Chapitre 5

"Au début, on ne peut qu'essayer de nommer les choses, une à une, platement, les énumérer, de la manière la plus banale possible, de la manière la plus précise possible, en essayant de ne rien oublier".
Récits d'Ellis Island de G.Perec avec R. Bobert P.O.L

Classifier en terrain mouvant : Approche prototypique pour la construction du sens de moulin à N2

5.1. "Parti de rien, virgule..."

La linguistique informatique a depuis longtemps[1] adopté les mécanismes basés sur l'héritage (taxonomie de classes) pour la représentation des connaissances. Ce type de mécanisme se révèle efficace pour modéliser des domaines de connaissances stabilisées. Dans les domaines de représentation des connaissances où on ne dispose pas encore de savoirs figés, une approche objet à taxonomie de classes nécessite, dans l'état actuel de nos connaissances sur les méta-représentations ou sur les connaissances capables de créer de nouvelles connaissances, des choix initiaux de représentations qu'il peut être coûteux de modifier : les modifications superficielles pouvant entraîner une refonte complète de l'existant. Si la représentation hiérarchique souvent adoptée en linguistique informatique n'est pas adaptée au problème de représentation des connaissances mouvantes, évolutives ou instables, il est légitime de la remettre en question. L'idée importante dans la notion de prototype est celle de la non-rigidité du système qui utilise ce type d'objets : dans ce cadre les rapports entre objets n'ont aucune part de prédéfinition. Une autre notion importante dans les langages à prototypes (qui rejoint d'ailleurs la précédente) est celle de l'éclatement des connaissances au sein d'une quantité aléatoire d'objets. Ce type de langages permet d'envisager le développement de programmes utilisant aisément des données modulaires.

[1] La dizaine d'années qui nous sépare des textes fondateurs de Gazdar (1985-1990) est une quasi éternité en temps informatique.

5.2. Prototypes et Analyse Automatique

5.2.1. Préliminaires

Le travail présenté ici amorce la mise au point d'une analyse automatique qui utilise les potentialités des langages à prototypes (-> incise 1) : au niveau de la représentation des connaissances de la langue et au niveau des éléments pertinents pour l'analyse. Les choix faits ci-dessous pour la représentation des entités linguistiques pour cette phase expérimentale d'utilisation des prototypes à la Self dans une analyse automatique ne visent absolument pas à donner une réponse ou une illustration de choix théoriques pré-supposés. Il s'agit d'établir tout d'abord un parallèle avec un travail en cours de réalisation dans un modèle objet à taxonomie de classes. Les choix faits s'inspirent d'une part de la tradition (récente) en linguistique de traiter de la typicalité (en privilégiant le champ d'étude autour des propriétés du système lexical) et d'autre part de la prise en compte des travaux existants sur l'analyse automatique dirigée par le lexique (WEP en particulier dont nous reparlerons (-> présentation de WEP)).

5.2.2. Cadre linguistique

Le point d'ancrage linguistique pour l'expérimentation d'une analyse automatique prototypique est celui de l'étude syntaxique des dénominations complexes de la forme `N1 à N2` définie par (Habert & Fabre 1993). Ce travail présente *"une approche hiérarchique de la construction du sens de dénominations complexes de la forme N1 à N2"* qui utilise les mécanismes mis en place par DATR. Le modèle proposé permet de formuler des hypothèses sur l'occupabilité des places argumentales dans le patron `moulin à N2`. Une partition des possibilités peut se matérialiser par les ensembles suivants :

moulin POUR/sortie moulin POUR/entrée moulin AVEC/énergie moulin AVEC/partie-de

L'ordre d'écriture des ensembles ainsi défini de réalisations des structures possibles sur le patron `moulin à N2` correspond à la réalisation quantitative de telles structures : les occurrences attestées et regroupées dans les ensembles ci-dessous diminuent quand on lit les ensembles dans l'ordre donné. Cette analyse permet d'une part d'éliminer des séquences données si celles-ci ne rentrent pas dans le cadre ainsi défini, d'autre part de classer les séquences `moulin à N2` en fonction des compatibilités observées. De plus, on peut ainsi définir des bases pour des heuristiques évaluant la classification des lexèmes à entrées multiples dans le cadre défini par la classification supra. Le travail présenté par (Habert & Fabre 1993) s'inscrit dans un cadre d'analyse tel que la construction du sens s'établit non pas par la juxtaposition et la combinaison des sens des entrées lexicales mais par la construction progressive en précisant pas à pas les hypothèses initiales, quitte à rester dans l'indétermination quand les informations disponibles ne permettent d'aller plus loin. Il s'agit de mettre en place un travail d'affinage des constructions développées par l'analyse qui progresse du plus général vers le particulier. Il s'agit donc aussi d'évaluer en parallèle les résultats obtenus dans une approche de représentation des connaissances qui utilise les notions d'objets et de structuration hiérarchique au sens de la PàO (c'est le cas du travail présenté par (Habert & Fabre 1993)) et les résultats que l'on va construire ici dans une approche de représentation des connaissances avec des prototypes.

5.2.3. Approches pour une Analyse automatique prototypique avec Self

5.2.3.1. Phase0 : Filtrage Catégoriel

Tous les éléments constitutifs du système mis en place pour cette phase introductive à une analyse automatique prototypique ont été définis comme des prototypes suivant les règles de programmation définies par Self 2.0 puis Self 3.0 (-> incises 1, 2, 3). Cette phase introductive peut être considérée comme une phase de familiarisation avec l'environnement informatique proposé par Self via une analyse syntaxique simplifiée à l'extrême.

Existe-t-il des éléments prototypiques dans la langue ?

Il ne s'agit évidemment pas de fournir une réponse définitive à cette question d'entrée de jeu. Cette question n'était d'ailleurs pas l'objet principal de ce travail, même si l'ombre de cette question revient souvent au cours des réflexions sur les développements successifs de ce travail. Le choix fait de donner la primauté à une approche prototypique lexicalisée s'est révélée la plus évidente, mais ce choix n'est évidemment pas une réponse définitive à la question initiale : c'est simplement la vision la plus élémentaire et la plus immédiate qui a permis de démarrer.

Prototype et analyse de mots

Il s'agit dans un premier temps de fournir une chaîne de caractères représentant une dénomination complexe de la forme `N1 à N2` du type `moulin à paroles` et de construire à partir de cette chaîne un objet prototypique associé à celle-ci regroupant en son sein trois prototypes distincts : un prototype de type `N` associé à la chaîne "moulin" , un prototype de type `Prep` associé à la chaîne "à" et un prototype de type `N` associé à la chaîne "paroles".

Représentation du lexique et des catégories grammaticales

La démarche retenue dans un premier temps consiste à associer à chaque entrée lexicale un prototype en utilisant pour cela les catégories grammaticales classiques. Les choix de représentation des connaissances lexicales dans cette phase de démarrage n'ont pas été formalisés de manière définitive. Il s'agit de disposer de représentation minimale qui privilégie l'association d'une entrée lexicale donnée et de la catégorie grammaticale associée à cette entrée lexicale (i.e sans prendre en compte les ambiguïtés possibles) : les catégories retenues ici sont (`Nom, Préposition`). De plus chacun des prototypes construits porte des attributs[2] qui correspondent aux valeurs catégorielles habituellement associées à ces entrées lexicales : `genre`, `nombre`, `semantique` et un attribut `forme` qui portera la valeur de l'entrée lexicale associée au prototype. On associe aux entrées lexicales de type `Nom` un prototype dont la définition reprend celle des noms communs. Le prototype `catN` possède les attributs précités. Il possède aussi un attribut `parent` qui lui permet de déléguer certains comportements (en l'occurrence, le clonage et l'impression des valeurs attribuées aux attributs donnés) au `traits catN`. Ces attributs ont pour valeur des prototypes nommés respectivement `genre, nombre, semantique et forme` (qui délèguent eux aussi les mêmes types de comportement à des `traits` respectivement de même nom). On définit de même les prototypes `catPrep, catNaN`. On donne en annexe (-> annexe partie 2 chapitre 5) le code correspondant à la création des `traits` et `prototypes catN`.

Contrôle et analyse dans un monde de prototypes

Dès cette première phase de notre implémentation, la **notion de contrôle** sur les entités construites s'est révélée cruciale. En effet si l'on veut pouvoir évaluer, tracer et contrôler les activités des prototypes construits, il est impératif de pouvoir les retrouver aisément. La difficulté étant que la PàP tend à définir des unités a priori peu connectées entre elles. La première difficulté a consisté à pouvoir "mettre la main" sur les prototypes ainsi définis. Une première approche de parsage a défini un prototype de parseur chargé de "détecter" parmi tous les prototypes construits dans une session de travail ceux qui possèdent un attribut `forme`.

[2] Ces éléments seront explicités dans les phases suivantes.

Ce prototype de parseur doit vérifier si la valeur associée au champ `forme` du prototype lexical rencontré correspond à la chaîne de caractère donnée en entrée. Il doit ensuite pouvoir "stocker" les prototypes qui répondent à cette recherche. La même chose est ensuite mise au point avec une suite de chaînes de caractères (une phrase). On obtient ainsi en sortie une liste regroupant les entrées de type `string` donnée en entrée puis la liste des vecteurs composés des `miroirs`[3] pointant sur les prototypes associés à chacune des chaînes de caractères lues. La solution retenue pour exercer un premier type de contrôle sur les prototypes construits a donc consisté à utiliser très rapidement des processus de **méta-représentation** pour garder la main sur les prototypes construits et ainsi interroger leur structure. Nous reviendrons infra sur la présentation de tels processus et nous verrons aussi que l'utilisation de ces processus va s'avérer nécessaire tout au long de ce travail d'implémentation.

La deuxième étape a consisté à construire les prototypes capables d'effectuer une analyse sur un morceau de phrase et de produire un résultat plus facilement exploitable. Un prototype `regle` est introduit, il regroupe en fin d'analyse tous les prototypes construits suivant le modèle syntaxique associé à ce prototype (-> annexe partie 2 chapitre 5). Dans notre cadre de travail, le modèle syntaxique retenu peut être représenté par la règle hors contexte : `N1àN2 -> N1 Prep N2 (où N1 = moulin)`. Le démarrage de chaque analyse initialise d'une part les attributs du prototype associé à la règle à reconnaître/construire et d'autre part enregistre la suite de mots à analyser (-> annexe partie 2 chapitre 5).

L'analyse commence et se déroule donc de la manière suivante :

- 1. Lecture de la suite de mots à analyser.
- 2. On prend les entrées lexicales enregistrées une par une et on recherche les prototypes lexicaux associés qui permettent de mettre à jour la règle à construire.

Le déroulement de l'analyse ainsi développée est représenté dans la figure 5.1 :

- 1. Lecture de la suite de mots à analyser de longueur `InputTextSize :=3`.
- 2. Pour chaque mot, recherche de tous les prototypes existants ayant un attribut `forme`, sélection de celui ayant un attribut `forme` dont la valeur est identique au mot examiné.
- 3. Si le prototype retenu (via le prototype `category` contenu dans l'attribut `categorie`) est de même catégorie grammaticale que celle attendue par la règle à construire, on ajoute celui-ci dans la règle via l'attribut `RhsContent` qui contiendra tous les prototypes associés aux mots lus.
- 4. En fin de lecture des entrées lexicales, (5a) on vérifie si les tailles respectives de la règle à construire (`rhs`) et du contenu prototypique de la règle à construire coïncident. Si tel est la cas (5b), on peut construire la partie gauche (`N1 à N2`) : on récupère pour cela, chacun des prototypes reconnus précédemment et on les affecte à leur position respective dans le prototype `N1 à N2` à construire.

[3] La notion de `miroir` fait appel au mécanisme de réflexivité de structure disponible en Self (Cointe & Mulet 1993). Les `miroirs` sont des entités qui rendent visibles au niveau du langage les informations contenues dans la représentation interne des objets. Les `miroirs` sont donc des objets, qui en fait ne contiennent aucune information en eux-mêmes, mais qui permettent de donner accès en lecture aux informations des objets du système via l'utilisation d'un ensemble de primitives prédéfinies.

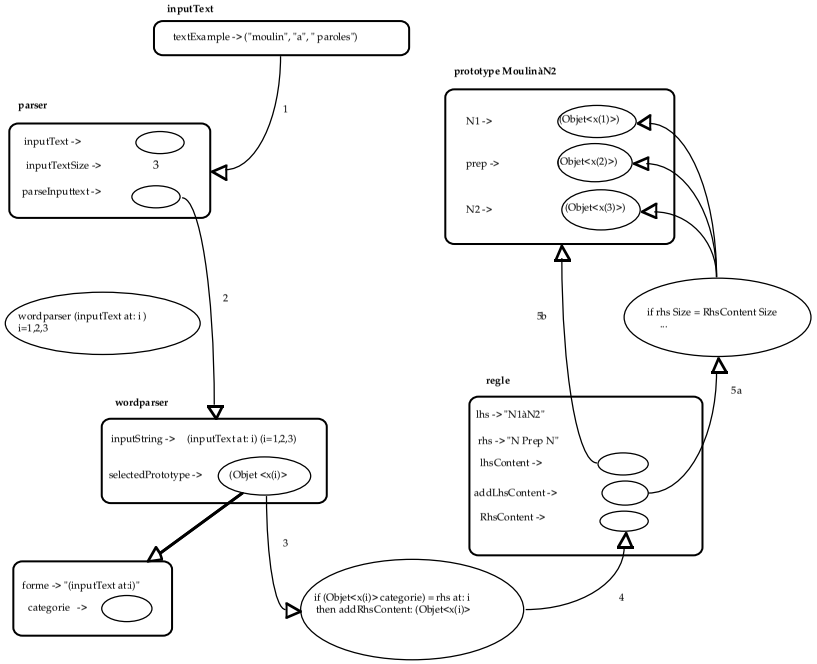

Figure 5.1 : Analyse prototypique Phase0.

L'utilisation de la version Self 3.0 a permis d'utiliser un environnement de programmation avec une interface graphique adéquate pour la mise au point et pour la visualisation des résultats construits par l'analyse. On présente infra (-> annexe partie 2 chapitres 6, 7, 8, 9) une version graphique des résultats obtenus par la suite en utilisant donc une version du code précédent modifié pour tenir compte de cette nouvelle version.

5.2.3.2. Phase1 : Affinement du filtrage

La phase d'expérimentation précédente s'est développée afin d'établir un parallèle avec le travail de (Habert & Fabre 1993). On a retenu le cadre sémantique défini pour la classification des dénominations complexes de la forme `N1 à N2` à savoir la partition des ces formes suivant les quatre classes suivantes :

moulin POUR/sortie moulin POUR/entrée moulins AVEC/énergie moulin AVEC/partie-de

Quatre prototypes sémantiques correspondant chacun à une des classes données ci-dessus sont définis. Ces prototypes comportent un attribut contenant la valeur de la classe à laquelle ils appartiennent sous la forme d'une chaîne de texte. De même, tous les éléments lexicaux de type `Nom` qui peuvent entrer dans le cadre syntaxique `N1 à N2` (en position `N2`) comportent un attribut `semantique` qui recevra comme valeur un des prototypes `semantique` ainsi définis.

La figure suivante reprend la présentation des objets lexicaux faite ci-dessus :

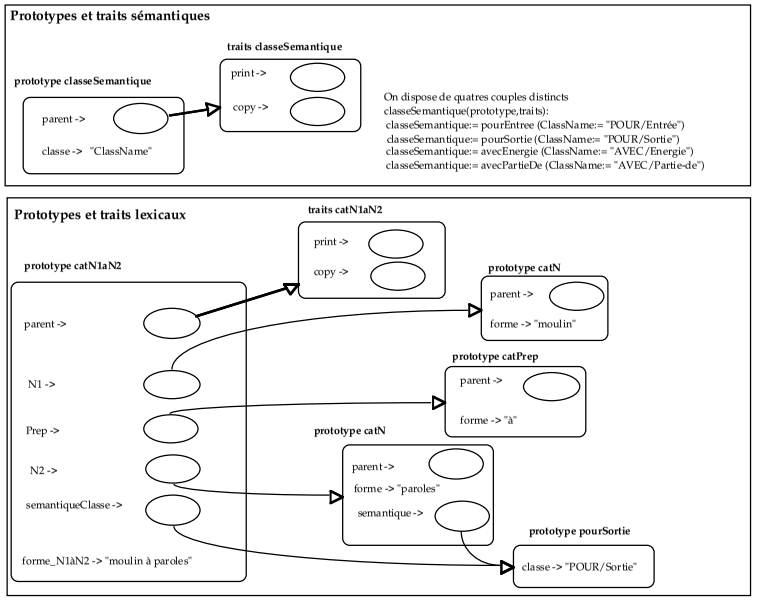

Figure 5.2 : Prototypes et traits pour analyse prototypique Phase1.

