Modélisation orientée-objets d'une grammaire de type Gouvernement et Liage dans le cadre d'un système de traduction multilingue

Cathy Berthouzoz
LATL/Université de Genève
berthouzoz@latl.unige.ch

mai 1996

1 Introduction

Ce travail expose une modélisation de la grammaire universelle, décrite dans la théorie Gouvernement et Liage, à l'aide d'une approche orientée-objets. Il a un double but : pratique, il s'espère résoudre des problèmes de génie logiciel rencontrés lors du développement du système de traduction ITS-2, théorique, il veut démontrer un certain type de modularité de ce genre de grammaire.

Les problèmes de génie logiciel - maintenance difficile et redondance du code - sont liés au fait que le développement d'un système de traduction multilingue implique l'écriture de programmes volumineux[1], la création et/ou l'adaptation de gigantesques bases de données lexicales[2]. Ce processus est incrémental : quand tous les composants sont prêts pour un couple de langues donné (analyseurs, traducteurs, lexiques), ils sont incorporés au système initial. Le développement du système est aussi incrémental à l'intérieur de chaque langue (en analyse) ou couple de langues (en traduction) : on commence avec les constructions de base, l'incorporation de constructions grammaticales plus compliquées se faisant par étapes. Avec un langage de programmation traditionnel[3], rajouter une langue ou une construction grammaticale implique l'adaptation plus ou moins lourde des programmes existants[4]. Le problème de maintenance est encore aggravé par la grande redondance du code. En effet, certaines fonctionnalités sont répliquées pour chaque langue, alors qu'elles ne diffèrent que par de légères nuances[5]. Nous voulons montrer dans ce travail qu'une approche orientée-objets - grâce avant tout à l'extensibilité et la réutilisabilité des objets - permet de résoudre les problèmes pratiques de maintenance et de redondance du système de traduction.

Dans une approche orientée-objets, l'accent n'est pas mis sur les fonctionnalités du système, mais sur les données traitées. Dans un système de traduction, les données sont le texte à traduire, les lexiques et surtout la grammaire. Celle-ci doit être réaliste, tant du point de vue de la couverture grammaticale - qui doit être la plus large possible - que du point de vue psycholinguistique, d'où le choix d'une grammaire générative de type *Principes et Paramètres*, plus spécifiquement la grammaire universelle (GU

[1] FIPS, l'analyseur du français en cours de développement au LATL, comporte environ 8'200 lignes, IPS, l'analyseur de l'anglais 5'500 lignes (non pas parce que l'anglais est moins difficile à analyser, mais parce que l'analyseur est moins développé). Pour ce couple de langues, ITS, le traducteur anglais-français, totalise 9'200 lignes et FITS, le traducteur français-anglais 11'500 lignes. Même si des commentaires occupent le tiers des lignes, il reste encore plus de 20'000 lignes à gérer et maintenir.

[2] Par exemple, le lexique français comporte 165'000 mots et 20'000 lexèmes, (9,1 Mo), le lexique anglais 81'000 mots et 46'000 lexèmes, (6,7 Mo), le lexique bilingue 22'000 correspondances (2,3 Mo).

[3] Le système ITS-2 est implémenté en Modula-2, un langage procédural modulaire développé par Wirth (1985).

[4] Si, comme dans le système ITS-2, les grammaires sont codées dans les algorithmes, le problème est encore plus complexe car il devient difficile de différencier les instructions spécifiques à la grammaire de celles des processus d'analyse ou de traduction.

[5] Un exemple flagrant est la gestion des lexiques : les procédures d'accès ou de mise à jour des entrées lexicales sont répliquées pour chaque langue alors que le format des entrées est équivalent, seuls les traits spécifiques à certaines catégories de mots étant différents.

dans la suite du texte) décrite par la théorie *Gouvernement et Liage* (Government and Binding, GB dans la suite du texte) (Chomsky 1981, 1986a, 1986b). En principe, une telle grammaire est universelle[6], paramétrisable selon les spécificités de chaque langue, et modulaire. La grammaire générative a été développée dans le but d'expliquer la production du langage - correspondant au processus de génération en traitement automatique du langage naturel (TALN) - en aucun cas dans le but d'expliquer la compréhension - l'analyse en TALN. Mais une grammaire de type principes et paramètres semble bien adaptée au développement d'analyseurs syntaxiques modulaires[7], plus efficaces que leurs correspondants non modulaires, comme le montrent entre autres Wehrli (1988) et Wehrli & Laenzlinger (1993).

La modularité du point de vue linguistique signifie que les principes sont organisés en sous-systèmes spécialisés, tels que par exemple le module thématique (assignation des rôles thématiques aux syntagmes nominaux) et le module du cas (assignation des cas aux syntagmes nominaux). Mais elle ne rejoint pas complètement la modularité du point de vue informatique, qui signifie en plus de la spécialisation des modules, dissimulation de l'information et interface minimale entre les modules. Or, les modules de la GU interagissent fortement entre eux - par exemple, par le Principe de Projection, l'information lexicale est présente à tous les niveaux de représentation - ce qui signifie qu'une correspondance 1:1 entre modules linguistiques et modules informatiques implique le passage de lourdes structures de données à travers les modules. L'analyseur résultant ne peut plus être considéré comme modulaire. Différentes études (Berwick et al. 1991, Merlo 1992, Walther 1994 entre autres) ont en outre montré que la GU ne peut pas être modulaire dans le sens informatique du terme, même si elle l'est conceptuellement.

Nous allons montrer qu'avec une approche orientée-objets, grâce à la notion d'héritage et de polymorphisme, il est possible de remodulariser la GU, non pas selon ses sous-systèmes de principes, mais selon les objets linguistiques sur lesquels s'appliquent les principes : catégories grammaticales et syntaxiques et chaînes. Nous rejoignons avec cette approche la conclusion à laquelle est arrivée Merlo (1995), à savoir : " La modularité, si elle existe, doit être cherchée dans le contenu linguistique, et non pas dans l'organisation de la théorie. "

L'approche orientée-objets permet non seulement de concevoir une grammaire modulaire, mais encore universelle, grâce au parallèle que l'on peut établir entre principes et paramètres de la théorie GB et message et polymorphisme du paradigme objet. En effet, quelle que soit la langue, les mêmes messages sont envoyés aux objets (principes) qui y répondent selon leur spécificité (paramètres). Une grammaire universelle permet de concevoir un analyseur et un générateur universels, ce qui réduit considérablement le nombre de programmes du système de traduction multilingue - nombre qui ne varie plus selon le nombre de langues incorporées - facilitant ainsi le

[6]Le terme universel ne doit pas être pris dans son sens le plus fort - applicable à toutes les langues existantes - mais dans le sens nuancé d'applicable à un sous-ensemble de langues partageant les mêmes propriétés, telles que les langues indo-européennes par exemple.

[7]Un analyseur est modulaire s'il y a une correspondance 1:1 entre les modules de la grammaire et les modules de l'analyseur.

développement incrémental du système et sa maintenance, et réduisant la redondance du code.

La suite du document est structurée de la façon suivante. La section 2 décrit le paradigme objet, la section 3 expose dans les grandes lignes la GU (3.1), ainsi que le parallèle entre la théorie GB et le paradigme objet (3.2). La section 4 concerne la description structurale du système de traduction. Le système lui-même est exposé brièvement à la section 4.1 pour établir le cadre général, les lexiques sont décrits à la section 4.2, car les informations lexicales sont primordiales dans la théorie GB. La grammaire universelle est détaillée à la section 4.3. Deux exemples d'analyse sont brièvement décrits à la section 5, pour montrer l'adéquation de cette modélisation. Enfin, la conclusion est présentée à la section 6.

2 Le paradigme objet

Le paradigme objet a émergé en réponse à la crise du logiciel du début des années 80, caractérisée par les problèmes de réalisation, de maintenance, voire même de conception des larges systèmes informatiques, problèmes difficilement solubles avec les techniques traditionnelles de développement de logiciel.

L'orientation-objet s'est développée tout au long des années 80 en théorie des bases de données, en génie logiciel ou en programmation pour être de plus en plus utilisée dans tous les domaines de l'informatique, y compris en linguistique informatique. Elle se caractérise par la modularité, l'extensibilité et la réutilisabilité des composants informatiques.

2.1 Analyse et conception orientées-objets

La modélisation sous forme d'objets permet de définir clairement le vocabulaire et de conceptualiser facilement le domaine d'application du problème à résoudre. Le modèle étant abstrait du monde réel, il forme une base plus solide pour la suite du développement que celui de l'approche traditionnelle, où il est en fait un point de vue du concepteur du système. Plusieurs méthodes d'analyse et de conception orientées-objets ont été proposées par Booch (1991), Coad & Yourdon (1990, 1991), Rumbaugh et al. (1991) et Schlaer & Mellor (1988). Mises à part certaines différences notationnelles, elles comprennent toutes :

1. une **spécification structurelle** qui décrit, à l'aide d'un ou plusieurs diagramme(s), les objets du domaine, leurs attributs et les relations entre les objets;

2. une **spécification comportementale** qui décrit, à l'aide de réseaux de transition ou de diagrammes d'événements, le comportement des objets (changement d'état) à travers leur cycle de vie.

Des méthodes d'abstraction des objets du monde réel ont été proposées, entre autre la méthode textuelle d'Abbot (1983), qui préconise de décrire

le système à l'aide du langage naturel et de créer un objet pour chaque nom, les attributs étant constitués par les adjectifs, et les interactions par les verbes. Mais cette opération semble plus être une affaire de doigté et d'expérience que de méthodologie. Il y a aussi la méthode de " conception par prise de responsabilités " (Wirfs-Brock 1989) qui définit pour chaque classe du système ses responsabilités (ce qu'elle doit faire) et ses collaborateurs (classes qui lui offrent des services). En partant des classes les plus évidentes, ou connues, on établit des scénarios (que se passe-t-il si ...) qui permettent de dériver les autres classes. Cette méthode est plutôt utilisée dans le monde de la programmation, les autres méthodes l'étant dans le monde des bases de données.

Comme le but de la modélisation est de fournir une description assez détaillée pour qu'elle puisse être facilement implémentée, peu importent la méthode et le formalisme utilisés, pourvu que le résultat soit compréhensible et bien documenté. Les méthodes proposées ne sont pas les seules, mais elles servent de point de départ à sa méthode personnelle.

2.2 Propriétés des objets

L'état du système est distribué entre les objets et chaque objet gère son propre état (ses attributs), qu'il dissimule aux autres. Les opérations permettent de s'informer sur l'état de l'objet ou de le modifier. Un changement significatif dans l'état d'un objet est appelé événement. Les objets communiquent entre eux à l'aide de messages (opération, responsabilité) plutôt qu'à l'aide de variables partagées, ce qui garantit une plus grande *sécurité* du système. Un objet répond à un message à l'aide de la méthode (spécification de l'opération) appropriée, qu'il propose dans son interface. Chaque objet est une entité indépendante, dont la représentation interne peut être modifiée sans faire référence aux autres objets, ce qui en fait une brique de base *réutilisable* selon les besoins. Des classes (types d'objets) regroupent les objets qui possèdent les mêmes caractéristiques. Une classe peut être étendue par de nouveaux attributs et de nouvelles méthodes, les attributs et méthodes de la classe de base, ou super-classe, étant hérités par la sous-classe. Le polymorphisme (plusieurs objets ont des méthodes qui ont le même nom, mais pas forcement la même fonctionnalité) permet de faire réagir différemment des objets en réponse au même message.

Bien que le concept de *modularité* soit à la base des objets, englobant encapsulation des données et des opérations, interactions minimales, spécialisation, réutilisabilité, les propriétés spécifiques aux objets, classes, héritage, messages, polymorphisme, leur donnent une plus grande puissance que les modules.

2.3 Programmation orientée-objets

Bien que l'analyse et la conception soient indépendantes de l'implémentation, un logiciel modélisé à l'aide d'une approche orientée-objets peut être implémenté à l'aide d'un langage de programmation traditionnel, il est pourtant plus simple d'utiliser un langage de programmation orienté-objets. Un

Terme orienté-objet	Terme traditionnel
Classe	Type de données abstrait extensible
Objet	Variable de ce type (Instance d'une classe)
Message	Appel de procédure (liée dynamiquement)
Méthode	Procédure liée à ce type

Table 1: Terminologie orientée-objets

tel langage doit posséder les caractéristiques suivantes (Mössenböck 1993) :

1. **Dissimulation de l'information** (*information hiding*) : l'implémentation de données complexes est encapsulée dans les objets. Les clients n'ont qu'une vue abstraite des objets et ne peuvent accéder aux données que par les méthodes mises à disposition par l'objet. On parle aussi d'encapsulation des données.

2. **Types de données abstraits** (*data abstraction*) : c'est une unité consistant en une structure de données et les opérations qui y sont applicables. On peut avoir plusieurs instances de ce type.

3. **Héritage** (*inheritance*) : un type de donnée abstrait peut être étendu dans un nouveau type qui hérite de toutes les données et procédures du type de base. Le nouveau type peut inclure de nouvelles données et opérations et peut même modifier des opérations héritées. Une propriété importante est la compatibilité entre le nouveau type et le type original.

4. **Liage dynamique** (*dynamic binding*) des messages (requêtes) aux méthodes : ce n'est que lors de l'exécution que le message est interprété pour exécuter la méthode de l'objet correspondant au type dynamique. Une conséquence en est le polymorphisme des variables, qui peuvent contenir un objet du type de base ou d'une de ses extensions.

Les deux premières caractéristiques ne sont pas uniquement liées aux langages orientés-objets, elles se retrouvent aussi dans les langages dits modulaires, comme Modula-2 et Ada.

Un type de données abstrait extensible qui accepte des messages liés dynamiquement est appelé une classe, qui forme la brique de base de la programmation orientée-objets.

La terminologie de la programmation orientée-objets diffère de celle de la programmation traditionnelle, même si elles peuvent désigner la même chose. La table 1 ci-dessous propose quelques termes très souvent utilisés en OOP et leur correspondants en programmation traditionnelle.

On peut dire que la programmation orientée-objets est l'évolution logique de la programmation modulaire, comme le montre parfaitement Wirth (1992), bien qu'il refuse cette terminologie au profit de programmation extensible.

Tout le monde semble d'accord sur les bienfaits de la programmation orientée-objets, extensibilité et réutilisabilité des classes existantes, lorsqu'elles

sont utilisées à bon escient. Il faut toutefois noter qu'elle comporte aussi certains inconvénients, tels efforts d'apprentissage et de compréhension des notions nouvelles et des classes existantes, et des problèmes d'efficacité, tant à l'exécution que d'utilisation de la mémoire, si les classes et les messages sont utilisés à tort et à travers (Mössenböck 1993).

2.4 Orientation-objets en TALN

Même si les systèmes de traitement automatique du langage naturel sont encore souvent développés à l'aide de langages non orientés-objets - les langages de programmation logique tels que Prolog ou de traitement symbolique comme Lisp étant les langages de prédilection du domaine - de nombreux chercheurs ont évoqué les avantages que pourraient présenter des approches par objets : voir par exemple Zampolli (1991) pour la réutilisabilité des bases de connaissances linguistiques, et Zajac (1992) pour l'application des techniques du génie logiciel au traitement du langage naturel, en particulier le développement incrémental des grammaires.

L'approche orientée-objets est particulièrement appropriée lorsque l'extensibilité et la portabilité d'un composant vers d'autres applications sont primordiales, comme le système de génération SEMSYN (Rösner 1988), ou des systèmes de traduction multilingues tels que la maquette LIDIA décrite dans Boitet (1993) et Blanchon (1994). Ce type d'approche est aussi adopté par Berrendonner et al. (1992) et Véronis (1992) pour la représentation, l'extraction et l'exploitation des connaissances dans le contexte du traitement des langues naturelles, et par Stefanini et al. (1992) pour la réalisation d'un environnement d'analyse syntaxique.

Le concept typique de l'orientation-objets, à savoir l'héritage, est de plus en plus utilisé en dehors de l'approche par objets : voir par exemple (Carpenter & Pollard 1991) pour son utilisation dans un sytème basé sur une grammaire syntagmatique guidée par les têtes (*Head-driven Phrase Structure Grammar*), (Russel et al. 1991) dans un lexique basé sur l'unification, (Becker et al. 1991, Evans et al. 1995) dans un système basé sur les grammaires d'arbres adjoints (*Tree Adjoining Grammar*), (Mitamura & Nyberg 1992, Weisweber 1992) dans des systèmes de traduction automatique, (Böttcher 1993) dans un système à structure de traits, et (Reinhard & Gibbon, 1991) dans le domaine de la morphologie. Pour avoir une idée plus complète de l'utilisation de l'héritage dans le TALN, voir (Daelemans & Gazdar, 1990).

3 La Grammaire Universelle

3.1 Principes et paramètres

Une alternative aux grammaires indépendantes du contexte, inadaptées aux phénomènes complexes du langage naturel, consiste en l'utilisation de grammaires génératives modernes de type principes et paramètres, en particulier la grammaire universelle de la théorie GB, développée initialement par Chomsky (1981) et constamment raffinée depuis, voir entre autres Chomsky (1986a, 1986b). Certains analyseurs basés sur la théorie des principes et paramètres

sont décrits dans (Door 1990, Berwick 1991, Fong 1991, Merlo 1992). La théorie GB vise à l'élaboration d'une grammaire universelle, compatible avec la diversité des langues existantes, en tant que modèle de la compétence linguistique de l'être humain. Elle se veut aussi un modèle de l'acquisition du langage, compatible avec les théories psycholinguistiques. Au niveau le plus général, une telle grammaire permet d'exprimer l'association entre les représentations de la forme et celles du sens.

