Chapitre 3 : Modélisation et Conception

Ce chapitre présente la modélisation des fonctionnalités du système et l'organisation des différents modules entre eux. Les schémas utilisent le formalisme du langage de modélisation UML (cas d'utilisation élaborés et diagramme de classes).

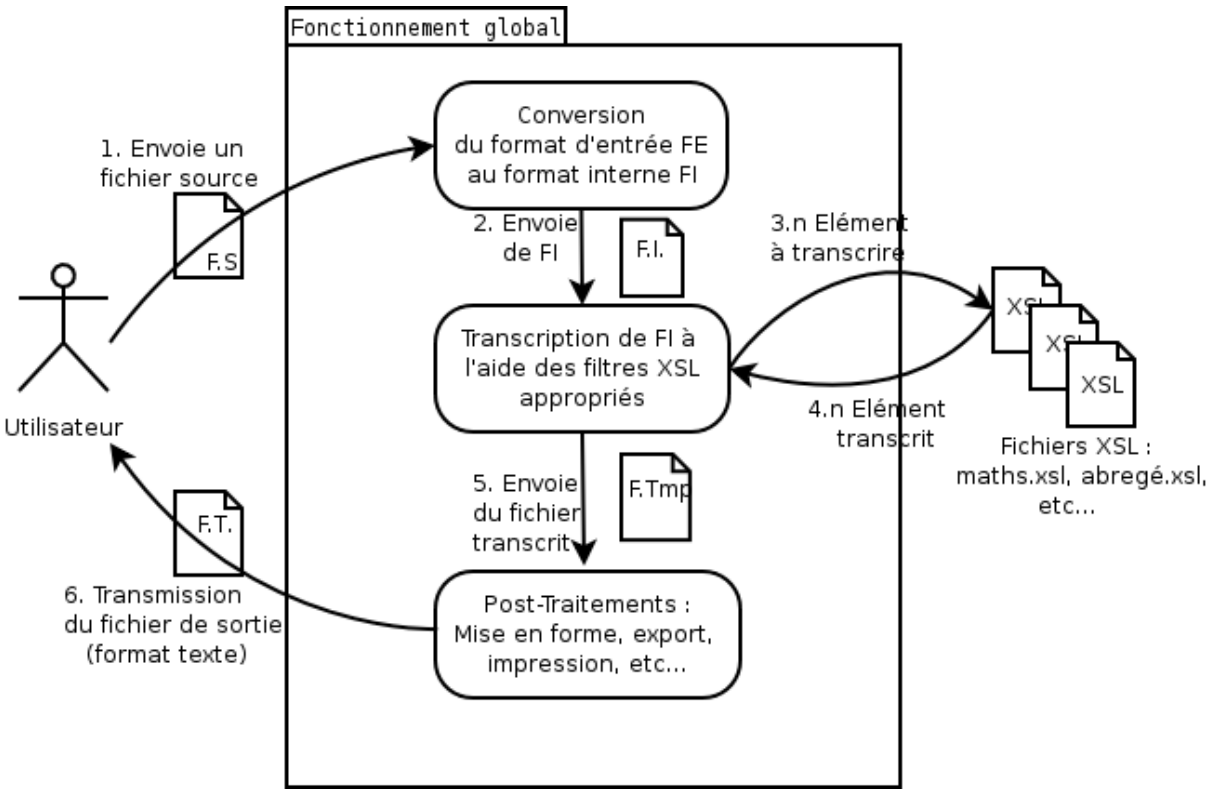
Nous terminons par l'explication des choix des environnements de développement puis d'exécution et des technologies utilisées.

3.1 Fonctionnement général du système

Nous présentons dans cette partie l'organisation des différents modules entre eux ainsi que les principaux formats et flux de données.

3.1.1 Architecture générale du système

Afin de répondre efficacement aux contraintes de simplicité d'utilisation, nous souhaitons minimiser au maximum les interactions de l'utilisateur avec le système. Dans l'idéal, l'utilisateur se contentera de donner un fichier source (F.S.) et obtiendra en retour un fichier transcrit (F.T.).



Nous avons identifié trois modules principaux dans la chaîne de traitement du fichier source :

- **le module de conversion** dont le rôle consistera à transformer le fichier source en un fichier au format interne;
- **le module de transcription** qui effectuera le transcodage du fichier source en braille;

- **le module de post-traitement** qui sera chargé des aménagements à apporter au fichier transcodé afin de le rendre directement exploitable par l'utilisateur selon ses exigences.

3.1.2 Principaux formats et flux de données

Le système sera amené a effectuer de nombreuses transformations tout au long de la chaîne de traitements. Voici les principaux fichiers et formats qu'il utilisera :

- F.S. : Fichier Source ou fichier d'entrée (format texte ou binaire). Ce fichier est transmis directement au logiciel par l'utilisateur et contient des données susceptibles d'être transcrites en braille (typiquement, un fichier provenant d'un logiciel de traitement de texte);
- F.I. : Fichier au format Interne (format texte, en XML). Il permet de présenter les données des formats hétérogènes d'entrée de la même manière, et ainsi de ne pas complexifier la transcription;
- XSL : Fichier XSL filtre (format texte, en XSL);
- F.Tmp : Fichier transcrit, avant post-traitement (format texte). C'est le résultat du transcodage de FI par les filtres XSL.;
- F.T. : Fichier transcrit (format texte). C'est le fichier de sortie remis à l'utilisateur.

Le logiciel doit être capable de prendre en compte un très grand nombre de format de fichiers d'entrée. Il manipulera donc des fichiers sources textes (documents textes, XML, HTML...) ou binaires (documents provenant de logiciels spécifiques d'édition comme Openoffice Writer ou Microsoft Word, ou de formats standards comme RTF).

En revanche, les modules de transcription et de post-traitement n'utiliseront en entrée qu'un seul type de fichiers (formats internes).

En sortie, le module de post-traitement pourra produire différents types de fichiers texte suivant sa configuration. La possibilité de produire des fichiers binaires n'est pas à exclure pour les développements futurs : certaines embosseuses ou imprimantes pourraient ainsi recevoir directement du code machine.

3.1.3 Technologies utilisées lors de la transcription

Le mécanisme de transcription se base sur les outils de transcodage qu'offre le processeur XSLT : à chaque type de contenu, nous ferons correspondre un filtre XSL chargé du transcodage spécifique.

Le choix de cette technologie et son utilisation exacte sont détaillées plus bas.

Afin d'intégrer facilement XSLT dans le processus de transcription, nous avons à notre disposition plusieurs API et plusieurs langages.

Cependant, JAVA propose le plus grand choix d'implémentation robustes d'XML et de XSL (les API SAX et DOM par exemple). Nous intégrerons donc probablement nos feuilles de style grace à une de ces API.

3.2 Fonctionnement détaillé des modules du système

Nous présentons de manière exhaustive dans cette partie les différents modules identifiés précédemment et terminons par la définition des options à prendre en compte.

3.2.1 Conversion du format d'entrée

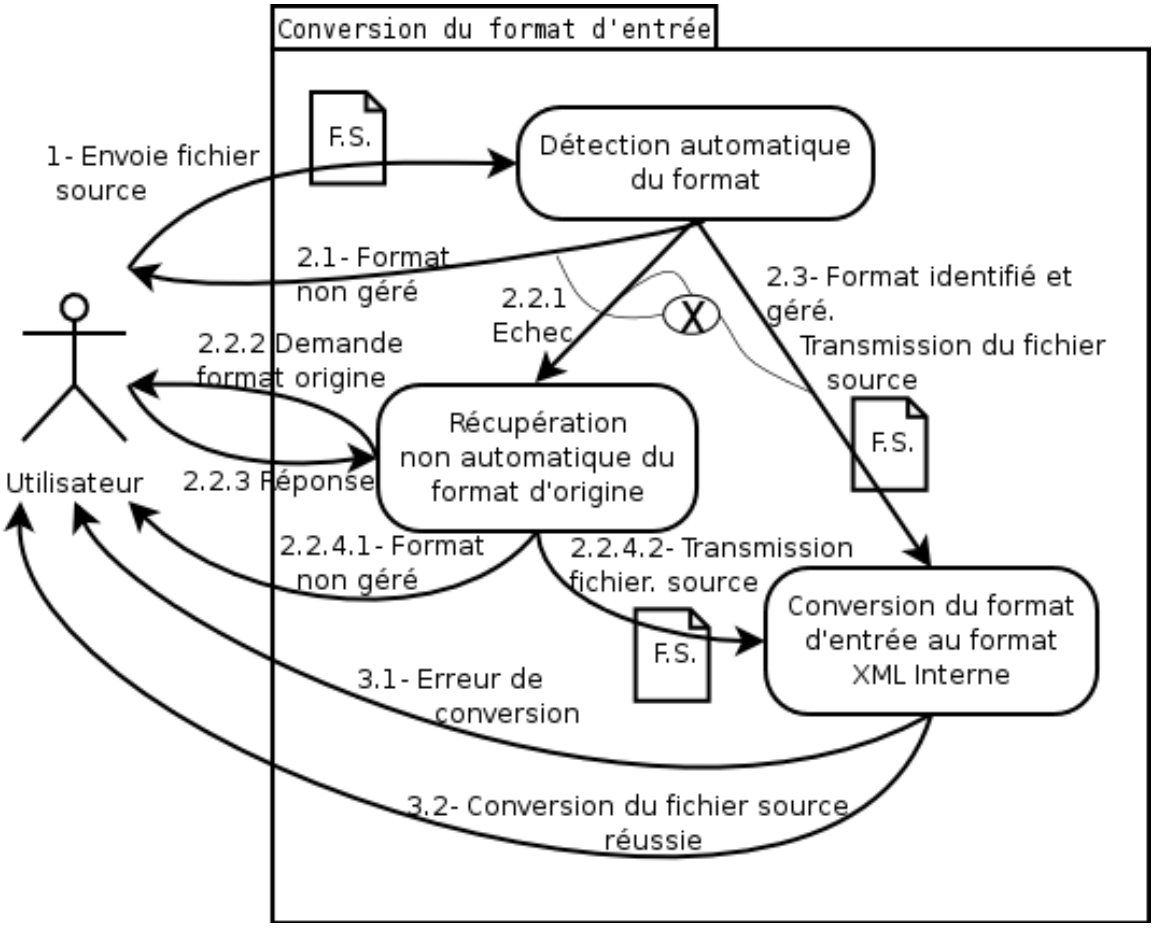
L'objectif de ce module est de convertir le fichier source au format interne.

