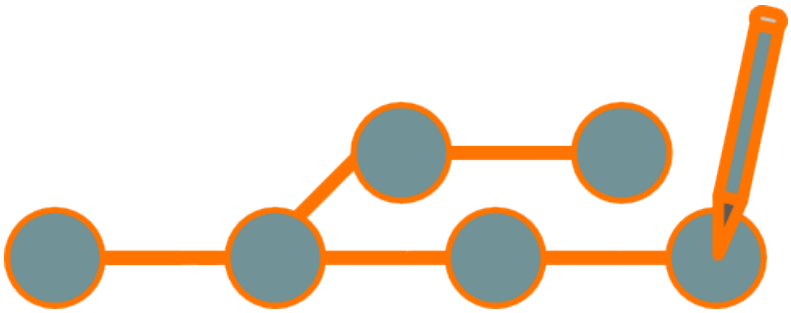


Processus de rédaction technique

Version 1.5

carrereo@gmail.com

Table des matières

1. Documentation technique : diminuer les coûts, améliorer la satisfaction client

La documentation technique, c'est comme une ampoule : une ampoule basse consommation demande un investissement plus important en début de cycle de vie, mais a rapidement un coût plus faible.

Comme une ampoule basse consommation, un processus de rédaction technique industriel diminue les coûts. Il réduit également le *time to market*. À coût initial légèrement supérieur ou égal, il améliore également la qualité.

Coût moindre
- moins de volume à créer
- suppression des mises à jour répétitives
- moins de volume à traduire

***Time to market* réduit**
- réutilisation maximale du contenu
- risque zéro de perte de données

Qualité améliorée
- briques d'information facilement optimisables
- cohérence parfaite du contenu d'entreprise

Une documentation industrielle repose sur :
- un format documentaire modulaire,
- un format de rédaction structurée,
- une chaîne de production et de publication documentaire fiable.

Si la chaîne de création et de publication choisie repose sur des logiciels *open-source*, le coût de mise en place et d'apprentissage peut même être compensé par l'économie sur les licences de logiciels. En tout état de cause, de trop nombreuses sociétés de haute technologie ont industrialisé leurs processus métier, mais laissent en friche la création, la gestion et la publication de leur contenu d'entreprise. Les coûts cachés (rédaction par des ingénieurs et non par un rédacteur technique compétent, mauvaise exploitation du capital immatériel, diminution de la satisfaction client, augmentation des coûts de support, etc.) peuvent être considérables. Pourtant, les solutions et les compétences existent.

De la rédaction à la communication technique

Le but de la communication technique est de transformer les prospects en clients satisfaits. Le rédacteur technique fournit au marché l'information dont il a besoin pour sélectionner, évaluer et utiliser une solution de haute technologie. Au sein de l'entreprise, il est l'interface entre les services R&D et marketing. À l'extérieur, il crée le dialogue entre l'entreprise et ses différents publics.

La communication technique est souvent réduite à la rédaction technique. La rédaction technique est destinée à fournir la documentation des produits, et intervient en aval de la vente. La communication technique intervient dès l'amont du processus de vente et accompagne le produit tout au long de son cycle de vie. Destinée autant au grand public, aux journalistes et aux prospects qu'aux clients, elle dépasse et englobe la rédaction technique, destinée uniquement aux utilisateurs.

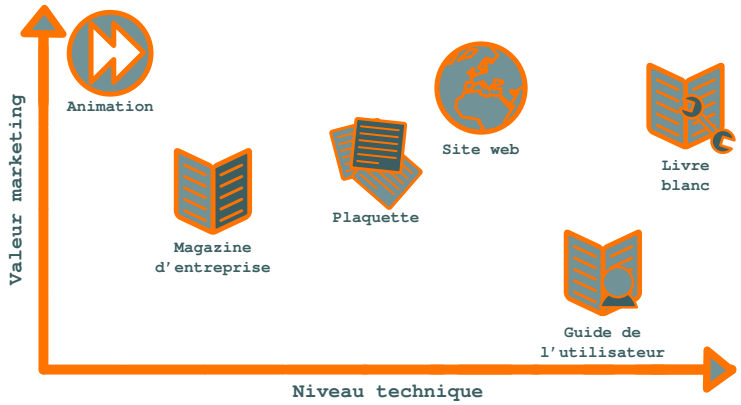


Fig. 1.1 – *Supports de rédaction marketing et technique*

La communication technique a pour but de montrer l'adéquation d'un produit aux besoins de sa cible. Elle recourt pour cela à différents supports, plus ou moins adaptés au niveau d'expertise de son public et à son statut par rapport à l'entreprise (grand public, journalistes, prospects, clients...). Le rédacteur technique doit adapter son message à chaque public. Utilisant toutes les ressources de la communication (rédaction, illustrations, films, animations...) il prend constamment en compte la dimension marketing. Pour augmenter les ventes, tout support de communication doit être un outil marketing.

Mais peut-on être à la fois logique et créatif ? C'est nécessaire dans les domaines de la composition musicale, de l'architecture et du développement informatique. C'est également le cas pour un rédacteur technique.

Ceci nécessite :
- une étude de l'adéquation entre les besoins du public et les moyens de l'entreprise,
- une bonne capacité de création et de rédaction,
- une gestion de projet rigoureuse,
- un processus industriel de production et de valorisation du contenu.

Ce document présente quelques exemples de supports de communication technique, leur intérêt marketing, leur adéquation au public et les modalités de leur valorisation.

Valoriser un contenu signifie :
- produire un contenu de qualité adapté à sa cible,
- conserver tout le contenu existant, dans ses différentes versions,
- réutiliser ou recycler à bon escient le contenu existant.

Les trois niveaux de la documentation technique

Si l'on compare la documentation technique à un jardin, on peut la classifier selon les niveaux suivants :

Friche
- Pas d'utilisation de processus documentaire.
- Création de la documentation par des équipes non dédiées.
- Utilisation de formats non adaptés ou utilisation incohérente de formats adaptés.

Jardin à l'anglaise
- Utilisation de processus documentaires fiables.
- Création de la documentation par des équipes dédiées.
- Utilisation cohérente de formats adaptés, mais non structurés.

Jardin à la française
- Utilisation de processus documentaires fiables.
- Création de la documentation par des équipes dédiées.
- Utilisation cohérente de formats structurés.

Les formats non adaptés à la rédaction technique sont par exemple les formats de traitement de texte, qui ne dissocient pas suffisamment la mise en page du contenu.

Les formats adaptés sont les formats de type FrameMaker, qui dissocient (relativement) la mise en page du contenu, mais ne sont pas sémantiques.

Les formats structurés sont les formats sémantiques de type DocBook ou DITA (Darwin Information Typing Architecture) XML.

Le stade du jardin à l'anglaise est déjà très satisfaisant et garantit qu'une information de qualité est fournie à l'utilisateur. Celui du jardin à la française permet en outre à l'entreprise de mieux maîtriser son contenu et de réduire les coûts de production.

À elle seule, la présence d'une des trois composantes (processus, équipe dédiée et format adapté) ne peut garantir un résultat satisfaisant. Confiez par exemple des outils permettant de générer du contenu au format DITA XML à des collaborateurs dont la communication technique n'est pas le métier ou sans mettre en place de processus de gestion du cycle de vie de la documentation technique, et vous obtiendrez des résultats décevants. Seule la présence conjointe de ces trois éléments fournira un résultat optimal.

Un index est-il utile dans un PDF ?

À l'heure des documents dématérialisés, un index est-il un élément indispensable d'une bonne documentation technique ?

La recherche en plein texte semble avoir détrôné l'index. Les notices techniques ne sont plus qu'exceptionnellement fournies aux clients sous forme papier. Ce drôle d'objet qu'est le PDF, format d'échange entre un format source non destiné aux clients et une version imprimée qui ne l'est que de manière marginale, est entré dans les mœurs. La séquence de touches *Ctrl+F* est un réflexe plus naturel aujourd'hui pour qui recherche une information.

Un texte destiné au Web recourra à une grande dispersion terminologique pour accroître sa visibilité sur les moteurs de recherche. L'emploi des synonymes est de rigueur pour donner au lecteur potentiel plusieurs chemins d'accès à la source d'information qui peut l'intéresser. Les moteurs de recherche ont rendu l'index caduc.

Si la documentation technique utilise une terminologie cohérente, l'efficacité de la recherche en plein texte est réduite : si le rédacteur technique a utilisé uniquement le terme *répertoire*, le lecteur qui recherche le mot *dossier* passera à côté de l'information qu'il recherche.

L'index, s'il est bien réalisé, a alors toute son utilité. Seul problème : créer un bon index demande un effort important en toute fin de projet, juste avant l'heure de la livraison. Et fournir un mauvais index n'a aucun intérêt ni pour le client, ni pour l'entreprise.

Un index est donc paradoxalement plus utile pour une bonne documentation que pour une mauvaise (du moins, une documentation dont la terminologie n'est pas cohérente). Mais son rapport coût/utilité est faible. C'est un luxe que l'entreprise peut rarement s'offrir, mais certainement pas le premier aspect qualitatif qu'il faut améliorer. Un index est la cerise sur le gâteau d'une documentation technique - le plus important reste le gâteau.

Principe de simplicité KISS

Le principe KISS (Keep it simple stupid), n'est pas spécifique à la rédaction technique. Il s'agit d'un principe général d'ingénierie, qui postule que tout objet matériel ou virtuel construit par l'homme est plus fiable et plus facile à maintenir et à faire évoluer si sa complexité est volontairement réduite. C'est le principe mis en exergue par antiphrase par les Shadocks : *Pourquoi faire simple quand on peut faire compliqué* ?

C'est ce principe qui a prévalu à la conception des montres *Swatch*, dont le cahier des charges stipulait qu'elles devaient embarquer deux fois moins de pièces que leurs consœurs. Résultat : des montres moins chères, plus fiables et disponibles en versions sans cesse renouvelées.

Ce qui a spectaculairement fonctionné pour des montres peut être appliqué avec le même succès à la documentation technique. Quel que soit le format utilisé [1] , le rédacteur technique peut tout aussi bien construire un document élégamment architecturé, qu'une usine à gaz. Avec DITA XML, il lui suffit de ne pas centraliser les *conref* ou de les imbriquer exagérément. Sous FrameMaker, c'est encore plus simple, peu de garde-fous étant posés : la multiplication des styles et des *overrides* peut rapidement rendre ingérable n'importe quel document.

De même, dans la formulation de ses phrases, le rédacteur technique doit toujours avoir le principe KISS à l'esprit. Il est facile de construire des phrases alambiquées qui dénotent plus une mécompréhension du sujet qu'un raffinement de l'écriture [2] . Construire une phrase simple demande un effort de compréhension de son sujet. La rédaction du contenu et son appréhension par son destinataire deviennent alors aisées. C'est un aspect fondamental du métier de rédacteur technique. Le rédacteur technique apporte ainsi une véritable valeur ajoutée au produit qu'il documente.

1. Même si les formats qui distinguent le contenu de la mise en page sont dans leur principe plus aptes à la mise en œuvre de la philosophie KISS.
2. Ce genre de phrase est d'ailleurs souvent impossible à traduire.

Formats et outils

Lorsqu'une entreprise décide d'industrialiser la rédaction technique, elle se pose d'emblée la question des outils. Or, plutôt que les outils, ce sont les formats sous-jacents qui sont le point essentiel.

La plupart des éditeurs, afin de disposer d'un marché captif obligé de régulièrement payer des mises à jour de leurs produits, ont en effet développé des formats propriétaires que seuls leurs logiciels sont à même de modifier. Un fichier MS Word ou un fichier FrameMaker ne peuvent ainsi être modifiés que *via* les outils éponymes. Choisir un tel format risque donc de limiter les choix ultérieurs de l'entreprise et de se révéler coûteux : il faut une licence par utilisateur, qu'il soit rédacteur technique, contributeur occasionnel ou traducteur.

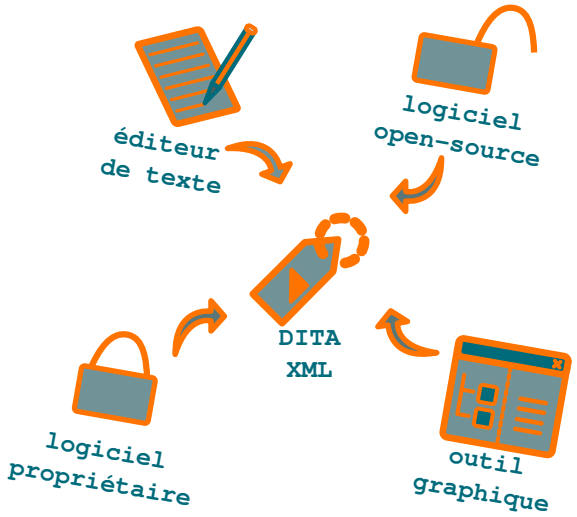


Fig. 1.2 – *Un format standard laisse le choix de l'outil.*

Si l'on réfléchit en termes de formats, en revanche, il est possible de mettre en place des solutions évolutives. Un format ouvert tel que OpenDocument [3] ou DITA XML, par exemple (seul le second étant un format industriel de rédaction technique), n'est pas lié à un outil donné. Il est donc possible de le modifier et de le manipuler à l'aide de différents logiciels. Les formats structurés de type DocBook et DITA XML liés à un schéma XSD normalisée peuvent notamment être facilement gérés, de la création à la publication, à l'aide de toute une panoplie d'outils, de l'éditeur de texte libre à la suite logicielle propriétaire et graphique.

3. http ://fr.wikipedia.org/wiki/OpenDocument

2. Rédaction technique : un processus industriel

La rédaction technique repose sur des processus rationnels. Trop souvent associée à un fort aspect littéraire, elle est fréquemment laissée à l'improvisation et à l'inspiration du rédacteur technique. Le rédacteur technique, comme les autres intervenants de l'entreprise, doit répondre à ses objectifs de manière prévisible et reproductible.

Ce processus repose sur une méthodologie rigoureuse et une chaîne de production fiable.

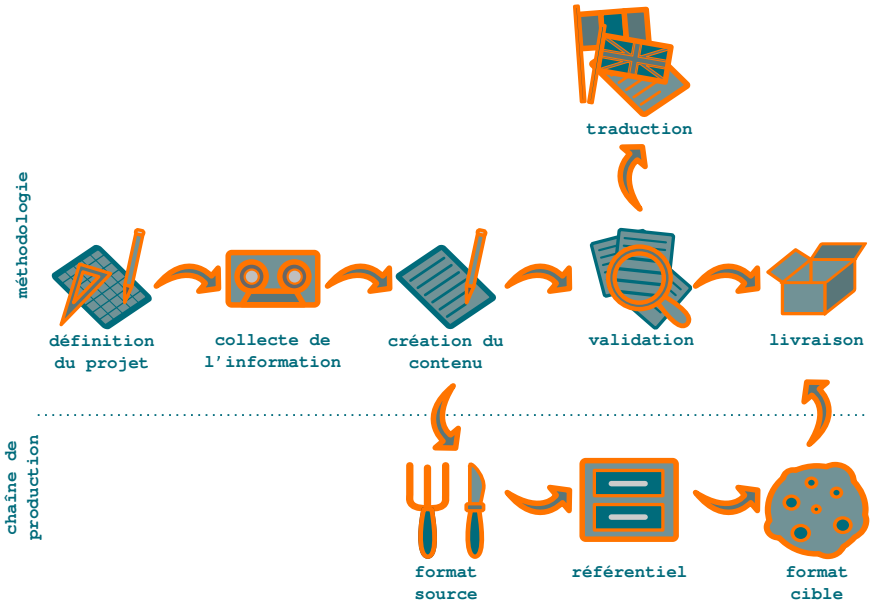


Fig. 2.1 – *Processus de rédaction technique*

Pour créer et valoriser un contenu à forte valeur ajoutée pour l'entreprise, le rédacteur technique dialogue constamment non seulement avec tous les acteurs internes de la société, mais aussi avec son écosystème : partenaires, journalistes, utilisateurs, etc. Il fournit ainsi aux différents publics l'information dont ils ont besoin. Ceci renforce l'image de marque de la société, améliore la satisfaction client et facilite la perception des avantages produit par les prospects. Le rédacteur technique s'appuie sur une chaîne de production aussi automatisée que possible. En mettant en place un processus industriel et reproductible, il diminue les coûts de production et fournit un niveau de qualité constant, adapté aux buts de l'entreprise.

Définition du projet

Un projet de rédaction technique apporte une valeur ajoutée aux produits et aide l'entreprise à mieux commercialiser son offre sur son marché. Mais, comme pour les projets de R&D ou de marketing, la définition du projet permet d'en estimer le budget et les retombées.

Communiquer des informations techniques sans savoir à qui ni dans quel but est un effort vain. Avant d'initier un projet de rédaction technique, il est indispensable de clairement le définir. Il convient notamment de déterminer :

son objectif Augmenter la notoriété de l'entreprise, accroître sa couverture médias, amener les prospects à prendre contact avec la société, réduire les coûts de support technique…

sa cible Grand public, journalistes, prospects, clients…

sa forme Livre blanc, mode d'emploi ou guide de l'utilisateur, brochure et *flyer*, site *web*, magazine d'entreprise, *print* ou *online*…

sa langue Suivant votre domaine d'activité, le projet sera décliné en une ou plusieurs langues (principalement l'anglais dans le secteur informatique).

son mode de diffusion Le document final peut être publié sur un site Internet ou Extranet, envoyé sous forme de fichier joint par *mail*, remis en mains propres au format papier, etc.

L'analyse des résultats du projet donne ensuite de précieux renseignements pour améliorer encore l'impact des projets suivants.

Collecte de l'information

Le rédacteur technique collecte l'information auprès de différentes sources, internes et externes à l'entreprise.

Une fois le projet de rédaction technique clairement défini, le rédacteur technique collecte toute l'information disponible :

— spécifications du produit, Intranet, pages Trac [4],
— interview du service R&D,
— manipulation du produit,
— interview du service marketing,
— interview de clients,
— analyse de la concurrence,
— lecture de la presse spécialisée.

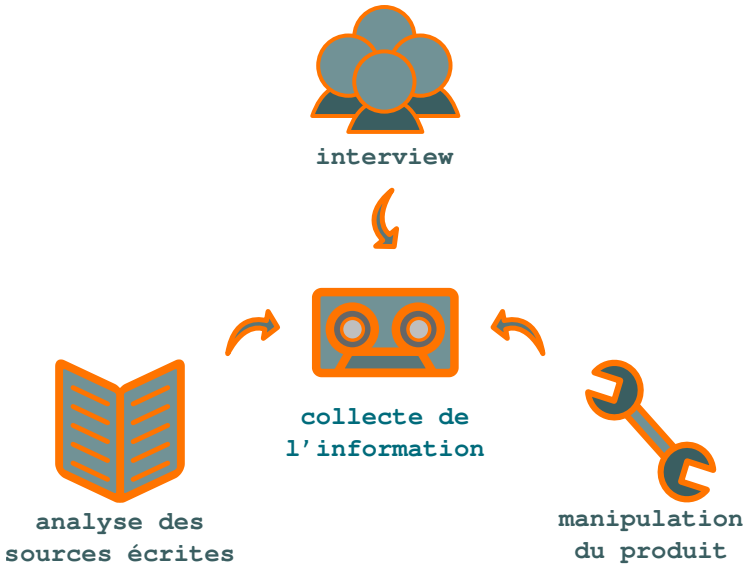


Fig. 2.2 – *Collecte de l'information*

Les informations doivent être recoupées pour minimiser le risque de transmettre des informations erronées ou plus à jour. Le rédacteur technique doit se livrer à un véritable travail d'enquête. En se mettant à la place de l'utilisateur, il vérifie chaque information et fait le tri entre les données pertinentes et celles qui ne seront que du bruit perturbant le message.

Premier utilisateur des solutions développées par la société, le rédacteur technique a le rôle du *candide* qui remet chaque aspect de l'information à transmettre dans son contexte. Il peut par exemple décider, contre l'avis de la direction technique, de passer sous silence des informations très techniques dans le guide de l'utilisateur. Inversement, il pourra étayer une brochure commerciale de données techniques précises pour étayer le discours marketing.

Tester les produits pour les documenter

Le rédacteur technique ne peut fournir une documentation utile aux clients de l'entreprise s'il se contente de mettre en forme des informations glanées auprès des différents acteurs de la société. Jouant le rôle de Candide, il est le premier représentant des utilisateurs et se doit de tester les produits dans des conditions proches des leurs.

Un conte chinois narre comment des aveugles se sont retrouvés confrontés à un éléphant. Aucun d'entre eux, et pour cause, n'ayant une perception globale de l'animal, chacun en eut une image différente : celui qui en tenait une patte le

4. http ://trac.edgewall.org

prenait pour un arbre, celui qui en étreignait la trompe le confondait avec un serpent, celui qui avait empoigné une défense l'identifiait à une lance, et celui qui s'agrippait à une de ses oreilles croyait qu'il s'agissait d'un éventail.

Fig. 2.3 – *Conte des aveugles et de l'éléphant*

Le rédacteur technique qui demande aux différents intervenants de l'entreprise à quoi sert le produit dont il doit créer la documentation et comment il fonctionne se retrouve comme celui qui demande aux aveugles à quoi ressemble un éléphant : pour la R&D, il s'agit de code élégamment rédigé, pour le marketing, d'une offre à positionner face à la concurrence sur son marché, pour le support technique, d'un exécutable dont il faut corriger les bugs, etc.

Pour avoir une vision réaliste de l'objet qu'il est censé décrire, le rédacteur technique doit donc l'appréhender par lui-même et se faire sa propre opinion, qu'il pourra ensuite confronter à celle des autres acteurs de l'entreprise. Le rédacteur technique est un pragmatique qui s'intéresse à la pratique, non à la théorie. S'il ne consulte que les développeurs, par exemple, il aura peu de chance de pouvoir créer une documentation satisfaisante pour l'utilisateur :

- d'une part, les développeurs ont souvent une vision idéaliste du fonctionnement de leur produit, différente du comportement de ce dernier en conditions réelles d'utilisation,
- d'autre part, une déperdition d'information se produit nécessairement entre :
  - ce que le développeur sait,
  - ce que le développeur exprime,
  - ce que le rédacteur technique comprend,
  - ce que le rédacteur technique rédige,
  - ce que l'utilisateur comprend.

Si le rédacteur technique teste réellement le comportement du produit dans des conditions aussi proches que possible de celles rencontrées par l'utilisateur, les trois premières causes de déperdition d'information sont quasiment inexistantes. Pour réduire les deux dernières, il ne lui reste plus qu'à filtrer, organiser et exprimer l'information qu'il a recueillie de manière adaptée au média qui la véhiculera et aux connaissances techniques de son destinataire.

Dans les faits, une telle demande peut sembler de prime abord incongrue en interne et se heurter à la lourdeur de la mise en place d'une plateforme de test. Ce n'est généralement qu'après les premiers retours clients ou les tests produits dans la presse que les différents interlocuteurs comprennent pleinement l'apport de cette démarche. C'est souvent seulement à partir de ce moment là que la rédaction technique gagne ses lettres de noblesse. Et que la documentation technique n'est plus seulement vue comme un mal nécessaire, mais comme une véritable valeur ajoutée.

Création du contenu

Le rédacteur technique crée le contenu du projet de rédaction technique dans un dialogue constant avec les différents acteurs de la société : services R&D, marketing. Il prend en compte en amont les différentes contraintes liées au cycle de vie des supports de rédaction technique.

En particulier, le rédacteur technique a soin de :

- faire valider le contenu à ses interlocuteurs afin d'apporter les modifications nécessaires aussi tôt que possible ; ceci garantit que le résultat sera conforme au projet initialement défini,

— minimiser le volume de texte et d'images sources afin de réduire les coûts de production, de maintenance et de traduction,
— prendre en compte dès le début du projet les contraintes d'internationalisation.

Format source

Le contenu d'un projet de rédaction technique est créé dans un format source, différent du format des livrables, le format cible. Pour reprendre une image fréquemment utilisée en développement logiciel, le format source est la recette de cuisine, le format cible, le plat. En photographie, le format source est le format RAW [5], qui est généré par l'appareil photo, et sur lequel les photographes professionnels préféreront apporter les retouches, et le format cible, le format JPEG.

Les traitements de texte nous ont déshabitués à distinguer le fond de la forme. Mais confondre les deux entraîne beaucoup d'erreurs et de perte de temps.

