

# Modifier le DOM avec JavaScript

Alexandre Niveau

GREYC — Université de Caen

En partie adapté des cours de Jean-Marc Lecarpentier et Hervé Le Crosnier

## Rappels

- Pour les navigateurs, une page web est un arbre, dont les éléments sont des nœuds
- Le travail du « moteur de rendu » du navigateur est de construire une représentation graphique de cet arbre
- Le HTML n'est qu'un *langage de sérialisation* de cet arbre

## Nécessité du DOM

- Les programmes en JavaScript sont exécutés par le navigateur
- Leur but : modifier la page web en fonction des actions de l'internaute
  - modifier la page = modifier les nœuds de l'arbre (changer leurs caractéristiques, les déplacer, en ajouter, en supprimer...)
- Il faut donc avoir un *modèle* de la page et de sa structure, ainsi que des fonctions permettant de manipuler ce modèle : une API (*application programming interface*)
- Le modèle des pages HTML (et XML) s'appelle le DOM, *document object model*

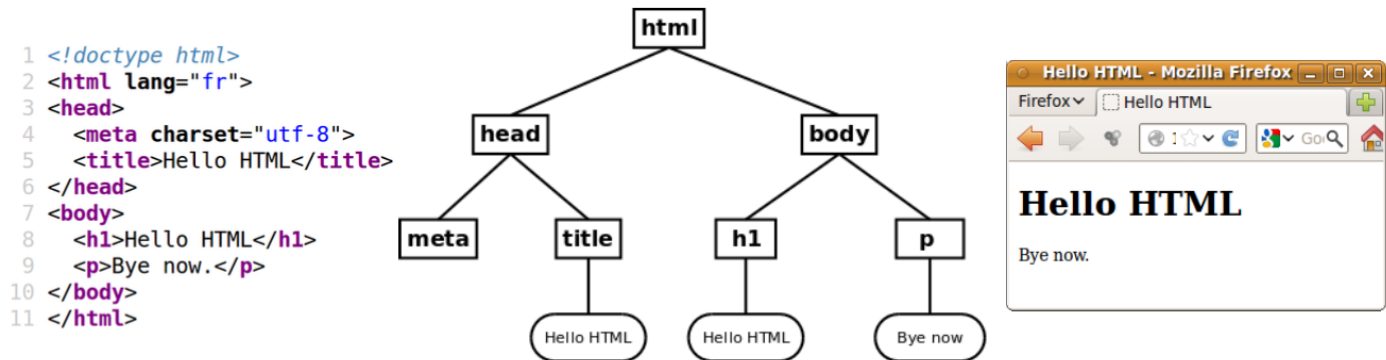
## Historique du DOM

- 1996 : sortie de JavaScript avec Netscape 2.0 puis de JScript avec IE 3.0
  - Les deux versions ont un DOM majoritairement compatible, JScript étant un portage de JavaScript
  - On l'appelle souvent DOM niveau 0
- 1997 : Netscape et IE versions 4.0, développés en parallèle, introduisent des modifications incompatibles dans leurs DOM
  - Il fallait donc plusieurs versions de chaque programme pour une même page web...
- 1998 : Standardisation par le W3C du DOM niveau 1
- DOM 2 en 2000, DOM 3 en 2004, DOM 4 en 2014

- Il y a également le « DOM Living Standard » édité par le WhatWG, qui unifie les anciennes normes et les implémentations existantes dans les navigateurs

