La mise en œuvre et l'évaluation du système TIMINF

1) Introduction

Dans ce chapitre nous allons expliquer l'installation des différents outils utilisés pour aboutir à notre objectif. Nous donnons aussi un exemple de déroulement de notre système qui résume les principales spécifications de notre projet et montre comment les différents modules peuvent être mis en œuvre dans un système de test d'inférence textuelle intégrant l'aspect temporel dans ses décisions. Nous finissons ce chapitre avec l'évaluation de notre système.

2) Environnement et outils utilisés

2.1) Python

Pour concevoir notre système nous avons choisi d'utiliser le langage de programmation Python qui a fait ses preuves dans la programmation de nombres applications du TALN.
Python est un langage portable, dynamique, extensible, gratuit, qui permet une approche modulaire et orientée objet de la programmation. Python est développé depuis 1989 par Guido van Rossum et de nombreux contributeurs bénévoles (Swinnen, 2005).

L'interpréteur peut être lancé directement depuis la ligne de commande (dans un « shell » *Linux*, ou bien dans une fenêtre *DOS* sous *Windows*) : il suffit d'y taper la commande **"python"** (en supposant que le logiciel lui-même ait été correctement installé).
Nous utilisons une interface graphique telle que *Windows*. Pour cela nous avons préféré travailler dans un environnement de travail spécialisé tel que ***IDLE***.
Avec *IDLE* sous *Windows*, notre environnement de travail ressemblera à celui-ci :
Les trois caractères « supérieur à » constituent le signal d'invité, ou *prompt principal*, lequel indique que Python est prêt à exécuter une commande.

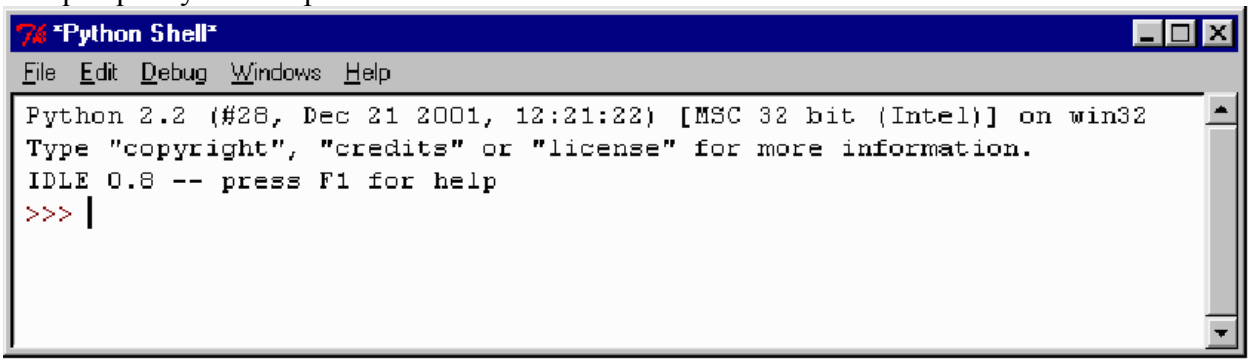

Figure 5.1 : Shell python

Pour rédiger nos séquences d'instructions nous avons utilisé l'éditeur incorporé dans une interface de développement telle que *IDLE).* Il serait parfaitement possible d'utiliser un système de traitement de textes, à la condition d'effectuer la sauvegarde sous un format "texte pur" (sans balises de mise en page). Il est cependant préférable d'utiliser un véritable éditeur ANSI "intelligent" tel que *nedit* ou *IDLE*, muni d'une fonction de coloration syntaxique pour Python, qui aide à éviter les fautes de syntaxe.
La figure ci-dessous illustre l'utilisation de l'éditeur *IDLE).* Sous (*windows)* :

