Smals

Codage et conversion de l'information

Préserver l'information numérique

Clients & Services

Section Recherches

Date : juin 2008
Deliverable : 2008/TRIM2/02
Statut : final
Auteur : Arnaud Hulstaert

Koninklijke Prinsstraat 102
1050 Brussel

Rue du Prince Royal 102
1050 Bruxelles

Tel : 02/787.57.11
Fax : 02/511.12.42

Management Summary

L'informatisation de notre société et la dématérialisation de l'information qui accompagne son développement actuel ont entraîné la gestion de quantités de données sous forme numérique. Ces informations, issues d'applications diverses, sont souvent hétérogènes dans leur format, de sorte que leurs échanges posent des problèmes importants pouvant profondément modifier la qualité des données.

Le problème est connu depuis de nombreuses années. Cependant, l'interconnexion des réseaux et des applications gérant des données similaires et devant de plus en plus interagir le rendent davantage stratégique aujourd'hui que par le passé. Cette situation rend ainsi problématique la multiplicité des solutions apportées et des formats utilisés et constitue un frein à un traitement optimal des données tout au long de la chaîne de traitement informationnel (stockage, indexation, recherche), tel que l'identification d'un travailleur présent dans une déclaration par comparaison avec les données situées dans d'autres bases de données de la sécurité sociale.

Le problème est aussi bien technique que conceptuel. Il faut bien entendu tenir compte des formats de codage disponibles et de leur support, tant au niveau du *software* que du *hardware*. Par ailleurs, beaucoup de formats disposent de nombreuses variantes nationales, généralement incompatibles sans conversion de l'information. Certains formats étant moins riches que d'autres, il est parfois nécessaire d'appauvrir l'information, par exemple en supprimant les caractères accentués et spéciaux. Cependant, les approches généralement locales du problème et de sa résolution ont entraîné, au sein des applications, l'apparition de nombreux formats et processus de conversions différents sans que les contraintes techniques ne le justifient à chaque fois.

A certains de ces problèmes, Unicode peut apporter des réponses mais il ne constitue en rien la solution parfaite et unique. Cependant, la richesse relative de l'information qu'il permet, sa compatibilité avec d'autres normes plus anciennes et largement répandues, ainsi que sa souplesse dans certaines manipulations de chaîne en font une évolution inévitable qu'il est nécessaire d'appréhender.

Afin de résoudre ces difficultés, il convient d'adopter une approche plus globale du traitement de l'information en définissant une politique de codage (afin d'en éviter la multiplicité et par là les conversions nécessaires) et une politique de conversion (pour gérer la qualité de l'information de manière cohérente) appliquées à l'ensemble des données.

Par ailleurs, dans le cadre de la préservation à long terme de l'information numérique, l'adoption de formats standards et ouverts, ainsi qu'une rationalisation de leur nombre est généralement considéré comme un pré requis. Outre cela, préserver la qualité de l'information, y compris en choisissant des formats permettant une compatibilité ascendante comme Unicode, permet d'en faciliter la conservation et la compréhension pour les futurs utilisateurs, tout en appréhendant mieux les changements technologiques.

Contenu

But et structure du document

Ce document a pour objectif de souligner les problèmes techniques et conceptuels que pose la gestion de formats hétérogènes de codage des caractères, principalement en ce qui concerne la qualité des données et du traitement de l'information.

Après une introduction à la problématique, le deuxième chapitre abordera les concepts de base des formats d'encodage des caractères, tels que la notation des caractères, la différence entre caractère et glyphe ainsi que la définition des composants d'un format de codage.

Il nous a semblé important et utile de retracer au chapitre trois l'évolution des principaux formats de codage des caractères occidentaux ainsi que de présenter le format Unicode et ses principales caractéristiques. De cette manière, nous pourrons constater les difficultés liées à la conversion des informations entre formats hétérogènes.

La quatrième partie resituera la problématique du codage et de la conversion de l'information dans le cadre de l'étude sur la préservation à long terme de l'information numérique, dont ce document représente une première partie.

Ensuite, le quatrième chapitre abordera le problème dans le contexte plus global de l'e-government. Des exemples simples et concrets nous permettront d'illustrer notre propos et de montrer les difficultés que posent le codage et la conversion des caractères dans le cadre du stockage, de l'indexation, de la recherche et de l'affichage des informations. Cette partie se terminera par un examen des solutions envisageables ainsi que des apports et des limites d'Unicode.

Une conclusion nous permettra de faire une synthèse des problèmes, des enjeux ainsi que des recommandations.

Enfin, en annexe, le lecteur intéressé trouvera plusieurs références de base sur la problématique, quelques informations complémentaires ainsi qu'un glossaire des termes techniques et acronymes utilisés.

1. Introduction

L'informatisation de notre société et la dématérialisation de l'information qui accompagne son développement actuel ont entraîné la gestion de quantités de données sous forme numérique. Ces informations, issues d'applications diverses, sont souvent hétérogènes dans leur format et leurs échanges posent dès lors des problèmes importants. Aussi le présent document a-t-il pour objectif d'analyser les problèmes que cette hétérogénéité des formats de codage, et les conversions que celle-ci implique, peut entraîner sur la qualité de l'information tout au long de la chaîne de traitement informationnel.

C'est ainsi que lors des caucus du Michigan aux Etats-Unis en 2004, des centaines de personnes dont le nom comportait une apostrophe (O'Connors), une espace (Van Kemps) ou un signe spécial (Ibañez) ont vu leur vote annulé en raison de l'incapacité du système à gérer ces caractères. Des problèmes similaires se posent pour les personnes dont le nom comporte un signe semblable lorsqu'elles souhaitent effectuer des réservations en ligne ou lorsqu'on recherche leurs données médicales. De même, Jessica Van Campen n'a pas pu, dans un premier temps, obtenir son diplôme de fin d'études parce que ses examens avaient été enregistrés sous le nom « Campen », le système ayant ignoré le « Van » en raison de la présence d'une espace.

"Basically, there are a lot of programmers out there who forget that a growing portion of the American public are not called John Smith or Mary White."

M. Rais, *Apostrophes in Names Stir Lot O'Trouble*, Dépêche AP, 21/02/08

Bien que ces caractères soient admis dans plusieurs applications de la sécurité sociale belge, ces problèmes d'hétérogénéité des formats ne s'en posent pas moins puisque certaines de ces applications reçoivent leurs données d'autres systèmes, parfois extérieurs à la sécurité sociale, voire à l'administration publique.

De surcroît, comme dans beaucoup d'autres domaines, ces applications, bien que distinctes, sont couramment amenées à gérer des données identiques. Par exemple, lorsqu'un employeur introduit une déclaration pour ses employés, ceux-ci seront identifiés par confrontation des données les concernant avec d'autres bases de données, telles celles référençant les personnes physiques ou les déclarations que l'employeur a dû introduire lors de l'engagement du salarié. Pour que ces données puissent être confrontées, il est nécessaire que, au moment de la comparaison, elles aient le même format de codage.

En effet, pour pouvoir être stockée et véhiculée, toute information doit être codée sous une forme quelconque (ici informatique). Ce processus de codage repose sur la définition préalable d'un vocabulaire et d'une syntaxe qu'il est nécessaire de maîtriser pour comprendre l'information et en assurer ainsi la compréhension et la conversion, si non parfaite du moins optimale, en une autre forme pouvant être interprétée et comprise par un tiers.

Pour tout format de codage informatique, il convient de distinguer deux éléments : l'ensemble des caractères qu'il permet de représenter et le processus technique par lequel ils seront codés, puisque deux codes ayant le même répertoire de caractères n'ont pas nécessairement la même technique d'encodage.

Ces deux éléments ont évidemment évolué avec le développement de l'informatique. D'un côté, le nombre de caractères représentables à l'aide d'un même format n'a cessé de s'accroître en raison des progrès informatiques. De l'autre, la création continue de nouveaux formats a entraîné une multiplication des techniques d'encodage qu'il convient donc de maîtriser.

Cette situation implique que, en circulant, l'information sera amenée à subir des transformations pour s'adapter à son nouveau format, transformation tant au niveau du fond (avec un appauvrissement linguistique éventuel de l'information, par exemple par la suppression des accents ou des caractères spéciaux) qu'au niveau de la forme (le format changeant, une même séquence de bits ne désigne plus la même information, et il faudra donc l'adapter).

Comment dès lors préserver la qualité de l'information, si possible tout au long de la chaîne de traitement informationnel (stockage, indexation, recherche, affichage) ? Et ce d'autant plus que l'utilisateur final (humain ou informatique) peut parfois se situer très loin du producteur de l'information, l'empêchant ainsi d'en évaluer les modifications.

Par ailleurs, dans le cadre de la préservation à long terme de l'information numérique, dont le présent document constitue une première partie, l'adoption de formats standards et ouverts, ainsi qu'une rationalisation de leur nombre est généralement considéré comme un pré requis. Outre cela, préserver la qualité de l'information, y compris en choisissant des formats permettant une compatibilité ascendante comme Unicode, permet d'en faciliter la conservation et la compréhension pour les futurs utilisateurs, tout en appréhendant mieux les changements technologiques.

Partant de ces constats, notre étude comprendra une première partie qui exposera tout d'abord le vocabulaire utilisé dans le cadre de formats de caractères, car de nombreuses confusions existent et peuvent avoir des répercussions importantes. Ensuite, nous examinerons l'évolution des formats de codage de l'information textuelle. Cette partie se terminera par un aperçu plus approfondi d'Unicode puisque ce format est aujourd'hui vu par certains comme la solution aux problèmes de codage et de conversion de l'information.

Une seconde partie sera consacrée à la problématique du codage et de la conversion des données au sein de l'e-government. Pour illustrer cette partie, nous utiliserons trois exemples : l'identification des travailleurs au sein de la sécurité sociale belge, l'application Limosa et la volonté de l'Union européenne de favoriser la coopération en diverses matières, telles que la circulation des personnes sur le marché européen.

Cette seconde partie s'achèvera par un examen des solutions envisageables pour remédier aux problèmes du codage et de la conversion des données, et en particulier les apports et les limites d'Unicode en la matière.

2. Notions importantes

Avant de passer à l'examen des principaux formats de codage encore utilisés aujourd'hui, il est important de définir quelques notions en rapport avec notre problématique.

2.1. Notation des caractères

Notation binaire (base 2) : les caractères sont représentés à l'aide d'une succession de bits, généralement regroupés en octets. Le nombre de bits utilisés par le format de codage définit le nombre de possibilités dont on dispose pour coder des caractères.

Notation décimale (base 10) : les caractères sont représentés à l'aide des chiffres arabes exprimés en base 10, cette notation est utilisée en programmation.

Notation hexadécimale (base 16) : la notation binaire étant peu lisible par un être humain, la notation hexadécimale est utilisée. Elle consiste à remplacer chaque groupe de quatre bits par un symbole. Seize symboles sont donc nécessaires. Pour les 10 premiers, les chiffres décimaux 0 à 9 sont utilisés, suivis par les 6 premières lettres de l'alphabet.

$$\left(\underbrace{0101}_{5}\,\underbrace{1101}_{D}\,\underbrace{0100}_{4}\right)_2 = (5D4)_{16}$$

Figure 1 : exemple de conversion d'un code binaire en hexadécimal

Le tableau ci-dessous illustre ces trois bases :

Base 10	Base 2	Base 16
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9

10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Comme il est très facile de confondre un nombre exprimé en base 10 et en base 16, on prendra donc soin de mentionner la base en indice. Par exemple, dans Unicode, 0059 désignera en base 16 le caractère « lettre latine Y majuscule » alors qu'en base 10 il se rapportera au point-virgule. On notera donc $(00)59_{10}$ ou $(00)59_{16}$.

2.2. Caractère et glyphe

Il est important de ne pas confondre un caractère et sa représentation graphique, le glyphe.

2.2.1. Définitions

Un caractère est une « unité d'information abstraite utilisée pour organiser, commander ou représenter des données textuelles »[1].

Un glyphe est une « forme géométrique utilisée pour présenter graphiquement des morceaux de texte ».[2]

Il faut donc distinguer le caractère abstrait « A majuscule » de ses représentations graphiques : A, **A**, A…

2.2.2. Relation caractère-glyphe

La plupart du temps, à un caractère correspondent un ou plusieurs glyphes, mais :

- Il y a des caractères sans glyphes, tels que les caractères de commandes.
- Un caractère peut être représenté par plusieurs glyphes tels que les caractères combinatoires[3] (a + ^ = a accent circonflexe).
- Un glyphe peut représenter plusieurs caractères, par exemple la ligature « fl » peut être un glyphe pour la suite de caractères « f » « l ».
- La relation caractère-glyphe n'est pas biunivoque ; par exemple, le glyphe B peut représenter les caractères « lettre latine B majuscule » ou « lettre grecque BETA majuscule ».
- Cette relation peut dépendre de la direction d'écriture. Ainsi, en français, au caractère « parenthèse ouvrante » correspond le glyphe « (» tandis qu'en arabe ce sera «) ».

> *"An "A" (or any other character) is something like a Platonic entity: it is the idea of an "A" and not the "A" itself."*
>
> Michael E. Cohen: *Text and Fonts in a Multi-lingual Cross-platform World*

[1] P. ANDRIES, *Unicode et ISO 10646 en français*, ch. 3, p. 48, D3, http://hapax.qc.ca/, traduction de la norme Unicode 3.1 par P. A.
[2] J. ANDRE, « Caractères, codage et normalisation. De Chappe à Unicode », *Document numérique*, vol. 6, n. 3-4, 2002, p. 15.
[3] Un caractère combinatoire est un caractère qui se combine graphiquement avec le caractère de base précédent. P. ANDRIES, *Unicode et ISO 10646 en français*, Glossaires, p.852, http://hapax.qc.ca/, traduction de la norme Unicode 3.1 par P. A.

Outre de souligner la difficulté de définir le mot caractère, cette distinction caractère-glyphes conduit à deux principes :

1. Même si des candidats au codage sont graphiquement identiques et donc pourraient être représentés par un même glyphe, ils doivent être codés de manière distincte pour garantir une correspondance biunivoque entre majuscule et minuscule dans un alphabet donné ainsi qu'entre normes différentes.
2. Les variations de forme (glyphes) ne doivent pas être codées de manière distincte puisqu'elles représentent le même caractère abstrait.

Ces deux principes ne sont malheureusement pas toujours respectés pour des raisons de place dans les formats de codage ou de compromis et de compatibilité avec les formats existants.

2.3. Format de codage et conversion

La constitution d'un format de codage (*character set*) passe par la définition :

- d'un répertoire de caractères (*character repertoire*) ou jeu de caractères, qui reprend l'ensemble des caractères abstraits qui doivent être codés ;
- d'une table de référence (*character code*) qui attribue à chaque caractère du répertoire un code numérique unique ;
- d'une technique d'encodage (*character encoding*) qui permet de transformer le code numérique de chaque caractère sous forme binaire, c'est-à-dire d'octet(s).

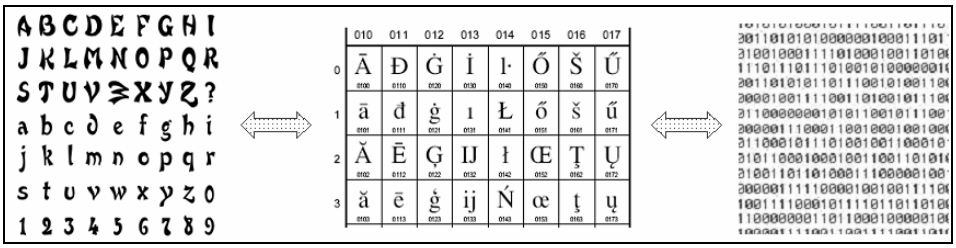


Figure 2 : les trois composants d'un format de codage : un ensemble de caractères à coder, une table de référence et une technique d'encodage

Ainsi, dès lors qu'un texte est saisi avec un ordinateur, ce texte est numérisé par la machine à l'aide d'une table de référence. Cette phase d'encodage donne naissance à une séquence binaire plus ou moins longue appelée, éventuellement, à être stockée à des fins de sauvegarde ou de transmission.

Avant toute réutilisation de cette séquence, une phase de décodage intervient alors, dans le but de retransformer le code binaire en texte compréhensible. Pour cela, l'ordinateur doit pouvoir s'appuyer sur la même table de référence que celle qui a servi à l'encodage, pour autant qu'elle soit connue et disponible.

De nombreux formats de codage étant disponibles, il faut généralement passer par une phase de transcodage, c'est-à-dire le fait de convertir le code numérique du caractère vers un code numérique correspondant dans un autre format.

3. Evolution des formats de codage des caractères

De très nombreux codes de transmission des textes ont été mis au point depuis le début de l'histoire humaine. Cependant, à la fin du 19e siècle et durant la première moitié du 20e siècle, apparaissent diverses inventions qui vont profondément modifier la transmission de l'information : le télégraphe de Morse et les cartes perforées construisent les bases de l'informatique et donc des méthodes modernes de transmission. Maîtrisant toujours davantage la technique, les ingénieurs définissent des codes de plus en plus riches passant de 4 à 5, puis 6 bits. Ces codes ne sont plus directement utilisés aujourd'hui, aussi avons-nous choisi de commencer l'évolution des formats avec les codes à 7 bits, encore abondamment utilisés à l'heure actuelle.

Ce tour d'horizon des différents formats nous permettra de comprendre les difficultés rencontrées lors de la transmission de textes entre formats hétérogènes.

3.1. Codages à 7 bits : ASCII et ISO/IEC 646

Le codage à 7 bits est le plus ancien format de codage encore utilisé aujourd'hui. Il s'agit principalement, voire uniquement, du format ASCII.