La grammaire universelle est un système de principes universels, qui ne varient pas d'une langue à l'autre, et de paramètres qui tiennent compte des propriétés spécifiques à chaque langue. Les valeurs des paramètres sont proposées par la GU parmi un choix restreint. Par exemple, les composants syntaxiques des phrases, sujet, verbe, objet,...sont communs à toutes les langues, par contre l'ordre de ces composants dépend de la langue. La grammaire d'un langage donné est donc une instantiation de la GU avec une certaine valeur des paramètres.

3.1.1 Niveaux de représentation

La théorie GB définit quatre niveaux de représentation des expressions du langage, ou descriptions structurales :

1. la **structure-D**, ou structure profonde, représente les relations de base entre les mots d'une phrase, plus spécifiquement la relation entre prédicat et arguments, et l'assignation des rôles thématiques aux arguments. La structure-D est générée par projection de l'information lexicale selon la catégorie lexicale, la grille thématique et la structure argumentale des prédicats selon les règles de la théorie $\overline{X}$.

2. la **structure-S**, ou structure de surface, représente les propriétés superficielles telles que l'ordre final des mots, leur cas, les phénomènes d'accord. La structure-S est dérivée de la structure-D par l'application de la règle Move-α dont les instantiations possibles sont le mouvement de tête, de syntagmes nominaux ou d'éléments interrogatifs.

3. la **forme logique** représente les propriétés logico-sémantiques de la phrase, un niveau entre la structure-S et la représentation sémantique. Elle représente l'interprétation des éléments qui ont une certaine portée dans la phrase, telle que les quantifieurs ou les éléments interrogatifs. La forme logique est dérivée de la structure-S par l'application de la règle Move-a, dont les instantiations possibles sont la montée des quantifieurs et des éléments interrogatifs.

4. la **forme phonologique** encode la suite de phonèmes constituant la phrase, la **forme graphique** la suite de graphèmes, qui dans le premier cas peut être prononcée, dans l'autre écrite. La forme phonologique est dérivée de la structure-S par linéarisation de la structure et application des règles d'élision et de liaison, de même que la forme graphique, selon les informations phonologico-graphiques contenues dans le lexique (consonne latente, voyelles faibles, Schwa,...).

Figure 1: Niveaux de représentation de GB

A chaque niveau l'information lexicale est présente, par le Principe de Projection Etendu (Chomsky 1986a), ce qui rend le lexique un composant central de la théorie GB, puisqu'il doit contenir les informations pertinentes pour tous les niveaux de représentation de la GU, mais chaque niveau n'a accès qu'à un sous-ensemble restreint de ces informations.

Le passage d'un niveau de représentation à l'autre détermine un composant de la grammaire, ou sous-système de règles : composant catégoriel, transformationnel, phonologico-graphique et logique. La figure 1 établit le lien entre les différents niveaux de représentation. Le sens des flèches indique le processus de génération de phrase. En analyse, le schéma se lit à l'envers, en partant d'une forme phonologique ou graphique et en construisant une structure-S au fur et à mesure de l'arrivée des mots. En traduction, un pas de plus est effectué en extrayant une structure-D de la structure-S.

L'application des règles de chaque composant ne se fait pas à la légère car chaque niveau de représentation doit satisfaire les contraintes imposées par les principes de la GU.

3.1.2 Sous-théories de la GU

Les principes de la GU s'organisent en sous-systèmes, ou sous-théories, spécialisés que l'on nomme modules. Chaque sous-système s'occupe d'un processus ou d'un phénomène particulier, mais les interactions étant fortes entre les différents sous-systèmes, ils ne constituent pas des modules dans le sens informatique du terme.

Les sous-théories comprennent la théorie Thêta, qui règle l'assignation des rôles thématiques aux arguments, la théorie du Cas, qui règle la distribution des syntagmes nominaux dans la phrase, la théorie du Liage, leur

interprétation référentielle. La théorie Xbar définit la structure hiérarchique de la phrase, la théorie du Gouvernement les relations structurales entre les constituants. La théorie du Mouvement définit les différents mouvements possibles, la théorie des Bornes les contraintes de localité du mouvement, la théorie des Chaînes la constitution des chaînes entre les éléments déplacés et leurs traces. La théorie du Contrôle règle la distribution et l'interprétation de PRO, une catégorie vide, tandis que les traces sont soumises au Principe des Catégories Vides.

3.2 La GU dans le paradigme objet

Les interactions entre les principes de la GU posent un problème pour une approche modulaire, car la GU n'est pas modulaire dans le sens informatique : chaque sous-théorie, même si elle regroupe des fonctionnalités spécifiques et reliées entre elles, n'a pas de structures de données propres, ni une interface minimale avec les autres sous-théories : sur le plan informatique, cela suppose le partage de nombreuses variables complexes entre les modules[8]. Il serait donc utopique de vouloir créer un module pour chaque sous-système de principes. Par contre, la modularisation de la GU est conceptuellement, et pratiquement, possible grâce à l'orientation-objets, car même si l'orientation-objets est basée sur une partie des principes de la modularité[9], les objets sont définis différemment des modules, et surtout, ils ont un comportement particulier : la notion d'héritage permet de définir les objets en cercles concentriques, par subsomption, du plus général à l'intérieur, vers le plus particulier à l'extérieur, sans interface entre eux puisque les objets intérieurs sont présents dans tous les objets qui les englobent. La notion de polymorphisme permet à un même message d'être traité différemment selon l'objet qui le reçoit, au contraire de la procédure qui doit être spécifique à chaque structure de données définie dans un module. Une propriété additionnelle, la persistance, permet aussi de traiter le contexte, en vue d'un traitement pragmatique ultérieur.

3.2.1 Classes - catégories grammaticales et syntaxiques

D'après la définition, une classe regroupe un ensemble d'objets qui possèdent les mêmes caractéristiques. On a donc une correspondance directe entre les classes et les catégories grammaticales, *noms*, *verbes*, *adjectifs*, etc., et syntaxiques, *NP*, *VP*, *AP*, etc... De même que chaque mot d'une phrase fait partie d'une catégorie grammaticale, il générera un objet de la classe correspondante - plusieurs dans les cas d'ambiguité - avec les attributs et les méthodes caractéristiques de sa classe. Par exemple, la phrase " Le chat mange la souris. " générera les objets lexicaux suivants :

- MDet(le), MNom(le), MNom(chat), MVerbe(mange), MDet (la), MNom(la), MNom(souris), MVerbe(souris).

[8] Dans l'implémentation actuelle, la structure de données de projection est présente dans son intégralité dans presque toutes les opérations effectuées lors de l'analyse, du transfert et de la génération.

[9] Encapsulation des données, interface minimale, spécialisation, réutilisabilité.

Comme les classes cachent leur représentation interne, si pour une raison ou une autre une classe doit être modifiée, cela peut se faire sans interférence avec les autres, gardant ainsi l'intégrité du système. D'autre part, chaque objet gérant son propre état, et n'acceptant que des messages, il n'est pas possible d'accéder directement à ses attributs, on ne passe plus de lourdes structures de données à travers les modules.

3.2.2 Héritage - universalité

La notion d'héritage permet de dériver, ou étendre, des classes à partir de classes existantes. Comme les classes dérivées héritent des attributs et méthodes de la classe de base, il n'est pas nécessaire de les réécrire, ce qui est une solution au problème de redondance du code. L'héritage est utilisé sur deux niveaux. Tout d'abord à l'intérieur même de la GU, pour définir une hiérarchie de classes compatibles entre elles. Les classes de base définissent les opérations applicables à toutes les classes de la hiérarchie, les classes dérivées implémentant les opérations spécifiques de la sous-classe. Par exemple, le schéma Xbar s'applique à toutes les catégories syntaxiques, il définit donc une opération de la classe de base. Par contre l'assignation de rôle thématique concerne quelques catégories seulement qui ont cette propriété; il définit donc une opération des classes dérivées concernées. L'héritage est aussi utilisé au niveau inter-linguistique : il permet de définir des classes de base abstraites correspondant aux composants de la GU, et de les étendre dans des classes plus spécifiques correspondant aux grammaires particulières, qui fixent la valeur des paramètres et implémentent les règles spécifiques de chaque langue. L'héritage garantissant la compatibilité entre les objets de la classe de base et ceux de ses sous-classes, les grammaires particulières sont compatibles avec la GU. Une conséquence importante est que le développement incrémental du système est rendu possible : ajouter une construction spécifique ou une nouvelle langue dans le système peut se faire à tout moment, sans qu'il soit besoin de modifier les classes existantes, et/ou les algorithmes pour autant que les nouvelles classes dérivées ont la même interface que celles de base.

3.2.3 Messages - principes

Les messages sont le seul moyen de communication entre les objets. Le message envoyé dit *ce* qu'il faut faire, l'objet détermine *comment* il faut le faire, c'est-à-dire quelle méthode invoquer, d'après son type dynamique. On peut voir une correspondance quasi directe entre les messages et les principes de la théorie GB. En effet, les principes sont des *contraintes* de bonne formation sur la structure de la phrase, ils disent quelles conditions les mots doivent remplir pour former une phrase correcte, selon leur catégorie grammaticale ou syntaxique et selon la langue pour les principes sur lesquels s'appliquent les paramètres. Par exemple, la théorie Xbar définit la projection maximale XP composée de Xbar et de ses spécifieurs, Xbar étant composée de X et de ses compléments. L'ordre d'attachement ainsi que le nombre et la catégorie des spécifieurs et des compléments potentiels sont des paramètres dépendant de la langue et/ou de la catégorie de X. Le même message est envoyé à tous les

objets, qui l'interpréteront différemment selon leur catégorie et leur langue. Des messages peuvent être envoyés à tous les objets ou seulement à certains d'entre eux, comme par exemple les messages de la théorie du Cas qui ne s'appliquent qu'aux Nps. Par exemple, la théorie du Liage définit les anaphores, pronoms et expressions référentielles, trois sortes de NPs. Appliquer le principe du Liage, c'est envoyer le message *Lier* à tous les NPs, quels qu'ils soient.

3.2.4 Polymorphisme - régularité

Le polymorphisme est en relation directe avec la notion de message : des objets différents peuvent répondre aux mêmes messages parce qu'ils ont les mêmes méthodes dans leur interface, leur implémentation pouvant être différente. On retrouve cette notion dans la théorie GB à deux niveaux. Tout d'abord au niveau de la structure de la phrase, par le principe de projection qui dit que l'information lexicale est représentée syntaxiquement, quelle que soit sa catégorie grammaticale, celle-ci déterminant le genre d'information disponible, et par la théorie Xbar, qui construit la projection maximale de chaque mot selon le même schéma. Enfin, la définition même de la grammaire GB comme grammaire universelle, dont les principes s'appliquent à toutes les langues, ceux-ci étant interprétés différemment selon la valeur de certains paramètres propres à chaque langue. Pour illustrer le premier point, la classe de base ItemLexical (cf. 4.2.2) contient une opération *Projeter* qui projette l'information lexicale dans une structure syntaxique. Envoyer le message ItemLexical.Projeter sera interprété différemment selon la catégorie de l'item lexical, i.e. $\underline{N} \rightarrow \underline{N}P$, $\underline{D} \rightarrow \underline{D}P$, $\underline{V} \rightarrow \underline{V}P$, etc... Le même message est envoyé à tous les objets qui y répondent selon leurs spécificités.

3.2.5 Persistance - contexte

La persistance est la propriété qu'ont les objets de survivre au programme qui les a créés. Ce n'est pas une propriété essentielle de la programmation orientée-objets, mais elle se montre d'une grande utilité à plusieurs niveaux. Tout d'abord au niveau purement informatique, elle permet de sauver de la place mémoire en cours d'exécution, en déchargeant de la mémoire de l'ordinateur les structures générées sans les perdre, celles-ci étant très gourmandes par rapport à cette ressource. Par la suite, on peut récupérer les objets générés précédemment, soit dans le même programme, soit dans un autre programme. Elle permet aussi de sauver du temps de calcul en gardant la valeur des paramètres de la grammaire entre deux exécutions, s'il n'est pas utile de les modifier, par exemple lorsque les langages source et cible sont les mêmes en traduction automatique. Au niveau linguistique - lorsqu'un traitement pragmatique est prévu -elle permet d'implémenter la notion de contexte pour déterminer les antécédents des pronoms, gérérer les bonnes correspondances temporelles et désambiguïser des phrases globalement ambiguës.

4 Description structurale

Le but de ce travail n'est pas de fournir une modélisation orientée-objets de tout le système de traduction, mais principalement de la grammaire utilisée dans le système. Il paraît cependant utile de le décrire dans les grandes lignes et d'en donner une modélisation orientée-objets schématique. Nous prenons comme modèle général le système ITS-2 représenté dans la figure 2, duquel on va extraire les classes, déterminer leurs attributs et opérations, puis les messages qui leurs sont envoyés. La méthode de décomposition étant un procédé itératif, des retours en arrière sont possibles, car la définition de classes de plus bas niveau peut influencer celle de classes de plus haut niveau. Des diagrammes montreront visuellement la structure hiérarchique - les relations d'héritage - et la structure dynamique - les interactions des classes entre elles.

Rappelons que le modèle développé ci-après est très général : il décrit le processus de traduction en principe applicable à toutes les langues - du moins un sous-ensemble de langues ayant les mêmes propriétés.

4.1 Le système de traduction

Dans le système de la figure 2, la traduction se base sur le principe du transfert et de la génération, dont les fondements théoriques sont les suivants :

1. on peut trouver pour chaque phrase source une phrase cible équivalente du point de vue du sens si les langues ne sont pas trop distantes du point de vue linguistique et culturel;

2. le principe de la préservation des structures impose de préserver la forme et le style de la langue source dans la langue cible autant que possible. Sinon, on recourt à des transpositions. Pour plus de détails, voir Wehrli (1993b) pour les spécifications et Pinnagoda et Ramluckun (1993) pour l'implémentation.

Le système se compose de quatre processus principaux :

- l'**analyse** (source) retourne une ou plusieurs structure-S sources correspondant à toutes les analyses possibles d'une phrase d'entrée, selon les spécifications de la grammaire source, des unités lexicales du langage source et des informations fournies par l'utilisateur lorsque l'analyseur ne peut résoudre un problème seul. Elle se décompose en analyse lexicale, qui reconnaît les mots et les frontières de phrases du texte, et analyse syntaxique, qui construit la(les) structure(s) adéquate(s) pour chaque phrase. L'analyseur doit être réaliste du point de vue pratique, i.e. l'analyse d'une phrase doit être de courte durée. Un des choix de recherche est l'adoption d'un point de vue psycholinguistique, ce qui signifie qu'il est souhaitable qu'il fonctionne plus ou moins comme l'analyseur humain - du moins dans les limites de notre compréhension actuelle - ce qui implique les propriétés suivantes (Laenzlinger & Wehrli 1991) : il traite les phrases dans l'ordre de production du langage; synchrone, l'analyse lexicale et l'analyse syntaxique se font en parallèle sur

les mots disponibles; incrémental, il construit les structures au fur et à mesure de l'arrivée de nouveaux constituants[10].

- le **transfert** (source-cible) met en correspondance à un certain niveau d'abstraction une structure-S source et une structure-D cible sur la base des correspondances entre items lexicaux cible et source établies dans un lexique bilingue et de la projection des informations lexicales cibles suivant la grammaire cible et des informations de l'utilisateur lors de problèmes insolubles. La première opération consiste donc à extraire la structure-D de la structure-S source, qui contient le prédicat verbal et ses compléments (arguments et ajouts). Pour préserver les structures, le transfert établit les transformations syntaxiques à appliquer à la structure-D cible pour dériver la structure-S cible correspondant à la structure-S source.

- la **génération transformationnelle** (cible) dérive la structure-S cible à partir d'une structure-D cible, des spécifications de la grammaire cible, du lexique cible et d'informations données par l'utilisateur, en appliquant les transformations syntaxiques nécessaires pour préserver la structure source. Des transpositions sont nécessaires si des structures sources n'ont pas de correspondance dans la langue cible ou si leur usage ne leur permet pas d'être préservées. Dans la théorie GB actuelle, il n'y a plus qu'une seule transformation, définie par la règle Move-a, qui rend compte de toutes les transformations possibles, comme le passif, par exemple. Sur le plan informatique, appliquer une seule transformation reviendrait à générer une multitude de structures qu'il faudrait filtrer par la suite. C'est pourquoi la génération transformationnelle sera établie sur le modèle classique des transformations, à savoir pour chaque construction sera associé un groupe d'opérations élémentaires telles que déplacement de constituant, substitution, effacement, etc... Pour avoir des détails sur les différentes transformations, voir Laenzlinger & Werhli (1991).

- la **génération morphologique** (cible) dérive les phrase(s) de sortie cible(s) à partir des structure(s)-S cible(s) en choisissant dans le lexique cible les bonnes représentations morphologiques, avec des règles d'adaptation spécifiées par la grammaire cible. Les mots sont cherchés selon la catégorie grammaticale et les traits morphologiques déterminants pour cette catégorie, calculés au cours du transfert et des transformations. La génération morphologique correspond au composant PF de la grammaire GB.

La figure 3 montre la relation entre les composants du système de traduction et les niveaux de représentation de l'information structurale, i.e. la structure-D et la structure-S.

A partir du modèle fonctionnel de la figure 2, un premier niveau de classes peut être dérivé selon les règles suivantes :

[10]Les analyseurs [F/D]IPS implémentés au LATL sont en outre basés sur une stratégie mixte ascendante - ils sont dirigés par les données - avec un filtre descendant, et parallèle - ils poursuivent toutes les alternatives possibles en même temps.

Figure 2: Modèle fonctionnel du système de traduction ITS-2

- les sources de données, symbolisées par des rectangles forment les classes,
- les processus, symbolisés par des cercles forment des opérations qui s'appliquent sur des objets de la classe origine de la flèche grasse, et qui produisent des objets de la classe à l'arrivée de la flèche. Les opérations sont rattachées à la classe d'entrée.
- les classes à l'origine des flèches simples sont des collaborateurs fournissant des services à la classe comportant l'opération.