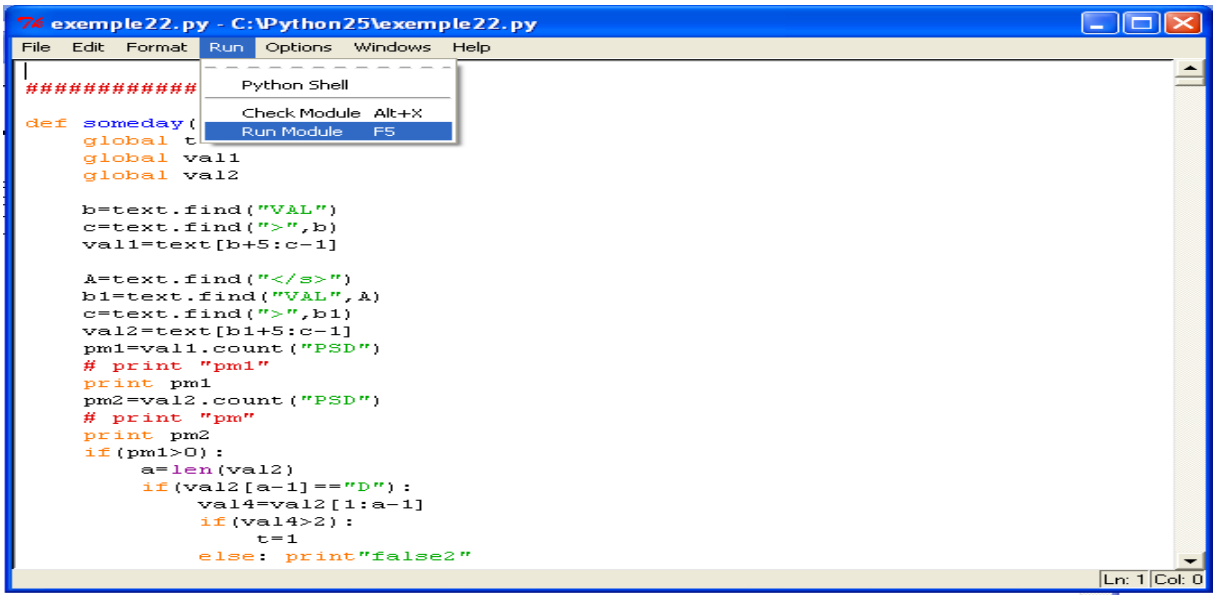

Figure 5.2 : Comment exécuter un programme

Par la suite, pour tester l'exécution de notre programme, il nous suffit de lancer l'interpréteur Python en lui fournissant (comme argument) le nom du fichier qui contient le script.
Par exemple, si nous avons placé un script dans un fichier nommé « MonScript », il suffira d'entrer la commande suivante dans une fenêtre de terminal pour que ce script s'exécute :
python MonScript
Dans l'*explorateur Windows,* nous pouvons lancer l'exécution de notre script en effectuant un simple clic de souris sur l'icône correspondante ou dans *IDLE*, en lançant l'exécution du script en cours d'édition, directement à l'aide de la combinaison de touches <Ctrl-F5>.

2.2) TARSQI

Nous décrivons dans ce qui suit le processus d'installation de TARSQI dans un environnement Linux puisqu'il n'existe pas actuellement une version Windows de TARSQI. Toutefois, le code est écrit pour être multiplateforme. Le groupe TIMEML travaille actuellement sur une version de TARSQI adapté pour Windows qui sera publiée dès que possible.

2.2.1) L'installation

La boîte à outils requiert au moins la version 2,3 de Python et la version 5,8 de Perl. La boîte à outils a été testée sur les plates-formes suivantes:
Red Hat Linux 5, avec Python 2.4.3 et Perl 5.8.8
Mac OS X, avec Python 2.3.5 et Perl 5.8.8

Pour installer TARSQI, nous avons d'abord téléchargé et décompresser l'archive dans un répertoire et, taper dans l'invité de commande ce qui suit :

```
% Gunzip-c TTK-1.0.tar.gz | tar xp
```

Cette commande permet de décompresser le contenu dans un répertoire nommé `TTK-1,0,` qui est un répertoire choisi par nous.

La boîte à outils TARSQI est conçue pour fonctionner de façon transparente avec le SGI TreeTagger. Le TreeTagger doit être installé dans
`ttk-1.0/code/components/preprocessing/treetagger/`
Ce répertoire doit avoir des sous-répertoires `bin` et `lib`.

2.2.2) L'utilisation de la boite à outils TARSQI

Pour exécuter l'outil TARSQI, nous devons ouvrir un terminal, aller au répertoire où se trouve le fichier tarsqi.py et taper :

```
python tarsqi.py <input_type> [drapeaux] <infile> <outfile>
```

`<input_type>`: Il existe deux formats d'entrée de TARSQI : `simple-xml` et `rte3`.

`[drapeaux]`: Avec les drapeaux nous pouvons exécuter un seul ou plusieurs module de TARSQI où l'ordre des modules est important. En voici un exemple:

`[drapeaux]=` préprocesseur, GUTIME, EVITA

L'exemple montre une demande d'exécution des 3 premiers modules de TARSQI.

2.2.3) L'utilisation de la boite à outils d'interface graphique

La Boîte à outils d'interface graphique peut être utilisée en tapant :

```
% Pythonw gui.py
```

L'interface graphique a trois avantages sur l'utilisation de la version en ligne de commande:

- Il est plus rapide lors de l'utilisation sur un fichier par fichier, parce que toutes les bibliothèques sont chargées soit au démarrage ou lorsque le premier fichier est traité.
- Il est plus facile à utiliser.
- Il permet à l'utilisateur de taper certains points d'entrée et voir ce qui se passe.

Le principal inconvénient est qu'il n'est pas possible de traiter tous les fichiers dans un répertoire. Voici une capture d'écran:

