Université Paris Diderot

Mémoire de master

Pour l'obtention du master Langues, Littératures et Civilisations Étrangères et Régionales (LLCER), spécialité Études japonaises, parcours Linguistique et didactique

Pour une modélisation de dictionnaires de japonais sous forme de graphe

Louis Lecailliez

dirigé par
Akiko Nakajima

22 septembre 2016

Table des matières

Chapitre 1

Introduction

1.1 Résumé

L'apprentissage du japonais est long et complexe pour un apprenant francophone. Puisque le vocabulaire, tout comme l'écriture, est radicalement différent de ce qu'il connait – exception faite des récents emprunts à l'anglais – il lui est nécessaire de consulter fréquemment des dictionnaires afin de se renseigner sur le sens et la lecture du vocabulaire et des caractères qu'il rencontre.

J'ai remarqué en utilisant pendant plusieurs années des ouvrages papiers et informatisés, et en programmant plusieurs de ces derniers, que ceux-ci pouvaient largement être améliorés. Hors contenu, qui nécessite un travail lexicographique classique, c'est en particulier au niveau des possibilités d'interactions que se trouve des pistes intéressantes d'amélioration.

Les dictionnaires ont déjà profité de l'informatisation, car celle-ci a permis de régler ce qui était probablement un des plus gros problèmes de la consultation de ces ouvrages, le temps de recherche. Pour autant, ce processus ne s'est pas accompagné d'une manière radicalement différente de faire les dictionnaires : la plupart des projets informatisés sont des transpositions de leur version papier. Leur contenu a été placé dans une base de données ou un fichier et une interface permet leur consultation sans apporter beaucoup plus à la consultation que la rapidité de recherche déjà évoquée.

Le présent mémoire se propose de repenser la création de dictionnaires numériques, en définissant un nouveau modèle de structuration des données et de détailler certains des avantages qu'il octroie. L'objectif visé est de pro-

poser un modèle qui permette d'améliorer significativement la navigation dans un dictionnaire et d'en regrouper plusieurs dans une seule application, fournissant ainsi dans un unique logiciel plusieurs types d'informations que pourrait rechercher un apprenant.

1.2 Résumé en japonais

日本語を勉強するフランス語の話者にとって日本語を学ぶ事は長くて複雑なものだ。語彙さらに文字がロマンス諸語とローマ字と異なるため（外来語以外に）しばしば言葉の意味を辞典で探さなければならない。

電子辞典は今まで書物のように作れていた。探す速さの他にコンピューターの能力を使えないため改善の可能性がありそうだ。

本論文では見出語にナビゲーションを大いに改善ために電子辞典の新しいグラフを使うデータモデルを提案する。

1.3 Contexte et objectifs

La démocratisation des solutions informatiques personnelles a permis de mettre les dictionnaires à portée de main de tout un chacun. Le changement de support, en passant de l'encombrant papier à de l'information numérique ne semble toutefois pas s'être accompagné d'une transformation des fonctions de ceux-ci. Ainsi, à de nombreux égards, il ne s'agit que d'une simple transposition, le dictionnaire électronique n'étant que le produit informatisé de sa version physique, où seul l'accès aux articles est radicalement modifié par l'augmentation de la vitesse de recherche de plusieurs ordres de grandeur.

Pourtant, à la manière dont le web, au moyen d'hyperliens, a modifié la conception et l'accès des textes, il est probablement possible d'utiliser les capacités offertes par l'informatique pour aller plus loin que ce qui est fait actuellement. C'est d'autant plus surprenant que l'encyclopédie, de nature proche du dictionnaire, a réussi à franchir ce pas comme le montre Wikipedia, dont la navigation d'article à article et l'ouverture du processus d'édition ont permis d'apporter au projet bien plus qu'une simple mise en ligne du contenu de ses prédécesseurs.

Il me semble intéressant de se pencher sur ce que pourrait être le dictionnaire électronique augmenté de fonctionnalités propres au numérique et de raisonner sur des exemples concrets, ici liés au japonais, afin d'explorer cette

voie. Les principes clefs qui semblent permettre de donner vie à cela sont : la navigabilité, la simplicité et l'extensibilité.

1.3.1 Navigabilité

Définissons le principe de navigabilité comme la capacité de pouvoir passer directement d'une fiche de dictionnaire à une autre, c'est-à-dire sans repasser par une recherche. La navigabilité est un concept puissant qu'on retrouve sur le web : c'est cette possibilité de pouvoir passer d'une page à une autre en un clic, de naviguer au travers des sites selon son appétence, qui en fait une grande partie de son attrait.

Dans le cadre d'un dictionnaire, il y a à première vue moins de liens évidents entre les objets qu'il contient. En dehors du traditionnel renvoi, qui est d'ailleurs dans certains dictionnaires déjà navigable, ne serait-il pas plaisant de pouvoir accéder d'un tapement de doigt aux mots sources d'un néologisme ? Ou de se rendre immédiatement sur la fiche d'un des synonymes listés ? Ou encore d'accéder à la traduction dans une autre langue d'un des sens listés dans la fiche d'un dictionnaire bilingue ?

Tout cela est probablement réalisable, mais nécessite un modèle de structuration de l'information adapté. Si l'existant ne le permet pas, c'est justement parce que le modèle de représentation informatique utilisé ne rend pas évident à créer ou utiliser les liens latents entre ces données.

1.3.2 Simplicité

Si le concept de navigabilité touchait à la façon de se servir du dictionnaire, c'est-à-dire son usage du côté de l'utilisateur final, le second concept qui me paraît important à la réussite d'un tel projet touche quand à lui au travail des lexicographes. Dans la mesure du possible, l'approche proposée doit être la plus simple possible.

Ceci pour plusieurs raisons. Tout d'abord, un modèle facilement compréhensible a plus de chance d'intéresser et de pouvoir être utilisé avec succès par des équipes extérieures. Le web fourni ici encore une illustration exemplaire de la nécessité de technologies simples : le web sémantique, qui se veut le successeur du web actuel, n'a toujours pas décollé en pratique à cause de la complexité des concepts [1] et des technologies monopolisées.

1. Dont certains sont pourtant très proche de ceux qui seront mobilisés dans ce mémoire.

Ainsi, si l'idée de modéliser des ressources lexicales sous forme de graphe, à destination des humains et des applications de traitement automatique des langues, est déjà décrite par [Spohr, 2012] dans son livre qui prolonge sa thèse de 2010 sur le même sujet, j'ai fait le choix opposé de ne pas me baser sur les technologies existantes du web sémantique. D'abord parce que ce serait mettre des détails d'implémentation avant le cadre théorique, mais surtout car cela permet de réfléchir au problème sans idées préconçues sur la façon de représenter certains phénomènes [2].

Mais en plus, il ne faut pas oublier que pour abstraites que soient les idées exposées ici, elles devront *in fine* être mises en pratique informatiquement pour pouvoir être utilisées et arriver dans les mains des utilisateurs finaux. Si la mise en oeuvre technique s'avère trop difficile car le modèle choisi est trop complexe, nous n'aurons finalement pas vraiment avancé sur l'amélioration effective des dictionnaires électroniques.

1.3.3 Versatilité et extensibilité

Enfin, le modèle doit permettre de représenter une partie conséquente des différents dictionnaires existants sans quoi son utilité serait de fait assez limitée et le but fixé non atteint. L'extensibilité du modèle peut être décrite sous plusieurs angles.

Il faut d'une part pouvoir ajouter des données de type hétérogène si on veut pouvoir fusionner les informations de plusieurs dictionnaires, ce qui est une opération nécessaire à la création d'un dictionnaire qui combine des informations lexicographiques en provenance de sources différentes.

D'autre part, c'est l'ajout même de nouvelles langues qui doit pouvoir être réalisé de manière aisée. En effet, il peut être intéressant de combiner plusieurs dictionnaires de langues distincts pour les utilisateurs qui en connaissent plusieurs, notamment car c'est une façon peu coûteuse de pallier à l'absence ou la faible quantité de données disponibles pour certaines langues. Enfin, cela devrait également permettre d'ajouter des dialectes facilement, ce qui est pour l'instant un aspect négligé des dictionnaires disponibles.

Ces deux dernières problématiques, qui sans être oubliées, ne seront pas décrites en détails. Les exemples fournis se concentreront sur le japonais.

2. Ainsi le concept de hiérarchie de classe n'est pas utilisé, aucun type n'est standardisé et la notion d'hyper-arc est introduite.

1.4 Plan du mémoire

Ce mémoire est divisé en trois parties, chacune structurée et ayant un but différent.

Puisque ce mémoire utilise des notions de différents domaines de la connaissance et de la technique (mathématiques, linguistique, informatique), la première partie s'attache à présenter les notions importantes qui pourraient être étrangères au linguiste spécialisé en japonais mais qui sont nécessaires à la bonne compréhension des deux parties suivantes.

La seconde partie est le coeur du mémoire : elle présente et détaille les constituants du modèle de données en graphe proposé.

La dernière partie est formée d'un ensemble d'applications du modèle à différents jeux de données lexicographiques. Non seulement cela a pour but d'illustrer comment le modèle peut être utilisé en pratique pour modéliser des dictionnaires ou des ouvrages assimilés, mais aussi de relever des limites au modèle rencontrées lors de sa mise en œuvre.

Chapitre 2

Rappels de lexicographie, mathématiques et informatique

Ce mémoire faisant usage de notions afférentes à différents champs de la connaissance humaine, il me semble nécessaire de rappeler, d'évoquer ou de discuter de celles qui seront principalement utilisées.

Ainsi des notions liées à nos objets de travail seront rappelées : dictionnaires, graphes et applications informatiques. Nous aborderons rapidement les dictionnaires (définition, structure et types existants), notamment ceux que l'on peut rencontrer en langue japonaise et nous listerons ceux que nous utiliserons dans les cas d'application.

2.1 Dictionnaires

2.1.1 Définition

Un dictionnaire est un « ouvrage didactique constitué par un ensemble d'articles dont l'entrée constitue un mot, indépendants les uns des autres et rangés dans un ordre déterminé, le plus souvent alphabétique. (Abréviation familière : dico.) » nous indique le Larousse en ligne [Larousse, 2016]. Bien que donnant un aperçu de ce qu'est un dictionnaire, cette définition comporte deux problèmes lorsqu'elle indique que les entrées sont constituées de mots. Pour un dictionnaire de langue française, le terme n'est pas assez précis, linguistiquement parlant, mais cela tient à la portée didactique à laquelle n'échappe pas le dictionnaire fournissant cette définition.

Le second problème nous concerne plus directement : un dictionnaire peut être une collection d'autres éléments que des mots. Typiquement en japonais, il existe des dictionnaires de caractères chinois (sinogrammes ou kanjis) qui sont tous aussi nécessaires à un apprenant que ne l'est un dictionnaire bilingue.

Nous allons donc aborder l'une après l'autre ces deux problématiques. Premièrement celle qui touche à la définition d'un « mot », car elle est nécessaire pour savoir quel objet est principalement manipulé par le modèle de structuration des données proposées. Dans un deuxième temps nous listerons les différents types de dictionnaires que l'on rencontre couramment lorsqu'on étudie le japonais, afin de définir plus précisément le type d'ouvrages que nous tenterons de représenter.

2.1.2 Mots et lexies

Le problème de la définition d'un mot est crucial pour la création de dictionnaires puisqu'elle conditionne ce qui y est effectivement recensé en tant qu'entrée principale. Sans plonger profondément dans cette problématique, l'*Introduction à la lexicologie explicative et combinatoire* (l'ILEC) [Mel'čuk *et al.*, 1995] donne la définition d'un concept intéressant : celui de **lexie**.

Celui-ci étant décrit comme « fort complexe », il n'est pas discouru en détails et fait l'objet des remarques suivantes : « Le concept de lexie est une formalisation et, simultanément, une généralisation de la notion de MOT [...] Pour le moment, il nous suffit de dire qu'une lexie ou unité lexicale, est soit un mot pris dans une acception bien spécifique (= lexème), soit encore une locution, elle aussi prise dans une acception bien spécifique (= phrasème) ».

L'entreprise des auteurs de ce livre est de définir les moyens scientifiques de parvenir à un dictionnaire de lexies. Du fait que les expressions idiomatiques figées, semi-figées ou les cooccurrences très fréquentes entre des mots sont nécessaires à la bonne compréhension et à l'utilisation d'une langue, il semble effectivement tout à fait pertinent de suivre ce même modèle dans l'optique de créer un dictionnaire à destination des japonisants et donc de ne pas se limiter à la nomenclature traditionnelle des dictionnaires, qui ne contiennent que des mots graphiquement isolés.

Mais contrairement aux langues européennes, le français notamment, le japonais comporte un objet lexicologique à part entière : le caractère chinois. On ne peut donc pas simplement envisager de ne référencer que des lexies,

du fait de la présence de ces caractères, qui constituent une difficulté à part entière de l'apprentissage de la langue, par ailleurs partagée avec le chinois. Le rassemblement dans un seul ouvrage des définitions de mots et de sinogrammes existe déjà dans certains dictionnaires ; on peut citer le Grand Ricci [Ricci, 2001] pour un dictionnaire papier, et Pleco [Pleco Software, 2000] pour un dictionnaire sur téléphone.

Puisque le projet est informatisé, le problème de stocker physiquement des informations appartenant à des dictionnaires papiers différents ne se pose pas. En effet, ces derniers, de par leur objectif pratique, se limitent généralement à un volume [1] (qui peut cependant être particulièrement large, par exemple le Kōjien).

Cependant, comme l'entreprise lexicographique doit être menée en un temps limité par une équipe limitée, il convient de fixer d'avance le type d'ouvrages à intégrer ; bien que le modèle proposé ici soit justement assez extensible pour rajouter après coup des ouvrages de nature très différente, qui ne faisaient pas partie du projet à l'origine. Ainsi pour chaque exemple d'application, nous définirons le ou les types de dictionnaire qui y figureront et le processus pour élaborer un dictionnaire sous forme de graphe à partir de ceux-ci.

2.1.3 Types de dictionnaires

Pour le japonais, on est susceptible d'avoir à manipuler des dictionnaires de nature différente : dictionnaire de langue (en japonais), dictionnaire de bilingue (depuis et vers le japonais), dictionnaire de caractères chinois (en japonais), dictionnaire de sinogrammes (en langue étrangère), dictionnaire de langue classique, dictionnaire de mots composés [2]. On peut également rencontrer des ouvrages plus spécialisés comportant moins de représentants : dictionnaire étymologique, dictionnaire de style de caractère chinois (calligraphie), dictionnaire d'accentuation. D'autres types d'ouvrages existent en sus de ceux-là, mais répondent à des besoins beaucoup trop précis pour être considérés d'usage courant : ce sont les dictionnaires de spécialité. Toute une gramme d'ouvrages porte le nom de dictionnaire dans leur titre mais n'en

1. Quand cette limite ne s'impose pas, on arrive à des dictionnaires très imposants tel le Grand Ricci et ses sept volumes, plus un volume d'annexes.

2. 漢語辞典 (kango jiten) et 四字熟語 (yojijukugo jiten), respectivement des dictionnaires de mots formés à l'aide de kanji et d'expression en quatre kanjis.

étant pas vraiment [3], il ne sont pas inclus ici.

Pour chacune de ces catégories, listons un ou deux représentants.

Type de dictionnaire	Exemple
Dictionnaire de langue	Kōjien
Dict. de kanji (japonais)	Daikanwan Jiten
Dict. de kanji (autre langue)	New Nelson Japanese-English Character Dictionary
Dict. français-japonais	Royal Dictionnaire français-japonais
Dict. de prononciation	NHK Accent Dictionary

FIGURE 2.1 – Quelques types de dictionnaire avec exemple

2.1.4 Dictionnaires virtuels

Le terme de dictionnaire virtuel est utilisé, expliqué et illustré dans l'article de Polguère intitulé *Lexicographie des dictionnaires virtuels* [Polguère, 2012]. Il y explique, dès le résumé, que la structure du lexique d'une langue est celle d'un graphe d'entités lexicales fortement connectées les unes aux autres et que par conséquent, le futur de l'activité lexicographique doit, selon lui, être celle de la création de lexiques modelés sur cette structure et non plus seulement la rédaction de « dictionnaires textes ».

Le présent mémoire s'inscrit dans cette lexicographie du futur. Cependant plusieurs différences notables par rapport à l'idée esquissée par Polguère y sont présentes. Tout d'abord, le but de l'étude n'est pas la représentation « véritable (ou plausible) des lexiques » mais de ce qui peut servir pratiquement à la création d'un dictionnaire bilingue, voir d'autres types de dictionnaires. C'est donc par le but explicitement pratique — il ne s'agit pas de rechercher la modélisation la plus efficace ou probable d'un ou des lexiques — et son caractère multilingue que l'idée s'en démarque.

En second lieu, le processus de création de dictionnaires textuels à partir du lexique-graphe est énoncé dans cet article comme un processus ayant lieu une fois en amont, pour produire un artefact immutable. Ici, l'idée d'application qui sous-tend l'étude est un processus dynamique : certes le même processus de sélection d'une partie du lexique-graphe pour en générer un sous-ensemble d'un domaine limité peut être appliqué, mais en plus ce processus peut être effectué, à multiples reprises et à une granulité très fine, par

3. [Mel'čuk *et al.*, 1995] p. 20.

l'utilisateur directement dans l'application [4], lui permettant alors de créer exactement le dictionnaire dont il a besoin.

Nous pourrions ainsi parler de *dictionnaire virtuel dynamique* ou de *dictionnaire personnel*, ce dictionnaire virtuel dont le processus lexicographique s'étend jusqu'à l'utilisateur final pour lui permettre une adéquation à ses usages inégalée jusque-là.

2.1.5 Micro-structure

La micro structure d'un dictionnaire, c'est l'organisation d'un seul article de dictionnaire [5]. Cette structure contient tous les champs d'informations susceptibles d'être contenus dans une entrée, avec en plus des règles concernant le nombre d'éléments que ces champs peuvent contenir (par exemple si un champ doit obligatoirement être rempli, ou s'il y a une limite au nombre de mots qu'il peut contenir) et les règles de rédaction à destination des lexicographes. En outre, la microstructure définit également la disposition graphique de ces différents champs pour former une entrée. Dans un dictionnaire papier, ou informatisé à partir d'un ouvrage papier, la microstructure est indissociable de son affichage.

En revanche, dans le modèle en graphe, cette notion d'ordre d'affichage n'existe plus explicitement : l'organisation de la micro-structure est découplée de la problématique de l'affichage sur un écran ou d'une impression papier. Cela reste toutefois une problématique importante, puisque qu'un utilisateur attend de pouvoir consulter les informations du dictionnaire sous une forme dont il est coutumier. La problématique l'affichage est donc traitée dans une section qui lui est dédiée : 3.6 **Affichage d'une fiche de dictionnaire**.

Remarques sur la micro-structure

Dans de nombreux dictionnaires, en particulier unilingues et bilingues, la micro-structure peut être compliquée à exprimer formellement. Les entrées, bien que structurées, présentent une grande variabilité, imposée par la nature différente (verbe, nom, adjectif, ...) des entités décrites, une grande variabilité d'informations qui ne se retrouvent que dans un nombre restreint d'entrées

4. Nous parlons ici d'application informatique au sens général, bien que le choix du terme par rapport à d'autres (logiciel, programme) soit évidemment influencé par le fait que les prototypes créés pour tester la validité des concepts soient des *applications mobiles*.

5. [Mel'čuk *et al.*, 1995] p.32.

(parce quelles nécessitent plus d'explications que d'autres), possiblement sous la forme d'encart distingué du reste par une typographie, des couleurs, un formatage différent.

Ce n'est pas un problème pour le lecteur humain, qui arrive intuitivement à comprendre leur raison d'être, mais cela complexifie énormément la tâche d'établissement d'une grammaire formelle de la micro-structure des entrées des dictionnaires en question. Pour illustration, on peut citer la tentative partielle de formalisation de la micro-structure d'un dictionnaire effectuée par [Lecailliez, 2015a].