La notation utilisée pour décrire les classes est la suivante : les classes sont des noms commençant par une majuscule et soulignés, les attributs sont des noms ou des adjectifs commençant par une minuscules et en italique, et les opérations sont des verbes commençant par une majuscule et en italique. Une classe suivant une opération et introduite par une flèche est le résultat de l'opération, les classes listées sur la dernière ligne de la description sont les collaborateurs de la classe.

L'application des règles produit les classes suivantes, dans la notation proposée. Pour ne pas augmenter la complexité, le niveau structure-D et le niveau structure-S sont confondus dans une même structure (la motivation en est donnée au paragraphe 4.3.1), de même que la génération transformationnelle et morphologique sont combinées en une seule opération :

Les classes Lexique et LexiqueBilingue sont explicitées à la section 4.2, les classes Structure et Grammaire à la section 4.3. La classe Utilisateur représente l'interface utilisateur, elle n'est pas développée dans ce travail. Les autres classes ainsi que celles qu'elles génèrent ou qu'elles utilisent de manière interne sont décrites ci-dessous dans leur généralité, leur description détaillée dépassant le cadre de ce travail.

Figure 3: Système de traduction du point de vue niveaux de représentation

Classe	Actions	Collaborateurs
Texte	Traduire(→ Texte) Analyser(→ Structure)	Grammaire, Lexique, Utilisateur, Stru
Structure	Transférer(→ Structure) Générer(→ Structure, → Texte)	Utilisateur, LexiqueBilingue, Gramma Lexique
Utilisateur		
Lexique		
LexiqueBilingue		
Grammaire		
SystèmeDeTraduction		Utilisateur, Texte

Table 2: Classes principales

4.1.1 Classe SystèmeDeTraduction

Dans un système multilingue, on devrait pouvoir spécifier plusieurs langues cibles pour ne pas analyser plusieurs fois le même texte source s'il doit être traduit dans plusieurs langues, car le transfert s'effectue sur les mêmes structures de surface. Les attributs de la classe SystèmeDeTraduction sont donc *langueSource*, *langueCibles* et *degréInteraction* sélectionnés par l'utilisateur. Il n'y a qu'une seule instance de cette classe, qui a comme principale fonction de coordonner les autres objets. C'est lui qui va instancier les lexiques et les grammaires, et lancer la traduction, en lançant le message *Traduire* à un objet de type Texte et produit un objet du même type dans la langue cible. Il peut fonctionner comme analyseur, dans ce cas il envoie le message *Analyser* à un objet de type Texte et produit une série d'objets de type Structure en langue cible pour chaque phrase constituant le texte. S'il y a plusieurs langues cibles, le système envoie un message *Analyser* à un objet de type

Texte et envoie les messages *Transférer* et *Générer* à chaque objet de type Structure produit par l'analyse.

4.1.2 Classe Texte

Cette classe comprend l'attribut *contenu*, la suite de caractères constituant le texte, et les opérations *Ecrire*, pour afficher sur un support adéquat le contenu alphanumérique du texte, *Traduire (→ Texte)* qui produit un texte en langue cible et *Analyser (→ Structure)* qui produit la structure syntaxique du texte source. Comme les analyses lexicale et syntaxique sont synchrones, l'analyse syntaxique est lancée dès qu'un symbole de type SMot est disponible. La classe Texte comporte l'opération *ProchainSymbole(→ symbole)* qui retourne le prochain symbole du texte.

4.1.3 Classe Symbole

Le texte passé en entrée au système est une suite de phrases, chacune constituée de caractères alphabétiques constituant des mots, de caractères numériques constituant des nombres et de signes de ponctuation dont certains sont importants car ils déterminent les limites de phrases (point, point d'exclamation, point d'interrogation) ou ils relient entre eux deux mots pour en faire un mot composé (tiret, apostrophe) ou une abréviation (point)[11]. Le premier but de l'analyse lexicale d'un texte est de retourner la liste des symboles constituant le texte. Un symbole est un mot s'il existe au moins une entrée dans la base de données des mots, l'entrée étant créée dynamiquement dans le cas de mots inconnus. La classe Symbole comprend comme attribut chaîne, le ou les caractères constituant le symbole. Elle est étendue dans deux sous-classes, SMot et SPonctuation. L'attribut propre de SMot est *listeMots*, la liste des entrées lexicales correspondant à une chaîne de caractères. C'est une liste d'objets de la classe ItemLexical, qui est développée au paragraphe 4.2.2. L'attribut propre de SPonctuation est un booléen, *finPhrase* qui permet de détecter les frontières de phrases.

4.2 Les lexiques

La description textuelle des composants du système existant permet d'extraire les classes les plus évidentes. Nous commençons par l'information lexicale, car le lexique occupe une position centrale dans la théorie GB d'une part, l'information lexicale étant présente à tous les niveaux de représentation, et car tous les processus du système utilisent des informations lexicales d'autre part. L'analyse et la génération utilisent des informations propres aux mots d'une langue, stockées dans un lexique monolingue, le transfert utilise des informations qui mettent en correspondance des mots (ou groupes de mots) de la langue source avec des mots (ou groupes de mots) de la langue cible, stockées dans un lexique bilingue.

Pour des raisons conceptuelles et d'efficacité dans la recherche, l'information composant le lexique monolingue d'une langue est partagée entre

[11] Les exemples donnés ne sont pas exhaustifs et peuvent être différents selon la langue traitée.

trois classes d'objets (pour une description détaillée, voir Walther (1991) et Walther & Wehrli (1995)) :

1. les **mots** contiennent l'information morphologique (toutes les variantes morphologiques d'un mot donné sont répertoriées) et grammaticale (traits de conjugaison, d'accord, de cas, etc....), ainsi que la représentation phonétique.

2. les **lexèmes** contiennent l'information syntaxique (sélection, sous-catégorisation,...) et sémantiques (rôles thématiques), qui reste invariante pour les différentes formes morphologiques d'un mot donné[12].

3. les **expressions idiomatiques** contiennent les groupes de mots ayant un sens particulier et devant être reconnus comme tels.

Les relations morphologiques entre les mots sont exprimées par un ensemble de relations entre les entrées de la base de données des mots. L'appartenance d'un mot à une expression idiomatique est une information sémantique exprimée par une relation entre le lexème et la base de données des expressions idiomatiques, car l'information morphologique d'une expression idiomatique ne diffère pas de l'information morphologique des mots qui la constituant. Les relations connectant les mots aux lexèmes expriment l'inflexion, toutes les formes morphologiques d'un lexème donné, et l'homographie, toutes les lectures (sens) d'un mot donné.

Le lexique bilingue est utilisé lors du transfert source-cible. Ce qui est pertinent à ce niveau-là, c'est l'information syntaxique et sémantique, c'est pourquoi il met en correspondance des lexèmes sources et cibles. Mais on a vu que certaines catégories lexicales n'ont pas de lexème, la correspondance dans ce cas-là se fait entre les mots directement. Un troisième type de correspondance se fait entre des expressions idiomatiques, source et cible. Une description détaillée de l'information bilingue est donnée dans Wehrli (1995).

4.2.1 Classe LexiqueMonolingue

Quelle que soit la langue à laquelle appartient le lexique, l'organisation est la même, que ce soit l'organisation externe, (partage entre trois bases de données) ou interne (attributs d'une entrée), seul le contenu informatif change. C'est pourquoi les lexiques source et cible sont des instances de la classe LexiqueMonolingue.

Un système informatique est composé d'objets logiciels, qui n'interagissent qu'avec d'autres objets du programme, et d'objets matériels, qui interfacent des objets physiques (un instrument de mesure, par exemple). Les lexiques peuvent être considérés comme des objets matériels, dans la mesure où les bases de données les composant sont implémentées sous forme de fichiers (indexés séquentiels, une clé étant un nombre unique). Les objets de la classe LexiqueMonolingue représentent l'interface avec ces bases de données. Un

[12] Les items lexicaux qui n'ont pas ou peu de variations morphologiques (prépositions, déterminants) n'ont pas de lexème, ceci pour des raisons pratiques d'implémentation, ils contiennent donc aussi l'information syntaxique et sémantique.

premier attribut de cette classe permet de déterminer la langue du lexique, *langue*. Nous supposons pour le moment que c'est une chaîne de caractères, même s'il est implémenté différemment. Un autre attribut de cette classe est la location des différentes bases de données. Chaque utilisateur ayant le droit de définir son propre ensemble de fichiers, on différentie les fichiers " standards ", ceux du système, des fichiers " privés ", ceux de l'utilisateur. L'attribut *location* est un attribut complexe composé de six noms de fichiers, à savoir *motsStandards*, *motsPrivés*, *lexèmesStandard*, *lexèmesPrivés*, *idiomes-Standards*, *idiomesPrivés*.

La propriété de dissimulation de l'information cache l'état (constitué par les attributs) des objets au monde extérieur, qui ne peut qu'utiliser les services mis à disposition par l'objet. Ceux-ci permettent en principe deux opérations : la lecture et l'écriture d'un ou plusieurs attributs. L'utilisateur n'a pas à savoir l'organisation externe des lexiques : on suppose que les fichiers standards se trouvent à un endroit bien précis et que c'est la langue qui détermine leur nom. Pour les fichiers privés, l'utilisateur peut les stocker dans le répertoire de son choix, mais les noms sont générés par le logiciel. Pour qu'un utilisateur puisse avoir plusieurs fichiers privés dans le même répertoire, ce serait le cas d'un traducteur qui possède des lexiques spécifiques à certains sous-domaines particuliers, on admet la possibilité d'un préfixe qui discrimine entre les différents domaines. La classe met à disposition l'opération *FixerLocation(langue, répertoirePrivé, préfixe)* pour générer les 6 fichiers d'une langue donnée et d'un utilisateur particulier. Si *répertoirePrivé* est une chaîne de caractère vide, il n'y a pas de fichiers privés, si préfixe est vide, les fichiers privés sont les noms de fichiers standard préfixés par la chaîne "priv_", sinon ils sont préfixés par la chaîne "préfixe_". L'opération *Modifier-Location(répertoirePrivé, préfixe)*[13] permet de changer l'ensemble des fichiers privés entre deux traductions. L'opération *QuelleEstLocation(→ langue, → répertoirePrivé, → préfixe)* permet de récupérer les informations sur la langue du lexique (source ou cible) et si l'utilisateur possède des lexiques privés ou non, si oui, s'ils sont préfixés ou non.

Dans une vision globale, qui définit les classes en toute généralité, pour être réutilisables dans n'importe quelle application, on définirait toutes les opérations possibles sur elles. Dans le cas de la classe LexiqueMonolingue, on définirait les opérations d'insertion, de mise à jour, de destruction et de recherche d'éléments. Ce n'est pas le but de ce travail, c'est pourquoi nous ne nous intéressons qu'à la recherche d'informations, qui est différente selon le processus en jeu : en analyse (lexicographique), on cherche si une forme graphique correspond à un mot de la langue, auquel cas on veut récupérer toutes les informations lexicales sur ce mot, tandis qu'en génération (morphologique), on veut retrouver la forme graphique correspondant à des traits morphologiques particuliers pour un lexème donné. La classe LexiqueMonolingue met à disposition les deux opérations de recherche *ChercherMot(chaîne, → listeMots)* qui correspond au premier type de recherche, listeMots étant la liste des entrées dans la base de données des mots pour une même forme graphique, et *ChercherForme(lexème, traits, →chaîne)* cor-

[13]Le symbole → signifie que l'attribut qui suit est retourné comme résultat de l'opération.

respondant au deuxième type, *chaîne* étant une chaîne de caractères, *traits* un ensemble de traits morphologiques, *lexème* un lexème donné et *itemLexical* l'entrée combinée des trois bases de données. Les trois derniers attributs sont des attributs complexes qui seront détaillés ci-dessous.

4.2.2 Classe ItemLexical

Un objet de la classe ItemLexical correspond à l'entrée lexicale d'un mot, partagée entre les trois bases de données. Les attributs de cette classe sont la forme graphique du mot, *clé*, le numéro du mot dans la base de données des mots, *indexMot*, le lexème du mot, *lexème*, et la prochaine forme morphologique associée au même lexème, *suivant*. Les caractéristiques morphologiques dépendant de la catégorie lexicale du mot, cette classe est étendue dans les sous-classes MNom, MVerbe, MAdjectif, MDéterminant, MAdverbe, MPréposition, MConjonction, qui ont comme attributs propres les caractéristiques associées à leur catégorie, par exemple *genre* et *nombre* pour MNom, *temps*, *mode*, *personne* pour MVerbe, etc...A ce stade de la conception, il n'y a pas besoin d'entrer dans les détails.

Les opérations mises à disposition de la classe ItemLexical sont *FixerClé(chaîne)* qui initialise un objet de la classe avec la chaîne de caractères donnée en paramètre, et *QuelleEstClé(→chaîne)* qui retourne dans une chaîne de caractères la forme graphique du mot. La théorie GB nous dit que l'information lexicale est projetée dans une structure syntaxique de même catégorie. ItemLexical met donc à disposition une opération *Projeter(→ projMax)*, dont l'attribut généré *projMax* sera détaillé au paragraphe 4.3.1.

4.2.3 Classe Lexème

Cette classe regroupe les informations sémantique et syntaxique invariantes pour les différentes formes morphologiques d'un mot, informations partagées entre les deux bases de données des lexèmes et des idiomes. Ses attributs sont le numéro du mot dans la base de données des lexèmes, *indexLexème*, le numéro de la première forme morphologique dans la base de données des mots, *indexMot*, le numéro de l'entrée dans la base de données des expressions idiomatiques si le mot fait partie d'une expression idiomatique, *indexIdiome*, et le prochain lexème s'il comporte des homographes, *suivant*. Les caractéristiques sémantiques et syntaxiques dépendant de la catégorie du lexème, cette classe est étendue dans les sous-classes LNom, LVerbe et LAdjectif, les autres catégories n'ayant que peu de variation, elles ne comportent pas de lexème. Les détails importent peu à ce stade de la conception.

4.2.4 Classe Idiome

Cette classe regroupe les informations concernant les expressions idiomatiques. Ses attributs sont le numéro de l'expression dans la base de données des expressions idiomatiques, *indexIdiome*, le numéro du premier lexème sur lequel est basée l'expression, *indexLexème1*, le numéro du deuxième lexème, *indexLexème2*, les traits caractéristiques de l'expression, *traits*, le nombre de morceaux dont est constituée l'expression, *nbMorceaux*, la table contenant les

morceaux, *tableMorceaux*, et la chaîne de caractères constituant l'expression, *chaîne*. Chaque morceau est déterminé par sa *position* dans la chaîne de caractères, nombre encodant le début et la fin du morceau[14], par le type de la *préposition* s'il y en a une et par des *contraintes*.

Pour illustrer, l'expression " casser sa pipe " contient deux morceaux, " casser " en position 5, et " pipe " en position 10013, les traits généraux de l'expression sont " pas_de_passif ", tandis qu'une contrainte associée au deuxième morceau est " possessif ". La position dans la chaîne de caractères est utilisée pour faire du pattern matching, lorsque la contrainte est " littéral ", évitant ainsi de décrire une expression avec une profusion de traits d'une part, et pour être sûr que la chaîne d'entrée ne contient pas d'autres constituants qui ne font pas partie de l'expression d'autre part. Par exemple, " Il a cassé sa pipe jaune " conserve sa valeur littérale, mais si on ne testait que le nombre, singulier, il serait considéré comme une expression idiomatique.

4.2.5 Classe Traits

Cette classe regroupe des traits morphologiques. Les détails dépendent de l'implémentation, ils importent peu ici.

4.2.6 Classe LexiqueBilingue

La classe LexiqueBilingue représente l'interface avec la base de données contenant toutes les correspondances entre lexèmes (mots et expressions idiomatiques) d'un couple de langues, sans spécification du sens source-cible. C'est pourquoi la classe contient deux attributs, *langueSource* et *langueCible* qui permettent de déterminer l'ordre de lecture des entrées. Un autre attribut de cette classe est la *location* de la base de données. De même que pour les lexiques monolingues, l'utilisateur peut spécifier un lexique bilingue privé. L'attribut *location* est un attribut complexe comprenant deux noms de fichiers, à savoir *bilingueStandard* et *bilingue Privé*.

Le fichier standard se trouve à un endroit bien précis et ce sont les langues source et cible qui déterminent son nom. Pour le fichier privé, l'utilisateur peut le stocker dans le répertoire de son choix, mais le nom est généré par le logiciel. Pour qu'un utilisateur puisse avoir plusieurs fichiers privés dans le même répertoire, ce serait le cas d'un traducteur qui possède des lexiques bilingues spécifiques à certains sous-domaines particuliers, on admet la possibilité d'un préfixe qui discrimine entre les différents domaines. La classe met à disposition l'opération *FixerLocation(langueSource, langueCible, répertoirePrivé, préfixe)* pour générer le fichier bilingue d'un couple de langues données et d'un utilisateur particulier. Si *répertoirePrivé* est une chaîne de caractère vide, il n'y a pas de fichiers privés, si *préfixe* est vide, le fichier privé est le nom du fichier standard préfixé par la chaîne "priv_", sinon il est préfixé par la chaîne "préfixe_". L'opération *ModifierLocation(répertoirePrivé, préfixe)* permet de changer de fichier privé entre deux traductions. L'opération *QuelleEstLocation(→ langueSource, → langueCible, →*

[14]La position est déterminée par la formule suivante : position = début * 1000 + fin. Pour retrouver le début et la fin avec position : début = position DIV 1000 et fin = position MOD 1000.

répertoirePrivé, → préfixe) permet de récupérer les informations sur les langues source et cible du lexique bilingue et si l'utilisateur possède un lexique bilingue privé ou non, si oui, s'il est préfixé ou non.