Analyse et reconnaissance/construction des prototypes `N1 à N2`

Le mécanisme général de parsage décrit supra est maintenu, seule la règle construite au cours de l'analyse est modifiée pour prendre en compte l'approche sémantique choisie. L'analyse se déroule donc ainsi :

- 1. On lit la forme complexe `N1 à N2` donnée, par exemple `moulin à paroles.`
- 2. Le parseur lit les entrées lexicales une à une, cherche les prototypes lexicaux associés et enregistre les résultats temporaires dans la règle à construire.
- 3. La fin de lecture correspond à la phase de création du prototype `N1 à N2` dont les attributs pointeront sur les différents prototypes reconnus au cours de l'analyse. La classe sémantique du `N1 à N2` construit sera déterminée par la valeur sémantique contenue dans le prototype lexical `N2`, dans notre exemple le prototype `paroles` associé à la chaîne "paroles" possède un attribut `semantique` pointant sur un prototype `semantique` du type `pourSortie` dont l'attribut `classe` donne la valeur de la classe sémantique associée au prototype `N1 à N2` construit à savoir la classe `POUR/Sortie`. De même, l'attribut `forme` du prototype résultant sera construit par la lecture des différents attributs `forme` puis par leur concaténation.

La figure 5.3 illustre toutes ces opérations :

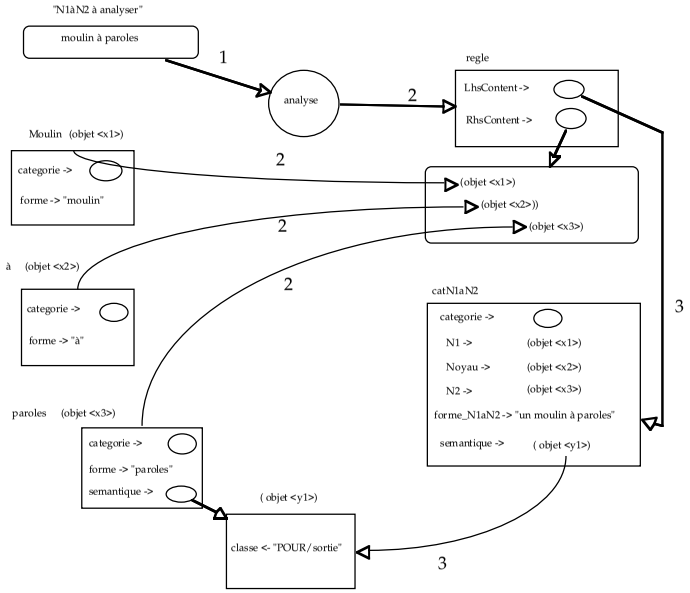

Figure 5.3 : Analyse prototypique Phase1.

L'attribut `semantique` associé aux éléments lexicaux manipulés ici ne concerne que la part de sémantique dont ils sont porteurs dans le cadre défini plus haut : la construction de formes `N1 à N2` suivant les quatre classes sémantiques retenues. Le prototype `N2` est porteur de la valeur sémantique attribuée au prototype `N1 à N2` résultant de l'analyse, et non pas d'une valeur sémantique caractérisant l'entrée lexicale attachée au prototype `N2`. Il est évidemment possible et nécessaire d'enrichir la sémantique de ces éléments par des spécifications supplémentaires : cette phase de travail sera faite infra. Le fait de partager la valeur de l'attribut `semantique` entre l'élément prototypique en position `N2` et le prototype construit `N1 à N2` n'est pas la solution définitive qui sera adoptée : la sémantique du `N1 à N2` sera déterminée par la valeur sémantique attachée au prototype `N2` initial (non pas en termes de résultat pré-construit comme ci-dessus, mais en fonction d'un calcul fondé sur une classification sémantique donnée). D'une manière générale, il faut donner aux objets manipulés dans une analyse automatique la possibilité de leur propre auto-modification en fonction d'évolutions dynamiques de leurs états successifs. L'ajout de spécification sémantique au prototype en position `N2` doit entraîner une spécification du prototype sémantique associé au prototype `N1 à N2`. Dans notre exemple, toute modification des objets `x1, x2, x3` et `y1` et donc de tous les prototypes attachés à ces derniers sera implicitement connue du prototype `N1 à N2` construit. Si ces modifications affectent les valeurs des attributs de celui-ci, il faut mettre en place des processus de réactualisation de ces attributs pour que ces spécialisations deviennent explicites au sein du prototype `N1 à N2`.

5.2.3.3. Phase2 : Vers une représentation prototypique des connaissances

La phase précédente d'analyse consiste simplement à construire le sens d'une séquence `N1 à N2` par appariement de la classe sémantique attachée à ce prototype : le `N2` est porteur de la

classe sémantique du prototype `N1 à N2` dont il est un des composants. Pour mener à bien une tentative de construction du sens des dénominations complexes `N1 à N2` (où `N1` est fixé), nous avons essayé d'adapter les principes de classification syntaxiques et sémantiques retenus par (Habert & Fabre 1993). Le "calcul" de la sémantique attachée au prototype `N1 à N2` construit est déterminé par la valeur sémantique contenue dans le prototype `N2` (cette valeur sémantique caractérisant des facettes sémantiques de `N2`). Le principe de classification sémantique présenté dans (Habert & Fabre 1993) a été repris : il utilise des classes sémantiques telles que aliment, noté `ali`, synthétique (pour des référents synthétiques, non naturels), noté `synth`, permettant de résumer un ensemble de traits élémentaires tels que `concret`, `comptable` :

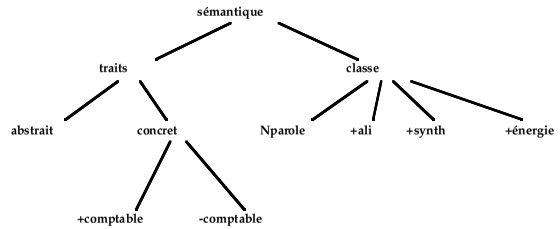

Figure 5.4 : Classification sémantique.

(Habert & Fabre 1993) associe à `moulin` un prédicat à quatre arguments :

`PRODUIRE(Entrée,Sortie,Energie,Partie-De)`

Le modèle sémantique proposé (fondé sur un modèle de représentation à base d'héritage) a permis d'affiner les résultats existants sur la classification des `N1 à N2` (Poncet 1991) en intégrant notamment la classe jusqu'alors résiduelle des séquences du type `moulin à paroles`, celle-ci s'intégrant au modèle proposé suivant les contraintes sémantiques précisées dans la figure 5.5.

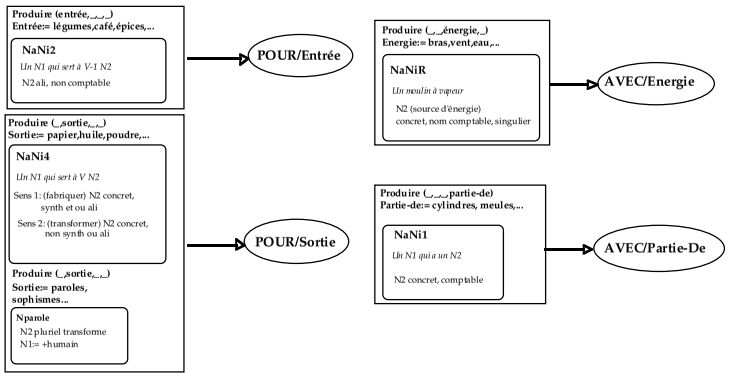

Figure 5.5 : Modèle sémantique pour les séquences `N1 à N2`.

L'extension de notre analyseur prototypique adapte ce mécanisme de construction du sens des formes `moulin à N2`.

- Le prototype `moulin à N2` construit résulte de l'association syntaxique des prototypes attachés aux trois chaînes lexicales données suivant le mécanisme décrit supra.

- La sémantique de ce prototype résulte de l'identication de la définition sémantique du prototype `N2` retenu avec la valeur sémantique attachée aux classes sémantiques définies ci-dessus.

Le calcul de la sémantique du prototype `N1 à N2` résultant s'effectue suivant les valeurs sémantiques du `N2` initial :

Figure 5.6 : Sémantiques `N2` initial et `N1 à N2` résultant.

Les prototypes de type `CatN` ont donc été redéfinis de telle sorte que l'attribut `semantique` dont ils sont porteurs, soit valué sur un prototype `semantique` qui prend en compte le principe de classification choisi. On donne ci-dessous une illustration de la définition sémantique sur des éléments du lexique :

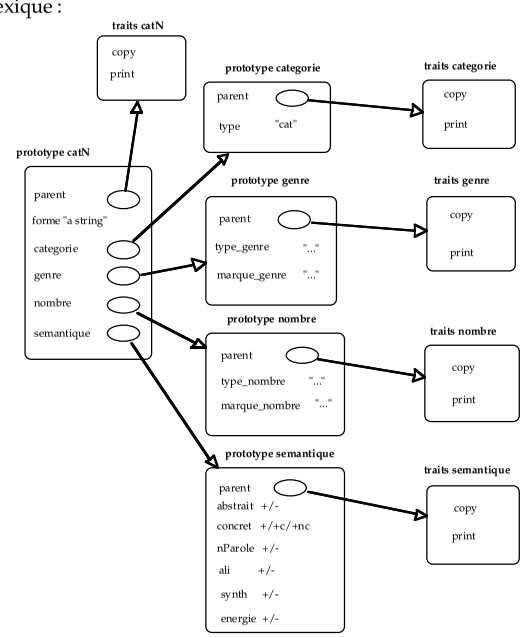

Figure 5.7 : Traits et prototypes pour analyse prototypique Phase2.

L'analyse et la construction du sens se déroule de la manière suivante :

- 1. Lecture des entrées lexicales.
- 2. Recherche des prototypes lexicaux associés et mise à jour de la règle.
- 3. Création du prototype `moulin à N2` :

  3a. Calcul de la sémantique : début.
  3b. Lecture de la sémantique de `<x3> (N2)` et des sémantiques définies dans le modèle.
  3c. Après identification, la valeur sémantique retenue est enregistrée.

- 4. Test `NParole` : si `(<x3> semantique nParole) = '+'`, ajout de `Nhumain <- '+'` au prototype sémantique attaché au prototype `moulin à N2` résultant.

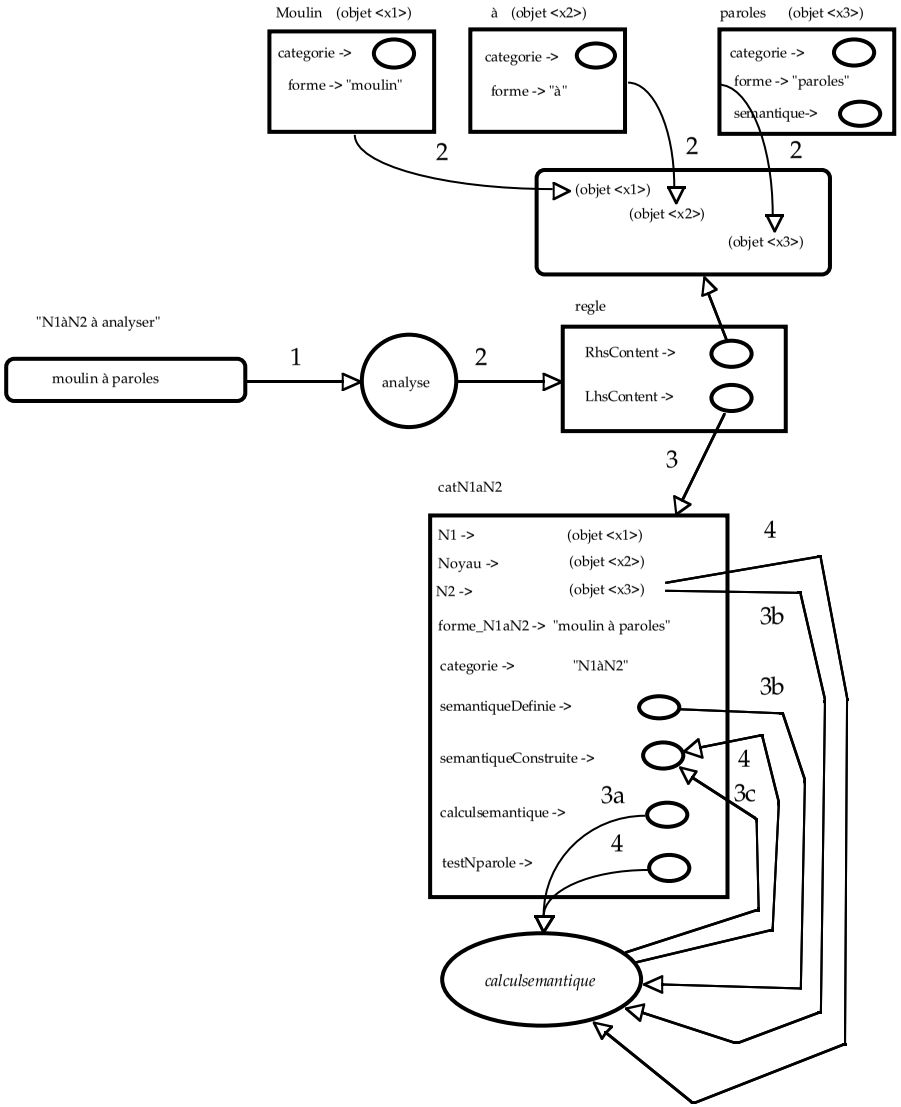

Figure 5.8 : Analyse prototypique Phase2.

5.2.3.4. Phase 0-2 : Premiers résultats acquis, perspectives

Les prototypes construits peuvent être ajustés automatiquement (i.e. recatégorisés)

Ces premières phases de travail donnent en effet une illustration simple de l'ajustement dynamique qu'il est possible de mettre en place sur les structures de représentations construites dans le cadre de la construction du sens des composés `moulin à N2`. Dans la figure 5.8, la sémantique du prototype `moulin à N2` construit est modifiée dynamiquement par l'ajout d'un attribut `Nhumain<-'+'`. C'est ce genre de mécanisme de construction du sens qu'il faut mettre en place, si l'on veut affiner progressivement les connaissances sémantiques des objets construits. Il s'agit désormais de progresser dans la représentation des éléments définis pour une analyse automatique expérimentale avec des prototypes : du point de vue du code développé mais aussi du point de vue de la représentation des connaissances proposée. On peut ainsi envisager de définir, pour les prototypes `moulin à N2`, des comportements généraux de mise à jour du sens construit qui tiennent compte de la sémantique des composants initiaux (au niveau des `traits catN1àN2`). En effet, la mise à jour proposée (via les attributs `calculsemantique` et `testNparole`) est définie en termes de méthodes construites au niveau du prototype `catN1àN2`. S'il possible de "stabiliser"/généraliser ce type de comportement, il serait souhaitable pour le développement d'un code cohérent (pour la lisibilité selon Self, et pour d'éventuelles mises à jours "économiques") de définir ces comportements au niveau des `traits` associés. On peut ainsi considérer que le calcul d'identification de la sémantique du `N2` avec une des classes sémantiques définies est un mécanisme général de construction du sens des séquences `moulin à N2`. De ce fait, ce comportement commun devrait être redéfini au niveau de l'objet `traits catN1àN2`. Une construction progressive de stabilisation des résultats doit entraîner une classification[4] nouvelle des connaissances sur les objets prédéfinis. Cette réorganisation des représentations initiales introduit une nouvelle fois dans ce travail la nécessité de définir un "méta-regard" sur les constructions en cours pour évaluer les résultats et prendre en compte les régularités ou généralisations qui se dessinent afin de modifier les représentations initiales. Nous y reviendrons infra.

Elargir la couverture sémantique du modèle

Il s'agit aussi de mettre à l'épreuve les mécanismes de calcul et de représentation choisis sur le corpus des formes `moulin à N2` pour évaluer les évolutions possibles. On sait en effet que certains exemplaires de `moulin à N2` ne rentrent pas dans le cadre du prédicat à quatre places retenu : `scie, foulon, ventilateur`. Il s'agit pour ces derniers, non pas de ce que produit le moulin, mais de l'instrument constitué par le moulin : l'énergie du moulin sert à scier, à fouler, à ventiler. Ces trois mots sont des déverbaux et ce fait permet de sélectionner une interprétation : `instrument dont l'énergie est apportée par le moulin et non Partie-De`.

