Tkacz: un système de gestion de connaissances

Thibault Polge*

Révision: 8 novembre 2014 à 12:27.

Résumé Ce document présente les fonctions fondamentales d'un outil d'organisation de données en recherche en histoire. Ce logiciel, en cours de conception, est écrit dans le cadre de ma thèse, c'est-à-dire qu'il vise à couvrir tout particulièrement les besoins d'un historien des sciences qui travaille sur la période moderne.

*Doctorant contractuel chargé d'enseignement en philosophie (PhiCo/EXeCO, ED 280, EA 3562, Paris 1 Panthéon Sorbonne). thibault.polge@univ-paris1.fr

Première partie

Présentation générale

1 Introduction

Tkacz veut modéliser les différentes activités d'organisation des données trouvées en bibliothèque ou aux archives. Il présuppose que ces activités consistent principalement en la prise en notes d'informations trouvées sur des sources papier ou numérisées. Il suppose que la prise de notes sur des livres ou des documents d'archives s'organise selon un flux de travail à (au moins) deux temps : 1) La prise rapide de notes peu structurées, directement à la lecture du document. Ces notes peuvent porter sur le document lui-même s'il constitue une source primaire, mais aussi potentiellement renvoyer vers d'autres sources à consulter, apporter des informations sur un objet tiers (une personne, un évènement, un autre document…), etc. Elles peuvent donc prendre différentes formes, comme une référence bibliographique, une citation, un évènement ou toute autre forme. Ensuite, 2) l'organisation et la mise au propre des points importants, et notamment la recopie de certaines informations sur des fiches spécifiques. Ces informations peuvent être une liste de sources à consulter citées ou évoquées dans le document, des données nouvelles sur une personne, etc.

Tkacz a pour objectifs:

— De faciliter la manipulation de données sémantiquement structurées, sans que cette structure ne soit un obstacle à l'enregistrement d'informations complexes.
— De réduire la nécessité d'une reprise (temps 2) des informations acquises sur un document sur de nouveaux supports, en facilitant l'orga
— De maintenir vivant le graphe des relations entre informations, autrement dit de ne pas disposer que de fiches isolées mais toujours un « répertoire » vivant et organisé.

2 L'objet fondamental : la fiche

2.1 L'objet fiche

La **fiche** est l'unité atomique de Tkacz. Une fiche a un **type** (cf. **??**) ; la liste complète des types n'est pas fixée *a priori*. Chaque fiche a un **titre**, permet la prise de notes, et est identifiée par un numéro unique attribué séquententiellement à la création.

Le titre devrait être unique et identifie la fiche de façon pour l'utilisateur. Sa forme est par défaut libre, mais peut être imposée par certains types. Par exemple, une fiche d'entrée bibliographique produit son titre automatiquement (mais de façon configurable) par rapport aux données structurées qu'elle contient. Un tel titre peut prendre la forme:

The efficacy of AZT in the treatment of patients with AIDS. . . 1987
Fischl, Richman, Grieco *et al*. N Engl J Med

Automatiquement calculée à partir des noms des auteurs, de l'année et du titre (éventuellement, comme ici, abrégé) d'une entrée bibliographique. Le titre d'une fiche personne prendra une forme du type:

FOUCAULT, Michel 1926–1984

Des notes, écrites dans un format inspiré de Markdown.

2.2 Attributs et liens

Une fiche, à l'exception du type minimal (qui modélise réellement une feuille blanche avec un titre), contient un certain nombre d'attributs, qui sont les données sémantiques, et de relations.

2.2.1 Attributs

Les attributs sont les propriétés sémantiques qui décrivent l'objet modélisé par une fiche. Certains attributs sont des liens vers d'autres fiches. L'auteur d'un livre (dans une fiche de notice bibliographique) est manipulé comme un lien vers une fiche personne ; si cette fiche n'existe pas, elle est créée automatiquement avec les valeurs disponibles.

Il existe trois espèces d'attributs: - Il peut être un objet primitif, comme une chaîne, un nombre ou une date, ou un ensemble d'objets primitifs, comme une liste ou un ensemble.

Dans une entrée de type « notice bibliographique », le numéro d'édition est un tel attribut. Ils sont relativement rares.

— Il peut être d'un type complexe, lui-même composé d'autres attributs (voir **??**).

Dans une entrée de type « personne », le nom de la personne est un type complexe formé de chaînes qui distingue les composants du nom. Ce type a aussi du code qui lui permet de lire et de mettre en forme un nom.

— Il peut être un **lien** vers une autre fiche d'un type donné.

