Informatique pour le TAL 2 (SL02244X)
Manipuler des données XML en Perl

Franck Sajous/CLLE-ERSS

29 mars 2012

http://w3.erss.univ-tlse2.fr/membre/fsajous/

Produire vs. Lire du XML

Deux processus différents

- Lire (et utiliser) des données au format XML (à partir d'un fichier) ;
- Produire des données au format XML (dans un fichier, sur la sortie standard).

Produire vs. Lire du XML

Deux processus différents

- Lire (et utiliser) des données au format XML (à partir d'un fichier) ;
- Produire des données au format XML (dans un fichier, sur la sortie standard).

Produire : facile

```
print "<document>\n";
print "\t<p id='1'>Ceci est un paragraphe</p>\n";
print "\t<p id='2'>Ceci est un autre paragraphe</p>\n";
print "</document>\n";
```

$\longrightarrow$

```
<document>
    <p id='1'>Ceci est un paragraphe</p>
    <p id='2'>Ceci est un autre paragraphe</p>
</document>
```

Exploiter des données XML dans un programme

Exemples

- Dans Morphalou, lister toutes les formes fléchies et les lemmes correspondant des entrées homonymiques ?
- Dans le corpus AirFrance, afficher les tours de parole imbriqués.

Exploiter des données XML dans un programme

Exemples

- Dans Morphalou, lister toutes les formes fléchies et les lemmes correspondant des entrées homonymiques ?
- Dans le corpus AirFrance, afficher les tours de parole imbriqués.

Comment ?

Une première idée

- Ouvrir le fichier, le lire ligne à ligne
- Détecter les balises XML avec des expressions régulières

Exploiter des données XML dans un programme

Exemples

- Dans Morphalou, lister toutes les formes fléchies et les lemmes correspondant des entrées homonymiques ?
- Dans le corpus AirFrance, afficher les tours de parole imbriqués.

Comment ?

Une première idée

- Ouvrir le fichier, le lire ligne à ligne
- Détecter les balises XML avec des expressions régulières

Essayez... Bon courage !

Parseurs prêts à l'emploi

Utiliser l'API : plusieurs familles de parseurs

- Les parseurs de type DOM (Document Object Model)
- Les parseurs *« à la SAX »*

DOM

- DOM = représentation standardisée des documents XML/XHTML
- document = arbre où chaque balise et chaque élément de texte sont considérés comme des nœuds
- pour un nœud (élément) donné, possibilité d'accèder aux attributs de l'élément, à d'autres nœuds en exprimant des relations de parenté (e.g. filiation dans l'arborescence), etc.

DOM (1)

Accèder à un/des noeuds

```
my $parser = new XML::DOM::Parser;
my $doc = $parseur->parsefile(‘‘monfichier.xml’’);
my @listeEntrees = @{$doc->getElementByTagName(‘‘lexicalEntry’’)};
foreach my $noeud (@listeEntrees)
{
  ...
}
```

DOM (1)

Accèder à un/des noeuds

```
my $parser = new XML::DOM::Parser;
my $doc = $parseur->parsefile(‘‘monfichier.xml’’);
my @listeEntrees = @{$doc->getElementByTagName(‘‘lexicalEntry’’)};
foreach my $noeud (@listeEntrees)
{
  ...
}
```

Depuis un noeud

Accès aux autres nœuds :

- `$noeud->getAttribute("id")`
- `$noeud->getParentNode()`
- `$noeud->getFirstChild()`
- `$noeud->getChildNodes()`

DOM (1)

Accèder à un/des noeuds

```
my $parser = new XML::DOM::Parser;
my $doc = $parseur->parsefile(‘‘monfichier.xml’’);
my @listeEntrees = @{$doc->getElementByTagName(‘‘lexicalEntry’’)};
foreach my $noeud (@listeEntrees)
{
  ...
}
```

Depuis un noeud

Accès aux autres nœuds :

- `$noeud->getAttribute("id")`
- `$noeud->getParentNode()`
- `$noeud->getFirstChild()`
- `$noeud->getChildNodes()`

Modification :

- `$noeud->appendChild($autreNoeud)`
- `$noeud->removeChild($fils)`
- `$noeud->setAttribute ("id", $nouvelId)`

DOM (2)

- Pratique, mais. . .
- Nécessite de stocker tout l'arbre en mémoire
- Impossible pour les documents volumineux

SAX (Simple API for XML)

Principes

- Traitement par flux
- Déclenchement d'« événements » à chaque :
  - ouverture de balise
  - fermeture de balise
  - élément de texte
- Le programmeur (utilisateur du module SAX) écrit des procédures qui « interceptent » ces événements
- Ces procédures sont appelées par le module ; elles peuvent « réagir » (ou pas) aux événements déclenchés

