Un Sistema Genèric de Cerca de la Resposta

Edgar Gonzàlez i Pellicer

If my doctor told me I had only six minutes to live,
I wouldn't brood.
I'd type a little faster....

Isaac Asimov

Sobre Aquesta Memòria

Sobre el Glossari de la Memòria

Al llarg del text d'aquesta memòria, ocasionalment apareixeran mots amb un asterisc* al seu costat. Aquests mots es troben definits al Glossari a l'Apèndix A. En la majoria es tracta de termes d'ús freqüent en el domini del Processament del Llenguatge Natural, però que poden resultar estranys a algú que no hi estigui avesat. He optat per aquesta distribució per a permetre a tothom seguir la meva memòria però sense molestar aquells que ja estan acostumats al vocabulari emprat.

Sobre la Guia d'Ús

El document que apareix a l'Apèndix C reprodueix una petita Guia que vaig redactar a mig termini del desenvolupament del projecte per a explicar a altres desenvolupadors que poguessin utilitzar les classes de què es parla a l'Apèndix B. Encara que el document no es troba en una versió definitiva, he considerat afegir-la aquí com a descripció addicional de la feina duta a terme, així com per a l'objectiu perquè va ser escrita: com a referència per als usuaris de la llibreria.

Sobre la Redacció de la Memòria

- La Redacció d'aquesta memòria ha tingut lloc usant el sistema de redacció **LaTeX**, en concret la implementació **teTeX**[1].
- Com a editor de textos s'ha utilitzat **GNU Emacs**[2], el mateix sistema usat durant tot el procés de desenvolupament del projecte.
- Per a fer els gràfics, ha estat de menester **Xfig**[3], amb què generàvem fitxers en PostScript Encapsulat per a importar a **LaTeX**.
- Els diagrames UML s'han desenvolupat amb **ArgoUML**[4], i també exportat a PostScript Encapsulat.
- Tot aquest procés ha tingut lloc funcionant sobre màquines equipades amb **Linux**[5], o, en el seu defecte, **Cygwin**[6] funcionant sobre **Windows XP**[7].

[1] http://www.tug.org/tetex/
[2] http://www.gnu.org/software/emacs/emacs.html
[3] http://www.xfig.org/
[4] http://argouml.tigris.org
[5] Incomptables...
[6] http://www.cygwin.com/
[7] http://www.microsoft.com/

Índex

1 Introducció **11**
- 1.1 Context Històric . . . 11
  - 1.1.1 La Mineria de Textos . . . 11
  - 1.1.2 Naixement de la CR . . . 12
  - 1.1.3 Les Conferències TREC, MUC i CLEF . . . 13
- 1.2 Alguns Sistemes de CR Online . . . 16
- 1.3 Tipologia de la CR . . . 16
  - 1.3.1 Classificació dels Sistemes de CR . . . 18
- 1.4 Arquitectura dels Sistemes de CR . . . 19
  - 1.4.1 Anàlisi de la Pregunta . . . 19
  - 1.4.2 Recuperació de Passatges . . . 20
  - 1.4.3 Extracció de la Resposta . . . 22
- 1.5 La Metodologia TALP - UdG . . . 22
  - 1.5.1 Anàlisi de la Pregunta . . . 22
  - 1.5.2 Recuperació de Passatges . . . 24
  - 1.5.3 Extracció de la Resposta . . . 25
  - 1.5.4 Resultats Obtinguts . . . 27
- 1.6 El Projecte CHIL . . . 27
- 1.7 Integració . . . 28

2 Anàlisi de Requeriments **31**
- 2.1 Característiques de la Implementació Existent . . . 31
- 2.2 Objectius a Assolir . . . 32
- 2.3 Altres Requeriments . . . 32

3 Especificació i Disseny **35**
- 3.1 El·lecció de l'Arquitectura del Sistema . . . 35
  - 3.1.1 Arquitectures Possibles . . . 35
  - 3.1.2 Arquitectura Escollida . . . 35
  - 3.1.3 Avantatges de l'Arquitectura Proposada . . . 36
  - 3.1.4 Sistemes amb Arquitectures Similars . . . 37
- 3.2 Representació d'Informació de Tipus Lingüístic . . . 38
  - 3.2.1 Format dels Fitxers . . . 38
- 3.3 El MetaServidor . . . 40
  - 3.3.1 El Protocol de Comunicació . . . 42
  - 3.3.2 El Sistema de Regles . . . 46
  - 3.3.3 Disseny . . . 49
- 3.4 Els Servidors . . . 49
  - 3.4.1 Format Esperat de les Peticions . . . 49
  - 3.4.2 Paràmetres d'Iniciació i de Procés . . . 50
  - 3.4.3 Esquema de Funcionament dels Servidors . . . 51
- 3.5 Els Clients . . . 51

4 Implementació **57**
4.1 Consideracions Generals . . . 57
4.1.1 Llenguatge de Programació . . . 57
4.1.2 Procés de Desenvolupament . . . 58
4.1.3 Prova i Detecció d'Errors . . . 59
4.2 Implementació del MetaServidor . . . 59
4.3 Implementació de la Part Comuna dels Servidors . . . 59
4.4 Implementació de la Part Comuna dels Clients . . . 60
4.5 Implementació del Sistema TALP-UdG . . . 61
4.5.1 TnT . . . 63
4.5.2 Lemmatizer WordNet2 . . . 65
4.5.3 Abionet . . . 65
4.5.4 Alembic . . . 66
4.5.5 Anotador Semàntic basat en WordNet . . . 67
4.5.6 EngSegmenter . . . 67
4.5.7 FreeLing . . . 67
4.5.8 PreProces . . . 67
4.5.9 MkSents . . . 69
4.5.10 Filter . . . 69
4.5.11 SubCat . . . 70
4.5.12 Minipar . . . 70
4.5.13 MkSints . . . 70
4.5.14 Cst . . . 72
4.5.15 MkChunks . . . 72
4.5.16 MkEnv . . . 72
4.5.17 PreProcesProlog . . . 72
4.5.18 QClassify . . . 74
4.5.19 PreProcesPregunta . . . 74
4.5.20 MgRetr . . . 75
4.5.21 Csc . . . 75
4.5.22 Xtract . . . 75
4.5.23 Millor . . . 75
4.5.24 QA . . . 75
4.6 Conclusions . . . 75

5 Direccions Futures **81**
5.1 Treball a Curt Termini . . . 81
5.2 Treball a Mig Termini . . . 81
5.3 Treball a Llarg Termini . . . 82

6 Distribució del Temps Emprat **85**
6.1 Planificació Inicial del Projecte . . . 85
6.2 Distribució Final del Temps . . . 86

A Glossari **87**

B Representació d'Informació de Tipus Lingüístic **93**
B.1 Requeriments . . . 93
B.2 Disseny . . . 94
B.3 Format de Fitxer . . . 94

C Guia d'Ús de la Llibreria Libfuf **95**
C.1 Objectiu del Document 95
C.2 Descripció de les Classes 95
C.2.1 Models 95
C.2.2 Identificadors 96
C.2.3 Iteradors 98
C.2.4 Posicions 98
C.2.5 Elements 98
C.2.6 Atributs 98
C.2.7 Entrada/Sortida 98
C.2.8 Excepcions 102
C.3 Llibreria C++ 102
C.3.1 Convenis 102
C.4 CookBook 103
C.4.1 Avatar 103
C.4.2 Posicions 106
C.4.3 Elements 107
C.4.4 Atributs 109
C.4.5 Entrada/Sortida 110
C.5 El Format FUF 110

Capítol 1

Introducció

El Projecte que es presenta en aquesta memòria es desenvolupa al voltant de les tecnologies conegudes com a Cerca de Resposta (*Question Answering*, CR), concretament, al voltant de la metodologia desenvolupada pel centre de recerca TALP[1], de la UPC[2], i la UdG[3]. Tanmateix, abans d'entrar en detall en el projecte i el seu desenvolupament, pot ser interessant d'explicar què és la Cerca de Resposta i de veure quin procés històric ha seguit fins arribar al seu estat actual.

1.1 Context Històric

1.1.1 La Mineria de Textos

El punt de partida de les tecnologies que es coneixen amb el nom genèric de Mineria de Textos*, dins les quals es troba la Cerca de Resposta, i que tenen com a objectiu comú tractar grans col·leccions documentals de forma útil (encara que els conceptes de *què* és útil puguin ser dispars), és el creixement exponencial experimentat en els darrers anys en la quantitat d'informació disponible en forma digital.

Ha esdevingut doncs necessari desenvolupar tècniques que permetin un accés intel·ligent a la informació per part de qualsevol tipus d'usuari, fins i tot aquells amb poca experiència amb la Informàtica. Tenint en compte que molta d'aquesta informació es troba en forma textual i no estructurada (en contrast amb la informació d'una Base de Dades, que es troba estructurada) i que per a facilitar l'accés a aquests sistemes les peticions dels usuaris han de poder ser fetes en Llenguatge Natural*, sembla lògic que la comunitat científica dedicada al Processament del Llenguatge Natural*(PLN) mostrés interès en aquesta problemàtica i comencés a desenvolupar tècniques per a resoldre-la.

L'objectiu, exposat per exemple en la Tesi Doctoral de José Luís Vicedo ([Vi02], juntament amb una explicació molt més detallada de la història i les tècniques aquí exposades), era i segueix essent aconseguir un sistema ideal, capaç de localitzar la informació, processar-la, integrar-la i retornar-la a l'usuari seguint els requeriments expressats per aquest. Tanmateix, en paraules del propi Vicedo, *...aunque las investigaciones avanzan en buena dirección, todavía no existe hoy ningún sistema operacional que cumpla todos estos requisitos.*

Inicialment, la investigació es va centrar en dos camps:

Recuperació d'Informació (*Information Retrieval*, RI) Sistemes que, a partir d'una consulta formulada per l'usuari, sel·leccionen els documents* d'una certa col·lecció que, seguint uns certs criteris de valoració, poden respondre millor a les necessitats expressades en la consulta.

[1] Tecnologies Aplicades al Llenguatge i la Parla
[2] Universitat Politècnica de Catalunya
[3] Universitat de Girona

L'exemple clàssic d'aquests sistemes són els cercadors d'Internet (**Google**[4], **Yahoo**[5], **AltaVista**[6]...) que a partir d'una sèrie de paraules clau*, operadors booleans i criteris addicionals (idioma, domini, país de publicació...) retornen una llista de documents, ordenats segons la rellevància que el sistema considera que poden tenir. Aquestes mesures de rellevància poden ser funció de paràmetres com ara el nombre d'aparicions de les paraules clau o el d'enllaços que apunten cap al *site* on es troba el document.

La principal restricció d'aquests sistemes és la velocitat: cal atendre moltes peticions en poc temps. I el principal inconvenient: l'usuari és qui ha de recórrer un a un els documents retornats pel sistema, potencialment llargs, per a trobar la informació que realment li interessa. Per a mitigar una mica aquest problema, han aparegut tècniques de Recuperació de Passatges (*Passage Retrieval*), que a canvi d'un major cost computacional processen els documents a nivell de passatge*, i permeten identificar els fragments de text interessants i recuperar documents llargs on la informació es troba només en una petita part.

Extracció d'Informació (*Information Extraction*, EI) Sistemes que processen col·leccions de documents relatius a un domini restringit per a extreure'n la informació rellevant de forma estructurada. Són sistemes que es dissenyen de forma específica per a un cert domini. Reben una plantilla definida que cal emplenar amb la informació que s'extreu del text.

Un exemple seria un sistema que, a partir dels articles d'una Enciclopèdia biogràfica, obtingués el nom, la data i lloc de naixement i les principals activitats i obres dels artistes que apareguin. A partir de les plantilles emplenades es podria generar una base de dades posteriorment consultable de la manera habitual.

El principal inconvenient de l'EI és la seva poca portabilitat entre dominis. Un sistema d'EI dissenyat per a treballar en un cert domini sovint és difícil d'adaptar per a fer-ho en un altre. Per altra banda, no es pot tractar amb textos en domini obert ja que les plantilles es dissenyen per a una aplicació concreta.

1.1.2 Naixement de la CR

Encara que els sistemes de RI i d'EI han resultat realment útils per a les tasques per a què han estat concebuts, no tenen el mateix objectiu que la CR: la localització de fragments de text concrets que responguin a preguntes concretes formulades per l'usuari en llenguatge natural.

Els primers intents d'abordar el problema apareixen a finals dels 70, en el sistema **QUALM** desenvolupat per Wendy Lehnert ([Le80]). En un principi es van intentar utilitzar mètodes propis de la Intel·ligència Artificial*(IA) per a resoldre el problema, però des d'aquesta perspectiva només es van obtenir bon resultats restringint (i molt) el domini. Sovint es tractava de sistemes que eren una interfície en llenguatge natural sobre una base de dades. Només a partir dels 90 s'ha començat a treballar amb mètodes d'IA sobre textos no estructurats, desenvolupant sistemes com **The UNIX Consultant**([CL94]) o **Lilog**([HR91]).

Va ser quan els experts en RI i EI van decidir aventurar-se en aquest camp, introduint les seves tècniques, que es van començar a obtenir bons resultats en dominis no restringits. El primer sistema d'aquestes característiques va ser **MURAX**, desenvolupat a meitat dels 90 per Kupiek (i descrit a [Ku93]). **MURAX** acceptava preguntes en llenguatge natural sobre l'Enciclopèdia Grolier, retornant a l'usuari el sintagma nominal que considerava la resposta.

A partir d'aquest moment es va iniciar una intensa activitat investigadora en aquests sistemes, en gran part gràcies a l'impuls donat per les conferències TREC (veure secció 1.1.3).

A Internet existeixen sistemes d'aquest tipus oberts al públic, com ara els que apareixen a la secció 1.2.

Paral·lelament, van sorgir sistemes amb característiques lleugerament diferents dels de CR pròpiament dits. Entre ells podem incloure els de:

[4] `http://www.google.com/`
[5] `http://www.yahoo.com/`
[6] `http://www.altavista.com/`

Cerca de Preguntes Freqüents (*Frequently Asked Questions Finding*) Sistemes que treballen sobre les llistes formades per preguntes i respostes (un exemple típic són les llistes de FAQ que es poden trobar a Internet, a llocs com `http://www.faqs.org`). Per a respondre a les preguntes de l'usuari localitzen les preguntes de la llista més semblants a la formulada i en retornen les respostes.

El **FAQFinder**[7] o l'**AskJeeves**[8] són sistemes d'aquest tipus que es poden consultar a Internet.

Comprensió de Textos (*Reading Comprehension Tests*) Sistemes que treballen sobre preguntes referents a un text donat.

Les proves formades per un cert text i una sèrie de preguntes relatives a ells són força usades per a avaluar el grau de comprensió que assoleix un lector en un determinat idioma. Per als sistemes computacionals de CR, aquest tipus de tests proporcionen una manera alternativa d'avaluació, diferent de la usada habitualment (veure secció 1.1.3). Per altra banda, el fet de treballar sobre un únic document fa que les tècniques hagin de ser diferents a les usades sobre grans col·leccions.

1.1.3 Les Conferències TREC, MUC i CLEF

En el desenvolupament de les tècniques de Mineria de Textos és de destacar el paper dut a terme per les diverses conferències que, en els diferents camps, han servit de plataforma per a la posada en contacte de la comunitat investigadora i per al desenvolupament de *benchmarks** i mesures per a la comparació dels diferents sistemes:

- En el camp de la RI, és de destacar el paper de les conferències TREC[9] (*Text Retrieval Conference*), sota els auspicis del NIST (*National Institute of Standards and Technology*) i el Departament de Defensa dels EUA. El seu objectiu és estimular la recerca en el tractament de grans col·leccions de documents en Llenguatge Natural proveint la infrastructura necessària per a poder fer avaluacions dels diferents sistemes.

  La primera conferència TREC va tenir lloc el 1992, i des de llavors són una cita anual on es troba la comunitat científica per a exposar els darrers avenços en el camp i per a provar els sistemes per a les diferents tasques proposades. Les tasques varien cada any: se n'afegeixen i eliminen segons evoluciona la recerca en els diferents problemes. Algunes de les que hi ha o hi ha hagut són:

  Cross-Language Track Sistemes capaços de recuperar documents referents a un cert tema amb independència de l'idioma en què estiguin. Va desaparèixer el 2002.

  Filtering Track Sistemes en què es coneix una demanda d'informació de l'usuari i uns quants documents rellevants, i per a cada document d'un flux cal decidir si mereix ser recuperat o no. Va desaparèixer el 2002.

  Interactive Track Sistemes interactius. Els diversos grups participants acorden un protocol i posteriorment es duen a terme experiments amb usuaris reals. La col·lecció de documents i les preguntes inicials són comunes.

  Es tractava d'un *track* adjunta a la *Web Track* que va fer una aparició puntual l'any 2003.

  Web Track Sistemes de cerca on la col·lecció documental està extreta directament de la *World Wide Web*.

- El 1999, la 8ena conferència TREC va introduir com a novetat una tasca de CR (*Question Answering Track*).

[7] `http://faqfinder.cs.uchicago.edu:8001/`
[8] `http://www.askjeeves.com/`
[9] `http://trec.nist.gov/`

Tal com s'explica a [Vo99], la tasca era obtenir la resposta a 200 preguntes que demanaven per una entitat concreta (de tipus factual, com veurem a la secció 1.3) sobre una col·lecció documental (en concret, la col·lecció *ad hoc* del TREC-8, formada majoritàriament per articles de diaris). S'assegurava que per a cada pregunta hi havia com a mínim un document de la col·lecció on trobar la resposta.

Els sistemes havien de localitzar per a cada pregunta 5 fragments de text de 250 o 50 caràcters que continguessin la resposta (es van fer proves en ambdues categories), ja fossin extrets d'algun document de la col·lecció o bé generats a partir d'informació d'aquesta. Per exemple, si la pregunta era:

– *Qui va conquerir València als musulmans?*

i en la col·lecció teníem un document amb el següent passatge:

> ...Jaume I (1208-1276), dit el Conqueridor, rei d'Aragó, va dur a terme la conquesta de València l'any 1238, incorporant el territori a la Corona d'Aragó i dotant-la de la condició de regne...

Possibles respostes serien:

250 caràcters, extrets *tot el passatge.*

50 caràcters, extrets Jaume I (1208-1276), dit el Conqueridor.

50 caràcters, generats Jaume I el Conqueridor d'Aragó.

Els fragments havien d'estar ordenats per rellevància que el sistema considerava que tenien.

Aquests fragments eren examinats per experts humans, que determinaven si la resposta era correcta o no. La manera d'avaluar els sistemes va ser mitjançant la mesura de Rang Recíproc Mitjà (MRR, *Mean Reciprocal Rank*): per a cada pregunta la puntuació era 0 si cap dels 5 fragments era correcte, o bé l'invers del número d'ordre del primer fragment correcte (és a dir, 1/1 si el 1r fragment era correcte, 1/2 si era el 2n, 1/3 si era el 3r...). La puntuació del sistema era la mitja aritmètica de la seva puntuació per a cada pregunta.

Com a referència, comentar que el millor sistema en la categoria de 250 caràcters per fragment va ser el de la Southern Methodist University, amb un MRR de 0.646 i 44 preguntes amb puntuació 0. En la categoria de 50 caràcters, es va imposar el de la companyia Cymfony Inc, amb MRR de 0.660 i 54 preguntes amb puntuació 0.

En successives conferències TREC la tasca ha anat creixent en complexitat. Així, a l'edició de l'any 2003, la *Question Answering Track* presentava els següents requeriments (molts d'ells existents ja en edicions anteriors, i tots detallats a [Vo03]):

– La col·lecció de documents usada va ser el corpus AQUAINT [10], format per aproximadament un milió de notícies extretes del servei de notícies AP, el servei de notícies del New York Times i l'agència de notícies Xinhua. En total, 3Gb de text.
– Es van incloure preguntes de tipus definicional i enumeracional, a part de les factuals. Tanmateix, s'indicava per a cada pregunta el tipus a què pertanyia, de manera que els sistemes no havien de determinar-ho.
– Es van avaluar dues subtasques: la de passatges, en què calia retornar un extracte d'un document de la col·lecció com a resposta a cadascuna de les 413 preguntes de tipus factual que es proposaven; i la principal, en què calia retornar la resposta exacta (sense necessitat d'ésser un extracte de cap document) a les mateixes 413 preguntes, 37 d'enumeracionals i 50 de definicionals.

[10] *Advanced Question and Aswering for Intelligence*: projecte que estimula la investigació en Cerca de Resposta sobre col·leccions heterogènies de documents, amb informació estructurada, no estructurada, imatges, video... `http://www.ic-arda.org/InfoExploit/aquaint`

  - Es van incloure preguntes sense resposta a la col·lecció.
  - Es van incloure noves mesures per a quantificar els nous tipus de pregunta, així com la precisió* i el *recall** a l'hora d'indicar les preguntes sense resposta.

  Altres idees, com ara la del TREC-10 d'incloure sèries de preguntes sobre un mateix esdeveniment, amb referències al context (per exemple, *Quan va néixer Alexandre Dumas?*, *Quin va ser el primer llibre que va publicar?*) s'han anat descartant bé per considerar-se ja superades, bé perquè la manera d'avaluar-les no reflectia la veritable dificultat del problema (l'explicació dels problemes amb les sèries de preguntes apareix a [Vo01]).

  Per altra banda, ben segur que futures edicions incorporaran nous problemes. I es que un dels objectius de les conferències és traçar un pla de desenvolupament del camp de la CR a llarg termini, esbossant els objectius que es volen assolir i fent plans a mig termini per a orientar investigacions futures.

- La major part de la tasca d'incentivació duta a terme per les conferències TREC ha estat per a sistemes en llengua anglesa. A nivell multilingüe, el CLEF [11] (*Cross-Lingual Evaluation Forum*) intenta estimular el desenvolupament de sistemes mono i multilingües de RI en les diverses llengües europees.

  El CLEF també inclou una tasca de Question Answering, tant monolingüe com multilingüe. Se'n pot trobar informació a la web `http://clef-qa.itc.it/2004`.

- Pel que fa a l'EI, les conferències MUC [12] (*Message Understanding Conference*) van servir de punt de trobada, comparació i avaluació dels diferents sistemes, de forma similar al TREC per a la RI i la CR. A la tesi doctoral de Jordi Turmo ([Tu02]) se'ns fa una exposició de les característiques de les diferents MUC i de l'evolució dels problemes proposats.

  Les MUC eren organitzades pel Grup de Recerca i Desenvolupament Naval (NRaD, *Naval Research and Development Group*) del NCCOSC (*Naval Command, Control and Ocean Surveillance Centre*), entitat depenent de la Marina dels EUA.

  La primera MUC va tenir lloc l'any 1987, i va tenir un caràcter més aviat exploratori: l'organització no va establir cap mena de tasca ni de criteri d'avaluació. L'únic que es va proposar va ser el domini del documents: operacions tàctiques navals.

  Tanmateix, la segona MUC l'any 1989 va definir la tasca d'emplenat de plantilles: es donava una plantilla per a l'esdeveniment d'un avistament de vaixells. La plantilla consistia en 10 camps (tipus d'esdeveniment, agent, temps, lloc, efecte, etc.) i els sistemes havien d'omplir-ne una per a cada avistament que aparegués als documents. Fins al MUC-5 es va mantenir aquesta tasca, variant la mida i el domini de la col·lecció, introduint mesures per a comparar el rendiment dels diversos sistemes i afegint altres idiomes com el Japonès.

  A partir del MUC-6, celebrat el 1995 es va incloure l'avaluació d'altres subtasques, a part de l'avaluació d'EI habitual (que es va passar a anomenar *Scenario Template*). En concret, es van avaluar el Reconeixement de *Named Entities** la Resolució de la Correferència*i el *Template Element* (estandardització dels conceptes de més baix nivell, com ara persones o organitzacions). Aquesta avaluació separada es va idear tenint en compte el fet que els sistemes d'EI depenen en gran mesura de la precisió i el *recall* d'aquestes subtasques.

  Tanmateix, no es van cel·lebrar més conferències després del MUC-7, el 1998. Altres trobades recullen ara la comunitat investigadora del camp, com ara les DUC[13] (*Document Understanding Conference*).

[11] `http://clef.iei.pi.cnr.it:2002/`
[12] `http://www.itl.nist.gov/iaui/894.02/related_projects/muc`
[13] `http://duc.nist.gov`

1.2 Alguns Sistemes de CR Online

Els sistemes de CR són una alternativa interessant als tradicionals cercadors d'Internet. Per altra banda, la Xarxa és un bon banc de proves per a aquests sistemes, ja que proporciona un gran nombre de documents i un gran nombre d'usuaris disposats a fer preguntes. Així que no ens ha de sorprendre que puguem trobar uns quants sistemes de CR *on-line* que cerquen la Web per a respondre les nostres preguntes. A títol de curiositat per aquells que vulguin provar-los, poso una petita llista d'alguns que he pogut provar, extreta de [Win]:

Answer Bus `http://www.answerbus.com/`

- Desenvolupat per Zhiping Zheng, de la Universitat de Saarbrücken.
- Funciona en Anglès, Francès, Espanyol, Alemany, Italià i Portuguès.
- Retorna el fragment on es troba la resposta.

Ask Jeeves `http://www.ask.com/`

- Desenvolupat per Ask Jeeves Inc., empresa dedicada a la RI.
- Actua com a cercador general o, en certs casos, com a cercador en FAQs.

Brain Boost `http://www.brainboost.com/`

- Retorna el fragment on es troba la resposta.

ExtrAns `http://www.ifi.unizh.ch/CL/extrans/`

- Desenvolupat per la Universitat de Zürich.
- Indexa pàgines **man**.
- Retorna el fragment on es troba la resposta.

IONaut `http://www.ionaut.com/`

- Desenvolupat pel projecte IO.
- Retorna la resposta exacta i els fragments on es troba la resposta.

START `http://www.ai.mit.edu/projects/infolab/`

- Desenvolupat per l'MIT[14].
- Retorna la resposta exacta i la font.

1.3 Tipologia de la CR

Els sistemes de CR poden haver de treballar en condicions diverses, com ja hem pogut veure explicant la seva història. Tanmateix, fóra bo fer-ne una tipologia de forma més detallada.

Una de les primeres distincions és el tipus de preguntes a què cal trobar resposta i la manera com cal proporcionar-la:

Factuals En aquests casos, la resposta a la pregunta és una entitat concret, bé sigui una persona, animal, objecte, lloc, quantitat, instant de temps. Habitualment són preguntes de tipus *Qui?*, *Què?* (no definicionals), *On?*, *Quan?* o *Quant?*.

Els sistemes que han de donar respostes factuals poden haver de donar la resposta exacta, sense cap paraula de més, o bé un fragment de text de major o menor tamany (encara que no excessivament grans, per a mantenir l'esperit de la Cerca de Resposta) en què es trobi aquesta. Per altra banda, en alguns casos es requereix que el text sigui un extracte d'algun

[14] Massachussets Institute of Technology

document de la col·lecció, mentre que altres cops es permet que la resposta sigui construïda lliurement.

Exemple: *Qui va escriure Tirant lo Blanc?*

- Joanot Martorell.
- ...Para I. Grifoll Joanot Martorell devia escriure el Tirant lo Blanc entre l'any 1460...

Exemple: *On va néixer Immanuel Kant?*

- Königsberg
- ...Kant, Immanuel (Königsberg 1724-*id.* 1804) Filòsof alemany...

Definicionals Són preguntes que demanen una descripció d'un cert concepte o persona. La resposta a proporcionar també pot ser únicament la definició o bé el fragment de text on aquesta es troba. Es tracta de les preguntes del tipus *Què és?* o *Qui és?*.

Exemple: *Què és un Erlenmeyer?*

- Un Erlenmeyer és un vas cònic de coll estret.
- ...un Erlenmeyer és un vas cònic de coll estret trobat a gairebé qualsevol laboratori de química...

Exemple: *Qui va ser Fausto Coppi?*

- Fausto Coppi va ser el guanyador del Tour del 1949.
- ...el guanyador del Tour del 1949, Fausto Coppi...

Enumeracionals Es tracta de preguntes la resposta de les quals és una llista d'elements que acompleixen una certa propietat. La resposta pot haver de requerir trobar tots els ens que apareguin a la col·lecció amb les característiques demanades, o bé només donar-ne un cert nombre. En quant a la manera de retornar-los, pot ser que només vulguem els elements o bé els fragments que els contenen, de forma similar a les preguntes factuals.

Exemple: *Digues tres monjos Zen.*

- Mumon, Bassui, Bodhidharma.
- 1. ...El monjo Zen Bassui va escriure una carta...
  2. ...el monjo Mumon, al segle tretze, va recopilar...
  3. ...els sis venerats monjos fundadors del Budisme Zen, de què Bodhidharma és el primer...

De resposta estesa Són preguntes en què la resposta és un procés. Corresponen típicament a les preguntes *Com?* i *Per què?*. Les respostes poden ser extractes directes de fragments de la col·lecció o, fins i tot, poden involucrar un procés de resum d'un fragment massa gran.

Exemple: *Com es fabrica el pa?*.

- Per a fabricar pa, es necessita farina, sal, oli, llevat i aigua. Es barregen els ingredients i es posen al forn.

Per altra banda, una altra distinció important a fer és si podem estar segurs o no que la pregunta té resposta a la col·lecció. Si no en podem estar segurs, cal afegir als sistemes algun mecanisme per a discriminar quan els candidats a resposta no són prou significatius com per a acceptar-los i indicar que no s'ha trobat resposta (situació que sol conèixer-se com a resposta NIL).

Un altre aspecte que pot influir són les característiques de la col·lecció de documents en què cerquem. La seva mida pot anar des de grans col·leccions fins a un únic document, com en els sistemes de Comprensió de Textos. Les tècniques que cal usar en un cas i altre difereixen força. Per altra banda, ens podem trobar amb col·leccions de textos de diferent qualitat lingüística:

1. Textos gramaticals: provinents d'enciclopèdies, serveis de notícies... El sistema **MURAX**, que usava l'Enciclopèdia Grolier, i els que treballen sobre la col·lecció AQUAINT pertanyen a aquesta primera categoria.

2. Textos amb agramaticalitats: com amb textos provinents d'internet o en general de fonts menys fiables. Les oracions agramaticals poden donar problemes als components menys lingüísticament robustos del sistema (com ara els Parsers), a part que és possible que hi hagi informació no gramatical o fins i tot no textual intercalada (taules, llistes, imatges). Aquesta informació addicional pot descartar-se o bé intentar aprofitar-la per a la CR (o fins i tot fer CR sobre ella: *Dona'm un mapa de Gran Bretanya.*. El sistema **START** del MIT, que apareix a la secció 1.2, per exemple, respon a preguntes sobre imatges).

3. Textos amb característiques especials: com ara els provinents de transcripcions orals, en què no hi ha ni signes de puntuació ni majúscules. A part, els sistemes de Reconeixement de la Parla (ASR, *Automatic Speech Recognition*)* tenen encara una baixa fiabilitat en situacions com la parla espontània (al voltant del 50%). Cercar respostes en documents on la meitat de les paraules són incorrectes adquireix una dificultat addicional.

   Un sistema que haurà de funcionar sota aquestes condicions serà el del projecte CHIL, de què parlarem després (secció 1.6).

4. Textos molt estructurats: provinents de transparències, presentacions, esquemes, documents en XML*... En aquest cas cal tenir en compte la disposició dels textos dins l'estructura del document, ja que molta informació que permet respondre les preguntes està en el format. En el mateix projecte CHIL també es vol intentar abordar aquest problema, en concret per al cas de les transparències.

Evidentment, algunes d'aquestes classificacions només tenen sentit en àmbits de recerca, i en situacions com les de les conferències de 1.1.3. Un sistema obert al públic ambiciós hauria d'estar en disposició de tractar qualsevol tipus de pregunta, eventualment sense resposta a la col·lecció.

1.3.1 Classificació dels Sistemes de CR

La tipologia de la secció anterior (1.3) fa referència al tipus de preguntes a què ha de trobar resposta el sistema i sota quines condicions haurà de fer-ho. Pel que fa a la classificació segons les característiques i les capacitats del sistema, l'habitual és usar la presentada a [MH99], que els discrimina segons tres criteris:

1. Bases de Coneixement*.

2. Capacitat de Raonament*.

3. Tècniques d'Indexació i PLN.

En funció d'aquests paràmetres distingeix 5 categories de sistemes, que a la vegada delimiten 5 categories de preguntes, segons les capacitats necessàries pels sistemes per a poder resoldre-les. Aquesta classificació està reproduïda a les taules 1.1 i 1.2.

Per contra, Vicedo, a la seva tesi [Vi02], proposa una classificació en 4 nivells a partir del criteri de les eines de PLN que usen:

1. Sistemes que no usen tècniques de PLN.

2. Sistemes que usen tècniques de tipus lèxico-semàntic.

3. Sistemes que usen tècniques de tipus semàntic.

4. Sistemes que usen tècniques de tipus contextual (correferències, coneixement general del món).

El mateix Vicedo fa una detallada exposició de cada categoria i dels sistemes existents en ella.