Comme pour les lexiques monolingues, l'opération pertinente offerte aux autres objets est la recherche d'information, grâce à l'opération mise à disposition par la classe LexiqueBilingue *ChercherCorrespondance(itemSource, →itemBilingue)* qui retourne dans *itemBilingue* l'entrée du fichier correspondant à un *itemSource* , le numéro d'un lexème, d'un mot ou d'une expression idiomatique source. Les numéros sont formés des numéros d'entrées dans les bases de données auquel on ajoute un préfixe qui détermine la base de données.

4.2.7 Classe ItemBilingue

Un objet de cette classe représente une entrée du fichier bilingue, qui est une correspondance entre lexèmes, mots ou expressions idiomatiques. Ses attributs sont (voir Wehrli 1995 pour la description détaillée des attributs) :

item_1, *item_2* les numéros d'entrées mises en correspondance,

descripteur_1, *descripteur_2* des chaînes de caractères permettant de spécifier le sens d'homonymes,

fréquence_1, *fréquence_2* les fréquences relatives

contexte le sous-domaine d'utilisation de la correspondance

arguments la mise en correspondance des arguments lorsque les numéros correspondent à des entrées verbales

4.2.8 Classes de l'information lexicale

Dans les paragraphes précédents, nous avons extrait les classes des lexiques. Dans ce paragraphe, nous résumons les attributs et opérations pour chaque classe. Les informations extraites sont illustrées à l'aide des diagrammes objets des figures 4 pour la hiérarchie des classes et 5 pour les relations entre les classes.

LexiqueMonolingue

- langue, location
- FixerLocation(langue, répertoirePrivé, préfixe)
- QuelleEstLocation(→ langue, → répertoirePrivé, → préfixe)
- ModifierLocation(répertoirePrivé, préfixe)
- ChercherMot(chaîne, →listeMots)
- ChercherForme(lexème, traits, → chaîne)

ItemLexical

- indexMot, lexème, suivant

Figure 4: Hiérarchie des classes de l'information lexicale

- FixerClé(chaîne)
- QuelleEstClé(→ chaîne)
- Projeter(→ projMax)

MNom, MVerbe, MAdjectif, MDéterminant, MAdverbe, MPréposition, MConjonctio
(sous-classes de ItemLexical)

Lexème

- indexLexème, indexMot, indexIdiome, suivant

LNom, LVerbe, Ladjectif (sous-classes de Lexème)

Idiome

- indexIdiome, indexLexème1, indexLexème2, traits, nbMorceaux, tableMorceaux, chaîne

LexiqueBilingue

- langueSource, langueCible, location
- FixerLocation(langueSource, langueCible, répertoirePrivé, préfixe)
- ModifierLocation(répertoirePrivé, préfixe)
- QuelleEstLocation(→langueSource, →langueCible, →répertoirePrivé, →préfixe)

Figure 5: Relations entre les différentes classes de l'information lexicale

- ChercherCorrespondance(itemSource, →itemBilingue)

ItemBilingue

- item_1, item_2, descripteur_1, descripteur_2, fréquence_1, fréquence_2, contexte, arguments

4.3 La grammaire

La grammaire détermine la deuxième sorte d'information, l'information structurale du système. Elle est utilisée par tous les processus de la traduction : grammaire de la langue source en analyse, grammaire de la langue cible en transfert et génération. Dans la théorie GB, elle est une instance de la grammaire universelle dans laquelle on a donné une valeur spécifique aux paramètres. La théorie GB n'ayant pas été développée dans un but informatique, il n'existe pas vraiment de formalisation de la GU. De plus, sa perpétuelle évolution et le manque d'unité dans les différentes propositions théoriques rendent la spécification de la GU très difficile. Pour détailler les différentes sous-théories et tenter d'extraire les classes adéquates, nous nous basons essentiellement sur Haegeman (1994), Laenzlinger & Wehrli (1991) et Wehrli (1993b, 1993c), les autres sources étant citées explicitement.

4.3.1 La théorie $\overline{X}$

La théorie $\overline{X}$ définit la structure hiérarchique de la phrase, son squelette. Le principe principal de cette théorie est le schéma $\overline{X}$, qui regroupe les règles de dérivation des syntagmes.

Figure 6: Schéma $\overline{X}$ pour le français et l'anglais

Le schéma $\overline{X}$

Le schéma $\overline{X}$ s'applique aux syntagmes constituant la phrase et à la phrase elle-même : pour toute tête lexicale et fonctionnelle X, on définit la projection intermédiaire $\overline{X}$ et la projection maximale XP selon les règles donnée dans la définition 1 :

Définition 1 Schéma $\overline{X}$

1. $XP \rightarrow Spec\ \overline{X}$
2. $\overline{X} \rightarrow X\ Compl$

L'utilisation des accolades signifie que l'ordre des constituants n'est pas spécifié par la GU : il est déterminé par deux paramètres dont la valeur dépend de chaque langue :

1. *specAvantXbar* : s'il a la valeur *vrai*, l'ordre est Spec – $\overline{X}$, sinon $\overline{X}$ – Spec
2. *têteAvantCompl* : s'il a la valeur *vrai*, l'ordre est X – Compl, sinon Compl – X.

Pour le français et l'anglais, les deux paramètres ont la valeur *vrai*, quelque soit la catégorie de X, ce qui donne le schéma de la figure 6.

Dans certaines langues, les paramètres peuvent avoir une valeur différente selon la catégorie de X : en allemand, *têteAvantCompl* a la valeur *faux* pour les catégories T, V et Adj, ce qui donne le schéma $\overline{X}$ de la figure 7 pour l'allemand.

Certaines langues dites à ordre libre des mots, tel le Warlpiri, n'ont pas de valeur fixe pour ces paramètres. Pour tenir compte de cette particularité, la valeur des paramètres ne sera pas simplement binaire, mais ternaire (*vrai*, *faux*, *indéterminé*).

Spec, pour spécifieur, et Compl, pour complément, sont tous deux des projections maximales, qui sont dans une relation de localité très forte avec la tête X : le principe de la binarité stricte (Kayne 1984) implique qu'ils sont uniques pour toute catégorie lexicale ou fonctionnelle[15].

[15] Ce qui ne correspond pas à ce qui est implémenté dans F/IPS, où Spec et Compl sont tous deux des listes de projections maximales.

Figure 7: Schéma $\overline{X}$ pour l'allemand

Le type du complément est déterminé par les traits de sous-catégorisation de l'élément lexical. Par contre, la position spécifieur est une position ouverte, c'est-à-dire qu'a priori n'importe quel élément peut y être inséré dans la mesure où sa présence ne viole aucun principe de la grammaire.

Les catégories lexicales sont fixes, elles se composent de noms (N), verbes (V), adjectifs (Adj), prépositions (P), déterminant (D), adverbes (A) et conjonctions (Conj). Les catégories fonctionnelles par c ontre dépendent des différents courants de la théorie et des chercheurs, mais elles comportent au minimum l'inflexion verbale (Infl ou I), qui contient les traits d'accord verbal (AGR) et le temps (T), qui permet de construire des phrases tensées ou infinitives, et le complémenteur (C) qui permet la construction de phrases subordonnées et/ou interrogatives. La structure complète d'une phrase est donnée par les règles de la définition 2 :

Définition 2 Structure d'une phrase complète

1. $CP \rightarrow Spec\ \overline{C}$
2. $\overline{C} \rightarrow C\ IP$
3. $IP \rightarrow Spec\ \overline{I}$
4. $\overline{I} \rightarrow I\ VP$

Certains auteurs adoptent l'hypothèse de l'Inflexion articulée (*Split-Infl hypothesis*) de Pollock (1989), qui décompose Infl selon ses traits en deux têtes indépendantes, Agr_S, pour les traits d'accord verbal et T pour le temps. Ces mêmes auteurs utilisent une tête fonctionnelle Neg pour les phrases négatives. Cette tête contient la particule de négation (" ne " en français). Il peut y avoir encore d'autres têtes fonctionnelles, comme F pour les phrases prédicatives autres que verbales, (Je considère *Marie intelligente*), et Agr, une tête contenant des traits d'accord, utilisée pour l'accord du sujet, de l'objet, etc... Et il peut y en avoir encore d'autres. L'intérêt d'une approche orientée-objets est qu'elle permet de compléter la grammaire selon les différents courants de la théorie. Pour le moment, seuls I, C et F sont gardés.

L'hypothèse-DP d'Abney (1987) est assumée, c'est-à-dire que le syntagme nominal est réinterprété comme DP, avec le déterminant comme tête et le NP comme complément :

Figure 8: VP-Shell

Définition 3 Hypothèse-DP

1. $DP \rightarrow Spec\ \overline{D}$
2. $\overline{D} \rightarrow D\ NP$

Toutes les positions dans le schéma $\overline{X}$ n'ont pas le même statut : certaines positions peuvent recevoir des arguments d'un prédicat, ce sont les positions argumentales ou positions-A, d'autres ne le peuvent pas, ce sont les positions non argumentales ou positions-$\overline{A}$. Les positions [Spec,VP], [Spec,IP] et [Compl,$\overline{V}$] sont des positions argumentales, [Spec, CP] est typiquement une position-$\overline{A}$. Les arguments ne peuvent se trouver que dans des positions-A.

VP-Shell

Puisqu'un VP ne contient que deux positions argumentales, si le verbe requiert trois arguments, puisque les arguments doivent se trouver en position-A, on utilise la notion de VP-shell de Larson (1988) : le VP déjà projeté est attaché comme complément d'un VP supérieur dont la tête est vide et la position de spécifieur peut recevoir le troisième argument. La structure de VP-shell est donnée dans la figure 8.

Quels arguments sont projetés dans quelle position sera vue au paragraphe 4.3.2.

Adjonction

Si une catégorie lexicale peut admettre plusieurs spécifieurs ou compléments autres que des arguments, ceux-ci sont intégrés à la structure par adjonction, une opération qui crée une catégorie multi-segmentale à partir d'une catégorie unique (May 1985). Le principe de préservation des structures implique que la catégorie est soit une catégorie de degré 0, i.e. une tête, soit une projection maximale (Chomsky 1986). Il n'y a en aucun cas d'adjonction à $\overline{X}$. C'est un processus récursif sur la droite ou sur la gauche de XP ou de X. Le schéma $\overline{X}$ est complété par la règle 4[16], résultant dans les deux schémas de la figure 9 :

[16]L'adjonction de tête n'est pas intégrée dans notre schéma Xbar car elle l'alourdirait sans apporter de bénéfice important. Si une tête doit s'adjoindre à une autre tête, elle seront simplement "concaténées".

Figure 9: Adjonction à gauche et à droite

Définition 4 Structure d'adjonction
$XP \rightarrow YP\ XP$

Pour illustrer cela[17], la phrase donnée en (1a) comporte deux compléments, un complément d'objet direct et un complément circonstanciel de lieu, elle a la structure donnée en (1b) :

(1)a. Jean a mangé une pomme dans le jardin.

b. [IP Jean [I' a [VP mangé [DP une pomme]] [PP dans le jardin]]]

L'exemple 2 montre le cas de deux spécifieurs, avec deux solutions, l'une au moyen d'un ajout (2b), l'autre au moyen d'une phrase prédicative (2c), qui est plus à même de rendre compte de l'accord entre le nom et l'adjectif postposé.

(2)a. le beau chat noir.

b. [DP le [NP [AdjP beau] chat] [AdjP noir]]]

c. [DP le [NP [AdjP beau] chati] [FP ei [F' [AdjP noir]]]]

L'adjonction est un processus coûteux qui est utilisé en dernier recours lorsqu'il n'y a pas de position disponible dans le schéma $\overline{X}$, que ce soit pour des spécifieurs ou compléments multiples, ou pour des éléments qui se déplacent. (voir Baltin 1982, May 1985, Chomsky 1986b, 1992, Laenzlinger 1993, pour des compléments sur la théorie de l'adjonction).

Structure de Projection

Dans une optique dérivationnelle, le schéma $\overline{X}$ s'applique à la structure-D, de laquelle est dérivée la structure-S par l'application de la règle Move-a. Le Principe de Préservation des Structures de Emonds (1976) implique que toutes les positions syntaxiques requises au niveau structure-S existent au niveau structure-D, que les catégories ne peuvent pas être modifiées par les mouvements, et que les mouvements laissent des traces dans leur position

[17]Il est à noter que dans F/IPS, cela ne se passe pas comme ça, puisque les compléments circonstanciels, des ajouts en fait, sont mis dans la liste des compléments.

d'origine. Cela implique que la structure-S inclut la structure-D et que, dans la structure-S, les mouvements sont en quelque sorte codés dans les chaînes constituées des éléments déplacés et de leurs traces. L'utilisation d'un seul niveau de représentation, la structure-S, et la substitution de la notion de chaîne à celle de mouvement constitue ce qu'on appelle l'approche représentationnelle de la théorie GB. Ce type d'approche est bien adapté à l'analyse d'une phrase, puisque le point de départ est une forme phonétique ou graphique, beaucoup plus proche d'une structure-S que d'une structure-D. Par contre, l'optique dérivationnelle est mieux adaptée à la génération, puisqu'il faut, pour le transfert, extraire de la structure-S les informations pertinentes, donc la structure-D, en replaçant les éléments déplacés dans leur position d'origine. La structure définie dans ce paragraphe est une structure-S.

Vu la régularité du schéma $\overline{X}$, la structure de projection peut éliminer le niveau intermédiaire $\overline{X}$: Laenzlinger & Wehrli (1991) définissent une projection maximale XP comprenant une tête lexicale ou fonctionnelle X, une liste de spécifieurs et une liste de compléments, listes qui peuvent comprendre des ajouts et des arguments, mais sans faire la différence structurellement. Pour garder la différence structurelle, la structure de projection retenue privilégie les relations locales entre la tête, le spécifieur et le complément : elle comprend une tête, un spécifieur et un complément, tous deux des projections maximales, et deux listes d'ajouts à XP, l'une pour l'adjonction à gauche, et l'autre pour l'adjonction à droite, car quelque soit la valeur des paramètres *têteAvantCompl* et *specAvant*$\overline{X}$, les ajouts à gauche sont toujours les éléments les plus à gauche, et les ajouts à droite les éléments les plus à droite. La structure hiérarchique peut être retrouvée sans problème : dans la liste d'ajouts à gauche, le premier constituant est le noeud le plus haut de la structure, dans la liste d'ajouts à droite, le premier est le plus bas. La linéarisation peut s'opérer facilement : on parcourt la liste des ajouts à gauche, puis, selon la valeur des paramètres d'ordre, la tête, le spécifieur et le complément, et enfin la liste des ajouts à droite.

Classes ProjectionMaximale et Tête

Le schéma $\overline{X}$ définit une structure de données arborescente, la classe de base ProjectionMaximale[18], abrégée en PM dans la suite du texte, dont les attributs sont *spec*, *compl*, deux objets de la classe de base, *ajoutsGauche* et *ajoutsDroite*, deux listes de projections maximales correspondant à des ajouts à droite et des ajouts à gauche, les deux paramètres d'ordre *têteAvantCompl* et *specAvant*$\overline{X}$, et *tête*, la tête de la projection. Le type de cette dernière détermine le type de la projection maximale : lexicale, la tête est un objet de type ItemLexical, fonctionnelle, la tête est un objet de type Traits. Mais dans certains cas, la tête fonctionnelle peut dominer un élément lexical, par exemple C lorsqu'il contient le complémenteur d'une phrase subordonnée. De plus, les traits sont différents si la tête est I, elle contient les traits d'accord verbal et de temps, C, elle contient le trait [+/- WH], etc... C'est pourquoi *tête* est un objet de la classe Tête, contenant les deux attributs *traits*, un objet de la classe (ou de ses sous-classes) Traits, et *itemLexical*, un objet

[18]Remplace la classe Structure du paragraphe 4.1

de la classe (ou de ses sous-classes) ItemLexical, qui peuvent être nuls dans certains cas.

Les différents syntagmes de la théorie Xbar sont des classes dérivées de la classe de base, les projections lexicales NP, VP, AdjP, DP, PrepP, AP, ConjP, et fonctionnelles IP, CP, FP, etc... Les projections lexicales sont projetées à partir de l'information lexicale, grâce à l'opération *Projeter(→ projMax)* de la classe ItemLexical. Cette opération reste abstraite dans la classe de base, car le type de la projection dépend de la catégorie du mot : elle est redéfinie dans chaque sous-classe MNom, MVerbe, MAdjectif, MDéterminant, MAdverbe, MPréposition et MConjonction. Les projections fonctionnelles sont projetées à partir de certaines projections maximales, IP est projeté à partir de VP, CP est projeté à partir de IP, DP peut être projeté de NP si le nom n'a pas de déterminant, grâce à l'opération *Projeter(→ projMax)* de la classe PM[19]. Mais certaines projections fonctionnelles sont projetées à partir d'éléments lexicaux, tels les conjonctions de type complémenteur qui projettent des CP et les adverbes de type particule de négation qui projettent des NegP si on adopte l'hypothèse de l'inflexion décomposée[20].