Suite au développement des ordinateurs durant la Seconde Guerre mondiale et l'après-guerre, les codages se sont multipliés, chaque constructeur définissant son propre système. Aussi, au début des années 60, ces constructeurs se sont-ils regroupés pour définir un code commun à 7 bits, adopté dans un premier temps en 1963, puis publié en 1967 par l'*American Standard Association* sous le nom d'*American Standard Code for Information Interchange* (ASCII).

Le code ASCII fut adopté aux Etats-Unis par la plupart des acteurs de l'informatique. Comme les constructeurs américains dominaient le marché, l'ASCII est rapidement devenu un standard international de fait.

Le souhait fut donc exprimé d'adapter ce code aux autres langages du monde et plus particulièrement de l'Europe occidentale. Ce travail fut effectué par l'ISO, qui publia en 1967 la norme ISO/IEC 646. De ce fait, l'ISO « désaméricanisa » quelque peu la table en ouvrant certaines positions à des variantes nationales.

Etant un format à 7 bits, l'ASCII (c'est-à-dire la norme ISO/IEC 646) offre 2^7 possibilités, soit 128, puisque chaque bit peut prendre la valeur « 0 » ou « 1 ». Parmi elles, seuls 95 caractères sont « imprimables » ; les autres servent de caractères de commande pour des périphériques (écrans, perforateurs de ruban, etc.). Certains de ces caractères sont liés à la technologie de l'époque et n'ont plus d'utilité aujourd'hui, comme le caractère BEL qui servait à activer la sonnerie d'un télex.

Les 95 caractères « imprimables » peuvent être répartis en trois groupes :

- 83 caractères obligatoires : espace, 52 lettres de l'alphabet (majuscule/minuscule), 10 chiffres et 20 signes de ponctuation ou autres.
- 2 caractères au choix # ou £ et $ ou ¤ (symbole monétaire international).
- 10 positions réservées à des caractères nationaux.

Dès sa conception, l'ASCII comprenait donc une version internationale de référence (IRV) et des variantes nationales (parfois même plusieurs pour un seul pays comme ce fut le cas en France) comme le montre le tableau suivant :

Version de référence (IRV)	#	¤	@	[	\	]	^	`	{	|	}	~
Allemagne (DIN 66003)	#	$	§	Ä	Ö	Ü	^	`	ä	ö	ü	ß
Belgique	#	$	à	°	ç	§	^	`	é	ij	è	~
Espagne	#	$	·	¡	Ñ	Ç	¿	`	‘	ñ	ç	″
France (NF Z62010/1982)	£	$	à	°	ç	$	^		é	ù	è	″
Grande-Bretagne	£	$	@	[	\	]	^	`	{	|	}	~
Suisse romande			à		ç				é	ù		~
USA (US-ASCII)	#	$	@	[	\	]	^	`	{	|	}	~

L'existence de ces variantes rend les échanges d'informations plus complexes et peut modifier sensiblement l'information comme le montre l'image suivante qui illustre la transformation subie par un nom allemand lors de son passage de la version allemande à la version belge de la norme :

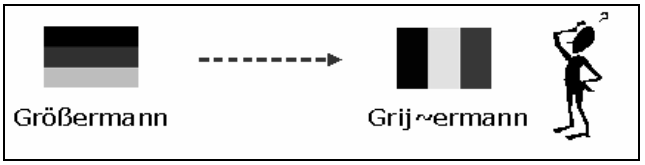


Figure 3 : transformation entre deux variantes d'ISO/IEC 646

Cependant, comme la version internationale ne comprenait pas le symbole dollar, la version américaine (US-ASCII) prit rapidement un poids important, aussi bien dans le milieu informatique que comptable, par rapport à la version internationale, qui ne fut guère utilisée.

Outre l'existence de variantes, l'incohérence entre les versions francophones et leur inadaptation partielle, puisqu'elles ne comportent pas les caractères accentués dans la partie invariante, a entraîné le souhait d'un format plus riche.

3.2. Codages à 8 bits : EBCDIC et ISO/IEC 8859

En réalité, les formats de codage à 7 bits en utilisaient 8. Cependant, le 8ème bit servait de caractère de contrôle pour vérifier la validité de la transmission.

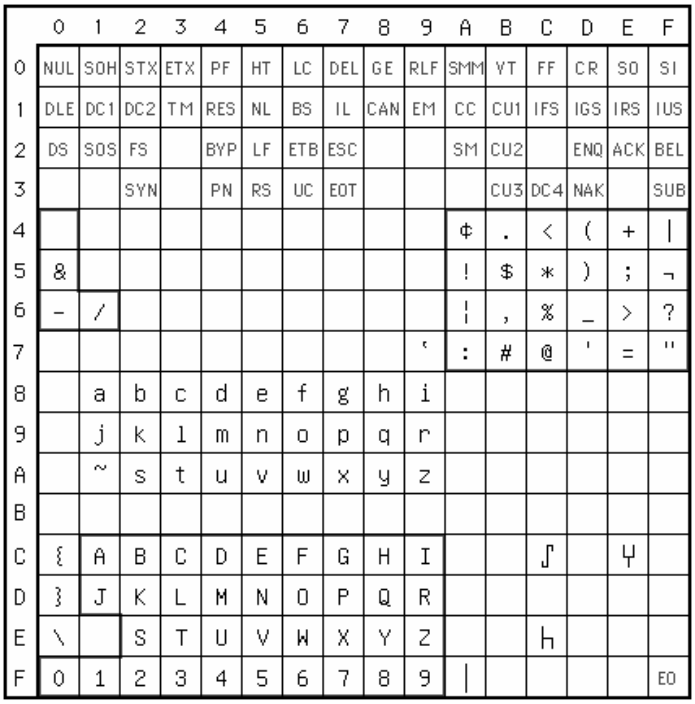
La qualité des transmissions augmentant, ce bit de contrôle put être utilisé à d'autres fins, ce qui permit de doubler le nombre de caractères disponibles, passant ainsi à 8 bits, soit 256 possibilités.

3.2.1. EBCDIC

L'*Extended Binary-Coded Decimal Interchange Code* (EBCDIC) est un format de codage à 8 bits créé en 1964 par IBM pour équiper la série 360 de ses mainframes. Bien que partie prenante dans la standardisation d'ASCII, IBM n'eut pas le temps de rendre la série compatible avec ses périphériques et

commercialisa donc le tout avec EBCDIC. Le succès de la série a assuré la pérennité d'EBCDIC encore utilisé aujourd'hui.

Malgré son codage à 8 bits, ce format (voir figure 4) ne propose guère plus de caractères que l'ASCII, laissant ainsi beaucoup de place perdue et rendant les opérations de conversion plus complexes puisqu'il faut d'abord supprimer les espaces vides.[4] Par ailleurs, il existe de très nombreuses variantes nationales (57 !) généralement incompatibles entre elles.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	PF	HT	LC	DEL	GE	RLF	SMM	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	TM	RES	NL	BS	IL	CAN	EM	CC	CU1	IFS	IGS	IRS	IUS
2	DS	SOS	FS		BYP	LF	ETB	ESC			SM	CU2		ENQ	ACK	BEL
3			SYN		PN	RS	UC	EOT				CU3	DC4	NAK		SUB
4											¢	.	<	(	+	|
5	&										!	$	*	)	;	¬
6	-	/									¦	,	%	_	>	?
7										`	:	#	@	'	=	"
8		a	b	c	d	e	f	g	h	i						
9		j	k	l	m	n	o	p	q	r						
A		~	s	t	u	v	w	x	y	z						
B																
C	{	A	B	C	D	E	F	G	H	I			∫		⊔	
D	}	J	K	L	M	N	O	P	Q	R						
E	\		S	T	U	V	W	X	Y	Z			⊢			
F	0	1	2	3	4	5	6	7	8	9	|					EO

Figure 4 : table de code EBCDIC de base. Le code hexadécimal de chaque caractère est formé par son numéro de ligne suivi de son numéro de colonne.

Ce format fut complété par la suite (EBCDIC augmenté) pour inclure certains caractères accentués et spéciaux.

3.2.2. ISO/IEC 8859

ASCII est un code à **7 bits**. L'extension de cette dénomination pour tout autre format est une erreur et doit être proscrite, car elle peut engendrer des problèmes de compréhension.

La norme ISO/IEC 646 étant jugée trop « pauvre » linguistiquement, l'ISO a travaillé à partir de l'ASCII sur une nouvelle norme étendue à 8 bits. Ce travail a donné naissance à la série de normes ISO/IEC 8859. Chacune d'elles ne pouvant servir pour l'ensemble des langues européennes, la décision fut prise de constituer des variantes (16 actuellement). De cette manière, chaque variante couvre un ensemble de pays regroupés par affinités politiques et commerciales. Par exemple, ISO/IEC 8859-1 correspond à la zone occidentale, -2 à la zone orientale, etc.

On trouvera en annexe (8.1) la liste des langues couvertes par chaque variante de la norme.

A nouveau, trois parties peuvent être distinguées dans cette série de normes :

- Les 128 premières positions dans la table de référence sont identiques à l'ASCII (qui est identique à la version internationale de la norme ISO/IEC 646 dans sa version de 1991 et non plus 1983).

[4] Exemple de chaîne de caractères devant être convertie en stream imprimable en supprimant les caractères non imprimables : 9 lignes de programmation avec une chaîne ASCII et 23 avec EBCDIC. C. LONGMORE, *ASCII and EBCDIC Compared*, http://www.dynamoo.com/technical/ascii-ebcdic.htm, novembre 2004 (consulté le 21 mai 08).

- Les 32 suivantes sont des caractères de commande, identiques pour toutes les normes ISO/IEC 8859.
- Les 96 restantes sont propres à chaque variante.

Ce format étant conçu pour faciliter l'échange d'informations, des efforts furent faits pour rendre les différences entre versions de la norme aussi « lisses » que possible. Par exemple, les signes propres à la langue allemande ont la même position dans les différentes tables. Ainsi, l'eszett « ß » est représenté par la valeur DF_{16} quelle que soit la table. De plus, certains caractères de même position diffèrent seulement par leur signe diacritique[5], tel que le N majuscule tildé « Ñ » de Latin-1 et Latin-3 ($D1_{16}$) qui deviendra un N majuscule accentué aigu « Ń » en Latin-2, un N majuscule avec cédille « Ņ » en Latin-4. Malheureusement, ce n'est pas toujours le cas.

ISO/ IEC 8859-1

Parmi la série de normes ISO/IEC 8859, la plus utilisée dans notre pays est ISO/IEC 8859-1 (aussi appelé Latin-1) qui permet de couvrir les langues suivantes : l'allemand, l'anglais, le basque, le catalan, le danois, l'écossais, l'espagnol, le féringien[6], le finnois (partiellement), le français (partiellement), l'islandais, l'irlandais, l'italien, le néerlandais (partiellement), le norvégien, le portugais, le rhéto-roman[7], le suédois, certaines langues européennes sud-orientales (l'albanais), ainsi que des langues africaines (l'afrikaans[8] et le swahili[9]).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A	nbs	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Figure 5 : les 96 caractères propres à ISO/IEC 8859-1

ISO/IEC 8859-15

Pour combler quelques lacunes et introduire le caractère euro « € », ISO/IEC 8859-1 fut révisée en 1999 par ISO/IEC 8859-15 (Latin-9). Ne pouvant augmenter le nombre de positions, certains caractères de ISO/IEC 8859-1, jugés moins utiles, furent remplacés comme le montre le tableau suivant :

Code hexadécimal	*ISO/IEC 8859-1 (Latin 1)*	*Remplacé par le caractère de Latin-9*	*ISO/IEC 8859-15 (Latin-9)*
A4	¤	symbole euro	€
A6	¦	lettre majuscule latine s caron	Š
A8	¨	lettre minuscule latine s caron	š
B4	´	lettre majuscule latine z caron	Ž
B8	¸	lettre minuscule latine z caron	ž
BC	¼	digramme soudé majuscule Œ	Œ
BD	½	digramme soudé minuscule Œ	œ
BE	¾	lettre majuscule latine Y tréma	Ÿ

[5] Un signe diacritique est un signe accompagnant une lettre. Il peut être placé au-dessus (diacritique suscrit), au-dessous (diacritique souscrit), dans ou à travers (diacritique inscrit), après (diacritique adscrit), devant (diacritique prescrit) ou tout autour (diacritique circumscrit). Son objectif est de modifier la valeur phonétique de la lettre, permettre une lecture plus précise ou éviter une ambiguïté entre des homographes.
[6] Langue des Îles Féroé, groupe d'îles situées dans l'océan Atlantique entre l'Écosse et l'Islande, et dépendantes du Danemark.
[7] Groupe de langues parlées en Suisse et dans le nord de l'Italie.
[8] Parlé en Afrique du Sud et en Namibie, anciennes colonies néerlandaises, l'afrikaans est une langue issue du néerlandais.
[9] Les langues swahilis sont un groupe de langues bantoues de l'Afrique de l'Est qui sont le fruit d'un métissage de langues africaines et d'arabe.

Cette version plus complète devrait donc théoriquement remplacer la précédente. Cependant, dans la pratique, son adoption tardive par l'ISO et l'arrivée du nouveau standard Unicode n'ont guère favorisé son expansion. Elle reste donc fort peu répandue à l'heure actuelle.

3.3. Unicode et ISO/IEC 10646

Comme nous venons de le voir, les normes en matière de codage des caractères étaient très diverses et les échanges d'informations souvent complexes entre ces normes hétérogènes. « *Les codages de caractères préexistants (à Unicode) étaient à la fois incomplets et incohérents : à chaque fois que du texte devait passer d'un programme ou d'un système à un autre, il y avait un sérieux risque de corruption. La gestion de codages particuliers était codée en dur dans les programmes, rendant le développement, les tests et la maintenance de ces versions internationales difficiles, coûteux, voire cauchemardesques. En fin de compte, les lancements de produit dans les marchés étrangers étaient à la fois onéreux et tardifs, au grand dam autant des éditeurs que de leurs clients.* »[10]

Avec l'interconnexion des réseaux, les échanges constants d'informations et un marché économiquement mondial, le problème n'a fait qu'empirer.

Pour éviter que cette situation ne se perpétue, l'ISO a constitué en 1984 un groupe de travail auquel il a donné l'objectif suivant : « *élaborer une norme établissant un répertoire de caractères graphiques des langues écrites du monde et son codage* ». Parallèlement, suite aux travaux de Joe Becker chez Xerox qui travaillait sur un jeu universel de caractères qu'il nommait Unicode, plusieurs industriels vont se réunir pour former le consortium Unicode[11]. Conscients des enjeux, les deux groupes collaborèrent étroitement pour que les deux répertoires coïncident totalement. Le standard Unicode et la norme ISO/IEC 10646 ont donc des répertoires rigoureusement identiques et évoluent conjointement depuis 1993, chaque nouvelle version d'Unicode donnant lieu à une mise à jour de la norme. La compatibilité ascendante entre versions est assurée puisque il a été décidé de ne jamais modifier un caractère défini dans les versions antérieures même en cas d'erreur ![12]

3.3.1. Principe de base et espace de codage

« Le standard Unicode (y compris la norme ISO/IEC 10646) est un mécanisme universel de codage de caractères. Il définit une manière cohérente de coder des textes multilingues, facilite l'échange de données textuelles à l'échelle planétaire et crée ainsi les prémisses pour tout logiciel international. »

Le principe de base d'Unicode est d'attribuer à chaque caractère abstrait un numéro et un nom. En cela, il ne diffère guère des autres standards ou normes de codage des caractères.

Cependant, son but étant de pouvoir coder l'ensemble des caractères du monde, les codages précédents de 7 et 8 bits étaient largement insuffisants. La décision fut donc prise de coder l'information, dans un premier temps, sur 16 bits

[10] M. DAVIS, Préface du livre de P. ANDRIES, *Unicode 5.0 en pratique. Codage des caractères et internationalisation des logiciels et des documents*, Québec, 2008.
[11] Le consortium Unicode rassemble les acteurs majeurs du secteur informatique. Le site internet du consortium (www.unicode.org) rassemble une foule d'informations utiles. Aussi nous a-t-il semblé intéressant de le présenter brièvement en annexe 7.2.
[12] P. ANDRIES, Unicode 5.0 en pratique. *Codage des caractères et internationalisation des logiciels et des documents*, Québec, 2008, p. 74.

(soit 2^{16} = 65.536 possibilités) et, dans un second temps, sur 31 bits[13] (soit 2^{31} = 2.147.483.648 possibilités) !

Actuellement, l'espace de codage d'Unicode se « limite » à 1.114.112 possibilités, avec des codes numériques allant en hexadécimal de 0 à $10FFFF_{16}$, dont seulement quelque 20 % ont été assignés à des caractères, soit encore environ 875.000 positions libres.

Par facilité, les concepteurs d'Unicode ont divisé cet espace de codage en plans, soit actuellement 17 plans (sur les 256 possibles dans le futur !) de 65.536 possibilités chacun (256 lignes x 256 colonnes).