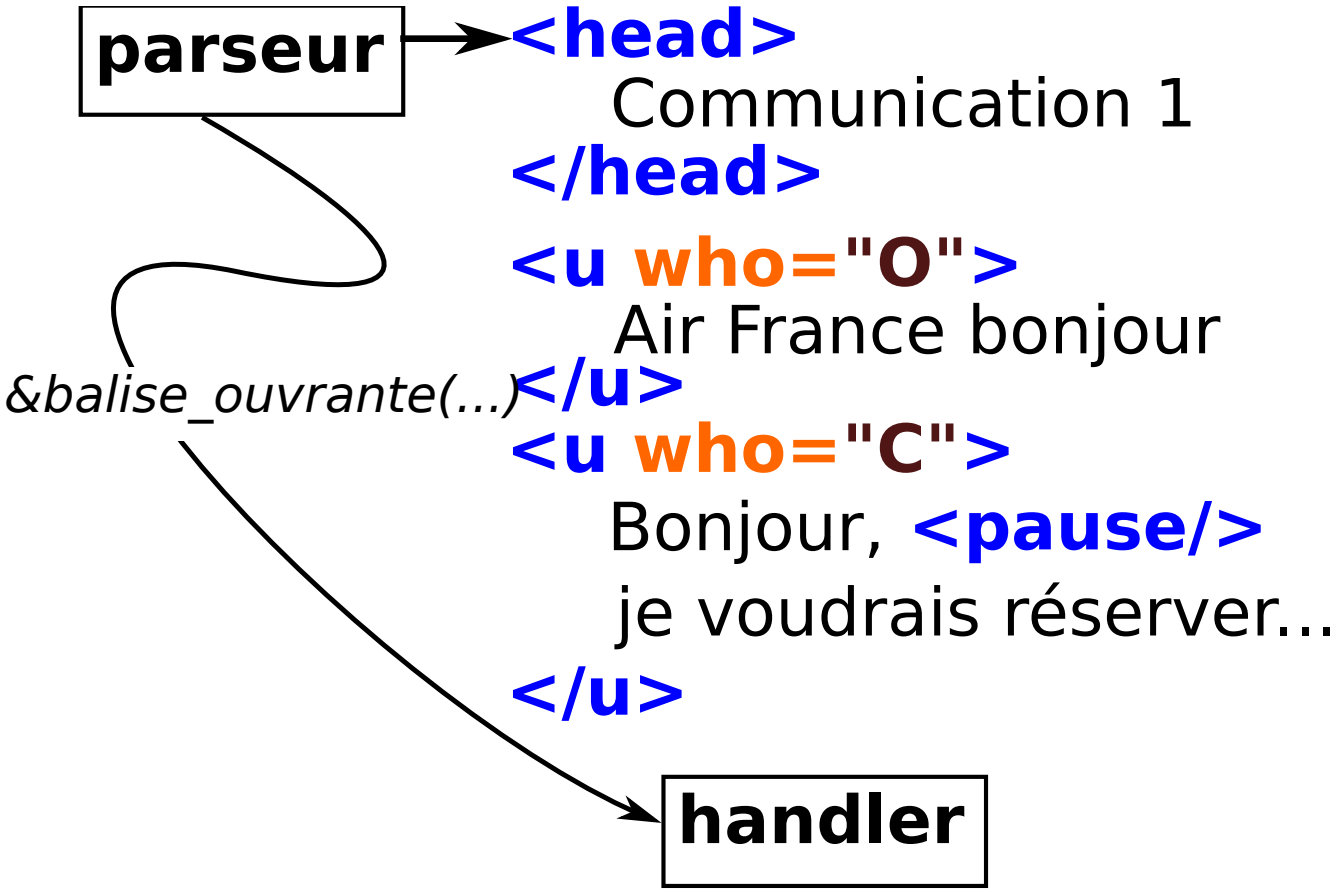
SAX : utilisation

Template

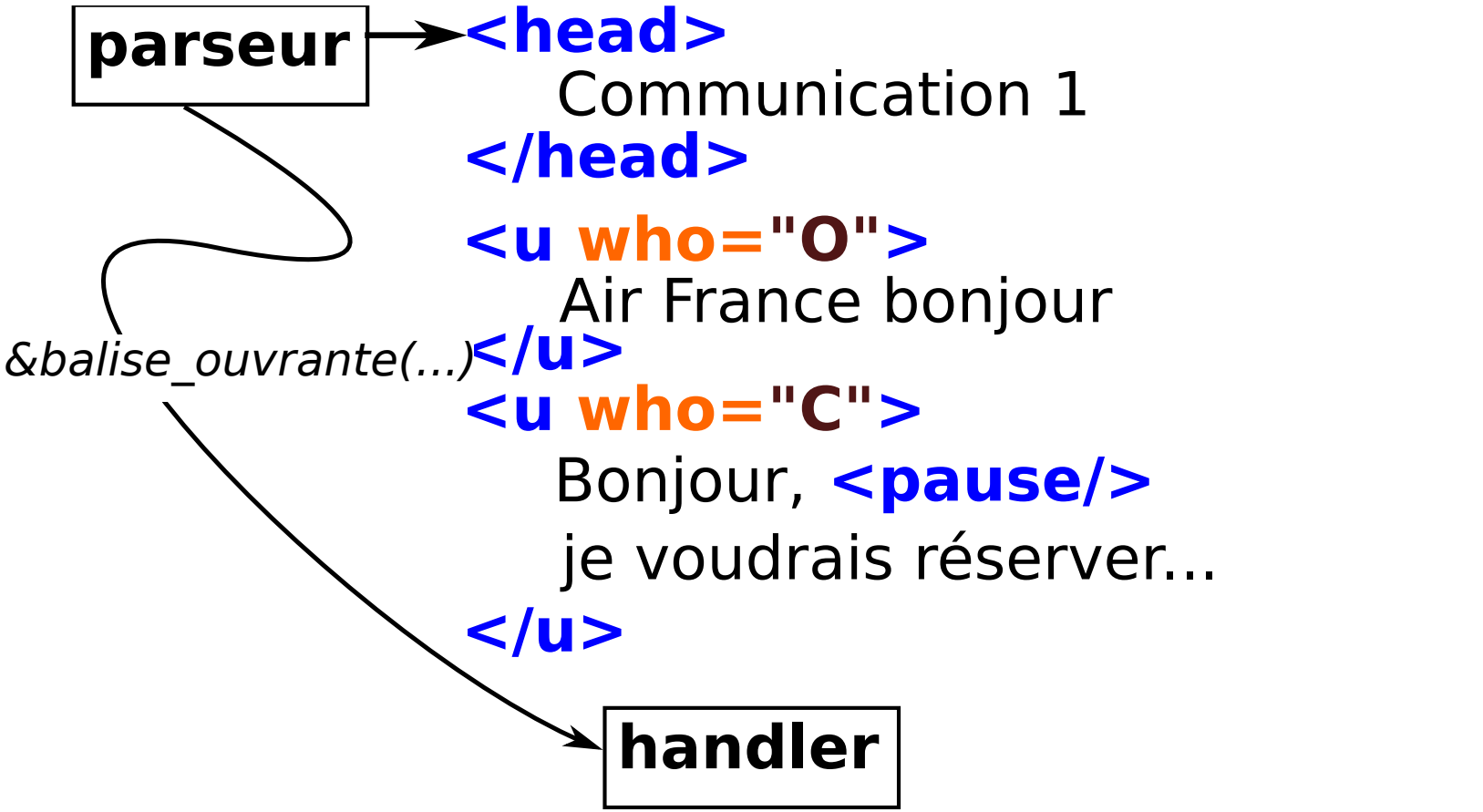
```
use XML::Parser;

my $parseur = new XML::Parser;
$parseur -> setHandlers (
    Start => \&balise_ouvrante,
    End => \&balise_fermante,
    Char => \&segment_texte
    );

$parseur->parsefile("monfichier.xml");

sub balise_ouvrante {
    # traitement balise ouvrante
}

sub balise_fermante {
    # traitement balise fermante
}

sub segment_texte {
    # traitement segment texte
}
```

