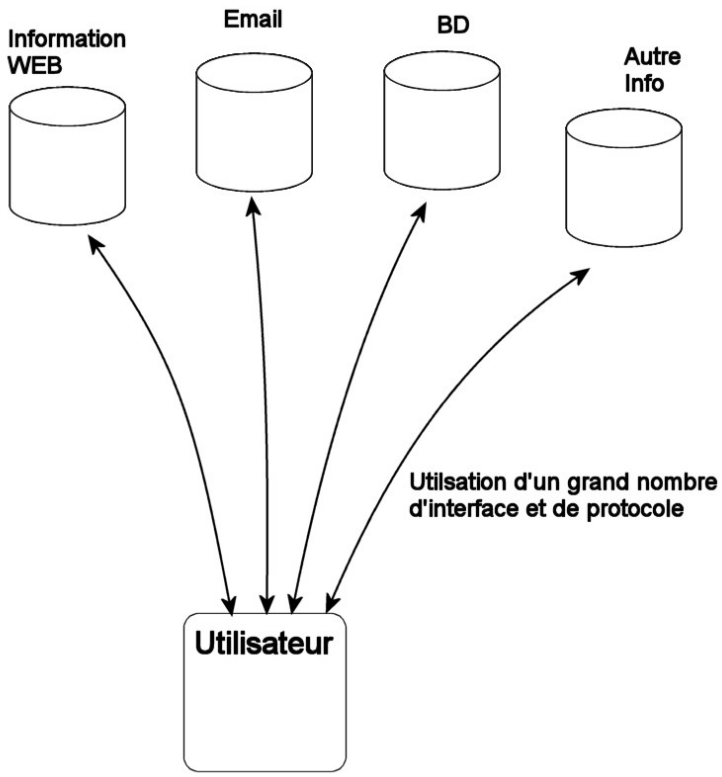
- Le but de ces portails est de faciliter et de regrouper l'accès aux différents systèmes d'information d'une entreprise.
  - Pour faciliter la lecture et la mise à jour par les employés.
  - Pour présenter de l'information à des clients ou à des partenaires.
- Beaucoup de produits commerciaux spécifiques existent pour créer ces types de portails.
  - www.portail-entreprises.fr
  - Site du pôle technologique la Gazelle

# Développement de portails

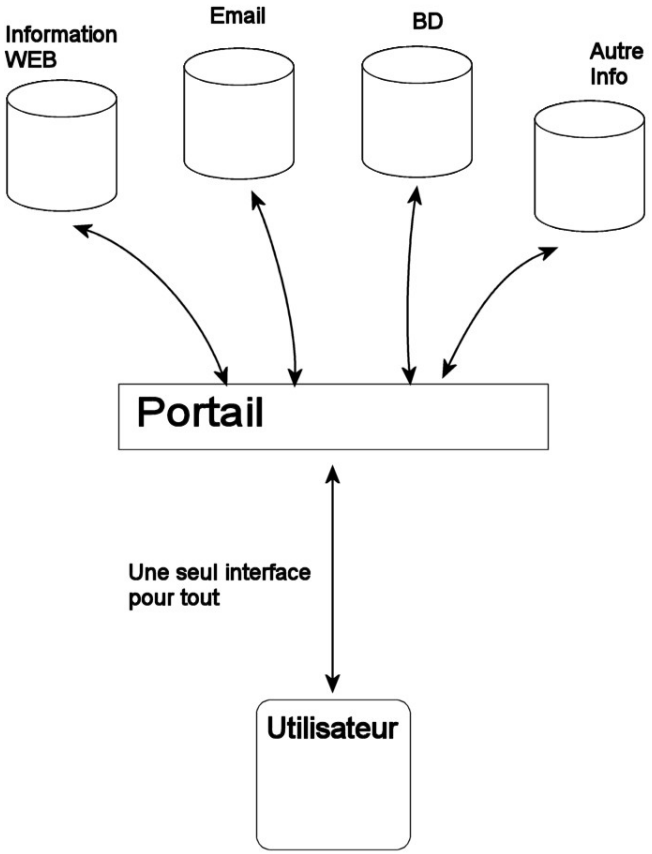
Pourquoi un portail
Les outils de développements

# Pourquoi développer un portail dans une entreprise

- Un utilisateur doit utiliser un grand nombre d'applications pour accéder à toutes les informations de l'entreprise.
- Multiplicité des formats de données, des IHM, informations présentées de façon désordonnées.



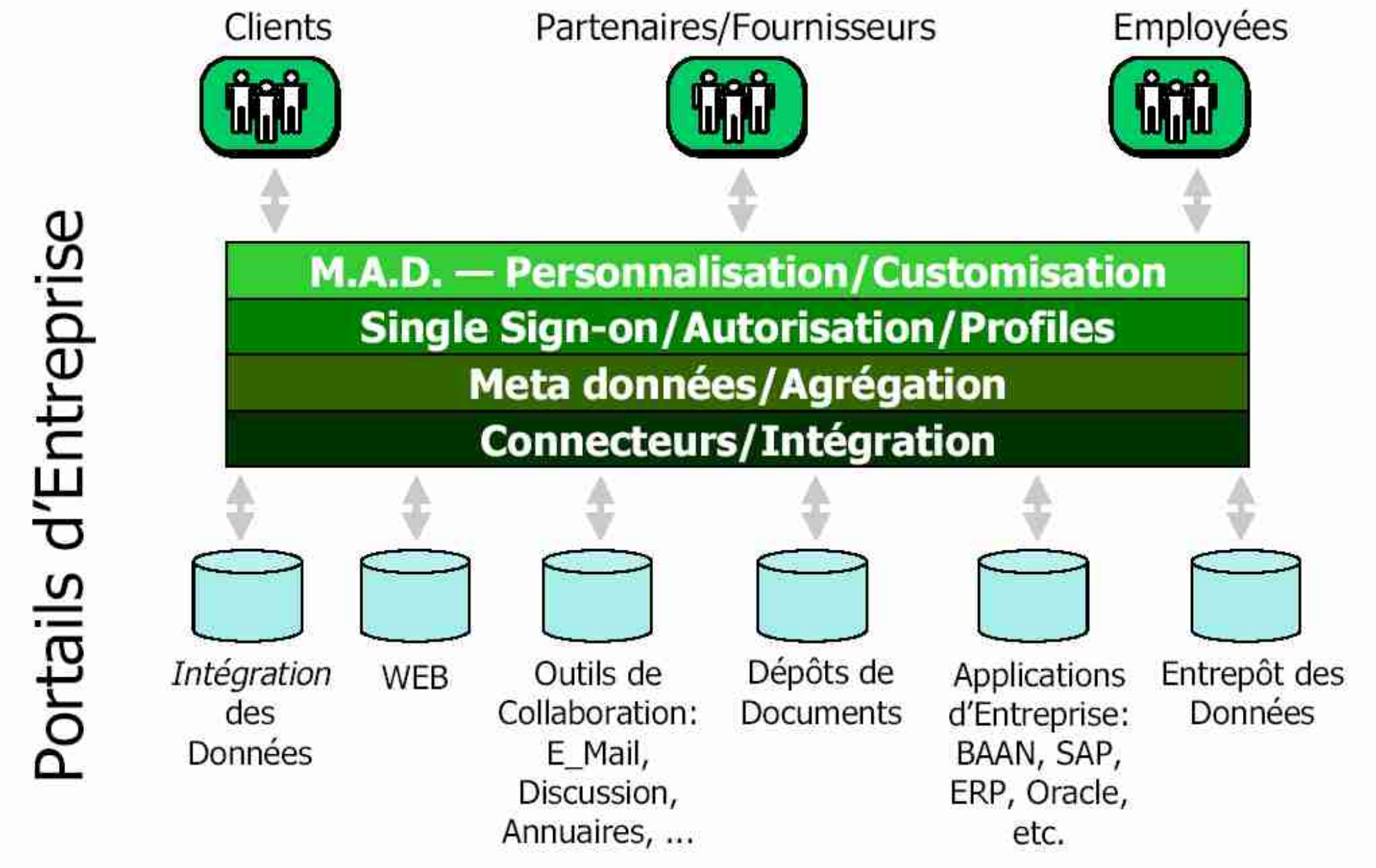
# Pourquoi développer un portail dans une entreprise




- Le but est qu'une application interroge elle-même les applications, puis traite les données pour les présenter de façon ordonnée et unifiée. L'utilisateur y accédera d'une façon unique, en général via un navigateur.

# Apport d'un système de portail

- Un accès simplifié aux informations et aux applications intranet
- Intégration de contenus tiers
- Constitution d'espaces de travail et d'accès à l'information
- Personnalisation des services
- Gestion de l'accès au différentes données de l'entreprise (single sign-on)


Portails d'Entreprise
Clients
Partenaires/Fournisseurs
Employées
M.A.D. — Personnalisation/Customisation
Single Sign-on/Autorisation/Profiles
Meta données/Agrégation
Connecteurs/Intégration
Intégration des Données
WEB
Outils de Collaboration: E_Mail, Discussion, Annuaires, ...
Dépôts de Documents
Applications d'Entreprise: BAAN, SAP, ERP, Oracle, etc.
Entrepôt des Données

Company Logo

Welcome, toto - Personnaliser | Mon profil | Alertes | Fermer session

## Sample Stock Portfolio

**My First Portfolio**

| | | |
|---|---|---|
| DJIA | 8896.09 | -35.59 |
| CAC-40 | 3326.65 | -3.34 |
| DAX | 3320.32 | -40.44 |
| HSI | 10069.87 | -77.97 |
| NIKKEI | 9215.56 | +38.78 |
| NYSE | 495.27 | -1.02 |
| S&P 500 | 936.31 | -2.56 |
| YHOO | 18.27 | -0.12 |

U.S. Markets closed

Get Quotes

Symbol Lookup

Quote data provided by Reuters

Quotes are delayed 20 minutes.
Get Real-Time Quotes.
* = news in the last 24hrs

## Auto Loan Monitor

**National Average**

| | Lowest | Average | Highest |
|---|---|---|---|
| New | 4.5% | 6.18% | 12.95% |
| Used | 4.5% | 6.63% | 14.25% |

**California**

| | Lowest | Average | Highest |
|---|---|---|---|
| New | 4.9% | 6.39% | 12.95% |
| Used | 4.9% | 6.8% | 14.25% |

more metros from *Bankrate.com*

## Sample Industry Outlook

YHOO — Nov 29 8:35am PT

- [audio] File swappers billed for copyright infringements. [1.0 min] (at CBS MarketWatch)
- [external] The Five Dumbest Things on Wall Street This Week (at TheStreet.com)
- [external] CSFB emails show research-bank ties (at CBS MarketWatch)
- UPDATE - Terra Lycos U.S. head quits, portal narrows focus (Reuters)
- Growth stocks back in vogue after bear mauling (Reuters Securities)

## Industry Performance

| | Price Change | Market Cap |
|---|---|---|
| Technology | 3.78% | 2308.9B |
| Healthcare | 1.84% | 1970.5B |
| Major Drugs | 1.81% | 1306.4B |
| Financial | 2.74% | 3353.2B |
| Insurance (Accident & Health) | 2.79% | 97.8B |

as of Nov 27

## HR Self-Service


Travel


Payroll


Option Plan

# Fonctionnalités attendues des différents systèmes de portails

- Agrégation de contenu
  - Collecte d'informations sur des sources internes ou externes, puis présentation de ces données de façon unifiée.
- Organisation de contenu
  - Classement des données de l'entreprises.
- Personnalisation des services
  - Adapter l'accès aux informations par l'intermédiaire de profils utilisateurs
- Accès au contenu
  - Moteur de recherche et indexation

# Fonctionnalités attendues des différents systèmes de portails

- Information et diffusion du contenu
  - Service de diffusions sélectives d'informations ou d'alertes
- Communication et travail collaboratif
  - Mail, forum, agenda partagé, éditeur de documents collaboratifs
- Services à valeur ajoutée
  - Synthèse de documents évolués (analyse sémantique), traduction automatique
- Administration et sécurité
  - Contrôles des accès, annuaire d'entreprise, statistiques, pare-feu

# Fonctionnalités attendues des différents systèmes de portails

- Les différents produits fournissent un ou plusieurs de ces services.
- Avant de choisir un produit il faut avoir bien cerné les besoins de l'entreprise choisir un produit en fonction de ses priorités. Peu de produits fournissent tous types de services. Ou alors on peut utiliser des produits plus généraux qui nécessiteront le développement des différents services.

# Les différents produits de création de portails

- 2 types d'outils de création de portails.
  - Des infrastructures de portails. Elles fournissent les outils de bases à la création de tout types de portails.
  - Des portails beaucoup plus spécifiques souvent très spécialisés dans un domaine

# Différents types de portails

## Le portail web

- entrée unique sur un large panel de ressources et services dans un domaine et une communauté particulière
- accès généralement gratuit
- dans un logique web 2.0 chacun peut contribuer à l'enrichissement du portail (production de contenus, évaluation, commentaire, discussions, etc.).

## Le portail commercial

- site multiservice
- accès à des contenus et des services majoritairement payants (abonnements).

## Le portail institutionnel

- plate-forme intranet/extranet
- accès à des données de l'organisme
- accès des ressources du système d'information

# La métaphore du portail

## Un passage obligé pour entrer dans un endroit

## Le portail peut être fermé

- Ne rentre pas qui veut (contrôler)
- Périmètre fermé (se cacher)

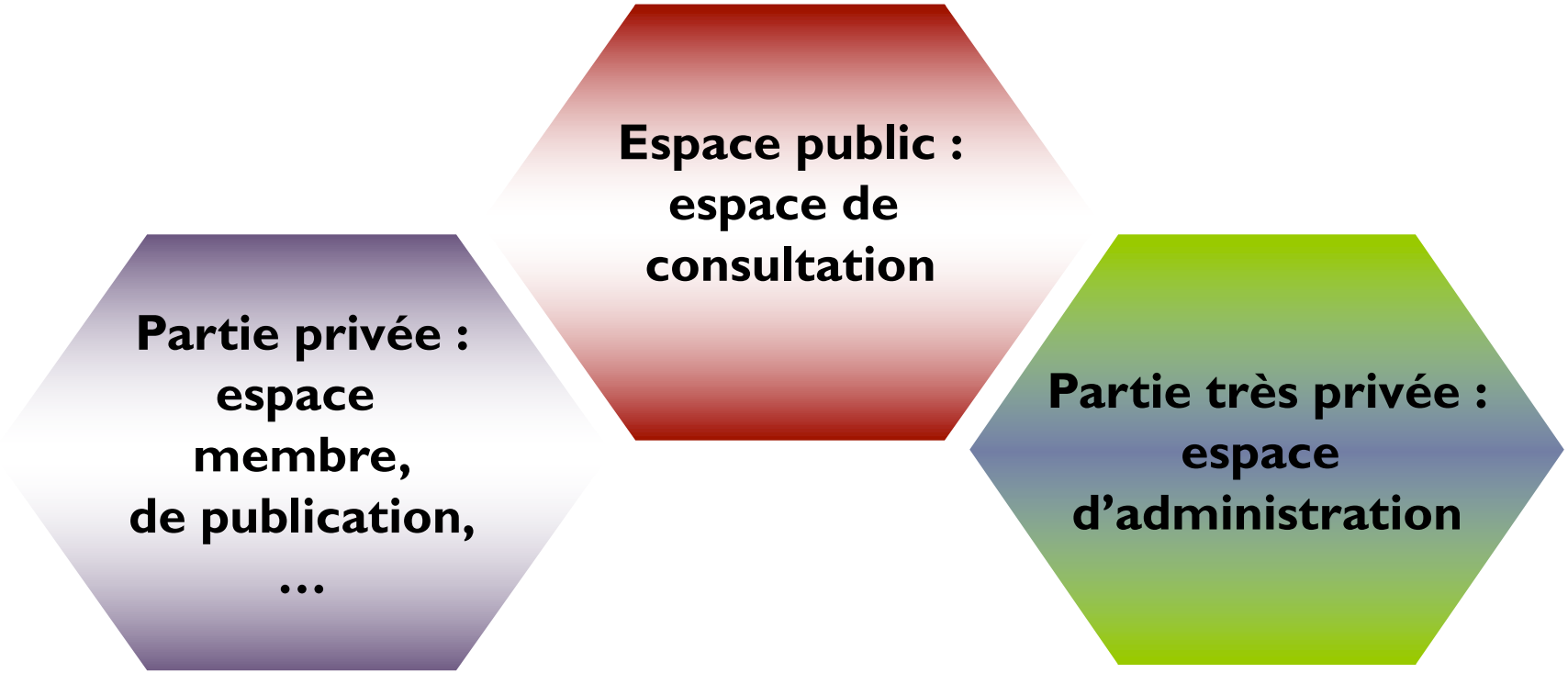
## Le portail peut être ouvert

- Informations importantes (nom, but, news, etc.) accessibles directement

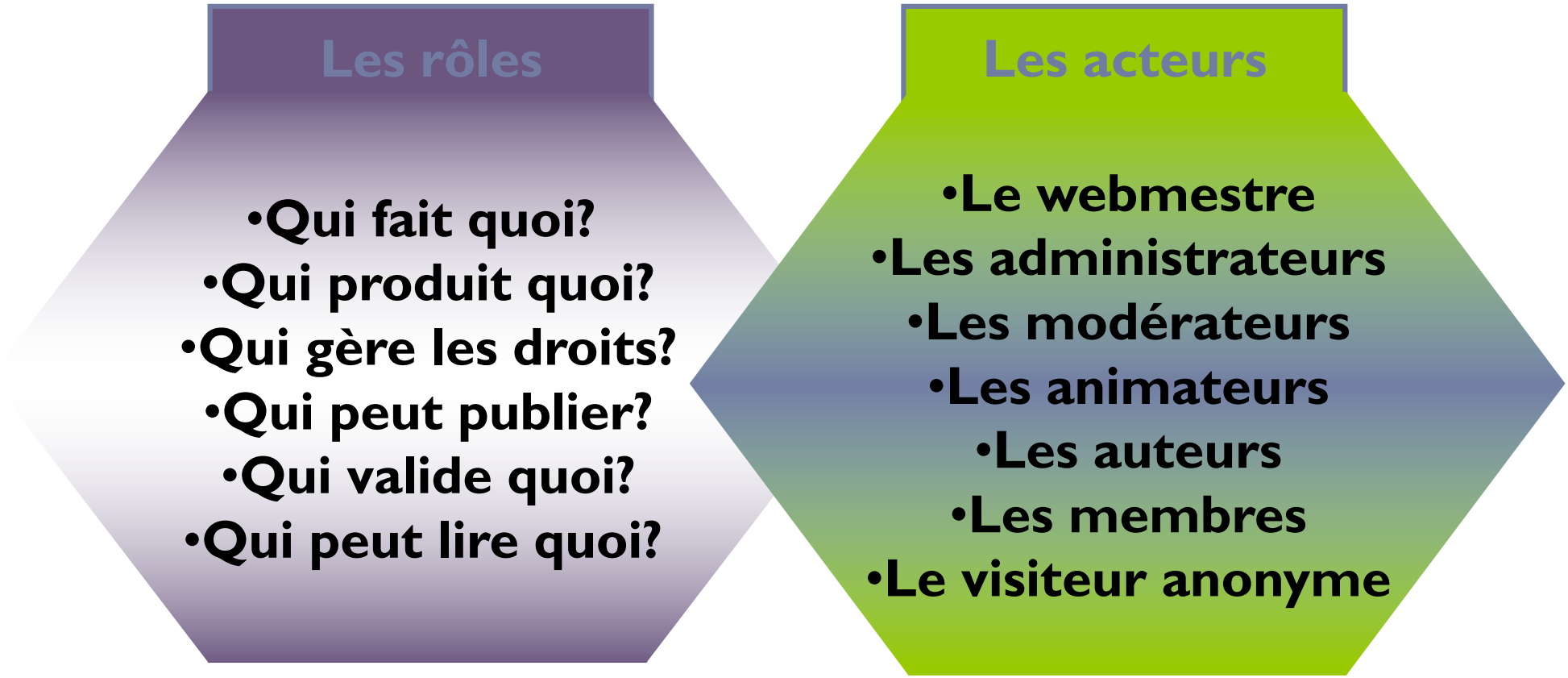
## Sans un portail, on a une dispersion :

- Informations/outils d'accès
- Quelle information est d'actualité? Où est-elle?
- Quel logiciel pour y accéder?

# Différents accès



# Différents rôles et acteurs



# Une production de contenu simplifiée

*Du « dur et solitaire » au « souple et communautaire »…*

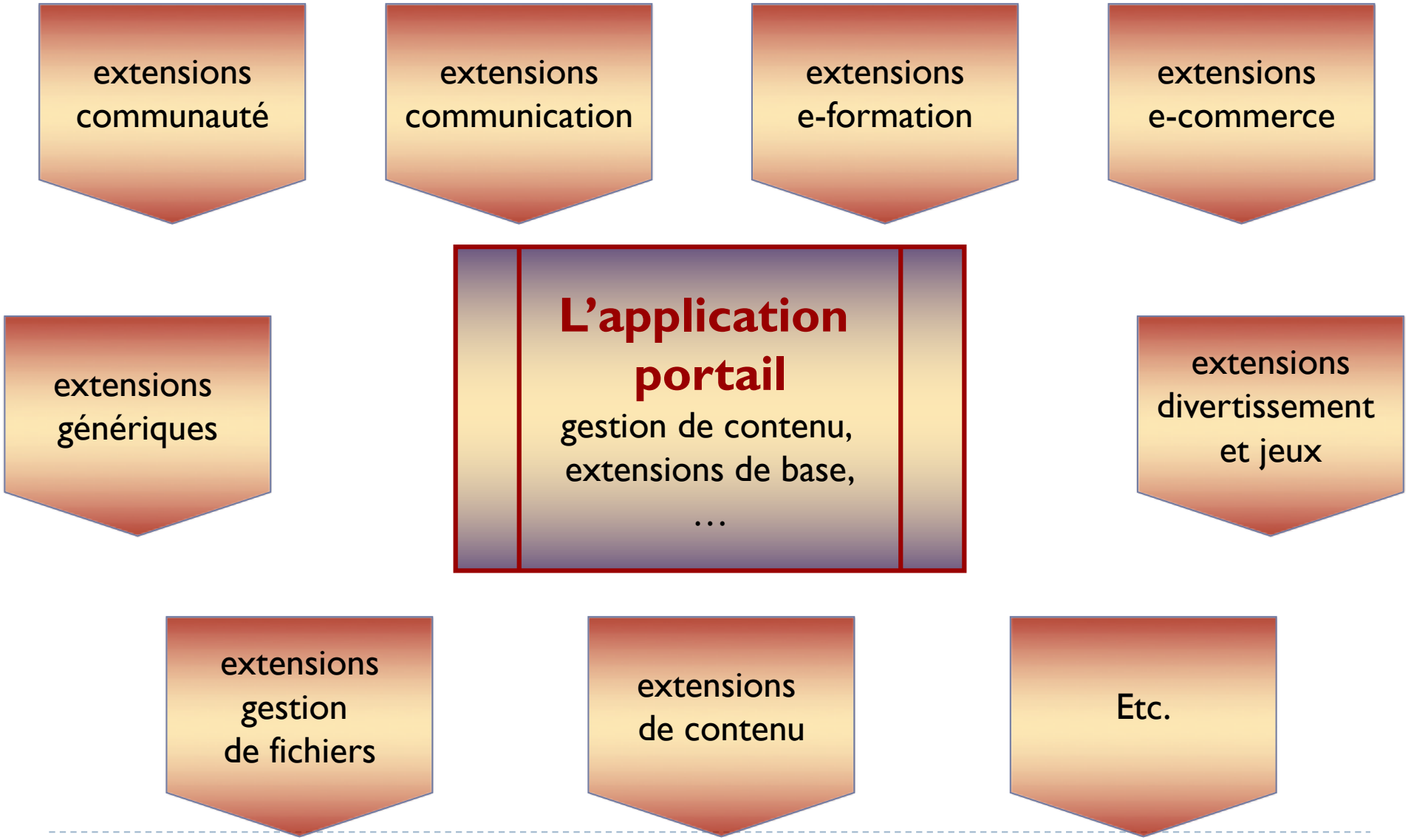
**Dans le temps : technologie en « dur » et l'homme-clef**

- 1 webmaster = homme-clef (technique + contenu)
- Des logiciels : Dreamweaver, Zend, …
- Développements informatiques complexes pour rajouter des composants

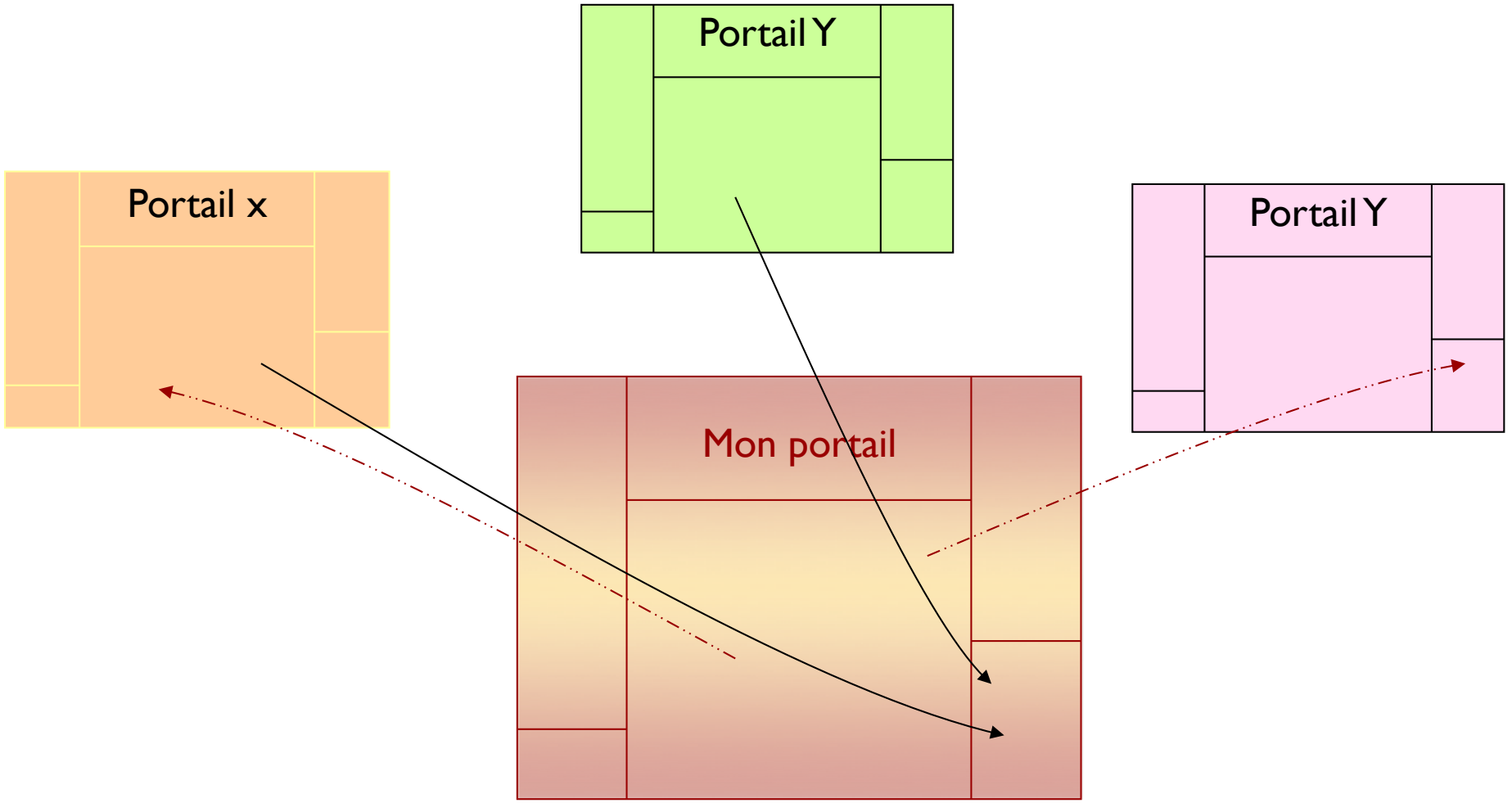
**Actuellement : gestion d'un portail accessible à des non informaticiens (sauf pour la sécurité) par une équipe d'acteurs**

- 1 webmaster éditorial, des administrateurs, des auteurs contributeurs
- Publication décentralisée simple sans logiciel à installer
- Facilité d'ajout d'extensions dans le portail

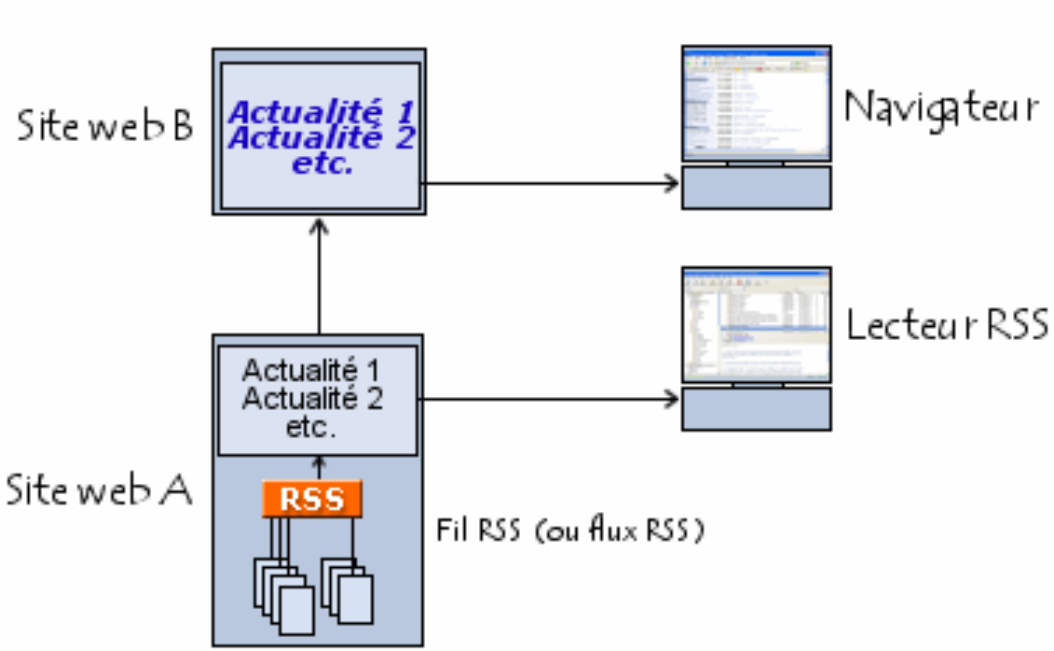
# Principe modulaire du portail



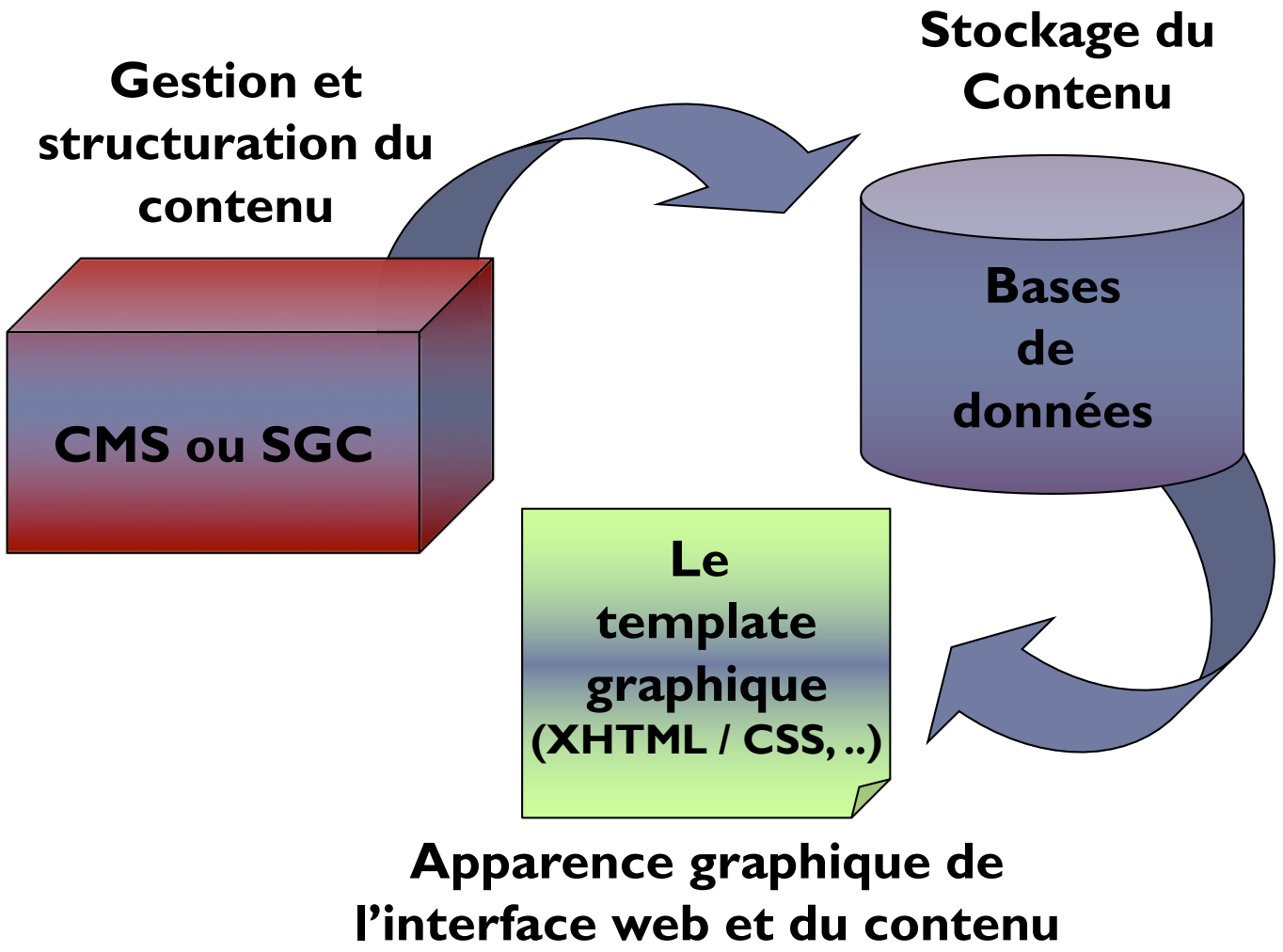
# Syndication des contenus



# Syndication des contenus



# Séparation contenu et présentation – principes de base



# Séparation contenu et présentation – les templates

**Le template graphique d'un portail est un ensemble de fichiers**

- qui structure et positionne visuellement les éléments des pages (php, html, …)
- qui décrivent la forme graphique de ces éléments (feuilles de style css)
- qui enrichissent et personnalisent le design (les images)

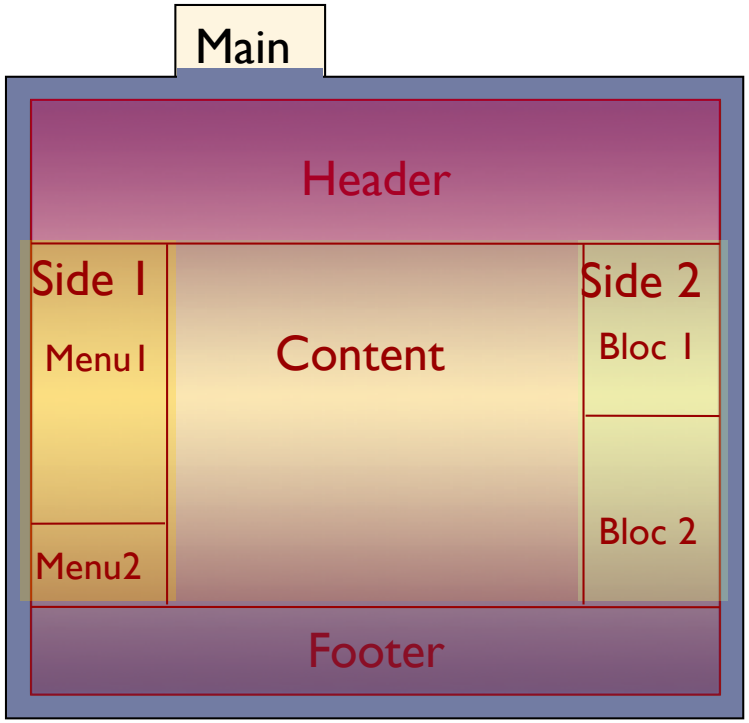
**et qui permettent la cohérence visuelle du portail.**

**On peut changer l'apparence d'un portail sans changer le contenu, en changeant le template.**

- Exemple : Theme Forest !