De même, l'étude des emplois métaphoriques peut élargir la limitation faite ici au `NParole`. On peut ainsi avoir des `moulin à N2` métaphoriques avec (`N2 pluriel` : "coups, thèmes, images, sophismes" ou avec (`N2 massif singulier` : "ennui"), dans les deux cas, on a une production massive. La présentation faite ici doit donc prendre en compte ces exemplaires de `moulin à N2` qui apportent des précisions au modèle initial.

[4] Il en va d'ailleurs de même dans toute activité d'intelligence humaine : même si les individus manipulent plutôt des entités individualisées dans des conditions empiriques données, le recours à une classification est nécessaire à terme pour structurer les informations acquises par apprentissage ou par expérience.

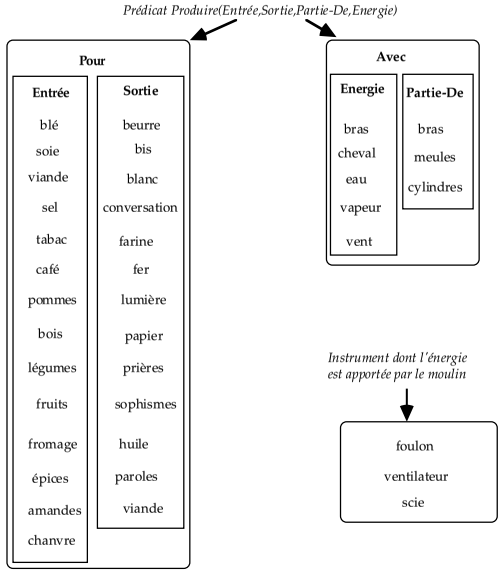

Figure 5.9 : Répartition lexicale suivant le modèle sémantique retenu.

5.2.4. Analyse automatique et construction des prototypes `N1 à N2`

5.2.4.1. Préliminaires

Cette phase d'extension de l'analyse automatique présentée supra accentue le rôle des prototypes dans la conduite et le déroulement de l'analyse. Les prototypes mis en oeuvre ne sont plus uniquement des entités passives, en particulier les prototypes résultants de l'analyse seront porteurs de savoirs lexicaux ou syntaxiques de plus en plus précis afin de leur permettre d'infléchir le déroulement de l'analyse. Un prototype `manager` a pour fonction de superviser les activités prototypiques mises en oeuvre par l'analyse, de mémoriser et de stocker les résultats construits. Cette solution permet d'accroître notre **contrôle** sur les entités manipulées et construites lors de l'analyse. Le `manager` constitue en effet un point d'entrée central pour interroger les prototypes qu'il gère au cours de l'analyse. Comme précédemment, le manager utilise des **méta-processus** pour interroger l'ensemble des prototypes définis. Il est de plus chargé de "garder la main" sur les prototypes qui répondent positivement à ses requêtes.

5.2.4.2. Phase3.0 : Filtrage pour la construction du prototype `moulin à N2`

Cette première étape restreint le modèle syntaxique initial à la règle hors contexte suivante :

```
moulinàN2 -> moulin Prep N2 (où Prep = à).
```

Le prototype `moulin à N2` construit connaît déjà deux de ses constituants lexicaux (`N1` et `Prep`) ainsi qu'un comportement du calcul sémantique. L'analyse qui suit opère un filtrage sur le `N2`. Les propriétés sémantiques du `N2` détermineront la classe sémantique du prototype

`moulin à N2` résultant suivant le modèle présenté supra qui associe à `moulin` le prédicat `PRODUIRE(Pour/Avec)`.

L'analyse se déroule de la manière suivante :

Le prototype `manager` lit une entrée lexicale et déclenche le processus de filtrage de cette entrée lexicale dans le but de construire un prototype `moulin à N2` qui entre dans la classification prédéfinie.

- 1. Création d'un prototype `moulin à N2` (l'objet à construire) par clonage du prototype `moulin à N2` prédéfini. La forme lexicale lue est affectée à l'attribut `forme` du prototype en position `N2` dans le prototype `moulin à N2` en cours de construction.

- 2.(a) Le comportement qui effectue le calcul sémantique du prototype résultant opère une identification de la sémantique du prototype en position `N2` avec le modèle sémantique défini par les prototypes `moulin à N2`, i.e `PRODUIRE(Avec,Pour)` (b).

Le résultat produit un prototype `moulin à N2` qui soit vérifie le prédicat proposé (sa sémantique est alors définie par l'une des quatre classes sémantiques) soit ne vérifie pas le modèle proposé (i.e. les traits sémantiques associés au `N2` ne sont pas reconnus par les filtres mis en place), dans ce cas sa sémantique reste indéfinie (-> figure 5.10).

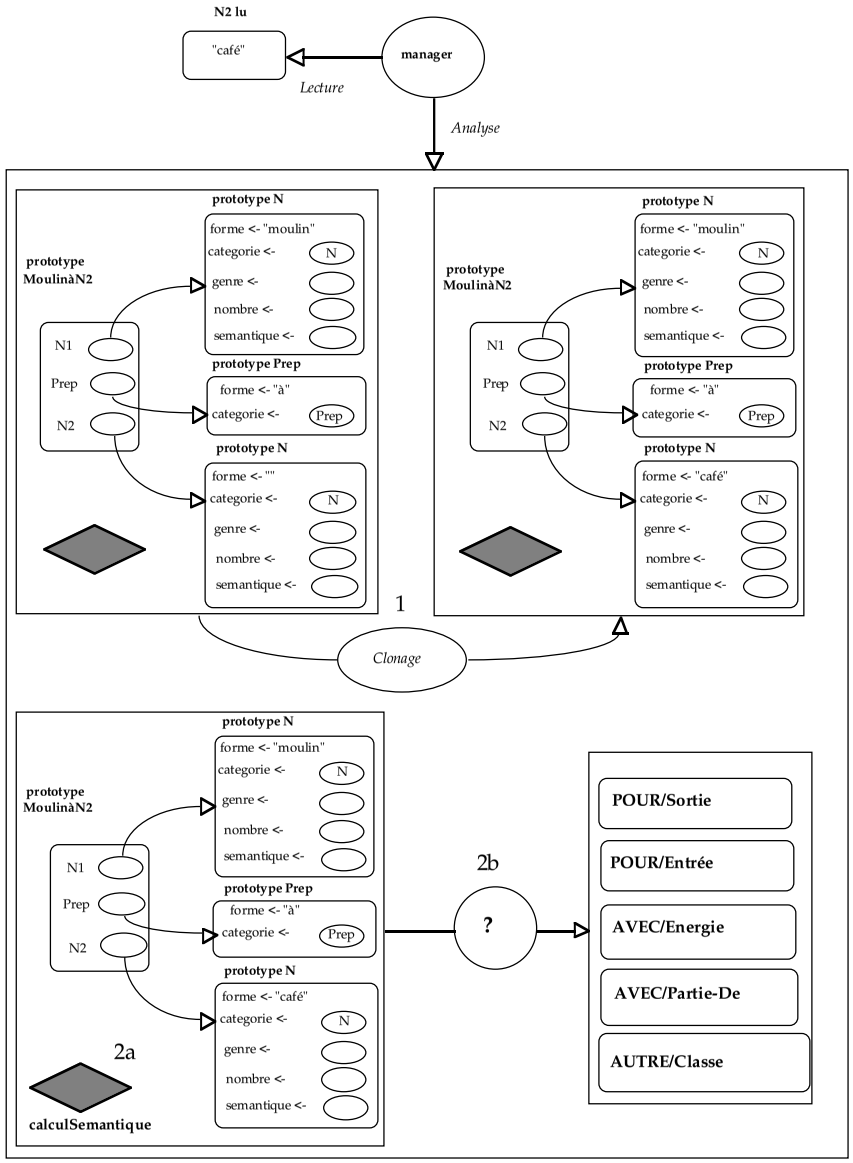

Figure 5.10 : Analyse prototypique Phase3.0.

5.2.4.3. Phase3.1 : Filtrage pour la construction du prototype `N1 à N2`

Cette deuxième étape reprend le modèle syntaxique initial : `N1àN2 -> N1 Prep N2`, et ne restreint plus la forme lexicale du `N1`. Il s'agit de construire des prototypes `N1 à N2` en tenant compte des résultats obtenus supra. Si la forme lexicale lue est "moulin", la phase précédente nous permet d'envisager la construction éventuelle d'un prototype `moulin à N2` suivant les contraintes définies. Si ces contraintes ne sont pas vérifiées, l'analyse construit un prototype `N1 à N2` pour lequel aucune construction de sens n'est envisagée pour le moment (Si la forme lexicale lue est de type "boite", par exemple). Le prototype `N1 à N2` initialement construit est donc porteur de différents niveaux de savoirs. Le savoir lexical qui porte sur la forme lexicale du `N1` permet soit de déclencher une analyse sémantique du type de la phase 3.0, soit de rester dans un cadre plus général d'analyse pour lequel aucun traitement sémantique précis n'est encore défini.

L'analyse se déroule de la manière suivante :

Le prototype `manager` lit les entrées lexicales et déclenche le processus d'analyse. Le premier mot lu est porteur d'un certain type de savoir syntaxique : il peut entrer dans la construction de prototype `N1 à N2` en position `N1`. Le `manager` créé donc un prototype `N1 à N2` (l'objet <x1>) par clonage du prototype `N1 à N2` prédéfini. Le prototype `N1 à N2` est quant à lui porteur d'un savoir lexical lui indiquant que pour certains prototypes `N` en position `N1`, leur forme lexicale peut spécialiser l'analyse vers un autre type de prototype à construire. C'est le cas des prototypes `N` en position `N1` dans un prototype `N1 à N2` dont l'attribut `forme` est égal à "moulin".

A. Le prototype `N` en position `N1` permet de construire un prototype `moulin à N2` :

- 1. Dans ce cas, le prototype `N1 à N2` active la création d'un prototype `moulin à N2`, (l'objet <x2>) par clonage du prototype `moulin à N2` prédéfini, et l'analyse se poursuit comme dans la phase3.0 pour la construction de ce prototype. Le prototype `manager` distribue, par filtrage catégoriel, les entrées lexicales restantes aux prototypes en position `Prep` et `N2` dans le prototype `moulin à N2` en cours de construction ainsi que dans le prototype `N1 à N2` initial. On garde "sous la main" ce prototype au cas où la construction sémantique du prototype `moulin à N2` ne serait pas définie.

La construction du prototype `moulin à N2` se déroule en suivant les étapes suivantes :

- Ajout d'un prototype `N` (`forme := 'moulin'`) en position `N1`.
- Ajout d'un prototype `Prep` en position `Prep`.
- Ajout d'un prototype `N` en position `N2`.
- Si les étapes précédentes se sont déroulées, le calcul sémantique est effectué, et le `manager` "considère" que la construction du prototype `moulin à N2` est effective, i.e le prototype `N1 à N2` initial n'est pas conservé par le `manager`.

De même, la construction du prototype `N1 à N2` se déroule en suivant les étapes suivantes :

- Ajout d'un prototype `N` (`forme := 'moulin'`) en position `N1`.
- Ajout d'un prototype `Prep` en position `Prep`.

- Ajout d'un prototype `N` en position `N2`.

- Si le prototype `moulin à N2` créé par le prototype `N1 à N2` n'est pas effectivement construit (i.e sémantique non définie), le `manager` considère que la construction du prototype `N1 à N2` est effective.

La validation de construction du prototype `N1 à N2` effectue en effet au préalable la vérification de construction du prototype `moulin à N2` qu'il a déclenchée.

- 2. Le comportement sémantique, spécifique aux prototypes `moulin à N2`, effectue ensuite le calcul sémantique du prototype résultant. Il opère donc une identification de la sémantique du prototype associé à l'attribut `N2` avec le modèle sémantique défini par les prototypes `moulin à N2`.

- 3. Le résultat produit est soit un prototype `moulin à N2` qui vérifie le prédicat proposé (sa sémantique est l'une des quatre classes sémantiques) soit un prototype `N1 à N2` dont la sémantique n'est pas encore identifiée.

La figure 5.11 donne une présentation détaillée du déroulement de l'analyse ainsi décrite.

B. Le prototype `N` en position `N1` ne permet pas de construire un prototype `moulin à N2` :

Dans ce cas, (1) l'analyse effectue, par filtrage catégoriel, un simple appariement des prototypes associés aux entrées lexicales restantes vers les attributs `Prep` et `N2` du prototype `N1 à N2` construit, (2) la sémantique de ce dernier reste non identifiée (-> figure 5.12).

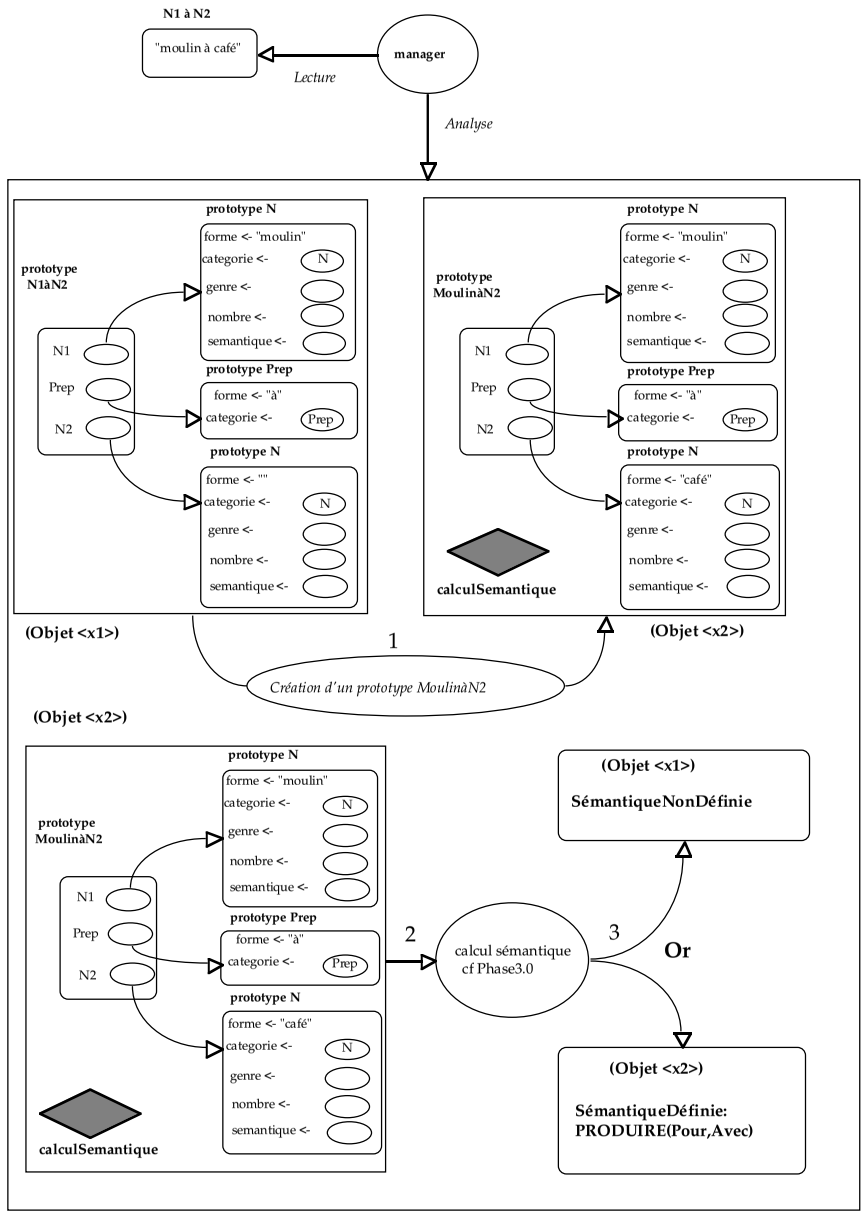

Figure 5.11 : Analyse prototypique Phase3.1(A).