En effet, il semble plus facile de gérer dès le début des traitements la multiplicités des formats d'entrée possibles dans un module spécifique, plutôt que de déléguer aux modules suivants la charge d'adapter leurs traitements en fonction des différents formats.

L'utilisation d'un format interne offre deux intérêts majeurs :

- garantir au module de transcription un format unique et syntaxiquement correct puisqu'il résulte d'un traitement en amont;
- disposer d'un format créé sur mesure par le système, donc offrant s'il est bien conçu la meilleure présentation possible des données pour les traitements à effectuer.

L'inconvénient d'une telle organisation est qu'il faut prévoir pour chaque type de format d'entrée un traitement spécifique. Devant la multiplicité des formats possibles, il conviendra donc dans un premier temps de ne prendre en compte que les plus utiles et les plus représentatifs d'entre eux.



La conversion du document d'entrée peut générer trois types d'erreurs :

- le format du fichier n'est pas géré : l'erreur est alors bloquante;
- le format du fichier n'est pas reconnu automatiquement : il faudra alors demander à l'utilisateur de préciser quel format il utilise, l'erreur n'est pas bloquante;
- la conversion du document ne s'est pas déroulée correctement (cas d'un fichier endommagé par exemple ou mal détecté) : l'erreur est bloquante.

Une erreur bloquante entraînera nécessairement la fin du traitement et sera notifiée à l'utilisateur.

De même, la réussite de la conversion devra être confirmée, même si 'utilisateur n'a pas à intervenir dans la suite des opérations.

3.2.2 Transcription

Le module de transcription gère à n'en pas douter les fonctionnalités principales de notre logiciel. C'est logiquement le module le plus complexe, tant au niveau des fichiers manipulés qu'au niveau des traitements à réaliser.

Nous reviendrons donc sur les formats et les flux de données spécifiques à ce module après en avoir clarifier le fonctionnement.

3.2.2.1 Fonctionnement général

Le module de transcription comporte quatre phases successives réparties en cinq sous-modules :

- **la phase de configuration** se charge de la récupérations des options de transcription;
- **la phase d'initialisation** prépare la transcription en collectant les filtres utiles et en créant un scénario de transcription;
- **la phase de transcription** réalise le transcodage en appliquant le scénario;
- **la phase terminale** rédige le rapport de transcription et informe l'utilisateur sur le déroulement des traitements effectués.

La phase d'initialisation permet une adaptation fine au fichier à transcrire plutôt que d'appliquer systématiquement le même traitement : supposons qu'un document ne contienne que des écritures littéraires, il serait inutile de charger les filtres de conversion des mathématiques, de la chimie, ou de la musique.

De même le programme s'adapte dynamiquement à la configuration de l'utilisateur.

La mise en place du scénario permet d'une part d'optimiser le logiciel, d'autre part de le rendre facilement paramétrable. C'est le scénario qui déterminera les traitements à effectuer en fonction du contexte de transcription.

De plus, certains filtres sont à appliquer prioritairement suivant les configurations (par exemple, l'abrégé mathématique est prioritaire sur l'abrégé littéraire).

Ainsi préparée, la phase de transcription n'a plus qu'à suivre le scénario à la lettre. Il est d'autant plus important d'alléger cette étape qu'elle reste la plus difficile à implémenter techniquement.

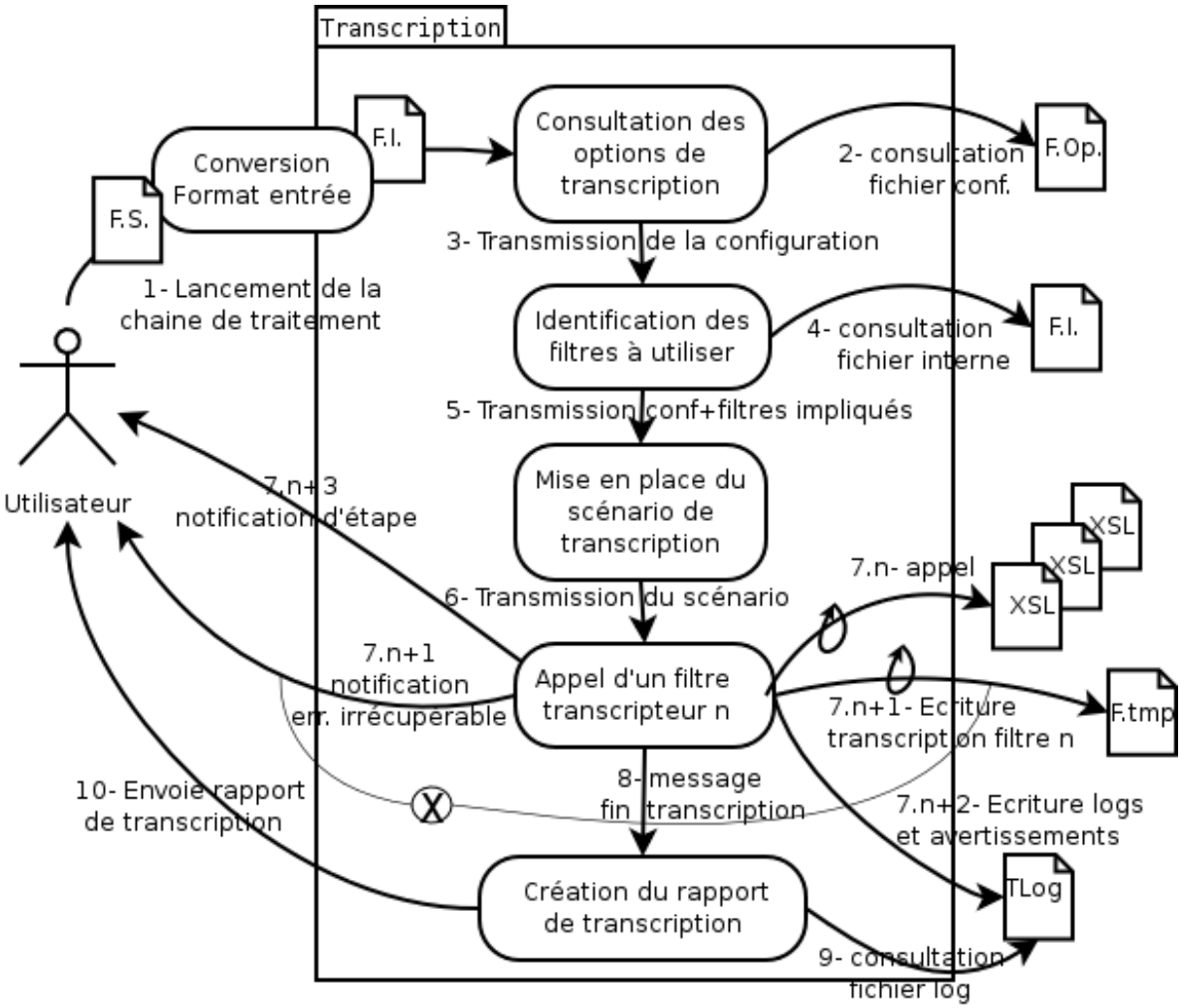
Le rapport de transcription permet à l'utilisateur d'obtenir un retour sur le déroulement de cette étape. Ce rapport sera à adapter en fonction des exigences de l'utilisateur : un transcripteur professionnel souhaitera un rapport fonctionnel détaillé, un développeur

préférera un rapport technique exhaustif alors qu'un utilisateur occasionnel ou peu compétent n'aura besoin que des grandes lignes.

Deux types d'erreurs peuvent être générés par le processus de transcription :

- des erreurs bloquantes (par exemple un code de caractère non reconnu ou une entité non prise en compte dans un filtre) : la transcription est alors arrêtées;
- des erreurs non-bloquantes (ambiguïté dans une transcription, plusieurs choix de transcriptions possibles, etc.) : ces erreurs seront notifiées si l'utilisateur l'a demandé et consignées dans le rapport.

L'utilisateur doit s'il le désire recevoir les informations sur le déroulement de la transcription en direct, on lui notifiera donc les différentes étapes du processus. Toutefois, il ne sera pas en mesure d'interagir avec le système durant la procédure si tout se déroule normalement.