Classe	Coneixement	Raonament	Indexació/PLN
1	Diccionaris	Simple Heurístiques* *pattern matching**	Noms Complexos Aposicions Semàntica Simple Indexació de Paraules Clau
2	Ontologies*	de Baix Nivell	Nominalització de Verbs Semàntica Coherència Context
3	Grans Bases de Coneixement	de Nivell Mitjà	Tècniques Avançades de PLN Indexació Semàntica
4	Adquisició i Classificació de Coneixement del Domini Bases de Coneixement d'Alta Precisió	d'Alt Nivell	
5	Coneixement del Món	de Molt Alt Nivell de Propòsit Específic	

Taula 1.1: Classificació dels Sistemes de CR

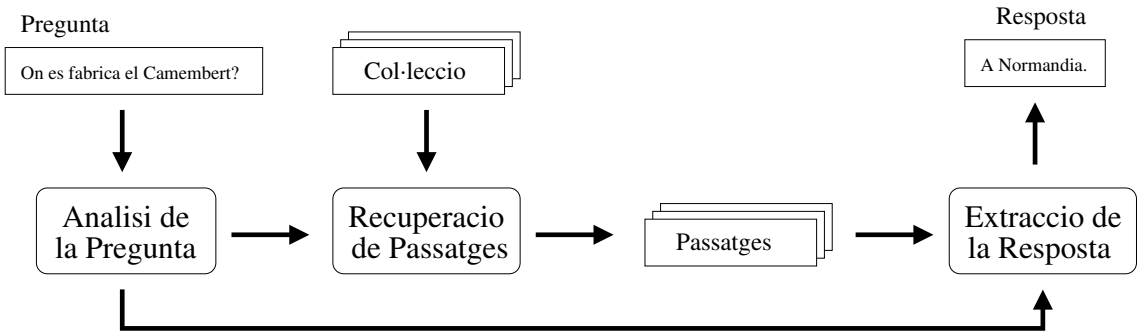


Figura 1.1: Arquitectura en 3 Etapes

1.4 Arquitectura dels Sistemes de CR

La majoria del sistemes de CR en Domini Obert segueixen una arquitectura en tres etapes. Aquesta arquitectura és per exemple la que usa el sistema del TALP-UdG (secció 1.5) o el sistema Lasso de la Southern Methodist University ([MH99]), i és la que apareix a la figura 1.1

Les etapes d'aquests sistemes són:

1.4.1 Anàlisi de la Pregunta

En aquesta primera etapa es pren la pregunta formulada per l'usuari i se n'extreu la informació necessària per a poder realitzar la Recuperació de Passatges i l'Extracció de la Resposta. Les característiques extretes poden variar segons els sistemes.

Tanmateix, un aspecte que la majoria dels sistemes utilitzen són les paraules clau (*keywords*) de la pregunta, per a la posterior Recuperació de Passatges. Fins i tot els sistemes que utilitzen només tècniques de RI sense PLN apliquen mètodes per a obtenir les paraules clau (Matthew Purver ho explica a la seva tesi doctoral [Pu00], parlant dels primers sistemes del TREC-8)

Per altra banda, els sistemes que usen tècniques de PLN acostumen a necessitar el Tipus de Pregunta (*Question Type*), terme amb què s'engloben factors com la tipificació de la pregunta

Classe	Exemples de Pregunta
1	*Qui va inventar el telègraf?* La resposta es troba en un fragment de text.
2	*Com es fa el Pa?* La resposta es troba en diverses oracions d'un paràgraf.
3	*Arguments a favor i en contra del PHN.* La resposta es troba repartida en diversos textos.
4	*El BCE hauria d'apujar els tipus d'interès?* La resposta es troba en un gran nombre de documents, sobre què cal raonar. La informació sobre el domini ha de ser adquirida automàticament.
5	*Què hauria de fer el Govern per aturar l'especulació immobiliària?* La resposta és la solució a un escenari complex, que involucra un gran nombre de factors.

Taula 1.2: Exemples de Preguntes per a la Classificació Anterior

(factual, enumeracional, definicional o de resposta extesa), la naturalesa esperada de la resposta (persones, objectes, llocs, organitzacions, quantitats...) o refinaments sobre aquests conceptes. Els diversos sistemes tenen diferents maneres de tractar el Tipus de Pregunta, però és freqüent que es defineixin taxonomies de tipus, que es poden consultar en el moment de l'Extracció de la Resposta (secció 1.4.3).

Aquesta informació pot obtenir-se de diverses formes, però en els sistemes que usen PLN s'acostumen a aplicar les tècniques estàndard com Tokenització*, Tagging*, Chunking*o Parsing*, per a posteriorment treballar amb tècniques més específiques, tant de la tasca com de la implementació.

1.4.2 Recuperació de Passatges

La Recuperació de Passatges pren les Paraules Clau obtingudes en l'etapa anterior, i eventualment algun altre tipus d'informació, i proporciona aquells passatges de la col·lecció de documents sobre què s'està fent CR que es consideren més prometedors a l'hora d'obtenir una resposta.

Per a una bona Recuperació de Passatges és important tant que es recuperin els passatges que contenen la resposta com que no es recuperin molts passatges irrellevants. En el cas que es perdin els passatges amb la resposta, serà impossible trobar-la; mentre que si s'obtenen molts passatges el rendiment del sistema se'n veurà penalitzat, ja que el procés d'Extracció de la Resposta sol ser costós i no es pot aplicar a un nombre arbitràriament alt de passatges.

Els sistemes de RI ofereixen funcionalitats que poden millorar la precisió i el *recall* i la majoria de sistemes de CR usen estratègies més enllà d'una recuperació directa de la Col·lecció.

Els sistemes de RI ofereixen consultes(*Queries*)* de diversos tipus:

Booleanes Obtenir els documents de la col·lecció en què apareix una expressió booleana de termes.

Exemple: $(rovelló \& llenega) \mid (fong \& !floridura)$

- Retorna els documents en què apareixen els mots **rovelló** i **llenega**, o bé hi apareix el mot **fong** i no el mot **floridura**.

Ranked Obtenir els documents en què apareixen una sèrie de termes, ordenats segons algun tipus de mesura de similaritat (per exemple, la mesura del cosinus proposada a [Sa89]).

Ponderades Consultes en què els diversos termes poden obtenir una ponderació diferent a l'hora de determinar la rellevància d'un document.

Fuzzy Obtenir els documents en què apareix un terme similar a un terme proposat.

De Termes Compostos Obtenir els documents en què un conjunt de paraules apareix de forma consecutiva.

Així, es poden trobar els documents en què aparegui l'expressió **Consell de Ministres**, i no tots aquells en què ho facin els mots **Consell**, **de** i **Ministres** de forma independent.

De Proximitat Obtenir els document en què un conjunt de paraules apareix en una secció de mida menor a un cert valor.

Per exemple, es pot cercar els documents en què apareguin els mots **President**, **Generalitat** i **Catalunya** en com a molt 10 paraules.

Amb *Wildcards* Obtenir els documents en què apareixen termes que satisfan un cert patró (per exemple, construïts amb els caràcters ? i $*$ habituals dels *shells**).

Així, **gener$*$o$*$** trobaria documents en què apareguin termes com **generosos**, **generacional** o **generacionals**.

De Rang Obtenir els documents en què apareguin termes dins un cert rang.

Per exemple, trobar els documents en què hi hagi termes entre **a** i **b**, és a dir, qualsevol que comenci per **a** o el mot **b**.

Un sistema real de RI que ofereix totes aquestes classes de Consulta és el **Lucene** del projecte Apache Jakarta ([Luc]).

Per altra banda, en alguns sistemes de RI es poden aplicar altres processos per a millorar la qualitat de la recuperació. Un dels més habituals és l'*stemming**, per a indexar la col·lecció per arrels, més que per paraules (per a més informació sobre l'*stemming*, es pot consultar l'article de Porter [Po80] o la seva web [PorW]).

Posteriorment, els documents recuperats es parteixen en passatges. També existeix un ventall de possibilitats a l'hora de dur a terme aquesta partició:

- Considerar passatges de mida fixa o variable.
- Imposar que als passatges apareguin totes les *Keywords* o bé permetre que n'hi hagi només un subconjunt.
- Generar els passatges amb major o menor solapament entre ells.

Els sistemes de CR utilitzen a més a més en aquesta etapa tècniques com:

- Combinar els resultats de diverses recuperacions. Per exemple, es pot realitzar una consulta amb tots els mots i una altra amb només les *Named Entities* de la pregunta, i ajuntar els resultats de l'una i l'altra.
- Usar *FeedBack*: si els passatges obtinguts no es consideren de prou qualitat o n'hi ha una quantitat massa gran o massa petita, es pot repetir el procés de recuperació afegint o traient paraules clau de la pregunta, o bé relaxant o enfortint les restriccions de la recuperació.
- Realitzar expansió de *Queries*: si alguna paraula clau és un acrònim o un nom propi, es pot ampliar la consulta amb les altres formes en què aquest terme pot aparèixer. Per exemple, si un terme és **Facultat d'Informàtica de Barcelona** pot resultar interessant buscar també documents en què figuri l'acrònim **FIB**. Per altra banda, si volem cercar documents sobre **Jaume I**, es pot fer la consulta amb els termes **Jaume I**, **Jaume el Conqueridor**, **El Rei Jaume** o **Jaume d'Aragó**.

1.4.3 Extracció de la Resposta

El darrer pas del procés de CR és l'extracció de la resposta a partir dels passatges obtinguts en la Recuperació de Passatges i de la informació extreta en l'Anàlisi de la Pregunta (especialment el Tipus de Pregunta). El resultat final ha de ser el o els fragments que es retornaran a l'usuari com a resposta, possiblement ordenats per rellevància.

Aquesta darrera etapa sol ser la més costosa, ja que cal utilitzar processos de PLN sobre tots els passatges obtinguts. Per aquesta raó la cobertura de la Recuperació de Passatges resulta determinant en el rendiment del sistema.

Les tècniques usades aquí són força diverses, i van des de l'utilització de tècniques simples de *pattern matching* fins la de mesures de similaritat entre l'estructura de la pregunta i la de les oracions dels passatges.

1.5 La Metodologia TALP - UdG

De cara al TREC-12 i al CLEF-2003, la Universitat de Girona i el centre TALP de la UPC van desenvolupar una metodologia de CR i van fer-ne una implementació per a prendre part en l'avaluació de la tasca de CR de les dues conferències.

El sistema estava pensat per a l'Anglès i l'Espanyol, cercava resposta a preguntes de tipus factual i tenia l'arquitectura en 3 capes explicada a 1.4. Se'n pot trobar una descripció detallada a [FM03]. Tanmateix, a continuació farem un petit esbós dels seus components.

1.5.1 Anàlisi de la Pregunta

El primer pas que tenia lloc era aplicar processos de PLN a la pregunta. La informació obtinguda era necessària per a les fases posteriors, i incloïa:

Informació morfològica El text era tokenitzat*, i de cada token se'n trobava el tag (categoria gramatical)* i el lema (arrel del mot)*. A part, les *Named Entities* es detectaven i s'unien els termes compostos per més d'un token en un de sol. Amb el text tokenitzat també es buscava la segmentació en oracions.

Les eines usades per a aquest procés diferien segons l'idioma:

- En Espanyol, en primer lloc s'aplicava l'analitzador morfosintàctic integrat **Ms-Analyze** desenvolupat pel TALP i el CLiC[15] de la UB[16], que duia a terme les tasques de Tokenització, Segmentació*, Tagging i Lematització*, detectant al mateix temps expressions de quantitat, temps i dates ([AC98]).

  A continuació d'usava el detector i classificador de *Named Entities* **Abionet** ([CM02]), produït també al TALP.

- Per a l'Anglès, com a tagger es va utilitzar el **TnT**, de Thorsten Brants ([Br00]). Com que el **TnT** no proporcionava els lemes, es va aprofitar el lematitzador de **WordNet*** **1.7**([Wn98]). I el mateix **Abionet**, amb un model entrenat per a la llengua anglesa, va ser l'eina amb què es detectaven les *Named Entities*.

  Per a la tokenització i segmentació es van utilitzar Autòmats Finits.

Informació semàntica Per a cada token es buscaven els sentits que el mot podia tenir. Per a l'anotació de sentits s'usaven els synsets (conjunt de mots que actuen com a sinònims d'un determinat concepte, identificat amb un codi)*de **WordNet** (base de dades lèxica d'ús habitual en el domini del PLN, [Wn98])*. No s'intentava fer cap mena de desambiguació dels sentits.

[15] Centre de Lingüística Computacional
[16] Universitat de Barcelona

Un altre aspecte semàntic que es buscava era la determinació del tipus de *Named Entity*. En concret es distingia entre 4 categories: persones, organitzacions, llocs i altres. Per a l'Anglès, addicionalment, els llocs es subclassificaven de forma més fina en categories com ciutat, pic, riu, port...

També es consultaven aspectes com si el mot era un actor d'un cert verb (*pintor* → *pintar*) o si era un gentilici (*barceloní* → *Barcelona*).

Les eines usades per a dur a terme aquesta part del procés van ser:

- En tots dos casos es va usar la versió corresponent de **WordNet** (espanyola o anglesa) per a l'anotació dels synsets.
- De la classificació de les *Named Entities* en les 4 categories proposades se n'encarregava el mateix **Abionet**.
- Per a la subcategorització dels llocs en Anglès, es va usar una sèrie de classificadors binaris apresos utilitzats el sistema d'*Inductive Logic Programming** **Foil**([Qu93]). El sistema està descrit a [FM04a].
- Els actors i gentilicis es buscaven en gazetteers*, generats mitjançant l'estudi de corpus per als gentilicis i mitjançant una anàlisi de les glosses de **WordNet** per als actors.

Informació Semàntica Es duia a terme el parsing (cerca de l'estructura sintàctica)* o chunking (cerca de sintagmes) * de la pregunta.

En concret, es va usar:

- Per a l'Espanyol, el chunker **Tacat**, desenvolupat al TALP.
- Per a l'Anglès, el parser **Minipar**([Li98]), adaptant la seva sortida al format del **Tacat**.

Com a resultat d'aquest procés, s'obtenien dues estructures que contenien informació sobre la pregunta:

Sent Contenia la informació pròpia de cada token*: forma, lema, categoria gramatical, classe de *Named Entity*(Persona, Organització, Lloc (subclassificat si era possible), o Altres) si era aplicable; synsets de **WordNet** amb la seva cadena d'hiperònims, TCO i codis de Magnini; i, finalment, la informació relativa als actors i gentilicis.

Sint Contenia informació sobre els components sintàctics de la pregunta (bàsicament sintagmes nominals, preposicionals i verbals) i sobre les dependències i altres relacions entre aquests components.

Amb aquestes dues estructures es determinaven quatre aspectes de la pregunta:

Tipus de Pregunta Es va entrenar un sistema usant **Foil** per a classificar cada pregunta en una categoria de les 50 que es van definir per a preguntes factuals (entre les quals hi havia, per exemple, *Who_action*, *Who_person_quality*, *When_begins* o *When_action*) . El Tipus de Pregunta intentava capturar la naturalesa esperada de la resposta i restriccions addicionals sobre ella (distingint per exemple, si es buscava una persona que tingués una qualitat o be que hagués fet una acció).

Paraules Clau Al mateix temps que es determinava el Tipus de Pregunta, s'extreien els termes que actuarien com a *keyword* en l'etapa de Recuperació de Passatges. Aquests termes havien de ser noms, adjectius, verbs o numerals.

Entorn(*Environment*) Intentava representar les relacions i categories semàntiques existents entre els diversos components de la pregunta. Es van definir 100 classes semàntiques i 25 relacions entre elles, formant dues jerarquies. Aquestes classes estaven relacionades amb els Tipus de Pregunta definits, de manera que en la fase d'Extracció s'usaven les relacions existents (com veurem a 1.5.3).

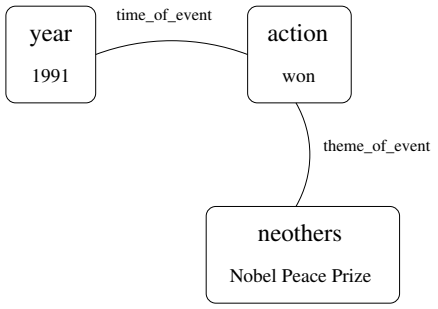


Figura 1.2: Entorn de *Who won the Nobel Peace Prize in 1991*

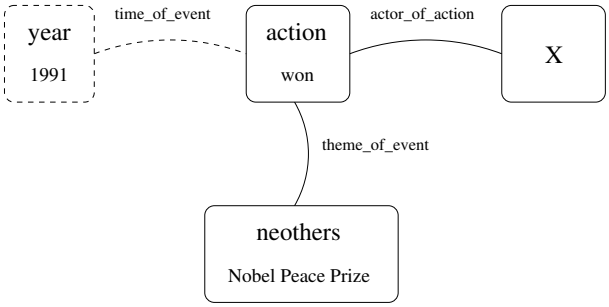


Figura 1.3: Restriccions de *Who won the Nobel Peace Prize in 1991*

Per exemple, una pregunta de tipus *Who_action* havia de trobar una instància de la classe *Human_action* en el text de la pregunta, i el passatge on havia de ser la resposta havia de tenir una instància de *Human_action*, una de *Human* i una relació *Actor_of_action* entre l'una i l'altra.

La determinació de l'Entorn tenia lloc mitjançant un conjunt d'unes 150 regles manuals.

Per exemple, l'Entorn de la pregunta *Who won the Nobel Peace Prize in 1991?* seria el que apareix a la figura 1.2.

Restriccions(*Constraints*) A partir de l'Entorn i del Tipus de la pregunta, calia determinar les restriccions que havia d'acomplir l'Entorn de la Resposta. Tanmateix, no totes les restriccions que apareixien a l'Entorn de la Pregunta eren necessàries per a l'extracció, de vegades calia afegir-ne d'addicionals, i algunes altres relacions havien de ser exteses, refinades o modificades. Per aquesta raó, es generava una llista de Restriccions, que al mateix temps es classificaven en obligatòries i opcionals.

Així, les Restriccions de la pregunta anterior series les de la figura 1.3

1.5.2 Recuperació de Passatges

Com a sistema de RI es va usar el sistema MG, descrit a [WM99]. En concret, es va usar la versió 1.3.1x, extensió no oficial feta pública per la Universitat de Utah a la seva web `http://www.math.utah.edu/pub/mg`.

Com a preprocés de la recuperació de passatges es va indexar la col·lecció AQUAINT de dues maneres diferents:

1. Tenint en compte els tokens i les *Named Entities* (**Joan Salvat-Papasseit** ja no eren dos termes sinó un únic terme **Joan_Salvat-Papasseit**).

2. Tenint en compte les *Named Entities* però no els tokens.

3. Tenint en compte els tokens però no les *Named Entities*.

La Recuperació de Passatges rebia les paraules clau de la pregunta que s'havien detectat a l'etapa d'Anàlisi, dividides segons si es tractava o no de *Named Entities*.

En primera instància, es feien dues recuperacions: una de tipus *Ranked* usant totes les paraules clau i el primer índex, i una altra de tipus booleà amb només les paraules clau *Named Entity* i usant el segon índex. En tots dos casos es fixava un màxim de documents a recuperar. En la *Ranked* addicionalment es podia imposar un llindar: només s'admetien els documents la puntuació del qual estigués per sobre d'una certa fracció de la puntuació del primer.

Addicionalment, per a la *query* de *Named Entities* es podia realitzar expansió d'acrònims i d'àlies, fent la consulta no només amb els termes que apareixien a la pregunta, sinó també amb les seves variants (veure secció 1.4.2). La determinació de les expansions dels acrònims i àlies va tenir lloc de forma prèvia (el procés es descriu a [FM04b]).

Amb aquestes dues recuperacions s'obtenien dues llistes amb els identificadors dels documents sel·leccionats. Aquestes dues llistes es fusionaven (eliminant els repetits) i es recuperava el text original dels documents usant el tercer índex. Sobre aquest text s'aplicava la generació de passatges.

Com a passatges es prenien seccions de mida fixa dels documents (en la versió operativa es va optar per 3 paràgrafs) amb un cert solapament entre elles (per exemple, de 1 paràgraf, fet que significava que el primer paràgraf de cada passatge era el mateix que el darrer de l'anterior).

Tots els passatges així generats s'indexaven a nivell de token, sense agrupar les *Named Entites*, i sobre ells es feia una darrera recuperació de tipus *Ranked*. En ella també es podia imposar un nombre màxim de passatges i un llindar en la puntuació, i la consulta usava totes les paraules clau.

Un exemple de tot aquest procés és el que apareix a la figura 1.4.

Els passatges obtinguts al final d'aquest procés eren els que passaven a la tercera i última etapa com a candidats a proporcionar una resposta.

1.5.3 Extracció de la Resposta

El primer pas amb els passatges obtinguts a l'etapa anterior era aplicar-los la mateixa cadena de processadors lingüístics que a la pregunta, segmentar-los en oracions i llavors puntuar les oracions usant el que Vicedo a la seva tesi anomena *Contingut Semàntic del Concepte*. (CSC, veure [Vi02]).

L'extracció tenia lloc només en les oracions dels passatges obtinguts en la recuperació, i consistia en construir l'Entorn de cada oració candidata i intentar fer un mapeig entre les relacions semàntiques d'aquest Entorn i les restriccions extretes de la pregunta. Les restriccions obligatòries havien de ser satisfetes per a considerar la candidata, mentre que les opcionals només augmentaven la seva puntuació. Sobre aquestes restriccions es podia aplicar relaxació: es podia transformar les restriccions obligatòries en opcionals i/o canviar-les per restriccions que es trobessin més amunt en la taxonomia de restriccions (generalitzar-les).

L'algorisme d'Extracció era:

1. Obtenir les restriccions de la pregunta.

2. Per a cada oració dels passatges candidats:

3. Obtenir l'entorn de l'oració.

4. Comprovar si s'acompleixen les restriccions obligatòries, i si s'acompleix alguna d'opcional.

5. Si no s'acompleixen, descartar l'oració.

6. Si s'acompleixen, intentar aplicar regles per a trobar una resposta en aquella oració.

7. Un cop processades totes les oracions,

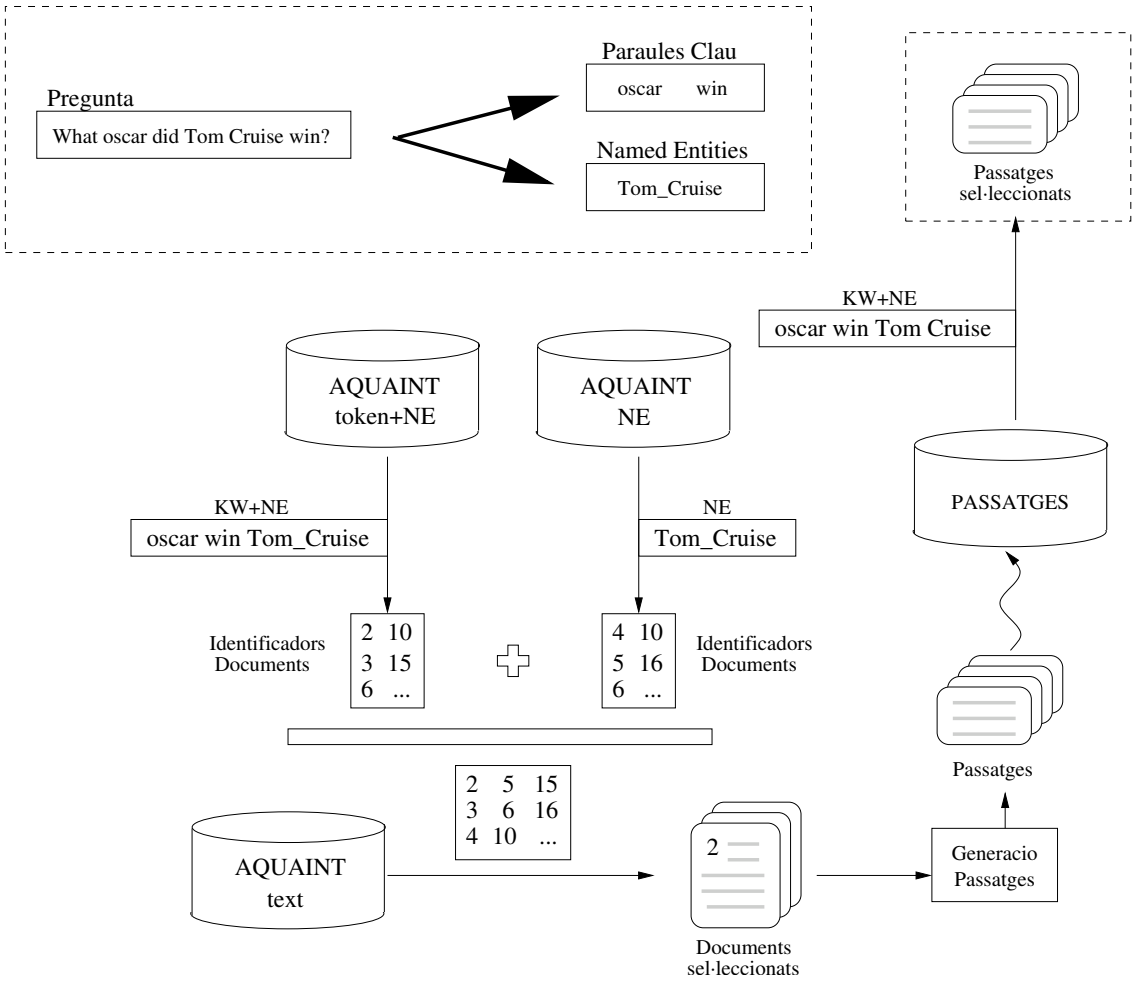


Figura 1.4: Recuperació de Passatges

8. Si no s'ha trobat com a mínim una resposta i hi ha relaxacions possibles, relaxar les restriccions i tornar a 2.

9. Si no s'ha trobat com a mínim una resposta i no hi ha més relaxacions possibles, retornar 'sense resposta'.

10. Altrament, retornar les respostes obtingudes.

Les regles que s'aplicaven en l'extracció eren, per exemple:

1. Per a preguntes de tipus **howmuch**:

2. si

   - hi ha un **event** C a l'Entorn de la resposta,
   - hi ha una relació de **partipant_in_event** entre aquesta C i un element U,
   - aquest element U té la propietat **entity_has_quality**,
   - hi ha una relació **which_quality** entre U i un element Q,
   - i aquesta Q té una relació **which_value** amb un element A,

3. llavors,

   - aquest element A és una possible resposta.

Si hi havia més d'una resposta possible, s'utilitzaven factors com el CSC de la oració, el nombre de Restriccions opcionals satisfetes, la fiabilitat de les regles usades per a obtenir-la... per a dur a terme un darrer procés de votació i, finalment, tria de la resposta que es retornava a l'usuari.

1.5.4 Resultats Obtinguts

Com hem comentat, el sistema TALP-UdG va prendre part en la *Question Answering Track* del TREC-12 i en l'avaluació de Cerca de Resposta del CLEF-2003. Quins van ser els resultats? Al mateix article [FM03] ens apareixen per al cas concret del TREC-12:

De les 413 preguntes a què va donar resposta, 22 van ser jutjades com a resposta exacta, i 2 més van ser considerades incorrectes. Amb aquestes xifres, es veu que la precisió era del 5.3%. Pel que fa a la capacitat de reconeixement de preguntes sense resposta, es va assolir una precisió del 9.2%, i un *recall* del 43.3%. Això és degut a que quan el sistema era incapaç de trobar una resposta, assumia que no en tenia, fet que va succeir en 141 preguntes, i va fer augmentar el *recall* de forma significativa.

Pel que fa a les diverses fases, la determinació del Tipus de Pregunta va obtenir una precisió del 69%, mentre que la Recuperació de Passatges només va proporcionar els passatges on es trobava la resposta a 157 de les 383 preguntes que tenien resposta en la col·lecció. De manera que aquesta etapa tenia una precisió del 38%. Pel que fa a l'Extracció de la Resposta, la precisió va ser del 8.3%.

Els resultats doncs mostraven que calia continuar treballant sobre el sistema. A ulls dels creadors, això era esperable, ja que es tractava més aviat d'un prototipus molt bàsic que d'un sistema funcional. Alguns components necessitaven un intens procés de millora.

1.6 El Projecte CHIL

El projecte CHIL(*Computers in the Human Interaction Loop*)[17] és un Projecte Integrat (*Integrated Project*, IP) dins el 6è Programa Marc per a la recerca de la Unió Europea. És un projecte que involucra 15 universitats i empreses europees i nord-americanes.

[17] `http://chil.server.de/`

L'objectiu d'aquest projecte és crear entorns en què persones estiguin interactuant amb altres persones i hi hagi un sistema que monitoritzi l'activitat humana i hi reaccioni proactivament, detectant les necessitats humanes i satisfent-les amb els seus serveis, però sense ser intrusiu. La idea és que la màquina sigui capaç d'entendre el context perceptual humà , i de proporcionar ajuda de forma implícita, amb un mínim d'atenció per part de les persones.

Per aconseguir-ho, el projecte es concentra en la recerca sobre tecnologies associades a:

- Interfícies Perceptuals d'Usuari Multimodals, que observin, reconeguin i interpretin tota la informació necessària per a detectar i explicar les activitats i intencions humanes.
- Serveis basats en la comprensió de la activitat humana i la detecció del context perceptual. Aquests serveis han d'equilibrar les intervencions implícites i explícites de la màquina, i proporcionar la informació en una forma adequada. La idea és oferir serveis com a suport a la memòria humana, facilitació de les comunicacions humà-humà i suport a reunions.
- Una Infrastructura que suporti els serveis mencionats, així com computació autònoma, *software* auto-reparable, arquitectures flexibles i comunicació en xarxa de dispositius de forma intermitent i dinàmica.

Un escenari que s'ha proposat per a provar l'evolució de la recerca que fomenta el CHIL és el següent:

> *Imaginem una sala dotada de càmeres i micròfons, així com pantalles individuals i col·lectives per a mostrar informació. Dins té lloc una reunió amb diversos participants humans. El sistema capta els moviments i la veu de les persones, detecta què es diu i qui ho diu, així com els gestos que es fan. La idea és que el sistema registra l'activitat que té lloc en la sala, i és capaç de donar suport a la reunió. Així, és capaç de:*
>
> - *Respondre a preguntes fetes pels participants.*
> - *Generar resums i actes de la reunió quan aquesta acaba.*
> - *Posar en contacte els participants amb altra gent que es trobi fora (trucant per telèfon o enviant un mail).*
> - *Recordar a cada participant informació sobre la resta de la gent, o sobre reunions anteriors.*
> - *I molt més...*

La UPC és un dels principals *partners* del projecte CHIL. El centre TALP hi participa tant en la seva secció de Processament de la Veu com la de Processament de Llenguatge Natural, i també hi pren part el grup de Tractament d'Imatge de la UPC, del departament de Teoria del Senyal i Comunicacions (TSC).

Bàsicament, l'aportació del grup de PLN del TALP dins CHIL és l'estudi de tècniques eficaces i eficients per a Resum Automàtic, Detecció del Tema* i CR sobre transcripcions orals o textos escrits.

1.7 Integració

El meu objectiu en aquest Projecte Final de Carrera ha estat el de dur a terme la integració de la implementació existent del prototips de CR desenvolupat per la UdG i el TALP.

Calia redissenyar i reimplementar els elements de la cadena de procés i integrar-los en un únic sistema, agrupant els diversos processadors (analitzadors morfològics, sintàctics i semàntics; recuperadors d'informació...) i recursos lingüístics (diccionaris, lexicons* , WordNet...), i dotant al sistema d'una estructura modular.

Per altra banda, com que la meva feina es desenvolupava en el marc del projecte CHIL, el sistema produït havia d'orientar-se cap a les necessitats d'aquest projecte. Recordem que una de

les aportacions del grup de PLN del TALP dins CHIL és l'estudi de la CR sobre transcripcions orals o textos escrits. Era necessari doncs tractar textos amb característiques molt heterogènies i, com a conseqüència, necessitàvem produir un sistema de CR integrat i parametritzat.

Possibles paràmetres serien:

- l'idioma dels documents
- el tipus de document: transcripcions oral, textos gramaticals, pàgines web, tranparències...
- el registre: més o menys formal, més o menys tècnic...
- el tipus de pregunta (tal com havíem explicat anteriorment)

Addicionalment, es volia aconseguir un disseny flexible que permetés:

- Canviar fàcilment mòduls per altres que tinguin la mateixa funcionalitat, i integrar-ne de nous, així com modificar els paràmetres i/o models usats en cada mòdul (per a adaptar-los a textos de diferents fonts).
- Tenir un manteniment fàcil i barat del sistema.
- Reutilitzar el *lingware* produït en aquest projecte de cara a implementar altres tipus d'eines de Processament del Llenguatge Natural, com ara resumidors, classificadors de documents, extractors d'informació...

Va ser amb aquest objectiu en ment que es va iniciar la meva feina en el present projecte, i els resultats dels mesos de treball són els que es recullen en aquesta memòria.

Capítol 2

Anàlisi de Requeriments

El desenvolupament del projecte es va iniciar amb un estudi de la implementació ja existent, i amb una anàlisi dels requeriments del nou sistema, per a poder dissenyar una arquitectura capaç d'acomplir-los.

2.1 Característiques de la Implementació Existent

- **Estava estructurada com un conjunt de subprogrames amb poca relació entre ells.**

  Per a poder-lo executar, calia conèixer tots els programes a cridar i l'ordre concret en què fer-ho. Hi havia alguns *scripts** que agrupaven algunes etapes, però no hi havia una forma directa d'invocar tot el procés.

- **Estava geogràficament dispersa.**

  Degut a que el sistema havia estat desenvolupat de forma conjunta pel centre TALP i la Universitat de Girona, els diferents components es trobaven distribuïts entre les màquines del dos grups, i calia traslladar els fitxers entre unes i altres. Aquest fet feia encara més difícil l'execució del sistema.

- **Estava pensada per a treballar en mode *batch**.**

  Buscava respostes a una llista de preguntes sobre una certa col·lecció. Per a respondre a preguntes de forma individual, calia recórrer tot el procés per a cadascuna d'elles, assolint els costos d'inicialització de les etapes cada cop, costos que podien ser elevats en alguns casos respecte al cost de procés pròpiament dit per a una sola pregunta.

- **Els paràmetres de configuració dels diferents components havien d'especificar-se com a opcions de línia de comandes.**

  Aquest fet feia que per a retocar les característiques d'un cert component calgués resseguir els *scripts* on s'activava, i anar canviant els paràmetres.

  Els programes regulats per *switchos** suposen sempre un esforç d'atenció a l'hora d'invocar-los, i per altra banda aquests paràmetres no es podien canviar en temps d'execució: calia aturar el procés i tornar-lo a engegar.

- **Estava formada per components heterogenis.**

  Els diversos subprogrames procedien de diverses fonts i tenien diferents característiques: hi havia codi en **Prolog**, **Perl** i **C**, i binaris de què no es disposava del codi.

- **Requeria una estructura concreta dels directoris i els noms dels fitxers on es guardaven les dades**

Així, el nom dels fitxers havia de ser un prefix més el número de la pregunta a què pertanyien, i estar en subdirectoris d'un únic directori amb noms concrets, com ara **sent**, **ca**, **subcat**...

Aquesta estructura rígida feia difícil adaptar components individuals per a altres tasques o canviar l'arquitectura del sistema.