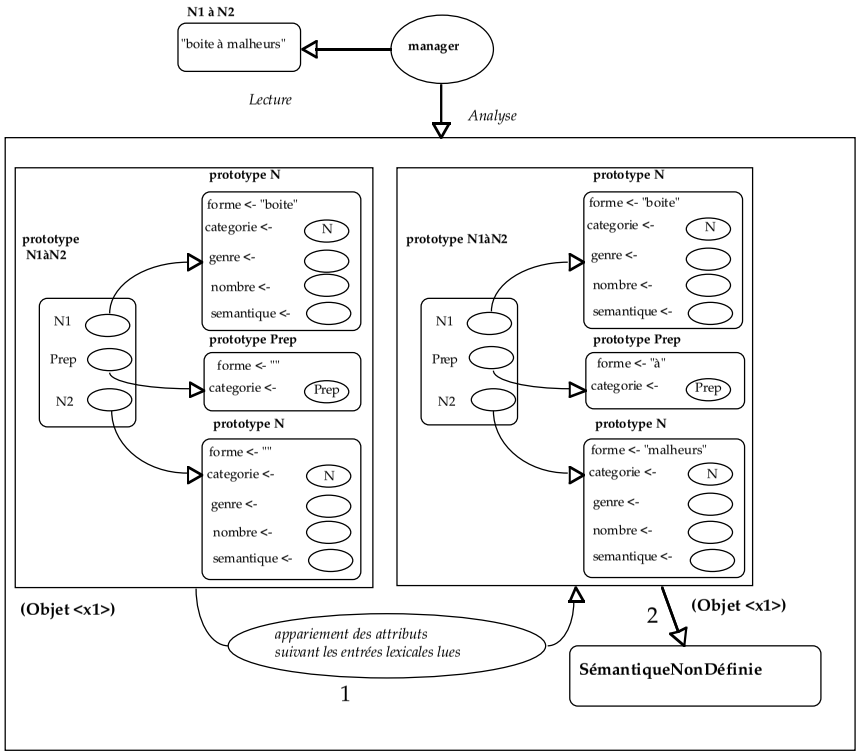

Figure 5.12 : Analyse prototypique Phase3.1(B).

La figure 5.13 détaille les activités du `manager` lors de l'analyse. La numérotation qui suit suit fait référence aux indications données dans cette figure :

- 1 Le `manager` lit la phase à analyser et pour chaque entrée lexicale.
- 2 Il déclencle une recherche de tous les prototypes lexicaux ayant un attribut `forme` de même valeur que l'entrée lue.
- 3 Cette recherche permet de sélectionner un certain nombre de prototypes.
- 4(0) Les prototypes lexicaux dont l'attribut `forme` est 'moulin' et en particulier un prototype `N` (`categorie type := 'nom'`),
- 4(1) Les prototypes lexicaux dont l'attribut `forme` est 'a' et en particulier un prototype `Prep` (`categorie type := 'Prep'`),
- 4(2) Les prototypes lexicaux dont l'attribut `forme` est 'paroles' et en particulier un prototype `N`, (`categorie type := 'nom'`).

- 5(0) Pour chaque prototype, on fait une recherche des constructions éventuelles à déclencher à partir de ces prototypes. Dans notre cas, seule la construction des prototypes `N1 à N2` est envisagée. Certains prototypes possèdent donc un attribut indiquant le type de prototype "complexe" qu'il est possible de construire partir de ce prototype lexical initial . Dans notre exemple, le prototype `N` associé à 'moulin' déclenche la création d'un prototype `N1 à N2` (par clonage).
- 5(1) De plus les prototypes `N1 à N2` permettent la création d'un prototype spécialisé si le prototype en position `N1` a un attribut `forme` de valeur égale à 'moulin' : dans ce cas le prototype `N1 à N2` génère un prototype `moulin à N2` et le manager poursuivra l'analyse en construisant en parallèle ces deux prototypes.
- 6 Les prototypes en cours de construction sont stockés, les opérations 2, 3, 4, 5 sont répétées.
- 7 Pour chacune des entrées lexicales, si le filtrage catégoriel est vérifié, les prototypes en cours de construction sont mis à jour (on leur adjoint les prototypes lexicaux correspondants).
- 8 En fin de lecture, le `manager` ne contient que les prototypes qui ont vérifié le filtrage catégoriel (dans notre cas `Nom Prep Nom`), i.e soit un prototype `N1 à N2`, soit un couple de prototypes (`N1 à N2, moulin à N2`).
- 9 Dans ce dernier cas, si le prototype `N1 à N2` a permis la construction d'un prototype `moulin à N2` (c'est le cas dans l'exemple donné), le prototype `N1 à N2` vérifie si le prototype `moulin à N2` qu'il a créé est effectivement construit.
- 10 Si le calcul sémantique qu'il permet produit un résultat cohérent avec le modèle sémantique choisi, si tel est le cas le prototype `N1 à N2` "considère" qu'il n'est pas construit puisqu'un prototype plus précis peut lui être associé.
- 11(0) C'est le prototype `moulin à N2` qui est retourné au `manager`.
- 11(1) Ou le prototype `N1 à N2` qui est effectivement construit et retourné au `manager`.

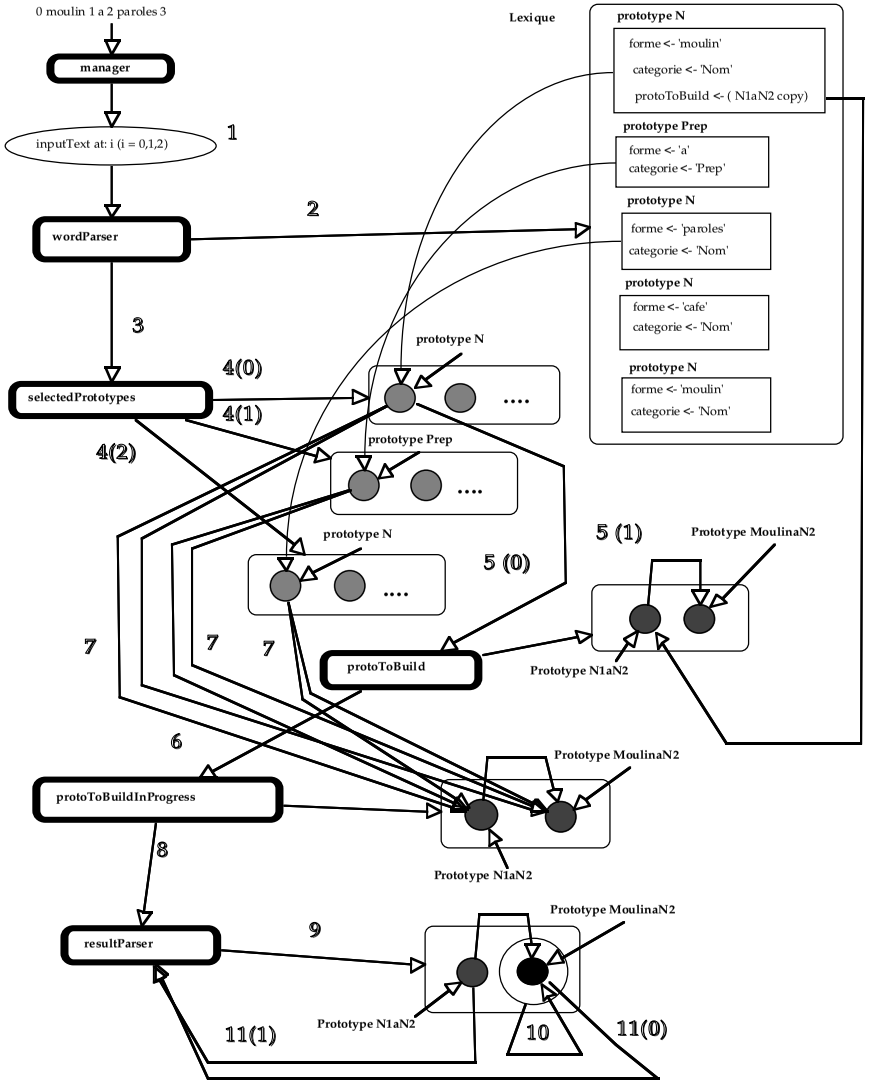

Figure 5.13 : Analyse Phase3.1.

5.2.4.4. Phase3.2 : Filtrage pour une construction spécialisée des prototypes `N1 à N2`

Cette troisième phase doit enrichir le modèle de construction du sens présenté dans la phrase précédente. Si les restrictions faites pour la construction du sens des prototypes `N1 à N2` ne permettent pas de prendre en compte un grand nombre de ces formes, il faut pouvoir enrichir les règles existantes. Les deux phases précédentes permettent d'effectuer un première approche de construction spécialisée des séquences `N1 à N2` : pour de telles formes où `N1 = moulin` et pour lesquelles le calcul sémantique reconnaît une forme sémantique qui entre dans la classification prédéfinie, l'analyse construit une forme prototypique de `N1 à N2` plus précise que si les contraintes précitées n'étaient pas vérifiées. Cette première approche demande a être précisée dans la mesure où, évidemment, un grand nombre de séquences `N1 à N2` ne sont pas prises en compte par l'analyse ainsi présentée. C'est justement grâce à cette approche prototypique de représentation des éléments de l'analyse qu'il nous semble possible de poursuivre cette étude. Puisque la classification de telles formes n'est pas encore figée, les connaissances "individuelles" sur de telles formes doivent nous permettre de progresser en affinant les résultats construits. Il faut pour cela prolonger les règles existantes et ajouter à celles-ci des éléments qui permettent de préciser plus avant les filtrages opérés. On a déjà vu par exemple que certaines séquences de `moulin à N2` ne vérifient pas le prédicat `Produire(Avec/Pour)`, à l'image de"scie, foulon, ventilateur". Il s'agit pour ces derniers, non pas de ce que produit le moulin, mais de l'instrument constitué par le moulin : l'énergie du moulin sert à scier, à fouler, à ventiler. Ces trois mots sont des déverbaux et ce fait permet de sélectionner une interprétation. Parmi les prototypes `N1 à N2` de sémantique non définie construits à l'issue de la Phase3.1(A), on peut donc opérér un filtrage suppléméntaire pour prendre en compte ce type de construction sémantique dans le cas où le `N2` initial est un déverbal.

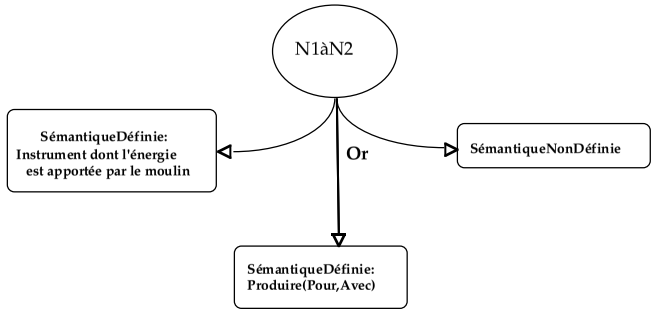

Figure 5.14 : Analyse prototypique et construction progressive du sens des séquences `N1 à N2`.

5.2.5. Une Analyse automatique prototypique étendue

5.2.5.1. Analyse par un réseau de prototypes

Cette étape vise à développer le mécanisme de l'analyse pour disposer d'un analyseur plus "puissant". Celui-ci n'est plus restreint à l'analyse des formes `N1 à N2`, il est capable d'analyser des phrases complexes. De plus un traçage des opérations effectuées par l'analyseur est disponible.

Mise en place d'un réseau de prototypes

L'analyseur ne dispose pas de grammaire initiale. Ce sont les prototypes définis pour l'analyse qui portent le savoir syntaxique.

Déroulement de l'analyse

Comme précédemment, un prototype nommé `manager` est chargé de conduire l'analyse puis de fournir les résultats obtenus.

- 1. Le `manager` lit la phrase à analyser : une liste de mots.
- 2. Pour chaque entrée lue, on recherche tous les prototypes lexicaux dont le champ `forme` a une valeur identique à cette chaîne. Le `manager` enregistre ces prototypes et leur position dans la phrase.
- 3. Pour chacun des prototypes initiaux enregistrés, on vérifie si leur champ `protoToBuild` contient un prototype ou plusieurs qu'il est possible de construire à partir de ce prototype initial. Dans les figures qui suivent, un prototype lexical de type `catDet` permet la création d'un prototype `catSn`, représentant un groupe nominal. Le prototype `catDet` sait donc implicitement qu'il peut permettre la création d'un prototype `catSn` regroupant un prototype `catDet` et d'un prototype `catN`, le prototype `CatSn` se construisant de "droite à gauche" : il attend tout d'abord un prototype de catégorie `Det` puis un prototype de catégorie `Nom` (`Sn -> Det N`). Le prototype `catSn` pouvant lui aussi être porteur de savoir sur le type de prototype dont il peut déclencher la création (en l'occurrence un prototype `catP` dans les exemples donnés représentant un phrase). Ces prototypes à construire enregistrent donc les prototypes dont ils sont issus et leur position dans la phrase. Le `manager` stockent ces prototypes à construire.
- 4. Le prototype `manager` possède à ce stade d'une liste de prototypes construits (les prototypes initiaux) et d'une liste de prototypes à construire (partiellement construite à l'aide des prototypes initiaux). Pour chacun des prototypes à construire, le `manager` vérifie s'il existe un prototype qui peut entrer dans la construction de celui-ci parmi les prototypes disponibles construits (au départ il ne dispose que des prototypes construits au point (2) puis des prototypes construits par l'analyse). La sélection d'un prototype à adjoindre au prototype à construire doit s'assurer d'une part de la cohérence des places respectives du prototype à construire et du prototype à adjoindre dans la phrase, et d'autre part de la concordance du type de catégorie attendue par le prototype à construire avec celle du prototype disponible sélectionné. Si ces deux contraintes sont vérifiées, le prototype sélectionné est ajouté au prototype à construire dans la position qui lui convient.
- 5. Le `manager` répète les opérations décrites au point (4) (sélection d'un prototype à construire et recherche d'un prototype à adjoindre) tant que l'analyse créé de nouveaux prototypes à construire ou tant qu'elle adjoint un prototype à un autre.

Une trace complète de ces mécanismes est donnée en annexe (-> annexe partie 2 chapitre 5). Et les figures 5.15, 5.16, 5.17 illustrent le réseau de prototypes que l'analyse met en place.

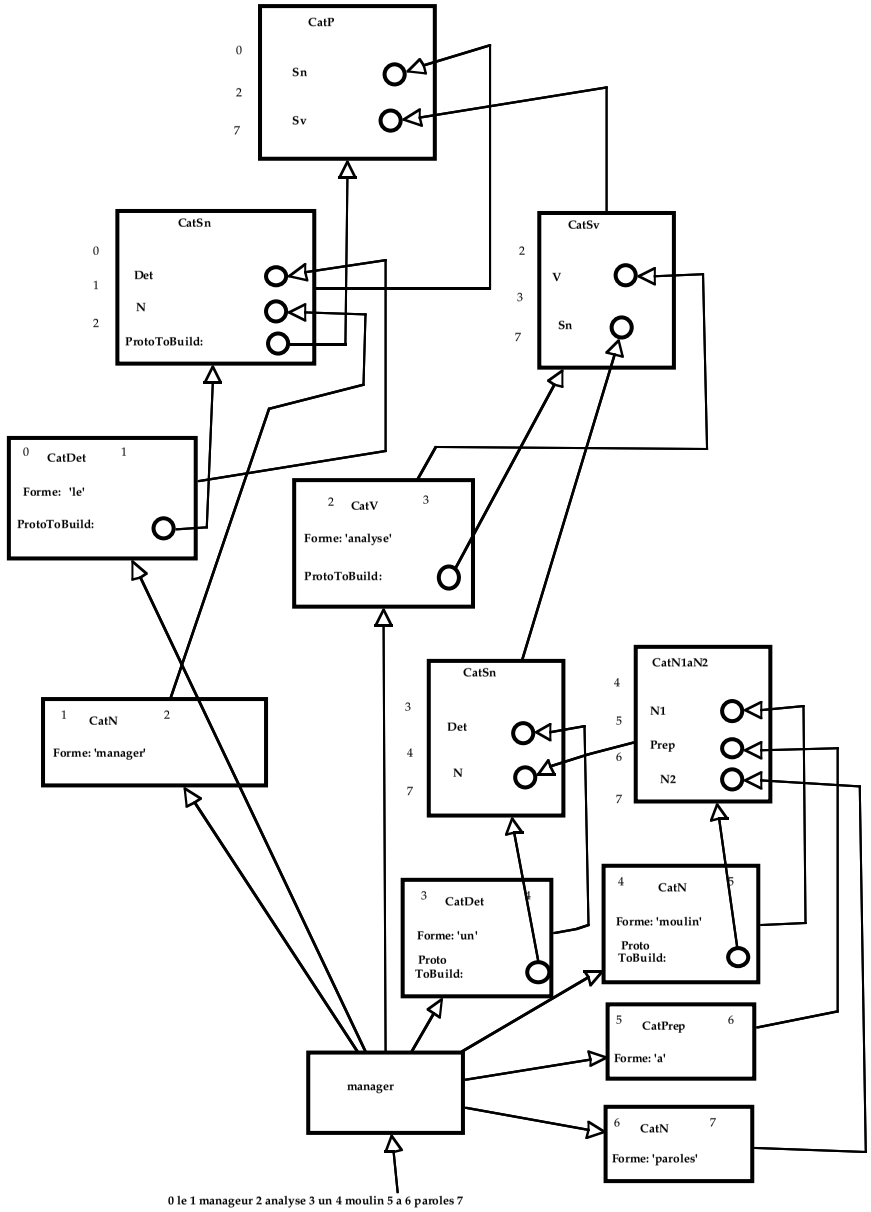

Figure 5.15 : Analyse par un réseau de prototypes.

