Vers l'utilisation des méthodes formelles pour le développement de linguiciels

Bilel Gargouri, Mohamed Jmaiel, Abdelmajid Ben Hamadou

Laboratoire LARIS
FSEG-SFAX, B.P. 1088
3018 SFAX, TUNISIA
E-mail: {Abdelmajid.Benhamadou@fsegs.rnu.tn}

Abstract

Formal methods have'nt been applied enough in the development process of lingware although their advantages have been proved in many other domains. In this framework, we have investigated some applications dealing with different processing levels (lexical analyses, morphology, syntax, semantic and pragmatic). These investigations has mainly led to the following observations. First of all, we have noticed a lack of use of methodologies that cover all the life cycle of a software development. The formal specification has not been used in the first development phases. In addition, we have noticed the lack of formal validation and consequently the insufficient guarantee of the developed software results. Moreover, there has been no appeal to rigorous methods of integration to solve the dichotomy of data and processing problem. However, the use of the formal aspect in the Natural Language Processing (NLP) has generally been limited to describing the natural language knowledge (i.e., grammars) and specifying the treatments using algorithmic languages. Few are those who have used a high level specification language.

This paper focuses on the contributions of formal methods in developing natural language software starting from an experimentation carried out on a real application and which consists in specifying and validating the system CORTEXA (Correction ORthographique des TEXtes Arabes) using the VDM formal method.

First of all, we review the advantages of formal methods in the general software development process. Then, we present the experimentation and the obtained results. After that, we place the formal methods advantages in the context of NLP. Finally, we give some methodological criteria that allow the choice of an appropriate formal method.

Résumé :

Les méthodes formelles n'ont pas été suffisamment utilisées dans le processus de développement des linguiciels, alors qu'elles ont fait leurs preuves dans d'autres domaines. Le présent article essaye de mettre en relief les avantages des méthodes formelles dans le contexte des langues naturelles, partant des résultats d'une expérience réalisée sur une application réelle. Dans un premier temps, nous rappellons les avantages globaux des méthodes formelles dans le processus de développement d'un logiciel. Ensuite, nous plaçons ces avantages dans le contexte des langues naturelles. Enfin, nous donnons les critères méthodologiques pour le choix d'une méthode formelle appropriée.

1 Introduction

L'automatisation des langues naturelles a bénéficié jusqu'à nos jours de nombreuses années de recherches et continue encore à faire l'objet de plusieurs travaux, notamment dans le domaine du génie linguistique pour le développement d'applications spécifiques.

L'étude des approches de développement des applications liées au Traitement Automatique des Langues Naturelles (TALN), à tous ses niveaux (i.e., lexical, morphologique, syntaxique, sémantique et pragmatique), (Fuchs, 1993; Sabah, 1989) nous a permis de constater une quasi-absence de l'utilisation de méthodologies de développement qui intègrent toutes les phases du cycle de vie d'un logiciel. En particulier, au niveau des premières étapes, nous avons constaté l'absence quasi-totale de la phase de spécification formelle.

D'un autre côté, nous avons constaté une difficulté, voire absence de validation formelle des approches utilisées dans le développement et par conséquent de garantie sur les performances des résultats obtenus. De même, nous avons remarqué le non recours à des méthodes rigoureuses d'intégration pour résoudre le problème de la dichotomie données-traitements.

L'utilisation des outils formels s'est limitée, dans la plupart des cas, à la description du langage (i.e., les grammaires) et à la spécification des traitements réduite, généralement, à l'usage d'un langage algorithmique. Rares sont ceux qui ont utilisé un langage de spécification formelle de haut niveau (Zajac, 1986; Jensen et al., 1993).

Après une présentation des avantages qu'offrent les méthodes formelles dans le processus de développement d'un logiciel, d'une manière générale, cet article essaye de mettre en relief les avantages specifiques au domaine de TALN partant d'une expérience menée au sein de notre équipe en utilisant la méthode VDM (Dawes, 1991; Jones, 1986). Il donne, à la fin, des critères permettant le choix d'une méthode formelle appropriée.