2.2 Objectius a Assolir

- **Volíem un sistema integrat.**

  Havia de permetre executar tot el procés de CR de forma senzilla.

- **Volíem un sistema flexible.**

  Havia de ser possible configurar i adaptar cadascun dels components del sistema de forma fàcil, així com canviar-los per altres d'homòlegs si esdevenia necessari. Per altra banda, també havia de ser possible canviar l'estructura de la cadena de processos sense haver de canviar tots els mòduls.

- **Volíem un sistema que permetés aprofitar el codi ja existent.**

  Per a facilitar la migració de la implementació actual a la nova, resultava fonamental poder reciclar-ne el codi.

- **Volíem un sistema que permetés la cohabitació de codi en diversos llenguatges, així com de binaris de què no es disposi de codi font.**

  Tant per a poder assolir el requisit anterior com per a permetre el treball amb eines externes de què no es disposa el codi, havia de ser possible integrar components de característiques dispars en aquest aspecte. Per altra banda, el poder treballar amb diversos llenguatges permet explotar els avantatges de cadascun d'ells en aquells aspectes on siguin més adequats.

- **Volíem un sistema que evités haver d'inicialitzar repetidament els components.**

  Si es treballa en mode *batch*, com es feia anteriorment, els costos d'inicialització no són un problema, però si es fan processa de forma individual cada pregunta, com quan es treballa en mode interactiu*,poden passar a representar la major part del cost d'alguna etapa (un exemple n'és l'analitzador morfosintàctic **FreeLing** (veure [CC04]), per a què el temps d'inicialització es de diversos segons mentre que per a taggejar* una frase triga unes centèsimes). Per aquesta raó resultaria interessant poder mantenir els components en un estat latent mentre estan actius altres.

- **Volíem un sistema que funcionés sobre el Sistema Operatiu Linux**

  Linux, a part de ser un dels Sistemes Operatius capdavanters en el Món de la Informàtica avui dia, amb totes els avantatges que suposa el ser un Sistema Operatiu de codi obert, tant a nivell de cost com de qualitat, havia estat l'escollit com a plataforma per al projecte CHIL, i el sistema a desenvolupar havia de córrer sobre ell.

2.3 Altres Requeriments

Addicionalment als requeriments imposats pels objectius de la secció 2.2, a l'hora de pensar en l'Arquitectura i el Disseny del Sistema que finalment ha estat implementat, es van tenir en compte aquests altres criteris:

- **Volíem que l'arquitectura i els components desenvolupats poguessin aprofitar-se per a altres sistemes de PLN, encara que no fossin de CR.**

  D'aquesta manera, s'afavoriria la reutilització del *software*, amb tots els beneficis que aquesta comporta (components més provats i, per tant, menys propensos a error; disminució dels costos de producció...).

- **Volíem que el sistema es produís seguint metodologies d'Enginyeria del Software***

  L'Enginyeria del Software ofereix mètodes provats que resolen els problemes que un informàtic es troba en el seu dia a dia, evitant haver de buscar solucions noves a cada problema. D'aquesta manera es disminueixen els costos i augmenta la qualitat del producte final.

  Addicionalment, com que aquest sistema ha de funcionar com una plataforma on s'afegiran noves components, és important que els nous desenvolupadors puguin entendre la seva estructura i usar els recursos que aquesta posa a la seva disposició. En aquest sentit, l'Enginyeria del Software esdevé important ja que les sol·lucions que ofereix poden considerar-se estàndard i, per exemple, entendre un sistema dissenyat utilitzant patrons* resulta més senzill que fer-ho amb un sistema creat de forma totalment *ad hoc*. Amb els patrons, hi ha un substrat comú que tots els desenvolupadors comparteixen (encara que sigui parcialment) i la comprensió no ha de començar d'un nivell tan baix com si cada programador utilitza les seves pròpies solucions. Per la mateixa raó, aquest substrat compartit també facilita la comunicació entre membres d'un mateix equip.

  Un altre aspecte a tenir en compte és el de les proves: l'Enginyeria del Software ofereix metodologies de prova (de caixa blanca, de caixa negra, d'integració...) que seguides de forma sistemàtica permeten reduir la probabilitat d'aparició de *bugs**.

  Així doncs, l'Enginyeria del Software ens ofereix respostes mínimament estàndard a gran part dels problemes que ens apareixen durant el desenvolupament d'un projecte, i és per aquesta raó que considerem l'ús de les seves eines un requeriment del projecte (a [GH94] apareixen altres arguments a favor de l'ús de les eines de l'Enginyeria del Software i, especialment, dels Patrons de Disseny).

- **Volíem un sistema que no depengués d'una disposició concreta de fitxers en els directoris del disc**

  Volíem evitar la rigidesa imposada per l'altre sistema en l'arbre de directoris i els noms dels fitxers de cara als diferents processos. L'ideal seria que els noms i les localitzacions dels diferents fitxers usats i produïts al llarg del procés poguessin ser qualssevol, per a permetre als usuaris emmagatzemar la seva informació de la manera desitjada per ells. Per altra banda, d'aquesta manera seria possible aprofitar els components desenvolupats per a altres tasques sense haver d'estar retocant els noms de documents ja existents, ni haver de moure de forma innecessària els fitxers generats a punts concrets de l'arbre de directoris.

Capítol 3

Especificació i Disseny

3.1 El·lecció de l'Arquitectura del Sistema

Un cop determinats els requeriments del sistema, la primera decisió a prendre va ser l'el·lecció de l'arquitectura sota què organitzaríem i coordinaríem els diversos components del sistema.

3.1.1 Arquitectures Possibles

Les alternatives principals que es van considerar foren:

- Mantenir l'arquitectura de *pipeline** de processos independents comunicats per pipes i amb els resultats intermitjos emmagatzemats en fitxers a disc, ja que són útils per a detectar i resoldre problemes.

- Dissenyar el sistema com un únic programa seguint el paradigma d'Orientació a Objectes. Cada mòdul i la comunicació entre ells serien dissenyats com a classes d'objectes.

  Per exemple, es va pensar en tenir una jerarquia que comencés en la classe Procés, de què heretessin les classes Tokenitzador, Tagger, Analitzador Semàntic... A la vegada, els diferents mòduls d'un mateix tipus, com ara, els diferents Taggers (**TnT**, **Brill**, **FreeLing**...) estarien cadascun en una classe, filla de la classe del tipus de Procés corresponent (en aquest cas, de la classe Tagger).

- Emprar l'Orientació a Objectes sobre una arquitectura de Client/Servidor, transformant cada mòdul en un Servidor, i l'aplicació en un Client que va demanant a cada Servidor que tracti el text.

  Seguint l'exemple anterior, els diferents Taggers estarien cadascun en un procés, a l'espera de Clients que els demanessin de taggejar els seus textos.

3.1.2 Arquitectura Escollida

Finalment, es va optar per l'arquitectura Client/Servidor descrita anteriorment ja que, com veurem més endavant, és la que s'adapta millor a les nostres necessitats. Tanmateix, per a poder obtenir els màxims avantatges d'ella, es van prendre les següents decisions de disseny:

- La comunicació entre Clients i Servidors té lloc mitjançant un servidor intermig, que podem anomenar *MetaServidor*. El MetaServidor té la tasca de determinar quin és el Servidor que pot atendre millor les peticions d'un cert Client. Aquesta decisió es fa mitjançant un conjunt de regles llegides d'un fitxer de configuració i que inicien un cert Servidor si la petició acompleix una sèrie de condicions. Es tracta d'un sistema senzill però versàtil, i com que aquestes regles es troben al costat Servidor es poden controlar alguns problemes de seguretat

causats per Clients maliciosos: el codi que finalment s'executa està determinat per la persona que configura el Servidor. La naturalesa d'aquestes regles es descriu a la secció 3.3.2.

- Els Clients es comuniquen amb el MetaServidor mitjançant sockets TCP. Aquest mètode permet connexions i desconnexions de forma dinàmica, i sense que ni tan sols Client i MetaServidor siguin a la mateixa màquina.

  Per altra banda, com que el MetaServidor és qui crea els Servidors com a processos fills seus, la comunicació entre el primer i aquests té lloc mitjançant *pipes**, capturant l'entrada i sortida estàndard dels Servidors.

- Les peticions del Client al Servidor i al Metaservidor, així com les respostes que es donen tots ells, segueixen un protocol de text pla (inspirat en el **HTTP**), per a facilitar el processat, i evitar els problemes associats a comunicacions binaries, com ara mides i ordenació de bits dels enters en diferents plataformes (encara que ara per ara el sistema ha de funcionar sobre una sola màquina, en un futur aquesta arquitectura permetria treballar també sobre un *cluster* de màquines, com ara el que s'acaba de construir al TALP. Tanmateix, una ampliació així requeriria una extensió del protocol per a tractar aspectes com la comunicació entre MetaServidors, de manera que queda com a possibilitat de treball futur (capítol 5)).

- Els textos a tractar es troben en fitxers a disc. El format d'aquests fitxers és un aspecte que es tracta detingudament a la secció 3.2.1, però seguint la filosofia de fer-ho portable entre plataformes, és important que siguin de text pla.

- De cara a facilitar el reaprofitament del codi ja existent, el requeriment que els Servidors i Clients segueixin el paradigma d'Orientació a Objectes podem relaxar-lo, transformant-ho en una recomanació. D'aquesta manera, serà important que donem una infrastructura Orientada a Objectes per als nous components que es dissenyin, però també serà important que puguin integrar-se components implementats seguint altres filosofies, des de la programació estructurada fins la funcional o la lògica.

3.1.3 Avantatges de l'Arquitectura Proposada

Aquest disseny té els següents avantatges que el fan interessant per a les nostres necessitats:

- El fet que cada servidor i cada client siguin processos independents unit a que la comunicació es faci mitjançant un protocol de text pla i fitxers a disc facilita el canvi dels diversos mòduls, així com la coexistència de mòduls homòlegs amb diferents característiques.

  Per altra banda, els diversos mòduls poden estar escrits en diferents llenguatges de programació. Això és necessari si es vol aprofitar el codi de la implementació ja existent, que barreja components en **C**, **Perl** i **Prolog**, i pot resultar interessant, ja que permet aprofitar els beneficis de cada llenguatge. Amb una arquitectura d'un únic programa hauríem hagut d'usar APIs de comunicació externa (interfícies **C-Prolog**, **C-Perl**...), que d'aquesta manera evitem.

- Resulta senzill adaptar el codi ja existent, ja que els servidors només necessiten estar atents a la seva entrada estàndard per a processar els textos sota demanda dels clients. El codi d'atenció a peticions té una part comuna per a tots ells, que es pot mantenir en una llibreria; i la part específica és la que es pot aprofitar.

- El procés no té perquè ser una cadena lineal com seria en el cas de l'arquitectura de *pipeline*, sinó que es pot ramificar segons les necessitats. Aquesta facilitat permet seguir metodologies més complexes i, per tant, dóna més versatilitat al nostre sistema.

- El treball mitjançant arxius a disc permet treballar amb quantitats d'informació majors que a memòria. Això resulta especialment important en aplicacions de Tractament del Llenguatge Natural, ja que la mida dels textos i/o el seu nombre pot ser enorme. Per altra banda,

disposar de resultats intermitjos a disc permet seguir el tractament de les dades i examinar la feina de cada component, mentre que si es treballa només en memòria cal anar fent volcats d'aquesta cada cert temps.

- El manteniment es fa més senzill, ja que és fàcil fer proves individuals i debugar cada component.
- El fet que la comunicació entre Clients i Servidors passi pel MetaServidor fa que tant uns com altres puguin escriure's de forma independent que la plataforma funcioni sobre una sola màquina o sobre un *cluster*. L'únic que canviaria seria el MetaServidor.

3.1.4 Sistemes amb Arquitectures Similars

Aquest tipus d'Arquitectura en què hi ha un conjunt de servidors que atenen peticions de clients de forma distribuïda és un model que ha entrat amb força al món de la informàtica, especialment arran de l'eclosió d'Internet i del revulsiu que ha suposat per a les tecnologies distribuïdes.

Algunes plataformes que permeten desenvolupar sistemes amb arquitectures similars són:

CORBA `http://www.omg.org/corba/`

CORBA és un *middleware** desenvolupat per l'OMG(Object Management Group), organització que agrupa un gran nombre d'empreses de la indústria de la Informàtica, i que permet treballar amb objectes de forma remota: ofereix mecanismes per a fer disponibles i localitzar objectes a altres màquines de la xarxa (com ara un Servei de Noms) i, un cop es tenen aquests objectes, cridar-ne a mètodes. A part, ofereix funcionalitats per a mantenir un repositori de les interfícies que poden tenir aquests objectes, així com un conjunt de serveis habituals que conformen el que anomenen **Object Management Architecture**.

Per a desenvolupar aplicacions sobre **CORBA**, és necessari utilitzar un compilador que, a partir de la interfície que es vol usar, generi els fitxers de comunicació (*Stubs** i *Skeletons**) en el llenguatge de programació desitjat (**CORBA** inclou *mappings* a un gran nombre de llenguatges, tant Orientats a Objectes com no: **C++**, **C**, **Java**, **SmallTalk**, **Eiffel**, **Perl**...), i incloure aquest codi en els Clients i els Servidors.

Cada objecte té un identificador (l'IOR), i a cada màquina del sistema hi ha un ORB(*Object Request Broker*) que s'encarrega d'encaminar les crides a mètodes cap als objectes de la seva màquina que s'amaguen rere els identificadors.

Jini `http://www.jini.org/`

Jini és una alternativa a **CORBA** pensada per a **Java**, aprofitant les característiques que aquest llenguatge té. En aquest cas, tenim un servei de *lookup*, on els proveïdors de serveis es registren, i a on els clients van a buscar-los. La particularitat de Jini és que el codi del servei es diposita al servei de *lookup* en registrar-se (en forma de fitxer **.class**) i els clients el baixen i l'executen localment quan demanen el Servei.

Aquest funcionament és possible (i mínimament segur) per les característiques pròpies de la Java Virtual Machine, com ara la càrrega de classes en temps d'execució, la portabilitat del codi entre diverses plataformes i la possibilitat d'executar-lo en una *sandbox**.

Per a no haver de fer circular per la xarxa tot el codi del servei, la classe que es baixen els clients acostuma a ser un *Proxy*, que es comunica amb el proveïdor del servei, que és qui realment du a terme la feina.

Així com **CORBA** manté un enfocament més proper a la programació Orientada a Objectes tradicional, **Jini** centra més l'enfocament en el concepte de Servei i de proveïdors i clients d'aquests Serveis.

Open Agent Architecture `http://www.ai.sri.com/oaa/`

L'**Open Agent Architecture** és un projecte de l'Stanford Research Institute que proporciona una arquitectura per a treball amb Sistemes Basats en Agents. Cada Agent és un procés independent especialitzat en dur a terme algun servei, i tots treballen de forma coordinada, fent peticions quan necessiten algun servei i sinó esperant per a poder processar-ne.

Com que l'**Open Agent Architecture** està pensada per a la comunitat de recerca en Intel·ligència Artificial, el sistema de comunicació entre Agents és usant l'**Interagent Comunication Language**, basat en el llenguatge **Prolog**. D'aquesta manera, els serveis oferts i les peticions fetes pels Agents s'expressen en forma de predicats **Prolog**.

L'encarregat de fer arribar les peticions de serveis a l'Agent (o Agents capaços d'atendre-les) és un Agent especialitzat anomenat *Facilitator*, que addicionalment és capaç d'emmagatzemar dades compartides pels Agents. Els Agents registren els seus serveis al *Facilitator*, i quan algun altre Agent fa una petició, el *Facilitator* fa arribar a l'Agent capaç de resoldre-la un event amb la informació d'aquesta, i finalment retorna la resposta al primer.

L'**Open Agent Architecture** té particular interès per a nosaltres ja que és l'arquitectura triada per al projecte CHIL com a suport dels serveis desenvolupats per aquest. Encara que el sistema desenvolupat en segueixi una altra, és bo recordar l'arquitectura de l'**OAA** per a facilitar un futur transport cap a ella (capítol 5).

3.2 Representació d'Informació de Tipus Lingüístic

En un sistema de CR, com en qualsevol altre tipus d'aplicació de PLN, la major part de la informació amb què es treballa és de tipus lingüístic (encara que també ens podem trobar amb aplicacions de tipus purament estadístic).

De cara a la integració del sistema, es va pensar en fer un disseny que inclogués les classes corresponents als objectes típics del Llenguatge Natural: Tokens, Chunks, Oracions, Passatges... D'aquesta manera, tots els components del sistema podrien compartir la mateixa visió del Domini, i augmentaria el grau d'integració. Per altra banda, nous components podrien aprofitar totes aquestes classes i no caldria construir per a cadascun d'ells una nova representació.

De la mateixa manera, es va pensar en unificar el format dels fitxers emprats al llarg de la cadena. Es volia un format flexible i que permetés reunir en un sol fitxer la informació que es trobava dispersa en diversos i amb diferents formats. Per altra banda, si s'enriquia el format, s'hi podien afegir funcionalitats de què en aquell moment no es disposava.

Finalment, per raons de temps i per simplicitat a l'hora d'aprofitar el codi s'ha mantingut la dispersió en diversos fitxers, amb els formats ja existents; i s'ha optat per mantenir també les representacions internes de cada mòdul. Tanmateix, el format dels fitxers i la llibreria de classes es troben dissenyades, n'hi ha una implementació en **C++** i romanen a disposició de qui els vulgui usar per a noves components que puguin crear-se o per a altres projectes.

Per aquesta raó, a l'Apèndix B es fa una breu exposició del seu procés de desenvolupament. Per altra banda, a l'Apèndix C es reprodueix una Guia que es va escriure per a exposar el disseny de les classes i el funcionament de la llibreria en **C++**.

Comentar també que al centre TALP hi ha una altra eina pensada com a integració d'Analitzadors Lingüístics i una jerarquia de classes per a representar conceptes de Llenguatge Natural: el Sistema **FreeLing** ([CC04]). **FreeLing** ha acabt essent una altra de les eines integrades en aquest projecte.

3.2.1 Format dels Fitxers

En la implementació existent s'utilitzava un conjunt de fitxers de text pla:

- Una part d'ells contenia informació individual sobre cada token dels passatges. Cada filera del fitxer corresponia a un token, i cada columna contenia una certa informació, estant les columnes separades per espais en blanc.

- Una altra part contenia informació sobre cada oració dels passatges, emmagatzemada en forma de predicats **Prolog**.
- Per últim, alguns fitxers contenien informació molt concreta, com ara els Chunks que apareixien als passatges, i seguien el seu propi format.

Exemples del contingut dels dos tipus de fitxers poden ser els que apareixen a les figures 3.1, 3.2 i 3.3, respectivament.

```
1_1 This this DT
1_2 is be BE
1_3 the the DT
1_4 end end NN
1_5 . . Fp
...
```

Figura 3.1: pass1.lem

```
:- multifile sent/3.
:- dynamic sent/3.

sent(1,1,[(word('This'),lemma('this'),pos('DT')),
          (word('is'),lemma('be'),pos('BE')),
          (word('the'),lemma('the'),pos('DT')),
          (word('end'),lemma('end'),pos('NN')),
          (word('.'),lemma('.'),pos('Fp'))]).
...
```

Figura 3.2: pass1.sent

```
nuc:1|
nuc:2|
nuc:4|mod:3|
...
```

Figura 3.3: pass1.chunks

Com hem comentat, de cara a la integració es volia canviar aquest format pel descrit a la secció B.3. Tanmateix, hi ha una sèrie d'avantatges en la forma de treballar existent:

- El format de un token per filera amb els seus atributs separats per espais s'usa en moltes aplicacions de Llenguatge Natural: Taggers, Parsers, Reconeixedors de Named Entities, Etiquetadors Genèrics... D'aquesta manera, el treball amb aplicacions externes evita la necessitat de construir *Wrappers** per a canviar els formats, tant d'entrada des del nostre format, com de sortida de retorn envers ell. Aquests *Wrappers* suposen, a més a més, un *overhead** en temps d'execució que així s'evita.
- Els fitxers en format **Prolog** resulten particularment útils per a ésser importats directament des de codi escrit en aquest llenguatge. Les rutines d'importació de codi en **Prolog** es troben implementades directament a l'intèrpret, mentre que si es vol codificar la importació de dades en un altre format, cal treballar amb l'Entrada/Sortida en **Prolog**, i com a resultat, a part d'una major complexitat de desenvolupament, es veu penalitzada l'eficiència.

- La conversió del codi existent resulta més senzilla, ja que no cal retocar la part d'Entrada/Sortida de cada mòdul.

Aquests factors van fer desestimar en un primer terme la idea del canvi de format, que ha acabat com una eventual tasca futura, previ estudi de la seva viabilitat i interès (veure capítol 5)
.

Així doncs, en la Implementació que s'ha fet del nou sistema s'ha mantingut la distribució i format dels fitxers tal com estava a la implementació original, si bé en alguns punts s'ha aprofitat que es feia una revisió de tot el codi per a combinar algunes etapes i reduir el nombre de fitxers intermitjos generats.

3.3 El MetaServidor

El punt central de l'Arquitectura proposada és el servidor específic que posa en contacte els Clients i els Servidors que proveeixen els serveis lingüístics i que nosaltres hem anomenat *MetaServidor*. L'estructura del MetaServidor és la que apareix a la figura 3.4.

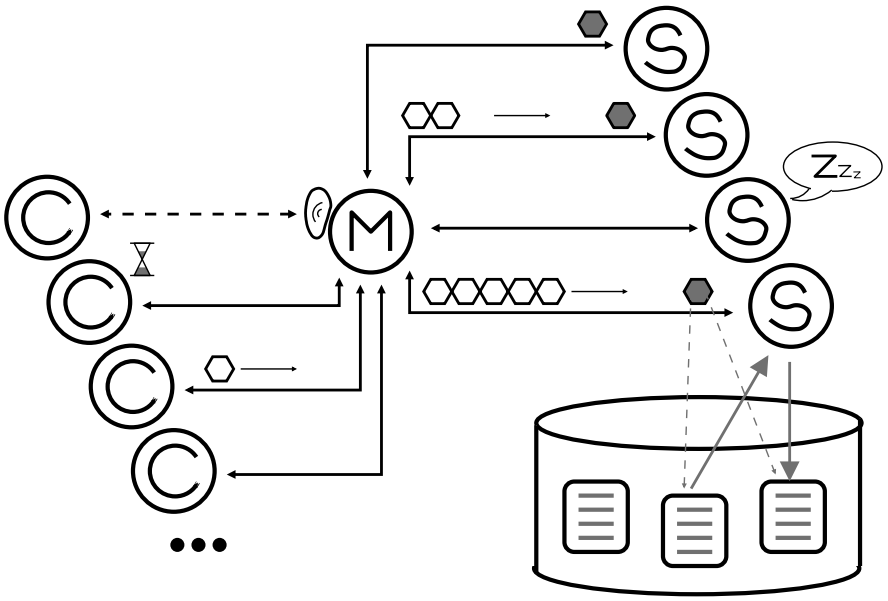


Figura 3.4: Estructura del MetaServidor

El Metaservidor accepta connexions dels Clients a través d'un *socket** TCP. Quan aquests es connecten el primer que han de fer si volen realitzar alguna tasca és localitzar i reservar un servidor d'acord amb les seves necessitats. La determinació del servidor adequat es fa seguint un sistema de Regles (secció 3.3.2). Si ja existeix un servidor en marxa amb les característiques desitjades, es registra que un nou Client vol usar-lo i es retorna al Client el seu identificador. En cas contrari, el MetaServidor sap com posar en marxa un nou Servidor, capturant la seva Entrada i Sortida estàndard amb una *pipe*, i un cop el Servidor està llest per a atendre peticions li ho comunica al Client.

Un cop el Client té accés a un Servidor (diem que està registrat al Servidor, o el Servidor el té registrat), pot demanar-li una petició, sempre a través del MetaServidor (la comunicació entre Clients i Servidors sempre passa pel MetaServidor). Les peticions rebudes per a cada Servidor s'acumulen en una cua, i el MetaServidor les hi va donant perquè les atengui per ordre d'arribada. Quan acaba de processar-ne una, el MetaServidor informa el Client que estava esperant per ella i dóna al Servidor la següent Petició de la cua (si no està buida).

Quan el Client ja no necessita més un Servidor, pot alliberar-lo. Si no hi ha més Clients registrats al Servidor, aquest pot aturar-se. Altrament, o bé segueix atenent Peticions o bé es queda en estat d'espera fins que li n'arribi alguna.

Tota la comunicació, tant entre Clients i MetaServidor com entre MetaServidor i Servidors, segueix el protocol exposat a la secció següent (3.3.1). Tanmateix, abans d'entrar en detalls val la pena analitzar alguns aspectes d'aquesta Arquitectura:

- **El MetaServidor és el Coll d'Ampolla.**

  Com que tota la Comunicació del sistema passa a través del MetaServidor, podem pensar que el MetaServidor posa un límit al rendiment del Sistema i de les Aplicacions que usin la infrastructura que aquest proporciona. I efectivament és així.

  Tanmateix, per la naturalesa de les aplicacions de PLN, i especialment dels sistemes de CR, el temps de comunicació representa una petita fracció del temps de procés total. En la major part dels casos, cal dur a terme processos sobre volums importants de text (com ara tots els passatges recuperats en la segona etapa de la CR) o bé processos més aviat costosos (com els d'extracció de la resposta, o d'anotació semàntica de textos). Per aquesta raó, en una situació de càrrega del sistema, aquesta càrrega es concentrarà en la part de procés, és a dir, en els Servidors, mentre que el MetaServidor només haurà d'intervenir de tant en tant, quan els processos acabin, per a informar el Client, rebre les seves Peticions i fer-les arribar als Servidors.

  Només per al cas de molts Clients fent peticions molt breus simultàniament les esperes pel MetaServidor poden passar a significar una part important del temps d'atenció de les Peticions, i degradar el rendiment del sistema. Tanmateix, aquestes situacions són poc probables tenint en compte les aplicacions per a què s'ha pensat l'Arquitectura, i en el cas que les Peticions més habituals siguin d'aquesta indole potser sigui més convenient buscar un altre tipus de solució.

  Evidentment, si augmentem el nombre d'aplicacions corrent simultàniament sobre el mateix MetaServidor les possibilitats de congestió augmenten. Tot i això, en aquest cas també entraria en joc la càrrega extra que rebria la màquina, que ja de per sí degradaria el rendiment, i la influència del MetaServidor seria menor.

  Per a una solució més escalable, cal considerar el funcionament de diversos MetaServidor sobre un *cluster*.

- **Les Cues de Peticions no tenen perquè ser FIFO**

  Tot i que en la implementació actual les Peticions s'atenen en funció del seu ordre d'arribada, es poden implementar cues de prioritat seguint algun tipus d'algorisme de *scheduling**. Tant es pot optar per atendre en primer lloc les Peticions que a priori han de requerir menys temps (*Shortest Job First*), com per establir un ordre de prioritats entre els diferents Clients i atendre primer la Petició del Client més prioritari. Aquestes prioritats poden ser estàtiques o bé dinàmiques, en funció de paràmetres com ara el temps que porta el Client en el sistema, la durada mitjana de les seves peticions anteriors...

  El tema ofereix un ampli ventall de possibilitats, i està relacionat amb els algorismes de *scheduling* en Sistemes Operatius. Es pot consultar la bibliografia del tema per a més informació (un llibre de nivell bàsic pot ser per exemple [TW97]).

- **No té perquè haver-hi un únic Servidor de cada tipus**

  En el cas que les cues creixin molt de mida, es pot pensar en afegir un nou Servidor del mateix tipus per anar atenent les Peticions en paral·lel. Segons les característiques del Servidor (freqüència i distribució dels bloquejos per Entrada/Sortida en relació amb les etapes de treball al processador, bàsicament) el fet de tenir dos processos iguals atenent Peticions simultàniament pot millorar el *throughput** total.

  Aquest Servidor addicional pot romandre en espera un cop el nombre de Peticions a la cua disminueixi o bé aturar-se quan ja no és més necessari.

- **El Servidor no té perquè aturar-se quan cap Client no el necessita**

  Relacionada amb el darrer paràgraf de l'aspecte anterior es troba la decisió més general de quan aturar un Servidor. En la Implementació actual, un Servidor s'atura immediatament en el moment en què només li queda un Client registrat i aquest es desconnecta o l'allibera. Tanmateix, si el Sistema no està molt carregat pot resultar interessant mantenir el Servidor en espera per si es donés el cas que un altre Client el necessita al cap de poc temps i d'aquesta manera s'estalvia el temps d'inicialització. Els Servidors podrien aturar-se realment quan hagués passat un *timeout** sense que cap Client els hagi demanat. O bé aquesta aturada podria fer-se quan els recursos del Sistema baixessin per sota d'un cert llindar, començant per exemple per aquells que portessin més temps inactius.

Aquests tres darrers aspectes es troben al llindar entre el disseny d'aplicacions i el de Sistemes Operatius, ja que es poden aprofitar resultats i tècniques provinents dels segons per a millorar el rendiment de les primeres. L'estudi de totes les possibilitats aquí esbossades pot proporcionar beneficis interessants, i s'hi deixa la porta oberta com a possible treball futur sobre el projecte (capítol 5).

3.3.1 El Protocol de Comunicació

El Protocol de comunicació i el format de les Peticions que circulen entre Clients i MetaServidor i que aquest redirigeix cap als Servidors esta inspirat en el protocol HTTP ([HT99]).

El tipus de contingut que s'envia en la comunicació és Text Pla, separant les línies pels separadors de línia estàndard de UNIX (Caràcter 10 ASCII). S'ha descartat usar el separador de línia Windows, l'usat habitualment en els Protocols de Xarxa (Caràcters 13 i 10 consecutius), ja que la plataforma d'implementació és el Sistema Operatiu **Linux** (com hem vist en els Objectius del capítol 2).

Quan el Client desitja fer una Petició al MetaServidor, comença per enviar una línia de capçalera que conté informació sobre el tipus de Petició que es vol iniciar i el Servidor a què es vol implicar. Actualment, se suporten tres tipus de Peticions:

REQ Requesta d'un Servidor. És el tipus de Petició que fa el Client per a reservar un Servidor per a una certa tasca. La línia de capçalera té la forma

- `REQ`.

PROC Procés en un Servidor. És la Petició que fa un Client un cop té accés a un Servidor i vol que aquest realitzi algun procés sobre un fitxer. La capçalera té la forma

- `PROC <n>`

on `<n>` és l'id del Servidor que hem obtingut en fer la Requesta.

REL Alliberament d'un Servidor. És la petició que fa un Client quan no necessita més un cert Servidor i vol alliberar-lo. El Servidor quedarà desregistrat de la llista del Client i recíprocament, i si no hi ha més Clients amb aquest Servidor registrat, s'aturarà. La Capçalera té la forma

- `REL <n>`

amb `<n>` de nou l'id del Servidor obtingut en fer la Requesta.

A continuació de la Capçalera, per a les peticions `REQ` i `PROC`, poden venir els paràmetres de la Petició. Aquests venen cadascun en una línia, i estan formats per un nom i un valor, separats per dos punts(`:`). Si hi ha diversos dos punts en la línia, s'agafen com a separadors els de més a l'esquerra. Per altra banda, els espais en blanc inicials o finals, i els que hi hagi a banda i banda del separador s'ignoren.

L'ordre dels paràmetres no importa, però en el cas que hi hagi més d'un valor per a un mateix nom preval el darrer.

La fi dels paràmetres ve indicada per una línia en blanc. Quan el MetaServidor rep aquesta línia en blanc comença a processar la Petició. En el cas de la Petició `REL`, que no necessita paràmetres, també és necessària la línia en blanc.

Les respostes del Servidor poden ser:

1. Si el MetaServidor rep alguna Petició la capçalera de la qual no segueixi un d'aquests tres formats respon informant d'un error de Sintaxi (Retorna una línia formada per la primera paraula de la Capçalera seguida de `KO SYNERR`).

2. Per al tipus de Petició `REQ`, les respostes possibles són:

   - `REQ KO NOSERV`
     Si no s'ha trobat cap Servidor que pugui satisfer les demandes de la Petició.
   - `REQ KO <error>`
     Si s'ha localitzat un Servidor però hi ha hagut un error en iniciar el procés fill (reportat al camp `<error>`).
   - `REQ OK <id>`
     Si s'ha trobat un Servidor (nou o ja iniciat) que satisfeia les demandes de la Petició. El seu id és l'indicat a `<id>`.

3. Per a les Peticions `PROC <id>`, podem obtenir:

   - `PROC KO SYNERR`
     La capçalera no contenia, després del `PROC`, un `<id>` de Servidor vàlid.
   - `PROC KO NOSERV`
     No s'ha trobat el Servidor amb id `<id>`.
   - `PROC KO NOACCESS`
     No es té accés al Servidor amb id `<id>`. Aquest error apareix quan s'intenta demanar un Procés a un Servidor que no té registrat aquest Client.
   - `PROC WAIT`
     Quan el MetaServidor encua la Petició a la cua del Servidor corresponent, informa al Client amb aquesta línia. Tanmateix, la Comunicació no acaba aquí.
   - `PROC KO PARAM <nom>`
     El Servidor ha trobat un error en processar el paràmetre `<nom>`. Pot ser que es tracti d'un paràmetre obligatori que no hi és o bé d'algun que segueixi un format incorrecte.
   - `PROC START`
     Quan el Servidor ha processat tots els Paràmetres i inicia el procés, envia aquesta notificació al MetaServidor, que la reenvia al Client. La Comunicació segueix.
   - `PROC KO <error>`
     Hi ha hagut un error durant el procés, i el Servidor ha aturat el processat de la Petició. La cadena `<error>` informa del tipus d'error.
   - `PROC OK`
     El Procés s'ha dut a terme amb èxit.

   La resposta a una Petició `PROC` pot contenir més d'una capçalera (en el cas que tot sigui correcte es rebran les 3 línies `PROC WAIT`, `PROC START` i `PROC OK`). Per altra banda, després del `PROC OK` es poden rebre paràmetres de sortida, amb el mateix format nom, dos punts, valor dels d'entrada. La fi de la resposta a una Petició `PROC` ve marcada per una línia en blanc.