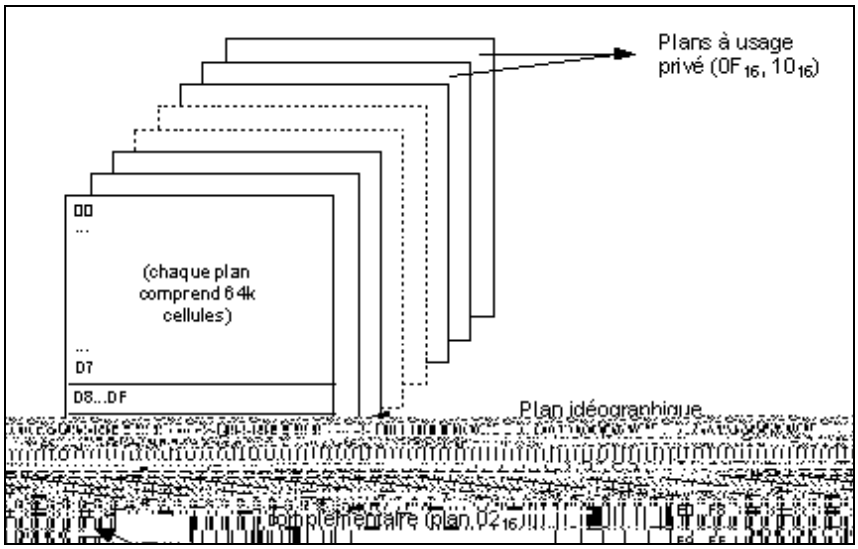


Figure 6 : plans et zones de codage

Parmi ces plans, le plus important est le BMP (*Basis Multilingual Plan*) ou plan multilingue de base comprenant les caractères entre 0000_{16} et $FFFF_{16}$. Il contient les caractères les plus courants et la majorité des caractères Unicode. Dans ce plan, les caractères sont regroupés par bloc de caractères consécutifs (appelés « zones ») en fonction de leur appartenance linguistique ou fonctionnelle (par exemple « Latin étendu A », « Birman » ou « Ponctuation »). Non seulement cette disposition facilite la consultation des tableaux de caractères, mais elle permet également des mises en œuvre plus compactes. Le lecteur intéressé trouvera une représentation de ce plan en annexe 8.3.

De plus, afin de faciliter la compatibilité avec les anciennes codifications, les 256 premières positions (bloc « Latin de base » et « Supplément Latin-1 ») reprennent exactement les caractères d'ISO/CEI 8859-1 (Latin-1).[14] Dès lors, les anciens textes codés en ASCII ou ISO/IEC 8859-1 sont compatibles Unicode.

3.3.2. Les dix principes d'Unicode

La conception et le développement d'Unicode respectent dix principes :

1. *Universalité* : Unicode décrit un seul jeu de caractères qui comprend tous les caractères nécessaires aux écritures du monde. Les caractères de toutes les écritures peuvent se mêler sans encombre.
2. *Efficacité* : afin de permettre des mises en œuvre efficaces, Unicode n'inclut pas de caractères d'échappement[15]. Tous les caractères Unicode ont la

[13] En réalité, il s'agit d'un codage sur 32 bits, mais le 1er doit toujours être 0.
[14] Pour rappel, dans cette norme ISO, les 128 premières positions reprennent exactement la table ASCII internationale dans sa version de 1993.
[15] Un caractère d'échappement est un caractère textuel placé devant un autre caractère textuel afin d'en changer la fonction.

même qualité, tous les codes sont d'une égale facilité d'accès. Cette universalité et cette uniformité de codage permettent une analyse, un tri, un affichage, un repérage et une édition efficaces des chaînes textuelles Unicode.

3. *Caractère et non glyphe* : Unicode distingue les caractères, qui forment les unités atomiques d'une langue écrite, des glyphes qui représentent les différentes formes visuelles qu'un ou plusieurs caractères (dans le cas d'une ligature) peuvent prendre. De même, de multiples glyphes peuvent représenter un même caractère.

4. *Sémantique* : chaque caractère est accompagné d'informations sémantiques telles que la casse du caractère (majuscule, minuscule ou casse de titre), sa directionnalité (gauche, droite…), s'il est à symétrie miroir (par exemple « → - flèche vers la droite » sera ajusté selon la directionnalité du texte), s'il s'agit d'un numéral ou s'il s'agit d'un combinatoire (¨, ^…).

5. *Texte brut* : Unicode ne code que du texte brut sans aucun caractère de balisage spécifiant des informations sur la police de caractère, la couleur du texte, etc.

6. *Ordre logique* : les textes en Unicode sont stockés en mémoire en ordre logique. Parfois, l'ordre des caractères à l'affichage ou à l'impression diffère de l'ordre logique. Par exemple, l'i bref dévanâgarî[16] (ि s ('affiche res quèavant les caract'il suit dans l'.ordre logique

7. *Unification* : le standard Unicode ne code qu'une seule fois le même caractère et ne lui attribue qu'un seul numéro de code quel que soit le nombre de langues qui l'utilisent (le a de chat, Katze, gato et kat est donc codé avec un seul numéro : 0061_{16}). Lors du codage, Unicode ne conserve pas d'informations sur la langue. Donc l'i grec français, le ypsilon allemand et le wye anglais sont tous représentés par le même code de caractère, 0057_{16} (« Y »). Cependant, le principe suivant s'oppose parfois à celui-ci.

8. *Convertibilité* : lors de la création d'Unicode, le consortium a décidé de définir des méthodes d'importation et d'exportation de données codées dans des jeux de caractères fondamentaux à l'époque. Un grand soin a été apporté afin de garantir une conversion aller-retour sans perte entre Unicode et ces jeux de caractères. Lorsque des variantes de forme (voire la même forme) se sont vues affecter des numéros de caractère différents dans une norme de base, Unicode les garde séparées. Cette décision garantit l'existence d'une correspondance bijective entre le standard Unicode et les normes de base. Par exemple, « Å » peut avoir le code $00C5_{16}$ ou $212B_{16}$.

9. *Composition dynamique* : Unicode permet de composer dynamiquement les formes accentuées. Ainsi le tréma « ¨ » ne sera codé qu'une seule fois et pourra être combiné avec n'importe quelle lettre.

10. *Séquence équivalente* : certains éléments textuels peuvent être codés de plusieurs façons : une forme précomposée ou une ou plusieurs compositions dynamiques. Pour des raisons de compatibilité avec les jeux de caractères actuels, on a inclus dans le standard de nombreuses formes précomposées. Les caractères précomposés sont équivalents à leur suite de caractères décomposée. Ainsi « â » est-il équivalent à « a » + « ^ ». Unicode fournit pour chaque forme précomposée incluse dans la norme son équivalent canonique décomposé.

De plus, généralement, des suites de diacritiques sont considérées distinctes si les diacritiques se présentent dans un ordre différent. Pour éviter ce problème, Unicode définit un algorithme de mise en ordre canonique (cf. p. 37).

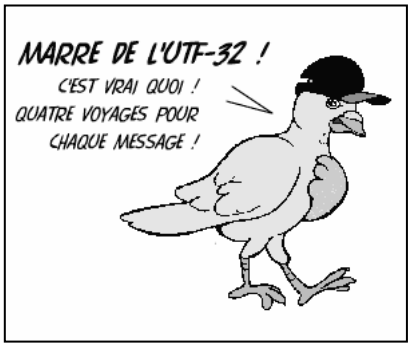
[16] La dévanâgarî est une écriture utilisée pour le sanskrit, le hindî, le népalais et plusieurs autres langues indiennes. C'est une des écritures les plus employées en Inde et au Népal.

3.3.3. Forme en mémoire des caractères

De par sa volonté de couvrir l'ensemble des caractères du monde, Unicode rend le codage des caractères plus complexe. En effet, auparavant, seuls 8 bits étaient utilisés, ce qui ne posait guère de problème puisque tout code étant inférieur à 256, il était donc très facilement représentable sous forme de simple octet, ce qui n'est plus le cas avec Unicode.

Dès lors, le consortium Unicode a développé trois techniques d'encodage appelées « *Unicode Transformation Format* » (UTF). Bien que très différents dans leur mise en œuvre, le principe de base est le même : tout code numérique Unicode doit pouvoir être retrouvé sans ambiguïté à partir de sa version transformée.

UTF-32



Chaque code numérique est représenté par une suite de 32 bits, soit 4 octets. Une transcription nécessite donc uniquement une conversion en binaire de la valeur hexadécimale du caractère.

Cette forme de codage a comme caractéristiques :

- Il s'agit du format le plus lourd puisque chaque caractère, que ce soit un « e » quasi-universel ou un idéogramme médiéval sub-indonésien, nécessitera quatre octets pour être codé.
- Pour des raisons de compatibilité avec UTF-16, l'espace de codage est volontairement limité à $0..10FFFF_{16}$.
- UTF-32 est l'équivalent de la forme UCS-4 d'ISO/IEC 10646 (Universal Character Set). Cependant, déclarer un texte en UTF-32 plutôt qu'en UCS-4 permet de spécifier l'application des règles sémantiques présentes dans Unicode mais pas dans ISO/IEC 10646.

UTF-16

Chaque numéro de caractère est codé sur un seizet (= 16 bits). Les caractères du PMB sont codés sur un seul seizet, tandis que ceux des autres plans sont codés à l'aide de deux seizets d'indirection.

Ces seizets d'indirection appartiennent au PMB et n'ont été attribués à aucun caractère. Chaque paire d'indirection est composée d'un seizet d'indirection supérieur (dont la valeur hexadécimale est comprise en D800..DBFF, soit 1024 possibilités) et d'un seizet d'indirection inférieur (dont la valeur est comprise entre DC00..DFFF, soit 1024 possibilités). Cette méthode offre donc 1.179.648 possibilités de codage (les 65.536 du PMB – les 2048 seizets d'indirection + 1.048.576 possibilités permises par la combinaison des seizets d'indirection), ce qui est largement suffisant pour la majorité des applications.

La conséquence de ce mode d'encodage est que, si en consultant une séquence de caractères encodée au format UTF-16, l'ordinateur trouve un seizet :

- compris dans les intervalles 0000..D7FF ou E000..FFFF, c'est qu'il s'agit obligatoirement d'un seizet codant un caractère du PMB ;
- compris dans l'intervalle D800..DBFF, c'est qu'il s'agit obligatoirement du premier seizet d'un caractère situé hors du PMB, qui doit donc être obligatoirement suivi par un seizet compris entre DC00..DFFF ;
- compris dans l'intervalle DC00..DFFF, c'est qu'il s'agit obligatoirement du second seizet d'un caractère situé hors du PMB, qui doit donc être obligatoirement précédé par un seizet compris entre D800..DBFF.

UTF-8

L'UTF-8 est une forme de codage à 8 bits afin de permettre son utilisation dans les systèmes architecturés sur la base de l'ASCII ou toute autre forme de codage à un octet.

Elle vise donc à sérialiser tout caractère Unicode uniquement à l'aide d'octets : un, deux, trois ou quatre selon les besoins.

Cependant, cette sérialisation n'est pas aisée. Par exemple, le caractère $FFFF_{16}$ correspond à la séquence de bits 1111111111111111_2. On pourrait aisément la sérialiser en deux octets, chacun étant de valeur 11111111_2. Or, cette valeur correspond au caractère FF_{16}. De ce fait, une confusion est possible, car en prenant l'octet en question, le système ne peut savoir s'il s'agit du caractère FF_{16} ou d'une partie du caractère $FFFF_{16}$. Il y a donc ambiguïté, ce que le standard Unicode n'accepte pas puisque en prenant au hasard un octet dans une séquence codée en UTF-8, on doit pouvoir de manière infaillible établir si celui-ci code à lui seul un caractère ou s'il fait partie d'une séquence.

Unicode prévoit donc de sérialiser le caractère en répartissant les bits dans différents octets en indiquant dans chacun d'eux sa « fonction ». Seuls les caractères U+0000 à U+007F (soit les 128 premières possibilités) sont codés sur un octet. S'il y a plusieurs octets, les bits les plus significatifs du premier octet d'une séquence donnent le nombre d'octets de la séquence : 110xxxxx pour une séquence de deux octets, 1110xxxx pour une séquence de trois octets, 11110xxx pour une séquence de quatre octets. Les octets suivants commencent obligatoirement par 10xxxxxx. Au sein de cette séquence, les bits du caractère à coder sont répartis en rajoutant éventuellement des zéros devant si nécessaire. Le tableau suivant illustre le processus de sérialisation mis en œuvre en UTF-8 :

Point de code (notation hexadécimale)	Point de code (notation décimale)	Point de code (notation binaire)	Séquence UTF-8 correspondante
U+0000 - U+007F	0 - 127	00000000 00000000 0xxxxxx	0xxxxxxx
U+0080 - U+07FF	128 – 2047	00000000 00000yyy yyxxxxxx	110yyyyy 10xxxxxx
U+0800 - U+D7FF U+E000 - U+FFFD	2048 – 55295 57344 - 65533	00000000 zzzzyyyy yyxxxxxx	1110zzzz 10yyyyyy 10xxxxxx
U+10000 - U+10FFFF	65536 - 1114111	000uuuuu zzzzyyyy yyxxxxxx	11110uuu 10uuzzzz 10yyyyyy 10xxxxxx

Prenons par exemple le caractère Å dont la valeur est $C5_{16}$, 197_{10} ou 11000101_2. Le caractère sera codé sur 2 octets, soit 11000011 (en rajoutant des zéros) suivi de 10000101, ce qui correspond à $C3_{16}$ suivi de 85_{16}. Or, $C3_{16}$ est non ambigu puisque s'il s'agissait d'un caractère, il serait lui-même codé sur deux octets, soit 11000011_2 suivi de 10000011_2, c'est-à-dire $C3_{16}$ suivi de 83_{16}.

Le principal avantage de ce format de transformation est qu'il est compact en termes d'octets puisque leur nombre varie selon les besoins, ce qui est particulièrement avantageux dans un monde informatique qui se caractérise par de nombreux échanges d'informations, ce qui en a fait le format d'encodage par défaut de XML.

Cependant, cette variabilité dans le nombre d'octets nécessaire au codage ralentit les routines de manipulation de chaînes (à savoir calcul de longueur, extraction, tri...).

4. Formats de codage et préservation à long terme de l'information

La préservation à long terme de l'information numérique est un problème vaste et complexe.[17] L'une des causes majeures en est l'obsolescence technologique, l'évolution des formats et l'évolution des types d'encodage. Il est donc conseillé de choisir, si possible, des formats ouverts et documentés, dont les normes représentent le meilleur exemple. De cette manière, dans le pire des cas, il sera toujours possible de reconstruire le format afin de lire les fichiers.

A ce titre, la norme OAIS (*Open Archival Information System* - ISO 14721) recommande l'inclusion de la documentation nécessaire à une relecture des fichiers dans le système. Il s'agit du concept « d'information de représentation », défini comme l'information qui traduit un Objet-données en des concepts plus explicites. L'Objet-données, interprété à l'aide de l'information de représentation, produit un Objet-information, comme le montre l'illustration suivant[18] :

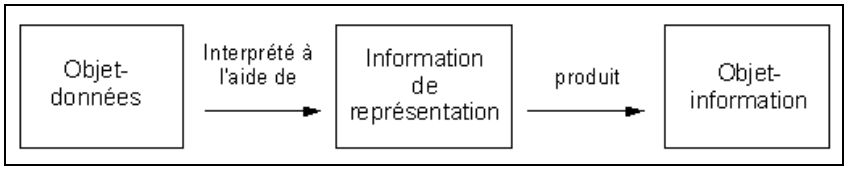


Figure 7 : obtention d'Information à partir de Données

Le choix de formats normalisés et non propriétaires est donc un avantage. En effet, tandis que la documentation des formats propriétaires n'est généralement pas ouverte et connue de tous, la documentation des standards est accessible et publique, rendant ainsi inutile son inclusion dans le système d'archivage.

Par ailleurs, l'émergence d'Unicode favorise un échange et un traitement homogène de l'information. Par son absence d'ambiguïté (une séquence de bits correspondant à une position désigne un seul caractère abstrait sans confusion possible) et sa globalité (il a pour ambition finale de coder l'ensemble des caractères du monde), il évite toute confusion puisqu'il permet de préserver la qualité de l'information en évitant les conversions dues à l'utilisation de différents formats de codage des caractères.

[17] Voir l'étude d'I. BOYDENS, La préservation à long terme de l'information numérique, *Techno, publication technique de Smals*, septembre 2004, n°28.
[18] Comité consultatif pour les systèmes de données spatiales. *Modèle de référence pour un Système ouvert d'archivage d'information (OAIS)*, livre bleu, mars 2005, p. 2-4.

Son utilisation facilite ainsi une compréhension ultérieure du texte et une préservation de son intégrité comme la législation et les normes en matière de conservation le prescrivent. De cette manière, les opérations de migration sont facilitées en ce qui concerne cet aspect.

A ce titre, la norme ISO PDF/A-1a, autrement dit le niveau « a » de la norme pdf pour l'archivage à long terme, impose que les polices utilisées et incorporées dans le document pdf soient configurées sur la base d'Unicode (Unicode mappings) afin de permettre une recherche non ambiguë dans le fichier, entre autres grâce aux informations sémantiques contenues dans le format Unicode.

De ce fait, il nous semble qu'une « best practice » en matière de format de codage serait d'utiliser les trois normes suivantes : ASCII, ISO/IEC 8859-1 et Unicode/ISO 10646.

5. Problématique des codages dans le contexte de l'e-government

Dans ce chapitre, nous analyserons les problèmes posés par les codages et conversions multiples de l'information dans le cadre de l'e-governement, et ce tout au long de la chaîne de traitement informationnel : stockage, indexation, recherche et affichage (point 5.3 à 5.7). Au préalable, après un bref exposé des cas concrets qui illustreront notre propos (point 5.1), nous verrons les raisons pour lesquelles cette problématique a longtemps été minimisée (5.2). Nous examinerons ensuite les solutions envisageables (5.8) et en particulier les apports et les limites d'Unicode en la matière.

5.1. Introduction

Comme la grande majorité des environnements applicatifs d'aujourd'hui, celui des administrations en Belgique se caractérise par l'existence conjointe de nombreuses applications informatiques, réparties entre de nombreuses institutions et services et dont la gestion elle-même est éclatée.

Ces diverses applications sont généralement amenées à collaborer en échangeant de (très) nombreuses données. Ces échanges se produisent à chaque étape d'une recherche d'informations : stockage, indexation, recherche, confrontation et affichage des données.

Ces nombreux flux d'informations impliquent souvent qu'une même donnée sera introduite et utilisée dans plusieurs applications. Or, les nombreuses transformations (conversion et transcodage[19]) opérées sur les données en diminuent parfois la qualité.

