ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΒΙΟΪΑΤΡΙΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

Δομημένη εξαγωγή πληροφοριών από σελίδες ιατρικών και διατροφικών δεδομένων με χρήση του Information Extraction

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Παντερής Γεώργιος

Επιβλέπων: Δημήτριος Κουτσούρης
Καθηγητής ΕΜΠ

Αθήνα, Ιούλιος 2014

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΒΙΟΪΑΤΡΙΚΗΣ ΤΕΧΝΟΛΟΓΙΑΣ

Δομημένη εξαγωγή πληροφοριών από σελίδες ιατρικών και διατροφικών δεδομένων με χρήση του Information Extraction

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Παντερής Γεώργιος

Επιβλέπων: Δημήτριος Κουτσούρης
Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 21η Ιουλίου 2014.

Δ. Κουτσούρης Καθηγητής	Δ. Φωτιάδης Καθηγητής	Γ. Ματσόπουλος Επικ. Καθηγητής

Αθήνα, Ιούλιος 2014

Παντερής Γεώργιος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Αντικείμενο της παρούσας διπλωματικής εργασίας αποτελεί η μελέτη και εφαρμογή μεθόδων για την εξαγωγή πληροφοριών (Information Extraction, IE) από κείμενα σχετικά με ιατρικοφαρμαευτικά και διατροφικά δεδομένα και τη δομημένη παρουσίασή τους σε μορφή html πινάκων. Γίνεται, επίσης, χρήση ενός λογισμικού που διατίθεται ελεύθερα στους χρήστες (GATE), καθώς και ενός συνόλου κανόνων και γραμματικών (JAPE) που υποβοηθούν στον εντοπισμό των δεδομένων που ζητούνται. Παρουσιάζεται, παράλληλα, μία εφαρμογή (pipeline) που είναι σχεδιασμένη κατάλληλα για να τρέχει πάνω σε ένα σύνολο εγγράφων (corpus) και να δημιουργεί ένα σύνολο σχολιασμών (annotations). Τέλος, εφαρμόζεται ένα στάδιο που λαμβάνει τα αποτελέσματα της εφαρμογής και τα κατηγοριοποιεί στην επιθυμητή μορφή. Η εργασία στο συνολό της περιλαμβάνει πέρα από ένα γενικό θεωρητικό υπόβαθρο, τη χρήση ειδικών διαδραστικών περιβαλλόντων (Integrated Development Environment, IDE) απαραίτητων για την επίτευξη του αρχικού στόχου.

Στο κεφάλαιο 1, γίνεται μια γενική εισαγωγή σε τρόπους επεξεργασίας φυσικών γλωσσών (Natural Language Processing, NLP) εισάγωντας το γενικότερο πλαίσιο στο οποίο θα κινηθεί η εν λόγω εργασία. Προσδιορίζονται, επίσης, και έννοιες απαραίτητες για τη συνέχεια, όπως αυτή της εξαγωγής δεδομένων (IE).

Στο κεφάλαιο 2, παρουσιάζεται το κυριότερο θεωρητικό υπόβαθρο της εργασίας, καθώς περιλαμβάνει την ανάλυση του GATE, ενός συστηματος λογισμικού στο οποίο θα στηριχθεί η εξαγωγή των δεδομένων μας. Περιλαμβάνει, ακόμα, και την ανάλυση του διαδραστικού περιβάλλοντος του GATE (GATE Developer) και του IE συστήματός του (ANNIE), που θα χρησιμοποιηθούν στη συνέχεια.

Στο κεφάλαιο 3, περιγράφεται σε θεωρητικό επίπεδο η γλώσσα που παράγει τους σχολιασμούς κειμένων με χρήση ειδικών κανόνων και γραμματικών. Αποτελεί ένα βασικό εργαλείο για τη δημιουργία της εφαρμογής που θα εξάγει τα δεδομένα.

Στο κεφάλαιο 4, προσδιορίζεται ο τρόπος σύνδεσης όλων των ανώτερων στοιχείων και πληροφοριών για τη δημιουργία της εφαρμογής που επιτυγχάνει τον τελικό στόχο. Αναφέρεται, δηλαδή, σε ένα αναλυτικό βαθμό ο τρόπος σχεδιασμού της εφαρμογής, παρουσιάζοντας ταυτόχρονα τμήματα κώδικα προγραμματισμού της και αποτελέσματα από την εκτελεσή της.

Στο κεφάλαιο 5, γίνεται μία γενική σύνοψη της εργασίας , προβολή συμπερασμάτων που προέκυψαν, καθώς και προτροπή νέων ιδεών για μελλοντική ανάπτυξη της δουλειάς που παρουσιάστηκε.

Λέξεις Κλειδιά

Επεξεργασία φυσικής γλώσσας, Εξαγωγή πληροφοριών, γενική αρχιτεκτονική για επεξεργασία κειμένων, μηχανή δημιουργίας μοτίβων σχολιασμών σε Java, σχολιασμός, σύνολα εγγράφων, Εξαγωγή πληροφοριών (IE), σύστημα εξαγωγής πληροφοριών του GATE

Abstract

The purpose of this diploma thesis is the design and implementation of methods that achieve Information Extraction (IE) from texts, relevant to biomedical and nutritional data and their structured presentation in html tables. Additionally, an open-source software for text engineering (GATE) is used, and a set of rules and grammars (JAPE) to assist in identifying the requested data. Furthermore, an application (pipeline) is utilized that is designed especially to run on a set of documents (corpus) and create a set of annotations that facilitate superior work. A final stage is applied, that takes the results of the above application and categorizes them in a desired form. The document includes apart from a theoretical background review of the basic concepts involved in the aforementioned tool flow, the description of the use of special Integrated Development Environments (IDEs) necessary to achieve the original goal.

In Chapter 1, a general introduction is made, covering the field of Natural Language Processing (NLP), providing motivation for this work. Moreover, lots of necessary concepts, such as the Information extraction (IE) are determined, critical for the comprehension of the rest of the chapters.

Chapter 2 presents the main theoretical background of the entire diploma thesis, as it involves the analysis of GATE, a general architecture of text engineering which will support the information extraction of our data. It also includes the analysis of the integrated development environment of GATE (GATE Developer) and the information extraction system of GATE (ANNIE), which will be subsequently used.

Chapter 3 describes the way of creating text annotations using the Java Annotation Pattern Engine (JAPE). It is an essential tool for the creation of our application which will perform the IE.

In Chapter 4, we analyze the way of connecting all data and information mentioned, to create the application that achieves the ultimate goal. Additionally, a detailed process of the application design is highlighted, while presenting snippets of its code and results of its execution.

In Chapter 5, the diploma thesis is summarised, giving conclusions and introducing new ideas for further development of the presented work.

Keywords

Natural language processing (NLP), General Architecture for Text Engineering (GATE), Java Annotation Pattern Engine (JAPE), annotation, corpus, Information Extraction (IE), A Nearly-New Information Extraction system (ANNIE)

Ευχαριστίες/Acknowledgements

Για την εκπόνηση της παρούσας διπλωματικής εργασίας θα ήθελα να εκφράσω τις ειλικρινείς μου ευχαριστίες προς τον επιβλέποντα καθηγητή κ. Δ. Κουτσούρη ο οποίος εμπιστεύθηκε στο πρόσωπο μου την ανάθεση αυτού του επιστημονικού έργου. Επιπλέον, αυτή η εργασία δεν θα είχε έλθει εις πέρας χωρίς την πολύτιμη βοήθεια και καθοδήγηση των υποψήφιων διδακτόρων Χάρη Τσίρμπα και Αθανάσιο Αναστασίου που βοήθησαν τα μέγιστα με τις συμβουλές, τις γνώσεις και την υπομονή τους.

Πίνακας Περιεχομένων

Πίνακας Εικόνων

Κεφάλαιο 1

Natural Language Processing (NLP)

1.1.Εισαγωγή στο NLP

"Έχετε ποτέ αναρωτηθεί πώς μπορούμε να δημιουργήσουμε ένα σύστημα που κάνει τις μεταφράσεις αυτόματα μεταξύ των γλωσσών; Ή ένα σύστημα που να μπορεί να καταλάβει γλωσσικές οδηγίες από έναν άνθρωπο; Η επεξεργασία φυσικής γλώσσας (NLP) θα καλύψει τις βασικές αρχές των μαθηματικών και υπολογιστικών μοντέλων της γλώσσας, καθώς και τις εφαρμογές των μοντέλων για την επίλυση αυτών των προβλήματων." (Ανώνυμος)

Το Natural Language Processing (NLP) είναι ένα πεδίο της επιστήμης των υπολογιστών, της τεχνητής νοημοσύνης και της γλωσσολογίας, που ασχολείται με τις αλληλεπιδράσεις μεταξύ υπολογιστών και της ανθρώπινης (φυσικής) γλώσσας. Ως εκ τούτου, το NLP έχει άμεση σχέση με τον τομέα της αλληλεπίδρασης ανθρώπου-υπολογιστή. Οι μεθοδολογίες και τεχνικές που χρησιμοποιούνται, υποθέτουν ότι τα μοτίβα στη γραμματική και οι εννοιολογικές σχέσεις ανάμεσα στις λέξεις οποιασδήποτε γλώσσας, μπορούν να διαρθρωθούν επιστημονικά. Ο απώτερος στόχος του NLP είναι να καθοριστεί ένα σύστημα συμβόλων, σχέσεων και εννοιολογικών πληροφοριών που μπορούν να χρησιμοποιηθούν από τη λογική του υπολογιστή με σκοπό τη δημιουργία μιας τεχνητής γλώσσας ερμηνείας. Πολλές προκλήσεις του NLP περιλαμβάνουν κατανόηση της φυσικής γλώσσας, επιτρέποντας στους υπολογιστές να αντλήσουν νόημα από τον άνθρωπο ή από γλωσσικές είσοδους (natural language input),ενώ άλλες αφορούν την παραγωγή φυσικής γλώσσας (natural language generation). [3] [5]

1.2. Ιστορική Αναδρομή

Η ιστορία του NLP ξεκινά συνήθως στη δεκαετία του 1950, παρόλο που αντίστοιχο έργο μπορεί να βρεθεί από τις προηγούμενες περιόδους. Το 1950, ο Alan Turing δημοσίευσε ένα άρθρο με τίτλο "Computing Machinery and Intelligence", το οποίο πρότεινε αυτό που καλείται τώρα το τεστ Turing[1] ως κριτήριο της νοημοσύνης.

Το αρχικό κίνητρο για την ουσιαστική ανάπτυξη του NLP ήταν οι μεταφράσεις κειμενων. Έτσι, μια πρώτη προσέγγιση ήταν το πείραμα Georgetown το 1954, το οποίο περιλάμβανε πλήρως αυτόματη μετάφραση περισσότερων από εξήντα ρωσικών φράσεων στα αγγλικά . Οι συγγραφείς υποστήριξαν ότι μέσα σε τρία ή πέντε χρόνια , η αυτόματη μετάφραση θα αποτελούσε ένα πρόβλημα του παρελθόντος . Ωστόσο, η πραγματική πρόοδος ήταν πολύ αργή , και μετά την έκθεση

[1] *Turing test* : Πρόκειται για ένα εμπειρικό test, ένα παιχνίδι, στο οποίο η χρήση της γλώσσας από ένα υπολογιστή θα αποτελούσε τη βάση για τον καθορισμό αν το μηχάνημα θα μπορούσε να σκεφτεί. Εάν το μηχάνημα μπορούσε να κερδίσει το παιχνίδι, θα κρινόταν ως έξυπνο. Πιο συγκεκριμένα, στο παιχνίδι αυτό υπάρχουν τρεις συμμετέχοντες: δύο άνθρωποι και ένας υπολογιστής. Ένας από τους ανθρώπους θα παίζει το ρόλο του κριτή. Για να κερδίσει ο κριτής, θα πρέπει να αποφασίσει ποιος από τους άλλους δύο συμμετέχοντες είναι ο υπολογιστής, κάνοντας μια σειρά από ερωτήσεις μέσω τηλέτυπου. Το έργο του υπολογιστή είναι να ξεγελάσει τον κριτή κάνοντάς τον να πιστεύει ότι είναι άνθρωπος, απαντώντας ανάλογα στις ερωτήσεις. Έργο του δεύτερου, ανθρώπινου συμμετέχοντα, είναι να πείσει τον κριτή ότι ο άλλος συμμετέχων είναι ο υπολογιστής και ότι αυτός είναι ο άνθρωπος.

alpac το 1966 , η οποία διαπίστωσε ότι η μακρά έρευνα δέκα χρόνων είχε αποτύχει να εκπληρώσει τις προσδοκίες , το αποτέλεσμα ήταν η δραματική μείωση της χρηματοδότησης για την αυτόματη μετάφραση. Μικρή περαιτέρω έρευνα στην αυτόματη μετάφραση διεξήχθη τα επόμενα χρόνια, μέχρι που στα τέλη της δεκαετίας του 1980 , αναπτύχθηκαν τα πρώτα στατιστικά συστήματα[2] που πραγματοποιούσαν αυτόματη μετάφραση.

Στο μεσοδιάστημα, μερικά από τα ιδιαίτερα επιτυχημένα συστήματα NLP που αναπτύχθηκαν ήταν στη δεκαετία του 1960 το SHRDLU , ένα σύστημα φυσικής γλώσσας που εργάζονται σε περιορισμένα " blocks worlds[3] " με περιορισμένα λεξιλόγια , και το ELIZA , μια προσομοίωση ενός ροτζεριανού ψυχοθεραπευτή , που γράφτηκε από τον Joseph Weizenbaum μεταξύ 1964-1966 . Χρησιμοποιώντας σχεδόν καθόλου πληροφορίες για την ανθρώπινη σκέψη ή τα συναισθήματα , το ELIZA παρείχε μερικές φορές εκπληκτικά ανθρώπινες αλληλεπίδρασεις.

Κατά τη διάρκεια της δεκαετίας του 1970 πολλοί προγραμματιστές άρχισαν να γράφουν " εννοιολογικές οντολογίες ", το οποίο δόμησε πληροφορίες του πραγματικού κόσμου σε κατανοητά ηλεκτρονικά δεδομένα. Παραδείγματα είναι το MARGIE (Schank , 1975) , το SAM (Cullingford , 1978) , το PAM (Wilensky , 1978) , το TaleSpin (Meehan , 1976) , το QUALM (Lehnert , 1977) , το Politics (Carbonell , 1979) , και το Plot Units (Lehnert 1981) . Κατά τη διάρκεια αυτής της

---

[2] *Statistical NLP* : Γίνεται χρήση στοχαστικών, πιθανολογικών και στατιστικών μεθόδων για την επίλυση διαφόρων ζητημάτων, ιδίως εκείνων που οφείλονται στο γεγονός ότι οι μεγάλες προτάσεις είναι ιδιαίτερα ασαφής όταν υποβάλλονται σε επεξεργασία με ρεαλιστικές γραμματικές, αποδίδοντας χιλιάδες ή εκατομμύρια πιθανές αναλύσεις. Μέθοδοι για την αποσαφήνιση συχνά περιλαμβάνουν τη χρήση των corpora (βλέπε υποσημείωση 7-2, σελίδα 16) και των μοντέλων Markov. Το στατιστικό NLP περιλαμβάνει όλες τις ποσοτικές προσεγγίσεις στην αυτοματοποιημένη επεξεργασία γλώσσών, συμπεριλαμβανομένης της πιθανολογικής μοντελοποίησης, της θεωρίας της πληροφορίας, και της γραμμικής άλγεβρας. Η τεχνολογία κυρίως προέρχεται κυρίως από το machine learning corpora (βλέπε υποσημείωση 8, σελίδα 16) και το data mining[2-2], τα οποία είναι τα πεδία της τεχνητής νοημοσύνης που αφορούν τη μάθηση από τα δεδομένα. [3]

[2-2] *data mining* : αποτελεί ένα διεπιστημονικό υποπεδίο της επιστήμης των υπολογιστών και ουσιαστικά είναι η υπολογιστική διαδικασία της ανακάλυψης προτύπων σε μεγάλα σύνολα δεδομένων, που αφορούν τις μεθόδους για διασταύρωση της τεχνητής νοημοσύνης, του machine learning, στατιστικών στοιχείων, και συστημάτων βάσεων δεδομένων. Ο συνολικός στόχος της διαδικασίας του data mining (ξόρυξη δεδομένων) είναι να αποσπαστούν πληροφορίες από ένα σύνολο δεδομένων και να μετατραπούν σε μια κατανοητή δομή για περαιτέρω χρήση. [20]

[3] *Block world* : είναι ένας από τους πιο διάσημους τομείς σχεδιασμού στον τομέα της τεχνητής νοημοσύνης. Φανταστείτε μια σειρά από κύβους (μπλοκ) να κάθονται σε ένα τραπέζι. Ο στόχος είναι να κατασκευάσουμε μία ή περισσότερες κατακόρυφες στοίβες των μπλοκ. Η σύλληψη είναι ότι μόνο ένα τετράγωνο μπορεί να μετακινηθεί σε μια στιγμή: μπορεί είτε να τοποθετηθεί στο τραπέζι ή να τοποθετηθεί πάνω από ένα άλλο μπλοκ. Εξαιτίας αυτού, κάθε μπλοκ που είναι, σε μια δεδομένη στιγμή, κάτω από ένα άλλο μπλοκ δεν μπορεί να μετακινηθεί. [8]

περιόδου , πολλά chatterbot[4] γράφτηκαν συμπεριλαμβανομένων των PARRY , Racter και Jabberwacky .

Μέχρι τη δεκαετία του 1980 , τα περισσότερα συστήματα NLP βασίστηκαν σε πολύπλοκα σύνολα από χειρόγραφους κανόνες . Ξεκινώντας στα τέλη της δεκαετίας του 1980 , ωστόσο , υπήρξε μια επανάσταση στον τομέα του NLP με την εισαγωγή των αλγορίθμων μηχανικής μάθησης για την επεξεργασία της γλώσσας. Αυτό οφείλεται τόσο στη σταθερή αύξηση της υπολογιστικής ισχύος που προκύπτουν από το Νόμο του Moore[5] και τη σταδιακή αποδυνάμωση της κυριαρχίας των θεωριών της γλωσσολογίας του Chomsky[6] (π.χ. μετασχηματιστική γραμματική), του οποίου το θεωρητικό υπόβαθρο αποθάρρυνε το είδος του corpus linguistic[7], το οποίο επισημαίνει την προσέγγιση του machine -learning[8] στον τομέα της επεξεργασίας γλώσσας (language processing). Μερικοί από τους πρώτους σε εφαρμογή αλγορίθμους του machine-learning , όπως τα decision trees[9], παρήγαγαν συστήματα

---

[4] *Chatterbot*: είναι ένα πρόγραμμα υπολογιστή σχεδιασμένο για την προσομοίωση έξυπνης συνομιλίας με έναν ή περισσότερους χρήστες μέσω οπτικοακουστικών μεθόδων, κυρίως για συμμετοχή σε μικρή συζήτηση. [9]

[5] *Moore's law* : Ο νόμος του Moore είναι η παρατήρηση ότι, καθόλη την ιστορία των υπολογιστών, ο αριθμός των τρανζίστορ σε ολοκληρωμένα κυκλώματα διπλασιάζεται περίπου κάθε δύο χρόνια. [10]

[6] *Chomsky's Theory* : Η βάση για τη γλωσσική θεωρία του Τσόμσκι είναι ότι οι αρχές που διέπουν τη δομή της γλώσσας είναι βιολογικά καθορισμένες στο ανθρώπινο μυαλό και ως εκ τούτου γενετικά μεταδιδόμενες. [11]

[7] *Corpus linguistic* : είναι η μελέτη της γλώσσας, όπως εκφράζεται σε δείγματα (corpora[7-2]) κειμένων του "πραγματικού κόσμου". Αυτή η μέθοδος αντιπροσωπεύει μια άμεση προσέγγιση για την εξαγωγή ενός συνόλου από αφηρημένους κανόνες, από τους οποίους μια φυσική γλώσσα διέπεται ή με τους οποίους μπορεί να αναφέρθεί σε άλλη γλώσσα. Αρχικά η διαδικασία αυτή γινόταν με το χέρι, ενώ στην πορεία έγινε μια αυτοματοποιημένη διαδικασία. [12]

[7-2] ***Corpus-corpora*** : Στη γλωσσολογία, ένα corpus (πληθυντικός corpora) είναι ένα μεγάλο και διαρθρωμένο σύνολο κειμένων (σήμερα συνήθως αποθηκεύονται με ηλεκτρονικά μέσα και επεξεργάζονται). Χρησιμοποιούνται για να κάνουν στατιστική ανάλυση και έλεγχο υποθέσεων, να έλεγχουν συμβάντα ή να επικυρώνουν γλωσσικούς κανόνες σε μια συγκεκριμένη γλωσσική περιοχή.

[8] *Machine learning*: είναι ένας κλάδος της τεχνητής νοημοσύνης, που αφορά την κατασκευή και μελέτη των συστημάτων, τα οποία στηρίζονται στη γνώση των δεδομένων. Για παράδειγμα, ένα σύστημα machine learning θα μπορούσε να εκπαιδευτεί σε μηνύματα ηλεκτρονικού ταχυδρομείου για να μάθει να διακρίνει τα spam και μη-spam μηνύματα. Στο επόμενο στάδιο, θα μπορούσε να χρησιμοποιηθεί για να ταξινομήσει τα νέα μηνύματα ηλεκτρονικού ταχυδρομείου σε spam και μη-spam φακέλους. [26]

[9] *Decision tree*: Ένα δέντρο απόφασης είναι ένα εργαλείο επιλογής αποφάσεων που χρησιμοποιεί ένα δέντρο που μοιάζει με γράφημα ή το μοντέλο των αποφάσεων και τις πιθανές συνέπειές τους, συμπεριλαμβανομένων των αποτελεσμάτων τυχαίων γεγονότων, τις

αυστηρών if-then κανόνων, παρόμοιους με τους ήδη υπάρχοντες χειρόγραφους κανόνες . Ωστόσο, η έρευνα επικεντρώνεται όλο και περισσότερο σε στατιστικά μοντέλα , τα οποία κάνουν ‘’χαλαρές’’ , πιθανολογικές αποφάσεις με βάση πραγματικές τιμές για τα χαρακτηριστικά που συνθέτουν τα δεδομένα εισόδου . Τα μοντέλα cache language[10], τα οποία χρησιμοποιούνται σε πολλά συστήματα αναγνώρισης ομιλίας[11], πλέον αποτελούν παραδείγματα τέτοιων στατιστικών μοντέλων. Τέτοια μοντέλα είναι γενικά πιο ισχυρά όταν εισάγεται άγνωστη είσοδος, ειδικά όταν η είσοδος περιέχει σφάλματα (όπως είναι πολύ κοινό για πραγματικά δεδομένα) , ενώ παράγουν πιο αξιόπιστα αποτελέσματα όταν ενσωματώνονται σε ευρύτερα συστήματα που περιλαμβάνουν πολλαπλές δευτερεύουσες εργασίες.

Πολλές σημαντικές αρχικές επιτυχίες σημειώθηκαν στον τομέα της αυτόματης μετάφρασης , λόγω κυρίως της δουλειάς της IBM Research , όπου αναπτύχθηκαν διαδοχικά πιο περίπλοκα στατιστικά μοντέλα . Τα συστήματα αυτά ήταν σε θέση να επωφεληθούν από τα υπάρχοντα πολύγλωσσα corpora που είχαν παραχθεί από το Κοινοβούλιο του Καναδά και της Ευρωπαϊκής Ένωσης ως αποτέλεσμα των νόμων για τη μετάφραση όλων των κυβερνητικών διαδικασιών σε όλες τις επίσημες γλώσσες των αντίστοιχων συστημάτων διακυβέρνησης. Ωστόσο, τα περισσότερα άλλα συστήματα εξαρτώνται από corpora ειδικά αναπτυγμένα για τις εργασίες που εφαρμόζονται από τα συστήματα αυτά, τα οποία ήταν (και συχνά συνεχίζουν να είναι) ένας σημαντικός περιορισμός για την επιτυχία τους. Ως αποτέλεσμα, μια μεγάλη έρευνα έχει γίνει σε μεθόδους πιο αποτελεσματικης μάθησης από περιορισμένες ποσότητες δεδομένων.