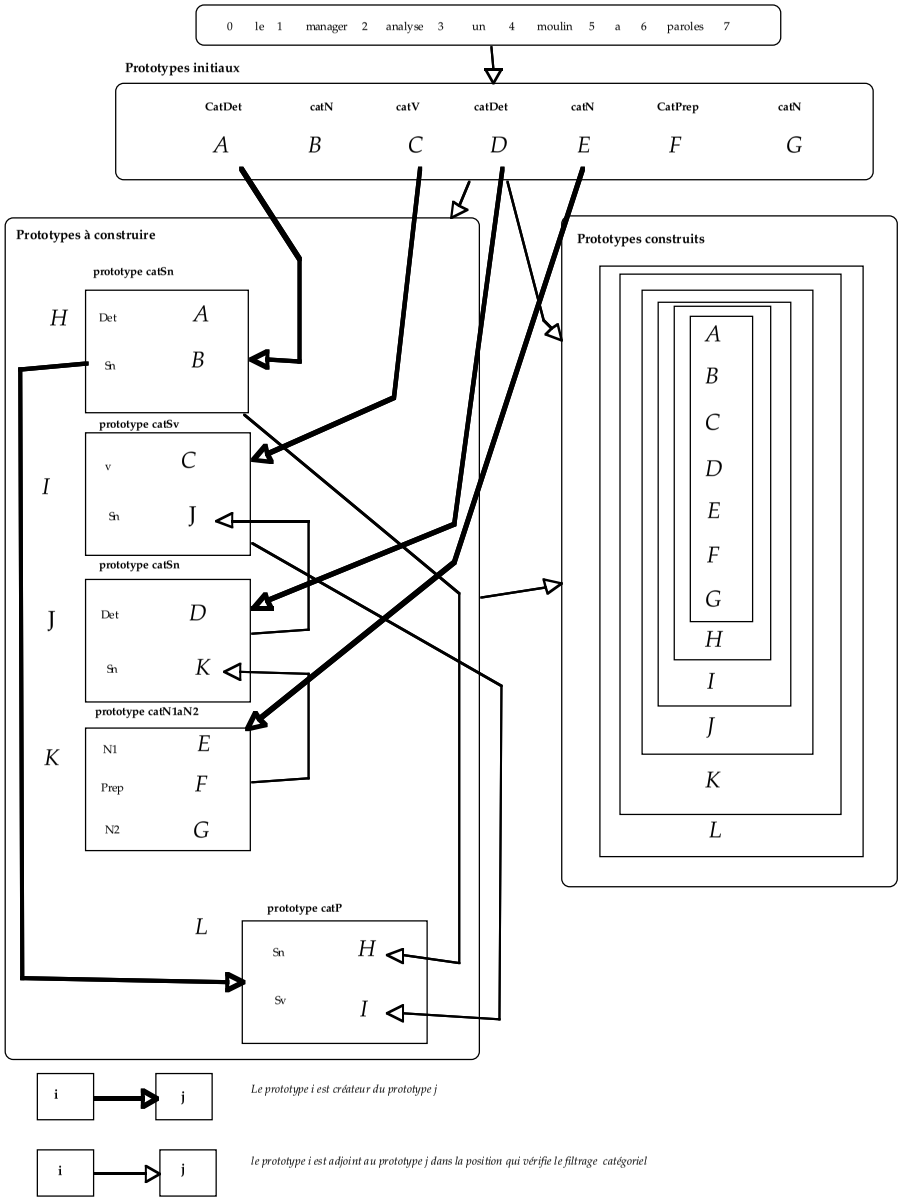

Figure 5.16 : Mise en place du réseau de prototypes pour l'analyse.

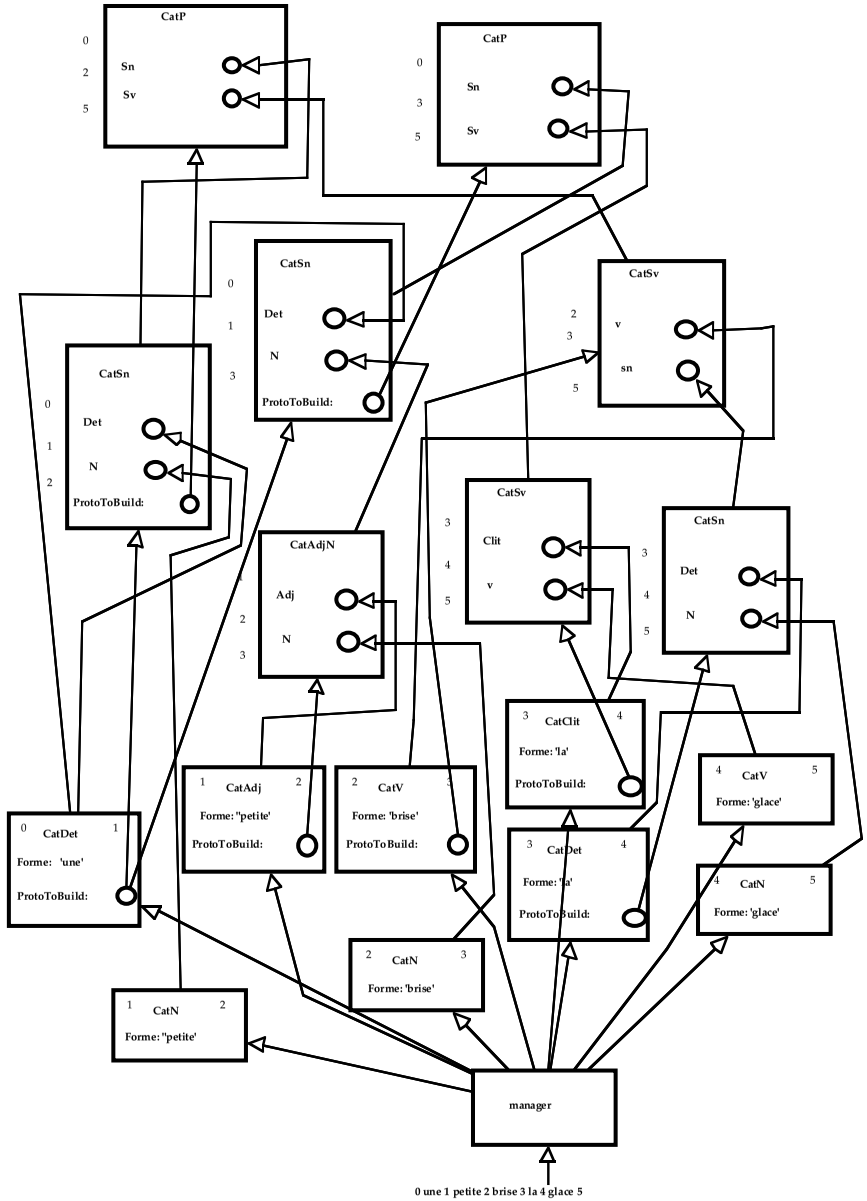

Figure 5.17 : Une petite brise la glace (1)

Cette figure est une illustration du résultat d'analyse obtenu via l'interface graphique de Self-3.0. Cette phase d'analyse nous a permis en effet d'intégrer aux processus d'analyse des mécanismes qui activent une **trace graphique des activités du `manager`**. L'utilisateur peut ainsi suivre dans cette fenêtre une "version animée" de l'analyse réalisée. Les prototypes construits apparaissent au rythme de leur construction et en fin d'analyse l'utilisateur peut ausculter les objets, les ajuster ou les remodeler à sa guise. Dans l'implémentation construite, on a choisi de ne présenter (de manière prédéfinie) via cette interface graphique qu'une partie des attributs associés aux objets construits, l'utilisateur peut évidemment étendre la présentation des objets présents dans la fenêtre graphique en suivant les protocoles[5] définis par Self pour l'utilisation de cette interface.

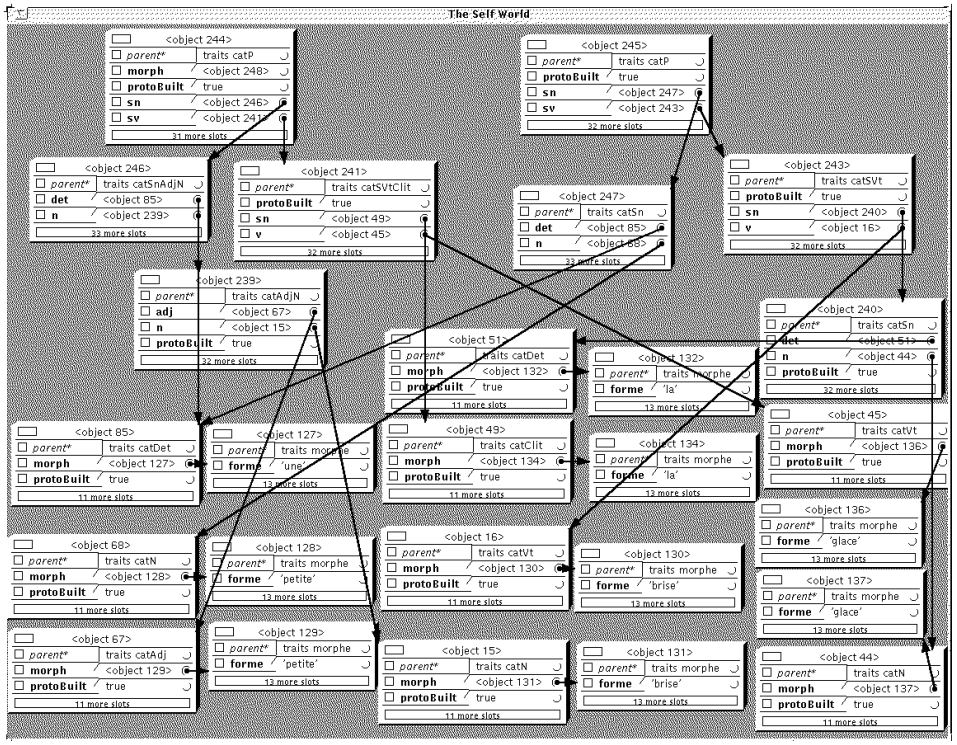

Figure 5.17 : Une petite brise la glace (2)

5.2.5.2. L'utilisateur peut contrôler la construction du sens des séquences en `moulin à N2`

Le travail présenté jusque là a consisté à développer des outils pour une analyse syntaxique prototypique dans le but de disposer d'outils fiables mais aussi "agréables" pour la présentation des résultats. Le détour fait supra vers une analyse syntaxique de phrases complexes a ainsi permis de mettre au point un prototype de parseur qui répond partiellement à ces contraintes (notamment pour une présentation lisible des résultats). Cette phase de

[5] Tous les objets sont en fait réactifs à la souris (-> tutorial Self).

travail a été poursuivie pour affiner les traces produites lors d'une analyse, en particulier pour évaluer les opérations effectuées lors d'une analyse. Le point d'ancrage linguistique de ce travail, à savoir l'étude des formes complexes `N1 à N2`, va de nouveau être le centre d'intérêt principal dans les phases de travail présentées ci-dessous. L'étude de la construction de ces formes complexes par affinage progressif des connaissances représentées a nécessité la mise en place d'outils qui permettent de réaliser ce travail d'affinage des connaissances quand celles-ci s'avèrent insuffisantes pour construire une forme complexe dans le cadre sémantique déjà présenté. Lorsqu'une analyse d'une forme `moulin à N2` ne construit pas un représentant d'une classe sémantique déjà connue, il paraît souhaitable de rendre la main à l'utilisateur pour que celui-ci affine les représentations initiales, si cela est possible afin de construire une forme dont le sens peut être reconnu par le calcul sémantique mis en place. Plus précisément, si une forme complexe du type `N1 à N2` permet de construire un prototype `moulin à N2` (<-spécialisation d'un prototype `N1 à N2`), i.e si `N1='moulin'`, le calcul sémantique attaché au prototype `moulin à N2` est effectuée. Si ce calcul produit un résultat cohérent avec le modèle `Produire(Pour,Avec)`, l'analyse ira à son terme sans intervention de l'utilisateur. Par contre, si ce résultat n'est pas le représentant d'une classe sémantique définie l'analyseur propose à l'utilisateur d'agir sur la sémantique des prototypes attachés aux composants de la forme complexe en cours de construction. Cette intervention au cours de la construction du prototype résultant permet ou non d'infléchir l'analyse. Elle est aussi guidé par le `manager` qui propose à l'utilisateur un protocole pour réaliser une éventuelle mise à jour des prototypes qui interviennent dans l'analyse.

Dans un premier temps le calcul présenté précédemment pour le calcul de la sémantique d'un prototype `moulin à N2` a été maintenu : celui-ci est déterminé uniquement par les valeurs sémantiques (exprimées sous forme de traits) associées au prototype lexical en position `N2`. Ce choix peut être remis en question. En effet il peut être souhaitable de donner un plus grand poids à chacun des composants de la forme résultante au lieu de privilégier le prototype en position `N2`. Ce point sera repris infra. On donne en annexe (-> annexe partie 2 chapitre 5) une illustration d'une analyse qui propose à l'utilisateur de modifier la sémantique du prototype lexical en position `N2` dont la sémantique ne permet de reconnaitre un des représentants des classes sémantiques définies. Toute l'analyse est gérée via l'utilisation de prototypes qui proposent un menu de choix, aussi bien pour charger les données de l'analyse que pour présenter les éléments lexicaux ou encore pour permettre au cours de l'analyse d'intervenir si le résultat produit n'est pas sémantiquement satisfaisant. On introduit ici le phénomène crucial du contrôle dans des processus de représentation des connaissances. L'analyse permet ici de rendre la main à l'utilisateur pour que son intervention modifie les valeurs sémantiques initiales afin de construire un résultat sémantique cohérent.

5.3. Approche prototypique de la construction du sens des séquences `moulin à N2` : résumés et perspectives

Pour le moment on opère un appariement de traits sur un nombre restreint de prototypes sémantiques de référence. Le risque est grand de devoir utiliser un très grand nombre de prototypes de référence si le nombre de traits sémantiques utilisés augmente. Les prototypes de référence pourraient être des points d'entrée très généraux qui se préciseraient petit à petit si l'inférence le nécessite. Dans le cas où l'inférence ne conduit pas à un résultat satisfaisant, on pourrait ainsi disposer d'un début de chemin "à travers" un prototype de référence (du point d'entrée vers le point critique où l'inférence échoue). Il serait ainsi possible de conserver en mémoire ce parcours puis de le modifier en ajoutant les informations nécessaires pour aller plus loin : définir un chemin (i.e une nouvelle classe). Les problèmes à résoudre sont :

- Comment pouvoir mémoriser un nouveau chemin (i.e une nouvelle classe) ?

- Si ce qui permet de reconnaitre une classe est un mécanisme d'appariement, comment figer et inclure un appariement sur un nombre croissant de valeurs sémantiques? La fonction `Sem` définie plus haut opère un appariement sur un nombre restreint et fixe de traits sémantiques. Il peut être nécessaire de faire évoluer ce nombre de traits sémantiques. Il serait donc souhaitable de pouvoir ajuster la fonction `Sem` ainsi que les représentations initiales des prototypes sémantiques de référence

Il faut donc pouvoir mettre en place un formatage récurrent, une rétroaction sur les représentations construites et des fonctions nouvelles sur ces représentations. Si on dispose d'un grand nombre de formes `moulin à N2` non reconnues par le modèle initial, il faut pouvoir évaluer les valeurs des traits et les comparer à toutes celles qui sont déjà stockées et essayer de les classer. On atteint là une étape cruciale dans notre démarche de construction des processus de représentation avec la PàP qui nous oblige à revoir le type de représentation à définir pour réaliser les objectifs annoncés.