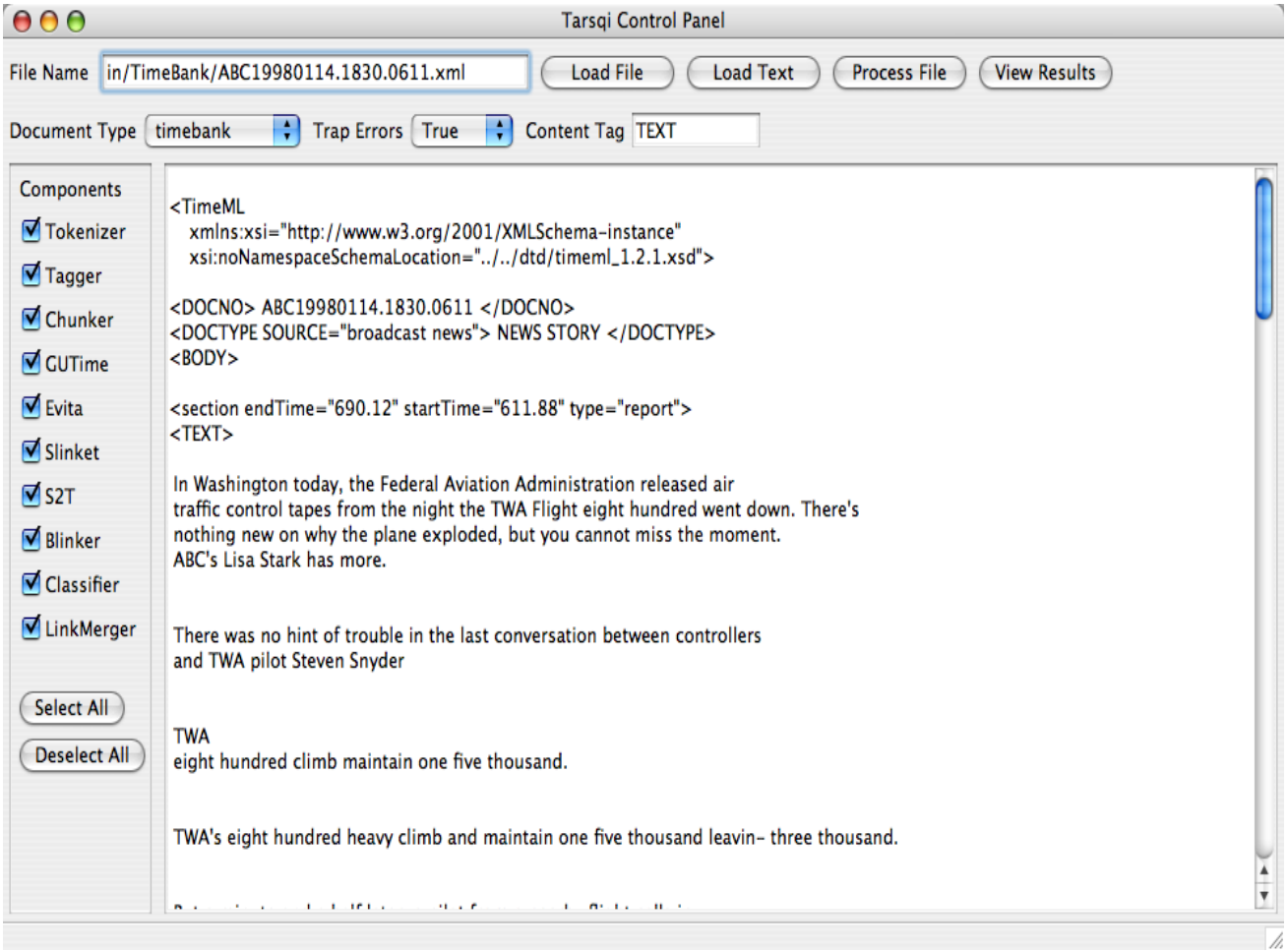

Figure 5.3 : Capture d'écran de l'interface TARSQI

Les fonctionnalités peuvent être résumées comme suit:

- Utilisez "Chargez le fichier" pour sélectionner un fichier à traiter.
- Utilisez "Texte de charge" à saisir du texte. Cette opération va créer un fichier dans `le` dossier data `/ en /` répertoire `utilisateur,` qui est ensuite sélectionné comme fichier d'entrée.
- Utilisez « Processus de dossier » pour traiter le fichier d'entrée conformes aux paramètres sélectionnés.

2.3) Link Parseur

L'installation de ce module n'est pas difficile, puisque après avoir téléchargé le système de puis le lien suivant (http://www.abisource.org/projects/link-grammar/), nous avons décompressé le contenu dans un répertoire de notre choix. Il suffit d'un click sur l'exécutable contenu dans le répertoire.

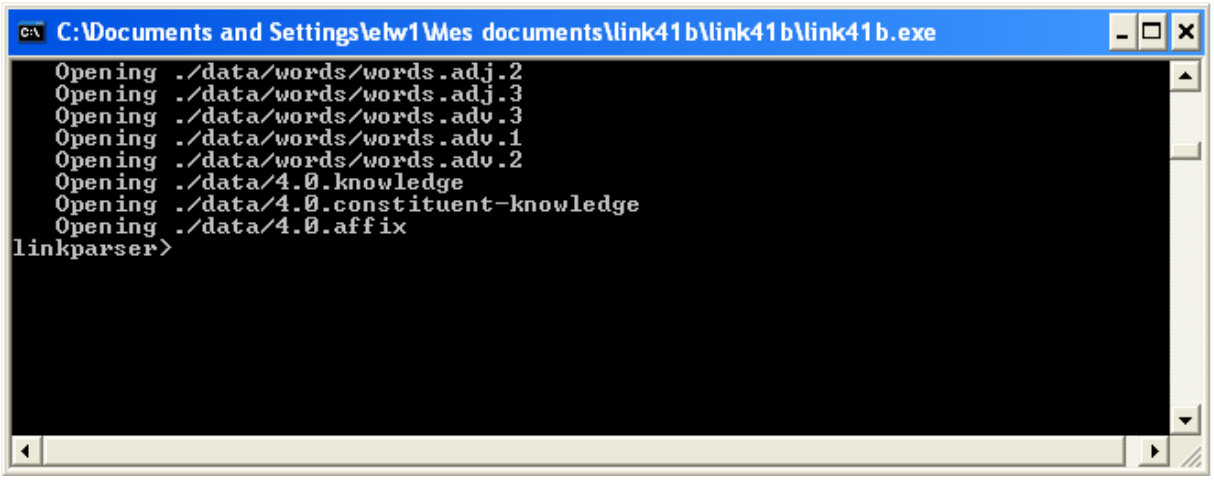

Figure 5.4 : Capture d'écran de l'interface de link parser

Il suffit d'écrire le texte que nous voulons analyser et nous aurons l'analyse syntaxique.