En effet, le document présenté à l'utilisateur présente deux aspects fondamentaux :
— le contenu,
— la mise en page.
Au cours du développement d'une documentation technique, ces deux aspects doivent être clairement distincts. Ils peuvent être pris en charge par deux intervenants différents :
— le rédacteur technique,
— le graphiste [6].
Lorsque la mise en page a une importance équivalente à celle du contenu, ou lorsqu'elle doit être variée, comme dans le cas d'une brochure commerciale, la rédaction et la mise en page s'opèrent sous des outils différents :
— éditeur de texte,
— logiciel de PAO, par exemple InDesign ou Scribus.
Lorsque la mise en page a une importance moindre que celle du contenu, ou lorsqu'elle doit être homogène, comme dans le cas d'une documentation technique, la rédaction et la mise en page s'opèrent sur :

les mêmes fichiers par exemple, des fichiers FrameMaker,

des fichiers différents par exemple, des fichiers de contenu XML et une feuille de style XSLT.

Dans un fichier FrameMaker, la séparation du fond et de la forme est élevée mais pas totale : le contenu et la mise en page sont placés dans le même fichier. FrameMaker applique une maquette de page homogène à tout un fichier, mais autorise l'ajout manuel d'éléments de mise en page. La même maquette peut être dupliquée pour tout le document, ou une maquette différente peut être utilisée pour chaque fichier qui compose ce dernier.

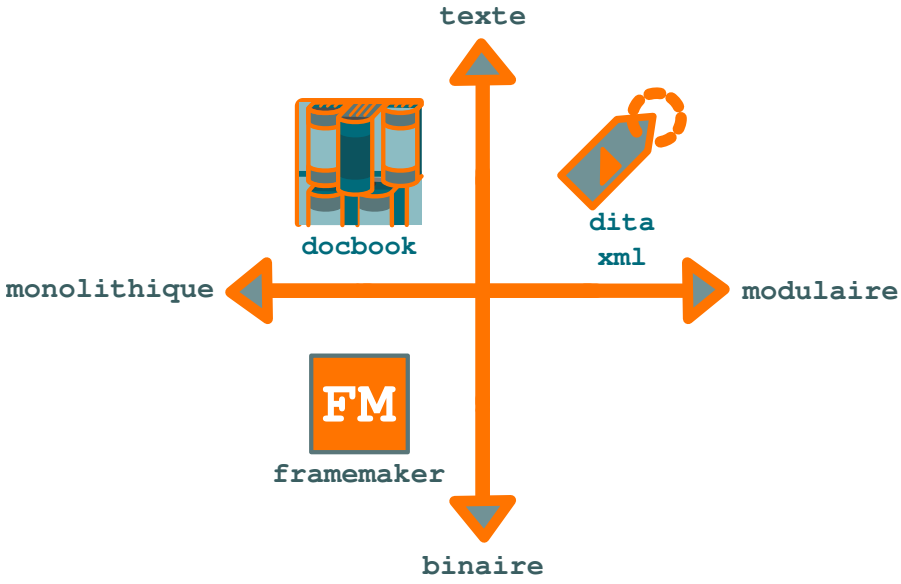


Fig. 2.4 – *Formats sources : degré de modularité et format*

Les formats sources peuvent être classés selon leur degré de modularité et leur format de fichier.

5. http ://fr.wikipedia.org/wiki/RAW_(format_d%27image)
6. Si le rédacteur technique met lui-même en page ses documents, il change de rôle lorsqu'il effectue cette opération.

Les formats XML structurés DocBook et DITA XML appliquent une maquette de page homogène à tout un document, et n'autorisent pas l'ajout manuel d'éléments de mise en page [7], ni l'application de maquettes différentes aux différents fichiers qui composent le document.

Format	Application d'une mise en page homogène	Possibilité de mise en page manuelle
MS Word	Non	Oui
FrameMaker	Oui	Oui
DITA XML	Oui	Non

Si contenu et mise en page sont intimement liés, comme sous un traitement de texte, il est difficile de modifier le contenu sans perturber la mise en page. Résultat : à chaque publication d'une nouvelle version d'une documentation technique, l'équipe de rédaction technique passe de longues heures à corriger les erreurs de mise en page générées par le logiciel. Le phénomène est moindre sous FrameMaker mais reste important. Il est nul avec les formats DITA XML et DocBook (les seules erreurs qui peuvent se produire sont des erreurs de compilation dues à une syntaxe XML erronée ; ces erreurs sont facilement rectifiables).

Les fichiers sources d'une documentation technique sont au format :
- binaire ou,
- texte.

Ce format est également :
- WYSIWYG ou,
- structuré.

Enfin, ce format est :
- modulaire ou,
- monolithique.

Ce dernier aspect détermine la manière dont le format gère le *single-sourcing* :
- selon une logique *livre vers aide en ligne* ou,
- selon une logique *aide en ligne vers livre*.

Les formats disponibles peuvent donc être classés selon le tableau suivant :

Format	Texte	Structuré	Modulaire
FrameMaker natif	Non	Non	Limité
DocBook	Oui	Oui	Limité
DITA XML	Oui	Oui	Oui

FrameMaker et DocBook ne sont pas pleinement modulaires, car les plus petits éléments d'information manipulables ne sont pas génériques : ils contiennent des informations telles que la structure de table des matières ou les références croisées qui ne sont valables que dans un nombre limité de contextes.

Documents monolithiques ou modulaires

Le format source peut reposer sur des fichiers monolithiques ou sur des grappes de fichiers modulaires.

Les fichiers monolithiques (par exemple MS Word, LibreOffice ou FrameMaker) centralisent tout le contenu dans un seul fichier, facile à manier, mais qui limite le partage du contenu ; le risque de disposer d'informations incohérentes ou en doublon est alors important.

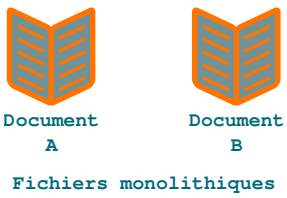


Fig. 2.5 – *Format source de rédaction technique monolithique*

Les grappes de fichiers modulaires (par exemple DITA XML) agrègent le contenu de multiples fichiers, ce qui favorise le partage et la réutilisation de blocs de contenu. Un tel système est difficile à mettre en place au niveau de toute l'entreprise, mais devrait être la norme pour une équipe de rédaction technique.

7. Ou très peu : dans les fichiers de contenu, il est seulement possible de mettre du texte en gras ou en italique, pas d'en changer la police, le corps ou la couleur.

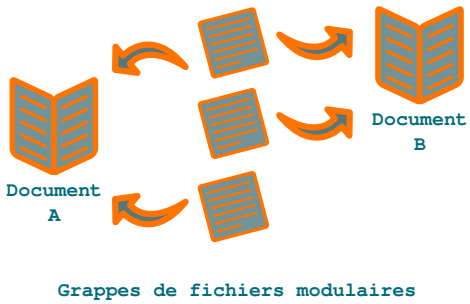


Fig. 2.6 – *Format source de rédaction technique modulaire*

Certains traitements de texte proposent de gérer des documents modulaires, mais ils le font mal. Inversement, un document DocBook ou DITA XML, par exemple, peut être monolithique, mais perd alors de sa souplesse.

Qu'est-ce qu'un module d'information ?

Le système modulaire le plus connu au monde est certainement celui des briques Lego. Adapté à la documentation technique, le principe des modules permet d'améliorer la qualité des manuels techniques et la productivité du rédacteur technique.

Mais suffit-il de convertir sa documentation de FrameMaker vers un format structuré tel que DITA XML pour obtenir une documentation modulaire ? Hélas, non. Si le contenu de départ mélange les informations de tout type (concepts, procédures pas à pas, référence), il sera toujours possible de le convertir au format DITA XML en ne respectant pas rigoureusement la sémantique DITA XML. Voire en modifiant les feuilles de style XSLT ou en spécialisant les XSD pour les rendre plus laxistes.

Or, si l'on obtient au final un document se basant sur des fichiers correpondant chacun à un schéma XSD différent (*concept*, *task*, ou *reference*), on n'obtient pas forcément ainsi une véritable documentation modulaire. En effet, essayez de construire alors un document ne regroupant que les fichiers d'un seul type : votre document aura toutes les chances d'être incomplet et incohérent.

Cette documentation n'est pas modulaire, car elle ne repose pas sur de véritables modules d'information. Un module est un élément atomique complet et cohérent qui peut être réutilisé dans différents contextes. Si vous avez divisé votre document monolithique original en une multitude de fichiers, vous n'avez pas encore créé de modules d'information. La seconde étape consiste à ré-écrire chaque fichier (selon par exemple l'approche minimaliste) pour le rendre plus générique et en faire un véritable module. Il faut évidemment adopter une approche structuraliste et décider du contenu de chaque module dans la perspective de l'architecture documentaire globale. De même, des mentions telles que *Voir la section suivante* devront être remplacées par des références croisées. Idéalement, ces références croisées ne se situent pas dans les fichiers de contenu proprement dit sous la forme :

```
<related-links>
  <link href="content.dita#content"/>
</related-links>
```

mais dans une section *reltable* propre à chaque fichier *ditamap*.

Les modules sont ainsi parfaitement décontextualisés, et les informations de structure telles que les références croisées sont placés dans des fichiers ne comportant pas de contenu textuel.

Fichiers binaires ou texte

Les formats sources sont des formats binaires ou texte.

Les formats binaires sont *opaques* :	si on les ouvre avec un éditeur de texte de type *notepad*, tout ce que l'on voit est une suite de caractères hiéroglyphiques ; il n'est donc la plupart du temps possible de les modifier qu'avec un seul logiciel.
Les formats texte sont *transparents* :	si on les ouvre avec un éditeur de texte, on voit du texte et des balises ; il est donc possible de les modifier avec différents logiciels et de leur appliquer des opérations de traitement par lot en ligne de commande, sans même les ouvrir, et d'utiliser de puissantes expressions rationnelles[8].

Référentiel

Le contenu est le capital immatériel de la société et doit être protégé comme tel. Il peut être géré dans différents référentiels : répertoires, mais aussi outils de gestion de contenu d'entreprise et logiciels de gestion de versions.

Git : du fichier au contenu

Vous êtes habitué à manipuler des fichiers ? Git vous invite à penser autrement. Avantage : vous avez une maîtrise beaucoup plus grande de votre contenu.

Qu'est-ce qu'un fichier ? Pour vous, un contenu, image, texte, feuille de calcul ou autre, identifié par un nom. Pour votre système d'exploitation, une suite de bits sur le disque dur à laquelle sont associés un nom de fichier et un chemin de répertoires. Si vous souhaitez gérer votre projet en termes de fichiers sous Git, vous allez au-devant de maintes difficultés. Si vous pensez plutôt en termes de contenu, tout devient beaucoup plus simple.

Si vous donnez un fichier à Git, il le scinde directement en deux choses :
- un contenu (suite de bits, ou *blob*),
- un arbre (lien entre le nom de fichier et le contenu).

Il le stocke ensuite dans l'une des deux zones suivantes :
- l'index (zone temporaire),
- la base de données d'objets (zone persistante).

Lorsque vous ajoutez un fichier (*git add <fichier>*) :
- l'arbre est placé dans l'index,
- le contenu est placé dans la base d'objets.

Lorsque vous *commitez* un fichier (*git commit*) :
- l'arbre est placé dans la base d'objets.

Git ne compare jamais deux fichiers entre eux. Il compare leur résumé, qui est un nombre unique calculé à partir de leur contenu. Si le résumé de deux fichiers est identique, le contenu de ces fichiers est indentique (au bit près).

L'historique de votre projet n'est pas forcément linéaire : vous pouvez lui faire suivre plusieurs routes parallèles, les branches.

Vous ne pouvez créer des branches qu'à partir d'un *commit*. Il faut voir les *commits* comme des ronds-points (la route étant l'historique de votre projet) à partir desquels vous pouvez, si vous le souhaitez, prendre une autre direction dans votre projet.

Si vous créez une branche, disons *test*, alors que des modifications de votre espace de travail ne sont pas *commitées* dans votre branche *master*, les modifications que vous effectuerez s'appliqueront aux fichiers non *commités* de votre espace de travail. Si vous faites une erreur, vous ne pourrez pas retrouver le *statu quo ante* de vos fichiers en revenant à la branche *master*.

Si vous voulez enregistrer votre travail au fil de l'eau afin de pouvoir revenir à tout moment à un état antérieur, il vous faut donc *committer* régulièrement et sauvegarder votre espace de travail, répertoire *.git* y compris, par exemple *via* rsync. Lorsque vous déciderez de partager votre travail, vous pourrez déplacer, fusionner ou supprimer vos *commits* avant de les envoyer sous forme de patchs ou de les déposer sur un dépôt central.

8. http ://fr.wikipedia.org/wiki/Expression_rationnelle

Faire sauter les goulets d'étranglement avec les branches

Les branches *Git* permettent de facilement effectuer en parallèle plusieurs tâches non liées :

Imaginons le scénario de travail suivant :
- On vous demande de migrer une section d'un document à un autre.
- Vous envoyez votre proposition pour validation.
- La validation se fait attendre et vous devez avancer sur d'autres parties des documents.

Comment faire sauter ce goulot d'étranglement ? C'est (relativement) simple :

1. Par défaut, vous travaillez sur la branche *master*. Votre espace de travail contient des modifications que vous ne souhaitez pas *committer* avant validation.
2. Créez une nouvelle branche : *git checkout -b ma-branche*.
3. *Committez* vos modifications sur la nouvelle branche : *git add mes-fichiers*, *git commit -m "mon message de commit"*.
4. Vous repassez sur la branche master *git checkout master* et passez à votre deuxième tâche. 5a. Si votre première tâche n'est pas validée, vous repassez sur la branche provisoire : *git checkout ma-branche* et faites un nouveau commit (que vous pourrez fusionner avec le ou les précédents après validation).
5. Lorsque vous recevez la validation de la première tâche, vous mettez votre travail en cours de côté : *git stash*.
6. Vous fusionnez la branche provisoire avec la branche master : *git merge ma-branche*.
7. Vous récupérez votre travail en cours : *git stash pop*.

Si vous n'avez pas besoin d'effectuer deux lots de tâches en parallèle, vous pouvez sans problème travailler dans votre espace local. Si vous devez revenir sur vos modifications, appellez la commande *git reset –hard HEAD* pour écraser vos fichiers non *commités* du répertoire local par ceux du dernier *commit*.

Organiser son historique avec Git rebase

Git est d'un abord déroutant. Ses *workflows* s'appliquent à du contenu plutôt qu'à des fichiers. Résultat : le travail de groupe et la gestion de différentes versions concurrentes d'un même contenu deviennent beaucoup plus simples.

Git effectue des *commits* atomiques : il applique des lots de modifications sur un contenu souvent réparti sur plusieurs fichiers, au lieu de gérer des *fichiers* proprement dits. Il nous invite à raisonner par lots de tâches sur un contenu et non par fichier.

Ce fonctionnement peut sembler peu intuitif si l'on a l'habitude de travailler fichier par fichier et non tâche par tâche. Mais une fois que l'on a adapté ses habitudes de travail à ce *workflow*, on s'aperçoit :
- que l'on dispose d'un historique beaucoup plus facilement exploitable,
- qu'il est beaucoup plus facile de gérer des versions concurrentes d'un même contenu dans des branches de développement parallèles.

Imaginons que vous ayez identifié deux types de modifications majeurs à apporter à votre contenu :
- les synopsis d'un programme en ligne de commande,
- les corrections grammaticales du texte.

Si votre contenu est réparti dans un ensemble de fichiers modulaires, vous pourriez décider d'apporter en même temps les deux types de modifications dans chaque fichier un à un. Pour répartir le travail sur un groupe de rédacteurs techniques, il vous suffit d'allouer à chacun un lot de fichiers.

Ce *workflow* n'est pas le plus adapté à Git. Si vous utilisez ce système de gestion de versions, il est préférable de diviser le travail en deux lots de tâches, que l'on appelera *synopsis* et *texte*, appliqués concurremment sur tous les fichiers.

Les contraintes de production vous obligeront souvent à scinder ces deux lots de tâches en sous-lots, que vous serez obligé de faire alterner.

Vous *committez* chaque sous-lot à chaque fois qu'il est achevé. Votre historique de *commit* ressemble alors au schéma suivant :

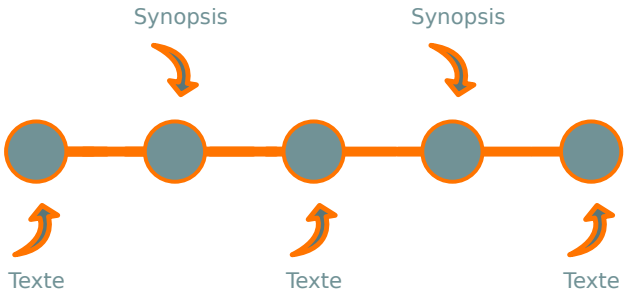


Fig. 2.7 – *Historique Git*

Lorsque vous placerez vos *commits* sur le dépôt central, certains *commits* représenteront une étape intermédiaire de l'une des tâches. Votre historique et vos branches seront donc plus difficiles à exploiter. D'autant plus que les tâches inachevées alternent. Pour en récupérer une seule, il faudra donc choisir soigneusement les *commits via* la commande *git cherry-pick*.

Heureusement, Git vous permet de réorganiser facilement vos *commits* avant de les partager. Lancez la commande *git rebase -i HEAD~5* pour réorganiser les *commits*, de la version en cours aux cinq précédentes, par exemple.

Attention : La commande *rebase* est potentiellement destructive ; veillez à sauvegarder votre espace de travail, répertoire *.git* compris, avant de l'exécuter, sous risque de perdre des données ; vous pouvez également créer une branche de sauvegarde provisoire.

Vous pouvez alors réécrire l'histoire pour proposer à vos collaborateurs un *commit* pour chaque tâche réalisée en son entier, comme sur le schéma suivant :

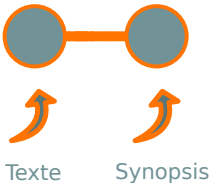


Fig. 2.8 – *Historique Git*

Les *commits* ont tout d'abord été regroupés par type sur la *flèche du temps* de Git, puis fusionnés.

Note : Si vous avez effectué simultanément les deux tâches sur un ou plusieurs fichiers, pas de panique : grâce à la commande *git add -p* vous pouvez répartir vos modifications imbriquées sur les *commits* idoines. Lorsque vous lancez *git status*, vous vous apercevez alors que vos fichiers sont à la fois prêts et non prêts à être *commités* : il y a deux états des fichiers, chaque état représentant un stade partiel de votre travail et la somme des deux représentant la totalité des modifications que vous avez apportées.

Évidemment, vous n'avez plus accès aux *commits* intermédiaires, mais c'est ce que vous souhaitiez : chaque *commit* unique représente un état cohérent de votre contenu.

Ce *workflow* facilite également le travail d'équipe : vous pouvez confier ces tâches à deux membres différents de votre équipe, chacun travaillant dans son espace local. Les modifications du premier sont ensuite fusionnées avec celles du second dans son espace local *via* des *patches*. Enfin, les *commits* sont refactorisés avant de les placer sur le dépôt central.

Important : Moins vous réorganiserez vos *commits* (surtout chronologiquement), plus le risque de devoir corriger manuellement des conflits sera faible. Autrement dit, *git rebase* ne doit pas être une excuse pour ne pas planifier rationnellement son travail.

Quel référentiel pour le travail de groupe ?

Le référentiel le plus fréquemment utilisé pour stocker des fichiers informatiques est le dossier, ou répertoire. Si ce dépôt est parfaitement adapté à la gestion de fichiers par un utilisateur unique sur son disque dur local, il montre rapidement ses

limites pour le travail de groupe.

Pour travailler sur un fichier, le rédacteur technique utilise un programme qui lit le fichier sur son disque dur et en charge une copie en mémoire vive. Les modifications s'effectuent sur cette copie. Lorsque le rédacteur technique enregistre ses modifications, le programme écrase sur le disque dur la version précédente du fichier. La version précédente est donc définitivement supprimée, sauf si le programme a créé une copie de sauvegarde ou si le rédacteur technique a utilisé la fonction *Enregistrer sous*, et non *Enregistrer*, pour créer une nouvelle version du fichier. Dans le premier cas, il n'existe que deux versions du fichier à un instant donné : la version n et la version n-1. Dans le second cas, le rédacteur technique peut créer autant de versions qu'il le souhaite, par exemple en ajoutant le suffixe -1, -2, etc. au nom du fichier.

Les programmes ne gèrent cependant pas la modification concurrente d'un même fichier par plusieurs rédacteurs techniques. Dans le cas d'un fichier disponible sur un disque réseau, imaginons qu'Arsène et Louise ouvrent la même version de ce fichier sous un éditeur de texte. Chacun apporte des modifications différentes dans sa copie chargée en mémoire vive, puis enregistre son travail. Arsène enregistre tout d'abord ses modifications, puis Louise. À la prochaine ouverture du fichier, seules les modifications de Louise figureront dans le fichier.

Pour éviter ce genre de situation, de nombreux programmes verrouillent les fichiers ouverts. Ils ne sont donc disponibles qu'en lecture tant que l'utilisateur qui les modifie en a une copie en mémoire vive (c'est-à-dire, tant qu'il ne l'a pas fermé). Il n'est donc pas possible avec ce système de travailler à plusieurs sur le même fichier et d'effectuer par exemple des modifications transverses par lot, comme modifier le chemin de toutes les images.

Si le programme utilisé ne verrouille pas les fichiers ouverts, une coordination de tous les instants doit s'instaurer entre les membres de l'équipe.

Les répertoires réseau partagés - peu adaptés au travail de groupe

Les fichiers partagés par une équipe de rédaction technique sont souvent stockés dans un répertoire partagé sur le réseau.

Les rédacteurs techniques travaillent directement sur les fichiers partagés, ce qui pose les problèmes suivants :
- risque de pertes de données en cas de défaillance du réseau,
- possibilités de travail off-line (déconnecté) limitées,
- verrouillage des fichiers par les membres de l'équipe qui les ont ouverts.

Même fréquemment sauvegardés, les répertoires ne sont pas un référentiel sûr pour les données : la granulométrie de la sauvegarde est le répertoire, sa fréquence n'est souvent que quotidienne. En cas de perte de données, la restauration se fait répertoire par répertoire, et non fichier par fichier et porte sur des versions dont l'ancienneté dépend de l'administrateur système, et non du rédacteur technique. Fouiller dans les archives est une opération fastidieuse qui peut elle-même être source d'erreurs : en l'absence d'une comparaison fiable et aisée entre plusieurs versions des fichiers, le rédacteur technique peut facilement supprimer des modifications qu'il aurait souhaité conserver en voulant en restaurer d'autres.

Copier un fichier du réseau pour le modifier sur son disque dur personnel, puis écraser la version du réseau par la version locale est une opération des plus périlleuses :
- les membres de l'équipe ne sont pas informés du fait qu'un autre membre modifie ou non le même fichier en même temps qu'eux ; l'un des rédacteurs techniques devra alors renoncer à toutes ses modifications ;
- lors d'une copie manuelle des fichiers, que ce soit *via* un gestionnaire de fichiers graphique ou en ligne de commande, le rédacteur technique peut facilement écraser la version la plus récente par la plus ancienne (on préférera alors avoir recours à un logiciel de synchronisation de fichiers tels que rsync [9] ou Unison [10] (ce dernier étant plus adapté à la synchronisation bidirectionnelle) en ligne de commande sous GNU/Linux ou Windows, ou à un équivalent graphique, tel SyncToy [11]. Cependant, ce type de logiciels se base sur la date de dernière modification des fichiers. Lorsque l'on met à jour ou publie un livre FrameMaker, notamment, ceci peut créer des conflits entre fichiers, FrameMaker enregistrant dans ces cas tous les fichiers du livre, même si leur contenu n'a pas été modifié).

Les systèmes de gestion de versions - rustiques mais fiables

Travailler sur des fichiers sources au format texte, et non binaire, est l'occasion pour le rédacteur technique de gérer son contenu comme les développeurs gèrent leur code : sous un système de gestion des sources tel que Git [12], Subversion ou SourceSafe.