4. Per últim, per a les Peticions `REL <id>`, les respostes poden ser:

- `REL KO SYNERR`
  La capçalera no contenia, després del `REL`, un `<id>` de Servidor vàlid.
- `REL KO NOSERV`
  No s'ha trobat el Servidor amb id `<id>`.
- `REL KO NOACCESS`
  No es té accés al Servidor amb id `<id>`. Aquest error apareix quan s'intenta alliberar un Servidor que no té registrat aquest Client.
- `REL OK`
  El Servidor ha estat alliberat.

Un diàleg típic entre Client i MetaServidor pot ser el mostrat a la figura 3.5. En ell, el Client fa una Petició per a localitzar un Tagger per a l'Anglès i li demana que Taggegi un Text que es troba al fitxer `/tmp/fitxer.tok` i que deixi el resultat a `/tmp/fitxer.tag`. Per últim, allibera el Tagger si ja no l'ha de fer servir més.

```
C: REQ
C: TIPUS: Tagger
C: IDIOMA: en
C:
M: REQ OK 1
C: PROC 1
C: INPUT: /tmp/fitxer.tok
C: OUTPUT: /tmp/fitxer.tag
C:
M: PROC WAIT
M: PROC START
M: PROC OK
M:
C: REL 1
C:
M: REL OK
...
```

Figura 3.5: Diàleg entre Client i MetaServidor

El protocol entre MetaServidor i Servidors, és bàsicament el mateix, ja que el MetaServidor el que fa és reenviar les Peticions rebudes dels Clients cap als Servidors que cal, i les Respostes donades per aquests cap al Client interessat.

En concret, quan un Servidor s'inicia la primera resposta que dóna és:

- `REQ KO <error>`
  Si en iniciar-se ha tingut algun problema i ha estat impossible de fer-ho.
- `REQ OK`
  Si, per contra, s'ha pogut iniciar sense problemes.

Com pot veure's, aquesta és la mateixa resposta que rep el Client (el que només tenint en compte els casos que no han estat ja respostos pel MetaServidor, com ara els que no troben quin Servidor iniciar).

A partir d'aquest moment, el Servidor queda a l'espera de Peticions dels clients. Quan el MetaServidor rep una Petició per a ell, la reenvia amb la mateixa informació rebuda del Client (no es reenvia exactament el mateix text, sinó la mateixa informació. Per exemple, si s'han donat

dos valors a un mateix paràmetre, només s'envia el que preval). El format de la Capçalera `PROC` i dels paràmetres és el mateix (amb la característica que el `<id>` de la Capçalera de les Peticions que rep un cert Servidor és sempre el mateix, ja que és el seu).

Les respostes del Servidor poden ser les que, comentant el protocol entre Client i MetaServidor, es podien rebre després d'un `PROC WAIT`:

- `PROC KO PARAM <nom>`
- `PROC START`
- `PROC KO <error>`
- `PROC OK`

amb el mateix significat. També, de la mateixa manera, el Servidor informa que ha acabat de processar la Petició amb l'enviament d'una línia en blanc.

Per últim, quan el MetaServidor rep una petició d'alliberament i el Servidor ha de ser aturat, el MetaServidor senzillament tanca la *pipe* que va dirigida cap a l'Entrada estàndard del Servidor i aquest sap que ha d'aturar-se en quant li sigui possible. No hi ha cap altre mena de comunicació, ja que des del moment en què el MetaServidor tanca la *pipe* considera que el Servidor es troba aturat i no li passa més Peticions.

Així, la seqüència de comunicació completa entre el Client, el MetaServidor i el Servidor per al cas anterior, suposant que no hi hagués cap altre Tagger per a l'Anglès en marxa ni cap altre Client interessat en ell seria la de la figura 3.6

```
C -> REQ                      -> M
C -> TIPUS: Tagger            -> M
C -> IDIOMA: en               -> M
C ->                          -> M
                                 M -> Inicia S...
                                 M <- REQ OK                  <- S
C <- REQ OK 1                 <- M
C -> PROC 1                   -> M
C -> INPUT: /tmp/fitxer.tok   -> M
C -> OUTPUT: /tmp/fitxer.tag  -> M
C ->                          -> M
                                 M -> PROC 1                  -> S
                                 M -> INPUT: /tmp/fitxer.tok  -> S
                                 M -> OUTPUT: /tmp/fitxer.tag -> S
                                 M ->                         -> S
C <- PROC WAIT                <- M
                                 M <- PROC START              <- S
C <- PROC START               <- M
                                 M <- PROC OK                 <- S
C <- PROC OK                  <- M
                                 M <-                         <- S
C <-                          <- M
C -> REL 1                    -> M
C ->                          -> M
                                 M -> Tanca la pipe amb S
C <- REL OK                   <- M
                                       S s'atura...
```

Figura 3.6: Diàleg entre Client, MetaServidor i Servidor

3.3.2 El Sistema de Regles

Com hem comentat abans, la determinació del Servidor adequat a cada Requesta dels Clients té lloc mitjançant un sistema de Regles. Cada regla consta d'un seguit de condicions, juntament amb un directori d'execució del Servidor i la línia de comandes amb què s'ha d'iniciar.

Aquestes regles, juntament amb una sèrie de valors que conformen el que anomenem Entorn Global, es llegeixen d'un fitxer de configuració en iniciar el MetaServidor, que per defecte és **metaServer.cfg**.

El mecanisme de localització del Servidor és el de recórrer de forma lineal les regles definides al fitxer de Configuració. Al mateix temps que es comproven les condicions d'aplicació de les regles, es crea una Petició modificada en què els Paràmetres irrellevants per a la regla han estat eliminats, i si algun Paràmetre havia de prendre un valor per defecte el té assignat.

Usant aquesta petició i la regla escollida es determina la línia de comanda a executar i el directori on fer-ho. Com veurem ara de forma més detallada, les regles permeten:

- Comparacions de igualtat, desigualtat, menor, major, menor o igual i major o igual entre valors numèrics o alfanumèrics.
- Valors per defecte.
- Substitució de variables i paràmetres en la línia de comandes.
- Expansió condicional d'expressions.

Format del Fitxer de Configuració

El fitxer de Configuració segueix també un format de Text pla, per a fer-ne fàcil la modificació. Si el primer caràcter d'una línia és un caràcter de sostingut(#), la línia es considera un comentari i és senzillament ignorada. El fitxer consta de seccions, que es delimiten amb una línia que comenci per **@@** i a continuació tingui una cadena identificant el tipus de secció. En la implementació actual es distingeixen dos tipus de seccions.

1. `@@ entorn`

   Ha de ser la primera secció del fitxer, només pot haver-n'hi una i en ella es descriu l'Entorn Global del MetaServidor. La seva primera línia ha de tenir la forma `@@ entorn`, i a continuació venen parelles de nom i valor, de forma similar a les usades al Protocol de comunicació de la secció anterior, però en aquest cas separats per un caràcter igual (=).

   L'Entorn Global defineix un conjunt de variables, de forma similar a l'Entorn que tenen els processos en un Sistema Operatiu. Aquestes variables serveixen per a definir paràmetres, constants, *paths** d'execució... de manera que quan vulguem fer un canvi en algun d'ells no sigui necessari recórrer tot el fitxer canviant-ne totes les aparicions, sinó que hi hagi prou amb canviar el valor de l'Entorn.

   Hi ha dues variables amb significat particular. La primera d'elles és `PORT` i que és consultada pel MetaServidor només inicialitzar-se per a decidir a quin Port iniciar el *socket* d'atenció als Clients. En el cas que no sigui definida, el port per defecte és el 54321[1].

   L'altra és `DEBUG`, que si pren un valor cert fa que el *log* que va generant el MetaServidor durant el seu funcionament sigui més detallat, per a facilitar el *debugging*.

2. `@@ servei <nomServei>`

   Cada secció d'aquest tipus identifica una regla per a posar en marxa un Servidor davant una Requesta dels Clients. El `<nomServei>` que hi ha com a argument és un identificador amb valor merament informatiu. El contingut d'una secció consta de tres parts:

[1] sense raó en especial

(a) En primer lloc, la llista de les condicions que és necessari que s'acompleixin perquè el Servidor es consideri adequat per satisfer una Requesta d'un Client. Aquestes condicions tenen de nou el format de nom i valor, en aquest cas separats per un operador que pot ser:

- Un operador de comparació, d'entre ==, ! =, <, <=, > i >=, amb els seus significats habituals. Perquè una comparació així se satisfaci, és necessari que la Petició del Client tingui el Paràmetre del nom indicat definit (aparegui de forma explícita a la Petició), en cas contrari la condició falla i no es pot aplicar la regla.
- Un operador de valor per defecte (? =). Aquest operador indica que el Paràmetre del nom indicat pot tenir qualsevol valor, però que si no està definit prengui l'aquí indicat.
- Un operador de valor ocult (:). Barreja entre l'operador de valor per defecte i la comparació d'igualtat, actua de la mateixa manera que la segona, però amb la diferència que si el Paràmetre no està definit li assigna el valor per defecte. Així doncs, una condició amb valor ocult pot tenir èxit per dues raons:
  i. El Paràmetre està definit i el seu valor és igual a l'indicat,
  ii. o bé el Paràmetre no està definit. En aquest cas, se li assigna el seu valor indicat.

Les condicions amb un operador de comparació o de valor ocult es consideren **obligatòries**, en el sentit que poden discriminar una Petició, mentre que les que tinguin un operador de valor per defecte, es consideren **optatives**.

Per a facilitar la construcció de les Regles i dels Clients, s'ha pres la convenció de tenir tres paràmetres amb nom i significat concret:

TIPUS Descriu el tipus de Servei que volem, sia Tokenitzador, Tagger, Extractor de la Resposta, Classificador de Preguntes...

IDIOMA Especifica l'idioma per al processador lingüístic. Com a codis d'idioma, es poden usar els codis de 2 lletres de la ISO (**ca** per al Català, **en** per a l'Anglès, **es** per a l'Espanyol....).

PROGR Dona el nom del programa, dins un mateix tipus de Servei. Així, dins els Taggers podem tenir el **FreeLing**, el **TnT**... i la seva activació ve en part determinada pel valor d'aquest paràmetre.

(b) En segon lloc, una línia començada per < que indica el directori on s'executarà el Servidor.

(c) Per últim, una línia començada per > que indica la línia de comandes a executar per a iniciar el Servidor.

En els dos camps de *path* i comanda es poden incloure variables i condicions perquè s'expandeixin en el moment en què la regla s'acompleixi. El format de les cadenes a expandir pot ser:

(a) `\${<nom>}`

Aquesta cadena s'expandirà com el valor de la variable de l'Entorn Global de nom `<nom>` o, si no existeix, el paràmetre de la Petició modificada (la que s'obté com a resultat d'una aplicació correcta de la Regla, i que té els valors per defecte i no té els paràmetres innecessaris) amb aquest nom.

(b) `?{<nom> <operador> <valor>}{<expressió>}`

Aquesta cadena s'expandirà com el resultat d'expandir l'expressió `<expressió>` (que, per tant, pot contenir altres variables i condicions a expandir) si el valor de la variable de l'Entorn Global o, si no existeix, el paràmetre de la Petició modificada de nom `<nom>` té el valor en relació al valor `<valor>` d'acord amb l'operador especificat.

Aquí és important l'el·lecció de l'ordre de cerca del nom `<nom>`: primer l'Entorn Global i després la Petició modificada. D'aquesta manera, s'evita que una Petició pugui canviar variables globals, com ara els *paths* d'execució de les comandes, tant per error com amb intencions malicioses.

De fet, el tema de la Seguretat d'aquest sistema ha estat poc estudiat, ja que ara per ara un client malvat podria introduir com a paràmetres que sap que després s'expandiran en la línia de comandes coses com `‘rm *‘`, que si s'executen des del *shell* poden tenir efectes destructius. Tot i que el sistema ha d'executar-se en un entorn cooperatiu, en el sentit que no ha d'estar obert al públic general sinó a un conjunt de persones conegut i controlat, no estaria de més detectar valors dels paràmetres que poden tenir males conseqüències i, o bé escapar els caràcters especials, o bé denegar les Peticions que intentin atemptar contra la integritat del sistema. Tot i així, com que els punts on s'executa directament codi rebut dels Clients són pocs, ha de ser fàcil localitzar-los i protegir-los adequadament. Ho deixem com a treball futur, però a curt termini.

Per altra banda, comentar que com que en el moment d'iniciar el Servidor intervenen Paràmetres que figuren a la Petició del Client, pot ser que tinguem un Servidor engegat mitjançant una determinada Regla, i que arribi una nova Petició que també entri per la mateixa Regla però que en canvi tingui Paràmetres diferents. En aquests casos, es considera que la Regla ha fet *matching* però és igualment necessari engegar un nou Servidor. Per aquesta raó, els paràmetres de determinació del Servidor han de ser només els que s'hagin de determinar en temps d'inicialització. Si en considerem masses, evitem que es puguin reaprofitar Servidors i eventualment podem augmentar el nombre de Servidors en espera a mantenir pel Sistema (tornarem a parlar d'aquest aspecte a 3.4.2).

Un exemple de fitxer de configuració, força simple, és el que apareix a la figura 3.7. En ell es defineixen dos Servidors del mateix tipus, però amb característiques diferents, i els directoris i comandes amb què executar-los[2].

Localització del Servidor

L'algorisme seguit per a localitzar el Servidor davant una Petició de tipus `REQ` és:

1. Situar-nos a la primera regla.
2. Comprovar que les condicions **obligatòries** s'acompleixin totes.
3. Si no s'acompleixen i queden més regles, ens situem a la següent i tornem al pas 2.
4. Si no s'acompleixen però no queden més regles, donem un error `REQ KO NOSERV` al Client i sortim de l'algorisme.
5. Si s'acompleixen, recorrem les condicions **opcionals** per a completar la Petició modificada amb els possibles valors per defecte.
6. Recorrem tots els Servidors que estiguin engegats per aquella Regla, comparant els Paràmetres de la Petició modificada que tenim entre mans amb els de la Petició modificada que els va engegar a cadascun.
7. Si tots els Paràmetres són iguals en algun d'ells, retornem al Client l'identificador del Servidor que ha coincidit (amb `REQ OK <id>`) i registrem que té autorització per a accedir-hi.
8. En cas contrari, és necessari iniciar un nou Servidor. Per a fer-ho:
9. Realitzem l'expansió del *path* de la Regla i canviem el directori actual al resultat obtingut.

[2]La línia partida amb el caràcter de barra apareix per a mostrar que es tracta d'una sola línia. La continuació de línies amb aquesta sintaxi no està suportada.

10. Realitzem l'expansió de la comanda de la Regla, i l'executem.
11. Si en iniciar el servidor tenim algun error, n'informem al Client retornant-li `REQ KO <error>` i sortim de l'algorisme.
12. Guardem la informació necessària en un objecte Servidor, registrant-hi el Client; li assignem el següent identificador disponible i el retornem al Client: `REQ OK <id>`.

3.3.3 Disseny

El disseny de classes que s'ha utilitzat per al MetaServidor és el que apareix a la figura 3.8. En ell poden veure's:

- Les classes dels elements que hem estat mencionant: Servidors (**Server**), Clients(**Client**), Regles (**OpcioServer**) i Peticions(**Peticio**).
- Una classe **Pendent** que agrupa una **Peticio** i un **Client** per a la cua de Peticions dels Servidors.
- Un parell de classes auxiliars: **Auxiliars**, que conté mètodes estàtics per ajudar al processat de les Peticions, comprovació de les condicions... i **BufferedReader**, que permet evitar els problemes que comporta la lectura a través de *pipes* i *sockets* i obtenir les línies a llegir una per una, independentment de si hi ha *buffering** o algun altre fenomen que pugui partir les línies enviades d'una banda a l'altre del canal, o bé fer que vàries d'elles es llegeixin juntes en una sola crida al sistema.
- Les associacions que hem comentat: els Servidors activats per una Regla, els Clients registrats pel Servidor, la cua de Peticions pendent, juntament amb algunes altres associacions secundàries.

Aquest disseny és el que s'ha implementat finalment, implementació en **Perl** que es comenta a la secció 4.2.

3.4 Els Servidors

Fins ara, les decisions preses de cara al Disseny del MetaServidor no han tingut en compte la forma de treballar de cara al PLN. El sistema de MetaServidor, Servidors i Clients permetria treballar amb qualsevol altre tipus d'Aplicació (si bé certes decisions s'han fet pensant més en el PLN).

3.4.1 Format Esperat de les Peticions

Per aquesta raó, hem d'acordar una forma de treballar conjunta entre Clients i Servidors, la manera com aquests es comunicaran a través de les Peticions. Com que el lloc on es trobaran les dades havíem decidit que serien fitxers a disc, resulta lògic pensar que les Peticions el que inclouran seran els noms d'aquests fitxers, de manera que en fer una Petició el Client especifiqui el fitxer o fitxers d'on s'han de llegir les dades i el fitxer o fitxers on s'han de deixar els resultats, juntament amb la resta de paràmetres de la Petició.

Per a facilitar l'escriptura de Servidors i Clients, s'han pres uns convenis a l'hora d'especificar els paràmetres de les Peticions de tipus `PROC`, de forma similar a les de les Peticions de tipus `REQ`:

INPUT Correspon al fitxer d'entrada per a casos en què només n'hi hagi un, o al principal si n'hi ha diversos (en els casos en què no es pugui destacar un fitxer d'entrada com a principal, s'ha optat per no mantenir el nom i posar-ne un altre de naturalesa més descriptiva).

OUTPUT Correspon al fitxer de sortida, de forma similar a l'`INPUT` per a l'entrada. També de la mateixa manera, en cas de diversos fitxers de sortida, correspon al principal, si aquest existeix.

TMP Per als serveis que requereixin deixar fitxers temporals, si a aquest paràmetre se li assigna un valor diferent de la cadena buida, l'utilitza com a prefix dels fitxers temporals. Un cop acabat el processat, els temporals es conservaran.

En cas que no s'especifiqui aquest Paràmetre, s'agafarà un temporal del directori temporal del sistema, i en acabar el procés de la Petició s'esborraran o es reaprofitaran per a una altra Petició.

GZIP Si s'opta per conservar els fitxers temporals, es pot especificar un valor cert (1 habitualment) per a indicar que un cop usats es comprimeixin, amb la utilitat **gZip** habitual de **UNIX**. Si no s'especifica el paràmetre, o se li dóna un valor fals (com 0), els fitxers es deixen sense comprimir.

La resta de paràmetres varien segons el tipus de Servei, i els seus noms i significats estan definits de forma específica.

Degut al fet que Clients i Servidors no tenen per què estar executant-se al mateix directori, **sempre** s'utilitzen *paths* absoluts en els paràmetres que contenen fitxers.

3.4.2 Paràmetres d'Iniciació i de Procés

Pot ser que aquí ens hagi sorgit la pregunta sobre perquè hi ha una divisió dels Paràmetres de les Peticions entre els que es proporcionen en la Petició `REQ` i els que es proporcionen en la Petició `PROC`. És a dir, per a parsejar un text en Anglès necessitem 2 Peticions:

1. 1 Petició `REQ` per a reservar un Parser per a l'Anglès.

2. 1 Petició `PROC` per a dir-li a aquest Parser que processi el fitxer.

Per què no podríem agrupar aquestes dues Peticions en una de sola?

1. 1 Petició per a dir que es parsegi un fitxer en Anglès.

La clau està en què dividint els paràmetres en dos grups es permet minimitzar l'impacte dels costos dels *overheads* d'inicialització dels diferents Servidors.

1. Els paràmetres que es demanen en fer la Requesta del Servidor en la Petició `REQ` són aquells un canvi en els quals resulta costós per al Servidor. Així, per exemple, la decisió de l'idioma del Parser pot resultar costosa, ja que caldrà carregar les estructures de dades del Parser per a l'Anglès en memòria. Un canvi d'idioma suposaria tornar a passar pel cost d'inicialització, i per aquesta raó preferim que si es vol parsejar un text en un altre idioma es reservi un nou Servidor.

2. Els paràmetres que es proporcionen en la Petició `PROC` són aquells que no són costosos. Així, un Paràmetre que indiqués el format del fitxer d'Entrada o el desitjat per al fitxer de Sortida no és costós (o no hauria de ser-ho), i té sentit que es pugui especificar de forma individual per a cada Petició sense haver de reservar un nou Servidor. Evidentment, Paràmetres com els fitxers amb les dades a processar a el destí on deixar els resultats està clar que pertanyen a aquesta segona categoria.

Exemples d'aquesta separació poden veure's en els Servidors que s'han implementat finalment per al sistema de CR (secció 4.5).

3.4.3 Esquema de Funcionament dels Servidors

Per a facilitar l'escriptura dels Servidors, es va fer un disseny que permetés factoritzar la part comuna de tots ells, de manera que es puguin implementar aquestes seccions en una llibreria de forma reusable.

Per a permetre-ho, es considera la classe **Wrapper**, de què poden heretar els Servidors que vulguin aprofitar-la com a base. El diagrama de seqüència del funcionament de **Wrapper** és el que apareix a la figura 3.9. La classe **Wrapper** segueix el Patró de disseny Plantilla (veure [GH94]), i defineix tres operacions ganxo, per al comportament específic de cada Servidor:

1. `initObjs()`, que realitza la tasca d'inicialització de l'Objecte Servidor, creant tots els objectes que aquest necessiti. És la primera funció que es crida, abans d'entrar al bucle principal.

2. `processarPeticio(p : Peticio)`, que atén una Petició rebuda d'un Client. És un *callback** que es va cridant des del bucle principal, `startMeUp()`, que va llegint Peticions de l'entrada estàndard i passant-les a `processarPetició(p : Peticio)` fins que el MetaServidor tanca l'entrada.

3. `deinitObjs()`, que permet alliberar de forma correcta els recursos reservats a `initObjs()`.

A part, els objectes **Petició**, incorporen un mètode `obtenirParams(inici : Iterator, final : Iterator)`, per a obtenir de forma senzilla els paràmetres de la petició, permetent definir també valors per defecte. La forma d'usar-la és construint un contenidor amb elements de la classe **Parametre**, que indiquen un nom de Paràmetre i, opcionalment, un valor per defecte. `obtenirParams(inici : Iterator, final : Iterator)` recorre el contenidor, comprova l'existència dels paràmetres obligatoris, i completa els que tenen valor per defecte, deixant els resultats en el mateix objecte **Paràmetre**. En el cas en què falti algun paràmetre, es llança una excepció per a informar el codi del Servidor.

Les classes usades apareixen a la figura 3.10. A l'apartat d'implementació (secció 4.3), es pot veure com a la implementació en **C++** s'ha seguit aquest disseny, mentre que per als servidors en **Perl** s'ha modificat per aprofitar les facilitats de treball amb *hashos** i llistes d'aquest llenguatge.

3.5 Els Clients

Pel que fa als clients, un cop determinada la forma de treball, així com l'estructura de les peticions, el que resta és dissenyar unes classes de manera que el treball amb el MetaServidor i la comunicació amb els Servidors sigui el més semblant possible al treball local. Per a fer-ho, utilitzem el patró representant (descrit també a [GH94]), i construïm dues classes *Proxy*, una per al treball amb el MetaServidor i una altra per al treball amb els Servidors. Aquestes classes són les que apareixen a la figura 3.11.

La classe **ProxyMetaServidor** ofereix les funcions:

- `new(addr : String, port : short)` (constructor) per a connectar-se a un MetaServidor que corri al port especificat de la màquina d'adreça IP `addr`.

- `requestServer(parametres : Hash)` per a fer una Petició de tipus `REQ` al MetaServidor, amb els Paràmetres especificats al *Hash*.

- `newServer(tipus : String, idioma : String, parametres : Hash)`. Degut a que, com havíem comentat anteriorment (secció 3.3.2), en les regles s'usen habitualment els paràmetres `TIPUS` i `IDIOMA` per a determinar el Servidor a activar, resulta útil tenir una manera més compacta de passar aquests dos paràmetres. De fet, l'únic que fa aquest mètode és afegir als paràmetres els dos indicats, i cridar a `requestServer(parametres : Hash)`.

- `requestProc(id : int, parametres : Hash)` per a fer una Petició de tipus `PROC` amb els Paràmetres especificats al *Hash*, i al Servidor d'identificador `id`.

- `requestRelease(id : int)` per a fer una Petició de tipus `REL` del Servidor d'identificador `id`.
- `close()` per a finalitzar la connexió amb el MetaServidor, i alliberar tots els Servidors a què el Client estigués registrat.
- `setWaitHook(codi : Hook)` i `setStartHook(codi : Hook)`. El **ProxyMetaServer** permet definir **Hooks**, és a dir, ganxos per què siguin cridats en moments concrets del procés d'una Petició. En concret, es poden definir dos **Hooks**:
  - El **WaitHook**, cridat quan es rep el `PROC WAIT` del MetaServidor conforme que la Petició ha estat encuada a la cua del Servidor.
  - L'**StartHook**, cridat quan es rep el `PROC START` conforme que el Servidor ha iniciat el processat de la Petició.

  Aquests mètodes permeten especificar el **Hook** a cridar en aquests dos moments[3].
- `sockExplore(cadena : String)`, `getSock()` i `flushSock()`, per a la gestió del Socket de connexió amb el MetaServidor. Es tracta d'operacions internes, que no han de ser utilitzades pels Clients.

Per altra banda, la classe **ProxyServer**, que se'ns retorna en demanar un Servidor al **ProxyMetaServer**, facilita més la demanda de Peticions de tipus `PROC` i `REL`:

- `new(_id : int, _metaServer : ProxyMetaServer)` (constructor), que inicialitza els atributs de l'Objecte. Aquest constructor és el que usa el **ProxyMetaServer** quan retorna un objecte **ProxyServer**, i habitualment no ha de ser cridat directament pel Client.
- `proc(parametres : Hash)` per a fer una Petició de tipus `PROC` amb els paràmetres indicats al *Hash*. De fet, crida a `requestProc(id : int, parametres : Hash)` del **ProxyMetaServer**.
- `procFile(input : String, output : String, parametres : Hash)`. De forma similar al que dèiem per a **newServer(...)**, com que tal com hem definit les Peticions `PROC` els paràmetres `INPUT` i `OUTPUT` tenen significat especial, resulta útil assignar-los valor de forma ràpida. Aquesta funció també crida a `requestProc(id : int, parametres : Hash)` del **ProxyMetaServer**, però afegint els valors de `INPUT` i `OUTPUT` a les Peticions.
- `release()` per a fer una Petició de tipus `REL`. Es transforma en una crida a `requestRelease(id : int)` del **ProxyMetaServer**.

Es pot pensar en crear subclasses de **ProxyServer**, per a crear Proxies per a tipus concrets de Servidors. Tanmateix, per a mantenir la senzillesa, hem optat per usar directament **ProxyServer**, ja que les subclasses podrien oferir bàsicament interfícies més adaptades a cada Servidor (funcions amb el nombre exacte de paràmetres, i amb noms més descriptius, o per accedir als resultats). D'aquestes funcionalitats addicionals se n'encarrega cada client en concret.

[3] A la implementació, per aprofitar les característiques del llenguatge **Perl**, s'ha optat per usar referències a codi, i la classe **Hook** no s'ha utilitzat

```
# Configuracio MetaServer
# Exemple

###############
# Entorn Base #
###############

@@ entorn
PORT        = 45322

# Directoris d'Execucio
PATHMETASERVER = /usr/lib/metaServer
PATHSERVERS    = ${PATHMETASERVER}/servers

# Directoris de Dades
DADES          = /usr/share/metaServer
PATHMODELS     = ${DADES}/models
PATHMODELSCA   = ${DADES}/modelsCa


############
# Serveis #
############

# CatAnalyler

@@ servei CatAnalyzer
TIPUS == Analitzador
PROGR : CatAnalyzer
IDIOMA ?= ca
FIABILITAT < 20
<${PATHSERVERS}/catAnalyzer
>./start ${FIABILITAT} ?{IDIOMA == ca}{${PATHMODELSCA}/model.dat}\
?{IDIOMA != ca}{${PATHMODELS}/model.dat}


# GenAnalyzer

TIPUS == Analitzador
PROGR : GenAnalyzer
IDIOMA == en
<${PATHSERVERS}/genAnalyzer
>./WrapGenAnalyzer.pl ${PATHMODELSGEN}/${IDIOMA}.dat
```

Figura 3.7: Fitxer de configuració del MetaServidor

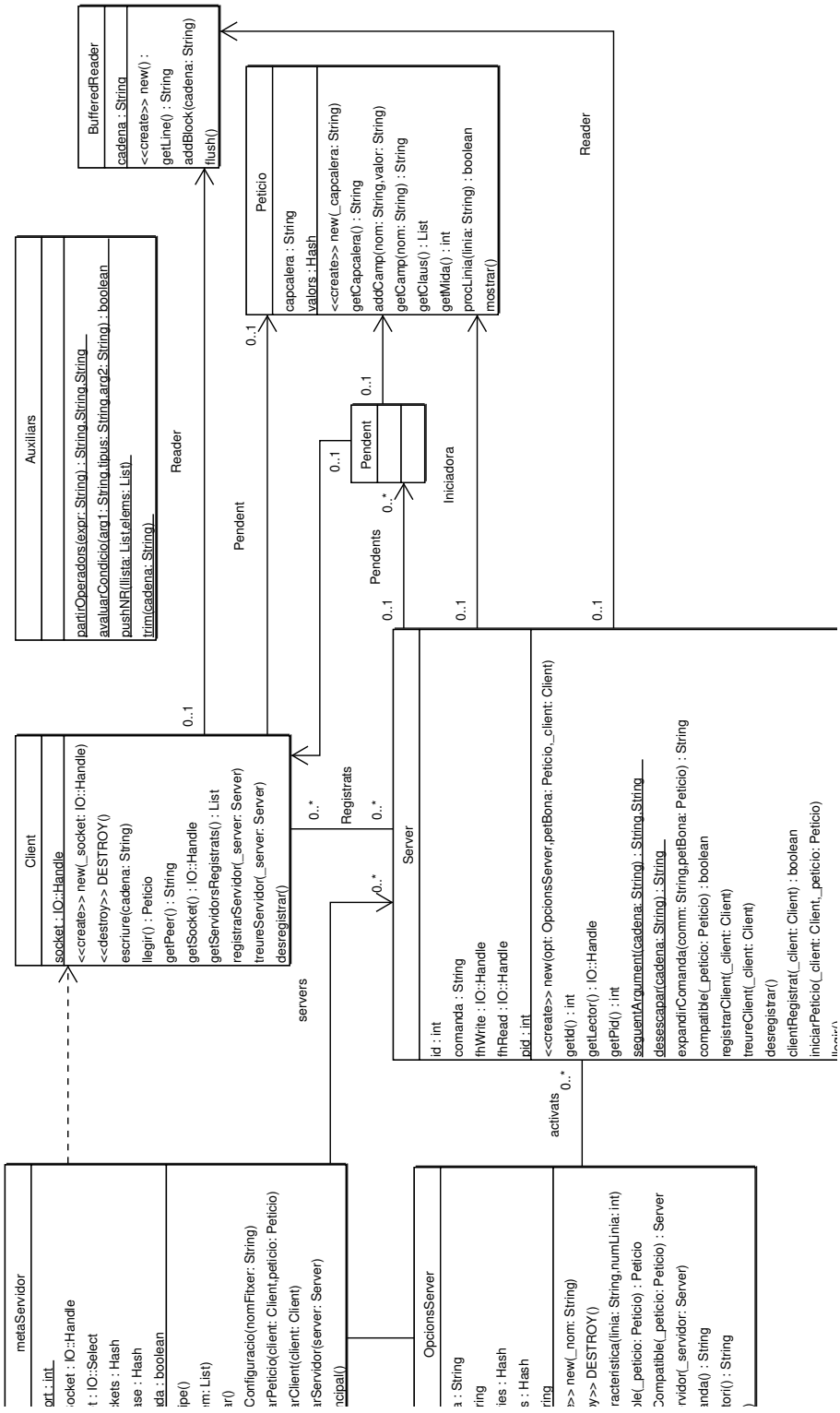


Figura 3.8: Disseny del MetaServidor

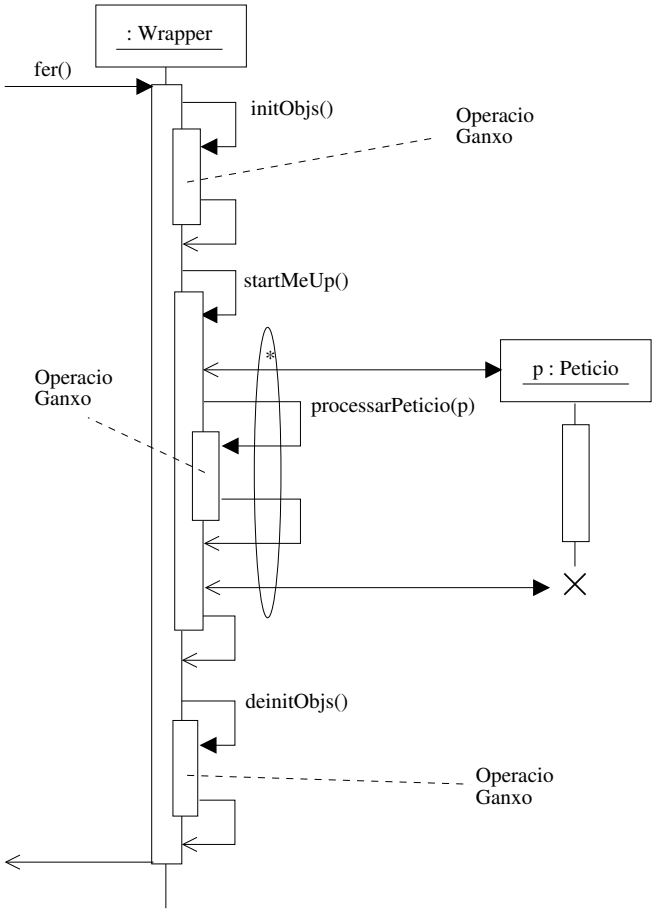


Figura 3.9: Diagrama de Seqüència de Wrapper

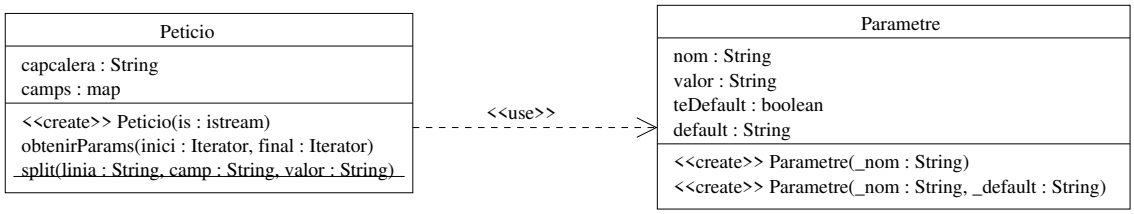


Figura 3.10: Diagrama de Classes per a les Peticions als Servidors

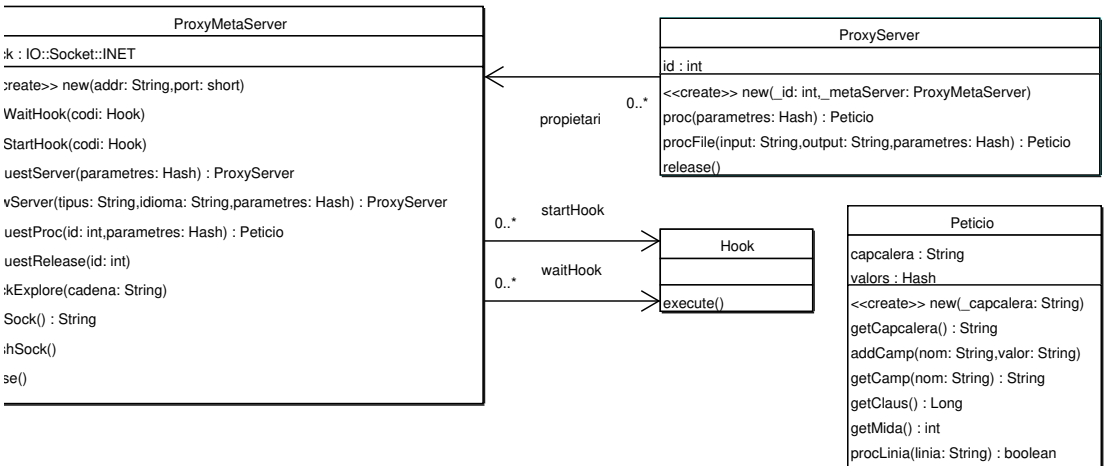


Figura 3.11: Diagrama de Classes per als Proxies

Capítol 4

Implementació

Un cop dissenyada l'arquitectura que havia de servir de suport al sistema de CR, es va començar la implementació dels diversos components, en el següent ordre:

1. MetaServidor
2. Part comuna dels Servidors (**Wrapper**, **Petició**...)
3. Part comuna dels Clients (**ProxyMetaServer**, **ProxyServer**...)
4. Servidors per als diversos processos del sistema de CR.
5. Clients

En aquest capítol descrivim les característiques de la implementació, tant a nivell general com a nivell de cada component.