3.2.2.2 Description détaillée des fichiers impliqués

- F.S. (Fichier Source), F.I. (Fichier au format Interne), XSL (Fichier XSL filtre), F.Tmp (Fichier transcrit, avant post-traitement) : voir plus haut;
- F.Op : Fichier d'Options, au format texte (possibilité d'utiliser un document XML). Il stocke les préférences des utilisateurs et la configuration en cours.

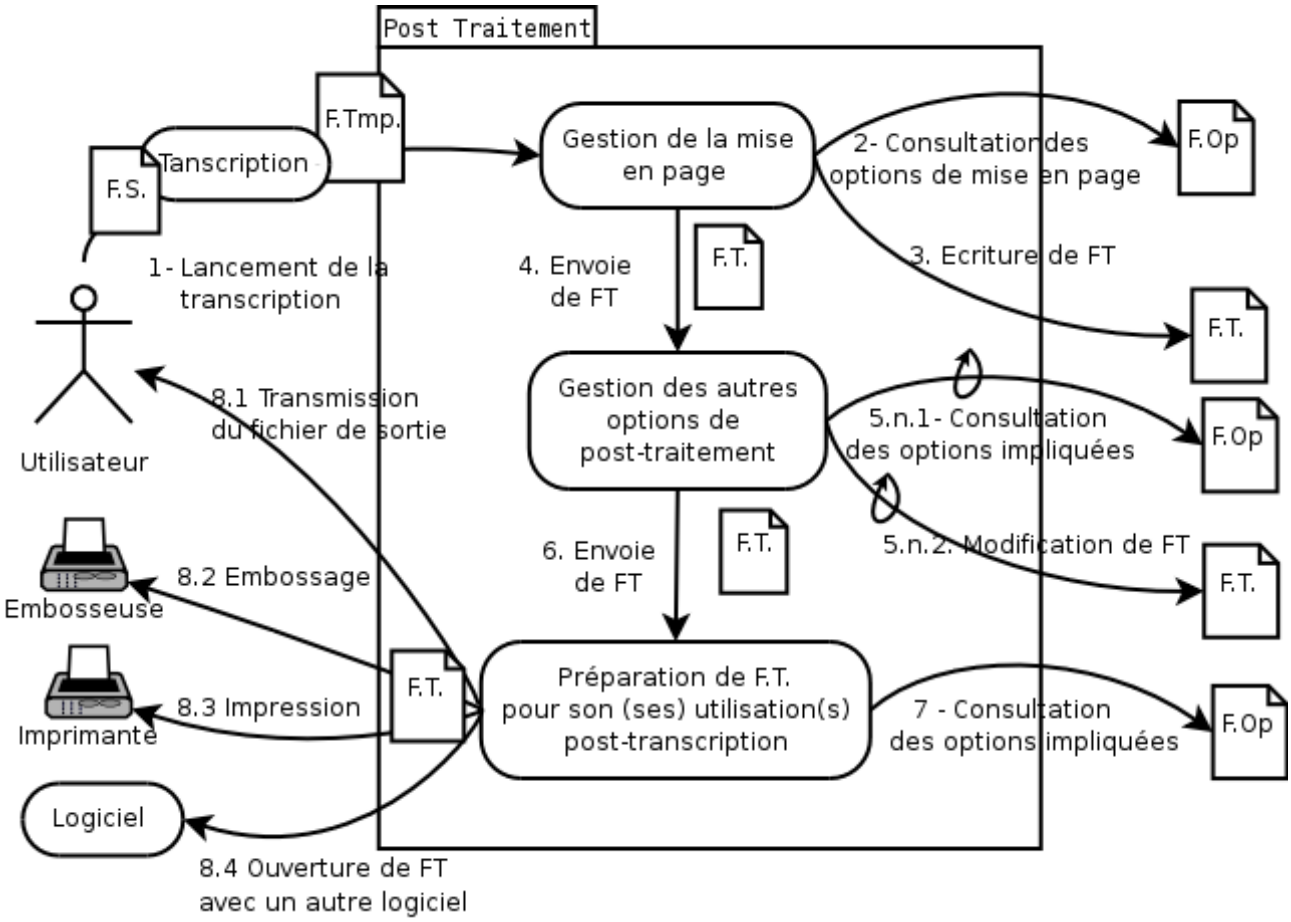
- Tlog : Fichier de logs de la transcription (format texte). Il contient l'ensemble des messages générés lors de la transcription par le processeur XSLT et par le logiciel. Il doit distinguer les différents types de messages afin de pouvoir identifier l'origine, le niveau de détail et l'importance de chaque message.

3.2.3 Post-Traitements

Le module de post-traitement à plusieurs objectifs :

- s'assurer de la bonne présentation du document, en représentant par exemple les structures (titre, listes,...) et les mises en formes (alignements, indentations,...).
- préparer le document afin de le rendre compatible avec le mode de sortie spécifié par l'utilisateur (embossage, impression, lecture sur plage braille, traitement avec un autre logiciel comme par exemple un logiciel d'impression spécifique)
- pouvoir intégrer d'autres fonctionnalités de post-traitement non prévues pour l'instant.

Une fois encore, afin de faciliter l'utilisation du logiciel, cette étape sera entièrement paramétrée à l'aide du fichier d'options.



3.2.4 Paramétrages

Un des critères les plus recherchés par les transcripteurs est la facilité d'adaptation du logiciel à leurs besoins spécifiques. Nous devrons donc dans la mesure du possible offrir une grande variété de paramètres tout en restant pertinent.

Les principales options à gérer seront :

- l'édition de la table Braille;

- le choix des filtres à utiliser pour un contenu donné (abrégé ou intégral par exemple);
- la possibilité de choisir les contenus à traiter;
- permettre la configuration indépendante pour chaque filtre;
- maintenir un dictionnaire d'éléments à ignorer lors de la conversion;
- autoriser ou non les coupures (nombre caractères/lignes, coupure mathématique);
- offrir différents niveaux de détail pour le compte-rendu de transcription.

A ces paramètres s'ajoutent encore d'autres réglages très spécifiques ou techniques que nous ne présenterons pas à ce niveau de la modélisation.

3.3 Modélisation de la structure du format d'échange interne

Cette partie détaille la méthode suivie pour réaliser la modélisation du format de document interne. Après une modélisation générique d'un document puis la mise en évidences des différences entre un document noir et un document braille, nous examinons la possibilité d'utiliser des formats existants.

Nous terminons par la spécification de notre format interne.

3.3.1 Modélisation de documents noir et Braille

Le diagramme suivant propose une modélisation générique d'un document " en noir "quelconque.

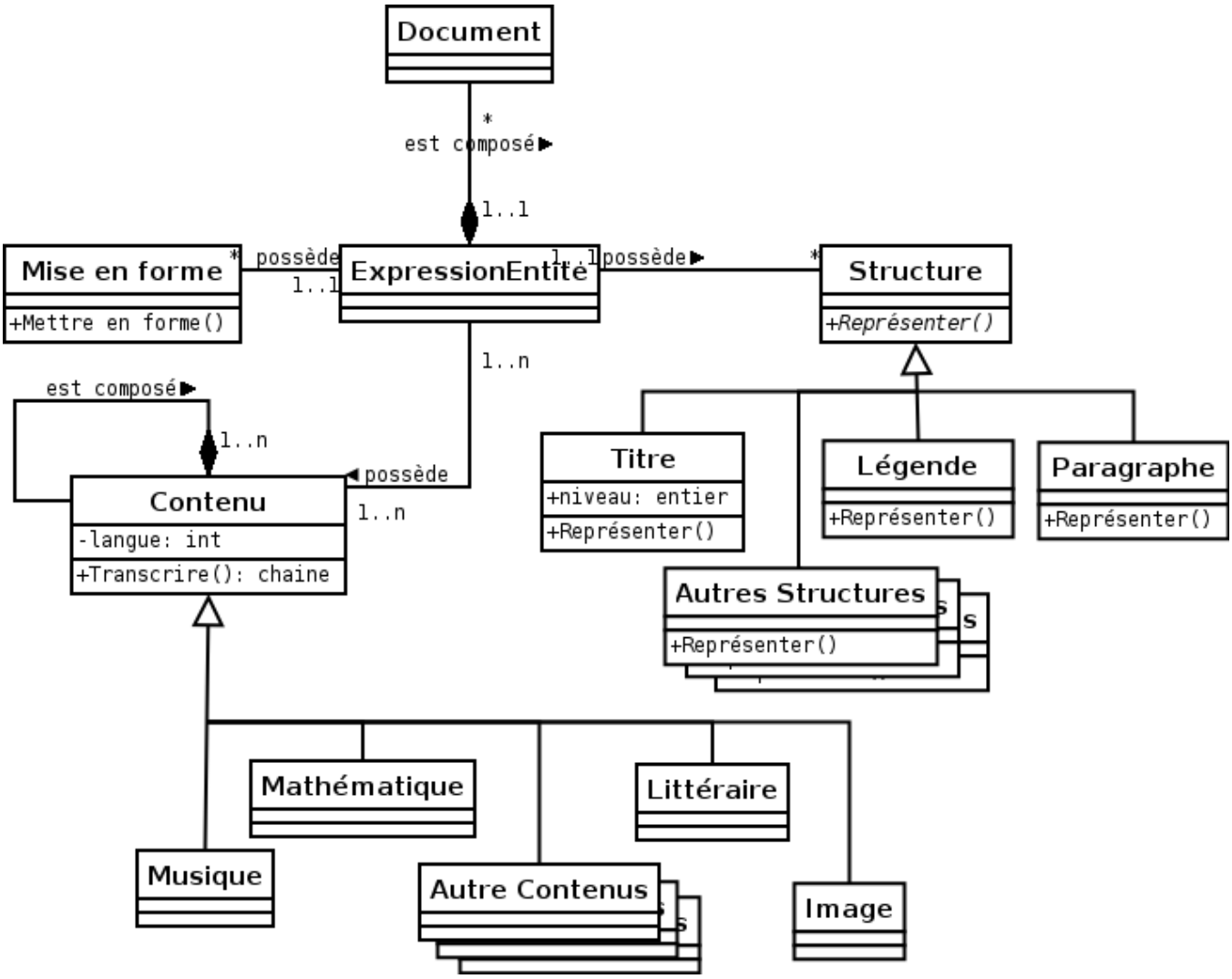
Bien que de nombreuses modélisations de documents existent déjà, nous nous sommes inspirés des différentes réflexions menées sur l'accessibilité informatique aux personnes handicapées pour établir notre modèle.