SAX : événements

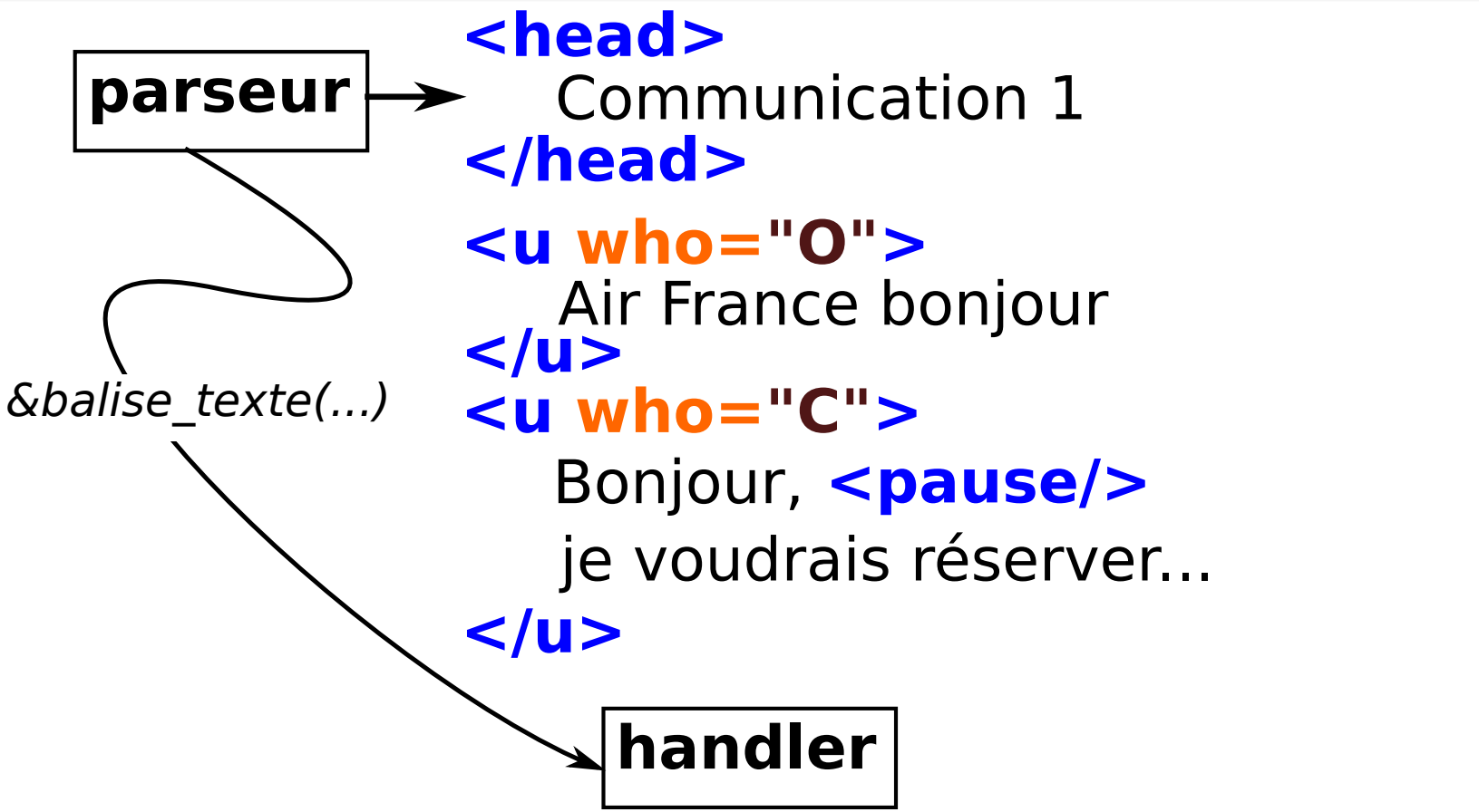
SAX : événements



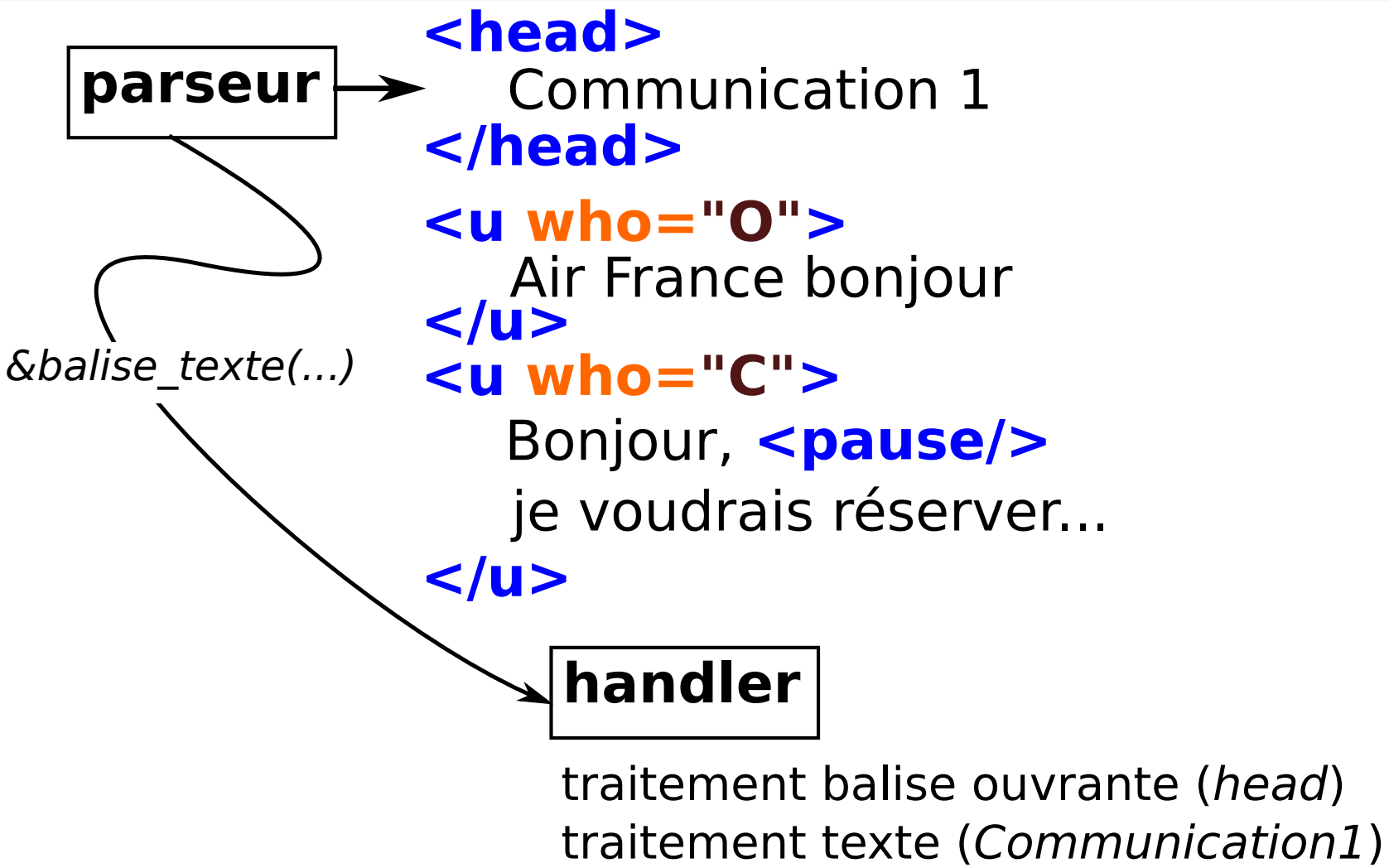
SAX : événements



SAX : événements



SAX : événements