4.1 Consideracions Generals

4.1.1 Llenguatge de Programació

Una de les primeres el·leccions en iniciar la implementació de qualsevol sistema sol ser la del Llenguatge de Programació a utilitzar.

Pel que fa als Servidors, com que estàvem interessats en aprofitar el codi ja existent, els llenguatges ja ens venien fixats d'avantmà: **C++**, **Perl** i **Prolog**. De manera que de la Part comuna n'hauríem de tenir versions per a cada llenguatge. Tanmateix, degut a les característiques particulars del llenguatge **Prolog** es va optar per construir els Servidors amb codi en **Prolog** com un **Wrapper** en **C++** que rep les peticions i gestiona l'Entrada/Sortida (aspectes més difícils i costosos en **Prolog**) i que a l'hora de processar les peticions crida el codi **Prolog** a través d'una API **C-Prolog**. Per a facilitar aquesta comunicació, es va crear una classe senzilla **PrologPred** que oculta els detalls de l'API i permet cridar un predicat **Prolog** des de **C++** (veure figura 4.1. En concret, aquí apareix el disseny per a l'API amb **SWI-Prolog**, però els canvis a realitzar per a altres versions de **Prolog** no haurien de modificar la interfície).

En els Clients la decisió era similar: no podem restringir-nos a un llenguatge en concret, de manera que les llibreries a usar pels clients havien de ser disponibles en **C++** i **Perl**, com a mínim.

Pel que fa al MetaServidor, aquí l'el·lecció era estrictament nostra, i ens vam decantar pel llenguatge **Perl**, per les següents raons:

- Resulta més ràpid desenvolupar prototipus en **Perl** que no pas en **C++**.
- En ser un llenguatge interpretat, el cicle de desenvolupament és més ràpid: els canvis en el codi poden provar-se immediatament, sense haver d'estar compilant.

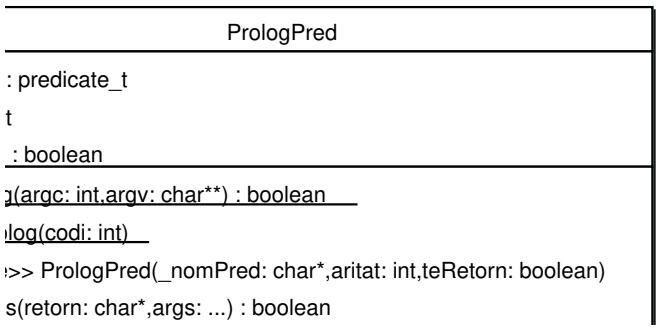


Figura 4.1: Classe PrologPred

- El llenguatge **Perl** sobresurt com a *glue language**, i per tant té molt bones facilitats de comunicació entre processos, així com per a posar en marxa processos fills. Això el fa especialment adequat per a desenvolupar el MetaServidor, una de les tasques del qual serà posar en marxa els Servidors i comunicar-se amb ells per la sortida i entrada estàndard.

Tanmateix, un cop el MetaServidor pugui considerar-se estable, pot resultar interessant passar-lo a **C++**, bàsicament per la millora en quant a eficiència que suposa un llenguatge compilat sobre un d'interpretat (treball futur, capítol 5).

El procés seria semblant al d'altres aplicacions, que comencen amb un prototipus en un llenguatge interpretat, i quan s'estabilitzen es passen a un llenguatge compilat, habitualment **C** o **C++**.

En concret, les distribucions i versions utilitzades van ser les següents:

C++ Compilador gcc de la GNU, versió 3.2
`http://gcc.gnu.org/`

Perl Perl versió 5.8.0
`http://www.perl.com/`

Prolog SWI-Prolog, versió 5.2.13
`http://www.swi-prolog.org/`

4.1.2 Procés de Desenvolupament

No es va voler utilitzar cap IDE (*Integrated Development Environment**) per a la implementació, sinó que es va optar per usar l'editor **GNU Emacs**[1]. Es tracta d'una decisió personal, i basada en què la familiaritat amb l'entorn permet augmentar de forma significativa la velocitat de desenvolupament, a vegades més que no pas les facilitats que aquest entorn ofereixi. Per altra banda, es tracta d'un editor usat per un gran nombre de programadors arreu del món, amb un gran nombre de funcionalitats i a més a més de codi lliure.

De cara al procés de *debugging*, per a **C++** es va utilitzar el **Data Display Debugger (DDD)**, *frontend** per al **GNU Gdb**. Es va disposar de la versió 3.3.1 d'aquest *debugger*.

Per al *debugging* en **Prolog**, es va utilitzar el *debugger* que ve amb la llibreria **XPCE**, distribuïda de forma conjunta amb el **SWI-Prolog**.

La màquina sobre què es va estar treballant la major part del temps va ser un PC equipat amb el Sistema Operatiu **GNU/Linux**. En concret, es tractava de la distribució **Debian** amb un kernel 2.4.24.

[1] http://www.gnu.org/software/emacs/emacs.html

4.1.3 Prova i Detecció d'Errors

Per a intentar minimitzar els *bugs*, el procés que s'ha seguit ha estat el de proves individuals de cada component, paral·lelament a la seva codificació. A mida que s'anaven implementant funcionalitats, aquestes es provaven, en la major part dels casos aprofitant l'arquitectura per a simular situacions concretes:

- Ens podem connectar mitjançant la utilitat **telnet** al MetaServidor per a fer-li Peticions directament, sense utilitzar les classes de costat Client.
- Podem iniciar directament els Servidors i escriure per la seva entrada estàndard les Peticions que desitgem, sense haver de passar pel MetaServidor.
- És fàcil monitoritzar la comunicació entre els diversos components, ja que tota passa pel MetaServidor i es pot fer *logging** de tot el que aquest rep i envia.

A mida que s'anaven desenvolupant els mòduls, també s'han fet proves de comunicació entre ells (MetaServidor - Servidors, Clients - MetaServidor, entre Servidors que hagin d'agafar l'un el resultat de l'altre...).

Per altra banda, com que l'objectiu era fer una reescriptura del sistema de CR ja existent, els resultats parcials obtinguts s'han anat comparant amb els que teníem del sistema ja existent, per a evitar que canviant l'estructura dels processadors lingüístics es canviés també el seu funcionament. Quan s'ha detectat una diferència, s'ha donat credibilitat al resultat de la implementació anterior.

4.2 Implementació del MetaServidor

Com hem comentat, la implementació del MetaServidor va comportar la decisió del Llenguatge usat per a ella. Per la resta, es va seguir el disseny proposat a la secció 3.3.3, sense major complicació. Només la classe **Pendent** es va substituir per un Array amb 2 elements (de fet, la implementació subjacent en **Perl** hauria estat la mateixa), ja que no requeria cap mètode més enllà dels accessors.

També mereix la pena mencionar alguns problemes amb el mòdul **IPC::Open3** de la llibreria estàndard de **Perl**: quan s'intenta obrir un procés fill i, internament dins l'intèrpret, la variant de la crida `exec(..)` utilitzada falla, en comptes de rebre un error, passem a tenir dos intèrprets que s'executen com dos processos independents, ja que la crida `fork()` prèvia sí que s'ha executat. Aquest fet no es troba documentat i va ser la causa d'un bon maldecap.

Aquest problema, juntament els problemes de sincronització que comporta el treball amb processos fills de què s'ha capturat l'entrada i la sortida estàndard van fer que en la implementació de mòduls posteriors intentés evitar aquesta situació en la mesura del possible.

4.3 Implementació de la Part Comuna dels Servidors

La implementació de la classe **Wrapper** no té majors secrets: s'ha fet una traducció bastant directa del disseny a versions en **C++** i **Perl**. L'únic que ha variat ha estat el mètode `obtenirParams(...)` de la classe **Peticio**:

- En **C++** s'ha mantingut el disseny, i cal passar al mètode l'iterador d'inici i de final d'un contenidor (els Servidors implementats utilitzen un **vector<Parametre>** de la STL*) amb els objectes **Parametre**.
- En canvi, en **Perl** el mètode rep una llista (aprofitant les facilitats que el llenguatge ofereix), en què cada element pot ser un escalar (el nom d'un paràmetre) o una referència a un array amb dos escalars (el nom d'un paràmetre i el seu valor per defecte).

  Els valors que prenen els paràmetres es retornen en la llista de retorn, en el mateix ordre en què apareixien als arguments del mètode.

A la figura 4.2 apareixen les dues formes d'obtenir els paràmetres. Com pot veure's, en tots dos casos s'aprofita el mecanisme d'excepcions del llenguatge per a informar dels errors en el procés d'obtenció.

```
// C++
vector<Parametre> vp;
vp.push_back(Parametre("INPUT"));
vp.push_back(Parametre("OUTPUT"));
vp.push_back(Parametre("MODEL", "genmodel.dat"));

...

try {
    peticio.obtenirParams(vp.begin(), vp.end());

    // Valors dels parametres
    string input  = vp[0].valor;
    string output = vp[1].valor;
    string model  = vp[2].valor;

} catch (string excepcio) {
    // Error en els parametres
}

-----------------------------------------------------------------

# Perl

my ($input, $output, $model);

eval {
    ($input, $output, $model) =
         $peticio->obtenirParams('INPUT', 'OUTPUT',
                                 [ 'MODEL', 'genmodel.dat' ]);
};

if ($@) {
    # Error en els parametres
}
```

Figura 4.2: Obtenció de paràmetres

El programa principal dels Servidors només ha de crear l'objecte de la Subclasse de Wrapper corresponent, passant-li els arguments de la línia de comandes com a paràmetre, i cridar sobre ell el mètode `fer()`.

4.4 Implementació de la Part Comuna dels Clients

Les classes per als Clients han estat implementades en **Perl** seguint el disseny presentat. L'únic que s'ha modificat ha estat la classe **Hook**, que no s'ha implementat, ja que el **Perl** permet el pas de referències a codi i s'ha optat per aquesta opció com una més adequada a les característiques del llenguatge.

Pel que fa a la implementació en **C++**, estarà disponible en breu.

4.5 Implementació del Sistema TALP-UdG

Finalment, un cop implementats els components que podríem considerar de base, es va començar la implementació del sistema de CR TALP-UdG. El sistema segueix d'a prop la metodologia exposada a 1.5, amb alguns canvis menors.

El sistema s'ha estructurat dividit en els Servidors que apareixen a la figura 4.3, juntament amb el recorregut que segueixen les dades (que, per analogia amb als component electrònics, podríem anomenar el *data path**). Alguns dels Servidors són en també Clients (res no prohibeix que un Servidor sigui al mateix temps Client) que tenen el seu propi *data path*. La seva estructura apareix a les figures 4.4, 4.5 i 4.6. De fet, tot el procés de CR és un Servidor que respon preguntes sobre una certa Col·lecció.

Aquesta implementació no té caràcter definitiu, sino que ha de servir de base sobre què construir diversos tipus de sistemes de CR. Per altra banda, els paràmetres que reben els diversos Servidors estan subjectes a canvi, segons calgui afegir-ne o eliminar-ne en funció de les necessitats dels sistemes.

Cada Servidor té les seves opcions en temps d'iniciació, i reconeix uns certs Paràmetres en les Peticions. Ara farem un petit recorregut per tots ells, detallant aquests aspectes i comentant algun altre aspecte de la implementació si és meritori de fer-ho.

EngTok

Es tracta d'un tokenitzador per a l'Anglès, adaptat de la utilitat del mateix nom de Dani Ferrés. El servidor tokenitza un fitxer de text, amb unes poques opcions.

Arguments de Línia de Comandes	
no en té	
Paràmetres de la Petició	
INPUT	Fitxer d'Entrada. S'espera un fitxer de text pla.
OUTPUT	Fitxer de Sortida. És un fitxer tokenitzat, en què cada token està en una línia, opcionalment precedit pel seu índex i el de la seva oració.
SPLIT	Booleà (per defecte fals) que indica si al mateix temps que es tokenitza cal partir les oracions. En cas afirmatiu, entre oracions hi haurà una línia en blanc al fitxer de sortida.
NUMBER	Booleà (per defecte fals) que indica si cal donar a cada token el índex de la posició que ocupa al document i, si hi ha partició, l'índex de la oració.
PASSAGES	Booleà (per defecte fals) que indica si l'entrada està formada per diversos passatges, de mode que cal eliminar les marques de separació a la sortida.

Taula 4.1: EngTok

La marca de separació de passatges segueix el format:

```
##############DOCNO:XIE19971013.0101_P1_1.3495_0_2.481329
```

1. `DOCNO:XIE19971013.0101`: Identificador del document de què prové el passatge.
2. `P1`: Número de Passatge.
3. `1.3495`: Puntuació del Document a la Recuperació *Ranked* amb totes les *Keywords*.
4. `0`: Puntuació del Document a la Recuperació amb només *Named Entities*.

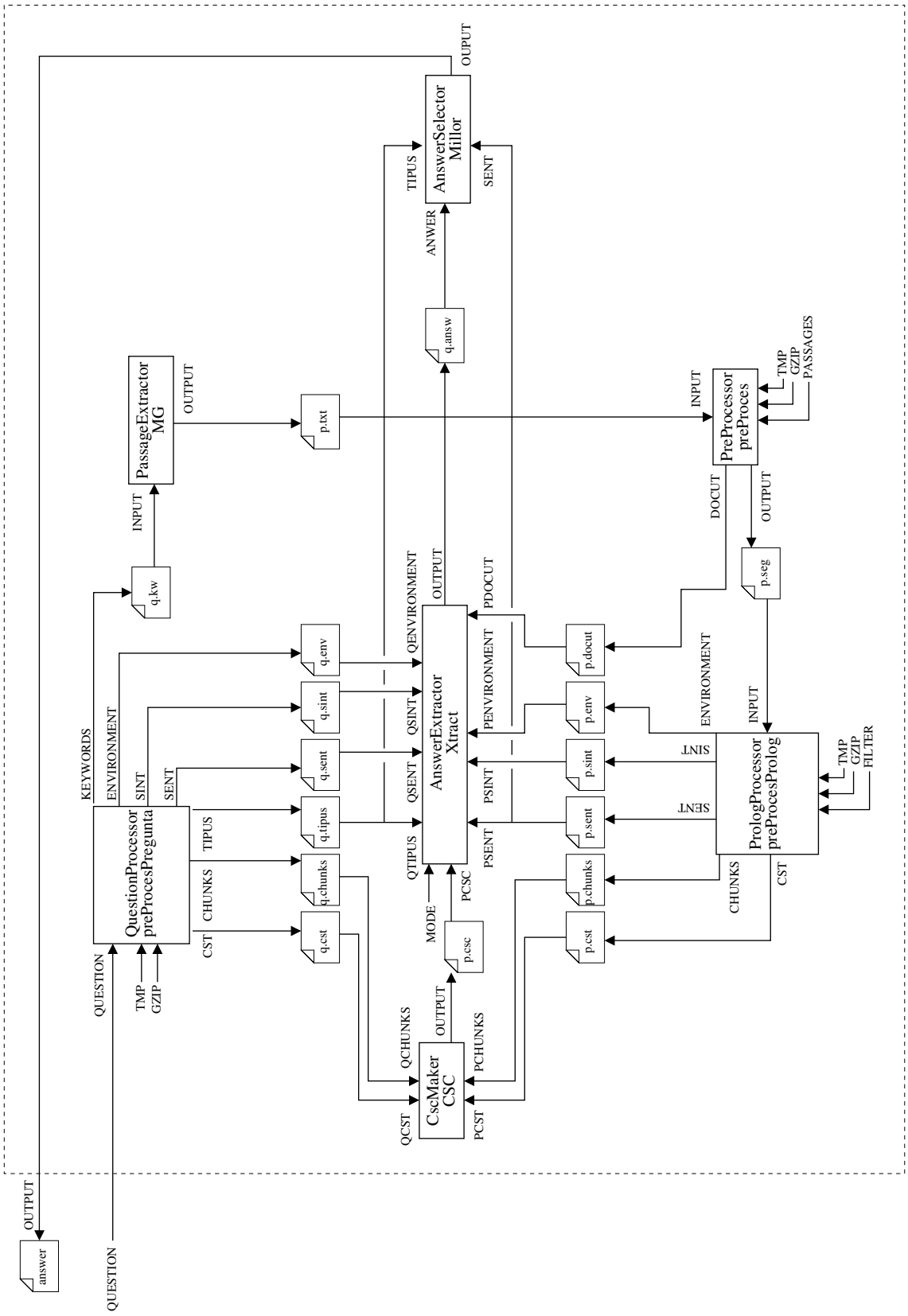


Figura 4.3: Estructura del Servidor de CR

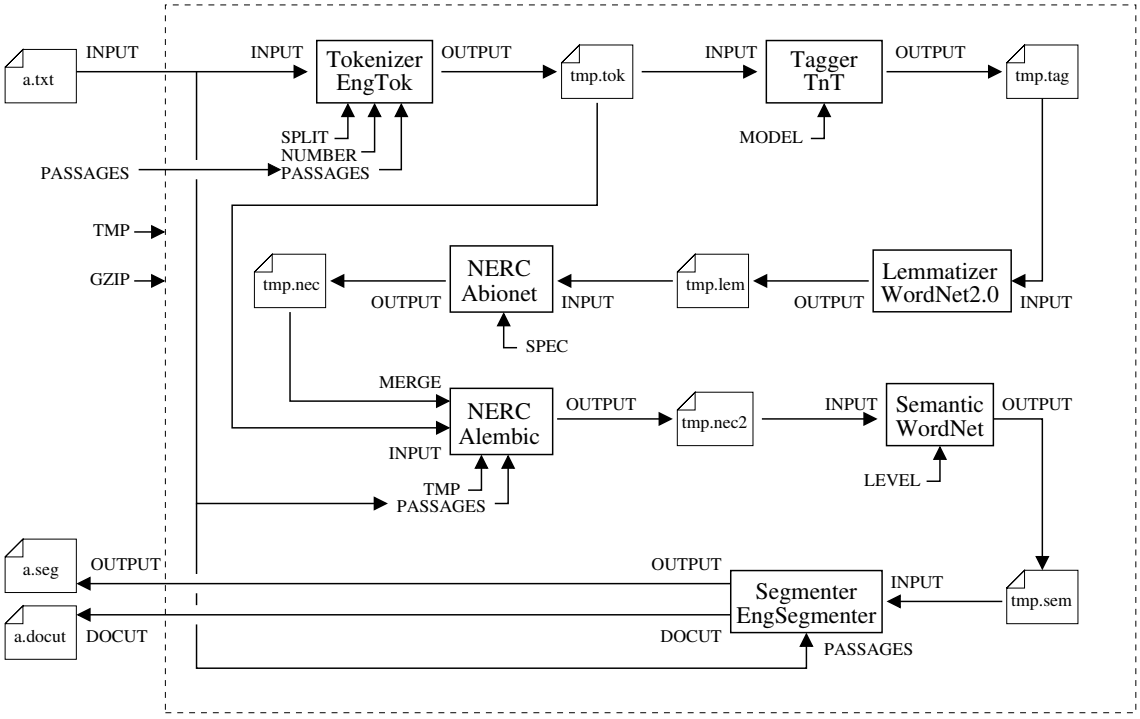


Figura 4.4: Estructura Interna del PreProcessador

5. `2.481329`: Puntuació del Passatge a la Recuperació.

La implementació és en **Perl** pur.

4.5.1 TnT

És un *Wrapper* per al Tagger **TnT**, de Thorsten Brants ([Br00]). Permet especificar el model amb què taggejar el text, així com traduir els Tags del Penn Tagset a altres tipus de Tag.

Arguments de Línia de Comandes	
1	*Path* a l'executable **TnT**.
2	Model a usar per defecte.
3	Directori per defecte dels models.

Paràmetres de la Petició	
INPUT	Fitxer d'Entrada. S'espera un fitxer tokenitzat no numerat.
OUTPUT	Fitxer de Sortida. És un fitxer taggejat: està tokenitzat i a continuació de la forma del Token hi ha el seu Tag.
MODEL	*Path* d'un model a utilitzar en comptes del per defecte. Si és un *path* relatiu, es considera que és relatiu al directori per defecte dels models

Taula 4.2: TnT

La implementació és en **Perl**, però és crea un procés fill **TnT** a cada Petició.

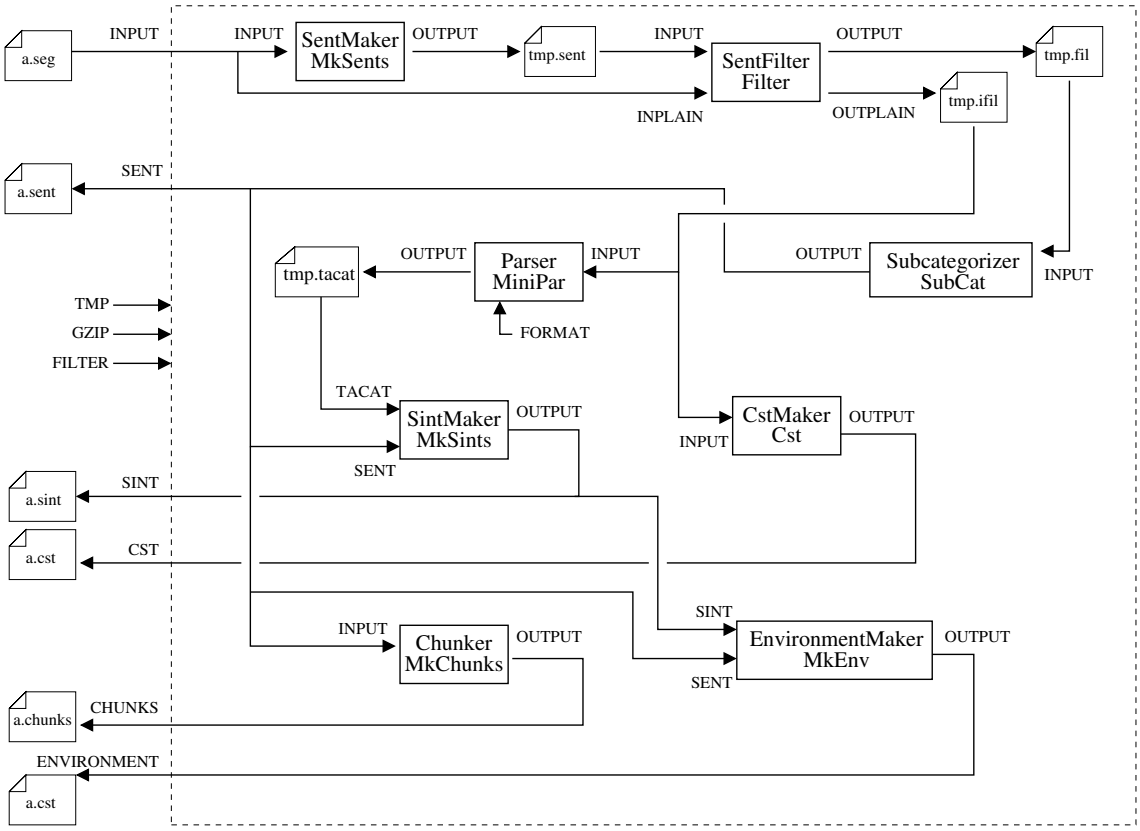


Figura 4.5: Estructura Interna del Processador Prolog

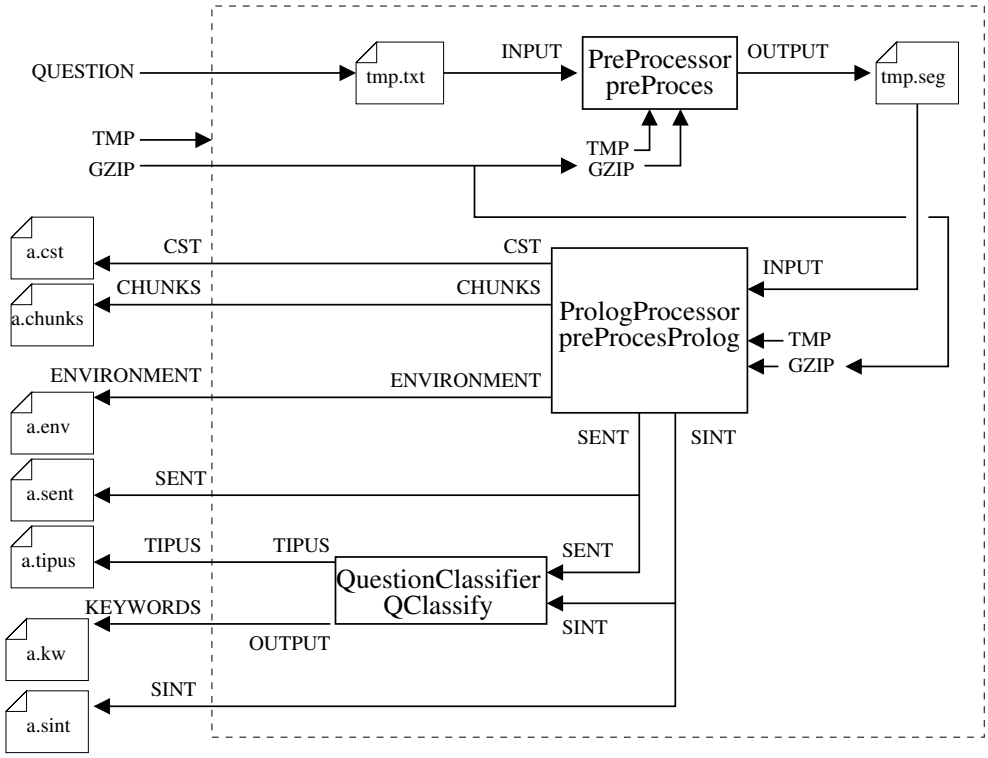


Figura 4.6: Estructura Interna del Processador de la Pregunta

4.5.2 Lemmatizer WordNet2

És un Servidor que utilitza el lemmatitzador de **WordNet 2.0** per a afegir el lema als mots que venen del **TnT**.

Arguments de Línia de Comandes	
1	*Path* a les dades de **WordNet 2.0**.
Paràmetres de la Petició	
INPUT	Fitxer d'Entrada. S'espera un fitxer taggejat no numerat.
OUTPUT	Fitxer de Sortida. És un fitxer lematitzat: tokenitzat amb la forma del Token, i el seu lema i Tag, precedit tot d'un asterisc per a indicar el lloc on anirà l'índex.

Taula 4.3: Lemmatizer WordNet2

La implementació és en **C++**, utilitzant l'API que el propi **WordNet** proporciona per accedir a les seves dades.

4.5.3 Abionet

És un *Named Entity Classifier* desenvolupat al TALP ([CM02]) per Xavier Carreras. L'**Abionet** pròpiament dit només etiqueta cada mot com a *Beginning*, *Inside* o *Outside* una *Named Entity*, i el Servidor s'encarrega d'ajuntar els diversos tokens que formen part d'una Named Entity en un sol terme, a part de transformar la classificació que en dona l'**Abionet** en un Tag.

El Servidor és un *Wrapper* en **Perl** que crida a l'executable d'**Abionet** i en processa la sortida, ajuntant les *Named Entities* multiterme i transformant els tags.

Arguments de Línia de Comandes	
1	*Path* a l'executable **Abionet**.
2	Codi de l'idioma.
3	Model a usar per defecte.
4	Directori per defecte dels models.
Paràmetres de la Petició	
INPUT	Fitxer d'Entrada. S'espera un fitxer lematitzat i amb la primera columna indicant l'índex del token (o un asterisc per a reservar l'espai)
OUTPUT	Fitxer de Sortida. És un fitxer tokenitzat en què hi ha índex (o un asterisc), forma, lema i tag del token, i el tipus de *Named Entity* que és, o un asterisc si no ho és.
SPEC	*Path* al Model a utilitzar en comptes del per defecte. Si és relatiu, es considera relatiu al directori per defecte dels models.

Taula 4.4: Abionet

4.5.4 Alembic

És un *workbench* del grup MITRE, que s'utilitza com a *Named Entity Recognizer* per a Dates i Temps, complementant el que s'ha detectat fins ara. El Servidor que s'ha implementat està molt pensat per a aquest ús.

Arguments de Línia de Comandes	
1	*Path* al directori de l'**Alembic**.
Paràmetres de la Petició	
INPUT	Fitxer d'Entrada. S'espera un fitxer tokenitzat o de text pla.
MERGE	Fitxer a fusionar. S'espera un fitxer tal com el treu l'**Abionet**. Si apareix aquest paràmetre, s'activa el mode de fusió. Si no, es treballa en mode normal.
OUTPUT	Fitxer de Sortida. Si es treballa en mode Merge, el format és el mateix que el de l'**Abionet**. En cas contrari, és un fitxer de text pla amb marcat XML de les *Named Entities*
TMP	Paràmetre TMP tal com definit a la secció 3.4.1.
PASSAGES	Booleà (per defecte fals) que indica si el fitxer d'entrada té marques de Passatge, que cal treure abans de passar l'**Alembic**.

Taula 4.5: Alembic

El Servidor primerament crida l'**Alembic** perquè processi el fitxer d'Entrada (prèviament filtrat si hi havia marques de Passatge). Llavors ve un procés de fusió de les *Named Entities*, en què a les que ja hi havia al fitxer a fusionar se li afegeixen les de tipus Data i Temps que hagi trobat l'**Alembic**. En el cas en què no hi hagués fitxer a fusionar, el resultat és directament el que ens dóna l'**Alembic**.

4.5.5 Anotador Semàntic basat en WordNet

En aquest mòdul s'afegeix a cada token (de les categories gramaticals per a què es disposa d'informació semàntica, en concret per a l'anglès verbs i noms) la informació referent als sentits que pot tenir. Com s'ha comentat anteriorment, per a representar els sentits d'un mot s'usen els *synsets* de WordNet. La decisió es fa únicament en funció del lema i la categoria gramatical.

També s'afegeix la informació sobre si el token és un actor o un gentilici.

Arguments de Línia de Comandes	
1	Codi de l'idioma.
2	Versió de WordNet a utilitzar (disponibles 1.5 i 1.6).
3	Nivell d'anàlisi per defecte. Es pot triar entre tres nivells: 1 : Actors/gentilicis, *synsets* i camí d'hiperònims. 2 : Informació nivell 1 més *Top Concept Ontology*. 3 : Informació nivell 2 més Codis de Magnini.
4	*Path* a les dades de WordNet.
Paràmetres de la Petició	
INPUT	Fitxer a fusionar. S'espera un fitxer tal com el treu l'**Abionet**.
OUTPUT	Fitxer de Sortida. És un fitxer tokenitzat amb tota la informació que ja tenia més dues, tres o quatre columnes (segons el nivell d'anàlisi que volem) amb la informació semàntica.
LEVEL	Indica el nivell d'anàlisi que volem per a aquest fitxer. Si no s'especifica, s'usa el per defecte.

Taula 4.6: Anotador Semàntic basat en WordNet

Es tracta d'un mòdul implementat de forma pura en **Perl**. Per accedir a WordNet, es fan servir volcats en format **DB File**, típicament utilitzats al món **Perl**.

4.5.6 EngSegmenter

És un Segmentador per a l'Anglès, adaptat d'una altra utilitat de Dani Ferrés. Addicionalment, genera un fitxer amb format **Prolog** indicant a quin passatge pertany cada oració en què se segmenta el text.