2 Rappel des principaux avantages des méthodes formelles

L'intégration des méthodes formelles dans le processus de développement de certaines applications critiques comme les systèmes temps réel et les systèmes distribués a donné ses preuves ces dernières années (Barroca and Dermid, 1992; Dick and Woods, 1997; Ledru, 1993). C'est ce qui a motivé leur utilisation dans le développement de logiciels traitant des problèmes complexes au niveau industriel (Hui et al., 1997).

Une méthode formelle est considérée comme une démarche de développement de logiciels basée sur des notations mathématiques et des preuves de validation formelles (Habrias, 1995). Cette démarche utilise un processus de raffinement qui part d'une spécification abstraite des besoins pour déboucher sur une spécification raffinée et exécutable (ou directement codable en un langage de programmation). Les principaux avantages des méthodes formelles peuvent être résumés dans les points suivants :
La précision et la non ambiguité : l'utilisation d'un langage basé sur des notations formelles et précises permet d'éviter toute ambiguité et toute redondance dans la spécification.
La détection d'erreurs conceptuelles le plus tôt possible : l'application de preuves de validation de la spécification tout le long du processus de raffinement de cette dernière, garanti la détection des erreurs de conception le plus tôt possible dans le processus de développement de l'application. En l'absence d'une telle validation, les erreurs de conception ne seront

détectées qu'après la phase d'implémentation ce qui engendrera un côut supplémentaire.
La satisfaction de la conception (éventuellement de l'implémentation) par rapport aux besoins : elle est garantie grâce au processus de raffinement qui part d'une spécification des besoins et applique des règles cohérentes de transformation pour aboutir à la conception finale.
Le contrôle de la cohérence données-traitements : qui est directement pris en charge grâce aux preuves de validation.
La réutilisation : le raffinement des spécifications formelles et leurs décompositions successives permettent de mettre en évidence des niveaux d'abstraction intéressants pour la résolution du problème et pour promouvoir la réutilisation (des spécifications).

3 Présentation et résultats de l'expérimentation

3.1 Choix et démarche utilisée

Pour mesurer l'impact de l'utilisation des méthodes formelles dans le contexte du TALN, nous avons effectué la spécification complète et validée du système CORTEXA (Correction ORthographique des TEXtes Arabes) (Ben-Hamadou, 1993) développé au sein de notre laboratoire.

Outre la disponibilité de la documentation, en matière de conception et d'implémentation, le choix du système CORTEXA est aussi motivé par la diversité des approches utilisées pour la représentation des connaissances et des traitements. En effet, il se compose :

- d'un module de détection des erreurs basé sur une analyse affixale qui utilise un système à états finis (les réseaux de transitions augmentées : ATN). L'analyse affixale effectue la décomposition d'un mot en ses composants premiers : préfixe, infixe, suffixe et racine en se référant à un ensemble de lexiques et de structures de données,
- d'un module de correction des erreurs orthographiques qui utilise un système à base de règles et
- d'un autre module de correction des erreurs typographiques qui se base sur un système mixte.

Le choix de VDM pour la spécification de CORTEXA est motivé, d'une part, par le fait que cette méthode se base sur les prédicats qui donnent un haut pouvoir expressif, et d'autre part, pour sa notation simple et riche. Aussi, VDM a fait ses preuves dans le développement de plusieurs systèmes d'information. Contrairement aux environnements de spécification des données linguistiques tels que D-PATR (Karttunen, 1986), EAGLES (Erbach et al., 1996), etc, VDM permet de spécifier à la fois des traitements et des données (dans notre contexte des données linguistiques) et offre une méthodologie de développement d'applications se basant sur des raffinements et des transformations validées.

Partant de la description informelle des besoins, nous avons développé la spécification abstraite du système CORTEXA (appelée aussi spécification implicite) qui englobe, entre autres, la spécification formelle de ses fonctions, de ses actions et de ses règles de correction. Cette spécification a été, ensuite, validée en utilisant des preuves formelles. Enfin, nous avons généralisé la spécification de conception (appelée aussi spécification explicite ou directe) à partir de la spécification abstraite moyennant des règles relatives à la méthode VDM. Cette spécification de conception est facilement transformée en code pour réaliser la phase d'implémentation.