2.4) PyWordNet

Dans notre module nous avons choisi d'utiliser une version de WordNet qui correspond au choix de notre langage de programmation.
En effet, PyWordNet est une interface Python pour la base de données WordNet qui permet avec des fonctions du langage python de consulter la base de données WordNet.
Exemple :
Si nous tapons l'expression suivante dans l'invité de commande python :

```
>>> N['dog'] dog(n.)
>>> N['dog'].getSenses()
```

Nous interrogeons la base de données sur les différents sens du mot « dog ».

```
{'dog' in {noun: dog, domestic dog, Canis familiaris}
```

2.4.1) L'installation

Pour installer **Pywordnet**, il nous a fallu d'abord Télécharger et installer WordNet de 2.0 qui est disponible sur le site http://www.cogsci.princeton.edu/ wn ~ /. Aussi nous avons téléchargé PyWordNet de http://sourceforge.net/projects/pywordnet et décompresser dans le répertoire. Ensuite avec l'invité de commande nous accédons au répertoire contenant les fichiers décompressés et nous tapons `python setup.py`.

Cette commande va permettre concrètement d'installer les deux bibliothèques nécessaires au bon fonctionnement du système. Les deux bibliothèques sont respectivement wordnet.py contient la base de données et wntools.py contient les fonctions qui permettent de consulter la base de données.

2.4.2) L'utilisation de PyWordNet dans notre système

Pour savoir si les mots sont antonyme ou synonymie, qui est l'objet du module inférence entre sujet et événement, nous avons utilisé la fonction meet (mot1, mot2, Synonymie) de la bibliothèque PyWordNet qui permet de donner vrai s'il y a une synonymie entre les deux.

La même chose pour l'antonyme mot et meet (mot1, mot2, antonymie) qui permet de donner vrai si 'il y a une antonymie entre les deux mots.

Cette figure représente la fonction qui détecte s'il y a une antonymie entre deux mots programmés en Python :

sujet5.py - C:\Python25\sujet5.py

File Edit Format Run Options Windows Help

```
def sujet():
    global EV1
    global EV2
    EV11="drink"
    EV22="kiss"
    x1=N[EV11].getSenses()
    y1=N[EV22].getSenses()

    x=N[EV1].getSenses() # avoire les synonyme de ev1

    z1=len(x)
    Y=N[EV2].getSenses()# avoire les synonyme de ev2

    z2=len(EV2)
    B=meet(x1[1],y1[1])
    print B
    i=0
    j=0
    p=0
    while(j<z2):
        if(p==1):
            break
        else:
            while(i<z1):
                D=meet(Y[j],x[i],ANTONYM)# relation entonymie entre Y,X
                if(D!=B):
                    print"sa march "
                    print D
                    p=1
                    break
                else:
                    i=i+1
    j=j+1
```

Ln: 1 Col: 0

Figure 5.5 : La fonction d'interfaçage avec WordNet

Dans ce qui suit nous allons illustrer nos travaux avec le déroulement d'un exemple du corpus sur notre système. Nous allons citer les différentes phases de traitement de la paire de textes.

3) Exemple d'exécution du TIMINF sur un exemple du corpus

Nous avons choisi pour l'exemple, la paire numéro 8 du corpus de développement.

Comme il est montré dans l'exemple ci-dessous la première étape est de transformer le texte brut en format simple-xml, pour cela nous avons balisé manuellement les paires du corpus.

Entré du système TIMINF

```
Exemple d'entrée simple-xml.
<DOC>
<DOCID> Simple Test </DOCID>
<TEXT>
the building collapsed at 2 o'clock p.m.
In the afternoon, the building collapsed.
</TEXT>
</DOC>
```

Figure 4.6 : Entré simple-xml

3.1) TARSQI

Le module TARSQI va permettre de détecter les deux événements de la paire de deux textes (T,H) qui correspondent dans l'exemple au verbe **collapsed** qui est l'évènement des deux segments de textes.

La commande qui permet d'enclencher le module TARSQI avec le format simpel-xml dans un environnement UNIX c'est :

```
python tarsqi.py simple-xml (le nom du fichier contenant les deux segments de textes)  (le nom du fichier de sortie).
```

Ci-dessous nous montrons la sortie TARSQI correspondant à l'exemple :

Sortie TARSQI

```
<TEXT><s>segment T</s>
 <s><lex pos="IN">in</lex> <NG><lex pos="DT">the</lex><TIMEX3 tid="t1" TYPE="TIME"
     VAL="TAFTERNOON">
<lex pos="NN">afternoon</lex></NG>
 <NG><lex pos="DT">the</lex> <lex pos="NN">building</lex></NG> <VG><lex pos="VBN">
 <EVENT eid="e2" class="OCCURRENCE">collapsed</EVENT></lex>
<MAKEINSTANCE eventID="e2" polarity="POS" pos="VERB" eiid="ei2"
     tense="PASTPART"aspect="NONE"></MAKEINSTANCE></VG>
 <lex pos=".">.</lex> </s>
```