## HTML ⇒ DOM ⇒ Vue

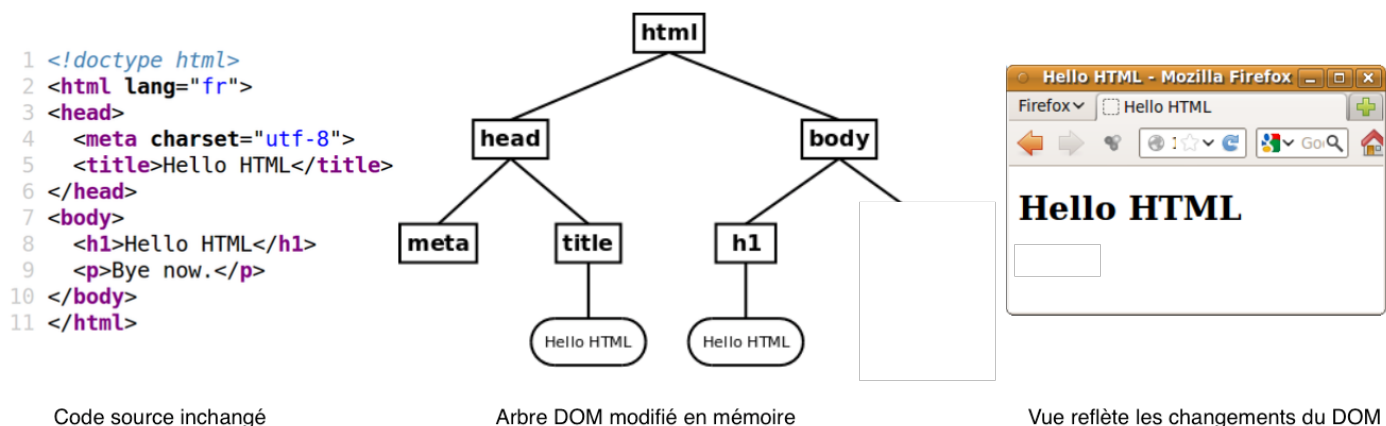
- Navigateur = parseur HTML + moteur graphique
  - Parseur HTML : construit l'arbre DOM en mémoire
  - Moteur graphique : construit une représentation de l'arbre DOM, suivant les règles données dans les CSS



Passage du HTML au DOM puis à la représentation graphique

## Modification du DOM

- JavaScript : implémente l'API DOM ⇒ possibilité de transformer l'arbre
- Toute modification de l'arbre DOM est immédiatement répercutée dans la représentation graphique
- Attention : l'arbre DOM est modifié dans la *mémoire* du navigateur, mais « afficher le code source » montre toujours le code de départ !
- Pour voir le code HTML correspondant à l'état réel du DOM à tout instant, il faut des outils particuliers (comme l'inspecteur de Firefox ou Chrome)