3.2 Résultats obtenus

L'utilisation de la méthode formelle VDM pour la spécification complète et validée du système CORTEXA a conduit, entre autres, aux constats suivants :
Insuffisance en règles : l'utilisation des preuves formelles nous a permis de mettre en relief, par rapport à la spécification initiale, certaines situations non prises en compte. En particulier, les preuves qui permettent de s'assurer que pour chaque type d'erreur doit exister au moins une règle de correction nous ont permis de constater que l'ensemble des règles de correction, initialement proposé, ne permet pas de prendre en charge toute la typologie d'erreurs.
Exemple 1: preuve relative à l'erreur de sup-

pression

$$(\forall w' \notin CH,\ \forall w \in Lex).(Del(w,\ w') \wedge w' \notin Lex) \Longrightarrow (\exists R \in Reg).w \in R(w')$$

où
Lex : le lexique de référence
CH : l'ensemble des séquences de chaines de caractères
Reg : l'ensemble des règles de correction
R(w) : l'application de la règle R sur la chaine w. On représente une règle en VDM par une fonction
Del() : un prédicat qui vérifie l'erreur de suppression de caractère.

La précision et la concision de la spécification : en comparant la spécification informelle du système CORTEXA, présentée dans la documentation, avec la spécification formelle développée, nous remarquons que cette dernière est plus précise et plus concise. L'exemple 2, donné ci-après, qui présente la spécification formelle de la fonction de génération des décompositions affixales possibles d'un mot w, illustre ce constat.

Exemple 2:

$$Isdecomp(w, p, i, s, root : CH)r : B$$
pre $True$
post $\exists a, b \in CH\ (w = p \bullet a \bullet i \bullet b \bullet s \wedge root = a \bullet b) \wedge (Sprefixe(w, p) \wedge Ssuffix(w, s) \wedge Sinfixe(w, i))$

où
B : le type booléen
Sinfixe() (respectivement *Sprefixe()* et *Ssuffixe()*) : un prédicat qui vérifie la propriété d'un infixe (respectivement d'un préfixe et d'un suffixe) pour une chaine.

Facilité du développement du code : la spécification de conception obtenue est suffisamment explicite pour les données et algorithmique pour les traitements. Elle est donc facilement codable en un langage de programmation. L'exemple 3, illustre l'usage d'une notation algorithmique dans la spécification des fonctions. Il présente la fonction S-Radical de vérification de la propriété d'un radical (formé par la racine et l'infixe).

Exemple 3:

$$Sradical : CH \times CH \longrightarrow B$$
$$Sradical(s_1, s_2) =_{Def} \textbf{if } s_1 = [\,] \textbf{ then } False \textbf{ else if } Sprefixe(s_1, s_2) \textbf{ then } True \textbf{ else } Sradical(tl(s_1), s_2)$$

où
tl() : une fonction VDM qui retourne la séquence en entrée privée de sa tête.

Unicité de la notation : les méthodes formelles permettent d'utiliser la même notation pour décrire aussi bien les données que les traitements. En effet, avec le langage VDM-SL, associé à VDM, nous avons pu spécifier toutes les fonctions et les données de référence de CORTEXA. Les exemples 4 et 5 illustrent cette unicité pour la représentation des données composées et des fonctions.

Exemple 4 : l'enregistrement relatif aux données d'une décomposition d'un mot en un préfixe, un infixe, un suffixe et une racine.

$$Decomp :: p : CH \quad i : CH \quad s : CH \quad r : CH$$

Exemple 5: spécification de l'action qui génère les propositions de correction des suffixes par suppression de caractère

$$A_3s(p : CH, c : CHAR)SCand : set\ of\ CH$$
pre $True$
post $\exists\ a, b, p_1 \in CH\ p = a \bullet c \bullet b \wedge p_1 = a \bullet b \wedge p_1 \in Suff \Longrightarrow \{p_1\} \subseteq SCand$

où
CHAR : l'ensemble des caractères
SCand : les suffixes candidats à la correction
Suff : l'ensemble des suffixes.