Ces modifications répétées entraînent plusieurs problèmes lors des différentes étapes d'une recherche d'informations, par exemple lors de la confrontation des données en vue de l'identification d'un travailleur.

Nous illustrerons notre propos à l'aide de trois exemples : l'identification des travailleurs au sein de la sécurité sociale, l'application Limosa chargée de procéder à l'enregistrement des travailleurs étrangers et la volonté de l'Union européenne de favoriser la coopération et l'échange d'informations entre pays membres.

[19] Nous distinguons ici la « conversion », opération modifiant la valeur sémantique des caractères (â → a), du « transcodage », opération qui consiste à changer le format des caractères.

5.1.1. Identification des travailleurs

Dans le cadre de la sécurité sociale, les employeurs sont tenus de fournir des informations sur les prestations, les rémunérations, les engagements, les risques sociaux, etc. de leurs employés. A cette fin, la sécurité sociale a développé des applications électroniques.

Dans quelques-unes de ces applications, il est nécessaire de procéder à l'identification des travailleurs inscrits dans la déclaration. Cette identification permettra de compléter le dossier de l'employeur et de l'employé en vue de leur accorder certains droits sociaux.

Le processus d'identification est le suivant :

- un employeur (ou son mandataire) envoie une déclaration électronique via un des canaux disponibles ;
- le service de l'identification compétent exécute les contrôles (automatiques et/ou humains) nécessaires à l'identification en comparant les données envoyées avec les données stockées dans les sources authentiques dont il dispose ;
- l'application renvoie les résultats de l'identification au destinataire (institutions de la sécurité sociale, employeur, mandataire) de diverses manières.

Etant donné qu'un même travailleur peut être mentionné dans les diverses déclarations, il semblerait judicieux que les données le concernant soient traitées de manière identique et que les résultats de l'identification soient similaires. Or, ce n'est pas le cas à l'heure actuelle.

Plusieurs raisons expliquent cette situation ; nous n'évoquons ici que celles qui concernent le codage et la conversion de l'information. La plupart d'entre elles sont dues au fait que, dans le cas de l'identification des travailleurs, l'environnement applicatif est extrêmement éclaté : de nombreuses applications et bases de données sont concernées, tout en traitant les mêmes données, comme le montre la figure 8. Les bases de données représentent les sources authentiques dont disposent les applications pour procéder à l'identification. Comme on le constate, même si ces applications sont susceptibles de gérer des informations identiques, les sources authentiques qui permettront l'identification, peuvent différer.

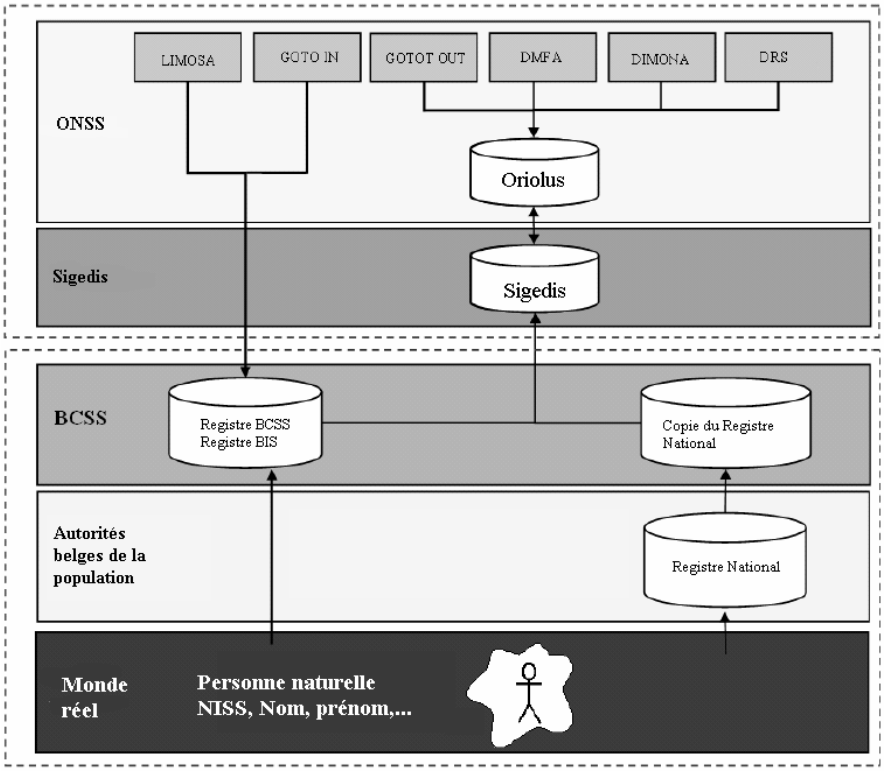


Figure 8 : environnement applicatif général de l'identification des travailleurs au sein de la sécurité sociale belge

5.1.2. Limosa

L'Union européenne applique le principe de la libre circulation des personnes et des services ainsi que la liberté d'établissement en tant qu'indépendant, mais dans le respect des conditions de travail fixées par les Etats membres.

Cette situation a engendré beaucoup de fraudes, avec pour conséquences une distorsion de la concurrence, une exploitation des travailleurs, un dumping social et une pression croissante sur les mécanismes belges de solidarité.

Aussi le gouvernement belge a-t-il décidé en 2005 de renforcer les contrôles en développant un système électronique de monitoring et de contrôle de toutes les formes d'occupation étrangère en Belgique tout en simplifiant les démarches administratives à effectuer par les parties concernées.

Cette volonté politique a donné naissance à l'application Limosa, du nom d'un groupe d'oiseaux migrateurs. Tout travailleur salarié, indépendant ou stagiaire qui vient travailler temporairement en Belgique doit dorénavant le communiquer, via cette application, à la sécurité sociale belge, sauf s'il en est exceptionnellement exempté. Ainsi, une entreprise étrangère (par exemple chinoise) qui envoie une personne travailler en Belgique peut souhaiter remplir depuis son pays les différentes modalités administratives nécessaires à ce déplacement.

Il serait cependant difficile et fastidieux de mettre en place une application permettant de gérer l'ensemble des caractères du monde. Certes, Unicode le permettrait, mais des processus de conversion devraient tout de même être définis pour que l'information puisse être traitée par les institutions belges. Pour simplifier la tâche, la problématique peut être réduite à l'espace Schengen[20]. De

[20] Depuis l'adhésion de certains des nouveaux pays membres de l'Union européenne, l'espace Schengen comprend les pays suivants : l'Allemagne, l'Autriche, la Belgique, le Danemark, l'Espagne, la France, la Finlande, la Grèce, l'Italie, le Luxembourg, les Pays-Bas, le Portugal, la Suède, l'Estonie, la Hongrie, la Lettonie, la Lituanie, Malte, la Pologne, la Slovaquie, la Slovénie et la

fait, quand des travailleurs, hors Schengen, souhaitent venir travailler en Belgique, ils doivent être munis d'un passeport, sur lequel figurent différentes informations, telles que le nom, le prénom, etc., transcrites dans l'alphabet latin. Cette transcription est en charge du pays émetteur (ou de son ambassade). Ainsi, il serait possible, pour ces pays, de reprendre les informations enregistrées sur le passeport.

Pour gérer les travailleurs au sein de l'espace Schengen, il serait donc utile de choisir un format de codage qui permette à tous les pays membres de procéder aux déclarations dans leur propre langue.

Par ailleurs, si à l'heure actuelle, l'identification est opérée à l'échelle nationale, il est prévu à terme, dans le cadre de la coopération européenne, de procéder à l'identification des travailleurs à l'échelle européenne en consultant les applications des différents pays membres.

Le même principe devrait être appliqué à la coopération judiciaire.

5.1.3. La coopération européenne

Afin de faciliter la coopération entre pays membres, l'Union européenne favorise le développement d'applications de type portail afin de permettre à un service d'accéder simultanément aux informations contenues dans plusieurs bases de données gérées par les différents participants.

Ainsi dans le cadre de la coopération judiciaire, une application pourrait permettre aux services judiciaires et de police d'accéder aux fichiers des autres pays afin de vérifier si un criminel a déjà commis des méfaits dans les autres pays membres.

Les applications de type portail accordent généralement l'accès à plusieurs ressources par la combinaison d'un index centralisé et de recherches distribuées.

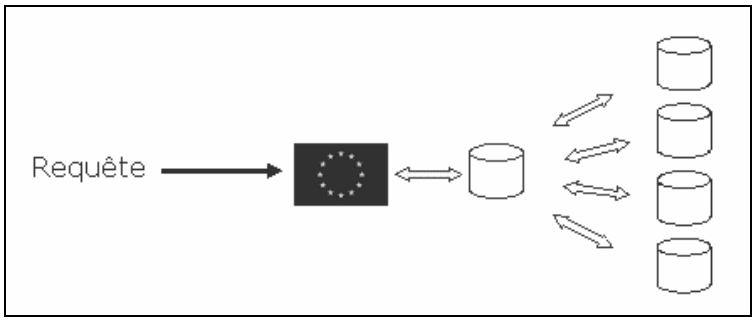


Figure 9 : schéma d'une recherche d'informations sur la base d'un portail

5.2. Une problématique longtemps minimisée

Longtemps, le monde de l'informatique ne s'est guère soucié du codage et de la conversion des données. Certes, la problématique était connue, mais le problème n'a jamais été jugé suffisamment stratégique pour faire l'objet d'une solution organisée et structurelle.

Plusieurs raisons expliquent cette situation :

- historique : comme nous venons de le voir, au début de l'informatique, les codages étaient linguistiquement pauvres. Or, à chaque évolution en informatique, il est nécessaire de tenir compte de l'existant en mettant en

République tchèque ainsi que trois pays associés : la Norvège, l'Islande et prochainement la Suisse où les dispositions entreront en application durant l'année 2008.

place des processus de compatibilité ascendante, qui se sont généralement traduits, dans ce cas-ci, par une simplification linguistique.

- technologique :
  - le mélange de caractères majuscules, minuscules et/ou accentués a tendance à diminuer les performances des applications et des périphériques.
  - les traitements d'information se font généralement sur des systèmes centraux (mainframes) autour desquels sont développés de nombreuses applications chargées d'envoyer et de traiter les résultats. Changer le codage et la richesse des données sur l'ordinateur central nécessite donc la réécriture de ces applications qui se comptent par milliers, voire dizaines de milliers, dans certaines entreprises ou dans l'administration.
  - organisationnelle : malgré les bonnes pratiques en la matière, les développements des applications au sein des entreprises ICT sont trop souvent pris en charge par une équipe qui n'a que peu de contacts avec les équipes travaillant sur d'autres applications. Chaque équipe fait face au problème du codage et de la conversion des données de manière individuelle. L'hétérogénéité des codages et des conversions est donc rarement mise en évidence en raison de cette vision davantage centrée sur les applications que sur les informations.[21]

5.3. Stockage de l'information

Avant de s'interroger sur les problèmes « linguistiques » de l'indexation de l'information, il faut s'interroger sur son stockage, puisque l'indexation ne pourra se faire que sur la base de l'information entreposée dans le système.

La question est d'ordre technologique et conceptuel.

Le nombre de formats disponibles, tant du point de vue du hardware que du software, est généralement limité. Il s'agit donc d'une première contrainte. A titre d'exemple, citons les mainframes d'IBM qui ont été développés avec le format EBCDIC, avant de supporter progressivement Unicode. Or, comme nous l'avons rappelé ci-dessus, les mainframes sont des éléments importants dans les grandes entreprises et le format qu'ils supportent conditionne, en partie, la richesse potentielle de l'information.

5.3.1. Identification des travailleurs

Dans le cadre de l'identification des travailleurs au sein de la sécurité sociale, les diverses applications concernées stockent dans un premier temps les informations introduites par l'utilisateur. Deux types de conversions peuvent être appliqués.

Tout d'abord, lorsqu'un utilisateur introduit dans le système un caractère non admis (car l'application n'accepte éventuellement qu'un sous-ensemble du format utilisé) ou non existant dans le format utilisé (par exemple le caractère « 𓏲 » chiffre tamoul 5 -$0BEB_{16}$- n'est pas défini dans ISO/IEC 8859-1), plusieurs cas de figure se présentent :

- L'utilisateur en a connaissance, soit par consultation des caractères autorisés (par exemple sur le portail de la sécurité sociale), soit par notification du système. Dans ce cas, il procèdera lui-même à la conversion, en-dehors de

[21] A ce titre, il convient de souligner l'initiative de Smals d'initier une démarche globale à toute la société, qui viserait à mettre en place une structure fonctionnelle rassemblant sous une même direction toutes les personnes amenées à travailler au niveau des données.

toute règle préétablie, laissant ainsi les habitudes linguistiques prendre le relais. Ainsi, un Allemand aura tendance à convertir « Müller » en « Mueller » selon les préceptes de la langue allemande.

- Le système convertit automatiquement le caractère dans le caractère le plus proche. De cette manière, « Đ » est converti en « D ».
- Le système accepte le caractère dont le code est identique à un caractère présent dans le format utilisé par le système. Il sera considéré comme ce caractère.[22] Par exemple, pour ISO/IEC 8859-2 (format utilisé en Europe centrale, entre autres pour configurer les claviers), la position $C8_{16}$ équivaut à « Č » alors que, selon ISO/IEC 8859-1, le caractère sera « È ». Le système interprète donc mal le caractère puisque sa « grammaire » de référence n'est pas la même. En fonction du traitement réservé au caractère, les conséquences peuvent être importantes comme nous le verrons dans le cas de l'application Limosa.

La seconde conversion se produit suite à l'introduction des données dans le système pour permettre leur stockage et leur traitement, notamment via le mainframe. Elles sont donc converties dans une forme moins riche. La conversion consiste soit à remplacer un caractère par un autre (en majuscule et exempt de signe diacritique), soit à le remplacer par une espace blanc. Quelques symboles sont acceptés (/, @…).

Or, la conversion n'est pas nécessairement exécutée de la même manière dans les différentes applications. Cette situation résulte directement du manque de prise en compte de cette problématique et notamment de l'organisation des développements des applications au sein des entreprises ICT (cf. 5.1.2). Chaque équipe a donc dressé sa propre table de conversion sans concertation avec les autres équipes (d'autant plus que les applications concernées n'ont pas été créées aux mêmes moments).

Par exemple, dans l'application Dimona[23], le nombre de caractères accentués ou spéciaux pour lesquels est prévu un caractère de remplacement est plus limité que dans la DmfA[24], car l'équipe de Dimona n'était pas assurée de la validité de certaines conversions. Ainsi le « Ø » (lettre majuscule O barré obliquement) scandinave est converti en « blanc » dans Dimona et en « O » dans la DmfA.

Donc, le nom du travailleur « Hågstrøm, Ragnhild » sera converti en :

- « H(blanc)GSTR(blanc)M, RAGNHILD » dans Dimona
- « HAGSTROM, RAGNHILD » dans la DmfA

ce qui pourrait entraîner des résultats différents lors de la recherche d'informations, tant dans le processus d'identification des travailleurs que dans les contrôles croisés Dimona-DmfA.

[22] Situation courante dans les e-mails. A titre d'exemple, voici l'extrait d'un e-mail ayant subi cette transformation : « a engendrÃ© de sÃ©rieux problÃ¨mes de compatibilitÃ© ». Il s'agit ici d'un e-mail envoyé en UTF-8 et interprété en ISO/IEC 8859-1.

[23] La déclaration immédiate de l'emploi ou DIMONA (*Déclaration Immédiate/Onmiddellijke aangifte*) est un message électronique par lequel l'employeur communique les entrées et les sorties de service de son personnel à l'ONSS. La dimension électronique du message rend celui-ci immédiat, c'est-à-dire direct et instantané.

[24] La DmfA (*Déclaration multifonctionnelle/ multifunctionele Aangifte*) permet à l'employeur de transmettre les données de salaire et de temps de travail relatives à ses travailleurs. Cette déclaration est appelée « multifonctionnelle » parce qu'elle peut être utilisée par toutes les institutions. En effet, la DmfA ne se limite pas à la déclaration et au calcul des cotisations de sécurité sociale dues et à celui des réductions. Elle constitue aussi la source des données pour les institutions de sécurité sociale chargées de l'attribution des droits dans la sécurité sociale et du paiement des indemnités. Les secteurs suivants font usage de ces données : l'assurance maladie, le chômage, les pensions, les accidents de travail, les maladies professionnelles, les allocations familiales et les vacances annuelles.

5.3.2. L'application Limosa

Le second problème que nous avions soulevé est le suivant : un caractère X pourrait être accepté dans le système, car il possède le même code que le caractère Y qui, lui, est défini dans la table utilisée par le système. Cette situation peut aisément se produire avec l'application Limosa.

Au vu du traitement que nous venons de décrire et de l'exemple donné ci-dessus, cette situation peut engendrer des modifications importantes de l'information.

Prenons l'exemple d'un travailleur tchèque nommé Ondřej Černý, introduit dans le système à l'aide d'un clavier tchèque (c'est-à-dire configuré sur la base d'ISO/IEC 8859-2). Limosa utilisant la version 8859-1, en fonction des configurations, ce nom pourrait être converti une première fois en « Ondøej Èerný », ensuite une seconde fois en :

- OND(blanc)EJ (blanc)ERN(blanc) dans Limosa
- OND(blanc)EJ EERN(blanc) dans la DmfA

Lors de l'introduction d'une déclaration Limosa, si aucun numéro de sécurité sociale n'est introduit (NISS), le système vérifie les données suivantes qui doivent obligatoirement être présentes dans la déclaration : nom et prénom du travailleur. Si l'identification n'est pas positive, un numéro est créé dans le registre BIS[25].

De plus, dans la DmfA, l'information a été fortement modifiée, passant de Černý à EERN(blanc).