# Séparation contenu et présentation – Codage du template

**Structure visuelle de la page et des zones**




**Extrait du code XHTML de la page web**

```
<div id="Main">
<div id="Header"></div>
<div id="Side 1">
<div id="Menu 1"></div>
<div id="Menu 2"></div>
</div>
<div id="Content"></div>
<div id="Side 2">
<div id="Bloc 1"></div>
<div id="Bloc 2"></div>
</div>
<div id="Footer"></div>
</div>
```

**Extrait de la feuille de style**

```
#Main{
width:970px;
text-align:left;
margin-left:auto;
margin-right:auto;
background: transparent
url(../images/bg.gif) repeat-y ;
margin-bottom: 20px;
}

#header {
height: 116px;
width: 100%;
 }

…
```

# Design de portails

- **Identification rapide du portail et de sa thématique** (logo, base line, nom)
- **Un design épuré voir minimaliste**
- **Un découpage en bandeau, colonnes et blocs.**
- **Priorité au contenu**
- **Une navigation principale par blocs de contenus + une navigation classique par menu**
- **Une page d'accueil avec de multiples entrées et une mise en avant des actualités**
- **Cohérence graphique** (grâce au template)

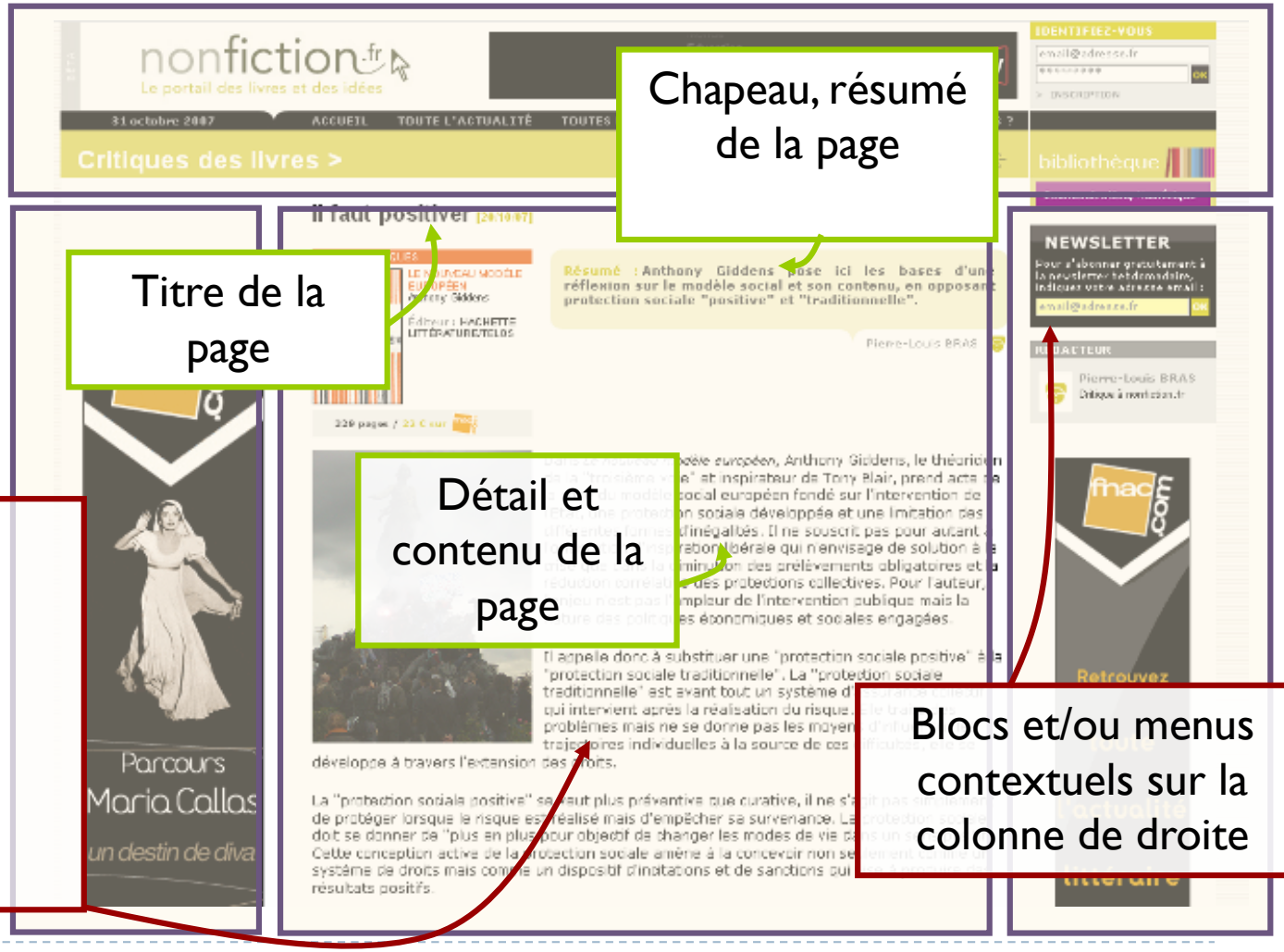
# Design de portails –
## page d'accueil

# Design de portails –
## page de contenu

Découpage global
en bandeau
(logo + menu)
et colonnes

Colonne centrale
épurée pour
un confort de lecture.

zone très textuelle



# Design de portails –
## Responsive Design

http://www.adamkaplan.me/grid/

# Écriture web dans les portails

- **Les règles linguistiques et typographiques**
- **Les procédés stylistiques**
  - Styles et techniques rhétoriques
  - Modes de lecture : recherche ou consommation
  - Lecture à l'écran
- **La pyramide inversée**
  - Placer le message principal en amont et détailler ensuite
    Ex :Titre, chapo, développement
  - Un paragraphe = une idée
  - Effet pyramide grâce à la navigation hypertexte
- **La règle des 5 « W »**
  - Who? What? When? Where? Why?
- **Les guidelines**
  - Graphiques : cohérence grâce aux feuilles de style
  - Éditoriales : cohérence de la ligne éditoriale et aussi de chaque type de contenu
- **Visiter le site :** http://www.ecrirepourleweb.com

# Accessibilité Web

## Accessibilité Web aux handicapés

- 5% des internautes concernés par une situation handicapante générant des difficultés quotidiennes tant physiques que sensorielles, intellectuelles ou mentales
- 2 % d'internautes utilisent des aides techniques spécialisées pour la consultation et l'usage d'Internet.
- Des directives publiées par la WAI du W3C pour rendre les sites accessibles
- égalité des chances ?

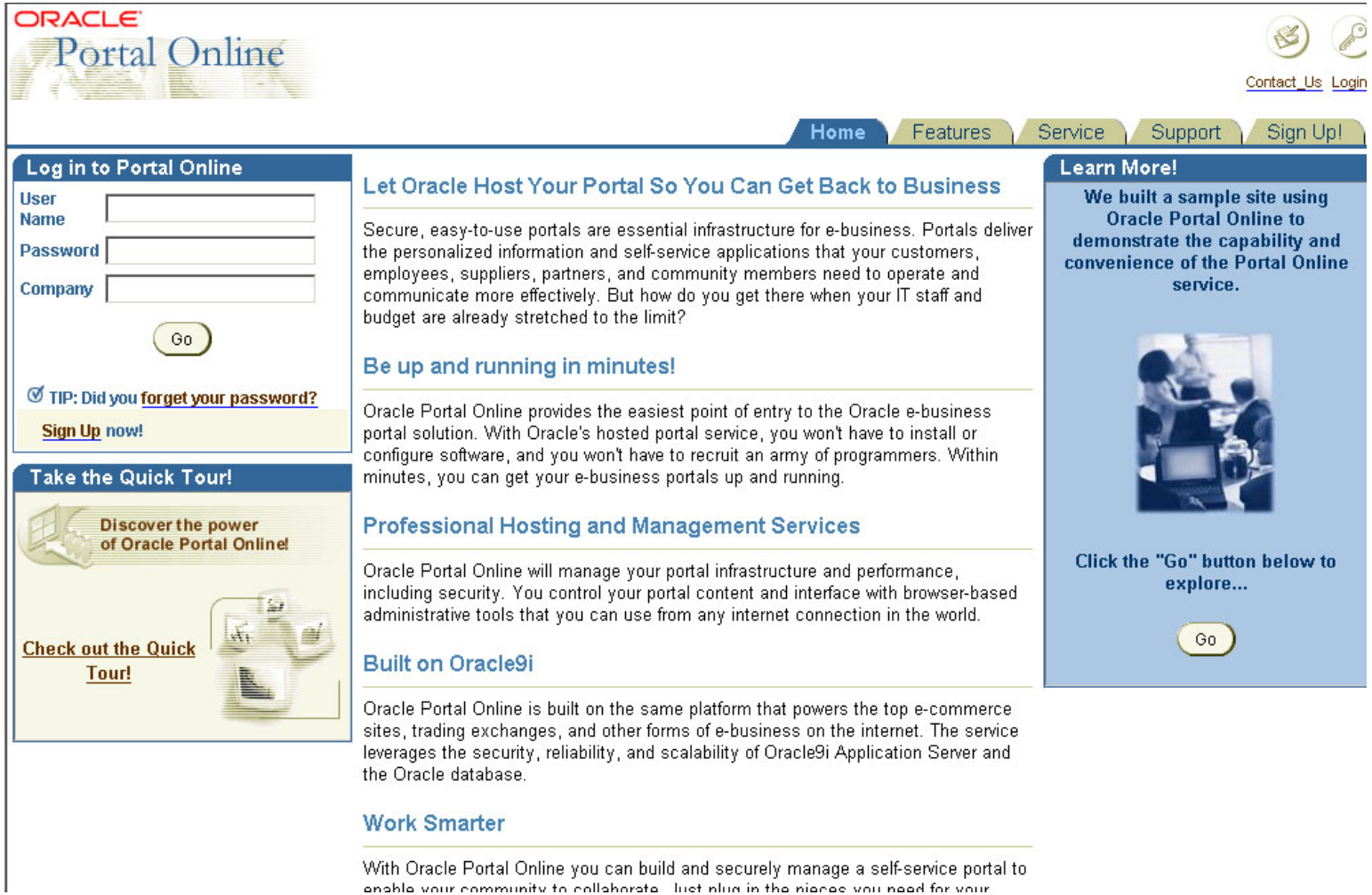
## Accessibilité pour tous et pour tous les environnements

- OS, navigateurs, PDA, smartphones, logiciel vocal pour handicapés (text to speech / speech to text), etc.…

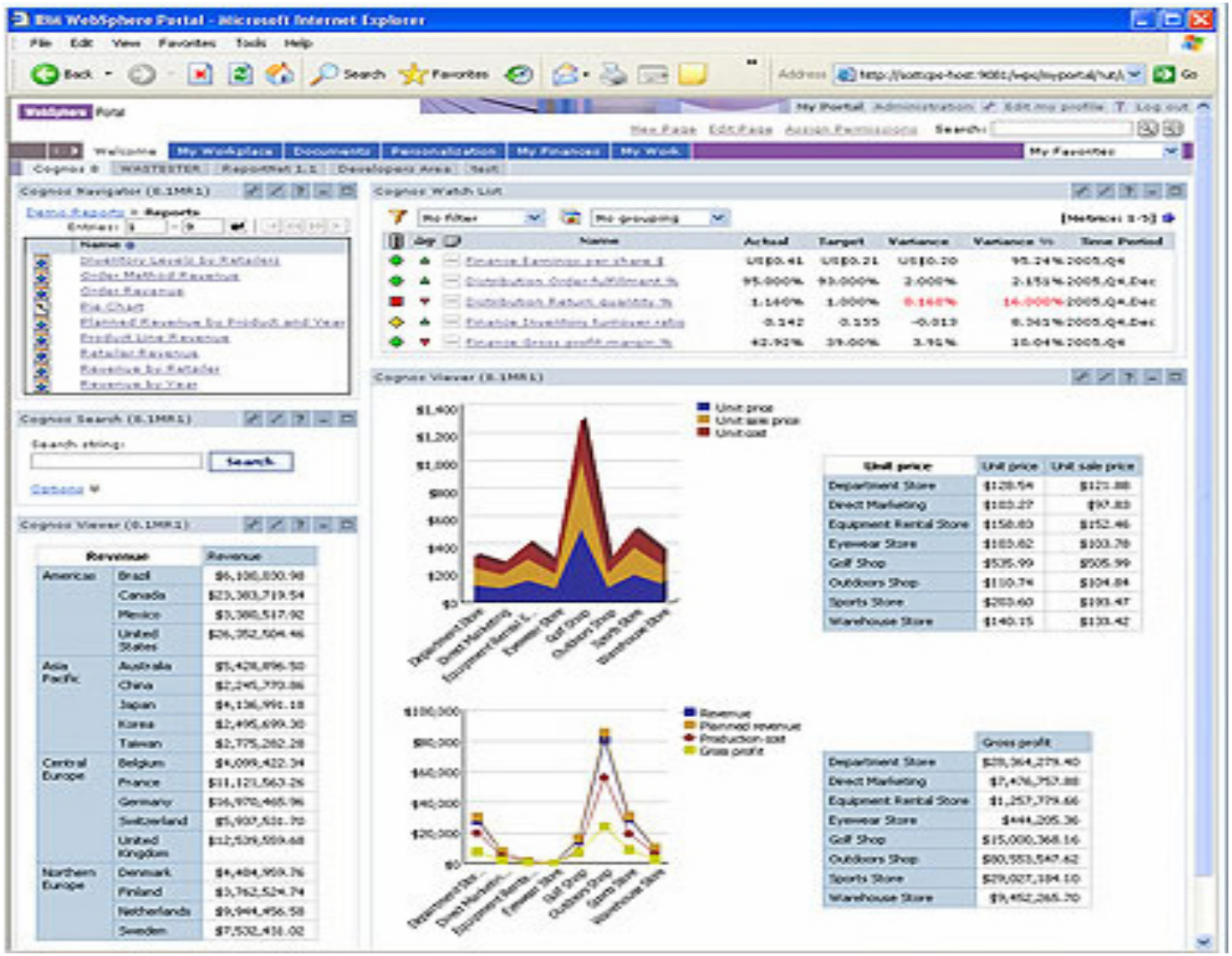
# Exemple de portails et démonstration

- Wikipedia une encyclopédie portail
- Nonfiction le portail des livres et des idées

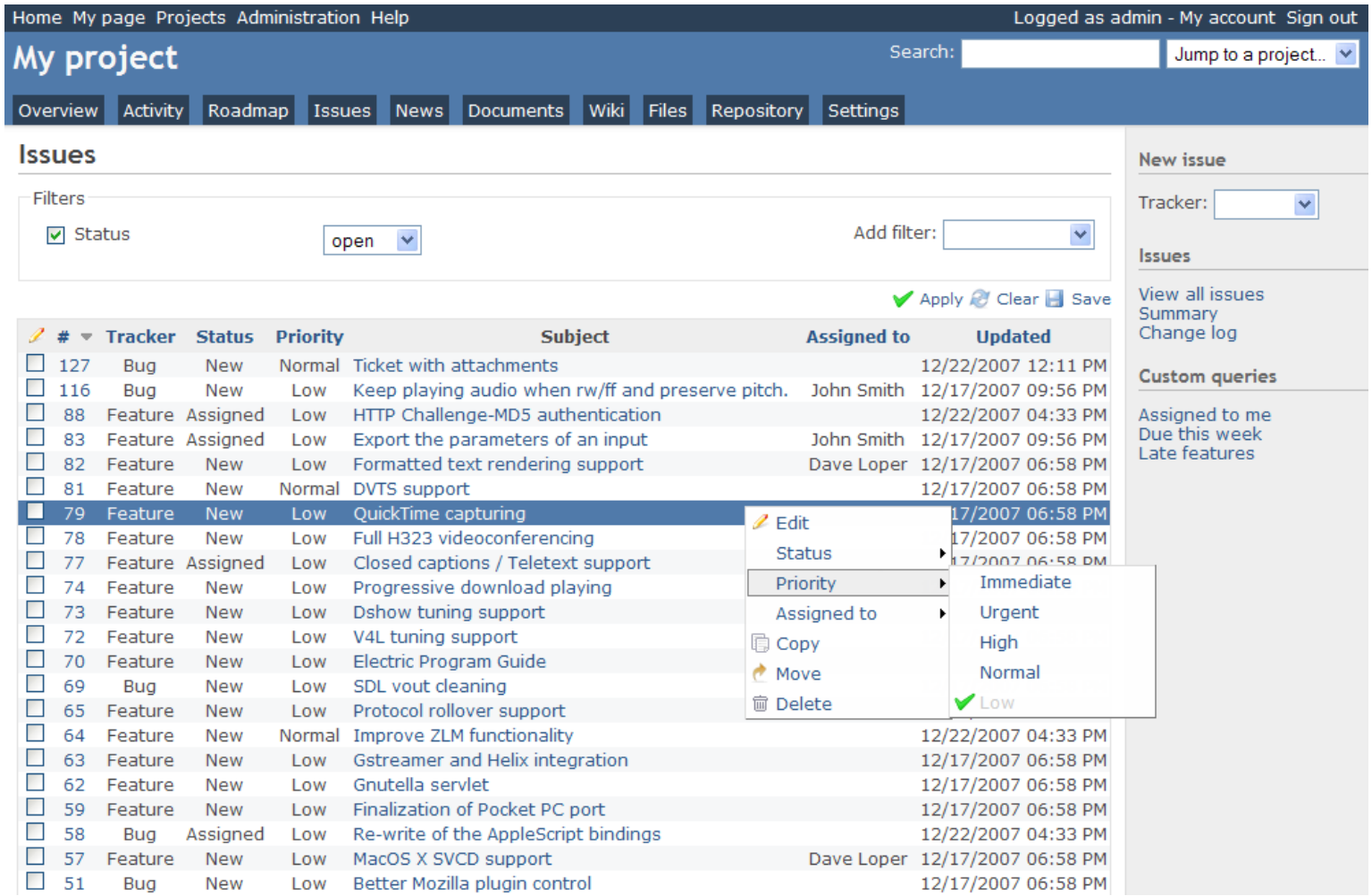
# Exemples de portails
# **Oracle Portal Online**



# Exemples de portails
# **Websphere**



# Exemples de portails
# **Redmine**



# Exemples de portails
# **Sharepoint**

# XML

# XML

- ***Extensible Markup Language*** : langage extensible de balisage.

- Reconnu comme recommandation par le W3C (World Wide Web Consortium) :
  - XML 1.0 depuis 1998
  - XML 1.1 depuis 2004
- XML 2.0 ??
- XML est une méthode pour représenter les données. Celles-ci sont écrites entre des balises ou sous forme d'attributs, et l'ensemble est écrit sous forme d'un arbre.

# XML

- Dans un document **XML**, la mise en forme des données (le contenant) est totalement séparée des données (le contenu).
  → plusieurs types de sortie pour un même fichier de données, en fonction de l'utilisateur ou de l'application demandeuse (autre document **XML**, tableau, graphique, image, animation multimédia, fichier **HTML**, fichier **PDF**...).

- Possibilité de créer les éléments que l'on désire permet de rendre le fichier lui-même lisible -et modifiable- par un être humain : on peut donner aux informations contenues dans un tel fichier les étiquettes que l'on veut, et les ordonner selon son désir.

# XML

- XML est un langage HTML amélioré.
- Contrairement à HTML, qui est à considérer comme un langage défini et figé (avec un nombre de balises limité), XML peut être considéré comme un **métalangage** permettant de définir d'autres langages
- XML est un format de description des données (contenu) et non de leur représentation, comme c'est le cas avec HTML.

# XML

Un fichier **XML** est composé de :

1. **Un prologue :** constitué de la déclaration **XML**, puis éventuellement d'une déclaration de type de document (une **DTD**).
2. **Un élément racine :** où on trouve les éléments.
3. **Un arbre :** constitué d'éléments imbriqués les uns dans les autres (ayant une relation parent-enfant) et d'éléments adjacents.

# Exemple :

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- La ligne ci-dessus est le prologue -->
<!-- Élément racine -->
<biblio>
  <!-- Premier enfant -->
  <livre>
    <!-- Élément enfant titre -->
    <titre>Les Misérables</titre>
    <auteur>Victor Hugo</auteur>
    <nb_tomes>3</nb_tomes>
  </livre>
  <livre>
    <titre>L'Assomoir</titre>
    <auteur>Émile Zola</auteur>
  </livre>
  <livre lang="en">
    <titre>David Copperfield</titre>
    <auteur>Charles Dickens</auteur>
    <nb_tomes>3</nb_tomes>
  </livre>
</biblio>
```

# 1) Le Prologue

## 1.1) Déclaration XML

Déclaration facultative, fait partie des « instructions de traitement ». Exemple de déclaration **XML** :

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

- **version** : version du **XML** utilisée dans le document;
- **encoding** : le jeu de codage de caractères utilisé. Par défaut, l'attribut **encoding** a la valeur **UTF-8**.
- **standalone** : dépendance du document par rapport à une DTD. Si la valeur est **yes**, le processeur de l'application n'attend aucune déclaration de type de document extérieure au document. Sinon, le processeur attend une référence de déclaration de type de document. La valeur par défaut est **no**.

## 1.2) Instructions de traitement

Instruction interprétée par l'application servant à traiter le document **XML**. Elle ne fait pas totalement partie du document.

Exemple d'instruction de traitement :

```
<?xml-stylesheet type="text/xsl" href="biblio.xsl"?>
```

- L'application = xml-stylesheet
- Deux feuilles de style différentes peuvent être utilisées, les **XSL** (propres au **XML**) ainsi que les **CSS** (feuilles de style apparues avec le **HTML**).
- L'attribut **href** = l'URL du fichier; utilisée par les navigateurs Internet pour la mise en forme du document.

## 1.3) Déclaration de type de document (DTD)

- permet de définir la structure du document.
- peut être externe (fichier à part : permet de partager la DTD entre plusieurs documents **XML**.)ou interne (incorporée au document **XML)**. L'autre type de document permettant de définir la structure d'un fichier, le schéma XML.

Exemple de déclaration de type de document externe :

```
<!DOCTYPE biblio SYSTEM "biblio.dtd">
```

→ définit l'ensemble des éléments utilisables dans le document, y compris l'élément-racine (ici **biblio**) ainsi que l'emplacement où se trouve le fichier **biblio.dtd** dans lequel se trouve définie la structure du document.

→ Utile pour vérifier la validité du document **XML**.

# 2) Les commentaires

Idem que pour **HTML**.

Ils commencent par **<!--** et se terminent par **-->**.

Ils peuvent être placés à n'importe quel endroit tant qu'ils se trouvent à l'extérieur d'une autre balise.

Exemples de commentaires valides :

```
<!-- ceci est correct -->
<elt> <!-- ceci est correct aussi -->
Un peu de texte </elt>
```

# 3) L'arbre d'éléments

Un document **XML** peut se représenter sous la forme d'une arborescence d'*éléments*. Cette arborescence comporte une racine (unique), des branches et des feuilles. Reprenons l'exemple précédent.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- La ligne ci-dessus est le prologue -->
<!-- Élément racine -->
<biblio>
  <!-- Premier enfant -->
  <livre>
    <!-- Élément enfant titre -->
    <titre>Les Misérables</titre>
    <auteur>Victor Hugo</auteur>
    <nb_tomes>3</nb_tomes>
  </livre>
  <livre>
    <titre>L'Assomoir</titre>
    <auteur>Émile Zola</auteur>
  </livre>
  <livre lang="en">
    <titre>David Copperfield</titre>
    <auteur>Charles Dickens</auteur>
    <nb_tomes>3</nb_tomes>
  </livre>
</biblio>
```

## 1.1) Élément racine

- base du document **XML**.
- unique et englobe tous les autres éléments.
- s'ouvre juste après le prologue, et se ferme à la toute fin du document.

Dans l'exemple, l'élément racine est **biblio** :

```
<biblio>
</biblio>
```

## 1.2) Les éléments

Ils forment la structure même du document : ce sont les branches et les feuilles de l'arborescence.

- peuvent contenir du texte :

```
<titre>Les Misérables</titre>
```

- peuvent contenir d'autres éléments, appelés éléments enfants:

```
<livre>
<titre>L'Assommoir</titre>
<auteur>Émile Zola</auteur>
<couverture couleur="rouge" />
 </livre>
```

- peuvent être vide, ils ne contiennent pas d'élément-enfant:

```
<couverture couleur="rouge" />
```

## 1.3) Les attributs

- Tous les éléments peuvent contenir un ou plusieurs attributs.
- Chaque élément ne peut contenir qu'une fois le même attribut.
- Un attribut est composé d'un nom et d'une valeur.
- Il ne peut être présent que dans la balise *ouvrante* de l'élément (par exemple, on n'a pas le droit d'écrire **</livre lang="en">**).

- Exemple d'utilisation d'un élément avec attribut :

```
<instrument type="vent">trompette</instrument>
```

- Exemple d'utilisation d'un élément vide avec attributs :

```
<img src="ours.png" alt="Gros ours" width="56"
height="100" />
```

## 1.3) Les entités

Il existe des entités définissables et définies. Elles peuvent être analysables ou non, internes ou externes. La déclaration des entités s'effectue au sein de la **DTD**. Elles peuvent être utilisées aussi bien dans la **DTD** que dans le document **XML**.

Certains caractères ayant un sens précis en **XML**, il est nécessaire de leur trouver un remplaçant lorsque l'on a besoin de les insérer dans un document. On a recours dans ce cas à des entités prédéfinies.

Ces entités sont :

| Caractère | Entités |
|---|---|
| & | &amp; |
| < | &lt; |
| > | &gt; |
| " | &quot; |
| ‘ | &aquot; |

# 4) Règles de composition

- ✓ Un nom d'élément ne peut commencer par un chiffre.
- ✓ Si le nom n'est composé que d'un seul caractère, ce doit être une lettre.
- ✓ S'il est composé d'au moins deux caractères, le premier peut être « _ » ou « : ». Le nom peut ensuite être composé de lettres, chiffres, tirets, tirets bas et deux points. La syntaxe **XML** est sensible à la casse (le format distingue majuscules et minuscules).
- ✓ Toutes les balises portant un contenu non vide doivent être fermées. La balise de début, la balise de terminaison et le contenu entre deux sont globalement appelés élément ;
- ✓ Les balises n'ayant pas de contenu doivent se terminer par /> (voir la balise **<img .../>** ci-dessus) ;
- ✓ Les valeurs d'attributs doivent être entre *guillemets* ;

# Document Type Definition

- Pour associer un document XML à une DTD, il faut le faire avec l'élément <!DOCTYPE>
  Ex : <!DOCTYPE livre SYSTEM "livre.dtd" >
- On considère qu'un document XML est constitué de deux choses : La DTD et la donnée.
- Un document XML bien formé peut ne pas être composé de DTD
- Un document XML valide doit avoir une DTD

# Data Type Definition

- Une DTD est définie dans l'élément
  <!DOCTYPE name SYSTEM "fichier.dtd" [delaration] >
  - name est le nom du type de document
  - SYSTEM "fichier.dtd" est une référence vers le fichier.dtd. Ce fichier contient tout ou une partie de la DTD. C'est la partie externe de la DTD
  - [declaration] permet de définir tout ou une partie de la DTD. C 'est la partie interne de la DTD

# Contenu d'une DTD

- Déclarations des entités
- Déclarations des éléments et de leurs contenus
- Déclarations des attributs des éléments
- Commentaires

# Exemple de DTD

→ Fichier XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
   <!DOCTYPE annuaire SYSTEM "annuaire.dtd">
   <annuaire>
      <personne type="étudiant">
         <nom>BELAID</nom>
         <prenom>Chokri</prenom>
         <email>quiatue@chokri.com</email>
      </personne>
      <personne type="chanteur">
         <nom>MIGALO</nom>
         <prenom>HABIB</prenom>
         <email>seyeskhouk@mosaiquefm.net</email>
      </personne>
   </annuaire>
```

# Exemple de DTD

→ DTD

```
<?xml version="1.1" encoding="ISO-8859-1"?>
    <!ELEMENT annuaire     (personne*)>
    <!ELEMENT personne  (nom,prenom,email+)>
    <!ATTLIST personne type (étudiant | professeur | chanteur | musicien) "étudiant">
    <!ELEMENT nom       (#PCDATA)>
    <!ELEMENT prenom    (#PCDATA)>
    <!ELEMENT email (#PCDATA)>
```

# Exemple DTD Interne

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE DOCUMENT [
<!ELEMENT DOCUMENT(PERSONNE*)>
<!ELEMENT PERSONNE (#PCDATA)>
<!ATTLIST PERSONNE PNUM ID #REQUIRED>
<!ATTLIST PERSONNE MERE IDREF #IMPLIED>
<!ATTLIST PERSONNE PERE  IDREF #IMPLIED>
]>
<DOCUMENT>
    <PERSONNE PNUM = "P1">Marie</PERSONNE>
    <PERSONNE PNUM = "P2">Jean</PERSONNE>
    <PERSONNE PNUM = "P3" MERE="P1" PERE="P2">Pierre</PERSONNE>
    <PERSONNE PNUM = "P4" MERE="P1" PERE="P2">Julie</PERSONNE>
</DOCUMENT>
```

# Les entités

- Une entité est un couple (nom, valeur). On référence le nom pour utiliser la valeur. Ex : & (amp,&)
- Une entité peut être interne (définie dans le document) ou externe (définie ailleurs)
- Les entités internes ont des valeurs de chaînes de caractères XML et sont utilisées comme moyen d'abréviation.
- Les entités externes peuvent être de type
  - XML : utilisé pour segmenter un document en parties
  - non XML : utilisé pour inclure des sons, images ...

# Déclaration des entités

- Les entités doivent être déclarées dans la DTD
- Un document peut référencer n'importe quelle entité qui a été déclaré dans la DTD mais aussi les entités prédéfinis (amp, apos, …)
- La déclaration d'une entité se fait de la manière suivante : <!ENTITY nom valeur>
  - La valeur doit être entre guillemets

# Les entités paramètres

- Ces entités sont définies dans une DTD mais elles sont aussi utilisées dans une DTD
- Une entités paramètre est forcement XML
- Une entités paramètre se définie comme une entité générale si ce n'est qu'il y a le caractère %.
  Ex : <!ENTITY % nom valeur>
- Pour référencer une entité paramètre on utilise le caractère %
  Ex : %nom;

# Les entités paramètres

- <!ENTITY % nom "chaîne_de_remplacement">
- Ces entités apparaissent uniquement dans les DTD. Ce sont des raccourcis vers des parties de déclarations de la DTD.
- Par exemple dans la DTD de XHML1.0 on a la déclaration d'entité :
  <!ENTITY % Shape "(rect|circle|poly|default)">
- qui est ensuite utilisée dans la déclaration d'attributs pour l'élément area :

```
<!ATTLIST area
  %attrs;
  shape         %Shape;         "rect"
  coords        %Coords;        #IMPLIED
  href          %URI;           #IMPLIED
  nohref        (nohref)        #IMPLIED
  alt           %Text;          #REQUIRED
  tabindex      %Number;        #IMPLIED
  accesskey     %Character;     #IMPLIED
  onfocus       %Script;        #IMPLIED
  onblur        %Script;        #IMPLIED
  >
```

# Les entités caractères

- <!ENTITY nom "&#code;">
- Ces entités associent un alias à un caractère non disponible sur le clavier, dans le document l'appel s'effectue avec&nom;.
- Rappelons que les caractères réservées &, <, >, ", ', sont déjà définis et n'ont pas à être redéfinis par des entités dans la DTD
- La DTD XHTML1.0 définit le caractère mathématique "il existe" et le caractère de l'alphabet grec "delta" :
- <!ENTITY exist "∃">

# Les entités générales

- <!ENTITY nom "texte">
- Transforme l'alias (nom) en une phrase incluse dans la déclaration de l'entité. C'est l'équivalent d'une macro.
- Par exemple <!ENTITY wysiwyg "What You See Is What You Get">
- permet d'utiliser dans le document XML :les éditeurs de type &wysiwyg;
- ce qui deviendra dans un navigateur : les éditeurs de type What You See Is What You Get.

# Les entités externes

- Les entités externes représentent des données contenues dans des fichiers séparés par rapport au document XML.

# Les entités externes Analysées

- <!ENTITY nom SYSTEM "URI">
- Les fichiers contenant des données textuelles sont analysés. Par exemple un livre peut être constitué des documents toc.xml, chapters/c1.xml, chapters/c2.xml et index.xml :

```
<!doctype book SYSTEM "book.dtd"
[
    <!entity toc SYSTEM "toc.xml">
    <!entity chap1 SYSTEM "chapters/c1.xml">
    <!entity chap2 SYSTEM "chapters/c2.xml">
    <!entity index SYSTEM "index.xml">
]>
<book>
    <head>&toc;</head>
    <body>
        &chap1;
        &chap2;
        &index;
    </body>
</book>
```

# Les entités externes non Analysées

- <!ENTITY nom SYSTEM "URI" NDATA type_notation>
- Les entités externes non analysées permettent de déclarer un contenu non XML dans le document XML (fichiers binaires images, sons...).
- Le mot clé NDATA (Notation DATA) précise le type d'entité non analysée que le processeur XML doit traiter.
- Par exemple, la déclaration d'entité ci-dessous attribue un alias (vacances) à une entité (images/plage.gif) de type GIF89a sauvegardée sur le disque (SYSTEM) :
- <!ENTITY vacances SYSTEM "images/plage.gif" NDATA GIF89a>
- Les fichiers binaires (images, sons...) doivent être identifiés par une notation qui a été déclarée dans la DTD. Par exemple, la notation GIF89a utilisée dans l'exemple précédent est déclarée dans la DTD :
  <!NOTATION GIF89a PUBLIC "-//Compuserve//NOTATION Graphics Interchange Format 89a//EN">

# Les entités externes non Analysées

- Les entités binaires ne peuvent être appelées que dans la valeur d'un attribut, lui-même déclaré comme étant du type ENTITY ou ENTITIES dans la DTD. La DTD comportera donc :
- <!NOTATION GIF89a PUBLIC "-//Compuserve//NOTATION Graphics Interchange Format 89a//EN">
- <!ATTLIST image source ENTITY #REQUIRED>
- <!ENTITY vacances SYSTEM "images/plage.gif" NDATA GIF89a>
- Ce qui permettra dans le document XML d'appeler l'image :
- <image source="vacances">

# Les éléments dans la DTD

- La déclaration d'un élément se fait de la manière suivante :
  <!ELEMENT nom modèle>
- Le nom correspond au nom de l'élément
- Le modèle défini le contenu de l'élément :
  - éléments fils
  - données
  - mélanges éléments fils et données
  - n'importe quoi respectant la syntaxe XML
  - élément vide

# Les éléments dans la DTD

- Pour décrire les éléments fils d'un élément, il suffit de donner leurs noms entre parenthèses.
- Si l'ordre des élément fils est important on les sépare par des virgules sinon par des barres verticales.
- Pour préciser les multiplicités des éléments fils on utilise : (soit fils le nom de l'élément fils) :
  - fils = une fois et une seule
  - fils? = 0 ou 1 fois
  - fils* = 0, 1 ou plus d'une fois
  - fils+ = au moins un fois
- il est possible de faire des groupes d'éléments fils en utilisant des parenthèses.

# Les éléments dans la DTD

- Exemples de définition d'élément :
  <!ELEMENT livre (titre, intro, chapitre+)>
  <!ELEMENT chapitre (titre , (titre_partie, partie)+)>
  ...