Πρόσφατη έρευνα επικεντρώθηκε περισσότερο χωρίς ή με μικρή επίβλεψη αλγορίθμων μάθησης. Τέτοιοι αλγόριθμοι είναι σε θέση να μάθουν από δεδομένα που δεν είναι σημειωμένα με τις επιθυμητές απαντήσεις , ή χρησιμοποιώντας ένα συνδυασμό σχολιασμένων και μη σχολιασμένων δεδομένων. Σε γενικές γραμμές , το έργο αυτό είναι πολύ πιο δύσκολο απ’ ό,τι η επιβλεπόμενη μάθηση , και συνήθως

δαπάνες των πόρων, και τη χρησιμότητα τους. Είναι ένας τρόπος εμφάνισης ένος αλγόριθμου. [13]

[10] *Cache language model* : είναι ένα είδος στατιστικού γλωσσικού προτύπου. Αυτά τα μοντέλα εμφανίζονται κυρίως στο υποπεδίο του NLP της επιστήμης των υπολογιστών και εισάγουν πιθανότητες σε ακολουθίες λέξεων μέσω μιας δοσμένης κατανομής πιθανότητας[10-2]. Τα στατιστικά γλωσσικά μοντέλα αποτελούν βασικά στοιχεία των συστημάτων αναγνώρισης ομιλίας (speech recognition) και πολλών συστημάτων αυτόματης μετάφρασης. Το ιδιαίτερο χαρακτηριστικό ενός cache language μοντέλου είναι ότι περιέχει ένα χώρο προσωρινής αποθήκευσης και μπορεί να εκχωρήσει σχετικά υψηλές πιθανοτήτες σε λέξεις ή ακολουθίες λέξεων που εμφανίζονται σε ένα συγκεκριμένο κείμενο. Σήμερα, όμως, η κύρια αλλά σε καμία περίπτωση αποκλειστική χρήση ενός cache language μοντέλου, είναι σε συστήματα αναγνώρισης ομιλίας. [14]

[10-2] *Κατανομή πιθανότητας* : Στον τομέα των πιθανοτήτων και της στατιστικής, μια κατανομή πιθανοτήτας εκχωρεί μια πιθανότητα σε κάθε μετρήσιμο υποσύνολο των πιθανών αποτελεσμάτων ενός τυχαίου πειράματος, έρευνας, ή διαδικασίας της επαγωγικής στατιστικής. [15]

[11] *Speech recognition- αναγώριση ομιλίας* : Στην επιστήμη των υπολογιστών είναι η μετάφραση των προφορικών λέξεων σε κείμενο. [16]

παράγει λιγότερο ακριβή αποτελέσματα για δεδομένη ποσότητα δεδομένων εισόδου. Ωστόσο, υπάρχει ένας τεράστιος αριθμός μη σχολιασμένων αρχείων (συμπεριλαμβανομένου, μεταξύ άλλων , του συνόλου του περιεχομένου του World Wide Web), τα οποία είναι διαθέσιμα και μπορούν συχνά να χρησιμοποιηθούν για τα κάτωθι αποτελέσματα. [3]

1.3. Μια δεύτερη ματιά – εξερεύνηση σε βάθος

1.3.1. Σημασία και στόχος του κλάδου NLP

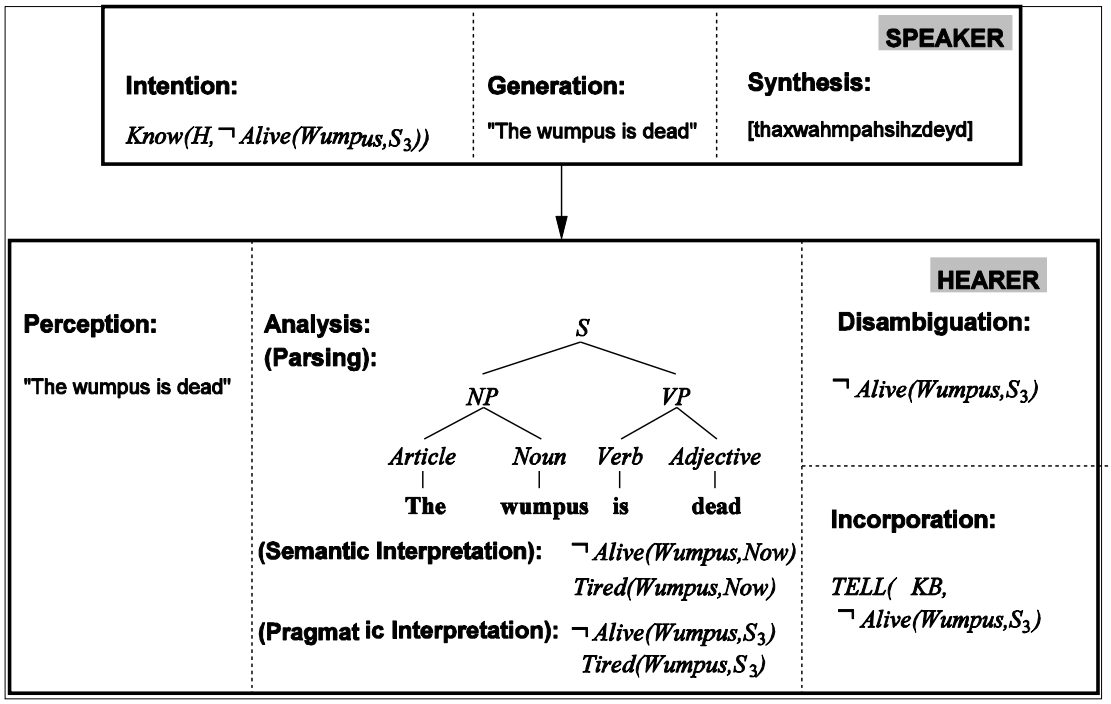
Το NLP είναι η τεχνολογία για την αντιμετώπιση ενός διαχρονικού προϊόντος: της ανθρώπινης γλώσσας, όπως φαίνεται στα μηνύματα ηλεκτρονικού ταχυδρομείου, τις ιστοσελίδες, τα tweets, τις περιγραφές προϊόντων, τις ιστορίες εφημερίδων, τα social media, και τα επιστημονικά άρθρα, σε χιλιάδες γλώσσες και ποικιλίες. Κατά την τελευταία δεκαετία, επιτυχημένες εφαρμογές του NLP έχουν γίνει μέρος της καθημερινής εμπειρίας μας, από την ορθογραφία και τη διόρθωση της γραμματικής σε επεξεργαστές κειμένου στην αυτόματη μετάφραση στο διαδίκτυο, από την ανίχνευση ανεπιθύμητων μηνυμάτων ηλεκτρονικού ταχυδρομείου στις αυτόματες ερωταποκρίσεις, από τη συλλογή των απόψεων των ανθρώπων σχετικά με τα προϊόντα ή τις υπηρεσίες στην εξαγωγή ραντεβού με χρήση email. Έτσι, η μελέτη και ενασχόληση με αυτόν το κλάδο θα οδηγήσει στη δημιουργία θεμελιώδων αλγορίθμων και μαθηματικών μοντέλων για την επεξεργασία δεδομένων ανθρώπινης γλώσσας και τρόπους χρήσης τους για την επίλυση πρακτικών προβλημάτων που ανακύπτουν. [4]

Ως αποτέλεσμα των παραπάνω προκύπτει ότι ο κύριος στόχος της παραγωγής και κατανόησης της φυσικής γλώσσας είναι η γενικότερη επικοινωνία. Αυτή με τη σειρά της μπορεί να χωριστεί σε δύο υποκατηγορίες, ανάλογα με τον τρόπο προσέγγισης:

- Επικοινωνία για τον ομιλητή
  - Πρόθεση - Intention: Περιλαμβάνει την απόφαση σχετικά με το χρονικό πλαίσο και την επιλογή των πληροφορίων που πρόκειται να διαβιβαστούν. Γενικά, απαιτεί σωστό σχεδιασμό και αξιολόγηση σχετικά με τους στόχους και τις πεποιθήσεις του ομιλητή.
  - Παραγωγή - Generation: Περιλαμβάνει τη μετάφραση των πληροφοριών, με σκοπό τη δημιουργία μιας ακολουθίας λέξεων στην επιθυμητή φυσική γλώσσα.
  - Σύνθεση - Synthesis: Παρουσίαση της ανώτερης ακολουθίας στην επιθυμητή μορφή.
- Επικοινωνία για τον ακροατή
  - Αντίληψη - Perception: Αντιστοίχιση των δεδομένων εισόδου σε μια ακολουθία λέξεων, π.χ. με χρήση της οπτικής αναγνώρισης χαρακτήρων ή της αναγνώρισης ομιλίας.
  - Ανάλυση - Analysis: Προσδιορίζεται το περιεχόμενο των πληροφοριών της ακολουθίας.
    - Συντακτική ερμηνεία (parsing): Εύρεση του σωστού συντακτικού δένδρου που δείχνει τη δομή της φράσης της ακολουθίας.
    - Σημασιολογική Ερμηνεία: Απόσπαση της κυριολεκτικής έννοιας/ερμηνείας της ακολουθίας (λογική μορφή).

- Ρεαλιστική Ερμηνεία: Λαμβάνεται υπόψην η επίδραση του γενικού πλαισίου του κειμένου στην τροποποίηση της κυριολεκτικής ερμηνείας της πρότασης.

- Ενσωμάτωση - Incorporation: Απόφαση σχετικά με την εγκυρότητα του περιεχομένου της ακολουθίας ενσωμάτωση της νέας πληροφορίας. [6]



Εικόνα 1.1. Επικοινωνία Ακροατή και ομιλητή [6]

1.3.2. Οι ρίζες του NLP

Η επεξεργασία της φυσικής γλώσσας έχει τις ρίζες της σε ένα τομέα που λέγεται semiotics (ή semiology), που σημαίνει μελέτη των σημείων. Η σημειολογία αναπτύχθηκε από τον Charles Sanders Peirce (επισήμονας λογικής και φιλόσοφος) και τον Ferdinand de Saussure (γλωσσολόγος). Η σημειολογία χωρίζεται σε τρεις κλάδους: τη σύνταξη, τη σημασιολογία και την πραγματολογία.

Η εξαγωγή της ερμηνείας ενός οποιουδήποτε κειμένου κάποιας γλώσσας, πραγματοποιείται από ένα πλήρη επεξεργαστής φυσικής γλώσσας μετά από μια σειριακή πορεία ανάλυσης τουλάχιστον επτά επίπεδων. Ωστόσο, θα εστιαστούν τα τέσσερα κύρια επίπεδα:

- Morphological : Ένα **μόρφημα** είναι το μικρότερο τμήμα μιας λέξης που μπορεί να μεταφέρει μια διακριτή έννοια. Για παράδειγμα, μορφήματα είναι τα εξής “και”, “τραβώ”, “σε”, “carry”, “pre”, “ly”, “s” κ.α. Η μορφολογική ανάλυση λειτουργεί με τις λέξεις σε αυτό το επίπεδο. Τυπικά, ένας επεξεργαστής φυσικής γλώσσας ξέρει πώς να κατανοήσει πολλαπλές μορφές μιας λέξης, για παράδειγμα τον πληθυντικό και ενικό της.
- Syntactic : Σε αυτό το επίπεδο, οι επεξεργαστές φυσικής γλώσσας επικεντρώνονται σε δομικές πληροφορίες και σχέσεις. Ουσιαστικά αφορά τη σωστή σειρά των λέξεων και την επίδρασή της στο γενικό νόημα.

Π.χ.

- The dog bit the boy
- The boy bit the dog
- Bit boy dog the the

➢ Semantic : Οι επεξεργαστές φυσικής γλώσσας αντλούν τον απόλυτο ορισμό (ορισμό λεξικού) από το κείμενο. Ουσιαστικά αφορά το κυριολεκτικό νόημα των λέξεων, φράσεων και προτάσεων.

Π.χ.

- "plant" ως φωτοσυνθετικός οργανισμός
- "plant" ως μονάδα παραγωγής
- "plant" ως πράξη της σποράς

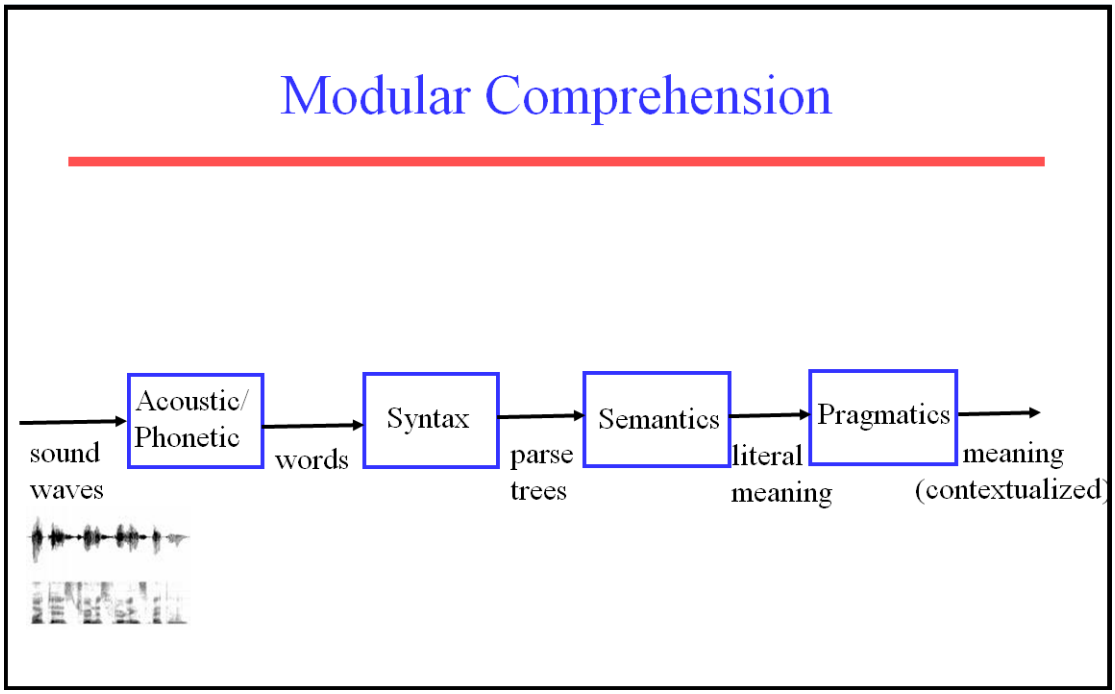
➢ Pragmatic : Επεξεργαστές φυσικής γλώσσας αντλούν γνώση από εξωτερικές γενικές πληροφορίες. Ουσιαστικά αφορά το συνολικό επικοινωνιακό και κοινωνικό πλαίσιο και τις επιπτώσεις αυτού στην ερμηνεία του κειμένου.

Π.χ.

- The ham sandwich wants another beer. (αναφορά)

John thinks vanilla. (Ελλειψη)

[5][6]



Εικόνα 1.2. Επίπεδα επεξεργασίας ηχητικών κυμάτων ομιλίας [6]

1.3.3.Ένα βασικό εμπόδιο – Ambiguity

Ένα θεμελιώδες ζήτημα που αποτελεί εμπόδιο στην επεξεργασία των γλωσσικών δεδομένων και χρήζει άμεσης αντιμετώπισης είναι η ασάφεια (**Ambiguity**). Λέμε ότι κάποιο input είναι ambiguous (διφορούμενο/ασαφές) εάν πολλαπλές, εναλλακτικές γλωσσικές δομές μπορούν να κατασκευαστούν γι' αυτό. Για παράδειγμα η φράση *"I made her duck"* έχει τις εξής διαφορετικές σημασίες:

1. *I cooked waterfowl for her*: Μαγείρεψα την πάπια για εκείνη.

2. *I cooked waterfowl belonging to her*: Μαγείρεψα την πάπια που ανήκει σε εκείνη. [3]
3. *I created the (plaster?) duck she owns*: έφτιαξα την πάπια που της ανήκει, π.χ. μια γύψινη πάπια.
4. *I caused her to quickly lower her head or body*: την ανάγκασα να χαμηλώσει γρήγορα το κεφάλι ή το σώμα της.
5. *I waved my magic wand and turned her into undifferentiated waterfowl*: κυμάτισα με μαγεία το ραβδί μου και τη μεταμόρφωσα σε πάπια.

Αυτές οι διαφορετικές σημασίες προκαλούνται από μια σειρά από ασάφειες. Πρώτον, οι λέξεις "duck" και "her" είναι μορφολογικά ή συντακτικά διφορούμενες ως προς το μέρος του λόγου που αποτελούν. Το duck μπορεί να είναι ρήμα ή ουσιαστικό, ενώ το her μπορεί να είναι δοτική ή κτητική αντωνυμία. Δεύτερον, η λέξη make είναι σημασιολογικά ασαφής - μπορεί να σημαίνει δημιουργώ ή μαγειρεύω. Επίσης, το make είναι ταυτόχρονα και συντακτικά ασαφές με έναν διαφορετικό τρόπο: το make ως ρήμα μπορεί να είναι μεταβατικό λαμβάνοντας ένα άμεσο αντικείμενο (επιλογή 2), ή μπορεί να λαμβάνει δύο αντικείμενα (επιλογή 5), που σημαίνει ότι το δεύτερο αντικείμενο (duck) έγινε κατηγορούμενο στο πρώτο αντικείμενο her. Τέλος, το "make" μπορεί να πάρει ένα άμεσο αντικείμενο και μια ρηματική φράση (επιλογή 4), πράγμα που σημαίνει ότι το αντικείμενο her πρόκειται να εκτελέσει τη ρηματική δράση duck. Επιπλέον, στον προφορικό λόγο υπάρχει ένα ακόμα βαθύτερο είδος ασάφειας : η πρώτη λέξη " I " θα μπορούσε να μπερδευτεί με το "eye", ενώ η δεύτερη λέξη "made" με τη λέξη "maid". [1]

Ασαφείς συνθέσεις μπορούν να δημιουργήσουν τεράστιο αριθμό πιθανών ερμηνειών. Για παράδειγμα, στην αγγλική γλώσσα, μια πρόταση που σε n προθετικές φράσεις έχει πάνω από 2^n συντακτικές ερμηνείες. Έτσι θεωρώντας τις παρακάτω φράσεις παίρνουμε και τον αντίστοιχο αριθμό ερμηνειών λόγω ασάφειας :

- *"I saw the man with the telescope"* → **2 ερμηνείες**
- *"I saw the man on the hill with the telescope"* → **5 ερμηνείες**
- *"I saw the man on the hill in Texas with the telescope"* →**14 ερμηνείες**
- *"I saw the man on the hill in Texas with the telescope at noon"* → **42 ερμηνείες**
- *"I saw the man on the hill in Texas with the telescope at noon on Monday"* → **132 ερμηνείες**

Παρόλο που οι ασάφειες δυσχεραίνουν το έργο του NLP, η ύπαρξη τους μπορεί εύκολα να δικαιολογηθεί, καθώς οφείλεται σε ένα σύνολο πραγμάτων: *α)* έχοντας μια μοναδική γλωσσική έκφραση για κάθε πιθανή σύλληψη που θα μπορούσε να χρησιμοποιηθεί, θα έκανε ιδιαίτερα περίπλοκη τη γλώσσα, ενώ οι γλωσσικές εκφράσεις θα προέκυπταν υπερβολικά μακροσκελείς. *β)* επιτρέποντας αναλύσιμη ασάφεια, δημιουργούνται μικρότερες γλωσσικές εκφράσεις, δηλαδή έχουμε συμπίεση δεδομένων. *γ)* η γλώσσα στηρίζεται στην ικανότητα των ανθρώπων να χρησιμοποιούν τις γνώσεις τους και τις ικανότητές τους, ώστε να επιλύουν σωστά τις ασάφειες που προκύπτουν. *δ)* σπανίως η αποσαφήνιση αποτυγχάνει. Γενικά, η σωστή επιλογή των ερμηνειών των γλωσσικών εκφράσεων απαιτεί :

a) Σύνταξη :
   - Ένα όνομα είναι συνήθως το υποκείμενο του ρήματος
b) Σημασιολογία :
   - Το Γιώργος και το Ελένη είναι ονόματα ανθρώπων.
   - Το Μαργαρίτα είναι όνομα ανθρώπου και ταυτόχρονα ένα λουλούδι.
   - Toyota είναι μάρκα αυτοκινήτων, ενώ το Avensis είναι ένα μοντέλο.

c) Πραγματολογία - Γνώση του κόσμου : χρήση πληροφοριών γενικού περιεχομένου για τη σωστή ερμηνεία του κειμένου
   - Οι πιστωτικές κάρτες απαιτούν από τους χρήστες να πληρώσουν τόκους. [6]

Με σκοπό την επίλυση αυτού του προβλήματος των ασαφειών, έχουν δημιουργηθεί και σχεδιαστεί ειδικά μοντέλα και αλγόριθμοι. Για παράδειγμα, η επιλογή για το αν το duck θα είναι ρήμα ή ουσιαστικό θα μπορεί να δοθεί από ένα εργαλείο που λέγεται part-of-speech tagging. Η απόφαση για το κατά πόσον το make σημαίνει δημιουργώ ή μαγειρεύω μπορεί να επιλυθεί με ένα εργαλείο που λέγεται word sense disambiguation. Όλα αυτά και άλλα εργαλεία θα αναλυθούν στη συνέχεια του κεφαλαίου και ειδικότερα στα επόμενα κεφάλαια. [1]

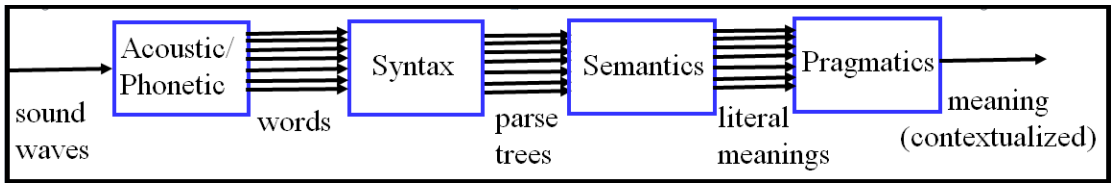
1.3.4. Pipelining Problem

Υποθέτοντας ξεχωριστά ανεξάρτητα τμήματα (components) επεξεργασίας για την αναγνώριση ομιλίας, τη σύνταξη, τη σημασιολογία, την πραγματολογία, κλπ. γίνεται πιο βολική η "σπονδυλωτή" (σε πολλά επίπεδα) ανάπτυξη λογισμικού. Ωστόσο, οι αποφάσεις συχνά από το "υψηλότερο επίπεδο" διαδικασιών απαιτούνται για την αποσαφήνιση των "χαμηλότερων επιπέδων" διαδικασιών. Ένα παράδειγμα για τη συντακτική αποσαφήνιση που στηρίζεται στη σημασιολογική είναι το εξής:

- Στο ζωολογικό κήπο, κάποιοι άνδρες έδειχναν σε μια ομάδα φοιτητών διάφορα είδη ιπτάμενων ζώων. Ξαφνικά, ένας από τους φοιτητές χτύπησε τον άνδρα *με το ρόπαλο*.

Εδώ γίνεται εμφανές ότι υπάρχει δυσκολία στην απόφαση του ρόλου της τελευταίας φράσης (αυτής σε πλάγια γραφή). Γενικά, αν σε κάθε επίπεδο παίρνονται δύσκολες αποφάσεις, τότε όταν σε μεταγενέστερο στάδιο φανεί ότι υπήρξε κάποια εσφαλμένη απόφαση, δεν μπορεί να γίνει επανεξέταση των δεδομένων και αλλαγές αποφάσεων. Αυτό είναι το **Pipelining Problem**, δηλώνοντας δηλαδή ότι υπάρχει κάποια ροή που δεν μπορεί να αλλάξει. Έτσι, στο συγκεκριμένο παράδειγμα, αν κατά τη διάρκεια της συντακτικής ανάλυσης γίνει σύνδεση της φράσης "με το ρόπαλο" με τη λέξη "χτύπησε", τότε δε μπορεί να γίνει επανασύνδεση με τη λέξη "άνθρωπο" , όταν φανεί κατά τη διάρκεια της σημασιολογικής ή πραγματολογικής ανάλυσης ότι η αρχική συσχέτιση ήταν ασαφής.

Σε αυτό το σημείο, λύση μπορεί να δοθεί με αύξηση του εύρους (bandwidth) των μοντέλων παραγωγής αποφάσεων. Αυτό σημαίνει ότι κάθε component θα δύναται να παράξει πολλαπλές ερμηνείες των φράσεων που λαμβάνει ως είσοδο, με αποτέλεσμα προηγούμενα components να μπορούν να επανεξετάσουν τα δεδομένα και να επαναπροσδιορίσουν τις ερμηνείες τους σε περιπτώσεις ασαφειών.

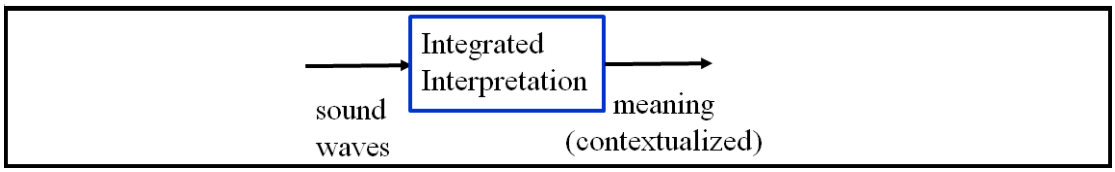


Εικόνα 1.3.Επίπεδα επεξεργασίας ηχητικών κυμάτων με αυξημενο εύρος [6]

Βέβαια, ακόμα και η παραπάνω προσέγγιση ενέχει κινδύνους, καθώς ο αριθμός των ερμηνειών, πλέον, αυξάνεται συνδυαστικά. Ανακύπτει, λοιπόν, ανάγκη για αποδοτική

κωδικοποίηση του συνδυασμού των ερμηνειών, γεγονός που μπορεί να επιτευχθεί με διαφόρους τρόπους, όπως word lattices[12] κ.α.