Ces systèmes :

9. http ://rsync.samba.org
10. http ://www.cis.upenn.edu/~bcpierce/unison
11. http ://www.microsoft.com/en-us/download/details.aspx ?id=15155
12. http ://www.git-scm.com

— favorisent le travail de groupe,
— suppriment les copies de fichiers en doublons et
— réduisent le risque de perte de données à presque zéro.

Sur des fichiers texte, et non binaires, un système de gestion de version offre des fonctionnalités supérieures :

— pas de risque de pertes de données en cas de défaillance du réseau [13],
— possibilités de travail *off-line* (déconnecté) poussées [14],
— non-verrouillage des fichiers par les membres de l'équipe qui les ont ouverts.
— possibilité de restauration très fine et dans le temps (depuis le dernier dépôt du fichier sur le référentiel) et en termes de quantité de travail [15].

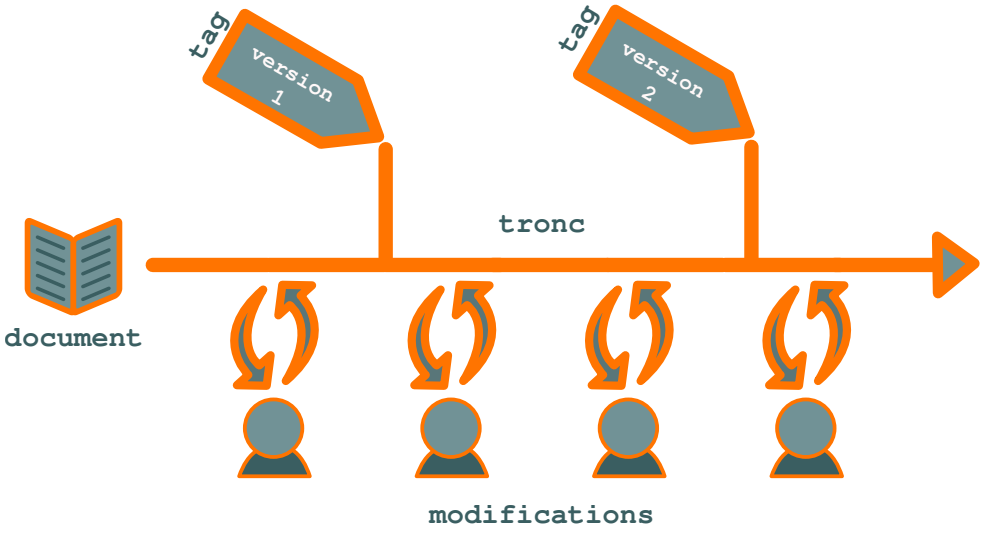


Fig. 2.9 – *Le système de gestion de versions conserve l'historique des modifications.*

Des interfaces graphiques permettent d'utiliser directement sous les gestionnaires de fichiers ces outils originellement conçus pour être utilisés en ligne de commande. Cependant, le paradigme sur lequel ils reposent est parfois difficile à appréhender pour les publics les moins technophiles [16].

Les systèmes de gestion des sources utilisent les concepts suivants :

Tronc Dépôt principal conservant toutes les versions des fichiers placées au cours du temps par le rédacteur technique (ou plus fréquemment, le développeur).

Branche Dépôt secondaire créé à partir de la version principale du code source.

Tag Instantané du tronc ou d'une branche à un instant *t*. Permet de figer facilement une version, par exemple, la version publiée, et de créer une archive.

Si l'on s'en donne la peine, il est également possible de mal utiliser les systèmes de gestion de version et de perdre des données. Mais, en pratique, à même niveau d'effort et de compétence, le risque de perdre de données est nettement moindre si le rédacteur technique manipule des fichiers texte sous un gestionnaire de version plutôt que des fichiers binaires sur un répertoire partagé.

Les systèmes de gestion de code source ont une fiabilité éprouvée et gèrent des millions de lignes de code. Tout comme les systèmes de fichiers (Ext4, Btrfs, etc.) ils évoluent lentement, selon une politique conservatrice, et ne sont proposés en production que lorsqu'ils ont été exhaustivement débogués. Si les plus grands projets de développement informatique, tel que GNU/Linux, par exemple, leur font confiance, pourquoi ne pas également leur confier la documentation technique ?

Un bémol cependant : ces outils ne sont pas destinés spécifiquement au format XML et effectuent des comparaisons ligne par ligne entre les fichiers, et non pas nœud par nœud, ce qui multiplie inutilement les conflits entre les *commits* ou les branches.

Voir aussi :

— *Git : du fichier au contenu* (page 11)

13. En cas d'incident réseau, l'utilisateur est averti que la transaction destinée à placer le fichier modifié sur le dépôt a échoué ; il peut alors procéder à une nouvelle transaction, sa copie locale du fichier étant intacte.
14. Surtout sous Git, conçu explicitement dans ce but.
15. Les systèmes de gestion de version favorisent un dépôt fréquent de modifications atomiques.
16. Même si Apple a contribué à en populariser certains aspects avec son application *Time machine*.

Utiliser les branches des systèmes de gestion de sources

Les systèmes de gestion de sources proposent de créer des branches d'un projet : si à un moment donné, un projet se divise en deux projets incompatibles, une branche est créée à partir du projet principal. Le rédacteur technique peut ainsi gérer les différentes traductions de la documentation technique.

Le système des branches peut servir en théorie à gérer :
- les différentes traductions d'une documentation technique,
- les différentes variations d'une même documentation technique.

En pratique, cependant, il vaut mieux gérer les déclinaisons d'une même documentation à l'aide des mécanismes de partage de sections et de filtrage de texte conditionnel des outils de documentation.

D'autre part, le système de gestion des branches est plus ou moins adapté à la gestion des traductions selon le gestionnaire de sources que l'on utilise.

La principale différence entre les systèmes de gestion de sources Git [17] et Subversion, c'est leur manière de gérer les branches. Créer une branche sous Subversion revient à dupliquer un répertoire. Les fichiers des deux répertoires évoluent ensuite séparément. Sous Git, en revanche, la création de branche se fait sans duplication de données. Sur un même répertoire local, une commande permet de changer de branche.

Créer une traduction d'une documentation consiste à *forker*, soit créer une branche, le document initial. Si l'on utilise Git se pose alors le choix entre :
- copier le répertoire de la langue source,
- créer une branche sur le répertoire de la langue source.

La solution de la branche permet en théorie d'effectuer du *Cherry picking* et d'appliquer facilement à toutes les langues cibles des modifications affectant uniquement le code XML du projet.

Par exemple, une modification de

```
<image href="filter.png" placement="break"/>
```

en

```
<image href="filter.png" placement="break" scalefit="yes"/>
```

de la version anglaise de la documentation peut facilement être appliquée aux versions chinoise, française, allemande ou autre si elle a fait l'objet d'un *commit* distinct. En pratique, cependant, cette opération peut s'avérer délicate et n'être réellement utile que si l'on doit gérer un grand nombre de différentes versions linguistiques. En tout cas, la solution des branches autorise de telles opérations, non celle des répertoires. Elle est cependant plus difficile à appréhender et à utiliser par l'équipe de rédaction technique.

Voir aussi :

— *Git : du fichier au contenu* (page 11)

Les CMS : le *workflow* en prime, mais une fiabilité à tester

Les CMS (Content Management System), ainsi dénommés pour des raisons purement marketing, mais dont la fonction se comprend mieux avec l'acronyme GED (système de gestion électronique de documents), apportent des notions de *workflow* et de gestion des liens qui s'avèrent précieuses lorsque l'on gère des documents modulaires.

S'ils utilisent des formats monolithiques tels que FrameMaker, les rédacteurs techniques peuvent utiliser des CMS tels que SharePoint, Alfresco ou consorts pour :

1. télécharger sur leur disque dur une copie locale des fichiers partagés,
2. effectuer leurs modifications sur la copie locale,
3. déposer la copie modifiée sur le dépôt central.

Cette solution est plus satisfaisante que le partage sur un simple serveur de fichiers, ne serait-ce que parce que la fréquence à laquelle les fichiers transitent sur le réseau est bien moindre [18]. Il est cependant toujours nécessaire de verrouiller les fichiers en cours de modification, ce dont se charge le CMS.

Originellement destinés aux documents monolithiques, de nombreux CMS prennent aujourd'hui en compte la modularisation des documentations techniques. Des solutions telles que DocZone ou Componize, cette dernière bâtie sur Alfresco,

17. http ://www.git-scm.com
18. À chaque dépôt du fichier sur le CMS, et non à chaque enregistrement de son travail par le rédacteur technique.

sont par exemple explicitement destinées à gérer des documentations modulaires basées sur l'architecture XML DITA XML.

Mais comment croire que ces solutions, qui sont fréquemment disponibles sous de nouvelles versions, marketing oblige, sont toutes d'une fiabilité optimale ?

J'aurais quelques scrupules, et quelques inquiétudes, sur le fait de leur confier entièrement la gestion et l'archivage des fichiers sources de la documentation. Une sélection rigoureuse de la solution s'impose, associée à une procédure de sauvegarde et de restauration éprouvée.

Base de données SQL

Dans le cas d'un CMS de type Drupal, Joomla ou WordPress, le référentiel est une base de données SQL. Il conserve un historique, mais uniquement article par article et ne permet pas la recherche et le remplacement de texte à travers tout le contenu. Cela peut justifier le choix de gérer le contenu sous-jacent du CMS sous forme de fichiers texte.

Un référentiel unique ?

Idéalement, tout le contenu peut être placé sous un référentiel unique, par exemple le logiciel de gestion de versions Git [19]. Ceci en maximise la réutilisation, la cohérence et la qualité. Si le contenu est au format DITA XML ou DocBook, par exemple, on peut exploiter au mieux les capacités de *single-sourcing* de ces formats pour le publier sous la forme appropriée.

Le contenu devient un réseau de modules d'information ; il faut alors gérer les relations au sein de ce réseau, notamment lors des mises à jour.

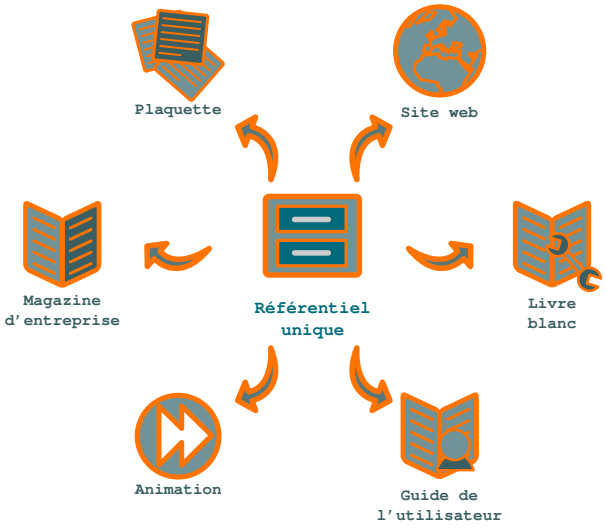


Fig. 2.10 – *Référentiel unique*

En pratique, il semble que rares sont les entreprises qui ont franchi ce pas. Il est vrai que tant que les formats structurés ne seront pas enseignés dans le secondaire, il paraît utopique de vouloir y convertir tous les acteurs de la société, surtout si le turn-over y est important.

Voir aussi :

— *Git : du fichier au contenu* (page 11)

Validation et contrôle qualité

Un support de rédaction technique doit être soumis à un contrôle qualité rigoureux avant d'être communiqué à ses différentes cibles.

19. http ://www.git-scm.com

Le contenu doit être validé avant livraison. Cela paraît évident, mais cela demande de bien impliquer en amont les personnes chargées de la validation. Idéalement, la phase de validation se déroule en parallèle de la phase de création : plus les modifications interviennent tôt, moins elles sont coûteuses. Un outil de gestion de contenu d'entreprise tel qu'Alfresco [20] peut sembler intéressant afin de mettre en place des *workflows*, sur le papier du moins. Dans les faits, une telle solution peut s'avérer lourde. Elle est de plus peu compatible avec certains formats sources basés sur des documents modulaires et non monolithiques et avec les logiciels de gestion de versions (le projet Componize [21] se propose cependant de gérer des projets DITA XML sous Alfresco). Il reste cependant impératif de mettre en place des étapes de validation tout au long du projet. Associés à un système de gestion de versions, les outils de comparaison sont très utiles pour valider les mises à jour. Une version « taguée » d'un projet DITA XML et la version en cours peuvent par exemple être exportées au format RTF, puis comparées sous un traitement de texte. Cela est bien moins fastidieux qu'une relecture comparée. Comparer les modules d'information directement sous le système de gestion de versions n'est pas suffisant, car ils ne sont que les « briques » du document final.

Workflow de création et validation

Un processus de création et de mise à jour de la documentation technique qui repose sur la mémoire des acteurs humains est peu fiable. Un rédacteur technique peut être fatigué, souffrant, en congés, oublier des données lorsqu'il est saturé d'informations ou avoir quitté la société. L'information entre deux personnes peut également mal circuler ou être mal comprise. C'est pour pallier ces faiblesses que l'homme a créé des outils. En revanche, il est créatif, à l'inverse des machines.

Face à cet état de fait, il convient de mettre un système de gestion de l'information relative à l'évolution de la documentation qui soit tolérant à l'erreur humaine. Il faut donc soit :

- mettre en œuvre des *workflows* sous un CMS,
- utiliser le système de gestion de tickets utilisés pour la gestion des nouvelles fonctionnalités du produit documenté (par exemple, Trac) :
  - création d'un ticket par un développeur,
  - mise en œuvre du ticket par un rédacteur technique,
  - fermeture du ticket par le créateur du ticket,
  - publication de la documentation lorsque tous les tickets critiques sont fermés.

Les fonctions principales d'un CMS sont les suivantes :

- gestion des métadonnées,
- *workflows*,
- traçabilité,

Quel qu'il soit, le système de suivi doit offrir une visibilité et une traçabilité totales des modifications apportées à la documentation technique (quoi, qui, quand).

Ce système doit être unique et exhaustif : il doit centraliser toutes les demandes de modification de la documentation technique.

Si le document est disponible en plusieurs langues, chaque ticket doit être dupliqué pour chaque langue ou, dans le cas d'un CMS, à chaque langue doit correspondre un *workflow* distinct.

Traduction

Les contraintes de traduction doivent être prises en compte en amont du processus rédactionnel. Elles ont des implications autant sur le style rédactionnel que sur l'organisation du référentiel.

Il n'y a pas de recette miracle : la livraison d'informations dans plusieurs langues demande un suivi constant. Mais la prise en compte des contraintes en amont et l'utilisation d'une méthodologie appropriée permettent d'améliorer la qualité et de diminuer les coûts et les délais de livraison des versions multilingues. La traduction doit être intégrée au *workflow* documentaire. Il faut également faire communiquer avec les traducteurs les différents acteurs : rédacteurs techniques, mais également ingénieurs, experts et concepteurs.

Si la documentation repose sur un ensemble de modules, la traduction peut se faire en parallèle de la rédaction, ce qui réduit les délais de livraison.

20. http ://www.alfresco.com/fr
21. http ://www.componize.com

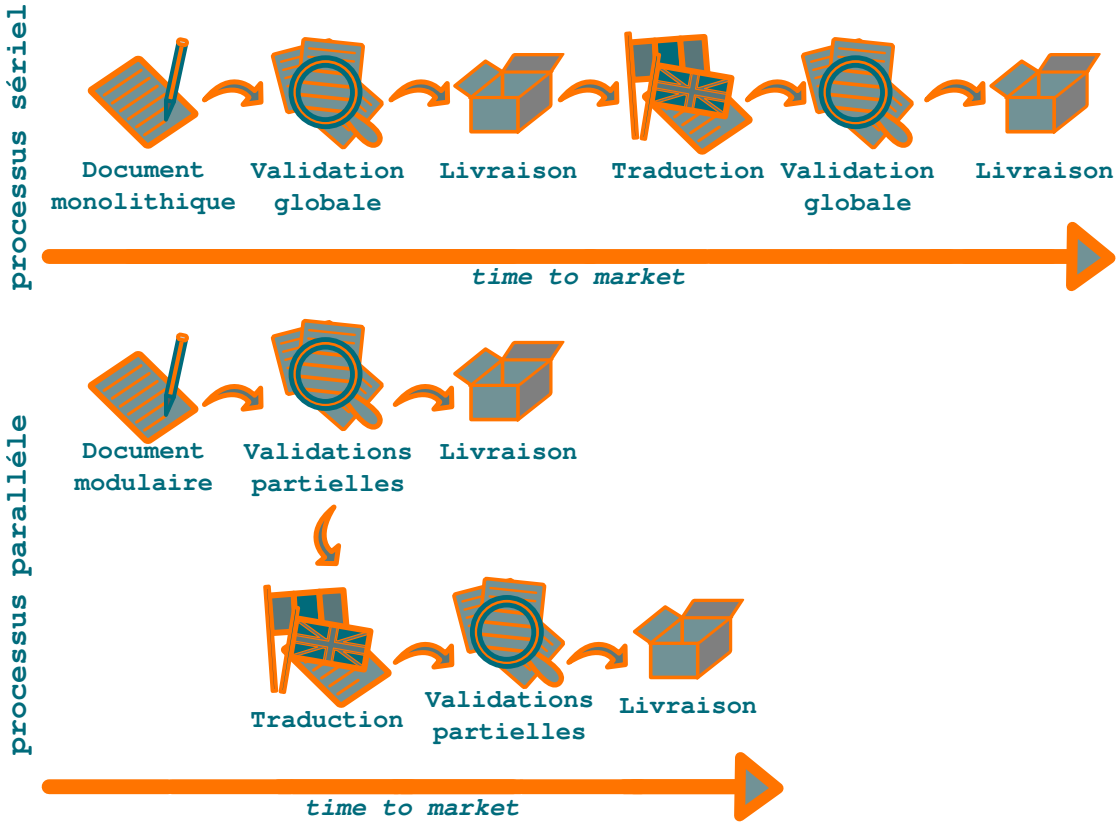


Fig. 2.11 – *Parallèlisation de la rédaction et de la traduction*

En ce qui concerne le référentiel des fichiers sources, vaut-il mieux placer les répertoires de langue en amont ou en aval des répertoires de projets documentaires ? Autrement dit, vaut-il mieux adopter la structure suivante :

- english
  - produit 1
  - produit 2
- francais
  - produit 1
  - produit 2

ou la suivante :

- produit 1
  - english
  - francais
- produit 2
  - english
  - francais

Dans la plupart des cas, il est préférable de placer la distinction entre les langues le plus en amont possible. Pour reprendre une terminologie utilisée dans le développement logiciel, créer une traduction d'un ensemble d'informations équivaut à créer une *branche* de cet ensemble. Comme il est plus facile de manipuler une branche par sa racine que par ses ramifications, à l'usage, il est beaucoup plus facile de manipuler des répertoires complets, ne serait-ce que pour les fournir aux traducteurs, qu'un ensemble de sous-répertoires.

Une fois la traduction réalisée, les modifications apportées à la version source ou à la version traduite ne peuvent être appliquées automatiquement à l'autre version. Pour continuer dans la terminologie du monde logiciel, la nouvelle branche est un *fork* : il devient impossible d'appliquer automatiquement à l'une les modifications apportées à l'autre. Pour fournir les mêmes informations dans les différentes langues, il est donc crucial de suivre efficacement les mises à jour de la version d'origine.

Format cible

Le format cible [22] d'un support de rédaction technique est celui sous lequel l'audience du message y accédera. Il est différent de celui sous lequel le rédacteur technique crée le contenu. Le *single-sourcing* permet de générer plusieurs livrables à des formats différents à partir d'un même format source.

À partir des fichiers sources validés, les livrables sont générés selon l'une des méthodes suivantes :

Totalement automatique Par exemple, livre blanc du format structuré DITA XML au format cible PDF *via* DITA-OT (DITA Open Toolkit).

Semi automatique Par exemple, contenu au format DITA XML exporté en HTML puis collé sous un CMS [24].

Manuelle Par exemple, plaquette marketing au format traitement de texte ou DITA XML mise en page sous Indesign, exportée en PDF, puis imprimée ; selon la fréquence de publication du document final, des filtres d'import XML peuvent également être mis en place.

Plus le processus est automatisé, plus le risque d'erreur est faible et plus la publication et la mise à jour sont aisées. L'automatisation facilite également le *single-sourcing*, qui consiste à générer plusieurs livrables à des formats cibles différents à partir d'un même format source. Un projet au format DITA XML peut ainsi être livré sous forme de fichier PDF, d'aide compilée Windows, d'aide JavaHelp, de site en HTML, etc. Le XML offre en ce domaine des possibilités quasi illimitées.

Livraison

Le rédacteur technique livre le document à son destinataire de la manière appropriée :

— animation publiée sur un site de streaming,
— plaquette distribuée dans les salons ou laissée en clientèle par les ingénieurs commerciaux,
— journaux envoyés aux clients,
— site Internet mis à jour,
— document mis en ligne en PDF ou distribué sous forme de guide imprimé…

22. Dans le cas d'une photo, le format cible est le format JPEG qui est utilisé pour l'affichage Web ou l'impression et sur lequel les modifications ne peuvent être annulées une fois fermé le logiciel de retouches.
24. Ceci est automatisable par un script ; le CMS Drupal propose également un module DITA integration for Drupal.

3. Format structuré DITA XML

Diminuer les coûts de production et de traduction, réduire les délais de mise sur le marché (ou *time to market*) et améliorer la qualité de la documentation. Voilà les défis que doit relever aujourd'hui le rédacteur technique. L'un des meilleurs moyens d'y parvenir consiste à réduire le volume source de la documentation et à mieux gérer le contenu d'entreprise.

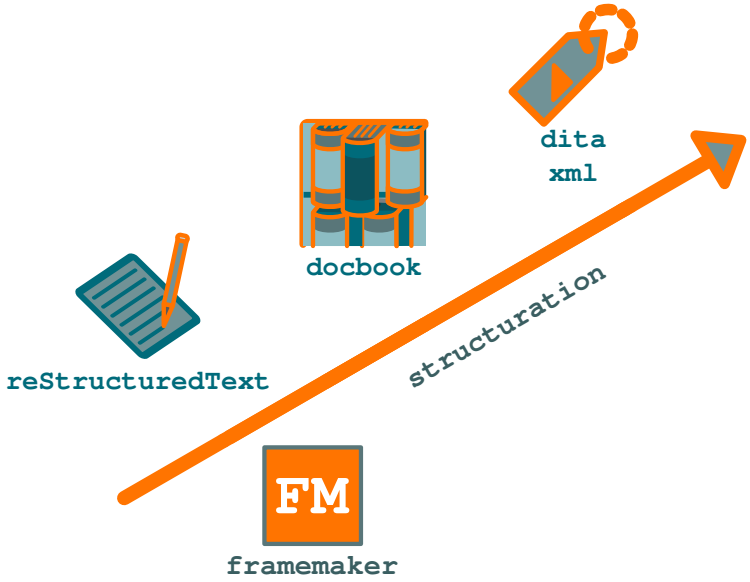


Fig. 3.1 – *Formats de documentation technique : degrés de structuration*

L'information que le rédacteur technique fournit au public *doit* être redondante : l'entreprise doit présenter à chacun de ses publics toute l'information dont il a besoin, sur le support qui lui est adapté. Pour prendre l'exemple le plus simple, chaque support d'information doit mentionner les coordonnées de la société. Mais jusqu'à 50 % de l'information disséminée par l'entreprise est répétée.

En revanche, la redondance de l'information en interne engendre des coûts supplémentaires, allonge les cycles de production et entraîne une baisse de l'homogénéité, et donc de la qualité, du contenu. Il est donc primordial de diminuer et de mieux partager le contenu source et d'en diminuer le volume. Le rédacteur technique doit diviser l'information en briques autonomes uniques, standardisées et génériques, pour pouvoir l'assembler à la demande. Il doit donc utiliser des modules structurés de manière homogène qui peuvent être facilement manipulés par des applications.