Cette tâche se heurte également à la définition du contenu des champs en langue naturelle, qui comporte un grand nombre de caractères différents. Même une langue dont on dit couramment qu'elle s'écrit à l'aide de 26 lettres fait en réalité usage de plus d'une centaine de caractères quand on prend en compte son bicaméralisme, les lettres diacritées, les chiffres et les symboles de ponctuation. Ces derniers sont en outre particulièrement présents dans les dictionnaires et nécessitent une attention particulière par rapport à d'autres types de textes.

Cette formalisation est pourtant importante. En effet, comment peut-on comparer les dictionnaires entre eux, en particulier papier et électronique, s'il n'existe pas de base de comparaison ? Des travaux ont notamment été menés par [Doan-Nguyen, 1998] concernant l'importation et la transformation de dictionnaires via l'écriture d'une grammaire de reconnaissance de leurs entrées.

La problématique est cependant trop large pour être plus qu'évoquée ici. On notera toutefois que dans ce travail, la micro-structure d'un dictionnaire modélisé sous forme de graphe est formalisée sous la notion de **Graphe de structure** (3.2.2). Ainsi que le fait que le graphe est une structure mathématiques pour laquelle il existe des opérations définies (union, réécriture, etc.), ce qui facilite l'expression des transformations qu'on pourrait vouloir lui apporter.

2.1.6 Macro-structure

La macro-structure est quant à elle l'agencement de toutes les entrées du dictionnaire pour former l'ouvrage final. Dans un dictionnaire de langue française, c'est l'ordre alphabétique qui fait office de macro-structure. En japonais, cela peut être l'ordre phonétique dit des 50 sons (五十音, gojūon) ou plus anciennement celui du Iroha (いろは, iroha). Le cas des dictionnaires

de kanjis est plus complexe, mais l'ordre généralement adopté se base sur un système « de clefs », qui sert à effectuer un premier tri, puis au sein de chacun des ensembles définis par ces clefs, un second tri est organisé en fonction du nombre de traits qui n'apparaissaient pas déjà dans la clef. Parmi les dictionnaires de chinois, on en trouve maintenant dont les entrées sont triées alphabétiquement selon leur romanisation en pinyin.

Ceci étant, cette définition me paraît trop limitée. En effet, l'objectif de la macro-structure est de faciliter la recherche dans un dictionnaire, si tant est que l'ordre de classement soit connu de l'utilisateur. Si celui-ci arrive sur une entrée qui n'est pas celle qu'il cherche, il peut utiliser sa connaissance de l'ordre de tri afin de se déplacer vers une autre entrée, et cela jusqu'à trouver celle désirée. Les index servent à la même chose : faciliter la recherche depuis une donnée (clé) connue. Chaque index a une structuration propre, c'est à dire un ordre de tri, qui n'est rien de moins que ce qui est défini comme la macro-structure. Ainsi, j'étendrai la définition de la macro-structure à l'ordonnancement principal des entrées du dictionnaire plus celui de ses index. Ce n'est pas innocent : dans le modèle proposé, il n'existe pas d'ordre de tri principal par lequel les entrées seraient « naturellement » accessibles ; seuls les index vont permettre de chercher un article à partir d'une donnée extérieure.

2.2 Rappel sur les graphes

2.2.1 Définitions

Un graphe est une structure mathématique particulièrement versatile adaptée à la modélisation de nombreux problèmes, ce qui est un préalable à leur résolution au moyen de l'informatique. C'est une structure qui permet de représenter des situations concrètes communes, par exemple les réseaux (informatiques, de personnes, le Web) et qui est utilisée en mathématiques, physique, linguistique informatique, sciences sociales, etc.

En outre, il est possible pour des cas simples [6] de les représenter visuellement, ce que nous ne nous priverons pas de faire. Dans la suite des explications, les notions mathématiques abordées sont consultables plus en détails dans l'ouvrage *Theory - Undergraduate Mathematics* [Koh *et al.*, 2015] ainsi que [Müller, 2012] qui est une introduction en français à la théorie des graphes, adressée aux lycéens.

6. En fait, pour les graphe dits planaires.

Graph non orienté

Un **graphe non orienté** G est formé par un couple (S, A) tel que :
- S est un ensemble fini de **sommets**,
- A est un ensemble de pairs $\{s_i, s_j\} \in S^2$ où $i \neq j$ appelées **arêtes**.

Un graphe tel que défini [7] est appelé **graphe simple** : il ne peut pas comporter de boucle, c'est à dire d'arête entre un sommet et lui-même.

Dans le cas contraire, on parle de **multigraphe**. Dans un multigraphe, A est un multi-ensemble d'arêtes. C'est une généralisation de la notion d'ensemble où un élément peut apparaître à plusieurs reprises.

On dit que les sommets s_i et s_j sont **adjacents** dans le graphe si $\{s_i, s_j\} \in A$, c'est-à-dire s'il existe une arête qui les relie.

Graph orienté

La définition d'un graph orienté est très semblable à celle d'un graph non orienté. Il s'agit également d'un couple formé par un ensemble de sommets et d'arcs. La différence se situe dans la définition d'un arc, qui est cette fois basée sur la notion de couple et non de paire. L'ordre des éléments du couple a une importance, contrairement à la paire.

Un **graphe orienté** G est formé par un couple (S, A) tel que :
- S est un ensemble fini de **sommets**,
- A est un ensemble de couples $(s_i, s_j) \in S^2$ où $i \neq j$ appelées **arcs**.

La seconde condition peut être retirée, auquel cas le graph peut comporter des arcs qui bouclent sur un sommet et on est en présence d'un multigraphe orienté.

Chaîne et composante connexe

Dans un graphe non orienté, une **chaîne** de x à y sommets du graphe est une suite finie d'arêtes consécutives reliant x à y. Dans un graphe orienté, on parle de **chemin**.

7. Cette définition est très inspirée de celle trouvée dans le support de cours `http://liris.cnrs.fr/~snndiaye/fichiers/App_Graphes.pdf`.

« Une **composante connexe** C d'un graphe $G = (S, A)$ est un sous-ensemble maximal de sommets tels que deux quelconques d'entre eux soient reliés par une chaîne : si $x \in C$, alors

$\forall y \in C$, il existe une chaîne reliant x à y,

$\forall z \in S \setminus C$, il n'existe pas de chaîne reliant x à z. » [8]

Pour le dire plus simplement, une composante connexe est un ensemble de sommets qui sont tous reliés directement (il existe une arête entre eux) ou indirectement (il existe une chaîne entre eux) les uns aux autres. Tout noeud hors de la composante connexe n'est relié directement ou indirectement à aucun noeud de la composante en question.

2.2.2 Hypergraphe

Un **hypergraphe** est une généralisation du graphe non orienté dans laquelle les liens ne sont plus limités à un ou deux sommets. On peut ainsi relier trois sommets ou plus à l'aide d'une arête, appelée **hyper-arête** ou plus simplement arête.

Formellement, la traduction française de [Bondy et Murty, 2007] par F. Havet [9] nous indique qu'un hypergraphe est un couple $(V, \mathcal{F})$, où l'ensemble des sommets et $\mathcal{F}$ est une famille de sous-ensembles de V qui forme l'ensemble des hyper-arêtes du graph.

2.2.3 Parcours et plus court chemin

Un des avantages à utiliser une structure de données standard est qu'on peut profiter des années de recherche sur les algorithmes qui les utilisent. Les graphes sont encore aujourd'hui un sujet de recherche, mais il existe d'ores et déjà de nombreux algorithmes publiés sur le sujet.

Deux problèmes classique sont celui du parcours du graphe, et celui du plus court chemin. Le parcours d'un graphe consiste à visiter chacun de ses nœuds. Le calcul du plus court chemin consiste à trouver le plus court chemin entre deux nœuds donnés.

8. `http://www.math.u-psud.fr/~montcouq/Enseignements/Apprentis/chemins.pdf`

9. `http://www-sop.inria.fr/members/Frederic.Havet/Traduction-Bondy-Murty.pdf`

Ces problèmes et les algorithmes qui y répondent sont importants dans la théorie des graphes. À ce stade du travail, le parcours du graphe entier n'est nécessaire que dans un cas : la création des index (cf. 3.5.1 **Index**).

2.3 Notions logicielles

2.3.1 Architecture logicielle en couches

Les logiciels informatiques sont souvent structurés – on parle d'architecture logicielle – selon des principes généraux et reconnus. Ceci a pour but de créer des programmes qui peuvent être plus facilement maintenus et déployés. Une des architectures logicielles la plus utilisée est l'architecture en couches. On pourra utilement se référer au chapitre 5 et les suivants du guide publié par Microsoft [MSDN, 2016] [10] pour une explication détaillée.

Dans cette architecture, chaque couche, qui est un concept abstrait, regroupe des composants (des parties du logiciel) qui servent à répondre à une problématique précise et distincte des autres couches. Les couches sont empilées les unes sur les autres et interagissent avec celles avec lesquelles elles sont en contact, c'est-à-dire la couche immédiatement supérieure et inférieure.

Elles imposent une séparation logique du code du programme qui peut même se retrouver physiquement : une couche peut être programmée avec un langage différent ou être déployée sur une autre machine, par exemple.

Architecture en trois couches

L'architecture en couche la plus usitée est probablement celle en trois couches, appelées : couche présentation (*presentation layer*), couche métier (*business layer*) et couche d'accès aux données (*data layer*). La couche présentation est celle qui est responsable de l'affichage et de l'interaction avec l'utilisateur. C'est par elle qu'est affiché le contenu d'un programme et que sont utilisées les fonctionnalités exposées par la couche métier.

10. D'où provient la figure 2.2

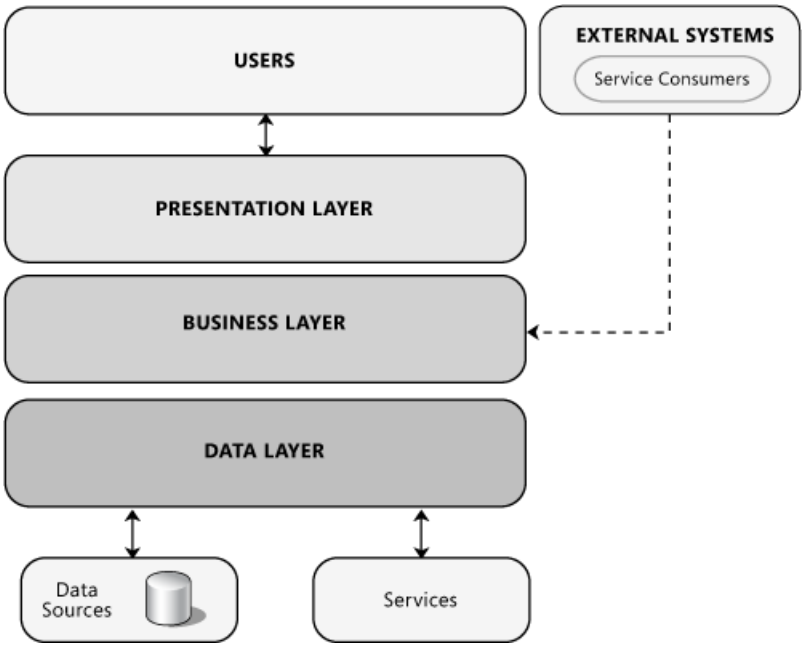


FIGURE 2.2 – Vue haut niveau d'une architecture en trois couches

La couche métier implémente le cœur logique de l'application. C'est là que sont groupés les composants qui réalisent les traitements liés au domaine du problème à résoudre (dans notre cas, représenter et manipuler des dictionnaires). Enfin la couche d'accès aux données se charge de fournir à la couche métier les données nécessaires à son activité. Elle permet d'abstraire la façon et l'endroit où sont stockées les données : fichiers, base de données locale ou distante, service web, etc.

Utilisation dans le mémoire

Le présent mémoire définit le modèle de données destiné à être implémenté par la couche métier. Il ne s'occupe pas du tout des responsabilités de la couche d'accès aux données. Ainsi, contrairement à d'autres travaux sur les dictionnaires tels que [Mangeot, 2004] et [Gader *et al.*, 2012], nous ne détaillerons aucun procédé utilisé pour stocker les données [11]. Il n'y aura donc

11. Pour information cependant, les prototypes se servent de fichiers textes. Le format de ceux-ci est spécifié formellement dans sa seconde version.

pas d'exemples de fichiers XML ou de schémas de base de données, puisque ceux-ci peuvent de toute façon être interchangés sans changement dans la couche métier.

En revanche puisque la finalité de l'étude est éminemment pratique, la couche présentation, qui se trouve à la périphérie du questionnement mérite d'être abordée là où cela est utile. Les illustrations liées à cette couche seront tirées des prototypes réalisés et ne seront donc pas seulement de pieux vœux abstraits.

2.3.2 Modèle de représentation « traditionnel » de l'information

Le paradigme le plus couramment utilisé en informatique est actuellement celui de la programmation impérative. Cela pour des raisons historiques et techniques : les processeurs eux-mêmes, en charge des calculs, sont de nature impérative.

Structures

Dans beaucoup de ces langages, on retrouve la notion de **structure** (*record* ou *struct* en anglais), qui est une structure de données basique qui permet d'agréger différents **champs**. Ceux-ci sont fixés d'avance, peuvent être de types différents à travers la structure et sont nommés. Une structure se démarque donc sur deux points d'un tableau, où les éléments sont de type identique et sont accédés par un indice numérique.

On peut représenter une structure par un tableau à trois colonnes. L'entête indique le nom de la structure. Chaque champ de la structure est matérialisé par une ligne dans le tableau. La première colonne représente son nom champ, la seconde son type et la dernière sa valeur. La figure 2.3 [12] fournit un exemple de structure remplie avec les données d'un des livres de la bibliographie.

12. Dans le tableau les types de données sont ceux du langage C#. *String* est une chaîne de caractère et *int* et *long* permettent de représenter des entiers relatifs.

Structure **Livre**		
Nom du champ	*Type du champ*	*Valeur du champ*
titre	string	La phonologie du japonais
author	string	Laurence Labrune
date_publication	int	2006
nb_volumes	int	1
nb_pages	int	305
isbn10	int	9042918055
isbn13	long	9789042918054
editeur	string	Peeters

FIGURE 2.3 – Exemple de structure informatique pour un livre

Des structures peuvent comporter des champs qui référencent ou incluent directement d'autres structures. Cela permet de modéliser des situations plus complexes.

Impact sur les dictionnaires électroniques

Il y a deux points clefs qu'il est important de souligner pour comprendre en quoi utiliser directement ce modèle de structuration des données pour réaliser des dictionnaires informatisés se distingue de celui qui est proposé dans le reste du mémoire.

Premièrement, des ouvrages différents vont être représentés de manière très différente, c'est-à-dire avec des structures dotées de champs qui diffèrent en nombre et nom. Un dictionnaire de kanji et un dictionnaire bilingue par exemple n'ont pas du tout besoin des mêmes structures puisqu'ils ne détaillent pas les mêmes objets lexicographiques.

Comme le nombre et le nom de champs sont différents il est impossible de définir d'avance un moyen standard de traiter des données encodées de cette manière. Bien qu'il existe dans certains langages de programmation un moyen d'accéder aux noms des champs d'une structure par une technique appelée introspection (*reflexion* en anglais), cela ne rend pas pour autant aisé l'accès à un champ quelconque, ni ne permet de comprendre la sémantique qui leur est associée.

Deuxièmement, les références vers d'autres structures sont contenues dans la définition des structures elles-mêmes. De manière similaire au point précédent, cela rend compliqué leur accès automatique. Le graphe des relations

entre entités est enfoui dans les références que les instances de ces structures maintiennent entre elles.

Un des intérêts de modéliser un dictionnaire sous forme de graphe de la façon présentée dans le chapitre suivant est d'avoir à disposition des données sur les données (méta-données) qui permettront de traiter automatiquement, par exemple pour leur affichage, des dictionnaires dont le contenu n'est pas connu d'avance, tout en autorisant des traitements plus complexes le cas échéant.

L'autre intérêt est d'expliciter la structure et les relations entre les objets lexicographiques contenues dans un dictionnaire, soit pour la présenter à l'utilisateur qui peut y voir un intérêt, soit pour effectuer divers calculs dessus : distance sémantique, création d'un nouveau dictionnaire comme le fait [Tanaka-Ishii *et al.*, 1998], etc.

Chapitre 3

Modélisation en graphe d'un dictionnaire

3.1 Vue d'ensemble

Comme l'introduction et les rappels sur les graphes l'ont laissés entendre, la modélisation proposée pour un dictionnaire virtuel de japonais se base sur un graphe. Les éléments de ce graphe-dictionnaire vont être énoncés : il s'agit d'expliciter les propriétés et la structure de ceux-ci ainsi que la façon dont ils s'assemblent.

Très grossièrement, les sommets (nœuds) du graphe comprennent l'essentiel de l'information lexicographique atomique du dictionnaire (formes graphiques, définitions, images, etc.) Ils sont reliés par des arcs – aussi appelés relations ou liens – qui contiennent, tout comme les nœuds, des métadonnées. Les interfaces de manipulation de ces éléments sont standardisées et cela concourt à permettre leur traitement générique.

Le graphe utilisé est un hypergraphe dont toutes les arêtes ne relient pas forcément le même nombre de sommets [1]. De plus, il permet de mélanger arêtes, arcs et hyper-arêtes en fonction des besoins du dictionnaire à modéliser.

Enfin le modèle définit également deux infrastructures supplémentaires : les annotations qui permettent d'enrichir les nœuds et les liens d'informations qu'il serait peu pertinent de modéliser à leur moyen, et les index qui servent à trouver rapidement un nœud du graphe.

1. Il ne s'agit donc pas d'un hypergraphe k-uniforme.

3.2 Typage

3.2.1 Types

Nécessité du typage

Les objets que peut contenir le graphe [2] qui nous servent à modéliser un dictionnaire virtuel sont de nature différente et doivent donc pouvoir être différencié les uns des autres. Intuitivement, on voit par exemple qu'une lexie vedette et un texte d'explication ne sont pas conceptuellement identiques, il faut donc un moyen de conserver cette distinction dans le système d'information. C'est pourquoi nous ferons appelle à la notion de **type**.

Ce concept est omniprésent en informatique au travers des différents systèmes de type des langages de programmation et a une théorie dédiée en mathématiques. Il va donc sans dire qu'il est hors de portée de définir ici ce qu'est un type. Nous utiliserons donc une définition simplifiée, qui suffit aux objectifs visés.

Définition

Le type d'un objet du graphe est une structure de données contenant les informations suivantes : un nom, une description, un identifiant unique, le type d'objet (nœud ou lien) auquel il se rattache et une liste optionnelle de contraintes dans le cas où il s'agit d'un type qui peut être associé à un arc.

Nom

Le nom d'un type est une chaîne de caractère non nulle, non vide et contenant au moins un caractère visible. Celui-ci est destiné à être manipulé et lu par des humains : à la fois dans ce travail lorsqu'il sera question de discourir sur un type déterminé, mais aussi dans une implémentation informatique réelle où il est visible dans les fichiers de données, le débogueur, etc.

Description

Comme un nom (*a fortiori* court) ne permet pas de lever toutes les ambiguïtés qui pourraient exister quant à son interprétation, une explication

2. Le terme **objets du graphe** signifie nœuds et arcs du graphe-dictionnaire.

détaillée est associée à chaque type. Cette explication, qui est rédigée à destination des lexicographes, explicite ce qu'est censé contenir et représenter un nœud de ce type. C'est le pendant des explications à destination des lexicographes associées à un champ dans la micro-structure d'une fiche de dictionnaire traditionnel.

Identifiant

L'identifiant unique, dont la nature et la forme sont un choix d'implémentation [3], est l'information qui permet de tester si deux types sont uniques ou non. Le fait de ne pas reposer sur le nom du type pour cela est motivé par le fait que celui-ci peut exister dans plusieurs projets différents pour indiquer des objets de natures différentes, ce qui peut poser problème si les données de ces projets ont vocation à être fusionnées, et par la possibilité que cela offre de renommer un type sans que cela n'impacte le reste du système.

Propriété des types

À un objet du graphe est associé un unique type. Les types sont mutuellement exclusifs. Ils sont totalement indépendants les uns des autres ; en particulier il n'existe aucune relation d'héritage (parenté) telle qu'on peut la retrouver dans les langages de programmation orienté objet.