L'approche choisie lors de la modélisation s'appuie sur la séparation du contenu, de la forme et de la structure d'une expression. Si de nombreuses études prônent déjà la dissociation du contenu et de la forme, nous présentons un niveau supplémentaire d'organisation en distinguant – dans ce qu'on appelle " forme " – structure et présentation d'une expression.

La structure d'une expression lui donne son rôle dans le document : corps de texte, titre, note de bas de page, etc. La présentation d'une expression lui donne son aspect : police de caractère, alignement, etc. Ces deux aspects sont différents par nature : choisir de les traiter indépendamment dans un processus de transcription semble donc logique.

Ce choix nous offre également la possibilité de traiter et de paramétrer séparément chaque caractéristique d'une expression. L'utilisation de filtres XSL spécialisés permettra de réaliser ces opérations.

Modélisation d'un document " en noir "



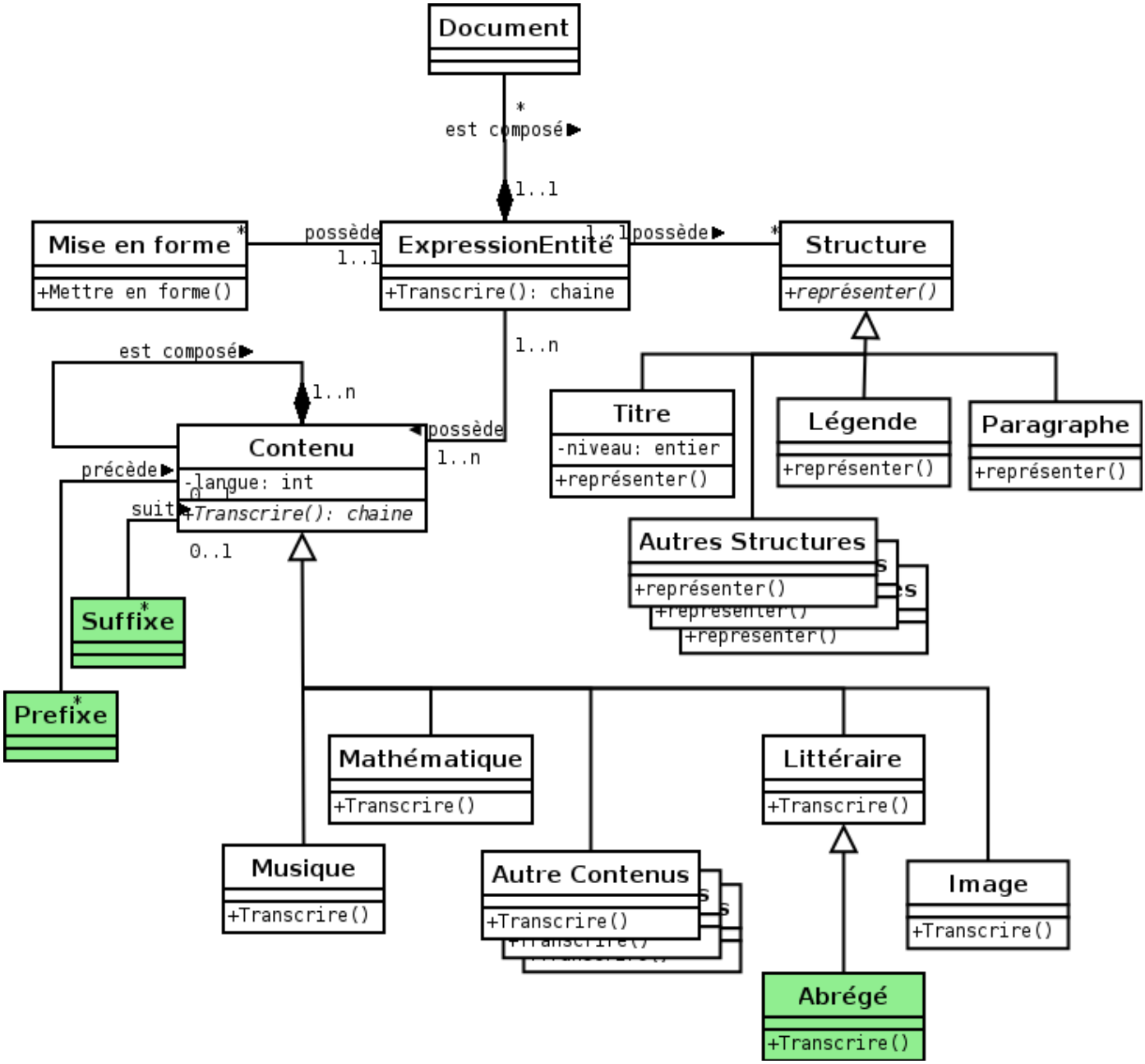
L'autre diagramme (" Modélisation d'un document Braille ") met en évidence les différences qui existent entre un document " en noir " et un document braille.

Contrairement à ce qu'on pourrait penser, le modèle braille obtenu est plus complexe que le modèle noir. Il ne faut pas perdre de vue :

- que notre point de vue de modélisateurs n'est pas objectif, car nous cherchons à mettre en place un transcripteur pour le braille;
- que nous ne modélisons que ce dont nous avons besoin;
- que le braille dérive de formes d'écritures plus anciennes que lui et peut donc être vue comme une amélioration de ces écritures (du point de vue du modèle);
- que le braille est essentiellement linéaire, contrairement à certaines écritures qu'il transcrit qui sont elles bidimensionnelles (mathématique, musique...) : il doit donc intégrer ces projections de la dimension 2 vers la dimension 1.

Ces deux modélisations vont nous servir de point de départ pour le choix d'un format interne compatible avec les deux modèles. La bonne spécification de ce format est cruciale car elle va conditionner les performances en termes de rapidité, de facilité d'implémentation et d'efficacité du logiciel.

Dans un premier temps, notre objectif concerne uniquement le contenu mais nous ne devons pas oublier pour autant les autres composantes d'une entité-expression qui seront pris en compte lors d'une version suivante du logiciel.



Modélisation d'un document en Braille

3.3.2 Réflexions sur les formats existants

Notre recherche de formats déjà existant s'est logiquement concentrée sur les formats XML définis par des DTD ou des schémas XML. Les traitements principaux se basant sur l'application de feuilles de style, autant utilise en entrée un document optimisé pour cette technologie.

Nous avons retenu 3 formats XML particulièrement intéressants :

- le format DTBook;
- le format Daisy;
- le format OpenOffice 1.

Malgré leur nombreuses fonctionnalités, ces formats ne répondent pas suffisement bien à nos attentes. Leur principal défaut est probablement l'utilisation de la phrase comme unité minimale.

Il nous semble préférable de spécifier nous-mêmes un format sur mesure en s'inspirant de nos modèles précédents plutôt que de chercher à rendre compatible un format complexe avec notre logiciel.

Néanmoins, l'utilisation de ces formats comme sorties possibles est à retenir.