Projeter signifie créer un objet de la sous-classe appropriée de PM. Mais le détail de la classe n'étant pas connu à l'extérieur, PM doit mettre à disposition des opérations d'initialisation, *FixerLex(itemLexical)* qui affecte un itemLexical à sa tête, *FixerFonc(traits)* qui affecte des traits à sa tête, et *FixerOrdre(têteAvantCompl, specAvant$\overline{X}$)* qui donne une valeur aux paramètres *têteAvantCompl* et *specAvant$\overline{X}$*. Si on peut donner une valeur aux attributs, on veut pouvoir aussi lire leur valeur, avec les opérations inverses, *QuelEstLex (→ itemLexical)* qui retourne l'itemLexical associé à sa tête, *QuelEstFonc (→ traits)* qui retourne les traits contenus associés à sa tête, et *QuelEstOrdre (→têteAvantCompl, →specAvant$\overline{X}$)* qui retourne la valeur des paramètres d'ordre. Les valeurs sont données aux paramètres d'ordre lors de l'application de l'opération de projection.

La classe Tête doit aussi mettre à disposition des opérations d'accès à son contenu, typiquement *FixerLex (itemLexical)*, *QuelEstLex (→ itemLexical)*, *FixerFonc (traits)* et *QuelEstFonc (→ traits)*, utilisées par PM. Les différentes catégories de têtes sont des sous-classes de Tête : N, V, Adj, Prép, D, Conj, A, I, F et C.

Pour donner une valeur à ses autres éléments, PM met à disposition les opérations *FixerSpec (projMax)*, *projMax* devient spécifieur de PM, *FixerCompl (projMax)*, *projMax* devient complément de PM, et *FixerAjout (projMax,sens)*, *projMax* est insérée dans la liste des ajouts selon la valeur de *sens*, qui est prise dans (*gauche*, *droite*). Elle met aussi à disposition les opérations inverses *QuelEstSpec (→ projMax)*, qui retourne la projection maximale spécifieur, *QuelEstCompl (→ projMax)*, qui retourne la projection maximale complément et *QuelEstAjout (sens, → projMax, → fin)*, qui retourne la projection maximale ajout selon le sens voulu et une valeur qui indique s'il y en a d'autres (*fin = faux*) ou non (*fin = vrai*).

L'écriture de la projection se fait au moyen de l'opération *Ecrire (→*

[19]Le polymorphisme permet de définir des opérations ayant le même nom mais s'appliquant à des classes différentes.

[20]Ce qui n'est pas le cas ici.

chaîne), définie dans la classe de base, qui retourne une chaîne de caractères contenant les noeuds terminaux de la projection. Elle effectue en quelque sorte une linéarisation de la projection maximale selon les paramètres *têteAvantCompl* et *specAvant*$\overline{X}$. Cette opération fait partie du composant PF de la grammaire, elle dépend en grande partie de la langue, c'est pourquoi elle reste à un niveau plus ou moins abstrait dans la classe de base, et sera redéfinie dans les classes dérivées.

4.3.2 Principe de Projection Etendu et hiérarchie thématique

Le principe de projection étendu dit deux choses : tout d'abord que l'information lexicale se retrouve au niveau syntaxique, deuxièmement que toute phrase comporte un sujet.

La première partie du principe fait le lien entre le lexique et le composant structural de la syntaxe. C'est par lui que l'on sait que le lexique doit comprendre toutes les informations morphologiques, grammaticales, syntaxiques et sémantiques, que les trois composants de la grammaire (syntaxe, PF et LF) accèdent à ces informations, mais pas forcément aux mêmes. La classe ItemLexical contient toutes ces informations, l'opération *Projeter (→ projMax)* de cette classe effectuant la projection de l'information lexicale sur la structure adéquate.

Le prédicat verbal donne le corps à la phrase : le verbe et ses arguments déterminent les constituants indispensables de la phrase. Les relations sémantiques entre le prédicat et les arguments sont spécifiées dans la grille thématique du prédicat, qui est une liste non ordonnée de rôles thématiques. Par exemple, le verbe "manger" en (3a) spécifie deux rôles thématiques, l'agent (qui effectue l'action) et le thème (ce sur quoi porte l'action), remplis respectivement par le sujet et l'objet direct. Le verbe "donner" en (3b) spécifie trois rôles thématiques : l'agent, le thème et le bénéficiaire (à qui profite l'action), respectivement remplis par le sujet, l'objet direct et l'objet indirect.

(3)a. Jean mange une pomme.

b. Jean donne un livre à Marie.

Les verbes "craindre", "préoccuper" et "plaire" sont des verbes dits psychologiques. Ils comportent tous les trois deux rôles thématiques identiques, l'expérient (qui expérimente un état mental) et le thème (l'état ou le contenu de l'état mental), mais ils sont réalisés par des arguments différents : pour "craindre" (4a), l'expérient est le sujet et le thème est un objet direct, pour "préoccuper" (4b), c'est le thème qui est le sujet et l'expérient qui est objet direct, tandis que pour "plaire" (4c), le thème est aussi réalisé par le sujet, mais l'expérient l'est par un objet indirect.

(4)a. Jean craint Marie.

b. Marie préoccupe Jean.

c. Marie plaît à Jean.

Il y a encore d'autres rôles thématiques, tels que le but, la source, la destination,... Selon l'hypothèse de l'uniformité de l'assignation thématique (UTAH, *Uniformity of Theta Assignment Hypothesis*) de Baker (1986), des relations thématiques identiques sont représentées au niveau structure-D par des relations structurales identiques. Cette hypothèse se vérifie pour les verbes d'action, mais semble inadéquate pour les verbes psychologiques. Belletti &Rizzi (1988) donnent une solution à ce problème en postulant d'une part que l'entrée lexicale des verbes comprend une grille thématique et une grille de cas inhérents, d'autre part que les verbes psychologiques possèdent des grilles thématiques différentes, car dans un cas il y a spécification d'un rôle thématique externe, qui sera projeté dans la position sujet, et des grilles de cas inhérents aussi différentes. Larson (1988) propose une autre solution en postulant une sorte de passivisation pour les verbes des classes de "préoccuper" et de "plaire". Quoiqu'il en soit, dans les deux solutions l'expérient qui n'est pas sujet du verbe est projeté dans une position adjointe à VP, ce qui est possible avec la structure $\overline{X}$ définie dans le paragraphe précédent. Tous les deux proposent aussi une hiérarchie thématique à la Jackendoff (1972) ou Carrier-Duncan (1985) ou encore Grimshaw (1990) du genre de celle définie dans la définition 5 :

Définition 5 Hiérarchie thématique
Agent > *Expérient* > *Thème* > *Bénéficiaire* > ...

Larson émet un principe de projection très général qui tient compte de cette hiérarchie :

Définition 6 Principe de projection étendu de Larson
Si un prédicat a détermine des rôles thématiques t1,...tn, alors le rôle le plus bas dans la hiérarchie thématique est assigné à l'argument le plus bas dans la structure, etc... jusqu'au rôle le plus haut qui est assigné à l'argument le plus haut.

Ce principe s'applique très bien aux verbes à deux arguments ou plus : dans le premier cas, la position complément est remplie par le rôle le plus bas dans la hiérarchie et la position spécifieur par le rôle le plus haut. Si le rôle le plus haut n'est pas le rôle externe, comme dans le cas de certains verbes psychologiques, il y a passivisation et il est projeté dans une position adjointe à VP. Dans le cas de trois arguments, il y a construction d'un VP-shell, et le rôle le plus haut est projeté dans la position spécifieur du VP englobant. Dans le cas de verbes à un argument, Larson projette le rôle dans la position spécifieur si c'est un argument externe, dans la position complément si c'est un argument interne (cas des verbes ergatifs).

Avec l'hypothèse du sujet interne au VP, voir Koopman&Sportiche (1991), tous les sujets sont générés en structure-D dans la position [Spec,VP] et se déplacent en position [Spec, IP][21] pour des raisons de cas. Tous les arguments, sujet compris, sont internes au VP. Si l'on ne veut plus faire explicitement la distinction entre rôle thématique interne et rôle externe dans le lexique,

[21]Ou [Spec,AgrVP] si on adopte l'hypothèse de l'Inflection articulée

le principe de projection ne peut pas s'appliquer de la même manière sur les verbes à un seul argument. Dans ce cas, il est court-circuité par la hiérarchie thématique : si l'unique rôle est un agent ou un expérient, il est projeté en position spécifieur, si c'est un thème, il est projeté dans la position complément. Pour les verbes psychologiques, on suppose que l'expérient est projeté en position spécifieur et que s'il est marqué d'un cas inhérent, il reste dans cette position, le thème se déplaçant en position [spec,IP]. C'est une hypothèse qui n'est malheureusement pas encore vérifiée, mais qui aurait le mérite de la simplicité.

La première partie du Principe de Projection Etendu ne donne pas lieu à la création d'une classe, ni à la modification d'une classe existante, mais il concerne le corps de l'opération *Projeter (→ projMax)* de la classe ItemLexical, en particulier ceux de type V. C'est une bonne chose, car les différentes hypothèses de projection pourront être implémentées et testées sans conséquence sur l'interface des classes.

La deuxième partie du principe dit que toute phrase doit avoir un sujet, autrement dit que la position [Spec, IP] soit remplie. La position sujet peut être remplie de plusieurs façons : par un DP réalisé lexicalement (5a), une clause (5b), le pronom nul *PRO* (5c) et le pronom nul *pro* (5d) :

(5)a. [DP Jean] a mangé une pomme dans le jardin.

b. [CP The story that I've heard] is amazing.

c. L'histoire que j'ai entendue est étonnante.

d. PRO partir en voyage est toujours une aventure.

e. pro ho telefonato a Maria.

f. J'ai téléphoné à Marie.

Dans l'hypothèse du sujet interne au VP, le sujet est généré (en structure profonde) en position [spec, VP] et se déplace en position [spec,IP] pour des raisons de cas. pro et PRO sont aussi générés dans la position [spec,VP] et se déplacent pour des raisons de gouvernement : PRO ne doit pas être gouverné, et la position [spec,IP] ne peut pas être gouvernée par I car elle n'est pas assez forte (traits d'accord inexistants dans le cas des infinitives), tandis que pro doit se déplacer pour être gouverné par une tête aux traits très riches qui peut lui donner un contenu (l'inflexion des verbes dans les langues à sujet nul est très riche).

Classe IP : complément

pro ne peut apparaître comme sujet que si la langue est dite à sujet nul, comme l'italien par exemple. La classe IP possède un attribut *proDrop*, qui sert de paramètre pour déterminer ce type de langues, l'opération *FixerPro (proDrop)* qui donne la valeur *vrai* ou *faux* au paramètre, et *QuelEstPro (→ proDrop)* pour retourner la valeur du paramètre. PRO quant à lui est soumis

à la théorie du Contrôle, développée au paragraphe 4.3.8, et les DP lexicaux à la théorie du Cas développée au paragraphe 4.3.5.

La deuxième partie du Principe de Projection Etendu agit comme filtre, car les constructions qui ne remplissent pas la condition ne forment pas des phrases bien formées du langage, et sont par conséquent éliminées. C'est un filtre concernant la classe IP exclusivement, appliqué par l'opération *Existe-Sujet (→ résultat)*, qui retourne dans *résultat* la valeur *vrai* s'il y a un sujet, la valeur *faux* sinon.

Classes Syntagme, pro et PRO

L'application de ce filtre implique l'insertion de pro ou de PRO, selon les conditions énoncées plus haut, c'est-à-dire la création de deux objets vides, mais compatibles avec des PM, puisqu'ils apparaissent en position spécifieur. En faire des objets de la classe DP alourdirait le traitement, puisque les objets héritent de tous les attributs de leur classe, alors que tous les éléments, du moins ceux vus jusqu'ici, sont inutilisés. C'est pourquoi PM n'est plus la classe de base, mais une classe dérivée d'une super-classe abstraite Syntagme qui rassemble les classes PM, pro et PRO, les deux catégories vides découlant du Principe de Projection Etendu. L'attribut *spec* de la classe PM doit être déclaré de type Syntagme pour que le spécifieur puisse être un objet de la classe PM, pro ou PRO.

La classe Syntagme prévoit l'opération d'écriture, *Ecrire (→ chaîne)*, mais cette opération reste abstraite car elle dépend des attributs des classes dérivées : elle est redéfinie dans les sous-classes de Syntagme (on avait déjà vu ce mécanisme pour la classe PM).

Classe pro

pro remplace un sujet pronom personnel dans les langues à sujet nul comme l'italien ou l'espagnol. Dans ce cas, il est gouverné par la tête Infl qui lui donne un contenu, c'est-à-dire les traits d'accord. Les attributs de pro sont donc *genre*, *nombre* et *personne*.

Le paramètre pro-drop est plus général puisqu'il dit que pro doit être gouverné par une tête, sans préciser laquelle. A priori pro peut être autre chose qu'un sujet. Un exemple est donné pour l'Italien en (6) dans Rizzi (1986) : pro peut être objet, donc gouverné par V :

(6) Questo conduce pro_i a [PRO_i concludere quanto segue].
Ceci conduit " les gens " à conclure ce qui suit.

Dans ce cas, il reçoit une interprétation arbitraire (les traits d'accord sont fixes, 3ème personne, masculin pluriel.). La classe VP possède aussi un attribut *proDrop*, qui sert de paramètre pour déterminer si pro peut être objet, l'opération *FixerPro (proDrop)* lui donne la valeur *vrai* ou *faux*, et *QuelEstPro (→ proDrop)* retourne sa valeur. Cette valeur est déterminée par le choix de la langue, donc elle est déterminée au moment de la création d'objets de type VP ou IP, c'est-à-dire lors de l'application de l'opération *Projeter* des Mmots. L'opération *ExisteObjet (→ résultat)* de la classe VP permet de déterminer si le verbe a pu décharger un rôle thématique généralement associé à l'objet.

4.3.3 La théorie des Chaînes

Les deux catégories vides pro et PRO, représentées par les classes pro et PRO, sous-classes de Syntagme, découlent du Principe de Projection Etendu. D'autres principes de la grammaire exigent que des éléments, projections maximales ou têtes, ne restent pas dans leur position canonique, mais se déplacent dans d'autres positions. Le Principe de Préservation des Structures exige que la position de base continue d'exister, remplie par une trace de l'élément déplacé. La trace est une catégorie vide qui est en relation avec son antécédent dans une chaîne, dont la définition est donnée en 7 :

Définition 7 Chaîne

1. *$C = (x_1, x_n)$ est une chaîne ssi, pour $1 \leq i < n$, x_i est un lieur local de x_{i+1}*

2. *x est un lieur pour y ssi, pour x et y de même catégorie, x et y sont coindicés et x c-commande y*

3. *x est un lieur local pour y ssi x est un lieur pour y et il n'y a pas de z tel que z est un lieur pour y et z n'est pas un lieur pour x.*

Le premier élément de la chaîne est appelé tête de la chaîne, c'est une projection maximale ou une tête, le dernier est appelé pied de la chaîne, c'est une trace qui se trouve dans la position de base, les autres étant des traces intermédiaires. La tête de la chaîne détermine son type : une projection maximale en position-A détermine une chaîne-A, une projection maximale en position-$\overline{A}$ détermine une chaîne-$\overline{A}$, une tête détermine une chaîne-X^0. Les traces respectives, traces-NP, traces-qu ou traces-X^0 respectivement, sont toutes les trois soumises au Principe des Catégories Vides (ECP) donné dans la définition 8 :

Définition 8 Principe des catégories vides *Les traces doivent être proprement gouvernées. A gouverne proprement B ssi A théta-gouverne B et/ou A antécédent-gouverne B.*

Les traces d'arguments doivent être théta-gouvernées et antécédent-gouvernées (Chomsky 1981), les traces des ajouts ne sont forcément qu'antécédent-gouvernées. La c-commande et le gouvernement sont vus au paragraphe 4.3.4.

Des exemples de traces-NP (7a), traces-qu (7b) et traces-X0 (7c) sont donnés ci-dessous. La tête de la chaîne peut éventuellement être une catégorie vide, si elle fait partie d'une autre chaîne, ou un opérateur nul en position [Spec, CP], noté OP, qui sert de gouverneur aux traces dans certaines phrases relatives (7d) et infinitives (7e) :

(7)a. [IP Cette histoire$_i$ est racontée t_i par tous les habitants].

b. [CP Que$_i$ [C' veux-tu manger t_i ce soir]]?

c. [CP [C' Veux$_i$- [IP tu t_i [VP manger une glace]]]]?

d. This is the man [CP1 OP_i that [IP1 John claims [CP2 t'_i that [IP2 he will invite t_i]]]].
" C'est l'homme que John prétend qu'il invitera."
C'est l'homme que John prétend inviter.

e. I_j need a man [CP OP_i [IP PRO_j to love t_i]].
J'ai besoin d'un homme à aimer.

Il y a un paramètre linguistique qui permet de contracter le complémenteur avec son spécifieur, en anglais avec l'opérateur nul, en français avec une trace, pour permettre au complémenteur de gouverner une trace en position sujet d'une relative. Ce paramètre est exprimé dans la définition 9 respectivement pour l'anglais et le français, pour rendre compte des phrases (8), où l'opérateur nul ne peut pas antécédent-gouverner la trace en position sujet à cause du complémenteur :

Définition 9 Contraction du complémenteur

1. *en anglais : OP_i that $\rightarrow$ $that_i$*

2. *en français : XP_i que $\rightarrow$ qui_i /- [IP t_i*

En français, XP peut être l'opérateur nul (8b) ou une trace intermédiaire (8c) :

(8)a. This is the letter [CP $that_i$ [IP t_i will surprise Poirot]].

b. L'homme [CP qui_i [IP t_i a été arrêté t_i]].

c. L'homme [CP OP_i que [IP je pense [CP qui_i [IP t_i a été arrêté t_i]]]].

Les trous parasites (*parasitic gaps*), dont un exemple est donné en (9), sont considérés comme des traces d'opérateurs nuls. Les phrases contenant des trous parasites sont composées de deux chaînes-$\overline{A}$: une composée d'un élément-qu et de sa trace, l'autre de l'opérateur nul et de sa trace. Les deux chaînes sont unifiées par le processus de composition de chaînes (les deux chaînes sont inclues dans une seule chaîne), pour donner une interprétation au trou parasite.