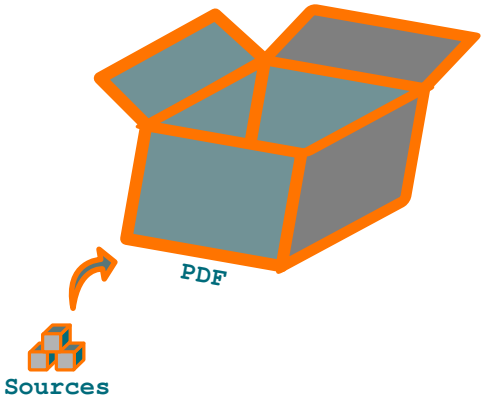


Fig. 3.2 – *Les sources de la documentation doivent être moins volumineuses que les livrables.*

DITA XML est une architecture XML de rédaction structurée destinée à la création de documents modulaires et à la

réutilisation du contenu. À partir d'une base commune de modules d'information atomiques DITA XML, le rédacteur technique peut fournir en temps réel toute l'information dont chaque utilisateur a besoin, sur tout type de média, du support d'e-learning au document PDF ou papier, en passant par le site Web.

DITA XML applique le principe de non-redondance des informations propre aux bases de données normalisées. Cette architecture transpose à la documentation technique la révolution introduite dans l'industrie par la standardisation : si l'on peut construire des automobiles de modèles différents à partir d'un ensemble de pièces identiques, de même, le rédacteur technique peut publier des documents différents à partir d'un ensemble de briques d'information standardisées.

Cas concrets d'utilisation de DITA XML

L'utilisation quotidienne du format de rédaction structurée DITA XML sur des projets multilingues en tant que rédacteur technique m'a amené à développer certaines solutions et astuces que je vous livre ici. Tout retour d'expérience est le bienvenu !

Formats structurés et non structurés

Les formats structurés favorisent la création de documents minimalistes, complets et cohérents. Ils permettent au rédacteur technique de se concentrer sur le contenu et d'améliorer l'expérience utilisateur et l'utilisabilité de la documentation technique.

Les informations contenues dans un document technique peuvent être catégorisées selon leur sens. Par défaut, DITA XML propose trois types de base :

concept Introduction ou présentation d'un concept.

task Procédure pas à pas, séquentielle et numérotée, destinée à réaliser une tâche.

reference Informations de référence sur une liste d'éléments tels que des options d'un programme.

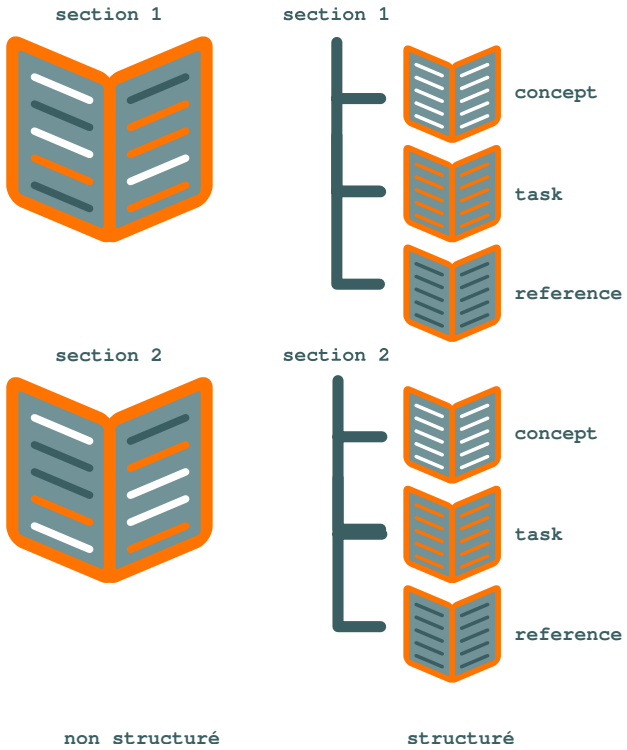


Fig. 3.3 – *Formats structurés et non structurés*

Sous un format non structuré tel que le format traditionnel de FrameMaker [26], rien ne contraint le rédacteur technique à organiser l'information selon son sens. Si des règles de rédaction rigoureuses ne sont pas scrupuleusement suivies, l'information fournie à l'utilisateur risque d'être peu claire et difficile à parcourir rapidement.

26. http ://en.wikipedia.org/wiki/Adobe_FrameMaker

Avec des formats structurés tels que DITA XML, en revanche :

— le rédacteur technique se concentre sur le contenu,
— l'information est présentée à l'utilisateur selon une organisation cohérente et prévisible,
— l'accès à l'information est séquentiel et rapide,
— l'information peut facilement être réorganisée selon les besoins,
— l'utilisabilité du support d'information fourni est optimale.

Les types d'information de haut niveau tels que *task* sont divisés en types de plus bas niveau, par exemple :

prereq Liste de points obligatoires préalables à la réalisation d'une tâche.

steps Série d'étapes de la procédure.

stepxmp Exemple de réalisation d'une étape.

Les règles syntaxiques interdisent au rédacteur technique de faire figurer une procédure pas à pas dans une section d'un autre type que *task*. Le rédacteur technique dispose donc d'un véritable modèle de rédaction qui l'aide à présenter des informations :

Minimalistes Selon le principe de design *less is more*, l'utilisateur ne dispose *que* de l'information dont il a besoin : une section *task*, par exemple, ne contient que des prérequis, une procédure et quelques autres éléments spécifiques ; toutes les informations conceptuelles ou de référence sont placées dans des sections à part.

Complètes L'utilisateur dispose de *toute* l'information dont il a besoin ; une section de type *task* sans procédure n'est pas une section DITA XML valide et ne pourra pas être publiée ; il est même possible de mettre en œuvre un mécanisme vérifiant automatiquement avant publication la présence de blocs d'information facultatifs selon le schéma XSD [27] DITA XML, mais que le rédacteur technique juge obligatoires, tels que le résultat d'une procédure.

Cohérentes Les informations de même type sont présentées dans le même ordre et avec la même mise en page ; les blocs d'information identiques répétés à différents endroits, tels qu'une remarque, sont issus d'une seule et même source et sont donc strictement identiques.

DocBook ou DITA XML ?

Certaines entreprises ont parfois un contenu existant au format DocBook [28]. Géré souvent par les acteurs les plus techniques de la société, il coexiste la plupart du temps avec d'autres contenus au format FrameMaker ou traitement de texte. S'il est décidé de fédérer tout le contenu d'entreprise sous un seul format, il semble naturel de capitaliser les efforts fournis sur la chaîne de création et de publication DocBook et de sélectionner ce format. C'est pourtant se priver des gains de productivité spectaculaires offerts par DITA XML.

Il est facile de générer du DocBook à partir de DITA XML. DITA-OT propose par défaut ce format cible, au même titre que le PDF ou le HTML. L'opération inverse ne peut pas être totalement automatisée. Pourquoi ?

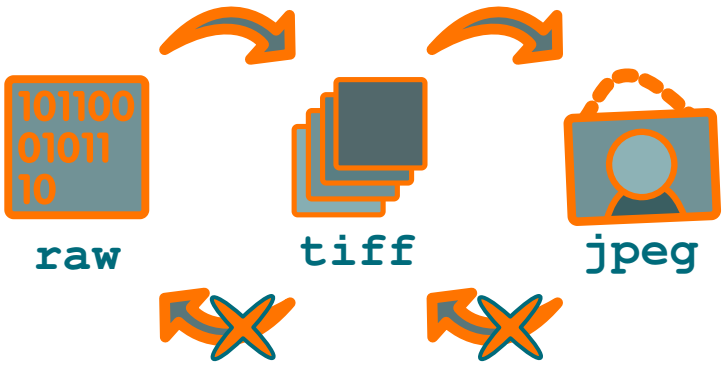


Fig. 3.4 – *Un processus non réversible*

Il n'est pas possible de migrer automatiquement des données de formats pauvres vers des format riches en information.

Tout simplement parce que le contenu au format DITA XML contient plus d'informations. Passer d'un format plus riche à un format plus pauvre en information est une opération entropique qui peut facilement être automatisée. Par exemple, générer un PDF à partir de DITA XML. Effectuer l'opération inverse exige d'injecter de l'intelligence, opération que seul l'être humain peut aujourd'hui effectuer.

27. http ://fr.wikipedia.org/wiki/XML_Schema
28. https ://github.com/olivier-carrere/redaction-technique.org/tree/DocBook

Si votre contenu était une photo, nous pourrions faire l'analogie suivante :

Format de contenu	Format de photo
DITA XML	RAW[29] [32]
DocBook	TIFF[30]
PDF	JPEG[31]

Le passage de RAW en TIFF et de TIFF en JPEG est destructif et ne peut se faire en sens inverse [33].

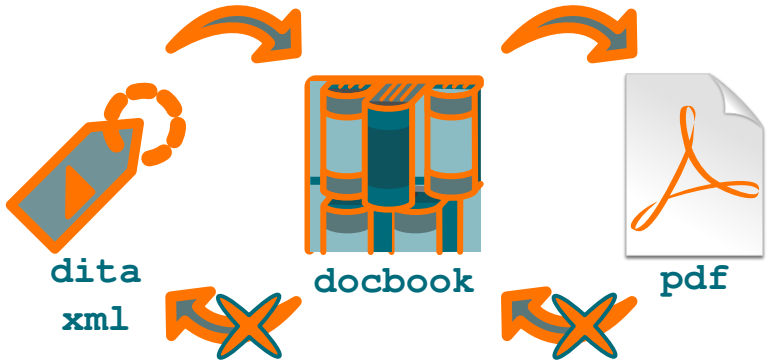


Fig. 3.5 – *Un processus non réversible*

Le PDF est sémantiquement plus pauvre que DocBook, lui-même plus pauvre que DITA XML [34].

Si votre entreprise tient absolument à utiliser du DocBook, il est toujours loisible de générer le contenu DocBook à partir d'un contenu source au format DITA XML. À condition que le contenu source reste au format DITA XML (c'est à dire, à condition qu'aucune modification apportée au contenu DocBook ne soit sauvegardée) et que le format DocBook ne soit qu'une étape de la génération des livrables, au même titre que le format FO, vous bénéficiez ainsi des fonctionnalités avancées de réutilisation du contenu que propose DITA XML.

L'effort de migration d'un format non structuré est certes un peu plus important vers DITA XML que vers DocBook, puisque vous devez injecter plus d'informations sémantiques. Vous devez également migrer le contenu DocBook vers DITA XML, ce qui représente également un effort, quoique plus faible. Mais votre contenu est immédiatement de meilleure qualité, car plus structuré. Et vous pourrez rapidement cueillir tous les fruits de votre labeur, notamment si une traduction de votre contenu dans une nouvelle langue est envisagée.

De manière générale, un professionnel a toujours intérêt à travailler sur le format le plus riche, ne serait-ce que pour être pro-actif et anticiper sur les nouveaux besoins.

Migration de FrameMaker vers DITA XML

Migrer de FrameMaker vers DITA XML, ce n'est pas comme enregistrer un document MS Word au format LibreOffice [35]. Aucun processus automatique ne permet de migrer un document non structuré vers un format structuré. Dans le pire des cas, selon la qualité de votre document de départ, cela peut s'apparenter à transformer une friche en jardin à la française. Mais une migration bien planifiée permet de passer au nouveau format sans perturber le rythme des livraisons.

Pour filer la métaphore, si l'on se fixe pour but de convertir un marécage en parterre du château de Versailles, il convient de passer par l'étape du jardin à l'anglaise - soit un endroit certes non rigoureusement architecturé, mais très agréable à vivre. Bonne nouvelle : si le rédacteur technique a utilisé de manière cohérente un jeu de styles limité et organisé rationnellement son contenu FrameMaker, il est déjà certainement très proche de ce stade.

29. http ://fr.wikipedia.org/wiki/RAW_(format_d%27image)
32. Ce n'est bien sûr qu'une analogie, DITA XML étant un standard, à la différence du format RAW.
30. http ://fr.wikipedia.org/wiki/Tagged_Image_File_Format
31. http ://fr.wikipedia.org/wiki/Jpeg
33. Pour être aussi exact que possible, vous pouvez enregistrer une image JPEG au format TIFF ; mais cette image aura une qualité égale à celle de l'image JPEG, inférieure à la qualité habituelle des images TIFF. En revanche, on ne peut à ma connaissance pas enregistrer une image TIFF sous un format RAW.
34. Le PDF est cependant plus riche en informations de mise en page, appliquées automatiquement à partir d'une feuille de style.
35. LibreOffice propose une fonction d'enregistrement au format DocBook, mais très imparfaite ; le XML qu'elle produit peut servir de base à la création d'une version DocBook, avec beaucoup d'efforts… Sauf à maintenir deux versions du même contenu, le processus de migration de LibreOffice vers DocBook exige donc un arrêt temporaire des livraisons des nouvelles versions de la documentation ; il doit donc être soigneusement planifié.

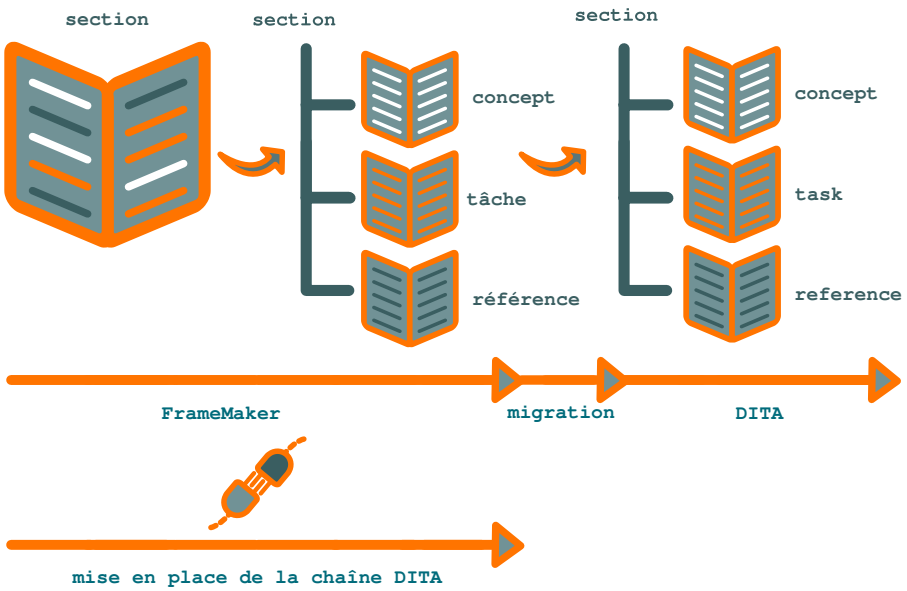


Fig. 3.6 – *Migration de FrameMaker vers DITA XML*

D'ailleurs, si, pour une raison quelconque, votre projet de migration devait s'arrêter là, les rédacteurs techniques, l'entreprise et les utilisateurs y auraient déjà beaucoup gagné, respectivement en :

— facilité de mise à jour,
— cohérence et rapidité de publication des nouvelles versions,
— facilité d'accès à l'information.

Restructuration du contenu FrameMaker

La partie automatisée d'une migration de FrameMaker[36] vers DITA XML consiste à appliquer une table de conversion entre les styles FrameMaker et les structures DITA XML.

Un important travail de restructuration du document FrameMaker doit cependant être effectué en amont :

— restructuration de l'information selon les trois catégories *concept*, *tâche* et *référence*,
— suppression des *overrides* (propriétés de texte appliquées manuellement et écrasant les styles ; ce genre d'hérésie est, sinon impossible, du moins très limité sous un format structuré),
— harmonisation et simplification des styles FrameMaker pour les limiter et les faire correspondre aux balises DITA XML qui seront utilisées (par exemple, un style *note_important* vers la balise <note type="”important>> ; il faut donc au préalable analyser le contenu existant et décider quel ensemble de balises sera utilisé parmi les centaines de balises proposées par DITA XML : il est en effet fortement déconseillé de les utiliser toutes).

36. http ://en.wikipedia.org/wiki/Adobe_FrameMaker

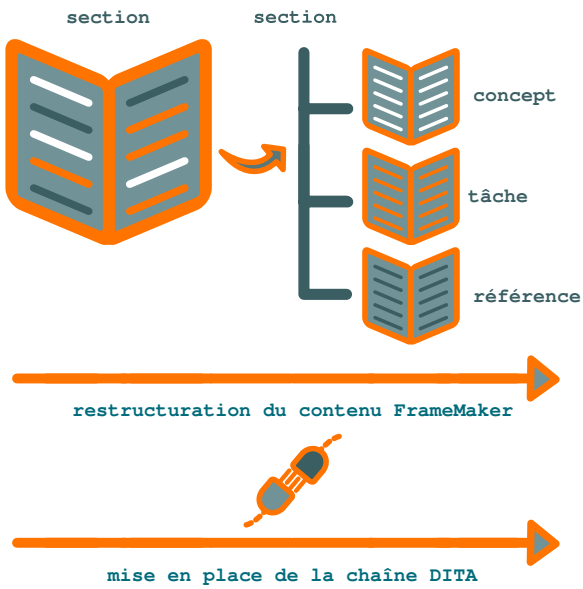


Fig. 3.7 – *Restructuration du contenu FrameMaker et mise en place de la chaîne DITA XML*

Ce travail d'harmonisation peut se faire en parallèle avec la mise à jour et la publication du document FrameMaker. La qualité de ce document n'en sera que meilleure. En même temps que cette réorganisation du contenu, vous pouvez mettre en place la chaîne complète de création, gestion et publication DITA XML sur un échantillon de votre contenu :

— mise en place des outils,
— réalisation des feuilles de style des différents formats de sortie,
— formation des rédacteurs techniques, graphistes et traducteurs,
— formation et sensibilisation des autres acteurs de l'entreprise.

Ce n'est qu'une fois que sa chaîne est fiable et acceptée, voire attendue par les autres acteurs de l'entreprise, que le rédacteur technique peut envisager la migration.

Si vos documents sont disponibles en plusieurs langues, vous devez modifier les fichiers FrameMaker et effectuer la migration pour chaque langue. Si un projet de traduction dans une nouvelle langue se profile, mieux vaut effectuer la migration avant !

Table de conversion FrameMaker vers DITA XML

Lorsque les fichiers FrameMaker [37] sont prêts pour la migration et que la chaîne DITA XML est parfaitement intégrée aux processus techniques et humains de la société, le rédacteur technique peut appliquer la table de conversion [38].

Vous devriez maintenant être à même d'archiver les fichiers FrameMaker, puis de basculer totalement vers le format DITA XML.

37. http ://en.wikipedia.org/wiki/Adobe_FrameMaker
38. Bien que ce processus doive être rapide, je vous conseille de le faire juste après une livraison d'une nouvelle version du document pour avoir la marge de temps suffisante avant la livraison suivante, des petits ajustements étant toujours nécessaires.

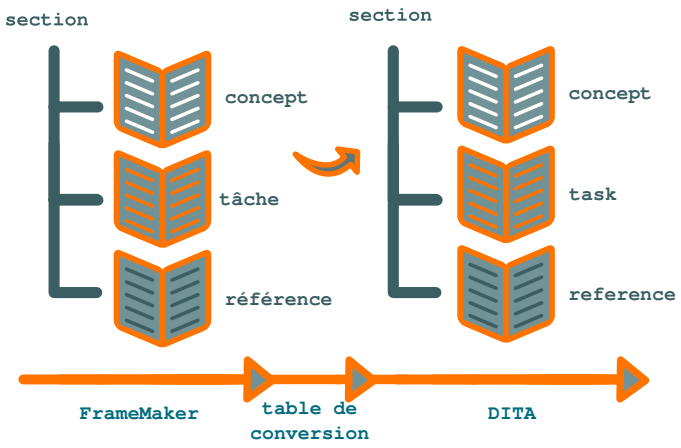


Fig. 3.8 – *Application d'une table de conversion de FrameMaker vers DITA XML*

Appliquez bien sûr ce processus à un petit jeu de documents [39], qui ne soit pas, si possible, d'une importance critique. Après ce premier succès, vous pourrez appliquer le processus aux autres jeux de documents.

Vous pouvez maintenant progressivement modulariser et partager votre contenu dans le nouveau format afin de tirer parti au maximum de DITA XML. Vous pouvez pendant cette phase continuer à publier de nouvelles versions du document ; la publication devrait d'ailleurs être beaucoup plus simple que sous FrameMaker.

Migrer de FrameMaker vers DITA XML

Le but de cette procédure est de :

— migrer son contenu FrameMaker vers DITA XML sans se plonger dans les arcanes des *EDD* FrameMaker (petits projets uniquement !),

— gérer la documentation technique au format DITA XML sans utiliser FrameMaker **structuré**.

1. Restructurez le contenu et les styles de vos fichiers de contenu FrameMaker selon les concepts DITA XML.
2. Créez un document FrameMaker vide et importez-y tous les styles existants dans les fichiers à migrer.
3. Appliquez tous les styles disponibles à des paragraphes vides du document FrameMaker vide.
4. Enregistrez le document FrameMaker vide sous le nom `styles.fm`.
5. Ouvrez FrameMaker **structuré 11** et créez un nouveau fichier DITA XML de type *topic*.
6. Choisissez **StructureTools > Exporter le catalogue d'éléments en tant qu'EDD** et sauvegardez la nouvelle EDD sous le nom `DITA-topic-edd.fm`.
7. Ouvrez le fichier `styles.fm`, puis choisissez **Fichier > Importer les définitions d'éléments** et importez les définitions d'éléments à partir de `DITA-topic-edd.fm`.
8. Répétez les trois étapes ci-dessus pour les autres types de topics DITA XML (*task*, *reference*, etc.), en modifiant les noms de fichiers comme il se doit.
9. Ouvrez le fichier `styles.fm`, puis choisissez **StructureTools > Générer le tableau de conversion**.
10. Modifiez le fichier de conversion et faites correspondre chaque style FrameMaker à une balise DITA XML.
11. Enregistrez le tableau de conversion sous le nom `DITA2FM-conversion-table.fm`.
12. Ouvrez un fichier de contenu FrameMaker sous FrameMaker structuré 11 et choisissez **StructureTools > Utilitaires > Structurer le document en cours**.
13. Sélectionnez `DITA2FM-conversion-table.fm` et cliquez sur **Ajouter structure**.
14. Enregistrez le fichier de contenu FrameMaker au format XML sans sélectionner d'application.
15. Ouvrez le fichier XML généré sous un éditeur DITA XML et corrigez la syntaxe DITA XML. Certains aspects de cette étape sont scriptables, mais il faut également procéder à des opérations manuelles de restructuration du contenu. Il vous faudra notamment placer à la main les références croisées, de préférence dans une *reltable*.

39. J'appelle *jeu de documents* tout ensemble d'informations liées qui ne partage aucun contenu avec un autre ensemble ; si par exemple le document *A* partage une section avec le document *B*, le jeu de documents est *A*+*B* ; si vous dupliquez la section partagée afin qu'elle ne soit plus commune à *A* et *B*, *A* et *B* deviennent des jeux distincts.

Pour générer les éléments permettant de construire un fichier *ditamap*, vous pouvez par exemple utiliser des scripts Perl du type :

> **Attention :** Ne lancez ce type de scripts que sur une copie de vos fichiers et non sur les fichiers originaux.

```
#!/usr/bin/perl
open(INPUT,"<$ARGV[0]") or die;
@input_array=<INPUT>;
close(INPUT);
$input_scalar=join("",@input_array);
# substitution
$input_scalar =~ s#\<body>(.|\n)*?</body>##ig;
open(OUTPUT,>$ARGV[0]") or die;
print(OUTPUT $input_scalar);
close(OUTPUT);
```

Vous pouvez également modulariser facilement le contenu à l'aide des ciseaux XML xml_split [40], ou utiliser le module Perl XML::Twig [41], ou encore ce *one-liner* Bash pour renommer les fichiers `.dita` d'après leur titre :

```
$ ack "<title>" *.dita| sed "s# #_#g;" |
tr '[:upper:]' '[:lower:]' |
sed -E "s#(.*.dita)#mv \1#g;" |
sed -E "s#\.dita.*<title>(.*)</title>#.dita \1.dita#g;"
```

Une architecture documentaire trop complexe ?

DITA XML permet des gains de productivité importants par la réduction du volume source que le rédacteur technique crée, traduit et maintient. Ce gain de productivité se fait au prix d'une plus grande complexité.

Si les projets DITA XML sont plus *complexes*, ils sont cependant moins *compliqués* que des projets reposant sur des formats plus traditionnels de type FrameMaker. En effet, DITA XML est une architecture rationnelle. Le rédacteur technique se trouve donc face à un comportement prédictible des outils qu'il utilise, loin des *trucs et astuces* destinés à contourner les bugs ou les fonctionnements erratiques des outils plus lourds.