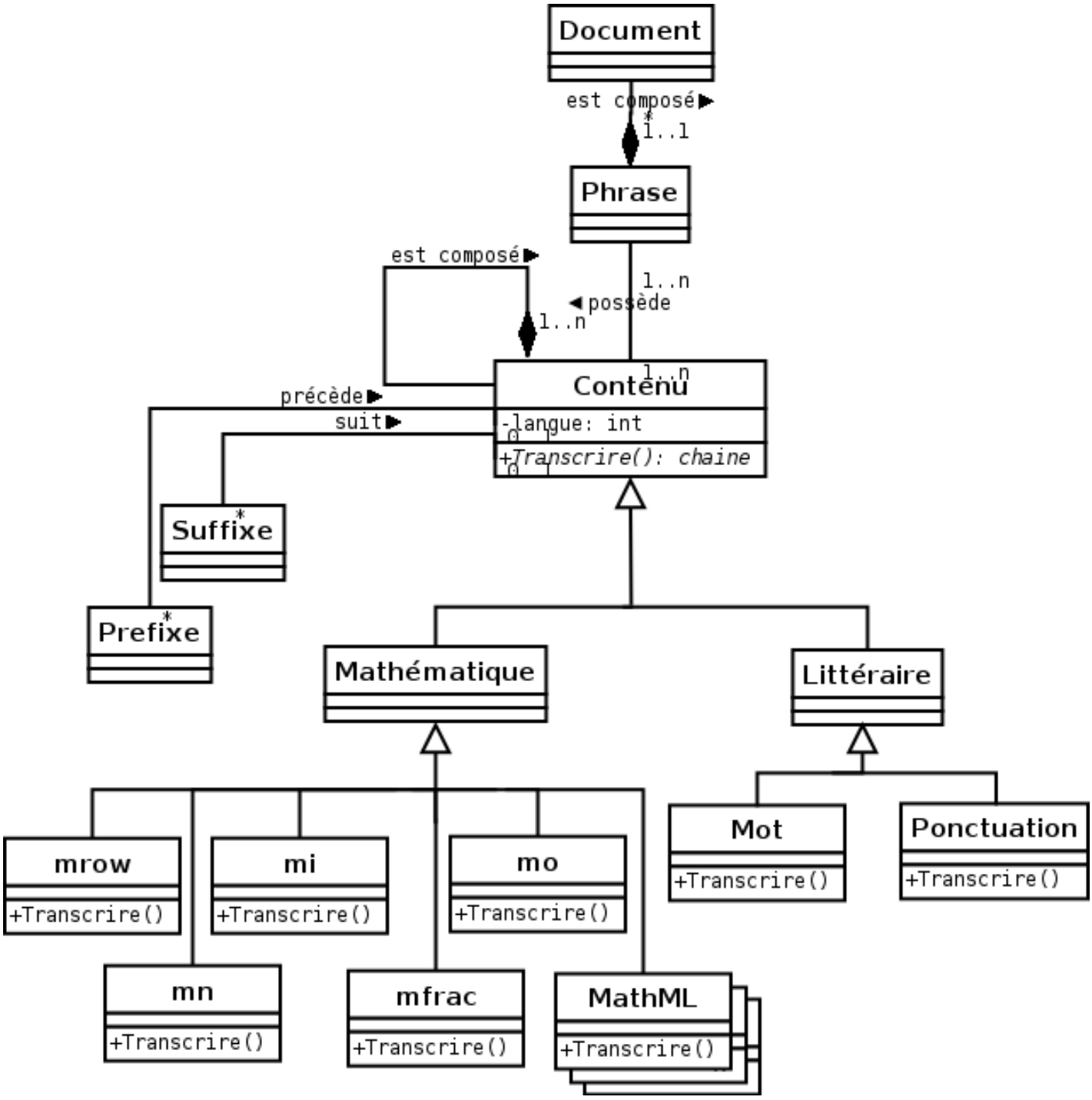
3.3.3 Proposition d'un format interne et justification des choix effectués

Le format interne retenu pour nos besoins actuels est modélisé sur le diagramme de classe suivant.

Nous nous sommes concentrés sur la spécification des contenus, littéraires et mathématiques :

- Nous utiliserons le standard MathML pour les expressions mathématiques;
- Nous utiliserons un format simple pour les expressions littéraires, de niveau suffisement fin pour permettre un traitement mot à mots. Nous distinguerons les signes de ponctuation des mots afin de faciliter l'algorithmie.

Les aspects mise en forme et structure pourront être modéliser ultérieurement, sans remettre en question l'organisation actuelle. Nous avons préféré nous concentrer dans un premier temps sur le contenu.

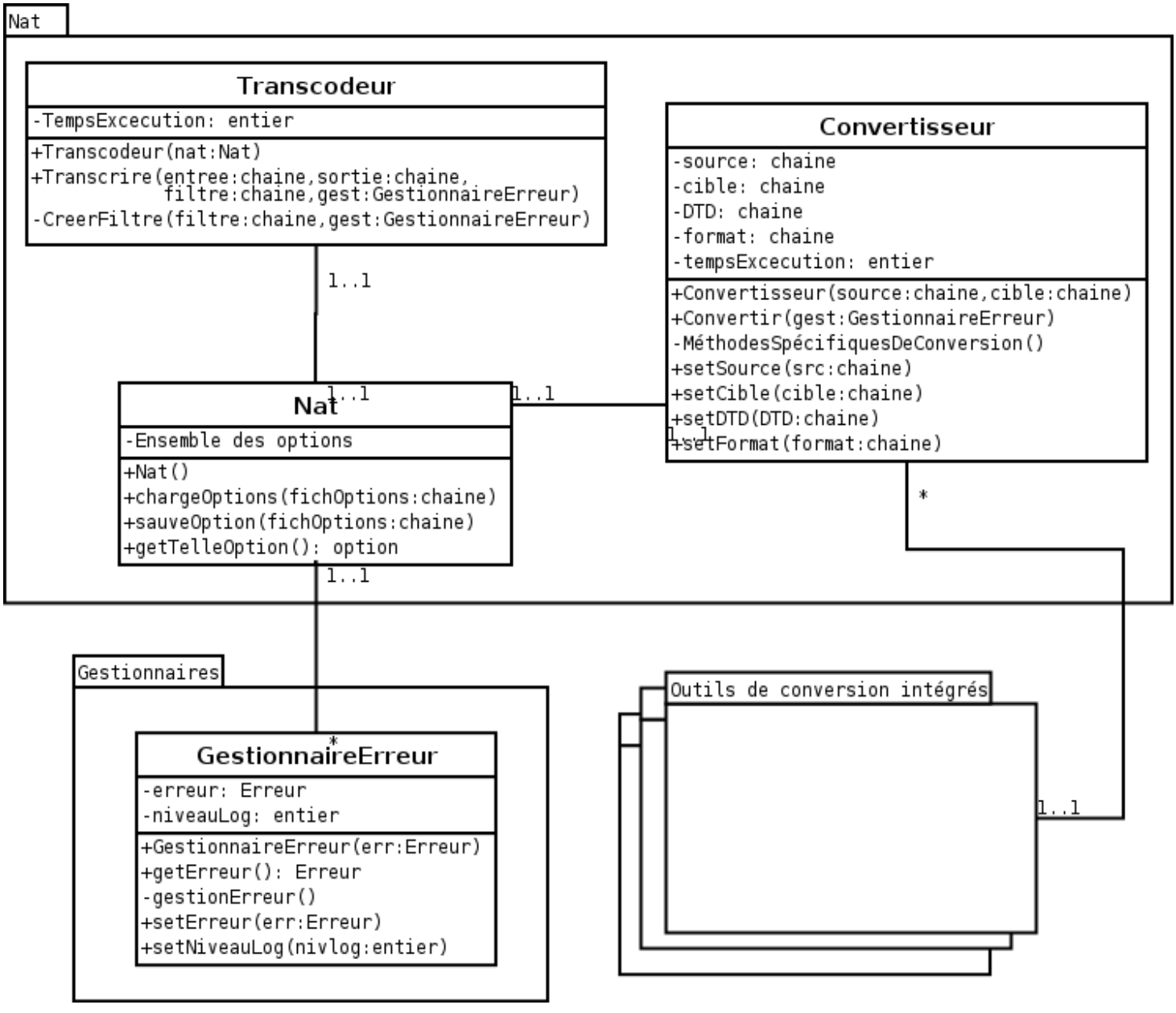


Modélisation du format de document interne.

3.4 Modélisation générale des fonctionnalités du système

Ce chapitre présente la modélisation des fonctionnalités du système qui seront par la suite programmés à l'aide d'un langage objet.

Cette modélisation découle directement de l'analyse des cas d'utilisation précédemment proposés. Elle a pour but d'organiser le développement et l'implémentation des fonctionnalités.



Modélisation des fonctionnalités du système

Le modèle est divisé en trois paquetages distincts :

- le paquetage Nat, qui contient la classe principale (Nat) et les deux classes de traitements (Convertisseur et Transcodeur);
- le paquetage Gestionnaires qui sera sollicité pour la gestion des différents types d'erreurs possibles;
- la collections de paquetages intégrés : ce sont les outils existants que nous utiliseront lors des conversions de documents et qui seront directement liés au convertisseur.

La classe principale Nat sera chargée du paramétrage, les options (attribut Ensemble des options) et leur méthode d'accès en lecture (getTelleOption()) n'ont cependant pas été représentées pour alléger la lecture.

Nous avons choisi une approche fonctionnelle pour notre modèle.