Τέλος, λύση στο pipelining problem μπορεί να δοθεί με μια μέθοδο που ονομάζεται Integrated Interpretation (ολοκληρωμένη ερμηνεία), η οποία θα συνδυάζει φωνητικούς, συντακτικούς, σημασιολογικούς και πραγματολογικούς περιορισμούς. Αυτή, όμως, η διαδικασία παρουσιάζει δυσκολίες σχεδίασης και εφαρμογής και απαιτεί δυνητικά υπολογιστικά συστήματα. [6]



Εικόνα 1.4. Επεξεργασία ηχητικών κυμάτων με χρήση ενός ολοκληρωμένου επιπέδου[6]

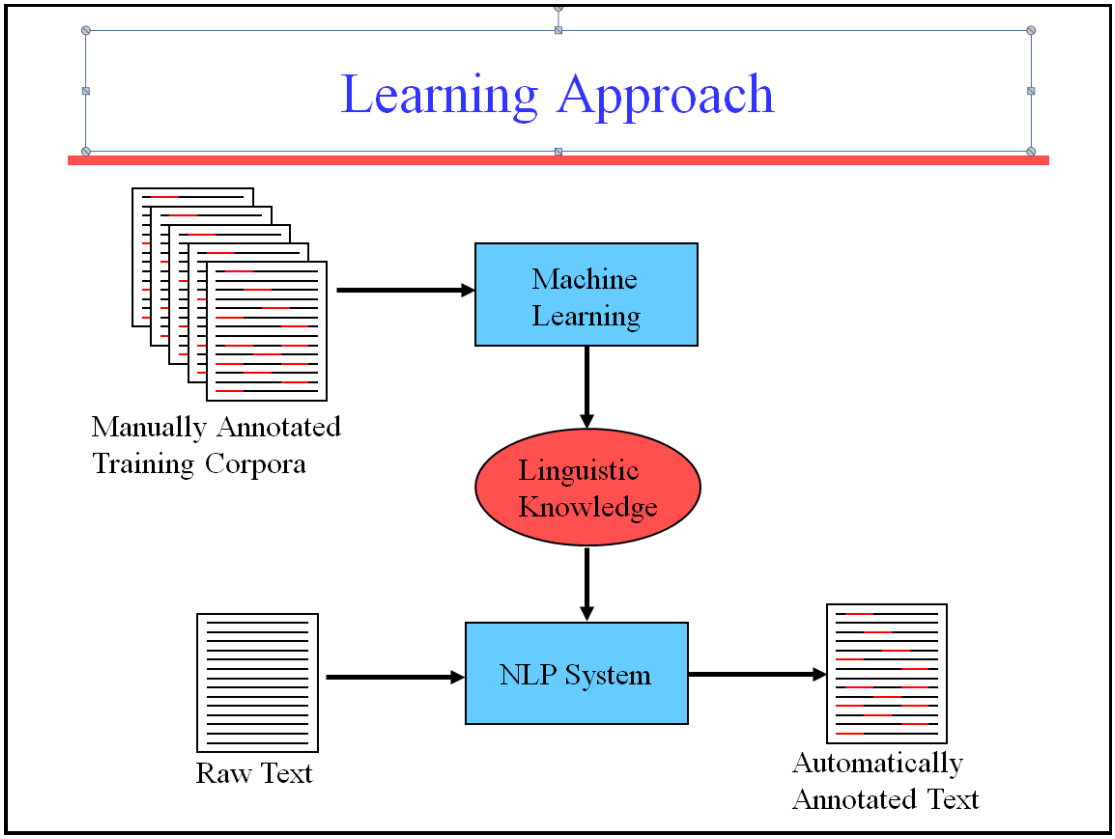
1.4. Σύγχρονες τάσεις και εφαρμογές

1.4.1. Χρήση του machine learning

Οι πρώτες προσεγγίσεις για την επεξεργασία της γλώσσας γίνονταν με τον παραδοσιακό και ορθολογιστικό τρόπο και απαιτούσαν ειδικούς, για τον προσδιορισμό και την επισημοποίηση των απαιτούμενων γνώσεων. Αρχικές υλοποιήσεις βασίζονταν συνήθως στην άμεση κωδικοποίηση στο χέρι μεγάλου συνόλου κανόνων. Το Manual knowledge engineering (διαδικασία στο χέρι παραγωγής γνώσεων), είναι μια δύσκολη, χρονοβόρα και επιρρεπής σε λάθη διαδικασία, καθώς οι "κανόνες" σε κάθε γλώσσα έχουν πολλές εξαιρέσεις και παρατυπίες. Έτσι, τα χειροκίνητα συστήματα ήταν ακριβά για να αναπτυχθούν και οι ικανότητές τους ήταν περιορισμένες και "εύθραυστες" (όχι ισχυρές). […]

Σε αντιδιαστολή, οι σύγχρονοι αλγόριθμοι NLP βασίζονται στο machine learning, ιδιαίτερα σε στατιστική μορφή. Τα παραδείγματα του machine learning είναι διαφορετικά από αυτά των περισσότερων προηγούμενων προσπαθειών στον τομέα της γλωσσικής επεξεργασίας. Επιτρέπουν, αντί για τη χρήση γενικών αλγορίθμων μάθησης, που συχνά (αν και όχι πάντα) στηρίζονται στη στατιστική συμπερασματολογία, την αυτόματη δημιουργία κανόνων, μέσω της ανάλυσης μεγάλων corpora από χαρακτηριστικά παραδείγματα του πραγματικού κόσμου. [3]

[12] *Word lattice* : είναι ένας κατευθυνόμενος ακυκλικός γράφος με ένα ενιαίο σημείο εκκίνησης και ακμές που επισημαίνονται με λέξεις και βάρη. Τα word lattices μπορούν να αντιπροσωπεύσουν οποιοδήποτε πεπερασμένο σύνολο από string (αν και αυτή η γενικότητα κάνει τα word lattices ελαφρώς λιγότερο αποδοτικά στο χώρο). [7]



Εικόνα 1.5. Χρήση Machine Learning σε NLP συστήματα [6]

Τα συστήματα αυτά έχουν πλεονεκτήματα σε σχέση με αυτά που βασίζονται στην παραγωγή κανόνων στο χέρι :

- Μεγάλες ποσότητες ηλεκτρονικών κειμένων είναι πλέον διαθέσιμες.
- Ο σχολιασμός των corpora είναι πιο εύκολος και απαιτεί λιγότερη εμπειρία από το manual knowledge engineering. [6]
- Οι διαδικασίες μάθησης που χρησιμοποιούνται κατά τη διάρκεια του machine learning εστιάζουν αυτόματα στις πιο συνήθεις περιπτώσεις, ενώ κατά τη σύνταξη των κανόνων με το χέρι, συχνά δεν είναι καθόλου εμφανές το που πρέπει να κατευθυνθούν οι προσπάθειες.
- Οι αυτόματες διαδικασίες μάθησης μπορούν να κάνουν χρήση των στατιστικών αλγορίθμων για να παράγουν μοντέλα που μπορούν να λειτουργήσουν ακόμα και σε άγνωστες εισόδους (π.χ. που περιέχουν λέξεις ή δομές που δεν έχουν δει ποτέ πριν) ή και σε λανθασμένες εισόδους (π.χ. λέξεις με ορθογραφικά λάθη ή λέξεις που παραλήφθηκαν κατά λάθος). Σε γενικές γραμμές, ο χειρισμός αυτών των εισόδων με χειρόγραφους κανόνες ή, γενικότερα, η δημιουργία συστημάτων με χειρόγραφους κανόνες που παίρνουν σίγουρες αποφάσεις, είναι εξαιρετικά δύσκολα πραγματοποιήσιμη, επιρρεπής σε λάθη και χρονοβόρα περίπτωση.
- Τα συστήματα που βασίζονται στην αυτόματη εκμάθηση των κανόνων μπορούν να γίνουν πιο ακριβή απλώς με την παροχή περισσότερων δεδομένων εισόδου. Ωστόσο, τα συστήματα που βασίζονται σε χειρόγραφους κανόνες μπορούν να γίνουν πιο ακριβή με την αύξηση της πολυπλοκότητας των κανόνων, το οποίο είναι αρκετά πιο δύσκολο

έργο. Ειδικότερα, υπάρχει ένα όριο στην πολυπλοκότητα των συστημάτων που βασίζονται σε χειρογραφους κανόνες, πέραν του οποίου τα συστήματα γίνονται όλο και πιο ανεξέλεγκτα. Ωστόσο, η δημιουργία περισσότερων δεδομένων εισόδου σε συστήματα machine learning, απαιτεί απλώς μια αντίστοιχη αύξηση στον αριθμό των δεδουλευμένων ωρών, συνήθως χωρίς σημαντικές αυξήσεις στην πολυπλοκότητα της διαδικασίας σχολιασμού. [3]

1.4.2. Βασικοί τομείς εφαρμογών

Το NLP ασχολείται με διάφορα ζητήματα, από τα οποία κάποια παράγουν αποτελέσματα με άμεση εφαρμογή, ενώ άλλα αποτελούν ενδιάμεσες εργασίες που χρησιμεύουν στην επίλυση μεγαλύτερων προβλημάτων. Παρακάτω γίνεται μια σύντομη περιγραφή των βασικών έργων του NLP:

- Automatic Summarization: είναι η διαδικασία της μείωσης ένος γραπτού κειμένου με χρήση ενός προγράμματος, προκειμένου να δημιουργηθεί μια περίληψη που θα διατηρεί τα πιο σημαντικά σημεία του πρωτότυπου εγγράφου. Συχνά χρησιμοποιείται για την δημιουργία περιλήψεων κειμένων γνωστού τύπου, όπως άρθρα ενός οικονομικού τμήματος εφημερίδας.
- Coreference resolution : Παίρνει μια πρόταση ή ένα μεγαλύτερο κομμάτι κειμένου και καθορίζει ποιες λέξεις αναφέρονται στα ίδια αντικείμενα. Το Anaphora resolution είναι ένα συγκεκριμένο παράδειγμα αυτού του τομέα, το οποίο αφορά ειδικά το ταίριασμα των αντωνυμιών με τα ουσιαστικά ή τα ονόματα που αναφέρονται.
  Π.χ. [6]

  – John put the carrot on the plate and ate it.

- Discourage analysis : είναι ένας γενικός όρος για μια σειρά από προσεγγίσεις στην ανάλυση γραπτών, φωνητικών και νοηματικών γλωσσικών δεδομένων. Περιλαμβάνει πολλά πεδία, όπως ο προσδιορισμός της δομής του λόγου, η αναγνώριση και ο χαρακτηρισμός μιας ομιλίας σε ένα κείμενο (π.χ. ερώτηση τύπου ναι-όχι, δήλωση, βεβαίωση) κ.α.
- Machine translation : Αυτόματη μετάφραση κειμένου από μία ανθρώπινη γλώσσα στην άλλη. Αυτό είναι ένα από τα πιο δύσκολα προβλήματα, και είναι μέλος μιας κατηγορίας προβλημάτων, που ονομάζεται "AI-complete[13]".

[13] *AI-complete problems* : Στον τομέα της τεχνητής νοημοσύνης, τα πιο δύσκολα προβλήματα είναι ανεπίσημα γνωστά ως AI-complete (πλήρη) ή AI-hard (σκληρά), πράγμα που σημαίνει ότι η δυσκολία των υπολογιστικών προβλημάτων αυτών είναι ισοδύναμη με εκείνη της επίλυσης του κεντρικού προβλήματος της τεχνητής νοημοσύνης : τη μετατροπή των υπολογιστών με νοημοσύνη, δηλαδή το σχεδιασμό τους, ώστε να είναι τόσο έξυπνοι όσο οι άνθρωποι. Για να θεωρηθεί ένα πρόβλημα AI-complete σημαίνει ότι δε θα μπορεί να λυθεί με ένα απλό ειδικό αλγόριθμο. Τα AI-complete προβλήματα θεωρείται ότι πρέπει να περιλαμβάνουν υπολογιστική όραση, κατανόηση φυσικής γλώσσας, και αντιμετώπιση

Π.χ. [6]

- Hasta la vista, bebé ⇒ See you later, baby. (τα λέμε μετά, μωρό μου)

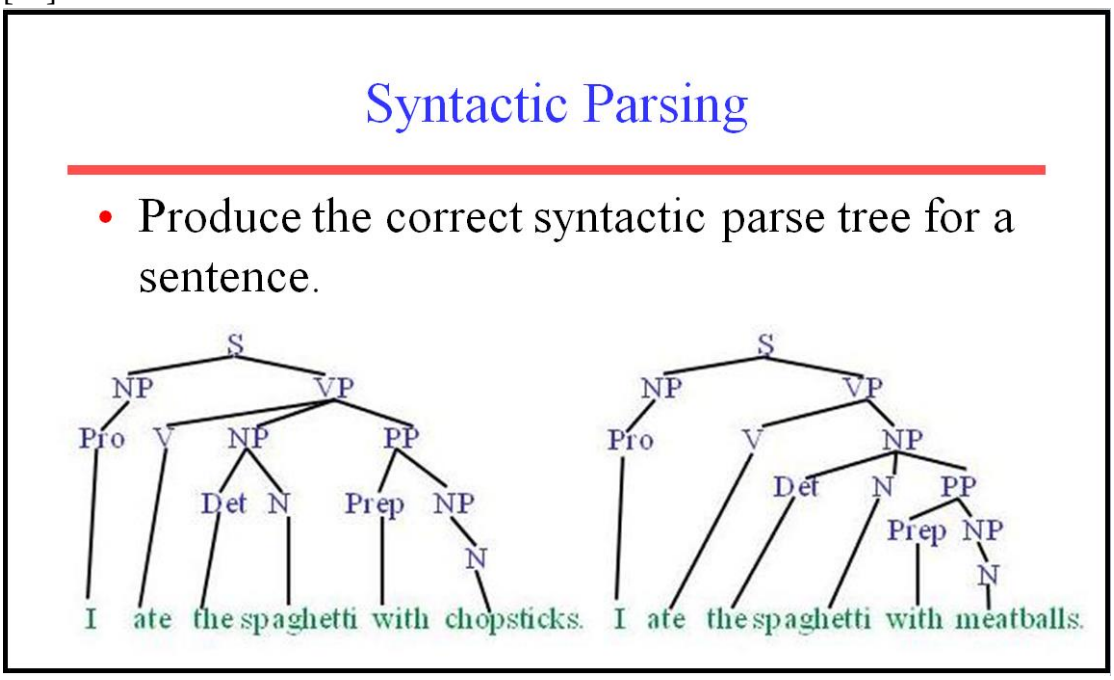
- Morphological segmentation : Χωρισμός λέξεων σε επιμέρους μορφήματα και προσδιορισμός της κατηγορίας τους. Η δυσκολία αυτής της εργασίας εξαρτάται σε μεγάλο βαθμό από την πολυπλοκότητα της μορφολογίας (δηλ. τη δομή των λέξεων) της γλώσσας που εξετάζεται. Τα Αγγλικά έχουν αρκετά απλή μορφολογία, συγκεκριμένα κλιτική μορφολογία, και έτσι είναι συχνά δυνατό να αγνοηθεί αυτό το έργο εξ ολοκλήρου και απλά να μοντελοποιηθούν όλες οι πιθανές μορφές μιας λέξης (π.χ. "open, opens, opened, opening") ως χωριστές λέξεις. Σε γλώσσες όπως η τουρκική, όμως, μια τέτοια προσέγγιση δεν είναι εφικτή, δεδομένου ότι κάθε καταχώρηση λεξικού έχει χιλιάδες πιθανές μορφές.
  Π.χ. [6]
  - carried ⇒ carry + ed (παρελθοντικός χρόνος)
  - independently ⇒ in + (depend + ent) + ly
  - Googlers ⇒ (Google + er) + s (πληθυντικός)
- Named entity recognition : Λαμβάνουμε ένα κομμάτι κειμένου, καθορίζουμε ποια στοιχεία του κειμένου ταιριάζουν με κύρια ονόματα, όπως τα άτομα ή οι τοποθεσίες, και ποιο είναι το είδος κάθε τέτοιου ονόματος (π.χ. πρόσωπο, θέση, οργάνωση). Σημειώστε ότι, αν και η κεφαλαιοποίηση μπορεί να βοηθήσει στην αναγνώριση των επώνυμων οντοτήτων σε γλώσσες όπως τα αγγλικά, η πληροφορία αυτή αφενώς δεν μπορεί να βοηθήσει στον καθορισμό του τύπου του ονόματος, και αφεταίρου σε πολλές περιπτώσεις προκύπτει ανακριβής ή ανεπαρκής. Για παράδειγμα, η πρώτη λέξη μιας πρότασης ξεκινάει πάντα με κεφαλαίο γράμμα και τις περισσότερες φορές δεν αποτελεί όνομα, ενώ υπάρχουν ονόματα που αποτελούνται από πολλές λέξεις και μόνο μερικές από αυτές κεφαλαιοποιούνται. Επιπλέον, πολλές άλλες γλώσσες (π.χ. κινέζικα ή αραβικά) δεν έχουν καμία κεφαλαιοποίηση, ενώ άλλες με κεφαλαιοποίηση τυγχαίνει να μην μπορούν να τη χρησιμοποιούν για να διακρίνουν ονόματα. Για παράδειγμα, η γερμανική γλώσσα κεφαλαιοποιεί όλα τα ουσιαστικά, ανεξάρτητα από το αν αναφέρονται σε ονόματα, ενώ γάλλοι και ισπανοί δεν κεφαλαιοποιούν τα ονόματα που χρησιμεύουν ως επίθετα.
- Natural language generation: Μετατροπή πληροφοριών από βάσεις δεδομένων υπολογιστή σε αναγνώσιμη από κάποια ανθρώπινη γλώσσα μορφή.
- Natural language understanding : Μετατροπή τμημάτων κειμένου σε πιο επίσημες παραστάσεις, όπως η λογική των δομών πρώτης τάξης, καθώς είναι πιο εύκολο για τα προγράμματα υπολογιστών να τις χειριστούν. Το natural language understanding περιλαμβάνει τον προσδιορισμό των σημασιολογιών που προέρχονται από μία γλωσσική έκφραση, η οποία παίρνει συνήθως τη μορφή οργανωμένου συμβολισμού των εννοιών των φυσικών γλωσσών. Μια ρητή τυποποίηση των σημασιολογιών χωρίς σύγχυση με έμμεσες παραδοχές, όπως η παραδοχή κλειστού κόσμου (Closed World Assumption) έναντι του ανοιχτού, ή η υποκειμενική Ναι / Όχι έναντι της αντικειμενικής True / False, αναμένεται για την κατασκευή μιας βάσης επισημοποίησης σημασιολογιών.

απρόβλεπτων περιστάσεων, καθώς γίνεται η επίλυση κάθε προβληματος πραγματικού κόσμου. [17]

- Optical character recognition: Λήψη μιας εικόνας που αντιπροσωπεύει κάποιο τυπωμένο κείμενο και καθορισμός του αντίστοιχου κειμένου.
- Part-of-speech (POS) tagging: Λήψη μιας πρότασης και καθορισμός για το τι μέρος του λόγου είναι κάθε λέξη. Πολλές λέξεις, ειδικά οι κοινές, μπορούν να χρησιμεύσουν ως διάφορα μέρη του λόγου. Για παράδειγμα, η λέξη "book" μπορεί να είναι ένα ουσιαστικό ("the book is on the table") ή ρήμα ("to book a flight"), το "set" μπορεί να είναι ουσιαστικό, ρήμα ή επίθετο,ενώ η λέξη "out" μπορεί να είναι οποιοδήποτε από τουλάχιστον πέντε διαφορετικά μέρη του λόγου. Μερικές γλώσσες έχουν μεγαλύτερη τέτοιου είδους ασάφεια από άλλες. Γλώσσες με μικρή κλιτική μορφολογία, όπως η αγγλική, είναι ιδιαίτερα επιρρεπείς σε τέτοια ασάφεια.
  Π.χ. [6]

	Ρήμα	Πρόθεση	Ουσιαστ.	Πρόθεση	Ουσιαστ.
–	*Έφαγα*	*τα*	*μακαρόνια*	*με*	*κρέας.*

- Parsing: Καθορισμός του δέντρου (γραμματική ανάλυση) μιας δεδομένης πρότασης. Η γραμματική των φυσικών γλωσσών είναι ασαφής και οι τυπικές προτάσεις έχουν πολλαπλές πιθανές αναλύσεις. Στην πραγματικότητα, για μια τυπική φράση μπορεί να υπάρχουν χιλιάδες δυνητικές αναλύσεις (οι περισσότερες από τις οποίες φαίνονται εντελώς παράλογες σε έναν άνθρωπο). [27]



Εικόνα 1.6. Συντακτικά δέντρα [6]

- Question answering : Άμεσες απαντήσεις σε ερωτήσεις των φυσικών γλωσσών, που βασίζονται σε πληροφορίες που παρουσιάζονται σε corpora εγγράφων κειμένου (π.χ. το διαδίκτυο). Γενικά, οι τυπικές ερωτήσεις ενδέχεται να έχουν μια συγκεκριμένη σωστή απάντηση (όπως "Ποια είναι η πρωτεύουσα του Καναδά;"). Ωστόσο, μερικές φορές εξετάζονται και ανοιχτές ερωτήσεις (όπως "Ποιο είναι το νόημα της ζωής;").
- Relationship extraction : Λήψη ενός κομματιού κειμένου και εντοπισμός των σχέσεων μεταξύ των ονομάτων (π.χ. ποιος είναι ο σύζυγος τίνος).