# Les éléments dans la DTD

- Lorsqu'un élément ne contient que des données (i.e. : une chaîne de caractère), on le défini de cette manière avec le mot clé : #PCDATA
- Ex :
  <!ELEMENT partie (#PCDATA)>

# Les éléments dans la DTD

- Il est possible qu'un élément ait des éléments fils mais aussi des données. Il est tout à fait possible de définir ce type d'élément dans une DTD.
- Ex :
  <!ELEMENT toto (#PCDATA, titi, tutu)*>

# Les éléments dans la DTD

- Lorsqu'un élément peut contenir soit du texte, soit n'importe quel ensemble d'élément fils, il faut utiliser le mot clé ANY
  Ex : <!ELEMENT elmt ANY>
- Lorsqu'un élément est un élément vide, il faut utiliser le mot clé EMPTY
  Ex : <!ELEMENT elmt EMPTY>

# Les attributs dans la DTD

- La déclaration des attributs d'un élément se fait de la manière suivante :
  <!ATTLIST element (attribut type default)+>
- element correspond au nom de l'élément qui contient l'attribut
- attribut correspond au nom de l'attribut
- type correspond au type de l'attribut
- default contient des indications sur la valeur par défaut

# Les attributs dans la DTD

- Le type de l'attribut peut être :
  - CDATA pour une chaîne de caractère
  - ID ou IDREF pour des références internes au document
  - ENTITY ou ENTITIES pour des références externes non-XML
  - NOTATION pour associer le contenu de l'élément à une application

# Les attributs dans la DTD

- L'indication sur la valeur par défaut peut être :
  - #REQUIRED indique que l'attribut doit avoir une valeur une fois le document créé.
  - #IMPLIED indique que l'attribut est optionnel.
  - #FIXED 'valeur' indique que l'attribut est une constante

# Les attributs dans la DTD

- Quelques exemples :

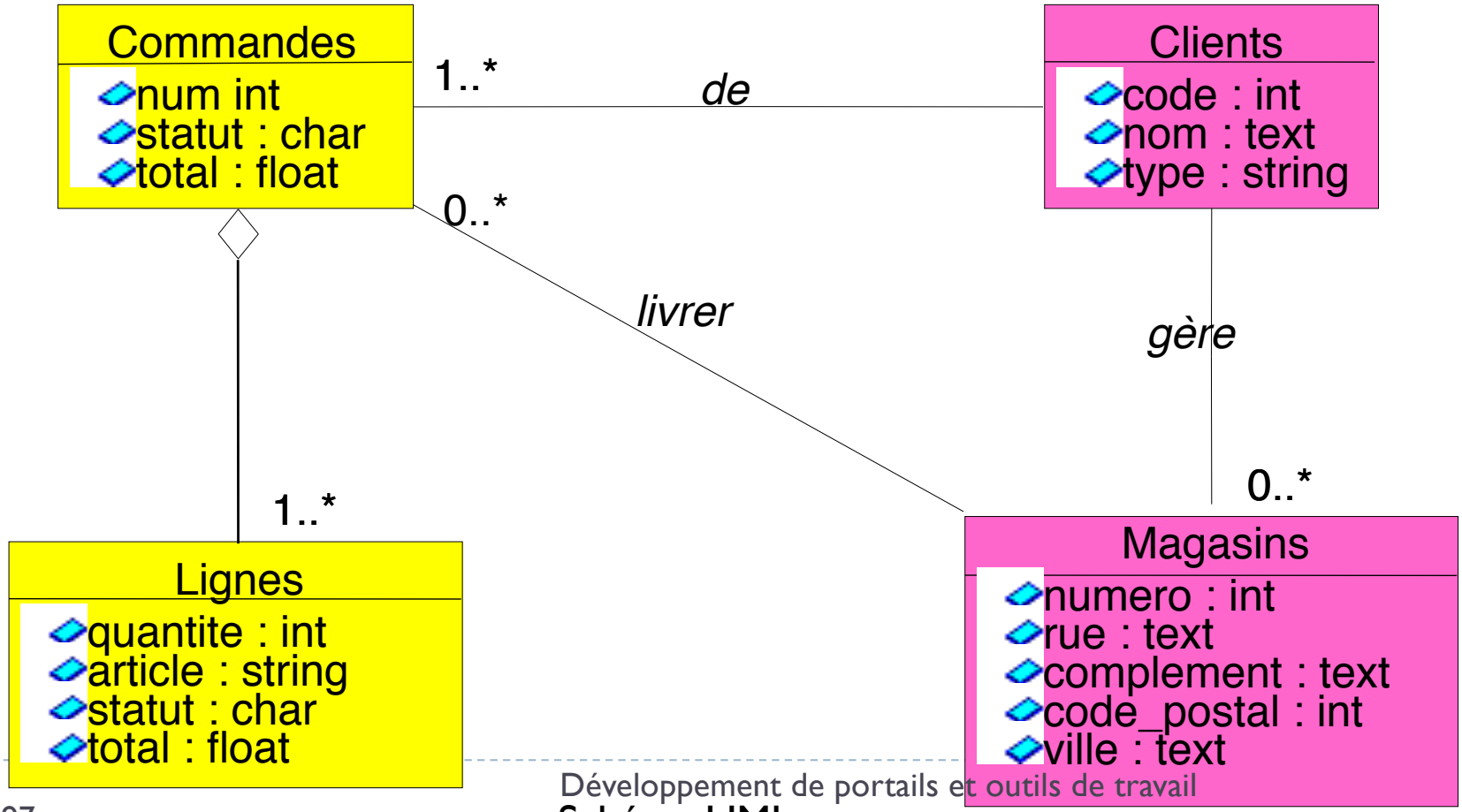
```
<!ATTLIST ex1 att1 CDATA #IMPLIED>
<!ATTLIST ex2 att1 CDATA #IMPLIED
              att2 ENTITY #REQUIRED>
...
```

# Sections conditionnelles

- <![ INCLUDE [ *contenu* ]]> fait que le *contenu* appartient à la DTD
- <![IGNORE [ *contenu* ]]> fait que le *contenu* n'appartient pas à la DTD
- En utilisant des entités paramètres et en les redéfinissant dans le document, il est alors possible d'avoir une DTD conditionnelle

# Exercice

- Définir les DTD pour publier une BD



# Définition des types et classes

```
<!-- Types de base-->
<!ENTITY % int "(#PCDATA)">
<!ENTITY % float "(#PCDATA)">
<!ENTITY % char "(#PCDATA)">
<!ENTITY % string "(#PCDATA)">

<!-- Classe Commande -->
<!ELEMENT Commande (cstatut, ctotal)>
<!ATTLIST Commande NUM ID #REQUIRED>
<!ELEMENT cstatut %char;>
<!ELEMENT ctotal %float;>

<!-- Classe Ligne -->
<!ELEMENT Ligne (article, quantite, statut?, total?)>
<!ELEMENT article %string;>
<!ELEMENT quantite %int;>
<!ELEMENT lstatut %char;>
<!ELEMENT ltotal %float;>
```

# Exercice

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE recette SYSTEM "http://www.cook.org/dtds/recette.dtd">
<recette titre="Flan de chez ta mère" auteur="Didier Donsez">
<ingredients>
 <ingredient mesure="6">oeuf</ingredient>
 <ingredient mesure="6" unite="tasse à café">lait</ingredient>
 <ingredient mesure="6" unite="cuillère à soupe">sucre</ingredient>
 <ingredient mesure="1" unite="zeste">citron</ingredient>
</ingredients>
<instructions>
 <instruction>mélanger et battre au fouet tous les ingrédients</instruction>
 <instruction>caraméliser le fond et les parois du moule</instruction>
 <instruction>verser dans le moule</instruction>
 <cuisson mode=" bain maire au four" temperature="160°C" duree="40 minutes"/>
 <instruction>sortir du four et laisser refroidir au réfrigérateur 6 heures</instruction>
</instructions>
<remarque>Rentre chez ta mère, elle t'a fait un flan !</remarque>
</recette>
```

# Correction

```
<!ELEMENT recette (ingredients,instructions,remarque?)>
<!ELEMENT ingredients (ingredient+)>
<!ELEMENT instructions (instruction|cuisson)+>
<!ELEMENT ingredient (#PCDATA)>
<!ELEMENT instruction (#PCDATA)>
<!ELEMENT cuisson EMPTY>
<!ELEMENT remarque (#PCDATA)>
<!ATTLIST recette
    titre CDATA #REQUIRED
    auteur CDATA #IMPLIED >
<!ATTLIST ingredient
    mesure CDATA #REQUIRED
    unite CDATA>
<!ATTLIST cuisson
    mode CDATA #REQUIRED
    temperature CDATA #REQUIRED
    duree CDATA #REQUIRED >
```

# XPATH

# Déplacements dans un document XPath

- XPath :
  - Un langage fonctionnel pour adresser les sous-arbres d'un arbre XML
- Norme utilisée dans de nombreux outils XML
  - **XSLT** : langage de transformation d'arbre XML
  - **XPointer** : Extension des URLs
  - **XQuery** : langage d'interrogation de documents
  - **Java, C++, Javascript** : évaluateurs Xpath

# Un langage fonctionnel

- **Programme** : une expression
  - 1 + 1 ➔ 2
  - 2.0* (4-1) ➔ 6
  - concat('bonjour', " " , "monsieur") ➔ *bonjour monsieur*
- **Exécution** : évaluation de l'expression
- **Résultat typé**
  - booléen
  - chaîne
  - nombre décimal signé
  - ensemble de nœuds
    - expression = chemins dans un arbre

# Opérandes et opérateurs pour les types simples

# Constantes

- Chaînes
  - `'Paris'`
  - `"That's rubbish"`
  - `'He said "Boo"'`
- Valeurs numériques
  - `12`
  - `3.05`
  - `- 5.25`

- Valeurs Booléennes
  - `true()`
  - `false()`

# Opérateurs

- Expression numériques :
  - `+, -, *, div, mod`
  - `position(), last(), count(nds),`
  - `string-length(expr)`
- Expression booléenne :
  - `or, and, not(...)`
  - `boolean(...),`
  - `=, !=, <, <=, >=` (à écrire < et >)
- Expression Chaîne
  - `...`

# Opérateurs (suite)

- Expressions chaîne
  - `string(exp)`
  - `concat(exp1, exp2, …)`
  - `substring(expr,start),`
  - `substring(expr,start,length)`
  - `substring-before(expr,expr)`
  - `substring-after(expr,expr)`

# Conversions de type

- Explicites

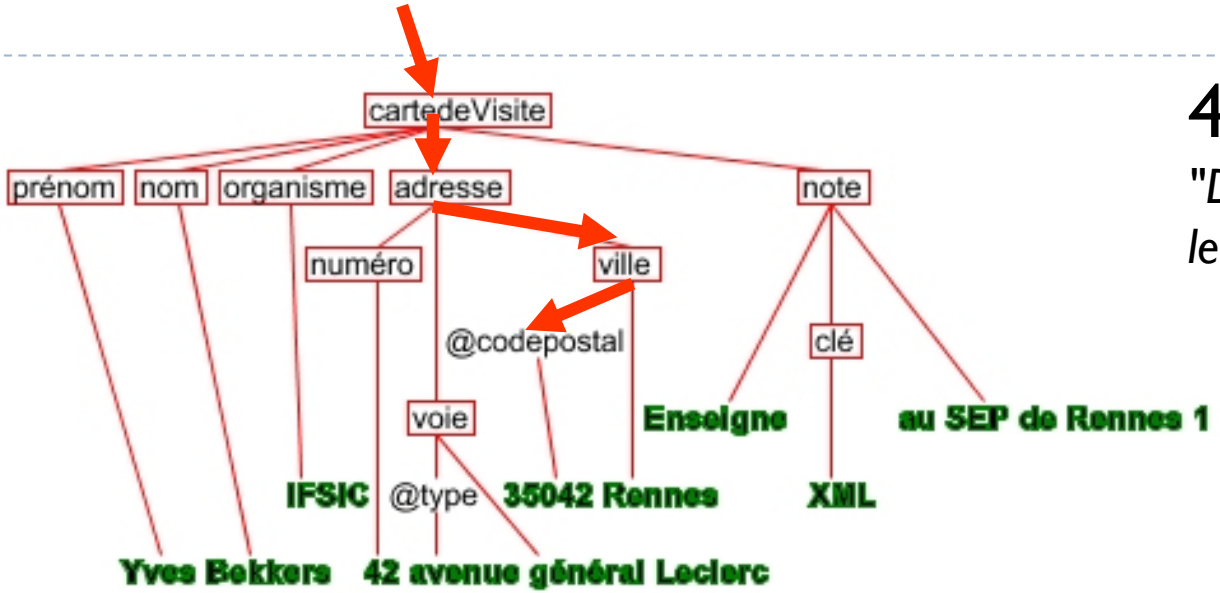
```
boolean(1 + 1) ➔ true
boolean(1 - 1) ➔ false
string-length("Boo") ➔ 3
```

- Implicites

```
boolean("Boo") ➔ true
```

# Type ensemble de nœuds

# Chemin absolu dans un arbre




## 4 étapes

*"Depuis la racine, le code postal"*

- Langage de chemin strict

```
/child::cartedeVisite/child::adresse
  /child::ville/attribute::codepostal
```

- Langage de chemin étendu

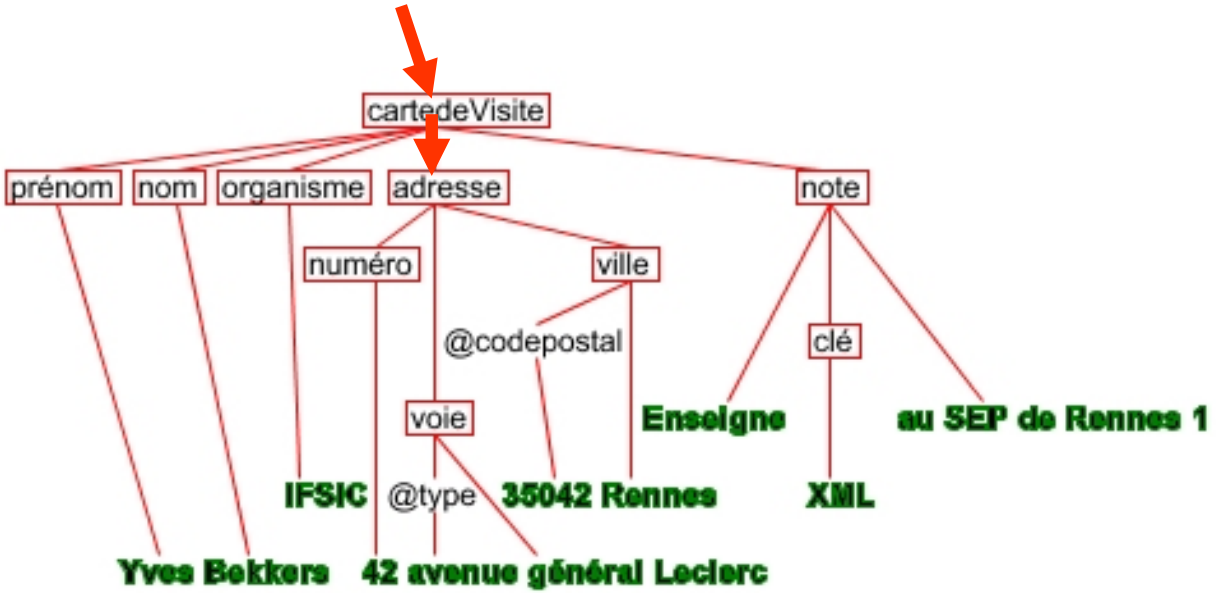
```
/cartedeVisite/adresse/ville/@codepostal
```

# Conversions de type

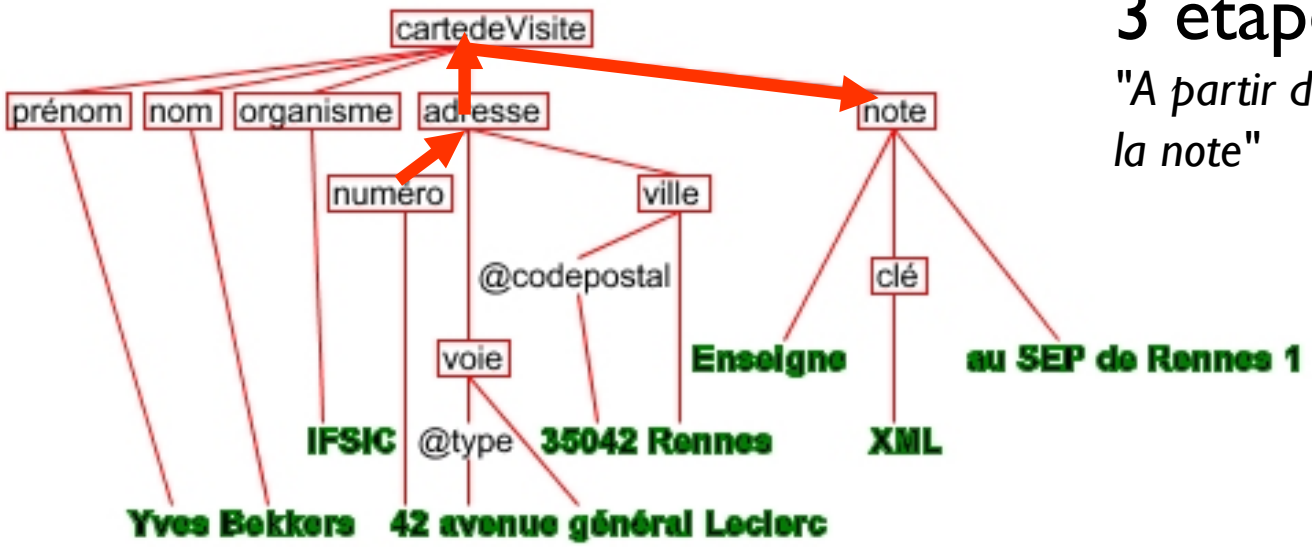
`boolean(/root)` ➔ `false`

`boolean(/cartedeVisite/adresse)` ➔ `true`

`count(/cartedeVisite)` ➔ `1`



# Chemin relatif




## 3 étapes

*"A partir du numéro, la note"*

- Langage de chemin strict

```
parent::*/parent::*/child::note
```

- Langage de chemin étendu

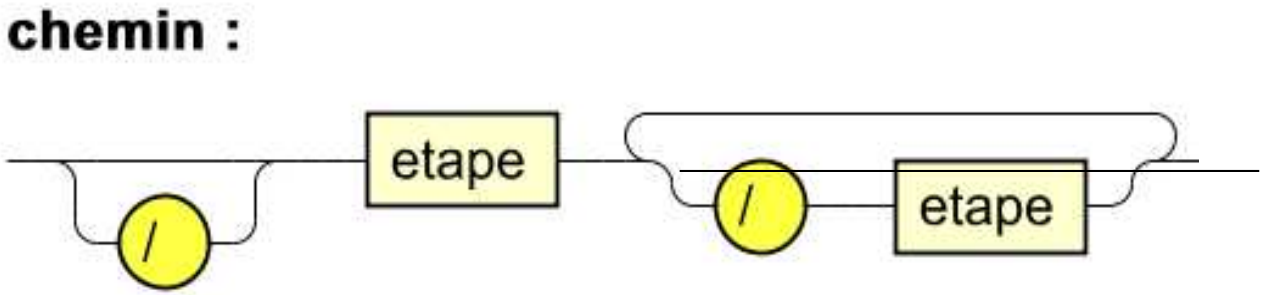
```
../../note
```

# Composition des étapes

- Un chemin est évalué comme la composition des étapes interprétées de gauche à droite avec
  - Un contexte initial fourni par l'application cliente
  - Tout nœud résultat d'une étape est utilisé comme nœud d'entrée de l'étape suivante
- Le résultat est un ensemble de nœuds

# Expression de chemin

- séquence d'étapes




- Chemin absolu
  - /étape1/étape2/étape3/ . . .
- Chemin relatif
  - étape1/étape2/étape3/ . . .

# Associativité

- Xpath

```
(carnetDAdresse/carteDeVisite)/nom
```

- Est équivalent à

```
carnetDAdresse/(carteDeVisite/nom)
```

# Racine d'un document

- La racine d'un document est au dessus de *l'élément racine du document*, elle contient
  - des commentaires éventuels
  - des instructions de traitements éventuelles
  - un et un seul élément (l'élément racine)

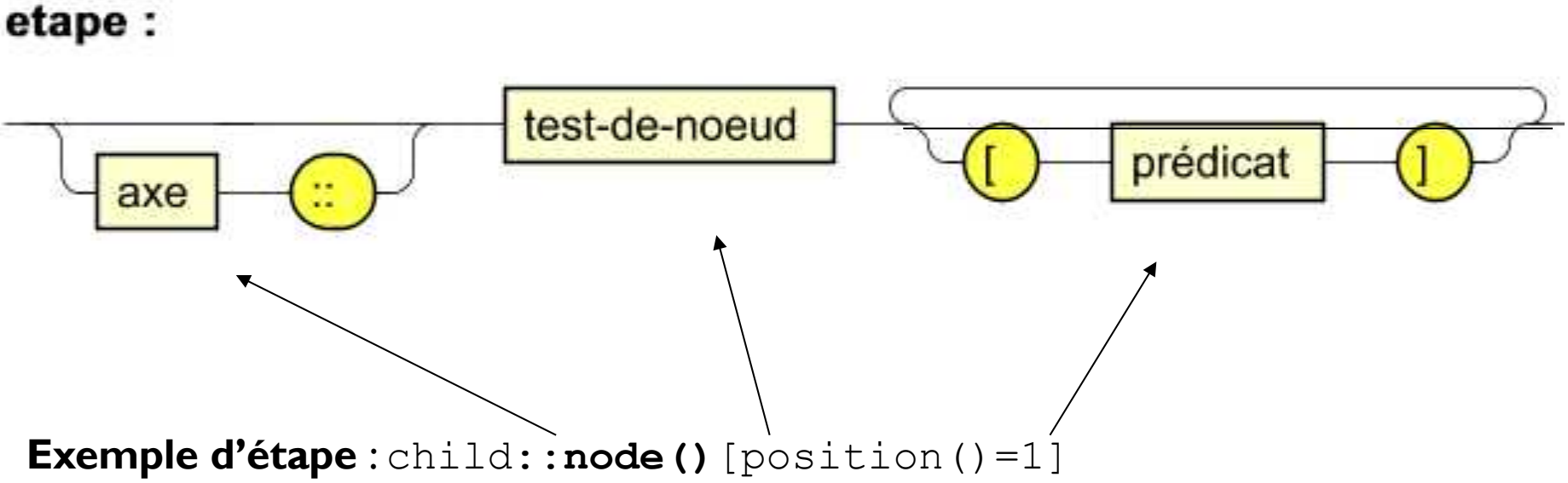
| | |
|---|---|
| `/` | Racine du document |
| `/child::html` | Elément racine du document si c'est html |
| `/child::*` | Elément racine du document |

# Contexte

- Un pas de localisation est évalué par rapport à un contexte courant
- Le contexte d'interprétation d'une étape XPath est composé de
  - Un nœud contextuel
  - Une position et une taille contextuelle (2 entiers)
  - Des liaisons de variables
  - Une librairie de fonctions
  - Des déclarations d'espaces de noms
- Le contexte est tenu à jour par les applications externes qui utilisent XPath (XPointer, XSLT, XQuery)

# Étape = sélectionner des nœuds

- Une étape est composée de trois parties
  - axe de déplacement (optionnel)
  - **Sélection de nœuds par leur type** (obligatoire)
  - prédicat (optionnel)




**Exemple d'étape** : `child::node()[position()=1]`

# Prédicat - exemples

```
[@codepostal='35700']
[nom[text()='Bekkers']]
[nom='Bekkers']
[position()=last()-1]
[not(position()=1)]
[boolean(clé)]
```

- Attention

  L'opérateur "=" accepte les ensembles en tant qu'opérande avec la sémantique suivante :

  "{a,b} = {c,d,e}" est *vrai* ssi il existe au moins couple d'éléments <x,y>, x dans {a,b} et y dans {c,d,e} tel que x=y

# Séquence de plusieurs prédicats

- le dernier fils du nœud courant pourvu que ce soit un élément « ville » ou un élément « rue »

```
[last()][ville or rue]
```

- le dernier des fils « ville » ou « rue » du nœud courant

```
[ville or rue][last()]
```

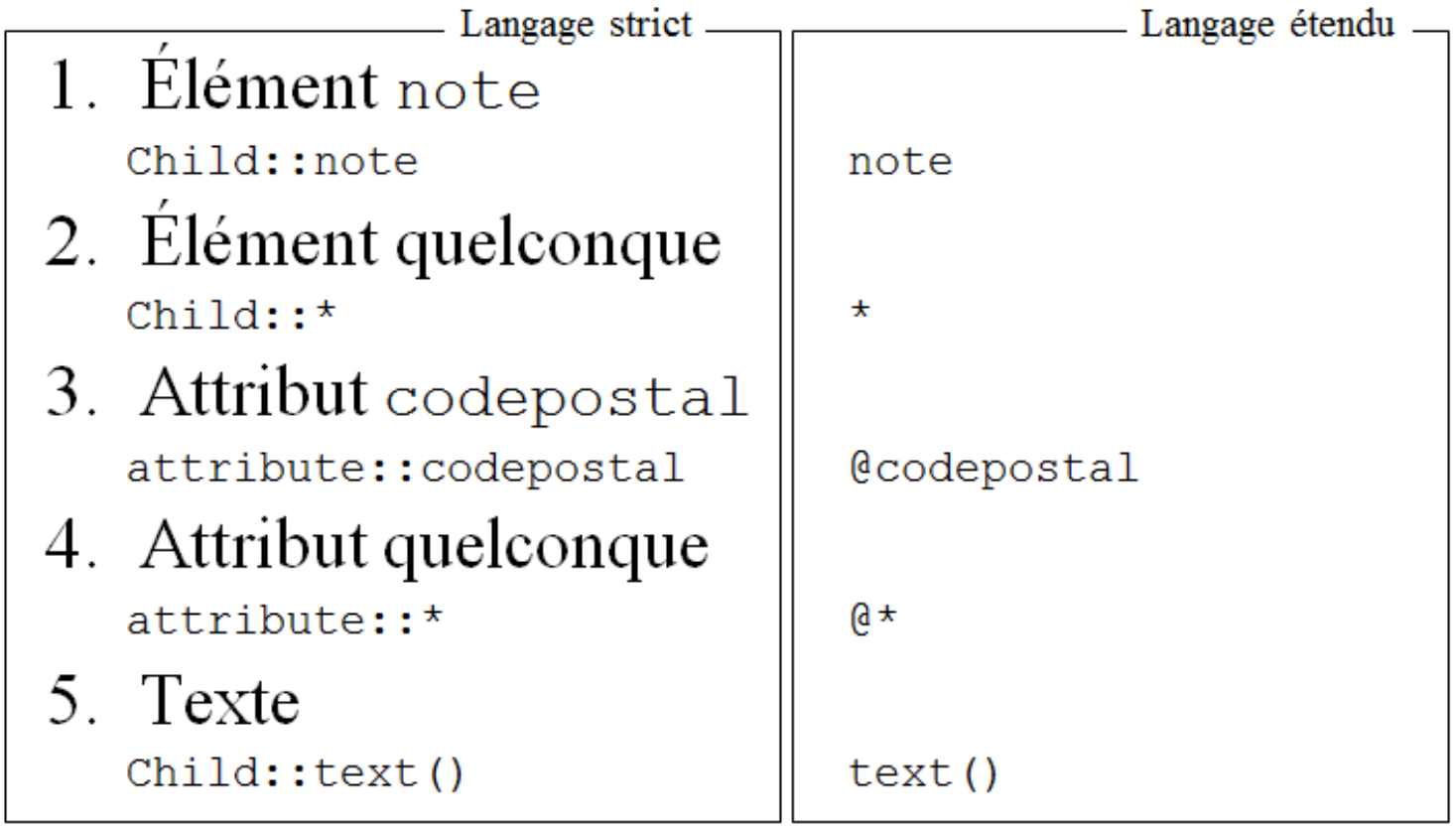
# SELECT DISTINCT de SQL

- tous les premiers éléments "ville" du document différents les uns des autres

```
//ville[not(text()=preceding::ville/text())]
```

- équivalent de *not in*

# Test de type de nœud



# Test de type de nœud (suite)

- Commentaire

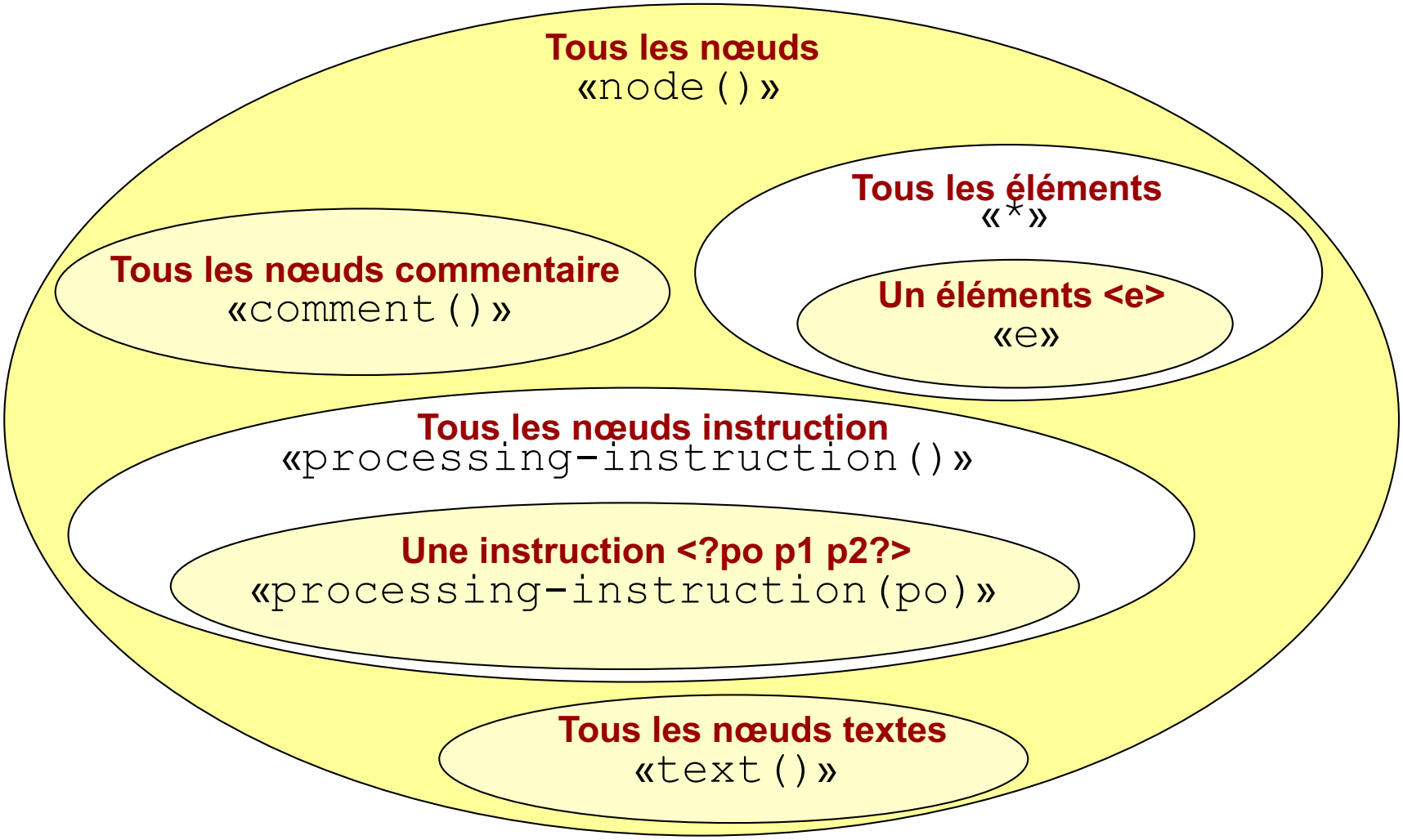
  `<!-- ceci est un commentaire -->`

  - `child::comment()`
- Instruction de traitement dont le nom est `monNom`

  `<?monNom arg1 arg2 arg3?>`

  - `child::processing-instruction(monNom)`
- Toute instruction de traitement
  - `child::processing-instruction()`

# Test de type de nœud

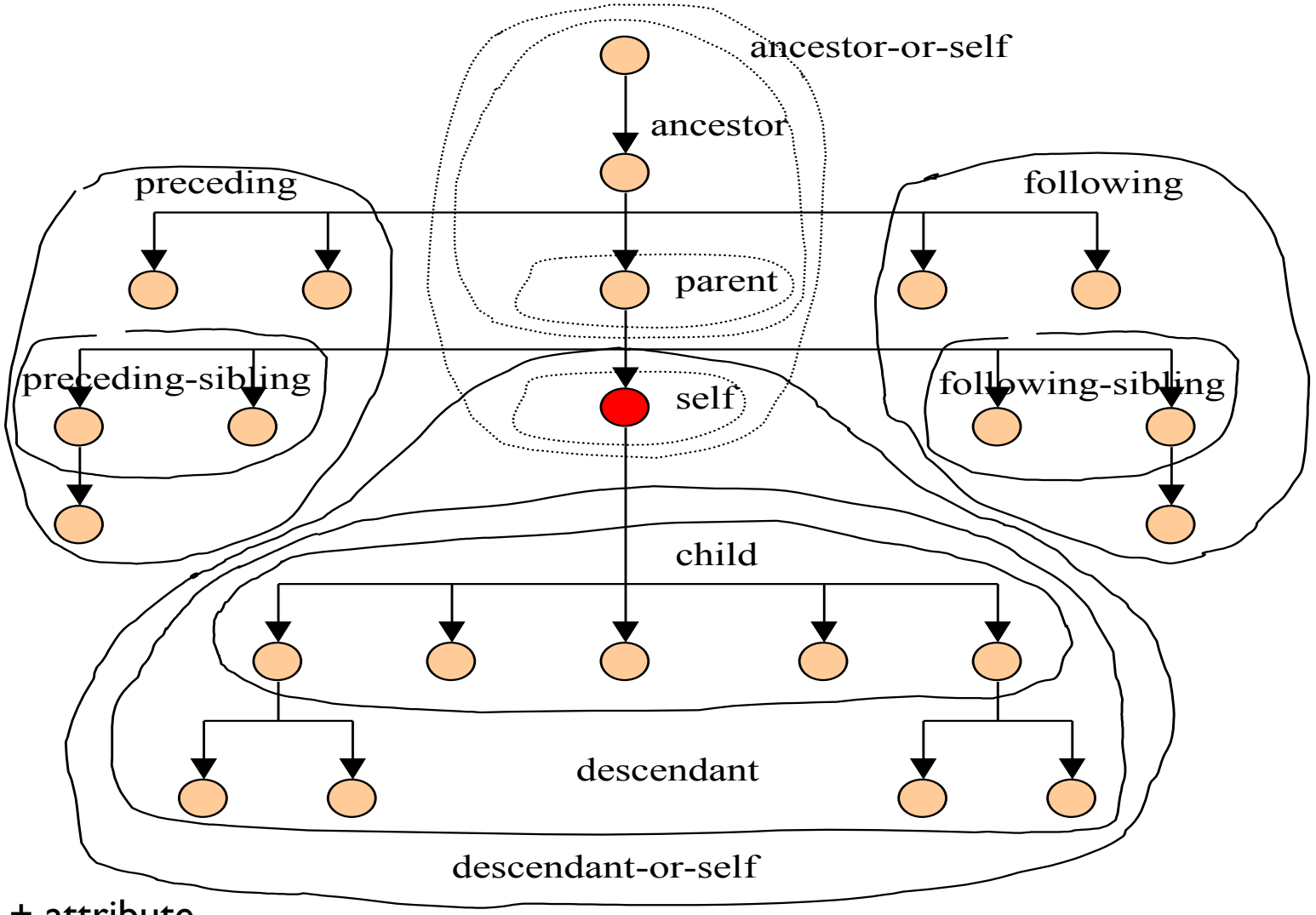



**Exemple d'étape** : `child::node()`

# Résumé

| Test | Nœuds sélectionnés |
|---|---|
| `*` | Tout élément |
| *`ville`* | Elément de nom `ville` |
| `text()` | Tout nœud de type texte |
| `processing-instruction()` | Toute instruction de traitement |
| `processing-instruction('proc')` | Processing instruction dont le nom est `proc` |
| `comment()` | Tout nœud commentaire |
| `node()` | Tout nœud |

# Axes de déplacement



# Axe following-sibling

```
<a>
  <b1/>
  <b2>
      <c21/>
      <c22/>
  </b2>
  <b3>
      <c31/>
      <c32/>
  </b3>
  <b4/>
</a>
```

## Expression Xpath

```
/descendant-or-self::node()
  /c21/following-sibling::*
```

## Réponse : 1 élément

**<c22/>**

*Raccourci*

/ descendant-or-self::node() = //

# Axe following

```
<a>
  <b1/>
  <b2>
      <c21/>
      <c22>
          <d/>
      </c22>
  </b2>
  <b3>
      <c31/>
      <c32/>
  </b3>
  <b4/>
</a>
```

## Expression Xpath

```
//c21/following::*
```

## Réponse :

**<c22/>**

**<d/>**

**<b3>**

**<c31/>**

**<c32/>**

**<b4/>**

# Axe ancestor-or-self

```
<a>
  <b1/>
  <b2>
      <c21/>
      <c22/>
  </b2>
  <b3>
      <c31/>
      <c32/>
  </b3>
  <b4/>
</a>
```

## Expression Xpath

```
//c22/ancestor-or-self::*
```

## Réponse : 3 éléments

**<a>**
**<b2>**

**<c22/>**

# Axe attribute

```
<a>
  <b1/>
  <b2>
      <c21/>
      <c22/>
  </b2>
  <b3 id='i1'>
      <c31/>
      <c32/>
  </b3>
  <b4/>
</a>
```

## Expression Xpath