3.5 Modélisation de l'interface graphique du système

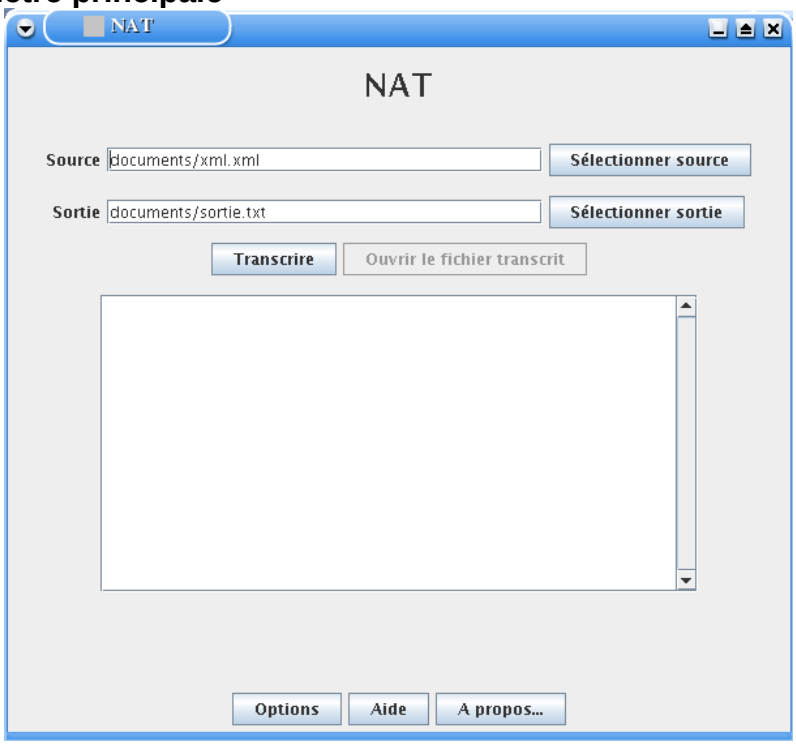
Cette partie propose une ébauche d'interface pour le système en conservant la notation UML pour les mêmes raisons que précédemment.

Après avoir présenté une maquette générale, nous l'organisons dans un diagramme de classe. Nous supposons que les composants graphiques standards (fenêtres, boutons, etc.) seront présents dans l'API du langage de programmation retenu.

3.5.1 Maquette de l'interface

Cette maquette a pour but de nous permettre d'identifier les composants à utiliser et de les organiser entre eux.

3.5.1.1 Fenêtre principale



La fenêtre principale doit permettre à l'utilisateur d'accéder à l'ensemble des fonctionnalités du logiciel, et interfacera le fonctionnement principal du logiciel.

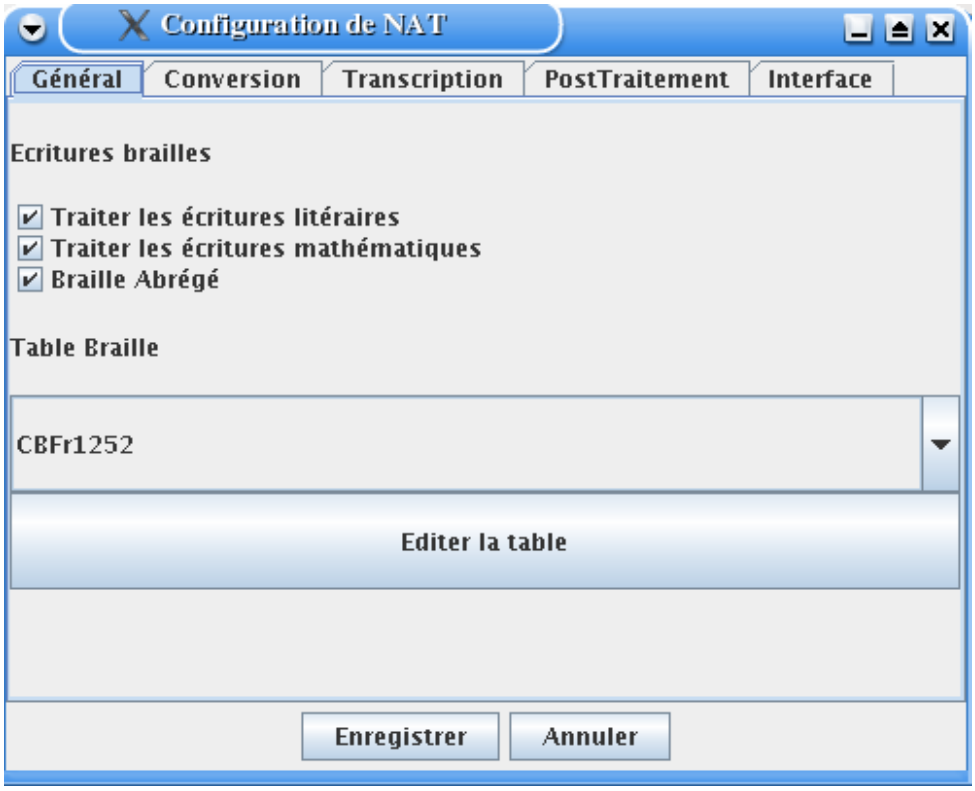
Elle proposera donc :

- une première zone destinée aux entrées (fichiers source/cible) et au lancement des traitements;
- une seconde zone destinée aux sorties (erreurs, état de la transcription, communication);
- une dernière zone permettant l'accès aux autres éléments de l'interface.

3.5.1.2 Fenêtre options

La fenêtre d'option regroupera l'ensemble des paramètres utilisés pour la transcription et la configuration du logiciel.

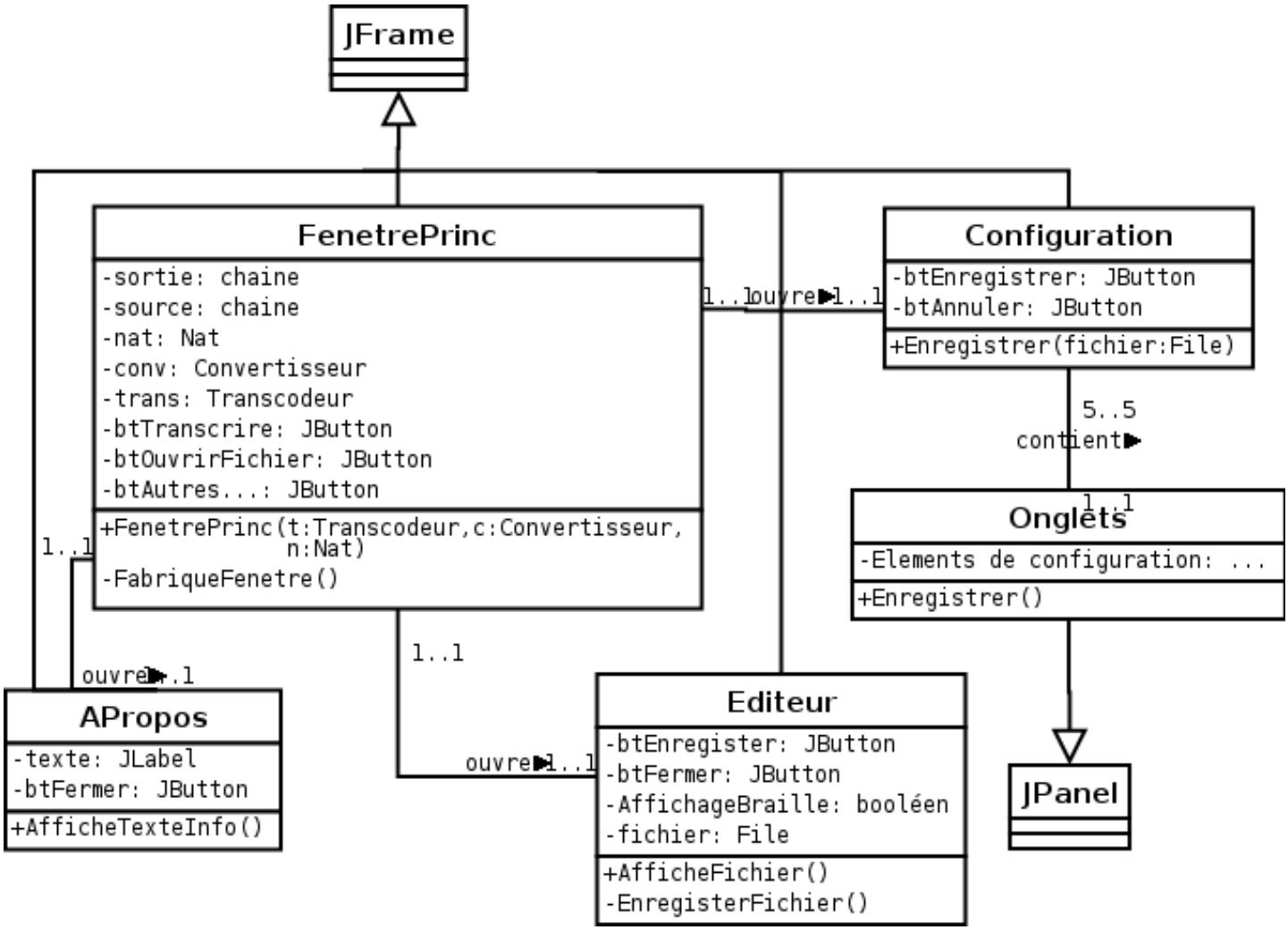
Nous les regrouperons thématiquement par onglets.



3.5.2 Modélisation de l'interface

Le diagramme de classe suivant présente l'organisation générale de l'interface graphique. Les composants standards utilisés sont ceux de l'API Swing de Java, mais pourraient tout aussi bien provenir d'API d'autres langages de programmation.

Les composants des onglets n'ont volontairement pas été détaillés afin de simplifier le modèles et laisser une plus grande liberté d'évolution aux développeurs..



3.6 Modélisation générale

Cette partie présente le modèle général et en justifie l'architecture.