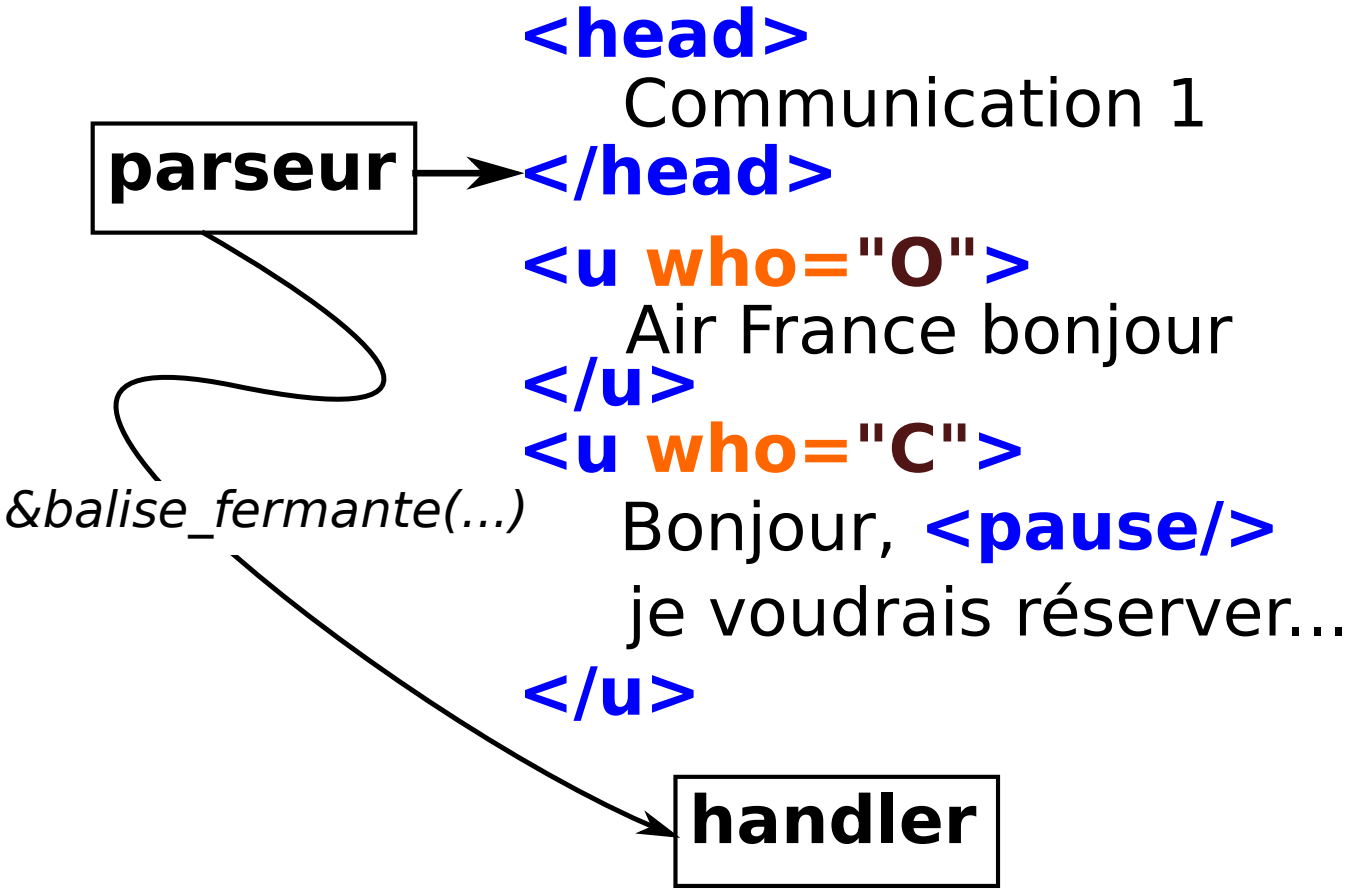
SAX : événements


```
<head>
    Communication 1
</head>
<u who="O">
    Air France bonjour
</u>
<u who="C">
    Bonjour, <pause/>
    je voudrais réserver...
</u>
```