(9) Poirot is a man [CP who_i [IP I interviewed t_i [before [CP OP_j [IP PRO hiring t_j]]]]].

Si une trace peut se trouver dans la position complément d'une préposition, phénomène que l'on appelle"preposition-strandin", est déterminé par un paramètre linguistique. En anglais, ce paramètre est vrai, comme le montre l'exemple (10) :

(10) $Whom_i$ will the police inspector give the money [PP to t_i]?

Classes CatégorieX^0, Tête et TraceX^0

Pour qu'une tête soit compatible avec une catégorie pleine ou vide, la classe Tête vue à la section 4.3.1 n'est plus une classe de base, mais une classe dérivée d'une super-classe abstraite CatégorieX^0, qui réunit les catégories pleines, Tête, et vides, TraceX^0. L'attribut *tête* de PM est déclaré de type CatégorieX^0 pour être compatible avec des objets de l'une ou l'autre classe. Les attributs de TraceX^0 sont *indice*, l'indice partagé entre la trace et son antécédent, et *catégorie*, la catégorie de la trace. On a vu que les différentes catégories de têtes "pleines" dérivent des sous-classes de Tête, car leur comportement est différent. Pour les traces, le comportement est le même quelque soit la catégorie, c'est pourquoi la catégorie n'est pas à la base de sous-classes, mais constitue un attribut de la classe. Les opérations de TraceX^0 associées aux attributs sont *FixerIndice (indice)*, FixerCatégorie (catégorie), *QuelEstIndice (→ indice)* et *QuelEstCatégorie (→ catégorie)*.

Classes TraceMax et OP

Les traces des projections maximales, traces-NP et traces-qu, sont des objets de la classe TraceMax, sous-classe de Syntagme. Elle comporte comme attributs *indice* et *catégorie*, et les opérations associées *FixerIndice (indice)*, *FixerCatégorie (catégorie)*, *QuelEstIndice (→ indice)* et *QuelEstCatégorie (→ catégorie)*. Un autre attribut est *type*, qui prend ses valeurs dans (*np*, *qu*), pour différentier les traces-NP des traces-qu. Les opérations associées sont *FixerType (type)* et *QuelEstType (→type)*.

La classe OP représente l'opérateur nul. C'est aussi une sous-classe de Syntagme.

Classe CP : complément

La classe CP contient un attribut supplémentaire, *contraction*, de type booléen, qui détermine si le complémenteur peut être contracté avec son spécifieur. Les différentes configurations, celles de la définition 9 pour le français et l'anglais entre autres, font partie de l'opération *ContracterComplémenteur (→ C)* qui modifie la tête de type C (celle-ci prend les traits du spécifieur).

Classes Chaîne et Séquence

La classe de base Chaîne contient l'attribut *indice*, qui identifie la chaîne, et les opérations associées, *FixerIndice (indice)* et *QuelEstIndice (→ indice)*, l'attribut *séquence*, un objet de la classe Séquence, qui est une liste d'éléments de type Noeud, pour pouvoir contenir des projections maximales, dans le cas de chaînes-A et de chaînes-$\overline{A}$, et des têtes, dans le cas de chaînes-X^0. Elle met à disposition l'opération abstraite *Insère (Noeud,position)* où *position* prend ses valeurs dans l'ensemble (*tête*, *suivant*, *précedent*,*pied*). Elle est étendue dans deux sous-classes ChaîneMax et ChaîneX^0, respectivement pour les chaînes-A et chaînes-$\overline{A}$ et pour les chaînes-X^0. Pour tenir compte des différences de comportement entre les chaînes-A et les chaînes-$\overline{A}$, ChaîneMax est étendue dans les deux sous-classes, ChaîneA et Chaîne$\overline{A}$. L'attribut *séquence* est une liste d'éléments de type Syntagme dans le cas de ces deux dernières sous-classes, tandis qu'il sera une liste d'éléments de type CatégorieX^0

dans le cas de la classe ChaîneX^0. L'opération *Insère* est redéfinie dans chaque sous-classe pour tenir compte du type de chaîne.

La classe Chaîne$\overline{A}$ contient une opération supplémentaire, *Composition (chaîne$\overline{A}$)*, qui intègre à la chaîne dont elle est appelée celle passée en paramètre. Cette opération est utilisée pour interpréter les trous parasites.

Classe PP : complément

Pour tenir compte du "preposition-stranding", la classe PP contient un attribut *prepStrand* de type booléen et les opérations associées *FixerPrepStrand (prepStrand)* et *QuelEstPrepStrand (→ prepStrand)* de lecture et d'écriture de l'attribut. S'il est *vrai*, le complément de PP peut être un objet de la classe TraceMax (pour autant que la chaîne à laquelle appartient la trace est bien formée). En anglais, cet attribut reçoit la valeur *vrai*, en français, la valeur *faux*.

4.3.4 Théorie du Gouvernement

Cette théorie gère les relations structurales dans la phrase. La notion de gouvernement est centrale à la théorie GB, puisqu'elle est utilisée par à peu près tous les principes. Elle est basée sur les relation de dominance, de c-commande, de barrière et de minimalité.

Dominance (Chomsky 1986b)

Définition 10 Dominance

Un noeud A est dominé par un noeud B si et seulement si le noeud A est dominé par chaque segment de B.

Cette définition de la dominance tient compte de l'adjonction à XP : dans la figure 9 du paragraphe 4.3.1, XP est composé de deux segments, XP1 et XP2, mais seul XP1 domine YP, donc YP n'est pas dominé par XP. En règle générale, les ajouts à XP ne sont pas dominés par XP, seules les relations suivantes sont valides à l'intérieur d'une projection maximale :

1. XP domine Spec, X et Compl, ainsi que les éléments qu'ils dominent,
2. Spec et Compl dominent leurs constituants respectifs,
3. X ne domine rien du tout.

Définition 11 c-commande (Chomsky 1986b)

A c-commande B si et seulement si :

1. *A ne domine pas B,*
2. *tous les X qui dominent A dominent aussi B*

Si X est le premier noeud branchant, on a affaire à la c-commande stricte, si X est la première projection maximale, on a affaire à la m-commande. Les relations suivantes s'appliquent dans une projection maximale :

1. Spec (c-/m-)commande X^0 et Compl
2. X^0 (c-/m-)commande Compl
3. Compl (c-/m-)commande X^0
4. X^0 m-commande Spec.

Gouvernement (Rizzi 1990)

Le gouvernement peut se faire de deux manières : par tête (12) ou par antécédent (13) :

Définition 12 Gouvernement par tête

X tête-gouverne Y ssi :

(a) X est une tête

(b) X m-commande[22] *Y*

1. X est soit N, V, Adj, P et I

(a) Il n'y a aucune barrière entre X et Y

(b) la Minimalité Relativisée est respectée.

Définition 13 Gouvernement par antécédent

X W-antécédent-gouverne Y (W = A, A', X0) ssi :

(a) X est dans une position-W

(b) X c-commande Y

1. X et Y sont coindicés

(a) Il n'y a aucune barrière entre X et Y

(b) la Minimalité Relativisée est respectée.

Les clauses (i) sont les conditions configurationnelles, la clause (ii), la condition de substance[23], et les clauses (iii), les conditions de localité. La m-commande est exigée lors du gouvernement de tête pour unifier l'assignation des rôles thématiques (assignation d'un rôle thématique par V au sujet dans [spec,VP], et des cas (assignation du nominatif par Infl au sujet dans [spec, IP]).

La notion de barrière est définie par Chomsky (1986b) dans la théorie des Barrières, qui réinterprète la Condition de Sous-Jacence de la théorie des Bornes traditionnelle et qui est complémentaire de la théorie de la Minimalité Relativisée de Rizzi (1990), bien qu'il semble y avoir des contradictions entre le passage d'une barrière, qui est permis lors du mouvement, et le gouvernement par antécédent, qui n'admet aucune barrière. La notion de barrière (14) est basée sur les notions de L-marquage, de catégorie bloquante (BC) et d'héritage.

[22] Avec la m-commande, on étend le domaine de la tête à son spécifieur.

[23] Cette clause définit un ensemble minimal de gouverneurs potentiels, elle est paramétrisable selon les langues.

Définition 14 Barrière

1. *A L-marque B ssi A est une catégorie lexicale qui théta-gouverne (gouverne et assigne un rôle thématique) B.*

2. *C est une BC (catégorie bloquante) pour B ssi C n'est pas L-marqué et C domine B.*

3. *c. A est une barrière pour B si (a) ou (b) :*

   (a) *A est une projection maximale et A domine immédiatement*[24] *C, C est une BC pour B (notion d'héritage).*

   (b) *A est une BC pour B, A n'est pas IP.*

Remarque Dans les langues où le verbe fléchi se déplace en I, en français par exemple, on considère que VP est L-marqué, donc qu'il ne constitue pas une barrière intrinsèque. Par contre dans les autres, comme l'anglais, il constitue une barrière.

La notion de minimalité relativisée (Rizzi 1990) est donnée dans la définition (15) :

Définition 15 Minimalité Relativisée *X a-gouverne*[25] *Y ssi il n'y a pas de Z tel que*

1. *Z est un a-gouverneur potentiel typique pour Y*

2. *Z c-commande Y et ne c-commande pas X*

La clause (i) doit être spécifiée pour les quatre cas de gouvernement (gouvernement de tête, gouvernement A-antécédent, $\overline{A}$-antécédent et X^0-antécédent respectivement)

La notion de gouverneur potentiel est explicitée dans la définition 16.

Définition 16 Gouverneurs

1. *Z est un tête-gouverneur potentiel typique pour Y* $\Rightarrow$ *Z est une tête m-commandant Y*

2. *Z est un antécédent-gouverneur potentiel typique pour Y,*

   (a) *Y est dans une chaîne-A, Z est un spécifieur-A c-commandant Y*

   (b) *Y est dans une chaîne-$\overline{A}$, Z est un spécifieur-$\overline{A}$ c-commandant Y*

   (c) *Y est dans une chaîne-X^0, Z est une tête c-commandant Y*

[24] Comme la notion de barrière concerne des projections maximales, les projections intermédiaires, Xbar, n'interviennent pas dans la notion de dominance immédiate.

[25] a est une variable qui peut prendre comme valeur tête ou antécédent

Classe Noeud, classe Tête : complément

Comme il y a des opérations qui s'appliquent autant à des projections maximales qu'à des têtes, la classe abstraite Noeud assure la compatibilité entre des objets de ces deux classes en rassemblant les classes Syntagme et CatégorieX^0.

Seules les projections maximales pleines peuvent dominer des éléments, mais comme un résultat négatif est aussi important, puisqu'il est utilisé dans la définition de la c-commande, et que les éléments dominés peuvent être des projections maximales ou des têtes, vides ou pleines, l'opération *Domine (noeud, → résultat)* est une opération de Noeud, qui vérifie si l'objet dont on appelle cette opération domine l'élément *noeud*. C'est bien évidemment une opération récursive.

L'opération *c-commande (noeud, → résultat)* de Noeud vérifie si l'objet dont elle est appelée c-commande l'élément *noeud*, l'opération *m-commande (noeud, → résultat)* de Noeud vérifie si l'objet dont elle est appelée m-commande l'élément *noeud*.

La relation de gouvernement par antécédent entre deux éléments est déterminée par les opérations *AntécédentGouverne (noeud, → résultat)* de la classe Noeud, qui est redéfinie pour chaque classe concernée, à savoir PM et Tête, les catégories vides ne pouvant pas être gouverneurs, la relation de gouvernement de tête est déterminée par l'opération *Gouverne (noeud, → résultat)* de la classe Tête. Les valeurs que peut prendre résultat sont prises dans l'ensemble (*ok*, *barrière*, *minimalité*, *nonApplicable*), qui signifient respectivement que toutes les conditions du gouvernement sont remplies, qu'il y a une barrière au gouvernement, que la minimalité n'est pas respectée ou que l'objet de la classe dont on appelle l'opération n'est pas un gouverneur potentiel pour noeud. Les clauses configurationnelles et de substance du gouvernement forment le corps de les opérations, elles ne sont pas détaillées ici.

La notion de barrière constitue le corps de l'opération *EstBarrière (noeud, → résultat)* de la classe PM. Pour vérifier la clause (iiia) de localité, il faut appeler l'opération *EstBarrière (y, → résultat)* pour toutes les projections maximales *xp* se situant à l'intérieur de la projection maximale contenant le gouverneur *g* que l'on est en train de tester. Si le résultat est *vrai*, alors il y a une barrière, la condition de localité n'est pas respectée et *g* ne gouverne pas *y*.

La clause (iiib) de localité est vérifiée en appelant l'opération *Gouverne (y, → résultat)* pour chaque gouverneur potentiel *x* se situant à l'intérieur de la projection maximale contenant le gouverneur *g* que l'on est en train de tester. Si résultat est *ok*, alors la localité n'est pas respectée, et *g* ne gouverne pas *y*.

4.3.5 Théorie du Cas

La théorie du cas gère la distribution des syntagmes nominaux à l'intérieur de la phrase. Elle fait la distinction entre cas abstrait et cas morphologique d'une part, entre cas structurel et cas inhérent d'autre part. Le cas abstrait est une caractéristique universelle, tandis que le cas morphologique correspondant est un paramètre variable selon les langues. Ainsi en français et en anglais,

les cas accusatif et datif morphologiques ne se retrouvent que dans le cas des pronoms (11a,b), à la différence de l'allemand où il existe aussi pour les noms (11c) :

(11)a. Marie$_{+NOM}$ donne une pomme$_{+ACC}$ à sa mère$_{+DAT}$.

b. Elle$_{+NOM}$ la$_{+ACC}$ lui$_{+DAT}$ donne.

c. Marie$_{+NOM}$ gibt ihrer Muter$_{+DAT}$ einen Apfel$_{+ACC}$.

Le cas inhérent fait partie de l'information lexicale, le cas structurel est purement configurationnel. Le cas inhérent domine le cas structurel : si un cas inhérent a été assigné en premier, il ne peut y avoir assignation d'un autre cas. Par contre, si un cas structurel a été assigné, un cas inhérent peut le remplacer. L'assignation de cas structurel se fait sous gouvernement de tête (Condition d'adjacence), l'assignation de cas inhérent se fait sous théta-gouvernement (Condition de Cas inhérent). Lorsqu'un NP a reçu un cas, il est rendu visible (Hypothèse de Visibilité, Chomsky 1981).

Le filtre du cas est donné dans la définition 17.

Définition 17 Filtre du cas *A chaque NP réalisé lexicalement est assigné un et un seul cas abstrait.*

Pour le remplir, les NP qui ne se trouvent pas dans une position où un cas peut être assigné, peuvent se déplacer dans une position libre. C'est typiquement le cas du sujet, qui se déplace de [Spec, VP], qui n'est pas une position de cas structurel, en [Spec,IP] (12a) où il reçoit le cas nominatif de Infl, ou le complément d'objet direct qui se déplace aussi en position [spec,IP] dans les constructions passives (12b) et à montée car le verbe ne peut plus assigner le cas structurel (Généralisation de Burzio, 1986). Le filtre du cas s'applique donc aux chaînes dont la tête est un NP. Une condition de bonne formation des chaînes est que chaque chaîne NP comporte un et un seul cas. Puisque les éléments d'une chaîne sont liés, les éléments-qu reçoivent leur cas par l'intermédiaire de leur trace (12c).

(12)a. [IP Jean$_{i+NOM}$ a [VP t_i mangé une pomme$_{+ACC}$]].

b. [IP Jean$_{i+NOM}$ a [VP été arrêté t_i]].

c. [CP Qui$_i$ as- tu$_{+NOM}$ [VP appelé t_{i+ACC}]]?

Le cas de l'inversion sujet-auxiliaire possible en anglais (13a), mais pas en français (13b) est une propriété de l'anglais (et d'autres langues) qui donne à C le statut de gouverneur lorsqu'il contient les traits d'inflexion du verbe :

(13)a. [CP Has$_i$ [IP John t_i [VP eaten]]]?

b. *[CP A$_i$ [IP Jean t_i [VP mangé]]]?

Classes DP et CP : complément

La classe DP contient un attribut supplémentaire, *cas*, qui prend ses valeurs dans l'ensemble (*nominatif*, *accusatif*, *datif*, *génitif*, *noCase*). Les opérations associées sont *FixerCas (cas)*, pour assigner un cas, et *QuelEstCas (→ cas)* pour retrouver la valeur du cas assigné au DP. Si la valeur de cas retournée par *QuelEstCas* est *noCase*, le DP ne passe pas le filtre du cas. Ce filtre est une opération individuelle, sur chaque DP, ou collective, sur tous les DP de la phrase. Dans ce cas, le filtre est associé à la classe CP, au moyen de l'opération *FiltrerCas (→ résultat, → dp)*, qui retourne dans *résultat* le résultat du filtrage (*vrai*, tous les DP ont un cas, *faux*, il existe au moins un DP qui n'en possède pas) et dans *dp* le premier DP qui ne passe pas le filtre.

Classe Tête : complément

L'assignation de cas structurel concerne les classes I (I assigne le cas nominatif à son spécifieur), et V (V assigne le cas accusatif à son complément, et dans certains cas, au spécifieur de son complément-IP, opération connue sous le nom de Marquage de Cas Exceptionnel). Les particularités des langues concernant l'assignation de cas depuis d'autres têtes (C en anglais) résulte dans la définition de cette opération pour d'autres classes de ces langues. L'assignation de cas se fait par l'opération *AssigneCasStructurel (cas)* de la classe Tête, opération abstraite qui est redéfinie pour tous les assigneurs de cas de la langue, typiquement en français I, V, en anglais I, V et C.