Texte balisé par TARSQI

```
 <TLINK relatedToTime="t1" lid="l4" relType="BEFORE" eventInstanceID="ei2"
     origin="CLASSIFIER 0.998739"></TLINK>
 <TLINK lid="l5" relatedToEventInstance="ei1" timeID="t1" relType="INCLUDES"
     origin="Blinker - Type 2 (at) i"></TLINK>
 <TLINK lid="l5" relatedToEventInstance="ei2" timeID="t2" relType="INCLUDES"
     origin="Blinker - Type 2 (at) i"></TLINK></TEXT>
```

Relation déduite par TARSQI

Figure 5.7 : Sortie TARSQI

3.2) L'analyse syntaxique

L'analyse syntaxique se fait en parallèle avec TARSQI et elle va permettre de détecter les sujets des deux segments de textes, qui correspond dans l'exemple à **the building.**

Ci-dessous nous montrons la sortie de LINK Parseur correspondante à l'exemple :

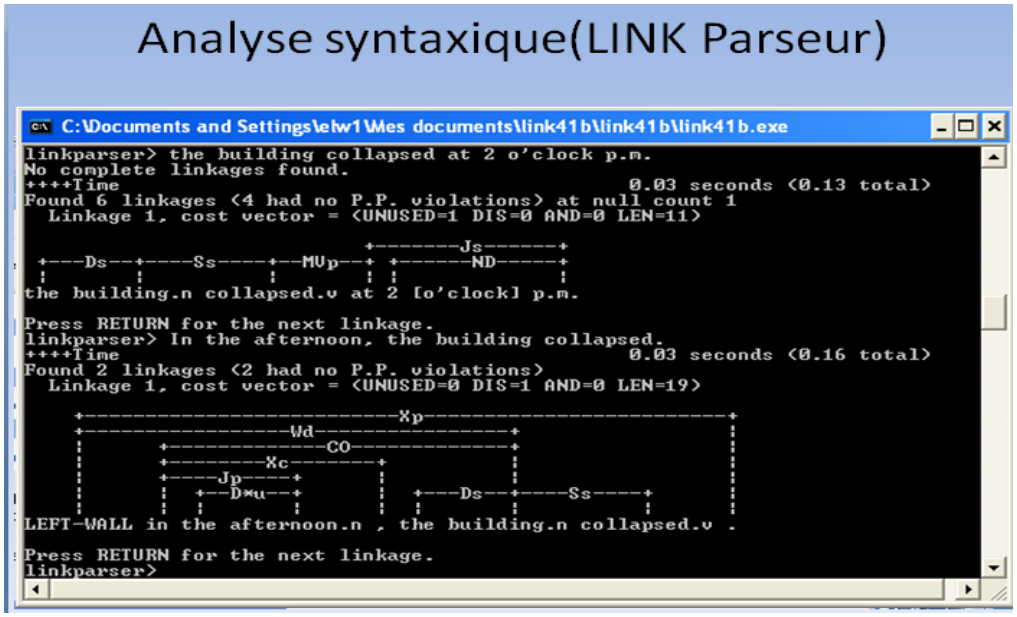

Figure 5.8: Sortie de l'analyseur syntaxique

3.3) L'inférence entre sujets et événements

Comme il est montré dans l'exemple ci-dessous le module de test d'inférence entre sujets et évènements va permettre de détecter l'équivalence entre les deux sujets et les deux évènements des segments T et H.

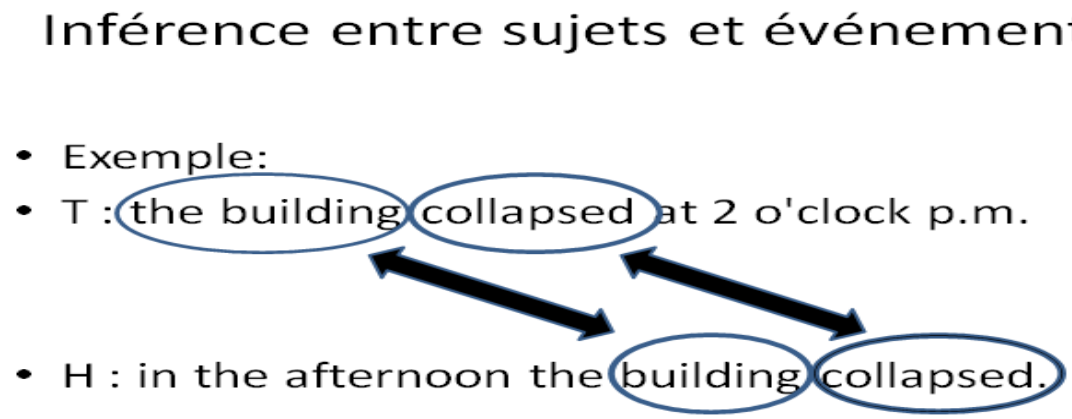

Figure 5.9 : Inférence entre sujets et évènements

3.4) Le balisages des expressions temporelles non détectées par TARSQI

Comme il est montré dans l'exemple ci-dessous, le balisage des expressions temporelles va permettre de positionner l'expression temporelle dans le temps. Dans l'exemple il détermine que **the afternoon** c'est l'intervalle temporel entre midi et 18 heures.