C'est une différence importante avec les travaux élaborés sur la base des technologies du web sémantique telles que Web Ontology Language (OWL). Dans celles-ci, la notion d'ontologie est centrale, et il existe des hiérarchies de types. Les types sont appelés *classe* dans la terminologie de ces standards.

Un brouillon de travail du W3C [W3C, 2006] traite de la modélisation en RDF/OWL de WordNet [Miller, 1995], un célèbre réseau lexical. Les hiérarchies de classes utilisées sont reproduites ci-dessous (figure 3.1).

3. Dans l'implémentation qui sert de référence à ce mémoire, c'est un GUID qui est utilisé.

Synset		
	AdjectiveSynset	
		AdjectiveSatelliteSynset
	AdverbSynset	
	NounSynset	
	VerbSynset	
WordSense		
	AdjectiveWordSense	
		AdjectiveSatelliteWordSense
	AdverbWordSense	
	NounWordSense	
	VerbWordSense	
Word		
	Collocation	

FIGURE 3.1 – Hiérarchies de classes WordNet (fragment)

Les classes de la première colonne sont les classes de base de leur hiérarchie respective. Les classes situées dans les colonnes plus à droite sont dérivées de la première classe rencontrée de la colonne immédiatement à leur gauche et en remontant par le haut. Ainsi *VerbSynset* est un dérivé de *Synset*, et *AdjectiveSatelliteWordSense* est une classe dérivée de *AdjectiveWordSense*, elle-même dérivée de *WordSense*.

Dans le modèle détaillé par ce mémoire, les types de nœuds sont tous au même niveau. Ils ne déclarent pas eux-même quels sont les types de nœuds avec lesquels ils sont en relation, que celle-ci soit de parenté ou autre.

En effet, celui nuit à l'extensibilité du modèle, car pour modifier la liste des relations, il faut modifier le type lui-même. Or, une modification du type entraîne en fait la création d'un nouveau type, pour éviter les problèmes de compatibilité. La gestion des relations entre nœuds est entièrement confiée aux **liens**.

Exemple d'un type de nœud

Le tableau ci-dessous (figure 3.2) illustre les différentes informations d'un type de nœud qu'on pourrait utiliser dans un graphe.

Élément	Valeur
Nom	Mot français
Objet concerné	Vertex
Description	Représente une lexie (mot, expression semi-figée ou idiomatique) française.
Identifiant	5688661d-ed5d-4909-b209-a43f9807943f

FIGURE 3.2 – Exemple d'un type de nœud

On remarque que contrairement à ce qui a été dit précédemment, la description n'y est pas très précise. Dans un projet réel, celle-ci pourrait être beaucoup plus conséquente, afin d'expliquer notamment la notion de lexie, et laisser le moins d'ambiguïtés possibles dans ce qui peut constituer ou non un nœud de ce type.

3.2.2 Graphe de structure

Une fois les types nécessaires à un projet définis, on peut former un graphe qui les représente graphiquement. Ce graphe est différent du graphe dictionnaire (qui stocke les données du dictionnaire proprement dites) dont on a parlé jusqu'à présent. Il est appelé **graphe de structure**.

Parce qu'il représente tout ou partie de la structure d'un dictionnaire établie suivant le modèle proposé ici, on peut dire que le graphe de structure est une représentation de tout ou partie de la micro-structure de ce dictionnaire.

Définition

Un graphe de structure est formé de la manière suivante : l'ensemble des sommets du graphe est constitué des types de nœuds en présence. Pour chaque paire (x, y) de sommets [4], il existe un arc entre x et y si et seulement s'il existe un type de relation dont les contraintes lui permettent de lier les sommets de ces types.

Intérêt

Le graphe de structure est un outil précieux pour rendre compte rapidement et facilement de la structure du graphe-dictionnaire qu'on définit ou

4. Un lien peut donc exister entre un sommet et lui-même.

qu'on manipule. En effet, il explicite tous les liens possibles entre les nœuds, ce que ne fait pas un exemple dans le cas général. De plus, puisqu'il s'agit d'un graphe, on peut le représenter graphiquement [5]. Un graphe de structure ne contenant que les types définis pour un projet, il reste de taille raisonnable. Il permet ainsi de raisonner sur le graphe-dictionnaire sans avoir à modéliser celui-ci complètement.

Illustration

Pour mieux comprendre la relation entre un graphe de structure et les graphes instanciés qui s'y rapportent, voici une image qui contient un graphe de structure et deux graphes instanciés à partir de celui-ci.

Les synonymes pris comme exemple sont tirés du *Dictionnaire Électronique des Synonymes* de l'université de Caen [CRISCO, 1998]. Notez que ce projet fait lui-même usage d'un graphe de synonymes pour calculer la proximité entre différents mots [6].

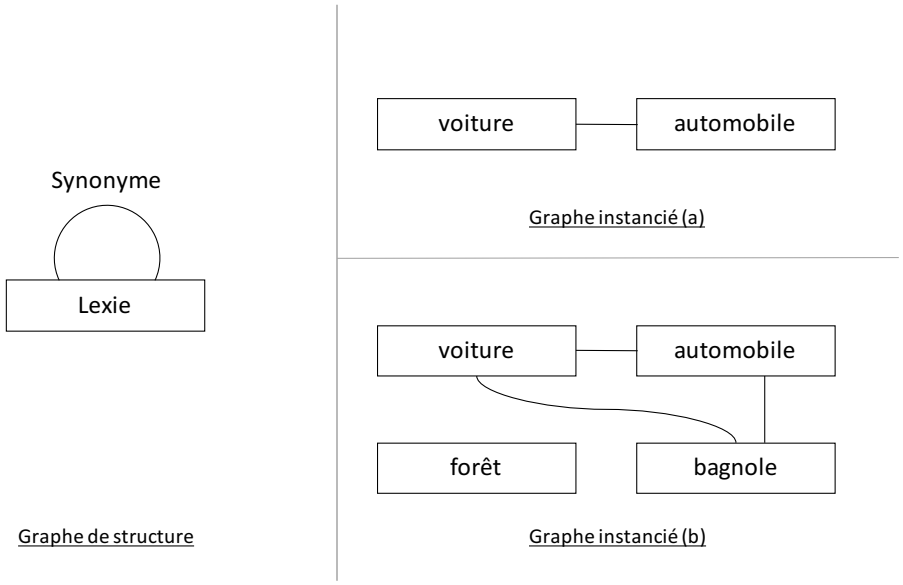


FIGURE 3.3 – Graphe de structure et deux instances

5. Sans croisement d'arête s'il s'agit d'un graphe planaire.
6. [François et Manguin, 2004] p. 12.

Le graphe de structure de la figure 3.3 représente une modélisation dans laquelle deux types sont respectivement définis : le type de nœud et le type d'arête *Lexie* et *Synonyme*.

Le graphe de structure représente les types de nœuds par des nœuds, les types d'arêtes par des arêtes et deux éléments sont étiquettés par leur nom. Au contraire, dans le graphe d'instance les nœuds contiennent les données (c'est-à-dire les lexies à intégrer au dictionnaire dans cet exemple) et les liens n'existent qu'entre les nœuds pour lesquels cette relation existe.

On voit donc que le graphe de structure permet de représenter précisément les graphes instanciés associés à un système de types donnés et est relativement économique en terme de place comparé à une illustration par des graphes instanciés – même très petit comme le graphe figure 3.3 (b) – de ce même système.

3.3 Nœuds

Le nœud est une des deux sortes d'objet que peut contenir le graphe-dictionnaire. Un nœud contient de l'information à propos d'une donnée du dictionnaire. Il possède un type. Les nœuds sont atomiques, ils contiennent de l'information qui ne peut pas être partagée ou liée directement par d'autres nœuds du graphe : pour cela il faut établir un lien entre le nœud lui-même, et un autre objet du graphe qui y fait référence.

Dans ce modèle de données un nœud peut contenir une structure et des données arbitrairement complexes, possiblement non textuelles. Ceci fait que les dictionnaires générés peuvent faire usage de données multimédias. Les dictionnaire papiers peuvent déjà être multimédias de manière limitée : les dictionnaires Larousse sont en effet célèbres pour les illustrations qu'ils contiennent. Le cas des nœuds de structure arbitrairement complexe est illustré en 4.3 **Nœuds complexes**.

3.3.1 Définition

Un nœud est une structure composée d'un type, d'un nom canonique, de chaînes indexables et d'un nombre quelconque de propriétés privées.

L'indexation des nœuds fait l'objet de la section 3.5.1 **Index**.

Type

Le type est un type tel que discuté précédemment en 3.2.1 **Types**.

Nom canonique

Le nom canonique d'un nœud est une chaîne de caractères (possiblement vide, mais non nulle) qui permet de le représenter textuellement. Elle n'est pas nécessairement unique au sein d'un dictionnaire.

Le nom canonique d'un nœud sert à fournir une représentation textuelle par défaut de celui-ci. Celle-ci peut être mise à profit lors du développement du dictionnaire (les données tout comme les clients qui s'en servent) s'il y a besoin d'afficher la valeur d'un nœud, par exemple dans un débogueur.

Lors de l'utilisation du dictionnaire lui-même, c'est le nom canonique d'un nœud qui est utilisé pour l'affichage de celui-ci en l'absence d'informations supplémentaires sur comment gérer son affichage, ainsi que dans la fiche générique (3.6.1).

En donnant une valeur sensée à ce nom canonique, il est possible de développer et de manipuler rapidement et facilement un graphe-dictionnaire.

3.3.2 Atomicité

Un point fondamental, et qui diverge de l'approche utilisée par [Polguère, 2014] est que les nœuds sont considérés comme atomiques du point de vue des autres objets du graphe. Un sommet peut en lier un autre mais il ne peut jamais référencer des données internes d'un autre nœud, même de type identique. En outre, il ne peut pas référencer depuis sa structure interne un autre nœud ou lien. Cette propriété est fondamentale dans l'approche pour deux raisons : la première est qu'elle force à utiliser des nœuds contenant relativement peu d'information, la seconde est liée à l'extensibilité du graphe.

En effet, puisqu'un nœud ne peut pas référencer le contenu d'un autre nœud, si ce contenu doit effectivement être référençable, il doit être placé dans un type de nœud identique ou différent mais dans une instance séparée. Puisque cette information est isolée dans un nœud propre, elle devient plus facilement réutilisable. Une plus grande réutilisation signifie un plus grand nombre de liens depuis et vers un nœud et donc une possibilité de navigation accrue entre les données.

Le second intérêt de cette propriété est l'extensibilité. Puisqu'un nœud

ne peut pas lier le contenu interne d'un autre nœud, on a un découplage total entre eux. On peut raisonner sur un type de nœuds ou sur un groupe de type de nœuds sans avoir à prendre en compte les autres types présents dans le graphe puisqu'ils n'ont pas accès à l'intérieur des types considérés. Et réciproquement les nœuds à propos desquels on raisonne n'ont pas accès à la structure interne des types de nœuds non considérés.

Ainsi, il est possible d'ajouter à un graphe des nouveaux types et de modifier les données privées de ceux déjà présents sans risquer de casser la compatibilité avec d'autres composants logiciels que ceux dédiés à l'interprétation des types en question.

Dans le cas d'un retrait ou d'une mise à jour (remplacement) d'un type existant, il faut bien entendu prendre en compte les types de liens qui peuvent relier les nœuds retirés, mais ces types eux-mêmes sont explicites et ne sont pas enfuis au sein de la structure interne d'autres nœuds.

3.4 Liens

Les liens sont la seconde famille d'objets qui peuplent le graphe. Ils permettent de relier des nœuds afin de spécifier qu'il existe entre eux une certaine relation. Celle-ci est indiquée au moyen d'un type et son interprétation est laissée libre au programme qui consomme le graphe.

Par rapport aux nœuds, les liens comportent deux informations supplémentaires : une indication sur son orientation (oui/non) et une liste de contraintes. L'orientation se réfère à la notion mathématique précédemment décrite et permet d'indiquer si celle-ci a une importance pour ce type de lien ou non.

3.4.1 Définition

Un lien est une structure composée d'un type de lien et des nœuds qu'il relie.

Contrairement à un nœud, un lien ne peut donc pas déclarer de chaînes indexables ni avoir de propriétés privées. En revanche grâce au mécanisme d'annotations (3.5.2) il est possible d'ajouter des informations à un lien telle qu'une pondération.

Remarques sur la définition

Notez qu'on aurait pu toutefois imaginer qu'un lien ait une définition plus proche du nœud et possède lui-même des chaînes indexées et des propriétés privées. En fait c'était une approche envisagée pour le modèle jusqu'à ce qu'une tentative de formalisation de l'affichage des fiches eut été effectuée. Il était plus simple à comprendre et à formaliser la présentation d'une fiche de dictionnaire à partir d'un sommet qu'à partir d'un sommet ou d'une relation. Cela reste cependant une piste de généralisation du modèle.

3.4.2 Lien simple

Un lien simple relie deux nœuds du graphe. Il correspond à un lien dans un graphe non orienté : les nœuds reliés y ont tous deux accès et il peut servir à passer d'un sommet à l'autre dans les deux sens.

3.4.3 Lien orienté

Un lien orienté relie deux nœuds du graphe. Il correspond à un lien dans un graphe orienté : il lie un nœud source à un nœud cible, et n'est traversable que dans ce sens.

3.4.4 Hyperlien

Un hyperlien est similaire à un lien simple mais il relie plus de deux nœuds du graphe.

3.4.5 Nécessité des hyper-liens

Les hyperliens sont nécessaires pour traiter de façon satisfaisante deux problèmes qui se pose en lexicographie japonaise : le fait que la lecture des mots de la langue n'est généralement pas évidente à cause de l'utilisation de caractères chinois (kanji) ainsi que les caractères chinois qui peuvent être considérés comme un objet lexicographique à part entière.

Dans le cas du français, un dictionnaire de langue comme le Larousse se permet de ne pas indiquer la prononciation des mots dans le cas général car les règles de lecture sont supposées connues du locuteur natif. Mais en japonais un mot listé par une vedette rédigée dans un syllabaire sera normalement

accompagné de sa forme en kanji en plus de la définition car celle-ci permet de distinguer des homophones. Et réciproquement, dans le cas où les vedettes sont formées de mots écrits en kanji, la lecture dans un syllabaire est présente en plus d'autres informations afin d'indiquer comment il se lit.

Problématique des caractères chinois

Le liens simples permettent de modéliser la situation, présente dans certains ouvrages tel que [Tokuhiro, 2014] où un caractère est glosé par une liste de prononciations ainsi que par une liste de sens sans que les deux ne soient mis en correspondance.

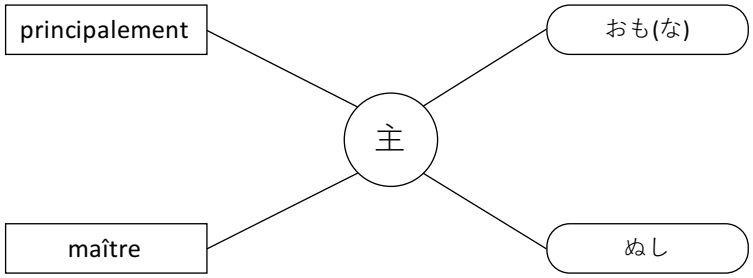


FIGURE 3.4 – Kanji modélisé à l'aide de liens simples

Cette représentation permet d'arriver à la même disposition que le papier : à partir du nœud qui représente un kanji, on établit la liste des ses lectures en parcourant les liens appropriés et de même pour former la liste des définitions ou des traductions.

Le problème est que cela ne permet pas de mettre en correspondance des lexies ou des morphèmes avec un signifié donné. Un programme qui se baserait sur la modélisation de la figure 3.4 pour inférer les triplets suivants {principalement, 主, おもな[7]}, {principalement, 主, ぬし[8]}, {maître, 主, おもな}, {maître, 主, ぬし} ferait deux déductions fausses, à savoir {principalement, 主, ぬし} et {maître, 主, おもな}.

C'est pourquoi les liens à trois nœuds ont été intégrés au modèle. Ils permettent de mettre en correspondance le sens associé à un caractère chinois

7. Lu onona.
8. Lu nushi.

à une lecture. On peut ainsi modéliser la présentation qu'adopte un dictionnaire de kanji comme le New Nelson [Haig *et al.*, 1997], dont un extrait est présenté en figure 3.5, et la recréer en parcourant le graphe sans en déduire de fausses associations.

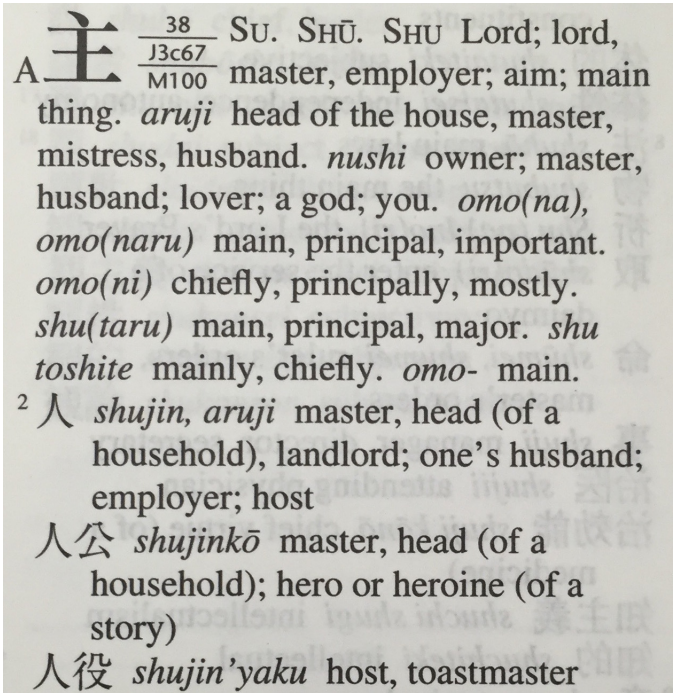

A 主 38 / J3c67 / M100 SU. SHŪ. SHU Lord; lord, master, employer; aim; main thing. *aruji* head of the house, master, mistress, husband. *nushi* owner; master, husband; lover; a god; you. *omo(na), omo(naru)* main, principal, important. *omo(ni)* chiefly, principally, mostly. *shu(taru)* main, principal, major. *shu toshite* mainly, chiefly. *omo-* main.
2 人 *shujin, aruji* master, head (of a household), landlord; one's husband; employer; host
人公 *shujinkō* master, head (of a household); hero or heroine (of a story)
人役 *shujin'yaku* host, toastmaster

FIGURE 3.5 – Extrait de dictionnaire pour le kanji 主

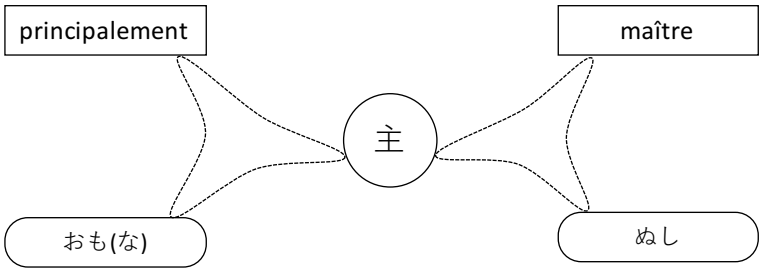


FIGURE 3.6 – Kanji modélisé à l'aide d'hyperlien

Sur la figure 3.6 les liens ont été modélisés par une sorte d'étoile à trois branches qui indique que les nœuds qu'elles touchent sont liés par une relation à trois éléments. Dans cette configuration, on peut déduire directement des triplets en parcourant une seule relation. Il s'agit dans cet exemple de {principalement, 主, おもな} et {maître, 主, ぬし}.

3.4.6 Contraintes

Les contraintes permettent de spécifier sous quelles conditions un type de lien peut être utilisé, c'est-à-dire instancié entre deux types nœuds. C'est le pendant formel de l'existence visuelle d'un lien dans un graphe de structure.