5.3.1. Lexique et prototypes

Que sont les comportements (au sens donné par Self de ce terme) communs à tous les exemplaires d'une catégorie ou, pour rester dans le domaine de la linguistique, d'une catégorie grammaticale? Cette question a-t-elle un sens? Dans ce qui a été présenté supra, les seuls comportements communs définis sur les "catégories" manipulées ne concernent pas des propriétés comportementales purement linguistiques, il s'agit simplement d'"artifices" informatiques qui permettent d'agir sur les objets utilisés soit pour permettre de dupliquer ces objets soit pour permettre des opérations d'impression des facettes déterminées sur ces objets. Peut-on envisager que les comportements (s'ils existent) des différentes catégories grammaticales ne soient pas dépendants de ces mêmes catégories? Le partage de certains comportements pourrait exister de manière transversale par rapport aux catégories. Il aurait pu en être ainsi dans ce qui est présenté supra puisque tous les prototypes utilisés déléguent au moins le comportement de clonage à un trait associé : une factorisation de ce comportement est donc envisageable. Là encore, il ne s'agit pas du partage de comportement purement linguistique : la mise en place de telles factorisations sur des comportements linguistiques communs à plusieurs catégories linguistiques est à faire. On atteint là le problème général de l'organisation des connaissances linguistiques à penser pour l'analyse dans un environnement informatique à la Self. La notion de comportement est fondamentale dans celui-ci (*"An object is what he does"*) et ne recouvre pas complètement les notions de catégories "linguistiques"

traditionnelles (privilégiant la représentation). On peut d'ailleurs modifier les prototypes `N1 à N2` utilisés précédemment et répartir différemment les attributs définis sur ces prototypes.

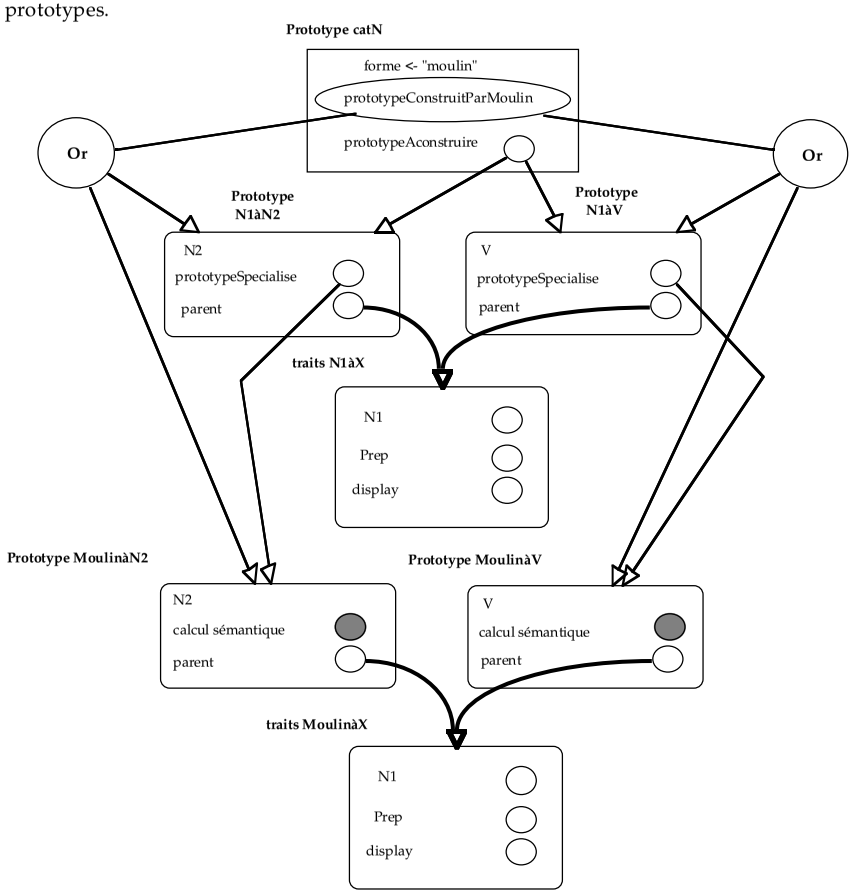

Figure 5.18 : Prototypes et représentation des composés `N1 à X`.

Puisque notre modèle privilégie le rôle du `N2`, on peut adopter une construction de ces prototypes en suivant le modèle que décrit la figure précédente. L'objet `traits CatN1aX` est destiné à recevoir les "comportements" communs aux prototypes `N1 à N2`, `N1 à V`. Notre modèle donnant aux constituants N2 et V un statut particulier (c'est lui qui permet une classification sémantique du composé), les attributs non spécifiques aux prototypes (`N1`, `Prep`, `display`) sont pris en charge par l'objet qui réalise le partage des connaissances entre ces prototypes. Les attributs spécifiques (`N2` et `V`, et les attributs réalisant le calcul sémantique) sont quant à eux maintenus dans la définition des prototypes `N1 à N2` et `N1 à V`. Chacun de ces prototypes est potentiellement créateur d'un prototype spécialisé, respectivement `moulin à N2` et `moulin à V`, si `N1 = moulin`.

Pour les mêmes raisons, les objets `traits` et les prototypes associés perpétuent cette évolution de représentation. Une catégorie ainsi définie serait donc un ensemble composé par :

- Un ensemble de comportements à tous les prototypes regroupé au sein d'un objet,
- Un ensemble de prototypes dont les attributs pointent sur des valeurs spécifiques et qui délèguent des comportements commmuns à l'objet précédemment défini.

La construction du prototype `N1 à X` (ou celle du prototype `moulin à X`) reste "un modèle de calcul particulier" pour tous les objets que nous cherchons à construire. Le prototype défini n'est pas un phénomène de surface[6], il est le filtre définitoire que les constituants éventuels devront traverser pour s'inscrire à leurs places respectives. Il s'agit bien sûr d'un filtre syntaxique qui ne dit rien explicitement sur la sémantique du prototype. Cette sémantique est en fait le résultat du comportement qui sera associé à chacun des prototypes construits c'est-à-dire qui auront déjà réussi à "traverser" ce filtre syntaxique. Cette approche de représentation s'éloigne assez nettement de la notion de prototype à la Rosch, en ce qu'elle met en avant une définition comportementale des prototypes : ce ne sont pas des traits distinctifs qui les distinguent mais plutôt des comportement différents dans un contexte donné. Notre approche construit des entités prototypiques qui ont une **ressemblance de famille structurelle** entre eux et dont la sémantique est définie individuellement : le prototype initial définit un calcul de cette sémantique qui est effectué et qui produit un résultat particulier en fonction des éléments du contexte rencontrés.

5.3.2. Syntaxe et prototypes

Une analyse syntaxique, telle que celle que réalise OLMES (Habert 1991c), dispose d'un savoir initial représenté par une grammaire donnée sous forme de règles et il s'agit au cours de l'analyse d'identifier les éléments lus en entrée aux différentes règles de cette grammaire. Notre approche prototypique n'a pas utilisé le même type de représentation du pré-savoir que constitue la grammaire. On a essayé au contraire de construire des prototypes qui encapsulent ce savoir de manière prototypique. Nous sommes restés pour le moment à une solution intermédiaire entre un savoir sous forme de règles et donc assez hiérarchique (et d'une portée assez longue sur une structure phrastique) et un savoir éclaté sur chaque élément d'une structure phrastique. On peut d'ailleurs voir le développement de notre travail comme une tentative de dissoudre un cadre syntaxique traditionnel (en l'occurrence celui représenté sous forme de règle) et de disséminer le savoir qui y est habituellement attaché sur plusieurs entités de l'analyse : **un effacement progressif de la notion de règle**.

Le but de l'analyse est de construire des prototypes déclenchés par les entrées lexicales lues sur la phrase à analyser. Et l'analyse est considérée comme terminée, quand aucune opération de création de nouveaux prototypes, ou d'adjonction de prototypes n'a lieu. L' opération de création de prototypes fait office de partie gauche d'une règle dans une grammaire traditionnelle : `moulin` peut déclencher la création d'un prototype `N1 à N2` ce que l'on peut traduire par `N1àN2 -> N1 Prep N2`; tandis que l'opération d'appariement de prototypes peut être assimilée à une lecture gauche-droite de la règle que le prototype déclenchée décrit structurellement : le prototype associé à "paroles" peut être intégré au prototype `N1 à N2` en position `N2`, si les prototypes en position `N1` et `Prep` l'ont déjà été à leur place respective. Le savoir syntaxique porté par les éléments prototypiques est resté restreint à des constructions de "courte portée" (ne dépassant la portée d'une phrase). Il semble malgré tout possible de poursuivre cette intuition de représentation qui se situe dans la perspective de construire une analyse automatique qui se ferait en l'absence d'une grammaire à vérifier mais dans une perspective d'une grammaire à construire. On part de rien "mais à chaque passage" on construit la grammaire et si ça marche on analyse le texte donné et on a construit parallèlement la grammaire de ce texte : en fait un prototype de grammaire!

5.3.3. Sémantique et prototypes

[6] On s'éloigne ici de la version étendue du prototype des cognitivistes.

La sémantique mise en place dans les différentes phases présentées est restée assez peu déterminée. On a adapté le modèle défini dans (Habert & Fabre 1993). Cette adaptation a consisté à associer aux éléments lexicaux représentés sous forme prototypique, un prototype sémantique regroupant un ensemble de traits donnant une définition minimale de cet élément lexical. Il est apparu très vite que ce choix initial n'est pas adéquat pour atteindre pleinement le but recherché. L'écriture du calcul sémantique défini pour la construction du sens des configurations nominales du type `moulin à N2` est restée très statique. Cela est dû en grande partie à l'absence de reflexion sur ce que pourrait être une représentation de la sémantique des éléments prototypiques lexicaux dans un environnement à la Self. La difficulté principale est de construire une représentation sémantique des éléments manipulés cohérente avec la notion de prototype. Self permet une structuration hybride des représentations : le partage via les `traits`, et une spécialisation des connaissances via les prototypes. Une modélisation de la sémantique qui utilise ces caractéristiques définitoires du langage est à faire. En particulier, l'utilisation d'une hiérarchie de concepts est à mettre en place si on veut pouvoir évaluer l'articulation qui est à l'oeuvre entre cette hiérarchie des concepts et la contextualisation des entrées lexicales.

De plus, suivant le choix fait de privilégier une représentation prototypique qui porte à la fois des éléments syntaxiques définitoires et sémantiques, c'est toute l'articulation de ces deux ordres qui est à penser. On en présente une illustration plus haut dans la création du prototype `moulin à X`. En effet, celui-ci porte un savoir syntaxique qui est "réparti" sur deux objets (via la délégation) et qui indique la valeur des composants qu'il peut recevoir dans telle position. Il porte aussi un savoir sémantique qui lui est attaché et qui dépend du composant en position `N2`. L'articulation forme-sens est ici restreinte à la portée du composé étudié, il n'est pas sûr qu'on puisse faire évoluer à plus longue distance cette articulation, tout au moins dans notre cadre de travail, cette étude est à faire. Il serait pourtant souhaitable de pouvoir dire par exemple que si le prototype `N1 à N2` de forme "moulin à cheval" (porteur d'un prototype spécialisé `moulin à N2` "moulin à cheval") dans un texte "prototypique" dont le thème général est de parler de phénomènes énergétiques, on puisse dans ce cas récupérer ce savoir contextuel et le transmettre au prototype `moulin à N2` (via le prototype `N1 à N2`) dont le calcul sémantique serait ainsi facilité. Si l'on pu mettre en évidence la possibilité de modifier les représentations initiales d'une entrée lexicale en fonction d'informations sémantiques relevées par l'analyse (ajout du trait `+humain` à `moulin` dans `moulin à paroles`), les insuffisances de modélisation des représentations mentionnées ci-dessus ne permettent pas de mesurer la portée réelle d'une telle potentialité. Même si l'on peut déjà considérer qu'il s'agit, a priori, d'un élément déterminant en faveur de ce type d'approche en comparaison avec une hiérarchie lexicale prédéterminée qui supporte mal l'affinage en fonction de résultats produits par une analyse.

5.3.4. Ajuster les mécanismes statiques/dynamiques de re/classification en/pour ajustant/ajuster les représentations initiales

Le dispositif que nous souhaitons construire doit mettre en place des opérations de catégorisation, c'est-à-dire qu'il doit définir les opérations qui permettent de classifier les séquences `moulin à N2` en les examinant une à une pour construire ce que Milner[7] décrit comme "*la taxinomie la plus complète, la plus fine et la plus économique possible*" (Milner 1989). On a déjà précisé plus haut les opérations qui permettent d'identifier certaines configurations nominales dans un modèle prédicatif. Le développement de notre travail doit répondre à la question :

[7] *"La littéralisation est d'autant meilleure qu'elle induit une taxonomie plus naturelle. Les difficultés empiriques ne doivent pas affecter les propositions générales : dans une science empirique littéralisée, une lettre vaut par l'ensemble des propositions empiriques qu'elle rend accessible; dans la science du langage, le noyau dur des propositions empiriques est de nature taxonomique; par conséquent, la science du langage doit se proposer au minimum de construire une littéralisation qui permette la taxinomie la plus complète, la plus fine et la plus économique possible"* (Milner 1989).

Comment à partir de représentations statiques du lexique,
mettre en place des opérations dynamiques pour l'interprétation ?

5.3.4.1. Ajuster les mécanismes de classification en ajustant les représentations initiales

Certains éléments de réponse ont déjà pu éclairer un affinage des représentations initiales. Avec par exemple l'ajout du trait `+humain` à `moulin` dans certaines configurations nominales du type `moulin à paroles`. Cela étant, les problèmes déjà mentionnés concernant le travail encore à faire pour une représentation des connaissances compatible et adéquate dans un environnement prototypique ne permettent pas encore de donner une illustration plus précise de phénomènes relevant "d'opérations dynamiques pour construire l'interprétation". Pour reprendre notre exemple d'ajout de trait à un élément lexical dans un contexte donné, on peut néammoins être amené à poursuivre notre réflexion suivant au moins deux axes.

Ne pas maintenir des représentations contraignantes et non éclairantes

Tout d'abord, si l'on maintient la notion de traits, une prise en compte d'un plus grand nombre de traits dans la définition sémantique des éléments lexicaux ainsi qu'une évaluation plus précise de la construction sémantique des composés, pourraient conduire à la mise en place d'opérations plus fines dans la construction sémantique des composés construits et notamment dans l'affinage de cette construction. Le maintien d'une représentation sémantique sous la forme d'une famille de traits sémantiques pose malgré tout, comme on l'a déjà souligné, de nombreux problèmes dans le développement du dispositif à construire. Aussi bien pour l'implémentation et l'évolution de cette implémentation, que pour une description fine de ce qui est représenté (la construction du sens en particulier). On peut en effet difficilement restreindre la construction du sens à un "simple" calcul sur un ensemble de valeurs discrètes, comme s'il s'agissait de distribuer une série de valeurs binaires dans une équation pour produire un résultat adéquat. On a d'ailleurs déjà évoqué le fait que la construction du sens met en place des phénomènes complexes qui s'appuie sur des mécanismes cognitifs qu'il est difficile d'assimiler à de simples équations.

Isoler les mécanismes qui participent à la construction du sens et les articuler hiérarchiquement

On peut aussi mettre l'accent sur la prise en compte des résultats produits pour affiner les représentations lexicales initiales avant de poursuivre une prise en compte d'autres phénomènes. Le modèle sémantique initial permet de traiter de nouvelles configurations nominales du type "moulin à paroles" (qui ajoutent un trait sémantique à la représentation initiale d'un élément lexical) ou "moulin à ventilateur" (dans ce cas c'est la prise en compte d''une "déviance" par rapport au modèle sémantique global qui a été initialisé). On peut donc envisager de considérer ces résultats comme des éléments définitoires dans la représentation du composé (et exécutables dans certains contextes donnés). Cette description de l'ajustement dans la construction du sens du prototype résultant ne doit donc plus être partie intégrante dans la définition du prototype et notamment dans le calcul sémantique qui lui est attaché : d'autant que ce calcul a été construit comme une globalité peu manipulable dont il est difficile d'isoler les différents actions effectuées. Il s'agit au contraire de l'extraire de cette représentation initiale. Puis de définir une description de cet ajustement sémantique comme un comportement sémantique définitoire de l'ensemble des prototypes qui partagent ce type de comportements. On peut ensuite inscrire ce comportement dans l'objet qui centralise les comportements des prototypes concernés. La prise en compte de résultats temporaires doit donc s'accompagner d'un

ajustement dans les représentations initialement développées. Ce type d'approche s'inscrit d'ailleurs parfaitement dans un environnement de programmation comme le met en place Self. Il est possible de jouer dynamiquement[8] sur les représentations initialement construites. On peut d'ailleurs procéder de la manière suivante :

- 1. Retrait de l'attribut `addSemantique` au prototype `moulin à N2` via la primitive `_RemoveSlot;`
- 2. Redéfinition de celui-ci , via la primitive `_AddSlots`, en ayant supprimé par exemple le test `Nparole` ou le test `Déverbal;`
- 3. Ajout d'un attribut particulier dans l'objet `traits moulin à N2` qui effectue le même type d'opérations.