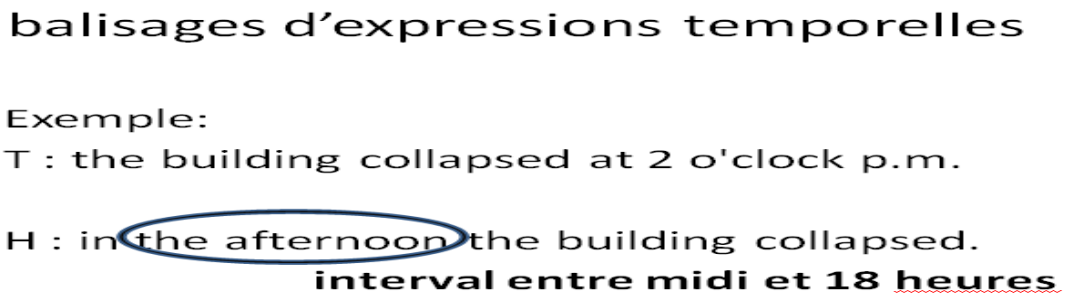

Figure 5.10: Balisages des expressions temporelles

3.5) Le superviseur

L'équivalence entre les sujets et les évènements est détectée par la phase d'inférence textuelle et d'ancrage entre les expressions temporelles (2 o'clock et the afternoon) est détecté par l'application de la régle R5 de la base de règles d'inférences qui stipule que si les différentes conditions se réunissent c'est-à-dire :

- détecter une équivalence entre les deux évènements (**e1**, **e2**)

- l'événement **e1** est relié avec une relation TLINK **e1→t2** et l'événement **e1** est relié avec la même relation TLINK à une durée **e2→d.**

- inclusion entre la date **t1** et la durée **d.**

Nous aurons une inférence temporelle et textuelle entre les segments **T** et **H.**

Des deux résultats précédents le superviseur décide qu'il y a une inférence textuelle entre les segments T et H.

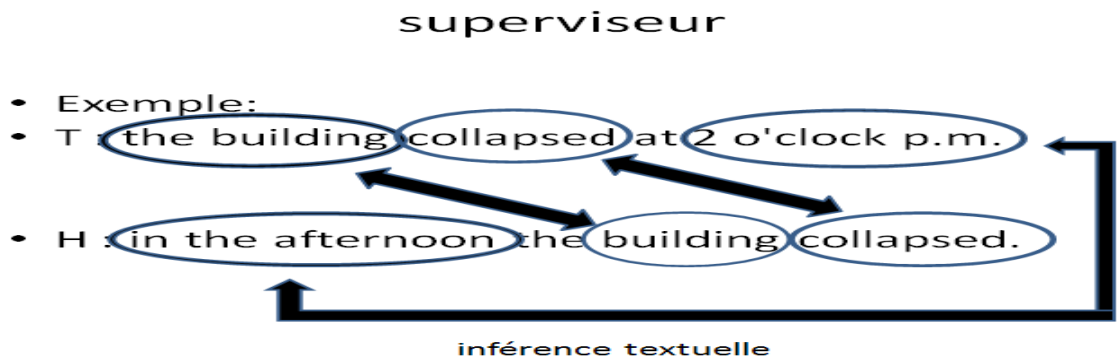

Figure 5.11 : Test d'inférences

Cette figure représente les différentes conditions nécessaires à une inférence textuelle.

4) L'évaluation de notre système

Notre objectif consiste à améliorer les systèmes d'inférences textuelles. Dans ce cadre, nous avons choisi d'évaluer notre système d'inférence avec le système d'évaluation adopté par le challenge RTE. Pour cela, nous devons évaluer le système par rapport au corpus de développement et aussi par rapport au corpus de test.

Chaque paire du corpus est lancée dans notre système qui donne en sortie s'il y a une inférence textuelle ou pas. Les résultats sont comparés au « **GOLD standard** » que nous avons établi dans notre étape de conception du corpus. Le pourcentage donnant le nombre de fois où il y a similitude entre notre système et le « **gold standard** » donne « **l'accuracy** » du système. l'accuracy est une mesure standard fréquemment utilisée dans les systèmes de traitements du langage naturel.

Dans ce qui suit, nous allons présenter les résultats préliminaires des évaluations des deux corpus.

4.1) L'évaluation du système sur le corpus de développement

Nous avons élaboré notre système d'après l'étude des inférences existantes dans le corpus de développement. Ce corpus nous a permis de tester notre système plusieurs fois en effectuant à chaque fois des modifications jusqu'à ce qu'on arrive à concevoir un système qui a donné 100% d'accuracy par rapport à ce corpus.

4.2) L'évaluation du système avec le corpus de test

Le corpus de test est constitué de 30 paires de textes, 15 d'entres elles sont évaluées comme contenant une inférence textuelle fausse et les autres sont évalués comme vrai.

Nous avons soumis ce corpus a notre système qui nous a permis de calculer l'accuracy.

Les résultats d'accuracy sont montrés dans le tableau suivant :

Les systémes	L'accuracy
Système	58 %

Tableau 2.1 : Le tableau représente l'accuracy du système

Les résultats de l'évaluation sont encouragent puisque nos résultats sont plus élevés que la moyenne de l'accuracy des systèmes participants au RTE 2 qui sont de 56.6 %.

Dans ce qui suit nous allons étudier les causes de défaillance de notre système.

4.4) L'analyse des erreurs causées par le système

D'après notre étude des résultats donnés par notre système nous avons pu élaborer un tableau contenant des statistiques concernant les causes d'échecs de notre système.

Problème	Pourcentage d'erreur dans le corpus
Analyse syntaxique	38 %
TARSQI	62 %

Tableau 3.2 : les causes d'erreurs du système

Nous remarquons dans ce tableau que les majeures parties des erreurs commises par notre système sont en générale causé par la déficience de l'outil TARSQI.