- Sentence breaking: Λήψη ενός κομματιού κειμένου και εύρεση των ορίων κάθε πρότασης. Συχνά αυτά τα όρια μαρκάρονται από περιόδους ή με χρήση άλλων σημείων στίξης, αν και αυτοί οι χαρακτήρες μπορούν να εξυπηρετούν και άλλους σκοπούς (π.χ. σήμανση συντομογραφιών).
- Sentiment analysis: Εξαγωγή υποκειμενικών πληροφοριών συνήθως από μια σειρά εγγράφων, χρησιμοποιώντας online κριτικές για τον καθορισμό της "πολικότητας" συγκεκριμένων αντικειμένων. Είναι ιδιαίτερα χρήσιμο για τον προσδιορισμό των τάσεων της κοινής γνώμης στα social media, με σκοπό το εμπόριο.
- Speech recognition: Λήψη ένας κλιπ ήχου ενός ατόμου ή ανθρώπων που μιλούν και καθορισμός της αναπαράστασης κειμένου της ομιλίας αυτής. Αυτό είναι το αντίθετο της μετατροπής κειμένου σε ομιλία και είναι κι αυτό ένα από τα εξαιρετικά δύσκολα προβλήματα που ήδη προαναφέραμε ως "AI-complete". Στη φυσική ομιλία δεν υπάρχουν σχεδόν καθόλου παύσεις μεταξύ των διαδοχικών λέξεων, και ως εκ τούτου η τμηματοποίηση του λόγου είναι ένα απαραίτητο υποπρόβλημα της αναγνώρισης ομιλίας (βλ. παρακάτω, speech segmentation). Σημειώστε, επίσης, ότι στις περισσότερες ομιλούμενες γλώσσες, οι ήχοι που εκπροσωπούν διαδοχικά γράμματα αναμειγνύονται μεταξύ τους σε μια διαδικασία που ονομάζεται συνάρθρωση, γεγονός που κάνει τη μετατροπή του αναλογικού σήματος σε διακριτούς χαρακτήρες μια πολύ δύσκολη διαδικασία.
- Speech segmentation: Λήψη ενός κλιπ ήχου από ένα άτομο ή ανθρώπους που μιλούν και διαχωρισμός του σε λέξεις. Αποτελεί μια υποεργασία της αναγνώρισης ομιλίας και συνήθως ομαδοποιείται με αυτή.
- Text summarization: Παράγει μια σύντομη περίληψη ενός μεγαλύτερου εγγράφου.
- Topic Segmentation: Λήψη ενός κομματιού κειμένου και διαχωρισμός του σε τμήματα, καθένα από τα οποία είναι αφιερωμένο σε ένα θέμα, καθώς και προσδιορισμός των θεμάτων αυτών.
- Word segmentation: Χωρισμός ενός μεγάλου κομματιού συνεχούς κειμένου σε χωριστές λέξεις. Για μια γλώσσα όπως τα Αγγλικά, αυτό είναι κάτι το τετριμμένο, αφού οι λέξεις είναι συνήθως χωρισμένες με κενά. Ωστόσο, υπάρχουν ορισμένες γραπτές γλώσσες, όπως τα Κινέζικα, τα Ιαπωνικά και τα Ταϊλανδέζικα, οι οποίες δε σηματοδοτούν τα όρια των λέξεων με τέτοιο τρόπο, και σε αυτές τις γλώσσες το word segmentation είναι ένα σημαντικό έργο που απαιτεί τη γνώση του λεξιλογίου και της μορφολογίας των λέξεων στη γλώσσα.
  Π.χ. [6]
  - jumptheshark.com ⇒ jump the shark .com
  - myspace.com/pluckerswingbar ⇒ myspace .com pluckers wing bar (όχι myspace .com plucker swing bar)
- Word sense disambiguation: Πολλές λέξεις έχουν περισσότερες από μία έννοια. Άρα προκύπτει το θέμα της επιλογής της έννοιας που ταιριάζει καλύτερο στο κείμενο. Για το πρόβλημα αυτό, μας δίνεται συνήθως μια λίστα από λέξεις με τις αντίστοιχες έννοιες τους, π.χ. ένα λεξικό ή online εφαρμογές, όπως το WordNet.
  Π.χ. [6]
  - Ellen has a strong *interest* in computational linguistics. (ενδιαφέρον)
  - Ellen pays a large amount of *interest* on her credit card. (τόκοι)

Υπάρχουν βέβαια και περιπτώσεις προβλημάτων που μπορούν να θεωρηθούν υποπροβλήματα του NLP ή και εντελώς ξεχωριστά ζητήματα. Μερικά από τα σημαντικότερα παραθέτονται παρακάτω :

- Information retrieval (IR): Έχει να κάνει με την αποθήκευση, αναζήτηση και ανάκτηση πληροφοριών. Πρόκειται για ένα ξεχωριστό πεδίο μέσα στην πληροφορική (πιο κοντά στις βάσεις δεδομένων), που ταυτόχρονα βασίζεται και σε ορισμένες μεθόδους του NLP (για παράδειγμα το stemming[14] - βλεπε παρακάτω). Μερικές πρόσφατες έρευνες και εφαρμογές προσπαθούν να γεφυρώσουν το χάσμα μεταξύ IR και NLP.
- Information Extraction (IE) : Αυτό αναφέρεται γενικά στην εξαγωγή σημασιολογικών πληροφοριών από τα κείμενα. Επίσης, προσδιορίζει τις φράσεις που αναφέρονται σε συγκεκριμένους τύπους των οντοτήτων και των σχέσεων στο κείμενο. Γενικά καλύπτει πολλούς τομείς, όπως το named entity recognition (αναδεικνύει ονόματα των ανθρώπων, τόπων, οργανώσεις, κλπ.), το Coreference resolution, το relationship extraction (προσδιορίζει συγκεκριμένες σχέσεις μεταξύ των οντοτήτων.), κλπ.
  Π.χ. [6]
  People organization **places**
  - *Michael Dell* is the CEO of Dell Computer Corporation and lives in **Austin Texas**.
- Speech processing: Αυτός ο τομέας περιλαμβάνει το speech recognition, το text-to-speech[15] , καθώς και άλλους σχετικούς τομείς. [3]

1.5. Το Μέλλον και περιορισμοί

1.5.1. Οι επόμενες σκέψεις

Το μέλλον στον τομέα του Natural Language Processing είναι άμεσα συνδεδεμένο με την ανάπτυξη του τομέα της τεχνητής νοημοσύνης. Καθώς η κατανόηση της φυσικής γλώσσας βελτιώνεται, οι υπολογιστές θα είναι σε θέση να μάθουν από τις πληροφορίες online και να εφαρμόσουν αυτά που έμαθαν στον πραγματικό κόσμο. Σε συνδυασμό με την παραγωγή φυσικής γλώσσας (natural

---

[14] *Stemming* : Στη γλωσσική μορφολογία και στην ανάκτηση πληροφοριών, το stemming είναι η διαδικασία μείωσης της λέξης και εύρεσης του στελέχους (stem) της, δηλαδή μια γραπτή μορφή της λέξης. Το stemming δεν χρειάζεται να είναι ταυτόσημο με τη μορφολογική ρίζα της λέξης. Συνήθως σχετικές λέξεις έχουν το ίδιο στέλεχος, έστω και αν το εν λόγω θέμα δεν είναι από μόνη της μια έγκυρη ρίζα. Για παράδειγμα το stem των λέξεων “fishing”, “fished”, “fisher” είναι το “fish” το οποίο τυγχάνει να είναι και ρίζα των ανώνερων λέξεων, ενώ το στέλεχος των “arguing”, “argued”, “argue” είναι το “argu” το οποίο δεν αποτελεί λέξη. [19]

[15] *Text-to-speech (TTS)* : Είναι το σύστημα που μετατρέπει ένα κανονικό κείμενο γλώσσας σε ομιλία. Αποτελεί ένα υποπεδίο του γενικού τομέα Speech synthesis, ο οποίος αναφέρεται στην τεχνιτή παραγωγή ανθρώπινης ομιλίας. [18]

language generation), οι υπολογιστές θα γίνονται όλο και πιο ικανοί να λαμβάνουν και να δίνουν οδηγίες.

Στο μέλλον, οι άνθρωποι μπορεί να μην χρειάζεται να κρυπτογραφούν προγράμματα, αλλά να υπαγορεύουν σε έναν υπολογιστή σε ανθρώπινη φυσική γλώσσα, και ο υπολογιστής να κατανοεί και να ενεργεί σύμφωνα με τις οδηγίες αυτές.

1.5.2. Περιορισμοί του NLP

Ένα από τα βασικά προβλήματα του σύγχρονου NLP είναι ότι οι περισσότεροι γλωσσολόγοι προσεγγίζουν το NLP σε ρεαλιστικό επίπεδο, συγκεντρώνοντας τεράστιες ποσότητες πληροφοριών σε μεγάλες βάσεις γνώσης που περιγράφουν τον κόσμο στο σύνολό του. [5]

Κεφάλαιο 2

Εισαγωγή στο GATE

2.1. Εισαγωγή

2.1.1 Τι είναι Gate

Το GATE (General Architecture for Text Engineering) είναι ένα από τα πλέον ευρέως διαδεδομένα συστήματα για την ανάπτυξη στοιχείων λογισμικού που επεξεργάζονται την ανθρώπινη γλώσσα. Εμφανίστηκε πρώτη φορά το 1996 και από τότε επανασχεδιάστηκε ολοκληρωτικά μέχρι να διαδοθεί εκ νέου το 2002. Πλέον, είναι σχεδόν 18 ετών και είναι σε ενεργή χρήση για όλους τους τύπους των υπολογιστικών εργασιών που αφορούν την ανθρώπινη γλώσσα. Στην ουσία, είναι ένα open source[16] free software[17] ικανό να επιλύσει ένα μεγάλο σύνολο προβλημάτων επεξεργασίας κειμένου. Το Gate υπερέχει σε ανάλυση κειμένου, καθώς αναφέρεται σε ποικίλα σχήματα και μεγέθη δεδομένων, λόγοι που το καθιστούν ίσως το καλύτερο σύστημα του είδους του. Αυτή τη στιγμή απασχολεί δεκάδες χιλιάδες ανθρώπους ανά τον κόσμο, έχοντας ήδη κοστίσει πολλά εκατομμύρια δολάρια για την εκπόνηση του. Σε αυτή την πορεία έχει εξελιχθεί αρκετά προσθέτοντας συνεχώς νέα εργαλεία λογισμικού, όπως λογισμικό για προσωπική χρήση σε desktop, web εφαρμογή, βιβλιοθήκη Java, αρχιτεκτονική και διεργασία. Πιο συγκεκριμένα το Gate περιλαμβάνει:

- Το GATE Developer: Ένα IDE[18] (integrated development environment-ολοκληρωμένο περιβάλλον ανάπτυξης) για γλωσσικά στοιχεία με χρήση ενός ευρέως χρησιμοποιημένου συστήματος IE (Information Extraction) και ενός συνόλου από διαφορετικά plugins.
- Το GATE Cloud: μια παράλληλα κατανεμημένη μηχανή επεξεργασίας που συνδυάζει το GATE Developer με μια πλήρως βελτιστοποιημένη υποδομή υπηρεσιών που λειτουργούν σε κάποιον υπερυπολογιστή, κυρίως για επεξεργασία μεγάλης κλίμακας κείμενων.
- Το GATE Teamware: ένα νέο web-based software, για την ακρίβεια ένα web-

---

[16] *Open source software*: είναι το λογισμικό με πηγαίο κώδικα που διατίθεται με άδεια στην οποία ο κάτοχος των πνευματικών δικαιωμάτων παρέχει τα δικαιώματα για τη μελέτη, την αλλαγή και τη διανομή του λογισμικού σε οποιονδήποτε και για οποιονδήποτε σκοπό. [21]

[17] *Free software*: ή libre software ή software libre, είναι λογισμικό που διανέμεται μαζί με τον πηγαίο κώδικα του και παρέχεται στο πλαίσιο μιας άδειας χρήσης λογισμικού, που εγγυώνται στους χρήστες την ελευθερία να εκτελούν το λογισμικό για οποιοδήποτε σκοπό, καθώς και για τη μελέτη, την προσαρμογή / τροποποιήση και τη διανομή του αρχικού λογισμικού και τις προσαρμοσμένες/διαφορετικές εκδόσεις του. Συχνά, το λογισμικό αναπτυσεται σε συνεργασία με εθελοντές προγραμματιστές υπολογιστών.

[18] *IDE - integrated development environment* (ολοκληρωμένο/διαδραστικό περιβάλλον ανάπτυξης): είναι μια εφαρμογή λογισμικού που παρέχει ολοκληρωμένες δυαντότητες σε προγραμματιστές υπολογιστών για ανάπτυξη λογισμικού. Ένα IDE αποτελείται συνήθως από ένα πρόγραμμα επεξεργασίας πηγαίου κώδικα, έτοιμα εργαλεία αυτοματισμού και ένα πρόγραμμα εντοπισμού σφαλμάτων (debugger). Τα περισσότερα σύγχρονα IDEs προσφέρουν χαρακτηριστικά ολοκλήρωσης ευφυή κώδικα. [24]

based framework[19] σχολιασμών, που επιτρέπει στους χρήστες να εκτελούν πολύπλοκα projects, με τη συμμετοχή λιγότερο εξειδικευμένων και φθηνότερων σχολιαστών, οι οποίοι εργάζονται εξ αποστάσεως μέσα από web browsers.

- Το GATE Mimir: είναι ένα πολυ-πρότυπο ευρετήριο και αποθετήριο διαχείρισης πληροφοριών, που μπορεί να χρησιμοποιηθεί για εύρεση και αναζήτηση σε κείμενα, σχόλια, σημασιολογικά σχήματα (οντολογίες), και σημασιολογικά μετα-δεδομένα (π.χ. δεδομένα). Επιτρέπει ερωτήματα, που συνδυάζουν αυθαίρετα ερωτήματα πλήρους-κειμένου, διαρθρωτικά, γλωσσικά και σημασιολογικά και αυτό μπορεί να κλιμακωθεί σε terabytes του κειμένου.
- Το GATE Embedded: μια βιβλιοθήκη java σχεδιασμένη για ένταξη σε ποικίλες εφαρμογές, παρέχοντας πρόσβαση σε όλες τις υπηρεσίες που χρησιμοποιούνται από το GATE Developer και όχι μόνο.
- Μια αρχιτεκτονική: μια υψηλού επιπέδου οργανωτική εικόνα του τρόπου σύνθεσης λογισμικού επεξεργασίας γλώσσας.
- Ένα Process: αναφέρει τους τρόπους με τους οποίους γίνεται η ανάθεση, ο σχεδιασμός, η ανάπτυξη, εφαρμογή, διατήρηση και αξιολόγηση μεγάλων ροών εργασίας επεξεργασίας κειμένου.

Ένα από τα κυριότερα κίνητρα της ομάδας του GATE ήταν να ανεξαρτητοποιήσει την έρευνα για αποτελέσματα διαφόρων εφαρμογών, από την ουσιαστική επίλυση ενός προβλήματος. Σύμφωνα με αυτή την ιδέα, οδηγήθηκε στις εξής βασικές λειτουργίες:

- μοντελοποίηση των εξειδικευμένων δομών δεδομένων
- μέτρηση, αξιολόγηση, συγκριτική ανάλυση δεδομένων
- οπτικοποίηση και επεξεργασία των annotations[20], ontologies[21], parse

---

[19] *Framework:* ένα framework λογισμικού είναι ένα abstraction[19-2] , στο οποίο το λογισμικό που παρέχει γενικές λειτουργίες, μπορεί να αλλάξει επιλεκτικά με έναν πρόσθετο κώδικα που έχει γράψει ο χρήστης, παρέχοντας έτσι ειδικό λογισμικό εφαρμογών. Ένα framework λογισμικού είναι μια καθολική, επαναχρησιμοποιήσιμη πλατφόρμα λογισμικού για την ανάπτυξη λογισμικού εφαρμογών, προϊόντων και λύσεων. Τα frameworks λογισμικού περιλαμβάνουν προγράμματα στήριξης, compilers, βιβλιοθήκες κώδικα, σετ εργαλείων, και διεπαφές προγραμματισμού εφαρμογών (APIs) που συγκεντρώνουν όλα τα διαφορετικά συστατικά για να καταστεί δυνατή η ανάπτυξη ενός project ή λύσης. [22]

[19-2] *Abstraction:* είναι η διαδικασία διαχωρισμού ιδέων συγκεκριμένων περιπτώσεων από τις ιδέες του χώρου εργασίας. Συγκεκριμένα, προσπαθεί να συνυπολογίσει τις λεπτομέρειες από ένα κοινό μοτίβο, έτσι ώστε οι προγραμματιστές να μπορούν να εργάζονται κοντά στο επίπεδο της ανθρώπινης σκέψης, αφήνοντας στην άκρη τις λεπτομέρειες που μπορεί να έχουν σημασία στην πράξη, αλλά είναι άνευ σημασίας για τη λύση του προβλήματος. [23]

[20] *Annotations* : είναι τα μεταδεδομένα (π.χ. ένα σχόλιο, επεξήγηση, σήμανση παρουσίασης) που συνδέονται με κάποιο κείμενο, εικόνα, ή άλλα δεδομένα. Συχνά τα annotations αναφέρονται σε ένα συγκεκριμένο τμήμα των αρχικών δεδομένων.

[21] *Ontology* : Στην επιστήμη των υπολογιστών και της πληροφορικής, μια οντολογία αναπαριστά επισήμως τη γνώση ως μια ιεραρχία εννοιών εντός ενός τομέα, χρησιμοποιώντας ένα κοινό λεξιλόγιο για να υποδηλώσει τα είδη, τις ιδιότητες και τις αλληλεξαρτήσεις αυτών των εννοιών. [25]

trees[22]κλπ.

- μια πεπερασμένη γλώσσα μεταγωγής για την ταχεία προτυποποίηση και την αποτελεσματική εφαρμογή των μεθόδων ανάλυσης (JAPE).
- Εξαγωγή αποτελεσμάτων από διάφορες περιπτώσεις για machine learning.

Στην κορυφή των βασικών του λειτουργιών, το GATE περιλαμβάνει εργαλεία για διάφορες εργασίες επεξεργασίας γλώσσων, π.χ. parsers[23], morphology[24], tagging[25], Information Retrieval tools[26], Information Extraction[27] εργαλεία για διάφορες γλώσσες, και πολλά άλλα. Το GATE Developer και το GATE Embedded παρέχονται με ένα Information Extraction system (ANNIE), το οποίο έχει προσαρμοστεί και αξιολογηθεί σε πολύ μεγάλο βαθμό σε διάφορα συστήματα.

2.1.2 Ορισμοί και γενικές αναφορές

Ορισμένοι χρήσιμοι ορισμοί για την περαιτέρω κατανόηση του GATE

---

[22] *Parse tree*: είναι ένα οργανωμένο δέντρο που αντιπροσωπεύει τη συντακτική δομή μιας συμβολοσειράς, σύμφωνα με κάποια γραμματική χωρίς συμφραζόμενα (context free).

[23] *Parser* / *Parsing* : είναι η διαδικασία της ανάλυσης μια σειράς από σύμβολα, είτε σε φυσική γλώσσα ή σε γλώσσα υπολογιστών, σύμφωνα με τους κανόνες μιας επίσημης γραμματικής. Ο όρος parsing προέρχεται από τη Λατινική pars (orationis), που σημαίνει μέρος (του λόγου).

[24] *Morphology* : η μελέτη του σχήματος, του μέγεθους, της υφής και της κατανομής των φυσικών αντικειμένων. [29]

[25] *Tagging* : Είναι η διαδικασία κατά την οποία βάζουμε tags (ετικέτες) σε ένα σύνολο πληροφορίων. Στα συστήματα πληροφορικής, το tag είναι μια μη ιεραρχική λέξη ή όρος που αποδίδεται σε ένα κομμάτι πληροφορίων (όπως ένα σελιδοδείκτη Internet, μια ψηφιακή εικόνα, ή ένα αρχείο του υπολογιστή). Αυτό το είδος των μεταδεδομένων βοηθά στην περιγραφή ένος στοιχείου και επιτρέπει να βρεθεί και πάλι από περιήγηση ή αναζήτηση. Οι ετικέτες γενικά επιλέγονται ανεπίσημα και προσωπικά από τον δημιουργό του στοιχείου ή το χρήστη, ανάλογα με το σύστημα. [30]

[26] *Information Retrieval tools* : είναι ένα σύνολο από εργαλεία που έχουν ως βασική δραστηριότητα την απόκτηση πόρων πληροφορίας σχετικών με κάποια ανάγκη για πληροφορίες από κάποια συλλογή πληροφοριακών πόρων. Οι αναζητήσεις μπορούν να βασίζονται σε μετα-δεδομένα ή σε πλήρες κείμενο. [31]

[27] *IE - Information Extraction* (Εξαγωγή πληροφοριών) : είναι η διαδικασία της αυτόματης εξαγωγής δομημένων πληροφοριών από μη δομημένα ή ημι-δομημένα αναγνώσιμα από μηχανήματα έγγραφα. Στις περισσότερες των περιπτώσεων αυτή η δραστηριότητα αφορά την επεξεργασία κειμένων ανθρώπινης γλώσσας μέσω της επεξεργασίας φυσικής γλώσσας (NLP). Πρόσφατες δραστηριότητες στην επεξεργασία των εγγράφων multimedia, όπως ο αυτόματος σχολιασμός και η εξαγωγή του περιεχομένου από εικόνες / ήχο / βίντεο θα μπορούσε να θεωρηθεί ως εξαγωγή πληροφοριών.

φαίνονται παρακάτω :

- **Computational Linguistics (CL)**: επιστήμη της γλώσσας που χρησιμοποιεί τον υπολογισμό ως εργαλείο έρευνας.
- **Language Engineering (LE)**: δημιουργία συστημάτων NLP των οποίων το κόστος και τα αποτελέσματα είναι μετρήσιμα και προβλέψιμα.
- **Software Architecture:** υποδομή πληροφορικής για την ανάπτυξη λογισμικού, συμπεριλαμβανομένης της ανάπτυξης περιβαλλόντων και πλαίσιων. Η πιο συνηθισμένη χρήση του όρου είναι για να υποδηλώσει μια μακρο-επίπεδη οργανωτική δομή για τα συστήματα λογισμικού
- **Software Architecture for Language engineering (SALE)**: υποδομή λογισμικού, αρχιτεκτονική και την ανάπτυξη εργαλείων για εφαρμοσμένα CL, NLP και LE συστήματα.

Συνεπώς, το GATE θα μπορούσε να εννοηθεί ως Software Architecture for Language Engineering. Στις επιστημονικές επιδιώξεις του NLP και CL, ο ρόλος του GATE είναι η υποστήριξη του πειραματισμού. Στο πλαίσιο αυτό, σημαντικά χαρακτηριστικά του GATE περιλαμβάνουν υποστήριξη για αυτοματοποιημένες μετρήσεις παρέχοντας τη δυνατότητα εύκολης επανάληψης των αποτελεσμάτων σε διαφορετικές τοποθεσίες και περιβάλλοντα, και τη μείωση των γενικών εξόδων της έρευνας με διάφορους τρόπους.

Το GATE ως αρχιτεκτονική υποδηλώνει ότι τα στοιχεία των συστημάτων λογισμικού που επεξεργάζονται τη φυσική γλώσσα, μπορούν να κατανεμηθούν σε διάφορες κατηγορίες από components, γνωστά και ως **resources**. Τα components είναι επαναχρησιμοποιήσιμα κομμάτια λογισμικού με σαφώς καθορισμένες διεπαφές, και είναι μια δημοφιλής αρχιτεκτονική μορφή, που χρησιμοποιείται στο Java Beans Sun και στο .Net της Microsoft. Τα components του GATE είναι εξειδικευμένοι τύποι του Java Bean και χωρίζονται σε τρεις κατηγορίες:

- **Language Resources (LRs):** αντιπροσωπεύουν οντότητες, όπως λεξικά, corpora, οντολογίες, κ.α.
- **Processing Resources (PRs):** αποτελούν οντότητες οι οποίες είναι κατά κύριο λόγο αλγοριθμικές, όπως αναλυτές, γεννήτριες, κ.α.
- **Visual Resources (VRs):** αποτελούν την οπτικοποίηση και την επεξεργασία στοιχείων που συμμετέχουν στο GUI[28].

Συλλογικά, το σύνολο των resources (πόρων) που ενσωματώνονται στο GATE είναι γνωστό ως **CREOLE** : Collection of REusable Objects for Language Engineering (δημαδή μια συλλογή από επαναχρησιμοποιήσιμα αντικείμενα για τη Γλωσσική Τεχνολογία) . Όλοι οι πόροι είναι συσκευασμένοι ως Java Archive (ή "JAR") αρχεία, συν κάποια δεδομένα διαμόρφωσης XML. Τα αρχεία JAR και XML διατίθενται στο GATE με την τοποθέτηση τους σε έναν web server, ή απλά τοποθετώντας τα στον τοπικό χώρο του αρχείου.

Όταν χρησιμοποιεί το GATE για την ανάπτυξη της γλωσσικά επεξεργαστικής λειτουργικότητας μιας εφαρμογής, ο προγραμματιστής θα χρησιμοποιήσει το GATE

---

[28] *GUI- Graphical User Interface* : είναι ένα είδος διεπαφής που επιτρέπει στους χρήστες να αλληλεπιδρούν με τις ηλεκτρονικές συσκευές μέσω γραφικών εικονιδίων και οπτικών ενδείξεων, όπως η δευτερογενής σημειογραφία, σε αντίθεση με τις διεπαφές βασιζόμενες σε κειμένα, τις δακτυλογραφημένες ετικέτες εντολών ή την περιήγηση κειμένων. Τα GUIs εισήχθησαν ως αντίδραση στην απότομη τάση εκμάθησης των διεπαφών γραμμής εντολών (Command-line interfaces, CLIS), που απαιτούσαν την αναγκαστική πληκτρολόγηση των εντολών. [32]

Developer και το GATE Embedded για την κατασκευή των πόρων των τριών προαναφερθέντων τύπων. Αυτό μπορεί να περιλαμβάνει τον προγραμματισμό ή την ανάπτυξη των Language Resources, όπως γραμματικές που χρησιμοποιούνται από τους ήδη υπάρχοντες Processing Resources, ή ένα μίγμα και των δύο. Το GATE Developer χρησιμοποιείται για την οπτικοποίηση των δομών δεδομένων που παράγονται και καταναλώνονται κατά τη διάρκεια της επεξεργασίας, καθώς και για τον εντοπισμό σφαλμάτων, τη μέτρηση των επιδόσεων και ούτω καθεξής. Το GATE Developer είναι ανάλογο με συστήματα όπως το Mathematica για τους μαθηματικούς, ή το JBuilder για τους προγραμματιστές σε Java: παρέχει ένα βολικό γραφικό περιβάλλον για την έρευνα και την ανάπτυξη λογισμικού επεξεργασίας γλώσσας. Όταν ένα κατάλληλο σύνολο από resources έχει αναπτυχθεί, μπορεί στη συνέχεια να ενσωματωθεί στην εφαρμογή του πελάτη χρησιμοποιώντας το GATE Embedded, το οποίο παρέχεται ως μια σειρά από JAR αρχεία.