Le tableau suivant présente les différents niveaux de complexité induits par DITA XML et les solutions qui permettent au rédacteur technique de les maîtriser plus facilement :

Complexité	Solution
Syntaxe DITA XML	IDE (Integrated Development Environment) tel que XMetal ou nXML
Gestion des relations entre des briques d'information atomiques	CMS dédié tel que Componize ou DocZone
Syntaxe de la feuille de style XSLT	Logiciel graphique de création de feuilles de style

Pour une petite équipe de rédaction technique, l'écueil principal sera la nécessité de mettre en œuvre la charte graphique de l'entreprise. Les autres aspects peuvent être gérés sans outil spécialisé, avec une bonne communication et une série de bonnes pratiques.

Du document à la base documentaire modulaire

Le modèle du livre est encore prédominant pour créer et gérer l'information. Mais le contenu d'entreprise est souvent disséminé dans de nombreux documents, sous des formats hétérogènes. Ceci se traduit par des doublons, des incohérences, un coût de mise à jour et de traduction élevé, et des retards de livraison. Le rédacteur technique dispose cependant d'autres modèles, plus efficaces.

Le format de rédaction structurée DITA XML propose de passer du modèle du livre à celui du de la base documentaire modulaire. Le contenu d'entreprise repose sur des briques uniques, qui peuvent être assemblées dynamiquement, à la demande, pour produire des documents sous différents formats cibles.

40. http ://search.cpan.org/dist/XML-Twig/tools/xml_split/xml_split
41. http ://www.xmltwig.org/xmltwig/

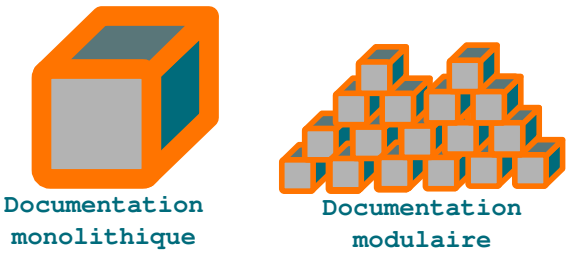


Fig. 3.9 – *Une documentation modulaire offre une souplesse inégalée.*

Le volume de contenu source est minimisé, ce qui diminue les coûts de création, mise à jour et traduction du contenu d'entreprise. De plus, le rédacteur technique peut gérer les processus de rédaction, validation et traduction module par module. Les *workflows* peuvent ainsi être parallélisés, ce qui réduit les délais de mise sur le marché.

Les fichiers DITA XML peuvent en outre être aisément centralisés sous un référentiel unique, tel qu'un ECM (système de gestion de contenu) ou un VCS (logiciel de gestion de versions). Le capital immatériel de la société est ainsi préservé.

Un langage à balises

DITA XML est un langage à balises : le rédacteur technique structure l'information dans des fichiers sources sans mise en page, similaires aux fichiers sources de code informatique. L'utilisateur reçoit un document cible, par exemple un fichier PDF, où les balises sont remplacées par une mise en forme typographique.

Si votre entreprise fournit à ses clients une documentation technique au format MS Word, le rédacteur technique et l'utilisateur disposent des mêmes supports d'information (il n'y a pas de différenciation entre le fichier source et le fichier cible). Ce qui semble a priori la solution la plus simple s'avère cependant peu efficace en termes de productivité de l'équipe de rédaction technique et de structuration de l'information.

Avec un format texte tel que DITA XML, le rédacteur technique et le lecteur disposent de supports largement différents :

Rédacteur technique Le rédacteur technique manipule des fichiers sources ; il utilise les balises pour construire le document en marquant les éléments d'information qu'il crée ou réutilise. Les balises sont imbriquées comme des poupées russes organisées selon une syntaxe rigoureuse. Le fichier source n'est pas au format WYSIWYG : la mise en page sera appliquée lors de la transformation des fichiers sources en fichiers cibles (autrement dit, lors de la génération des livrables). Tout au plus, certains logiciels graphiques tels XMetal, Oxygen ou FrameMaker structuré proposent-ils le format WYSIWYM (what you see is what you mean), où les balises sont remplacées à l'écran par une mise en forme générique, différente de l'aspect final du document. Je trouve cependant que l'un des intérêts d'avoir recours à un langage à balises est de voir exactement ce que l'on fait en manipulant soi-même les balises sans en déléguer l'interprétation à un logiciel graphique.

Utilisateur Seul le contenu est présenté au lecteur dans le fichier cible ; le texte marqué par des balises dans les fichiers sources a une mise en valeur typographique dont le sens est explicité dans la section *Conventions typographiques* du document final.

Un fichier source DITA XML mélange du texte et des balises, délimitées par les signes < et >. Le texte proprement dit est encapsulé dans un jeu de balises ouvrantes de type <balise> et de balises fermantes de type </balise> selon le schéma <balise>texte</balise>. Tout texte entré hors d'une balise ouvrante et fermante est incorrect et produit un fichier non valide.

Typologie de haut niveau de l'information

DITA XML propose au rédacteur technique une typologie de haut niveau qui est une véritable aide à la structuration du contenu.

S'il crée un nouveau document au format FrameMaker, DocBook ou traitement de texte, le rédacteur technique se trouve face à une page blanche. Selon sa rigueur professionnelle, l'information transmise à l'utilisateur oscillera entre les deux pôles suivants :

Organisation rationnelle L'utilisateur dispose d'un accès séquentiel rapide et aisé à l'information dont il a besoin.

Magma informatif L'utilisateur doit lire intégralement toute une section, voire le document en sa totalité pour espérer trouver des renseignements utiles.

Lorsqu'il crée un document DITA XML, en revanche, le rédacteur technique doit d'emblée choisir le modèle[42] qui correspond au type d'information qu'il veut présenter. De base, DITA XML propose les types d'information suivants[43] :

concept Texte généraliste du type introduction ou présentation.

task Procédure pas à pas destinée à réaliser une tâche.

reference Information de référence du type explication de paramètres de commandes.

Chacune de ces catégories de haut niveau propose un jeu de balises de plus bas niveau qui lui est propre. Si le rédacteur technique rédige un document technique, il y a toutes les chances pour que l'information qu'il a collectée et qu'il doit organiser fasse partie de l'une de ces trois catégories[44]. Cette division en types d'information oblige donc d'entrée de jeu le rédacteur technique à structurer l'information. L'utilisateur y gagne en facilité et rapidité d'accès à l'information et en utilisabilité globale de la documentation technique.

Organisation à la demande du contenu

Les briques d'information peuvent être assemblées à la demande dans des structures de table des matières externes, les *ditamap*.

L'organisation de l'information sous DITA XML n'est pas figée. Les briques peuvent être organisées dans différentes structures hiérarchiques, selon l'évolution des besoins. Si le rédacteur technique a pris soin de construire des briques d'information atomiques et génériques, il peut, à l'instar d'un constructeur automobile proposant sans cesse de nouveaux modèles par assemblage d'éléments standardisés, proposer par exemple les documents suivants :

Guide de l'utilisateur Thèmes systématiquement organisés en concept et procédures pas à pas.

Document de présentation Concepts.

Quikstart Procédures pas à pas.

Manuel de référence Informations de référence.

Pour ce faire, le rédacteur technique prendra soin de placer les éléments liés à un contexte particulier dans les structures *ditamap* et non dans les fichiers de contenu DITA XML. En particulier, les références croisées doivent être indiquées dans une *reltable* placée dans la *ditamap* : si le document *A* doit renvoyer au document *B* dans la *ditamap 1*, il doit pouvoir être également utilisé sans modification dans la *ditamap 2*, où le document *B* n'est pas inclus.

L'organisation des répertoires de travail doit également permettre l'utilisation de liens relatifs, notamment vers les images, qui ne seront jamais cassés.

Le single-sourcing : un format source, plusieurs formats cibles

Le *single-sourcing* est un sujet qui a longtemps divisé les rédacteurs techniques : des supports de rédaction technique différents, tels qu'une aide en ligne et un manuel imprimé, doivent-ils proposer un contenu radicalement différent ou peuvent-ils être générés à partir du même contenu source ?

Les contraintes de productivité et la réduction des coûts aidant, le débat a été tranché en faveur du *single-sourcing*. Le gain qualitatif, discutable, ne compense pas le coût de créer, maintenir et traduire une version source différente pour chaque version cible.

42. Dans la pratique, un schéma XSD.
43. DITA XML propose trois types d'information de base, tandis que la méthode Information Mapping en propose sept.
44. S'il s'avère qu'il a réellement besoin d'une autre catégorie, il peut la créer *via* une spécialisation.

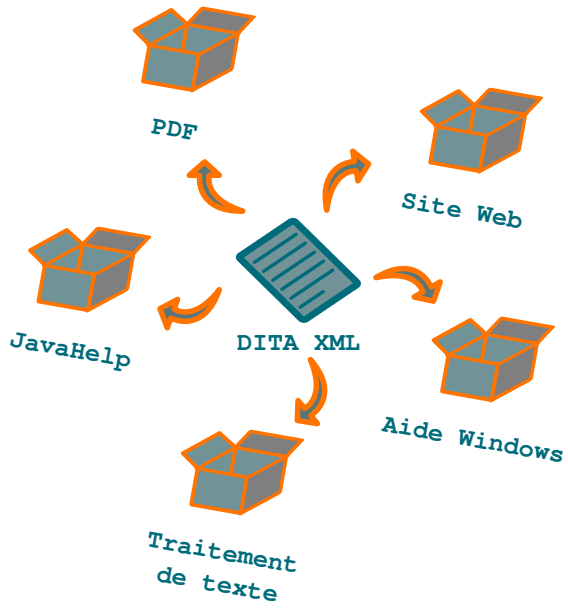


Fig. 3.10 – *Un seul jeu d'informations, une multiplicité de formats de sortie*

Si le rédacteur technique pratique le *single-sourcing*, il doit cependant sélectionner en début de projet le paradigme sur lequel il se base : le livre ou l'aide en ligne. Pendant longtemps, les outils proposés reposaient soit sur un document de type livre (MS Word, ou FrameMaker, essentiellement) qui pouvait être exporté au format d'aide en ligne, soit sur un fichier source (RTF) d'aide Windows, pour générer un PDF. Une forte perte d'information de navigation (index, références croisées, liens, etc.) intervenait souvent lors de l'exportation.

DITA XML propose un modèle agnostique quant au format cible. Les fichiers sources, bien que basés sur un modèle modulaire proche de celui de l'aide en ligne, peuvent facilement être exportés sous forme de fichier PDF, d'aide en ligne, de pages HTML liées ou autre, sans aucune perte d'information.

Les topics, modules d'information de base DITA XML

Les topics [45] sont les plus petites unités d'information autonomes gérées par DITA XML. Chaque *topic* a un titre et un corps de texte. Il ne traite que d'un seul sujet. Il appartient donc au rédacteur technique de se baser sur la modularité proposée par DITA XML pour bien structurer l'information.

Les *topics* sont sémantiquement typés. Il existe idéalement un type de *topic* par type d'information. DITA XML propose par défaut des *topics* adaptés à la documentation des logiciels (description de concepts et de tâches, liste de commandes, etc.), mais de nouveaux types de *topics* peuvent être créés pour répondre à d'autres besoins.

Les *topics* sont une des différences principales entre DITA XML et DocBook, qui ne propose pas de typologie des briques d'information.

Les *topics* sont généralement stockés *à plat* dans des répertoires divisés par type de *topic*. Ils sont organisés hiérarchiquement dans des fichiers *ditamap* et peuvent être partagés entre différents documents. Les titres des modules ne sont pas affectés d'un niveau de titre. La structure des modules étant parfaitement homogène, un module peut avoir un niveau 3 dans un document donné, et un niveau 1 dans un autre document, sans qu'il y ait besoin de modifier en quoi que ce soit les *topics*.

Les unités d'information atomiques [46] telles que des remarques, des paragraphes, voire des phrases ou des segments de phrase, qui ne peuvent pas être munis d'un titre, ne forment pas des *topics*. Elles peuvent être cependant partagées *via* le mécanisme *conref*, similaire au mécanisme *Xinclude* proposé par DocBook.

Gérer son contenu DITA XML avec ou sans CMS ?

L'architecture DITA XML ne propose pas de mécanisme de *workflow* documentaire natif. Les *workflows* sont pourtant un élément important d'un processus efficace de gestion du cycle de vie du contenu.

45. http ://docs.oasis-open.org/dita/v1.0/archspec/topicover.html
46. Pas au sens XPath.

Les CMS gèrent également les métadonnées, ce qui permet une recherche plus efficace de l'information existante, et les rétroliens [50].

La plupart des entreprises sont réticentes à mettre en place des CMS, outils dédiés aux *workflows*. Elles ont d'ailleurs parfois connu des échecs de mise en place de telles solutions part le passé.

De plus, l'un des grands avantages de DITA XML, c'est de s'intégrer directement dans le système d'information en place. Chez les éditeurs de logiciels, notamment, rien de plus facile que de venir se greffer sur le système de gestion des sources en place, qu'il s'agisse de Git [47] [51], de Subversion ou de SourceSafe. À budget quasi nul. Raison de plus pour ne pas investir du temps et de l'argent dans un CMS. Les gains de productivité spectaculaires reportés par certaines entreprises suite à la mise en place d'un CMS DITA XML ont cependant de quoi faire réfléchir. Ainsi, Epson America a pu réutiliser jusqu'à 90 % du contenu existant sur de nouveaux projets.

Si l'on opte pour un CMS, celui-ci doit clairement supporter DITA XML : on ne gère pas un jeu de briques d'information comme un document monolithique. Adieu donc SharePoint ou Alfresco, il faut se tourner vers des solutions dédiées telles que Componize [48] ou DocZone [49].

Quel que soit le choix initial, il est possible à tout instant de changer de stratégie, sans remettre en cause l'existant. L'architecture DITA XML n'est en effet liée à aucun référentiel particulier. Rien n'interdit donc de commencer à gérer ses projets sans CMS, puis d'avoir recours à une telle solution si les bénéfices de ce choix deviennent manifestes.

Voir aussi :

— *Git : du fichier au contenu* (page 11)

Cas concret : documentation de NuFirewall

La documentation de NuFirewall [52], qui a été perçue par la presse comme un point fort du produit [53], a été réalisée sous DITA XML.

Si je n'avais pas utilisé un format qui favorise au maximum la réutilisation de l'information, je n'aurais pas autant pu me consacrer à l'essentiel : le contenu.

Partager des blocs d'information atomiques avec les *conref*

Lorsque le rédacteur technique veut réutiliser des blocs d'information DITA XML plus petits qu'une section, il doit les partager au niveau des fichiers de contenu *dita* et non dans les structures de table des matières *ditamap*, grâce au mécanisme conref [54].

Le principe des *conref* est simple : lorsqu'un *conref* est mentionné au niveau d'un nœud XML donné, tout le contenu du nœud cible est remplacé par le contenu du nœud source.

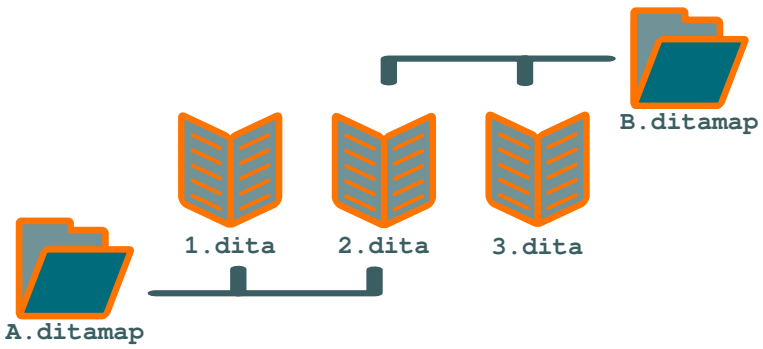


Fig. 3.11 – *Partage de blocs d'information de granulométrie large entre les* ditamap

50. Le rédacteur technique peut ainsi voir où un élément d'information est inclus ; lors de la mise à jour de cet élément, il peut alors juger si l'élément modifié sera toujours valable dans les différents contextes.
47. http ://www.git-scm.com
51. Il est possible, quoiqu'un peu complexe, de mettre en place des *workflows* sous Git *via* des branches.
48. http ://www.componize.com
49. http ://www.doczone.com
52. http ://linuxfr.org/news/nufirewall-le-pare-feu-libre-sans-prise-de-t%C3%AAte
53. http ://www.linformaticien.com/tests/id/20068/categoryid/48/edenwall-nufirewall-le-pare-feu-nouvelle-generation.aspx
54. http ://docs.oasis-open.org/dita/v1.1/OS/archspec/conref.html

Une différence notable entre le mécanisme des *conref* et le mécanisme XML des xinclude [55], c'est que le nœud source doit être conforme au schéma XSD du fichier source *et* du fichier cible. Ce formalisme rigoureux, s'il s'avère moins souple et oblige parfois à quelques acrobaties, rend les *conref* beaucoup plus lisibles que les *xinclude* et favorise leur utilisation.

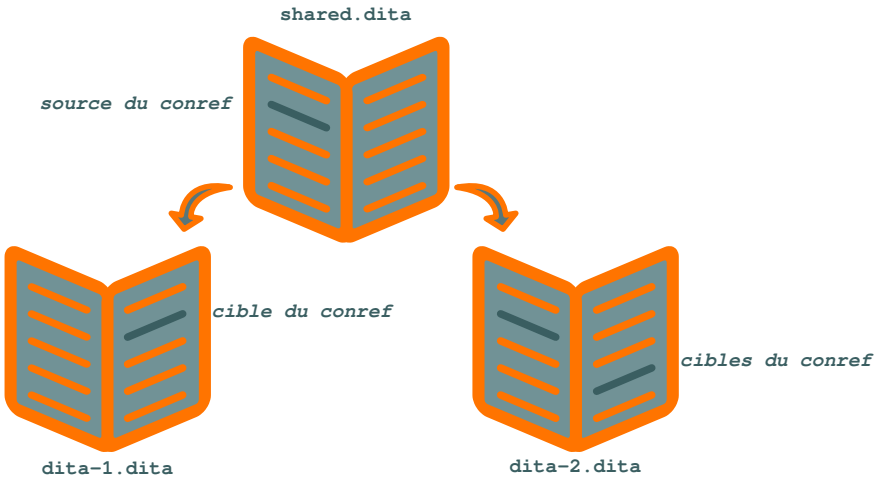


Fig. 3.12 – *Partage de blocs d'information de granulométrie fine entre les sections DITA XML*

Centraliser les *conref* dans un fichier unique

Pour favoriser l'utilisation des conref [56] au sein d'une équipe de rédacteurs techniques, et également pour simplifier la maintenance des *conref*, il s'avère très efficace de centraliser tous les *conref* dans un fichier DITA XML dédié.

Il est *a priori* plus simple, pour réutiliser un contenu d'un fichier DITA XML existant, de pointer vers ce contenu sans l'extraire de son contexte d'origine. Cependant, un des grands principes de la réutilisation du contenu est de décontextualiser le contenu. Il est donc à terme beaucoup plus efficace pour le rédacteur technique d'extraire le contenu réutilisé de son fichier d'origine et de le placer dans un fichier ne contenant que des sources de *conref*. Il est en effet beaucoup plus facile de placer tous les éléments sources dans un référentiel unique que de devoir chercher les différentes sources dans une multitude de fichiers.

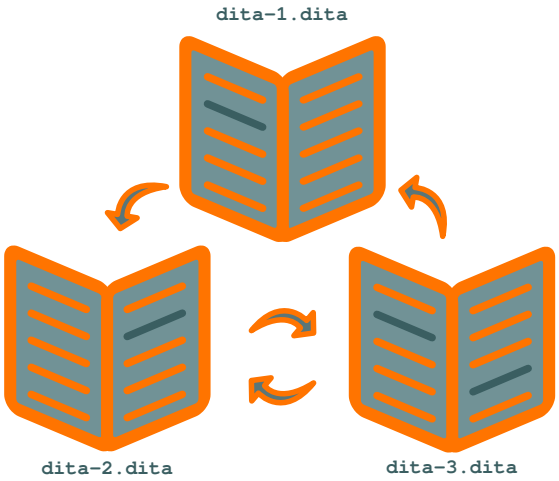


Fig. 3.13 – *Gérer les* conref *de manière décentralisée est peu efficace*

Les *conref* sont en effet résolus à la compilation même si les fichiers contenant les valeurs sources ne sont pas référencés dans le fichier *ditamap* permettant de générer le livrable (ce qui veut dire également que les fichiers contenant les valeurs sources des *conref* peuvent se trouver dans un répertoire de niveau supérieur à celui du *ditamap*).

55. http ://en.wikipedia.org/wiki/XInclude
56. http ://docs.oasis-open.org/dita/v1.1/OS/archspec/conref.html

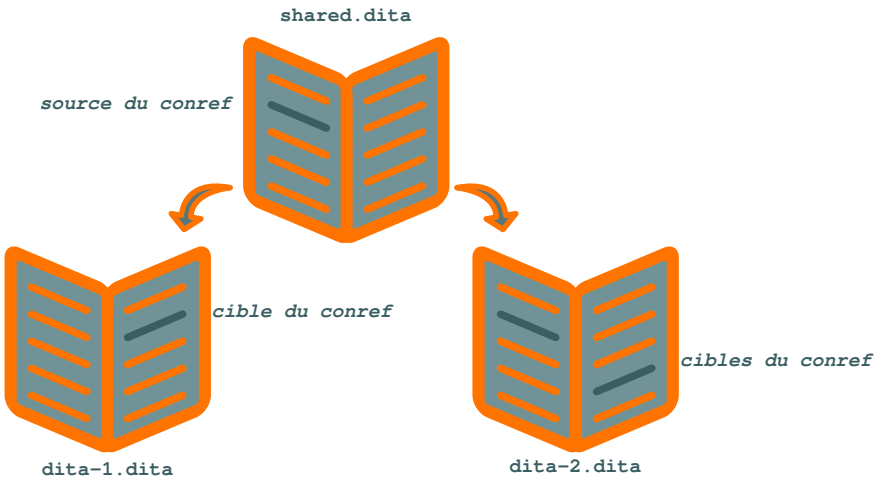


Fig. 3.14 – *Bonne gestion des* conref

Les fichiers de contenu référencés dans des structures *ditamap* ne contiennent donc que des *conref* cibles, et un fichier central fédère tous les *conref* sources ; il contient éventuellement également quelques références internes à des *conref* cibles.

Ce fichier central doit être de même type (*task*, *concept*, *reference*, etc.) que les fichiers de contenu, ou du moins du type *composite*, qui accepte tous types de structures DITA XML. Pour des raisons d'organisation, je trouve personnellement efficace de créer un fichier central par type de *topic* DITA XML, et donc de même type, pour partager les informations propres à chaque type. Je réserve le type *composite* à un fichier central *fourre-tout* contenant des informations partagées entre différents types de *topics*.

Tous les *conref* sources d'un fichier donné doivent avoir un ID unique dans ce fichier ; veillez à utiliser des noms explicites pour les humains, vos fichiers *dita* contenant des *conref* cibles deviendront sinon rapidement illisibles !

Utiliser le nœud XML de plus bas niveau

Le rédacteur technique doit utiliser comme source du conref [57] le nœud DITA XML de plus bas niveau contenant l'information à partager.

Le but des *conref* étant de gérer des blocs d'information de faibles dimensions, il est logique de les manipuler au niveau de la plus petite structure XML encapsulant l'information, même si cette structure, pour être compatible avec le schéma XSD de la section DITA XML où elle intervient, doit elle-même être incluse dans des structures XML plus grandes.