4.5.7 FreeLing

Es tracta d'un *suite** d'analitzadors lingüístics, desenvolupada al centre TALP ([CC04]). Inclou tokenitzador, segmentador, analitzador morfològic, detector de *Named Entitites*, reconeixedor de dates, temps i quantitats i tagger, així com chunking basat en *charts**.

Ara per ara, es troba implementat el Servidor que engloba tots aquests aspectes excepte les *Named Entities* i el chunking. Per una banda, el chunking, com veurem, té lloc en una fase posterior del processat; i per l'altra, **FreeLing** és una eina que també segueix en desenvolupament, i en el moment d'escriure el Servidor el chunking era una funcionalitat que encara mancava, mentre que les *Named Entities* encara es consideren en una etapa molt primerenca de desenvolupament.

La implementació ha estat en **C++**, aprofitant que **FreeLing** està estructurat com una llibreria a enllaçar amb l'aplicació que l'utilitza.

4.5.8 PreProces

Es tracta del primer Servidor-Client que trobem, i engloba tots els processos anteriors, adequant l'el·lecció dels Servidors i paràmetres d'acord amb la llengua (ara per ara; segurament, s'hi afegiran

Arguments de Línia de Comandes	
no en té	
Paràmetres de la Petició	
INPUT	Fitxer a fusionar. S'accepta qualsevol fitxer tokenitzat amb informació, sempre i quant tingui la forma a la 2a columna i el lema a la 4a. A la 1a hauria de tenir asteriscos per a indicar que hi ha l'espai per a l'índex de l'element i de l'oració.
OUTPUT	Fitxer de Sortida. Segueix el mateix format que el fitxer d'entrada, però a la primera columna té els índex de l'oració i del token, separats per un subratllat(_).
PASSAGES	Fitxer de text pla que conté el mateix text que ens arriba per INPUT, però conservant les separacions de Passatge. Si no s'especifica, no es té en compte la divisió en Passatges.
DOCUT	Si s'especifica fitxer de Passatges, es pot generar el fitxer que manté la relació de a quin Passatge correspon cada oració. Aquesta informació apareix en forma de predicats **Prolog**: `docut(Or,Pa)` significa que l'oració `Or` pertany al passatge `Pa`. Si no s'especifica, no es genera aquest fitxer.

Taula 4.7: EngSegmenter

Arguments de Línia de Comandes	
1	*Path* al fitxer de configuració de **FreeLing**.
Paràmetres de la Petició	
INPUT	Fitxer d'Entrada. Tan pot ser un fitxer de text pla com un de ja tokenitzat (però sense altra informació).
OUTPUT	Fitxer de Sortida. Segueix el mateix format que el fitxer d'entrada, però a la primera columna té els índex de l'oració i del token, separats per un subratllat(_).
TOKENIZED	Booleà (per defecte fals) que indica si el fitxer d'entrada ja està tokenitzat, o bé és un fitxer de text pla.
LINE	Booleà (per defecte fals) que indica si cada línia de l'entrada s'ha de processar per separat.

Taula 4.8: FreeLing

més paràmetres en un futur proper).

Arguments de Línia de Comandes	
1	Codi de l'idioma.
2	Adreça del host on es troba el MetaServidor.
3	Port on escolta connexions el MetaServidor.
Paràmetres de la Petició	
INPUT	Fitxer d'Entrada. S'espera un fitxer de text pla.
OUTPUT	Fitxer de Sortida. Es tracta del fitxer que retorna el Segmentador.
TMP	Paràmetre TMP tal com definit a la secció 3.4.1.
GZIP	Paràmetre GZIP tal com definit a la secció 3.4.1.
PASSAGES	Booleà (per defecte fals) que indica si el fitxer d'Entrada està organitzat en Passatges.
DOCUT	Fitxer on deixar la relació entre Oracions i Passatges, tal com la retorna **EngSegmenter**

Taula 4.9: PreProces

La implementació és en **Perl**, utilitzant les classes desenvolupades durant el Projecte.

4.5.9 MkSents

Es tracta d'un Servidor que transforma la informació que hem obtingut durant el PreProcés en predicats **Prolog**, per a fer la seva càrrega més senzilla des dels Servidors que estan implementats en aquest Llenguatge.

Arguments de Línia de Comandes	
no en té	
Paràmetres de la Petició	
INPUT	Fitxer d'Entrada. S'espera un fitxer tal com el retorna el **EngSegmenter**.
OUTPUT	Fitxer de Sortida. És un fitxer amb predicats **Prolog**, en concret, els predicats: `sent(Or,Ini,L)`, l'Oració `Or` comença a l'índex de token `Ini`, i la llista dels seus tokens és `L`. Cada token ve representat per una tupla amb els diversos atributs.

Taula 4.10: MkSents

Està implementat en **Perl** pur.

4.5.10 Filter

Es tracta d'un mòdul que elimina les oracions que apareixen repetides en el fitxer, per a evitar dur a terme els processos lingüístics, que poden ser costosos, més d'un cop sobre el mateix fragment. Com que en el procés de Recuperació de Passatges hi ha un solapament entre ells, és possible que la mateixa oració ens aparegui diversos cops.

El filtratge funciona sobre els `sent` obtinguts en el Servidor anterior. Tanmateix, també es filtra el fitxer tokenitzat, per a mantenir-los ambdós sincronitzats.

Arguments de Línia de Comandes	
no en té	
Paràmetres de la Petició	
INPUT	Fitxer d'Entrada. S'espera un fitxer amb `sents`.
OUTPUT	Fitxer de Sortida. És un fitxer amb els mateixos `sents`, però eliminant els repetits.
INPLAIN	Fitxer d'Entrada de Text. S'espera un fitxer tokenitzat, amb l'índex de l'oració i del token a la primera columna. Pot haver-hi més atributs.
OUTPLAIN	Fitxer de Sortida de Text. El format és el mateix de l'Entrada. El filtratge del fitxer de Text només té lloc si s'especifiquen tots dos paràmetres: INPLAIN i OUTPLAIN.

Taula 4.11: Filter

El codi principal està en **Prolog**, i tenim un *Wrapper* en **C++**, tal com hem comentat a 4.1.1. Aquest *Wrapper* rep la llista de les Oracions que **Prolog** ha filtrat, i se n'encarrega, addicionalment, de filtrar el Fitxer de Text, si se n'ha especificat.

4.5.11 SubCat

És un Servidor que Subcategoritza les *Named Entities* de tipus *Location*(lloc geogràfic) segons siguin ciutat, país, riu, muntanya... S'utilitza el sistema desenvolupat al TALP i explicat a [FM04a].

Arguments de Línia de Comandes	
no en té	
Paràmetres de la Petició	
INPUT	Fitxer d'Entrada. S'espera un fitxer amb `sents`.
OUTPUT	Fitxer de Sortida. És un fitxer amb els mateixos `sents`, però amb la subclassificació de *Named Entities*.

Taula 4.12: SubCat

El codi està en **Prolog**, incloent un fitxer amb regles generades automàticament per aprenentatge automàtic amb el sistema **Foil** de Ross Quinlan([Qu93]). Com a tots els Servidors en **Prolog**, hi ha el *Wrapper* en **C++**.

4.5.12 Minipar

Minipar és un parser per a la llengua Anglesa desenvolupat per Dekang Lin ([Li98]). El Servidor utilitza **Minipar** per a parsejar el text, i després retorna el resultat en el format que interessi d'entre el que imita la sortida de l'eina **tacat**[2], o el que retorna el parsing com a predicats **Prolog**.

La implementació és en **C++**, utilitzant la API que **Minipar** ofereix per a ser usat des d'Aplicacions.

4.5.13 MkSints

Es tracta de nou d'un conversor, que transforma la informació sintàctica obtinguda al pas anterior a un format més adequat, el de predicats `sint`.

En aquest cas, es tracta d'un *Wrapper* en **Perl** que crida a una aplicació externa.

[2]Chunker desenvolupat pel TALP

Arguments de Línia de Comandes	
1	*Path* a les dades de **Minipar**.

Paràmetres de la Petició	
INPUT	Fitxer d'Entrada de Text. S'espera un fitxer tokenitzat, amb l'índex de l'oració i el token a la primera columna, i la forma del mot a la segona.
FORMAT	Format de la Sortida. Pot ser 'PROLOG' o 'TACAT', per defecte essent el primer.
OUTPUT	Fitxer de Sortida. Segons el valor de FORMAT, té la forma de: - Predicats **Prolog** `tacat(Or,L)`, on `Or` és l'índex de l'Oració i `L` la llista amb l'anàlisi. - Estructures de claus com les que retornava el parser **tacat**.

Taula 4.13: Minipar

Arguments de Línia de Comandes	
1	*Path* a l'executable **mksints**.

Paràmetres de la Petició	
SENT	Fitxer d'Entrada de Sents. S'espera un fitxer amb `sents`.
TACAT	Fitxer d'Entrada del Parsing. S'espera un fitxer amb `tacats`.
OUTPUT	Fitxer de Sortida. És un fitxer amb predicats **Prolog**: `sint(Or,L)`, on `Or` és l'índex de l'Oració i `L` la llista amb l'anàlisi (en un format diferent a `tacat`.

Taula 4.14: MkSints

4.5.14 Cst

Es tracta d'un Servidor que ajuda a trobar el Contingut Semàntic del Terme, per a posteriorment trobar el Contingut Semàntic del Concepte. El Contingut Semàntic del Terme està format per tots els termes que poden substituir un terme donat, amb un cert valor cadascun d'ells. Els substituts provenen dels sinònims i hiperònims i hipònims de primer nivell del terme a **WordNet**. Aquesta informació s'usa posteriorment per al càlcul del Contingut Semàntic del Concepte (veure [Vi02] i secció 4.5.21).

Arguments de Línia de Comandes	
1	*Path* a les dades.
Paràmetres de la Petició	
INPUT	Fitxer d'Entrada. S'espera un fitxer tokenitzat tal com el treu **EngSegmenter**.
OUTPUT	Fitxer de Sortida. És un fitxer tokenitzat, amb els índexs d'oració i de token, forma, lema, tag, informació d'actor/gentilici i Cst de cada token.

Taula 4.15: Cst

Es tracta d'un Servidor en **Perl**, que accedeix a la informació necessària utilitzat **DB Files**.

4.5.15 MkChunks

Es tracta d'un chunker que localitza els sintagmes que apareixen al text utilitzant gramàtiques DCG ([PW80]).

Arguments de Línia de Comandes	
no en té	
Paràmetres de la Petició	
INPUT	Fitxer d'Entrada. S'espera un fitxer amb `sents`.
OUTPUT	Fitxer de Sortida. És un fitxer amb un format específic: cada chunk està en una línia, i apareixen els índexs de token del nucli del chunks i dels seus modificadors.

Taula 4.16: MkChunks

Es tracta d'un Servidor escrit en **Prolog** amb el *Wrapper* en **C++**.

4.5.16 MkEnv

Com s'explica a 1.5, l'Entorn captura els contingut semàntic de les oracions. Aquest Servidor s'encarrega de trobar l'Entorn de les Oracions d'un fitxer.

Es tracta d'un Servidor escrit en **Prolog** amb el *Wrapper* en **C++**.

4.5.17 PreProcesProlog

Es tracta del segon Servidor-Client, que agrupa els processos que segueixen al **PreProces**.

La implementació és en **Perl**, utilitzant les classes desenvolupades durant el Projecte.

Arguments de Línia de Comandes	
no en té	
Paràmetres de la Petició	
SENT	Fitxer d'Entrada de Sents. S'espera un fitxer amb `sents`.
SINT	Fitxer d'Entrada de Sints. S'espera un fitxer amb `sints`.
OUTPUT	Fitxer de Sortida. És un fitxer amb predicats **Prolog**: `env(Or,Env)`, on `Or` és l'índex de l'Oració i `Env` la llista que conté les restriccions de l'Entorn.

Taula 4.17: MkEnv

Arguments de Línia de Comandes	
1	Codi de l'idioma.
2	Adreça del host on es troba el MetaServidor.
3	Port on escolta connexions el MetaServidor.
Paràmetres de la Petició	
INPUT	Fitxer d'Entrada. S'espera un fitxer amb el format que retorna **EngSegmenter**.
SENT	Fitxer de Sortida de Sents. Conté predicats `sent`.
SINT	Fitxer de Sortida de Sints. Conté predicats `sint`.
CST	Fitxer de Sortida de Cst. Està tokenitzat segons el format que retorna **Cst**.
CST	Fitxer de Sortida de Chunks. Té el format que retorna **MkChunks**.
ENVIRONMENT	Fitxer de Sortida d'Entorns. Conté predicats `env`.
TMP	Paràmetre TMP tal com definit a la secció 3.4.1.
GZIP	Paràmetre GZIP tal com definit a la secció 3.4.1.
FILTER	Booleà (per defecte cert) que indica si cal fer el procés de Filtratge de les Oracions repetides.

Taula 4.18: PreProcesProlog

4.5.18 QClassify

Es tracta d'un classificador de les Preguntes, segons la taxonomia establerta per a la Metodologia TALP-UdG. Addicionalment, determina les paraules clau a ser usades per a la Recuperació de Passatges.

Arguments de Línia de Comandes	
no en té	
Paràmetres de la Petició	
SENT	Fitxer d'Entrada de Sents. S'espera que contingui predicats `sent`.
SINT	Fitxer d'Entrada de Sints. S'espera que contingui predicats `sint`.
OUTPUT	Fitxer de Sortida. Per a cada pregunta de l'entrada, treu una línia amb el seu índex, el seu tipus, la confiança de la regla amb què s'ha determinat el tipus i la llista de *keywords*.
TIPUS	Fitxer de Sortida de Tipus. Conté la informació sobre el tipus de pregunta en format **Prolog**: `tipus(P,T)` indica que la pregunta amb índex `P` té tipus `T`. Si no s'especifica aquest Paràmetre, aquest fitxer no es genera.

Taula 4.19: QClassify

Es tracta d'un Servidor escrit en **Prolog** amb el *Wrapper* en **C++**.

4.5.19 PreProcesPregunta

És el tercer Servidor-Client, i engloba tot el processat de la Pregunta. Permet que aquesta viatgi dins la Petició, en comptes d'estar escrita en un fitxer a disc.

Arguments de Línia de Comandes	
1	Codi de l'idioma.
2	Adreça del host on es troba el MetaServidor.
3	Port on escolta connexions el MetaServidor.
Paràmetres de la Petició	
QUESTION	Pregunta (no fitxer amb la Pregunta!).
SENT	Fitxer de Sortida de Sents. Conté predicats `sent`.
SINT	Fitxer de Sortida de Sints. Conté predicats `sint`.
CST	Fitxer de Sortida de Cst. Està tokenitzat segons el format que retorna **Cst**.
CHUNKS	Fitxer de Sortida de Chunks. Té el format que retorna **MkChunks**.
ENVIRONMENT	Fitxer de Sortida d'Entorns. Conté predicats `env`.
TIPUS	Fitxer de Sortida de Tipus. Conté predicats `tipus`.
KEYWORDS	Fitxer de Sortida de Keywords. Segueix el format de la sortida de **QClassify**.
TMP	Paràmetre TMP tal com definit a la secció 3.4.1.
GZIP	Paràmetre GZIP tal com definit a la secció 3.4.1.

Taula 4.20: PreProcesPregunta

Es tracta d'un Servidor escrit en **Prolog** amb el *Wrapper* en **C++**.

4.5.20 MgRetr

Es tracta d'un Servidor que realitza tot el procés de Recuperació de Passatges descrit a 1.5, utilitzant el sistema de Recuperació d'Informació **Mg**([WM99]).

En aquest cas, els arguments de línia de comandes es passen mitjançant *switchos*, ja que n'hi ha un nombre considerable (estan adaptats del mòdul homòleg de la implementació anterior).

El Servidor és un codi en **Perl** que s'encarrega de posar en marxa tres processos fills **mgquery** (utilitat de l'**Mg** per a fer consultes sobre col·leccions). Aquests processos es reaprofiten de Petició en Petició, ja que el seu temps d'inicialització és significatiu. La comunicació amb ells té lloc a través d'una *pipe* unidireccional, i la sincronització per a saber quan s'han atés les *queries* es fa amb una *named pipe**.

4.5.21 Csc

La construcció del Contingut Semàntic del Concepte té lloc un cop preguntes i passatges han estat processats, i s'han obtingut els seus Chunks i Cst. El valor que s'emmagatzema és la similaritat entre CSCs dels conceptes de la pregunta i dels que aparèixen a cada oració.

La implementació és directament en **Perl**, accedint a les dades necessàries per al calcul del Csc a través de **DB Files**.

4.5.22 Xtract

Es tracta del Servidor que finalment du a terme l'extracció de la Resposta. Per a fer-ho, utilitza gran part de la informació que hem anat obtenint al llarg de tot el procés.

El codi és en **Prolog**, i s'ha escrit un *Wrapper* en **C++**.

4.5.23 Millor

Aquest penúltim Servidor s'encarrega de triar la millor resposta d'entre totes les proporcionades per **Xtract**. Per a fer-ho, té en compte aspectes com el valor de Csc de les oracions on s'ha trobat la resposta, la confiança de les regles que **Xtract** ha utilitzat per a obtenir-les, el nombre de vegades que una mateixa resposta es troba a la col·leccio...

Es tracta de codi en **Prolog** amb el *Wrapper* en **C++**.

4.5.24 QA

Es tracta del Servidor-Client que engloba tot el procés descrit fins ara i, per tant, les opcions que pot oferir són innombrables. Tanmateix, ara per ara només n'hem considerat un petit subconjunt, a anar ampliant.

La implementació és en **Perl**, utilitzant les classes desenvolupades durant el Projecte.

4.6 Conclusions

La implementació del sistema de CR seguint la metodologia TALP-UdG ha permès acomplir un doble objectiu:

- Disposar d'un sistema de CR integrat, que pot servir de *baseline** per a Sistemes més desenvolupats i amb característiques determinades (com ara el que es necessita per al projecte CHIL, capaç de processar transcripcions orals i transparències).

- Avaluar la validesa del disseny dut a terme, implementant-hi un sistema de CR operatiu a sobre.

Arguments de Línia de Comandes	
-pathMg	*Path* a l'executable **Mg**.
-pathData	*Path* a les Dades de l'**Mg**.
-Dclust	*Path* del Directori amb els fitxers de clustering de *Named Entities*
-Facronims	*Path* del Fitxer d'expansió d'acronims.
-ExpQTextNE	Segueix l'expressió regular **i?e?** per a indicar si volem expansió de les *Named Entities* a la *query* de text i *Named Entities*. La i activa l'expansió interna i la e, l'externa. Per defecte, no hi ha expansió.
-ExpQNE	Equivalent al parametre anterior, però per a la *query* de només *Named Entities*.
-WTQ	Activa la ponderació dels termes de la *query*.
-Cdoc	Nom de la Col·leccio de Keywords.
-Cne	Nom de la Col·leccio de *Named Entities*. Si no s'especifica, no s'usa a la cerca.
-CdocRetrieval	Nom de la Col·leccio de Documents Textuals.
-NR	Número de documents a recuperar amb la Query Ranked. (per defecte, 200)
-Vdoc	Llindar de recuperació de la Query Ranked. (per defecte, 0%)
-NB	Número de documents a recuperar amb la Query Booleana. (per defecte, 200)
-Vdoc	Llindar de recuperació de la Query Ranked. (per defecte, 0%)
-K	Número de documents a recuperar amb la Query de Passatges. (per defecte, 100)
-Vpass	Llindar de recuperació de la Query de Passatges. (per defecte, 0%)
-M	Nombre de paràgrafs dels passatges a indexar. (per defecte, 3)
-S	Solapament en paràgrafs dels passatges a indexar. (per defecte, 1)
Paràmetres de la Petició	
INPUT	Fitxer d'Entrada. S'espera el format retornat per **QClassify**.
QUERY	Alternativa al Fitxer d'Entrada. En aquest cas, es passa directament la cadena amb la query, amb la forma: $kw_1\#tag_1 \ldots kw_n\#tag_1@ne_1\#tip_1 \ldots ne_m\#tip_m$ on kw_i són les *keywords* no *Named Entity*, i tag_i els seus tags. i ne_j són les *keywords Named Entity*, i tip_i el seu tipus.
OUTPUT	Fitxer de Sortida. És un fitxer de text pla amb els passatges recuperats, precedits pel seu identificador de Passatge.

Taula 4.21: MgRetr

Arguments de Línia de Comandes	
1	*Path* al directori de dades.

Paràmetres de la Petició	
QCHUNKS	Fitxer amb els Chunks de la Pregunta. S'espera el format retornat per **MkChunks**.
QCST	Fitxer amb els Cst de la Pregunta. S'espera el format retornat per **Cst**.
PCHUNKS	Fitxer amb els Chunks dels Passatges. S'espera el format retornat per **MkChunks**.
PCST	Fitxer amb els Cst dels Passatges. S'espera el format retornat per **Cst**.
OUTPUT	Fitxer de Sortida. És un fitxer amb predicats **Prolog**: `csc(Or, V)`, que indica que l'Oració `Or` té com a valor de Csc `V`.

Taula 4.22: Csc

Arguments de Línia de Comandes	
no en té	

Paràmetres de la Petició	
MODE	Mode de treball de l'Extractor. Determina l'estrategia de control a seguir en extreure la resposta. Pot ser 'standard' (valor per defecte), ' strict_exhaustiu', 'strict_relaxat', 'relaxat', 'global', 'global2' o 'global3' (per ara).
QSENT	Fitxer de Sents de la Pregunta. S'espera que contingui un predicat `sent`.
QSINT	Fitxer de Sints de la Pregunta. S'espera que contingui un predicat `sint`.
QENVIRONMENT	Fitxer de l'Entorn de la Pregunta. S'espera que contingui un predicat `env`.
QTIPUS	Fitxer amb el Tipus de la Pregunta. S'espera que contingui un predicat `tipus`.
PSENT	Fitxer de Sents dels Passatges. S'espera que contingui predicats `sent`.
PSINT	Fitxer de Sints dels Passatges. S'espera que contingui predicats `sint`.
PCSC	Fitxer de Csc's dels Passatges. S'espera que contingui predicats `csc`.
PENVIRONMENT	Fitxer d'Entorns dels Passatges. S'espera que contingui predicats `env`.
PDOCUT	Fitxer amb els Documents propietaris de les oracions dels Passatges. S'espera que contingui predicats `docut`.
OUTPUT	Fitxer de Sortida. Conté predicats **Prolog**: `resposta/12`, amb tota la informació sobre les possibles respostes trobades.

Taula 4.23: Xtract

Arguments de Línia de Comandes	
no en té	
Paràmetres de la Petició	
SENT	Fitxer de Sents dels Passatges. S'espera que contingui predicats `sent`.
ANSWER	Fitxer de Respostes. S'espera que contingui predicats `resposta`.
TIPUS	Fitxer de Tipus de la Pregunta. S'espera que contingui un predicat `tipus`
OUTPUT	Fitxer de Sortida. És un fitxer que conté una línia amb l'identificador del document d'on hem extret la resposta i aquesta, o bé 'NIL'si no s'ha pogut trobar resposta.

Taula 4.24: QClassify

Arguments de Línia de Comandes	
1	Codi de l'idioma.
2	Adreça del host on es troba el MetaServidor.
3	Port on escolta connexions el MetaServidor.
4	Nom de la Col·leccio de Keywords.
5	Nom de la Col·leccio de *Named Entities*. Si no s'especifica, no s'usa a la cerca.
6	Nom de la Col·leccio de Documents Textuals.
no en té	
Paràmetres de la Petició	
QUESTION	Text de la Pregunta.
OUTPUT	Fitxer de Sortida. Segueix el format de la sortida de **Millor**.
TMP	Paràmetre TMP tal com definit a la secció 3.4.1.
GZIP	Paràmetre GZIP tal com definit a la secció 3.4.1.

Taula 4.25: QA

Tanmateix, la feina no ha fet més que començar, ja que molts dels components necessiten un intens treball de recerca per a millorar els seus resultats. Aquesta feina ja està en marxa, implicant un bon nombre de recercadors del TALP.

Per altra banda, a molts dels Servidors caldrà afegir-los paràmetres, per a fer-los més customitzables, i possiblement esdevingui necessari afegir nous Servidors amb característiques concretes.

El capítol següent fa un petit esbòs de la feina encara per dur a terme.

Capítol 5

Direccions Futures

Al llarg de la memòria hem vist diversos aspectes que han quedat poc estudiats, o bé permetien anar més enllà i per raons obvies de temps han quedat relegats. Tots aquests temes han quedat com a possibilitats de treball futur. És gairebé segur que molts d'ells els acabaré tractant durant els mesos vinents, mentre segueixi el meu treball al TALP dins el projecte CHIL, especialment els esbossats a curt termini.

Els treballs a mig i llarg termini són més aviat possibilitats que ofereix el Sistema, que no necessàriament han d'acabar duent-se a terme. Segons les necessitats i la seva evolució pot ser que acabi triant-se alguna de les opcions aquí presentades.

5.1 Treball a Curt Termini

A curt termini, les tasques a desenvolupar són:

- **Millora dels diversos components de cara a la *Question Answering Track* del TREC-2004, la segona quinzena del mes de Juliol d'enguany**.

  La metodologia de CR encara té aspectes a millorar, tant en llengua anglesa com espanyola, i a nivell del TALP s'està treballant per a fer més bons els resultats respecte al TREC-2003.

- **Restructuració del codi dels mòduls**.

  El codi que s'ha adaptat per a alguns Servidors té petites mancances d'organització interna. Com que cada cop l'equip que treballa en CR engloba més gent, és necessari que una eina de comunicació entre ells com és el codi sigui clara i estructurada per a permetre que el màxim de gent possible pugui entendre'l i fer-hi aportacions.

- **Desenvolupament de Servidors per a treballar amb transcripcions orals**.

  Les característiques de les transcripcions orals són completament diferents a les dels textos directament escrits: per una banda, hi ha una taxa encara alta d'errors en el Reconeixement Automàtic de la Parla, especialment si es tracta de parla espontània. Per una altra, la llengua oral de per sí és sovint inconnexa, agramatical, es corregeix sovint el que un mateix diu... La conjunció d'aquests dos factors fa que sigui necessària molta recerca per a trobar nous mètodes per a tasques que en llengua escrita es consideren resoltes (com ara segmentació del text en oracions).

  El projecte CHIL se centra en aquesta problemàtica.

5.2 Treball a Mig Termini

- **Polítiques diferents en l'Atenció de Peticions**.

En parlar del disseny del MetaServidor (secció 3.3.3) hem comentat la possibilitat de canviar l'estructura que manté les Peticions en espera per ser ateses. En comptes d'una cua FIFO, hem esbossat la possibilitat d'usar alguna variant d'una cua amb prioritats. Els diversos algorismes per assignar les prioritats poden ser una línia d'exploració, ja que cal tenir en compte les característiques especials de les Peticions del Processament de Llenguatge Natural en general i de cada Servidor en particular.

Per altra banda, aspectes com si iniciar un altre Servidor de les mateixes característiques quan la cua d'algun d'ells estigui molt plena, o bé quan aturar els Servidors inactius també poden resultar interessants de cara a millorar el comportament del Servidor.

- **Pas del MetaServidor a C++**.

  Per altra banda, en parlar de la implementació del MetaServidor (secció 4.2), hem comentat que quan aquesta pugui considerar-se estable pot resultar interessant fer-ne un port cap al llenguatge **C++**. La raó principal és l'eficiència, aspecte que el llenguatge **C++** té com a objectiu fonamental.

  Encara que per al tipus d'Aplicacions per a què ha estat concebut el MetaServidor no té perquè suposar una penalització en quant a aquest aspecte, si es volgués aprofitar per a Sistemes en què el temps de procés fos proper al de comunicació, un canvi des de **Perl** cap a **C++** pot suposar una millora en quant a temps de resposta. Aquesta millora es pot notar especialment si es donen situacions de congestió, que d'aquesta manera potser es puguin evitar.

  En cas que no canviïn les característiques de l'aplicació, es possible que no valgui la pena fer la traducció. Només la relativa similaritat entre un llenguatge i l'altre pot fer-la més senzilla i per tant, més interessant.

- **Estudi de la viabilitat i interés del canvi de Format**.

  També hem parlat del canvi dels formats de les dades i dels fitxers intermitjos a la seccio 3.2. Un altre aspecte a considerar és si es continua mantenint els formats emprats fins ara o es fa el canvi cap a la forma de treballar exposada als Apèndixs B i C.

5.3 Treball a Llarg Termini

- **Extensió a una Arquitectura en *Cluster***

  Com hem comentat en parlar de l'arquitectura a la secció 3.1, el sistema desenvolupat permetria una extensió cap al treball en *Cluster*. La idea, que apareix a la figura 5.1, és de tenir un conjunt de MetaServidors en diverses màquines, connectats entre ells ja sigui amb un MetaServidor central[1] o bé formant un arbre d'expansió qualsevol, que col·laboressin en la localització dels Servidors que puguin demanar els Clients. Un requisit d'aquestes màquines seria que compartissin el sistema de fitxers, ja que a les Peticions s'envien els *paths* d'aquests. Si no fos així, s'hauria de fer alguna extensió al protocol per a salvar aquest problema.

  Les Peticions haurien de ser enrutades pels MetaServidors des dels Clients que les facin fins als Servidors que les hagin de rebre.

  Un sistema així permetria augmentar la productivitat del sistema, ja que algunes tasques poden dur-se a terme en paral·lel, i si es distibuís el procés en múltiples màquines s'aprofitarien els avantatges de tenir diverses CPUs.

- **Adaptació per a la Open Agent Architecture** Una alternativa a desenvolupar un sistema per a connectar els MetaServidors en *Cluster* és aprofitar alguna Arquitectura similar ja existent, com les que hem mencionat a la secció 3.1.4. D'entre elles, la que més interés

[1] que es guanyaria el nom de MetaMetaServidor

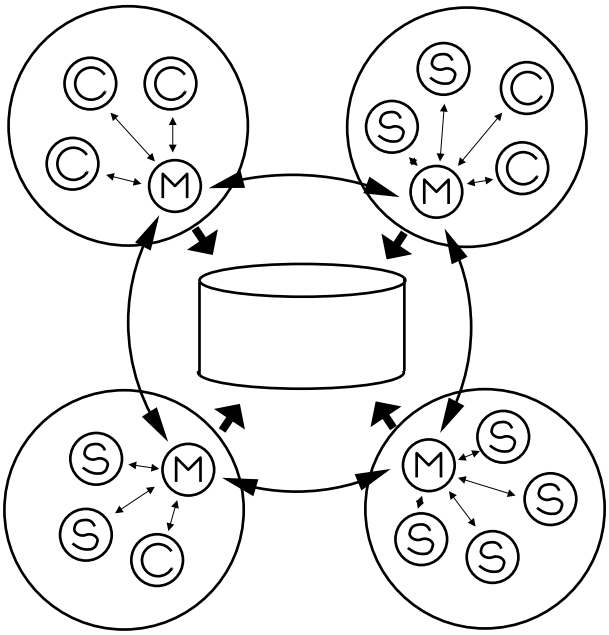


Figura 5.1: MetaServidors distribuits

té és l'**Open Agent Architecture**, especialment pel fet de ser l'escollida per al projecte CHIL.

Per a portar la nostra Arquitectura envers l'**OAA** seria necessari transformar tota la comunicació que ara es du a terme mitjançant Peticions en resolucions expressades en l'**Inter-agent Comunication Language**. Els paràmetres de les Peticions es transformarien en termes dins predicats **Prolog**, i s'hauria d'idear algun sistema per a activar i desactivar els Servidors (transformats ara en Agents) en funció de les demandes dels Clients. Possiblement seria necessari un altre Agent que dués a terme les tasques del MetaServidor que no desenvolupi el Facilitador, com ara la càrrega i descàrrega dinàmica dels Agents, segons el sistema de regles (que pot conservar-se o canviar-se per alguna altra forma de decisió).

Una dels objectius del disseny, si bé secundari però present igualment, ha estat que si finalment s'optés per aquesta opció els canvis a dur a terme siguin el més limitats possibles, bàsicament canviar la part de comunicació que ara utilitza el nostre protocol per crides a les llibreries de la **OAA**.

Capítol 6

Distribució del Temps Emprat

6.1 Planificació Inicial del Projecte

La taula 6.1 presenta la planificació que es va fer en el moment de l'entrega de l'informe (a principi d'Abril) del Projecte Final de Carrera. Es fam correspondre les diverses tasques a les fileres amb els períodes en què es podein realitzar, marcant amb un asterisc els períodes en què efectivament volíem realitzar cada tasca.

	1-15 Feb	16-29 Feb	1-15 Mar	16-31 Mar	1-15 Abr	16-30 Abr	1-15 Mai	16-31 Mai	1-15 Jun	16-30 Jun
Presa de Contacte	*									
Disseny		*	*							
Implementacio i Prova Individual				*	* (Informe)	*				
Prova General							*	*		
Redacció de Documentació								*	*	*

Taula 6.1: Planificació

Les Tasques en què es va dividir el Projecte eren:

Presa de Contacte Familiarització amb la metodologia, la implementació existent, i les diverses eines de PLN usades al TALP. Com pot observar-se, aquesta tasca havia tingut lloc durant la primera quinzena de Febrer.

Disseny Estudi més profund de la metodologia, i avaluació de les diferents arquitectures, decidint-nos per la de client/servidor exposada anteriorment. Havíem trigat un mes, durant la segona quinzena de Febrer i la primera de Març.

Implementació i Prova Individual Implementació dels diferents mòduls del sistema, i prova individual del codi resultant. Es tractava d'un procés amb força *feedback*, s'estava duent a terme des de la segona quinzena de Març i que s'esperava que em durés fins a finals d'Abril. Més detalladament, es va començar per implementar la llibreria d'accés a fitxers, i a continuació es va programar el MetaServidor. En el moment de l'informe s'estaven codificant els Servidors, i l'últim pas havia de ser el Client de Cerca de Resposta.

Prova General Prova del sistema en l'avaluació del CLEF. Havia de tenir lloc durant el mes de Maig.

Redacció de Documentació A partir de finals de Maig i fins la data de presentació del Projecte.

A la taula 6.3 figura el nombre d'hores (de forma molt aproximada) que s'esperava invertir en cada tasca, seguint la planificació exposada.