Το GATE περιλαμβάνει πόρους για κοινές δομές δεδομένων και αλγορίθμων LE, συμπεριλαμβανομένων των εγγράφων, των corpora και διαφόρων ειδών σχολιασμών, ενός συνόλου στοιχείων ανάλυσης γλώσσας για Information Extraction (IE) και μιας σειράς από δεδομένα οπτικοποίησης και στοιχεία επεξεργασίας. Το GATE υποστηρίζει έγγραφα σε διάφορες μορφές, όπως XML, RTF, e-mail, HTML, SGML και απλού κειμένου. Σε όλες τις περιπτώσεις η μορφή αναλύεται και μετατρέπεται σε ένα ενιαίο μοντέλο σχολιασμού (annotation). Τα έγγραφα GATE, τα corpora και τα σχόλια αποθηκεύονται σε βάσεις δεδομένων διαφόρων ειδών, οπτικοποιούνται μέσω της ανάπτυξης περιβάλλοντος, και γίνεται πρόσβαση σε επίπεδο κώδικα, μέσω του framework. Το GATE framework παρέχει τη δυνατότητα δημιουργίας ενός συνόλου ατομικών Jape transducers το ένα μετά το άλλο. Αυτή η αλυσιδωτή σύνδεση επιτρέπει αργότερα τους Jape μετατροπείς να λειτουργούν κατά την έξοδο των προηγούμενων μετατροπέων Jape, οικοδομώντας έτσι όλο και περισσότερο πολύπλοκους σχολιασμούς και ενσωματώνοντας τα περισσότερα από τα συμφραζόμενα (σημασιολογία) του εγγράφου στους νέους σχολιασμούς. [2]

2.2. Εφαρμογές του GATE

2.2.1 GATE Developer

Το GATE Developer είναι το IDE του GATE, δηλαδή το γραφικό περιβάλλον του χρήστη. Πιο συγκεκριμένα, είναι ένα βολικό περιβάλλον για την έρευνα και την ανάπτυξη λογισμικού επεξεργασίας γλώσσας. Εκτός του ότι είναι από μόνο του ένα ισχυρό εργαλείο για την έρευνα, είναι επίσης πολύ χρήσιμο σε συνδυασμό με το GATE Embedded (το GATE API[29] με το οποίο λειτουργικότητες του GATE μπορούν να συμπεριληφθούν σε εφαρμογές χρήστη). Αυτό σημαίνει για παράδειγμα, ότι μπορεί να χρησιμοποιηθεί το GATE Developer για τη δημιουργία εφαρμογών που στη συνέχεια να ενσωματωθούν μέσω του API.

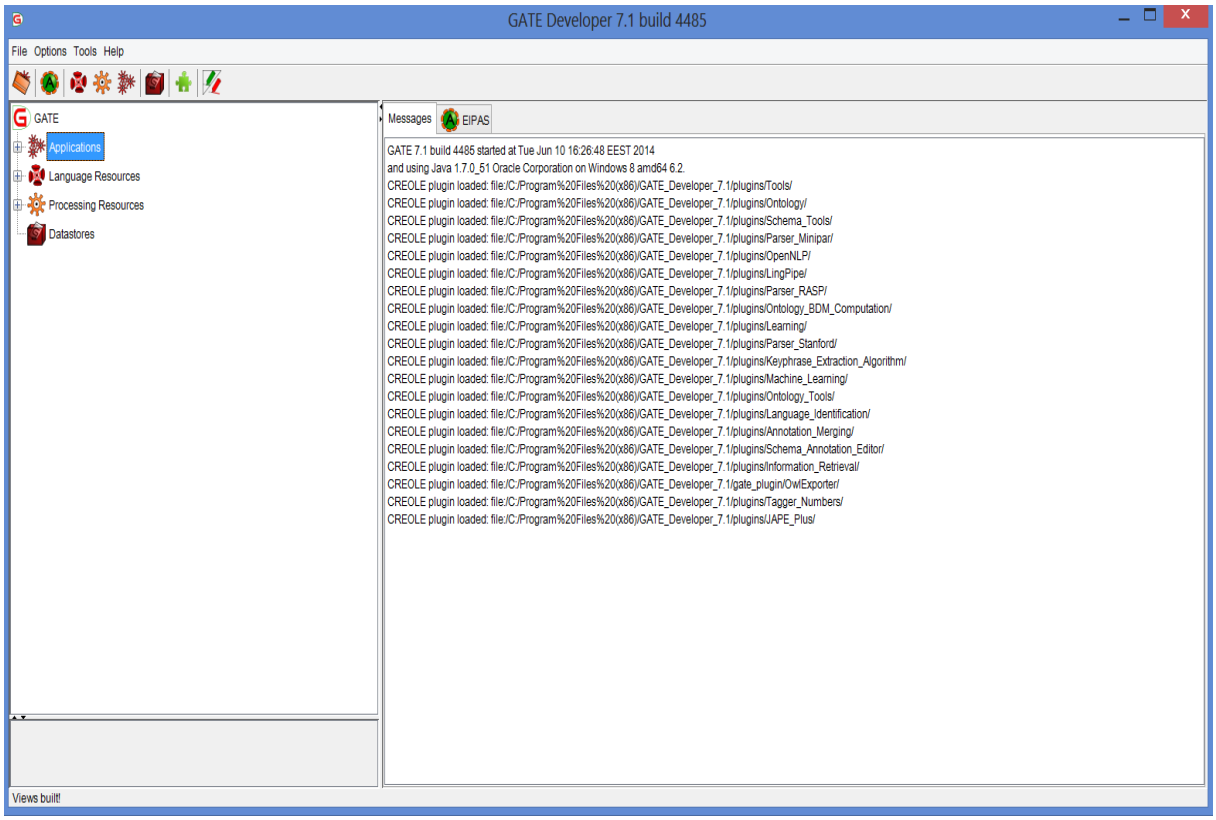
Η βασική δραστηριότητα του GATE είναι ο σχολιασμός εγγράφων

[29] *API* : Στον προγραμματισμό ηλεκτρονικών υπολογιστών, ένα application programming interface (μια διεπαφή προγραμματισμού εφαρμογών) καθορίζει πώς ορισμένα στοιχεία του λογισμικού θα πρέπει να αλληλεπιδρούν μεταξύ τους. [33]

(**annotation**), και όλη η λειτουργικότητα που θα εισαχθεί σε σχέση με αυτό. Όπως έχει προαναφερθεί σε ανώτερη υποσημείωση, οι σχολιασμοί αυτοί αποτελούν κάποιο είδος μεταδεδομένων (π.χ. σχόλια, επεξηγήσεις) που συνδέονται με κάποιο συγκεκριμένο τμήμα του κειμένου, της εικόνας ή άλλων δεδομένων με σκοπό να προσδιορίσουν κάποιες ιδιότητες και πληροφορίες σε αυτά. Με βάση τα ανώτερα, το GATE Developer περιλαμβάνει:

- Τα έγγραφα που πρέπει να σχολιαστούν (documents)
- Language Resources: δηλαδή corpora που περιέχουν σύνολα εγγράφων με σκοπό την ομαδοποίηση τους για εφαρμογή ενιαίων διαδικασιών σε αυτά
- Annotations: Σχολιασμούς που έχουν δημιουργηθεί σχετικά με τα έγγραφα
- Annotation types: Πεδία σχολιασμού, όπως «Όνομα» ή «Ημερομηνία», κ.α.
- Annotation Sets: Σύνολα σχολιασμού που περιλαμβάνουν ομάδες πεδίων σχολιασμού
- Processing Resources: Πόρους επεξεργασίας που δημιουργούν σχολιασμούς σε έγγραφα
- Applications: Εφαρμογές, που περιλαμβάνουν αλληλουχίες πόρων επεξεργασίας, και εφαρμόζονται σε ένα έγγραφο ή corpus.
- Datastore: περιλαμβάνει αποθετήρια για δεδομένα μεγάλου όγκου.

Στην παρακάτω εικόνα φαίνεται η μορφή του GATE Developer με τα αρχικά πεδία του (φαίνονται στην αριστερή στήλη).



Εικόνα 2.1. Αρχική σελίδα του GATE Developer

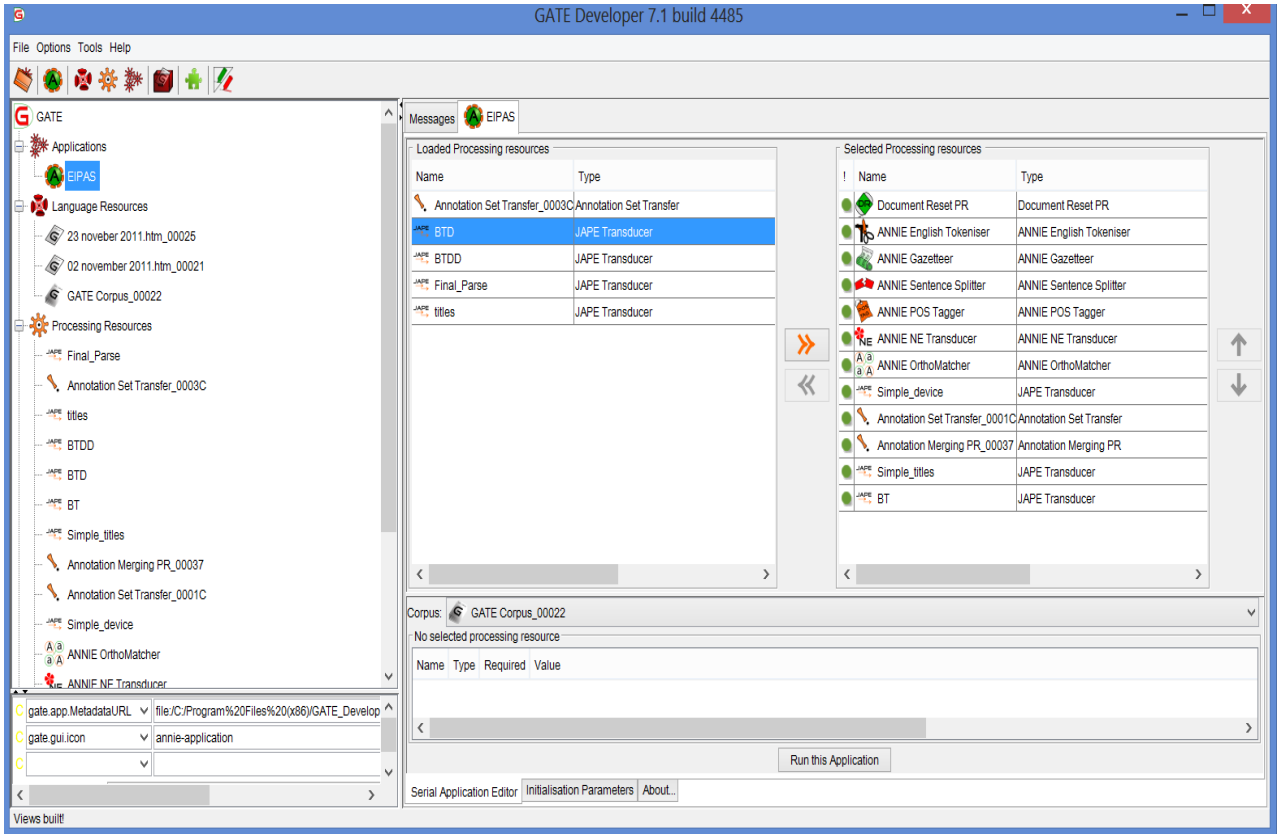
Η λειτουργία αυτού του προγράμματος απαιτεί τη ρύθμιση αρκετών από τα προαναφερθέντα πεδία. Αρχικά, πρέπει να εισαχθούν τα έγγραφα που θα γίνουν annotated. Αυτό μπορεί να γίνει με δύο τρόπους, είτε να δοθεί κάποια διεύθυνση

URL, είτε να γίνει χρήση κάποιου ήδη αποθηκευμένου εγγράφου σε μορφή XML. Αυτά για λόγους ευκολίας συνηθίζεται να ομαδοποιούνται σε κάποια σύνολα εγγράφων, που λέγονται corpora (corpus στον ενικό). Έτσι, μια εφαρμογή μπορεί να τρέξει σε κάποιο corpus με αποτέλεσμα να γίνουν τα annotations για όλα τα έγγραφα που περιέχει το corpus αυτό.

Στη συνέχεια, απαιτείται η δημιουργία της εφαρμογής που θα δημιουργήσει τα annotations (σχολιασμούς). Αυτή μπορεί να είναι είτε ένα πλήρες σύστημα εξαγωγής πληροφοριών ελεύθερο προς χρήση, που ονομάζεται ANNIE (a Nearly-New Information Extraction System), είτε κάποια άλλη εφαρμογή που θα έχει δημιουρήσει ο χρήστης. Υπάρχει, δηλαδή, η δυνατότητα ο ίδιος ο χρήστης του προγράμματος να ορίσει τη δική του εφαρμογή εξαρχής ή ακόμα να τροποποιήσει την ήδη υπάρχουσα (ANNIE). Γενικά υπάρχουν πέντε κατηγορίες εφαρμογών, οι εφαρμογές pipeline, οι corpus pipeline, οι conditional pipeline, οι conditional corpus pipeline και οι real-time corpus pipeline. Οι πρώτες απλώς ομαδοποιούν ένα σύνολο από PRs σε σειρά και τις εκτελούν σε αυτή τη σειρά σε κάποιο έγγραφο. Η κλάση που τις υλοποιεί λέγεται *SerialController*. Οι δεύτερες μπορούν να τρέξουν πάνω σε όλο το corpus, δηλαδή σε ένα σύνολο εγγράφων. Είναι ειδικά για LanguageAnalysers - PRs που εφαρμόζονται σε έγγραφα και corpora. Ένα corpus pipeline ανοίγει κάθε έγγραφο του corpus με σειρά, θέτει το έγγραφο αυτό ως παράμετρο χρόνου εκτέλεσης σε κάθε PR, τρέχει όλες τα PRs σχετικά με το corpus και στο τέλος κλείνει το έγγραφο. Η κλάση που τις υλοποιεί λέγεται *SerialAnalyserController*. Η διαφορά με τις **conditional** εφαρμογές είναι ότι σε αυτές δίνεται η δυνατότητα άμεσης επιλογής και αλλαγής των processing resources που θα τρέξουν ή όχι στο corpus ή το document. Αυτή η επιλογή μπορεί να χρησιμοποιηθεί για την προσωρινή και γρήγορη απενεργοποίηση ενός στοιχείου της εφαρμογής, για σκοπούς εντοπισμού σφαλμάτων (debugging) για παράδειγμα. Τέλος, όσον αφορά τις real-time εφαρμογές, υπάρχει μια επιπλέον παράμετρος (timeout parameter) που σετάρεται εξ αρχής και δηλώνει το μέγιστο χρόνο για την επεξεργασία ενός εγγράφου. Τα έγγραφα , λοιπόν, που χρειάζονται περισσότερο χρόνο για να ολοκληρωθεί η επεξεργασία τους, αγνοούνται και η εκτέλεση της εφαρμογής συνεχίζει στα επόμενα έγγραφα του corpus. Σε όλες, όμως, τις κατηγορίες εφαρμογών, προκύπτει ότι η βασική διαμόρφωση μιας εφαρμογης είναι τα resources τα οποία θα περιέχει σε συνδυασμό φυσικά με τη σειρά στην οποία θα εισαχθούν. Στην παρακάτω εικόνα φαίνεται ένα παράδειγμα μιας corpus pipeline εφαρμογής, των resources που έχουν γίνει Load, καθώς και αυτών που έχουν εισαχθεί στο corpus pipeline για να τρέξουν πάνω στα έγγραφα του επιλεγμένου corpus. Παρατηρούμε, λοιπόν, ότι κάποια resources, όπως το titles ή το BT έχουν γίνει load, αλλά δεν έχουν επιλεγεί ώστε να τρέξουν στην εφαρμογή. Αυτό θα εχει σαν αποτέλεσμα να μη συμμετάσχουν καθόλου στη διαδικασία δημιουργίας των annotations πάνω στο corpus GATE Corpus_00022. Φυσικά, δε θα μπορούσαμε να παραλήψουμε το γεγονός της pipeline ονομασίας των εφαρμογών, πράγμα που δεν είναι τυχαίο, αφού τα επιλεγμένα resources θα εκτελεστούν στη σειρά (από πάνω προς τα κάτω) μεταφέροντας πληροφορίες και δεδομένα από τα προηγούμενα στα επόμενα, διατηρωντας έτσι κάποιες ιδιότητες των pipelines[30] (σωληνώσεων).

[30] Software Pipeline: αγωγοί/σωληνώσεις που περιλαμβάνουν ένα σύνολο εντολών και η έξοδος της μιας λειτουργίας εντολής τροφοδοτείται αυτόματα στην επόμενη, μετά τη λειτουργία. [35]

Συνεπώς, στο προκείμενο θα εκτελεστεί πρώτα το resource Document Reset PR, μετά το ANNIE English Tokeniser, έπειτα το ANNIE Gazatteer κ.ο.κ.



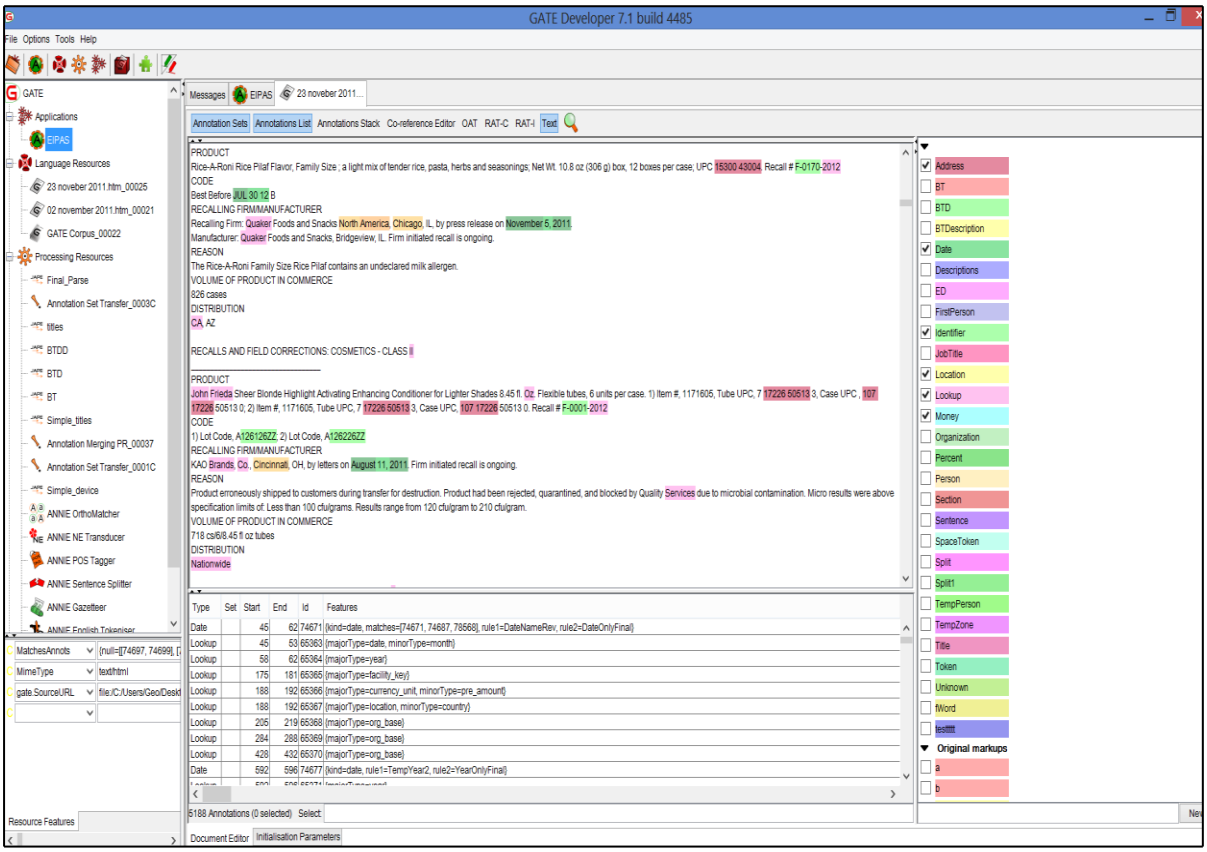
Εικόνα 2.2. Pipeline εφαρμογή με loaded και selected resources

Όσον αφορά τα processing resources, είναι αυτά που θα συγκροτήσουν την εφαρμογή. Γίνεται εμφανές, λοιπόν, ότι το μεγαλύτερο και ουσιαστικότερο ρόλο τον έχουν τα processing resources, καθώς αυτά είναι που θα περιλαμβάνουν τους κανόνες για τη δημιουργία και το χειρισμό των annotations στα έγγραφα.Υπάρχει μεγάλη πληθώρα από είδη αυτών των πόρων, λόγω των πολλαπλών λειτουργιών που επιτελούν, όπως συγχώνευση, μεταφορά, ανάλυση δεδομένων κ.α. Γι' αυτούς τους λόγους θα αναλυθούν στην επόμενη παράγραφο σχολαστικότερα. Βέβαια, πρέπει να γίνει κατανοητό το γεγονός ότι στις περισσότερες περιπτώσεις, προκειμένου να χρησιμοποιηθεί ένα συγκεκριμένο processing resource (και συγκεκριμένο language resource) θα πρέπει πρώτα να φορτωθεί το πρόσθετο (plugin) CREOLE που το περιέχει.

Σαν τελικό στάδιο υπάρχουν τα datastores, τα οποία βοηθούν στη φόρτωση και εκτέλεση των εγγράφων. Όταν τα corpora είναι μεγάλα, η διαθέσιμη μνήμη ενδέχεται να μην είναι επαρκής για το ταυτόχρονο άνοιγμα όλων των εγγράφων. Η λειτουργία του datastore παρέχει τη δυνατότητα να αποθηκευτούν έγγραφα στο δίσκο και να ανοιχτούν μόνο μία φορά για επεξεργασία. Αυτό σημαίνει ότι πολύ μεγαλύτερα corpora μπορούν να χρησιμοποιηθούν. Ένα datastore μπορεί επίσης να είναι χρήσιμο για την αποθήκευση εγγράφων με ένα αποτελεσματικό και χωρίς απώλειες τρόπο. Γενικά, με την εκτέλεση μιας εφαρμογής σε ένα datastore corpus, κάθε έγγραφο θα φορτωθεί, επεξεργαστεί, αποθηκευτεί και στη συνέχεια θα εκφορτωθεί. Έτσι, ανά πάσα στιγμή μόνο ένα έγγραφο θα είναι φορτωμένο από το

datastore corpus. Αυτό εμποδίζει μεν την έλλειψη μνήμης (memory shortage), όμως είναι λίγο πιο αργή διαδικασία σε σχέση με τη φόρτωση όλων των εγγράφων εξ'αρχής. Τα επεξεργασμένα έγγραφα αποθηκεύονται αυτόματα πίσω στο datastore, έτσι ώστε να μπορεί να γίνει χρήση των αντιγράφων του datastore για πολλαπλούς σκοπούς (π.χ. πειραματικούς).

Το σύνολο των annotations θα φανεί αφού πρώτα τρέξει η εφαρμογή πάνω στο επιλεγμένο corpus. Τότε, με το άνοιγμα κάθε εγγράφου που ανήκει σ'αυτό το corpus και με κατάλληλη επιλογή θα εμφανιζονται τα Annotation Sets, τα πεδία αυτών και πολλά άλλα στοιχεία. Τα πεδία που θα επιλεγούν (στο αριστερό τμήμα) θα εμφανίσουν τα αντίστοιχα κομμάτια του κειμένου σε έγχρωμο φόντο, ενώ γίνεται επίσης δυνατή η προβολή της λίστας από annotations ή άλλων επιλογών (π.χ. annotation stack). Στην παρακάτω εικόνα φαίνονται τα πεδία που έχουν επιλεγεί, τα annotations σε αντίστοιχα χρώματα καθώς και η λίστα τους στο κάτω μέρος.



Εικόνα 2.3. Επιλογή annotation sets και εμφάνιση των αντίστοιχων annotations

Τέλος, παρόλο που οι μέχρι τώρα αναφορές επεξηγούν πως οι εφαρμογές χρησιμοποιούνται για αυτόματο σχολιασμό των εγγράφων, υπάρχει ωστόσο και η δυνατότητα σχολιασμού στο χέρι, π.χ. από το χρήστη, ή ημι-αυτόματα, εκτελώντας μια εφαρμογή πάνω στο corpus και στη συνέχεια κάνοντας διόρθωση / προσθήκη νέων σχολίων με το χέρι.