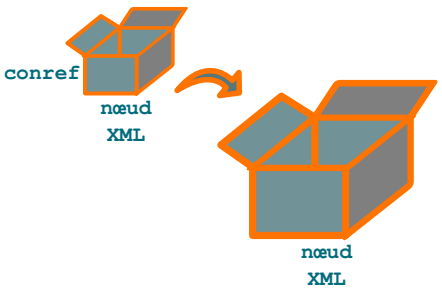


Fig. 3.15 – *Placement du* conref *sur le nœud XML de plus bas niveau*

Vous voulez par exemple réutiliser la phrase *Cliquez sur OK*. Vous ne pouvez cependant pas indiquer dans le fichier contenant les *conref* sources uniquement le code suivant :

```
<cmd>Cliquez sur OK.</cmd>
```

Pour être conforme au schéma XSD, votre code doit au moins être structuré comme suit :

57. http ://docs.oasis-open.org/dita/v1.1/OS/archspec/conref.html

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE task PUBLIC "-//OASIS//DTD DITA 1.2 Task//EN"
/usr/share/dita-ot/dtd/technicalContent/dtd/task.dtd"">
<task id="shared" xml:lang="fr-fr">
  <title>Conref source</title>
  <taskbody>
    <steps>
      <step>
        <cmd>
          Cliquez sur OK.
        </cmd>
      </step>
    </steps>
  </taskbody>
```

Il s'agit maintenant de placer un ID sur une structure XML afin de pouvoir réutiliser le contenu de cette structure. En l'occurrence, c'est une étape unique comprenant une commande unique que vous souhaitez réutiliser.

Il est alors préférable d'utiliser la syntaxe suivante :

```
<step>
  <cmd id="click-ok">
    Cliquez sur OK.
  </cmd>
</step>
```

plutôt que la suivante :

```
<step id="click-ok">
  <cmd>
    Cliquez sur OK.
  </cmd>
</step>
```

En effet, dans le premier cas, vous pourrez utiliser le *conref* même si le nœud supérieur (`<step>`) contient d'autres nœuds que `<step>` (par exemple `<info>`).

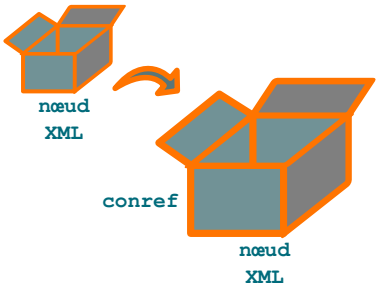


Fig. 3.16 – *Placement du* conref *sur le nœud XML de plus haut niveau*

Dans le 2e cas, tout le contenu du nœud `<step>` sera remplacé par la valeur du *conref* source. Par exemple, dans le cas suivant, tout le contenu du nœud sera absent des livrables :

```
<step id="click-ok">
  <cmd/>
  <info>
    Si vous ne savez pas lire, c'est le bouton vert.
  </info>
</step>
```

Prendre en compte les contraintes de traduction

L'unité d'information DITA XML la plus petite est le nœud `<ph>`. Le rédacteur technique doit cependant veiller à ne lui appliquer le mécanisme *conref* que pour une phrase complète ou un terme qui ne sera jamais traduit (par exemple, le nom de la société ou d'un produit). De gros problèmes apparaissent sinon lors de la traduction dans d'autres langues.

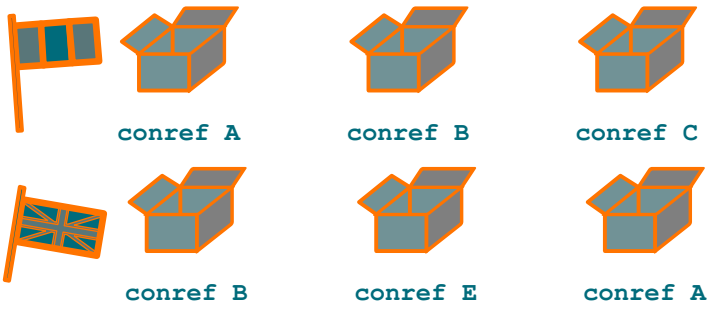


Fig. 3.17 – *Les phrases se découpent différemment selon les langues.*

Exemple

Si vous décidez de pousser la granulométrie au niveau du segment de phrase et que vous définissez les *conref* suivants :

```
<ph id="click">Click the</ph>

<ph id="blue">blue</ph>

<ph id="arrow">arrow</ph>
```

Vous pouvez maintenant utiliser le code suivant :

```
<p>
  <ph conref="shared.dita/click"/>
  <ph conref="shared.dita/blue"/>
  <ph conref="shared.dita/arrow"/>.
</p>
```

pour que soit générée la phrase *Click the blue arrow*.

Essayons maintenant de créer une version française de cette phrase. Nous traduisons donc les *conref* comme suit :

```
<ph id="click">Cliquez sur la</ph>

<ph id="blue">bleue</ph>

<ph id="arrow">flèche</ph>
```

Nous obtenons alors la phrase *Cliquez sur la bleue flèche*.

Pour pallier ce problème, il faudrait réorganiser l'ordre des *conref* dans le fichier DITA XML traduit, ce qui est difficilement gérable et fait perdre tout l'intérêt du mécanisme. Sans compter que des problèmes pires que ce cas d'école peuvent conduire à complètement abandonner dans la langue cible les *conref* utilisés dans la langue source (je n'ai pas d'exemple concret à offrir, ayant toujours évité de tomber dans ce genre de travers.)

Imbriquer les *conref*

Pour des raisons de facilité de mise à jour et de maintenance du contenu DITA XML, le rédacteur technique doit limiter l'effet *poupée russe* et ne pas trop imbriquer les conref[58]. Un seul niveau d'imbrication (un *conref* imbriqué dans un autre) me semble le seuil au-delà duquel le contenu peut vite devenir ingérable.

Dans l'exemple ci-dessous, le *conref* source *see-admin-guide* contient le *conref* cible *admin-guide-title* :

Exemple

```
<p id="see-admin-guide">
  Pour de plus amples informations, voir le <ph
  conref="shared.dita/admin-guide-title"/>.
</p>
```

58. http ://docs.oasis-open.org/dita/v1.1/OS/archspec/conref.html

Ce niveau de complexité est gérable. Mais si le *conref* source *admin-guide-title* contient lui même un *conref* cible, le code DITA XML devient un vrai plat de spaghettis (sans compter les risques de référence circulaire). Les *conref* peuvent théoriquement être combinés à l'infini, mais les problèmes pratiques que cela engendre peuvent également être infinis !

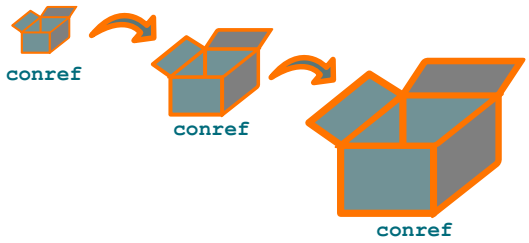


Fig. 3.18 – *Imbriquer les* conref *sur plusieurs niveaux : puissant, mais dangereux !*

Pour résumer la situation :

— Il est tout à fait possible d'imbriquer plusieurs *conref* sources. Le seul effet de bord négatif porte sur la lisibilité du fichier contenant les *conref*.
— L'imbrication de *conref* sources et cibles est possible mais rapidement ingérable.
— Il est impossible d'imbriquer des *conref* cibles : le contenu du *conref* du niveau supérieur écrasera les valeurs des *conref* du niveau inférieur.

Maximiser l'utilisation des *conref* pour faire baisser les coûts

Recourir aux conref [59] est le meilleur moyen dont dispose le rédacteur technique pour faire baisser spectaculairement les coûts et les délais de publication de son contenu DITA XML, surtout pour les documents multilingues.

De par la nature des informations qu'elles contiennent, les sections de type *task* ont un taux plus élevé de réutilisation du contenu que celles de type *concept* ou *reference*.

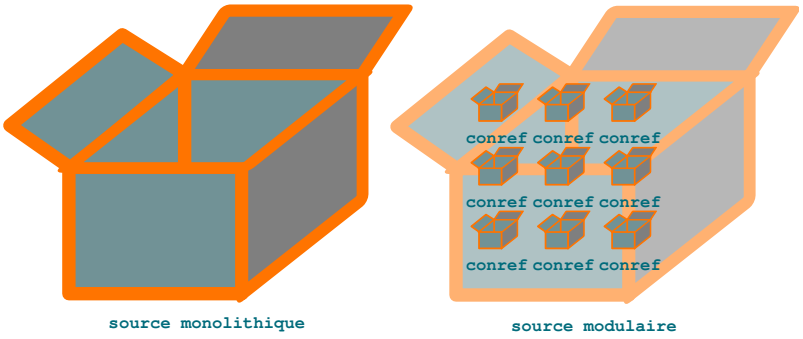


Fig. 3.19 – *Les* conref *modularisent de petits blocs d'information*

Comme dans l'exemple ci-dessous, il n'est pas rare d'obtenir rapidement des fichiers dont la seule valeur unique est le titre, le reste du contenu, *pourtant unique* (car il assemble de manière unique des blocs d'information non uniques), étant généré par des *conref*.

Exemple

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE task PUBLIC "-//OASIS//DTD DITA 1.2 Task//EN"
"/usr/share/dita-ot/dtd/technicalContent/dtd/task.dtd">
<task id="display-trends" xml:lang="fr-fr">
  <title>Afficher les tendances</title>
  <taskbody>
    <context audience="basic">
      <note type="restriction" audience="advanced">
        <ul>
```

59. http ://docs.oasis-open.org/dita/v1.1/OS/archspec/conref.html

```
            <li>
              <ph conref="shared.dita/ip-control"/>
            </li>
          </ul>
          <ph conref="../../shared/shared.dita/see-user-guide"
          audience="no-user-guide"/>
        </note>
      </context>
      <steps>
        <step>
          <cmd audience="basic">
            <menucascade>
              <uicontrol conref="shared.dita/logs"/>
            </menucascade>
          </cmd>
          <choices audience="advanced">
            <choice>
              <ph conref="shared.dita/physical-appliance"/>
              <menucascade>
                <uicontrol conref="shared.dita/logs"/>
              </menucascade>
            </choice>
            <choice>
              <ph conref="shared.dita/virtual-appliance"/>
              <menucascade>
                <uicontrol conref="shared.dita/server"/>
                <uicontrol conref="shared.dita/logs"/>
              </menucascade>
            </choice>
          </choices>
        </step>
        <step>
          <cmd>
            <menucascade>
              <uicontrol conref="shared.dita/all"/>
              <uicontrol conref="shared.dita/editfile"/>
            </menucascade>
          </cmd>
          <info>
            <ul conref="shared.dita/drill-down">
              <li/>
            </ul>
            <note conref="shared.dita/randomnames"/>
          </info>
        </step>
      </steps>
    </taskbody>
  </task>
```

Seul le texte en noir doit être traduit. Traduire ce type de fichier de contenu DITA XML consiste donc à traduire uniquement le titre de la section et l'intégralité des *conref* sources. Lorsqu'il traduit un ensemble d'unités d'information placées en vrac dans un fichier, le traducteur manque cependant cruellement de contexte. Le créateur du contenu initial doit donc lui fournir une assistance constante. La méthode la plus efficace consiste à faire travailler le traducteur en régie. Avantage supplémentaire : il pourra ainsi interroger non seulement le rédacteur technique, mais également les concepteurs du produit.

Note : Ne croyez pas qu'il s'agit là d'une contrainte spécifiquement induite par la modularisation poussée du contenu. Pour avoir fait une école de traduction reposant sur le principe simple mais efficace du *triangle du sens* (le traducteur doit comprendre le texte source pour le reformuler dans le texte cible et non transcrire une suite de mots d'une langue à l'autre) et avoir pratiqué la traduction technique durant plusieurs années, je sais que tout projet de traduction réussi repose sur une collaboration efficace entre concepteurs, rédacteurs et traducteurs.

Il est également possible de factoriser ainsi des éléments de structure, et non de contenu, tels que des en-têtes de tableaux. Vous pouvez ainsi présenter des informations de même type de manière homogène à moindre coût, c'est à dire sans recourir à la spécialisation [60].

60. http ://en.wikipedia.org/wiki/Darwin_Information_Typing_Architecture#Specialization

Protéger les informations confidentielles

Le puissant mécanisme conref[61] de DITA XML se prête à d'autres applications que la réduction des coûts. Par exemple, le rédacteur technique peut masquer des informations dans le code source.

Voici un cas original d'utilisation des *conref* : imaginez que vous devez faire traduire un fichier contenant des informations confidentielles qui ne doivent pas figurer dans la version traduite et auxquelles le traducteur ne doit pas avoir accès (une clause de confidentialité interdit aux clients de diffuser l'information dont ils disposent).

Comment faire ? Le filtrage à l'aide du mécanisme *ditaval* est conçu pour exclure des informations des livrables, non pour les masquer dans les fichiers sources. Allez-vous devoir créer deux jeux de fichiers sources, certains comportant les informations confidentielles, les autres non ? Adieu alors le *single-sourcing* et la réutilisation du contenu qui vous ont fait choisir DITA XML !

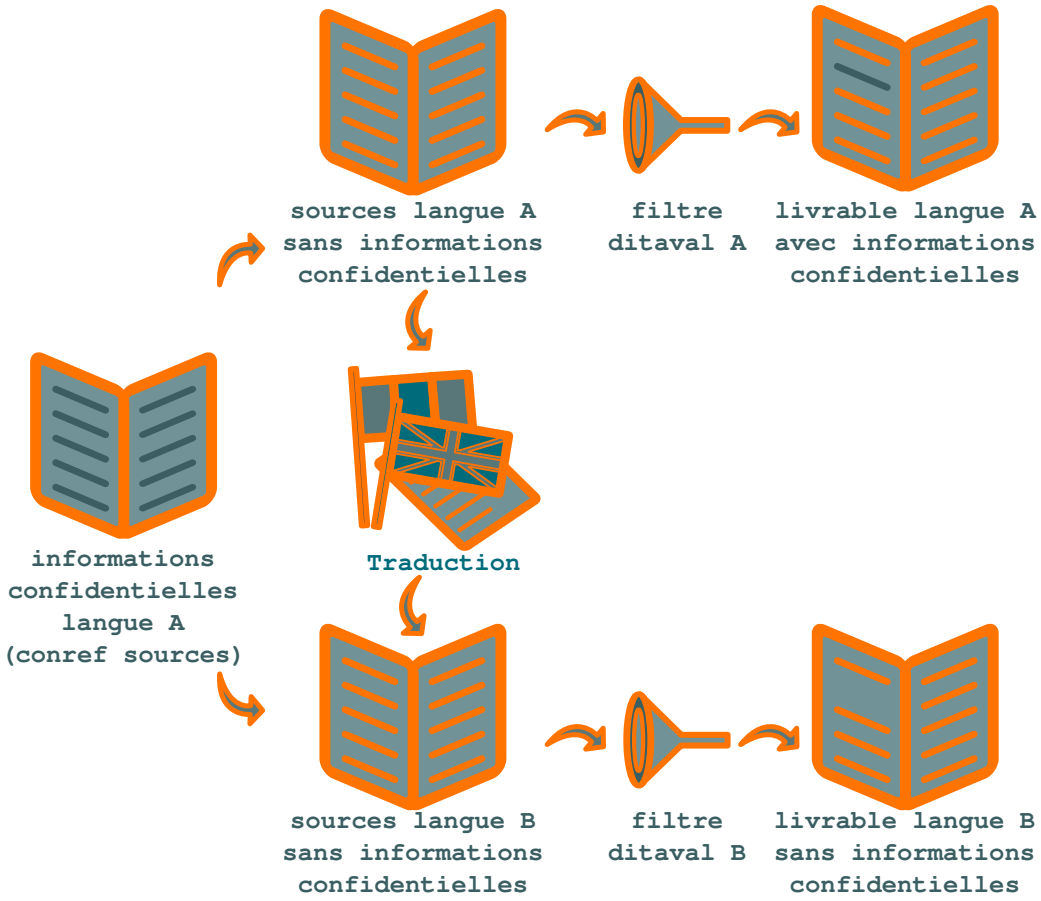


Fig. 3.20 – *Masquer des informations confidentielles aux traducteurs*

En plaçant le contenu confidentiel dans un fichier que vous appelez par exemple `confidentiel.dita` et en plaçant des *conref* assortis d'une clé de filtrage dans le fichier à traduire, vous avez résolu votre problème : le traducteur ne traduira que le texte non confidentiel, et le livrable généré dans la langue cible ne contiendra pas le texte confidentiel, noté comme conditionnel et exclu explicitement par le fichier *ditaval* passé en argument lors de la compilation.

Fournir une information ciblée avec le texte conditionnel ditaval

Un fichier ditaval[62] reprend le principe des lunettes que vous chaussez pour visualiser un film en 3D : le verre gauche masque une moitié de l'image, le verre droit en masque l'autre moitié. Mais seul le rédacteur technique dispose de lunettes 3D et a une vision complète de l'information contenue dans le projet DITA XML.

Les destinataires de l'information disposent de lunettes avec deux verres gauches ou deux verres droits. Ils ne voient donc qu'une partie de l'information. Loin d'être lésés par cet état de fait, ils ont ainsi un meilleur accès à l'information.

61. http ://docs.oasis-open.org/dita/v1.1/OS/archspec/conref.html
62. http ://docs.oasis-open.org/dita/v1.2/os/spec/common/about-ditaval.html

Le profilage réalisé masque à chaque public les informations dont ils n'ont *pas* besoin et qui ne seraient pour eux que du bruit. Chaque audience bénéficie donc d'un meilleur accès à l'information qui la concerne, selon le fameux concept minimaliste de *less is more*.

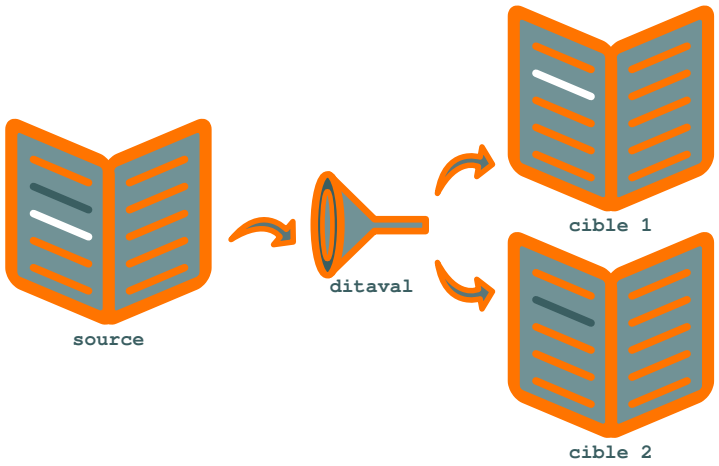


Fig. 3.21 – *Texte conditionnel avec DITA XML*

Concrètement, le mécanisme *ditaval* est basé sur des opérateurs binaires : vous marquez un bloc d'information avec un attribut et une valeur, puis incluez ou excluez ce bloc dans le livrable en passant un opérande lors de la compilation (le bloc est inclus par défaut si aucun opérande n'est spécifié). C'est le principe du texte conditionnel.

Gâce à ce mécanisme, il n'est pas nécessaire de créer deux fichiers différents lorsque leur contenu ne comporte que des variations mineures. C'est un outil de plus destiné à réduire la redondance du contenu source.

Vous pouvez appliquer des clés de filtrage en série (condition *et*) en indiquant plusieurs valeurs séparées par des espaces dans les attributs *product*, *audience* ou autre.

Exemple

Pour indiquer qu'une remarque est destinée à la fois à des électriciens et à des utilisateurs avancés en voulant profiler l'information selon les publics suivants :

- non électriciens,
- électriciens débutants,
- électriciens experts.

Vous pouvez utiliser la structure suivante :

```
<step audience="electricians advanced">
  <cmd> Ramenez l'intensité sous la dose létale de 150mA. </cmd>
</step>
```

Attention : Une clé de filtrage mal positionnée peut entraîner une erreur de compilation. En effet, si le code non filtré est conforme au schéma XSD DITA XML, le code filtré peut ne pas l'être.

Exemple

Le code suivant est correct avant filtrage :

```
<thead>
  <row product="a>
    <entry>Commande</entry>
    <entry>Description</entry>
  </row>
</thead>
```

Après filtrage, en revanche, on obtient le code suivant :

```
<thead>
</thead>
```

Or, selon le schéma XSD, les en-têtes de tableaux doivent contenir au moins une ligne :

```
<!ENTITY % thead.content "((%row;)+)>
```

Ce code est donc incorrect et entraîne l'échec de la compilation.

4. Le coin du *geek*

Je suis un rédacteur technique à tendance *geek*. Après une grande frustration ressentie lors de l'utilisation d'outils tels que FrameMaker ou Flare, j'ai progressivement eu envie de voir ce qui se passait « sous le capot » et de comprendre ce qui se cachait derrière les interfaces graphiques.

Cette approche facilite le dialogue avec les développeurs et donne une plus grande maîtrise des données. Elle se traduit par un certain nombre de caractéristiques qui se reflètent dans la pratique de mon métier :

Caractéristique	Application pratique
Recherche de rationnalisation	Industrialisation des formats, outils et processus pour minimiser la quantité de travail inutile.
Capacité d'adaptation	Vision du changement comme opportunité d'acquérir et d'utiliser de nouvelles connaissances.
Forte autonomie de travail	Créativité dans le choix des moyens mis en œuvre pour atteindre efficacement les objectifs.
Sens de l'innovation	Recherche constante de nouvelles solutions (dont il faut cependant évaluer le coût de mise en œuvre et l'impact humain).
Goût des rapports humains	Préférence pour le dialogue face à face pour éviter tout présupposé ou incompréhension.
Capacité d'apprentissage dans des domaines variés	Apprentissage « sur le tas » de formats de documentation, langages informatiques ou environnements en ligne de commande.
Recherche de l'excellence	Amélioration de la satisfaction client, minimisation de la maintenance et meilleure réutilisation du contenu existant.

Si vous vous reconnaissez dans ce portrait et n'avez pas peur du mode console, c'est par ici !

Voir aussi :

Being a Techie Writer [63]

Le Raspberry Pi 3 en tant que plateforme de documentation

Faut-il une débauche de puissance pour générer une documentation professionnelle ? Avec son unique giga-octet de mémoire vive et son processeur de *smartphone*, le Raspberry Pi 3 semble se positionner comme une bonne station bureautique des années 2000… À l'usage, il s'avère pourtant qu'une unité centrale d'une quarantaine d'euros suffit largement pour créer, gérer et générer une documentation aux formats PDF, HTML, ou autre.

Note : Les buts de ce billet sont de :

— Présenter un POC (Proof of Concept, démonstration de faisabilité) et utiliser des ressources minimales pour créer, gérer et publier une documentation professionnelle. La plupart des opérations se déroulent donc en mode texte, sous Linux. Si les solutions présentées ici fonctionnent également en mode graphique sous Windows, elles ne sont peut-être pas disponibles sous *Windows 10 IoT*, destiné au Raspberry Pi 3.

— Présenter un scénario d'utilisation aussi simple que possible, parfois au détriment de l'élégance technique.

Configurez le Raspberry Pi 3

Prérequis

— Carte micro-SD de 16 Go classe 10 (de préférence).

— Connexion Internet filaire ou Wi-Fi.

63. http ://techwhirl.com/techie-writer-series-part-i-being-a-techie-writer/

1. Installez la distribution Linux Raspbian sur votre Raspberry Pi 3 *via* NOOBS [64].
2. Sélectionnez **Menu > Preferences > Raspberry Pi Configuration**.
   La boîte de dialogue *Raspberry Pi Configuration* apparaît.
3. Sélectionnez l'onglet *Localisation*.
4. Cliquez sur **Set Locale**, sélectionnez les options suivantes, puis cliquez sur **OK** :

Option	Valeur
Language	fr (French)
Country	FR (France)
Character Set	UTF-8

5. Cliquez sur **Set Keyboard**, sélectionnez les valeurs correspondant à votre clavier, puis cliquez sur **OK**.
6. Cliquez sur **OK** dans la boîte de dialogue *Raspberry Pi Configuration*.
7. Sélectionnez **Menu > Accessories > Terminal**.
8. Mettez à jour le système :

```
$  sudo aptitude update && sudo aptitude safe-upgrade -y
```

   Le temps de lire un épisode du *Surfer d'argent*, et le système est mis à jour.
9. Sélectionnez **Menu > Shutdown > Reboot**.
   Le Raspberry Pi 3 redémarre.

Installez les logiciels nécessaires à la gestion de ce document

1. Sélectionnez **Menu > Accessoires > LXTerminal**.
2. Installez les paquets logiciels suivants :

```
$ sudo aptitude install -y calibre emacs gitk inkscape python3-sphinx texlive-full
```

   Le temps de lire cinq ou six épisodes de *The Amazing Spider-Man*, et les logiciels suivants sont installés :

Logiciel	Description
Calibre	Gestionnaire de livres numériques
Emacs	Environnement de développement intégré.
Gitk	Navigateur d'historique du logiciel de gestion de versions décentralisé.
Inkscape	Logiciel de dessin vectoriel.
Python Sphinx	Générateur de documentation basé sur le format reStructuredText.
Texlive	Environnement LaTeX complet pour la génération du document au format PDF.