Cohérence données-traitements : l'unicité de la notation, a permis d'appliquer des preuves formelles à la fois sur des données et des traitements et par conséquent de contrôler la cohérence de ces derniers. L'exemple 1 illustre ce contrôle dans le cas d'un système à base de règles.

La validation de chaque composant du système : pour chaque composant ou module du système CORTEXA, nous avons appliqué les preuves de validation appropriées, ce qui nous a permis de valider tous les résultats partiels du système. Le théorème de l'exemple 6, donné ci-après, permet de prouver qu'à la suite de l'application de la règle de correction d'une erreur de substitution, les propositions de correction obtenues appartiennent au lexique.

Exemple 6:

$$\forall w' \in CH, \forall w \in Lex.Sub(w, w') \Longrightarrow \exists R \in Reg.R(w') \subseteq Lex$$

où

Sub : un prédicat qui vérifie l'erreur de substitution de caractères.

4 Intérêts des méthodes formelles en génie linguistique

Cette expérimentation, bien qu'elle soit assez limitée dans le temps (elle a duré une année environ) et dans son contexte (elle s'est intéressé à un seul système et non à plusieurs), elle nous a permis d'apprécier à juste titre l'intérêt de recourir aux méthodes formelles dans le processus de développement des applications liées au TALN. Elle nous a aussi permis de dégager certains avantages globaux dédiés au domaine du TALN qui viennent consolider ceux que nous avons déjà cités dans un cadre général de développement des logiciels. Ces avantages spécifiques peuvent être résumés et argumentés dans les points qui suivent.

D'abord, au niveau de la spécification des besoins, les applications du TALN sont généralement très ambitieuses au départ. Or on connaît aujourd'hui les limites des modèles linguistiques et des outils de représentation des connaissances. L'utilisation d'outils formels dans les premières étapes de développement (i.e., analyse) permet de mettre très vite en évidence les limites du système à développer, en particulier, sur le plan de la couverture linguistique et par conséquent de partir pour l'étape de conception sur une version validée du système qui sera implémenté et de prévoir d'emblé les possibilités d'extention et de réutilisation.

Par ailleurs, la complexité des traitements liés au langage naturel et la diversité des données linguistiques et des fortes intéractions qui existent entre données et traitements rendent la tâche de conception très difficile et pouvant engendrer des problèmes d'incohérence. L'utilisation des méthodes formelles au niveau de la conception permet d'abord, de gérer la dichotomie données-traitements soit par l'intégration (i.e., en utilisation l'approche objet), soit par le contrôle de cohérence (i.e., par des preuves de validation) et ensuite de mettre en évidence, par des regroupements et des raffinements successifs, des abstractions intéressantes réutilisables telsque des modules ou des sous-systèmes pouvant être disponibles dans une bibliothèque (Darricau et al., 1997). Ces abstractions correspondent par exemple à des modules standards du TALN traitant le niveau phonétique, morphologique, syntaxique, etc. Notons à ce propos que, la réutilisation de spécifications (i.e., de conception) peut se faire directement ou moyennant des adaptations. Les méthodes formelles offrent des environnements qui facilitent ces adaptations (éditeurs,..) et qui permettent la validation des nouvelles spécifications.

Enfin, l'utilisation d'une notation uniforme donne la possibilité d'intégrer dans la même application une variété de connaissances sur la langue spécifiées avec des formalismes différents (i.e., grammaires d'unification, HPSG, Grammaires Formelles, etc). Ce qui permettera d'avoir une meilleure cohérence dans la spécification finale à produire.