Συμπερασματικά, το GATE Developer είναι ένα πρόγραμμα εύχρηστο και παράλληλα πολύ χρήσιμο για τον τομέα του language engineering. Αφενός, δίνει τη δυνατότητα μετατροπής του κεμένου σε ένα σύνολο από annotations, επιτρέποντας

έτσι την κατηγοριοποίηση του σύμφωνα με σημασιολογικούς, νοηματικούς, συντακτικούς και άλλους παράγοντες, καθώς και την άμεση προσπέλαση του από κάθε χρήστη - με εύκολο τρόπο μπορεί κάποιος να εισαγάγει ένα έγγραφο και να αναγνωρίσει, για παράδειγμα, όλες τις τοποθεσίες ή ημερομηνίες που αυτό περιέχει, απλά τρέχοντας την εφαρμογή ANNIE και επιλεγοντας τα κατάλληλα πεδία των annotation sets. Αφεταίρου, ίσως η πιο σημαντική συμβολή του είναι το γεγονός ότι μπορεί να συνδυαστεί με άλλα προγράμματα λογισμικού, κάνοντας έτσι πιο διαδραστική και κομβική τη λειτουργία του. Υπάρχει, για παράδειγμα, επιλογή που σώζει την υπάρχουσα κατάσταση, συμπεριλαμβανομένων των corpora, των processing resources και των εφαρμογών που υπάρχουν, σε ένα αρχείο εύκολα επαναχρησιμοποιήσιμο. Συγκεκριμένα, ένα "**.state**" αρχείο μπορεί να φορτωθεί οποιαδήποτε άλλη στιγμή στο GATE Developer με ακριβώς ίδια χαρακτηριστικά, ή ακόμα και να φορτωθεί σε κάποιο άλλο πρόγραμμα και να επεξεργαστεί εκεί. Μπορεί, δηλαδή, να γίνει χρήση του στο Eclipse, όπου με κατάλληλα imports και εντολές, να τρέξει παράγοντας τα ίδια αποτελέσματα. Το σημείο, όμως, που θα καταστήσει το GATE Developer ως χρήσιμο λογισμικό, είναι η παραγωγή ενδιάμεσων αποτελεσμάτων και η δυνατότητα της εύκολης χρήσης και περεταίρω επεξεργασίας τους με σκοπό την επίλυση ποικίλων προβλημάτων. Συνεπώς, ο χρήστης με σχετικά εύκολο τρόπο μπορεί να ορίσει μια εφαρμογή, να την τρέξει σε κάποιο corpus, να δει άμεσα τα αποτελεσματά της, να κάνει το αντίστοιχο debugging και έπειτα, αν βεβαιωθεί ότι λειτουργεί σωστά, να χρησιμοποιήσει αυτά τα αποτελέσματα για επιπλέον ανάλυση και επεξεργασία σε άλλα γραφικά περιβάλλοντα.

2.2.2 ANNIE: a Nearly-New Information Extraction System

Το GATE διανέμεται με ένα IE σύστημα που ονομάζεται ANNIE, a Nearly-New Information Extraction System (που αναπτύχθηκε από τους Hamish Cunningham, Valentin Tablan, Diana Maynard, Kalina Bontcheva, Marin Dimitrov και άλλους). Το ANNIE βασίζεται σε αλγόριθμους πεπερασμένης κατάστασης και στη γλώσσα Jape, στην οποία θα ανφερθούμε αναλυτικότερα σε επόμενο κεφάλαιο και αποτελεί το βασικό σύστημα του GATE για εξαγωγή πληροφοριών από έγγραφα. Ουσιαστικά, είναι μια εφαρμογή που περιλαμβάνει έξι processing resources με συγκεκριμένη σειρά και δύναται να δημιουργήσει annotations που θα κατηγοριοποιηθούν στις εξής ομάδες: Person, Location, Organization, Date, Address, Money και Percent. Υπάρχει φυσικά και η δυνατότητα της επέκτασης αυτής της εφαρμογής με την προσθήκη νέων processing resources με σκοπό την εξαγωγή νέων πληροφοριών. Οι πόροι επεξεργασίας που χρησιμοποιούνται by default (ως προεπιλογή) είναι οι εξής:

- ***Document Reset PR*:**

  Επιτρέπει το έγγραφο να επαναφέρθεί στην αρχική του κατάσταση, αφαιρώντας όλα τα σετ σχολιασμού (annotation sets) και το περιεχόμενό τους, εκτός από αυτό που περιέχει την ανάλυση μορφής του εγγράφου (Original Markups). Υπάρχουν, ακόμα, αρκετές επιλογές, λόγω των run-time παραμέτρων που μπορούν να αφαιρέσουν τα original markups, ή κάποιους άλλους τύπους από annotations, ή ακόμα να γίνει επιλογή των annotation sets που θα επιλεγούν

γενικότερα.

- ***ANNIE English Tokeniser*:**

Χωρίζει το κείμενο σε πολύ απλά γλωσσικά στοιχεία (tokens) όπως αριθμούς, σημεία στίξης και λέξεις διαφόρων τύπων. Για παράδειγμα, γίνεται διάκριση ανάμεσα στις λέξεις με κεφαλαία σε σχέση με αυτές με πεζά, καθώς και μεταξύ ορισμένων τύπων στίξης. Γενικά, ο στόχος είναι να περιοριστεί το έργο του tokeniser, να μεγιστοποιηθεί η απόδοση και να επιτευχθεί μεγαλύτερη ευελιξία, θέτοντας το βάρος στους κανόνες γραμματικής, οι οποίοι είναι πιο προσαρμόσιμοι.

Οι κανόνες του tokeniser έχουν δύο μέλη, το αριστερό (LHS – left hand side) και το δεξί (RHS – right hand side) . Το αριστερό είναι μια κανονική έκφραση, η οποία πρέπει να συνδυάζεται με τα δεδομένα εισόδου, ενώ το δεξί περιγράφει τα annotations που θα προστεθούν στο annotation set. Τα δύο αυτά μέλη χωρίζονται μεταξύ τους με το “>”, δηλαδή είναι της μορφής {LHS} > {RHS}. Για παράδειγμα, ο παρακάτω κανόνας αναφέρεται στο αν μια λέξη ξεκινάει με κεφαλαίο γράμμα:

```
‘UPPERCASE_LETTER’ ‘LOWERCASE_LETTER’* >
Token;orth=upperInitial;kind=word;
```

Ουσιαστικά, λέει ότι η συγκεκριμένη λέξη πρέπει να ξεκινάει με κεφαλαίο γράμμα (uppercase_letter), ενώ στη συνεχεια πρέπει να υπάρχει μια ακολουθία από μηδέν ή περισσότερα πεζά γράμματα (lowercase_letter*). Ο τύπος αυτού του annotation θα είναι “Token”, ενώ το χαρακτηριστικό “orth” (ορθογραφία) θα έχει την τιμή “upperInitial” και το “kind” την τιμή “word”.

Πιθανοί τύποι από **tokens** μπορεί να είναι:

- *Λέξεις* - Μια λέξη ορίζεται ως κάθε σύνολο συνεχόμενων κεφαλαίων ή πεζών γραμμάτων, συμπεριλαμβανομένης μιας παύλας (αλλά χωρίς άλλες μορφές στίξης). Μια λέξη έχει επίσης το χαρακτηριστικό “orth” το οποίο έχει τέσσερις πιθανές τιμές (upperInitial, allCaps, lowerCase, mixedCaps).
- *Αριθμός* - Ένας αριθμός ορίζεται ως οποιοδήποτε συνδυασμός από διαδοχικά ψηφία. Δεν υπάρχουν υποδιαιρέσεις των αριθμών.
- *Σύμβολο* – υπάρχουν δύο τύποι συμβόλων, οι οποίοι ορίζονται ως εξής: σύμβολα του νομίσματος (π.χ. «$», «£») και άλλα σύμβολα (π.χ. '&', '^'). Αυτά αντιπροσωπεύονται από οποιονδήποτε αριθμό διαδοχικών συμβόλων νομίσματος ή άλλων συμβόλων (αντίστοιχα).
- *Στίξη* - Τρεις τύποι των σημείων στίξης ορίζονται: start_punctuation (π.χ. “(”), end_punctuation (π.χ. “)”), και άλλα σημεία στίξης (π.χ. “:”). Κάθε σύμβολο στίξης είναι ένα ξεχωριστό token.
- *Κενός χαρακτήρας (space token)* - Τα λευκά διαστήματα χωρίζονται σε δύο τύπους SpaceToken, χώρου και ελέγχου, ανάλογα με το αν είναι καθαρά χωρικοί χαρακτήρες ή χαρακτήρες ελέγχου. Οποιοδήποτε συνεχόμενο (και ομογενές) σύνολο χωρικών χαρακτήρων ή χαρακτήρων ελέγχου ορίζεται ως SpaceToken.

Η παραπάνω περιγραφή ισχύει και για τον by default tokeniser (που υπάρχει ως προεπιλογή). Ωστόσο, εναλλακτικά tokenisers μπορούν να δημιουργηθούν εάν είναι απαραίτητο. Η επιλογή του tokeniser κατόπιν προσδιορίζεται κατά τη στιγμή της επεξεργασίας κειμένου. Συγκεκριμένα, όσον αφορά το English tokeniser που χρησιμοποιείται στο ANNIE, θα λέγαμε ότι είναι ένα processing resource που περιλαμβάνει ένα κανονικό tokeniser και ένα Jape μετατροπέα (βλ. παρακάτω). Ο μετατροπέας έχει το ρόλο της προσαρμογής της γενικής εξόδου του

tokeniser προς τις απαιτήσεις του αγγλικού μέρους του λόγου. Μια τέτοια προσαρμογή είναι η ένωση μαζί σε ένα token των κατασκευασμάτων όπως « '30 », « Cause », « 'em », « ’N » , « ’S », « ’s », « ’T », « ’d », « ’ll », « ’m », « ’re », « ’til », « ’ve », κλπ. Ένα άλλο καθήκον του μετατροπέα είναι να μετατρέπει αρνητικές δομές, όπως « don’t », από τρία token (« don », « ’ » και « t ») σε δύο (« do » και « n’t »). Το English Tokeniser πρέπει πάντα να χρησιμοποιείται σε αγγλικά κείμενα τα οποία πρέπει να υποστούν επεξεργασία μετά από το POS Tagger.

- ***ANNIE Gazetteer*:**

Ο ρόλος του gazetteer είναι να αναγνωρίσει την ύπαρξη ονομάτων στο κείμενο, τα οποία βασίζονται σε συγκεκριμένους καταλόγους. Οι κατάλογοι gazetteer που χρησιμοποιούνται, είναι αρχεία απλού κειμένου με μία καταχώρηση ανά γραμμή. Κάθε λίστα είναι ένα .lst αρχείο, που περιλαμβάνει μια σειρά από ονόματα, όπως τα ονόματα των πόλεων, των οργανώσεων, των ημερών της εβδομάδας, κλπ. Παρακάτω φαίνεται ένα μικρό μέρος της λίστας από τις χώρες του κόσμου (οι οποίες είναι σε αλφαβητική σειρά):

Afghanistan
Afrique
Albania
Albanie
Alderney
Algeria
Algérie
Allemagne
America
Amérique
Amériques
American Samoa
Andorra
Andorre
Angleterre
Anglo-Normandes
Angola
Anguilla
Antigua and Barbuda
Antigua et Barbuda
Antilles
Antilles Néerlandaises

Υπάρχουν ακόμα και ένα αρχείο .def, που χρησιμοποιείται ως ευρετήριο γι’ αυτές τις λίστες. Επίσης, για κάθε annotated λίστα απαραίτητος είναι ο ορισμός κάποιων τύπων (π.χ. major), ενώ για άλλους προαιρετικός (π.χ. minor). Από προεπιλογή, το Gazetteer processing resource δημιουργεί ένα annotation με όνομα Lookup για κάθε gazetteer είσοδο που βρίσκει στο κείμενο. Οι λίστες αυτές μεταγλωτίζονται σε μηχανές πεπερασμένων καταστάσεων. Κάθε token κειμένου που συνδυάζεται με αυτές τις μηχανές θα πρέπει γίνεται annotated με χαρακτηριστικά που καθορίζουν τους major και minor τύπους. Στη συνέχεια, κανόνες γραμματικής ορίζουν τους τύπους που πρέπει να προσδιορίζονται σε ειδικές περιστάσεις. Κάθε λίστα gazetteer θα πρέπει να βρίσκεται στον ίδιο κατάλογο με το αρχείο ευρετηρίου (αρχείο .def). Τέλος, υπάρχουν παράμετροι αρχικοποίησης (init-time parameters), αλλά και παράμετροι χρόνου εκτέλεσης

(run-time parameters) για τη ρύθμιση των επιλογών του Gazetteer και έτσι τον ακριβή πρσδιορισμό της λειτουργίας του.

- ***ANNIE Sentence Splitter*:**

Είναι ένα σύνολο από μετατροπείς πεπερασμένων καταστάσεων, οι οποίοι χωρίζουν το κείμενο σε προτάσεις. Αυτό το resource είναι απαραίτητο για τον tagger. Ο splitter χρησιμοποιεί μια λίστα gazetteer από συντομογραφίες, που βοηθάει στη διάκριση στοιχείων που χωρίζουν προτάσεις από άλλα είδη αυτών. Κάθε πρόταση γίνεται annotated με τον τύπο "Sentence", ενώ κάθε στοιχείο χωρισμού πρότασης (όπως μια τελεία) γίνεται annotated με τον τύπο "Split".

- ***ANNIE POS Tagger:***

Γενικά, ο tagger είναι μια τροποποιημένη έκδοση του Brill tagger, ο οποίος παράγει μια ετικέτα (tag) με το μέρος του λόγου κάθε λέξης ή συμβόλου ως σχολιασμό. Ο tagger χρησιμοποιεί ένα προεπιλεγμένο λεξικό και ένα σύνολο κανόνων. Και τα δύο αυτά μπορούν να τροποποιηθούν με το χέρι εάν είναι απαραίτητο. Υπάρχουν δύο επιπλέον λεξικά - ένα για τα κείμενα που όλα είναι κεφαλαία (lexicon_cap), και ένα για τα κείμενα που όλα είναι πεζά (lexicon_lower). Για τη χρήση κάποιου απ'αυτά, το προεπιλεγμένο λεξικό θα πρέπει να αντικατασταθεί με το κατάλληλο λεξικό κατά το χρόνο φόρτωσης. Το προεπιλεγμένο σύνολο κανόνων θα πρέπει να εξακολουθήσει να χρησιμοποιείται στην περίπτωση αυτή. Το ANNIE POS Tagger περιλαμβάνει και αυτό πολλές παραμέτρους (init-time και run-time) για ρύθμιση διαφόρων επιλογών, όπως το url για το αρχείο λεξικού, το όνομα του annotation set που θα χρησιμοοποιηθεία ως έξοδος κ.α.

Ο semantic tagger του ANNIE βασίζεται στη γλώσσα JAPE η οποία θα αναλυθεί στο επόμενο κεφάλαιο. Περιέχει κανόνες που ενεργούν σε annotations που έχουν ανατεθεί σε προηγούμενες φάσεις, προκειμένου να παράγουν έξοδο από σχολιασμένες (annotated) οντότητες. Οι προεπιλεγμένοι τύποι σχολιασμού (default annotation types), τα χαρακτηριστικά και πιθανές τιμές αυτών, που παράγονται από το ANNIE έχουν ως εξής:

- *Person* (άτομο)
  - gender: male (αρσενικό), female (θυλικό)
- *Location* (μέρος)
  - locType: region (περιοχή), airport (αεροδρόμιο), city (πόλη), country (χώρα), county (κομητεία), province (επαρχία), other
- *Organization*
  - orgType: company (εταιρία), department (υπουργείο), government (κυβέρνηση), newspaper (εφημερίδα), team (ομάδα), other
- *Money* (λεφτά)
- *Percent* (ποσοστό)
- *Date* (ημερομηνία)
  - kind: date (ημερομηνία), time (ώρα), dateTime (ώρα και ημερομηνία)
- *Address* (διεύθυνση)
  - kind: email (μήνυμα ηλεκτρονικού ταχυδρομείου), url (σελίδα στο internet), phone (τηλέφωνο), postcode (ταχυδρομικός κώδικας), complete, ip (διεύθυνση προτοκόλλου internet), other
- *Identifier* (αναγνωριστικό)
- *Unknown* (άγνωστο)

Σημειώστε ότι ορισμένες από αυτές τις τιμές χαρακτηριστικών δημιουργούνται αυτόματα από τις λίστες gazetteer, οπότε με την τροποποιήση του αρχείου ορισμού της λίστας gazetteer, αυτά θα μπορούσαν να αλλάξουν. Σημειώστε επίσης ότι άλλα σχόλια, χαρακτηριστικά και τιμές δημιουργούνται επίσης από το ANNIE για debugging σκοπούς: για παράδειγμα, τα περισσότερα σχόλια έχουν ένα κανόνα χαρακτηριστικού που δίνει πληροφορίες σχετικά με το ποιός κανόνας(ες) χρησιμοποιήθηκε για να δημιουργηθεί αυτό το annotation. Ο Unknown τύπος σχολιασμού χρησιμοποιείται από τον πόρο επεξεργασίας Orthomatcher και αποτελείται από οποιοδήποτε κατάλληλο ουσιαστικό που δεν έχει ήδη εντοπιστεί.

- ***ANNIE NE Transducer*** **:**

Στο πλαίσιο του GATE συστήματος και του NLP γενικότερα, ο transducer είναι μια πεπερασμένη κατάσταση μετατροπέα με δύο ταινίες: μια ταινία εισόδου και μια ταινία εξόδου. Από αυτή την άποψη, ένας μετατροπέας κάνει τη μεταγωγή (δηλαδή, τη μετάφραση) του περιεχομένου της ταινίας εισόδου στην ταινία εξόδου, αποδεχόμενη ένα string στην ταινία εισόδου και παράγοντας ένα άλλο string στην ταινία εξόδου. Γενικά, ο transducer δημιουργεί τους κανόνες σχηματισμού των annotations κάνοντας χρήση της γραμματικής Jape η οποία θα σχολιαστεί αναλυτικά στο επόμενο κεφαλαιο.

- ***ANNIE OrthoMatcher*** :

Προσθέτει σχέσεις ταυτότητας μεταξύ επώνυμων οντοτήτων που βρέθηκαν από το σημασιολογικό επισημειωτή (semantic tagger), προκειμένου να εκτελεστεί coreference , δηλαδή όπως προαναφέρθηκε ένα είδος ταιριάσματος (βλ. σελίδα 25). Δε θα βρει , λοιπόν, μια νέα επώνυμη οντότητα, αλλά μπορεί να αναθέσει έναν τύπο σε ένα αταξινόμητο όνομα (ένα Unknown annotation), χρησιμοποιώντας τον τύπο του ονόματος με το οποίο ταιριάζει. Οι κανόνες ταιριάσματος προβάλονται μόνον αν τα ονόματα που συγκρίνονται είναι και τα δύο του ίδιου τύπου, δηλαδή και τα δύο ήδη χαρακτηρισμένα ως π.χ. οργανώσεις, ή αν ένας από αυτούς έχει χαρακτηριστεί ως «unknown». Αυτό αποτρέπει την επανακατηγοριοποίηση ενός ήδη ταξινομημένου ονόματος.

Ένας πίνακας αντιστοιχίας από ψευδώνυμα (**aliases**) χρησιμοποιείται για την καταγραφή μη ταιριασμένων strings, τα οποία αντιπροσωπεύουν την ίδια οντότητα, π.χ. «IBM» και «Big Blue», «Coca-Cola» και «Coke». Υπάρχει επίσης ένας πίνακας πλαστών αντιστοιχίσεων, δηλαδή ταιριάζουν strings που δεν αντιπροσωπεύουν την ίδια οντότητα, π.χ. «BT Wireless» και «BT Cellnet» (που είναι δύο διαφορετικές οργανώσεις). Ο κατάλογος των πινάκων που πρέπει να χρησιμοποιείται είναι μια παράμετρος του χρόνου φόρτωσης του orthomatcher: μια προεπιλεγμένη λίστα έχει οριστεί, αλλά μπορεί να αλλάξει ανάλογα με τις ανάγκες.

Ο orthomatcher δημιουργεί ένα πίνακα από strings, τύπους και αναγνωριστικά (IDs) όλων των annotation των ονομάτων, το οποίο στη συνέχεια περνάει σε μια λειτουργία σύγκρισης συμβολοσειράς κατά ζεύγη όλων των καταχωρήσεων. [2]

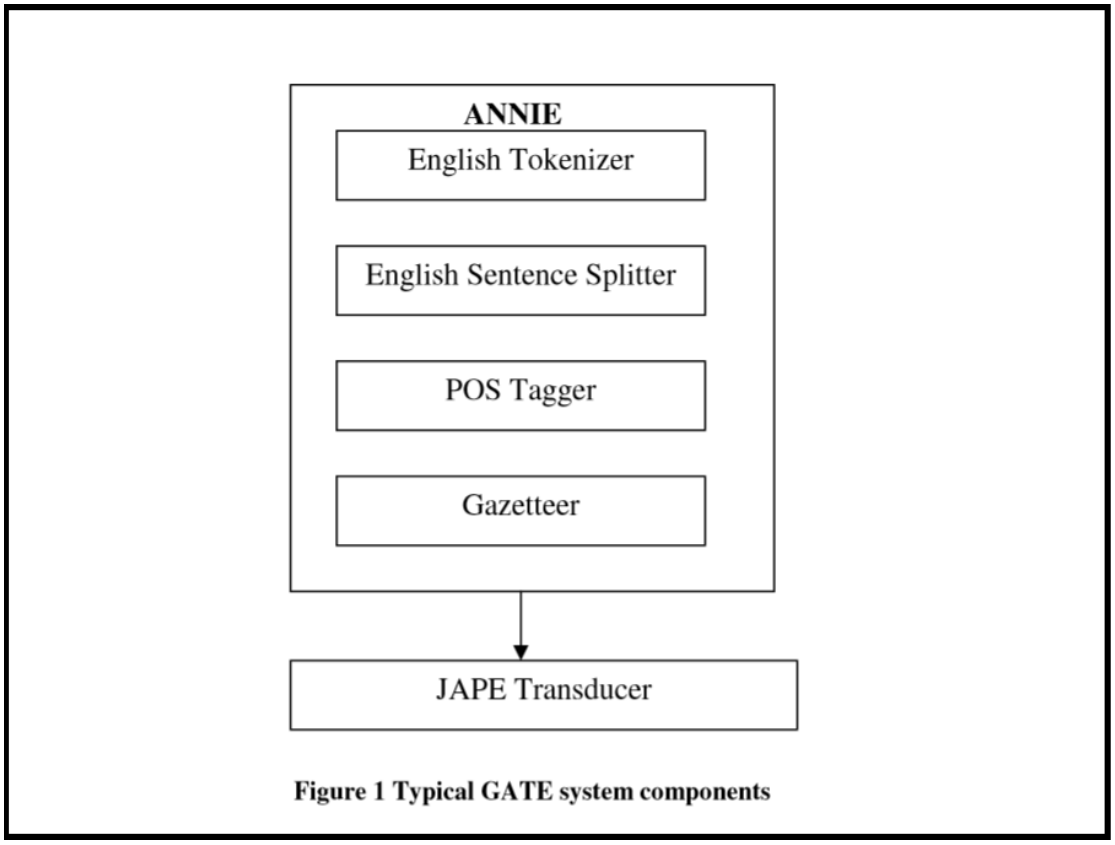


Figure 1 Typical GATE system components

Εικόνα 2.4. Τα στοιχεία που συνθέτουν ένα τυπικό σύστημα GATE [2]

2.3. Το GATE Embedded

2.3.1 Εισαγωγή στο GATE Embedded

Όπως έχει προαναφερθεί, η μεγαλύτερη χρησιμότητα του GATE είναι ότι μπορεί να ενσωματωθεί και σε άλλες εφαρμογές με σχετικά απλό τρόπο. Με χρήση, δηλαδή, κάποιον αρχείων .jar και κατάλληλη τροποποίηση του classpath, γίνεται να φορτωθεί και να τρέξουν εφαρμογές του (όπως το ANNIE) σε κάποιο πρόγραμμα Java, σαν το Eclipse. Παρακάτω φαίνονται διάφορες γενικές κλάσεις και διαπροσωπείες για χρήση του GATE σε κάποιο virtual machine της Java:

- Η αρχικοποίηση του: *gate.Gate.init()*
- δημιουργία του ANNIE ως corpus pipeline εφαρμογή:

```
SerialAnalyserController controller = (SerialAnalyserController)
PersistenceManager.loadObjectFromFile(new File(new File( Gate.getPluginsHome(), A
NNIEConstants.PLUGIN_DIR), ANNIEConstants.DEFAULT_FILE));
```

- Φόρτωση (load)των plugins για μετέπειτα χρήση διαφόρων resources:

```
Gate.getCreoleRegister().registerDirectories(New File(Gate.getPluginsHome(), "Tools").
toURL());
```

- Για δημιουργία ενός processing resource υπάρχουν δύο τρόποι:
  - Χρήση του Factory συμπεριλαμβανομένων των τιμών των διαφόρων παραμέτρων:

```
FeatureMap params = Factory.newFeatureMap();
ProcessingResource morpher = (ProcessingResource)
 Factory.createResource("gate.creole.morph.Morph",params);
```

Σε περίπτωση, όπως στην παραπάνω, όπου δεν έχουν οριστεί παράμετροι για τη δημιοργία του resource, υπάρχουν κάποιες default παράμετροι που εννοούνται.