En effet, le graphe de structure est un outil théorique [9] qui permet de comprendre qu'un lien l relie deux types t_1 et t_2, mais il faut un mécanisme qui permette de spécifier formellement cela et qui soit implémentable facilement.

Une contrainte est un objet qui expose une unique propriété : un booléen qui indique si elle est satisfaite ou non.

Les contraintes sont paramétrées, c'est-à-dire qu'elles nécessitent un ou plusieurs arguments pour être totalement définies. Il existe trois sortes de contraintes : contrainte de type, contrainte d'arité et contrainte booléenne.

Liste de contraintes

Une *liste de contraintes* est associée à chaque type de liens. Chacune des contraintes doit être vérifiée pour que la liste des contraintes soit considérée comme vérifiée. Autrement dit, la liste de contraintes peut se résumer à une unique contraire booléenne *ET* entre toutes les contraintes qu'elle contient.

Contrainte de type

La contrainte de type permet d'exprimer quel type d'objet peut être la source ou la cible d'un lien. Elle prend en paramètre un type d'objet du graphe. Tout type qui n'est pas spécifié comme étant accessible pour un lien

9. Du moins dans le cadre de ce mémoire. En fait, il est possible de construire un outillage de façon à pouvoir manipuler des graphes de structures et de générer les données des types de nœuds et de liens (qui contiennent les contraintes) appropriées et utilisable par une implémentation de dictionnaire sous forme de graphe.

ne l'est pas. Ainsi par défaut un type de lien qui ne spécifie pas les types d'objets qu'il peut lier est inutile car ininstanciable [10].

Soit l un lien reliant des objets du graphe $o_1, ..., o_n$ de type respectifs $T_1, ..., T_n$, la contrainte de type $CT(T_i)$ appliquée à l est satisfaite si $T_i \in \{T_1, ..., T_n\}$.

Par exemple le type de lien *Synonyme* de la figure 3.3 (section 3.2.2 **Graphe de structure**) possède une contrainte de type paramétrée par le type *Lexie*. La contrainte n'est pas violée dans les figures 3.3 (a) et (b) car les instances de *Synonyme* lient toutes au moins une instance de *Lexie*.

Contrainte d'arité

La contrainte d'arité permet de fixer le nombre d'objets du graphe que peut relier un lien. Elle est paramétrée par un entier naturel supérieur ou égal à 2.

Soit l un lien reliant des objets du graphe $o_1, ..., o_n$, la contrainte d'arité $CA(a) \mid a \in \mathbb{N}^* \setminus 1$ appliquée à l est satisfaite si $|\{o_1, ..., o_n\}| = a$.

Contraintes booléennes

Le troisième type de contrainte, dit booléenne, est formé d'un ensemble de trois contraintes distinctes qui ont pour but de composer entre elles les contraintes existantes afin de pouvoir exprimer des contraintes plus expressives.

La contrainte ET est paramétrée par deux contraintes. Elle est vérifiée si les deux contraintes qu'elle prend en paramètre sont vérifiées.

La contrainte OU est paramétrée par deux contraintes. Elle est vérifiée si au moins une des deux contraintes qu'elle prend en paramètre est vérifiée.

La contrainte NON est paramétrée par une contrainte. Elle est vérifiée si la contrainte qu'elle prend en paramètre ne l'est pas.

Les autres opérations booléennes classiques (ou exclusif, implication, équivalence, nand, ...) peuvent être exprimées à l'aide des trois opérations précédentes. De même, des opérations portant sur plus de deux paramètres pour ET et OU peuvent être formées à partir des opérations à deux arguments

10. En pratique on peut tout à fait autoriser à instancier un lien dont les contraintes ne sont pas satisfaites. On peut ensuite parcourir les objets du graphe pour vérifier si toutes ses contraintes sont satisfaites avant de réaliser une opération qui nécessite un état du graphe cohérent.

sans problème puisqu'elles sont associatives. De fait, les implémentations sont libres de définir et d'utiliser ce genre d'opérations qui raccourcit la notation des contraintes.

Intérêt

Les contraintes permettent de vérifier automatiquement qu'un dictionnaire-graphe est conforme au graphe de structure auquel il se rattache.

3.5 Autres composantes du graphe

3.5.1 Index

Un index est une structure de données qui permet un accès rapide à une information. Il associe une valeur à une clé. Il peut être implémenté de différentes manières ; parmi les plus courantes ont retrouve les arbres-B et les tables de hashage. Dans cette partie, nous considérerons l'utilisation d'une table de hashage.

Dictionnaire papier

Dans un ouvrage papier la clé est le plus souvent la donnée recherchée (une lexie, un caractère chinois par exemple), et la valeur associée à celle-ci dans un index est souvent le numéro de l'entrée dans le dictionnaire ou le numéro de la page où elle se situe. Il peut y avoir plusieurs index dans un même dictionnaire. Typiquement, on retrouve ainsi un index par prononciation et un index par radicaux (部首, bushu) dans les dictionnaires de kanji.

En outre, un index forme un niveau d'indirection par rapport à ce qu'on recherche dans le dictionnaire : il faut en effet tout d'abord chercher la clé dans l'index, puis à partir de la valeur retournée (un numéro de page par exemple), chercher celle-ci pour enfin trouver l'entrée du dictionnaire qui nous intéresse.

En informatique

Dans un programme, une telle indirection n'est pas nécessaire : on peut associer directement une entrée à une clé d'index et la retourner lorsqu'une

recherche correspondante est effectuée. L'association clé / valeur peut se réaliser à l'aide d'une structure de données bien connue : la table de hashage. On la retrouve sous des noms plus techniques tel que HashMap ou dictionnaire [11]. Comme ce dernier nom peut porter à confusion, nous utiliserons les termes de table de hashage, et plus simplement table, pour parler de cette structure.

Ceci étant, cette valeur n'est pas forcément unique : on peut vouloir associer à une clé plusieurs entrées du dictionnaire. Une clé d'index sera donc plutôt associée à une liste d'éléments qu'à un élément isolé.

Dans le modèle en graphe

> Nous définissons un index comme une table de hashage à qui une chaîne de caractères associe une liste d'objets du graphe.

C'est en effet les objets du graphe (nœuds et liens) qu'on cherche et qu'on veut pouvoir trouver efficacement pour affichage. Un élément pourrait avoir à rester en quelque sorte caché (non indexé), être indexé par plusieurs chaînes, ou encore figurer dans plusieurs index.

Pour arriver à cela, un élément du graphe déclare quelles chaînes il exporte en tant que chaîne à indexer. Confier cette responsabilité aux nœuds et liens eux-mêmes est dans la logique d'encapsulation qui a servi à prescrire la règle selon laquelle une donnée interne d'un nœud ne peut pas être référencée par un lien. Ici aussi on considère qu'un nœud « sait mieux que quiconque » sous quelles formes il doit être indexé. En fait, une donnée interne semblable à la chaîne à indexer pourrait très bien ne pas exister dans le nœud : par exemple dans le cas d'une image. Il n'est donc pas raisonnable dans le cas général de demander à un programme externe de se charger de cette tâche.

Processus d'indexation

Il vient d'être dit qu'un nœud peut être référencé par plusieurs index. Mais la routine chargée de compiler les index doit avoir un moyen de savoir dans quel index se place la chaîne et l'élément référencé. Le modèle doit en effet pouvoir gérer plusieurs index, d'une part pour gérer plusieurs langues différentes et permettre une recherche dans chacune d'elles, mais aussi pour

11. Cf. l'interface IDictionary<T> en .net par exemple.

gérer les cas des dictionnaires autre que multilingues qui comportent plusieurs index.

Pour cela, le nœud fournit un couple de chaîne (*index*, *valeur*) où *index* est le nom d'un index et *valeur* la chaîne à inclure dans cet index. Comme plusieurs valeurs doivent pouvoir être exportées, c'est en fait une liste de couples (*index*, *valeur*) qui est fournie par chaque nœud. Cette liste peut être vide.

Algorithme

Le processus d'indexation, c'est à dire de création et remplissage des index fonctionne de la façon suivante : la procédure parcourt le graphe et pour chaque élément indexable fait appel à la méthode [12] *GetIndexedStrings*.

Dans la liste ainsi renvoyée, l'indexeur traite chaque couple rencontré. Pour cela, si un nom d'index n'est pas connu, un nouvel index est créé, associé à ce nom, et le couple (*chaîne*, *liste*) est inséré dans cette table ; *chaîne* est la seconde valeur du couple renvoyé par la méthode *GetIndexedStrings* tandis que la liste est initialisée avec l'élément à indexer.

Dans le cas où l'index existe, on y ajoute de manière identique un couple (*chaîne*, *liste*) si *chaîne* n'existe pas dans la table, sinon c'est simplement l'élément qui est ajouté à la liste associée à la chaîne.

Illustration

Dans l'exemple de la figure (3.7), les objets du programme sont représentés par des rectangles. Il y a un objet Indexeur qui contient deux index (Index Hepburn et Index Nippon-Shiki), ainsi qu'un objet Graphe qui contient deux nœuds qui ont pour noms canoniques ふじさん (fujisan, le mont Fuji) et にほん (nihon, le Japon). Chacun de ces nœuds exporte deux chaînes à indexer, une pour un index nommé *hepburn* l'autre pour un index nommé *nippon*.

12. Cela peut être implémenté de façon différente suivant les possibilités du langage utilisé. Dans le prototype en C#, c'est une propriété éponyme et non une méthode qui est utilisée.

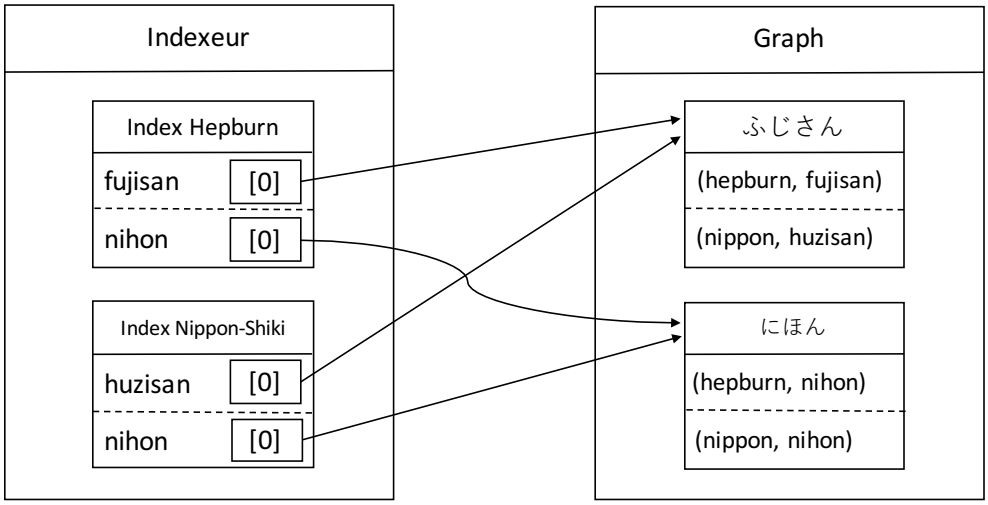


FIGURE 3.7 – Exemple d'indexation d'un graphe à deux éléments

En parcourant les nœuds du graphe, l'indexeur peut ajouter aux index correspondants les chaînes qui permettent de référencer les nœuds ; ici *fujisan* et *nihon* pour l'index Hepburn et *huzisan* et *nihon* pour l'index en Nippon-shiki. Comme il n'y a qu'un nœud à référencer par chaîne dans chaque index, la liste associée à chaque clé dans les tables de hachages ne contient qu'un élément, d'indice 0 et qui pointe vers le nœud correspondant (ceci est représenté par des flèches).

Cet exemple illustre un graphe à deux éléments qui fournissent chacun une chaîne de caractères à deux index différents. La séparation en deux index distincts pour ces deux chaînes permet de faire des recherches en romaji (lettres latines) à l'aide d'une ou l'autre de ces méthodes de transcription sans pour autant contenir, et donc renvoyer, des doublons lorsque cette romanisation est identique dans les deux méthodes.

Limites

L'utilisation d'une table de hashage pour les index a l'avantage d'être facilement implémentable puisque c'est une structure de données directement disponible dans de nombreux environnements de programmation. Elle comporte cependant un inconvénient majeur dans le cadre de l'élaboration d'un

dictionnaire.

Une table de hashage ne permet en effet que de retourner une valeur pour laquelle la clé exacte existe dans la structure. Cela pose un léger problème de normalisation (par exemple pour les mots comportant des lettres capitales, les abréviations qui comportent des points, etc.) qui peut généralement être résolu en indexant les formes les plus courantes et en pré-traitant la chaîne recherchée.

En revanche, la recherche par préfixe y est impossible. Ce comportement, qui liste les mots commençant par la chaîne recherchée est présent dans certains dictionnaires électroniques et nécessite d'utiliser d'autres structures de données, tel qu'un arbre préfixe (*trie*) ou un arbre radix (*Patricia trie*), pour être supporté.

Comme ces structures supportent également l'interface d'une table de hashage, on pourra les utiliser dans une implémentation pour fournir la fonctionnalité de recherche par préfixe, tout en restant dans le cadre de la définition énoncée plus précédemment pour les index.

On peut aussi se servir d'un système de gestion de base de données relationnelle (SGBDR) pour implémenter la couche d'accès aux données et utiliser les fonctionnalités qu'il fournit pour permettre une gamme de recherche (*fuzzy match*) plus étendue.

3.5.2 Annotations

Deux concepts ont été présentés : les nœuds et les arcs. Ils constituent les objets du graphe. S'ils permettent de représenter un grand nombre de situations, ils ne sont pas forcément le moyen le plus approprié pour tous les cas. Pour le montrer, prenons deux exemples : celui de la fréquence et celui de la catégorie d'un mot.

Ici la fréquence est exprimée par un ordre de fréquence, qui est un entier naturel positif, calculé à l'aide d'un corpus de référence. Pour associer une fréquence à un mot modélisé par un nœud du graphe, on peut créer un type de nœud dédié, et un type d'arc qui permet de faire le lien entre ces deux types de nœud. Est-ce cependant satisfaisant ?

Dans la mesure où un nœud représentant une lexie n'est relié qu'à un unique nœud d'ordre de fréquence et puisque les nœuds de fréquence sont indicatifs d'une fréquence bien particulière (par exemple la fréquence d'un kanji en japonais contemporain calculée avec un corpus donné), il est garanti qu'à un nœud de fréquence ne sera associé qu'un nœud d'un autre type donné.

Dans ce cas, la structuration en graphe n'a pas d'intérêt, si ce n'est d'alourdir la possible représentation graphique et de nécessiter la navigation d'un lien pour accéder à la donnée (ici la fréquence).

Dans le cas des parties du discours, un nœud de catégorie aura un très grand nombre de liens pour les catégories ouvertes. Ceci est à même de compliquer l'utilisation automatique du graphe, en obligeant un lien qui traverse un tel nœud (il rentre dans le nœud de catégorie puis effectue une action sur tous les liens qu'il contient) à effectuer un traitement potentiellement très long. On pourrait mitiger le problème en faisant usage de liens unidirectionnels partant d'une lexie et allant vers une catégorie — ce qui évite ainsi le parcours des arcs sortants du nœud de catégorie — mais on perd ainsi l'intéressante possibilité d'obtenir toutes les lexies d'une catégorie depuis le nœud qui la représente.

Définition

Ces deux types de problèmes, qui sont tous deux liés aux nombres de liens qu'un nœud possède ou peut posséder, justifient la création d'un concept supplémentaire : **les annotations**. Les annotations ne sont pas un objet du graphe : elles ne sont pas des entités indépendantes. Une annotation est liée à un objet du graphe (nœud ou arc) et se présente sous la forme d'un triplet (*espace de nom*, *clé*, *valeur*).

Clé et valeur

La clé est une chaîne de caractères qui indique ce que représente l'annotation. C'est par elle qu'on accède à une annotation donnée. Pour les exemples précédemment évoqués, on pourrait se servir des clés notées *fréquence* et *catégorie*.

La valeur est également une chaîne de caractère. Bien entendu, c'est au programme client d'interpréter cette valeur. Dans le cas de la fréquence, qui s'exprime sous forme numérique, le programme client a la responsabilité de la convertir dans un type de données numérique s'il veut s'en servir pour ordonner les données, ou pour d'autres traitements.

Espace de nom

Le modèle en graphe se veut modulaire et extensible, en particulier il doit pouvoir être relativement facile de fusionner des projets différents en un

seul. De ce point de vue, l'utilisation de noms simples pour les clés peut se révéler problématique si deux projets font usage des mêmes noms pour deux concepts différents.

Même au sein d'un seul projet, on peut vouloir utiliser la même clé pour plusieurs concepts totalement différents. Ainsi, pour faire la distinction entre ceux-ci, on les place dans des espaces de nom différents.

Un espace de nom est une chaîne de caractères non nulle, non vide, constituée d'au moins un caractère visible. Deux espaces de nom sont identiques si leur valeur normalisée dans l'encodage utilisé est identique.

3.6 Affichage d'une fiche de dictionnaire

Problématique

La question de l'affichage est cruciale pour l'applicabilité du modèle proposé : on ne peut en effet pas se contenter d'afficher un graphe à l'utilisateur, qui s'attend à une présentation plus classique des données.

La première chose à remarquer est qu'il n'y a pas de façon unique de gérer cet affichage : en dehors des considérations typographiques et des couleurs qui ne nous intéressent pas ici, c'est le public cible final qui va déterminer les informations à présenter. On a déjà insisté sur le fait qu'un dictionnaire-graphe pouvait rassembler plusieurs sortes de dictionnaires distincts. Il est donc tout à fait possible de ne souhaiter afficher que des informations spécifiques à l'un d'eux.

La seconde chose à noter est que le contenu du dictionnaire, même dans le cas où un seul type de dictionnaire est modélisé par le graphe, est fragmenté en différents nœuds qui constituent autant de points d'entrée pour lesquels il pourrait y avoir une fiche. Ainsi, avec une relation (mot japonais, graphie en kana, mot français) telle celle que l'on retrouve dans le premier cas d'application (4.1 **Application à un lexique**), on peut imaginer pouvoir afficher une fiche en considérant qu'un mot japonais ou qu'un mot français sont des mots vedette ; et cela même si les données proviennent à l'origine uniquement d'un lexique japonais-français.

Définitions

Cette situation nécessite de poser quelques définitions. Nous appelerons *fiche* le contenu affiché associé à un *nœud-vedette*. Une fiche commence forcé-

ment par le nom canonique du nœud-vedette à laquelle elle correspond puis contient le contenu potentiellement vide des nœuds qui lui sont liés.

Dans ce modèle, les vedettes sont donc choisies parmi les nœuds ; même si les liens sont également instanciés ils ne peuvent pas directement devenir vedette.

3.6.1 Fiche générique

La première étape pour fournir une visualisation des fiches est donc de choisir les types de nœuds susceptibles d'être des nœuds vedettes. Il existe une sorte de fiche pour chaque type de nœud vedette.

Prenons l'exemple d'un graphe très simple tel qu'il contienne les types *MotFrançais* et *MotJaponais* qui peuvent être reliés par une relation *Traduction*. Ce graphe bipartite a pour graphe de structure la figure 3.8 (a) et un exemple d'instance est fourni en figure 3.8 (b).