```
/*/*[attribute::id='i1']
```

```
/*/*[@id='i1']
```

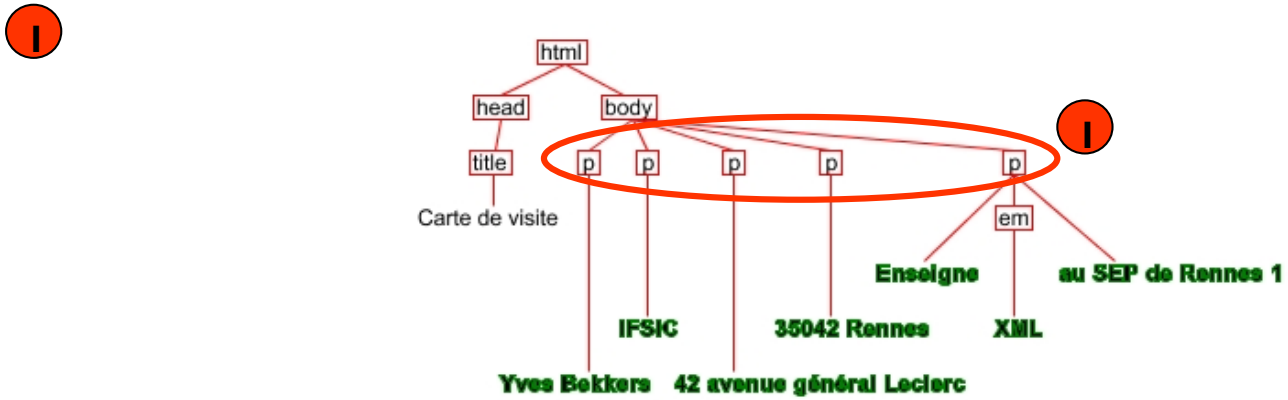
## Réponse : 1 élément

**<b3 id='i1'>**

# Filtrage à l'aide du prédicat

Une expression XPath s'évalue en un ensemble de nœuds

```
/html/body/p
```




Filtrage par la position

```
/html/body/p[position()=1]
```

Filtrage par le contenu

```
/html/body/p[em]
/html/body/p[position()=last()]
                              /em[text()='XML']
```

# Les raccourcis importants

**Langage strict**

```
child::nom
child::*
attribute::id
attribute::*

descendant-or-self::node()
self::node()
parent::node()
child::personne
  [string(nom)='Bekkers']
```

**Langage étendu**

```
nom
*
@id
@*

//
.
..
personne[nom='Bekkers']
```

# Conversions de type implicites

- Un prédicat en langage étendu

```
[.='Bekkers']
```

- En langage strict

```
[self::node()='Bekkers']
```

- En langage strict avec conversion explicite

```
[string(self::node())='Bekkers']
```

# Conversions de type implicites (2)

- Un prédicat en langage étendu

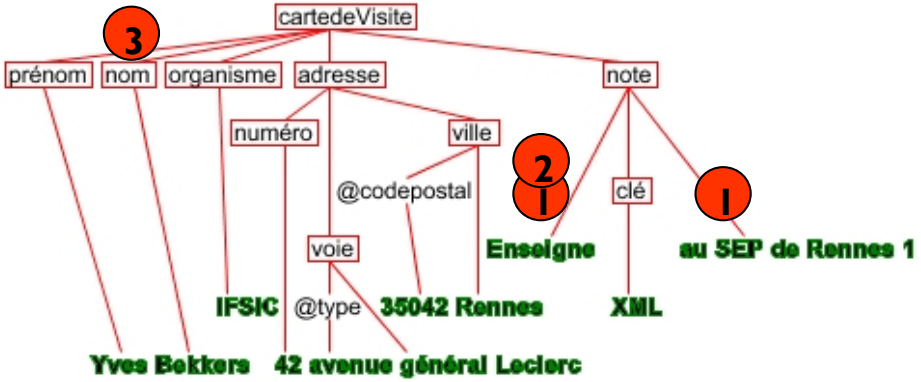
  `[clé]`
- En langage strict

  `[child::clé]`
- En langage strict avec conversion explicite

  `[boolean(child::clé)]`

# Quelques exemples

# Quelques exemples




1
```
/descendant-or-self::note/child::text()   langage stricte
//note/text()                             langage étendu
```

2
```
/descendant-or-self::note/child::text()[position()=1]
//note/text()[position()=1]
//note/text()[1]
```

3
```
/descendant-or-self::nom[string(self::nom)='Bekkers']
//nom[.='Bekkers']
```

# Quelques exemples

| | |
|---|---|
| `//figure` | Tous les éléments "figure" du document |
| `figure` | Tous les éléments "figure" fils du nd courant |
| `./figure` | Tous les éléments "figure" fils du nd courant |
| `*/figure` | Tous les éléments "figure" petit fils du nd courant |
| `adresse/@rue` | Les attributs "rue" des éléments "adresse" fils du nd courant |
| `adresse/text()` | Tous les textes situés directement sous l'élément "adresse" |

# Quelques exemples - suite

| | |
|---|---|
| `*[last()]` | Le dernier fils du nœud courant |
| `figure[lg]` | Tous les éléments "figure" fils du nd courant, pourvu qu'ils aient un fils "lg" |
| `ancestor::tst` | L'élément "tst" englobant le plus intérieur |
| `./@*` | Tous les attributs de l'élément courant |
| `XXX[@WIDTH and not(@WIDTH="20")]` | Les éléments XXX fils du nd courant pourvu qu'ils aient un attribut "WIDTH" avec une valeur différente de 20 |

# Opérateur union

- Tous les éléments pere ou mere du document

```
//pere | //mere
```

# Exercice

# Un fichier source

```
<carnetDAdresse>
  <carteDeVisite id="i1">
    <prénom>Yves</prénom>
    <nom>Bekkers</nom>
    <organisme>IFSIC</organisme>
    <adresse>
      <numéro>42</numéro>
      <voie type="avenue">général Leclerc</voie>
      <ville codepostal="35042">Rennes</ville>
    </adresse>
    <note>Enseigne <clé>XML</clé>
       au SEP de Rennes 1</note>
  </carteDeVisite>
...
```

# Questions

1. Tous les éléments `<note>`

1. Contenus et attributs des éléments `<note>`

2. Le troisième élément `<carteDeVisite>`

3. Le contenu de l'élément `<adresse>` de la troisième carte de visite

# Liens

Navigation XML

# Avec les attributs ID et IDREF

- L'attribut ID permet d'associer un identifiant à un élément
  Ex : <!ELEMENT cible (#PCDATA)>
  <!ATTLIST cible identifiant ID #IMPLIED>
- L 'attribut IDREF permet de référencer un élément qui possède un attribut ID
  Ex : <!ELEMENT référence EMPTY>
  <!ATTLIST référence ref IDREF #REQUIRED>

  D'où
  <cible identifiant='01234'> blabla </cible>
  <!-- une référence -->
  <référence ref='01234'/>

# XPointer XML Pointer Language

- XPointer est le standard qui permet de référencer des ressources
- Un XPointer est constitué d'une référence de base puis d'une cascade de références. Ainsi il est possible de référencer la valeur de l'attribut couleur du 3ème éléments fils de l'élément de type voiture.

# XPointer XML Pointer Language

- La référence de base d'un XPointer est :
  - root() : la racine du document cible
  - origin() : l'origine du pointeur.
  - id(nb) : l'élément ayant un attribut id avec nb comme valeur
  - html(val) : un élément <A> ayant un attribut name avec val comme valeur

# XPointer XML Pointer Language

- Les cascades de référence peuvent être :
  - child(), descendant(), ancestor(), …
    avec les paramètres suivant :
    - nb : le numéro de l 'occurrence
    - #element, #pi, #comment, #text, #all
    - nom : le nom d'un élément
    - un nom d'attribut et/ou une valeur d'attribut
  - attr(nom) pour la valeur d'un attribut nom
  - ...

# XPointer XML Pointer Language

- Un XPointer peut être précédé d'une URL, il faut utiliser le caractère # comme séparateur
- Exemples de XPointer
  http://un.deux.fr/toto.xml#root().child(1,titre)
  http://un.deux.fr/toto.xml#id(0123)
  ...

# XLink XML Linking Language

- Utiliser XML pour mémoriser des données est une approche intéressante et utile. Seulement, les données à manipuler sont parfois très volumineuses, voire sur des serveurs distants, fragmentées. Aussi, le W3C propose un mécanisme inspiré des hyperliens HTML : XLink (http://www.w3.org/XML/Linking). Ce «langage» (en fait une collection d'attributs) permet de mettre en relation une ou plusieurs ressources avec une ou plusieurs autres ressources. Couplé avec XPointer, les ressources peuvent être des éléments XML.

# XLink XML Linking Language

- Ce standard permet de mettre en relation plusieurs documents (ou ressources), pas nécessairement en XML. De plus, ces liens sont indépendant des documents mis en relation. XLink est une syntaxe basée sur des attributs XML définis sur l'espace de noms : «http://www.w3.org/1999/xlink».
- https://www.w3schools.com/xml/xml_namespaces.asp
- XLink propose deux versions de liens : les liens simples et les liens étendus. Il permet de représenter : des liens classiques «à la HTML» (liens simples), des tables des matières, des index...

# XLink XML Linking Language – Liens simples

- Le lien simple XLink est un ensemble d'attributs permettant de décrire un lien vers une ressource depuis un élément XML quelconque et son utilisation. Cette catégorie de lien est de même type que l'hyperlien HTML. Au minimum, deux attributs sont nécessaires :
  - «type» avec la valeur «simple» (c'est un lien simple) ;
  - «href» qui reçoit une URI (URL ou autre URI comme par exemple «uri:isbn:15513615698») ; c'est le lien proprement dit.

# XLink XML Linking Language – Liens simples

- Dans l'exemple ci-contre, les parties XLink sont colorées en rouge. Dans l'élément «identité», l'espace de nom XLink est défini et associé au préfixe «xlink». Ensuite, dans les éléments «site-web», l'attribut XLink «href», identifié par le préfixe spécifique, permet de référencer l'URL de ce site Web.

```
<identité xmlns:xlink="http://www.w3.org/1999/xlink">
  <nom>Asimov</nom>
    <prenom>Isaac</prenom>
    <date-naissance>1920-01-02</date-naissance>
    <date-deces>1992-04-06</date-deces>
    <nationalite>Russe/Américain</nationalite>
    <webographie>
    <site-web lang="fr"
          xlink:href="http://www.asimov.fr/"
          xlink:type="simple"/>
        <site-web lang="en"
           xlink:href="http://www.asimovonline.com/"
           xlink:type="simple"/>
      </webographie>
</identité>
```

# XLink XML Linking Language – Liens simples

- Il est possible de compléter les attributs précédents par d'autres décrivant le comportement des liens. Ainsi, il existe les deux attributs suivants :
- «show» qui indique la manière d'exploiter le lien quand il est activé :
  - «new» (par défaut) : affiche le contenu de l'URI dans une nouvelle fenêtre,
  - «replace» : affiche dans la fenêtre courante (remplace le contenu),
  - «embed» : inclut la ressource dans le document courant au niveau du lien,
  - «other» : autre comportement défini dans une balise dédiée,
  - «none» : ne rien faire ;

# XLink XML Linking Language – Liens simples

- «actuate» qui indique à quel moment activer le lien :
- «onLoad» : lien suivi dès que l'application le voit,
- «onRequest» (par défaut) : suivi sur demande de l'utilisateur,
- «other» : autre comportement défini dans une balise dédiée,
- «none» : pas d'explication.

# XLink XML Linking Language – Liens simples

- Un lien HTML classique sera : «actuate='onRequest'» et «show='replace'». Ainsi, l'exemple précédent peut devenir le document XML ci-après. D'autres attributs, plus orientés vers la sémantique existent :
  - «title» qui est un court texte décrivant la ressource ;
  - «role» donnant l'URI où trouver la description ou l'annotation de la ressource distante.

# XLink XML Linking Language – Liens simples

```
<identité xmlns:xlink="http://www.w3.org/1999/xlink">
   <nom>Asimov</nom>
   <prenom>Isaac</prenom>
   <date-naissance>1920-01-02</date-naissance>
   <date-deces>1992-04-06</date-deces>
   <nationalite>Russe/Américain</nationalite>
   <webographie>
     <site-web
        xlink:role="Principal site sur I. Asimov en français"
      xlink:title="ISAAC ASIMOV"
      xlink:href="http://www.asimov.fr/"
      xlink:type="simple"/>
     <site-web
        xlink:role="Site majeur sur I. Asimov en anglais"
      xlink:title="Isaac Asimov Home Page"
      xlink:href="http://www.asimovonline.com/"
      xlink:type="simple"/>
  </webographie>
</identité>
```

# XLink XML Linking Language – Xlink étendu

- XLink propose aussi des liens beaucoup plus complexes. Ici, nul besoin de se trouver «dans» une ressource. Le lien peut être indépendant du document de départ et du document visé. De plus, il permet de décrire une collection de ressources et des chemins entre ces ressources (chemins entre une ou plusieurs ressources).

# XLink XML Linking Language – Xlink étendu

- Pour cela, il propose de nouveaux attributs :
  - «title» et «role» sont possibles pour tous ces types ;
  - «type» prend alors «extended» sur l'élément englobant la description des liens et des ressources ;
- les ressources sont décrites dans l'élément principal avec les attributs suivants :
  - «type» vaut «ressource» (signifie une référence à une ressource locale) ou «locator» (signifie une référence à une ressource distante),
  - «href» donne l'URI de la ressource,
  - «label» sert à l'identification de la ressource, plusieurs ressources peuvent avoir le même label si elles sont prises ensembles dans le lien ;

# XLink XML Linking Language – Xlink étendu

- Les liens sont aussi décrits dans l'élément principal avec les attributs suivants :
  - «type» vaut «arc» détermine une connexion entre deux ressources,
  - «from» et «to» détermine les extrémités de l'arc (orienté), respectivement l'origine et l'extrémité.
- L'exemple ci-après présente un cas de liens étendus. On y retrouve :
  - (1) des ressources locales ;
  - (2) des ressources distantes sous forme d'URL ;
  - (3) des ressources distances sous forme de numéro ISBN (URI) ;
  - (4) des liens entre les ressources.

# XLink XML Linking Language

- Exemple

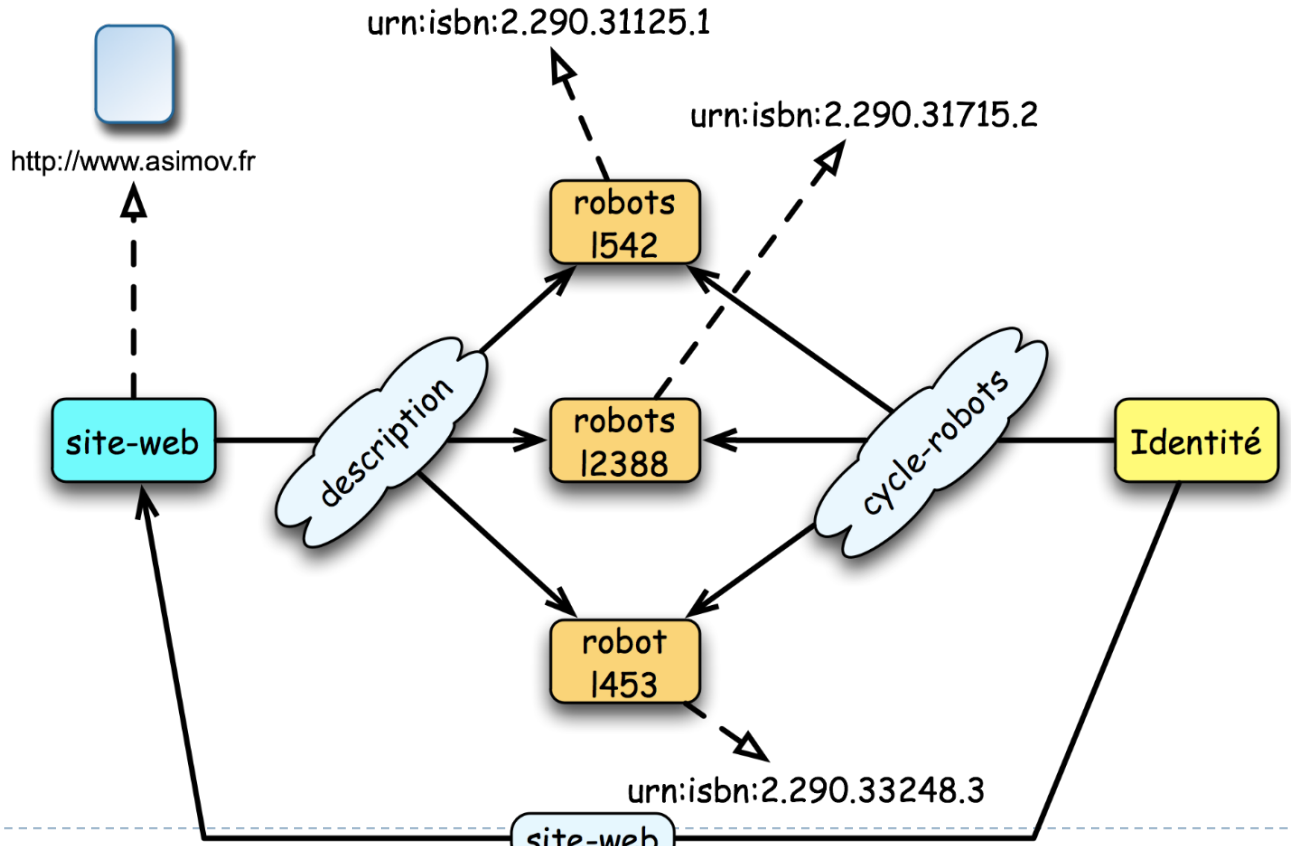
```
<?xml version="1.0" encoding="UTF-8"?>
<auteur xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="extended">
    <identité xlink:type="ressource"  xlink:label="identite">
        <nom>Asimov</nom><prenom>Isaac</prenom>
        <date-naissance>1920-01-02</date-naissance>                          ①
        <date-deces>1992-04-06</date-deces>
        <nationalite>Russe/Américain</nationalite>
    </identité>
    <site xlink:title="site d'un fan français"  xlink:label="site-web"
                  xlink:href="http://www.asimov.fr" xlink:type="locator"/>   ②

   <livre annee="1964" reference="1542" editeur="J'ai lu"
            xlink:type="locator" xlink:href="urn:isbn:2.290.31125.1"
            xlink:label="robots"> <titre>Un défilé de robots</titre>
    </livre>
   <livre annee="1950" reference="1453" editeur="J'ai lu"
            xlink:type="locator" xlink:href="urn:isbn:2.290.34248.3"          ③
            xlink:label="robots"><titre>I, robot</titre>
    </livre>
    <livre annee="2002" reference="12388" editeur="J'ai lu"
             xlink:type="locator" xlink:href="urn:isbn:2.290.31715.2"
             xlink:label="robots"><titre>Le robot qui rêvait</titre>
    </livre>

    <cycle-robots xlink:type="arc" xlink:from="identite" xlink:to="robots"
                        xlink:title="début du cycle des robots"/>            ④
    <site-web xlink:type="arc" xlink:from="identite" xlink:to="site-web"/>
    <description xlink:type="arc" xlink:from="site-web" xlink:to="robots"/>
</auteur>
```

# XLink XML Linking Language – Xlink étendu

- Graphiquement, cet exemple peut donner le schéma ci-dessous. Les flèches en pointillé indiquent les ressources externes. Les flèches pleines sont les liens XLink.



# XSLT

Langage de transformation d'arbre

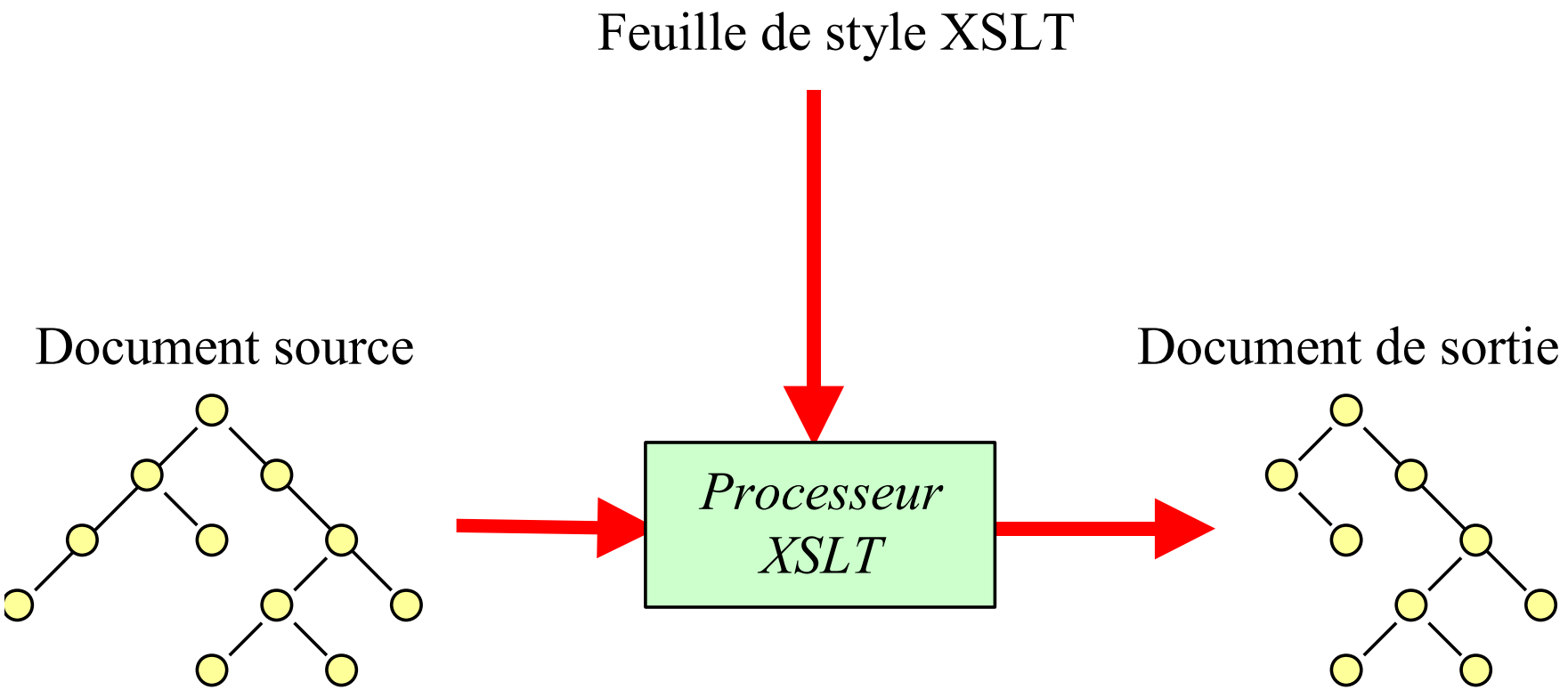
# Transformation de documents

- XSL (e***X****tensible* ***S****tylesheet* ***L****anguage*)
- Deux normes indépendantes
  - XSLT : langage de transformation
    - XPath : langage pour adresser les nœuds d'un arbre
  - XSL-FO : langage de formatage
    - Permet de spécifier un formatage plus fin que celui que l'on obtient à l'aide de HTML+CSS

# XSLT le langage de transformation

- Un langage déclaratif (Turing complet !)
  - avec une syntaxe XML !
- Les programmes XSLT s'appellent des *feuilles de styles*
  - Mais c'est beaucoup plus puissant que CSS
  - Exprime une transformation d'arbre en un autre arbre
- Modèle de calcul
  - Utilise une technique de *filtrage* à base de *motifs* (patterns) *et de modèles* (template) décrits dans des *règles* (template rules) pour transformer des arbres

# XSLT = Transformation d'arbre



# XSLT premier exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
      "http://www.w3.org/1999/XSL/Transform">
    <xsl:output method= "html" version="1.0"
      encoding="UTF-8"/>
     <xsl:template match="/">
          <html>
               <HEAD>
                    <TITLE>Bonjour</TITLE>
               </HEAD>
               <BODY>
                  <h1>Bonjour !</h1>
               </BODY>
          </html>
     </xsl:template>
</xsl:stylesheet>
```

# XSLT est un langage XML

- Les instructions sont des éléments XML

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
      "http://www.w3.org/1999/XSL/Transform">
   <!-- Format de sortie -->
   <xsl:output method="xml" version="1.0"
     encoding="UTF-8" indent="yes"/>

   <!-- ... règles XSLT ... -->

</xsl:stylesheet>
```

# XSLT un espace de noms

- Espace de nom XSLT
  - `http://www.w3.org/1999/XSL/Transform`
  - Préfixe recommandé `xsl:`

# Prélude d'une feuille de style

# Elément <xsl:stylesheet>

- Élement racine d'un document XSLT

```
<xsl:stylesheet
    version="1.0"
    xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform"
>
```

  - Attribut `version` : version de langage XSL (obligatoire)
  - Attribut `xmlns:xsl` : espace de nom XSL

# Elément <xsl:output>

- Format de sortie du document résultat

```
<xsl:output method="xml" version="1.0"
      encoding="UTF-8" indent="yes"/>
```

  - Attribut `method` : type du document en sortie
  - Attribut `encoding` : codage du document
  - Attribut `indent` : indentation en sortie

# Type de document en sortie

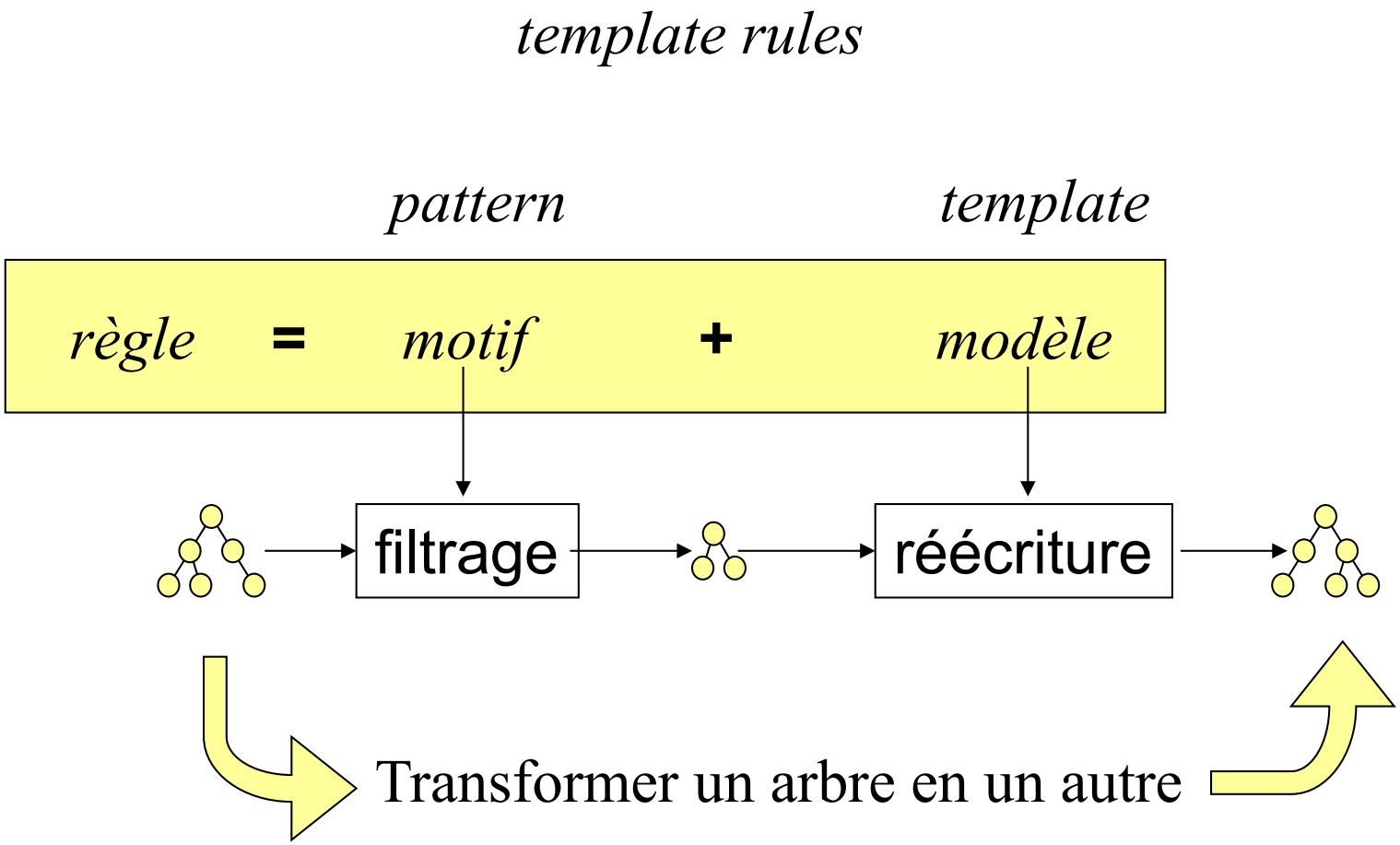
- Trois types de document en sortie
  - xml : vérifie que la sortie est bien formée
    - (*sortie par défaut*)
  - html : accepte les balises manquantes, génère les entités HTML (é ...)
    - (*sortie par défaut si XSL reconnaît l'arbre de sortie HTML4*)
  - text : tout autre format textuel :
    - du code Java, format Microsoft RTF, LaTex

# Parcours/transformation d'arbre

- Règle de réécriture : *template rules*
  - `<xsl:template>`
- Spécifier un parcours de l'arbre d'entrée
  - `<xsl:apply-templates>`
  - `<xsl:for-each>`
- Obtenir une valeur dans l'arbre source
  - `<xsl:value-of>`
  - les crochets dans un attribut

```
<a href="{@src}">
```

# Règles de réécriture



# Élément <xsl:template>

- Règle de réécriture *motif* + *modèle*

```
<xsl:template match="motif">
    … modèle …
</xsl:template>
```

  - Attribut `match` : expression XPATH
    - Un motif pour filtrer l'arbre d'entrée
  - Contenu de l'élément `<xsl:template>` :
    - Un modèle de sous-arbre en sortie
- Un programme XSLT est un ensemble de règles

# Premier exemple complet

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:output method="html"/>

    <xsl:template match="/">
        <HTML>
            <HEAD>
                <TITLE>Welcome</TITLE>
            </HEAD>
            <BODY>
                Welcome!
            </BODY>
        </HTML>
    </xsl:template>
</xsl:stylesheet>
```

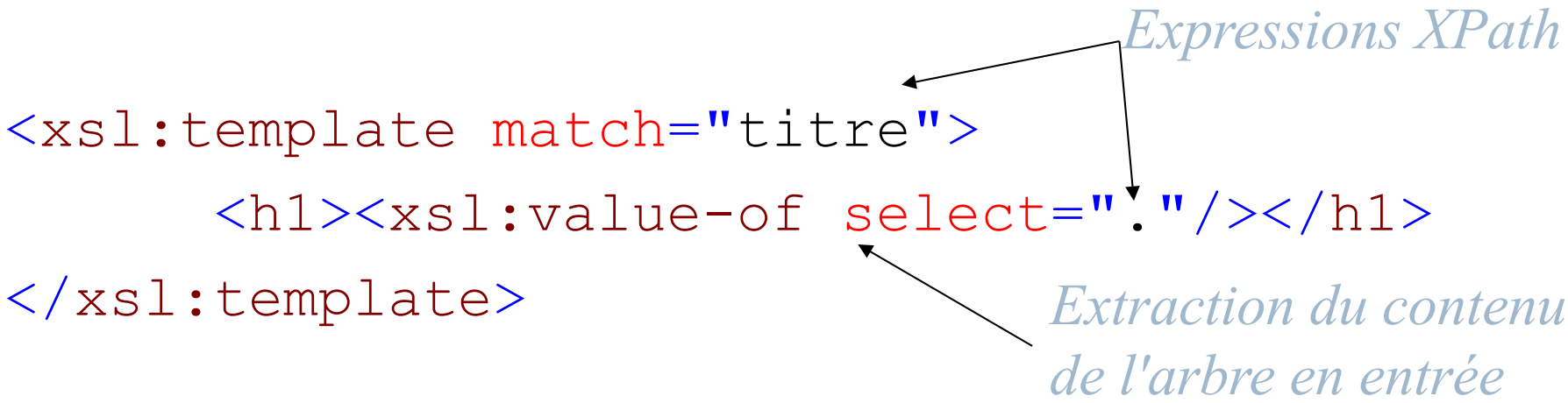
motif
« *filtre la racine du document d'entrée* »

règle

modèle
« *Document html*»

# Contenu de l'élément <xsl:template>

- Un élément <xsl:template> contient
  - Le modèle de texte HTML (ou XML ou texte simple)
    - *le texte XML doit être bien formé*
  - Des instructions XSLT (mélangées au modèle de sortie)
    - *pour générer un texte en relation avec le contenu du document source*
    - *pour extraire des informations du document source*



# Second exemple - le carnet d'adresse

**En entrée**

```
<carnetDAdresse>
  <carteDeVisite>
    <nom>Bekkers</nom>
    ...
  </carteDeVisite >
  <carteDeVisite>
    <nom> Bartold </nom>
    ...
  </carteDeVisite >
  ...
</ carnetDAdresse >
```

**En sortie**

```
<html>
 <body>
 <h1>Liste des Noms</h1>
       <p>Nom : Bekkers</p>
       <p>Nom : Bartold</p>
       <p>Nom : Letertre</p>
       <p>Nom : Apolon</p>
 </body>
</html>
```

# Second exemple (1)

```
<xsl:stylesheet version="1.0"
    xmlns:xsl=
        "http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" indent="no"
        encoding="iso-8859-1"/>

    <xsl:template match="/">
      <html>
        <xsl:apply-templates select="child::*"/>
      </html>
    </xsl:template>
```

Expression xpath

Modèle de sous-arbre

...