Imaginons maintenant que le même employeur (ou un autre), ayant entre-temps « appris » les caractères autorisés dans le système, réintroduise le même travailleur quelque temps (mois ou années) après en prenant soin de convertir lui-même les caractères problématiques. Notre travailleur tchèque serait donc introduit sous la forme suivante : Ondrej Cerný. La conversion donnerait ensuite ONDREJ CERN(blanc). Il est peu probable que le système fasse le rapprochement avec le travailleur déclaré précédemment et, à nouveau, un numéro sera créé au registre BIS, alors que le travailleur en possède déjà un.

Cette opération répétée de nombreuses fois entraîne une diminution de la qualité de l'information contenue aussi bien dans le registre que dans les applications de la sécurité sociale.

5.4. Indexation de l'information

Une fois stockée, l'information sera indexée afin de pouvoir faire l'objet d'une recherche. Cette indexation sera réalisée sur la base des informations stockées dans le système. Elle variera fortement en fonction des formats utilisés.

De plus, beaucoup d'index ne permettent actuellement pas d'indexer des caractères spéciaux[26] ou étrangers. Ceux-ci subissent donc des conversions en vue de leur indexation.

Comme nous l'avons déjà exposé plus haut, afin de faciliter la coopération entre pays membres, l'Union européenne favorise le développement d'applications de type portail afin de permettre l'interrogation de plusieurs bases de données gérées par les différents pays membres.

[25] Le registre « BIS » comprend toutes les personnes connues auprès des institutions belges de sécurité sociale mais non inscrites dans une commune belge ; il s'agit entre autres des travailleurs frontaliers. Depuis sa mise en production, 36 % des déclarations Limosa ont entraîné la création d'un numéro BIS, soit 32.807 numéros.

[26] Par caractère spécial, on entend tout caractère qui ne correspond ni à une lettre ni à un signe diacritique ou combinatoire, tel que : * $ ( ) ¼ %…

Un tel système à l'échelle européenne implique la gestion de cinq systèmes d'écriture différents : arménien, cyrillique, géorgien, grec et latin. Nous n'évoquerons ici que les problèmes liés à l'alphabet latin.

La plupart des langues employant le système d'écriture latin utilisent des signes diacritiques ou des caractères spéciaux. Initialement conçus pour des systèmes anglo-saxons, les logiciels d'indexation n'en tenaient pas compte. Aujourd'hui, l'informatique n'étant plus centrée exclusivement sur les pratiques anglo-saxonnes, les logiciels doivent prendre en compte les spécificités de chaque langue.

Or, plusieurs langues établissent certains caractères ou signes spéciaux comme des lettres à part entière et ne pouvant être réduites à leur équivalent anglophone sans perte d'information, ce qui n'est pas toujours souhaité. Ainsi, si le « ô » français est considéré comme un « o » accompagné d'un accent circonflexe, le « ø » est considéré lui comme un caractère distinct du « o » en danois et en norvégien, tout comme le « å » n'est pas équivalent à « a » en suédois, ni « Ł » à « L » en polonais.

Par ailleurs, un même caractère peut subir différentes transformations en fonction de la langue dans laquelle il est utilisé. Par exemple, le « ö » est employé, entre autres, en estonien, en allemand et en hongrois. Mais l'indexation différera puisqu'il sera respectivement indexé tel quel comme caractère distinct en estonien, converti en « oe » en allemand et en « o » en hongrois. De même que le ezsett allemand (ß) est soit converti en double « s », soit gardé tel quel (dans ce cas-ci, une recherche d'informations renverra généralement les deux possibilités, comme c'est le cas dans Google).

A titre d'exemple, le tableau qui suit montre les caractères des nouveaux pays membres de l'Union européenne qui existent indépendamment de leur équivalent français « simple » accompagné d'un signe diacritique :

République tchèque	Č Ř Š Ž dans la plupart des bases de données. Dans certaines, Á Ď É Ě í Í Ň Ó Ť Ú Ý ů son également maintenus tels quels
Estonie	Š Ž Õ Ä Ö Ü
Hongrie, Lettonie, Lituanie	Aucun, les signes diacritiques sont généralement ignorés
Pologne	Ą Ć Ę Ł Ń Ó Ś Ź Ż
Slovaquie	Č Ř Š Ž
Slovénie	Č Š Ž

Dans le cadre d'une coopération européenne, il est important que ces informations puissent être comparées. Si le nom d'un criminel est encodé dans une base de données belge et que la Belgique partage cette information avec les autres pays de l'Union, il faudrait idéalement que les caractères aient été traités de la même manière, ce qui pourrait ne pas être le cas comme le montre l'exemple suivant :

Exemple : Lötizer

Français	Lotizer
Estonien	Lötizer
Allemand	Loetizer
Hongrois	Lotizer

Ainsi la même personne sera indexée de manière différente en fonction du pays hébergeant l'information.

5.5. Recherche d'informations

Les problèmes que nous venons de soulever en ce qui concerne le stockage et l'indexation des données auront évidemment des répercussions lors de la phase de recherche. De plus, lors de cette recherche, les informations seront converties à plusieurs reprises pour permettre leur comparaison.

5.5.1. Identification des travailleurs

Dans le cadre de l'identification des travailleurs au sein de la sécurité sociale, la recherche d'informations consistera à vérifier les données fournies par le déclarant en les comparant avec les données contenues dans la source authentique à laquelle l'application a accès.

La source authentique varie en fonction des applications. Quatre d'entre elles (Gotot out[27], DmfA, Dimona, DRS[28]) procèdent à l'identification via le système Oriolus, deux le font par l'intermédiaire du registre de la BCSS, comme le montre la figure 8.

Après le contrôle du numéro d'identification de la sécurité sociale (NISS), pour plusieurs applications, un autre contrôle est effectué sur la base du nom et du prénom. Si ce deuxième processus réussit, l'identification est considérée comme positive.

Les identifications effectuées via Oriolus passent par le mainframe de Smals, BS2000 de Siemens, qui utilise le format EBCDIC, une variante Siemens et non la version IBM. Il est intéressant de souligner que ce mainframe utilise le format EBCDIC, qui n'est pas identique à la version de référence d'IBM. A titre d'exemple, dans le format de Siemens (OSD_EBCDIC_DF03_IRV), la position 4A est attribuée à l'accent grave, tandis que dans la version d'IBM, il s'agit du « Ø » scandinave. Ce format n'accepte que les majuscules et minuscules ainsi que quelques caractères spéciaux.

Pour procéder à la comparaison, Oriolus dispose des données du registre national ou des registres de la BCSS. Or, les données du registre national sont en EBCDIC-RN, c'est-à-dire une version du format EBCDIC modifiée pour les besoins du registre national et qui n'est pas non plus identique à la version utilisée par BS2000. Les données doivent donc être transcodées à plusieurs reprises. Or, la table de transcodage utilisée ne reprend pas à l'entrée la table du registre national. Par exemple, la position 192_{10} convertit le « é » en « E », alors que dans la version du registre national, la position n'est pas attribuée. De même, la position 90_{10} est attribuée au caractère « é », tandis que la table de conversion prévoit pour cette même position la conversion du « è » en « E ».

Quant à la position $B9_{16}$ d'EBCDIC-RN, elle est occupée par le signe degré « ° ». Lors de la conversion à partir du registre national, aucune conversion n'est prévue pour ce signe qui est remplacé par un blanc. Par contre, dans la DmfA, la table de conversion prévoit un remplacement par un « O » !

[27] L'application Gotot OUT (Gotot signifie « GrensOverschrijdende Tewerkstelling - Occupation Transfrontalière ») veille à ce que les travailleurs belges puissent travailler temporairement à l'étranger tout en continuant à jouir pleinement de leurs droits au sein de la sécurité sociale belge.

[28] Un risque social se présente lorsqu'un travailleur ne peut prétendre à son salaire, par exemple en cas de maladie, d'accident du travail, de maladie professionnelle, de chômage ou de maternité. Dans de telles circonstances, la sécurité sociale prévoit un large éventail d'allocations sociales. Pour que le travailleur puisse en bénéficier, l'employeur doit introduire une DRS (déclaration de risques sociaux) auprès d'une des institutions de sécurité sociale.

Certaines erreurs ne portent pas à conséquence, mais illustrent bien les difficultés que l'on peut rencontrer en raison de la multiplication des formats et des tables de conversion. Si nous reprenons le cas du signe degré « ° », il est généralement utilisé dans les adresses pour indiquer le numéro de l'habitation. De ce fait, l'adresse « rue du prince royal n°102 » sera convertie en :

- « rue du prince royal no102 » dans la DmfA ;
- « rue du prince royal n 102 » lors de la conversion de l'adresse en provenance du registre national pour la rendre compatible avec le mainframe BS2000 de Smals.

Plus problématiques sont les différences de conversion d'un même signe en fonction des applications comme nous l'avons déjà expliqué ci-dessus. Sans y revenir, le tableau suivant montre les différences de traitement d'un même signe dans les différentes applications concernées par l'identification des travailleurs et qui posent problème lors de la recherche d'informations :

Caractère	DmfA	Dimona	Cimire	BCSS
Á	A	blanc	non attribué	blanc
Đ	D	blanc	non attribué	blanc
Í	I	blanc	non attribué	blanc
Ò	O	blanc	non attribué	blanc
Ú	U	blanc	non attribué	blanc
Ý	Y	blanc	non attribué	blanc
Ã	A	A	non attribué	blanc
Ê	E	E	non attribué	blanc
Õ	O	O	non attribué	blanc
õ	O	O	non attribué	blanc
Ç	C	C	C	blanc

5.5.2. Coopération européenne

Dans un contexte européen, lors d'une recherche d'informations multilingue et multi-base effectuée à partir d'un portail central, la localisation[29] pose plusieurs problèmes.

Tout d'abord, les métadonnées permettant la recherche ne sont pas nécessairement encodées dans un même jeu de caractères, ce qui nécessite à nouveau une conversion de l'information et de la requête. Plusieurs situations se présentent :

- les caractères de la requête ont un équivalent dans les autres jeux de caractères. Par exemple, la série de normes ISO/IEC 8859 a prévu le même code pour « ä », « ö » et « ü ». Ces caractères peuvent donc être envoyés tels quels dans les requêtes distribuées, laissant aux systèmes interrogés le soin d'ignorer le signe diacritique ou non. Les systèmes sont en effet tolérants à « trop » de détails et acceptent les diacritiques dans les requêtes même si ceux-ci sont ignorés dans l'indexation.
- les caractères introduits n'ont pas d'équivalents dans les jeux de caractères locaux. Une conversion est effectuée et entraîne une perte d'information. Le caractère est converti en sa lettre ordinaire la plus proche ou ignoré.

Cependant, la perte d'information peut être bénéfique. Par exemple un utilisateur introduit « Łotz » (nom d'une ville de Pologne) comme critère de recherche. La requête est envoyée dans les bases de données ayant Unicode comme format de codage ou un format d'Europe centrale uniquement, puisque « Ł » est absent des autres jeux de caractères.

[29] La localisation est le fait de modifier certains paramètres en vue de les rendre compatibles avec la culture de l'utilisateur.

De même, un utilisateur peu au courant des spécificités de la langue polonaise utilisera sans doute la forme simplifiée comme critère de recherche, à savoir « Lotz ». Le danger est que les bases de données polonaises ne renverront que les enregistrements où le nom de la ville a été écrit de cette manière ou liés à un enregistrement où cette forme a été ajoutée comme renvoi. L'usager va ainsi manquer plusieurs informations, dans la mesure où ces bases de données contiendront probablement plus de documents pertinents.

Dans ces deux cas, le silence[30] peut être important comme le montre la figure suivante :

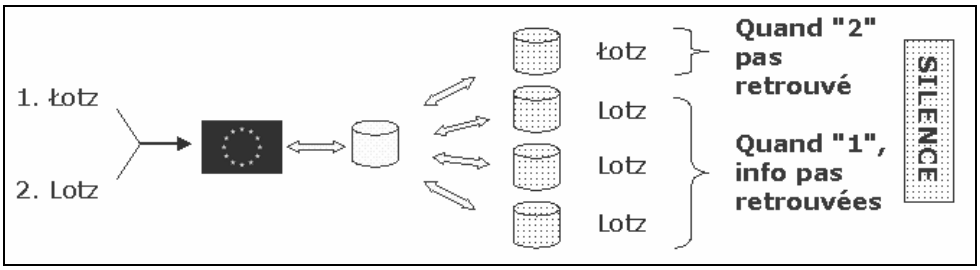


Figure 10 : silence lors de la recherche d'informations en raison des différences d'indexation

Enfin, même si un utilisateur connaît le polonais et souhaite introduire « Łotz » comme critère de recherche, le caractère ne se trouve pas sur le clavier dont il se sert, l'obligeant ainsi à complexifier les manipulations nécessaires à l'introduction de la requête.

5.6. Affichage des résultats

Après la recherche, certains systèmes procèdent à un tri alphabétique des résultats en vue de leur affichage. Deux problèmes se posent : les variations de tri par défaut en fonction des formats de codage et les variations de tri dans un contexte multilingue.

5.6.1. Variation du tri selon les formats de codage

Code ASCII	Code EBCDIC
August	august
Vice versa	co-op
Vice-president	container
august	coop
co-op	August
container	Vice versa
coop	Vice-president

Le premier problème est la variation du tri par défaut en fonction du format de codage. Le tri se base sur les valeurs binaires des caractères. Cette méthode produit des résultats différents selon les codages. L'illustration ci-contre montre les différences de tri alphabétique croissant entre l'ASCII et l'EBCDIC.

Si les données recherchées contiennent des majuscules et minuscules, une impression d'erreur se dégage de ce tri. Un peu de persévérance est nécessaire pour trouver la place de l'information cherchée. Le risque existe de ne pas la trouver.

5.6.2. Variation de tri dans un contexte multilingue

Dans un contexte multilingue, la diversité des caractères présents dans les systèmes ou des profils locaux[31] complique le tri alphabétique pour l'utilisateur qui court le risque de ne pas trouver l'information dont il a besoin car il ne la cherche pas au bon endroit. Par exemple, comparons les ordres alphabétiques des profils français, danois, estonien et slovaque :

[30] Le silence est défini comme les informations pertinentes non extraites lors de la recherche d'informations et dont l'utilisateur n'a donc pas connaissance.
[31] Un profil local est le résultat de la localisation (cf. note 29).

Français : a b c d e f g h i j k l m n o p q r s t u v w x y z
Danois : a b c d e f g h i j k l m n o p q r s t u v w x y z
Estonien : a b c d e f g h i j k l m n o p q r s
Slovaque : a b c č d e f g h i j k l m n o p q r ř s t u v w x y z

On constate que le danois contient trois lettres, l'estonien six lettres et le slovaque cinq lettres supplémentaires par rapport au français. Entre l'estonien et le slovaque, quatre adaptations locales sont nécessaires :

- certaines lettres (t, u, v, w, x, y) viennent après le z en estonien ;
- individualisation d'une lettre avec un signe diacritique après la lettre porteuse : č et ř en slovaque, š et ž dans les deux profils locaux ;
- individualisation d'une lettre avec signe diacritique à la fin de l'alphabet : õ, ä, ö et ü en estonien ;
- traitement d'une paire de lettres comme si elles étaient un seul caractère : ch en slovaque.

Or, dans le cadre d'une coopération européenne, l'utilisateur s'attend évidemment à ce que le tri corresponde à son attente « locale », même si le portail est centralisé. Si un utilisateur belge souhaite obtenir des informations sur un criminel danois nommé Åndersen, en interrogeant, via un portail, les bases de données danoises, il cherchera parmi les noms commençant par la première lettre de l'alphabet et non après le z !

5.7. Synthèse

Comme nous l'avons exposé durant les quatre premières parties de ce chapitre, la problématique du codage et de la conversion des données se rencontre à chaque étape de la chaîne de traitement de l'information.

Dès le stockage de l'information, les choix tant technologiques que conceptuels détermineront la richesse future de l'information. Lors de leur introduction, les données subissent des traitements automatiques ou manuels qui peuvent profondément modifier l'information.

Lors de l'indexation, les variations dans les pratiques d'indexation entraînent des modifications de l'information qui ne sont pas toujours souhaitées et qui peuvent entraîner une baisse de la qualité de la recherche, en raison du bruit[32] ou du silence engendré par les différences d'indexation.

Enfin, l'affichage des résultats est rendu plus complexe lors de la présence de caractères accentués ou spéciaux. Il n'est pas toujours possible de réduire une lettre spéciale à son équivalent « simplifié », sans perte d'information.

5.8. Solutions envisageables

Après l'examen des problèmes liés au codage et à la conversion des caractères, nous allons examiner les pistes de solutions envisageables pour remédier à certains d'entre eux en prenant soin d'examiner les apports et les limites d'Unicode.

[32] Le bruit est l'inverse du silence, il s'agit donc des informations non pertinentes extraites lors de la recherche d'informations et dont l'utilisateur n'a pas besoin.

5.8.1. Importance d'une politique de codage et de conversion des données

Comme nous l'avons déjà exposé ci-dessus, le problème du codage et de la conversion de l'information a été longtemps minimisé et n'a jamais été jugé suffisamment stratégique pour faire l'objet d'une solution organisée et structurelle.

Suite à la prise de conscience croissante des enjeux liés à cette problématique, une solution pourrait donc être de mettre sur pied :

- une politique standardisée de conversion des données ;
- une gestion globale de cette politique dans un processus centralisé.

Politique standardisée de codage et de conversion des données

La définition d'une politique uniforme de codage et de conversion des données est une première étape nécessaire à la résolution du problème.

Politique de codage

En ce qui concerne la politique de codage, l'hétérogénéité ne pourra être évitée totalement. Tout d'abord pour des raisons techniques puisque tous les hardware et software ne supportent pas tous les codages. Ensuite parce qu'il n'est pas nécessaire que toutes les applications disposent d'un format riche du point de vue linguistique.