5 Les critères de choix d'une méthode formelle pour le TALN

L'utilisation de la méthode VDM pour la spécification complète et validée du système CORTEXA a été à titre d'essai. Toutefois, le choix d'une méthode formelle pour le développement d'une application de TALN reste crucial. Ce choix doit tenir compte des spécificités du domaine des langues naturelles sur le plan du langage de spécification et sur celui de la méthodologie appliquée. Dans ce qui suit, nous donnons quelques critères que nous jugeons pertinents dans le choix d'une méthode formelle dans le contexte de TALN :

- Le pouvoir expressif de la méthode : possibilité d'intégrer dans la même spécification des connaissances linguistiques variées décrites avec des formalismes différents. Le langage de spécification doit pouvoir unifier la représentation des différentes expressions. Le pouvoir expressif concerne aussi la spécification conjointe des données linguistiques et les traitements qui leurs sont appliqués.
- Simplicité de la notation et de la méthodologie de développement.
- Couverture maximale du cycle de vie du logiciel à développer.
- Existance d'Ateliers de Génie Logiciel (AGLs) qui supportent la méthode.
- Possibilité de supporter l'architecture du système envisagé (i.e., séquentielle, distribuée, parallèle, etc).

6 Conclusion

L'utilisation des méthodes formelles dans le contexte des langues naturelles permet, non seulement de consolider les avantages globaux de ces methodes dans le cadre général de développement de logiciels, mais aussi de rapporter de nouveaux profits spécifiques au domaine. Cette utilisation concerne aussi bien le processus de développement des applications que leur maintenance. Cependant, le choix d'une méthode appropriée reste lié à la disponibilité d'outils logiciels associés qui facilitent sa mise en oeuvre et à la construction d'une bibliothèque de spécifications réutilisables.

Actuellement, nos travaux se concentrent sur la finalisation d'une approche que nous avons développée pour généraliser l'utilisation des méthodes formelles (VDM ou autres) dans le processus de développement des linguiciels. Cette approche intègre les principaux formalismes existants de description des connaissances linguistiques (i.e., Grammaires d'Unification, Grammaires Formelles, HPSG, etc).

References

L. M. Barroca and J. A. Mc Dermid. 1992. Formal methods : use and relevance for the development of safety-critical systems. *The Computer Journal*, 35(6).

A. BenHamadou. 1993. *Vérification et correction automatiques par analyse affixale des textes écrits en langage naturel : le cas de l'arabe non voyellé.* Ph.D. thesis, Faculté des Sciences de Tunis. Thèse Es-Sciences en Informatique.

M. Darricau, H. Hadj Mabrouk, and J.G. Ganascia. 1997. Une approche pour la réutilisation des spécifications de logiciels. *Génie Logiciel*, (45):21–27, September.

J. Dawes. 1991. *The VDM-SL reference guide.* Pitman Publishing.

J. Dick and E. Woods. 1997. Lessons learned from rigorous system software development. *Information and Software Technology*, 39:551–560.

G. Erbach, J. Dorre, S. Manandhar, and H. Uszkoreit. 1996. A report on the draft eagles encoding standard for hpsg. In *Actes de TALN-96*, Marseille, France, May.

C. Fuchs. 1993. *Linguistique et Traitements Automatiques des Langues.* Hachette.

H. Habrias. 1995. Les spécifications formelles pour les systèmes d'informations quoi ? pourquoi ?comments ? *Ingénierie des systèmes d'information*, 3(2):205–253.

J. Hui, L. Dong, and X. Xiren. 1997. Using formal specification language in industrial software development. In *Procedings of the IEEE International Conference on Intelligent Processsing Systems*, pages 1847–1851, Beijing, China, October.

K. Jensen, G.E. Heidorn, and S. D. Richardson. 1993. *NLP: The PLNLP Approach.* Kulwer academic publishers.

C. B. Jones. 1986. *Systematic software development using VDM.* Printice Hall.

L. Karttunen. 1986. D-patr : A development environment for unification-based grammars. In *In Proceedings of the 11th International Conference on Computational Linguistics*, pages 74–80, Bonn, Germany.

Y. Ledru. 1993. Developing reactive systems in a vdm framework. *Science of Computer Programming*, 20:51–71.

G. Sabah. 1989. *L'intelligence artificielle et le langage.* Hermes.

R. Zajac. 1986. Scsl : a linguistic specification language for mt. In *Procedings of COLING'86*, pages 25–92, Bonn, Germany, August.