# Second exemple (2)

```
<xsl:template match="carnetDAdresse">
  <body>
    <h1>Liste des Noms</h1>
    <xsl:apply-templates select="child::*"/>
  </body>
</xsl:template>

<xsl:template match="carteDeVisite">
  <p>Nom : <xsl:value-of select="nom"/>
  </p>
</xsl:template>

</xsl:stylesheet>
```

# Résultat

- Pour un document source contenant 4 cartes de visite

```
<html>
 <body>
 <h1>Liste des Noms</h1>
        <p>Nom : Bekkers</p>
        <p>Nom : Bartold</p>
        <p>Nom : Letertre</p>
        <p>Nom : Apolon</p>
 </body>
</html>
```

# Changement de contexte - Élément <xsl:apply-templates>

- Descente dans les fils d'un nœud

```
<xsl:template match="carnetDAdresse">
  <body>
    <h1>Liste des Noms</h1>
    <xsl:apply-templates
                    select="child::node()"/>
  </body>
</xsl:template>
```

Expression xpath

- Raccourci d'écriture
  - descente par défaut aux nœuds fils

```
<xsl:apply-templates/>
```

# Élément `<xsl:value-of>`

- Générer le contenu d'un élément

```
<xsl:template match="carteDeVisite">
    <p>Nom : <xsl:value-of select="nom"/>
    </p>
</xsl:template>
```

- Sélection de la valeur :
  - attribut `select` : expression xpath
  - ici : le texte contenu dans l'élément `nom` de l'élément `carteDeVisite`

# Résultat de <xsl:value-of> et type nœud

- Le nœud selectionné est un *élément*
  - Concaténation de tous les textes qui se trouvent comme contenu de cet élément et de ses descendants
- Le nœud est un nœud *text*
  - Texte du nœud lui même
- Le nœud est un *Attribut*
  - Valeur de l'attribut normalisée (pas d'espace de début et fin)
- Le nœud est une *Instruction de traitement*
  - Valeur de l'instruction de traitement

  (sans les marques <? et ?> et sans le nom)
- Le nœud est un *Commentaire*
  - Le texte du commentaire (sans les marques <!-- et -->)

# Exemple 1

- Arbre en entrée

```
<carteDeVisite>
  <nom>Bekkers</nom>
</carteDeVisite>
```

- Règle

```
<xsl:template match="carteDeVisite">
   <p>Nom : <xsl:value-of select="nom"/></p>
</xsl:template>
```

- Arbre en sortie

```
<p>Nom : Bekkers</p>
```

# Exemple 2

- Arbre en entrée

```
<note>enseigne <clé>XML</clé> au SEP</note>
```

- Régle

```
<xsl:template match="note">
        <xsl:value-of select="."/>
</xsl:template>
```

- En sortie

```
enseigne XML au SEP
```

# Exemple 3

- Arbre en entrée

```
<note>enseigne <clé>XML</clé> au SEP</note>
```

- Règle

```
<xsl:template match="note">
      <xsl:value-of select= "text()"/>
</xsl:template>
```

- En sortie

```
enseigne
```

Seul le premier élément sélectionné est produit

# Exemple 4

- Arbre en entrée

```
<note>enseigne <clé>XML</clé> au SEP</note>
```

- Règle

```
<xsl:template match="*">
        <xsl:value-of select="name()"/>
</xsl:template>
```

- En sortie

```
note
```

# Exemple 5

- Arbre en entrée

4 cartes de visite : Bekkers, Bartold, Letertre, Apolon

- Règle

```
<xsl:template match="/carnetDAdresse">
    <xsl:value-of select="carteDeVisite/nom"/>
</xsl:template>
```

- En sortie

Bekkers

Seul le premier élément sélectionné est produit

# Exemple 6

- Arbre en entrée

4 cartes de visite : Bekkers, Bartold, Letertre, Apolon

- Règle

```
<xsl:template
  match="/carnetDAdresse/carteDeVisite">
        <xsl:value-of select="nom"/>
</xsl:template>
```

- En sortie

BekkersBartoldLetertreApolon

Pour chaque carte de visite le template est appliqué

# Règles par défaut

# Règles par défaut (1)

Traverser la racine et tous les noeuds « élément »

```
<xsl:template match="*|/">
  <xsl:apply-templates/>
</xsl:template>
```

Sortir les feuilles « texte » et les « attributs »

```
<xsl:template match="text()|@*">
  <xsl:value-of select="."/>
</xsl:template>
```

# Règles par défaut (2)

- Commentaires et instructions de traitement

```
<xsl:template match="processing-
  instruction()|comment()"/>
```

- Ne rien faire

# Feuille de style minimum

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform">

     <xsl:output method="text"/>

</xsl:stylesheet>
```

- Traverse tout l'arbre et sort les feuilles (contenu d'élément texte et valeur d'attribut)

# Génération de contenu

# Résultat littéral ou non ?

# Méthodes de génération de contenu

- Deux méthodes de génération de contenu :

1) Résultat litéral

```
<xsl:template match="subtitle">
  <h2><xsl:apply-templates/></h2>
</xsl:template>
```

2) Résultat non litéral

(validation possible de la feuille de style)

```
<xsl:template match="subtitle">
  <xsl:element name="h2">
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>
```

# Autre exemple

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:foo="http://example.com">

  <xsl:output method="xml" indent="yes" />

  <xsl:template match="/">
    <root>
      <foo:element1 />
      <foo:element2 />
    </root>
  </xsl:template>
</xsl:stylesheet>
```

Si on change le litéral <root> par <xsl:element name="root" >

```
<?xml version="1.0"?>
<root xmlns:foo="http://example.com">
  <foo:element1/>
  <foo:element2/>
</root>
```

```
<?xml version="1.0"?>
<root>
  <foo:element1 xmlns:foo="http://example.com"/>
  <foo:element2 xmlns:foo="http://example.com"/>
</root>
```

# Valeur d'attribut par résultat littéral évalué

Évaluation d'expression xpath en accolades dans les valeurs d'attribut

- Arbre en entrée

```
<a href="fic.txt"/>
```

- Template

```
<xsl:template match="a">
    <b id="{@href}"/>
</xsl:template>
```

- En sortie

```
<b id="fic.txt"/>
```

# Résultat non littéral <xsl:attribute>

- Arbre en entrée

```
<a href="fic.txt"/>
```

- Template

```
<xsl:template match="a">
   <b><xsl:attribute name="id">
         <xsl:value-of select="@href"/>
      </xsl:attribute></b>
</xsl:template>
```

- En sortie

```
<b id="fic.txt"/>
```

# Parcours itératifs

# Élement <xsl:for-each>

- Itération sur un ensemble de nœuds

```
<xsl:template match="/carnetDAdresse">
    <xsl:for-each select="carteDeVisite">
        <p><xsl:value-of select="nom"/></p>
    </xsl:for-each>
</xsl:template>
```

# Deux styles de programmation

- Récursive

```
<xsl:apply-templates>
```

- Itérative

```
<xsl:for-each>
```

- Attribut `select` donne l'ensemble de nœuds vers lequel on se déplace

# Élément <xsl:comment>

- Sortir les commentaires à l'identique

```
<xsl:template match="comment()">
    <xsl:comment>
        <xsl:value-of select="."/>
    </xsl:comment>
</xsl:template>
```

# Élément <xsl:processing-instruction>

- Sortir les instructions de traîtement à l'identique

```
<xsl:template match="processing-
  instruction()">
   <xsl:processing-instruction name="./name()">
      <xsl:value-of select="."/>
   </xsl:processing-instruction>
</xsl:template>
```

# Conflits de Règles

- Règle implicite de priorité
  - La règle la plus sélective gagne
  - Parmi 2 templates de même sélectivité, le dernier dans la feuille de style gagne
- Exemple
  - `nom` est plus sélectif que `/|*`
  - `note[clé]` est plus sélectif que `note`
  - `ville[@codepostal='35000']` est plus sélectif que `ville[@codepostal]`

# Les modes

- Permet de déclarer plusieurs règles pour un même élément
- Chaque règle traite l'élément différemment

```
<xsl:template match="h1" mode="normal">


<xsl:template match="h1" mode="table-index">
```

# Attributs `mode`

- Dans un élément `apply-templates`

```
<xsl:apply-templates mode="passe1"/>
```

- Dans un élément `template`

```
<xsl:template match="carteDeVisite"
    mode="passe1">
    ...
</xsl:template>
```

- Attention un `apply-templates` n'hérite pas du mode du `template` englobant

# Autres outils

# Élément <xsl:if>

- Conditionnelle

```
<xsl:for-each select="carteDeVisite">
  <xsl:value-of select="nom"/>
  <xsl:if test="position()!=last()">,
  </xsl:if>
</xsl:for-each>
```

- Génère une virgule après chaque nom sauf pour le dernier
- En sortie

```
Bekkers, Bartold, Letertre, Apolon
```

# Élément <xsl:choose>

- Conditionnelle à choix multiple

```
<xsl:choose>
    <xsl:when test="start-with('35',@codep)">
        <!-- cas 1 -->
    </xsl:when>
    <xsl:when test="start-with('44',@codep)">
        <!-- cas 2 -->
    </xsl:when>
    <xsl:otherwise>
        <!-- autres cas -->
    </xsl:otherwise>
</xsl:choose>
```

# <xsl:variable>

- Déclaration de variable 1

```
<xsl:variable name="blackcolor"
  select="#FFFFCC" />
```

- Déclaration de variable 2

```
<xsl:variable
  name="blackcolor">#FFFFCC</xsl:variable>
```

- Référence à une variable

```
<BODY BGCOLOR='{$blackcolor}'>
```

# <xsl:variable>

- XSL est un langage à assignation unique
- Les « variables » sont des constantes à la manière des constantes `#define` de C
- Une variable ne peut être réaffectée
- La visibilité d'une variable est son élément père
- Une variable peut en cacher une autre

# Initialisation conditionnelle

- Exemple

Java

```
if (niveau > 20)
    code = 3;
else
    code = 5;
```

XSLT

```
<xsl:variable name="code">
  <xsl:choose>
    <xsl:when test="$niveau &gt; 20">
      <xsl:text>3</xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>5</xsl:text>
    </xsl:otherwise>
  </xsl:choose>
</xsl:variable>
```

# Les espaces

- Les espaces non significatifs dans l'arbre xsl ne sont pas produits

```
<xsl:template match="nom">
    <p><xsl:value-of select="."/>
        </p>
</xsl:template>
```

et

```
<xsl:template match="nom">
    <p><xsl:value-of select="."/></p>
</xsl:template>
```

ont le même effet

# Xquery

- http://www.louizi.com/xquery.ppt
- http://www.louizi.com/xquery.pdf

# Traitement de documents XML

DOM et SAX

# Parseur

- XML offre de nouveaux horizons applicatifs
- Il n'est pas possible d'imaginer l'étendue de ce domaine applicatif
- Cependant, toutes les applications auront besoin de parseur
- Il y a deux types de parseur normalisés
  - SAX
  - DOM

# SAX : Simple API for XML

- SAX est un type de parseur XML
- SAX a été construit par des internautes en 1997, ils voulaient standardiser ce type de parseur
- SAX est implanté en Java et en Python
- SAX est un parseur événementiel
  - Lorsque que le parseur lit un élément du document, il lance un événement
  - Il est utilisé par les applications qui considèrent les documents XML comme des flots de données

# SAX : Simple API for XML

- SAX étant un parseur événementiel, il propose un ensemble d'interface pour implanter des *handler*
  - DocumentHandler
    - startElement()
    - starDocument()
  - ErrorHandler
    - Error()
- Il suffit ensuite de créer un parseur XML, d'affecter les handler puis de commencer à scanner le document

# SAX : Simple API for XML

Exemple d'applications utilisant SAX

```
public class MyHandler extends DocumentHandler {
        public void startElement(String n, AttributeList att) {
                System.out.println("Start element :" + n);
        }
}

….
   DocumentHandler dh = new MyHandler();
   parser.serDocumentHandler(dh);
```

# DOM : Document Object Model

- DOM est un type de parseur XML
- DOM a été standardisé par le W3C
- L'API du DOM est définit en IDL CORBA
- DOM est un parseur compilé
  - Lecture de la totalité du document XML
  - Création d'une structure arborescente d'objet qui représente le document XML

# DOM : Document Object Model

- Le DOM est un parseur compilé, son API est composée de toutes les entités d'un document XML
  - Document : Objet qui représente le document XML
  - Element : Objet qui représente un élément XML
  - Node : Objet dont tous les autres objets héritent
- Pour construire une application qui utilise le DOM, il suffit de créer un parseur et d'appeler la méthode parse qui retourne un objet de type Document

# DOM : Document Object Model

Exemple d'applications utilisant DOM

```
try {
   DOMParserWrapper parser = new …
   Document docu = parser.parser(uri);
   …
```

# Conclusion pour les traitements

- SAX et DOM sont deux types de parseur standardisés par les internautes pour SAX et par le W3C pour DOM
- SAX est plus orienté traitement du document comme un flot d'information
- DOM est plus orienté transformation de document ou archivage
- Il existe de nombreuses implantations gratuites en Java de SAX et de DOM (Sun, IBM, …)

# Les protocoles d'intégration

# Introduction

- Les systèmes d'informations d'entreprises reposent sur des parcs aux technologies hétérogènes (J2EE, .NET, PHP, Ruby on rails). Cela rend le besoin d'interopérabilité de plus en plus indispensable. Jusqu'à présent, il a été possible de répondre à ces besoins en effectuant des développements spécifiques (JSP ou servlets), appelés de manière distante (en HTTP) ou l'inverse.

# Les solutions technologiques

- Plusieurs normes d'interopérabilité pourraient être utilisées, parmi ces normes les plus fréquemment citées, on trouve :
- **JSR-170** (et sa version plus récente, **JSR-283**) : il s'agit de deux normes permettant d'accéder à un *Content repository*, c'est-à-dire à un système de persistance de gestion de contenu. L'interface définie par cette norme est en Java, et donc il n'y a pas indépendance par rapport au langage des applications tierces. Ces normes sont limitées à la seule gestion des contenus.

# Les solutions technologiques

- **JSR-168** (et sa version plus récente, **JSR-286**) : ces deux normes permettent qu'un portail conteneur de portlets*, au sens de ces normes, présente des portlets développées à l'extérieur. Il s'agit, là encore, d'interopérabilité limitée au seul langage Java. L'étendue de l'interface est limitée aux portlets et ne permet pas d'interaction applicative ;

- **Un portlet est une application informatique que l'on peut placer dans un portail Web, qui sert alors de conteneur. C'est un objet qui affiche un bloc sur une page Web et qui est souvent émis par des servlets. Un portlet traite les requêtes d'une tâche ou d'un service donné et génère dynamiquement le contenu Web affiché à l'utilisateur.*

# Les solutions technologiques

- **WSRP** : cette norme a essentiellement le même objectif que celui des normes JSR-168 et 286, mais l'interface n'est pas en Java, mais en services web SOAP. Sa vocation est limitée à l'interopérabilité de portlets ;

# Les solutions technologiques

- **Services Web SOAP** : ***SOAP*** est une norme permettant de faire des appels de procédures à distance, de manière indépendante du protocole de transport sous-jacent. Lorsque cette norme est utilisée en corrélation avec HTTP, on parle de services web SOAP.

- Utiliser et développer des services web SOAP nécessite de la part des développeurs de disposer d'ateliers de développement (AGL ou outils RAD) supportant cette norme.

# Les solutions technologiques

- Dans la pratique, les services Web SOAP sont principalement utilisés dans des réseaux locaux, car ils utilisent beaucoup de bande passante et ils peuvent être utilisés, par exemple, en conjonction avec des EAI fonctionnant en services web SOAP (on parle alors d'ESB) ;
- ***EAI***
  - *Middleware chargé d'effectuer le transport, le routage et la transformation de messages entre plusieurs applications, l'EAI évite de multiplier les connexions point à point, très coûteuses à développer et entretenir. Jugés lourds et coûteux, les premiers EAI propriétaires – assimilables à des progiciels de l'intégration – ont laissé la place, dans un premier temps, à des outils dits « tactiques », basés sur des standards, notamment les services Web. De ce fait, ils se sont rapprochés sensiblement de la philosophie des ESB.*

# Les solutions technologiques

- ***ESB***
  - *L'Enterprise Service Bus ou ESB est une technique informatique intergicielle. Son but est avant tout de permettre la communication des applications qui à la base ne sont pas pensées pour fonctionner ensemble (deux ERP - Enterprise Resource Planning ou Progiciel de gestion intégré - provenant de deux éditeurs différents par exemple).*

  - *On peut considérer l'ESB comme une nouvelle génération d'EAI (en français, Intégration d'applications d'entreprise) construite sur des standards comme XML, JMS ou encore les services web. Aussi, la différence majeure avec l'EAI réside dans le fait que l'ESB propose une intégration complètement distribuée grâce à l'utilisation des conteneurs de services. Ces "mini-serveurs" contiennent la logique d'intégration et peuvent être déposés n'importe où sur le réseau.*

# Les solutions technologiques

- **Services Web REST** : il s'agit de services web basés sur HTTP, reposant sur l'utilisation pertinente des possibilités et du sens de chaque élément de la norme et des champs des requêtes et réponses HTTP.

# Services Web REST

- c'est le format d'interopérabilité qui est massivement adopté par les services web sur Internet, et qui est utilisé pour réaliser des *mashups* (Services Web Google Map, Amazon S3, Flickr, …) ;
- il est moins lourd au niveau développement de développer de tels services web que des services web SOAP ;
- Le fait d'utiliser des services web REST n'induit pas de restrictions fonctionnelles.

# Services Web REST

- REST n'est pas strictement un protocole, mais plutôt un ensemble de principes à respecter pour fournir une API de service web HTTP. On dit d'une API de services web respectant ces principes qu'elle est *RESTful* ou orientée REST. On parle aussi d'Architecture Orientée Ressources (ROA pour Resources Oriented Architecture).

# Services Web REST

- L'architecture orientée REST comprend quatre concepts et quatre principes.
- Les quatre concepts :
  - Les *ressources* : ce sont les objets que l'on gère dans une API REST ;
  - Les *URI (nom) des ressources* : une URI est le moyen universel d'identifier une ressource ;
  - Les *représentations* : une ressource peut avoir plusieurs représentations (comme : HTML, XML, ATOM, CSV, …), qui est déterminée, en HTTP, par la ou les valeurs dans le champ *Accept* de la requête ;
  - Les *liens* entre les ressources, par l'intermédiaire des URI présentes dans les représentations.

**Exemple Fichier ATOM :**

```
<?xml version="1.0" encoding="utf-8"?> <feed
xmlns="http://www.w3.org/2005/Atom"> <title>Fil d'exemple</title>
<subtitle>Un titre secondaire.</subtitle> <link href="http://example.org/"/>
<updated>2010-05-13T18:30:02Z</updated> <author> <name>Paul
Martin</name> <email>paulmartin@example.com</email> </author>
<id>urn:uuid:60a76c80-d399-11d9-b91C-0003939e0af6</id> <entry>
<title>Des robots propulsés par Atom deviennent fous</title> <link
href="http://example.org/2003/12/13/atom03"/> <id>urn:uuid:1225c695-cfb8-
4ebb-aaaa-80da344efa6a</id> <updated>2010-04-01T18:30:02Z</updated>
<summary>Poisson d'avril !</summary> </entry> </feed>
```

# Services Web REST

- Dans la pratique
  - On conçoit un modèle axé sur des *ressources*. Les ressources qui correspondent à des objets métiers sont les plus faciles à modéliser, puis au besoin, on conçoit d'autres ressources plus conceptuelles, comme celles qui représentent des transactions, voire toute l'application.

# Services Web REST

- Dans la pratique
  - Il y a également des ressources qui ne correspondent pas à ces données, comme par exemple une requête de recherche ;
  - Ces ressources doivent être *sans état*, c'est-à-dire qu'une requête doit comprendre toutes les informations nécessaires à son traitement. Il n'y a pas d'information d'état en session côté serveur. Il peut, par contre, y avoir des états et une notion de session côté client, mais cela ne passe pas dans les requêtes ;

# Services Web REST

- Dans la pratique
  - On réalise une conception de *template URI*, c'est-à-dire d'URI paramétriques pour déterminer l'ensemble des URI de ressources de même nature. Par exemple ici l'URI représentant un contenu d'identifiant c_4000 est :
    - [/<context-path>]/rest/data/c_4000
  - Le template URI est alors :
    - [/<context-path>]/rest/data/{id}

# Services Web REST

- Dans la pratique
  - L'interface uniforme implique que la *méthode HTTP* d'une requête indique la nature de l'opération sur la ressource :
    - GET : récupération d'une représentation d'une ressource ;
    - PUT : création ou modification de la ressource que l'on indique dans l'URI ;
    - DELETE : suppression de la ressource indiquée dans l'URI ;
    - POST : autres type d'actions. On utilise cette méthode, en particulier, pour créer une ressource pour laquelle (et c'est le cas en général) c'est le serveur qui détermine l'URI. Dans le cas où une création de ressource est faite suite à un POST, l'URL de la nouvelle ressource est indiquée dans l'en-tête de réponse *Location*.

# Services Web REST

- Exemple del.icio.us
  - http://www.louizi.com/cours/restfully-delicious.html
  - http://www.louizi.com/cours/RESTFUL.pdf
  - https://delicious.com/rss

# SSO

# Scénario

Lors d'un voyage

- Se connecter pour réserver un billet d'avion
- Se connecter pour réserver une chambre d'hôtel
- Se connecter pour louer une voiture

- Se connecter plusieurs fois est source de soucis
- N’est-il pas possible de se connecter une seule fois et accéder à toutes les ressources?
- Le Single Sign-On peut répondre à cette question.

# Introduction

- Définition du single sign-on
- Comment ça marche
- Quelques systèmes single sign-on
  - SAML
  - Microsoft passport
  - Facebook Connect / Twitter Sign In

# Définitions de Single Sign-On (SSO) sur le Web:

**Les utilisateurs se connectent à un site une seule et unique fois et un accès leur est accordé à une ou plusieurs applications se trouvant sur un pour plusieurs domaines.**

[www.cafesoft.com/support/security/glossary.html]

**Un mécanisme pour vérifier l'identité d'un utilisateur de plusieurs applications à travers une seule authentification. WebSphere Portal Server utilise la Java Authentication et les Authorization Services pour assurer le single sign-on.**

[www.ibm.com/software/webservers/portal/library/v12/InfoCenter/wps/glossary.html]

**Un seul login procure accès à toutes les ressources du réseau.**

[www.suliscommunication.com/language/ecommerce/ebus3.htm]

# Single Sign-On (SSO)

Il peut être représenté de deux manières différentes. La première consiste en la relation client / serveur, la deuxième est dans le domaine e-commerce concerné.

# Dans la relation Client / Serveur

- “Dans chaque relation **client/serveur**, single sign-on est un processus d ‘**authentication** session/utilisateur qui permet à un utilisateur de saisir un login et un mot de passer afin d’accéder à plusieurs applications.”

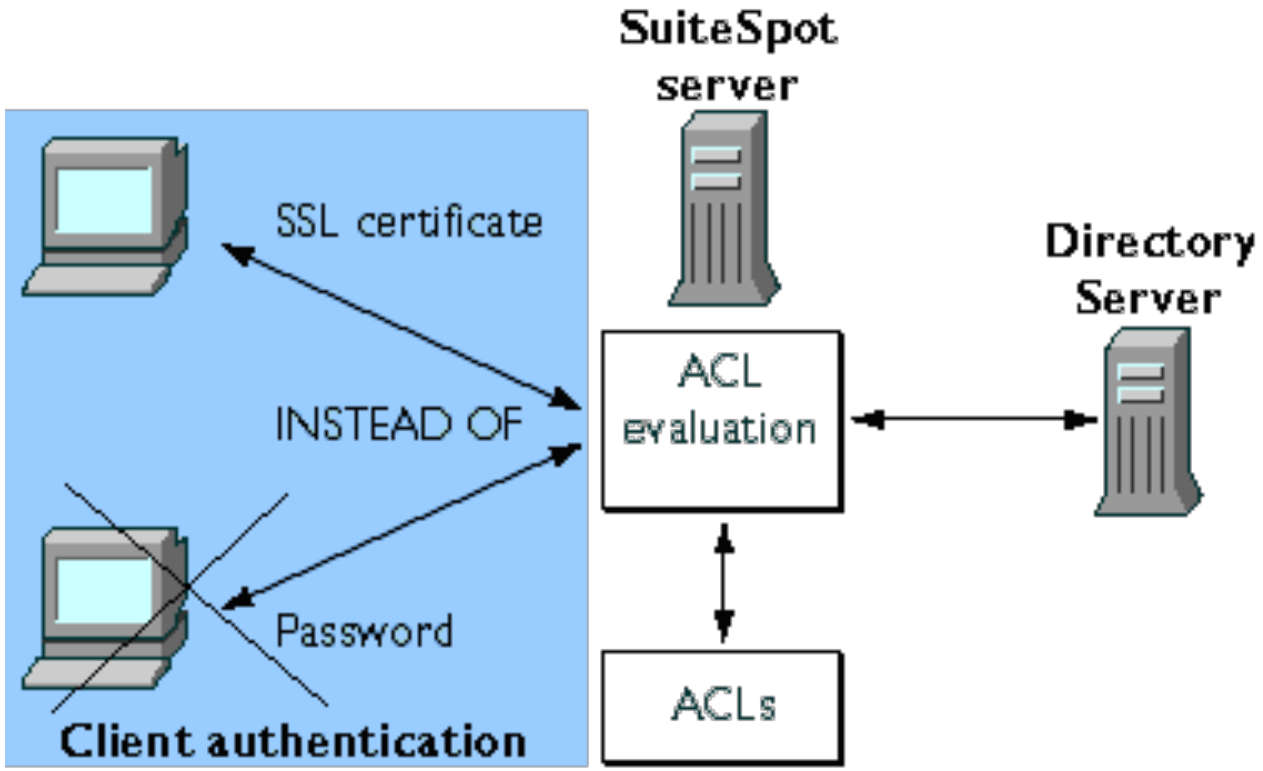
[http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci340859,00.html ]

# Dans le E-commerce

- “Dans le e-commerce, le single sign-on (souvent désigné comme SSO) est destiné à centraliser l’information financière de l’utilisateur dans un seul serveur, non seulement pour faciliter la tâche à l’utilisatuer, mais surtout pour offrir une sécurité accrue en limitant le nombre de fois où l’utilisateur saisit des informations sensibles et confidentielles (numéro de carte de bancaire par exemple) pour la facturation.”

[http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci340859,00.html ]

# Comment ça marche?



# Synchronisation des mots de passe

- Le processus de synchronisation des mots de passe consiste à changer chaque mot de passe des différentes applications par une seule et unique valeur. Une fois le logiciel de synchronisation des mots de passe installé, l'utilisateur va saisir le même mot de passe quand il se connecte à chacun des systèmes synchronisés (email, agenda, système financier…).

# Synchronisation des mots de passe VS Single sign-on

| | Synchronisation des mots de passe | Single Sign-on |
|---|---|---|
| Processus | Changer simplement le mot de passe pour toutes les applications. L'utilisateur continue à se loguer à chaque application séparément, mais en utilisant le même mot de passe. | Un seul nom d'utilisateur et un seul mot de passe pour se loguer à un site, l'authentification client à d'autres sites étant faite par un serveur spécifique. |
| Connexion | Plusieurs fois, suivant l'application | Une seule fois pour tous les domaines |

# Synchronisation des mots de passe VS Single sign-on

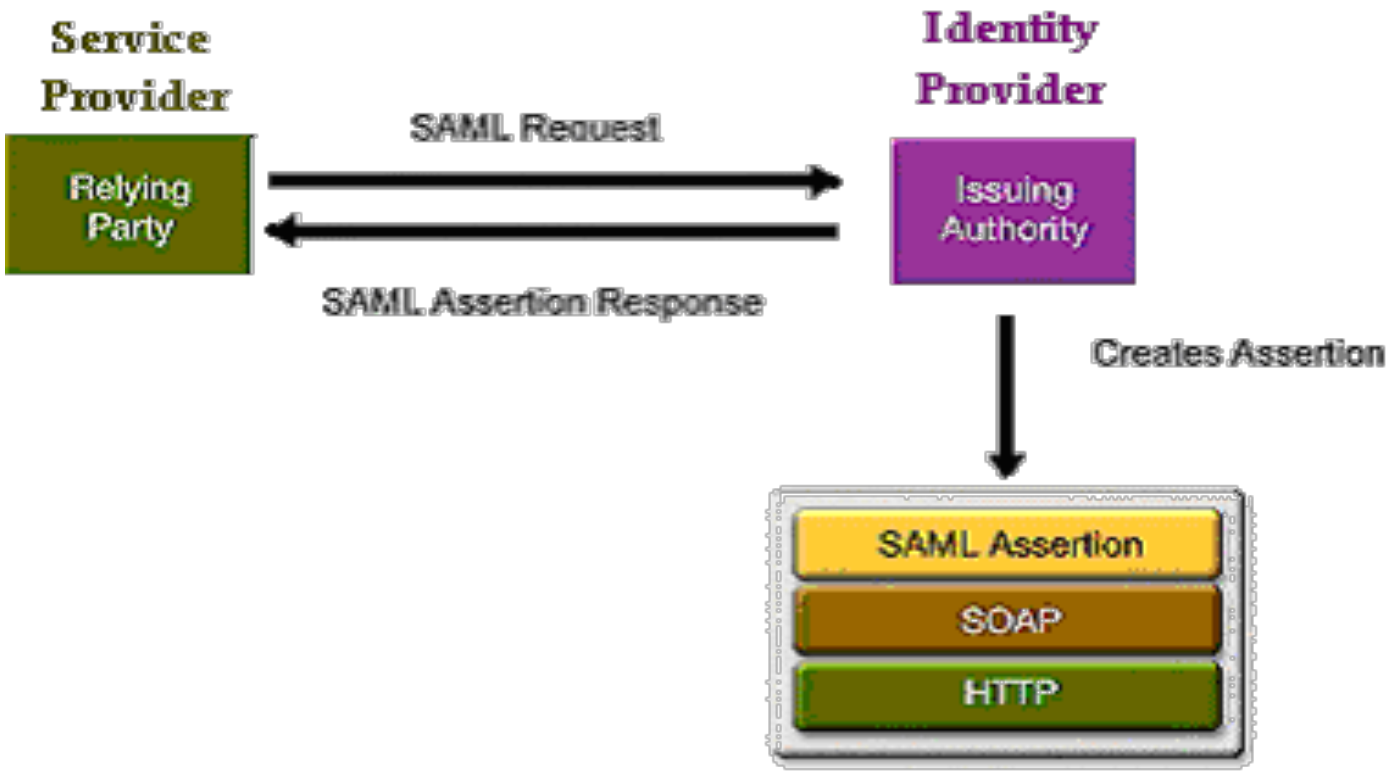
| | | |
|---|---|---|
| Gérer les données d'accès (accréditation) | Gérer les mots de passe uniquement | Utilisation d'un protocole spécifique pour gérer l'authentification du client et ses informations secrètes |
| Mot de passe faible | Correspond seulement à la politique du système le plus faible | Un seul mot de passe uniquement, peut être très sécurisée |
| Sécurité | Une fois une application compromise, toutes les autres seront accessibles, ainsi que les données confidentielles | Peut encoder les données sensibles et les envoyer via un canal SSL |

# SAML

C’est quoi?

SAML (Security Assertion Markup Language) est un framework XML pour l’échange des informations de sécurité sur Internet.

# Comment ça marche?




- 1. Le fournisseur d'accès reçoit la requête du client, et l'envoie à l'Identity Provider qui se chargera de l'authentification du client.
- 2.L'Identity provider autorise le client, crée l'assertion , et la renvoie au fournisseur d'accès. Les assertions SAML peuvent avoir en plus une entête SOAP et passer via le protocole HTTP.

# Requête du fournisseur d'accès

- Voici un exemple d'une requête compatible SAML qui est envoyée par un fournisseur d'accès demandant une authentification par mot de passe à l'identity provider

```
<samlp: Request ...>
<samlp: AttributeQuery>
<saml: Subject>
<saml: NameIdentifier SecurityDomain="sun. com"
  Name="rimap"/>
</ saml: Subject>
<saml: AttributeDesignator
  AttributeName="Employee_ ID"
  AttributeNamespace="sun. com">
</ saml: AttributeDesignator>
</ samlp: AttributeQuery>
</ samlp: Request>
```

# Réponse de l'identity provider

- En réponse, les autorités d'émission affirment que le sujet (S) a été authentifié par (M) au moment (T).

```
<samlp: Response
MajorVersion="1" MinorVersion="0"
RequestID="128.14.234.20.90123456"
InResponseTo="123.45.678.90.12345678"
StatusCode="/features/2002/05/Success">
 <saml: Assertion MajorVersion="1" MinorVersion="0"
  AssertionID="123.45.678.90.12345678" Issuer="Sun Microsystems,
  Inc." IssueInstant="2011- 01- 14T10: 00: 23Z">
 <saml: Conditions NotBefore="2002- 01- 14T10: 00: 30Z"
  NotAfter="2011- 01- 14T10: 15: 00Z" />
   <saml: AuthenticationStatement AuthenticationMethod="Password"
  AuthenticationInstant="2011- 01- 14T10: 00: 20Z">
    <saml: Subject>
      <saml: NameIdentifier SecurityDomain="sun. com" Name="rimap"
  />
    </ saml: Subject>
  </ saml: AuthenticationStatement>
 </ saml: Assertion>
</ samlp: Response>
```

# De quoi est composé SAML

- Assertions
- Protocoles de requêtes/réponses
- Contraintes (ou Bindings) (La méthode SOAP-over-HTTP pour transporter des requêtes et des réponses SAML)
- Profils (Pour  intégrer et extraire des assertions SAML dans un framework ou dans un protocole)

.net Passport

# .NET Passport

- Microsoft® .NET Passport
  - Service Passport single sign in

Passport fournit aux utilisateurs enregistrés un ticket électronique. Avec ce ticket, les utilisateurs sont autorisés à accéder à des pages des sites participants.

.net Passport

# .NET Passport

- Une implémentation du système Single Sign-On, basée sur le mécanisme des cookies.
- Emploi de techniques de prévention des attaques

  - **Captcha** différenciant les humains des machines

  - **Secure Sockets Layer (SSL)**

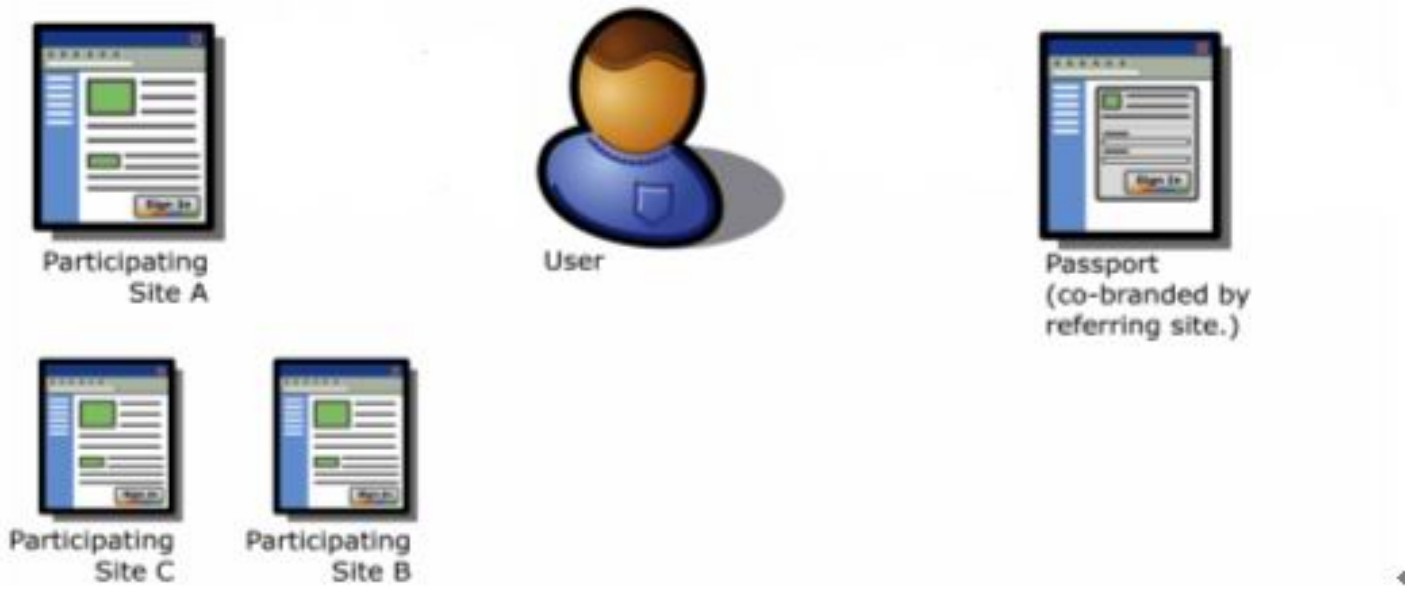
# .NET Passport

- Processus d'enregistrement
  - Information stockée dans un compte Passport
  - Captcha
  - Validation E-mail
- Processus d'authentication
  - Cookies écrits par passport
  - Navigation dans un autre site participant
  - Secure Sockets Layer (SSL)

.net Passport

# Service Passport

- Trois éléments importants dans le système



# Captcha

.net Passport

# Secure Sockets Layer (SSL)

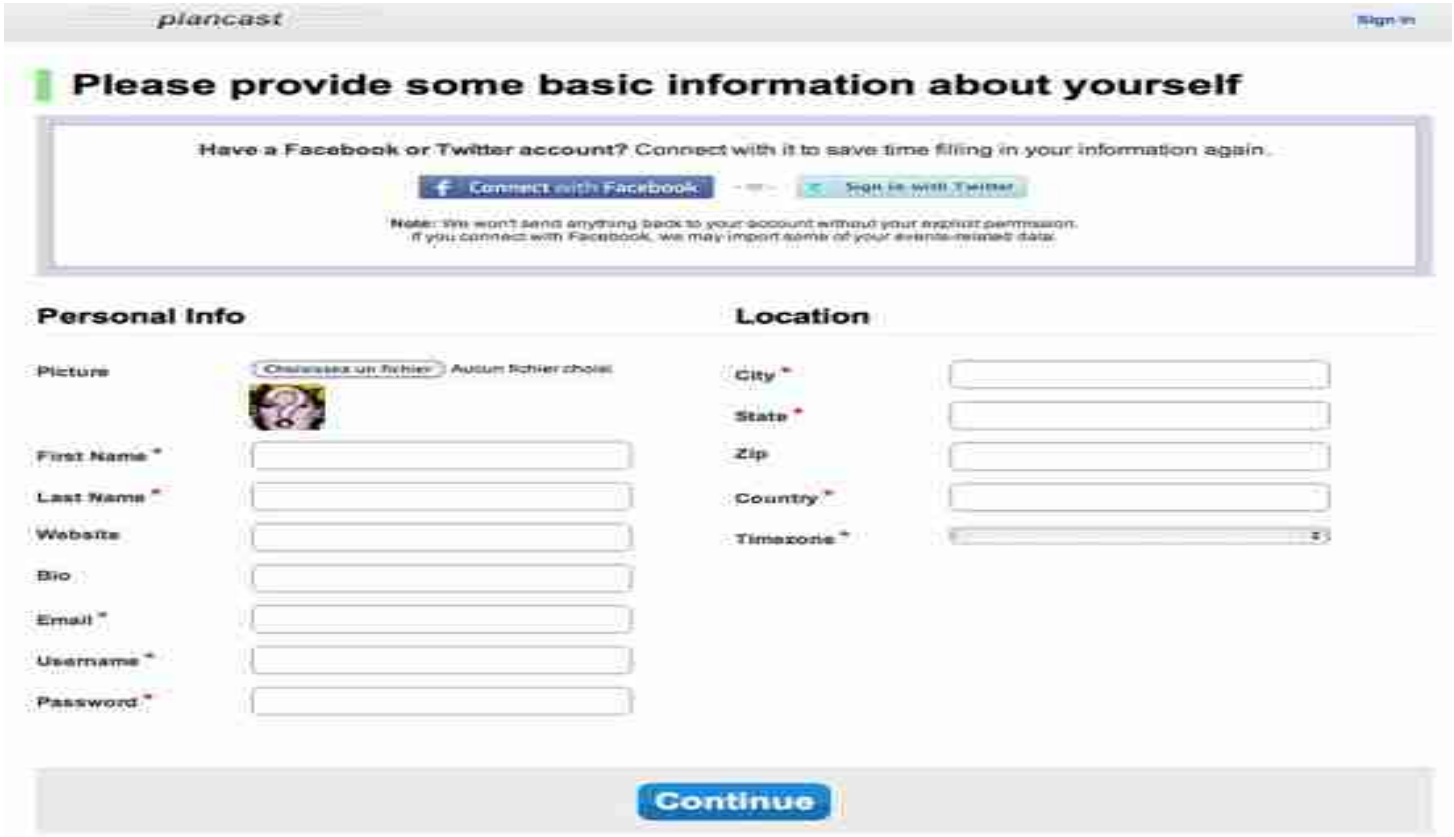
- **Secure Sockets Layer (SSL)** (devenu TLS (Transport Layer Security))

  est un protocole de sécurisation des échanges sur Internet qui fournit encodage des données, authentification serveur et intégrité des messages pour une connexion à Internet.