D'être assigneur de cas inhérent est une propriété des objets de la classe ItemLexical, propriété qui est passée à la Tête correspondante. L'assignation se fait par l'opération *AssigneCasInhérent (cas)* de la classe Tête, aussi une opération abstraite redéfinie pour tous les assigneurs concernés, typiquement P, V en français, et P, V et N en anglais.

4.3.6 Théorie Thématique

La théorie thématique règle l'assignation des rôles thématiques aux arguments. On a vu que le principe de projection construit la structure sur la base de la grille thématique des prédicats, et que les rôles thématiques sont assignés aux constituants se trouvant dans des positions argumentales. Dans l'exemple (14), le verbe "admirer" assigne le rôle thématique "Thème" à son complément "les étoiles", la préposition "avec" assigne le rôle thématique "Instrument" à son complément "un téléscope".

(14) J'ai [VP [VP admiré les étoiles$_{+THEME}$] [PP avec un téléscope$_{+INSTRUMENT}$]].

Le critère thématique dit que chaque argument doit porter un et un seul rôle thématique et chaque rôle thématique doit être assigné à un et un seul argument. Mais selon la Condition de Visibilité, un NP ne peut recevoir un rôle thématique que s'il porte un cas, un argument qui ne se trouve pas en position de cas doit se déplacer, laissant une trace en position argumentale, trace à laquelle est associé le rôle thématique. Le critère thématique peut être formulé comme une condition de bonne formation sur les chaînes-A (Rizzi 1986) :

Définition 18 Critère thématique

Une chaîne-A doit porter un et un seul rôle thématique, et chaque rôle thématique doit être assigné à une et une seule chaîne-A.

Un verbe peut perdre la possibilité d'assigner un rôle thématique au sujet s'il ne peut pas assigner le cas accusatif à son complément, tels que verbes passifs et à montée (Généralisation de Burzio, 1986).

Classes DP, PRO, pro et CP : complément

Les classes DP , pro et PRO contiennent un attribut supplémentaire, *rôle*, qui prend ses valeurs dans l'ensemble (*agent*, *expérient*, *thème*, *bénéficiaire*, *but*, *instrument*, *noRôle*). Les opérations associées sont *FixerRôle (rôle)*, pour assigner un rôle thématique, et *QuelEstRôle (→ rôle)* pour retrouver la valeur du rôle assigné au DP. Si la valeur de rôle retournée par *QuelEstRôle* est *noRôle*, le DP ne passe pas le critère thématique. Ce filtre est une opération individuelle, sur chaque DP, ou collective, sur tous les DP de la phrase. Dans ce cas, le filtre est associé à la classe CP, au moyen de l'opération FiltrerArguments (→ résultat, → dp), qui retourne dans *résultat* le résultat du filtrage (*vrai*, tous les DP arguments ont un rôle thématique, *faux*, il existe au moins un DP qui n'en possède pas) et dans *dp* le premier DP argument qui n'a pas de rôle thématique.

Classe Tête : complément

D'être assigneur de rôle thématique est une propriété des objets de la classe ItemLexical, propriété qui est passée à la Tête correspondante. L'assignation se fait par l'opération *AssignerRôle (rôle)* de la classe Tête, une opération abstraite redéfinie pour tous les assigneurs concernés, typiquement P, V et Adj. L'assignation se fait à un *syntagme*, car les NP, pro et PRO reçoivent un rôle thématique, mais il n'est pas nécessaire de spécifier le syntagme sur lequel porte l'assignation car il dépend du rôle. Chaque assigneur doit en outre décharger tous ses rôles thématiques. Ce contrôle est effectué par l'opération *FiltrerThéta (→ résultat, → rôle)* de la classe Tête concernée, qui retourne dans *résultat* le résultat du filtrage (*vrai*, tous les rôles ont été assignés, *faux*, il existe un rôle non assigné) et dans *rôle* le premier rôle qui n'a pas été assigné.

4.3.7 La théorie du Liage

La théorie du liage règle l'interprétation référentielle des syntagmes nominaux dans la phrase. Certaines expressions sont référentielles car elles correspondent à un objet du monde (par exemple Jean, un oiseau). Par contre, d'autres expressions n'ont de sens que par rapport à d'autres expressions (il, lui-même). Les NP sont classés selon les deux traits [anaphore, pronominal]. Selon la valeur de ces traits, on dénombre les anaphores, [+ anaphore, - pronominal], les pronoms, [- anaphore, + pronominal] et les expressions référentielles, [- anaphore, - pronominal] pour les NP réalisés lexicalement. Le quatrième ensemble de propriétés, [+ anaphore, + pronominal] détermine PRO, développé au paragraphe 4.3.8.

Les anaphores sont soumises au Principe A du Liage (19a), les pronoms au Principe B (19b) et les expressions référentielles au Principe C (19c) :

Définition 19 Principes du Liage

1. *Principe A : une anaphore (réfléchi, réciproque) est liée dans sa catégorie gouvernante.*

2. *Principe B : un pronom est libre dans sa catégorie gouvernante.*

3. *Principe C : une expression référentielle est libre partout.*

La relation de liage est définie comme une relation de coindexation entre un élément A et un élément B qu'il c-commande. La catégorie gouvernante pour un élément X est le domaine minimal contenant X, son gouverneur et un sujet/SUJET accessible. Un sujet est tout NP en position de spécifieur, tandis que le SUJET est la tête I, qui contient les traits d'accord. Un sujet/SUJET est dit accessible si sa coindexation avec X ne viole aucun principe grammatical.

Classes Référence, Anaphore, Pronom et DP : complément

La classe DP est dérivée en trois sous-classes selon leur type : Référence, Anaphore et Pronom. Les différents objets de cette classe sont générés sur la base des traits lexicaux, lors de la projection du DP à partir d'un ItemLexical. Les principes du Liage sont définit dans les opérations *Lier (→ résultat, → référence)* de chaque sous-classes, qui retourne le résultat de l'application du principe, et la référence trouvée. Typiquement, pour une anaphore, référence doit correspondre à un DP dans la catégorie gouvernante, pour un pronom, un DP en dehors de la catégorie gouvernante, et pour une expression référentielle, un objet nul. Les trois opérations sont des redéfinitions de l'opération abstraite *Lier* de la classe de base DP.

4.3.8 La théorie du Contrôle

La théorie de Contrôle règle la distribution et l'interprétation de PRO, qui est un DP non réalisé lexicalement qui apparaît comme sujet des clauses infinitives, pour satisfaire le Principe de Projection Etendu. Ses traits sont [+anaphore, +pronominal], il doit donc échapper au gouvernement, selon le théorème de PRO donné en (20), car sinon il devrait correspondre en même temps au Principe A et au Principe B du Liage :

Définition 20 Théorème de PRO

PRO doit être non gouverné.

Le contrôle désigne une relation de dépendance référentielle entre un sujet non exprimé (l'élément contrôlé) et un constituant exprimé ou non (le contrôleur), qui lui passe ses propriétés référentielles. Le contrôle peut être obligatoire, le contrôleur est dans ce cas un argument sujet (15a) ou objet (15b) qui le c-commande, ou optionnel (15c), auquel cas on parle de contrôle arbitraire. Les schémas de contrôle obligatoire dépendent des langues, ils sont spécifiés dans l'entrée lexicale des verbes.

(15)a. Jean$_i$ promet à Marie de [IP PRO$_i$ partir].

Figure 10: Hiérarchie des classes de l'information structurale

b. Jean dit à Marie$_i$ de [IP PRO$_i$ partir].

c. PRO partir en voyage est agréable.

Classe PRO : complément

Pour que PRO ne soit pas gouverné, il faut tester que l'opération *Gouverne* retourne comme résultat *nonApplicable*, pour toutes les têtes précédant la position d'insertion de PRO. La classe *PRO* contient l'attribut supplémentaire, *contrôleur*, et les opérations associées *FixerContrôleur (argument)* et *QuelEstContrôleur (→ argument)*, *argument* pouvant être un élément nul dans le cas de contrôle arbitraire.

Classe V : complément

La classe V contient un attribut supplémentaire, *contrôle*, qui peut prendre ses valeurs dans (*sujet*, *objet*, *arbitraire*) pour tenir compte des différents schémas de contrôle des verbes. Les opérations *FixerContrôle (contrôle)* et *QuelEstContrôle (→ contrôle)* permettent d'assigner une valeur et de récupérer la valeur de l'attribut respectivement.

Figure 11: Relations entre les classes

4.3.9 Classes de l'information structurale

Aux paragraphes précédents, nous avons extrait les classes des principes. Dans ce paragraphe, nous résumons les attributs et opérations pour chaque classe. Les informations extraites sont illustrées à l'aide des diagrammes objets des figures 10, pour la hiérarchie des classes, et 11 pour les relations entre les différentes classes.

Classes de compatibilité : Noeud, Syntagme et CatégorieX^0

Les classes de compatibilité sont définies dans un but de généralisation.

Les relations de gouvernement impliquent des projections maximales et des têtes, vides ou pleines : la classe Noeud est la classe de base compatible avec tous les noeuds de la structure. Elle comporte les opérations suivantes :

- Domine (noeud, → résultat)
- C-commande (noeud, → résultat)
- M-commande (noeud, → résultat)
- AntécédentGouverne (noeud, → résultat)

La compatibilité des catégories vides pro et PRO avec les projections maximales implique la classe de base abstraite Syntagme.

La compatibilité des catégories vides de type X^0 avec les têtes implique la classe de base abstraite CatégorieX^0.

ProjectionMaximale (sous-classe de Syntagme)

- spec, compl, ajoutsGauche, ajoutsDroite, tête,
- headFirst, specFirst
- Projeter (→ projMax)
- SetAjout (projMax, sens)
- GetAjout (sens,→projMax, fin)
- AntécédentGouverne (noeud, → résultat)
- EstBarrière (noeud, → résultat)
- Ecrire (→ chaîne)

VP (sous-classe de PM)

- pro-drop
- ExisteObjet (→ résultat)

IP (sous-classe de PM)

- contraction, proDrop
- ExisteSujet (→ résultat)

CP (sous-classe de PM)

- ContracterComplémenteur
- FiltrerArguments (→ résultat, → dp)
- FiltrerCas (→ résultat, → dp)

PP (sous-classe de PM)

- prepStrand

DP (sous-classe de PM)

- cas, rôle
- GetCas (→ cas)
- GetRôle (→ rôle)
- Lier (→ résultat, → référence)

Référence, Anaphore, Pronom (sous-classes de DP)

NP, AdjP, AP, ConjP, FP (sous-classes de PM)

pro (sous-classe de Syntagme)

- genre, nombre, personne, rôle
- GetRôle (→ rôle)

PRO (sous-classe de Syntagme)

- contrôleur, rôle
- GetRôle (→ rôle)
- EstGouverné (→ résultat)
- GetControleur (verbe, → contrôleur)

TraceMax (sous-classe de Syntagme)

- indice, catégorie, type

OP (sous-classe de Syntagme)

- indice

Tête (sous-classe de CatégorieX0)

- itemLexical, traits
- AntécédentGouverne (noeud, → résultat)
- Gouverne (noeud, → résultat)
- AssigneCasStructurel (cas)
- AssigneRôleThématique (role, syntagme, → résultat)
- FiltrerThéta (→ résultat, → rôle)
- AssigneCasInherent (cas)

V (sous-classe de Tête)

- contrôle

N, Adj, Prep, D, Conj, A, I, F et C (sous-classes de Tête)

TraceX^0 (sous-classe de CatégorieX^0)

- indice, catégorie

Chaîne

- indice, séquence
- Insère (Noeud, position)

ChaîneMax, ChaîneX^0 (sous-classes de Chaîne)

ChaîneA (sous-classe de ChaîneMax)

ChaîneAbar (sous-classe de ChaîneMax)

- Composition (chaîneAbar)

Séquence

5 La grammaire en action

Après avoir modélisé la grammaire universelle, voyons maintenant comment elle est utilisée dans les algorithmes du processus de traduction. L'algorithme de traduction est exposé birèvement à la section 5.1 pour donner un contexte d'utilisation. Une première ébauche de l'algorithme d'analyse est présentée à la section 5.2, illustrée par deux exemples en français.

5.1 Algorithme de traduction

Admettons pour le moment que le texte à traduire est constitué d'une seule phrase en langue source, P_S, qui génère un objet de type Texte (PS). Traduire la phrase P_S signifie lancer le message Texte (PS).*Traduire* ($\rightarrow$ Structure (S_{SC}), $\rightarrow$ Texte (P_C)), qui retourne un objet de type Texte (P_C), dont le contenu est la phrase traduite en langue cible, P_C, et un objet de type Structure (S_{SC}), dont le contenu est la structure de surface cible S_{SC}[26] Dans l'algorithme de traduction (16), S_{SS} est la structure-S source, S_{DC} la structure-D cible, la notation C.M signifie envoyer le message M à la classe C et le symbole $\rightarrow$ signifie que l'objet qui le suit est retourné en réponse au message.

(16) **Algorithme de traduction**

1. Texte (P_S).Analyser ($\rightarrow$ Structure (S_{SS}))
2. Structure (S_{SS}).Transférer ($\rightarrow$ Structure (S_{DC}))
3. Structure (S_{DC}).Générer ($\rightarrow$ Structure (S_{SC}), $\rightarrow$ Texte (P_C))

En exemple, prenons le français comme langue source et l'anglais comme la langue cible. Soit la phrase à traduire Le chat mange la souris, alors les variables de l'algorithme précédent peuvent être instanciées avec les valeurs données en (17).

(17) **Exemple de traduction**

P_S = Le chat mange la souris

S_{SS} = [CP [IP [DP le [NP chat]]$_j$ mange$_i$ [VP [DP e]$_j$ e$_i$ [DP la [NP souris]]]]]

S_{DC} = [CP [IP [VP [DP the [NP cat]] eat$_{+présentprogressif}$ [DP the [NP mouse]]]]]

S_{SC} = [CP [IP [DP the [NP cat]]$_i$ is [VP [DP e]$_i$ eating [DP the [NP mouse]]]]]

P_C = The cat is eating the mouse.

5.2 Analyse

L'analyse d'une phrase se déroule plus ou moins selon le modèle de l'analyseur décrit dans Laenzlinger & Wehrli (1991). Nous n'entrerons pas dans tous les détails, en particulier nous ne décrirons qu'une seule analyse, laissant de côté les analyses parasites générées par la stratégie d'analyse.

L'algorithme d'analyse (16) peut être décrit de la manière suivante : pour chaque nouveau mot, l'information lexicale est projetée dans une structure syntaxique, qui est combinée avec son contexte gauche. La projection de

[26]. Nous admettrons que la phrase source n'est pas ambigüe, donc qu'elle ne retourne qu'une seule traduction.

l'information lexicale peut donner lieu à une projection inhérente, c'est-à-dire qu'une projection maximale devient complément d'une nouvelle projection maximale. Combiner signifie soit attacher le constituant courant comme complément du constituant précédent, soit attacher le constituant précédent comme spécificateur du nouveau constituant, soit encore comme ajout à droite ou à gauche. Lorsqu'il n'y a plus de nouveau mot, on applique le filtre du cas et le critère thématique. Les bonnes analyses sont celles qui passent les deux filtres, c'est-à-dire pour lesquelles *résultat* = *ok* et $\underline{\text{D}}\text{P} = \underline{\text{N}}\text{IL}$. Finalement, les principes du Liage sont appliqués à chaque DP de la structure.

(18) **Algorithme d'analyse**

1. Pour chaque mot M_i, $i = \{1, \ldots n\}$
   (a) $\underline{\text{I}}$temLexical (M_i).Projeter ($\rightarrow \underline{\text{P}}\text{M}$ (M_i))
   (b) Pour chaque constituant précédent C_j ($\text{M}_1..\text{M}_{n-1}$), $j = 1, ..m$
      i. $\underline{\text{P}}\text{M}$ (C_j).SetCompl ($\underline{\text{P}}\text{M}$ (M_i))
      ii. $\underline{\text{P}}\text{M}$ (M_i).SetSpec ($\underline{\text{P}}\text{M}$ (C_ij))
      iii. $\underline{\text{P}}\text{M}$ (C_j).SetAjoutDroite ($\underline{\text{P}}\text{M}$ (M_i))
      iv. $\underline{\text{P}}\text{M}$ (C_j).SetAjoutGauche ($\underline{\text{P}}\text{M}$ (M_i))
2. Pour chaque $\underline{\text{C}}$P complet, i.e. comprenant les mots M_1 à M_n
   (a) $\underline{\text{C}}\text{P}$ ($\text{M}_1..\text{M}_n$).FiltrerCas ($\rightarrow$ résultat, $\rightarrow \underline{\text{D}}\text{P}$ ())
   (b) $\underline{\text{C}}\text{P}$ ($\text{M}_1..\text{M}_n$).FiltrerArguments ($\rightarrow$ résultat, $\rightarrow \underline{\text{D}}\text{P}$ ())
3. Pour chaque $\underline{\text{D}}$P, $\underline{\text{D}}$P.Lier ($\rightarrow$ résultat , $\rightarrow$ référence)

5.2.1 Un exemple simple

L'algorithme d'analyse est appliqué à la phrase simple (19a), pour donner la structure (19b). Le déroulement des opérations est donné en 20.