3.6.1 Organisation du système en MVC

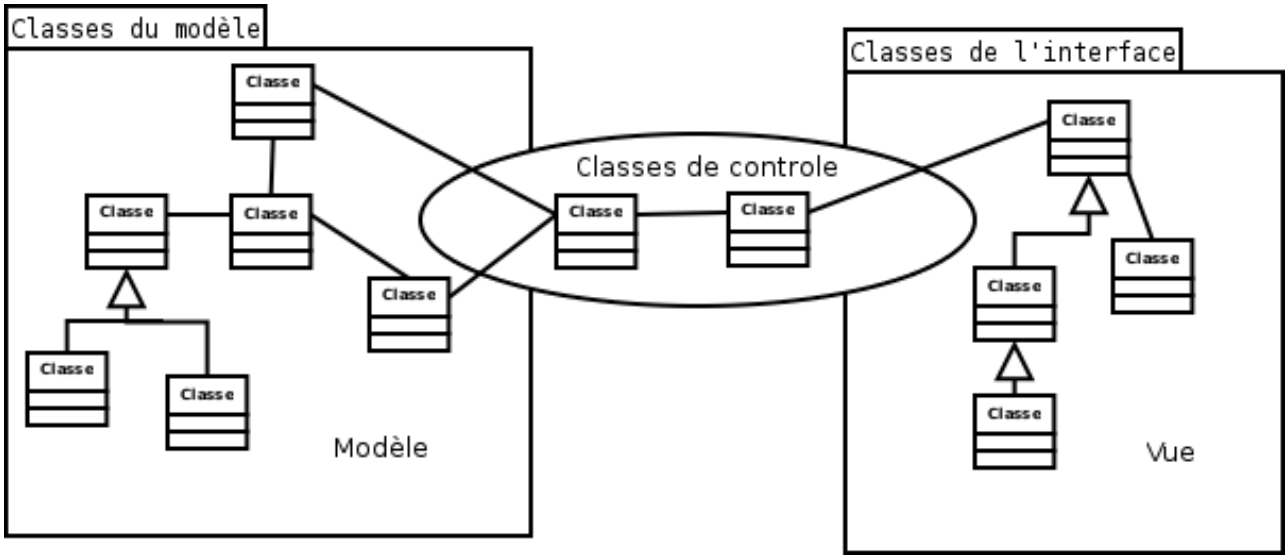
Grâce au développement des techniques de programmation et de modélisation, l'axiome " séparer le contenu de la manière de le présenter " s'impose de plus en plus en tant que " Best practice " que l'on pourrait traduire en français par " recommandation pour obtenir un programme bien réalisé ".

De ce concept est né l'idée de l'architecture MVC (Modèle, Vue, Contrôleur). Cette architecture permet de séparer les fonctionnalités du système de leur interfaçage, la communication entre les deux ensembles étant assurée par les contrôleurs.

Dans notre cas, nous devrons probablement adapter notre interface à différents type de public : il semble d'autant plus judicieux d'utiliser une telle architecture qui nous permettra pour un même ensemble de fonctionnalités de proposer plusieurs interfaces.

L'autre intérêt majeur de ce type d'organisation est la réutilisabilité des différents composants de manière indépendante : en cas d'intégration de notre logiciel, seuls les classes utiles seront récupérées. De plus, en cas de modifications fonctionnelle ou graphique, seul le package concerné sera modifié sans avoir à toucher aux autres.

Nous incitons le lecteur à se remémorer les critères d'accessibilité du W3C : il s'agit bien plus que d'une simple analogie...



L'architecture MVC

3.6.2 Modèle général

Le modèle général est disponible en annexe (Annexe A).

Il intègre les modèles précédemment proposés ainsi que les classes de contrôle nécessaires au bon fonctionnement du système.

3.7 Implémentation

Cette partie présente les choix techniques retenus pour l'instanciation de notre modèle et de ses environnements de développement et d'exécution.

3.7.1 Choix des environnements

3.7.1.1 Environnements de développement

Les développements seront effectués sous Linux (Debian Sarge [WEB 16]) à l'aide d'Eclipse [WEB 14] pour la programmation Java et de l'éditeur texte Kate [WEB 15].

Une partie du développement des feuilles de style a néanmoins été réalisé sous Microsoft XP à l'aide de l'éditeur texte NotePad.

3.7.1.2 Environnements d'exécution

L'objectif de notre logiciel est de servir au plus grand nombre sans trop contraindre les utilisateurs à adopter une configuration logicielle spécifique.

L'utilisation de technologies ouvertes et libres ainsi que de Java devrait permettre à n'importe quel système d'exploitation doté d'une machine virtuelle Java récente de faire fonctionner notre logiciel.

3.7.2 Choix des outils de programmation

L'étude préalable nous incite fortement à utiliser une solution XML-XSL interfacée par le langage de programmation JAVA.

Nous avons donc choisi les outils suivants :

- programme principal : Java
- transcription : appel à des feuilles XSL grace aux API DOM et SAX de Java

- interface graphique : API Swing de Java

Chapitre 4 : Développements

Ce chapitre présente les développements réalisés et les travaux associés.

4.1 Le module de transcription

4.1.1 Définition de la DTD

Bien que le XML Schema offre plus de fonctionnalités que la DTD, nous avons préféré dans un premier temps nous contenter d'une structuration par DTD, plus simple à mettre en place et surtout mieux interprétée par les différentes API.

Nous nous sommes focalisés sur l'aspect contenu, reléguant la définition des deux autres aspects d'une expression aux prochaines versions du logiciel.

```
<!-- DTD pour NAT -->
<!-- pour les conversions venant de xhtml -->
<!ENTITY % laDtdXHTML SYSTEM "xhtml11.dtd" >
%laDtdXHTML;
<!-- pour le contenu mathÃ©matique -->
<!ENTITY % laDtdMath SYSTEM "mathml.dtd" >
%laDtdMath;
<!-- pour le contenu litÃ©raire -->
<!ELEMENT mot (#PCDATA)>
<!ELEMENT ponctuation (#PCDATA)>
<!ELEMENT lit ((mot | ponctuation)*)>
<!-- Element conteneurs de contenus -->
<!ELEMENT phrase ((math | lit)*)>
<!-- Element de base -->
<!ELEMENT doc (phrase*)>
```

Bien que courte en apparence, cette DTD inclut les DTD XHTML1.1 et MATHM, soit plus de 3000 lignes de définitions...

4.1.2 Intégration de la feuille de style mathématique de BraMaNet

Pour la gestion des mathématiques, nous avons repris et amélioré la feuille de style BraMaTxt.xsl du logiciel libre BraMaNet.

Cette feuille de style, parfaite pour le MathMl généré pas MathType, présentait un grand nombre de lacunes pour s'adapter à la norme officielle MathMl.

Plutôt que de nous enfermer strictement dans la norme, nous avons préféré adapter cette feuille de style pour qu'elle reste compatible avec les différents MathMl existants.

Nous l'avons également augmentée d'expressions et de symboles non-traités jusqu'alors.

4.1.3 Création des feuilles de style intégral et abrégé pour le littéraire

Nous avons commencé par implémenter la feuille de style pour le braille intégral. Sa programmation quasi-trivial n'a pas posé beaucoup de problèmes techniques et nous a permis de régler rapidement les soucis de compatibilité de la feuille mathématique avec un autre filtre.

En revanche, la conception de la feuille de style pour l'abrégé s'est révélée beaucoup plus complexe. En effet, nous n'avons pas souhaité traiter l'abrégé par un immense dictionnaire, ce qui aurait été plus facile mais beaucoup moins évolutif et performant.

Après l'analyse des règles d'abréviation, nous les avons transcrites en XSL et ordonnées logiquement grace à l'algorithme de la page suivante. Les zones grisées correspondent à un contenu à analyser ou a une modification de ce contenu et les points cerclés à un état final (la transcription a été trouvée).

L'abrégé comprend deux ensembles de règles : les règles d'exception et les règles du cas général. Le temps imparti par le calendrier ne nous permettant pas de traiter l'ensemble des cas possibles, nous avons commencé par les règles d'exception, puisque ce sont elles qui s'appliquent en premier.

Les numéros utilisés dans l'algorithme correspondent à la numérotation des règles utilisée dans le document officiel de l'A.V.H. " Le braille abrégé " que l'on peut se procurer sur le site internet de l'association (http://www.avh.asso.fr) ou en écrivant au service commercial.

Une fois ces règles implémentées, nous avons optimisé les traitements en étudiant les fréquences d'apparition des mots-exceptions dans la langue française. Le classement alphabétique a donc été remplacé par un classement fréquentielle.

Malgré nos efforts, certaines ambiguïtés subsistent, qui ne pourront être levées que par une analyse étymologique du mot. Une partie de cette analyse est réalisable simplement par informatique, mais l'analyse exhaustive est elle très complexe à implémenter.

Les prochaines versions du logiciel devront être capables de détecter une ambiguïté et demanderont dans ce cas à l'utilisateur de choisir la bonne transcription parmi plusieurs propositions.