- Χρήση του GATE Developer και συγκεκριμένα κάποιου αποθηκευμένου .state αρχείου που θα περιέχει σεταρισμένη την εφαρμογή:

```
CorpusController controller = (CorpusController)
PersistenceManager.loadObjectFromFile(new File("savedState.state"));
```

- Για τη δημιουργία ενός document από κάποιο url:

```
URL u = new URL("http://gate.ac.uk/hamish/");
FeatureMap params = Factory.newFeatureMap();
params.put("sourceUrl", u);
Document doc = (Document) Factory.createResource("gate.corpora.DocumentI
mpl", params);
```

- Για τη διαγραφή ενός resource: *deleteResource (Resource res)*

Γενικά, προκειμένου να χρησιμοποιηθεί ένας πόρος CREOLE, το σχετικό CREOLE plugin πρέπει να φορτωθεί. Τα Processing Resources, Visual Resources και Language Resources εκτός του εγγράφου, του Corpus και του DataStore, απαιτούν πρώτα τη φόρτωση του κατάλληλου plugin. Κατά τη χρήση του εγγράφου, του Corpus ή του DataStore, δεν χρειάζεται να φορτωθεί αρχικά κανένα plugin.

2.3.2 Language resources

Αυτή η ενότητα περιγράφει την εφαρμογή των εγγράφων και των corpora στο GATE.

- **Documents**: Όσον αφορά τα έγγραφα, το περιεχόμενό τους μπορεί να επεξεργαστεί με τη μέθοδο *gate.DocumentContent* και με κατάλληλη επιλογή τιμών των παραμέτρων.
- **Feature Map**: Ένα Feature Map είναι ένας χάρτης Java (δηλαδή υλοποιείται από το περιβάλλον java.util.Map) που κατέχει τα ζεύγη <attribute-name, attribute-value>. Τα ονόματα των γνωρισμάτων είναι strings, ενώ οι τιμές μπορεί να είναι οποιαδήποτε αντικείμενα Java. Feature Maps δημιουργούνται με τη χρήση της μεθόδου *gate.Factory.newFeatureMap().*
- **Annotation Set**: Ένα έγγραφο GATE μπορεί να έχει ένα ή περισσότερα στρώματα (layers ή sets) από annotations - ένα ανώνυμο, (ονομάζεται επίσης και default), και όσα επονομασμένα είναι απαραίτητα. Ένα τέτοιο annotation set είναι οργανωμένο στη μορφή ενός Directed Acyclic Graph (DAG) – κατευθυνόμενος ακυκλικός γράφος - , στον οποίο οι κόμβοι είναι τα συγκεκριμένα σημεία κειμένου του εγγράφου, ενώ τα τόξα περιλαμβάνουν τις περιγραφές και τα χαρακτηριστικά των annotations, που αναφέρονται στο κομμάτι κειμένου από τη θέση που υποδεικνύεται από τον αρχικό κόμβο στον

κόμβο τέλους. Παρακάτω φαίνεται μια εικόνα περιγραφής αυτού του μοντέλου.

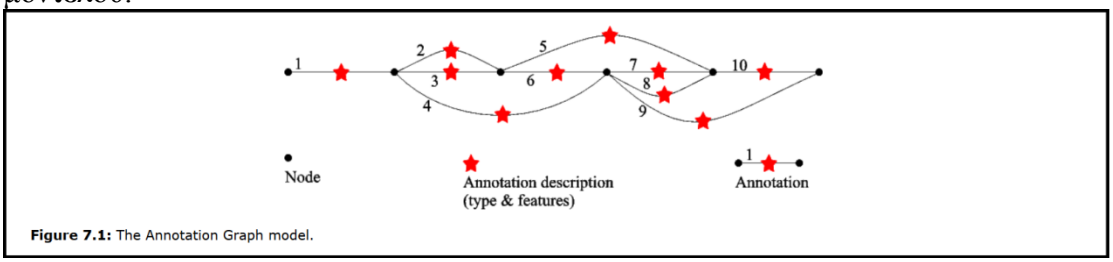


Figure 7.1: The Annotation Graph model.

Εικόνα 2.5. Annotation set με τη μορφή του DAG

Ένα annotation set κατέχει ένα αριθμό από annotations και διατηρεί μια σειρά δεικτών προκειμένου να παρέχουν γρήγορη πρόσβαση στους υπάρχοντες σχολιασμούς. Το annotation set του GATE ορίζεται από τη διαπροσωπεία *gate.AnnotationSet* και η προεπιλεγμένη κλάση στην οποία παρέχεται είναι η *gate.annotation.AnnotationSetImpl*. Τα σύνολα σχολιασμού δημιουργούνται από το έγγραφο, όπως απαιτείται. Την πρώτη φορά που ένα συγκεκριμένο σύνολο σχολιασμού ζητείται από ένα έγγραφο, θα δημιουργηθεί αν δεν υπάρχει. Στο επόμενο παράδειγμα φαίνεται ο τρόπος που επιλέγονται όλα τα annotations του συγκεκριμένου τύπου "person" από αρστερά προς τα δεξιά:

```
AnnotationSet annSet = ...;
String type = "Person";
//Get all person annotations
AnnotationSet persSet = annSet.get(type);
//Sort the annotations
List persList = new ArrayList(persSet);
Collections.sort(persList, new gate.util.OffsetComparator());
//Iterate
Iterator persIter = persList.iterator();
while(persIter.hasNext()){
...
}
```

- **Annotation**: Ένα annotation είναι μια μορφή μετα-δεδομένων που συνδέονται με ένα συγκεκριμένο τμήμα του περιεχομένου του εγγράφου. Η σύνδεση μεταξύ των annotations (σχολιασμών) και του περιεχομένου που αναφέρονται, γίνεται μέσω δύο δεικτών που αντιπροσωπεύουν τις περιοχές έναρξης και λήξης του καλυπτόμενου περιεχομένου. Ένας σχολιασμός πρέπει επίσης να έχει έναν τύπο (ή ένα όνομα), ο οποίος χρησιμοποιείται για τη δημιουργία κλάσεων από παρόμοια annotations, οι οποίες συνήθως συνδέονται μεταξύ τους με τη σημασιολογία τους. Γενικά, ένα annotation αποτελείται από:
    - *Start node*: περιοχή του εγγράφου που προσδιορίζεται από ένα αριθμό και δηλώνει την αρχή του annotation
    - *End node*: περιοχή του εγγράφου που προσδιορίζεται από ένα αριθμό και δηλώνει το τέλος του annotation
    - *Type*: ένα string που προσδιορίζει τον τύπο του annotation
    - *Features*: διάφορα χαρακτηριστικά, όπως αναφέρθηκαν προηγούμενα στο feature map
    - *ID*: ένας ακέραιος για να υποδηλώνει μοναδικά κάθε annotation

Τα annotations ορίζονται από τη διαπροσωπεία *gate.Annotation* και υλοποιούνται από την κλάση *gate.annotation.AnnotationImpl*. Τα annotations

υπάρχουν μόνο ως μέλη των annotation sets και δεν θα πρέπει να δημιουργούνται άμεσα από τη βοήθεια ενός κατασκευαστή. Η δημιουργία τους θα πρέπει πάντα να ανατεθεί στο annotation set που τα περιέχει.

- **Corpora**: Ένα corpus στο GATE είναι μια λίστα Java (δηλαδή μια εφαρμογή του java.util.List) των εγγράφων. Τα corpora του GATE ορίζονται από τη διαπροσωπεία *gate.Corpus* και οι ακόλουθες εφαρμογές είναι διαθέσιμες: *gate.corpora.CorpusImpl*, που χρησιμοποιούνται για παροδικά corpora και *gate.corpora.SerialCorpusImpl* , που χρησιμοποιούνται για μόνιμα corpora που είναι αποθηκευμένα σε μια σειριακή datastore (δηλαδή ενός καταλόγου σε ένα σύστημα αρχείων). Για παράδειγμα, παρακάτω φαίνεται η δημιουργία ενός corpus από όλα τα xml files ενός καταλόγου:

```
Corpus corpus = Factory.newCorpus("My XML Files");
File directory = ...;
ExtensionFileFilter filter = new ExtensionFileFilter("XML files", "xml");
URL url = directory.toURL();
corpus.populate(url, filter, null, false);
```

2.3.3 Processing resources

Τα Processing Resources (PRS) αποτελούν οντότητες οι οποίες είναι κατά κύριο λόγο αλγοριθμικές, όπως αναλυτές, γεννήτριες ή μοντέλα ngram[31] και δημιουργούνται με τη χρήση του *Factory* με παρόποιο τρόπο με τα LRs. Εκτός από τις παραμέτρους του χρόνου δημιουργίας (creation-time), έχουν επίσης μια σειρά από παραμέτρους χρόνου εκτέλεσης (run-time) που έχουν οριστεί από το σύστημα λίγο πριν από την εκτέλεσή τους.

2.3.4 Controllers

Οι controllers (ελεγκτές) χρησιμοποιούνται για τη δημιουργία εφαρμογών του GATE. Ένας ελεγκτής χειρίζεται ένα σύνολο από Processing Resources και μπορεί να τα εκτελέσει ακολουθώντας μια συγκεκριμένη στρατηγική. Γενικά, παρέχει μια σειρά από σειριακούς ελεγκτές (δηλ. ελεγκτές που τρέχουν τα PRs τους σε σειρά). Μερικά παραδείγματα φαίνονται παρακάτω:

- *gate.creole.SerialController*: ένας σειριακός ελεγκτής που παίρνει οποιοδήποτε είδος από PRs.
- *gate.creole.SerialAnalyserController*: : ένας σειριακός ελεγκτής που παίρνει μόνο Language Analysers ως PRs.
- *gate.creole.ConditionalSerialController*: ένας σειριακός ελεγκτής που δέχεται όλους τους τύπους PRs και ο οποίος επιτρέπει τη συμπερίληψη ή εξαίρεση κάποιον μελών των PRs από την αλυσίδα εκτέλεσης, σύμφωνα με ορισμένες προϋποθέσεις χρόνου εκτέλεσης.

---

[31] *Ngram model:* Στα πεδία της υπολογιστικής γλωσσολογίας και των πιθανοτήτων, ένα n-gram είναι μια συνεχόμενη ακολουθία από n αντικείμενα από μια δεδομένη ακολουθία κειμένου ή ομιλίας. Τα στοιχεία μπορεί να είναι επιφωνήματα, συλλαβές, γράμματα, λέξεις ή ζεύγη βάσεων ανάλογα με την εφαρμογή. Τα n-grams συνήθως συλλέγονται από ένα κείμενο ή από ένα speech corpus (μια βάση δεδομένων από ηχητικά αρχεία ομιλιών και κειμένων συστηματικής αναπαράστασης). [36]

2.3.5 Δημιουργία μιας ANNIE εφαρμογής και τρέξιμο αυτής σε κάποιο corpus

```
1. // αρχικά φορτώνουμε το ANNIE plugin
2. Gate.getCreoleRegister().registerDirectories(new File(
3. Gate.getPluginsHome(), "ANNIE").toURI().toURL());
4.
5. // δημιουργία του ελεγκτή για να τρέξουμε το ANNIE
6. SerialAnalyserController annieController =
7. (SerialAnalyserController) Factory.createResource(
8. "gate.creole.SerialAnalyserController",
9. Factory.newFeatureMap(),
10. Factory.newFeatureMap(), "ANNIE");
11.
12. // φορτώνουμε τα Processing Resources ακριβώς όπως είναι αποθηκευμένα
13. //στο ANNIEConstants
14. for(int i = 0; i < ANNIEConstants.PR_NAMES.length; i++) {
15. // χρήση των default παραμέτρων
16. FeatureMap params = Factory.newFeatureMap();
17. ProcessingResource pr = (ProcessingResource)
18. Factory.createResource(ANNIEConstants.PR_NAMES[i],
19. params);
20.
21. // προσθήκη του PR στον pipeline controller (ελεγκτή)
22. annieController.add(pr);
23. } // για κάθε ANNIE PR
24.
25. // Δημιουργία του Corpus στο οποίο θα τρέξουμε την ANNIE
26. Corpus corpus = ...;
27. annieController.setCorpus(corpus);
28.
29. // Τρέχουμε την ANNIE
30. annieController.execute();
```

2.3.6 Εμβάθυνση στο GATE Embedded

Μέχρι τώρα είδαμε γενικές πληροφορίες σύμφωνα με την ενσωμάτωση των εφαρμογών του GATE και χειρισμό αυτών σε κάποιο περιβάλλον Java. Ωστόσο, υπάρχει και επιπλέον ανάλυση για επίλυση πιο εξειδικευμένων θεμάτων.

Σε ορισμένες περιπτώσεις, είναι χρήσιμο να υπάρχει παραπάνω από ένα στρώμα μεταδεδομένων, που θα συνδέεται με τα ίδια annotations. Μια τέτοια περίπτωση είναι η μοντελοποίηση των σχέσεων μεταξύ των annotations. Ένα τυπικό

παράδειγμα των σχέσεων αυτών είναι το **co-reference**[32]. Έτσι, για τη δημιουργία των σχέσεων αυτών υπάρχει η διαπροσωπεία *gate.relations.Relation* με ένα σύνολο από μεθόδους για κατάλληλο χειρισμό των σχέσεων αυτών. Για παράδειγμα, η σύνδεση δύο annotations, token και sentence, μπορεί να γίνει με τη δημιουργία ενός relation με όνομα contained ως εξής:

```
relSet.addRelation("contained", new int[] {token.getId(), sentence.getId()});
```

Μερικές φορές, ιδιαίτερα σε μια πολυνηματική εφαρμογή, είναι χρήσιμο να μπορεί να δημιουργηθεί ένα ανεξάρτητο αντίγραφο ενός υπάρχοντος PR, ελεγκτή ή LR. Ο προφανής τρόπος για να γίνει αυτό είναι να γίνει κλήση του *createResource* πάλι, περνώντας το ίδιο όνομα κλάσης, τις παραμέτρους, τα χαρακτηριστικά και το όνομα, τα οποία σε πολλές περιπτώσεις είναι αρκετά. Ωστόσο, υπάρχουν ορισμένοι πόροι για τους οποίους αυτό μπορεί να είναι ανεπαρκές (π.χ. ελεγκτές, οι οποίοι πρέπει επίσης να επαναλάβουν τα PRs τους), ή μη ασφαλές (αν ένα PR χρησιμοποιεί προσωρινά αρχεία, για παράδειγμα), ή απλά αναποτελεσματικό. Για παράδειγμα, για ένα μεγάλο gazetteer αυτό θα συνεπαγόταν τη φόρτωση ενός δεύτερου αντιγράφου των καταλόγων στη μνήμη και τη μεταγλώττιση τους σε ένα δεύτερο όμοιο state machine[33], αλλά ένας πολύ πιο αποτελεσματικός τρόπος για να επιτευχθεί το ίδιο αποτέλεσμα θα ήταν να χρησιμοποιηθεί ένα *SharedDefaultGazetteer* , το οποίο μπορεί να επαναχρησιμοποιήσει το υπάρχον state machine. Το GATE Factory παρέχει μια μέθοδο για δημιουργία αντιγράφων η οποία λαμβάνει ένα υπάρχον resource και δημιουργεί και επιστρέφει ένα ανεξάρτητο αντίγραφο αυτού του resource (*Factory.duplicate*).

Επιπρόσθετα,υπάρχει και το @Sharable annotation (στο πακέτο *gate.creole.metadata*), το οποίο παρέχει έναν τρόπο στο resource για να σηματοδοτήσει ιδιότητες JavaBean, των οποίων οι τιμές θα πρέπει να μοιράζονται μεταξύ ενός resource και των διπλότυπών του. Τυπικά παραδείγματα των αντικειμένων που θα μπορούσαν να χαρακτηριστούν ως sharable περιλαμβάνουν μεγάλες ή ακριβές στη δημιουργία δομές δεδομένων που δημιουργούνται από ένα resource κατά το χρόνο εκκίνησης, ενώ στη συνέχεια χρησιμοποιούνται μόνο για ανάγνωση, thread-safe[34]

---

[32] *Co-reference:* Δύο annotations του ίδιου τύπου (π.χ. person) μπορεί να αναφέρονται στο ίδιο πραγματικό όνομα (π.χ. πρόσωπο), οπότε σε αυτή την περίπτωση λέμε ότι αυτά τα annotations είναι co-referring.

[33] *State machine:* Ένα finite-state machine (FSM) ή finite-state automaton (πληθυντικός: automata), ή απλά ένα state machine, είναι ένα μαθηματικό μοντέλο υπολογισμού που χρησιμοποιείται για το σχεδιασμό προγραμμάτων ηλεκτρονικών υπολογιστών αλλά και λογικών κυκλωμάτων. Έχει σχεδιαστεί ως μια αφηρημένη μηχανή που μπορεί να είναι σε μία από έναν πεπερασμένο αριθμό των καταστάσεων. Η μηχανή μπορεί να βρίσκεται σε μία μόνο κατάσταση κάθε στιγμή, οπότε η κατάσταση αυτή ονομάζεται current (τωρινή) state. Μπορεί, όμως, να αλλάξει από τη μια κατάταση στην άλλη, όταν εισαχθεί κάποιο κρίσιμο γεγονός ή κατάσταση, το οποίο ονομάζεται μετάβαση. Μια ιδιαίτερη FSM ορίζεται από μια λίστα των καταστάσεων της, και την κρίσιμη προϋπόθεση για κάθε μετάβαση. [37]

[34] *Thread-safety:* είναι μια έννοια προγραμματισμού ηλεκτρονικών υπολογιστών που εφαρμόζονται στο πλαίσιο των multi-threaded (πολυνηματικών) προγράμματων. Ένα κομμάτι του κώδικα είναι thread-safe, αν χειρίζεται μόνο κοινόχρηστες δομές δεδομένων κατά τρόπο που εγγυάται την ασφαλή εκτέλεση τους από πολλαπλά threads ταυτόχρονα. Υπάρχουν διάφορες στρατηγικές για την κατασκευή thread-safe δομών δεδομένων. [38]

cache μνήμες κάποιου είδους, ή καταστάσεις που χρησιμοποιούνται για τη δημιουργία καθολικά μοναδικών αναγνωριστικών (όπως AtomicInteger που αυξάνεται κάθε φορά που ένα νέο ID απαιτείται). Είναι σαφές ότι τυχόν αντικείμενα που είναι shared μεταξύ διαφορετικών περιπτώσεων πόρων πρέπει να προσεγγιστούν από όλες τις περιπτώσεις με τρόπο που να είναι thread-safe ή σωστά συγχρονισμένα.

Το GATE Embedded επιτρέπει την επίμονη αποθήκευση εφαρμογών σε μορφή που βασίζεται σε XML **serialisation**[35]. Αυτό είναι ιδιαίτερα χρήσιμο για την διαχείριση και διανομή των εφαρμογών. Ένας προγραμματιστής μπορεί να σώσει την κατάσταση μιας εφαρμογής όταν σταματήσει να εργάζεται και να συνεχίσει την ανάπτυξη της κάποια άλλη στιγμή. Όταν η εφαρμογή φθάσει σε τελικό επίπεδο, μπορεί να αναπτυχθεί στο χώρο του πελάτη, χρησιμοποιώντας την ίδια μέθοδο. Όταν μια εφαρμογή (δηλαδή ένας Controller) αποθηκεύεται, το GATE θα σώσει μόνο τις τιμές για τις παραμέτρους που χρησιμοποιούνται για τη δημιουργία των processing resources που περιέχονται στην εφαρμογή. Όταν αυτή ξαναφορτωθεί, όλα τα Prs θα δημιουργηθούν εκ νέου χρησιμοποιώντας τις αποθηκευμένες παραμέτρους. Παρακάτω φαίνεται ο τρόπος που θα σωθεί και θα φορτωθεί μια εφαρμογή:

```
//Μέρος που θα σωθεί η εφαρμογή
File file = ...;

//η εφαρμογή που θα σωθεί
Controller theApplication = ...;

// σώσιμο
gate.util.persistence.PersistenceManager. saveObjectToFile(theApplication, file);

//διαγραφή εφαρμογής
Factory.deleteResource(theApplication);
theApplication = null;

[...]
//φόρτωση εκ νέου της εφαρμογής
theApplication = gate.util.persistence.PersistenceManager.loadObjectFromFile(file);
```

Το GATE Embedded μπορεί να χρησιμοποιηθεί σε πολυνηματικές εφαρμογές, εφ' όσον τηρήσουν μερικούς περιορισμούς. Κατ 'αρχάς, θα πρέπει να γίνει η αρχικοποίηση του καλώντας Gate.init () ακριβώς μία φορά στην εφαρμογή, συνήθως στο στάδιο έναρξης εφαρμογής, πριν από την έναρξη οποιουδήποτε επεξεργαστικού νήματος. Δεύτερον, δεν πρέπει να γίνονται γενικά κλήσεις που επηρεάζουν την global κατάσταση του GATE (π.χ. φόρτωση ή εκφόρτωση plugins) σε περισσότερα από ένα νήματα κάθε στιγμή. Αντ'αυτού, συνήθως φορτώνονται όλα τα plugins που η εφαρμογή απαιτεί κατά το χρόνο εκκίνησης. Γενικά, είναι ασφαλές να δημιουργηθούν αναφορές (instances) των resources σε πολλαπλά νήματα

[35] *Serialization:* Στην επιστήμη των υπολογιστών, στο πλαίσιο της αποθήκευσης δεδομένων, το serialization είναι η διαδικασία της μετάφρασης των δομών δεδομένων ή των καταστάσεων των αντικειμένων σε μια μορφή που μπορεί να αποθηκευτεί (για παράδειγμα, σε ένα αρχείο ή στην προσωρινή μνήμη ή να μεταδοθεί σε ένα σύνδεσμο σύνδεσης με το δίκτυο) και να ανακατασκευαστεί αργότερα στο ίδιο ή σε άλλο περιβάλλον του υπολογιστή. [34]

ταυτόχρονα. Τρίτον, είναι σημαντικό να σημειωθεί ότι τα processing resources του GATE, τα language resources και οι ελεγκτές είναι από το σχεδιασμό τους όχι thread-safe, δηλαδή δεν είναι δυνατή η χρήση μιας αναφοράς ενός ελεγκτή / PR / LR σε πολλαπλά threads ταυτόχρονα. Μόνο για ένα καλογραμμένο resource θα μπορούσε να είναι δυνατή η χρήση πολλών διαφορετικών instances του ίδιου πόρου με τη μία, το καθένα σε διαφορετικό νήμα. Όλα τα τυποποιημένα PRs του ANNIE είναι ασφαλή όταν ανεξάρτητα instances χρησιμοποιούνται σε διαφορετικά νήματα ταυτόχρονα. Ένα τυπικό μοντέλο ανάπτυξης για μια πολυνηματική GATE εφαρμογή είναι:

- Ανάπτυξη της GATE εφαρμογής (processing pipeline) στο GATE Developer
- Αποθήκευση αυτής ως ένα .gapp (ή .state) αρχείο
- Στη φάση εκκίνησης της εφαρμογής, φόρτωση n αντιγράφων του pipeline χρησιμοποιώντας την εντολή *PersistenceManager.loadObjectFromFile ()* ή φόρτωση του pipeline μία φορά και στη συνέχεια δημιουργία αντιγράφων χρησιμοποιώντας το *Factory.duplicate*, όπως περιγράφηκε ανώτερα, ή παράδοση κάθε αντιτύπου σε ένα νήμα ή αποθήκευση αυτών
- Όταν θα πρέπει να επεξεργαστεί ένα κείμενο, παίρνει ένα αντίγραφο του pipeline απ'όπου αποθηκεύτηκε, και το επιστρέφει όταν έχει τελειώσει η επεξεργασία. [2]

Κεφάλαιο 3

JAPE Γραμματικές

3.1.Εισαγωγή στο Jape

Το Jape – Java Annotation Patterns Engine – είναι ένα βασικό στοιχείο που περιλαμβάνεται στην open-source πλατφόρμα του General Architecture for Text Engineering (GATE). Συγκεκριμένα, είναι μια μηχανή μεταγωγής (transduction) πεπερασμένων καταστάσεων που λειτουργεί στους σχολιασμούς και βασίζεται σε κανονικές εκφράσεις. Έτσι, θεωρείται χρήσιμη για διάφορους τομείς του language processing, όπως το pattern-matching[36], το semantic extraction, αλλά και σε πολλές άλλες λειτουργίες που αφορούν τα συντακτικά δέντρα (π.χ. αυτά που παράγονται από αναλυτές φυσικής γλώσσας). Το Jape είναι μια έκδοση του CPSL – Common Pattern Specification Language1, που επιτρέπει την αναγνώριση κανονικών εκφράσεων στα annotations των εγγράφων.