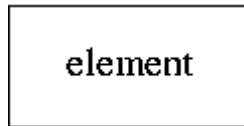


Modification du DOM reflétée dans la représentation graphique

## Représentation graphique

- Règles pour une représentation graphique de l'arbre DOM

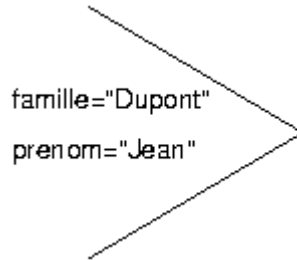
- Elements : rectangle



- Noeuds de texte : ellipse



- Attributs : triangle

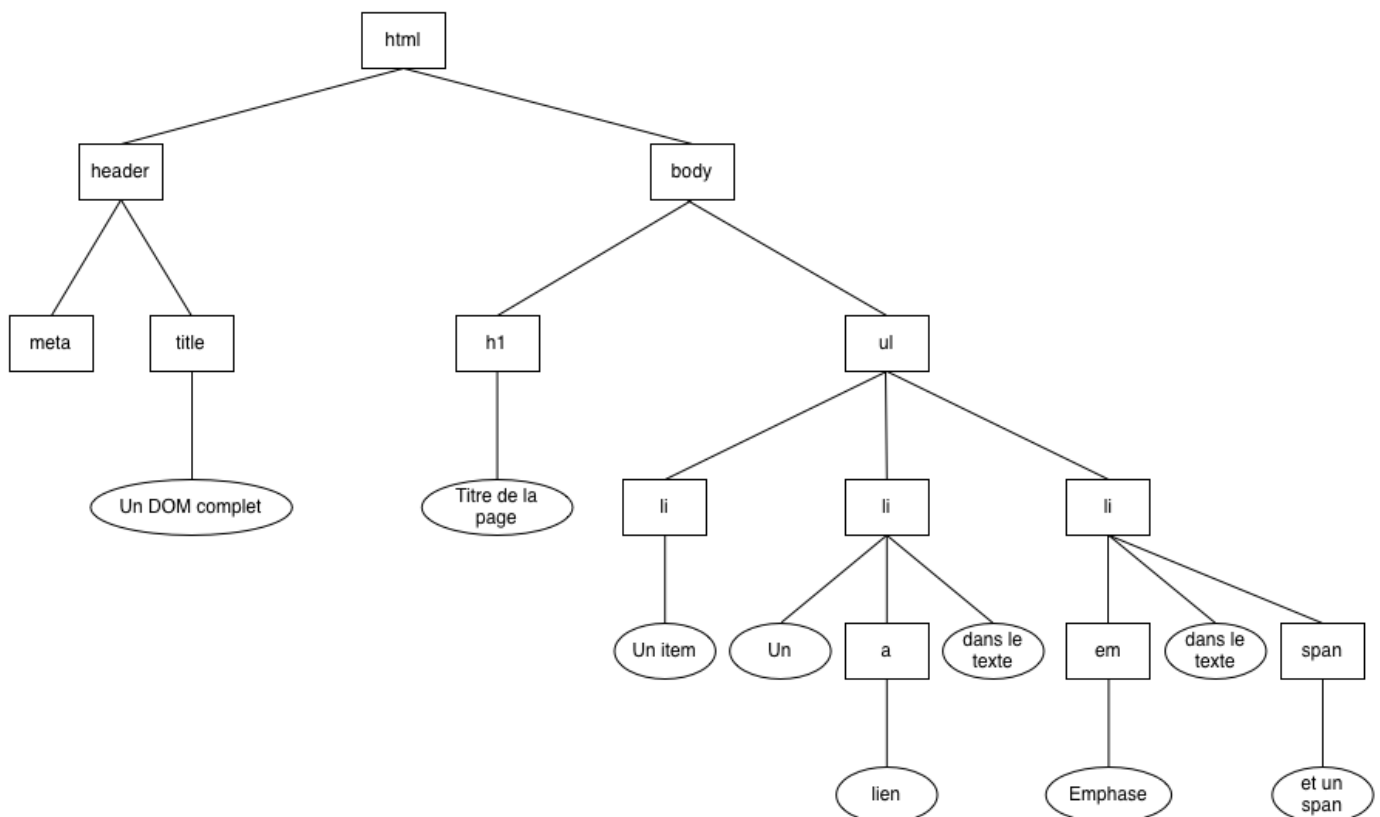


## Représentation de XML en DOM

```
<personne>
  <nom famille="Dupont" prenom="Jean" />
  <adresse>
    <rue numero="46" >Saint-Pierre</rue>
    <ville codePostal="14000">Caen</ville>
  </adresse>
  <telephone type="domicile">0231563459</telephone>
</personne>
```

Représentation graphique [img/personne.png]

## Types de nœuds



[Voir le code HTML de cet arbre](#) [demo/dom-complet.html]

- Noeud de type élément : représente un élément HTML
- Possibilité de changer les nœuds éléments, par ex. changer ses attributs
- Noeud de type texte : représente du texte
- Possibilité de changer le texte seulement
- Noeud de texte est forcément une feuille de l'arbre
- Noeud de type élément a des fils de type élément et/ou texte
- Noeud de type élément est une feuille de l'arbre si c'est un élément vide

## L'API du DOM

- [Recommandation du W3C](http://www.w3.org/TR/dom) [http://www.w3.org/TR/dom]
- Spécifie les fonctionnalités qui doivent être présentes
- Distincte des implémentations en divers langages (JavaScript, PHP, Java, C++, etc.)
- DOM définit les objets manipulables dans un document HTML, ou XML
- Les **objets** du DOM ont des
  - *propriétés* : contient les caractéristiques de l'objet
  - *méthodes* : fonctions qui permettent de manipuler l'objet (noms explicites en *camelCase*)

## Les principaux objets du DOM

- Voir [la référence sur MDN](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model) [https://developer.mozilla.org/en-US/docs/Web/API/Document\_Object\_Model]
- Document : le document (élément racine) duquel on a construit le DOM
- Node : les nœuds, qui peuvent être de différents types :
  - Element : nœuds éléments HTML (ou XML), contiennent d'autres nœuds (de type Element, Comment...)
  - CharacterData : nœuds de texte, contiennent du texte (objet Text)
- Event : les événements
- etc.
- Deux objets représentent des collections :
  - NodeList : liste de nœuds (par exemple la liste des fils d'un nœud)
  - HTMLCollection : liste d'éléments
- Une collection peut être statique ou « vivante » (*live*), c'est-à-dire que les changements du DOM y sont reflétés en permanence.