En procédant de la sorte, ce ne sont plus les valeurs sémantiques qu'il s'agit de préciser, mais il s'agit plutôt d'identifier puis d'articuler les mécanismes qui permettent la construction du sens.

C'est en ce sens que l'on peut caractériser les langages à prototypes comme des langages de programmation contextuelle et les langages à taxonomies de classes comme des langages de représentation. Dans ces derniers, toutes les choses du monde peuvent être décrites sous la forme d'un objet (au sens figé du terme) attaché à une classe définitoire : il s'agit de représenter le savoir que l'objet décrit, la structure définitoire auquel il est attaché est suffisante pour définir cet objet. A l'inverse une approche prototypique ne présume pas de toutes les facettes de ce qui est à représenter : un prototype n'est jamais complètement déterminé quand on le créé. Les circonstances peuvent infléchir la représentation initialement retenue : une redéfinition partielle voir totale de la structure définitoire d'un objet-prototype est toujours possible. A l'inverse, une modification de la définition structurelle d'un objet dans une taxonomie de classes implique le passage obligé par le filtre définitoire que constitue la classe[9] dont il est une instance. Et c'est donc toute la famille d'objets instanciées qui se trouve affectée par cette évolution de représentation.

5.3.4.2. Ajuster les mécanismes de classification pour ajuster les représentations initiales

Il s'agit ici de laisser les représentations ouvertes aux éléments contextuels ou autres qui éclairent leur construction. On a déjà mentionné précédemment un exemple de situation où des éléments contextuels pourraient infléchir la construction du sens d'un composé. C'est le cas de configurations du type "moulin à bras" et "moulin à cheval" pour lesquelles il n'est pas juste d'adapter la représentation des éléments lexicaux en ajoutant un sens `Nénergie` aux entrées lexicales "bras" et "cheval". Nous n'en sommes pas encore à une phase de développement de notre travail qui permette une prise en compte globale des phénomènes contextuels. Malgré tout, il ne faut pas présupposer les résultats à construire, la réflexion sur les mécanismes de catégorisation serait ainsi vite réglée. Il faut au contraire élargir les mécanismes de transferts d'informations entre les agents de l'analyse, même si certains d'entre eux sont considérés dans un premier temps comme vituellement présents même s'ils sont provisoirement incomplètement représentés. On peut ainsi envisager que telle configuration nominale `moulin à N2` ne soit pas effectivement construite sémantiquement car **en attente** d'informations supplémentaires que pourraient par exemple lui fournir des prototypes déjà construits ou à construire, le contexte sous une forme à déterminer, etc. On peut en effet déjà anticiper un modèle de représentation du type d'informations qui seraient pertinentes pour construire des configurations nominales prototypiques et des mécanismes d'envoi de messages qui leur seraient appliqués. Une évolution de ces représentations serait à faire à mesure que les éléments de construction d'une

[8] L'utilisation des primitives `_AddSlots:, _AddSlotsIfAbsent:, _RemoveSlot:` permet de modifier dynamiquement les prototypes créés.
[9] On peut aussi penser à la spécialisation de classe puis à l'instanciation de cette nouvelle classe pour créer un nouvel objet.

analyse globale se mettraient en place. On rejoint là certaines approches à l'oeuvre dans des formalismes par expert de mots qu'il convient de présenter pour mesurer la distance qui existe entre les approches retenues dans ce type de modèle.

5.3.4.3. Un analyseur qui manipule des mots experts : WEP

De même que notre approche privilégie le mot dans la conduite de l'analyse, WEP (Small 1987) considère chaque mot comme la source essentielle des connaissances pour la compréhension, qu'il s'agisse des connaissances que chaque mot a sur lui-même ou des connaissances que chaque mot a sur ses relations avec les autres mots ou les concepts. C'est ainsi que dans ce modèle on attache à chaque morphème linguistiquement intéressant un expert qui est un graphe de décision dont les noeuds correspondent à des actions ou à des questions. Un noeud action utilise des primitives de Langage d'Interaction Lexicale (LIL) ou des primitives du Langage de Différentiation du Sens (LDS) pour :

- Construire ou compléter un concept.
- Envoyer des messages.
- Modifier un état.
- Suspendre l'activité d'un expert.

Accès d'un expert aux informations

Un expert dispose d'un certain nombre d'informations qui concernent la compréhension des phrases précédentes (cette mise en oeuvre est théorique et est définie par des questions soumises à l'utilisateur), la liste des experts en attente d'information, l'état global de l'analyse, la situation des démons postés par les experts de mots. Un démon est un agent qui est mis en place par l'expert déclenché si celui-ci ne peut pas dans l'état présent de son activation répondre positivement par manque d'informations : *"Any word expert that desires some piece of information through interaction with one of its neighbors specifies the nature of that information in a data structure called a restart demon"* (Small 1987).

Activation d'un expert

- Par questionnement d'un autre expert.
- Par le biais d'un de ses démons de réactivation qui a pu construire un résultat.
- Par la prise en compte du mot suivant dans la phrase.

Représentation de la compréhension de la phrase

Un expert est capable de représenter les compréhension de la phrase sous la forme d'objets appelés *concepts*. Un *concept* possède des attributs qui décrivent :

- Une liste des éléments de la mémoire dont le concept est une occurrence.
- Une liste des éléments de la mémoire dont le concept peut-être une occurrence de l'un d'eux.
- Une liste des éléments de la mémoire dont le concept n'est pas une occurrence.

- Une liste de couples indiquant les rôles que peut jouer le concept pour d'autres concepts.
- Son rôle particulier dans la phrase.
- La forme lexicale associée à ce concept.

Comportement de l'expert

Un expert peut exécuter les actions suivantes :

1. Lecture du mot suivant de la phrase.
2. Demande d'informations à d'autres experts.
3. Créer un concept, en complèter un autre.
4. Changer l'état global de l'analyse.
5. Questionner le contexte ou solliciter des connaissances plus générales.
6. Suspendre son exécution.
7. Terminer son exécution.

Mise en oeuvre des actions d'un expert

Les langages LIL et LDS définissent des primitives qui décrivent les actions qu'un expert est susceptible de réaliser. Les primitives du langage LIL permettent de déclencher trois types d'action :

- Mise en place de démons.
- Regard en avant.
- Envoi de messages.

Les primitives du langage LDS permettent quant à elles aux experts :

- D'infléchir leur décision en questionnant le contexte.
- De construire des concepts.
- De placer des démons.

WEP et PàP : confrontations

S'il s'agit pour nous d'essayer de décrire les régularités de certaines configurations linguistiques, l'approche de WEP se situe au contraire dans une volonté de tout décrire en partant des irrégularités de la langue : les régularités de la langue ne décrivant qu'une partie des phénomènes à l'intérieur de l'ensemble des irrégularités. Si une telle approche est éclairante sur la mise en place de contrôle local sur les activités d'une analyse, elle pose malgré tout des problèmes fondamentaux dans la gestion et la cohérence des informations qui sont manipulées au cours de cette analyse.

Autant il est intéressant de pouvoir disposer d'éléments qui réalisent une articulation entre les différents domaines de connaissance sur des endroits critiques d'une analyse (c'est d'ailleurs le point qui nous interesse) et qui mettent en place les opérations de récupération de l'information manquant, autant il paraît difficile de concevoir dans le cadre de la construction du sens, un

contrôle dans le déroulement de l'analyse qui ne serait dispersé que sur les mots. D'une part, on perd toute idée d'un méta-regard sur l'ensemble de l'énoncé à construire, d'autre part, il ne permet pas de prendre en compte certaines régularités de la langue d'une portée plus longue et de ce fait de construire des méta-connaissances de plus grande portée sur les faits de langue. Les limites que l'on peut concéder à WEP sont en fait contenues dans l'approche retenue. En effet si on limite les problèmes de construction du sens d'un énoncé au niveau des mots, une telle approche peut se révéler cohérente mais reste condamnée à ne rendre compte que des phénomènes locaux, alors qu'on sait que les phénomènes à l'oeuvre dans une construction du sens d'un énoncé appréhendé dans sa globalité ne restreignent pas les interactions possibles au niveau du mot.

WEP s'inscrit dans une approche de la représentation des connaissances dans des domaines de savoir où la classification est floue ou mouvante. Si une telle approche priviligie la modélisation des irrégularités des faits de langue, il faut lui adjoindre un module qui "mette de l'ordre" là où il est possible de le faire. Celui-ci doit modéliser les régularités de la langue (et elles ne manquent pas). On dépasserait ainsi le mot et on pourrait prendre en compte des phénomènes plus importants. C'est d'ailleurs ce que nous souhaitons mettre en place dans notre approche prototypique. Plus précisément, les opérations mises en place par WEP peuvent éclairer notre approche prototypique pour résoudre la mise en oeuvre d'une remise à jour automatique des représentations construites. Si nous disposions d'éléments qui gèrent le questionnement, l'attente d'informations..., nous pourrions ne pas rejeter certaines configurations du fait simplement qu'elles n'ont pas encore reçu les informations pertinentes permettant leur construction.

Incise Chapitre 5

Incise 1. Environnement de programmation avec Self

Les éléments présentés infra font référence à la version 3.0 de Self. Pour certains résultats, il s'agit de calculs faits avec la version 2.0, des indications seront données dans ce cas pour identifier les différences minimes qui n'altèrent en rien la présentation générale de Self.

`Lobby`

Il s'agit de l'objet grâce auquel on accède au monde Self. Quand on veut un créer un nouvel objet, on adresse une requête qui doit être comprise dans le contexte défini par cet objet. Les attributs `traits, globals` (dans la version Self-3.0, ou `prototypes` dans les précédentes), `mixins` sont les racines d'objets accessibles à partir du `lobby`.

`Traits`

Cet attribut regroupe les objets qui définissent les comportements partagés par un ensemble de prototypes. Les prototypes se distinguent principalement en ce qu'ils héritent de deux types possibles de comportement : les `traits clonable` définissent la possibilité de multiplier les prototypes par clonage et les `traits oddball` qui spécifie que le prototype qui en hérite est défini comme unique. C'est le cas par exemple de l'objet `true`. A l'inverse, une liste, un vecteur sont des prototypes qu'il est possible de cloner.

`Globals`

Cet attribut regroupe les prototypes (pré-) definis.

`Mixins`

Cet attribut regroupe certains types de comportement qu'il est possible d'ajouter aux objets sans pour autant leur faire partager l'intégralité d'un comportement.

`Shell`

"In Order for a Self expression to be evaluated, the meaning of "self" must be defined. When an expression is evaluated by a read-eval-print loop, the shell object plays the role of self" (Self Group 1993).

On donne ci-contre une présentation graphique du monde Self qui présente l'organisation initiale disponible.

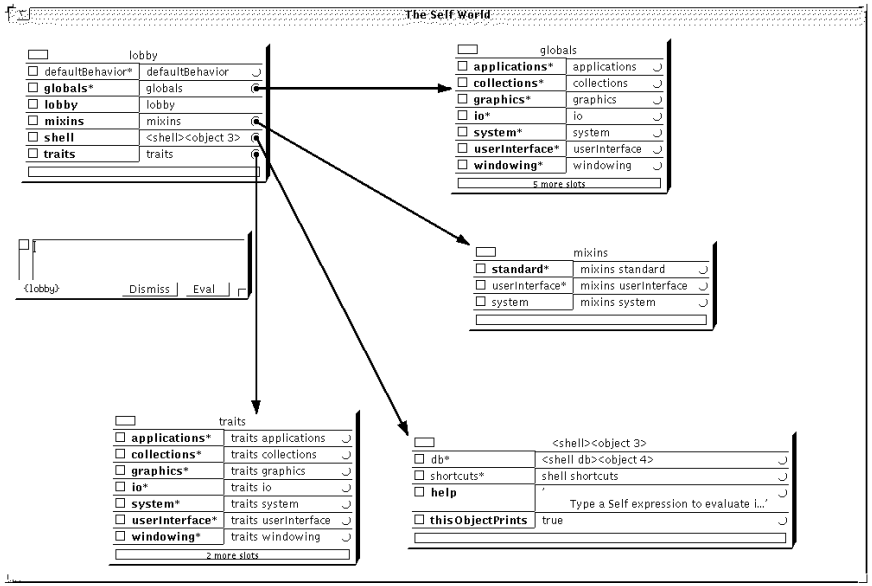

Figure 5.19 : Le monde Self.

"The lobby is a common parent of many object in the system and therefore provides a kind of linguistic common ground for the language. In particular, many useful objects such as true and all the prototype objects are accessible from the lobby. When the virtual machine evaluates the expression you type, messages implicitely sent to self get looked up first in the shell object, then in the lobby, then in the lobby's parents, and so on" (Self Group 1993).

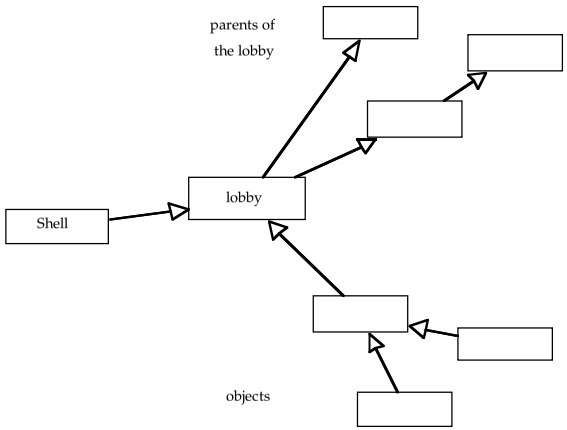

Figure 5.20 : Le `lobby` est au centre du monde Self.

I.1.1. Inspecter les prototypes

La méthode `inspect:` imprime le contenu de chaque attribut de l'objet donné en argument.

I.1.2. Ajouter, supprimer des attributs

Deux primitives permettent d'ajouter des attributs dans l'environnement Self : `_AddSlots`, `_AddSlotsIfAbsent`. Ces deux primitives ajoutent les attributs donnés en argument au receveur du message.

```
_AddSlotsIfAbsent: (| aPrototype = () | )
```

Dans cet exemple, le message d'ajout d'un attribut (si celui-ci n'existe pas) est adressé dans le contexte du `lobby` (par défaut).

```
aPrototype: _AddSlots (| x = 'valeur de l'attribut x du prototype aPrototype'. ^_ y
<- 'valeur de l'attribut y du prototype aPrototype'. | )
```

Ici on adresse au receveur `aPrototype` un message d'ajout de deux attributs notés `x` et `y`. Le premier attribut est constant, le second est un attribut statique public (^_)[10] et assignable. La primitive `_Print` permet de visualiser le prototype ainsi défini :

```
aPrototype _Print
<55>: ( | x =  'valeur de l'attribut x du prototype aPrototype'. ^ y = 'valeur de
l'attribut y du prototype aPrototype'. _ y: <-. | )
```

Chaque objet est numéroté, notre prototype est donc associé au nombre 55. Ce nombre permet de pouvoir faire appel à ce prototype en utilisant la primitive `_AsObject:`

```
(55 _AsObject) _Print
<55>: ( | x =  'valeur de l'attribut x du prototype aPrototype'. ^ y = 'valeur de
l'attribut y du prototype aPrototype'. _ y: <-. | )
```

La primitive `_RemoveSlots` supprime un attribut au receveur du message :

```
aPrototype _RemoveSlot: 'y:'
```

Le résultat de cette suppression de l'attribut d'assignation est le suivant :

```
aPrototype _Print
<55>: ( | x =  'valeur de l'attribut x du prototype aPrototype'. ^ y = 'valeur de
l'attribut y du prototype aPrototype'. | )
```

Les deux primitives `_AddSlots` et `_RemoveSlots` permettent de modifier ponctuellement les prototypes auxquels elles s'adressent. Il est possible de construire ou de redéfinir complètement un prototype en utilisant la primitive `_Define`. Celle-ci supprime tous les attributs du receveur, puis ajoute tous les attributs donnés en argument à ce même receveur.

```
aPrototype _Define: ( |  x = 'valeur de l'attribut x du prototype aPrototype'. y <-
'valeur de l'attribut y du prototype aPrototype'. | )
```

I.1.3. Définir des prototypes

[10] Voir infra pour une présentation complète des restrictions d'accès aux attributs.