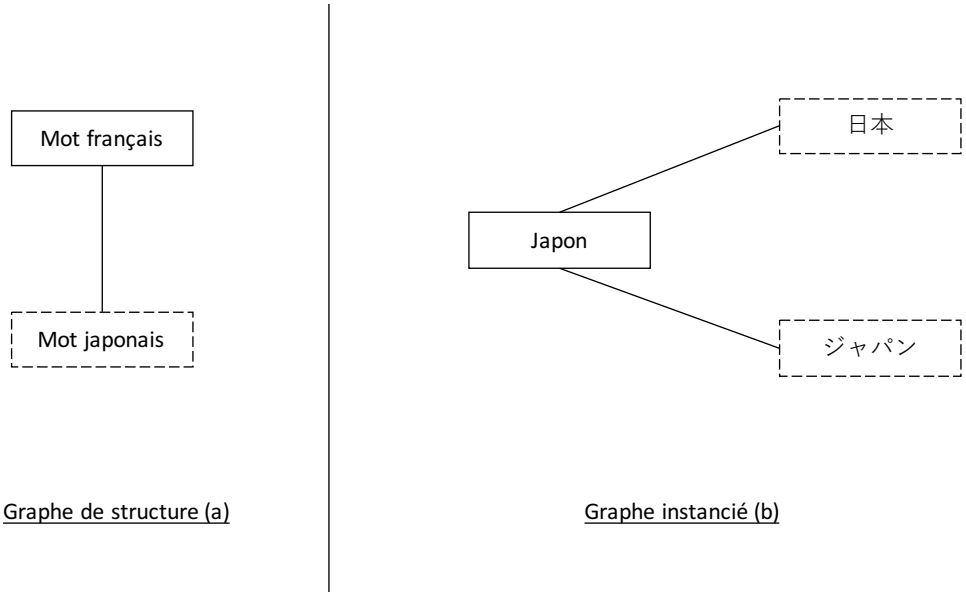


FIGURE 3.8 – Graphe de structure et exemple d'instance

On décide que les *MotFrançais* peuvent être des nœuds vedettes. On peut représenter la fiche associée à ce type de nœud de la façon suivante :

Nœud vedette (*MotFrançais*)
Nœud lié (*MotJaponais*)
...
Nœud lié (*MotJaponais*)

FIGURE 3.9 – Fiche vue depuis un nœud *MotFrançais*

Japon
日本
ジャパン

FIGURE 3.10 – Application à l'exemple

La liste des mots japonais est potentiellement vide, en revanche le nom canonique de la vedette est forcément présent, ce qui correspond à la définition d'une fiche posée précédemment.

On pourrait également décider que *MotJaponais* peut devenir un type de nœud vedette. Dans ce cas-là, la fiche associée pourrait revêtir cette forme :

Nœud vedette (*MotJaponais*)
Nœud lié (*MotFrançais*)
...
Nœud lié (*MotFrançais*)

FIGURE 3.11 – Fiche vue depuis un nœud *MotJaponais*

日本
Japon

FIGURE 3.12 – Application à l'exemple

On remarque qu'au changement de type de nœuds près, la structure de la fiche est la même que celle de la fiche de *MotFrançais*.

3.6.2 Blocs

La notion de bloc est liée à celle de lien. Lors de l'affichage d'une fiche, le contenu des nœuds liés à la vedette par des liens n'est pas affiché directement : les nœuds sont d'abord regroupés par type de liens qui y accèdent.

Prenons comme exemple un nœud *MotJaponais* de nom canonique « 日本 » [13] relié aux nœuds de nom canonique « Japon » par la relation *Traduction* et « 日 » [14] et « 本 » [15] par la relation *Kanji*.

13. Lu nihon, sens Japon.
14. Sens de jour, soleil.
15. Sens d'origine, racine, livre.

La fiche générique pour « 日本 » sera constituée de deux blocs : le premier qui contient « Japon » accédé via le lien *Traduction* et un second qui contient « 日 » et « 本 » accédé par les liens de type *Kanji*. Elle n'est donc pas constituée de trois blocs (« 日本 », « 日 » et « 本 ») qui correspondent à chacun des nœuds liés.

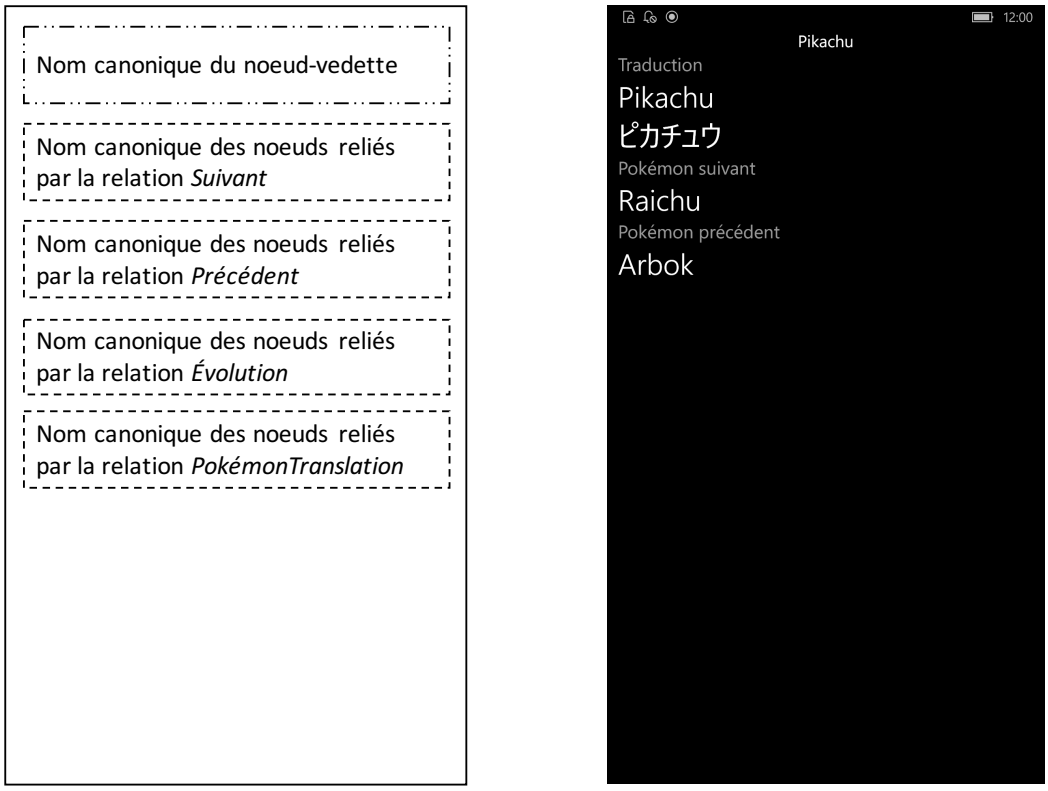
3.6.3 Fiche générique pour les liens simples et orientés

En généralisant cette observation, on en vient à définir la *fiche générique* pour les nœuds vedettes qui ne comportent que des liens simples ou orientés.

Cette fiche est formée du nom canonique du nœud vedette, puis pour chaque type de liens simples ou orientés, du nom canonique des nœuds liés. Le point important est que chaque fiche est donc formée, en plus de la vedette, de *blocs* qui affichent des informations obtenus via les instances de relations qui relient la vedette à d'autres nœuds.

Pour illustration, la figure 3.13 (a) représente la fiche générique abstraite d'un *PokémonFrench* du second cas d'application 4.2 **Implémentation d'un Pokédex**. Elle est secondée, pour comparaison, de la fiche réelle (b) obtenue à partir du prototype d'application mobile.

Dans la figure 3.13 (b) le bloc qui contient le nœud-vedette se situe tout en haut et est centré. Les blocs qui contiennent les informations obtenues à partir des liens (texte large et blanc) indiquent les relations associées à l'aide de texte plus petit et gris.



(a) fiche abstraite (b) fiche concrète

FIGURE 3.13 – Fiches génériques abstraite et concrète

On constate que l'ordre des blocs dans les deux images n'est pas le même. Ce n'est pas choquant car l'ordre d'apparition des blocs n'est pas spécifié pour la fiche générique. Une application concrète doit donc prévoir un mécanisme pour pallier à cela, soit en ajoutant des annotations au graphe, en définissant un ordre ou en permettant à l'utilisateur de remanier les blocs [16].

On remarque également que les nœuds sont effectivement groupés par relation : « Pikachu » et « ピカチュウ » (pikachu) sont des instances de types différents mais sont liés à ce nœud vedette par la même relation, de

16. C'est cette dernière solution qui a été retenue pour le prototype, bien que non implémentée. En outre, ces différentes solutions peuvent être combinées.

type *PokémonTranslation*. Ils sont donc regroupés dans le même bloc.

On note également que la fiche du prototype indique pour chaque bloc par quelle relation les nœuds ont été atteints. Il peut en effet être difficile pour l'utilisateur de deviner à quoi fait référence tel contenu sans cette indication, d'autant plus qu'entre les fiches associées à un même type de nœud-vedette certains blocs peuvent être absents (car la donnée n'est pas présente dans le graphe). C'est le cas ici du bloc représentant les nœuds *PokémonImage* atteint par des liens *HasImage*, qui ne serait toutefois représenté dans la fiche que sous la forme d'une chaîne de caractère (nom canonique) et non d'une image.

Navigation dans la fiche générique

Outre l'affichage d'un nœud et de ceux qui lui sont reliés, la fiche générique définit un comportement qui permet de naviguer d'un nœud à un autre.

Lorsqu'un clic [17] est effectué sur le contenu affiché depuis une relation, la fiche générique change de façon à ce que le nœud cliqué devienne le nœud-vedette. Les blocs qui affichent le contenu lié au nœud vedette sont également modifiés.

La figure 3.14 illustre la navigation dans la fiche générique à l'aide d'un exemple basé sur les données du second cas d'application (section 4.2) [18]. L'écran de la figure 3.14 (a) est généré sur le modèle de la fiche générique avec comme vedette un nœud qui a pour nom générique « Weedle ».

17. Ou une action assimilée, dépendante de l'implémentation et du type d'appareil concerné.

18. Les captures d'écran à l'origine en résolution 1080x1920 ont été tronquées en 1080x768 avant redimensionnement. L'espace tronqué verticalement ne contient aucune information supplémentaire.

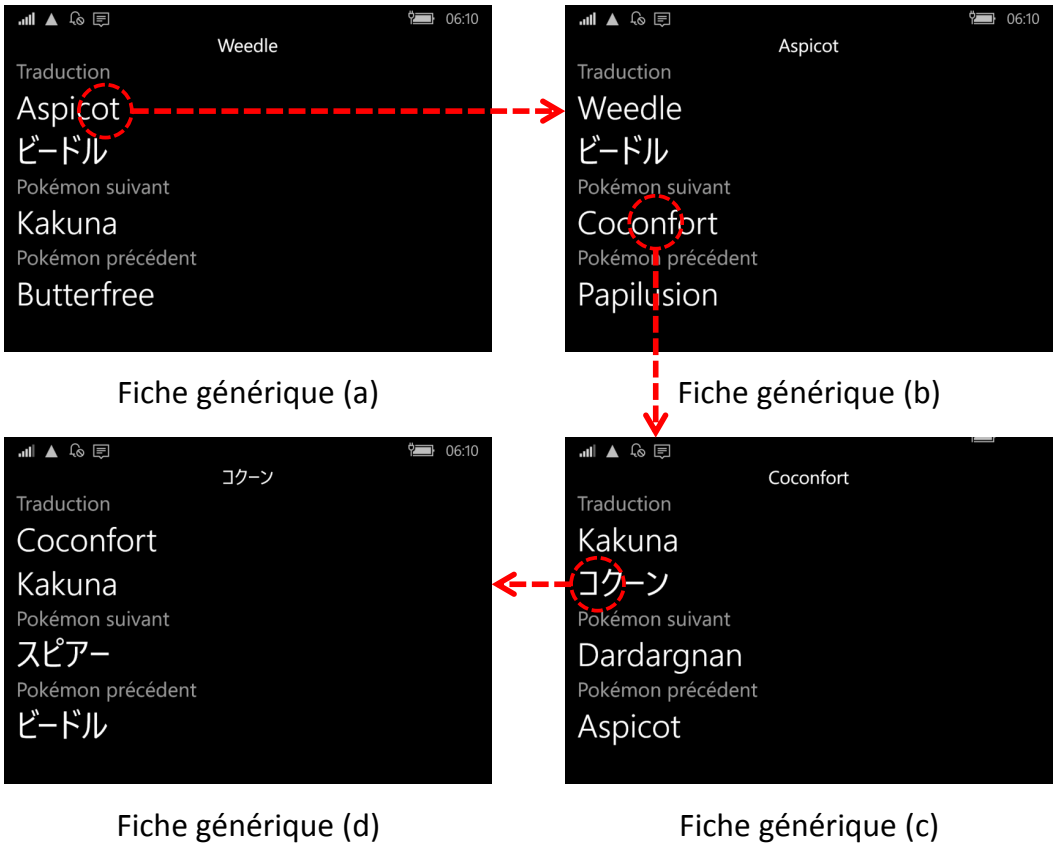


FIGURE 3.14 – Exemple de navigation dans la fiche générique

Un clic sur le nœud « Aspicot », qui est relié au nœud vedette par un lien de type *Traduction* va actualiser l'affichage de façon à présenter la fiche visible en figure 3.14 (b). Celle-ci est également générée sur le modèle de la fiche générique mais à pour vedette le nœud « Aspicot ».

Les autres relations présentes sur la fiche, ici *Pokémon suivant* et *Pokémon précédent* sont également cliquables et permettent de se rendre sur d'autres fiches, elles aussi générées sur le modèle de la fiche générique.

3.6.4 Composants d'affichage personnalisés

À elle seule, la fiche générique telle que définie précédemment n'adresse pas deux problématiques : d'une part l'affichage du contenu d'un nœud, et non de son nom canonique, d'autre part celle du traitement des hyperliens.

Jusqu'à présent, on a supposé que le nom canonique était suffisamment proche du contenu du nœud pour être affiché et compris par l'utilisateur à la place du contenu proprement dit [19].

Mais cette hypothèse n'est pas valide dans le cas général : la possibilité de définir des nœuds avec un contenu arbitrairement complexe existe précisément pour palier aux cas où une simple chaîne de caractère affichée directement ne suffit pas à représenter l'information lexicographique désirée.

C'est ici qu'entre en jeu une notion supplémentaire : celle de *composant d'affichage personnalisé* (appelé plus simplement composant personnalisé dans la suite), dont le rôle est de permettre d'afficher le contenu des nœuds présents dans un bloc.

Un composant personnalisé est un module de code informatique qui sert à lire, interpréter, afficher, paramétrer et interagir avec le contenu des nœuds appartenant à un bloc. Afin de pouvoir modifier facilement un projet par la suite, un tel composant ne devrait gérer qu'un seul type de bloc, c'est à dire de lien.

Puisque les instances de liens comportent tous un type qui contient un identifiant unique, et de même pour les instances de nœuds, il est possible de déclarer dans le code ou la configuration d'une application une association entre un type de lien et un composant personnalisé.

La figure 3.15 présente la fiche générique qui fait usage d'un composant personnalisé pour l'affichage d'une image. Le nom de la relation (*HasImage*) ainsi que son contenu (l'image d'un Pikachu) sont affichés par ce composant.

19. Dans la majorité des cas, on n'a en fait même pas défini le contenu des nœuds car le nom canonique suffisait amplement. La plupart des données utilisées par le prototype sont dans ce cas là.

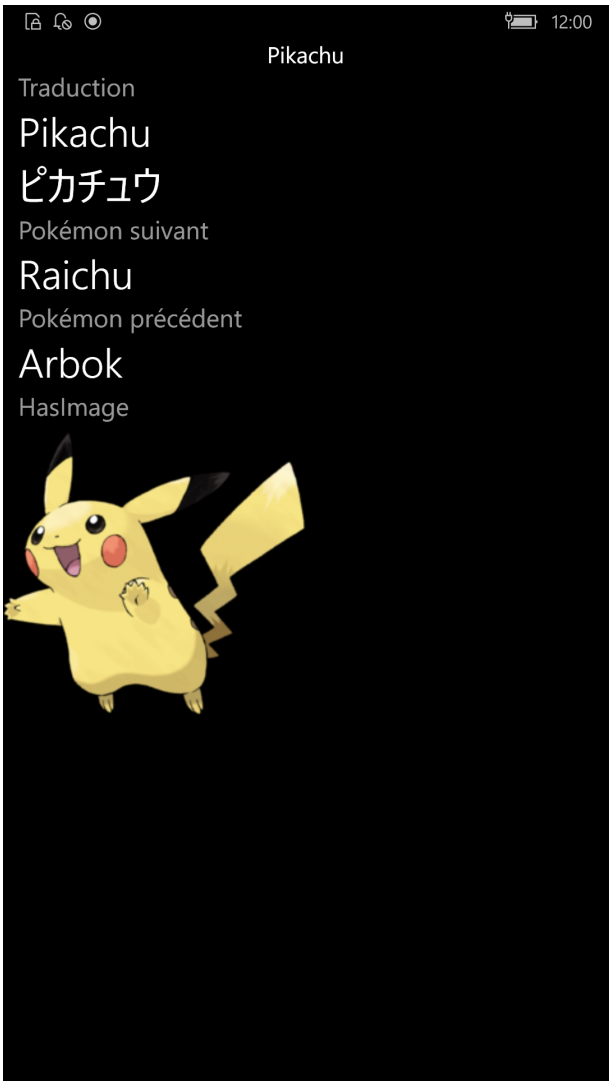


FIGURE 3.15 – Fiche générique avec usage d'un composant personnalisé

Hormis l'utilisation d'un composant personnalisé, le reste de la fiche est identique à celle de la figure 3.13 (b). Les deux ont en fait été générées à partir de la même infrastructure.

Chapitre 4

Cas d'usages : applications et exemples

4.1 Application à un lexique

4.1.1 Description rapide

En premier exemple d'application, nous allons tenter de modéliser sous forme de graphe les informations présentes dans l'ouvrage *100 fiches de vocabulaire japonais* [Peyrat, 2012]. Cet ouvrage n'est pas un dictionnaire à proprement parler mais il rassemble plus de 3000 mots japonais sous une forme, que nous examinerons en détail, qui est très courante parmi les ressources lexicales mises à disposition des apprenants du japonais.

C'est donc un exemple intéressant à étudier, car il peut être réutilisé avec peu de modifications pour traiter les différentes listes de vocabulaire pré-existantes mises à disposition du public sur le web.

En outre, la simple correspondance bilingue, si elle ne répond pas clairement pas au niveau de contenu et qualité des dictionnaires franco-japonais (Royal, Crown), se retrouve pour des couples de langue moins courants ; par exemple thailandais-japonais. C'est donc une base de travail correcte pour un dictionnaire très basique.

4.1.2 Micro-structure

Les listes de vocabulaire japonais se retrouvent le plus souvent sous la forme de triplets composés de la graphie la plus courante du mot susceptible

de contenir des kanjis, la lecture en hiragana de ce même mot japonais et sa traduction française.

Il y a plusieurs subtilités à relever avec ce format. Premièrement il existe le cas où un mot ne s'écrit pas couramment en kanji, soit parce qu'il n'a effectivement pas d'écriture qui fasse usage de sinogrammes, soit parce que celles-ci sont tombées en désuétude en japonais contemporain. Dans ce cas, le champ associé à l'écriture utilisant des kanjis est laissé vide [1].

Deuxièmement, un mot peut être écrit en katakana. Dans ce cas, c'est le champ de prononciation qui est utilisé, en katakana à la place des hiragana. La plupart des mots écrits en katakana n'ont pas d'écriture en kanjis, ce cas se cumule ainsi avec le point précédent.

Troisièmement, un terme peut avoir plusieurs écritures en kanjis et plusieurs lectures. C'est l'écriture en kanjis la plus courante qui est listée. Dans le cas d'une lecture multiple, plusieurs peuvent être indiquées successivement.

Dernièrement, l'écriture japonaise mélange allègrement plusieurs écriture et le champ qui contient le mot sous sa forme courante est donc susceptible de contenir des caractères des trois systèmes d'écritures propres au japonais.

Ainsi sous l'apparente simplicité du modèle se cachent plusieurs cas particuliers. Notons pour illustrer des exemples tirés de la page 57 de [Peyrat, 2012].

(1)	La fleur	花	はな
(2)	Le platane	—	プラタナス
(3)	Le pétale	花弁	はなびら/かべん
(4)	L'olivier	オリーブの木	オリーブのき

Le cas le plus courant est celui représenté par (1). Aucun des cas particuliers listés précédemment n'affecte ce genre d'entrée. La ligne (2) illustre le cas d'un mot qui s'écrit seulement en katakana. L'exemple (3) est représentatif d'un mot à plusieurs lectures, la première listée étant formée à partir de morphèmes purement japonais, la seconde fait quant à elle usage de radicaux de la strate sino-japonaise. Enfin la ligne (4) montre le mélange des trois écritures dans un même terme ainsi que le mélange des syllabaires hiragana et katakana pour la prononciation.