2.2.2 Liens et relations

Il y a deux façons de lier une fiche à une autre : le lien simple et l'expression d'une relation.

2.2.3 Le lien simple

est un « pointeur » vers la fiche cible. L'auteur d'un document, par exemple, est un lien vers une fiche personne, et pas une simple séquence de caractères. Le lien peut avoir un **corollaire**, c'est-à-dire que la liaison de A et B implique une relation d'une autre nature de B vers A. Dans l'exemple de l'auteur, le lien « a pour auteur » a pour corollaire « est auteur de ».

Le corollaire est généralement implicite. Il peut n'être fixable que depuis un seul des membres de la relation possible. Le lien d'auteur, par exemple, n'est manipulable que depuis la chose dont x est l'auteur, et pas depuis x.

2.2.4 La relation

est un attribut complexe, qui permet de lier des fiches entre elles de façon moins formelle et plus fine. Une relation a une **nature**, qui est l'équivalent du type d'une fiche. Dans certains cas, un lien peut être remplacé par une relation. Par exemple, dans un document historique, l'attribution de l'auteur peut être douteuse — « auteur probable » est une relation, car le lien simple ne suffit pas — il faut sans doute préciser la nature du doute, les différentes sources, etc.

Une relation peut aussi décrire des relations entre entités : « membre de » ou « ami de », « frère de », etc.

Une relation peut être réciproque ou non. Dans l'exemple qui précède, les deux dernières sont nécessairement réciproques. Une relation peut aussi avoir un **corollaire**. La relation `A est membre de B` (dans cet exemple, B est par exemple une personne morale) a pour corollaire « B a comme membre A »

Une relation est de type « cite » ou « évoque », qui permet de commenter un livre en le liant à ce qui fait son objet.

2.3 La notion de type

2.4 Les types de fiche standards

La description formelle de ces types est donnée en **??**.

2.4.1 Simple

Il s'agit d'une fiche a minima : titre et notes.

2.4.2 Référence bibliographique

Ce type de fiche peut représenter différents types d'objets entrant dans une bibliographie, qu'il s'agisse de sources primaires ou de sources secondaires, de supports imprimés ou audiovisuels.

2.4.3 Personne (physique ou morale)

3 L'organisation des fiches : la taxinomie

Cette organisation se fait sous la forme de **taxinomies**. Une taxinomie est une structure hiérarchique, comparable à un système de fichiers, dans les entrées duquel les fiches prennent place. Contrairement à un système de fichiers, par définition unique, plusieurs taxinomies peuvent cohabiter, aucune fiche ne réside nécessairement dans une taxinomie quelconque, et une fiche peut se trouver associée à plusieurs taxons.

Une taxinomie peut être créee manuellement ou automatiquement. Les types de fiche sont eux-mêmes des taxons, et créent une taxinomie automatique et non modifiable (ie, associer une fiche à un type revient à la faire rentrer dans une taxinomie)

Les taxons peuvent se voir associés un certain nombre de règles d'affectation:

— un taxon peut contenir la totalité du contenu (ie, les fiches) de ses enfants. Comme dans une taxinomie biologique, toutes les sous-classes de mammifères (theria et prototheria) *sont* des mammifères ; ou au contraire ne contenir que ce qui y est explicitement ajouté.

— Un taxon peut se voir ajouter directement (manuellement) du contenu, ou ne le recevoir que par affectation automatique (par le contenu de ses enfants ou d'autres moyens).

— Un taxon peut être incompatible avec un autre, ie la présence d'une fiche dans ce taxon rend impossible sa présence dans un autre. Par exemple, un mammifère ne peut pas être un poisson ; ou au contraire un taxon peut en impliquer automatiquement un ou plusieurs autres.

Un taxon peut fixer des règles pour lui-même et/ou ses enfants à un niveau n ou aux niveaux n à m.

Le lien d'une fiche à un taxon est lui-même une fiche, qui peut donc être commenté.

Le fait qu'une taxinomie est forcément hiérarchique n'implique pas nécessairement qu'elle soit manipulée comme telle. Il est possible de créer des taxinomies de « tags » ou tous les tags sont au même niveau.

Les taxinomies ne sont pas fortement indépendantes ; elles sont gérées en interne comme un unique arbre hiérarchique.

4 Sélection et recherche

La sélection et la recherche utilisent le mécanisme de la taxinomie pour rechercher des notes. Chaque taxon peut être conçu comme un ensembl de fiches. Les expressions de recherche prennent la forme suivante:

```
[Publications] 'Michel Foucault' date < {3 jan 1950}
```

La recherche répond à une logique globalement ensembliste ; les opérations fondamentales de la théorie des ensembles (intersection, union, différence, différence symétrique) forment les opérateurs principaux du mécanisme de recherche.