Le code ci dessous permet la définition d'un prototype représentant un point :

1.Définition des comportements des points :

```
traits applications _AddSlotsIfAbsent: (| pointTraits = () | )
```

La primitive `_AddSlotsIfAbsent` permet d'ajouter un attribut pour porter les comportements communs des points :

```
traits pointTraits _Define:  (| clone = ( _Clone).
                                print = (x print.String,
                                         '@',y printString. ). |)
```

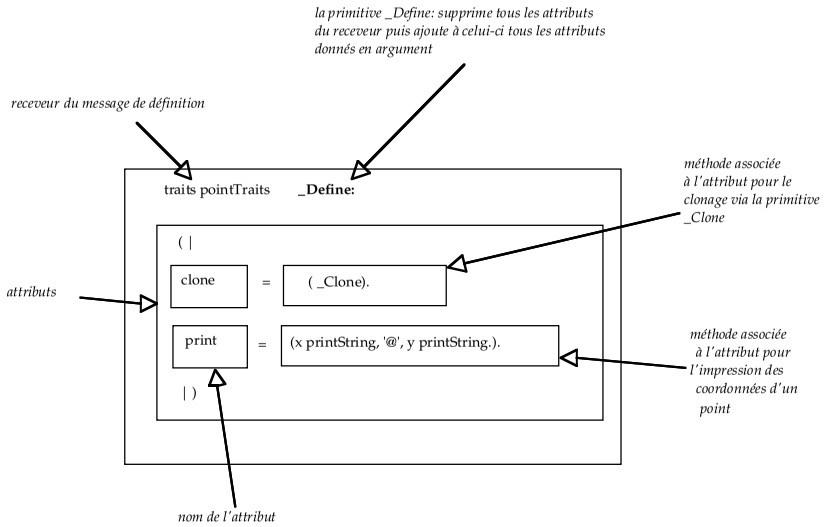

Figure 5.21 : Définition de l'objet regroupant les comportements des prototypes `point`.

2.Définition des prototypes `point` :

```
globals applications _AddSlotsIfAbsent: ( | pointProto = () | )

pointProto _Define: ( | parent* =  traits pointTraits.
                                x <- 0.
                                y <- 0.
                      | )
```


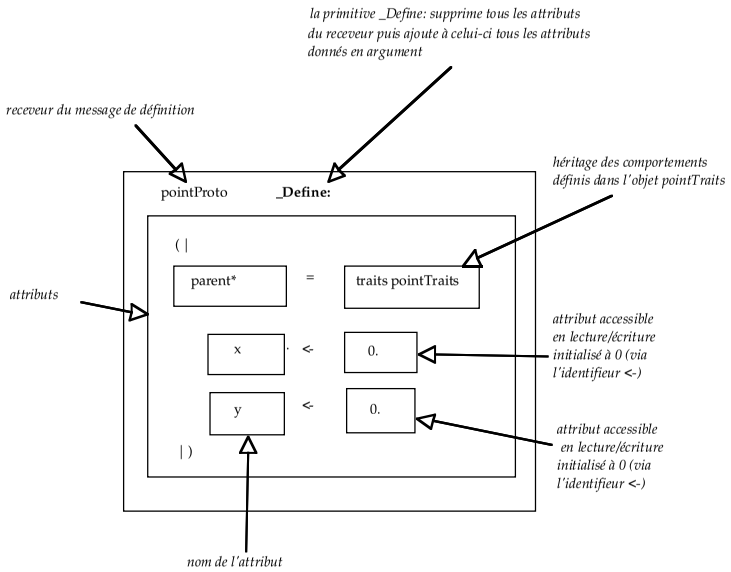

Figure 5.22 : Définition d'un prototype de `point`.

I.1.4. Créer des prototypes

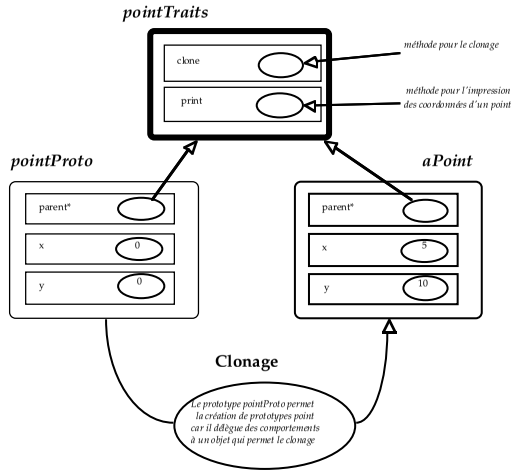

Figure 5.23 : Création d'un prototype de `point`.

La primitive `_AddSlots` ajoute un attribut dans le monde Self :

```
Self> _AddSlots: ( | aPoint | )
shell
```

On peut ensuite adresser à cet attribut les messages combinés de clonage du prototype `point` avec assignation des attributs `x` et `y` aux valeurs données en arguments :

```
Self> aPoint: ((pointProto clone) x: 5) y: 10)
<object 23>
```

Les comportements associés aux prototypes `point` sont décrits ci-dessous :

```
Self> traits pointTraits _Print
<5>: ( |  clone = <a method>.  print = <a method>. | )
nil
ou
Self> (5 _AsObject) _Print
<5>: ( |  clone = <a method>.  print = <a method>. | )
nil
```

La primitive `inspect` permet d'obtenir une visualisation précise de l'implémentation des objets qu'elle décrit :

```
Self> inspect: traits pointTraits
( |  clone = ( _Clone).
     print = (x printString, '@', y printString.). | )
<object 5>
Self> inspect: (5 _AsObject)
( |  clone = ( _Clone).
     print = (x printString, '@', y printString.). | )
<object 5>
```

Le prototype `point`, créé ci-dessus à partir du prototype initialement défini, délègue ses comportements à l'objet `traits pointTraits` (numéroté 5) et possède deux attributs de données `x` et `y`, accessibles en lecture et en écriture (décrit par l'identifieur **<-**) :

```
Self> aPoint _Print     (ou (23 _AsObject) _Print )
<7>: ( | parent* = <5>. x = 5. x: <-.  y = 10. y: <-. | )
nil
Self> inspect: aPoint   (ou  inspect: (23 _AsObject) )
( | parent* = traits pointTraits.
        x  <- 5.
        y <- 10. | )
<object 23>
```

I.1.5. Accès aux attributs des prototypes

```
Self> aPoint x (lecture)
5
Self> aPoint y (lecture)
10
Self> aPoint x: 6 (écriture)
<object 23>
Self> aPoint x
6
Self> inspect: aPoint
( | parent* = traits pointTraits.
           x  <- 6.
           y <- 10. | )
<object 23>
```

La méthode d'impression des coordoonnées d'un point est définie dans l'objet `traits pointTraits` (porteur des comportements communs aux prototypes `point`); on peut donc adresser aux prototypes représentant des points le message suivant :


```
Self> aPoint print
'6@10'
```

I.1.6. Documenter le code

Il est aussi possible de fournir des commentaires sur les objets manipulés :

```
{ ' Cet objet possède un attribut'
( | x = 0. | ) }
```

Incise 2. Illustration par la modélisation des points avec Self (Ungar & al 1987)

I.2.1. Définition d'un objet attaché aux comportements des objets

```
_DefineSlots: ( | traits =
              ( |clonable = ( |
                 copy = (_Clone).
                 |).
                |).
               |)
```

I.2.2. Définition d'un objet attaché aux prototypes

```
_DefineSlots: ( | prototypes = () |)
```

I.2.3. Définition des comportements attachés aux points

Dans le code qui suit on définit des opérations d'affichage et d'addition de point(s) (via les coordonnées de ces points), alors que la définition des prototypes `point` n'a pas été précisée. Cette définition des `traits point` tient compte d'une connaissance implicite de la définition des prototypes `point` qui délègueront les comportements définis ici aux `traits point`. Il est important de noter que l'utilisation des objets `traits` doit être considérée comme un mécanisme qui comble le manque d'organisation d'un langage à prototypes. L'utilisation des objets `traits` permet de stocker au sein de ces objets des comportements généraux utilisables par les prototypes qui permettent de tels comportements. Ce mécanisme permet d'une part d'alléger la définition structurelle des prototypes en ne conservant au sein de cette définition que les éléments propres aux prototypes et d'autre part de définir des comportements généraux accessibles par tous les prototypes définis par l'utilisateur.

```
traits _DefineSlots: (|
point = (|
1 parent* = traits clonable.
2 print = (x printString, '@', y PrintString).
3  +  = ( { méthode pour l'addition des points } ).
  |).
       |)
```

Le code précédent permet:
1 d'hériter d'un comportement,
2 de définir une méthode pour l'impression d'un point,
3 de définir une méthode pour l'addition des points,

I.2.4. Définition d'un prototype de point

```
prototypes _DefineSlots: (| point =
                           (| parent* = traits point.
                              x <- 0.
                              y <- 0.
                            |).
                          |)
```


On donne ci-dessous une illustration graphique de ce modèle de points en Self.

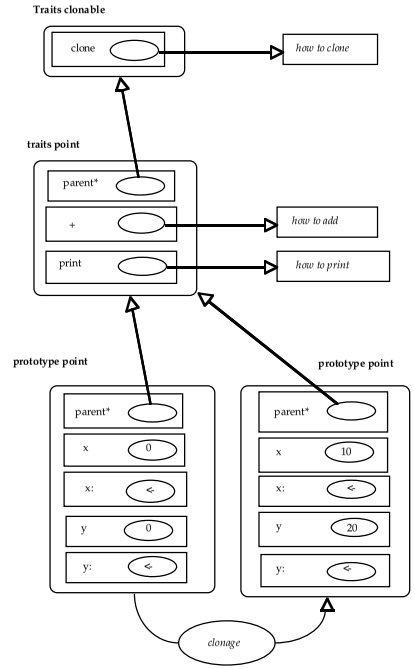

Figure 5.24 : Modèle de points en Self.

Dans ce type de modèle de prototypes, les points sont donc des objets possèdant deux attributs mutables `x` et `y`. Ils délèguent des comportements à l'objet `traits point` qui définit la somme de ces points ainsi que l'impression des coordonnées d'un point. Cet objet parent délégue le comportement de clonage à l'objet `traits clonable`.

Incise 3. Illustration par la modélisation des Noms avec Self

I.3.1. Définitions des objets attachés aux comportements des prototypes

```
1 traits  applications _AddSlotsIfAbsent: ( | catN = () | )
2 traits catN _Define: ( |
3                        parent* = traits clonable.
4                        copy    = (clone).
5                        categorie <- 'Nom'.
6                         display = ( ('<Catégorie: ', categorie printString,'>')
print.
                                     morph display.
                                     accord display.
                                ).
                         | )
```

Le code ci dessus permet:
1 d'adresser à l'objet `traits applications` un message de création d'un attribut si celui-ci n'existe pas,
2 d'adresser à l'attribut créé un message de définition,
3 de définir un héritage,
4 de définir une méthode de duplication,
5 de définir un attribut qui portera la valeur catégorielle,
6 de définir un attribut pour l'affichage des attributs des prototypes créés.

La définition de ce réservoir de comportements qui seront accessibles par les prototypes déléguant ce type de comportements à l'objet ainsi créé constitue en fait un regard en avant sur ce que sera une partie de la définition des prototypes non encore définis. On pré-suppose en effet que les prototypes déléguant les comportements définis possèderont certains attributs dont les noms sont utilisés avant même qu'ils ne soient inscrits dans la définition des prototypes. Le comportement `display` s'applique sur des attributs `morphologie` et `accord` qui n'ont pas encore été définis. On procède de même avec les objets suivants:

```
traits applications _AddSlotsIfAbsent: ( | morphologie = () | )
traits morphologie _Define: ( |
    parent* = traits clonable.
    copy    = (clone).
    display = (
                ( ' <Lemme: ', lemme  printString,'>
                    <Forme: ', forme printString,'>
                    <Forme Plur: 'formePlur printString,'>
                    <Forme Sing: ', formeSing printString, '>') print.
                ( ' <Classe: ', classe printString, '>') print.
              ).
| )
traits applications _AddSlotsIfAbsent: ( | accord = () | )
traits accord _Define: ( |
  _ parent* = traits clonable.
  ^ copy    = (clone).
  ^ display = (
     (' <Genre: ', genre type_genre printString,'>
        <Nombre: ', nombre type_nombre printString, '>') print.
             ).
| )
traits applications _AddSlotsIfAbsent: ( | genre = () | )
traits genre _Define: ( |
  _ parent* = traits clonable.
```



```
  ^ copy    = (clone).
| )
traits applications _AddSlotsIfAbsent: ( | nombre = () | )
traits nombre _Define: ( |
  _ parent* = traits clonable.
  ^ copy    = (clone).
| )
```

I.3.2. Définitions des prototypes

```
globals applications _AddSlotsIfAbsent: ( | catN = () | )
catN _Define: ( |
    parent* = traits catN.
    morph <- ''.
    accord <- ''.
| )
globals applications _AddSlotsIfAbsent: ( | accord = () | )
accord _Define: ( |
  _ parent* = traits accord.
  ^ genre <- ''.
  ^ nombre <- ''.
| )
globals applications _AddSlotsIfAbsent: ( | morphologie = () | )
morphologie _Define: ( |
  _ parent* = traits morphe.
  ^ lemme <- ''.
  ^ forme <- ''.
  ^ formeSing <- ''.
  ^ formePlur <- ''.
  ^ classe <- ''.
| )
globals applications _AddSlotsIfAbsent: ( | genre = () | )
genre _Define: ( |
  _ parent* = traits genre.
  ^ type_genre <- ''.
  ^ marque_genre <- ''.
| )
globals applications _AddSlotsIfAbsent: ( | nombre = () | )
nombre _Define: ( |
  _ parent* = traits nombre.
  ^ type_nombre <- ''.
  ^ marque_nombre <- ''.
| )
```

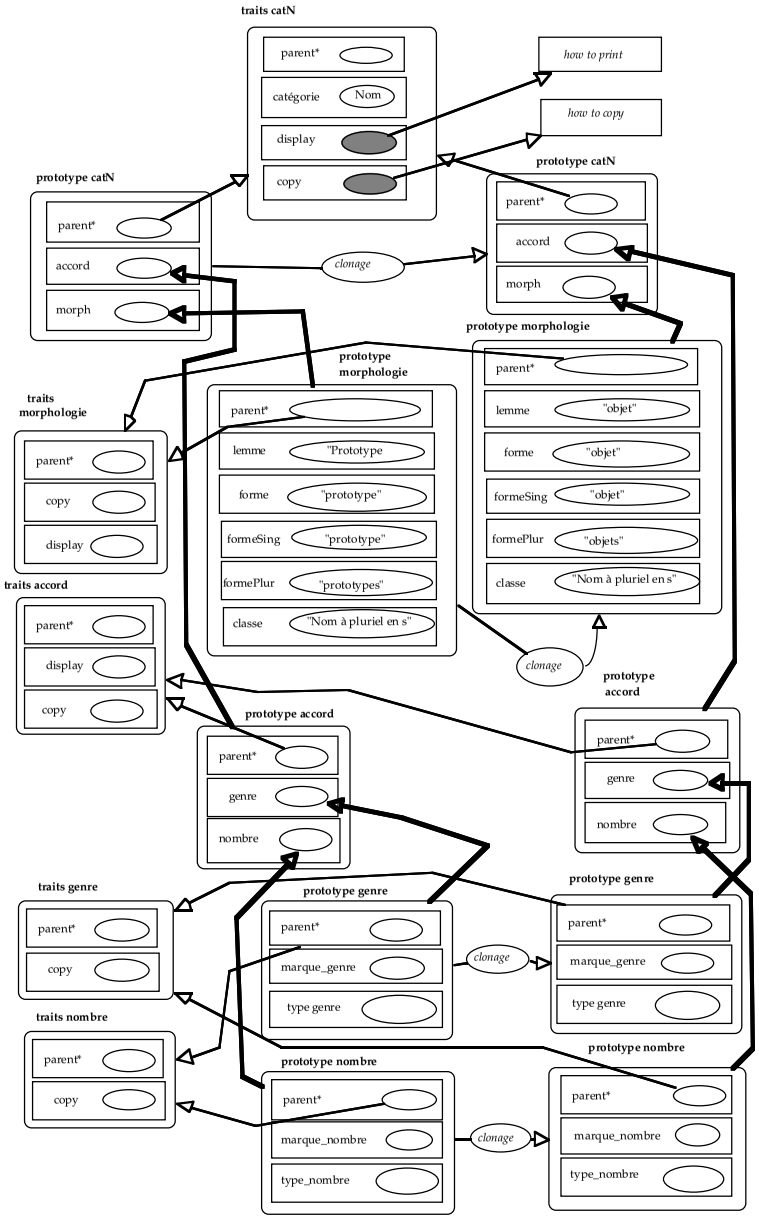

Figure 5.25 :Un modèle de représentation des `Noms` en Self.