Une particularité de l'ouvrage utilisé ici est que ses listes de vocabulaire font usage d'articles définis, ce qui a l'avantage de présenter immédiatement le genre des noms (élision mise à part). Mais ce n'est pas un choix couramment

1. Mais sera remplacé par un tiret dans l'exemple qui suivra.

adopté, notamment dans les listes de vocabulaire destinées aux révisions disponibles sur internet [2].

4.1.3 Modélisation en graphe

Premier modèle : types de nœuds

En se basant sur la forme en triplet, on peut proposer une première modélisation en graphe tout à fait basique : on crée trois types de nœuds, chacun correspondant au contenu d'une des colonnes des fiches. Le premier type sera nommé *Français* pour la première colonne, le second *JaponaisKanji*, et le troisième *JaponaisLecture*. Pour instancier ces types, on utilise seulement la forme textuelle présente dans les fiches.

Puisque les termes en français sont uniques, on n'a normalement aucun doublon, il n'y aura donc pas deux entrées fusionnées en un même nœud du graphe. Ce type de fusion, où un nœud correspond à plusieurs éléments distincts dans le matériel d'origine est un comportement recherché car il permet d'augmenter la densité des liens dans le graphe : au lieu d'avoir N liens répartis en m et n liens attachés à n_1 et n_2 nœuds, on a ces N liens qui relient un seul nœud au reste du graphe.

Pour le second champ, qui contient une graphie pouvant comporter des kanjis, on est à peu près sûr de retrouver un effet similaire (l'absence de regroupement). Il pourrait cependant y avoir quelques cas où un nœud de ce type est lié à deux nœuds *Français*, voire plus [3]. Enfin le troisième champ est le plus susceptible de regrouper en un nœud plusieurs apparitions d'une même lecture grâce à la présence d'homophones.

Sans même regarder quels liens on pourrait avoir à créer, on constate à partir de ce qui vient d'être expliqué que cette modélisation est très peu intéressante : une infime partie des données vont être reliées les unes aux autres à cause d'un manque de nœuds communs à plusieurs entrées dans le graphe.

2. Tel que [Waller, 2012] par exemple.

3. Encore une fois, cela n'a pas été vérifié extensivement dans l'ouvrage car son contenu n'est pas disponible informatiquement.

Second modèle

Tentons donc une autre modélisation. Dans celle-ci nous effectuerons des pré-traitements et nous nous servirons de quatre types de nœuds. Le premier type, *Français* correspondra aux éléments de la première colonne. Cependant, la présence de déterminatifs est gênante car cela ne ressemble pas à ce que l'on trouve classiquement dans les dictionnaires. Pour un francophone cette information est, cas exceptionnel, peu importante car connue. Nous ne représenterons donc pas le genre des noms dans notre graphe [4].

Contrairement à l'exemple précédent, nous créons un type qui ne correspond pas directement à une colonne dans nos données d'origine : *Kanji*. Celui-ci est destiné à contenir un caractère chinois. Dans la description humaine du type, il sera bien précisé que le nœud contient un unique caractère. Les types *JaponaisKanji* et *JaponaisLecture* sont identiques à ceux définis précédemment.

Les nœuds de type *Kanji* sont instanciés à partir du contenu des mots présents dans la seconde colonne des fiches. Pour chaque caractère de la chaîne, si c'est un caractère chinois et qu'un nœud *Kanji* de nom canonique correspondant n'existe pas, il est créé.

Contenu des nœuds *JaponaisLecture*

On pourrait se poser la question de la normalisation en un seul système syllabaire des données affectées au type *JaponaisLecture*. Utiliser un même jeu de caractères pourrait permettre de fusionner des nœuds où des homonymes ne sont différenciés que par l'écriture utilisée. Cette normalisation n'empêcherait pas les nœuds concernés d'être trouvables à partir d'une recherche dans leur écriture d'origine grâce au mécanisme d'indexation qui permet de déclarer des chaînes à indexer différentes du contenu réel du nœud.

Cependant, les mots écrits en katakana le sont souvent à cause de leur origine étrangère non chinoise. De fait, leur morphologie est distincte des autres mots de la langue et par conséquent on peut douter que de nombreux nœuds soient fusionnés grâce à une normalisation. Cet argument linguistique permet donc de trancher entre deux positions techniquement réalisables.

4. Mais il serait tout à fait possible de le faire à l'aide du mécanisme d'annotation.

Relations

Trois types de liens vont être utilisés : *Vocabulaire3*, *Vocabulaire2* et *JaponaisKanji-Kanji*. Le premier est un hyperlien à 3 éléments (*Français*, *JaponaisKanji*, *JaponaisLecture*) formé à partir des lignes comportants une valeur pour les 3 colonnes. *Vocabulaire2*, qui relie des sommets *Français* et *JaponaisLecture*, est utilisé pour les cas où une écriture contenant des kanjis n'est pas présente. Enfin *JaponaisKanji-Kanji* relie un nœud *JaponaisKanji* à un nœud *Kanji*, dont le contenu est présent dans l'écriture du nœud *JaponaisKanji*. Plusieurs instances de ces liens peuvent être affectées à chacun des nœuds du graphe.

La figure 4.1 est un graphe généré à partir de certaines lignes de la fiche 26, consacrée aux minéraux, page 58 de [Peyrat, 2012]. Parmi les lignes de la page, celles des mots suivants sont utilisées : minéraux, métal, or, argent, pierre, calcaire et pierre précieuse.

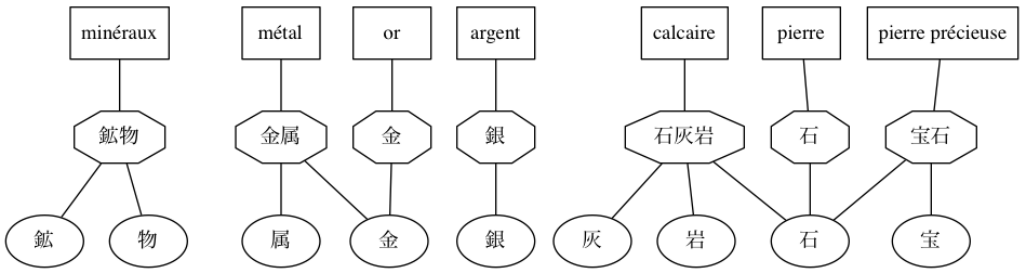


FIGURE 4.1 – Graphe lexical lié aux métaux et minéraux

Notons que les associations *Vocabulaire3* ne sont pas totalement représentées en figure 4.1 pour un souci de lisibilité. Elles sont graphiquement simplifiées en lien binaire entre les nœuds *Français* et *JaponaisKanji*, et les nœuds *JaponaisLecture* qui ont été écartés de la représentation ne sont pas affichés non plus.

Remarques

Le premier constat est que des nœuds de types différents peuvent avoir un contenu textuel identique ; c'est ici le cas des nœuds 金 (kane), 銀 (gin) et 石

(ishi) qu'on retrouve à la fois dans des instances *JaponaisKanji* et de *Kanji*. Comme ces nœuds représentent des objets lexicographiques différents, l'utilisation de types différents permet de les maintenir séparés ; on ne cherche en effet pas à rassembler dans un même nœud tout ce qui a une forme textuellement identique dans tout le graphe, mais seulement ce qui l'est au sein d'un même type.

Le second est que les nœuds *JaponaisKanji* n'ont aucun lien entre eux et les nœuds *Kanji* non plus, et pour cause : les types de liens définis ne permettent pas de créer de telles associations. On a ici des nœuds *JaponaisKanji* indirectement reliés à d'autres nœuds *JaponaisKanji* par l'intermédiaire d'un, mais possiblement plusieurs, nœuds *Kanji*. Ce sous-ensemble du graphe ne comportant que les nœuds *JaponaisKanji* et *Kanji* et les liens *JaponaisKanji-Kanji* est un graphe biparti.

Intérêt de la modélisation

Un programme peut tirer profit de ces liens pour l'affichage d'informations complémentaires à l'apprenant. Ainsi, on pourrait imaginer un logiciel qui s'en servirait pour lister les mots japonais comprenant les mêmes kanjis que celui qu'il est en train de consulter, d'apprendre ou de réviser.

La figure 4.2 illustre un système de recommandation automatique basé sur le graphe. À partir du mot japonais 宝石 (hōseki), il affiche les informations directement liées (pierre précieuse et 宝). Puis il suit le lien *JaponaisKanji-Kanji* vers le nœud *Kanji* 石, qui contient lui même trois liens *JaponaisKanji-Kanji* vers des nœuds *JaponaisKanji*. Un de ces liens a déjà été traversé (celui pour arriver sur le nœud), les deux autres liens permettent d'atteindre les nœuds *JaponaisKanji* 石灰岩 et 石, dont les informations liées sont à leur tour affichées en tant que recommandations.

Un tel système, couplé avec la possibilité de naviguer en cliquant sur les données tel que celui qui existe dans la fiche générique (3.6.1), permet de se rendre rapidement et facilement sur les fiches des termes reliés à celui consulté. Il est donc plus facile de prendre connaissance de l'existence de mots d'un même champ lexical et de consulter les détails de leur définition.

宝石　ほうせき　Pierre précieuse
Autres mots contenant les mêmes kanjis :
石　せき　Pierre
石灰岩　せっかいがん　Calcaire

FIGURE 4.2 – Illustration d'un système de recommandation

L'apprenant peut donc avoir accès à d'autres mots, souvent du même champ lexical, et par ce moyen apprendre des mots proches ou à tout le moins être informé de leur existence. Cela peut être difficilement apparent, notamment dans le cas de grandes listes de vocabulaire ou de listes triées selon la prononciation japonaise, où ces mots pourraient ne même pas être sur la même page.

Par rapport à une fiche de dictionnaire informatique classique, la présence de ces liens est un avantage : elle permet de présenter les mots liés et de les rendre cliquables pour accéder à la fiche concernée. Dans le dictionnaire japonais de référence sur iPhone, Imiwa, les kanjis d'un mot (composé) sont affichés et mènent vers la fiche dédiée à ce caractère (on observe bien là qu'on est en présence de deux fiches de type différents ce qui nous conforte dans l'approche observée, qui fait aussi usage de typage), il faut cliquer sur un second lien, « Compounds », afin d'avoir accès aux composés. Mais ce programme triche en quelque sorte, dans le sens où il se contente en fait de lancer une nouvelle recherche ; ce qui peut mener à quelques imprécisions dans les résultats.

Pour aller plus loin

On s'est pour l'instant contenté de modéliser des informations qui étaient toutes présentes dans le livre. Mais on pourrait compléter le graphe obtenu à l'aide d'informations externes avec pour objectif d'augmenter la densité des liens du graphe. La possibilité de représenter et de fusionner des informations de sources hétérogènes est d'ailleurs un des intérêts de la modélisation sous forme de graphe.

On remarque que beaucoup de kanjis partagent une même clef : 金 (kin, métal). Si on avait des nœuds permettant de représenter des portions de sinogrammes, on pourrait établir de nouveaux liens, qui permettraient de relier, par transitivité les kanjis 鉱 (kō, minerai), 金 (kin, or), 銀 (gin, argent) et 銅 (dō, cuivre), qui partagent tous cette clef dans leur structure.

[Lecailliez, 2015b] [5] propose un graphe de décomposition des caractères. On peut l'intégrer au présent graphe en ajoutant un nouveau type de nœud et un nouveau type de lien. Le type de nœud *Composante* représente une portion de caractères qui n'existe pas en tant que kanji ; au contraire des portions 金 et 広 contenue dans 鉱 qui existent sous forme de sinogrammes propres.

Le type de lien, orienté, *Décomposition* relie un nœud *Kanji* à un ou des nœuds *Composante* ou deux nœuds de type *Composante*. On obtient alors le graphe de structure présenté en figure 4.3.

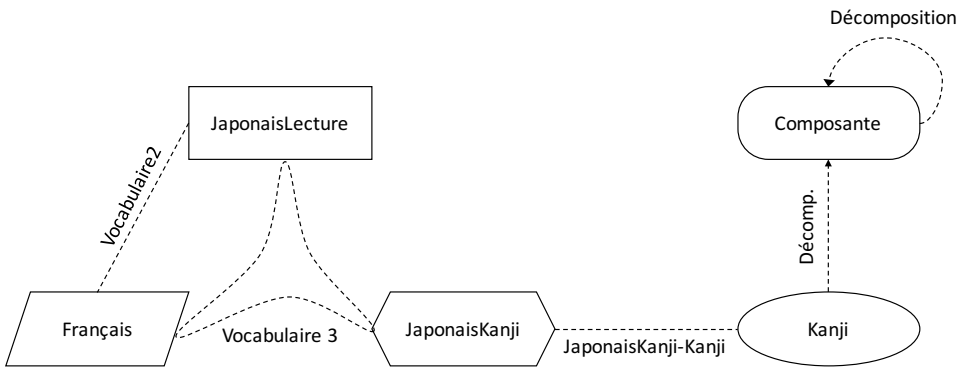


FIGURE 4.3 – Nouveau graphe de structure

Appliqué aux mots or et argent, on obtient le graphe d'instances présenté en figure 4.4.

5. P. 32-34.

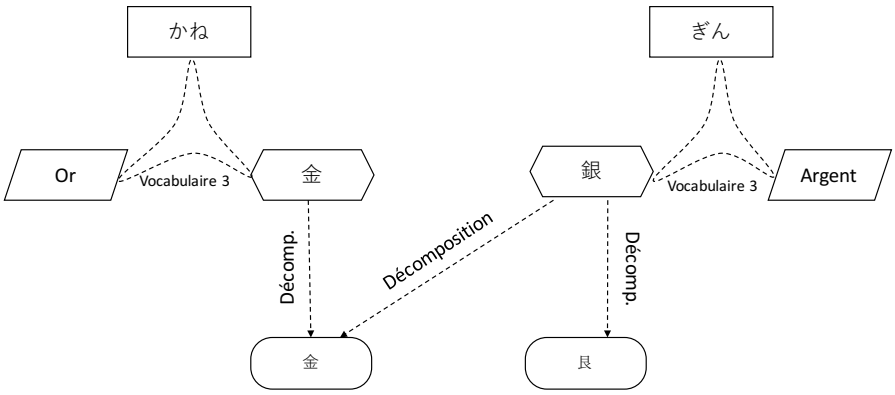


FIGURE 4.4 – Graphe d'instance

Une mise à jour de plus

Enfin on note que plusieurs des kanjis présents dans le figure 4.2 sont sous leur forme japonaise simplifiée (新字体, shinjitai). Or il peut être utile de connaître leur forme traditionnelle (旧字体, kyūjitai) quand on lit des textes datant d'avant 1945 dont l'écriture n'a pas été modernisée.

On pourrait créer un nouveau lien orienté nommé *Traditionnel* qui relie deux kanjis, le second étant la forme traditionnelle du premier. Pour obtenir les données nécessaires à l'ajout des liens en forme traditionnelle on peut utiliser la liste des jōyō kanjis fournie par Wikipédia [6] qui indique les formes traditionnelles dans une colonne dédiée.

Cependant l'utilisation d'une annotation pourrait être nécessaire pour déterminer directement si un caractère est sous forme traditionnelle ou non.

4.1.4 Résumé

Cette première tentative de modélisation d'un ouvrage de vocabulaire japonais, en apparence assez simple, a permis d'illustrer comment la modélisation sous forme de graphe peut se réaliser en pratique.

Il n'existe pas de solution unique ou de méthode toute faite pour passer d'un jeu de données lexicographiques à un modèle en graphe qui ait du sens et une utilité pour ce que l'on projette d'en faire. En effet, le premier modèle

6. `https://en.wikipedia.org/wiki/List_of_joyo_kanji`

proposé pêchait par manque de relations entre les données, qui sont pourtant une composante essentielle d'un graphe, ce qui ne permettait pas de tirer profit du modèle obtenu.

Un second modèle a pu être établi. Pour éviter l'écueil précédent, celui-ci a introduit un type qui n'existait pas « naturellement » dans l'ouvrage. De cette façon, les données ont commencé à être reliées entre elles. La création de ces liens est directement exploitable par des applications pour fournir, par exemple, des recommandations aux utilisateurs. On a donc déjà ici un gain net par rapport à une modélisation traditionnelle, qui ne permet de réaliser ce type de système que plus difficilement, et de manière moins générale.

Enfin, nous avons vu que le modèle pouvait être très facilement étendu pour établir des liens de nature nouvelle. Pour cela, des données externes au livre de référence utilisé ainsi que d'autres travaux universitaires ont été mobilisés et ont pu être ajoutés à la représentation avec une grande facilité. L'ajout de ces nouvelles données n'a toutefois pas d'impact sur une application qui n'en aurait pas besoin grâce au typage des objets du graphe qui permet de sélectionner et de n'utiliser que les portions de données dont a besoin une application.

4.2 Implémentation d'un Pokédex

4.2.1 Description rapide

Afin de prouver que le modèle est adaptable à des données très diverses, penchons nous sur un phénomène culturel originaire du Japon : Pokémon. L'ensemble des noms des monstres de poche de la franchise possède un faisceau de caractéristiques qui le rend intéressant à utiliser comme illustration. Il est de taille connue et restreinte sans être trivial (à présent 721), multilingue avec une correspondance un à un entre chaque traduction [7], et est très bien documenté.

De plus, de nombreuses données non linguistiques y sont associées : un numéro d'index dans le Pokédex national [8] qui permet de les identifier sans ambiguïté entre les langues et les versions du jeu, un ou deux types (eau,

7. Pour l'ensemble des langues considérées dans cet exemple. En effet avec la prise en compte du chinois, cela ne serait plus vérifié.

8. Il existe différents sous-ensembles du Pokédex national qui ordonnent les Pokémon d'une autre manière, en fonction des différentes versions (générations) du jeu principal.

sol, ténèbres, ...), les attaques qu'il peut apprendre, une ou plusieurs images le représentant, etc. la liste est très longue et des projets se voulant le plus complet possible sur la question on été initiés.

Mais ce qui nous intéresse ici, c'est le réseau multilingue formé par ces noms, les liens qu'on peut établir entre eux et les liens qu'on peut établir avec le reste du lexique d'une langue.

4.2.2 Micro-structure

Puisqu'il ne s'agit pas ici d'un ouvrage à proprement parler, il n'y a pas de micro-structure préexistante à analyser. On pourra toutefois se référer à la page [POKÉPÉDIA, 2007], qui liste les Pokémon dans l'ordre du Pokédex National dans plusieurs langues.

4.2.3 Modélisation en graphe

Le numéro unique d'un Pokémon n'est pas une donnée lexicographique pertinente ni indépendante, elle ne sera donc pas incluse directement dans le graphe. Nous voulons représenter les noms des monstres dans plusieurs langues, à savoir le japonais, le français et l'anglais.

Types de nœuds

La question qui se pose est de savoir quels sont les types à utiliser pour représenter la situation suivante : le nom d'un même Pokémon dans plusieurs langues. En l'état du modèle proposé, le seul moyen est de créer des types de nœuds différents. Un type de nœud par langue est crée : *PokémonFrench*, *PokémonEnglish*, *PokémonJapanese* respectivement pour les noms français, anglais et japonais.

On remarque que si on disposait d'une propriété *Langue* sur les nœuds, on pourrait n'avoir qu'un type unique, par exemple *PokemonName* et l'utiliser pour toutes ces langues en fixant cette propriété différemment en fonction de la langue concernée. C'est donc une extension du modèle qui pourrait être envisagée à l'avenir [9].

L'ajout d'une nouvelle langue se fera donc, si nécessaire, par l'ajout d'un nouveau type, associé à la langue en question.

9. En réalité c'est une composante qui a été retirée du modèle du présent mémoire pour le simplifier.

Contenu des nœuds et indexation

Les instances de nœuds des trois types déclarés précédemment ont pour nom canonique le nom du monstre qu'ils représentent. Le type est fixé en fonction de la langue associée au nom. Ces nœuds ne contiennent pas de propriétés privées.