Cependant, comme il est possible de ne gérer qu'un sous-ensemble de caractères avec Unicode, le passage à ce format pourrait garantir une homogénéité de formats entre les applications, évitant ainsi les transcodages et les problèmes de qualité qui peuvent se poser.

Par ailleurs, une solution visant à étendre les caractères autorisés sur les mainframes est une opération délicate et coûteuse, car elle nécessiterait un énorme travail de réécriture des nombreuses applications, initialement prévues pour accepter aussi bien des informations linguistiquement moins riches ainsi qu'un format bien précis.

Cette uniformisation doit cependant être favorisée. Pour prendre un exemple évident, si des applications en développement se basent sur le standard ASCII, on veillera à ce que la version soit la même (la version belge ou la version américaine – US-ASCII).

Politique de conversion

Une politique de conversion standardisée doit également être définie afin d'éviter des procédures différentes au sein d'applications amenées à gérer les mêmes données, et éventuellement à interagir, comme c'est le cas dans le processus d'identification des travailleurs ou dans le cadre de la coopération européenne.

Dans le cadre de la conversion entre formats linguistiquement différents, il faut distinguer la gestion de l'alphabet latin des autres alphabets.

En ce qui concerne l'alphabet latin, il n'existe pas de normes de conversion entre les caractères d'un même alphabet. Il faut donc opérer des choix. Il est possible de définir ses propres conversions ou de suivre les prescriptions grammaticales en la matière. A ce titre, le eszett allemand « ß », par exemple, se traduit généralement par un double s, de même que le « ü » allemand en « ue ». Cependant, la définition d'une telle conversion entraînera inévitablement une ambiguïté quant au « ue ». Est-ce sa forme originale ou le résultat d'une conversion ? Par ailleurs, cette forme est également utilisée dans l'alphabet turc comme lettre à part entière. Dans ce cas-ci, il n'est donc pas possible de la

convertir en « ue ». Il semble donc difficile de tenir compte des spécificités nationales pour établir les règles de conversion.

A terme, on peut penser qu'une expansion d'Unicode dans les applications permettra de gérer certaines de ces conversions, à condition que les outils de stockage, d'indexation et de recherche s'adaptent eux aussi. Cependant, cette situation n'est envisageable qu'à long terme et elle ne résoudra pas tous les problèmes de conversion.

Il est donc important que des choix concernant les conversions soient opérés pour tous les caractères accentués et spéciaux, nécessaires à une utilisation optimale des applications, et que ces choix soient appliqués à toutes les données.

Les applications actuelles ne gérant pas les alphabets cyrillique, grec, arménien, arabe ou asiatique, l'information doit être transcrite en alphabet latin. Pour opérer cette transcription (qui ensuite devra éventuellement être convertie selon les prescriptions internes à l'alphabet latin), plusieurs normes ISO existent en la matière. Elles sont reprises en annexe (8.4). Dans le contexte actuel, seules les langues grecque et éventuellement cyrillique doivent faire l'objet d'une conversion puisque des pays membres de l'espace Schengen les utilisent.

Gestion globale et centralisée de cette politique de conversion

Une fois cette politique standardisée établie, elle doit être gérée de manière globale. L'entreprise dans son ensemble doit être consciente des enjeux liés au codage et à la conversion des données, et ce critère doit être intégré dans la stratégie de développement de l'entreprise. L'insertion de cette problématique dans les exigences des projets pourrait être envisagée.

En ce qui concerne le projet d'identification des travailleurs, afin d'éviter la multiplicité des conversions et l'éparpillement de la gestion, la politique uniformisée de conversion pourrait être appliquée par l'intermédiaire d'un service informatique centralisé auquel chaque application ferait appel pour procéder à la conversion. Ce service centralisé gèrerait autant les conversions devant être effectuées à l'entrée (conversion des caractères en ASCII) que les conversions nécessaires à la comparaison des données avec les sources authentiques (EBCDIC d'Oriolus).

De cette manière, la politique définie par l'entreprise sera gérée et implémentée de manière centralisée, évitant ainsi les erreurs dues à l'éclatement de l'environnement applicatif.

5.8.2. Unicode : apports et limites

Unicode étant souvent considéré à l'heure actuelle comme LA solution au multilinguisme et au problème de conversion, il est utile de s'interroger sur les apports d'Unicode mais aussi sur ses limites.

Stockage de l'information

En ce qui concerne le stockage de l'information, il est évident qu'Unicode apporte un gain non négligeable puisqu'il permet de stocker une information linguistiquement plus riche et nécessitant donc moins de conversions.

Actuellement, l'application Limosa utilise le format de codage ISO/IEC 8859-1, à savoir la version de la norme ISO/IEC 8859 pour l'Europe occidentale. Or, ce format pose deux problèmes importants.

Tout d'abord, comme nous l'avons vu précédemment, cette version de la norme devrait idéalement être remplacée par la version 15 (Latin-9),

publiée en 1999. Cette version, prévue dès le départ pour remplacer Latin-1, permet l'encodage de lettres supplémentaires. Ces lettres supplémentaires, utiles pour l'encodage de noms étrangers (Š, ž, etc.) sont permises par le remplacement de plusieurs signes de la version 1.

Le second problème, plus important, est que cette version ne permet pas de couvrir l'ensemble des pays de l'espace Schengen, puisque les caractères des nouveaux pays de l'Europe de l'Est (Estonie, Hongrie, Lettonie, Lituanie, Pologne, Slovaquie, Slovénie et République tchèque) ne sont pas supportés, comme le montre la carte ci-contre (les pays en gris foncé font partie de l'espace Schengen ; les pays hachurés sont ceux dont la langue n'est pas couverte par la norme ISO/IEC 8859-1).

Dans ce cas-ci, le passage à Unicode permettrait à l'application de gérer l'ensemble des langues de l'espace Schengen, de sorte que l'ensemble de ces caractères puisse subir un traitement équivalent.

Par ailleurs, il serait possible de garder une trace de l'information telle qu'initialement introduite, autorisant une communication plus aisée, notamment dans le cadre de la coopération européenne, avec les différents pays de l'espace Schengen, qui disposent eux de l'information dans sa forme originale ou dans une forme convertie selon leur pratique (en Allemagne par exemple, le « ü » est stocké sous la forme « ue »).

Indexation et recherche de l'information

Equivalence et compatibilité de caractères Unicode

Tant pour des raisons de compromis linguistiques que pour assurer une compatibilité optimale entre Unicode et les formats précédents, certains éléments textuels ont été codés plusieurs fois. Il faut cependant distinguer deux cas de figure :

1. certains éléments distincts dans Unicode ne sont que des variantes visuelles d'un même caractère ;
2. un même élément peut être codé de plusieurs façons : forme précomposée ou une ou plusieurs compositions dynamiques.

Equivalence de compatibilité

Unicode définit parfois plusieurs codes qui correspondent à des entités qui ne sont que des variantes visuelles d'un même caractère. Ceci doit cependant être relativisé puisque, dans les langues concernées, certains caractères sont considérés comme des lettres à part entière. Par exemple, la ligature et digramme « ij » néerlandais est considérée comme une lettre puisque la majuscule sera notée « IJ » et non « Ij » et qu'elle se classera comme le « Y ». Cependant, par souci de compatibilité, Unicode a défini des **équivalences de compatibilité**. Dans le cas de notre exemple, son équivalent de compatibilité est « i » suivi de « j » (≈ 0069 006A). Toutefois, l'apparence de ces caractères étant légèrement différente, le remplacement d'un caractère par un autre peut entraîner une perte potentielle d'informations de formatage. Aussi certains caractères sont-ils accompagnés d'un balisage supplémentaire.

Exemple :

Original	Code	Nom	Equivalence de compatibilité
¼	00BC	Fraction un quart	≈ <fraction> 1 (0031) / (2044) 4 (0034)
E	FF25	Lettre majuscule latin E pleine chasse	≈ <large> E (0045)
ℋ	210B	Majuscule H de ronde	≈ <police> H (0048)
ﬀ	FB00	Ligature minuscule FF	≈ f (0066) f (0066)

Equivalence canonique

Unicode définit également des processus d'**équivalence canonique** puisque certains éléments textuels peuvent être codés de plusieurs façons : une forme précomposée ou une ou plusieurs compositions dynamiques. De plus, pour des raisons de compatibilité avec les standards préexistants, de nombreuses formes précomposées ont été intégrées dans Unicode. « Ä » est donc l'équivalent de « A » + « ¨ ». Aussi Unicode prévoit-il des équivalences canoniques qui permettent de considérer ces lettres comme équivalentes. De plus, certains éléments pouvant être codés dynamiquement de plusieurs manières, Unicode prévoit également un algorithme de mise en ordre canonique.

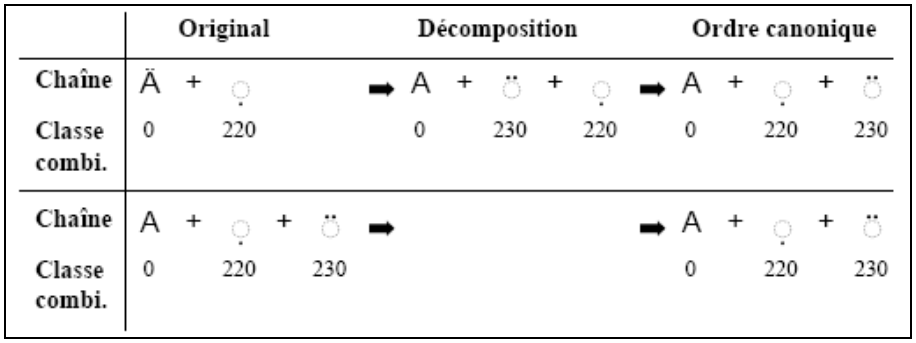


Figure 11 : mise en ordre canonique

Ces processus d'équivalence sont importants puisque ces deux types de conversion d'informations permettent une indexation et une recherche équivalentes pour des caractères initialement différenciés.

A partir de ces deux types d'équivalence, Unicode précise deux formes de décomposition des caractères nécessaires quand on désire comparer des chaînes de caractères.

La *décomposition canonique* d'un caractère est réversible et n'entraîne aucune perte d'information. Elle peut donc être utilisée dans l'échange normalisé de textes. La décomposition canonique d'une chaîne de caractères est la transposition successive et récursive de chaque caractère par sa valeur canonique correspondante jusqu'à ce que ces transpositions renvoient vers elles-mêmes, suivie de sa mise en ordre canonique.

La *décomposition de compatibilité* entraîne une perte d'information. Elle peut cependant être utile à maintes occasions puisqu'elle permet d'éliminer des différences qui ne sont pas toujours pertinentes. Certains caractères ont les mêmes décompositions de compatibilité et sont donc compatibles ; ils ne sont toutefois pas canoniquement équivalents. La décomposition de compatibilité d'une chaîne de caractères est la transposition successive et récursive de chaque caractère par sa valeur canonique et de compatibilité correspondante jusqu'à ce que ces transpositions renvoient vers elles-mêmes, suivie de sa mise en ordre canonique.

Ces deux formes transposent les caractères vers des caractères décomposés. Il existe également des formes de normalisation qui transposent les caractères en caractères composés (s'ils existent). Pour ce faire, il suffit de faire suivre les deux formes de décomposition introduites ci-dessus d'une composition canonique (E + ´ est donc transposé en É).

En faisant suivre ou non les deux formes de décomposition par une composition canonique, on obtient un total de quatre formes de normalisation. Ces quatre formes portent un nom. Les deux formes normalisées vers les caractères précomposés se nomment C et KC. Le résultat de l'une est l'équivalent canonique du texte d'origine, le résultat de l'autre est son équivalent de compatibilité. Le K de KD et KC représente le mot « compatibilité » (suggéré par

l'allemand « kompatibel ») alors que le C renvoie aux deux C de composition canonique. Le D lui se réfère à la décomposition.

	Sans composition canonique	Suivie d'une composition canonique
Décomposition canonique	D	C
Décomposition de compatibilité	KD	KC

Figure 12 : les quatre formes de normalisation

Des chaînes Unicode canoniquement équivalentes doivent toujours être considérées comme égales par les programmes. Une des façons les plus simples de s'en assurer est de normaliser toutes les chaînes de caractères, car si les chaînes sont normalisées, leurs formes canoniques (C ou D) ont exactement la même représentation binaire. La forme de normalisation à utiliser dépend de l'application et de la plate-forme en question.[33]

Illustrons ces formes à l'aide d'un exemple : LJadevič

Nom	Chaîne normalisée
D	LJ + a + d + e + v + i + c + ˇ
C	LJ + a + d + e + v + i + č
KD	L + J + a + d + e + v + i + c + ˇ
KC	L + J + a + d + e + v + i + č

Au vu de la nécessité de comparer des chaînes de caractères introduites par différentes personnes dans de nombreuses applications, il est indispensable de procéder au minimum à une décomposition de compatibilité et canonique, quelle que soit l'origine des données.

En effet, dans le cadre d'une identification des travailleurs étrangers à l'échelle européenne, les données seront utilisées aussi bien dans un cadre national qu'européen. L'utilisation nationale entraînera donc une décomposition de compatibilité de certaines ligatures étrangères. Or, ce processus est irréversible. On ne pourra donc pas procéder à l'opération inverse pour comparer les données avec les administrations européennes. Pour procéder à des recherches, il faudra opérer sur les données en provenance des autres pays membres les mêmes traitements que ceux opérés sur les données nationales.

L'avantage de la décomposition canonique est qu'elle permet de comparer par exemple des données accentuées avec des données dépourvues de signes diacritiques, puisqu'il est possible de construire des algorithmes qui les ignorent, tout en gardant la forme d'origine en mémoire. Nous y reviendrons au point qui traite de l'affichage des données.

Si la sécurité sociale adopte Unicode, cette méthode permettrait de continuer à comparer les nouvelles informations introduites (linguistiquement plus riches) avec les données existantes (linguistiquement plus pauvres), tout en facilitant un passage progressif d'un format à l'autre en récupérant l'existant.

Silence lors de la recherche

Ces décompositions ne permettent de résoudre les différences d'indexation et de recherche qu'entre caractères canoniquement équivalents ou compatibles.

Dès lors, certains caractères spéciaux ne sont pas rapprochés de leur homologue « simplifié », comme « Ø » et « O », ou « Ł » et « L » qui ne sont rapprochés

[33] La forme la plus fréquente est la forme C, car cette forme canonique recomposée est compacte et d'ordinaire compatible avec les jeux de caractères préexistants, c'est d'ailleurs la forme choisie par le W3C pour les standards du web.

d'aucune manière par Unicode. Or, cette simplification est opérée dans les applications de la sécurité sociale.

Cette situation pourrait entraîner un silence éventuellement important lors d'une recherche d'informations au niveau européen, comme nous l'avons déjà mentionné plus haut. Unicode n'apporte ici aucune réponse.

Une première solution pourrait être de sensibiliser les utilisateurs au fait que certains caractères spéciaux sont susceptibles d'être importants. Cependant, cette solution nécessite que l'utilisateur dispose de la forme « enrichie » de l'information qu'il cherche et elle ne convient guère aux processus automatisés.

Une deuxième solution serait de mettre au point des notices d'autorités enrichies en gérant parallèlement une forme enrichie et simplifiée de la donnée. Cette solution pose un problème de lourdeur en termes de gestion de l'information et nécessite des supports de stockage qui permettent de conserver la forme enrichie.

Enfin, une troisième solution pourrait être le doublement des index en spécifiant un index comme international, qui serait utilisé lors des recherches via un portail centralisé. Ce second index serait non localisé, c'est-à-dire exempt de signes diacritiques et de caractères spéciaux, qui seraient convertis en la lettre la plus proche.

D'une part, cette solution nécessite une politique de conversion de l'information telle que nous l'avons expliquée ci-dessus et, d'autre part, elle échange le silence contre le bruit puisque certaines données différenciées dans les profils locaux seraient indexés de la même manière. Elle nous semble cependant la solution la plus adéquate.

Significations différentes d'une même forme graphique

Le standard Unicode a été conçu pour gérer des caractères abstraits et non des glyphes. Nous avons vu que, dans Unicode, par compromis autant que par souci de compatibilité, on retrouve plusieurs codes numériques pour un même caractère.

Cependant, il est courant d'utiliser une même forme graphique pour représenter plusieurs caractères abstraits ayant des significations différentes. Or, si Unicode est un standard informatique, il ne faut pas oublier qu'il sera utilisé par des logiciels devant être eux-mêmes employés par des êtres humains qui ne peuvent dès lors pas nécessairement faire la différence entre des formes graphiques semblables qui ont des significations distinctes. Or, cette différence sémantique, non décelable par un humain, peut avoir des conséquences importantes du point de vue informatique.

Un exemple intéressant est le cas de l'apostrophe. Par souci de compatibilité avec l'ASCII, Unicode a attribué la position 0027_{16} à ce signe. Cependant, il mentionne également que le caractère recommandé pour indiquer l'apostrophe est 2019_{16} et que les caractères recommandés pour les guillemets appariés en anglais sont 2018_{16} et 2019_{16}. Nous remarquerons, en passant, l'ambiguïté de la position 2019 qui exprime à la fois l'apostrophe et le guillemet droit anglais.

Un autre exemple est le caractère « µ » qui renvoie soit à la lettre minuscule grecque « mu » (03BC) soit au symbole « micro » (00B5). Dans ce cas-ci, une équivalence de compatibilité est indiquée entre ces deux signes. Il conviendra donc d'être prudent lorsqu'on souhaite appliquer les équivalences de compatibilité de manière automatique en étudiant a priori le type d'informations géré par le système, ainsi que les besoins et les usages des données.