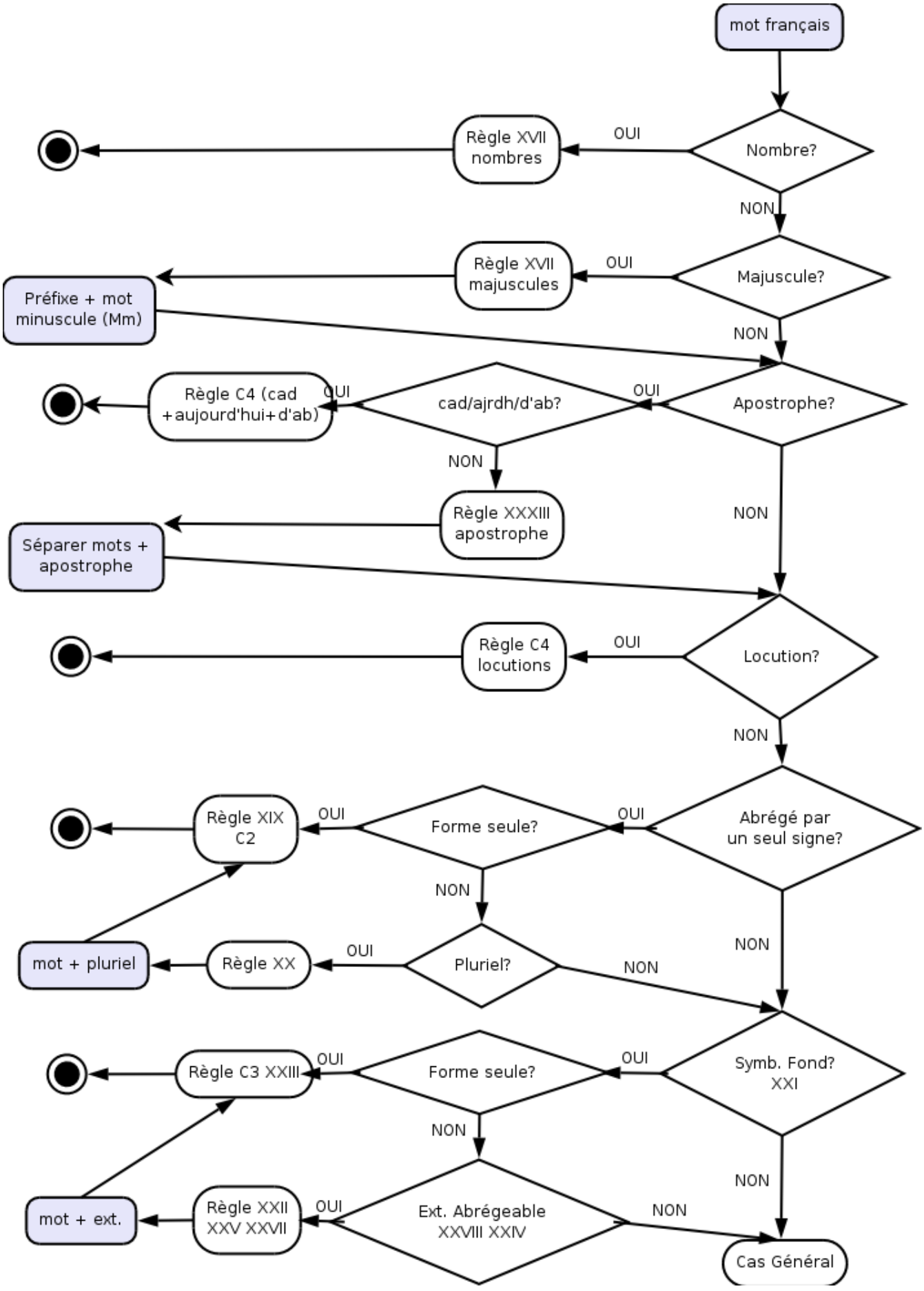
La même méthode de travail devra être appliquée pour la réalisation du cas général.

4.1.4 Implémentation du module avec Java

L'utilisation des API SAX et DOM, bien que théoriquement triviales, s'est révélées un peu plus ardue que prévue. En effet, dans notre soucis de compatibilité inter-plateforme, nous avons dû implémenter exhaustivement les interfaces, travail fastidieux et complexe.

Le résultat obtenu est très satisfaisant, comme l'on montré les premiers tests unitaires.

Afin de maîtriser l'ensemble du processus, nous avons identifié chaque erreur possible (Exception de Java) et la traitons de manière appropriée. L'utilisateur disposera donc d'un outil performant et compréhensible de gestion d'erreurs, et pourra suivre le fil des traitements très simplement.


mot français
Nombre?
OUI
Règle XVII nombres
NON
Majuscule?
OUI
Règle XVII majuscules
Préfixe + mot minuscule (Mm)
NON
Apostrophe?
OUI
cad/ajrdh/d'ab?
OUI
Règle C4 (cad +aujourd'hui+d'ab)
NON
Règle XXXIII apostrophe
Séparer mots + apostrophe
NON
Locution?
OUI
Règle C4 locutions
NON
Abrégé par un seul signe?
OUI
Forme seule?
OUI
Règle XIX C2
NON
Pluriel?
OUI
Règle XX
mot + pluriel
NON
Symb. Fond? XXI
OUI
Forme seule?
OUI
Règle C3 XXIII
NON
Ext. Abrégeable XXVIII XXIV
OUI
Règle XXII XXV XXVII
mot + ext.
NON
Cas Général

4. 2 Le module de conversion

4.2.1 Conversion d'un fichier texte

Le format le plus simple qui puisse exister est le format texte. Il est fréquemment utilisé par les non-voyants (mels , sortie de scanner à reconnaissance de forme, etc.) et c'est donc logiquement celui-ci que nous avons traité en premier.

La conversion au format interne a été réalisée uniquement en Java, le format d'entrée restant simple à appréhender.

Afin de conserver la compatibilité avec les fichiers exportés par MathType, nous avons également intégré une fonction de reconnaissance du MathMl. Un fichier préparé pour BraMaNet reste donc utilisable avec NAT.

4.2.2 Conversion d'un fichier open-office

La conversion open-office est bien plus difficile à réaliser. Nous avons choisi d'intégrer un logiciel libre, Writer2XHTML, permettant d'exporter proprement un fichier open-office en XHTML.

Malheureusement, la définition des entités mathématiques laissant fortement à désirer, nous avons donc reprogrammé l'ensemble du module chargé de gérer les formules open-office.

Une fois ce travail réalisé, nous obtenions un fichier XHTML correct incluant correctement des expressions en MathMl. Afin de le transformer en document interne, nous avons utilisé de nouveau un filtre XSL dédié.

4.3 Les interfaces homme-machine

4.3.1 Le mode console

Nous avons voulu permettre à un utilisateur d'utiliser NAT uniquement en mode console. Les intérêts de cette idée sont multiples :

- un utilisateur peut préférer le mode console pour diverses raisons (système d'exploitation en mode texte, connexion à un serveur par protocole texte comme ssh ou telnet, etc.);
- le logiciel pourra être exécuté par un autre programme, ce qui facilite son intégration dans d'autres outils;
- le logiciel devient compatible Linux, et pourra être incorporé dans les distributions de ce système d'exploitation libre.

Cependant, le mode console ne permet pas un paramétrage facile, puisqu'il faudra modifier directement les fichiers de configuration sans passer par une interface. Une évolution possible du logiciel serait la mise en place d'un outil de configuration textuel.

4.3.2 Le mode graphique

Le mode graphique offre plus de fonctionnalités que le mode console, notamment en ce qui concerne le paramétrage du logiciel.

Il a été réalisé à l'aide du paquetage SWING de Java, contenant des composants légers et accessibles pour les lecteurs d'écrans. Nous avons également implémenté l'interface d'accessibilité proposée par Java.

Les résultats de l'étude ergonomique étant arrivés trop tard (voir 4.5), un certain nombre d'éléments laissent encore à désirer, même si l'interface permet l'utilisation de toutes les fonctionnalités principales du logiciel.

Les copies d'écran des différentes fenêtres sont proposées en annexe.

4.4 Exécution des tests unitaires

Nous avons effectué deux types de tests unitaires :

- des tests fonctionnels, destinés à vérifier le bon comportement du logiciel en fonction des types de documents utilisés, des systèmes d'exploitation et des configurations logicielles diverses;
- des tests qualitatifs permettant de valider les documents transcrits.

Les tests fonctionnels ont été réalisés durant deux jours auprès d'utilisateurs expérimentés à la mission handicap de Lyon 1. La mission handicap de Lyon 1 dispose d'une grande variété d'environnements (systèmes d'exploitation, postes adaptés pour des non-voyants, matériels spécifiques...).

Le logiciel s'est chaque fois comporté de la manière souhaitée.

Les tests qualitatifs ont été réalisés à l'aide de spécialistes du braille abrégé et mathématiques. Ils nous ont permis de corriger quelques coquilles et une erreur importante dans la coupure des expressions mathématiques.

Les jeux de tests ont été réalisés à partir de jeux existants ayant servis lors de précédents tests de BraMaNet et de la mise en place du fichier d'abréviation du logiciel WinBraille par Françoise Perdoux (CNEFEI).

4.5 Étude ergonomique de l'interface

Nous avons demandé à l’équipe d’ergonomes qui participent à notre projet (Hélène DESLANDES, Vivian DUMOND et LAUTRETTE Jean-Pascal) de réaliser une étude ergonomique de l'interface.

Malheureusement, nos différents calendriers ne nous ont pas permis de prendre en compte l'ensemble des recommandations proposées, les premiers résultats nous ayant été communiqués deux jours avant la date de rendu des travaux.

Nous présentons néanmoins le travail réalisé en annexe afin de l'intégrer dans une prochaine version de NAT.

Une des difficultés d'intégration pourrait provenir du manque d'expertise des ergonomes dans le domaine spécifique du handicap, bon nombre de recommandations ne s'appliquant pas à la transcription mais plutôt aux contenus web.