Les nœuds de type *PokémonFrench* et *PokémonEnglish* exposent deux chaînes indexables : la première correspond à leur nom canonique et la second à ce même nom avec remplacement des lettres capitalisées en lettres minuscules. Les chaînes indexées par les nœuds des trois types sont placées dans des index distincts. Cela permet d'offrir une recherche par langue tout en permettant de créer un index global si besoin est.

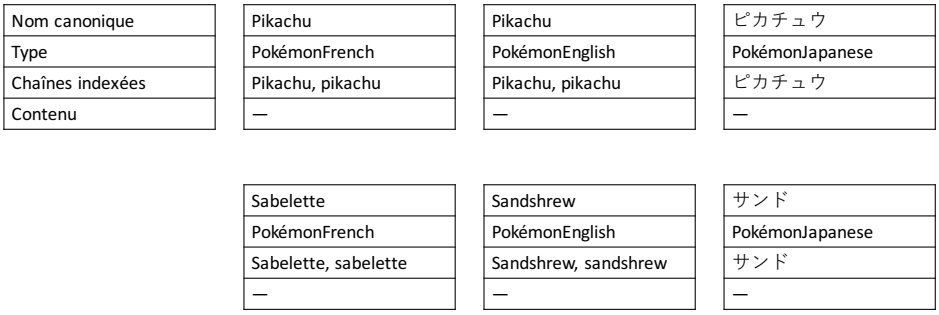


FIGURE 4.5 – Définition d'un nœud et six instances de nœuds

L'illustration précédente rappelle la structure d'un nœud et présente six instances de nœuds correspondant à deux créatures. On remarque que pour Pikachu, les instances en français et en anglais ne se distinguent, au niveau de l'information présente, que par le type qui leur est associé.

Lien de traduction

Afin de pouvoir accéder à la fiche d'un Pokémon dans les autres langues à partir d'un nœud donné, on crée la relation *PokémonTranslation* munie des contraintes suivantes : arité 2, nœuds liables (*PokémonFrench*, *PokémonEnglish*) ou (*PokémonEnglish*, *PokémonJapanese*) ou (*PokémonFrench*, *PokémonJapanese*).

On remarque tout de suite que la contrainte qui permet de relier deux à deux les traductions entre elles comporte un nombre de clauses fonction du

nombre de type de nœuds de langue et que ce nombre est égal au nombre d'arêtes dans un graphe complet K_n où n est le nombre de type de nœuds en question [10]. C'est donc une limitation évidente de cette façon de procéder, à moins de générer automatiquement la liste de contraintes dans le cas où plus de langues seraient à gérer.

Limite à l'ajout d'une langue

On remarque un autre problème : l'ajout d'une nouvelle langue nécessite la modification de la contrainte qui permet de lier ces nœuds. Mais cela n'est pas un scénario supporté par le modèle. En effet une fois qu'un type est crée et ses contraintes fixées, le type et ses dernières sont immutables. Permettre le contraire casserait la comptabilité avec l'existant lors d'un changement de contraintes.

Il y a deux moyens de régler le problème : soit créer un type de relation par paire de type de nœuds à relier, et donc trois dans cet exemple. L'avantage de cette façon de procéder est que la relation indique clairement, si elle est orientée, dans quel sens s'effectue la traduction, bien que cela soit déductible du type parcouru en premier et de celui atteint en traversant la relation.

L'autre possibilité, qui serait possible avec la propriété de langue énoncée plus haut, serait d'avoir une unique relation liant deux nœuds d'un unique type PokémonName. La contrainte serait alors que la langue des nœuds liés soit différente. Cela complexifierait donc également le système de contraintes en plus de la définition d'un nœud. Le gain étant important en terme de facilité d'utilisation, c'est une évolution envisageable du modèle.

Nouveaux types de liens

En plus de la relation de traduction déjà décrite il existe d'autres relations intéressantes à modéliser. Au sein d'un même type on définit le lien *Suivant* qui indique quel est le Pokémon suivant un autre dans l'ordre du Pokédex, ce qui est utile car les Pokémon qui sont les évolutions les uns des autres sont souvent (mais pas toujours) indexés de manière contiguë. Cela nous mène à deux autres types de relation : *Précédent*, qui relie le monstre précédent dans l'ordre du Pokédex et *Évolution*, qui indique qu'un Pokémon évolue à partir d'un autre.

On se retrouve alors avec le graphe de structure de la figure 4.6.

10. C'est à dire $\frac{n(n-1)}{2}$.

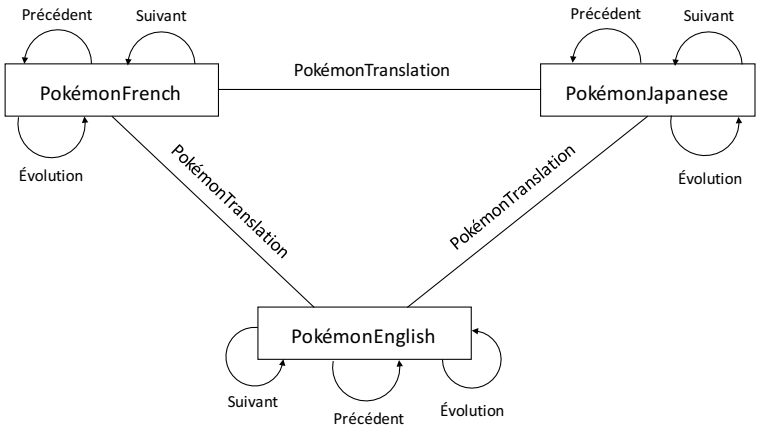


FIGURE 4.6 – Graphe de stucture du Pokédex

À partir de ce graphe, on peut implémenter une application qui affiche automatiquement les nœuds reliés à l'instance du nœud courant. Ainsi la figure 4.7 représente une fiche de dictionnaire générée automatiquement à partir du nœud « Raichu », qui est relié respectivement aux nœuds déjà décrits Pikachu et Sabelette par les relations *Précédent* et *Suivant*.

Comme c'est le nœud français qui est affiché, les traductions sont en anglais et japonais [11]. De même, les relations qui pointent vers les monstres suivant et précédent le font vers d'autres instances de *PokémonFrench*, conformément à ce qu'indique le graphe de structure.

Chacune des informations présentes sur la capture d'écran de ce prototype d'application mobile affiche le nom canonique d'un nœud. Tous sont cliquables et permettent de se rendre à la fiche correspondante.

11. Une implémentation commerciale devrait probablement comporter une indication de la langue des traductions pour éviter les ambiguïtés.

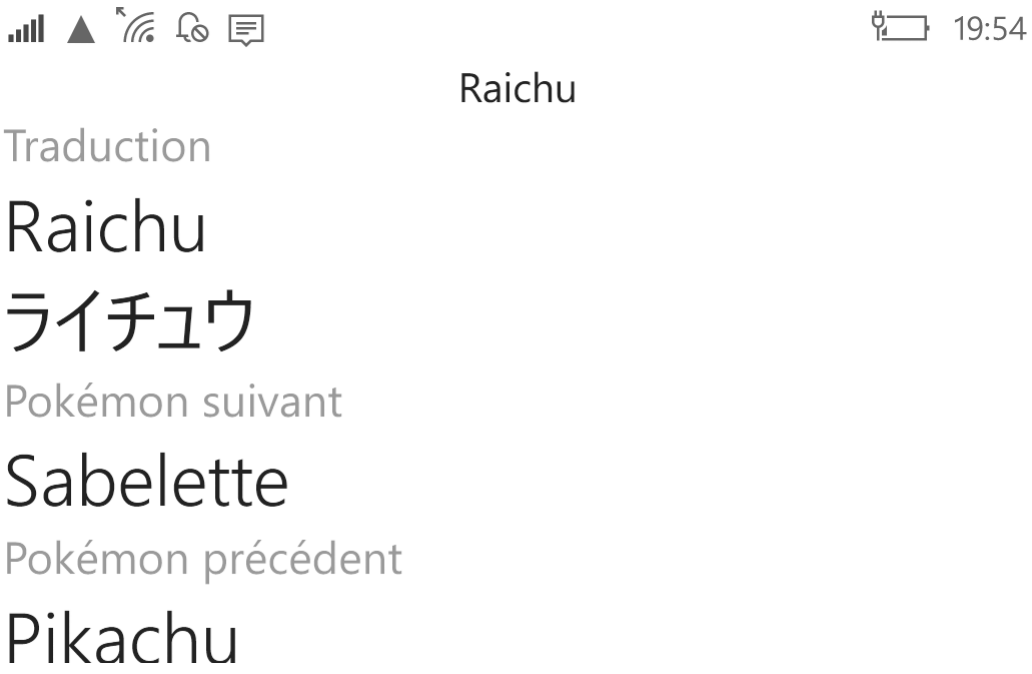


FIGURE 4.7 – Fiche d'un Pokémon dans le prototype d'application mobile

Mise à jour de la modélisation

Dans le paragraphe précédent, trois relations ont été successivement décrites, dont deux qui ont été imaginées à partir de l'énonciation d'une première. Dans un scénario lexicographique on peut également vouloir rajouter des relations à un modèle existant mais l'intervalle de temps entre la création d'un modèle et sa mise à jour peut être bien plus long. Le modèle démontre ici un de ses avantages : le fait d'avoir conçu et de vouloir mettre en place une nouvelle relation ne nécessite pas de toucher à l'existant.

Dans un modèle de structuration plus classique des données, où un Pokémon serait représenté par l'instance d'une classe (éponyme, tant qu'à faire) cela aurait nécessité la modification de la classe pour y ajouter un champ pointant vers le monstre précédent et d'un champ pointant vers ses évolutions. Ce genre de modifications nécessite *a minima* une recompilation du

programme et le plus généralement une migration du modèle de stockage des données. C'est donc une opération beaucoup plus lourde que celle qui est permise ici [12].

Intégration dans un graphe existant

On voit que ces données sont munies d'un certain nombre de relations internes, on peut donc naviguer entre elles et elles forment une composante connexe du graphe (2.2.1) tout à fait exploitable de manière autonome. Toutefois, qu'en est-il de son intégration dans un graphe plus large ? En l'état, il n'y a pas de relation avec des entités plus générales, des termes français par exemple. De fait on ne pourra atteindre des nœuds de cette composante qu'en arrivant directement sur un de ses nœuds via une recherche, et réciproquement on ne pourra pas en sortir simplement en naviguant à travers les relations accessibles depuis les nœuds correspondant à des noms de Pokémon.

Posons qu'il existe un graphe muni de deux types *Français* et *Japonais-Lecture* déjà définis en 4.1.3 **Modélisation en graphe** qui représentent respectivement des lexies françaises et japonaises. On cherche à relier certaines des instances de ces types à des nœuds de notre composante composée de noms de Pokémon. Ces noms sont le plus souvent tirés de jeux de mots, qui impliquent un ou plusieurs mots existants dans le lexique de ces langues (et parfois dans celui d'une autre langue).

On peut donc créer une notice explicative de l'étymologie en question, contenu dans un type *Étymologie*, et relier celle-ci d'une part au nom qui est expliqué, d'autre part aux lexies impliquées dans cette étymologie. De cette manière on crée un nombre significatif de liens vers des nœuds n'appartenant pas à l'origine à la composante et celle-ci se retrouve désormais reliée au reste du graphe.

12. Évidemment il n'est pas équitable de comparer le comportement d'un système réel avec la procédure à employer pour modifier un modèle théorique. Cependant le dit modèle peut être implémenté avec un système de plugins permettant d'éviter une recompilation et l'ajout *a posterio* de relations ne nécessite pas une migration des données existantes, d'où le fait qu'une telle comparaison ait été faite ici.

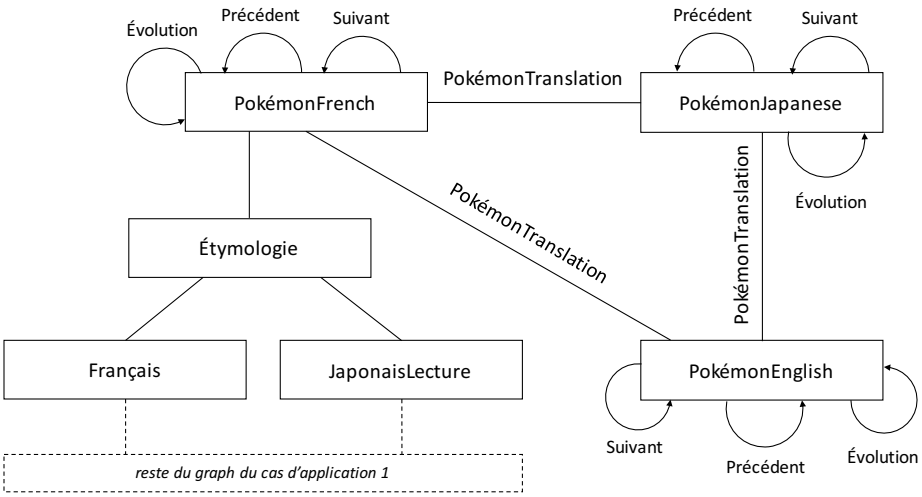


FIGURE 4.8 – Graphe de structure avec étymologie et liens vers un autre graphe

Le reste du graphe auquel appartient à l'origine les types *Français* et *JaponaisLecture* peut être aussi complexe que les différentes extensions qui ont été proposées. Ce qui importe ici c'est que ces types servent de jonction entre les deux graphes des deux cas d'application. Cela montre qu'on peut raisonner indépendamment sur deux types de graphe puis tenter de les rassembler pour former un graphe plus important.

Affichage

La chaîne *PokémonFrench*, *Étymologie*, *Français* (ou *JaponaisLecture*) du graphe de structure présente une particularité liée à son affichage : dans la fiche générée pour le monstre en question, on pourrait souhaiter afficher et rendre cliquable les lexies à l'origine de l'étymologie afin de pouvoir se rendre immédiatement sur les fiches associées. C'est cependant impossible avec un logiciel qui ne ferait que collecter les nœuds adjacents car les données d'étymologie font tampon entre ces deux types. En générant une fiche à partir des relations directement liées à un sommet, on ne parvient pas à l'effet escompté.

Dans l'autre sens, en partant d'une lexie pour aller vers un Pokémon (on souhaite lister les Pokémon dont le nom est formé à partir de cette lexie sur

sa fiche) on est interrompue par le type *Étymologie* qui n'a pas grand intérêt à être affiché sur une fiche instance de Lexie car c'est une information qui « appartient » au nom du Pokémon. Dans ce cas là, un lien orienté partant de l'étymologie vers les lexies serait plus approprié.

Il faudrait donc un mécanisme pour pallier à cet inconvénient qui existe du point de vue des deux types (*Pokémon* et *Étymologie*) concernés, mais s'y manifeste différemment. Enfin, une fiche dédiée à l'étymologie en tant qu'entrée lexicographique indépendante n'a pas grand sens. Toutes ces problématiques peuvent cependant être adressées par un client programmé spécifiquement pour un graphe structuré de cette façon. Il n'est pas donc pas nécessaire de détailler ici plus en détails une solution qui ne s'applique pas au cas général.

Ajout d'images

Finalement, on peut proposer une amélioration supplémentaire au graphe qui nous sert ici d'illustration : fournir une image pour chaque monstre. Étant un produit cross-média principalement établi sur des supports non textuels (jeux vidéos, dessins et films animés, cartes à jouer, ...) la franchise Pokémon a une composante visuelle importante qui lui est indissociable. Un consommateur peut donc légitimement s'attendre à ce que l'ensemble des données contenues et présentées par un logiciel relatif à Pokémon puisse afficher une image des créatures en question.

Conceptuellement, il est très simple d'ajouter cela à notre modèle : on crée un nouveau nœud et une relation qui permet de lier un nœud contenant un nom (peut importe la langue) et le nœud d'image. Le nouveau graphe de structure est visible en figure 4.9.

Comme l'image n'a pas vocation à être une entrée indépendante du dictionnaire final, le nœud de type *PokémonImage* ne renvoie aucune chaîne à indexer. En outre, les clients qui affichent le graphe peuvent être conçus pour ne pas permettre la navigation vers l'image, ce qui nécessiterait de pouvoir la considérer comme une vedette.

Le nom canonique du nœud peut contenir une description de l'image de façon à ce que son contenu puisse être compris sans avoir à afficher l'image elle-même. Les données elles-mêmes sont contenues dans le champ « contenu » du nœud [13].

13. Dans le prototype d'application mobile, qui ne gère que des nœuds ayant un contenu

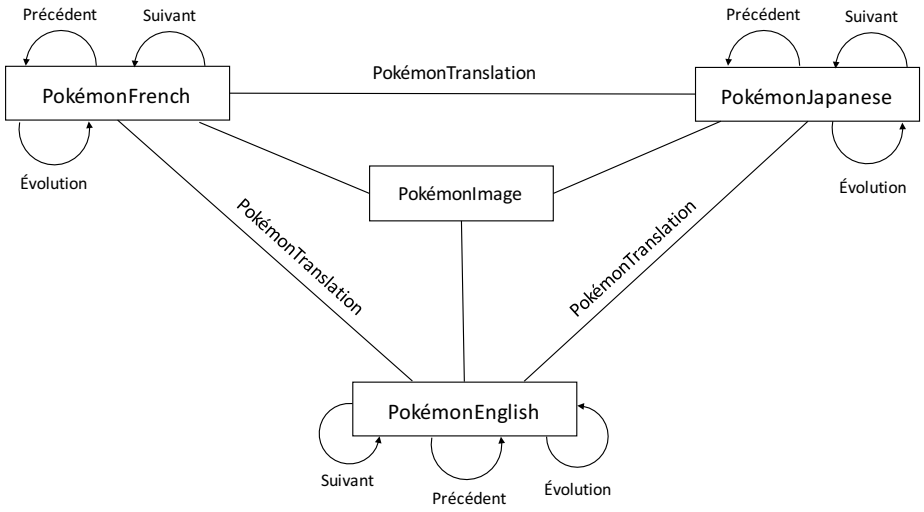


FIGURE 4.9 – Graphe de structure avec ajout d'une image

4.2.4 Résumé

Ce second cas d'application a permis de montrer que le formalisme est applicable à des données qui ne sont pas *stricto sensu* des données lexicographiques. Il est ainsi possible de modéliser des relations extra-linguistiques, ce qui est utile pour la réalisation de dictionnaires encyclopédiques notamment.

Une nouvelle fois, il a été illustré qu'ajouter de nouveaux types de nœuds ou de liens était facile et ne nécessitait pas de modifier les types existants. L'atomicité des liens qui cachent les détails d'implémentation permet aussi d'ajouter des éléments multimédias (ici des images) sans problème particulier.

De plus, l'extensibilité est également présente sous la forme de la capacité à interfacer un graphe avec un autre graphe existant modélisé selon le même formalisme (un graphe de lexies du français dans ce cas d'application).

Il a en outre était mis en lumière une faiblesse actuelle du modèle : la gestion de plus de deux langues. En outre, l'ajout postérieur à la définition d'un graphe de structure d'une nouvelle langue nécessite de modifier les contraintes de certains types de relations (ce qui n'est normalement pas autorisé) ou de

textuel, les données binaires d'une image PNG sont converties en texte en code Base64. Ces données sont ensuite décodées et affichées sous forme d'image par un composant dédié.

créer de nouveaux types de relations, ce qui n'est pas satisfaisant.

Enfin, la problématique de l'affichage final doit être prise en compte dès la conception du graphe si on souhaite obtenir un résultat satisfaisant avec un algorithme simple tel que celui de la fiche générique. On peut utiliser des mécanismes du modèle en graphe comme liens orientés ou laisser à l'application cliente l'entière responsabilité de produire un affichage ou une sortie cohérente.

4.3 Nœuds complexes

4.3.1 Description rapide

La simplicité des éléments de base qui forment le graphe n'implique pas que l'usage qui en soit fait se limite à des situations basiques. Pour illustrer le fait, énoncé dans la section 3.3 **Nœuds**, que la structure interne des nœuds peut être arbitrairement complexe pour coller aux besoins d'une application, nous allons nous pencher sur la modélisation de phénomènes phonologiques absent de l'écrasante majorité des dictionnaires de japonais : l'indication de la dévocalisation (母音の無声化, boin no museika) de voyelles et la présence d'une nasale vélaire (鼻濁音, bidakuon). Ces phénomènes sont expliqués en détails dans le livre de référence de [Labrune, 2006] sur la phonologie du japonais.