3. Libérez de l'espace disque :

```
$ sudo aptitude clean
```

Récupérez les sources de ce document

1. Clonez le dépôt *Git* des sources de ce document :

```
$ git clone https://github.com/olivier-carrere/redaction-technique.org.git
```

2. Placez-vous dans le répertoire des sources de ce document :

```
$ cd redaction-technique.org
```

Créez et modifiez le texte

1. Modifiez un fichier source modulaire de ce document :
   — à l'aide d'un éditeur de texte :

64. https ://www.raspberrypi.org/downloads/noobs/

```
$ leafpad *coin-du-geek.rst &
```

— ou à l'aide d'un environnement de développement :

```
$ emacs *coin-du-geek.rst &
```

— ou à l'aide d'un éditeur en ligne, par exemple :

```
$ sed -i "s/répertoire/dossier/g;" *.rst
```

Créez et modifiez les schémas

1. Modifiez un fichier source des images de ce document :
   — à l'aide d'un logiciel de dessin vectoriel :

```
$ inkscape graphics/modulaire-texte-monolithique-binaire.svg &
```

— ou à l'aide d'un éditeur en ligne :

```
$ sed -i "s/docbook/XML/g;" graphics/*.svg
```

Gérez les versions de votre documentation

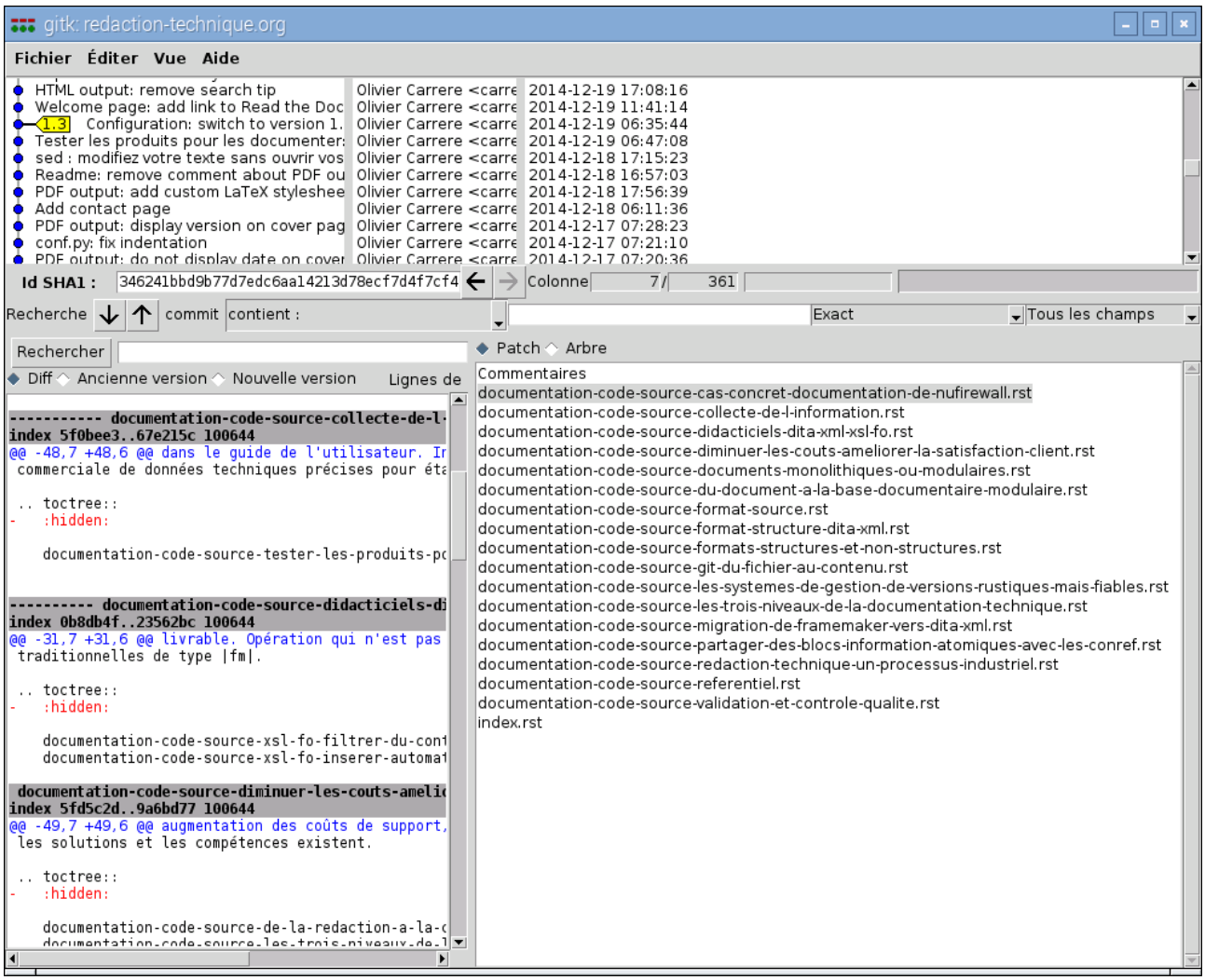
1. *Commitez* votre lot de modifications sous *Git* :

```
$ git config --global user.email "votre email"
$ git config --global user.name "votre nom"
$ git add *.rst
$ git commit -m "Mon lot de modifications de texte"
$ git add graphics/*.svg
$ git commit -m "Mon lot de modifications sur les images"
```

2. Affichez l'historique des modifications des sources de ce document :

```
$ gitk &
```

Ô surprise, vous avez sous les yeux, mais oui, une GUI (Graphical User Interface) ! C'est tellement beau, qu'on va faire une photo :



Un commit *atomique s'étendant sur une bonne quinzaine de fichiers*

Note :

— Vos modifications sont purement locales et ne sont pas appliquées sur le dépot distant *GitHub*.

— Si vos modifications apportent une réelle valeur ajoutée à ce document (correction de coquille, ajout d'information ou autre), n'hésitez pas à me la soumettre sous forme de patch *Git* ou *via* votre compte *GitHub*.

— *GitHub* n'est probablement pas hébergé sur un cluster de Raspberry Pi 3. Rien n'empêche cependant d'héberger un dépôt distant *Git* sur un Raspberry Pi 3 connecté au réseau et d'y accéder par connexion sécurisée SSH (Secure Shell).

Générez votre documentation

1. Revenez dans le terminal, puis récupérez la dernière version *taguée* de ce document :

```
$ git checkout $(git describe --tags $(git rev-list --tags --max-count=1))
```

Note : Oui, je sais, cette commande ne correspond pas exactement à la définition de *simple* donnée par le Larousse…

---

2. Générez la dernière version *taguée* de ce document aux format PDF, HTML et EPUB :

```
$ make all
```

3. Affichez le document au format PDF :

```
$ xpdf _build/latex/redaction-techniqueorg.pdf &
```

4. Affichez le document au format HTML :

```
$ epiphany _build/html/index.html &
```

5. Affichez le document au format EPUB :

```
$  ebook-viewer _build/epub/redaction-techniqueorg.epub &
```

Et voilà. En quelques minutes, vous avez :

— Appliqué des règles de texte conditionnel à des sources communes selon le format de publication. Ce contenu est par exemple appelé *livre électronique* dans la version EPUB, *document* dans la version PDF et encore autrement dans la version HTML.
— Généré dans trois formats différents une documentation d'une soixantaine de pages comprenant une quarantaine de schémas.

Note :

— Le fichier `Makefile` est assez brut de décoffrage et le temps de compilation peut facilement être optimisé.
— Nous pourrions mettre en place une solution complète de texte conditionnel avec opérateurs booléens et tout et tout grâce au moteur de *templating* Jinja [65].
— Les observateurs remarqueront que la version HTML du document version 1.5 ne comporte pas de table des matières dans la colonne de droite. C'est qu'en effet, cette version n'embarque pas le patch *1032292*. Je vous laisse chercher dans l'historique *Git*… voire créer une branche et le *cherry-picker* !

Le Raspberry Pi 3 est donc une plateforme de documentation tout à fait crédible… à condition de se passer, ou presque, d'interface graphique !

Le prochain test consistera à générer la version DITA XML de ce document.

Le prochain prochain test consistera à générer ce document sur un *smartphone* en installant une distribution Linux sur Android [66].

sed : modifiez votre texte sans ouvrir vos fichiers

Les clones d'Unix sont peu utilisés pour gérer la documentation technique. Ceci est étrange si l'on songe à la pléthore d'outils disponibles sous ces plateformes pour manipuler du texte dans tous les sens.

Prenons l'exemple du dialogue entre entre M. Jourdain et son maître de philosophie, dans le *Bourgeois gentilhomme* de Molière :

> ***MONSIEUR JOURDAIN :*** *[...] Je voudrais donc lui mettre dans un billet : « Belle marquise, vos beaux yeux me font mourir d'amour » ; mais je voudrais que cela fût mis d'une manière galante, que cela fût tourné gentiment.*

[...]

> ***MAÎTRE DE PHILOSOPHIE :*** *On les peut mettre premièrement comme vous avez dit : Belle marquise, vos beaux yeux me font mourir d'amour. Ou bien : D'amour mourir me font, belle marquise, vos beaux yeux. Ou bien : Vos yeux beaux d'amour me font, belle marquise, mourir. Ou bien : Mourir vos beaux yeux, belle marquise, d'amour me font. Ou bien : Me font vos yeux beaux mourir, belle marquise, d'amour.*

Commençons par afficher la phrase d'origine dans un terminal :

```
$ echo "Belle marquise, vos beaux \\
yeux me font mourir d'amour."
Belle marquise, vos beaux yeux me font mourir d'amour.
```

Il s'agit maintenant d'intervertir les mots de la phrase, pour en créer une nouvelle. Pour une simple transposition, on pourrait juger plus facile d'utiliser *awk*. *awk* ne gère en effet pas des lignes, mais des *champs* d'un *enregistrement* (d'une ligne), délimités par défaut par des espaces. Autrement dit, *awk* traite le texte comme une base de données. Il peut facilement afficher toute la ligne, ou seulement un ou plusieurs champs, dans l'ordre souhaité. Les champs sont indiqués sous la forme *$n*, où n indique la position du champ dans la ligne, à partir de la gauche. Ainsi *$1* indique le premier champ, *$2* le dernier, etc. *$0* correspond à toute la ligne.

65. http ://jinja.pocoo.org/
66. https ://korben.info/comment-installer-linux-sur-android.html

Nous allons donc donner la déclaration d'amour de M. Jourdain en entrée d'un programme *awk* d'une ligne, grâce au symbole de redirection *pipeline* (|).

```
$ echo "Belle marquise, vos beaux \\
yeux me font mourir d'amour." |
awk  '{print $9" "$8" "$6" "$7" "$1" "$2" "$3" "$4" "$5}'
d'amour. mourir me font Belle marquise, vos beaux yeux
```

La sortie de la commande *echo* n'est pas affichée. Ce qui est affiché, c'est la sortie du programme *awk*, dont la sortie de la commande *echo*, soit la déclaration d'amour de M. Jourdain, était l'entrée.

La sortie finale ne correspond cependant pas à ce que l'on souhaitait. Les *champs* ne correspondent pas trait pour trait à des mots. Il faudrait donc raffiner la commande *awk*.

Il est plus simple de se tourner vers *sed*. *sed* sélectionne dans des lignes des ensembles de caractères cités littéralement, ou *via* des méta-caractères dans des *expressions rationnelles* (ou *expressions régulières*). Un méta-caractère connu des expressions rationnelles est le signe *, indiquant, en ligne de commande, zéro ou un nombre indéfini de caractères, comme dans :

```
$ ls *.rst
```

sed gère également des *références arrières*, qui affichent à l'endroit où on le souhaite la valeur correspondant à une expression littérale ou rationnelle trouvée auparavant. Heureusement pour nous, la déclaration d'amour de M. Jourdain contient exactement neuf mots, ce qui correspond au nombre maximal de références arrières possibles.

```
$ echo "Belle marquise, vos beaux \\
yeux me font mourir d'amour." |
sed "s#\(.*\) \(.*\), \(.*\) \(.*\) \(.*\) \(.*\) \(.*\) \\
\(.*\)\(d'.*\)#\9 \8 \6 \7, \1 \2, \3 \4 \5#"
d'amour. mourir me font, Belle marquise, vos beaux yeux
```

Nous buttons sur le même problème : l'expression régulière .* ne correspond pas à un mot, mais à une suite de caractères, ponctuation comprise. Il faut alors utiliser la forme <.*>, qui correspond à un mot tel que ceux dont M. Jourdain se sert pour faire de la prose. Nous allons utiliser les caractères d'échappement (barre oblique inverse \) pour que les signes < et > ne soient pas interprétés littéralement sous certaines consoles, mais comme des méta-caractères ayant une fonction spéciale :

```
$ export \
p="\(\<.*\>\) \(\<.*\>\), \(\<.*\>\) \(\<.*\>\) \\
\(\<.*\>\) \(\<.*\>\) \(\<.*\>\) \(\<.*\>\) \(d'\<.*\>\)"
$ echo "Belle marquise, vos beaux \\
yeux me font mourir d'amour." |
sed "s#$p#\9 \8 \6 \7, \1 \2, \3 \4 \5#"
d'amour mourir me font, Belle marquise, vos beaux yeux.
```

Nous pourrions également utiliser la forme [[:alpha :]]* qui fait gagner en lisibilité, mais perdre en concision :

```
$ export a="[[:alpha:]]"
$ export n="\($a*\) \($a*\), \($a*\) \($a*\) \($a*\) \\
\($a*\) \($a*\) \($a*\) \(d'$a*\)"
$ echo "Belle marquise, vos beaux \\
yeux me font mourir d'amour." |
sed "s#$n#\9 \8 \6 \7, \1 \2, \3 \4 \5#"
d'amour mourir me font, Belle marquise, vos beaux yeux.
```

C'est mieux, mais nous avons un problème de capitalisation. Nous allons donc utiliser les opérateurs /u et /l placés judicieusement. Auparavant, nous allons exporter des variables pour rendre le script plus concis et plus lisible :

```
$ export w="\(\<.*\>\)"
$ export m="$w $w, $w $w $w $w $w $w"
```

```
$ echo "Belle marquise, vos beaux \\
yeux me font mourir d'amour." |
sed "s#$m \(d'\<.*\>\)#\u\9 \8 \6 \7, \l\1 \2, \3 \4 \5#"
D'amour mourir me font, belle marquise, vos beaux yeux.
```

Nous pouvons maintenant facilement redistribuer les références arrières pour obtenir toutes les variations du maître de philosophie :

```
$ echo "Belle marquise, vos beaux \\
yeux me font mourir d'amour." |
sed "s#$m \(d'\<.*\>\)#\u\3 \5 \4 \9 \6 \7, \l\1 \2, \8#"
Vos yeux beaux d'amour me font, belle marquise, mourir.
```

```
$ echo "Belle marquise, vos beaux \\
yeux me font mourir d'amour." |
sed "s#$m \(d'\<.*\>\)#\u\8 \3 \4 \5, \l\1 \2, \9 \6 \7#"
Mourir vos beaux yeux, belle marquise, d'amour me font.
```

```
$ echo "Belle marquise, vos beaux \\
yeux me font mourir d'amour." |
sed "s#$m \(d'\<.*\>\)#\u\6 \7 \3 \5 \4 \8, \l\1 \2, \9#"
Me font vos yeux beaux mourir, belle marquise, d'amour.
```

Molière et GNU/Linux

Réécrivons le dialogue de M. Jourdain et de son maître de philosophie en style *geek* :

MONSIEUR JOURDAIN : Je voudrais donc lui afficher sur la sortie standard :

```
$ Belle marquise, vos beaux yeux me font mourir d'amour.
```

Mais je voudrais que cela fût mis d'une manière galante, que cela fût tourné gentiment.

MAÎTRE DE PHILOSOPHIE : On les peut mettre premièrement comme vous avez dit :

```
$ echo "Belle marquise, vos beaux \\
yeux me font mourir d'amour."
```

Ou bien :

```
$ export declaration="Belle marquise, vos \\
beaux yeux me font mourir d'amour."
$ echo $declaration
```

Ou bien :

```
$ export w="\(\<.*\>\)"
$ export m="$w $w, $w $w $w $w $w $w"
$ echo $declaration |
sed "s#$m \(d'\<.*\>\)#\u\9 \8 \6 \7, \l\1 \2, \3 \4 \5#"
```

Ou bien :

```
$ echo $declaration |
sed "s#$m \(d'\<.*\>\)#\u\3 \5 \4 \9 \6 \7, \l\1 \2, \8#"
```

Ou bien :

```
$ echo $declaration |
sed "s#$m \(d'\<.*\>\)#\u\8 \3 \4 \5, \l\1 \2, \9 \6 \7#"
```

Ou bien :

```
$ echo $declaration |
sed "s#$m \(d'\<.*\>\)#\u\6 \7 \3 \5 \4 \8, \l\1 \2, \9#"
```

Beaucoup d'efforts…

Certes, beaucoup d'efforts pour pas grand chose, me direz-vous. Mais imaginons un fichier qui contiennent 1000 phrases de la même structure :

Cher docteur, ces grands malheurs vous font pleurer d'amertume. Petit garçon, cette bonne glace te fait saliver d'envie. Vaste océan, la forte houle te fait tanguer d'ivresse.

Ceci est en l'occurrence peu probable, mais il est en revanche monnaie courante de trouver dans la documentation technique des phrases de même structure, pour des raisons d'homogénéité stylistique.

Pour effectuer nos tests sur un échantillon, plaçons les trois phrases précédentes dans un fichier :

```
$ echo "Cher docteur, ces grands malheurs \\
vous font pleurer d'amertume." > variations.txt

$ echo "Petit garçon, cette bonne glace te \\
fait saliver d'envie." >> variations.txt

$ echo "Vaste océan, la forte houle te \\
fait tanguer d'ivresse." >> variations.txt
```

Plaçons les différentes commandes *sed* dans un script différent chacune :

```
$ echo "s#$p#\u\9 \8 \6 \7, \l\1 \2, \3 \4 \5#" > moliere1.sed
$ echo "s#$p#\u\3 \5 \4 \9 \6 \7, \l\1 \2, \8#" > moliere2.sed
$ echo "s#$p#\u\8 \3 \4 \5, \l\1 \2, \9 \6 \7#" > moliere3.sed
$ echo "s#$p#\u\6 \7 \3 \5 \4 \8, \l\1 \2, \9#" > moliere4.sed
```

Exécutons maintenant en boucle tous les scripts *sed* sur toutes les lignes du fichier :

```
$ for (( i=1; i<5; i++ )); do
   while read s;
    do echo "$s" |
     sed -f moliere$i.sed ;
    done < variations.txt
   done
D'amertume pleurer vous font, cher docteur, ces grands malheurs.
D'envie saliver te fait, petit garçon, cette bonne glace.
D'ivresse tanguer te fait, vaste océan, la forte houle.
Ces malheurs grands d'amertume vous font, cher docteur, pleurer.
Cette glace bonne d'envie te fait, petit garçon, saliver.
La houle forte d'ivresse te fait, vaste océan, tanguer.
Pleurer ces grands malheurs, cher docteur, d'amertume vous font.
Saliver cette bonne glace, petit garçon, d'envie te fait.
Tanguer la forte houle, vaste océan, d'ivresse te fait.
Vous font ces malheurs grands pleurer, cher docteur, d'amertume.
Te fait cette glace bonne saliver, petit garçon, d'envie.
Te fait la houle forte tanguer, vaste océan, d'ivresse.
```

Et voilà. En quelques instants, sans jamais ouvrir un seul fichier, nous appliquons une suite d'opérations complexes sur un nombre indéfini de phrases de même structure. Ce qui n'est pas possible sous un traitement de texte ou autre outil muni d'une interface graphique, ou sur des fichiers binaires.

Voir aussi :

— *Expressions régulières en Python* (page 49)

Expressions régulières en Python

Le langage Python offre de nombreuses bibliothèques de fonctions. Celle dédiée aux expressions régulières peut vous aider à manipuler du texte, notamment si vous n'êtes pas familier des utilitaires *sed* ou *awk*.

Le code suivant illustre comment inverser l'ordre des mots d'une phrase, selon le fameux exemple du *Bourgeois gentilhomme* :

```
#! /usr/bin/python

"""

   Demande à l'utilisateur de saisir la tirade de M. Jourdain et en affiche
   une variante. Fonctionne uniquement avec la tirade suivante :

   Belle marquise, vos beaux yeux me font mourir d'amour.

"""

# Importation de la bibliothèque d'expressions régulières.
import re

# Le programme demande à l'utilisateur de saisir la tirade et la place dans
# une variable.
texte_original = input("Entrez la tirade de M. Jourdain : ")
```

```
# Suppression du point final de la tirade.
texte_original_sans_point = texte_original.strip('.')
# Conversion de la chaîne de caractères en liste de mots.
texte_melange = re.split(' ',texte_original_sans_point)
# Création d'une variable globale contenant une chaîne de caractères vide.
texte_final = ""

# Après vérification que la tirade comporte bien 9 mots, affichage du premier
# mot de la tirade modifiée, avec une lettre capitale.
if len(texte_melange) == 9:
   print(str.capitalize(texte_melange[8]),end="")

   # Une boucle ajoute à la chaîne de caractères initialement vide les mots de
   # la tirade de M. Jourdain, à l'exception du premier et du dernier, dans
   # l'ordre voulu.

   for i in [7,5,6,0,1,2,3]:
      texte_final = texte_final + " " + str.lower(texte_melange[i])

   # Affichage du texte final, suivi d'un point.
   print(texte_final,end=" ")
   print(str.lower(texte_melange[4]),end="")
   print(".")

else:
   print("La tirade doit comporter neuf mots exactement.")
```

Voir aussi :

— *sed : modifiez votre texte sans ouvrir vos fichiers* (page 46)

Didacticiels DITA XML et XSL-FO

Les didacticiels suivants aideront le rédacteur technique à mettre en place et à utiliser une chaîne de création et de publication DITA XML libre.

DITA XML est un langage de rédaction structurée qui permet de créer des documents sans se soucier de leur aspect final sur différents supports. XSL-FO est un langage qui permet de réorganiser et filtrer le contenu XML et de lui appliquer une mise en page à l'aide d'une feuille de style.

Un ensemble de fichiers DITA XML contient tout le contenu, relatif par exemple à un produit. Différentes feuilles de style XSL-FO permettront de publier ce contenu en PDF, en HTML ou sous un autre format en appliquant des transformations complexes. Le résumé de chaque section du document final pourra par exemple apparaître dans la version HTML et non dans la version PDF.

De même, si un produit doit être fourni en marque blanche à différents clients, une mise en page totalement différente peu être appliquée à sa documentation en spécifiant simplement un autre jeu de feuilles de style lors de la génération du livrable. Opération qui n'est pas envisageable en pratique avec des solutions traditionnelles de type FrameMaker.

XSL-FO : filtrer du contenu selon des conditions « sauf » et « ou »

Imaginons que vous vouliez filtrer les nœuds enfants de la balise DITA XML `<example>` et afficher tout son contenu à l'exception du titre (situé entre les balises `<title>`).

Vous pouvez recourir alors à la syntaxe suivante :

```
<xsl:template match="*[contains(@class,' topic/example ')]">
  <fo:block>
    <xsl:apply-templates select="*[not(name()='title')]" />
  </fo:block>
</xsl:template>
```

Cette commande sélectionne tous les nœuds enfants du nœud `<example>`, à l'exception du nœud `<title>`. Cependant, le nœud `<example>` accepte le texte entré directement, sans être encapsulé dans des balises. Cette commande ne fera alors pas apparaître ce contenu.