En effet, TARSQI ne détecte pas plusieurs choses. Par exemple, au niveau de la détection des évènements où nous avons remarqué que TARSQI ne détecte pas les verbes composés comme un événement mais plutôt comme deux évènements indépendants.

Exemple: paire numéro 1 du corpus de test.
T: the First World War spent 7 years.

H: World War I, also known as the First World War, the Great War and the War To End All Wars, was a global military conflict which **took place** primarily in Europe from 1914 to 1918.

Dans l'exemple la détection de l'événement **took place** par TARSQI n'a pas pu se faire car **took place** est un verbe composé.

Aussi les erreurs de notre système viennent de l'analyse syntaxique effectuer en pré traitement par *link parser* où les sujets des verbes ne sont pas détectés.

Exemple: paire numéro 10 du corpus de test.

T: Protracted military S1: conflict between Iran and Iraq. **It** officially began on t1: Sept. 22, 1980, finally, in July, 1988, Iran was forced to accept a United Nations–mandated cease-fire.
H: With more than 100000 Iranian victims of Iraq**'s** chemical weapons during the ten-year war, Iran is one of the countries most severely afflicted by weapons.

Dans l'exemple précédant, la relation entre la date t1 et le sujet S1 n'est pas détecté par TARSQI puisque l'analyse syntaxique n'a pas pu auparavant relier entre **conflict** et **it.**

5) Conclusion

Nous avons présenté dans ce chapitre le processus d'installation de nos différents outils nécessaires au bon fonctionnement de notre système. Nous avons également présenté le déroulement de notre système sur un exemple du corpus qui a permis de montrer comment les différents modules étaient mis en œuvre dans notre système.

Enfin, nous avons donné les performances de notre système qui étaient encourageantes et nous avons étudié les différentes failles de notre système. Cela a permis de monter que l'inférence temporelle à un besoin inéluctable aux d'autres modules d'inférences.

Conclusion générale et perspective

Nous avons présenté, tout au long de ce manuscrit, notre démarche pour la conception d'un système d'inférence textuelle considérant l'inférence temporelle dans sa décision. Pour cela nous avons d'abord exploré l'apport du RTE dans les différentes applications du TAL (RI, QR, EI et RA) et étudié les différentes approches utilisées pour détecter l'inférence (lexical, lexico syntaxique, sémantique et logique). Puis nous avons analysé les approches des différents groupes de recherches qui ont participé aux trois challenges Pascal RTE. Cette étape nous a permis de découvrir les chemins qui n'ont pas encore été étudiés pour détecter l'inférence textuelle.

Ensuite, nous avons exploré la logique temporelle, ses applications dans le traitement du langage nature et les différents types d'inférences temporelles existantes. Cette étude nous a permis de constater qu'il n'y a pas de travail à nos jours liant l'inférence temporelle et la reconnaissance de l'inférence temporelle.

Nous avons élaboré un corpus contenant des paires de segments de textes integrant des relations temporelles et nous avons fait une classification des différents types d'inférences temporelles existants dans le corpus.

La suite logique de ce travail est de déduire des régles d'inférences temporelles et les intégrer à un systéme de reconnaissance d'inférence texuelle.

Une fois le systéme concu, nous avons évalué ses performances avec la méme stategie d'evaluation adoptée dans le challenge pascal RTE. Cette evaluation nous a donné des résultats encourageants.

Enfin, nous avons étudié les différentes failles de notre système. Cela a permis de prévoir plusieurs perspectives de recherches.

Contribution

Etant donné les objectifs que nous nous sommes fixés pour ce projet, les principales contributions de TIMINF peuvent être résumées comme suit :

L'élaboration d'un corpus à base d'inférence temporelle permettra d'évaluer les recherches futures dans ce Domaine.

L'étude du corpus nous a permis de classifier différents types d'inférence temporelle et de développer différentes règles d'inférences temporelles.

Aussi l'évaluation de notre système a permis de voir concrètement quel est l'apport de l'aspect temporel dans le RTE.

Perspectives et travaux futurs

Nous envisageons de poursuivre nos recherches futures dans trois directions principales.

Notre système ne permet pas de détecter les entités nommées et de gérer les anaphores. Pour cela, nous envisageons d'introduire un module permettant de détecter et de dater les entités nommées automatiquement. Aussi nous pensons à intégrer un module pour gérer les anaphores et étudier l'impacte de celui-ci sur la performance de notre système.

La seconde direction scientifique est d'évaluer le système prédicat argument comme prétraitement au lieu d'une simple analyse syntaxique.

Enfin, nous envisageons également de développer un système pouvant tester l'inférence textuelle dans des segments de textes plus grandes et utiliser un système qui utilise comme réponse trois sorties possibles (inférence vrai, inférence Fausse ou on ne c'est pas s'il y a une inférence) et nous associons chaque inférence vraie à une application du TALN (QR, RI, IE, PP…).

Références

(Baker, Fillmore et Lowe, 1998) Baker, Charles J. Fillmore, and John B. Lowe. 1998. The berkeley framenet project. In Proceedings of the COLING-ACL, Montreal.

(Bras, 1990) Myriam Bras. Calcul des Structures Temporelles du Discours. PhD thesis, IRIT, 1990.
(Benveniste, 1974) Benveniste Emile Problèmes de linguistique générale. Paris, Gallimard, vol. II.