Un dernier exemple illustre particulièrement bien les problèmes que cette situation peut engendrer. Visuellement, l'eszett allemand « ß » (00DF) ne se différencie pas toujours facilement du béta minuscule grec « β » (03B2). Imaginons qu'une personne introduise le nom d'une personne allemande dans une application et choisisse le béta grec à la place du caractère allemand. Lors de

la conversion, le caractère sera soit ignoré (si le système ne prend pas en charge les caractères grecs) soit converti en « b » et non en « ss », ce qui entraîne une modification importante de l'information initiale.

Cette situation engendre plusieurs problèmes de sécurité. Le lecteur intéressé en trouvera une explication et plusieurs pistes de solutions dans le *Unicode Technical Report #36 : Unicode Security Considerations* disponible sur le site du consortium (cf. annexe 8.2).

Unicode comme langage pivot

Dans le cadre d'un portail permettant de mener des recherches dans plusieurs bases de données utilisant des formats hétérogènes, Unicode peut être utilisé pour assurer l'interopérabilité aussi bien de la requête que des résultats. Chaque requête ou résultat est converti en son équivalent Unicode et ensuite reconverti au format utilisé dans la base de données.

Cette méthode évite de devoir spécifier une table de conversion de chaque format de codage vers tous les autres et réduit ainsi le nombre de tables de conversion nécessaires (de plusieurs milliers à une centaine), ce qui permettrait d'éviter les nombreuses erreurs dues aux problèmes de qualité contenus dans ces tables.

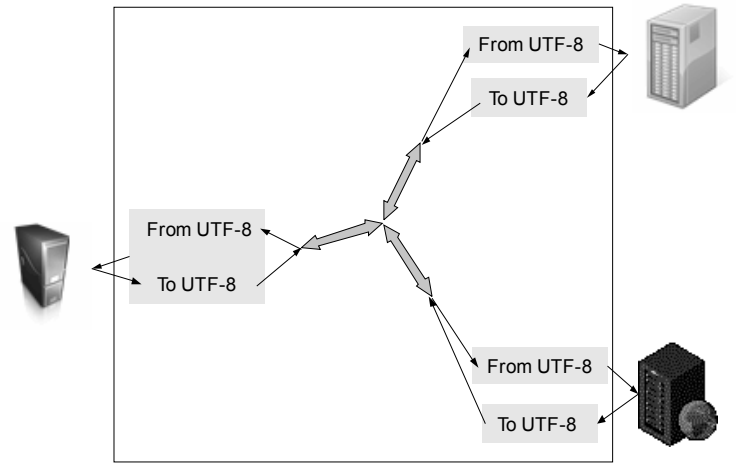


Figure 13 : Unicode comme langage pivot[34]

Affichage

Nous avons souligné les problèmes liés aux variations de tri dans un contexte multilingue. Le tri des chaînes de caractères est un processus largement tributaire de la langue du lecteur cible et même de l'application considérée. En effet, si l'allemand place le « Ä » au début de l'alphabet, le suédois le place après le « Z ». Toute norme de tri se doit de prendre en compte cette grande variabilité. Par ailleurs, le tri de caractères Unicode est rendu plus complexe par l'utilisation de plusieurs octets. Il n'est donc plus possible de trier les informations sur la base de leur valeur binaire.[35]

Aussi, Unicode (dans son rapport technique n°10) et l'ISO (dans sa norme 14651[36]) ont-ils défini des algorithmes de tri opérant sur des chaînes Unicode. Le

[34] D. HETHERINGTON, Case Study : « Unicode in Large Network Systems Management », *12th International Unicode Conference*, Tokyo, Avril 1998, figure 12.
[35] Ainsi, l'édition du *Soleil du Québec* du 18 décembre 2003 annonçait que 300.000 annuaires téléphoniques édités par la société TELUS étaient destinés au pilon suite à un tri effectué sur la valeur binaire des caractères, classant ainsi Dubé après Dubuisson et Bérard après Bzdera. cf. P. ANDRIES, *Unicode 5.0 dans la pratique. Codage des caractères et internationalisation des logiciels et des documents*, Paris, 2008, p. 126. Nous remercions l'auteur pour nous avoir transmis certains chapitres de son livre avant sa publication. Les informations ici présentées concernant le tri en proviennent.
[36] ISO/IEC 14651, Classement international de chaînes de caractères - Méthode de comparaison de chaînes de caractères et description du modèle commun et adaptable de classement. Disponible gratuitement à l'adresse suivante :

rapport technique n° 10 d'Unicode (*UTR-10*) est un « profil » de la norme ISO/IEC 14651. Cette dernière a comme caractéristique principale de ne pas imposer de table de tri implicite. Pour qu'un processus soit conforme, il faut explicitement déclarer un sous-tableau (un « delta ») pour la langue et la culture cibles. En cela, ISO/IEC 14651 reconnaît la nécessité d'adapter l'algorithme implicite à la culture et à la langue locales malgré une table qui propose au départ un modèle valable pour la majorité des langues. Dans cette optique, aucun tri n'est valable pour toutes les langues.

A l'intérieur d'une même langue comme le français, le tri est rendu complexe en raison de la présence d'espaces, de traits d'union, d'apostrophes et de casses différentes. Pour surmonter ces difficultés, la norme ISO/IEC 14651 utilise un modèle à plusieurs niveaux : lettre de base > accent > casse > caractères spéciaux.

Ce qui signifie : les lettres de base sont plus importantes que les accents que portent ces lettres, ces accents sont plus importants que la casse des lettres en question. Les caractères comme le trait d'union, l'espace ou l'apostrophe ou tout autre caractère spécial inclus dans une chaîne de caractères sont considérés au quatrième niveau quand toutes les chaînes sont égales aux trois premiers niveaux. Enfin, en cas d'égalité stricte aux quatre premiers niveaux, il est possible de départager les caractères grâce à leur numéro binaire ou à une autre convention indépendante du codage. Le tableau suivant fournit quelques exemples.

Niveau	Description	Exemples	
1	Caractère de base	e < i	jeune < jeunes < jeunisme
2	Accents	u < û	jeune < jeûne < jeunes
3	Casse	j < J	jeune < Jeune < jeûne
4	Ponctuation	coop < co-op	jeunisme < jeun-isme < Jeunisme

Le premier niveau traite donc les informations à trier sans égard à la casse, aux diacritiques ni aux caractères spéciaux. Le deuxième niveau résout les égalités entre quasi-homoglyphes. Ce tri devra s'adapter à la tradition lexicographique de la langue de l'utilisateur, puisque des différences existent entre l'anglais et le français, comme le montre l'exemple suivant. Si on trie les mots « cote », « coté », « côté » et « côte » :

- à l'anglaise, on obtient : cote < coté < côte < côté ;
- en français, par contre, on les triera dans l'ordre suivant : cote < côte < coté < côté.

Le troisième niveau distinguera les quasi-homoglyphes qui ne se distinguent que par la casse et enfin le quatrième résout les égalités restantes entre quasi-homoglyphes qui ne se distinguent que par la présence de caractère spéciaux.

Pour procéder au tri, d'un point de vue conceptuel, on attribue un poids lexicographique à chaque caractère sur la base de ces quatre niveaux. Par exemple, le « c » est défini par [c, <absence d'accent>, <minuscule>, <pas de caractère spécial>] alors que « É » aura comme élément de tri [e, <accent aigu>, <majuscule>, <pas de caractère spécial>]. Le tri se fera sur la base d'une comparaison des chaînes tout d'abord au premier niveau et puis seulement s'il y a égalité on passera au deuxième niveau.

Dans la pratique, les sous-clés ne seront pas codées de manière aussi verbeuse que celles mentionnées précédemment ; diverses techniques de réduction de sous-clés permettent de réduire considérablement les besoins en mémoire.

http://standards.iso.org/ittf/ PubliclyAvailableStandards/c025066_ISO_CEI_14651_2001%28F%29.zip

Unicode définit une table de tri par défaut, appelée DUCET[37]. Cette table définit une correspondance entre les caractères Unicode et des éléments de tri. Un élément de tri est une suite d'au moins trois poids (des entiers sur 16 bits). Le premier entier représente le poids de premier niveau du caractère (la lettre de base), le second entier représente le poids de deuxième niveau, etc. Cette table attribue donc une valeur numérique aux classes de caractères à chaque niveau. Ensuite, on concatène les sous-éléments de même niveau et on procède à une comparaison binaire.

Prenons un exemple : trions, selon les règles françaises, les deux chaînes « COTÉ » et « Côte ».

Tout d'abord, chaque chaîne a les valeurs suivantes :

- COTÉ : U+0043 U+004F U+0054 U+00C9
- Côte : U+0043 U+00F4 U+0074 U+0065

La première étape de l'algorithme de tri consiste à normaliser la chaîne pour éliminer les différences sans importance. On procède à une décomposition canonique, car elle permet facilement d'ignorer les accents au premier niveau. On obtient alors :

- COTÉ : U+0043 U+004F U+0054 U+0045 U+0301
- Côte : U+0043 U+006F U+0302 U+0074 U+0065

Ensuite on transpose les caractères vers les éléments de tri (le poids individuel de chaque caractère ou groupe de caractères). Nous supprimons les sous-éléments de niveau 4 qui ne sont pas nécessaires ici. Les valeurs des éléments de tri sont les suivantes :

- COTÉ : [**0706**.0020.*0008*] [**0820**.0020.*0008*] [**09FF**.0020.*0008*] [**08B1**.0020.*0008*] [0000.0032.*0002*]
- Côte : [**0706**.0020.*0008*] [**0820**.0020.*0002*] [**0000**.003C.*0002*] [**09FF**.0020.*0002*] [**08B1**.0020.*0002*]

Les éléments de tri qui commencent par [0000...] sont des diacritiques qui seront ignorés au premier niveau. Ensuite on concatène les sous-éléments de même niveau en séparant les niveaux par la valeur 0x0000 :

- COTÉ : **0706 0820 09FF 08B1** 0000 0032 0020 0020 0020 0020 0000 *0008 0008 0008 0008 0002*
- Côte : **0706 0820 09FF 08B1** 0000 0020 0020 003C 0020 0020 000 *0008 0002 0002 0002 0002*

La position du 0032 au deuxième niveau s'explique par l'inversion du stockage des sous-clés d'accents pour le français.

Ensuite on compare de manière binaire en progressant de manière séquentielle jusqu'à trouver une différence.

- COTÉ : 0706 0820 09FF 08B1 0000 **0032** 0020 0020 0020 0020 0000 0008 0008 0008 0008 0002
- Côte : 0706 0820 09FF 08B1 0000 **0020** 0020 003C 0020 0020 0000 0008 0002 0002 0002 0002

La sous-clé de premier niveau est identique, on passe alors au second niveau, les accents. Ici, on trouve immédiatement une différence. « COTÉ » a un accent de valeur 0032 là où « Côte » a une absence d'accent 0020. Or, 0032 est plus grand que 0020, « COTÉ » se trie donc après « Côte ».

Comme nous venons de le voir, il est donc possible de comparer des données accentuées avec des données sans accents sans faire de conversion.

[37] Disponible à l'adresse suivante : http://www.unicode.org/Public/UCA/latest/allkeys.txt

Ces lignes expliquent le principe général du tri selon la norme ISO et Unicode. De nombreuses autres subtilités doivent éventuellement être prises en compte pour personnaliser le tri. Cependant, il existe des bibliothèques (tel ICU) et des classes Java ou .NET qui permettent de trier selon l'algorithme d'Unicode et qui fournissent des tris personnalisés pour de nombreuses langues. Il est par ailleurs toujours possible de personnaliser le tri selon ses besoins.

6. Conclusions

Longtemps minimisée, la problématique du codage et de la conversion des données émerge progressivement en raison de l'interconnexion des réseaux et des applications.

L'évolution des formats de codage induisant la présence d'informations de plus en plus riches sur le plan linguistique a permis de résoudre divers problèmes liés au caractère « pauvre » des premiers codages. Cependant, la richesse croissante de ces formats, en particulier avec Unicode, entraîne autant la résolution d'anciens problèmes que l'apparition de nouvelles difficultés, telles la gestion d'une information plus complexe, la compatibilité avec les informations existantes, la compatibilité de ces informations plus riches avec des applications qui ne sont pas conçues pour les gérer, etc.

De plus, la multiplicité des formats entraîne inévitablement des conversions et transcodages multiples de l'information. Or, ces nombreuses modifications impliquent des éventuelles pertes d'informations comme nous l'avons montré au travers d'exemples simples. Ces pertes sont imputables tant à l'individualisation des solutions apportées à cette problématique qu'à une mauvaise documentation des conversions et des formats. La cause générale en est l'absence de prise en compte de cette problématique au sein des politiques de développement et des sociétés informatiques.

Dès lors, nous pensons que de nombreux problèmes peuvent être résolus, tout d'abord, par une prise de conscience de ce problème, et ce même dans des sociétés n'ayant pas pour vocation de gérer des informations et des applications vendues dans le monde, comme Smals ou les institutions fédérales belges. Par nos exemples, nous avons montré que le problème se pose également au niveau national et européen puisque certaines informations identiques sont traitées différemment selon les applications qui les gèrent alors même que celles-ci sont censées confronter ces données. Il serait donc opportun de définir une politique en la matière en introduisant la prise en compte du codage des données dans le développement des applications. Les conversions pourraient elles aussi faire l'objet d'une politique qui pourrait être gérée de manière centralisée (au minimum au niveau de l'entreprise, au mieux au niveau de la sécurité sociale ou des institutions fédérales) pour éviter la multiplicité des solutions et des incohérences.

Le stockage de l'information est une étape importante dans la résolution du problème puisque le format utilisé à ce stade définira la richesse initiale de l'information. Or, si le problème est surtout technique, il est également conceptuel. La gestion d'une plus grande richesse est également plus complexe. Aussi convient-il de s'interroger sur les besoins et les usages de l'application. Cependant, la réflexion doit être globale puisque certains formats permettront plus facilement des comparaisons avec des données stockées dans d'autres applications qui utilisent un format différent. Ainsi le format ISO/IEC 8859-1 doit être privilégié puisqu'il est compatible avec Unicode. De plus, ce format a

fait l'objet d'une norme et n'est pas propriétaire comme le sont les formats de Windows et d'IBM.

Dans certains cas, la gestion d'une information plus riche peut s'avérer avantageuse. Soit pour communiquer plus aisément avec les administrations ou employeurs étrangers ; dans ce cas, il faudra généralement prévoir également un format plus simple pour la même information de manière à la rendre utilisable tant par les applications que par les personnes qui doivent être en mesure de l'exploiter, ce qui alourdit la gestion de cette information. Soit pour pouvoir en gérer l'appauvrissement linguistique selon ses propres règles définies, nous le rappelons, au niveau de l'entreprise.

Les difficultés posées par l'indexation et la recherche ne peuvent être facilement résolues. Une première solution sera la définition d'une politique en la matière par l'entreprise. Elle permettra d'éviter les différences de traitement d'informations identiques et devant être confrontées.

L'enrichissement de l'information pose des problèmes liés au silence lors de la recherche, tout comme un appauvrissement collectif des informations, en vue d'en effacer les différences posées par la multiplicité des formats, entraînera inévitablement un bruit.

Concernant l'affichage des résultats lors d'une recherche, nous avons souligné la possibilité de le localiser dans un contexte multilingue (tant au niveau des applications qu'au niveau des données comme c'est le cas avec l'application Limosa).

Bien que posant ses propres problèmes, notamment en raison des compromis effectués lors de sa création, le format développé par le consortium Unicode et repris dans la norme ISO/IEC 14646 nous semble apporter des réponses à certains problèmes liés à l'utilisation des formats précédents.

Tout d'abord, vu son objectif de coder l'ensemble des caractères du monde, il facilite les échanges d'informations et supprime (dans la théorie puisqu'il est nécessaire de gérer l'existant) les difficultés liées à l'hétérogénéité des formats et des problèmes de qualité que les multiples conversions de l'information peuvent entraîner.

En ce qui concerne l'indexation et la recherche, les équivalences canonique et de compatibilité permettent de surmonter en partie les problèmes de bruit et de silence ainsi que de gérer des informations plus riches avec les informations existantes, pour autant qu'elles soient codées en ASCII ou ISO/IEC 8859-1 ou qu'une conversion soit effectuée vers ces formats.

A propos de l'affichage des résultats, le consortium Unicode et l'ISO apportent des solutions personnalisables. De plus, plusieurs bibliothèques et projets permettent d'accéder à des tris prêts à l'emploi, tel le projet ICU.

Par ailleurs, nous avons souligné le rôle qu'Unicode peut jouer en tant que langage pivot entre applications à formats hétérogènes. Cette technique peut permettre de réduire considérablement le nombre de tables de conversion et, par là, les problèmes de qualité que la multiplicité de celles-ci occasionne.

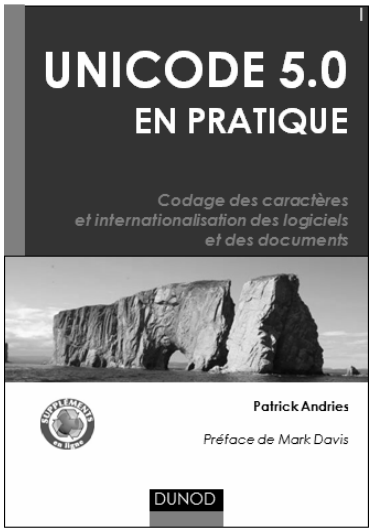
De surcroît, le passage à Unicode nous semble une évolution inévitable. Tout d'abord pour des raisons de gestion des caractères accentués et spéciaux, évitant ainsi certains problèmes causés par leur non gestion, et pour des raisons de facilité de transfert de l'information (en parallèle avec l'extension constante du format XML pour l'échange d'informations). Mais aussi parce que ce passage permet de prendre en compte les particularités des langues, assurant ainsi à chaque citoyen un traitement adapté des informations le concernant, et en premier lieu les informations qui le caractérisent le mieux, à savoir ses nom et prénom, surtout dans une société multiculturelle comme la nôtre. Le sentiment positif pour le citoyen dans ses contacts avec l'administration en sera immanquablement augmenté.