# Implémentation de l'API DOM avec JavaScript

- Les scripts en JavaScript permettent au navigateur d'agir sur l'arbre DOM du document en cours de visualisation
- Les objets du DOM sont implémentés par des *objets* en JavaScript
- En particulier, les éléments HTML sont des objets, les attributs HTML sont des propriétés de ces objets
- Les listes d'objets (NodeList et HTMLCollection) ne sont pas des tableaux, mais des objets particuliers qui ont aussi un attribut `length` et dont les éléments sont accessibles avec les crochets

## L'objet document

- Modélise le document manipulé
- L'élément racine (`html`) du document : `document.documentElement`
- L'élément `body` du document : `document.body`
- Obtenir un élément par son identifiant : `document.getElementById("toto")`
- ... et bien d'autres méthodes (pour accéder à des nœuds ou les modifier)

## Objets de type Node

- Modélise *tous les types* de nœuds, que ce soit des éléments ou non
- Propriétés des **objets Node** [<https://developer.mozilla.org/en-US/docs/Web/API/Node>] :
  - Liste des fils : `n.childNodes`
  - Nombre de fils : `n.childNodes.length`
  - Premier fils : `n.firstChild`
  - Dernier fils : `n.lastChild`
  - Nœud parent : `n.parentNode`
  - Frère suivant : `n.nextSibling`
  - Frère précédent : `n.previousSibling`
- **Attention** aux nœuds de texte vides !

# [#interface-Element]

## Objets de type Element

- Généralement, on préfère parcourir le DOM en ignorant les nœuds de texte (notamment à cause des nœuds de texte vides)
- Les propriétés de l'**interface Element** [<https://developer.mozilla.org/en-US/docs/Web/API/Element>] permettent de le faire simplement :

- Liste des *éléments* fils : `e.children`
- Nombre d'éléments fils : `e.childElementCount`
- Premier/dernier fils qui est un élément : `e.firstElementChild` et `e.lastElementChild`
- Frère suivant/précédent qui est un élément : `nextElementSibling` et `previousElementSibling`

## Attributs des éléments

- Pour accéder et modifier les attributs d'un élément HTML : méthodes `getAttribute("toto")` et `setAttribute("toto", "valeur")`, `hasAttribute("toto")`, `getAttributeNames()`... de l'interface `Element`
- La plupart des attributs HTML sont accessibles via des **propriétés** de l'objet JS représentant un élément :
  - `id` : identifiant d'un élément
  - `href` : attribut href, pour un lien
  - `src` : attribut src, pour une image
  - `style` : objet représentant le contenu de l'attribut style (voir plus loin)
  - `classList` : objet représentant le contenu de l'attribut class (voir plus loin)
  - etc.

Attention, la valeur de ces propriétés n'est pas forcément exactement identique à celle de l'attribut (par exemple, la propriété `src` d'un `HTMLImageElement` contient l'URL absolue vers l'image, telle que résolue par le navigateur)

## Accéder directement à un élément de l'arbre DOM

- `document.getElementById("toto")` : utilisation de l'identifiant `id` d'un élément

```
<body>
  <p class="intro">bla bla bla</p>
  <div id="toto">Bonjour le monde</div>

  <script>
    var maDiv = document.getElementById("toto");
    maDiv.setAttribute("style", "color: red;");
    // ou bien
    maDiv.style.color="red";
  </script>
</body>
```

Résultat [demo/getelem.html]

- `document.getElementsByTagName("h2")` : renvoie une liste « vivante » de tous les éléments `h2`
- `document.getElementsByClassName("erreur")` : renvoie une liste « vivante » de tous les éléments de classe `erreur`