À terme, il est prévu de faciliter l'usage de ce mécanisme de recherche par une interface graphique d'élaboration des requêtes et/ou une formulation des requêtes dans un langage formalisé proche du langage naturel.

4.1 Syntaxe

La totalité des opérateurs peuvent manipuler quatre types de propriétés, soit les trois types d'ensembles :

set Un ensemble de fiches ou de taxons (c'est pareil, un taxon n'est qu'un ensemble de fiches)
strset Un ensemble de chaînes.
attrset Un ensemble de noms d'attributs ou de relations.

Et un type complexe, repéré par les accolades, spécifiques à certain type d'attributs, par exemple les dates.

Il n'existe pas de type « fiche unique » ou « chaîne » : tout est un ensemble, qui peut ne contenir qu'un élément.

4.2 Opérateurs de groupement

[ ] Sélectionne un taxon par son nom : `[str]`.

Les ambiguités peuvent être résolues en donnant un parent du taxon au format `[parent/taxon]`, un parent plus lointain `[parent/.../taxon]` ou le nom de la taxinomie `[@taxinomie: parent/taxon]` ou une combinaison: `[@taxinomie: parent//taxon]`.

" " Sélectionne une fiche par son nom.

Une fiche peut être aussi sélectionnée directement par son numéro.

Ces deux premiers opérateurs utilisent la virgule comme séparateur. `[taxon1, taxon2]` est un ensemble de taxons, et donc renvoie une valeur de type `set`.

() Les parenthèses augmentent la priorité d'une expression (cf. **??**). Rien de très original. L'expression `A+B*C` sera évaluée implicitement comme l'union de `A` et de l'intersection de `B` et `C`. Avec des parenthèses telles que `(A+B)*C`, elle renverra l'intersection de `C` et de l'union de `A` et `B`.

4.3 Opérateurs binaires

: `attribut:set` ou `attribut:str` retourne l'ensemble des fiches dont l'attribut `a` a au moins une valeur dans `expr`.
:: `a::expr` retourne l'ensemble des fiches dont l'attribut `a` a toutes ses valeurs dans expr.
< Strictement inférieur à, pour les attributs où cela à un sens.
<= Inférieur ou égal où, pour les attributs où cela à un sens.
> Strictement supérieur à, pour les attributs où cela à un sens.
>= Supérieur ou égal où, pour les attributs où cela à un sens.
& intersection ($\cap$). c'est l'opérateur implicite. `A&B` ou `AB` retournent l'intersection des taxons `A` et `B`. L'intersection est symétrique ; `A&B = B&A`
| union ($\cup$). `A|B` retourne l'ensemble des fiches de `A` et de `B`. L'union est symétrique ; `A|B = B|A`

- différence (\). `A-B` renvoie l'ensemble `A` moins l'ensemble `B`. La différence n'est pas symétrique (l'intersection de `A-B` et `B-A` est vide : $(A \backslash B) \cap (B \backslash A) = \emptyset$.)

^ différence symétrique (△). `A^B` renvoie la totalité des fiches de `A` ou `B` mais pas les deux. `A^B = (A+B)-(B*A)`.

4.4 Opérateurs unaires

- Inverse. `-expr` renvoie la totalité des fiches non contenues dans `expr`. `-expr = {*-expr}`

: équivalent approximatif de : sans spécifier le nom de l'attribut : `*:expr` renvoie toutes les fiches liées à expr. `* :expr` doit être un ensemble.

:* `expr`

4.5 Autres termes

***** L'ensemble des fiches du dépôt.

[ ] Délimite une construction complexe spécifique à un type de données, par exemple une date.

" " Délimite une chaîne pour la recherche en plein texte.

Isolé, il renvoie l'ensemble des fiches dans lesquelles ce texte a été trouvé ; sinon il peut être utilisé pour la recherche par attributs (`:`, `::`)

**** Caractère d'échappement.

4.6 Priorité

Les expressions sont évaluées avec les priorités suivantes. (1) et (2) précisent qu'il s'agit, respectivement, de la version unaire ou binaire d'un opérateur.

Priorité	0	1	2	3	4	5
Opérateurs	Opérateurs de groupement	Opérateurs unaires	&	|	- (2)	/

4.7 Synonymie

Pour des raisons de clarté, les opérateurs natifs ont les synonymes suivants:

Opérateur	:	::	&	|	-	\	*
Synonymes	in		and	or	andnot	xor	all

Les versions localisées pourraient implémenter ces synonymes dans leur langue.