(19)a. Le chat mange la souris.

b. [CP [IP [DP Le [NP chat]]$_j$ mange$_i$ [VP [DP e]$_j$ e$_i$ [DP la [NP souris]]]]]

(20) Un exemple d'analyse simple

1. $\underline{\text{M}}$Det (le).Projeter ($\rightarrow \underline{\text{D}}\text{P}$ (le))
2. $\underline{\text{M}}$Nom (chat).Projeter ($\rightarrow \underline{\text{N}}\text{P}$ (chat))
3. $\underline{\text{D}}\text{P}$ (le).SetCompl ($\underline{\text{N}}\text{P}$ (chat)) $\Rightarrow \underline{\text{D}}\text{P}$ (le chat)
4. $\underline{\text{M}}$Verbe (mange).Projeter ($\rightarrow \underline{\text{V}}\text{P}$ (mange))
   (a) Un objet de type $\underline{\text{V}}$P provoque une projection inhérente de type $\underline{\text{I}}$P :
      - $\underline{\text{V}}\text{P}$ (mange).Projeter ($\rightarrow \underline{\text{I}}\text{P}$ (mange))

   Les opérations de projection inhérente sont :

i. création de l'objet IP
ii. IP ().SetCompl (VP (mange))
iii. IP ().SetTêteLex (mange) ⇒ IP (mange)

(b) Un objet de type IP provoque la création d'une chaîneX^0 de type v :
i. création de la chaîne
ii. ChaîneX0 (i,v).Insère (I (mange),tête)
iii. ChaîneX0 (i,v).Insère (TraceX^0 (i,v),pied)
iv. VP (mange).SetTêteLex (TraceX^0 (i,v)) ⇒ VP (e)

(c) Un objet de type IP provoque la projection inhérente d'un objet de type CP :
- IP (mange).Projeter (→ CP ()) avec CP ().compl := IP (mange)

(d) IP (mange).SetSpec (DP (le chat)) Assigner à I un spécificateur de type DP provoque les opérations suivantes :
i. Création d'une chaîne argumentale entre [spec,IP] et [spec,VP] :
A. ChaîneA (j,dp).Insère (DP (le chat),tête)
B. VP (e).SetSpec (TraceMax (j, dp, np))
C. ChaîneA (j,dp).Insère (TraceMax (j,dp,np),pied)
ii. Assignation du cas structurel par I :
- I.AssignerCasStructurel (nominatif) ⇒ DP (le chat).cas := nominatif

(e) Assignation par V du rôle thématique agent à son spécificateur, par l'intermédiaire de la trace
- V.AssignerRôle (agent) ⇒ DP (le chat).rôle := agent

5. MDet (la).Projeter (→ DP (la))
6. VP (e).SetCompl (DP (la))
Assigner un complément à V provoque les opérations suivantes :
(a) Assignation du cas structurel par V à son complément :
- V (e).AssignerCasStructurel (accusatif) ⇒ DP (la).cas := accusatif

(b) Assignation du rôle thématique thème par V à son complément
- V.AssignerRôleThématique (thème) ⇒ DP (la).rôle := thème

7. MNom (souris).Projeter (→ DP (souris))
8. DP (la).SetCompl (NP (souris)) ⇒ DP (la souris)
9. CP (le chat mange la souris).FiltrerCas ⇒ (ok, NIL)
10. CP (le chat mange la souris).FiltrerArguments ⇒ (ok, NIL)
11. DP (le chat).Lier (→ ok, →NIL)
12. DP (la souris).Lier (→ ok, → NIL)

5.2.2 Un exemple plus complexe

La phrase (21a), dont la structure est donnée en (21b) est plus complexe, car elle contient à la fois un passif et un verbe à montée du sujet. Le déroulement des opérations est donné en 22.

(21)a. Jean semble avoir été invité à la fête.

b. [CP [IP [DP [NP Jean]]$_j$ semble$_i$ [VP e$_i$ [CP [IP [DP e]$_j$ [VP avoir [VP été [VP [VP invité [DP e]$_j$] [PP à [DP la [NP fête]]]]]]]]]]

(22) Un exemple d'analyse plus compliqué

1. MNom (Jean).Projeter (→ NP (Jean))
   Un objet de type NP avec le trait "nom propre" provoque la projection inhérente d'un objet de type DP :
   - NP (Jean).Projeter (→ DP (Jean))
2. MVerbe (semble).Projeter (→ VP (semble))

   (a) Un objet de type VP provoque une projection inhérente de type IP (les opérations sont les mêmes que pour l'exemple précédent) :
   - VP (semble).Projeter (→ IP (semble))

   (b) Un objet de type IP provoque la création d'une ChaîneX^0 de type v :
   i. création de la chaîne
   ii. ChaîneX^0 (i,v).Insère (I (semble),tête)
   iii. ChaîneX^0 (i,v).Insère (TraceX^0 (i,v),pied)
   iv. VP (semble).SetTêteLex (TraceX^0 (i,v)) ⇒ VP (e)

   (c) Un objet de type IP provoque la projection inhérente d'un objet de type CP :
   - IP (semble).Projeter (→ CP ()) avec CP ().compl := IP (semble)
3. IP (semble).SetSpec (DP (Jean))
   Assigner à I un spécificateur de type DP provoque les opérations suivantes :

   (a) création d'une chaîne argumentale avec Jean comme tête, mais comme semble est un verbe à montée du sujet, il n'assigne pas de rôle thématique à un DP, donc [spec,VP] est vide, donc il faut attendre d'avoir un verbe avec un rôle thématique à décharger pour trouver le pied de la chaîne.
   - ChaîneA (j,dp).Insère (DP (Jean),tête)

   (b) Assignation du cas structurel par I :
   - I.AssignerCasStructurel (nominatif) ⇒ DP (Jean).cas := nominatif

4. MVerbe (avoir).Projeter (→ VP (avoir))

   (a) un objet de type V provoque une projection inhérente de type I, mais comme le verbe est non tensé, il n'y a pas mouvement de tête :

   - VP (semble).Projeter (→ IP ())

   (b) Un objet de type IP provoque la projection inhérente d'un objet de type CP :
   - IP ().Projeter (→ CP ()) avec CP ().compl := IP ()

5. VP (semble).SetCompl (CP ())
   comme semble est un verbe à montée du sujet, [spec,IP] est forcément une trace intermédiaire :
   - ChaîneA (j,dp).Insère (TraceMax (j,dp,np), suivant)
6. MVerbe (été).Projeter (→ VP (été))
7. VP (avoir).SetCompl (VP (été))
8. MVerbe (invité).Projeter (→ VP (invité))
9. VP (été).SetCompl (VP (invité))
   attacher le VP comme complément de l'auxiliaire être montre qu'on se trouve en présence d'un passif. Il n'y a donc pas de rôle thématique agent (caractérisé par un [spec,VP] inexistant). Le verbe perdant aussi la propriété d'assigner le rôle structurel accusatif à son complément, celui-ci doit se déplacer pour des raisons de cas et laisse une trace en [compl,VP]. Le complément s'est déplacé en [spec,IP] de la phrase enchâssée, mais comme le verbe principal est à montée du sujet, il y a laissé une trace intermédiaire lorsqu'il s'est déplacé en [spec,IP (semble)]. La trace en [compl,VP] est donc le pied de la chaîne j et sert d'intermédiaire à l'assignation du rôle thématique thème à Jean.
   (a) ChaîneA (j,dp).Insère (TraceMax (j,dp,np),pied)
   (b) VP (invité).SetCompl (TraceMax (j,dp,np))
   (c) V.AssignerRôleThématique (thème) ⇒ DP (Jean).rôle := thème
10. MPrep (à).Projeter (→ PP (à))
11. VP (invité).SetAjoutDroite (PP (à))
12. MDet (la).Projeter (→ DP (la))
13. PP (à).SetCompl (DP (la)) ⇒ PP (à la)
14. MNom (fête).Projeter (→ NP (fête))
15. DP (la).SetCompl (NP (fête)) ⇒ DP (la fête) ⇒ PP (à la fête)
16. CP (Jean a été invité à la fête).FiltrerCas ⇒ (ok, NIL)
17. CP (Jean a été invité à la fête).FiltrerArguments ⇒ (ok, NIL)

18. DP (Jean).Lier (→ ok, →NIL)
19. DP (la fête).Lier (→ ok, → NIL)

6 Conclusion

Nous avons proposé dans ce travail une modélisation orientée-objets de la grammaire universelle décrite par la théorie GB, utilisable dans un système de traduction multilingue. Le paradigme objet, grâce à l'héritage, aux messages et au polymorphisme, semble bien adapté à la conception d'une grammaire universelle, paramétrisable et modulaire.

Sur le plan linguistique, on a montré que la GU peut être modularisée, non pas dans le sens d'un module par sous-théorie, mais d'une classe par objet linguistique - catégories syntaxiques (lexicales et fonctionnelles) et chaînes. Les principes des différents modules définissent des opérations des classes sur lesquelles ils s'appliquent. Puisque ce sont des principes universels, ils s'appliquent de la même façon dans toutes les langues, du moins sur un sous-ensemble de langues partageant les mêmes propriétés. Les grammaires spécifiques à chaque langue sont des instances de ces classes, mais lorsque des paramètres modifient les principes, ou lorsqu'il existe des constructions spécifiques à une langue donnée, les classes de base sont étendues dans des sous-classes spécialisées, qui héritent de tous les attributs et opérations des classes de la GU. Ce sont les méthodes - l'implémentation des opérations dans les classes - qui sont modifiées, non pas l'interface de la classe. Ainsi, tous les objets répondent aux mêmes messages.

Sur le plan informatique, les processus du système, tels l'analyse, le transfert et la génération, ne donnent pas lieu à des classes, mais à des opérations des classes représentant les données traitées par les processus, l'approche orientée-objets étant basée sur les données présentes dans le système, et non pas sur les fonctionnalités comme l'approche structurée traditionnelle. La grammaire étant utilisée par tous les processus, concevoir une grammaire orientée-objets permet de résoudre les problèmes de génie logiciel tels que maintenance difficile et redondance du code.

Une implémentation orientée-objets semble possible après une phrase de conception détaillée, puisque le premier pas a été fait en extrayant les classes, avec leurs attributs et opérations, l'adéquation de cette modélisation ayant été démontrée par l'ébauche d'un algorithme d'analyse, illustrée avec deux exemples du français.

Bibliographie

Abbot R., 1983. Program Design by Informal English Descriptions. *CACM*, vol. 26, no. 11, novembre 1983, pp. 882-894.

Abney S., 1987. "The English noun phrase in its sentential form". Thèse non publiée. MIT.

Baker M., 1988. *Incorporation: A Theory of Grammatical Function Changing.*University of Chicago Press.

Becker T., Joshi A. K., Rambow O., 1991. Long-Distance Scrambling and Tree Adjoining Grammar. *Actes de l'EACL*, pp 21-26.

Belletti,A. & Rizzi L., 1988. Psych-verbs and q-Theory. *Natural Language and Linguistic Theory 6*, 291-352.

Berrendonner A., Fredj M., Oquendo F., Rouault J., 1992. Un système inférentiel orienté objet pour des applications en langues naturelles. *Actes de Coling*, pp 461-467.

Berwick R., Abney S., Tenny C. éditeurs, 1991. *Principle-Based Parsing: Computation and Psycholinguistics.* Kluwer Academic Press.

Boitet C., 1993. La TAO comme technologie scientifique : la traduction automatique fondée sur le dialogue, in *La traductique.* Pierrette Bouillon, André Clas (eds), Presses de l'Université de Montréal, pp 109-148.

Booch G., 1991. *Object-Oriented Design with Applications.* Benjamin/Cummings.

Böttcher M., 1993. Disjunctions and inheritance in the Context Feature Structure System. *Actes de l'EACL*, pp 54-60.

Burzio L., 1986. *Italian Syntax. A government and Binding Approach.* Reidel, Dordrecht.

Carpenter B., Pollard C., 1991. Inclusion, disjointness and choice: the logic of linguistic classification. *Actes de ACL*, pp 9-16.

Carrier-Duncan J., 1985. Linking of Thematic Roles in Derivational Word Formation. *Linguistic Inquiry 16*, 1-34.

Chomsky N., 1981. *Lectures on Government and Binding.* Foris Publications.

Chomsky N., 1986a. *Knowledge of Language : Its nature, Origin and Use.* Praeger.

Chomsky N., 1986b. *Barriers.* Linguistics Inquiry Monograph. MIT.

Coad P., Yourdon E., 1991. *Object-Oriented Design.* Prentice-Hall.

Coad P., Yourdon E., 1990. *Object-Oriented Analysis.* Prentice-Hall.

Colbert E., 1989. *The object Oriented Software Development Method : A practical Approach to Object Oriented Developement.* Absolute Software Co., Inc.

Daelemans W., Gazdar G. (eds), 1990. *Inheritance in Natural Language Processing.* Université de Tilburg.

Door B., 1990. "Lexical Conceptual Structure and Machine Translation". Thèse de doctorat. MIT.

Emonds J., 1976. *A transformational approach to English Syntax.* Academic Press.

Evans R., Gazdar G., Weir D., 1995. Encoding Lexicalized Tree Adjoining Grammars with a non monotonic Inheritance Hierarchy. *Actes des ACL*, pp 77-84.

Fong S., 1991. "Computational Properties of Principal-Based Grammatical Theories". Thèse de doctorat. MIT.

Grimshaw J., 1990. *Argument Structure.* Linguistic Inquiry Monograph. MIT Press.

Haegeman L., 1994. *Introduction to Government and Binding Theory.* Blackwell.

Jackendoff R., 1972. *Semantic interpretation in Generative Grammar*, MIT Press, Cambridge, Mass.

Kayne R., 1984. *Connectedness and Binary Branching.* Foris Publications.

Koopman H., Sportiche D., 1991. The position of Subjects . *Lingua 85*, 211-258.

Laenzlinger C., 1993. Principles for a formal account of Adverb Syntax. *Gen-GenP*, vol. 1, num. 2. Genève.

Laenzlinger C., Wehrli E., 1991. FIPS : Un analyseur interactif pour le français. *T.A. Informations 2*, pp 35-49.

Larson, R. K, 1988. On the double object construction. *Linguistic Inquiry 19*, 335-391.

Merlo P., 1992. "On modularity and compilation in a Government and Binding Parser". Thèse de doctorat distribuée comme Rapport Technique UMIACS-TR-92-115 CS-TR-2982. Université de Maryland.

Merlo P., 1995. Modularity and Information Content Classes in Principle-Bases Parsers. *Computational Linguistics*, vol. 21, num. 4, pp 515-541.

Mitamura T., Nyberg E. H., 1992. Hierarchical lexical structure and interpretative mapping in machine translation. *Actes de Coling*, pp 1254-1258.

Mössenböck H., 1993. *Object-Oriented Programming in Oberon-2.* Springer-Verlag.

Pinnagoda S., Ramluckun M., 1993. "Transfert et génération de phrases". Notes techniques LATL/Genève 93/16.

Pressman R., 1993. *A manager's guide to software Engineering.* Mc Graw Hill.

Reinhard S., Gibbon D., 1991. Prosodic inheritance and morphological generalisation. *Actes de l'EACL*, pp 131-136.

Rizzi L., 1986. Conditions de bonne formation sur les chaînes. *Modèles Linguistiques 7*, pp 119-157.

Rizzi L., 1990. *Relativized Minimality.* Linguistic Inquiry Monograph. MIT Press.

Rösner D., 1988. The SEMSYN Generation System: ingredients, applications, prospects. *Actes de l'ACL*, pp 25-32.

Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W., 1991. *Object-Oriented Modeling and Design.* Prentice-Hall.

Russel G., Carrol J., Warwick-Amstrong S., 1991. Multiple Default Inheritance in a unification-based Lexicon. *Actes de l'ACL.*

Schlaer S., Mellor S., 1988. *Object-Oriented System Analysis: Modeling the world in data.* Prentice-Hall.

Sodhi J., 1992. *Software requirements, analysis and specifications.* Mc Graw Hill.

Stefanini M.-H., Berrendoner A., Lallich G., Oquendo F., 1992. Talisman : A multi-agent system governed by linguistic laws for natural language processing. *Actes de Coling*, pp 490-497.

Topper A., Ouellette D., Jorgensen P., 1993. *Structured Methods, Merging Models Techniques and Case.* Mc Graw Hill.

Véronis J., 1992. A feature-based model for lexical databases. *Actes de Coling*, pp 588-594.

Walther C., 1994. "Modularity and Parsing". Thèse de doctorat. Université de Genève.

Walther C., 1991. "The lexical database for English. Progress Report". Notes techniques LATL/Genève 91/8.

Walther C., Wehrli E., 1995. A versatile multilingual lexical database. *Actes des IVèmes Journées scientifiques de l'AUPELF-UREF : Lexicomatique et dictionnairiques.* Lyon. 28-30 septembre.

Wehrli E., 1995. "L'information bilingue". Notes techniques LATL/Genève. 95/6.

Wehrli E., 1993a. "Le mécanisme de sélection dans les analyseurs IPS et FIPS". Notes techniques 93/1.

Wehrli E., 1993b. "Description du système ITS-2". Notes techniques LATL/Genève. 93/2.

Wehrli E., Laenzlinger C., 1993c. Théorie linguistique et traitement automatique du langage naturel. in *La traductique*, Pierrette Bouillon & André Clas (eds). Les presses de l'Université de Montréal.

Wehrli E., 1988. Parsing with a GB grammar. in *Natural Language Parsing and Linguistic Theories*. Reyle & Rohrer (éd), pp 177-201.

Weisweber W., 1992. Term rewriting as a basis for a uniform architecture in machine translation. *Actes de Coling*, pp 777-783.

Wirfs-Brock R., Wilkerson B., 1989. Object-Oriented Design: A Responsability-Driven Approach. *Proceedings of the 1989 OOPSLA*.

Wirth N., Reiser M., 1992. *Programming in Oberon. Steps beyond Pascal and Modula*. Addison-Wesley.

Wirth N., 1985. *Programming in Modula-2*. 3ème édition. Springer-Verlag

Zajac R., 1992. Toward computer-aided linguistic engineering. *Actes de Coling*, pp 828-834.

Zampolli A., 1991. Towards Reusable Linguistic Resources. *Actes de l'EACL*.

Table des illustrations

Liste des tables

Table des matières