## Utilisation de sélecteurs CSS

- Les techniques ci-dessus sont efficaces, mais ne fonctionnent que dans des cas particuliers
- Si on veut sélectionner des éléments de manière précise sans devoir rajouter des identifiants partout, la méthode la plus simple est d'utiliser un sélecteur CSS
- C'est possible avec deux fonctions standard et [qui fonctionnent partout](http://caniuse.com/#feat=queryselector) [http://caniuse.com/#feat=queryselector] (IE>9) :
  - `document.querySelector("#tutu div.erreur")` : renvoie le premier élément qui correspond au sélecteur CSS donné
  - `document.querySelectorAll("#tutu div.erreur")` : renvoie une liste statique de tous les éléments correspondant au sélecteur CSS donné
- Les sélecteurs autorisés sont ceux supportés par le navigateur, mais attention, les pseudo-éléments (`::before`, `::first-letter`... à ne pas confondre avec les pseudo-classes) sont inutiles, car ils ne correspondent à aucun élément de l'arbre et renvoient donc toujours `null`
- Attention, `querySelector` est moins efficace que les `getElementsBy`, en particulier pour [sélectionner un identifiant](http://jsperf.com/getelementbyid-vs-queryselector) [http://jsperf.com/getelementbyid-vs-queryselector]. En pratique, à moins que vous ne fassiez des modifications vraiment intensives du DOM, la différence est négligeable : *il est conseillé de préférer la simplicité d'implémentation que permet `querySelector`.*

## Modifier le style CSS

- On peut modifier les propriétés de style de chaque objet DOM :

```
document.getElementById("toto").style.color="green";  
document.getElementById("toto").style.backgroundColor="blue";  
document.getElementById("toto").style.display="none";
```

- Chaque propriété CSS correspond à un attribut (les tirets sont remplacés par du camelCase)
- La valeur est une chaîne de caractères, parsée comme une valeur CSS.

Démo [demo/style.html]

## Récupérer le style CSS

- Attention, les propriétés récupérées avec `.style` correspondent **uniquement** au contenu de l'attribut HTML `style="..."`
- Pour récupérer le style couramment appliqué par le navigateur (depuis une feuille de style par exemple), il faut utiliser `getComputedStyle(element)`

Démo [demo/style2.html]

## Manipuler les classes

- En général on ne manipulera pas directement le style : séparation entre présentation (CSS) et comportement (JS)
- La façon propre de faire est de passer par des classes, dont le style est défini indépendamment du script
- Pour manipuler les classes, on utilisera **la propriété `classList`** des éléments :

```
toto = document.getElementById("toto");
toto.classList.add("tutu");
toto.classList.remove("titi");
if (toto.classList.contains("foobar"))
    toto.classList.toggle("erreur");
```

## Modifier le texte d'un nœud

- Pour modifier le texte, par ex. d'un paragraphe, on peut récupérer son nœud textuel et modifier son attribut `nodeValue` :

```
var para = document.querySelector("p");
alert(para.firstChild.nodeValue); // affiche le texte
para.firstChild.nodeValue = "nouveau texte !";
```

- Pas très robuste, car le paragraphe peut contenir d'autres nœuds (par ex. un élément `em`), auquel cas on ne remplace pas tout.
- Solution plus simple : **attribut `textContent`**, qui correspond au texte concaténé de *tous* les descendants du nœud
- En modifiant `textContent` on remplace tous les descendants du nœud par un unique nœud de texte

Démo [demo/text.html]

# [#innerHTML]

## L'attribut `innerHTML`

- Parfois on veut remplacer le contenu du nœud par d'autres nœuds
- `innerHTML` fonctionne de la même façon que `textContent`, mais en « gardant les éléments HTML » :
  - en lecture, il renvoie une représentation HTML de l'arbre DOM qui descend du nœud
  - en écriture, il construit un arbre DOM avec le HTML donné et remplace l'arbre DOM qui descend du nœud par le nouvel arbre

démo [demo/inner.html]

- Moyennement propre et robuste
  - on ne manipule qu'une sérialisation du DOM ; en particulier, on ne retrouve pas exactement ce qu'on a mis (démo) [demo/inner2.html]
    - les anciens nœuds sont supprimés, et de nouveaux sont créés — *même* avec l'opérateur `+=` !



- Peut provoquer des failles de sécurité (injections) et n'est pas très efficace
- Très pratique pour les tests et les bidouilles rapides, mais :
  - pour modifier seulement le texte, aucune raison de ne pas utiliser `textContent`
  - il est très déconseillé de l'utiliser pour autre chose qu'un simple texte avec un peu de balisage (ou pour insérer du HTML d'une source externe *de confiance*)
  - **interdit dans le cadre de ce cours**
- Au passage, il existe aussi `outerHTML`, qui correspond au code HTML de l'élément entier, pas seulement son contenu.