5 Opérations du logiciel

5.1 Mode de consultation

5.2 Mode de recueil

Le mode de recueil suppose une source d'information (ou plusieurs ?) représentée par une fiche, d'où on recueille un certain nombre de données sur elle-même ou sur d'autres objets, représentés par d'autres fiches, qui peuvent être déjà existantes ou créées à la volée.

Il verrouille sur la source d'origine et associe toute information entrée à cette source.

6 Fonctions étendues

6.1 Export de bibliographies

Les fiches de types « référence bibliographiques » doivent pouvoir être exportées dans des formats manipulables par un gestionnaire de bases de données, en réduisant la complexité intrinsèque à Tkacz.

Deuxième partie

Utilisation et interface utilisateur

La fenêtre principale de Tkacz se présente de différentes façons selon le mode en cours. Le mode de consultation (par défaut, ou accessible via Meta 〉 Data (⌘+⌥+A,C)

7 Syntaxe des fiches

La syntaxe est dérivée de Markdown et d'autres langages de formatage rapide. Il y a néanmoins quelques différences dans le comportement standard du parser:

— Tkacz utilise les _*tirets de soulignement*_ pour la mise en italiques, et ***une seule étoile*** pour le gras.

— Un paragraphe indenté de quatre espaces ou plus n'est pas traité comme un bloc de code, mais est simplement indenté d'un niveau. Les blocs de code utilisent exclusivement la syntaxe «grillagée», en encadrant le bloc de ~~~ et en indiquant éventuellement le langage après la première série de tildes.

— La syntaxe des liens est supprimée.

— La syntaxe des blocs de description est modifiée.

— Le format des citations permet d'en préciser l'origine.

— Tkacz pourrait gèrer plusieurs tables des matières dans un seul document.

De plus, un certain nombre d'extensions spécifiques sont ajoutées.

7.1 Association clé-valeur

Toute fiche commence généralement par un préambule qui expose formellement son contenu. Ce préambule prend la forme

```
:name
    :first Michel
    :middle Paul :unused
    :last Foucault

:name Fuchs :aka
    Dans [Guibert 1980]

:birth 15 octobre 1926 @ Poitiers
:death 25 juin 1984 @ Paris
```

7.2 Liens et relations

```
<> friend [Didier.Éribon]
<> founder GIP
```

Troisième partie

Implémentation

8 Format de stockage

Un dépôt Tkacz est un répertoire du système de fichiers, qui n'est pas destiné à être manipulé par l'utilisateur. Il peut être présenté comme un bundle (sur OS X) ou stocké zippé (sur les autres systèmes) pour empêcher toute manipulation destructrice.

8.1 Format d'un dépôt

Un dépôt combine un dépôt git [1] et une base de données SQLite qui sert de cache. Un dépôt vide a donc la structure suivante:

```
/
├── .git
│   └── . . .
└── .tkacz
    ├── db ..... Fichiers SQLite
    │   └── . . .
    └── manifest ..... Déclarations Tkacz de base (Yaml)
```

Le fichier `.tkacz‣manifest` est une représentation JSON de la structure détaillée ci-dessous. évidemment

```
{
    "tkacz": {
        "frameworkVersion": [0,1,0],
        "formatVersion":    [0,1,0],
        "schemaId":         "core",
        "schemaVersion":    [0,1,0]
    },
    "repository": {
        "uuid":             "03095EEF-6C87-430B-A00E-440616196C31",
        "name":             "As set by the user"
    },
    "core": {}
}
```

Les clés `tkacz`, `repository` et `core` (options du schéma standard) et de façon générale toutes les clés racines vérifiant `[a-zA-Z]+[a-zA-Z0-9_]*` sont réservées, les extensions ou les schémas tiers peuvent inscrire leur paramétrage dans des clés au format `com.domaine.nom` (à la Java).

8.2 Stockage des fiches

Les fiches sont sauvegardées comme des fichiers gérés par Git, dans des dossiers correspondant à leurs types, par exemple `person‣collective`. Leur nom est un numéro attribué séquentiellement pour le dépôt entier (indépendemment du type donc). Tous les noms vérifiant `[0-9]+` sont donc réservés pour les fiches au niveau du suivi des versions.

Le dépôt n'utilise qu'une seule branche, `master`.

1. Dans ce document, le mot «dépôt» seul fait *toujours* référence à un dépôt Tkacz.

L'usage de Git fait ici est assez particulier: chaque message de commit décrit l'état des références, c'est à dire la fiche qui sert de source et la position dans cette fiche.