(Bourigault, 2000) BOURIGAULT D. Recent Advances in Computational Terminology, 2000.

(Bourigault et al., 2004) BOURIGAULT D. AUSSENAC-GILLES N. et CHARLET J. (2004). Construction de ressources terminologiques ou ontologiques à partir de textes : un cadre unificateur pour trois études de cas, Revue d'Intelligence Artificielle, 18(4), 24 pp.

(Charolles, 1997) Charolles M. « L'encadrement du discours – univers, champs, domaines et espaces », Cahier derecherche linguistique, 6, p. 1-73. 1997.

(Chaumartin, 2007) Francois-Regis chaumartin, wordnet et son ecosysteme, BDL-CA,2007, montreal.

(Cohen, 1960) Cohen J. : "A coefficient of agreement for nominal scales", Educ. Psychol. Meas.: 20, 27-46. 1960

(Dagan et al, 2005) Textual inference problems from the PASCAL RTE. Challenge, 2005.

(Len Schubert, 2002) Len Schubert. Can we derive general Word Knowledge from Texts ?. 2002.

(Helmut Schmid, 1994) Part-of-Speech Tagging with Neural Networks. Proceedings of the 15th International Conference on Computational Linguistics (COLING-94). August 1994.

(Joachims, 2003) T. Joachims, Information Retrieval and Language Technology (pdf), 2003, Cornell University.

(Kosseim., 2005). Leila Kosseim, Extraction d'information bilingue, 2005.

(Ligauzat , 1994) Gérard Ligauzat. Représentation des connaissances et linguistique. Armand Colin, Paris, 1994.

(Lin et Pantel, 2001) DeKang Lin and Patrick Pantel. 2001. Discovery of inference rules for Question Answering. Natural Language Engineering.

(Laurain et Marie, 2006) La traduction automatique. France. Septentrion Presses Universitaire, 1996. p. 15-16.

(Macleod et al., 1998) C.Macleod, R.Grishman, A.Meyers, L.Barrett and R. Reeves. 1998. Nomex : A lexicom of normalisations.in Proceedings of 8 the International Congress of the European association for lexicography.1998. liege, begium : EURALEX.

(Mani et Wilson, 2000) Mani and George Wilson. 2000. Processingof News. In Proceedings of the 38th Annual Meetingof the Association for Computational Linguistics(ACL2000), pages 69–76.

(Moldovan et Rus, 2001) Dan I. Moldovan and Vasile Rus. 2001. Logic form transformation of wordnet and its applicability to question answering. In Meeting of the Association for Computational Linguistics, pages 394-401.

(Moldovan and Rus, 2001). Moldovan and Rus Logic Forms can be utilized by a wide variety.2001.

(Miller, 1995) P. Miller. 1995. "Notes on phonology and orthography in several Katuic Mon-Khmer groups in Northeast Thailand." Mon-Khmer Studies 24: 27-51.

(Nugues, 2006) Pierre Nugues. An Introduction to Language Processing with Perl and Prolog. Springer Verlag, 2006.

(Nyberg et al, 2002) E. Nyberg, T.Mitamura, J. Carbonnell, J. Callan, K. Cllins-Thompson, K Czuba, M. Duggan, L. Hiyakumoto, N. Hu, Y. huang, J. Ko, L.V. Lita, S.Muratagh et V. Pedro. The JAVELIN Question-Ansewering System at TREC 2002. In Proceding of the 11th Text Retrieval conference (TREC-11), 2002.

(Pustejovsky et al., 2003) Paul Kiparsky and Carol Kiparsky. InManfred Bierwisch and Karl Erich Heidolph, editors, Progress in Linguistics. A collection of Papers, pages143–173. Mouton, Paris.

(Rodrigo et al., 2007) A. Rodrigo, A. Pe˜nas, J. Herrera and F. Verdejo..*The Effect of Entity Recognition on Answer Validation*.In Lecture Notes in Computer Science. In press 2007.

(Sleator et Temperley, 1991) Daniel Sleator and Davy Temperley. 1991. Parsing English with a Link Grammar. Carnegie Mellon University Computer Science technical report CMU-CS-91-196, October 1991.

(Swinnen, 2005) Gérard Swinnen Apprendre à programmer avec Python, Copyright 2005.

(Tatu et Moldovan,2007) Marta Tatu and Dan Moldovan. 2007 COGEX at the third recognising of textual entailement challenge. In proceeding of the wokshop on textual entailment, prague, June 2007.

(Tatu et al., 2006) Marta Tatu, B Iles, J. Slavick, A. Novischi, and D. Moldovan. 2006, COGEX at the third recognising of textual entailement challenge. In proceeding of the wokshop on textual entailment,Venice, Italy.

(vanderwende et al., 2005) Lucy vanderwende, deborah coughlin and bill dolan.
2005.what syntax contribute in entailment task. In proccedings of pascalchallange workshop on recogning texual entailment,2005 .

(Venhagen et al., 2005) M. Venhagen, I. Mani , R. Sauri, R. Knippen, J .Littman and J. Pustejovsky. 2005. Automating Tenporal Annotation With TARSQI. In Proceedings of ACL 2005. demo session.

(WOS, 1998) L. WOS. Automated Reasoning -33 Basic Research Problems. Prentice-Hall.
(Yvon, 2007) François Yvon . Une petite introduction au Traitement Automatique des Langues Naturelles, 2007.