  - Utilisation de la cryptographie à clé publique pour l'authentification
  - Mécanisme des certificats

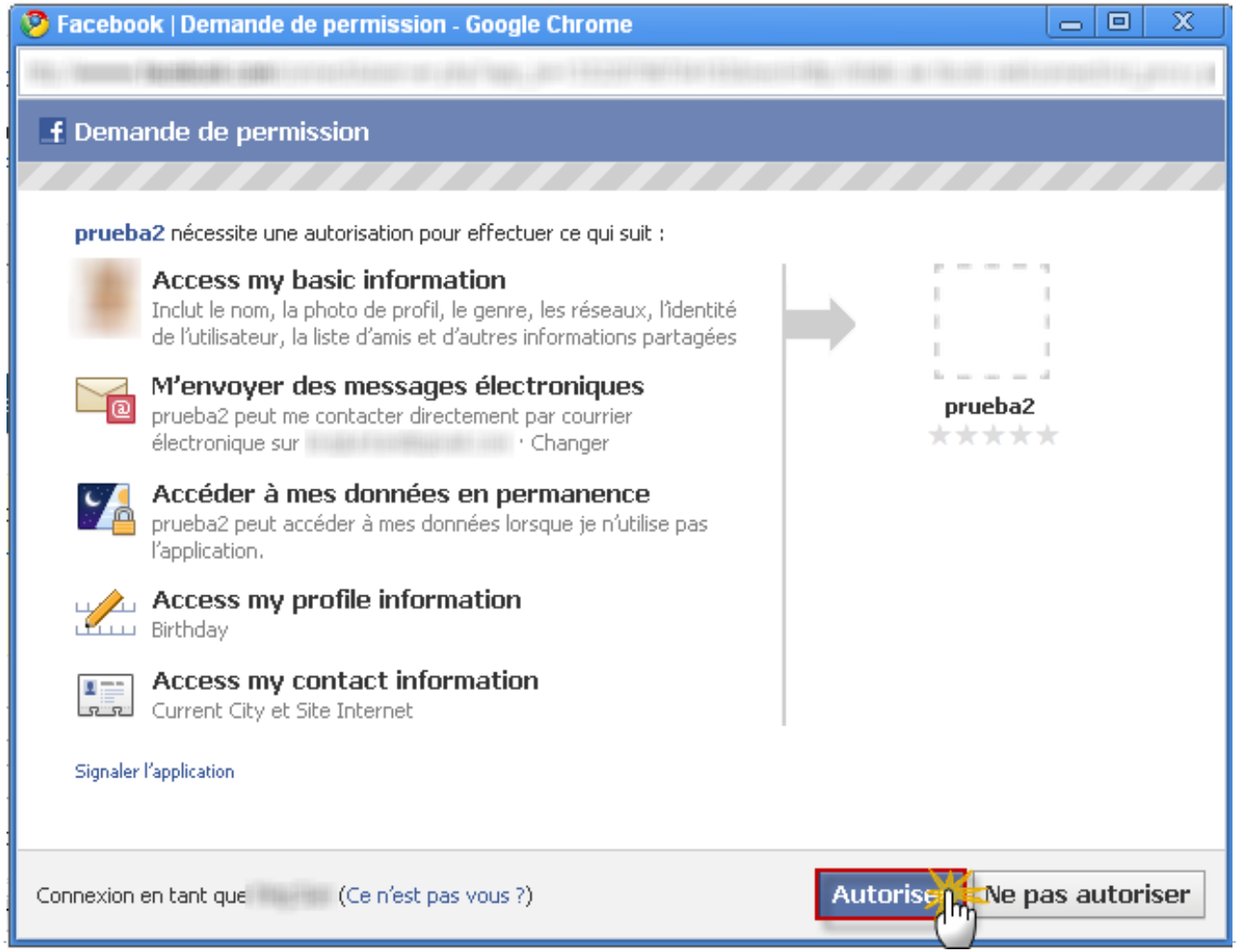
# Facebook Connect / Twitter Sign In

- **Modèle SSO**



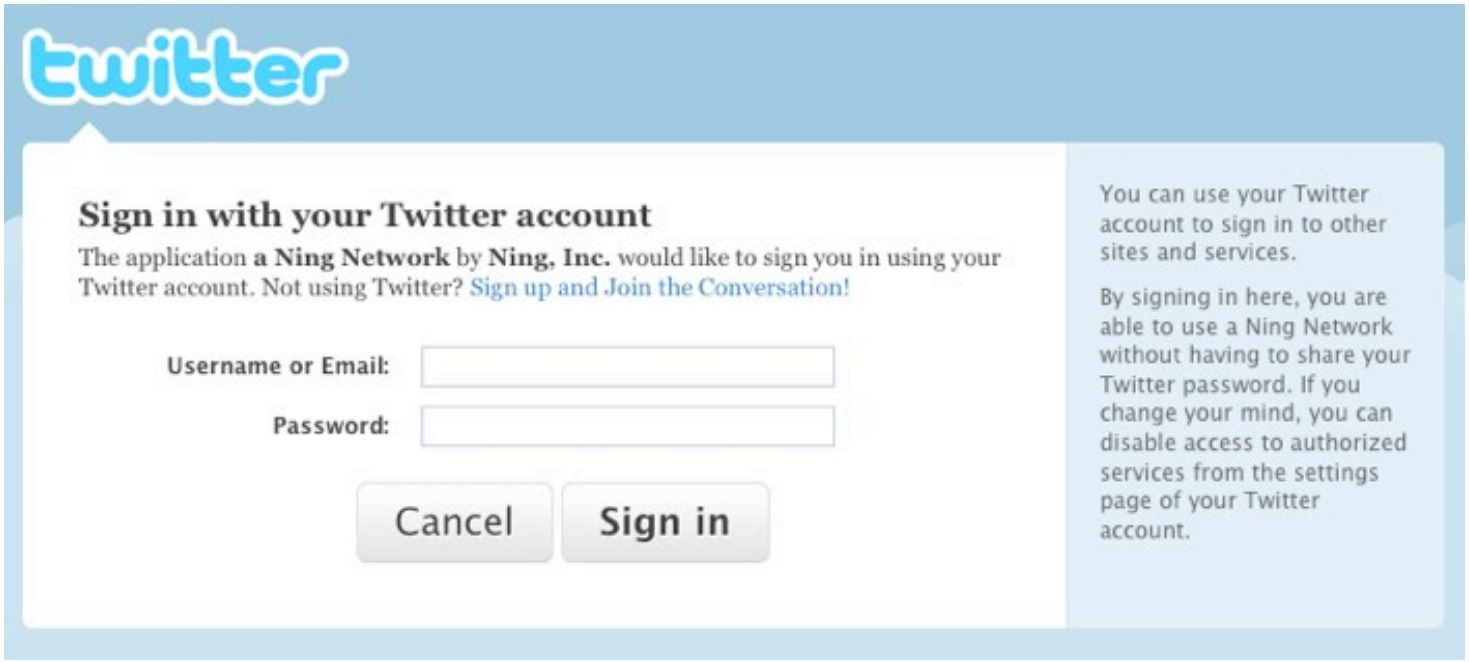
# Facebook Connect / Twitter Sign In

- **Modèle SSO**



# Facebook Connect / Twitter Sign In

- **Modèle SSO**



# Google!

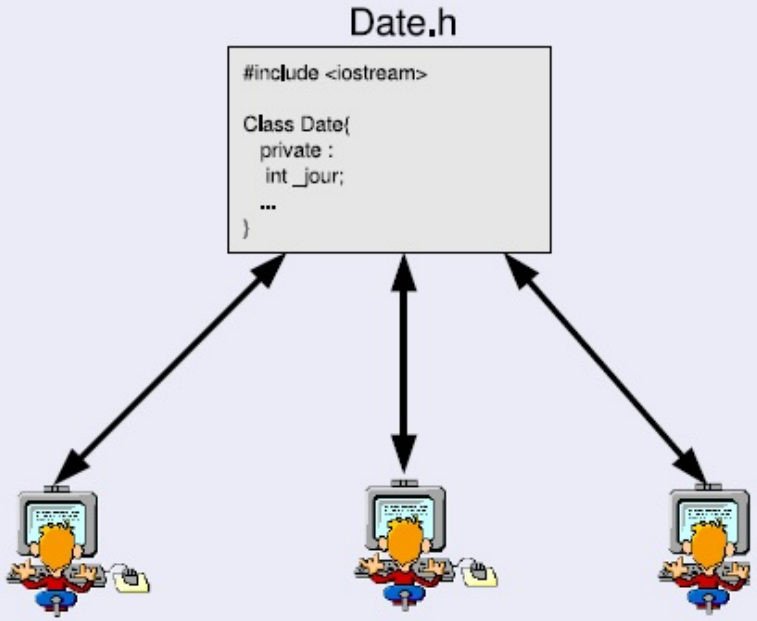
- **SAML**
- **Google Apps**

# Outils collaboratifs

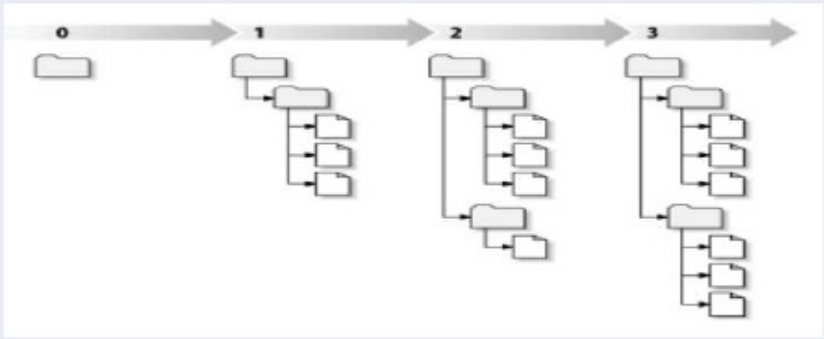
# Les systèmes de gestion des versions

# Objectifs d'un Système de Gestion des Versions

## Travailler à plusieurs



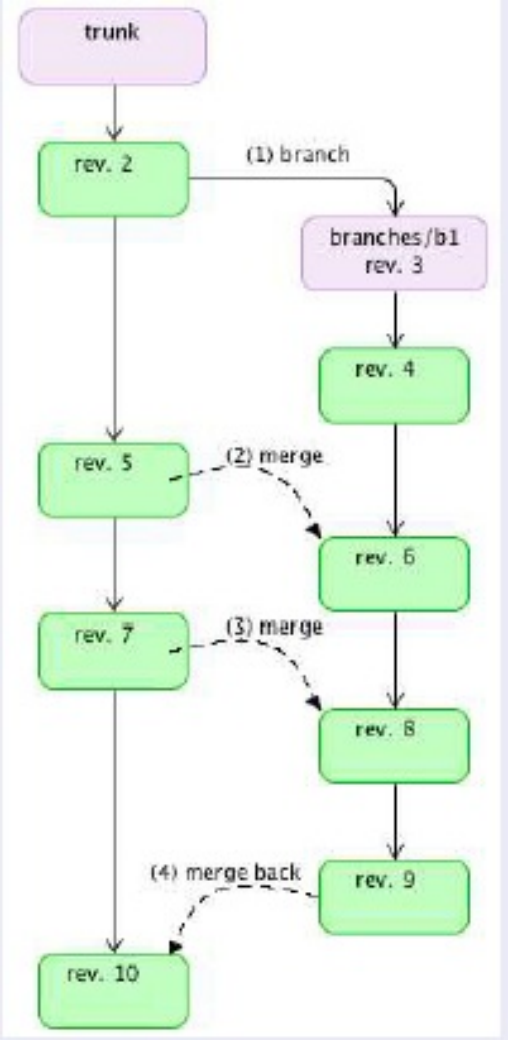

## Conserver l'historique




- Pouvoir revenir en arrière
- Qui a modifié pour la dernière fois ce fichier ?
- Quelles sont les différences entre 2 versions de ce fichier ?
- Quelle est la version du 15 mars 2007 ?

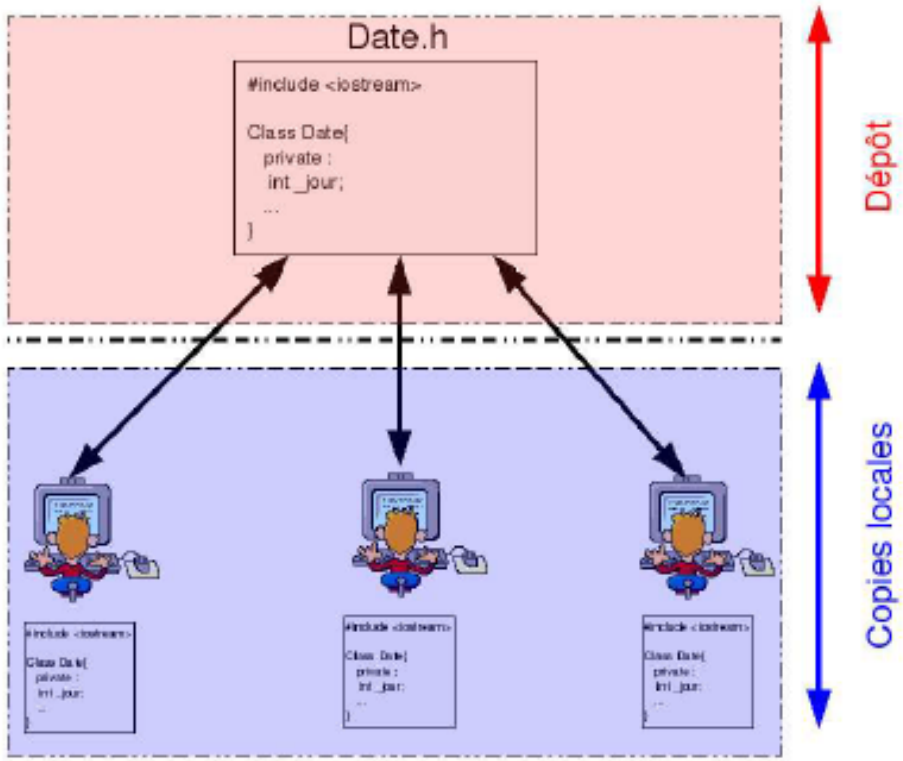
# Et aussi…

- **La gestion des branches**
  - Objectif : mener en parallèle plusieurs versions (stable, testing…)
- **L'utilisation des tags**
  - Objectif : donner un nom explicite à une version pour pouvoir y accéder facilement



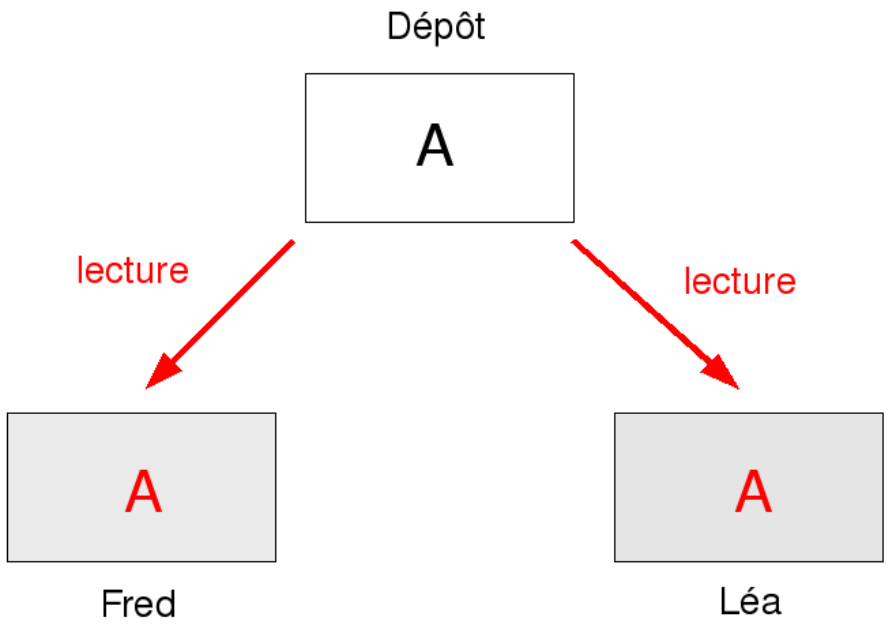
# Principe de base

- **Notion de dépôt et de copie locale**



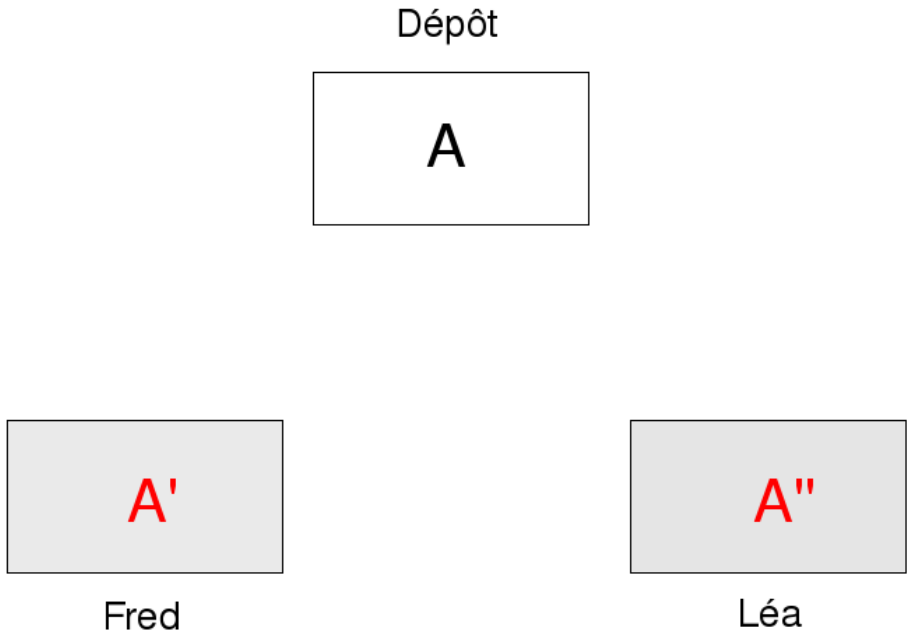

- Les accès (écriture/lecture) se font via le SGV

# Problème



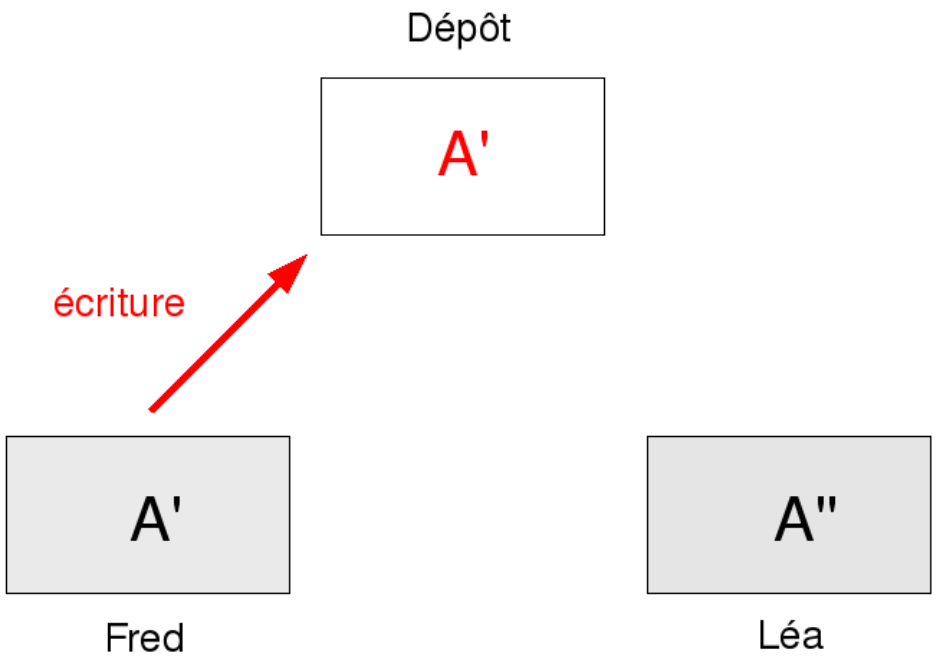

- Fred et Léa accèdent au même fichier et le copient chez eux

# Problème (suite)



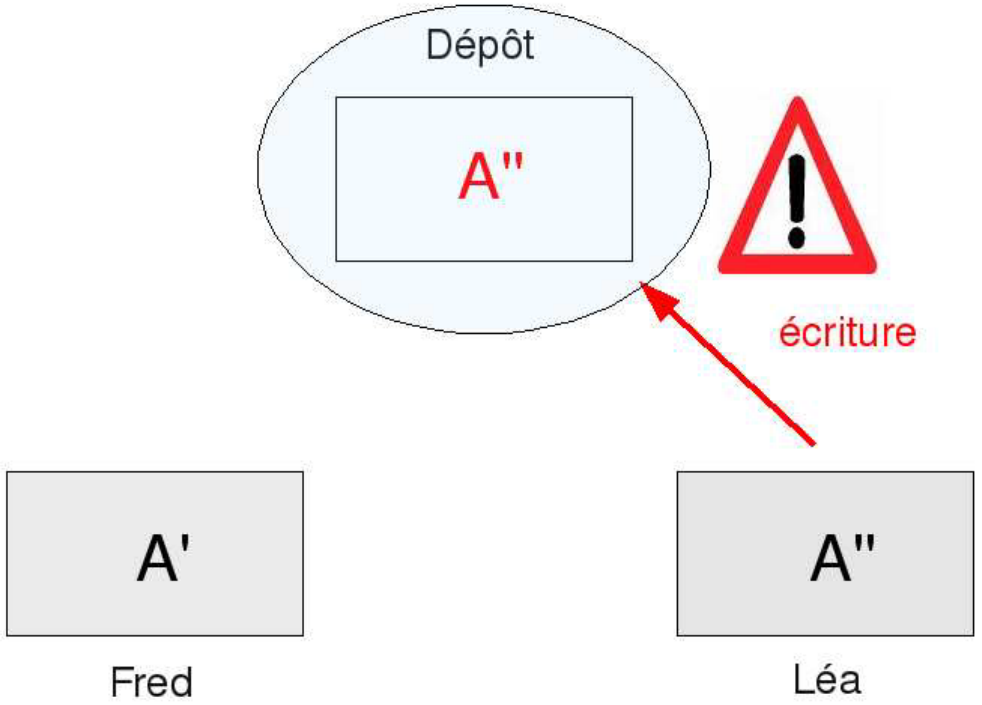

- Fred et Léa font chacun des modifications

# Problème (suite)



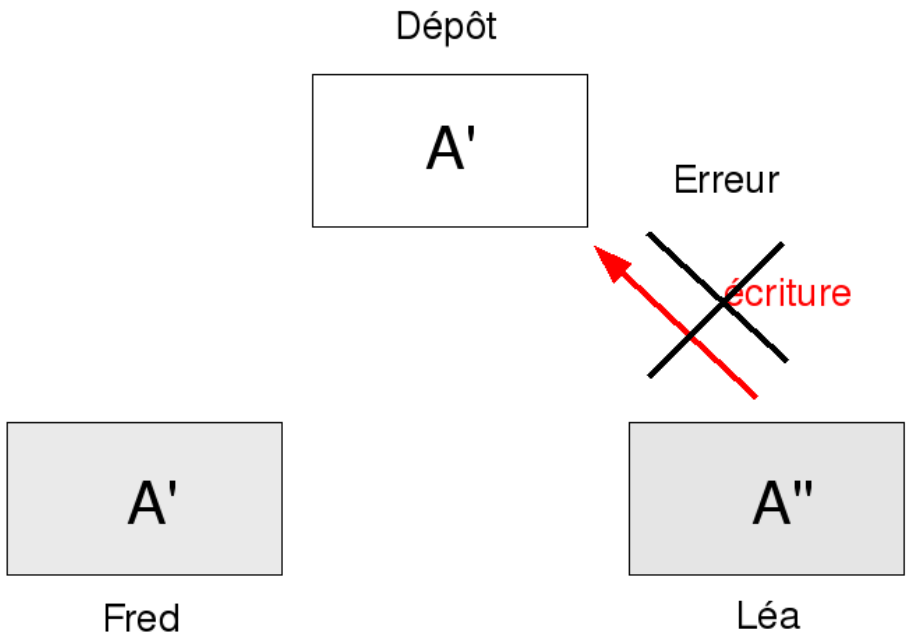

- Fred écrit sur le dépôt

# Problème (suite)



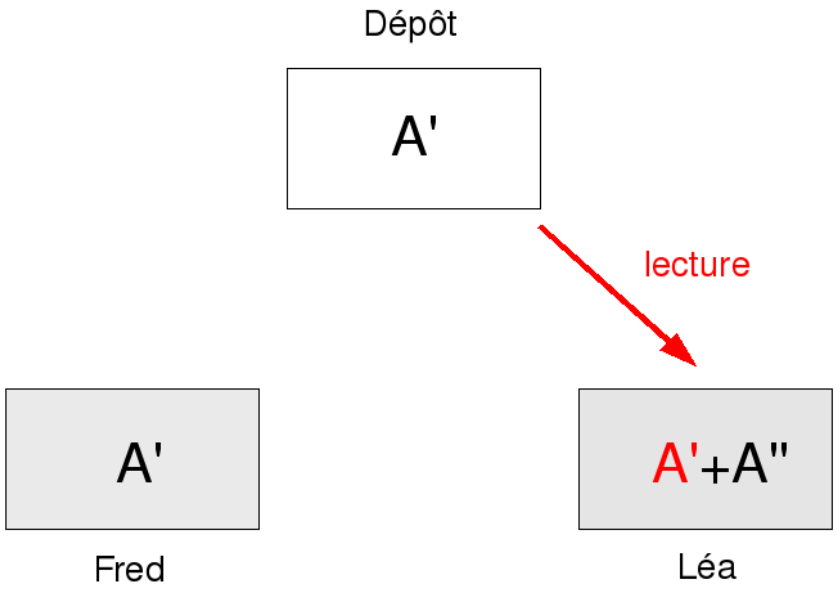

- Léa écrit sur le dépôt en écrasant la version de Fred

# Solution



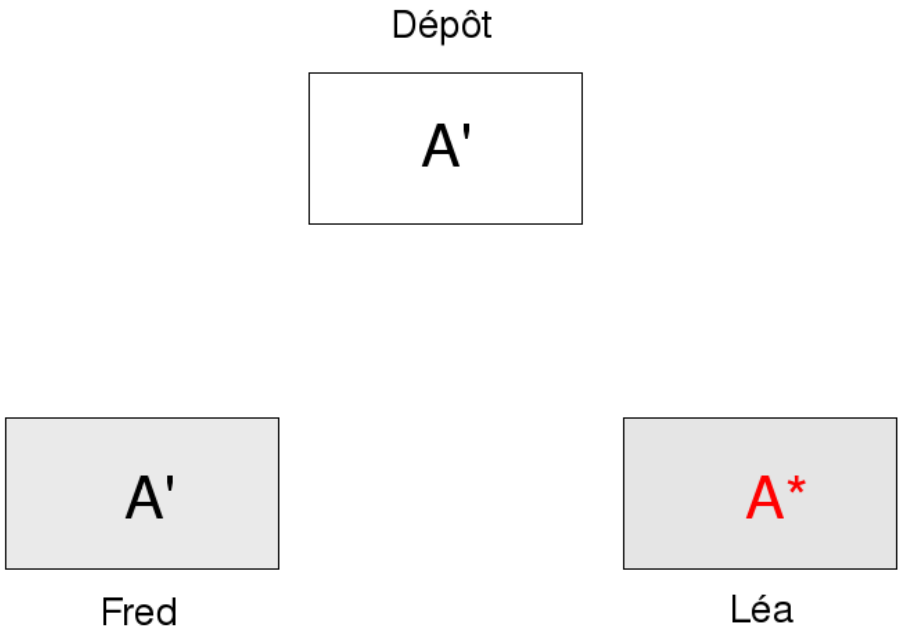

- Léa ne peut pas écrire sur le dépôt car sa version n'est pas à jour

# Solution (suite)



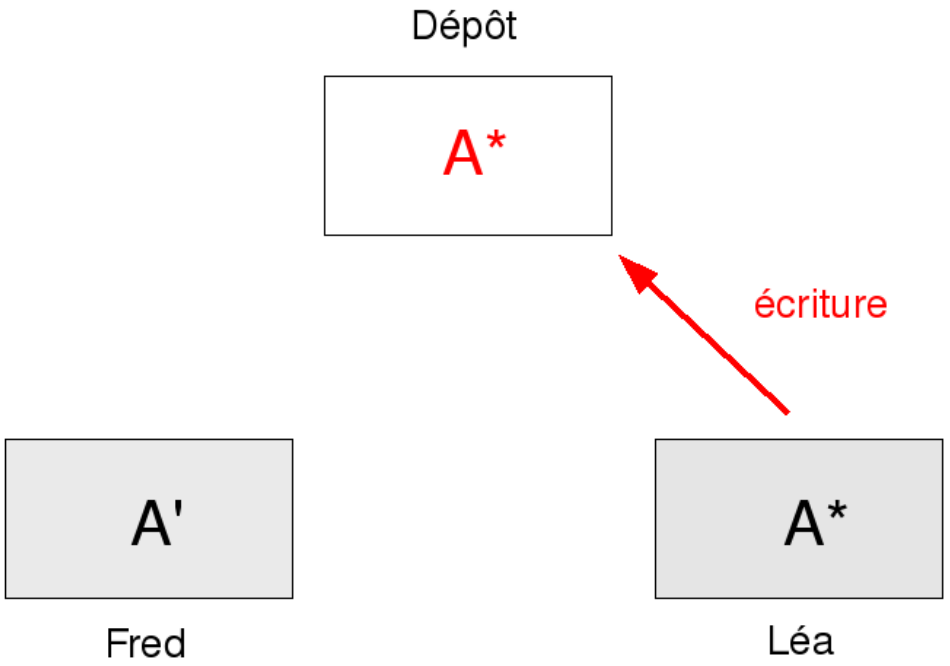

- Léa met à jour : elle récupère la version du dépôt sans perdre ses modifications

# Solution (suite)



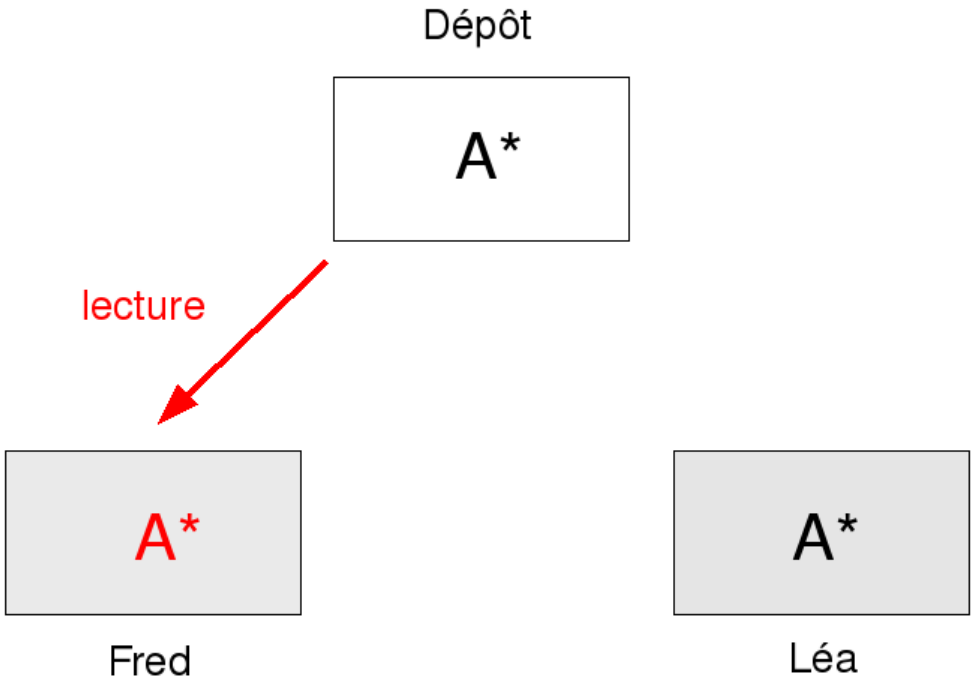

- Léa fusionne la version du dépôt (A') avec sa version (A'')
  A', A'' -> A*

# Solution (suite)




- Léa peut écrire sur le dépôt

# Solution (suite)




- Fred récupère la nouvelle version

# Système de Gestion des Versions (SGV)

- **Un SGV gère le mécanisme de lecture-fusion-écriture**
  - Les demandes de lecture, écriture se font via le SGV
  - La fusion automatique est possible si
    - il s'agit d'un fichier texte (diff)
    - les modifications sont assez éloignées les unes de autres
  - Le SGV conserve l'historique
  - Et aussi : gestion des branches, tags, . . .