Il nous semble donc indispensable que Smals et ses institutions membres continuent à suivre l'évolution de cette norme et les bénéfices qu'elle peut apporter, tout en restant conscient des problèmes qu'elle peut entraîner. Ceci d'autant plus qu'Unicode s'impose progressivement comme standard par défaut dans les applications et les échanges au travers de l'XML.

De ce fait, il serait judicieux de s'interroger systématiquement sur l'opportunité et l'utilité de passer au format Unicode, principalement lors des reengineering des applications et des bases de données actuellement en cours ou à venir, tels que le reengineering des applications Dimona et Oriolus.

En parallèle, il convient de fixer une politique de codage de l'information, ainsi qu'une politique de conversion à l'instar de celle actuellement développée par Smals dans le cadre du projet de l'harmonisation de l'identification des travailleurs au sein de la sécurité sociale belge.

7. Références



- P. ANDRIES, *Unicode 5.0 en pratique. Codage des caractères et internationalisation des logiciels et des documents*, Paris, 2008.
- Site web du consortium Unicode : www.unicode.org
- J. ANDRE et H. HUDRISIER, « Unicode, écriture du monde ? », *Document numérique*, 6, 3-4, Paris, 2002. Disponible en ligne à l'adresse suivante : http://www.cairn.info/sommaire.php?ID_REVUE=DN&ID_NUMPUBLIE=DN_063 dont les articles :
  - J. ANDRE, « Caractères, codages et normalisation. De Chappe à Unicode », p. 13-49.
  - E. GIGUET et N. LUCAS, « Intégration d'Unicode. Conception d'un agent de recherche d'information sur internet », p. 225-236.
- J. KORPELA, « A Tutorial on Character Code Issue », *IT and Communication*, http://www.cs.tut.fi/~jkorpela/indexen.html (consulté le 23 mai).
- Site web d'ARCANA PERCIPIO : http://www.arcanapercipio.com/index.html (consulté le 23 mai).

8. Annexes

8.1. Langues couvertes par les normes ISO/IEC 8859

Norme	Nom de l'alphabet	Description
-1	Alphabet latin 1	Europe occidentale
-2	Alphabet latin 2	Europe centrale
-3	Alphabet latin 3	Maltais et espéranto
-4	Alphabet latin 4	Balte (danois, estonien, finnois, suédois)
-5	Alphabet latin/cyrillique	Russe, bulgare, biélorusse
-6	Alphabet latin/arabe	Arabe
-7	Alphabet latin/grec	Grec moderne (monotonique)
-8	Alphabet latin/hébreu	Hébreu et yidiche (sans voyelle)
-9	Alphabet latin 5	Turc (Latin-1 moins 6 lettres islandaises, remplacées par 6 lettres turques)
-10	Alphabet latin 6	Nordique (lapon, islandais, esquimau)
-11	Alphabet latin/thaï	Thaï
-12	Initialement prévue pour le devanāgarī, mais le projet a été abandonné	
-13	Alphabet latin 7	Balte (estonien, finnois, letton, lituanien)
-14	Alphabet latin 8	Celte (gallois, cornique, gaélique)
-15	Alphabet latin 9	Version « euro » du latin 1, Europe occidentale mais particulièrement le français
-16	Alphabet latin 10	Balkans (roumain, croate et slovène)

8.2. Présentation du site internet du consortium Unicode

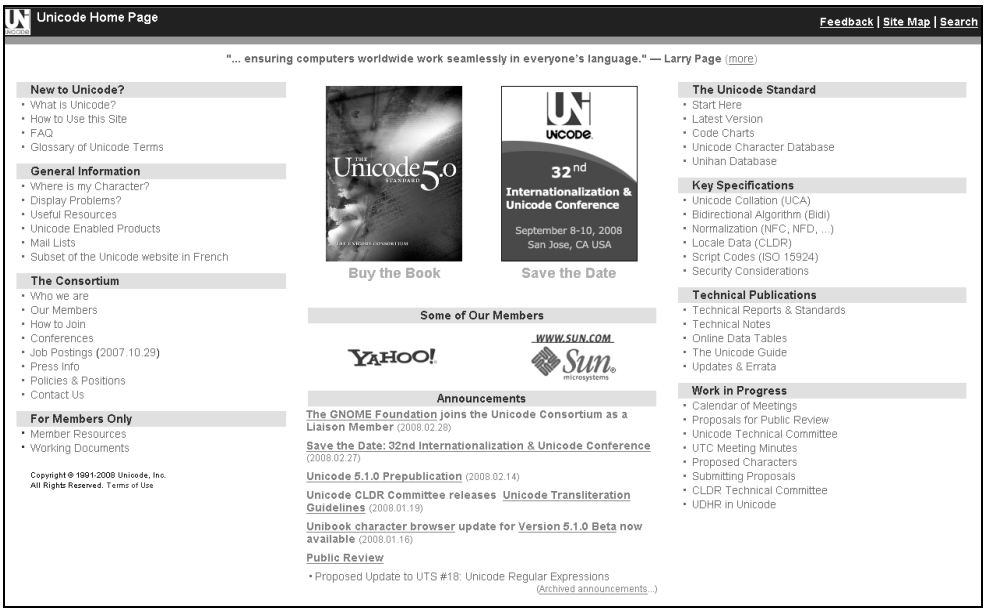


Figure 14 : page d'accueil du site du consortium Unicode

Le consortium Unicode a développé un site (www.unicode.org) extrêmement riche en informations sur les développements liés à Unicode, les tables des caractères, etc. Aussi nous a-t-il semblé intéressant d'en faire une courte présentation.

Les informations de présentation d'Unicode et du consortium se trouvent dans le menu gauche. Les informations liées aux aspects techniques et conceptuels se trouveront essentiellement dans le menu de droite.

Une première partie « The Unicode Standard » permet d'accéder à la dernière version du format, aux tables de caractères (*Code Charts*) ainsi qu'à la *Unicode Character Database* qui contient des fichiers sur les propriétés des caractères et l'organisation du format.

La seconde partie permet d'accéder rapidement aux principales spécifications du standard. Parmi elles, le *Common Locale Data Repository* a pour objectif de réunir et documenter les aspects locaux (format des dates, des heures, tri des textes, dénomination des langues, langues présentes dans chaque pays, etc.) et de les mettre à la disposition de tous pour soutenir le développement de logiciels et d'applications à travers le monde.

La troisième partie renvoie à l'ensemble des publications techniques (*Technical Publications*). Elles sont de deux types.

Les *Technical Reports* couvrent un large éventail de sujets liés à la mise en œuvre ou au développement du standard Unicode, elles-mêmes divisées en trois catégories, les *Unicode Standard Annex (UAX)* qui font partie intégrante du standard, les *Unicode Technical Standard (UTS)* qui sont des spécifications indépendantes et les *Unicode Technical Report (UTR)* qui contiennent des informations importantes (par exemple le *TR#36 Unicode Security Considerations*).

Les *Technical Notes* fournissent des informations sur une variété de sujets liés aux technologies de l'Unicode et de l'internationalisation.

Enfin une dernière partie fournit des informations sur les travaux en cours et leur calendrier.

8.3. Plan multilingue de base d'Unicode

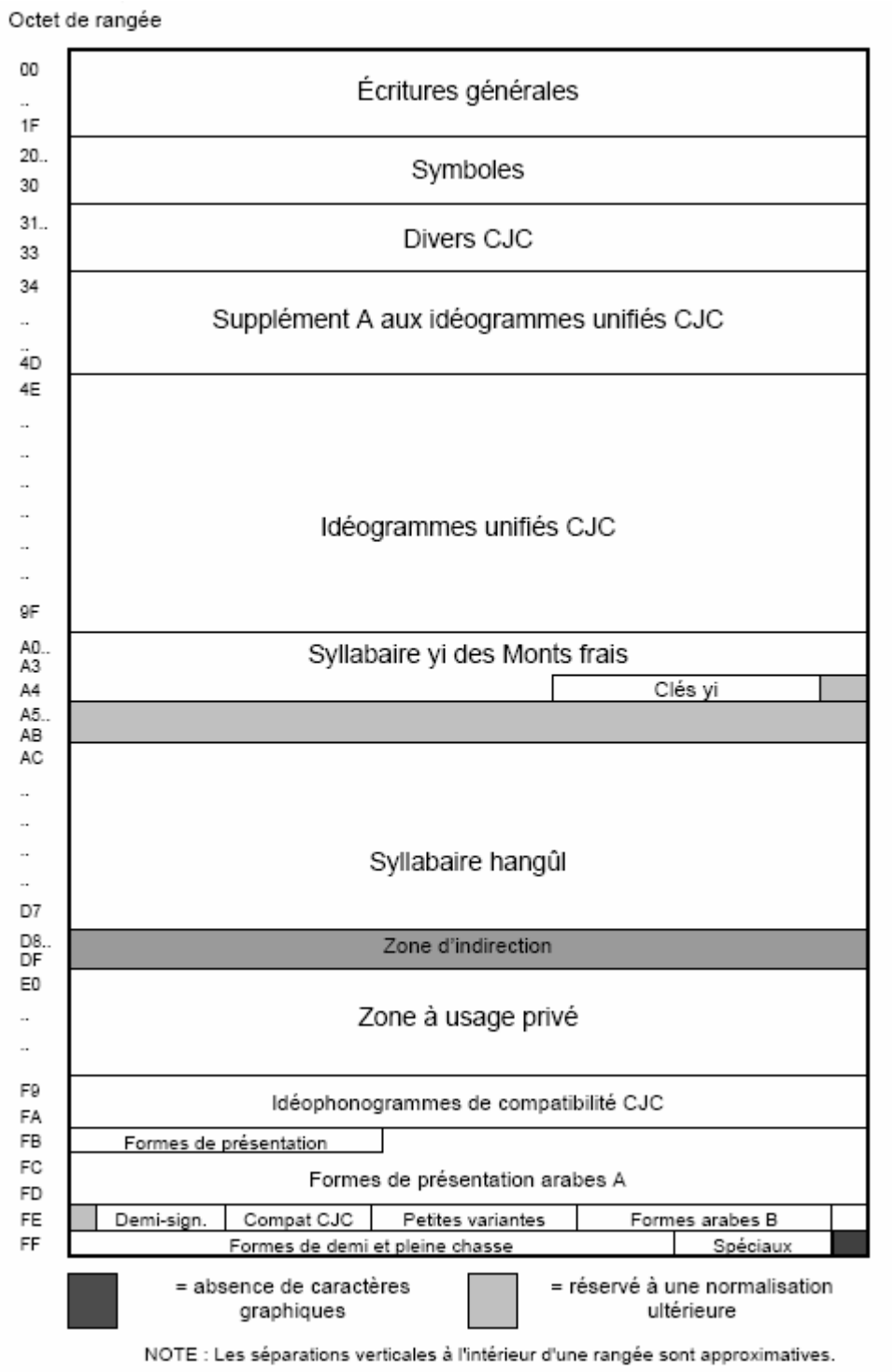


Figure 15 : structure générale du PMB

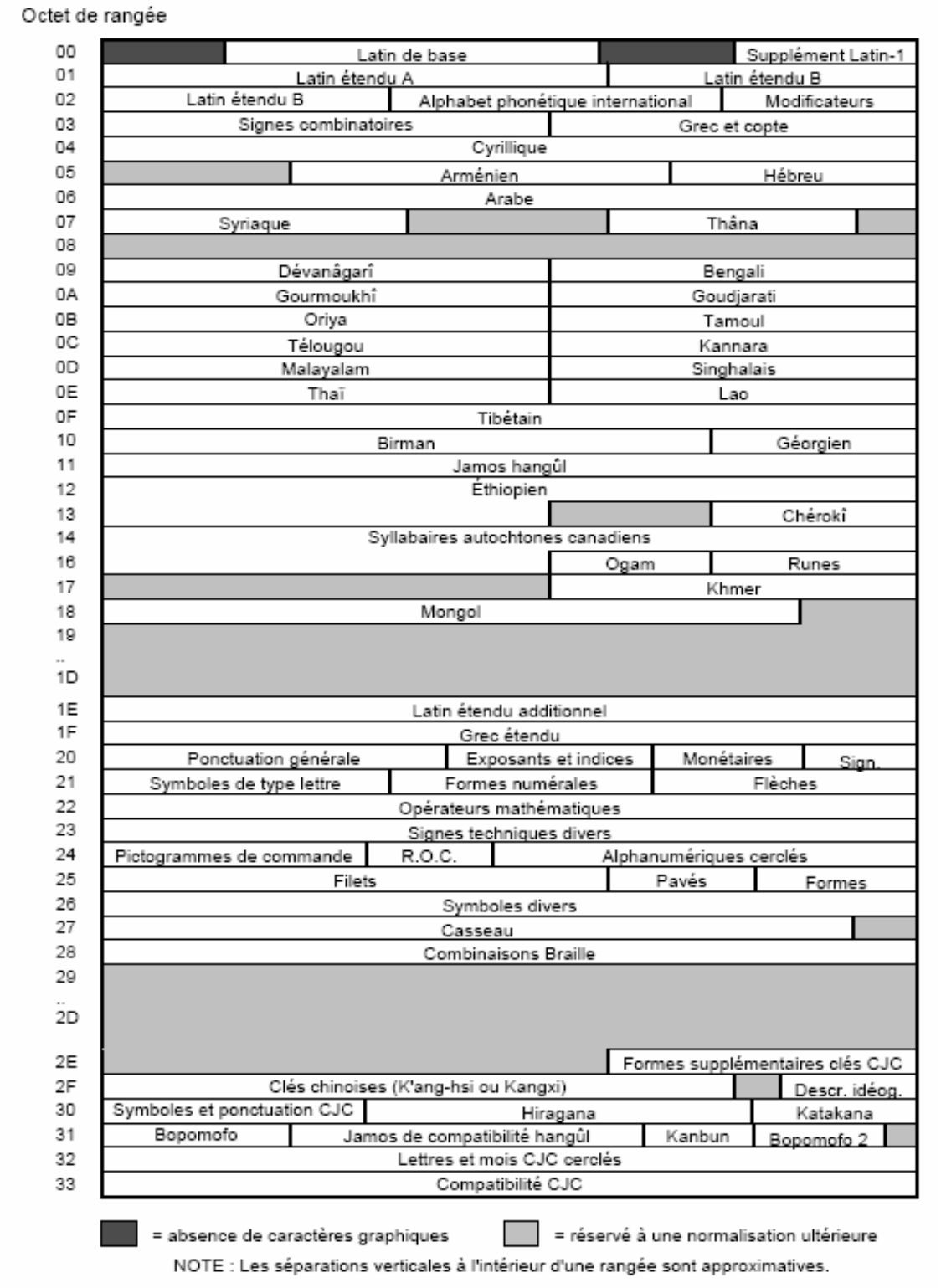


Figure 16 : zone des écritures générales du PMB

8.4. Normes de transcription dans l'alphabet latin

Référence	Titre	Année d'édition
ISO 233:1984	Information et documentation. Translittération des caractères **arabes** en caractères latins.	1984
ISO 233-2:1993	Information et documentation. Translittération des caractères arabes en caractères latins. Partie 2, Langue **arabe**. Translittération simplifiée.	1993
ISO 233-3:1999	Information et documentation. Translittération des caractères **arabes** en caractères latins. Partie 3 : **Persan**. Translittération simplifiée (disponible en anglais seulement).	1999
ISO 9985:1996	Information et documentation. Translittération des caractères **arméniens** en caractères latins.	1996
ISO 7098:1991	Information et documentation. Romanisation du **chinois**. - 2e édition	1991
ISO/TR 11941:1996	Information et documentation. Translittération de l'écriture **coréenne** en caractères latins.	1996
NF ISO 9 : 1995	Information et documentation. Translittération des caractères **cyrilliques** en caractères latins. Langues slaves et non slaves.	1995
ISO 15919:2001	Information et documentation. Translittération du **Devanagari** et des écritures indiennes liées en caractères latins (disponible en anglais seulement).	2001
ISO 9984:1996	Information et documentation. Translittération des caractères **géorgiens** en caractères latins.	1996
ISO 843:1997	Information et documentation. Conversion des caractères **grecs** en caractères latins.	1997
ISO 259:1984	Information et documentation. Translittération des caractères **hébraïques** en caractères latins.	1984
ISO 259-2:1994	Information et documentation. Translittération des caractères **hébreux** en caractères latins. Partie 2, Translittération simplifiée.	1994
ISO 3602:1989	Information et documentation. Romanisation du **japonais** (écriture en kana).	1989
ISO 11940:1998	Information and documentation. Translittération du **thaï**.	1998
ISO 11940-2:2007	Information et documentation. Translittération des caractères **thaï** en caractères latins. Partie 2, Transcription simplifiée de la langue thaï.	2007

9. Glossaire

ASCII	American Standard Code for Information Interchange
Dimona	Déclaration Immédiate/Onmiddellijke aangifte
DmfA	Déclaration multifonctionnelle/ multifunctionele Aangifte.
DRS	Déclaration des risques sociaux
DUCET	Default Unicode Collation Element Table
EBCDIC	Extended Binary Coded Decimal Interchange Code
Gotot IN	GrensOverschrijdende Tewerkstelling - Occupation Transfrontalière (pour les travailleurs entrants)
Gotot OUT	GrensOverschrijdende Tewerkstelling - Occupation Transfrontalière (pour les travailleurs sortants)
ICU	International Components for Unicode
ISO/IEC	International Organization for Standardization / International Electrotechnical Commission
Limosa	Déclaration obligatoire pour les travailleurs ou stagiaires qui effectuent des prestations en Belgique
OAIS	Open Archival Information System
Oriolus	Système d'identification de Cimire
UTR	Unicode Technical Report
UTF	Unicode Transformation Format