handler

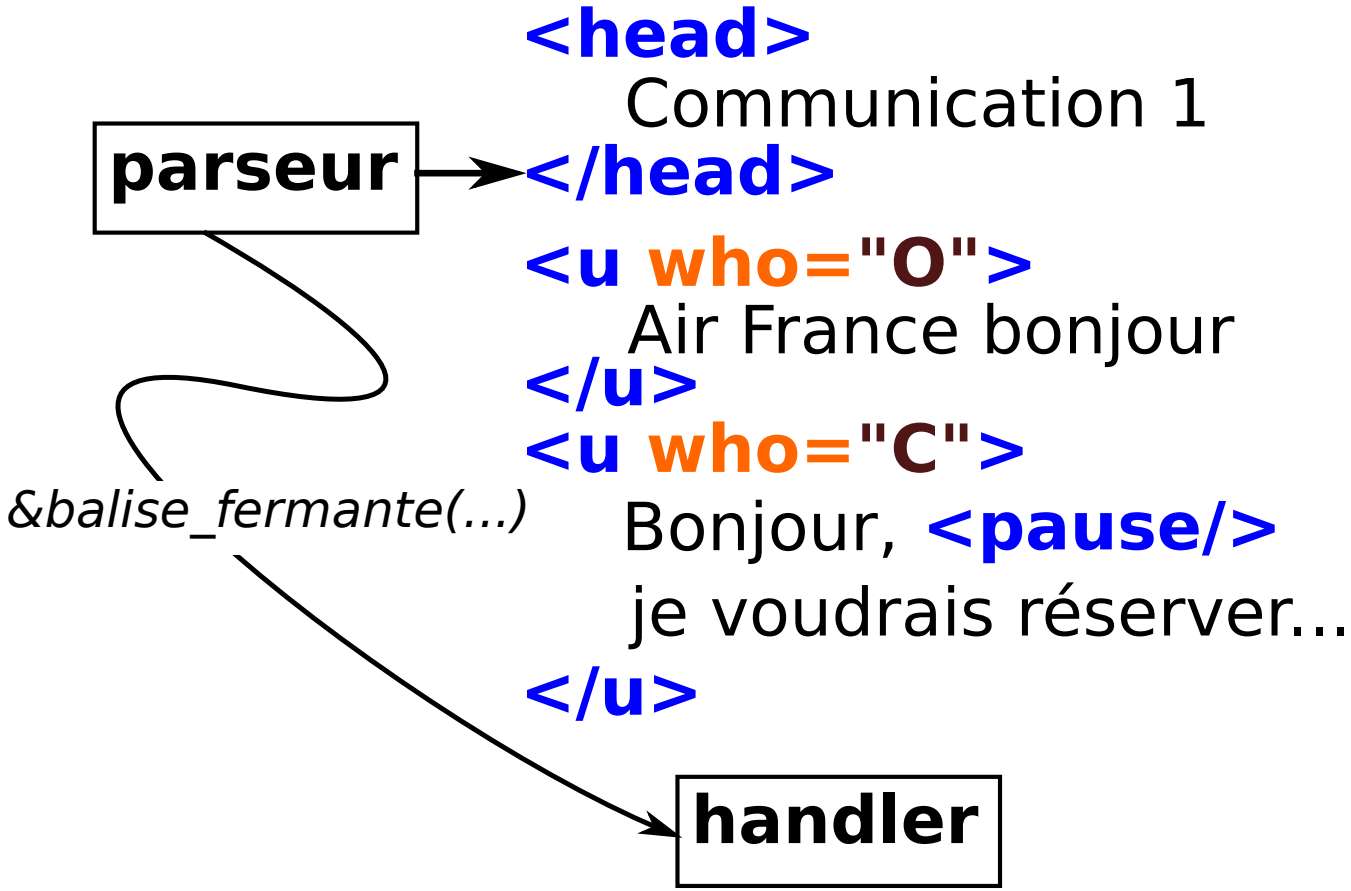
traitement balise ouvrante (*head*)
traitement texte (*Communication1*)

SAX : événements



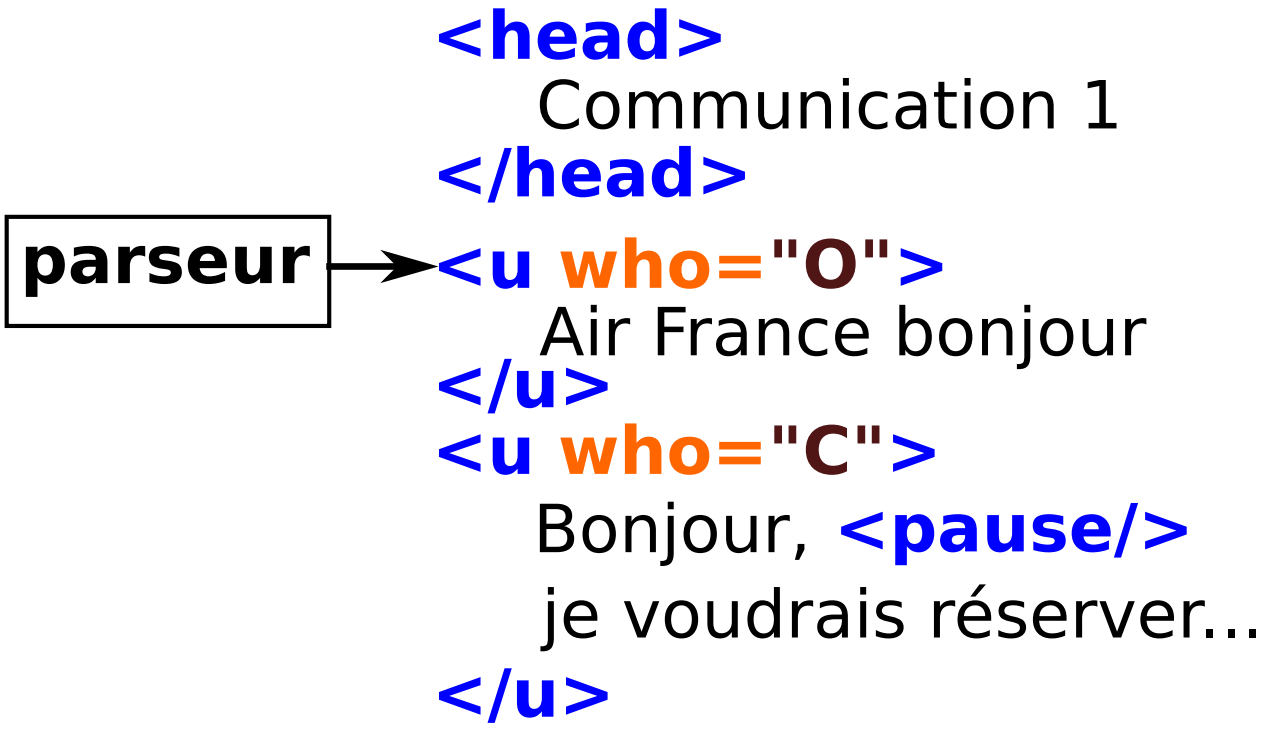
traitement balise ouvrante (*head*)
traitement texte (*Communication1*)