## Créer des nœuds de l'arbre DOM

- On peut créer un nœud DOM

```
var newP = document.createElement("p");
var newTxt = document.createTextNode("contenu");
newP.appendChild(newTxt);
```

- puis l'attacher comme *fil*s à un nœud existant

```
var maDiv = document.getElementById("toto");
maDiv.appendChild(newP);
```

- Tant qu'il n'est pas attaché, le nœud n'est pas dans l'arbre, donc on ne peut pas le voir (ni le récupérer avec les méthodes habituelles)
- Remarque : si on appelle `appendChild()` sur un nœud déjà attaché dans l'arbre, le nœud n'est pas copié mais *déplacé*

## Créer des éléments avec des attributs

- Les éléments que l'on crée n'ont aucun attribut, il faut les ajouter explicitement

```
var monA = document.createElement("a");
monA.setAttribute("href", "http://example.com");
monA.setAttribute("title", "Exemple");
var monText = document.createTextNode("Le site example.com");
monA.appendChild(monText);
maDiv.appendChild(monA);
```

## Autres manipulations

- Supprimer un nœud fils : `papa.removeChild(toto)`
  - supprimer un élément : `toto.parentNode.removeChild(toto)`
- Insérer un fils avant un autre : `papa.insertBefore(nouveau, toto)`
  - Plus général : `insertAdjacentElement` [<https://developer.mozilla.org/en-US/docs/Web/API/Element/insertAdjacentElement>], existe aussi en version texte (`insertAdjacentText`) et HTML (`insertAdjacentHTML` — mêmes remarques que pour `innerHTML`)

- Cloner un nœud : `toto.cloneNode(true)`
  - le paramètre indique qu'on veut cloner le nœud avec tous ses descendants (c'est généralement ce qu'on veut)
  - attention, il faut attacher le nouveau nœud dans l'arbre ensuite
- Remplacer un fils par un autre : `papa.replaceChild(nouveau, toto)`

# [#live-collections]

## Attention aux collections « vivantes »

- La plupart des `NodeList` et `HTMLCollection` sont « vivantes », c'est-à-dire qu'elles sont actualisées en direct
- Attention aux manipulations de l'arbre lors d'un parcours d'une telle collection !

Un exemple [demo/live.html]

- Solutions : itérer en partant de la fin, ou alors convertir la collection en tableau

```
liveCol = document.getElementsByTagName("li");
tableau = Array.prototype.slice.call(liveCol);
/* on parcourt le tableau, qui reste statique */
for (i = 0; i < tableau.length; i++)
    if (tableau[i].classList.contains("suppr"))
        tableau[i].parentNode.removeChild(tableau[i]);
```

- À noter : `querySelectorAll` renvoie une collection statique, mais c'est un cas rare.
- **Solution recommandée : n'utiliser que `querySelectorAll`**, à moins d'en avoir besoin dans une *tight loop*, auquel cas ça vaut le coup de se poser la question et d'évaluer les performances (par exemple avec [jsperf](https://jsperf.com/) [https://jsperf.com/])

## Bilan de cours

- Le DOM permet :
  - de construire une représentation en arbre d'un document
  - d'offrir une interface commune d'accès et modification du document, quelque soit le langage applicatif
- Grâce à l'implémentation de l'interface DOM dans le langage JavaScript, nous pouvons directement dans le navigateur :
  - repérer un élément dans l'arbre DOM
  - modifier ses attributs, notamment ses classes, pour modifier son apparence
  - ajouter ou supprimer des éléments du DOM
- L'interface DOM permet aussi de réagir aux événements : on verra ça au prochain cours

## Spécifications et normes

- Normes du W3C sur le DOM [http://www.w3.org/TR/dom]

## Références et guides

- [Document Object Model](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model) [https://developer.mozilla.org/en-US/docs/Web/API/Document\_Object\_Model] sur MDN
- [Web API Interfaces](https://developer.mozilla.org/en-US/docs/Web/API) [https://developer.mozilla.org/en-US/docs/Web/API] sur MDN



[<http://creativecommons.org/licenses/by-nc-sa/4.0/>]

Ce cours est mis à disposition selon les termes de la [licence Creative Commons Attribution — Pas d'utilisation commerciale — Partage dans les mêmes conditions 4.0 International](http://creativecommons.org/licenses/by-nc-sa/4.0/) [<http://creativecommons.org/licenses/by-nc-sa/4.0/>].