Supposons que le code source d'un de vos fichiers DITA XML soit le suivant :

```
<example>
  <title>
    XSL-FO
  </title>
  Voici mon exemple de chemin XPATH :
  <codeblock>
    ancestor-or-self
  </codeblock>
  Texte non encapsulé situé après un nœud enfant.
</example>
```

Le fichier PDF affichera l'exemple structuré comme suit :

```
ancestor-or-self
```

Le titre de l'exemple n'est pas affiché, ce qui correspond au résultat souhaité, mais le contenu non encapsulé dans des balises n'apparaît pas, ce qui est un effet de bord indésirable. Pour sélectionner ce contenu, il faut sélectionner les nœuds textuels avec la syntaxe `text()`. Il est alors tentant d'utiliser la syntaxe suivante :

```
<xsl:template match="*[contains(@class,' topic/example ')]">
  <fo:block>
    <xsl:apply-templates select="text()" />
    <xsl:apply-templates select="*[not(name()='title')]" />
  </fo:block>
</xsl:template>
```

Cependant, tous les éléments texte non encapsulés dans des balises enfant de la balise `<example>` seront placés en tête de l'exemple, avant les éléments encapsulés, même s'ils sont placés après dans le fichier source DITA XML.

Le fichier PDF affichera l'exemple structuré comme suit :

Voici mon exemple de chemin XPATH :Texte non encapsulé situé après un nœud enfant.

```
ancestor-or-self
```

Il faut alors utiliser la syntaxe *pipe* (condition booléenne *ou*) pour modifier le chemin XPATH [67] comme suit :

```
<xsl:apply-templates select="text()|*[not(name()='title')]" />
```

Le résultat final sera :

```
<xsl:template match="*[contains(@class,' topic/example ')]">
  <fo:block>
    <xsl:apply-templates select="text()|*[not(name()='title')]" />
  </fo:block>
</xsl:template>
```

Le fichier PDF affichera l'exemple structuré comme suit :

Voici mon exemple de chemin XPATH :

```
ancestor-or-self
```

Texte non encapsulé situé après un nœud enfant.

XSL-FO : insérer automatiquement un titre pour les exemples

Par défaut, DITA-OT n'insère pas automatiquement dans les fichiers PDF le texte *Exemple :* devant le titre d'un exemple contenu entre balises DITA XML `<example>`. La syntaxe XSL-FO [68] offre cependant cette possibilité.

Supposons que le code source d'un de vos fichiers DITA XML soit le suivant :

67. http ://fr.wikipedia.org/wiki/XPath
68. http ://fr.wikipedia.org/wiki/XSL-FO

```
<example>
  <title>
    XSL-FO
  </title>
  Voici mon exemple de chemin XPATH :
  <codeblock>
    ancestor-or-self
  </codeblock>
</example>
```

Vous souhaitez que le fichier PDF généré affiche l'exemple structuré comme suit :

Exemple : XSL-FO

Voici mon exemple de chemin XPATH :

```
ancestor-or-self
```

et que si l'exemple ne contient pas de titre, il soit structuré comme suit :

Exemple :

Voici mon exemple de chemin XPATH :

```
ancestor-or-self
```

Par défaut, cependant, ce contenu sera structuré comme suit dans le PDF par DITA-OT :

XSL-FO

Voici mon exemple de chemin XPATH :

```
ancestor-or-self
```

Il est toujours possible d'entrer le texte entre les balises `<example>`, mais XSL-FO offre une manière de procéder plus élégante et structurée.

Insérer automatiquement une variable de texte avant le titre des exemples

1. Remplacez dans la feuille de style `plugins/org.dita.pdf2/xsl/fo/commons.xsl` (sous DITA-OT 1.7.) le template suivant :

```
<xsl:template match="*[contains(@class,' topic/example')]/*
[contains(@class,' topic/title ')]>
  <fo:block xsl:use-attribute-sets="example.title>
    <xsl:call-template name="commonattributes"/>
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

   par le code suivant :

```
<xsl:template match="*[contains(@class,' topic/example ')]>
  <fo:block xsl:use-attribute-sets="example.title>
    <xsl:call-template name="insertVariable>
    <xsl:with-param name="theVariableID"
    select="'my-example-text'"/>
    </xsl:call-template>
    <xsl:apply-templates select="title"/>
  </fo:block>
  <fo:block>
  <xsl:apply-templates
  select="*[not(contains(@class, ' topic/title'))]
    |text()|processing-instruction()"/>
  </fo:block>
</xsl:template>
```

2. Définissez dans les fichiers contenant les variables de langue, tels que `plugins/org.dita.pdf2/cfg/common/vars/fr.xml`, les variables de texte à insérer automatiquement, par exemple :

```
<variable id="my-example-text>Exemple :</variable>
```

Pour obtenir un comportement homogène, vous devez désactiver ce traitement pour les exemples des types de *topics* spécifiques (*task*, notamment).

Générer un PDF avec DITA Open Toolkit sous GNU/Linux

Ce didacticiel DITA XML est destiné à vous guider dans la mise en place et l'utilisation de la chaîne de publication DITA-OT dans un environnement GNU/Linux (Ubuntu ou Debian).

Prérequis

— Ubuntu ou Debian sur une machine physique ou virtuelle avec le mot de passe administrateur,
— connexion Internet.

1. Téléchargez et décompressez l'archive DITA-OT :

```
$ export REPO="https://github.com/dita-ot/dita-ot"
$ wget $REPO/releases/download/2.1/dita-ot-2.1.0.tar.gz
$ tar -xzvf dita-ot-2.1.0.tar.gz
```

2. Générez votre premier PDF :

```
$ cd dita-ot-2.1.0
$ dita -f pdf -i samples/taskbook.ditamap
```

Félicitations, vous avez compilé votre premier projet DITA XML ! Le fichier PDF généré est `out/taskbook.pdf`. Vous pouvez maintenant compiler d'autres projets en ignorant les étapes 1 et 2.

Générer un PDF avec DITA Open Toolkit (Windows)

Ce didacticiel DITA XML est destiné à vous guider dans la mise en place et l'utilisation de la chaîne de publication DITA-OT dans un environnement Windows (testé sur Windows XP).

Prérequis

— Connexion Internet

1. Téléchargez Java [69], puis lancez le programme d'installation.
2. Téléchargez DITA Open Toolkit 1.5.4 [70] sur le bureau, puis décompressez `DITA-OT1.5.4_full_easy_install_bin.zip`.
3. Sélectionnez *Exécuter* dans le menu *Démarrer*, collez la commande suivante, puis appuyez sur **Entrée** :

```
cmd
```

   Un terminal apparaît.
4. Collez la commande suivante dans le terminal :

```
set full=DITA-OT1.5.4_full_easy_install_bin
cd Bureau\%full%\DITA-OT1.5.4
```

5. Collez la commande suivante :

```
startcmd.bat
```

   Un nouveau terminal apparaît.
6. Collez la commande suivante dans le nouveau terminal :

```
$ java -jar lib/dost.jar /i:samples/taskbook.ditamap \
/outdir:. /transtype:pdf2
```

   Cette commande génère un fichier PDF à partir d'un projet DITA XML d'exemple.

   Félicitations, vous avez compilé votre premier projet DITA XML ! Vous trouverez le fichier cible `taskbook.pdf` dans le répertoire `Bureau\%full%\DITA-OT1.5.4`. Vous pouvez maintenant compiler d'autres projets en ignorant les étapes 1 et 2.

69. http ://java.com/fr/download/manual.jsp ?locale=fr
70. http ://sourceforge.net/projects/dita-ot/files/DITA-OTStableRelease/DITAOpenToolkit1.5.4/DITA-OT1.5.4_full_easy_install_bin.zip/download

Gérer les projets de documentation multilingues DITA XML

DITA XML est un formidable format pour gérer les projets de documentation. Pour les projets multilingues, cependant, le rédacteur technique doit créer un fichier *ditamap*, qui contient la structure de table des matières des documents, par version. Ceci entraîne un risque d'erreurs et d'incohérences. Heureusement, une méthodologie appropriée et un script d'automatisation destiné à la chaîne de publication DITA-OT remédient à ce problème.

Méthodologie de gestion des projets de documentation multilingues DITA XML

1. Le fichier *ditamap* ne doit pas comporter de section *navtitle*, qui contient un titre en toutes lettres, au lieu d'extraire le titre de la section DITA XML correspondante, et est donc propre à chaque langue.
2. Dès le début de votre projet DITA XML, placez les fichiers de contenu DITA XML dans un sous-répertoire spécifique à la langue dans laquelle il est initialement rédigé.
   Par exemple :
   — product
   — en_US
   — images
   — tasks
   — topics
   et non :
   — product
   — images
   — tasks
   — topics
3. Remplacez dans le fichier *ditamap* toutes les occurrences du nom du répertoire propre à la langue par une chaîne unique provisoire.
   Par exemple, utilisez la chaîne *@language-code@* :

```
<topicref href="@language-code@/topics/managing-rights.dita"/>
```

   et non :

```
<topicref href="en_US/topics/managing-rights.dita"/>
```

4. Pour générer les fichiers cibles, vous pouvez maintenant :
   (a) modifier dans le fichier `demo/fo/build.xml` le paramètre `default.locale`,
   (b) remplacer dans le fichier *ditamap* la variable de langue par le nom du répertoire de langue,
   (c) modifier le paramètre de langue `xml:lang` dans le fichier *ditamap* et dans les fichiers de contenu DITA XML,
   (d) pour les fichiers cibles PDF, modifier les dimensions de page (A4 ou US letter, par exemple) selon la langue,
   (e) générer les fichiers cibles,
   (f) rétablir les valeurs initiales dans les fichiers sources.

Heureusement, un script Bash (GNU/Linux) simple permet d'automatiser cela.

Prérequis

— Vous avez installé DITA-OT.
— Votre projet DITA XML ne comporte qu'un fichier *ditamap*.
— Vos fichiers de contenu DITA XML ont l'extension `.dita`.
— Les noms des répertoires des versions linguistiques correspondent aux codes de langues supportés par Dita Open Toolkit (`fr_FR` ou `en_US`, par exemple).
— Vos fichiers de contenu DITA XML se trouvent dans des sous-répertoires des répertoires des versions linguistiques (par exemple, dans `fr_FR/tasks/` et `fr_FR/topics/`).

Les valeurs supportées pour la dimension des pages PDF sont `fr_FR` (A4) et `en_US` (US letter). Ce script peut être bien entendu facilement adapté, ou inspirer un nouveau script.

Attention : Ce script est fourni sans garantie. Avant toute exécution de ce script, effectuez une sauvegarde de l'ensemble de votre projet DITA XML, fichiers de configuration inclus (par exemple sous un système de gestion de versions). Assurez-vous de pouvoir restaurer facilement le projet dans son intégralité en cas d'erreur ou de comportement inattendu.

Pour utiliser ce script :

1. Téléchargez le script de génération multilingue DITA XML [71] dans le répertoire contenant le fichier *ditamap* du projet.
2. Dans un terminal, placez-vous dans ce répertoire, puis entrez :

```
$ chmod +x dita2target.sh
```

3. Dans le terminal, entrez :

```
$ mkdir out
```

pour créer le répertoire qui contiendra les fichiers cibles.

4. Entrez :

```
$ ./dita2target.sh <fichier ditamap>  \
<nom du répertoire de langue> <format cible>
```

pour générer les fichiers cibles.

L'argument *format cible* accepte les valeurs gérées par DITA-OT.

Exemple

```
./dita2target.sh firewall.ditamap  en_US pdf2
```

Le fichier PDF `firewall.pdf` est alors généré dans le répertoire `out` (spécifié *en dur* dans le script).

Créer des documents différents à partir des mêmes sources DITA XML (texte conditionnel)

DITA XML offre un mécanisme de texte conditionnel. Ce mécanisme favorise la réutilisation du contenu source et évite la redondance des informations. Ce didacticiel aidera le rédacteur technique à utiliser ce mécanisme en quelques minutes.

Prérequis

— Vous avez installé DITA-OT dans le répertoire `DITA-OT1.5.4` sous GNU/Linux ou Windows.

1. Collez le code suivant dans un fichier et enregistrez ce dernier sous le nom de `texte-conditionnel.dita` dans le répertoire `DITA-OT1.5.4` :

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE topic PUBLIC "-//OASIS//DTD DITA 1.2 Topic//EN"
"/usr/share/dita-ot/dtd/technicalContent/dtd/topic.dtd">
<topic id="exemple-topic" xml:lang="fr-fr">
  <title>Utilisation du texte conditionnel</title>
  <body>
    <hazardstatement>
      <messagepanel audience="electriciens">
        <typeofhazard>
          Danger pour les électriciens
        </typeofhazard>
        <consequence>
          Risque d'électrocution
        </consequence>
        <howtoavoid>
          Ne touchez pas les fils électriques.
        </howtoavoid>
      </messagepanel>
```

71. http ://www.redaction-technique.org/media/dita2target.sh

```
        <messagepanel audience="plombiers">
          <typeofhazard>
            Danger pour les plombiers
          </typeofhazard>
          <consequence>
            Risque de noyade
          </consequence>
          <howtoavoid>
            Ne plongez pas dans la piscine.
          </howtoavoid>
        </messagepanel>
      </hazardstatement>
      <p>
       Tout contenu placé entre balises ne comportant pas de
       valeur <i>audience</i> exclue dans un fichier
       <i>.ditaval</i> est publié dans les documents
       destinés aux plombiers et aux électriciens.
    </p>
    </body>
  </topic>
```

Ce code contient des balises DITA XML contenant des valeurs *audience* différentes : nous allons exclure le contenu d'une de ces deux balises lors de la génération du fichier cible en utilisant la clé *audience*.

2. Collez le code suivant dans un fichier et enregistrez ce dernier sous le nom de `texte-conditionnel.ditamap` dans le répertoire `DITA-OT1.5.4` :

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE bookmap PUBLIC "-//OASIS//DTD DITA BookMap//EN"
"/usr/share/dita-ot/dtd/bookmap/dtd/bookmap.dtd">
<bookmap id="texte-conditionnel">
  <booktitle>
    <mainbooktitle>
      Exemple de texte conditionnel
    </mainbooktitle>
  </booktitle>
  <chapter href="texte-conditionnel.dita"/>
</bookmap>
```

3. Collez le code suivant dans un fichier et enregistrez ce dernier sous le nom de `electriciens.ditaval` dans le répertoire `DITA-OT1.5.4` :

```
<?xml version="1.0" encoding="UTF-8"?>
<val>
  <prop att="audience" val="electriciens" action="include"/>
  <prop att="audience" val="plombiers" action="exclude"/>
</val>
```

4. Collez le code suivant dans un fichier et enregistrez ce dernier sous le nom de `plombiers.ditaval` dans le répertoire `DITA-OT1.5.4` :

```
<?xml version="1.0" encoding="UTF-8"?>
<val>
  <prop att="audience" val="electriciens" action="exclude"/>
  <prop att="audience" val="plombiers" action="include"/>
</val>
```

5. Ouvrez un terminal et entrez la commande suivante dans le répertoire `DITA-OT1.5.4` :

```
$ java -jar lib/dost.jar /i:texte-conditionnel.ditamap \
/filter:electriciens.ditaval /outdir:. /transtype:pdf2
```

Ouvrez le fichier `texte-conditionnel.pdf` ; il contient des informations destinées :
— aux plombiers et aux électriciens,
— uniquement aux électriciens.

6. Ouvrez un terminal et entrez la commande suivante dans le répertoire `DITA-OT1.5.4` :

```
$ java -jar lib/dost.jar /i:texte-conditionnel.ditamap \
/filter:plombiers.ditaval /outdir:. /transtype:pdf2
```

Ouvrez le fichier `texte-conditionnel.pdf` ; il contient des informations destinées :

— aux plombiers et aux électriciens,
— uniquement aux plombiers.

DITA Open Toolkit : afficher les références croisées dans les PDF

Les références croisées sont un élément important d'une documentation technique bien structurée. Elles permettent à l'utilisateur de naviguer facilement dans les briques d'information et sont un élément crucial de l'utilisabilité du document final. DITA-OT les gère très bien, à condition d'effectuer quelques réglages.

Vous avez placé des balises *related-links* correctement formatées dans vos fichiers de contenu DITA XML, ou mieux, une reltable [72] dans votre structure de table des matières *ditamap* (la *reltable* permet de décontextualiser votre contenu et donc de mieux le réutiliser). Vous lancez votre commande de génération du PDF et, mauvaise surprise, aucune section *Voir aussi* [73] n'apparaît dans le fichier cible ! Vous essayez alors de générer une version HTML de votre contenu et là, votre section *Voir aussi* est bien présente. DITA-OT ne supporterait-il pas les références croisées dans les PDF ?

Fort heureusement, non. Par défaut (allez savoir pourquoi), les références croisées ne sont pas générées dans les PDF par DITA-OT. Pour les afficher, attribuez la valeur *no* à la variable *disableRelatedLinks* du fichier `demo/fo/build_template.xml`. Si vous utilisez *ant*, il vous faudra également passer le paramètre *args.fo.include.rellinks=all* comme suit :

```
$ ant -Dargs.input=samples/sequence.ditamap -Doutput.dir=out/ \
-Dtranstype=pdf2 -Dargs.fo.include.rellinks=all
```

Afficher un index dans un PDF (mais pas sous DITA Open Toolkit)

Tout n'est pas parfait sous DITA-OT, le moteur de publication libre DITA XML. Vous avez méticuleusement inséré vos entrées d'index dans vos fichiers de contenu DITA XML. Vous générez une sortie PDF et l'index n'apparaît pas. Un message d'erreur de la compilation vous indique que, hélas, FOP ne supporte actuellement pas la génération des index.

Face à cette situation, vous avez quatre solutions :

— attendre que FOP supporte les index ; sans date de disponibilité, ce choix sera difficile à faire accepter par votre direction ;
— abandonner DITA XML ; avouez que ce serait dommage de renoncer aux formidables gains de productivité que permet ce format ;
— renoncer à afficher l'index dans le PDF ; les arguments en faveur d'un tel choix ont un certain poids : les index sont difficiles à maintenir et offre un surplus d'utilisabilité discutable dans un document qui ne sera consulté que marginalement sous forme imprimée ;
— abandonner DITA-OT et se tourner vers une solution propriétaire ; les logiciels non open-source, XMetal, par exemple, on souvent recours au moteur de publication XEP de RenderX qui lui, supporte parfaitement les index.

Le problème de l'index n'est donc pas un obstacle à l'adoption de DITA XML. Si votre support final est un document imprimé, les solutions existent. S'il s'agit d'un format électronique, l'absence d'un index est largement compensée par la fonction de recherche en plein texte.

Utiliser l'IDE nXML pour DITA XML

Le mode nXML propose de valider en temps réel les documents XML DocBook, XHTML ou autres. Plus la peine de connaître le schéma XML par cœur : votre éditeur de texte vous propose l'autocomplétion des balises XML selon le contexte. Il ne supporte cependant pas DITA XML par défaut. Ce didacticiel vous permettra d'utiliser ce mode Emacs pour DITA XML.

Prérequis

— Emacs
— La structure de répertoires de votre projet de documentation DITA XML doit être la suivante :
  — répertoire de langue
    — concepts
    — faq
    — reference
    — tasks

72. http ://docs.oasis-open.org/dita/v1.0/langspec/reltable.html
73. Pour des raisons de « décontextualisation », et pour se donner la possibilité de réutiliser le contenu ailleurs, les références croisées ne sont pas placées dans le corps du texte, mais en fin de section, dans une rubrique dédiée.

    — topics

où <*répertoire de langue*> a la valeur *en_US*, ou *fr_FR*, etc.

— Les instructions de ligne de commande sont conçues pour GNU/Linux ; elles doivent être adaptées pour être utilisées dans un autre environnement.

1. Effectuez une sauvegarde de l'ensemble de votre projet de documentation DITA XML.
2. Ouvrez un terminal et collez la suite de commandes suivante :

```
$ export THAI="http://www.thaiopensource.com/download"
$ export RED="http://www.redaction-technique.org/media"
$ cd && \
wget $THAI/nxml-mode-20041004.tar.gz && \
tar xzvf nxml-mode-20041004.tar.gz && \
wget $RED/nxml-mode-environmment.txt && \
cp .emacs .emacs.bak && \
cat .emacs | sed '$a\' > .emacs.tmp && \
mv .emacs.tmp .emacs && \
cat nxml-mode-environmment.txt >> .emacs && \
rm  nxml-mode-environmment.txt
```

Note : Si un message vous avertit que le fichier `.emacs` n'existe pas, collez les commandes suivantes, puis recommencez l'opération :

```
$ cd && touch .emacs
```

Cette suite de commandes :
— télécharge et décompresse le mode nXML,
— crée une copie de sauvegarde du fichier `.emacs` (`.emacs.bak`),
— écrit les variables d'environnement du mode nXML dans le fichier `.emacs`.

3. Téléchargez l'archive des schémas RelaxNG pour DITA XML [74] dans le répertoire racine de votre projet de documentation DITA XML.
4. Placez-vous dans le répertoire racine de votre projet de documentation DITA XML, puis collez la commande suivante :

```
$ tar xzvf rnc.tar.gz
```

Cette commande crée un répertoire `rnc` de même niveau que le <*répertoire de langue*>.

5. Téléchargez l'archive des fichiers schemas.xml [75] dans le répertoire racine de votre projet de documentation DITA XML, puis collez la suite de commandes ci-dessous en remplaçant <*répertoire de langue*> par la valeur appropriée, *en_US*, ou *fr_FR*, par exemple. Répétez cette étape pour tous vos répertoires de langue.

```
$ export DIR="schemas.redaction-technique.org"
$ tar xzvf $DIR.tar.gz && \
cd <répertoire de langue> && \
cp ../$DIR/concepts/schemas.xml concepts/ && \
cp ../$DIR/faq/schemas.xml faq/ && \
cp ../$DIR/reference/schemas.xml reference/ && \
cp ../$DIR/tasks/schemas.xml tasks/ && \
cp ../$DIR/tasks/schemas.xml tasks/ && \
cp ../$DIR/topics/schemas.xml topics/ && \
rm -rf ../$DIR/
```

Vos répertoires de langue doivent maintenant comporter les fichiers `schemas.xml` appropriés :
— fr_FR
    — concepts
        — schemas.xml
    — concepts
    — schemas.xml
    — faq
    — schemas.xml
    — reference
    — schemas.xml
    — tasks

74. http ://www.redaction-technique.org/media/rnc.tar.gz
75. http ://www.redaction-technique.org/media/schemas.redaction-technique.org.tar.gz

— schemas.xml
— topics
— schemas.xml

6. Ouvrez un fichier de contenu DITA XML (`.dita`) avec Emacs. La syntaxe DITA XML apparaît en couleurs. Les endroits où le schéma n'est pas respecté sont soulignés en rouge.
7. Pour insérer une nouvelle balise entrez <, puis appuyez sur Ctrl+Entrée. La liste des balises possibles apparaît.
8. Sélectionnez une balise, puis appuyez sur Entrée. Appuyez sur Ctrl+Entrée pour afficher la liste des attributs autorisés.
9. Pour insérer une balise fermante après du texte, entrez </, puis appuyez sur Ctrl+Entrée.

Accélérer sa saisie avec le mode Predictive pour Emacs

Ce didacticiel mode Predictive pour Emacs est destiné à vous guider dans la mise en place et l'utilisation du mode Emacs d'aide à la rédaction et d'autocomplétion des mots anglais et français Predictive dans un environnement GNU/Linux (en l'occurrence, Debian).

1. Installez make et texinfo :

```
$ sudo aptitude install make texinfo
```

2. Téléchargez Predictive [76].
3. Décompressez l'archive Predictive :

```
$ tar xzvf predictive-0.23.13.tar.gz
```

4. Placez-vous dans le répertoire `predictive` :

```
$ cd predictive
```

5. Compilez *predictive* :

```
$ make
```

6. Installez *predictive* :

```
$ sudo make install
```

7. Insérez le code suivant dans le fichier `.emacs` :

```
;; predictive install location
     (add-to-list 'load-path "~/.emacs.d/predictive/")
     ;; dictionary locations
     (add-to-list 'load-path "~/.emacs.d/predictive/latex/")
     (add-to-list 'load-path "~/.emacs.d/predictive/texinfo/")
     (add-to-list 'load-path "~/.emacs.d/predictive/html/")
     ;; load predictive package
     (require 'predictive)
```

8. Lancez Emacs, puis appuyez sur Alt+X et entrez :

```
predictive-mode
```

76. http ://www.dr-qubit.org/emacs.php#predictive-download

5. Contact

Vous pouvez me contacter *via* mes profils LinkedIn [77] et Viadeo [78].

77. http ://fr.linkedin.com/in/carrereolivier
78. http ://fr.viadeo.com/fr/profile/olivier.carrere