6.2 Distribució Final del Temps

Finalment, les tasques del projecte s'han dut a terme seguint la distribució que apareix a la taula 6.2. Com pot veure's, la part d'implementació s'ha allargat més del que estava pensat, fent que la prova general, la documentació i alguns petits detalls d'implementació hagin hagut de fer-se conjuntament durant el més de Juny.

Finalment no va ser possible tenir el sistema llest per a l'avaluació del CLEF, i la prova de foc definitiva serà el més de Juliol, amb la participació en el TREC.

	1-15 Feb	16-29 Feb	1-15 Mar	16-31 Mar	1-15 Abr	16-30 Abr	1-15 Mai	16-31 Mai	1-15 Jun	16-30 → Jun
Presa de Contacte	*									
Disseny		*	*							
Implementació i Prova Individual				*	*	*	*	*	*	
Prova General									*	*
Redacció de Documentació								*	*	

Taula 6.2: Distribució Final de la Feina

El canvi en la distribució de les hores pot apreciar-se també a la taula 6.3 (també es tracta de xifres aproximades).

Aspecte	Hores Estimades	Hores Emprades
Presa de Contacte	60 h	60h
Disseny	140 h	140 h
Implementació	220 h	310 h
Prova	130 h	80 h
Documentació	150 h	120 h
Total	700 h	710 h

Taula 6.3: Hores de Treball

Apèndix A

Glossari

Automatic Speech Recognition *Veure Reconeixement de la Parla.*

Base de Coneixement Estructura de Dades usada en aplicacions d'Intel·ligència Artificial en què s'emmagatzemema informació sobre el domini, tant a nivell general com sobre el problema concret a què s'està enfrontant.

Baseline Sistema que serveix de referència de rendiment per a futures millores.

Buffering Tècnica usada en molts aspectes de la Informàtica, que consisteix en emmagatzemar dades en un zona de memòria per a dur a terme processos en proporció més costosos (com per exemple operacions d'Entrada/Sortida o crides al sistema) un menor nombre de cops.

Callback Funció d'un programa que es passa com a paràmetre a una llibreria perquè sigui cridada com a resposta a algun event.

Chart Tècnica de parsing que construeix una taula per programació dinàmica per a millorar l'eficiència.

Chunking Procés de Llenguatge Natural consistent en agrupar els tokens* en sintagmes, però sense analitzar l'estructura interna d'aquests. També s'anomena *parsing** parcial o superficial (*shallow parsing*).

Per exemple, un chunking de la frase:

- El gos pelut menja carn de vedella anglesa.

pot ser:

- (El gos pelut) menja (carn) (de vedella anglesa).

Cluster (1) Grup de màquines que treballen de forma coordinada.

Cluster (2) Subconjunt d'elements d'un conjunt major que poden agrupar-se per raons de similaritat.

Codis de Magnini Codis assignats als synsets* de **WordNet*** que indiquen el domini sobre què tracten (**astronomia**, **medicina**, **indústria**, etc.).

Concordança de Patrons Tècniques la forma de treball de les quals és disposar d'una sèrie de regles que s'activen quan les dades segueixen una certa forma. El criteri d'activació ha de ser estrictament basat en la forma. En elles acostumen a utilitzar-se els *wildcards**.

Consulta Cadena en un llenguatge formal amb què es fa una petició a una Base de Dades o a un sistema de Recuperació d'Informació. Exemples en són les consultes SQL, o les cadenes introduïdes al buscador **Google**.

Correferència Relació que uneix dos elements lingüístics (termes o sintagmes) que fan referència a la mateixa entitat.

Detecció del Tema Procés de Llenguatge Natural que determina el tema sobre què parla un determinat fragment de text.

Enginyeria del Software Tècniques i metodologies basades en les usades històricament a les Enginyeries aplicades al procés de desenvolupament de Software.

EuroWordNet Xarxa similar a **WordNet*** que, a part de recollir relacions semàntiques (com sinonímia, hiperonímia o hiponímia) entre mots de diverses llengües europees, ofereix facilitats per a trobar els conceptes equivalents entre una llengua i una altra.

Frontend Aplicació la única funcionalitat de la qual és oferir una interfície més amigable a una altra aplicació.

Gazetteer Llista que conté *Named Entities* d'una determinada categoria, com ara una llista de noms d'empreses, o de ciutats.

Glue Language Llenguatge de Programació que permet de forma fàcil desenvolupar programes que integrin components amb característiques heterogènies, actuant de pont entre ells.

Hash (1) Funcions que mapegen un tipus complex (com per exemple una cadena de caràcters) sobre un rang dels enters. També es poden anomenar funcions de dispersió.

Hash (2) (Pròpiament, Taula de Hash) Estructura de dades que utilitza una funció de Hash per a emmagatzemar parells clau-valor i permetre-hi un accés eficient.

Heurística Regla de control d'un algorisme que es basa en propietats del problema a resoldre per a orientar l'algorisme cap a zones *a priori* més prometedores de l'espai de solucions.

Hiperònim Aquell terme que expressa un concepte més general que el d'un terme determinat. Exemple: **Edifici** és un hiperònim de **Masia**.

Hipònim Aquell terme que expressa un concepte més concret que el d'un terme determinat. La hiponímia és la relació inversa a la hipernonímia. Exemple: **Bicicleta** és un hipònim de **Vehicle**.

Inductive Logic Programming Tècniques que intenten obtenir programes en algun llenguatge lògic (habitualment **Prolog**) a partir d'un conjunt d'exemples, usant tècniques d'Aprenentatge automàtic.

Integrated Development Environment Aplicació que ofereix funcionalitats per a desenvolupar codi de forma integrada. Acostumen a incloure eines com editors de codi font guiats per la sintaxi, compilació i debugging integrat o gestors de projectes...

Intel·ligència Artificial Branca de la Informàtica que intenta obtenir programes i algorismes que imitin el comportament intel·ligent que (no sempre) mostren els humans.

Lema Forma bàsica d'un terme, la que es buscaria al diccionari. Per exemple: el lema de **catalanes** és **català**, i el de **aniríem** és **anar**.

Lematització Procés per què a un terme o conjunt de termes li són assignats els seus respectius lemes.

Lexicó Llista de mots usats per una Aplicació, en un domini o en una certa situació.

Llenguatge Natural Lenguatge parlat per les persones, per oposició als Llenguatges parlats per les màquines (*shells*, Llenguatges de Programació, etc.).

Log Traça de l'execució que deixa una aplicació. En ell s'hi poden enregistrar els events que hagin succeït, els errors que s'hagin produït, etc.

Mineria de Textos Tècniques que intenten obtenir informació útil a partir de grans col·leccions de documents.

Mode Batch Mode de treball d'una aplicació en què es processa de forma contínua i sense que intervingui l'usuari o usuària un conjunt de tasques similars.

Mode Interactiu Mode de treball d'una aplicació en què l'usuari o usuària intervé de forma habitual, i en què després de processar una tasca s'espera a que aquest o aquesta indiqui la següent.

Named Entities Termes que corresponen a un Nom Propi.

Named Pipe Mecanisme de comunicació entre processos que ofereix el Sistema Operatiu UNIX. Es tracta d'una *pipe** a què es pot accedir com un fitxer en el Sistema de Fitxers.

Ontologia Model formal de la realitat en un cert domini.

Overhead Cost addicional que suposa algun aspecte no relacionat directament amb la tasca principal sobre el cost total de processar aquesta.

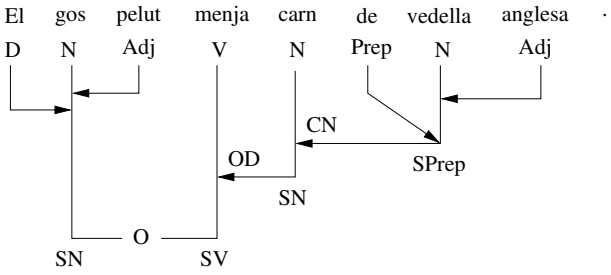
Paraula Clau Terme que s'usa en una consulta per a Recuperació d'Informació. La idea és que són les paraules que aporten el veritable significat a una oració.

Parsing Procés de llenguatge natural que obté les relacions i agrupacions sintàctiques dels diversos elements d'una oració.

Per exemple, el parsing de la frase:

- El gos pelut menja carn de vedella anglesa.

pot ser:



Passatge Fragment d'un document.

Path Seqüència dels noms dels directoris que cal recórrer en un Sistema de Fitxers per arribar a un determinat fitxer.

Patrons de Disseny Metodologies de l'Enginyeria del Software* que ofereixen una forma genèrica de dissenyar classes d'un sistema que presenti unes característiques determinades.

Pattern Matching Concordança de Patrons

Pipe Mecanisme de comunicació entre processos que ofereix el Sistema Operatiu UNIX. Es tracta d'un canal de què un o més processos poden llegir bytes i un o més, escriure'n.

Pipeline Arquitectura d'un Sistema en què els diversos components s'organitzen com processos independents i la informació flueix de l'un a l'altre.

Classificació entre 2 classes A i B

	Classificat A	Classificat B
Real A	n_{aa}	n_{ab}
Real B	n_{ba}	n_{bb}

$$\begin{aligned} precision &= \frac{n_{aa}}{n_{aa} + n_{ba}} \\ recall &= \frac{n_{aa}}{n_{aa} + n_{ab}} \end{aligned}$$

Taula A.1: Precisió i Recall

Precisió Mesura de la bondat d'una classificació. La seva definició apareix a la taula A.1.

Processament de Llenguatge Natural Tècniques informàtiques, habitualment incloses dins la Intel·ligència Artificial*, que s'usen per a tractar el Llenguatge Natural*.

Raonament Automàtic Tècniques informàtiques que intenten emular de forma algorísmica els processos del pensament humà.

Recall Mesura de la bondat d'una classificació. La seva definició apareix a la taula A.1.

Reconeixement de la Parla Tècniques que intenten obtenir mots

Script En general, programa escrit en un llenguatge interpretat.

Segmentació Procés de Llenguatge Natural que intenta trobar els límits entre oracions consecutives d'un text o flux de mots.

Shell Aplicació que permet controlar aspectes del Sistema Operatiu i iniciar i aturar processos en una màquina.

Skeleton Entitat informàtica que serveix com a estructura base d'altres entitats, que l'han d'extendre a través de mecanismes concrets (per exemple, una classe skeleton que cal extendre mitjançant herència i la implementació dels seus mètodes abstractes).

Socket Mecanisme de comunicació entre processos (en la mateixa màquina o en diferents) introduït a BSD UNIX. Es tracta d'un canal de comunicació bidireccional on poden escriure's i llegir-se bytes en un sentit i en l'altre.

Stub Entitat informàtica que en substitueix una altra sense tenir exactament la mateixa funcionalitat. Pot tractar-se de *Proxies* per a sistemes distribuïts, o bé de versions més simples per a provar altres mòduls.

Suite Conjunt d'Aplicacions amb finalitats relacionades i amb mecanismes de comunicació i integració entre elles.

Synset Conjunt de termes que són sinònims per a un cert concepte. Són la base de **WordNet***.

Switch Paràmetres que es pasen per la línia de comandes a una aplicació i que modifiquen el seu funcionament per defecte.

Stemming Procés per què a un terme o conjunt de termes li són assignats els seus respectius *stems*, arrels tals que:

- Mots que provenen de la mateixa arrel tenen el mateix *stem*.

- Mots que no provenen de la mateixa arrel tenen diferent *stem*.

Els *stem*s no tenen perquè correspondre a termes reals. De fet, la lemmatització* és un tipus concret de stemming que utilitza els lemes com a *stem*.

Standard Template Library (STL) Llibreria estàndard del llenguatge **C++** que conté, entre d'altres recursos, Estructures de Dades, Algorismes i funcions per a Entrada/Sortida.

Tag Categoria morfosintàctica que té un cert terme en un context determinat.

Tagging Procés de Llenguatge Natural per què a un terme o conjunt de termes li és assignat el seu tag*.

Templeta Plantilla que cal completar.

Threshold Valor llindar que s'imposa com a criteri en algun tipus de decisió.

Throughput Nombre de tasques acabades per unitat de temps.

Token Unitat lèxica de processament d'un text. Correspon a un mot, un signe de puntuació, un número, etc.

Tokenització Procés per què un text és dividit en tokens.

Top Concept Ontology Ontologia de conceptes generals (com ara Event, Entitat, Acció o Propietat) amb què poden relacionar-se els synsets*de WordNet*.

XML Un llenguatge de descripció de llenguatges. Els documents en llenguatges descrits amb XML estan formats per una sèrie de marques que delimiten seccions del document i el doten d'una estructura en arbre.

Wildcard Caràcter especial que pot substituir un o més caràcters qualssevol. Els més típics són el **?** i el $*$ dels *shells**.

WordNet Base de dades lèxica d'ús habitual en el domini del PLN. Està basada en el concepte de synset*, i recull les diverses relacions que poden existir entre ells (sinonímia, hiponímia, hiperonímia, etc.).

Wrapper Aplicació, classe o alguna altra entitat informàtica que no ofereix funcionalitat pròpia, sino que només canvia o adapta la interfície d'una altra entitat que és qui realment ofereix la funcionalitat.

Apèndix B

Representació d'Informació de Tipus Lingüístic

B.1 Requeriments

A l'hora de dissenyar la llibreria de classes i el format de fitxer per a la representació de les dades usades al llarg de tot el sistema de CR, es van proposar els objectius següents:

- **Dissenyar unes classes no restringides al sistema que s'estava desenvolupant**

  Es volia que les classes fossin prou genèriques com perquè altres aplicacions de PLN poguessin aprofitar-les, encara que no fossin de CR.

- **Dissenyar unes classes que permetessin ser exteses i modificades**

  Com que cada aplicació té les seves necessitats concretes, és impossible pensar en preveure-les totes de bon principi. Per aquesta raó, calia que fos possible ampliar el sistema de forma senzilla:

  - Calia que fos possible ampliar la informació emmagatzemada per a cada Element (afegir un Atribut als Tokens, afegir Relacions entre els Chunks i el Token que els fa de Cap...).
  - Calia que l'ús de les classes no anés condicionat a l'ús de cap format de fitxer concret, sinò que es pogués utilitzar el que es volgués.
  - Calia que es poguessin afegir noves classes a la jerarquia.

- **Dissenyar unes classes que permetessin representar ambigüitats**

  Encara que en el sistema a implementar inicialment no n'apareguessin, les ambigüitats apareixen en moltes aplicacions de PLN. Per aquesta raó, poder-les representar resulta un avantatge.

  Així, es va intentar que el model concebut pogués representar ambigüitats en els diversos elements (a nivell de Tagging, de Chunking, de Parsing, de Segmentació...).

- **Dissenyar unes classes que fossin fàcils d'utilitzar**

  Com que el nombre d'usuaris potencials d'aquestes classes podia ser gran, es va intentar que l'ús de les classes fos el més intuïtiu possible. Per altra banda, l'ús de patrons de disseny havia de fer que els conceptes que apareguessin al disseny fossin els d'ús habitual dins el món de l'Enginyeria del Software.

  Per altra banda, la nomenclatura de les classes i els mètodes segueix una sèrie de criteris pensats per a donar una interfície homogènia a tota la llibreria i disminuir així el temps d'aprenentatge requerit per a usar-la.

B.2 Disseny

Per aconseguir aquests objectius era bo minimitzar l'acoblament entre classes, així que es van usar en diversos punts Patrons de Disseny, dins la política general d'usar mètodes d'Enginyeria del Software.

El disseny finalment escollit és el que es descriu a l'apèndix següent.

B.3 Format de Fitxer

Pel que fa al format de fitxer, es va pensar en una continuació del format ja existent: el de fitxers de text pla on cada línia correspon a una unitat lèxica i els seus atributs. Tanmateix, la diferència és que en el nou format es pot emmagatzemar també informació jeràrquica (estructura del text en seccions, oracions, termes...) i de rel·lacions (arbres i rols sintàctics, coreferents...).

A aquest format el vam anomenar FUF (Format Unificat de Fitxer), i es descriu a la secció C.5 de l'apèndix següent.

Apèndix C

Guia d'Ús de la Llibreria Libfuf

C.1 Objectiu del Document

L'objectiu d'aquest document és descriure les classes creades per a la Integració del sistema de Question Answering del TALP, així com la seva API[1].

C.2 Descripció de les Classes

C.2.1 Models

El punt central del disseny és el Model: un Model inclou el text sobre què s'està treballant, les unitats que es creen sobre ell, així com la informació que està associada a aquestes unitats i les relacions entre elles. Segons el mòdul concret en què ens trobem, aquesta informació ens pot interessar representada d'una certa o altra manera, i aquestes representacions poden ser força dispars (des de predicats PROLOG fins l'estructura de classes descrita més avall). Per aquesta raó, són les subclasses de Model les que defineixen tota la seva funcionalitat.

La informació que es pot trobar en un model és la que apareix a la figura C.1.

Posicions Una Posició és la porció de text ocupada per un Token. Són el punt de Referència a l'hora de delimitar Segments.

Tokens Les unitats mínimes de text que considerem en els Models.

Segments Elements textuals formats per tots els Tokens entre dues Posicions. Poden ser Chunks, Oracions, Passatges, Seccions o Documents.

Arbres Elements textuals estructurats. Són una seqüència ordenada d'altres elements on cadascun té assignada una etiqueta de funció. Ara per ara només s'hi consideren Sintagmes, però podria haver-hi altres tipus. No tenen per què abarcar un rang continu de Tokens.

Atributs Tots els tipus d'Element Textual (Tokens, Segments i Arbres) poden tenir associats Atributs. Per a cada Tipus d'Element els Atributs que els seus membres poden tenir, així com el seu nom i tipus (Numèric, de Cadena, de Llista...), estan definits.

Relacions Entre dos Elements Textuals qualssevol pot haver-hi relacions, que són arestes dirigides i etiquetades per a indicar el Tipus de Relació.

La representació que es descriu en aquest document és la que hem anomenat *Avatar*[2]. En ella, cadascun dels Elements del Model està representat per un Objecte. L'Avatar, addicionalment,

[1] Application Programming Interface

[2] **Avatar**: *m.* Descens d'una deïtat a la terra i la seva encarnació en un home o un animal; encarnació, personificació. ([DIEC])

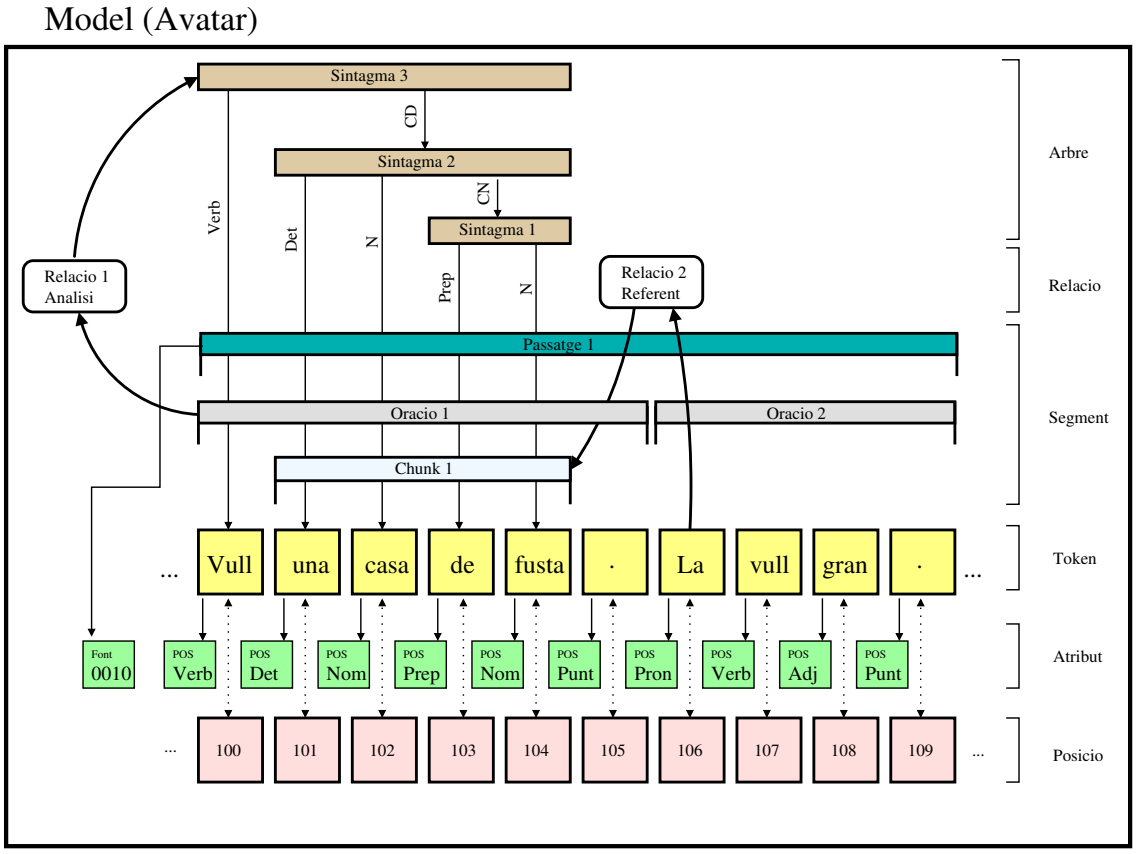


Figura C.1: Estructura d'un Model

està dissenyat seguint el patró Façana (exposat a [GH94]), i conté operacions per a treballar amb aspectes globals del Model (definició d'Atributs per als diversos Tipus, accés a un Element per identificador (veure secció C.2.2), creació de nous Elements...).

Addicionalment, hi ha una sèrie d'addicions sobre el Model original, que es descriuen a continuació.

C.2.2 Identificadors

En l'Avatar hi ha definits tota una sèrie d'identificadors que s'usen en diverses situacions:

- Cada Tipus d'Element Textual té un Identificador únic, l'anomenat *IdTipus*. Així, en la versió inicial hi ha 7 valors per a l'IdTipus: per a Tokens, Chunks, Oracions, Passatges, Seccions, Documents i Sintagmes.
- Paral·lelament, cada Element té un Identificador doble: per una banda l'IdTipus del tipus a què pertany, i per l'altra un *Id* propi, únic dins cada Tipus. D'aquesta manera, per a identificar un Element necessitem el seu IdTipus i el seu Id.
- A l'hora d'accedir als Atributs d'un Element, es pot utitzar l'accés per nom o bé obtenir un *IdAtribut*, que identifica l'Atribut dins un cert Tipus. Tots els Elements d'un mateix Tipus accedeixen als seus Atributs amb el mateix IdAtribut.
- Per últim, els Tipus d'Atribut s'identifiquen amb un *IdKin*. Aquest valor només s'usa quan s'afegeixen Atributs, per a indicar el seu Tipus (que per a distingir del tipus dels Elements anomenarem *Kin*).

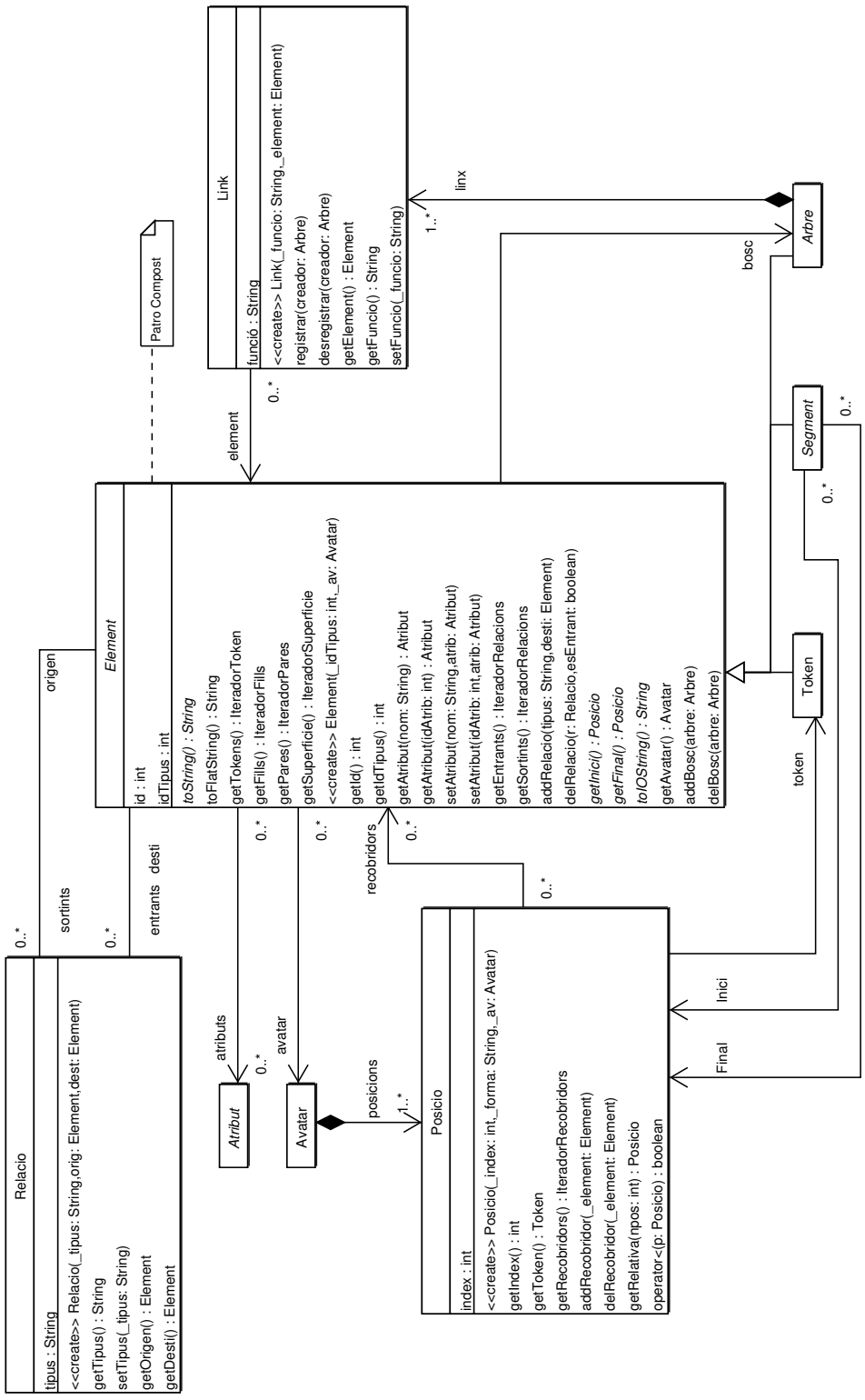


Figura C.2: Diagrama UML de les Classes de l'Avatar

C.2.3 Iteradors

Per a recórrer les diverses col·leccions s'utilitzen objectes Iterador (seguint de nou un patró de [GH94]). La idea és donar una interfície única per a simplificar la programació, i l'escollida és una simplificació de l'habitual:

- `Constructor`: Crea l'Iterador i el posa apuntant al primer element.
- `isDone():bool`: Indica si hem recorregut ja tota la col·lecció.
- `current():Tipus`: Retorna l'element actual
- `next()`: Avança l'Iterador un element.

Hi ha diversos tipus d'Iteradors, segons el que estiguem recorrent i/o com ho fem. Un ús concret és, per exemple, el d'Iteradors per a obtenir tots els Elements inclosos o que inclouen un altre.

D'aquesta manera, per a definir nous recorreguts n'hi ha prou amb afegir noves classes d'Iteradors i modificar la funció `next()`.

La jerarquia existent d'Iteradors apareix a la figura C.3.

C.2.4 Posicions

Les Posicions de l'Avatar mantenen un registre dels Elements que passen per elles, i ofereixen la funcionalitat d'obtenir tots els Elements que la cobreixen.

C.2.5 Elements

Des d'un Element es poden obtenir les Relacions que arriben i surten d'ell. A part, es poden obtenir els Iteradors per a accedir als seus Tokens, als seus Fills (elements que es troben continguts en ell), als seus Pares (elements en què es troba contingut) o a la seva Superfície (recorregut voraç que agafa a cada posició l'element més gran possible fins a recobrir-les totes).

També es pot obtenir sempre les Posicions d'Inici i Final de l'Element.

La jerarquia d'Elements apareix a la figura C.4.

C.2.6 Atributs

Es defineixen diverses classes que hereten d'Atribut per a diversos *Kin* d'Atribut. La jerarquia definida es pot veure a la figura C.5.

Per altra banda, s'utilitza el sistema de Comptatge de Referències descrit a [Me96], de manera que la compartició de valors d'Atributs pot dur-se a terme sense haver-se de preocupar per qui allivera finalment la memòria de l'Objecte[3].

C.2.7 Entrada/Sortida

Es proporciona una base per a qualsevol tipus d'Entrada/Sortida amb models de qualsevol tipus i en diversos formas. En concret es troben implementades les classes per a Serialitzar Avatars en el format FUF, dissenyat especialment per al sistema.

Les classes dissenyades apareixen a la figura C.6. Com es pot veure, s'ha utilitzat el patró Prototipus ([GH94]) per a permetre que la Configuració d'Entrada/Sortida pugui utilitzar qualsevol Escriptor o Lector, encara que no siguin els del format FUF per a Avatars. Per altra banda, el LectorFUF, per a permetre que el format FUF pugui ser importat cap algun altre tipus de Model a part del d'Avatar, utilitza el patró Constructor: a mida que llegeix les diferents parts del fitxer va cridant els mètodes corresponents del Constructor, i és responsabilitat d'aquest crear el Model corresponent, per a posteriorment retornar-lo.