Quand Tkacz contrôle l'éditeur de texte, il commite automatiquement les changements, sans contrôle possible de l'utilisateur, dans les situations suivantes:

1. La saisie ou l'effacement se poursuit, mais la position du curseur a changé. [`move`]
2. L'utilisateur commence à saisir du texte après en avoir effacé. [`insert`]
3. L'utilisateur commence à effacer du texte après en avoir saisi. [`delete`]
4. La référence ou l'emplacement dans la référence a changé. [`insert`]
5. L'utilisateur coupe du texte. [`cut`]
6. L'utilisateur colle du texte (au format Tkacz, sinon `insert`). [`paste`]

Si un éditeur externe est utilisé, un commit est réalisé à chaque modification du fichier[2].

8.2.1 Format des messages de commit

Commits sur modification Le format d'un message de commit en cours d'édition est une représentation JSON (minimisée) de la structure:

```
{
    "operation": "type",
    "using":     234,
    "position": {
        "page": "x"
    }
}
```

`operation` décrit l'action de l'utilisateur qui justifie le commit.

Avec cette réserve que le format de `position`, s'il est obligatoirement un dictionnaire, n'est pas déterminé par avance, et dépend du type de fiche. En imaginant des notes prises à la volée sur un enregistrement audio, `position` pourrait avoir un format du type:

```
{
    "position": {
        "tape": 0,
        "time": [0, 12, 34]
    }
}
```

Si une fiche est modifiée sans référence ouverte, le message de commit ne contient que la clé `operation`.

Commits vides Des messages supplémentaires sont produits à l'ouverture et à la fermeture d'une ressource, avec des commit vide:

```
{
    "opening": 123
}
```

2. Il est plausible qu'il soit en fait impossible d'utiliser un éditeur pour lequel Tkacz ne puisse pas suivre chaque modification. En effet, la fonctionnalité fondamentale de suivi des ajouts et modifications et de leur rattachement à des emplacements précis des sources nécessite un suivi extrêment précis des modifications. Autrement dit, seul peut sans doute être utilisé un éditeur qui sauvegarde à chaque caractère modifié.

et:

```
{
    "closing": 123
}
```

Un dernier type de message permet d'associer une donnée déjà saisie à une autre ressource:

```
{
    "link": 72,
    "from": [132, 21],
    "to":   [135, 12]
}
```

`from` et `to` sont ici des paires ligne/colonne.

8.2.2 Calcul des diffs, suivi de l'origine et déplacement de paragraphes

Le *déplacement* de blocs de texte est un problème sérieux que les diff classiques ne permettent pas de résoudre. Or, un bloc déplacé doit continuer à être associé à son origine. Plusieurs solutions semblent possibles:

— Commit automatique à chaque opération couper/copier/coller.

— "Lester" les copies/coupes avec les informations d'origine de la source et les porter dans le commit lors du collage. *Cette option exclut absolument d'utiliser un autre éditeur de texte que celui de Tkacz.*

8.3 Structure du cache

La totalité des données utiles se trouve dans les fiches elle-même, ce qui signifie que seul le dépôt git doit être copié pour cloner entièrement

9 Le système de types

Le mécanisme de types de Tkacz repose sur une correspondance terme à terme entre une hiérarchie de types et une hiérarchie d'objets instantiables. Ces deux hiérarchies sont définies comme suit:

Types		Objets	
Template (virtual)		Node	
EntityTemplate	PrimitiveTemplate<T>	—	—
	IntegerTemplate, StringTemplate…	Card	

À chaque instance d'un descendant d'Object correspond un objet Template du type correspondant (à une Primitive correspond un PrimitiveTemplate, etc.). L'objet Template:

— Initialise la structure de données de l'Object.

Quatrième partie

Annexes

A Définitions

Ce lexique décrit les termes employés dans la présent document, le lexique anglais équivalent pour l'implémentation, et le lexique anglais et français l'interface graphique quand ils divergent des termes retenus par ailleurs.

A.1 Termes employés dans ce document

%\begin{multicols}{2}

Attribut

Dépôt

Fiche

Lien Un lien est un type particulier d'attribut qui, au lieu d'être une donnée stockée en place, est un renvoi vers une autre fiche. L'attribut *auteur* d'une fiche *notice bibliographique* est un lien vers une fiche de type *personne*.

Nature L'équivalent pour une relation du type d'une fiche.

Relation {#relation}

Schéma

Taxinomie

Type %\end{multicols}

A.2 Termes anglais du code source

attribute : → *attribut*.
card: → *fiche*.
link: → *lien*.

nature : → *nature*.
relationship : → *relation*.
repository : → *dépôt*.

schema : A Schema implements a factory and a visitor.