Dans un dictionnaire papier l'absence de ces données s'explique aisément par la cible et la portée du dit ouvrage : c'est généralement une information qui surcharge la page si elle n'est pas souhaitée par le lecteur. Dans un dictionnaire électronique, on peut stocker ces données tout en ne les affichant qu'à la demande de l'utilisateur. Dans ce cas là, c'est à la surcharge de paramètres dans la page de configuration de l'application qu'il faut faire attention pour ne pas submerger le lecteur.

4.3.2 Indication de la dévocalisation

Le problème est le suivant : on cherche à représenter les voyelles touchées par une dévocalisation dans un mot japonais. Conformément à ce qu'on trouve dans la littérature japonaise [Nihon hōsō kyōkai hōsō bunka kenkyūjō, 1998], le mot sera écrit en kana et c'est le kana contenant la voyelle dévocalisée qui

sera marqué par un artefact ou un artifice graphique, de façon à être différencié des autres voyelles non touchées par le phénomène.

Pour cela on peut utiliser une structure telle que présentée en figure 4.10.

Données privées
Mot écrit en kana
Indication de la dévocalisation

FIGURE 4.10 – Données internes à un nœud (dévocalisation)

Le premier champ n'appelle pas à explications complémentaires. Le second contient un moyen d'indiquer quels sont les kanas qui sont à modifier de façon à indiquer que la voyelle qu'ils contiennent est dévocalisée.

Représentation informatique

On pourrait passer par un tableau de taille identique à celle en caractère – et non en more, si on veut passer par une colorisation dans l'interface graphique – de la chaîne à annoter qui contient pour chaque position si le caractère est à colorer différemment ou non. Mais c'est un gâchis de mémoire et cela augmente la complexité de lecture du champ, nous allons donc passer par un vecteur de bits, qui a exactement la même fonction mais qui permet de représenter cela de manière plus compacte et sans avoir à instancier un tableau.

Prenons par exemple le mot がくせい (gakusei, étudiant), dont la voyelle /u/ qu'il contient est affectée par la dévocalisation de sa première voyelle. Lorsque le mot est écrit en kana, c'est le caractère く (ku) en deuxième position qui est touché. En binaire, la suite de bits correspondante est 0100 (c'est-à-dire 4). Cependant, on va plutôt compter les positions à partir des bits de poids faible, ce qui permet d'encoder la valeur de manière identique quelque soit la taille de l'entier qui sert réellement à stocker la valeur [14]. On a donc le vecteur 0010 (les bits de poids faible sont ceux à la fin du nombre), qui correspond à 2 en base décimale. La structure résultante est présentée dans la figure 4.11.

14. En effet, dans le cas contraire un mot de 5 caractères dont le second est touché par un dévoisement aurait pour masque 01000 c'est-à-dire 8.

Champ	Valeur
Type	*JapaneseWordWithVoicing*
Nom canonique	がくせい
Chaînes indexées	がくせい, 学生
Mot écrit en kana (champ privé 1)	がくせい
Masque de dévocalisation (champ privé 2)	2

FIGURE 4.11 – Nœud encodant un dévoisement de voyelle

Les deux champs privés de la figure 4.11 correspondent aux deux champs de la structure présentée en figure 4.10.

La figure 4.12 présente le type associé à ce nœud. On remarque qu'elle détaille la structure interne de celui-ci afin de la documenter pour les lexicographes qui travaillent sur le projet.

Élément	Valeur
Nom	JapaneseWordWithVoicing
Objet concerné	Vertex
Description	Représente un mot japonais écrit en kana avec indication des voyelles dévoisées. Ses données internes sont composées d'un champ écrit en kana et d'un champ contenant le masque binaire des caractères affectés par la dévocalisation. Le masque est à interpréter en prenant le bit de poids faible comme le premier caractère de la chaîne. La valeur 1 indique une dévocalisation, 0 indique l'absence de dévocalisation.
Identifiant	d6a4fbf7-c322-4b45-b492-fdfcabb32e6c

FIGURE 4.12 – Description du type *JapaneseWordWithVoicing*

Affichage

Le composant d'affichage [15] associé au type de nœud qui contient la structure peut ensuite s'en servir pour présenter l'affichage du dévoisement. Cela peut éventuellement même être paramétrable pour laisser le choix à l'utilisateur.

15. Cf. 3.6.4 **Composants d'affichage personnalisés** pour la question de l'affichage des données internes à l'aide de composants personnalisés.

FIGURE 4.13 – Dévocalisation indiquée

FIGURE 4.14 – Dévoc. non indiquée

La figure 4.13 présente un fragment d'écran d'application mobile où la dévocalisation est indiquée à l'utilisateur par l'usage de la couleur grise pour le caractère touché, au lieu du blanc pour le reste du mot. Le même mot sans cette indication est illustré en figure 4.14.

4.3.3 Résumé

Les données lexicographiques à représenter peuvent être beaucoup plus riches qu'une simple chaîne de caractères. Pour adresser ces cas, les nœuds du graphe peuvent contenir des données quelconques, dont l'interprétation et l'utilisation sont laissées libre aux programmes qui consomment le graphe. On peut ainsi fournir un affichage soigné, ou configurable dans une application finale.

Cette manière de faire nécessite que le contenu de chaque type de nœud soit décrit très précisément, en particulier s'il doit pouvoir être utilisé dans d'autres projets.

Ces propriétés internes sont exposées aux logiciels qui utilisent le graphe, mais pas aux nœuds ou aux liens eux-mêmes. Le retrait d'un nœud du graphe ne nécessite donc pas d'avoir à modifier les données internes de nœuds présents dans le graphe.

Chapitre 5

Conclusion

5.1 Limitations et problèmes non adressés

5.1.1 Gestion du multilinguisme

En l'état, on a vu que le modèle ne fournissait aucun mécanisme explicitement destiné à gérer le multilinguisme. Il n'y a donc pas de moyen standardisé d'associer une langue ou un couple de langue à des données du graphe (nœuds et liens) ou de traiter automatiquement les langues en présence.

Le second cas d'application (4.2 **Implémentation d'un Pokédex**) a illustré à la fois comment gérer trois langues dans un graphe de structure et détaillé plus précisément les limitations que cela impliquait.

Cependant une généralisation du modèle pour pouvoir traiter un plus grand nombre de langues, mais aussi, pourquoi pas, adresser la problématique des dialectes, devrait contenir un mécanisme simple et fiable pour résoudre ce problème.

En attendant une telle amélioration on peut toutefois utiliser le mécanisme d'annotation afin d'étiqueter les données pour indiquer les langues qui leurs sont associées ou définir des types liés aux langues comme il a été fait dans le deuxième cas appliqué.

5.1.2 Orientation des hyperliens

Deux types de lien simple sont définis : orienté et non orienté. Leur définition est relativement similaire à la différence du concept mathématique

sous-jacent pour définir les liens ; dans un cas il s'agit d'ensembles, dans l'autre de n-uplets.

Pour les hyperliens en revanche, il n'en existe qu'une seule sorte, qui correspond à des liens non orientés. Cette définition permet de répondre à la problématique de la gestion des caractères chinois et des mots écrits en kanjis qui peuvent comporter des lectures différentes associées à un sens lui aussi différent. Dans ce genre de modélisation les éléments concernés ont tous un type différent, ce qui permet de les traiter séparément.

Cependant elle n'est pas apte à représenter des situations plus complexes, notamment celles où des éléments de types identiques apparaissent et où leur ordre de traitement a une importance pour l'interprétation du lien.

Des hyperliens « ordonnés » permettraient ainsi de représenter des problématiques plus complexes. Par exemple celle de représenter une locution en quatre caractères (四字熟語, yojijukugo) sous forme d'un hyperlien qui relie les hyperliens qui rassemblent un kanji, sa lecture et son sens en français. Cette façon de procéder à l'avantage d'expliciter quel est le sens du caractère utilisé, en contexte dans l'expression. Elle nécessite cependant de pouvoir ordonner les relations ainsi liées, ce qui n'est pour l'instant pas possible [1].

Notons que la même technique est applicable [2] à des textes entiers en chinois classique – langue d'où provient nombre de ces idiomes – car l'écriture ne présente aucune variation morphologique. Dans le cas du japonais, ce serait plus compliqué à mettre en oeuvre directement.

5.1.3 Expressivité des contraintes

Tel que défini actuellement, le système de contraintes ne permet pas de spécifier certaines situations. En particulier le cas où un lien doit pouvoir relier plusieurs instances d'un type en plus d'une d'un autre type donné.

Par exemple, soit les types de nœuds T_{N_1} et T_{N_2} et le type de lien T_L. On cherche à spécifier que le type T_L relie deux instances de T_{N_1} et une instance de T_{N_2} mais cela est impossible. Avec le système de contraintes tel qu'il existe, en combinant une contrainte d'arité et deux contraintes de type, on autorise également le lien à relier une instance de T_{N_1} et deux de T_{N_2}.

1. Dans le modèle théorique exposé ici. Comme l'implémentation fait usage de listes pour stocker les liens vers les objets impliqués, les liens sont *de facto* ordonnés.

2. En fait, elle a déjà été mise en application sous une forme proche lors de la réalisation d'une base de données et d'un site web. Le rapport afférent n'est pas disponible en ligne mais peut être obtenu en me contactant.

En outre, les liens orientés ne sont pas gérés, ce qui pose un problème évident de spécification d'un graphe. La problématique de l'orientation peut être vue comme un cas particulier à deux éléments du cas général de l'ordonnancement des sommets exprimé en 5.1.2 **Orientation des hyperliens**.

On a donc tout intérêt à tenter d'adresser ce cas plus général lors de l'amélioration du système de contraintes.

5.2 Vue d'ensemble du modèle proposé

Nous avons abordé une façon de modéliser les dictionnaires sous forme de graphe. Plus précisément, la micro-structure d'une entrée peut y être exprimée formellement par un graphe de structure. Un dictionnaire-graphe (graphe d'instances) est calqué sur un graphe de structure : il en reprend la structure mais contient les données lexicographiques concrètes du projet.

En outre, les composants du graphe (nœuds et liens) possèdent des propriétés communes et connues d'avance : nom canonique, chaînes indexées, ... en plus d'être associées à un type. Les types eux-mêmes sont clairement définis, à la fois pour les humains avec une description précise de ce qu'il est censé représenter, et à destination des machines à l'aide d'un identifiant unique.

Cela constitue un ensemble de méta-données qui permet d'automatiser les traitements même dans le cas où les données elles-mêmes ne sont pas interprétables. Un programme peut donc tirer partie des informations qu'il peut décoder sans être interrompu dans sa navigation des liens présents par les types qu'il ne sait pas gérer.

Enfin, le modèle permet une grande flexibilité d'usages à l'aide d'un modèle de représentation des données standardisé. Des programmes peuvent être réalisés pour des usages très différents : compilation d'un ouvrage pour le papier, calcul de proximité sémantique, application mobile, ...

Dans le cas d'une application de dictionnaire informatisée, le système permet de manière quasiment « clefs en main », une fois implémenté, d'offrir une expérience de navigation inédite. Celle-ci permet d'exposer clairement des relations existantes entre les mots d'un lexique et les mots de différentes langues et de permettre à un apprenant de naviguer grâce à elles entre les termes au gré de ses envies.

5.3 La lexie de la fin

Outre les avantages du côté du lecteur final en terme de navigation, ce modèle propose également une nouvelle façon de faire les dictionnaires, par une sorte d'inversion du processus lexicographique.

Généralement, les lexicographes établissent une micro-structure pour le dictionnaire. Une fois établie, il tentent de créer les premières fiches. Cette étape permet de soulever des problèmes ou des ambiguïtés dans la micro-structure, qui est alors modifiée pour prendre en compte les solutions proposées. Les fiches sont alors mises à niveau vers le nouveau format, et d'autres sont créées. À nouveau des problèmes peuvent émerger, nécessitant une correction de la structure. Après plusieurs itérations de ce cycle, la micro-structure est gelée et la création en volume de fiche peut commencer. La définition de la micro-structure a donc préséance sur la création de fiches et est bloquante : tant qu'elle n'est pas entièrement terminée, la création de fiche en masse ne peut pas commencer.

Cette approche est bien entendu toujours employable avec le modèle exposé ici. En revanche, une autre est également possible. Dans celle-ci les données sont répertoriées avant (ou pendant et après) le processus de définition de la micro-structure, c'est-à-dire du graphe de structure du projet. L'ajout de données qui ne seront finalement pas utilisées n'est pas un problème, car elles peuvent être conservées pour des projets ultérieurs sans parasiter celui en cours. La conversion des données vers un nouveau format est facilitée par le fait que tout est déjà documenté, finement discriminé et inséré dans un système informatisé. Le processus lexicographique est donc inversé : on part de données existantes et présentes dans le graphe, et on construit la partie de la micro-structure manquante sous forme de liens, l'autre étant celle constituée par les sommets.

La création d'une fiche est elle aussi modifiée de manière semblable. Dans un système classique, une fiche n'est réellement ajoutée au projet que quand ses champs obligatoires sont remplis. Auparavant, des données la concernant peuvent exister de manière informelle hors du système informatisé destiné à générer le dictionnaire final : notes de recherche, fichiers Word, etc. Ici l'information, même incomplète du point de vue d'une fiche, est ajoutée au dictionnaire en incubation sous forme de nœud dès que possible, puisqu'elle ne dépend pas directement d'autres données. Les avantages tirés sont la disponibilité pour d'autres parties du dictionnaire qui en auraient besoin, et la visibilité accrue de l'avancement du travail puisque les données ne sont plus

enfouies dans les documents personnels des collaborateurs.

Finalement, le processus lexicographique s'étend jusqu'à l'utilisateur final. Puisque les données sont finement égrainées, une application a tout loisir de lui permettre de modifier les informations et leur arrangement dans les fiches pour obtenir celle dont il a exactement besoin. Le dictionnaire personnel est à portée de main grâce à son versant technique, le dictionnaire-graphe.

Table des figures

Bibliographie

[Bondy et Murty, 2007] Bondy, J.-A. et Murty, U. S. R. (2007). *Graph theory*. Graduate texts in mathematics. Springer, New York, London. OHX.

[CRISCO, 1998] CRISCO (1998). [Site web ; accédé le 16 jui. 2016] `http://www.crisco.unicaen.fr/des/synonymes/`.

[Doan-Nguyen, 1998] Doan-Nguyen, H. (1998). *Techniques génériques d'accumulation d'ensembles lexicaux structurés à partir de ressources dictionnairiques informatisées multilingues hétérogènes*. Thèse de doctorat, Institut national polytechnique de grenoble.

[François et Manguin, 2004] François, J. et Manguin, J.-L. (2004). Le dictionnaire Électronique des synonymes du crisco – un mode d'emploi à trois niveaux. *Cahier du CRISCO 17, université de Caen [avec la collaboration de R. Eufe, L. Fesenmeier, C. Ozouf et M. Sénéchal]*.

[Gader *et al.*, 2012] Gader, N., Lux-Pogodalla, V. et Polguère, A. (2012). Hand-Crafting a Lexical Network With a Knowledge-Based Graph Editor. *In* Committee, T. C. . O., éditeur : *Third Workshop on Cognitive Aspects of the Lexicon (CogALex III)*, pages 109–125, Mumbai, India.

[Haig *et al.*, 1997] Haig, J., Nelson, A., of Hawaii at Manoa. Dept. of East Asian Languages, U. et Literatures (1997). *The new Nelson Japanese-English character dictionary : Shinpan Neruson Kan-Ei jiten*. C.E. Tuttle Co.

[Koh *et al.*, 2015] Koh, K. M., Dong, F., Ng, K. L. et Tay, E. G. (2015). *Graph theory. Undergraduate mathematics*. World Scientific Publishing.

[Labrune, 2006] Labrune, L. (2006). *La phonologie du japonais [version électronique révisée]*. Peeters.

[Larousse, 2016] Larousse (2016). [Site web ; accédé le 20 avr. 2016] http://larousse.fr/dictionnaires/francais/dictionnaire/25356?q=dictionnaire#25239.

[Lecailliez, 2015a] Lecailliez, L. (2015a). Approches pour une numérisation de qualité d'un dictionnaire vietnamien-français comprenant des caractères nôm.

[Lecailliez, 2015b] Lecailliez, L. (2015b). Méthode automatisée pour la recherche de caractères chinois complexes dans un dictionnaire électronique basée sur leur décomposition structurelle. Dossier de master 1, Université Paris Diderot.

[Mangeot, 2004] Mangeot, M. (2004). *Environnements centralisés et distribués pour lexicographes et lexicologues en contexte multilingue*. Thèse de doctorat, Université Joseph-Fourier - Grenoble I.

[Mel'čuk *et al.*, 1995] Mel'čuk, I., Clas, A. et Polguère, A. (1995). *Introduction à la lexicologie explicative et combinatoire*. Champs linguistiques. Duculot.

[Miller, 1995] Miller, G. A. (1995). Wordnet : A lexical database for english. *Communications of the ACM Vol. 38, No. 11 : 39-41*.

[MSDN, 2016] MSDN (2016). [Site web ; accédé le 14 jui. 2016] https://msdn.microsoft.com/en-us/library/ee658109.aspx?f=255&MSPPError=-2147217396.

[Müller, 2012] Müller, D. (2012). *Introduction à la théorie des graphes*. Cahier CRM.

[Nihon hōsō kyōkai hōsō bunka kenkyūjō, 1998] Nihon hōsō kyōkai hōsō bunka kenkyūjō (1998). *NHK nihongo atsuon akusento jiten shinban*. Nihon hōsō shuppan kyōkai.

[Peyrat, 2012] Peyrat, J. (2012). *100 fiches de vocabulaire japonais*. Studyrama.

[Pleco Software, 2000] Pleco Software (2000). [Site web ; accédé le 25 août 2016 https://www.pleco.com/about/].

[POKÉPÉDIA, 2007] POKÉPÉDIA (2007). [Site web ; accédé le 11 mai 2016] http://www.pokepedia.fr/Liste_des_Pok%C3%A9mon_dans_l'ordre_du_Pok%C3%A9dex_National.

[Polguère, 2014] Polguère, A. (2014). From Writing Dictionaries to Weaving Lexical Networks. *International Journal of Lexicography*, 27(4):396–418.

[Polguère, 2012] Polguère, A. (2012). Lexicographie des dictionnaires virtuels. *In* Apresjan, Y., Boguslavsky, I., L'Homme, M.-C., Iomdin, L., Milićević, J., Polguère, A. et Wanner, L., éditeurs : *Meanings, Texts, and Other Exciting Things. A Festschrift to Commemorate the 80th Anniversary of Professor Igor Alexandrovič Mel 'čuk*.

[Ricci, 2001] Ricci, I. (2001). *Grand dictionnaire Ricci de la langue chinoise*. Desclée de Brouwer, Paris.

[Spohr, 2012] Spohr, D. (2012). *Towards a Multifunctional Lexical Resource*, volume 141. De Gruyter.

[Tanaka-Ishii *et al.*, 1998] Tanaka-Ishii, K., Umemura, K. et Iwasaki, H. (1998). Daisangengo wo kaishita taihyakujiten no sakusei [Construction of Bilingual Dictionary Intermediated by a Third Language]. *Jōhō shori gakkai ronbonshi [Journal of the Information Processing Society of Japan]*, (6):1915–1924.

[Tokuhiro, 2014] Tokuhiro, Y. (2014). *Nihongo gakushū no tame no yoku tsukau jun kanji 2200 [Les 2200 kanjis fréquemment utilisés dans l'apprentissage du japonais]*. Sanseidō.

[W3C, 2006] W3C (2006). [Site web ; accédé le 20 avr. 2016] `https://www.w3.org/TR/2006/WD-wordnet-rdf-20060619/`.

[Waller, 2012] Waller, J. (2012). [Site web ; accédé le 10 sept. 2016] `http://www.tanos.co.uk/jlpt/`.