# Quel SGV choisir?

- Vaste choix
  - Technologie en pleine évolution
  - De nouveaux systèmes apparaissent régulièrement
- Eléments à prendre en compte
  - Pérennité : systèmes leaders VS systèmes émergents
  - Intégration dans des IDE
  - Proposé par des Forges
  - Interfaces graphiques
  - Portabilité (Multi OS)
  - Sécurité
  - Documentations abondantes
  - Outils connexes

# Subversion

- Un SVG très répandu
  - Documentations très riches, forums actifs
  - Interfaces graphiques
    - Linux : rapidsvn, kdesvn, esvn, Qsvn, …
    - Windows : intégré à l'explorateur via le plugin TortoiseSVN
  - Proposé dans les Forges et intégré dans certains IDE (Eclipse, Kdevelop)

# Subversion

- Le successeur de CVS
  - Reprend le modèle de CVS en comblant certains manques :
    - Renommage et déplacement de fichiers sans perte de l'historique
    - Gestion des répertoires
    - commits atomiques
    - Gestion des metadonnées (ex : permissions)
    - Possibilité de migrer de CVS vers SVN sans perte de l'historique (cvs2svn)
    - Protocoles réseaux sécurisés (HTTPS)

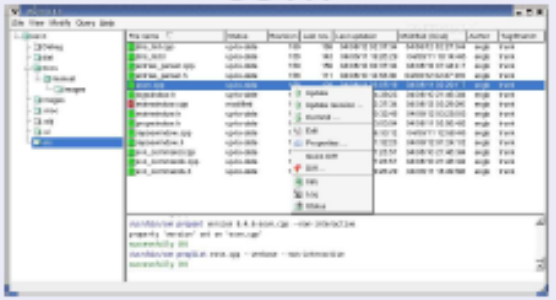
# Subversion

## Principales commandes

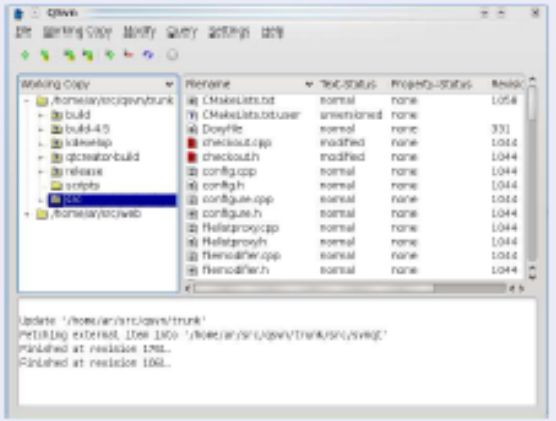
- svnadmin create : créer un nouveau dépôt
- svn import : importer un projet dans le dépôt
- svn checkout : lire tout un projet
- svn update : lire/mettre à jour depuis le dépôt
- svn commit : écrire/modifier le dépôt (nouvelle révision)
- svn status : état de la copie locale
- svn add : ajouter un fichier
- svn rm : enlever un fichier
- svn mv : déplacer un fichier

## Interfaces graphiques

esvn



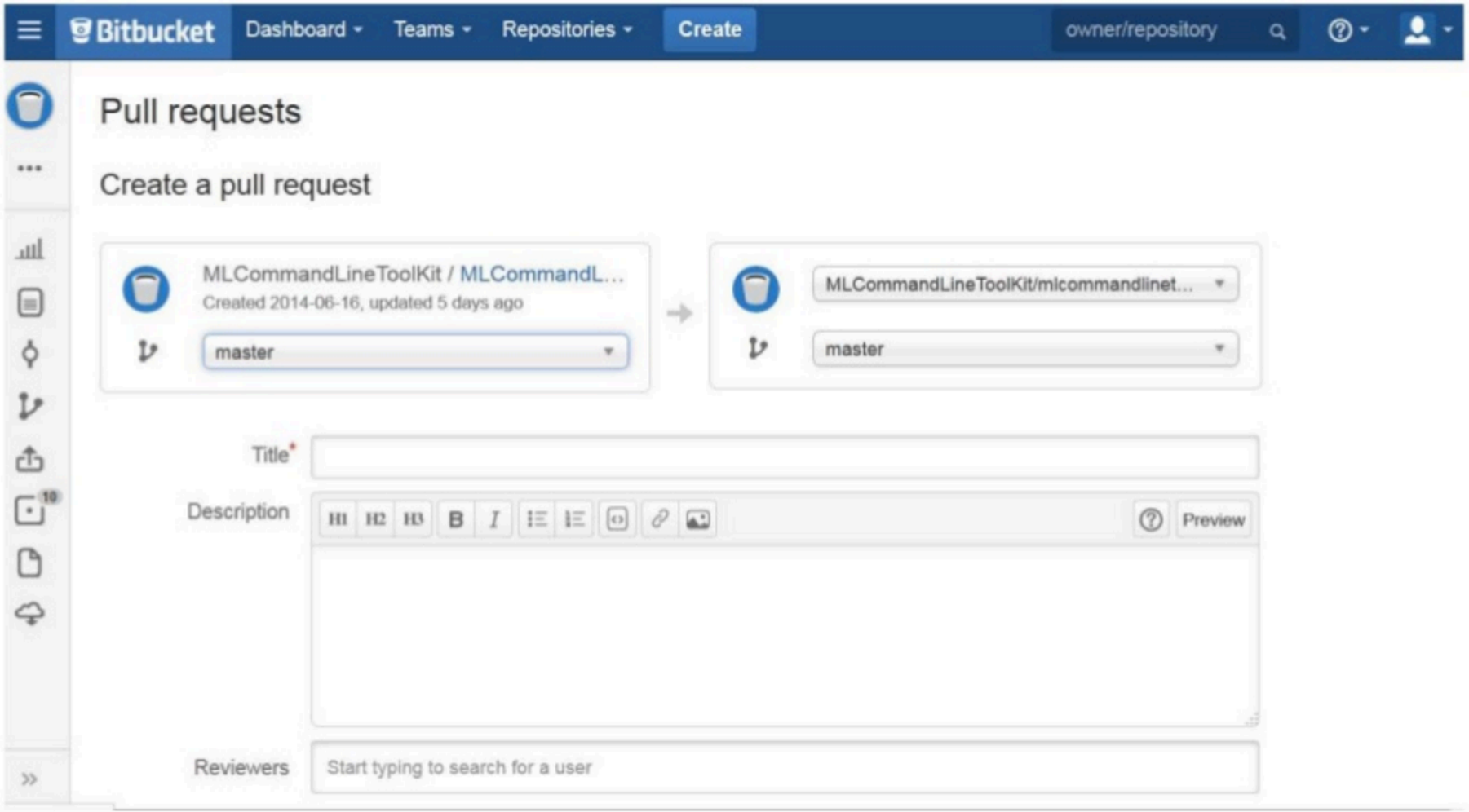
Qsvn



# Git

- Un système de contrôle de version distribué
- Bitbucket : Hébergement en ligne des projets qui utilisent Git. Il est développé en Python et est basé sur le framework Django.
  - Travail en équipe
  - Intégration à JIRA
  - Fork, Code, Pull
  - Etc.

# Bitbucket


Bitbucket
Dashboard
Teams
Repositories
Create
owner/repository
Pull requests
Create a pull request
MLCommandLineToolKit / MLCommandL...
Created 2014-06-16, updated 5 days ago
master
MLCommandLineToolKit/mlcommandlinet...
master
Title*
Description
Preview
Reviewers
Start typing to search for a user

# Les plateformes collaboratives

# Plateformes de gestion des projets / incidents

- L'objectif de la gestion des projets / incidents est la remise en service des applications, dans les délais les plus courts, en minimisant l'impact sur les utilisateurs.

# Un feu ⟺ Un bug

***Les délais de résolution d'un bug sont équivalent aux besoins pour éteindre un feu.***

*Pour éteindre un feu il faut :*

**La 1ère minute**

Au bout de **5 minutes**

Au bout de **1 heure**

Au bout de **6 heures**

AGENCE KARISMATIK

Doc-Aff-02 Affiche sensibilisation détection bug

*Pour corriger un bug il faut :*

Lorsqu'il est détecté par
**le développeur**

**2 minutes**

Lorsqu'il est détecté par
**le chef de projet**

**15 minutes**

Lorsqu'il est détecté par
**le pôle de test**

**1 heure**

Lorsqu'il est détecté par
**le client**

**1/2 journée**

- **Plus un bug est détecté tôt, plus il est rapide à corriger** car le code est frais dans la mémoire du développeur
- **Plus on attend pour le détecter, plus il faudra de temps** au développeur pour se remettre dans le code et le corrigé

***BIEN TESTER = GAGNER DU TEMPS AU FINAL***

# Plateformes de gestion des projets / incidents

- Le cycle de vie de l'incident et les étapes de la Gestion des incidents sont les suivantes :

  - Détection et enregistrement
  - Classification et support initial
  - Investigation
  - Résolution
  - Clôture

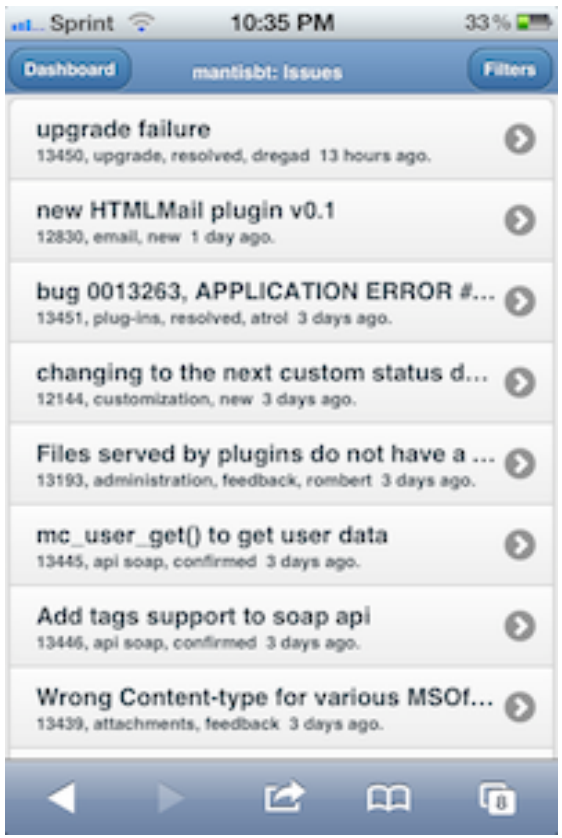
# Plateformes de gestion des projets / incidents

- Des logiciels existent aujourd'hui pour permettre de faciliter cette gestion. Ils offrent des fonctions de prise d'appel, de suivi des appels, d'aide au diagnostic, de statistiques, de capitalisation des solutions, etc...

# Plateformes de gestion des projets / incidents

- **Mantis**





# Plateformes de gestion des projets / incidents

- **Redmine**

# Plateformes de gestion des projets / incidents

- **Redmine** est une application web libre de gestion complète de projet en mode web, développé en Ruby sur la base du framework Ruby on Rails.

# Plateformes de gestion des projets / incidents

- **Ruby** est un langage de programmation libre. Il est interprété, orienté objet et multi-paradigme.

  - Un **paradigme de programmation** est un style fondamental de programmation informatique qui traite de la manière dont les solutions aux problèmes doivent être formulées dans un langage de programmation (à comparer à la méthodologie, qui est une manière de résoudre des problèmes spécifiques de génie logiciel).

# Plateformes de gestion des projets / incidents

- **Exemples de paradigmes (plusieurs familles)**
- **Types de programmation impérative (et dérivés) Programmation impérative, paradigme originel et le plus courant**
  - Programmation structurée, visant à structurer les programmes impératifs pour en supprimer les instructions *goto*
  - Programmation procédurale,
- **Types de programmation orientée objet (et dérivés) Programmation orientée objet, consistant en la définition et l'assemblage de briques logicielles appelées objets**
  - Programmation orientée prototype, qui simplifie et rend plus flexible la programmation orientée objet
  - Programmation orientée classe, à comparer à la Programmation orientée prototype (dans le contexte de la programmation orientée objet)
  - Programmation orientée composant (comme en OLE)
- **Types de programmation déclarative (et dérivés) Programmation déclarative, consistant à déclarer les données du problème, puis à demander au programme de le résoudre**
  - Programmation logique, consistant à exprimer les problèmes et les algorithmes sous forme de prédicats (comme en Prolog)
  - Programmation par contraintes, à comparer à la programmation logique

# Plateformes de gestion des projets / incidents

- **Ruby On Rails**
- également appelé **RoR** ou **Rails** est un framework web libre écrit en Ruby. Il suit le motif de conception Modèle-Vue-Contrôleur aussi nommé MVC. Il permet de créer des applications web rapidement, car il impose une structure au programmeur, et ainsi l'oblige à avoir une logique et une démarche qui favorise la réalisation de l'application. Il ajoute aussi un grand niveau d'abstraction dans la programmation de l'application, grâce à un ensemble de fonctions de haut niveau permettant de se concentrer surtout sur les fonctionnalités plutôt que sur la mécanique autour de ces fonctionnalités.

# Plateformes de gestion des projets / incidents

- **A connaître**
  - **ImageMagick**
    - **ImageMagick** est un logiciel libre, comprenant une bibliothèque, ainsi qu'un ensemble d'utilitaires en ligne de commande, permettant de créer, de convertir, de modifier et d'afficher des images dans un très grand nombre de formats. Les images peuvent être découpées, les couleurs peuvent être modifiées, différents effets peuvent être appliqués aux images, les images peuvent subir des rotations, il est possible d'y inclure du texte, des segments, des polygones, des ellipses et des courbes de Bézier, etc.

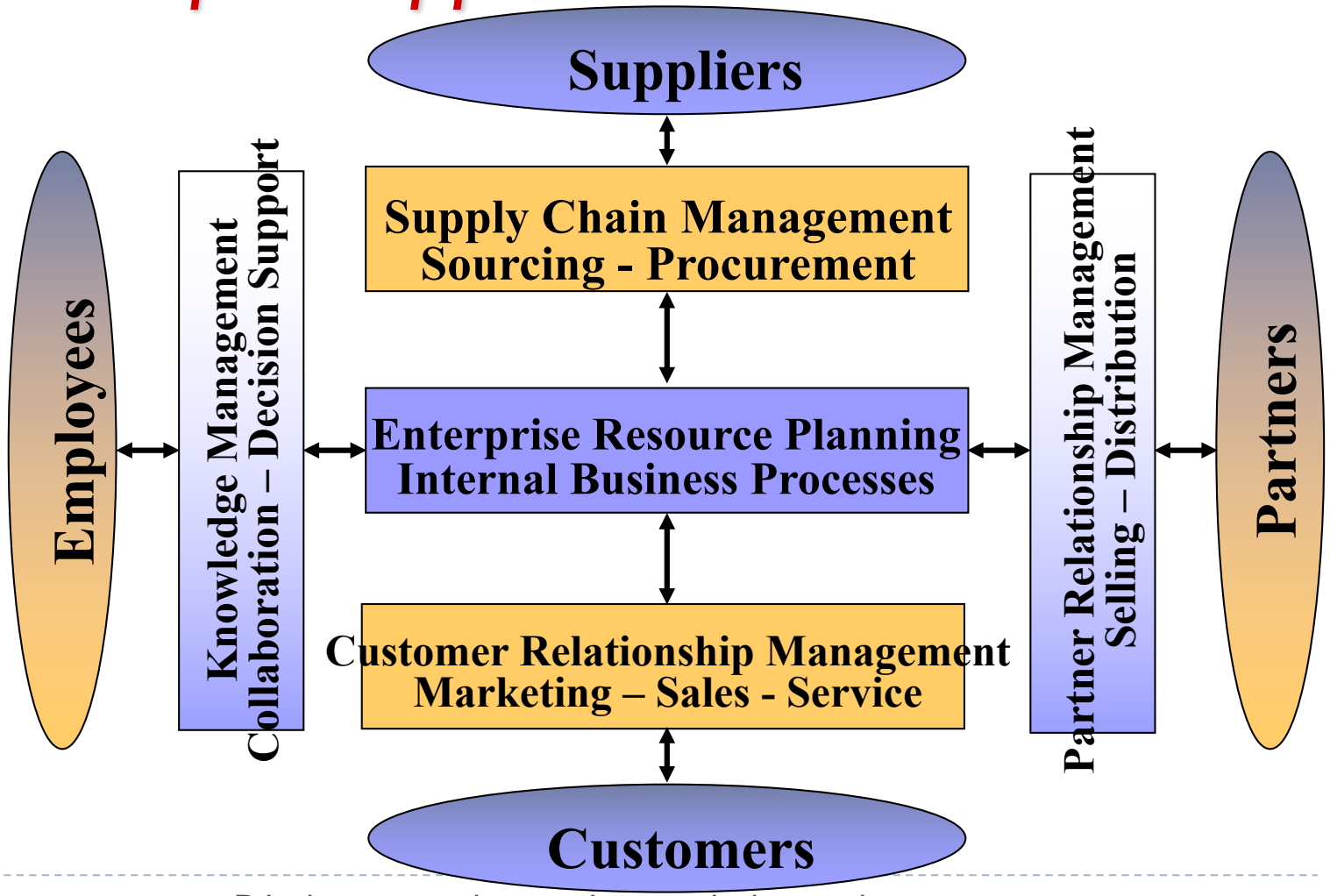
# Plateformes de gestion des projets / incidents

- **A connaître**
  - **GEM**
    - un format de paquet du gestionnaire de paquets pour la bibliothèque de langage de programmation Ruby
  - **Mongrel**
    - **Mongrel** est un serveur HTTP écrit en Ruby et en C. Il a été conçu pour être léger, rapide et sécurisé.
    - C'est un logiciel libre distribué selon les termes de la licence Ruby, compatible avec la licence GNU GPL.

# ERP & CRM

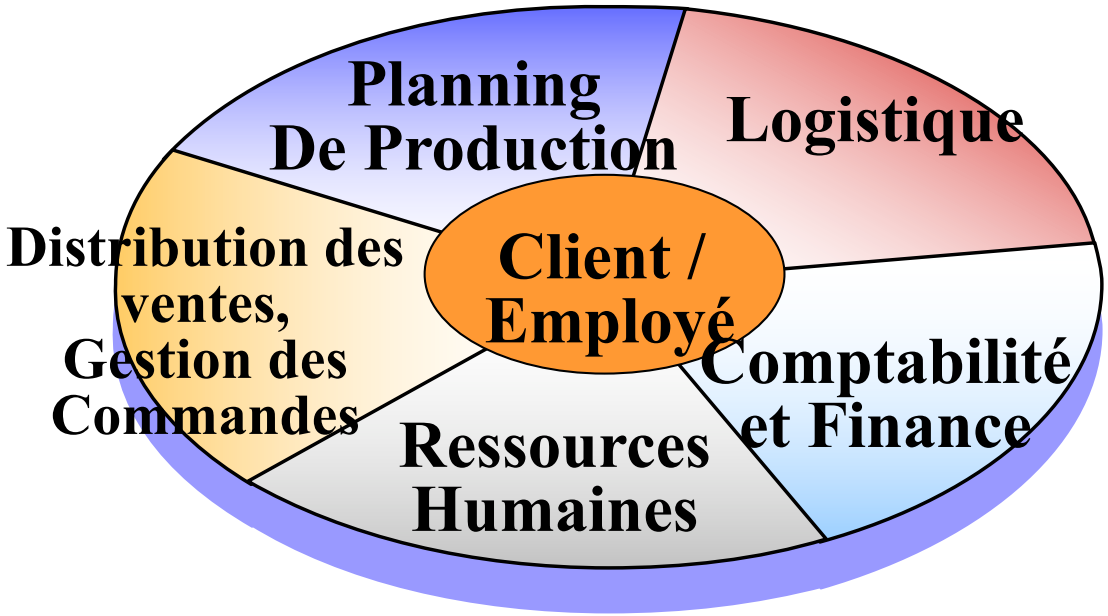
# Enterprise Business Systems

## Enterprise Application Architecture



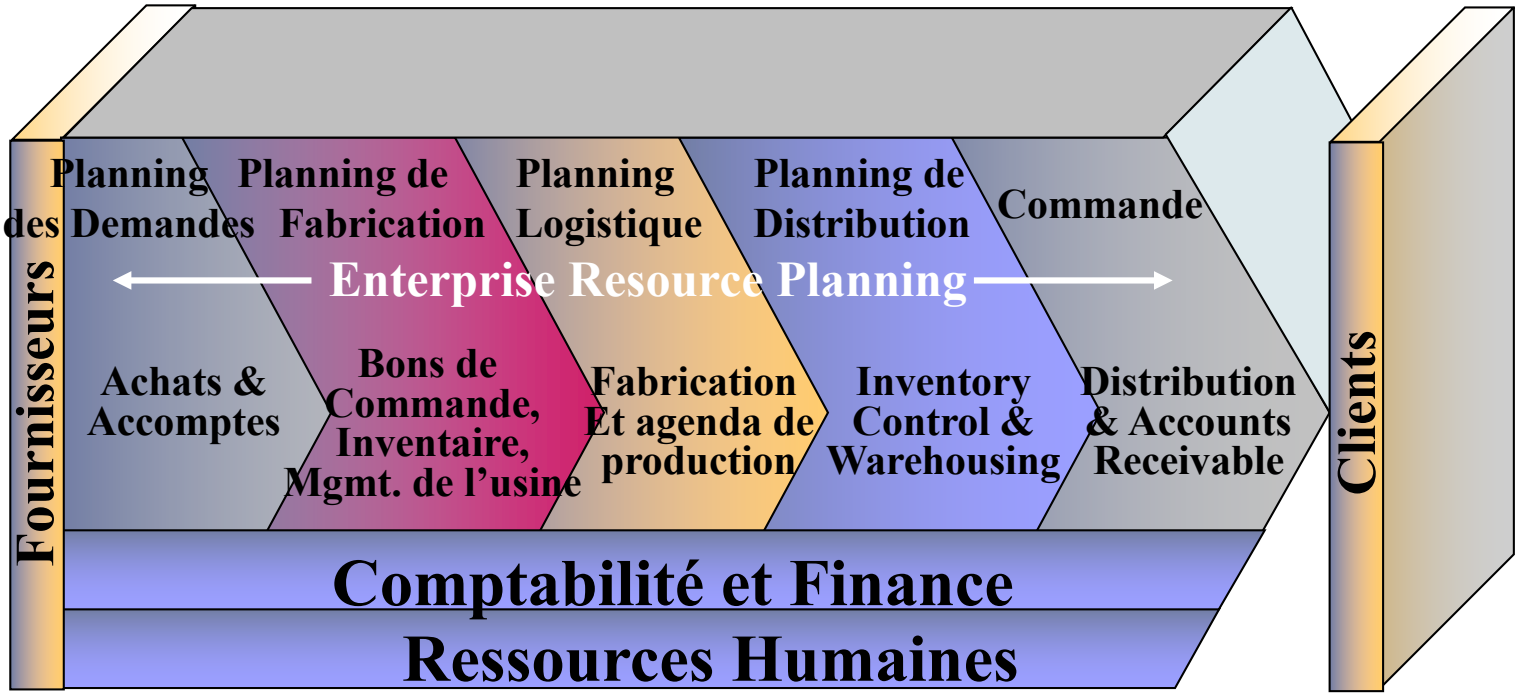
# ERP

- **Enterprise Resource Planning**
  - **C'est le pilier technologique du e-commerce**



# ERP

- **Enterprise Resource Planning**
  - **La valeur commerciale de l'ERP**



# ERP

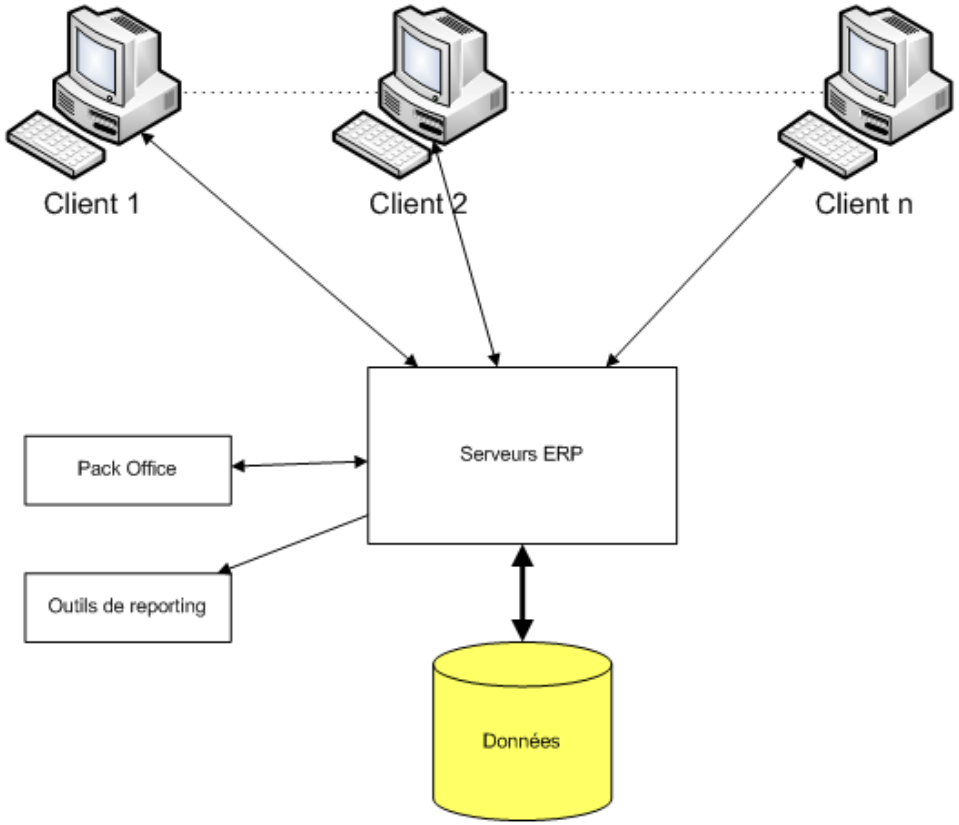
- **Enterprise Resource Planning**
  - **Les avantages**
    - **Qualité et efficacité**
    - **Coûts réduits**
    - **Aide à la décision**
    - **Enterprise Agility**

# ERP

- **Enterprise Resource Planning**
  - **Les inconvénients**
    - **Comprendre la complexité du planning, le développement, formation nécessaire**
    - **Difficulté à impliquer les employés concernés**
    - **Essayer de faire beaucoup très rapidement**
    - **Dépendance par rapport aux entreprises / développeurs de l'ERP**

# ERP

- **Enterprise Resource Planning**
  - **Architecture de base**



# ERP

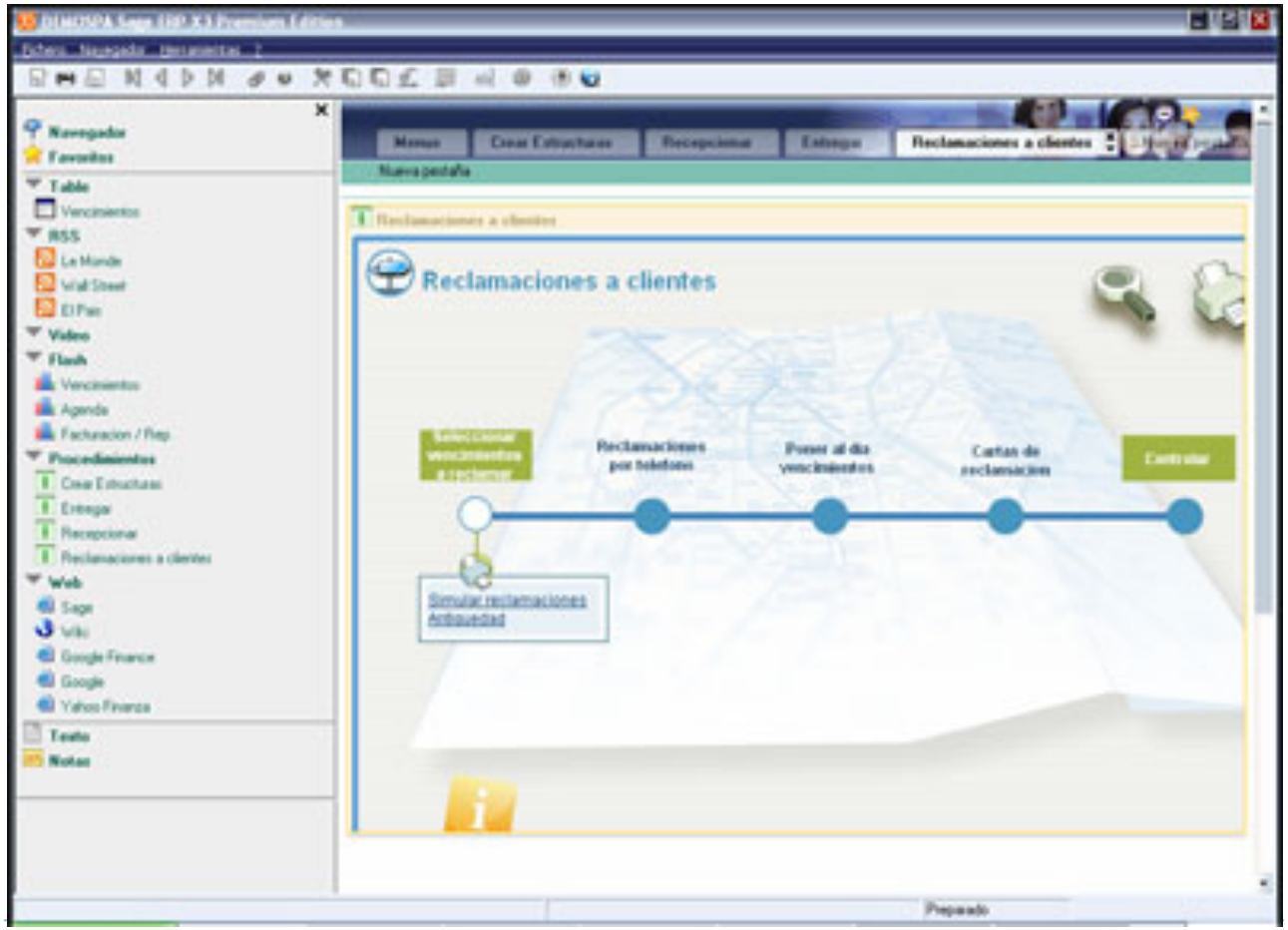
- **Enterprise Resource Planning**
  - **Comment choisir?**
    - **Etendue des fonctionnalités**
    - **Compatibilité avec les SGBD « classiques » : Oracle / PostgreSQL / MySQL**
    - **Support SOAP / XML RPC et WSDL**
    - **Le langage de programmation (Python, PHP, J2EE, Ruby)**

# ERP

- **Enterprise Resource Planning**
  - **Deux familles :**
    - **Propriétaires (Avec Licence)**
    - **Open Source**

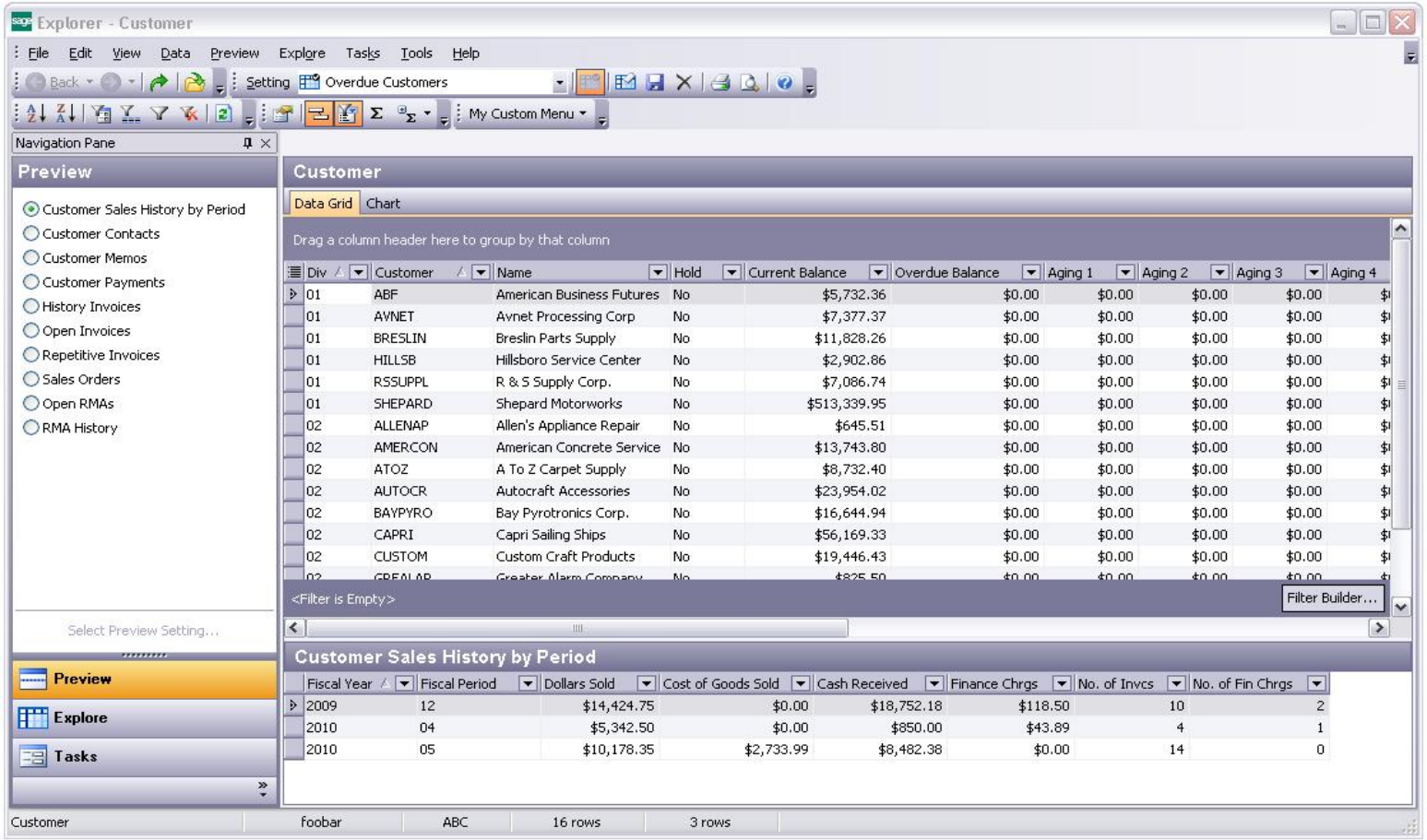
# ERP

- **Enterprise Resource Planning**
  - **ERP Propriétaire (SAGE X3)**



# ERP

- **Enterprise Resource Planning**
  - **ERP Propriétaire (SAGE X3)**

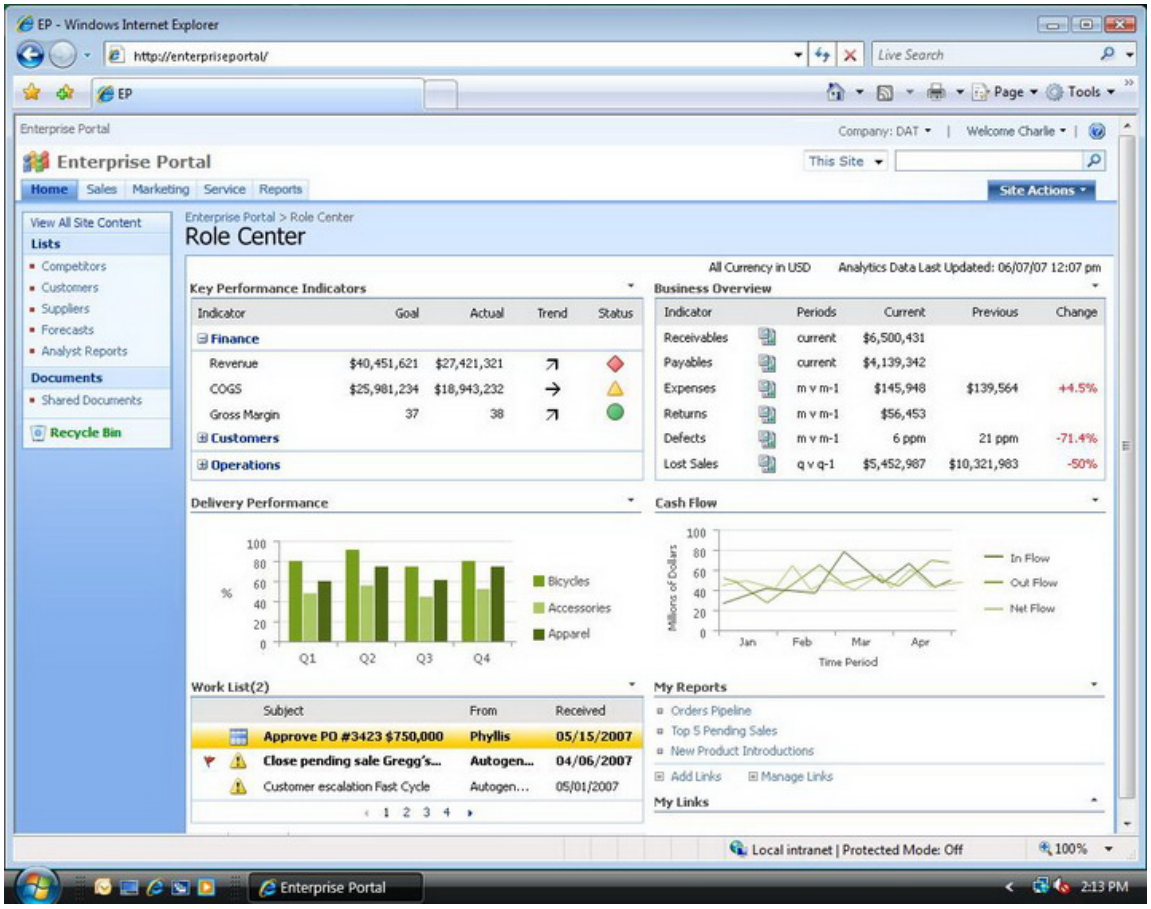


# ERP

- **Enterprise Resource Planning**
  - **ERP Propriétaire (Microsoft DYNAMICS)**

# ERP

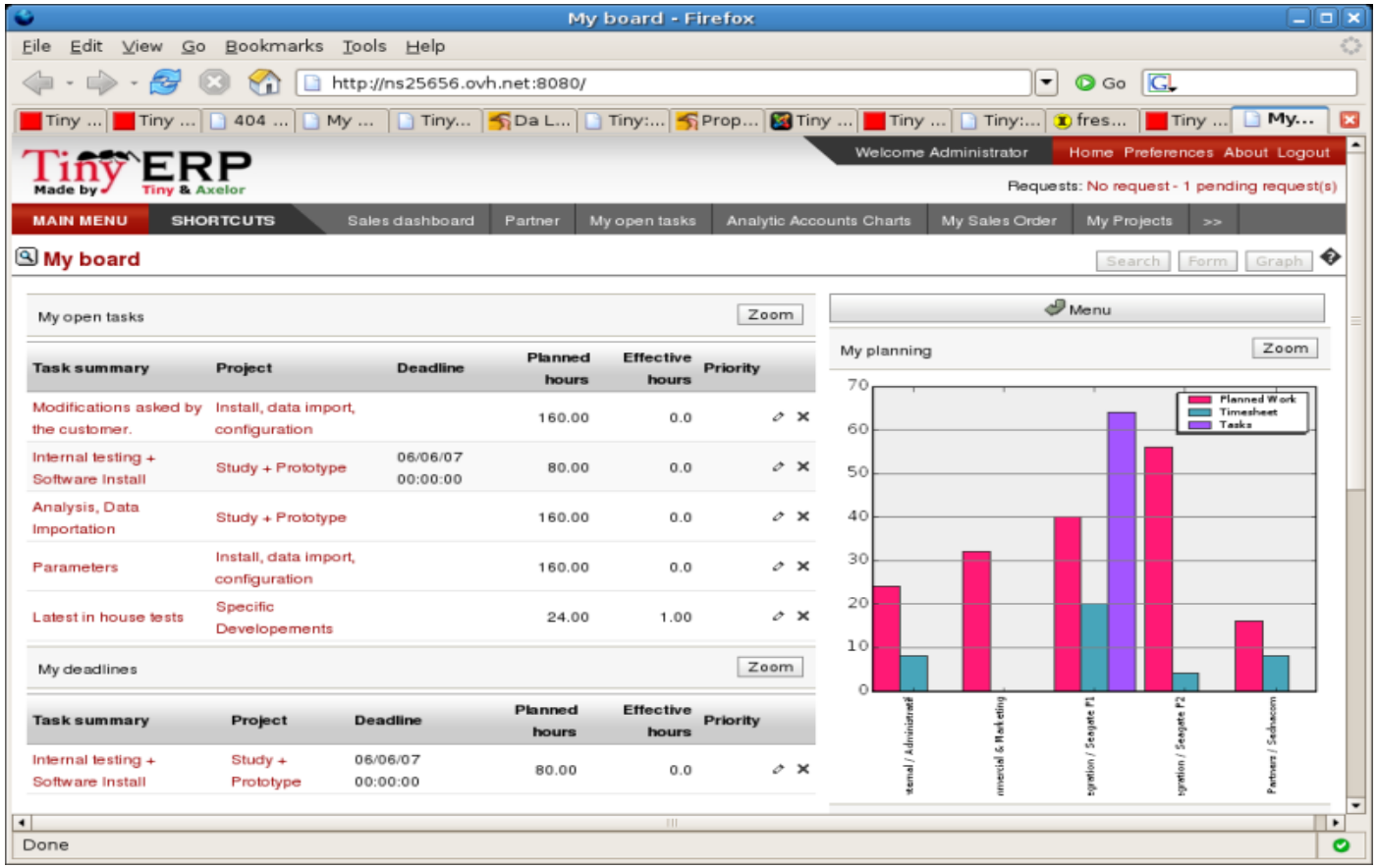
- **Enterprise Resource Planning**
  - **ERP Propriétaire (Microsoft DYNAMICS)**