SAX : événements



traitement balise ouvrante (*head*)
traitement texte (*Communication1*)
traitement balise fermante (*head*)

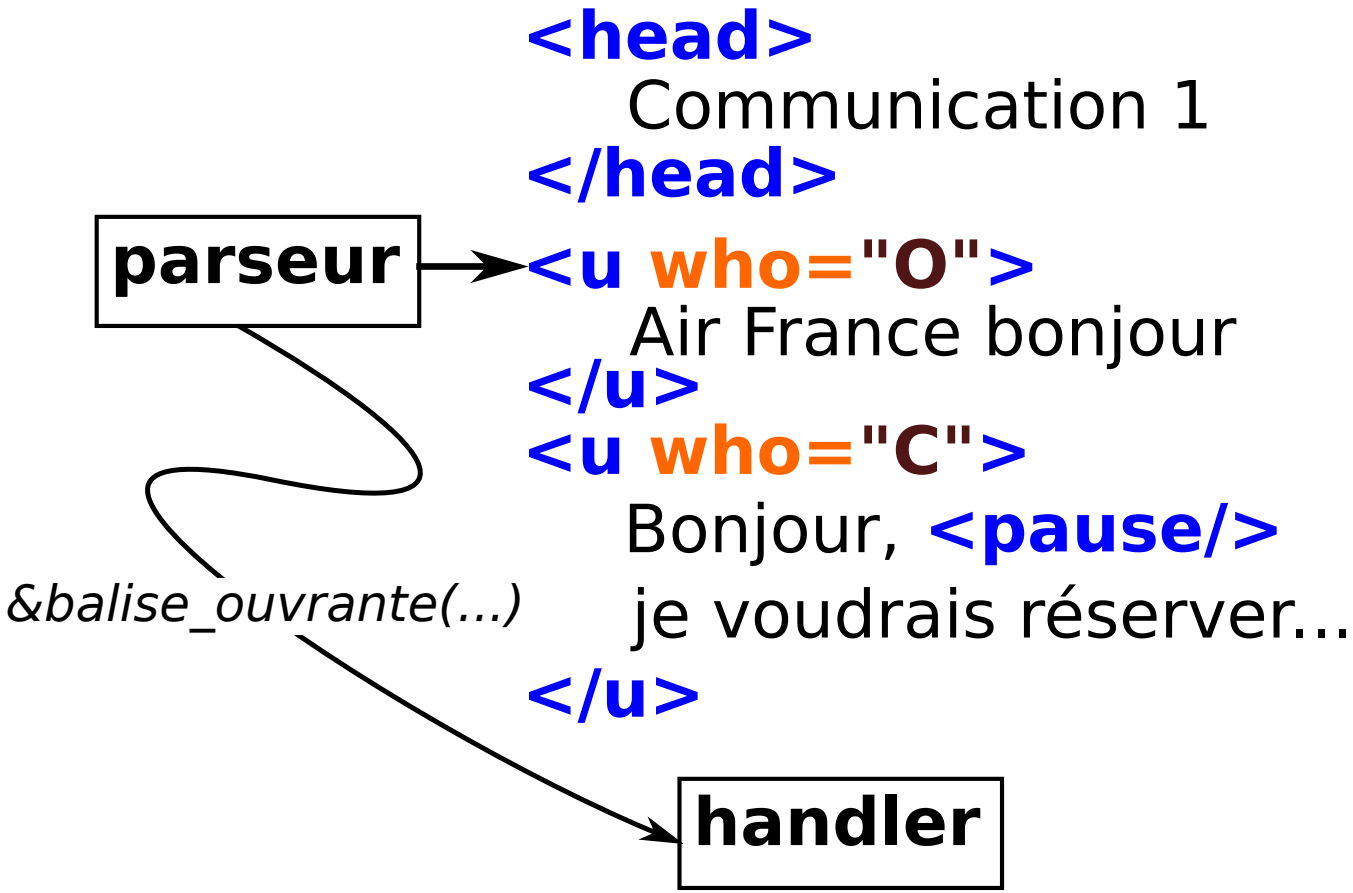
SAX : événements



handler

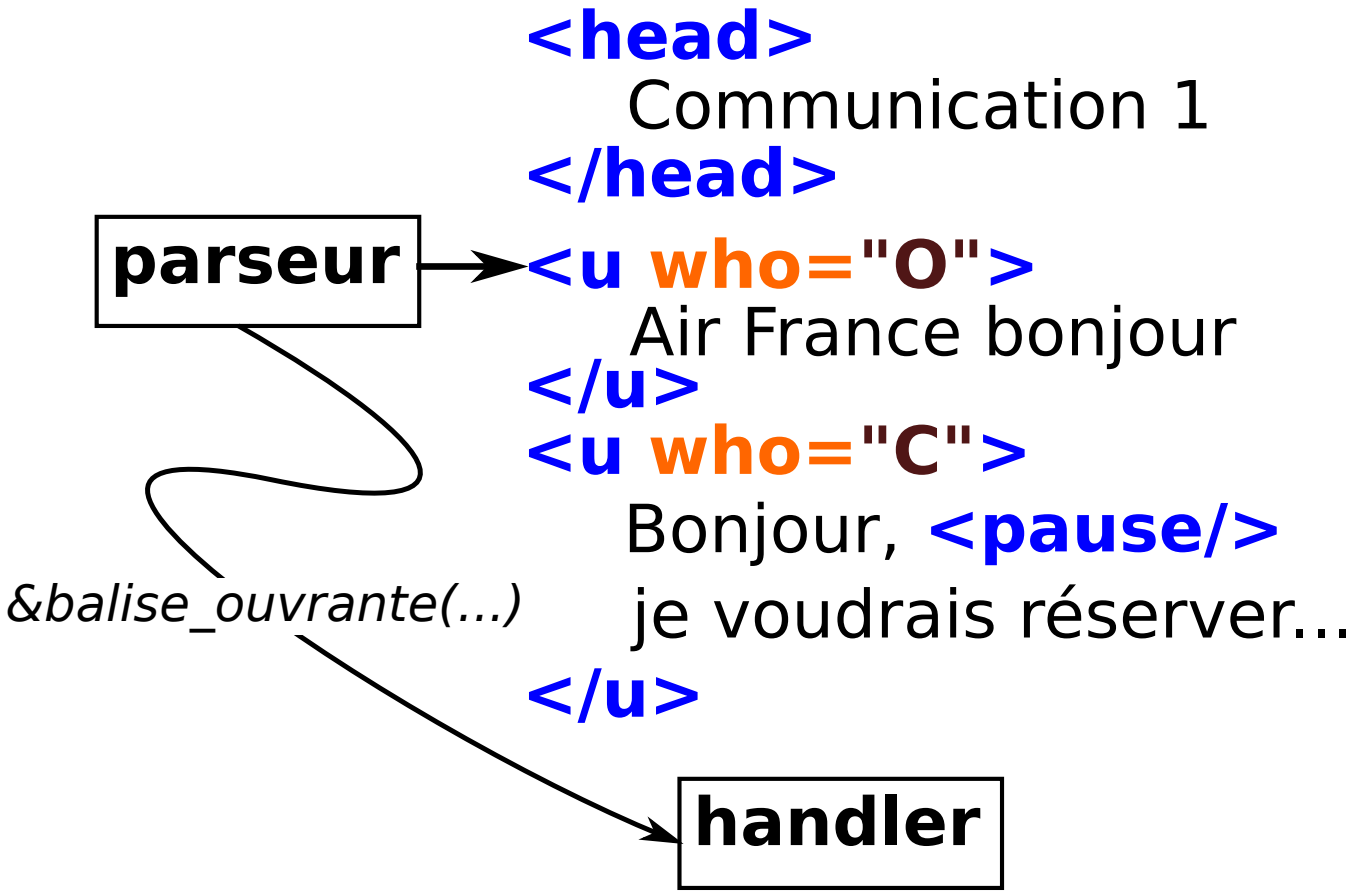
traitement balise ouvrante (*head*)
traitement texte (*Communication1*)
traitement balise fermante (*head*)

SAX : événements



traitement balise ouvrante (*head*)
traitement texte (*Communication1*)
traitement balise fermante (*head*)

SAX : événements



traitement balise ouvrante (*head*)
traitement texte (*Communication1*)
traitement balise fermante (*head*)
traitement balise ouvrante (*u*)

SAX : gérer les événements

Arguments des procédures appelées

- balise ouvrante :
  - Premier argument : une référence au parseur (on ne l'utilisera pas ici)
  - Deuxième argument : nom de la balise
  - Arguments suivants (2 à 2) : nom attribut, valeur attribut
- balise fermante :
  - Premier argument : une référence au parseur
  - Deuxième argument : nom de la balise
- segment de texte :
  - Premier argument : une référence au parseur
  - Deuxième argument : chaîne de caractère
  - Attention ! Un même élément de texte peut être découpé en plusieurs segments par le parseur et provoquer autant d'événements $\rightarrow$ nécessité de gérer une variable « *buffer* » pour stocker l'élément textuel en cours