[3] Gràcies a Mihai Surdeanu per donar-me la idea

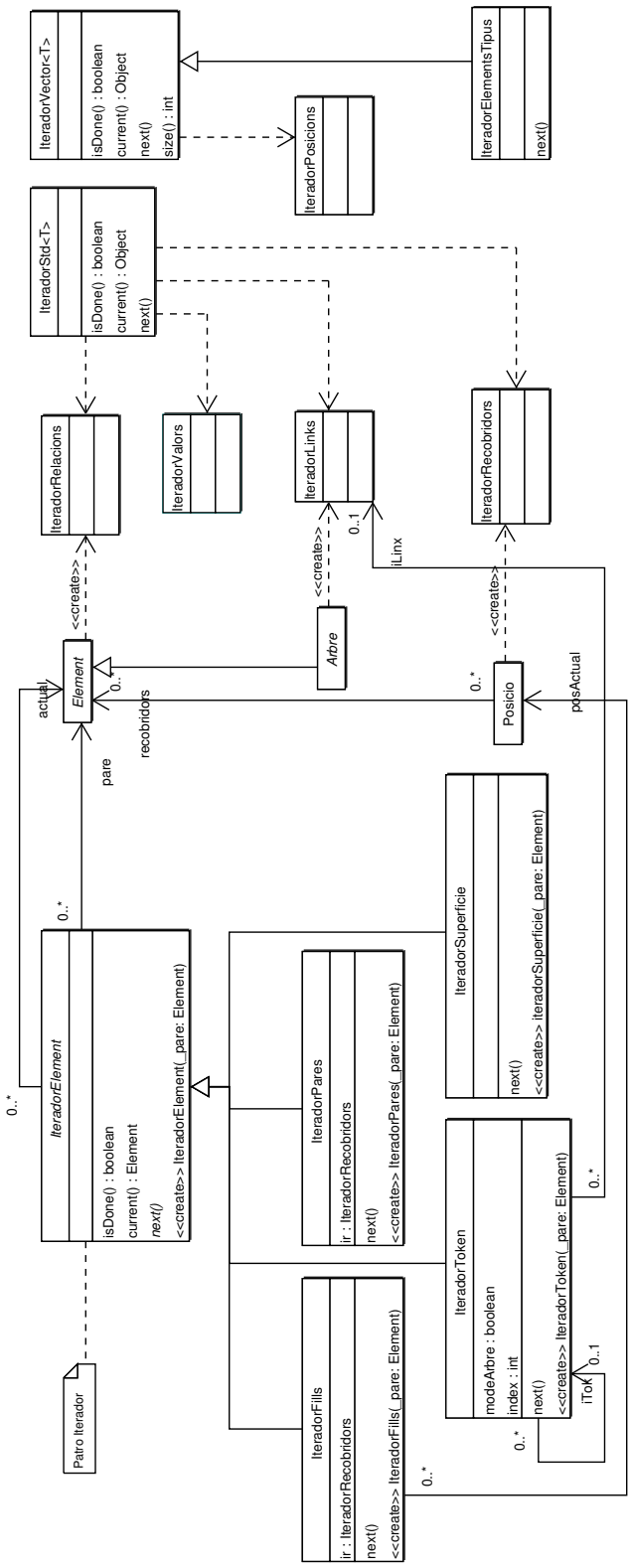


Figura C.3: Iteradors

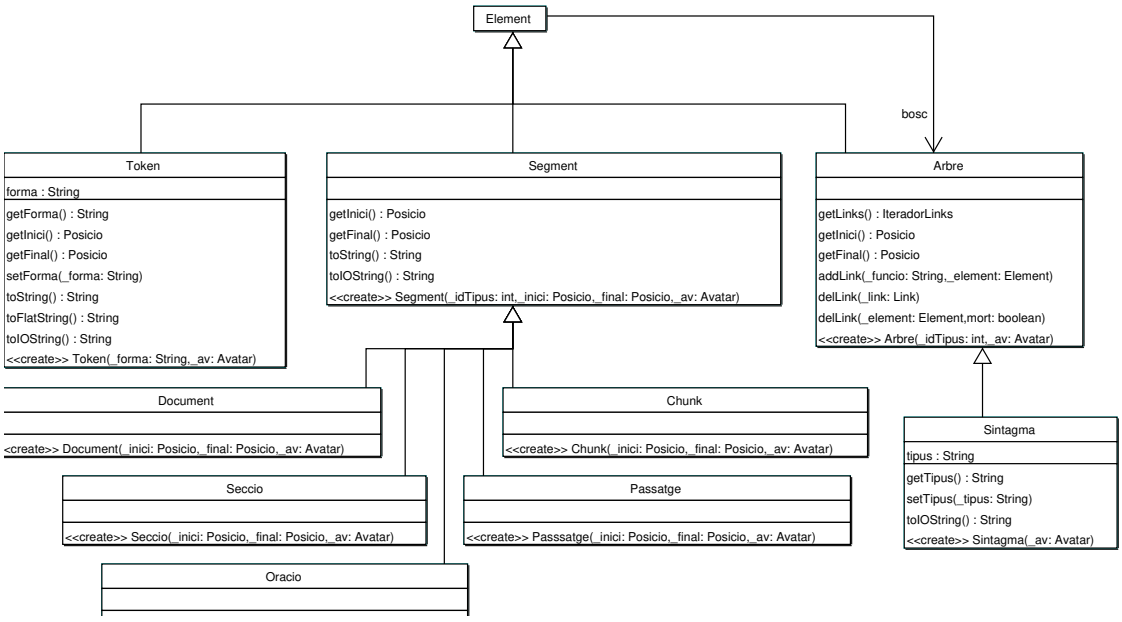


Figura C.4: Elements

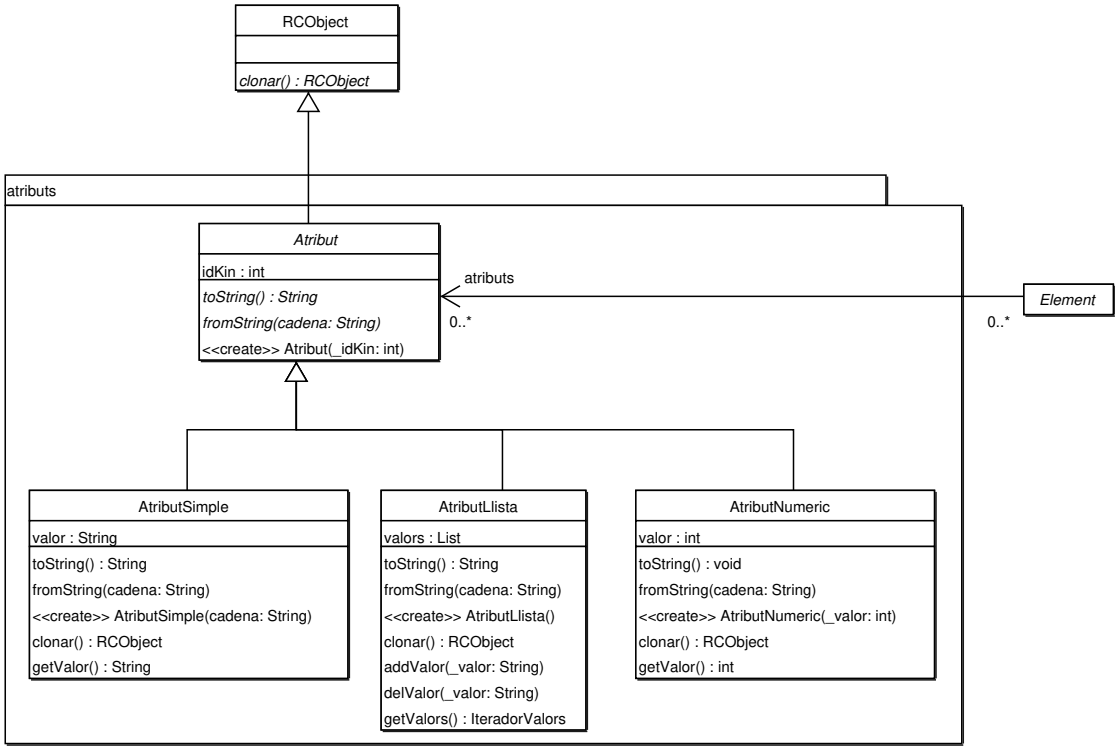


Figura C.5: Atributs

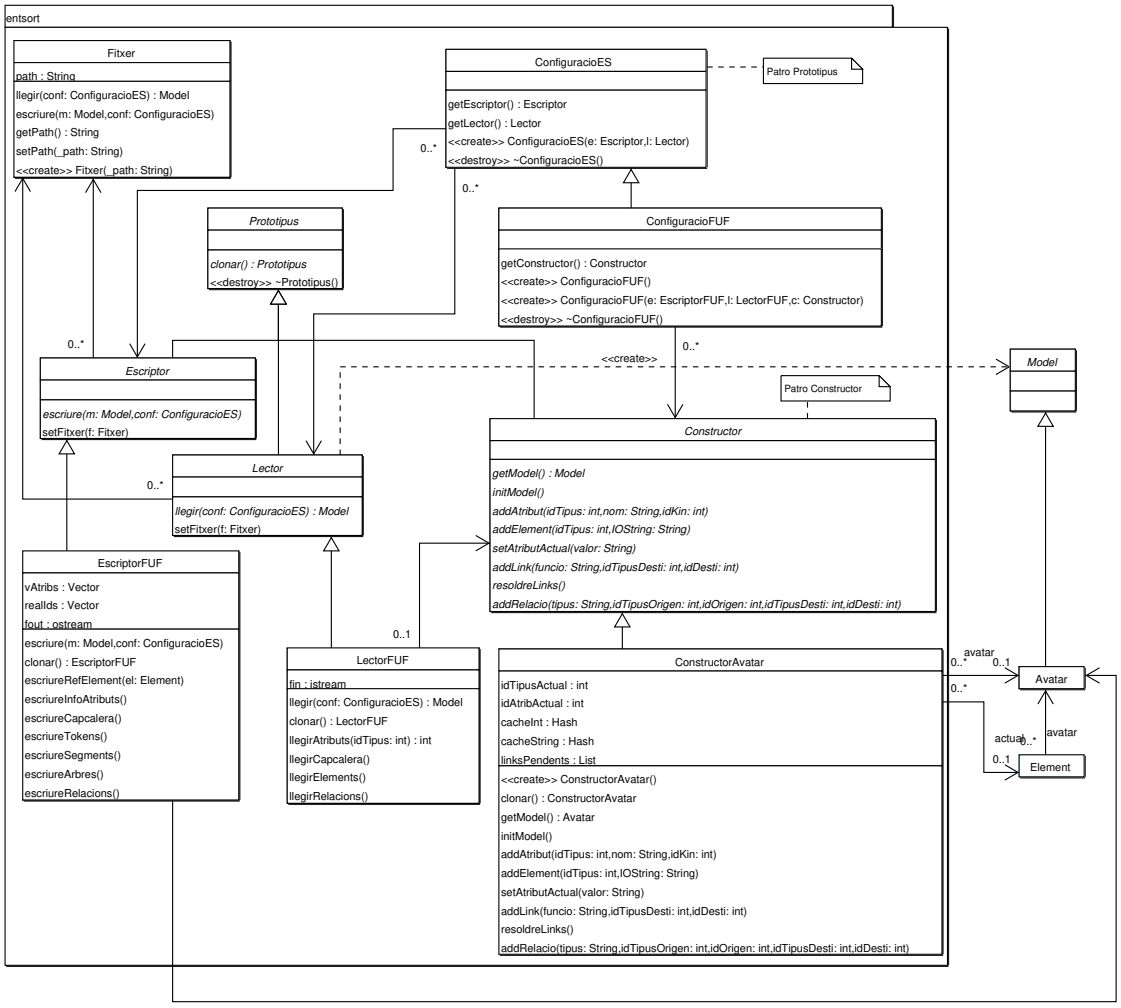


Figura C.6: Entrada/Sortida

C.2.8 Excepcions

Per últim, hi ha una classe Excepcio per a informar de les situacions anòmales que puguin esdevenir-se durant l'ús del sistema. Com a únic atribut té una string amb un missatge indicatiu.

C.3 Llibreria C++

La Llibreria ha estat implementada en C++ i provada utilitzant el compilador `gcc-3.2` de la GNU (per a informació sobre el `gcc`, vegeu [GNU]).

C.3.1 Convenis

Namespaces

Totes les funcionalitats de la llibreria es troben a l'espai de noms `libfuf`. Les classes d'atributs es troben a `libfuf::atributs` i les d'Entrada/Sortida a `libfuf::entsort`.

Ús de Majúscules i Minúscules

Els noms de les classes comencen sempre per Majúscules i els noms dels métodes ho fan en Minúscules. La resta del nom està en Minúscules, excepte la primera paraula de cada mot, si l'identificador està format per diversos mots, que torna a estar en Majúscules. Els mots estan directament juxtaposats sense cap mena de separador.

Ex: `IteradorPares`, `getElementsTipus`

En quant a les constants, estàn totes en Majúscules i amb les diverses parts separades per subratllats(_).

Ex: `T_TOKEN`, `K_NUMERIC`

Noms de Funcions

Els mètodes accessors dels atributs i relacions de les classes segueixen una sèrie de criteris:

- Els metodes consultors reben de nom `getXXX()` on `XXX` és el nom de l'atribut a consultar.

  Ex: `string Token::getForma()`, `IteradorLinks Arbre::getLinks()`.

  Com es veu, tant pot ser que es retorni un valor simple com un iterador, segons l'atribut o relació siguin únics o bé una col·lecció, respectivament.

- Per a valors simples modificables, el métode modificador rep el nom `setXXX(yyy)`.

  Ex: `void Link::setFuncio(string _funcio)`
  Ex: `void AtributNumeric::setValor(int _valor)`

- Per a col·leccions modificables, el métode per a afegir rep el nom de `addXXX(yyy)`, i el per a eliminar, `delXXX(yyy)`.

  Ex: `void Arbre::delLink(Element* _element, bool mort)`
  Ex: `Chunk* Avatar::addChunk(Posicio* _inici, Posicio* _final)`

Punters i Valors

Hi ha convencions sobre els valors de retorn i els paràmetres que involucren les classes del Sistema:

- Les Posicions, els Elements Textuals (Element, Token, Segments i Arbres) i les Relacions es reben i es passen sempre com un punter. La idea és que d'aquesta manera s'eviten còpies innecessàries d'objectes, i es manté sempre la seva identitat (per a un cert element, només n'existeix un objecte en el sistema).

- Els Iteradors es reben per valor.
- Els Atributs es reben a través d'un punter amb comptatge de referències (classe `RCPtr<T>`). El seu ús és igual al de un punter estàndard.

  A l'hora de subministrar Atributs es poden usar tant punters convencionals o `RCPtr`'s (hi ha conversió implícita de `T*` a `RCPtr<T>`). Tanmateix, si ens interessa realitzar compartició d'Atributs i guardar-ne punters fora dels Elements, és necessari que aquest punter extern sigui un `RCPtr`, ja que si no es pot donar una situació com la de la figura C.7.

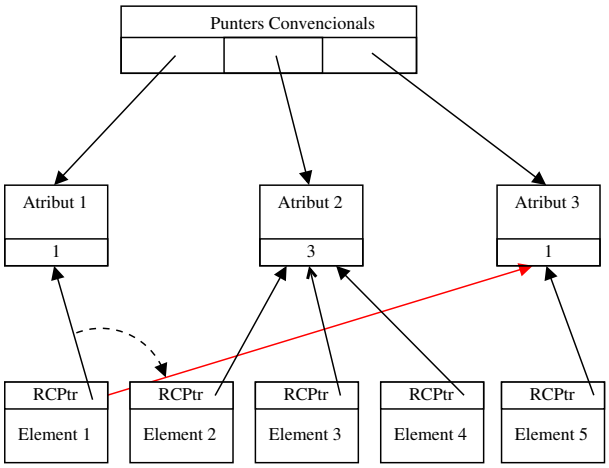


Figura C.7: Error en el Comptatge de Referències

En aquest cas, quan assignem a l'Element 1 l'Atribut 3, l'Atribut 1 passarà a tenir 0 referències (ja que els punters convencionals no compten) i s'invocarà `delete` sobre ell. Si ara assignessim a algun altre Element el punter d'Atribut 1 que hem guardat a la taula, es podria produir un accés a una posició de memòria ja alliverada i, per tant, un `segmentation fault`.

Deleció

També hi ha un criteri a l'hora de fer `delete` sobre els punters rebuts del sistema.

Quan es fa `delete` d'un punter s'actualitzen totes les associacions que involucraven l'objecte apuntat, de manera que fer `delete` és la manera d'eliminar realment un element del sistema. El `delete` es pot fer sobre punters a Relacions, Links (arestes dels Arbres), Segments o Arbres. Tanmateix, no es pot fer sobre Tokens o Posicions, ja que aquests, un cop afegits, ara per ara no es poden eliminar[4].

Sobre els `RCPtr<Atribut>` no és necessàri fer `delete`, aquest té lloc automàticament. Similarment, com que els Iteradors se'ns retornen per valor tampoc no cal alliverar-los.

Per últim, quan es fa `delete` o queda fora d'àmbit un Avatar, automàticament s'allivera tota la memòria usada pel Model.

C.4 CookBook

C.4.1 Avatar

Creació de l'Avatar

Hi ha dues maneres d'obtenir un Avatar:

[4]Intentaré arreglar això en una futura versió

1. Crear un Avatar buit:

```
Avatar av;
```

o bé

```
Avatar* av = new Avatar();
```

2. Carregar-lo d'un fitxer:

```
// Carrega de fitxer.dat amb els valors estàndard de configuracio
ConfiguracioFUF conf;
Fitxer f("fitxer.dat");

Avatar* av = dynamic_cast<Avatar*>(f.llegir(conf));
```

Afegir Elements a l'Avatar

1. Afegir un Token. Sempre s'afegeixen pel final.

```
// Afegeix el token 'prova' pel final
Token* tk = av->pushToken("prova");
```

2. Afegir un Segment. Cal donar-ne la posició inicial i final (veure secció C.4.1 sobre com obtenir posicions).

```
// tk1 i tk2 són els dos tokens entre què volem marcar un chunk
Posicio* p1 = tk1->getPosicio();
Posicio* p2 = tk2->getPosicio();

Chunk* ch1 = av->addChunk(p1, p2);
```

3. Afegir un Arbre. Es crea buit i després s'hi van afegint els Links (no cal inserir-los en ordre, s'ordenen automàticament).

```
// Volem crear un Sintagma amb el Token tk1, el Chunk ch1
// i el Sintagma s1
Sintagma* s2 = av->addSintagma();
s2->addLink("Det", tk1);
s2->addLink("N", ch1);
s2->addLink("CN", s1);
```

4. Afegir una Relacio entre dos Elements. A partir de l'Element orígen s'afegeix la relació envers el destí, tot indicant-ne el tipus.

```
// Volem fer una relacio del Token tk1 cap al Chunk ch1, indicant
// que es una relació de CoReferència
tk1->addRelacio("Coreferent", ch1);
```

Gestio d'Atributs

1. Afegir un atribut a un tipus. S'ha d'indicar a l'Avatar per a quin tipus d'Element és l'Atribut, el seu nom, i el *Kin* de l'Atribut. Els tipus d'Element són constants definides a `Avatar`, i els *Kin*, definides a `Atribut`.

```
// Afegir un Atribut Numèric de nom UT als Tokens
int idAtribUT = av->addAtribut(Avatar::T_TOKEN, "UT", Atribut::K_NUMERIC);
```

2. Obtenir l'idAtrib d'un Atribut. S'ha d'indicar el Tipus d'Element i el Nom de l'Atribut.

```
// Obtenir l'idAtrib de l'Atribut UT dels Tokens
int idAtribUT = av->getIdAtrib(Avatar::T_TOKEN, "UT");
```

3. Obtenir l'idKin d'un Atribut. S'ha d'indicar el Tipus d'Element i l'idAtrib.

```
// Obtenir l'idKin de l'Atribut UT dels Tokens
int idKinUT = av->getIdKin(Avatar::T_TOKEN, idAtribUT);
```

4. Eliminar un Atribut. Cal donar el Tipus d'Element i l'idAtrib o bé el Nom. Un cop eliminat, un Atribut es pot restaurar tornant a afegir un Atribut amb el mateix Nom i idKin, o bé amb el mateix idAtrib. La raó que l'idKin hagi de ser el mateix és que els Atributs esborrats no desapareixen immediatament, sino senzillament queden marcats com a esborrables, per a no haver de recórrer tots els Elements del tipus. D'aquesta manera, quan es restaura l'Atribut el que es fa una Resurrecció dels valors ja existent (així que esborrar i tornar a afegir no buida tots els valors...).

```
// Eliminem l'atribut UT
av->delAtribut(Avatar::T_TOKEN, idAtribUT);
// o bé
av->delAtribut(Avatar::T_TOKEN, "UT");

// Ara el Restaurem
av->addAtribut(Avatar::T_TOKEN, idAtribUT);
// o be
av->addAtribut(Avatar::T_TOKEN, "UT", Atribut::K_NUMERIC);

// Si haguessim fet...
av->addAtribut(Avatar::T_TOKEN, "UT", Atribut::K_SIMPLE);
// -> Excepcio!
```

5. Obtenir tota la informació sobre els Atributs d'un Tipus. Necessitem un vector de l'estructura InfoAtribut.

```
// Vector on guardarem la informacio
// Es un vector STL convencional
vector<InfoAtribut> vAtribs;

// Li ho demanem al Avatar
av->getInfoAtributs(Avatar::T_TOKEN, vAtribs);
```

Obtenció de Posicions

1. Obtenció d'una Posicio per índex.

```
// Obtenim la Posició número 2
Posicio* pos2 = av->getPosicio(2);
```

2. Obtenció d'una Posició relativa a una altra.

```
// Obtenir la Posició següent a la pos2
Posicio* pos3 = pos2->getRelativa(+1);
```

3. Obtenció de les Posicions d'inici o final d'un Element.

```
// Obtenir on comença i acaba el Chunk ch1
Posicio* iniciChunk = ch1->getInici();
Posicio* finalChunk = ch1->getFinal();
```

4. Obtenció de Totes les Posicions. Ens venen en forma d'Iterador.

```
// Obtenir un Iterador i recórrer totes les posicions
IteradorPosicions ip = av->getPosicions();

while(!ip.isDone()) {
  Posicio* actual = ip.current();
  // Fer alguna cosa amb actual
  ip.next();
}
```

Accés a Elements

1. Accedir a tots els Elements d'un Tipus. Necessitem l'idTipus.

```
// Obtenir tots els Chunks
IteradorElementsTipus iet = av->getElementsTipus(Avatar::T_CHUNK);
```

2. Obtenir un Element donat el seu idTipus i el seu id.

```
// Obtenir l'Oració de Id 3
// Necessitem fer un cast ja que el valor retornat és un Element*
Oracio* or = static_cast<Oracio*>(
                 av->getElementTipus(Avatar::T_ORACIO, 3));
```

Mètodes Interns

La resta de mètodes d'Avatar (`nouElement`, `elementEsborrat` i `addRelacio`) són per a mantenir les estructures de Dades Internes sincronitzades i no haurien d'usar-se des de l'Exterior.

Queden dos mètodes, l'estàtic `esTipusContigu`, que donat un idTipus diu si el tipus és Contigu (no és un Arbre), i `getTipus`, que serveix per a distingir entre diversos Models a l'hora d'escriure'ls. El seu ús, encara que no és en absolut perillòs, no és massa útil des de fora de la llibreria.

C.4.2 Posicions

Índex de la Posició

```
// Obtenir l'Índex de la Posició pos1
int index = pos1->getIndex();
```

Obtenció dels Recobridors

Per a obtenir els Elements que passen per una certa Posició, hem de recórrer un IteradorRecobridors. Els Elements es troben ordenats per posició d'acabament, primer els que acaben més lluny. L'Element que no es troba en aquest recorregut és el Token de la Posició, que s'ha d'obtenir després.

```
// Obtenir els Elements de la Posició pos1
IteradorRecobridors ir = pos1->getRecobridors();

// Després de recórrer ir, cal agafar el Token
Token* tk = pos1->getToken();
```

Comparació de Posicions

Com que les Posicions són úniques, per a saber si les posicions retornades per dues crides diferents són la mateixa, n'hi ha prou amb comparar els punters.

Per altra banda, hi ha un operador de menor definit entre Posicions, i una funció `menor` per a realitzar la mateixa comparació però amb els punters a les Posicions:

```
bool menor(Posicio* p1, Posicio* p2) {
  return *p1 < *p2;
}
```

L'ordre entre Posicions és l'ordre entre els seus índexos.

Mètodes Interns

`addRecobridor` i `delRecobridor` són mètodes interns i no haurien de cridar-se des de l'exterior.

C.4.3 Elements

Conversió a Cadenes

1. Conversió a cadena. Mostra el Text de l'Element de forma formatada (amb les Funcions si es tracta d'un Arbre, o entre claudàtors si és un Segment).

   ```
   // Obtenir una Representació del Sintagma s1
   string text = s1->toString();
   ```

2. Conversió a cadena de Text Plà. Agafa els Tokens que abarca l'Element i els juxtaposa separats per espais.

   ```
   // Obtenir el Text Pla de l'Oracio or1
   string textPla = or1->toFlatString();
   ```

3. Conversió a cadena per E/S. Emmagatzema la informació específica de cada tipus d'Element (no Atributs) en una cadena per a la Serialitzacio. És poc útil si no és per a escriure en algun suport permanent.

   ```
   // Obtenir la Informació Específica del Token tk1
   // (en aquest cas, la forma)
   string cadenaIO = tk1->getIOString();
   ```

Accés als Atributs

Tant per a accedir com per a modificar els Atributs d'un Element es pot usar l'idAtrib com el nom de l'Atribut. En cas de consultar un Atribut que no està definit per a l'Element, s'obté un RCPtr nul (que es pot detectar amb l'operador !).

```
// Posem l'Atribut POS del Token tk1 a "NP00P0"
tk1->setAtribut("POS", new AtributSimple("NP00P0");

// Ara obtenim l'atribut i assignem el mateix a un altre Token tk2
RCPtr<Atribut> atrib = tk1->getAtribut("POS");
tk2->setAtribut(idAtribPOS, atrib);

// Si ara assignem a tk1 i tk2 un altre valor, l'Atribut s'esborra
// automàticament. Cal perdre també la referència des de atrib
atrib = 0;
tk1->setAtribut(idAtribPOS, new AtributSimple("NP00A0");
tk2->setAtribut(idAtribPOS, tk1->getAtribut(idAtribPOS));
// Ara ha tingut lloc el delete

// Un atribut que tk1 que potser sigui nul
atrib = tk1->getAtribut("Lema");
if (!atrib)
    cout << "Lema no definit" << endl;
```

Accés a les Relacions

Hi ha dos mètodes que ens retornen un Iterador per a recórrer totes les relacions d'un Element. Un ens permet obtenir les relacions que surten de l'Element (és l'orígen) i l'altre les que hi entren (és el destí).

```
// Escriure totes les relacions que surten del Passatge pas1
IteradorRelacions ir = pas1->getSortints();
while (!ir.isDone) {
    Relacio* r = ir.current();
    cout << "->" << r->getTipus() << ':'
         << r->getDesti()->toString() << endl;
    ir.next();
}

// Podem repetir el mateix amb les entrants
ir = pas1->getEntrants();
// ...
```

Recorreguts

Hi ha una sèrie d'Iteradors predefinits que permeten recórrer els Elements que tenen una certa relació de continència amb algun altre Element. En concret, i com ja havíem comentat a C.2.5, estan definits els Iteradors:

IteradorToken Recorre tots els Tokens que cobreix l'Element.

IteradorPares Recorre els Elements que inclouen l'Element.

IteradorFills Recorre els Elements que l'Element inclou.

IteradorSuperfície Recorre els Elements que l'Element inclou, però triant-ne només un per a cobrir cada posició.

L'algorisme que segueix és:

1. Ens col·loquem a la posició d'inici de l'Element pare.
2. Triem l'Element més gran possible que comenci en aquella posició i que no acabi més enllà que on acaba el pare (i que no sigui el pare).
3. Aquest és l'element actual.
4. Quan ens demanen el següent, ens col·loquem a la posició següent a on acaba l'Element actual.
5. Si estem ja fora de les posicions cobertes pel pare, ja hem acabat. Si no, tornem al pas 2.

D'aquesta manera podem realitzar un recorregut per la Superfície d'un Element.

Encara que aquests Iteradors es poden construir passant per paràmetre al Constructor l'Element de referència, s'inclouen a la classe Element mètodes per a obtenir-los directament: `getTokens()`, `getPares()`, `getFills()` i `getSuperficie()`.

```
// Volem recórrer tots els Fills de l'Oracio or1
IteradorFills if = IteradorFills(or1);
// o be
IteradorFills if = or1->getFills();
```

C.4.4 Atributs

Accés als valors

1. Accés com a cadena.

   ```
   // Mostrar per pantalla la POS del token tk1
   cout << t1->getAtribut("POS")->toString();
   ```

2. Accés pel tipus que correspon. És necessari fer un casting previ del punter intern a l'RCPtr per a poder accedir al métode `getValue` de les classes `AtributSimple` i `AtributNumeric`.

   ```
   // Obtenir en un enter la UT del token tk1
   AtributNumeric* atrib = static_cast<AtributNumeric*>(
                              tk1->getAtribut("UT").getPunter());
   int ut = atrib->getValor();
   ```

3. Accés a una AtributLlista. Rebem un IteradorValors.

   ```
   // Processar els Valors de l'Atribut Synsets del token tk1
   AtributLlista* atrib = static_cast<AtributLlista*>(
                             tk1->getAtribut(idAtribSynsets).getPunter());
   IteradorValors iv = atrib->getValors();
   while (!iv.isDone()) {
       string actual = iv.current();
       // Fem el que toqui
       iv.next();
   }
   ```

C.4.5 Entrada/Sortida

Obtencio d'una Configuracio

1. Obtenció d'una configuració per defecte.

```
// Obtenim la ConfiguracioFUF per defecte
ConfiguracioFUF conf;
```

2. Obtenció d'una configuració amb un lector i un escriptor concret.

```
// Obtenim una configuració que usa les classes ElMeuEscriptor i
// ElMeuLector
ConfiguracioES conf(new ElMeuEscriptor(), new ElMeuLector());
```

3. Obtenció d'un fitxer

```
// Obtenim el fitxer "dades.txt"
Fitxer fitx("dades.txt");
```

4. Escriptura d'un Avatar en un fitxer

```
// Escrivim l'Avatar *av al fitxer fitx
fitx.escriure(av, conf);
```

5. Càrrega d'un Avatar d'un fitxer:

```
// Carrega d'un Avatar de fitx
Avatar* av = dynamic_cast<Avatar*>(fitx.llegir(conf));
```

C.5 El Format FUF

Els fitxers FUF són fitxers de text pla, i consten de les següents parts:

1. Una capçalera, formada per la línia `@@libfuf`, i per una linia `@@versio <x.y>`, indicant la versió del format (per a indicar possibles extensions). Actualment, s'indica la versió 1.0.
2. Per a cada tipus de element:
   (a) La capçalera, una línia que consta de:
   - L'identificador que comença la secció corresponent al tipus (`@@token`, `@@chunk`, `@@oracio`, etc.).
   - El nombre d'elements d'aquest tipus que apareixen al fitxer.
   - El nombre d'atributs que estan definits per als elements d'aquest tipus.
   - Per a cada atribut, el nom i l'*idKin*.

   Tots aquests elements estan separats per espais.
   (b) Per a cada element. una línia amb:
   - La informació particular de l'element. Per a un token, serà la seva forma; per a un segment, els índexs de les posicions d'inici i de final; i per a un sintagma, el seu tipus.
   - Els valors dels diferents atributs de l'element, separats per espais.
   - En el cas dels arbres, en la línia següent, la llista amb els seus links, indicant per a cadascun la seva funció i l'idTipus i id de l'Element cap al que apunten. La llista acaba amb un link de funció amb valor * i idTipus i id amb valor -1

3. Per a les relacions:
   (a) La capçalera, formada per la línia `@@relacio`.
   (b) Per a cada relacio. una línia amb el tipus de relació i els idTipus i id dels Elements d'orígen i destí.
   (c) Aquesta llista acaba amb una línia amb tipus de relació amb valor * i idTipus i id d'origen i desti amb valor -1.
4. La resta del fitxer s'ignora.

Bibliografia

[AB98] Antonietta Alonge, Francesca Bertanga, Laura Bloksma, Salvador Climent, Wim Peters, Horacio Rodríguez, Adriana Roventini, Piek Vossen, **The Top-Down Strategy for Building EuroWordNet: Vocabulary Coverage, Base Concepts and Top Ontology**, Computer and Humanities 32. Kluwer Academic Publishers, 1998.

[AC98] Jordi Atseries, Josep Carmona, Irene Castellón, Sergi Cervell, Montserrat Civit, Lluís Màrquez, Maria Antònia Martí, Lluís Padró, Roberto Placer, Horacio Rodríguez, Mariona Taulé, Jordi Turmo, **An Environment for Morphosyntactic Processing of Unrestricted Spanish Text**, Proceedings of the 1st International Conference on Language Resources and Evaluation. LREC-98, Granada, 1998.

[Br00] Thorsten Brants, **TNT, A Statistical Part-of-Speech Tagger**, Proceedings of the 6th ANLP-NAACL. Seattle, 2000.

[CC04] Xavi Carreras, Isaac Chao, Lluís Padró, Montserrat Padró, **FreeLing: An Open-Source Suite of Language Analyzers**, Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04). Lisboa, 2004.

[CL94] David Ngi Chin, Marc Luria, James H. Martin, James Mayfield, Robert Wilensky, Dekai Wu, **The Berkeley UNIX Consultant Project**, 1994.

[CM00] Bernardo Magnini, Gabriela Cavaglia, **Integrating Subject Field Codes into WordNet**, Proceedings LREC. Atenes, 2000.

[CM02] Xavier Carreras, Lluís Màrquez, Lluís Padró, **Named Entity Extraction Using Adaboost**, Proceedings of the 6th Conference on Computational Natural Language Learning (CoNLL 2002). Shared Task Contribution. Taipei, 2002.

[DIEC] **Diccionari de l'Institut d'Estudis Catalans**, `http://pdl.iec.es/entrada/diec.asp`

[FM04a] Daniel Ferrés, Marc Massot, Montserrat Padró, Horacio Rodríguez, Jordi Turmo, **Automatic Classification of Geographic Named Entities**, Enviat a 4th LREC, Lisboa 2004.

[FM04b] Daniel Ferrés, Marc Massot, Montserrat Padró, Horacio Rodríguez, Jordi Turmo, **Automatic Building Gazetteers of Co-referring Named Entities**, Enviat a 4th LREC, Lisboa 2004.

[FM03] Daniel Ferrés, Marc Massot, Horacio Rodríguez, **QA UdG-UPC System at TREC-12**, 2003.

[GH94] Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides, **Design Patterns. Elements of Reusable Object-Oriented Software**, Addison-Wesley, 1994.

[GNU] **GNU Compiler Collection**, `http://gcc.gnu.org`

[HR91] Otthein Herzog, Claus-Rainer Rollinger, **Text Understanding in LILOG: Integrating Computational Linguistics and Artificial Intelligence**, vol 546 de Lecture Notes in Computer Science, Springer-Verlag, 1991.

[HT99] Intenet Engineering Task Force, **Hypertext Transfer Protocol – HTTP/1.1**, 1999.

[Ku93] Julian Kupiec, **MURAX: A Robust Linguistic Approach for Question Answering Using an On-Line Encyclopedia**, 1993.

[Le80] Wendy Lehnert, **Question Answering in Natural Language Processing**, a **Natural Language Question Answering Systems**, Carl Hansen Verlag editor, 1980.

[Li98] Dekang Lin, **Dependency-based Evalutation of Minipar**, Proceedings of the Workshop on Evaluation of Parsing Systems, LREC-98. Granada, 1998.

[Luc] Apache Jakarta Project, **Lucene**, `http://jakarta.apache.org/lucene/docs/index.html`.

[Me96] Scott Meyers, **More Effective C++**, Addison-Wesley, 1996.

[MH99] Dan Moldovan, Sanda Harabagiu, Marius Pasca, Rada Mihalcea, Richard Goodrum, Roxana Gîrju y Vasile Rus, **LASSO: A Tool for Surfing the Answer Net**, a [Tr99], 1999.

[New] Jan Newmarch, **Jan Newmarch's Guide to JINI Technologies**, `http://jan.netcomp.monash.edu.au/java/jini/tutorial/Jini.html`

[Po80] Martin Porter, **An Algorithm for Suffix Stripping**, 1980.

[PorW] Martin Porter, **Porter Stemming Algorithm**, `http://www.tartaurus.org/~martin/PorterStemmer`.

[Pu00] Matthew Purver, **Simplistic Question Answering**, Tesi Doctoral, Universitat de Cambridge, 2000

[PW80] Fernando Pereira, David H. D. Warren, **Definite clause grammars for language analysis – a survey of the formalism and a comparison with augmented transition networks**, a Artificial Intelligence, 1980.

[Qu93] Ross Quinlan, **Foil: A Midterm Report**, Proceedings of the 6th European Conference on Machine Learning, Springer Verlag, 1993.

[Sa89] Gerard A. Salton, **Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer**, Addison-Wesley, 1989.

[TW97] Andrew S. Tanenbaum, Albert S. Woodhull, **Operating Systems: Design and Implementation**, Prentice-Hall, 1997.

[Tr99] **NIST Special Publication 500-246: The Eighth Text Retrieval Conference**, 1999.

[Tr01] **NIST Special Publication 500-250: The Tenth Text Retrieval Conference**, 2001.

[Tr03] **NIST Special Publication 500-255: The Twelfth Text Retrieval Conference**, 2003.

[Tu02] Jordi Turmo, **An Information Extraction System Portable to New Domains**, Tesi Doctoral, UPC, 2002.

[Vi02] José Luís Vicedo, **SEMQA: Un Modelo Semàntico Aplicado a los Sistemas de Búsqueda de Respuestas**, Tesi Doctoral, Universitat d'Alacant, 2002.

[Vo99] Ellen M. Voorhees, **The TREC-8 Question Answering Track Report**, a [Tr99], 1999.

[Vo01] Ellen M Voorhees, **Overview of the TREC-10 Question Answering Track**, [Tr01], 2001.

[Vo03] Ellen M Voorhees, **Overview of the TREC 2003 Question Answering Track**, a [Tr03], 2003.

[Win] Steve Winikoff, **Question Answering Links**, `http://alcor.concordia.ca/~smw/home/qa.html`.

[WM99] Ian H. Witten, Alistair Moffat, Timothy C. Bell, **Managing Gygabytes**, Morgan Kauffman, San Francisco, 1999.

[Wn98] **WordNet: An Electronic Lexical Database**, Editat per Christiane Fellbaum, MIT Press, 1998.