# ERP

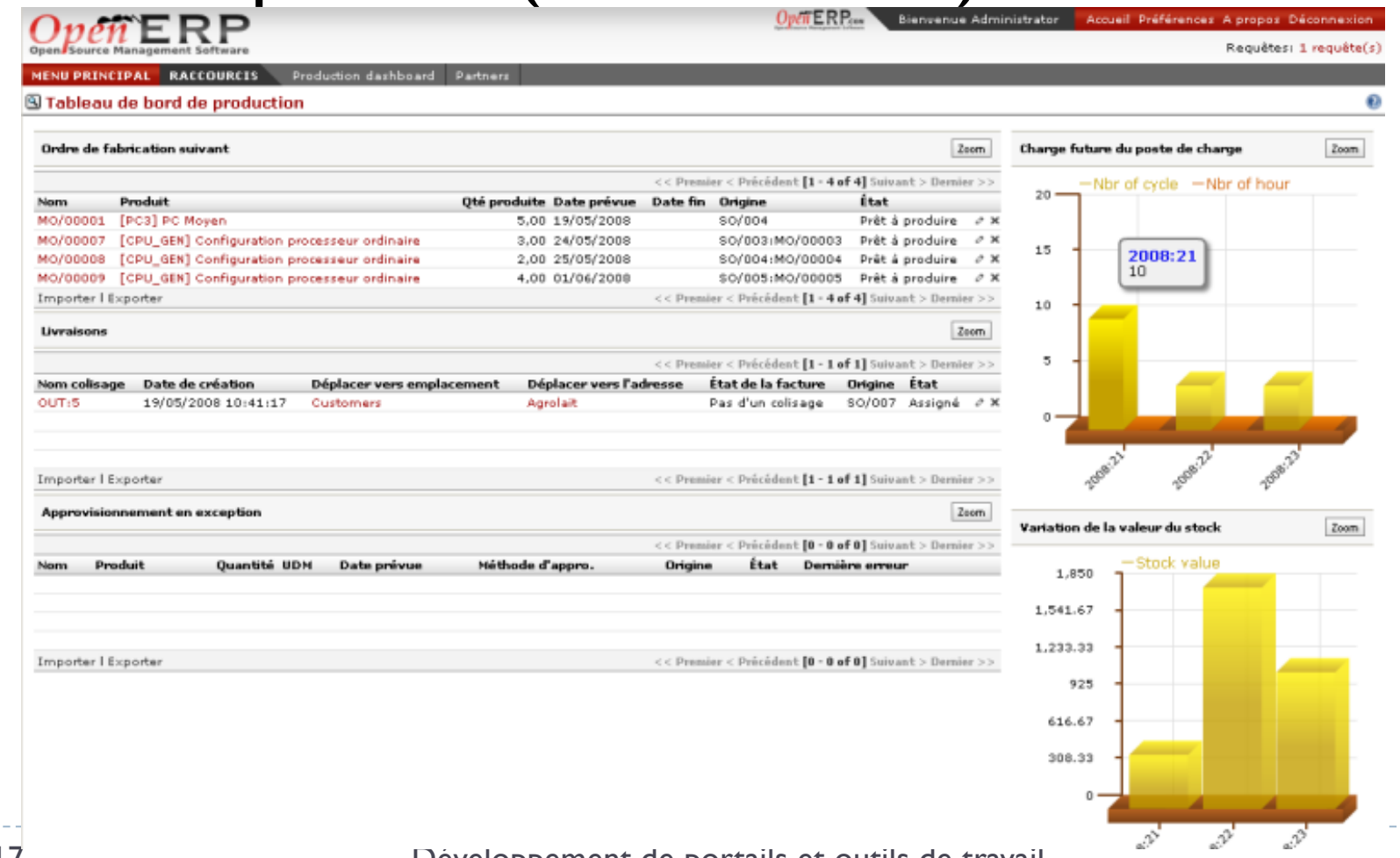
## Enterprise Resource Planning

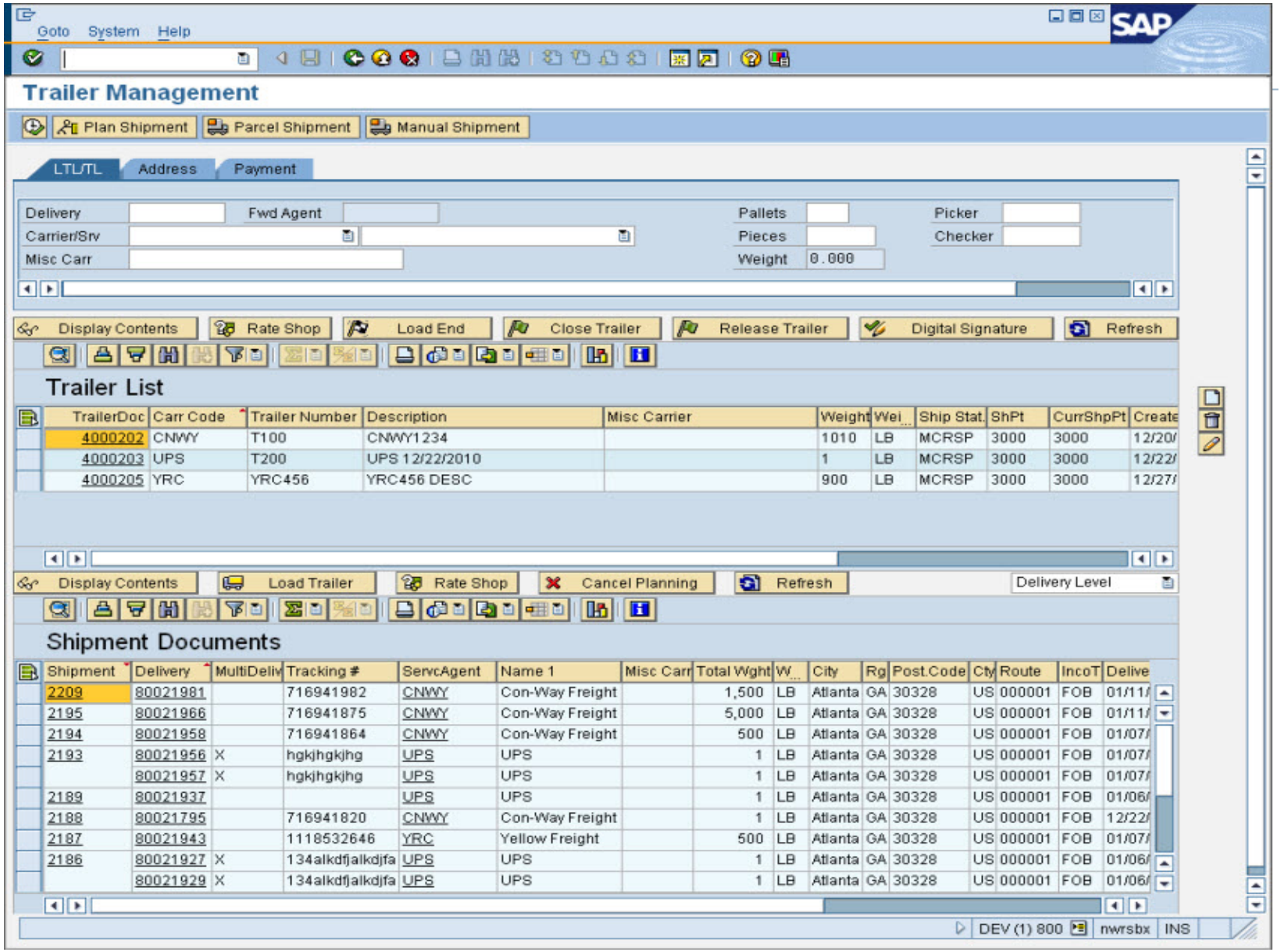
### ERP Open Source (Tiny ERP)



# ERP

- **Enterprise Resource Planning**
  - **ERP Open Source (SMILE OPEN ERP)**




Goto System Help
SAP
Trailer Management
Plan Shipment
Parcel Shipment
Manual Shipment
LTL/TL
Address
Payment
Delivery
Fwd Agent
Carrier/Srv
Misc Carr
Pallets
Pieces
Weight
0.000
Picker
Checker
Display Contents
Rate Shop
Load End
Close Trailer
Release Trailer
Digital Signature
Refresh
Trailer List
TrailerDoc
Carr Code
Trailer Number
Description
Misc Carrier
Weight
Wei...
Ship Stat.
ShPt
CurrShpPt
Create
4000202 CNWY T100 CNWY1234 1010 LB MCRSP 3000 3000 12/20/
4000203 UPS T200 UPS 12/22/2010 1 LB MCRSP 3000 3000 12/22/
4000205 YRC YRC456 YRC456 DESC 900 LB MCRSP 3000 3000 12/27/
Display Contents
Load Trailer
Rate Shop
Cancel Planning
Refresh
Delivery Level
Shipment Documents
Shipment
Delivery
MultiDeliv
Tracking #
ServcAgent
Name 1
Misc Carr
Total Wght
W...
City
Rg
Post.Code
Cty
Route
IncoT
Delive
2209 80021981 716941982 CNWY Con-Way Freight 1,500 LB Atlanta GA 30328 US 000001 FOB 01/11/
2195 80021966 716941875 CNWY Con-Way Freight 5,000 LB Atlanta GA 30328 US 000001 FOB 01/11/
2194 80021958 716941864 CNWY Con-Way Freight 500 LB Atlanta GA 30328 US 000001 FOB 01/07/
2193 80021956 X hgkjhgkjhg UPS UPS 1 LB Atlanta GA 30328 US 000001 FOB 01/07/
80021957 X hgkjhgkjhg UPS UPS 1 LB Atlanta GA 30328 US 000001 FOB 01/07/
2189 80021937 UPS UPS 1 LB Atlanta GA 30328 US 000001 FOB 01/06/
2188 80021795 716941820 CNWY Con-Way Freight 1 LB Atlanta GA 30328 US 000001 FOB 12/22/
2187 80021943 1118532646 YRC Yellow Freight 500 LB Atlanta GA 30328 US 000001 FOB 01/07/
2186 80021927 X 134alkdfjalkdjfa UPS UPS 1 LB Atlanta GA 30328 US 000001 FOB 01/06/
80021929 X 134alkdfjalkdjfa UPS UPS 1 LB Atlanta GA 30328 US 000001 FOB 01/06/
DEV (1) 800
nwrsbx
INS

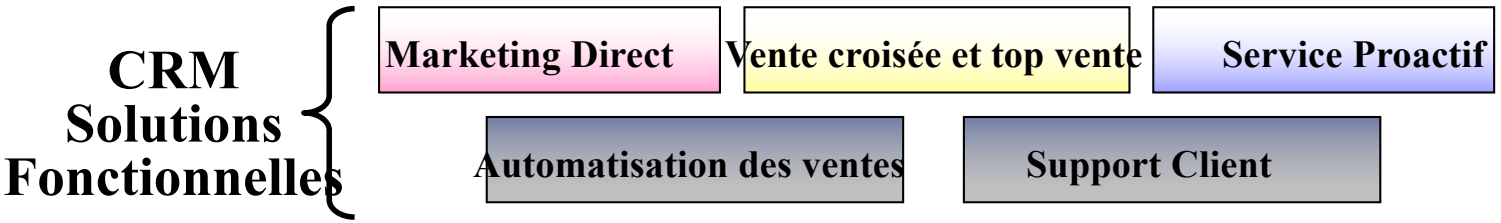
# CRM

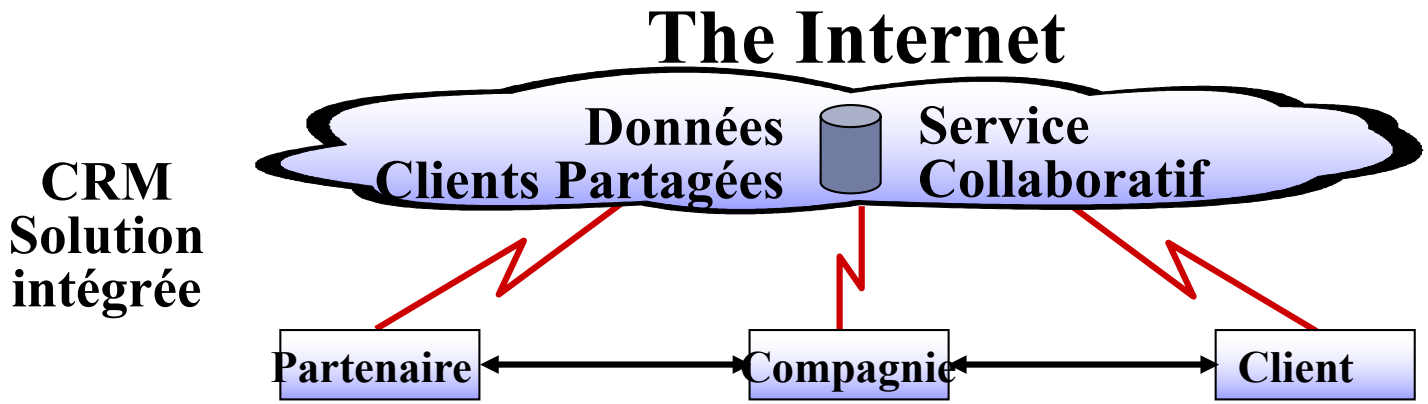
- **Customer Relationship Manager**
  - **Permet de se concentrer sur le commerce (Business Focus)**
  - **Permet une relation collaborative et intégrée entre une entreprise (un commerce) et ses différents clients**

# CRM

- **Customer Relationship Manager**

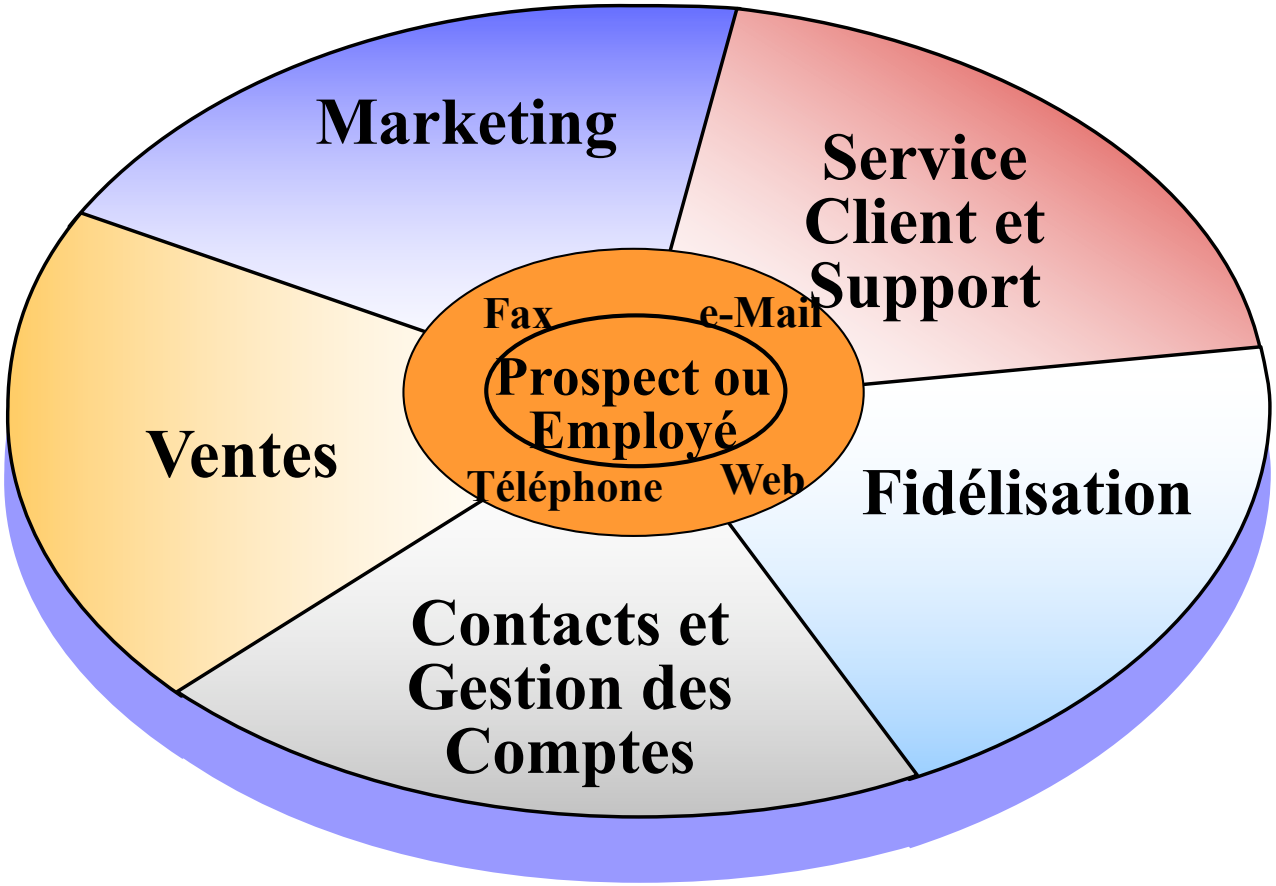
**Cycle de Vie D'un client**



# CRM

- **Customer Relationship Manager**



# CRM

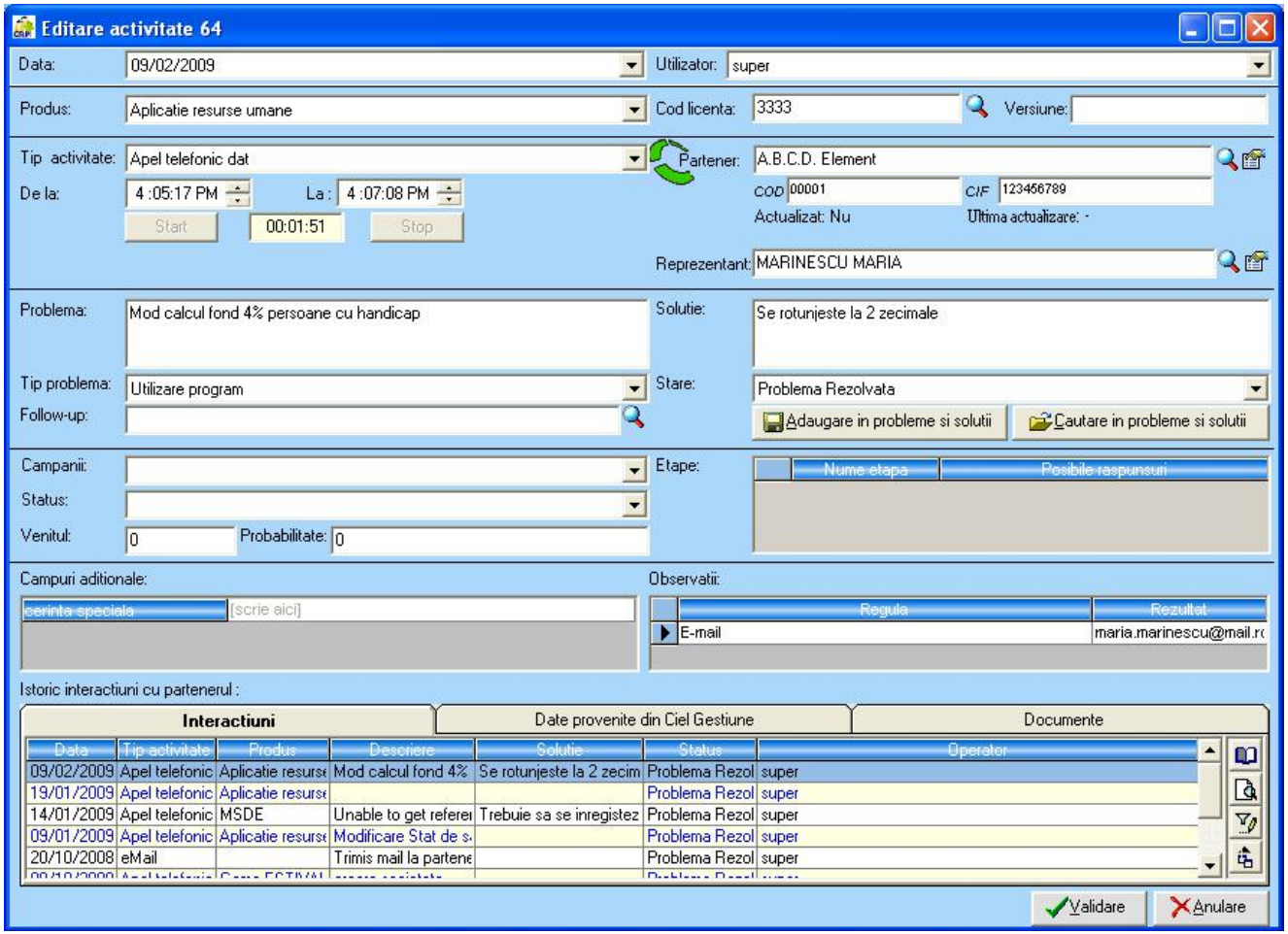
- **Customer Relationship Manager**
  - **Avantages**
    - **Identification et ciblage des meilleurs clients**
    - **Personnalisation et paramétrage des différents produits et services**
    - **Suivi du contact client**
  - **Inconvénients**
    - **50% des applications n'arrivent pas à répondre aux besoins**
    - **20% du temps le CRM endommage la relation client**
    - **Le manque de compréhension / préparation est souvent pointé du doigt**

# CRM

- **Customer Relationship Manager**
  - **Plusieurs familles**
    - **Propriétaire**
    - **Open Source Commercial**
    - **Open Source**
    - **Quelques CRM ont récemment ajouté le qualificatif « Social » pour l'intégration des réseaux sociaux (LinkedIn, Viadeo…)**

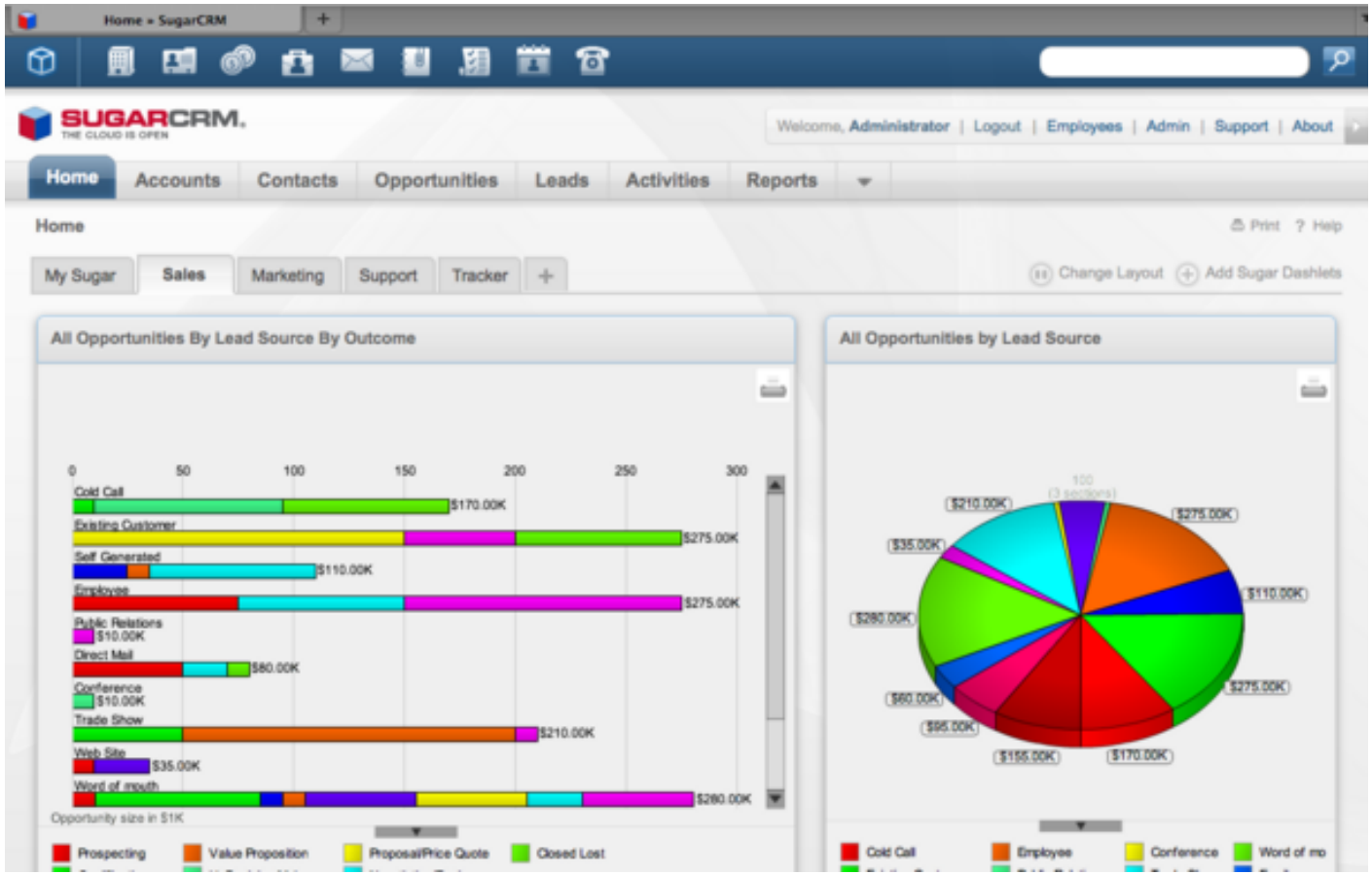
# CRM

- **Propriétaire : Ciel CRM**



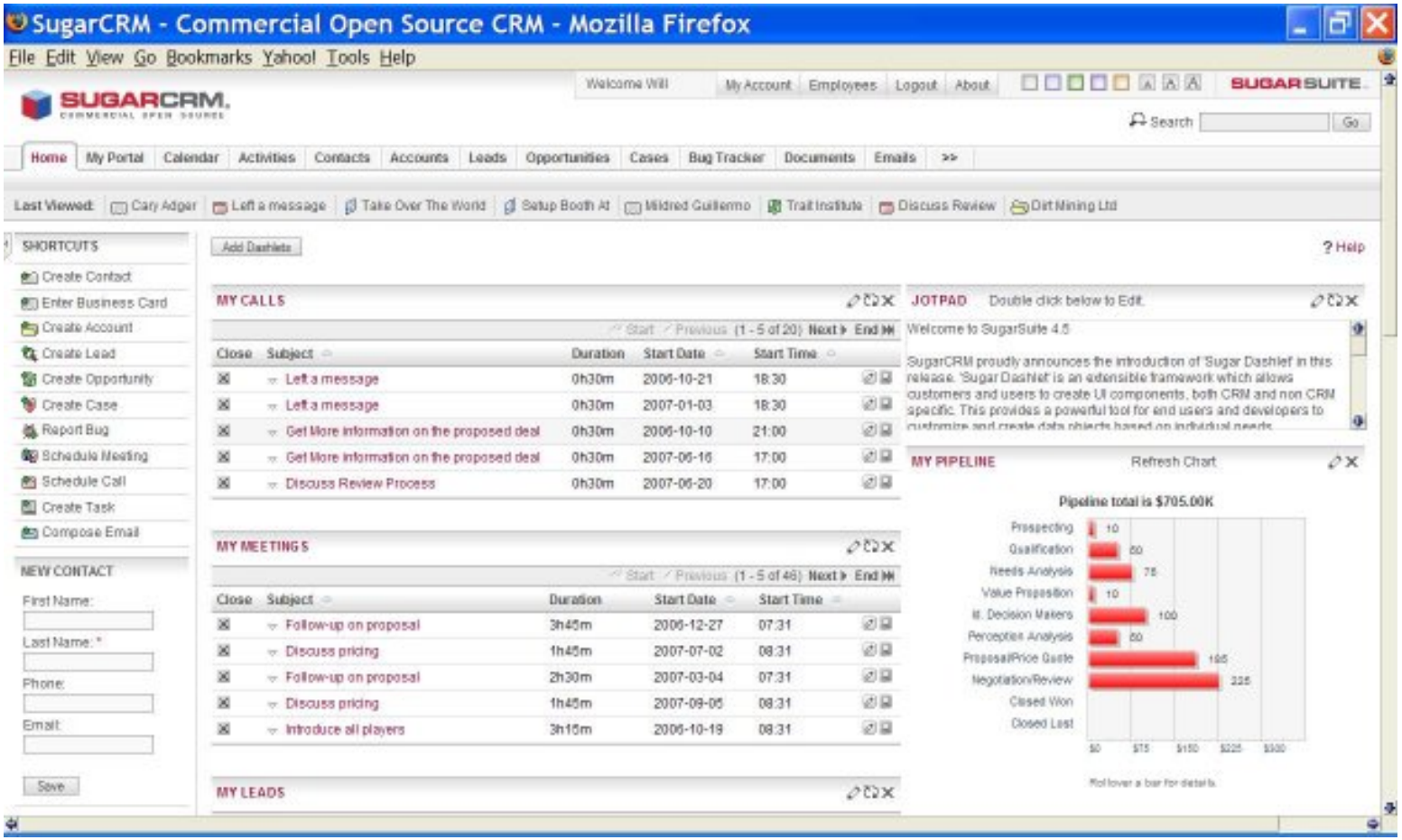
# CRM

- **Open Source Commercial : SugarCRM**



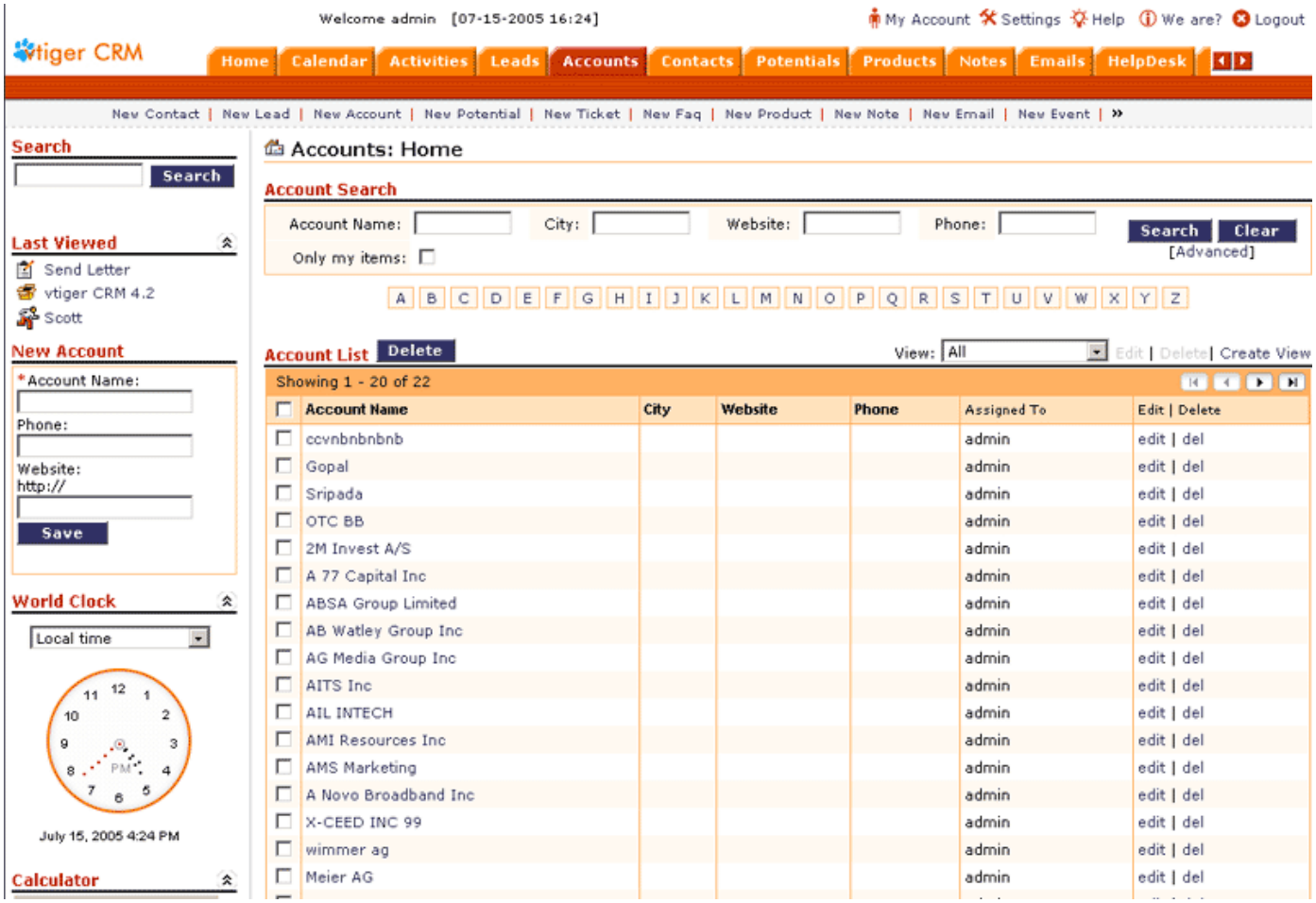
# CRM

- **Open Source Commercial : SugarCRM**



# CRM

- **Open Source Commercial : vTiger**



# CRM

- **Open Source : Dolibarr**