Μια γραμματική Jape αποτελείται από ένα σύνολο φάσεων, κάθε μία από τις οποίες αποτελείται από ένα σύνολο κανόνων μοτίβου (ή δράσης). Οι φάσεις τρέχουν διαδοχικά και αποτελούν ένα σύνολο από μετατροπείς πεπερασμένων καταστάσεων αναφερόμενοι στους σχολιασμούς. Το αριστερό μέρος (LHS) των κανόνων αποτελείται από μία περιγραφή μοτίβου των annotations. Το δεξί μέρος (RHS) αποτελείται από τις δηλώσεις χειρισμού των annotations. Τα annotations που ταιριάζουν στο LHS του κανόνα, μπορεί να αναφέρονται στο RHS μέσω των **ετικετών** που είναι συνδεμένα με στοιχεία του μοτίβου. Παρακάτω φαίνεται ένα παράδειγμα μιας γραμματικής jape που θα δείξει τη γενική μορφή σύνταξης τέτοιων κανόνων:

```
Phase: Jobtitle
Input: Lookup
Options: control = appelt debug = true

Rule: Jobtitle1
(
 {Lookup.majorType == jobtitle}
 (
  {Lookup.majorType == jobtitle}
 )?
)
:jobtitle
-->
 :jobtitle.JobTitle = {rule = "JobTitle1"}
```

Αρχικά, το LHS είναι το τμήμα που προηγείται του "-->" και το RHS είναι το μέρος μετά από αυτό. Το LHS καθορίζει ένα μοτίβο για να γίνει το ταίριασμα με το

---

[36] *Pattern-matching:* Στην επιστήμη των υπολογιστών, το pattern-matching είναι η πράξη ελέγχου μιας δεδομένης αλληλουχίας από tokens για την εύρεση κάποιων με συγκεκριμένο μοτίβο. Σε αντίθεση με την αναγνώριση προτύπων, το ταίριασμα πρέπει συνήθως να είναι ακριβές. Τα σχέδια έχουν συνήθως τη μορφή είτε αλληλουχιών ή δομών δέντρων. Χρήσεις του pattern-matching περιλαμβάνουν εξαγωγές των θέσεων (αν υπάρχουν) ενός μοτίβου μέσα σε μια συμβολική ακολουθία, την εξαγωγή κάποιου συστατικού του ταιριασμένου μοτίβου, και την υποκατάσταση του ταιριασμένου μοτίβου με κάποια άλλη συμβολική ακολουθία (δηλαδή, αναζήτηση και αντικατάσταση). [39]

annotated έγγραφο, ενώ το RHS καθορίζει τι πρέπει να γίνει με το κείμενο της αντιστοίχισης. Σε αυτό το παράδειγμα, έχουμε έναν κανόνα με τίτλο "Jobtitle1", ο οποίος θα ταιριάξει το κείμενο με τα εξής χαρακτηριστικά : να είναι annotated με το annotation "Lookup", έχοντας στο χαρακτηριστικό "majorType" την τιμή "jobtitle", ακολουθούμενο προαιρετικά από πρόσθετο κείμενο με ακριβώς τα ίδια χαρακτηριστικά (annotated ως "Lookup" και με χαρακτηριστικό "majorType" το "jobtitle"). Μόλις αυτός ο κανόνας ταιριάξει με μια ακολουθία κειμένου, σε ολόκληρη την αλληλουχία εκχωρείται μια ετικέτα από τον κανόνα, και εν προκειμένω, με την ετικέτα "jobtitle".

Στο RHS, αναφερόμαστε σε αυτό το κομμάτι κειμένου χρησιμοποιώντας την ετικέτα που δίνεται στο LHS, δηλαδή το "jobtitle".Έτσι, λέμε ότι στο κείμενο με αυτή την ετικέτα πρέπει να δοθεί το annotation του τύπου "JobTitle" και ως χαρακτηριστικό το "rule" με τιμή "JobTitle1".

Επίσης, παρατηρούμε ότι ξεκινήσαμε τη γραμματική Jape δίνοντάς της ένα όνομα φάσης. Γενικά, μια γραμματική Jape αποτελείται από ένα σύνολο φάσεων, καθέ μια από τις οποίες αποτελείται από ένα σύνολο κανόνων μοτίβου. Αυτές οι φάσεις πρέπει να επισημαίνονται μοναδικά με κάποιο όνομα (εδώ "Jobtitle"), σημειώνοντας ότι το όνομα αυτό μπορεί να είναι διαφορετικό από το όνομα του αρχείου της jape γραμματικής. Ωστόσο, το όνομα της φάσης αποτελεί μέρος του ονόματος της κλάσης Java για τις λειτουργίες του RHS που μεταγλωττίστηκαν. Εξαιτίας αυτού, θα πρέπει να περιέχει μόνο αλφαριθμητικούς χαρακτήρες και χαρακτήρες κάτω παύλας, και δεν μπορεί να ξεκινάει με αριθμό.

Πρέπει, επίσης, να ορίζεται μια λίστα από annotations ως είσοδος κατά την έναρξη κάθε γραμματικής, δηλώνοντας έτσι ότι αυτά είναι τα annotations με τα οποία θα πρέπει να ταιριάξει ο κανόνας. Εάν δεν οριστούν τέτοια, η προεπιλογή θα είναι τα Token, SpaceToken και Lookup annotations, καθώς από προεπιλογή, μόνο αυτοί οι σχολιασμοί θα εξεταστούν κατά την προσπάθεια ενός ταιριάσματος. Άρα, κάθε είδος σχολιασμού που πρόκειται να ταιριάζει με αυτή τη φάση γραμματικής πρέπει να περιλαμβάνεται στο σύνολο εισόδου (Input set). Συνεπώς, κάθε είδος σχολιασμού που δεν έχει οριστεί, θα αγνοηθεί κατά το ταίριασμα των μοτίβων. Εν προκειμένω, χρησιμοποιούμε "Input: Lookup", διότι το μόνο είδος σχολιασμού που χρησιμοποιούμε για το LHS είναι το Lookup annotation.

Υπάρχουν, ακόμα, και διάφορες επιπλέον επιλογές (options) που μπορούν να ρυθμιστούν και να δηλώσουν τα χαρακτηριστικά και τις μεθόδους των κανόνων και των γραμματικών. Ένα ευρύ φάσμα λειτουργικότητας μπορεί να χρησιμοποιηθεί με το Jape, καθιστώντας το ένα πολύ ισχυρό σύστημα. Παρακάτω φαίνονται αναλυτικότερα στοιχεία για τα δύο μέλη των γραμματικών, της λειτουργικότητας τους, τις προτεραιότητες μεταξύ των κανόνων, αλλά και τρόπους εισαγωγής Java κώδικα στο RHS για μεγαλύτερη αποτελεσματικότητα αυτών. [2]

3.2. Το αριστερό μέρος (LHS)

Το LHS (left-hand side) μιας γραμματικής Jape έχει ως στόχο να ταιριάξει κάποιο κομμάτι κειμένου ώστε να γίνει annotated, αποφεύγοντας ανεπιθύμητα ταιριάσματα. Υπάρχουν διάφορα εργαλεία που είναι διαθέσιμα γι' αυτό το σκοπό.

Ο απλούστερος κανόνας είναι να ταιριάξει κάθε μοναδικό annotation ενός συγκεκριμένου τύπου. Μπορούν να ταιριάξουν μόνο τύποι από annotations που έχουν καθοριστεί στη γραμμή "Input" στο επάνω μέρος του αρχείου. Για παράδειγμα, το ακόλουθο θα ταιριάξει κάθε Lookup annotation: *{Lookup}*

Μπορεί, ακόμα, να γίνει ο καθορίσμος των χαρακτηριστικών (και των τιμών) σε ένα annotation που θα ταιριάξουν. Παρακάτω φαίνονται διάφοροι τελεστές που επιτελούν αυτή τη λειτουργία:

- ισότητα και ανισότητα*: {Token.kind == "number"}, {Token.length != 4} –*.
- Τελεστές σύγκρισης: *{Token.string > "aardvark"}, {Token.length < 10}*. Υποστηρίζονται ακόμα και οι τελεστές <=, >= .
- έλεγχος των annotations στο πλαίσιο άλλων annotations: *{X contains Y}, {X notContains Y}, {X within Y}* και *{X notWithin Y}*
- κανονικές εκφράσεις: *{Token.string =~ "[Dd]ogs"}, {Token.string !~ "(?i)hello"}*. Ακόμα υποστηρίζονται και οι τελεστές *==~* και *!=~* για ολοκήρωτικό ταίριασμα των strings.

Ο παρακάτω κανόνας λειτουργεί όταν το χαρακτηριστικό "category" του annotation "Token" είναι ίσο με "NNP" :

```
Rule: Unknown
Priority: 50
(
 {Token.category == NNP}
)
:unknown
-->
 :unknown.Unknown = {kind = "PN", rule = Unknown}
```

Εκτός από τα χαρακτηριστικά αναφοράς των annotations, το Jape επιτρέπει την πρόσβαση και σε άλλες "μετα-ιδιότητες" των annotations. Αυτό γίνεται με τη χρήση του συμβόλου "@" μετά το όνομα του τύπου σχολιασμού. Οι τρεις μετα-ιδιότητες που είναι δημιουργημένες είναι:

- Length: επιστρέφει το μήκος που εκτείνεται το annotation
- String: επιστρέφει το string που εκτείνεται το annotation στο έγγραφο.
- cleanString: Όπως το string, αλλά με αφαίρεση του επιπλέον λευκού περιθωρίου.

```
{X@length > 5}:label-->:label.New = {}
```

3.2.1. Σύνθετα μοτίβα για annotations

Παρακάτω φαίνονται πολλά παραδείγματα και χρήσεις διαφόρων τελεστών για δημιουργία πιο απαιτητικών κανόνων:

- **Ακολουθία**

```
Rule: InLocation
(
 {Token.category == "IN"}
 {Location}
):inLoc
```

Στο παραπάνω παράδειγμα φαίνεται ένας κανόνας με όνομα InLocation, ο οποίος ταιριάζει ένα annotation τύπου "Token" με το χατακτηριστικό "category" να έχει τιμή "IN", ακολουθούμενο από ένα annotation τύπου "Location". Σημειώστε ότι το "ακολουθούμενο από" εξαρτάται από τα είδη σχολιασμού που καθορίζονται στη γραμμή εισόδου (Input line) - το παραπάνω μοτίβο ταιριάζει ένα annotation τύπου Token και ένα annotation τύπου Location υπό την προϋπόθεση ότι δεν υπάρχουν παρεμβατικά annotations των

τύπων που αναφέρονται στη γραμμή εισόδου. Το Token και το Location δεν είναι απαραίτητο να είναι συνεχόμενα (θα μπορούσαν πιθανότατα να διαχωρίζονται από ένα παρεμβαλλόμενο διάστημα). Ειδικότερα, το μοτίβο δεν θα ταιριάζει αν το "SpaceToken" annotation είχε καθοριστεί στη γραμμή εισόδου.

- **Εναλλακτικές**

```
Rule: InOrAdjective
(
 {Token.category == "IN"} | {Token.category == "JJ"}
):inLoc
```

Στο παραπάνω παράδειγμα φαίνεται η χρήση του τελεστή "|", ο οποίος χρησιμοποιείται για να υποδηλώσει εναλλακτικές λύσεις. Αυτό σημαίνει ότι ο κανόνας θα ταιριάξει ένα Token με χαρακτηριστικό category "IN" ή "JJ".

- **Ομαδοποίηση**

```
Rule: InLocation
(
 ({Token.category == "IN"} | {Token.category == "JJ"})
 {Location}
):inLoc
```

Η παρένθεση χρησιμοποιείται για την ομαδοποίηση των μοτίβων. Έτσι, στο παράδειγμα από πάνω ο κανόνας ταιριάζει ένα Token που θα έχει μία από τις δύο τιμές του χαρακτηριστικού category ("IN" ή "JJ") ακολουθούμενο από ένα Location.

```
Rule: InLocation
(
 {Token.category == "IN"} |
 ( {Token.category == "JJ"}
   {Location} )
):inLoc
```

Εδώ φαίνεται η διαφορά χρήσης της παρένθεσης, αφού τώρα ο κανόνας θα αντιστοιχίσει ή ένα "In" Token ή μια ακολουθία από "JJ" Token και Location.

- **Επανάληψη**

Το Jape παρέχει, επίσης, τελεστές επανάληψης για να επιτρέψει σε ένα μοτίβο σε παρενθέσεις να είναι προαιρετικό (?), ή να ταιριάζει με κανένα ή περισσότερα (*), ένα ή περισσότερα (+) ή κάποιο καθορισμένο αριθμό φορών. Στο παρακάτω παράδειγμα, μπορείτε να δείτε τους τελεστές "|" και "?" να χρησιμοποιούνται:

```
Rule: LocOrganization
Priority: 50

(
 ({Lookup.majorType == location} |
  {Lookup.majorType == country_adj})
{Lookup.majorType == organization}
({Lookup.majorType == organization})?
)
:orgName -->
  :orgName.TempOrganization = {kind = "orgName", rule=LocOrganization}
```

- **Εύρος αναφορών**
  Τα εύρη επανάληψεων καθορίζονται χρησιμοποιώντας αγκύλες.

```
({Token})[1,3]
({Token.kind == number})[3]
```

  Το πρώτο ταιριάζει ένα ή τρία Tokens στη σειρά, ενώ το δεύτερο ακριβώς τρία Token στη σειρά.

- **Ταίριασμα κειμένου**
  Το Jape λειτουργεί πάνω από τα annotations και έτσι δεν μπορεί να ταιριάξει άμεσα με strings του κειμένου στο έγγραφο. Για να ταιριάξει ένα string θα πρέπει να ταιριάξει με το annotation που καλύπτει αυτό το string, συνήθως ένα "Token". Το GATE Tokeniser προσθέτει ένα "string" χαρακτηριστικό σε όλα τα Token annotations περιέχοντας τη συμβολοσειρά που το Token καλύπτει, ώστε να μπορεί να χρησιμοποιηθεί αυτό για να ταιριάξει το κείμενο του εγγράφου.

```
Phase: UrlPre
Input:  Token SpaceToken
Options: control = appelt

Rule: Urlpre

( (({Token.string == "http"} |
  {Token.string == "ftp"})
 {Token.string == ":"}
 {Token.string == "/"}
       {Token.string == "/"}
      ) |
({Token.string == "www"}
       {Token.string == "."}
      )
):urlpre
-->
:urlpre.UrlPre = {rule = "UrlPre"}
```

  Από τη στιγμή που γίνεται ταίριασμα των annotations και όχι του κειμένου, απαιτείται προσοχή στο γεγονός ότι οι συμβολοσειρές αυτές είναι μονά tokens. Στο παραπάνω παράδειγμα, το {Token.string == ":/ /"} δεν θα ταιριάξει ποτέ (αν υποθέσουμε ότι έχουμε τον προεπιλεγμένο ANNIE Tokeniser), αφού και οι τρεις χαρακτήρες αντιμετωπίζονται ως ξεχωριστά tokens.

- **Templates**
  Στις περιπτώσεις όπου μια γραμματική περιέχει πολλά παρόμοια ή πανομοιότυπα strings ή άλλες κυριολεκτικές τιμές, το Jape υποστηρίζει την ιδέα των templates. Ένα template είναι μια επώνυμη τιμή που δηλώνεται στο αρχείο γραμματικής, παρόμοια με μια μεταβλητή στη Java ή άλλες γλώσσες προγραμματισμού, η οποία μπορεί να αναφέρεται σε οποιοδήποτε σημείο θα μπορούσε να χρησιμοποιηθεί μία κανονική συμβολοσειρά, μία boolean ή μία αριθμητική τιμή, στο αριστερό ή το δεξί μέρος του κανόνα. Στην απλούστερη περίπτωση τα templates μπορούν να είναι σταθερές:

```
Template: source = "Interesting entity finder"
Template: threshold = 0.6
```

Τα templates μπορούν να χρησιμοποιηθούν σε κανόνες, παρέχοντας τα ονόματά τους σε αγκύλες:

```
Rule: InterestingLocation
(
 {Location.score >= [threshold]}
):loc
-->
:loc.Entity = { type = Location, source = [source] }
```

Ο parser της γραμματικής Jape αντικαθιστά τις τιμές των templates για τις αναφορές τους, όταν η γραμματική σαρώνεται. Έτσι, ο παραπάνω κανόνας είναι ισοδύναμος με τον εξής:

```
Rule: InterestingLocation
(
 {Location.score >= 0.6}
):loc
-->
:loc.Entity = { type = Location,
 source = "Interesting entity finder" }
```

Το πλεονέκτημα της χρήσης των templates είναι ότι αν υπάρχουν πολλοί κανόνες της γραμματικής στους οποίους γίνεται μεγάλη αναφορά του template "threshold", τότε είναι δυνατό να αλλάξει το threshold για όλους τους κανόνες, απλά αλλάζοντας τον ορισμό του template. Ακόμα, τα templates των οποίων η τιμή είναι μια συμβολοσειρά μπορεί να περιέχει και παραμέτρους, που ορίζονται με το συμβολισμό *${name}*. Σε μια Jape γραμματική πολλαπλών φάσεων, τα templates που ορίζονται στις προηγούμενες φάσεις μπορούν να αναφέρονται σε μεταγενέστερα στάδια. Αυτό καθιστά δυνατή τη δήλωση των σταθερών σε ένα μέρος και την αναφορά τους με χρήση μιας πολύπλοκης γραμματικής.

- **Πολλαπλά μοτίβα**
  Είναι επίσης δυνατόν να έχουμε περισσότερα από ένα μοτίβα και αντίστοιχη δράση, όπως φαίνεται στον παρακάτω κανόνα. Στο LHS κάθε μοτίβο περικλείεται από μια σειρά από παρενθέσεις και έχει μια μοναδική ετικέτα, ενώ στο RHS, κάθε ετικέτα συνδέεται με μια ενέργεια. Στο παράδειγμα που ακολουθεί, το Lookup annotation με majorType την τιμή "jobtitle" είναι χαρακτηρισμένο με την ετικέτα "jobtitle" και του δίνεται το νέο annotation "JobTitle", ενώ το annotation TempPerson είναι χαρακτηρισμένα με την ετικέτα "person" και του δίνεται το νέο annotation "Person".

```
Rule: PersonJobTitle
Priority: 20

(
 {Lookup.majorType == jobtitle}
):jobtitle
(
 {TempPerson}
):person
```

```
-->
   :jobtitle.JobTitle = {rule = "PersonJobTitle"},
   :person.Person = {kind = "personName", rule = "PersonJobTitle"}
```

Ομοίως, τα μοτίβα που επισημαίνονται μπορούν να είναι φωλιασμένα, όπως στο παρακάτω παράδειγμα, όπου το συνολικό μοτίβο είναι annotated ως Person, ενώ μέσα στο μοτίβο, το jobtitle είναι annotated ως JobTitle.

```
Rule: PersonJobTitle2
Priority: 20

(
(
 {Lookup.majorType == jobtitle}
):jobtitle
 {TempPerson}
):person
-->
   :jobtitle.JobTitle = {rule = "PersonJobTitle"},
   :person.Person = {kind = "personName", rule = "PersonJobTitle"}
```

- **Μακροεντολές**
  Οι μακροεντολές επιτρέπουν τη δημιουργία ενός ορισμού που μπορεί στη συνέχεια να χρησιμοποιηθεί πολλές φορές στους κανόνες Jape. Στην παρακάτω Jape γραμματική, έχουμε ένα σύνολο μακροεντολών που χρησιμοποιούνται. Η μακροεντολή "AMOUNT_NUMBER" κάνει χρήση των μακροεντολών "MILLION_BILLION" και "NUMBER_WORD", ενώ ο κανόνας "MoneyCurrencyUnit" κάνει χρήση του "AMOUNT_NUMBER":

```
Phase: Number
Input: Token Lookup
Options: control = appelt

Macro: MILLION_BILLION
({Token.string == "m"}|
{Token.string == "million"}|
{Token.string == "b"}|
{Token.string == "billion"}|
{Token.string == "bn"}|
{Token.string == "k"}|
{Token.string == "K"} )

Macro: NUMBER_WORDS
(
 (({Lookup.majorType == number}
  ({Token.string == "-"})?
 )*
  {Lookup.majorType == number}
  {Token.string == "and"}
 )*
 ({Lookup.majorType == number}
  ({Token.string == "-"})?
```

```
 )*
  {Lookup.majorType == number}
)

Macro: AMOUNT_NUMBER
(({Token.kind == number}
 (({Token.string == ","}|
  {Token.string == "."}
  )

  {Token.kind == number}
 )*
 |
 (NUMBER_WORDS)
 )
 (MILLION_BILLION)?
)

Rule: MoneyCurrencyUnit
 (
    (AMOUNT_NUMBER)
    ({Lookup.majorType == currency_unit})
 )
:number -->
 :number.Money = {kind = "number", rule = "MoneyCurrencyUnit"}
```

- **Πολυπεριοριστικές δηλώσεις**
  Είναι εξίσου αποδεκτό να υπάρχουν πολλαπλοί περιορισμοί σε μια δήλωση. Σε αυτό το παράδειγμα, το "majorType" του "Lookup" πρέπει να έχει την τιμή "name" και το "minorType" την τιμή "surname".

```
Rule: Surname
(
 {Lookup.majorType == "name",
  Lookup.minorType == "surname"}
):surname
-->
 :surname.Surname = {}
```

Σε αυτές τις περιπτώσεις, οι πολλαπλοί περιορισμοί για το ίδιο είδος σχολιασμού, πρέπει να ικανοποιηθούν όλοι με το ίδιο annotation, ώστε να ταιριάξει το μοτίβο. Οι περιορισμοί μπορούν να αναφέρονται σε διαφορετικά annotations, και για να ταιριάξει το μοτίβο στο σύνολό του, οι περιορισμοί θα πρέπει να ικανοποιούνται από τα annotations που ξεκινούν στην ίδια θέση στο έγγραφο. Σε αυτό το παράδειγμα, εκτός από τους περιορισμούς σχετικά με τα "majorType" και "minorType" του "Lookup", έχουμε επίσης έναν περιορισμό για το "string" των "Token":

```
Rule: SurnameStartingWithDe
(
 {Token.string == "de",
  Lookup.majorType == "name",
 Lookup.minorType == "surname"}
):de
```

```
-->
 :de.Surname = {prefix = "de"}
```

Ο κανόνας αυτός θα ταιριάζει όπου ένα Token με string “de” και Lookup με majorType “name” και minorType “surname” ξεκινάει στην ίδια θέση στο κείμενο. Τόσο το Lookup όσο και το Token annotation θα συμπεριληφθούν στο *:de*, οπότε το Surname που δημιουργείται θα καλύπτει το μεγαλύτερο μήκος από τα δύο. Όπως και πριν, οι περιορισμοί για το ίδιο είδος annotation πρέπει να ικανοποιούνται από ένα μόνο annotation, έτσι σε αυτό το παράδειγμα πρέπει να υπάρχει ένα ενιαίο Lookup που θα ταιριάζουν τόσο οι major και minor τύποι - ο κανόνας δεν θα ταιριάζει αν υπήρχαν δύο διαφορετικά Lookups στην ίδια θέση , όπου ένας από αυτούς θα ικανοποιούσε κάθε περιορισμό.

- **Χρήση συμφραζόμενων**
  Τα συμφραζόμενα μπορούν να αντιμετωπιστούν με τους κανόνες γραμματικής με τον ακόλουθο τρόπο. Το μοτίβο που πρέπει να είναι annotated περικλείεται πάντα από ένα σύνολο παρενθέσεων. Εάν το προηγούμενο πλαίσιο πρέπει να περιλαμβάνεται στον κανόνα, αυτό τοποθετείται πριν από το σύνολο των παρενθέσεων. Το πλαίσιο αυτό περιγράφεται με τον ίδιο ακριβώς τρόπο όπως και το μοτίβο, ώστε να συνδυαστούν. Εάν το πλαίσιο πρέπει να συμπεριληφθεί μετά το μοτίβο, τοποθετείται μετά την ετικέτα που δόθηκε στο annotation. Το πλαίσιο χρησιμοποιείται όταν ένα μοτίβο πρέπει να αναγνωρίζεται μόνο εφόσον συμβεί μια συγκεκριμένη κατάσταση, αλλά το ίδιο το πλαίσιο δεν αποτελεί μέρος του μοτίβου που πρόκειται να γίνει annotated.

```
Rule: YearContext1

({Token.string == "in"}|
 {Token.string == "by"}
)
(YEAR)
:date -->
 :date.Timex = {kind = "date", rule = "YearContext1"}
```

Στο παραπάνω παράδειγμα, φαίνεται ότι ο κανόνας (θεωρώντας κάποια μακροεντολή για το “year”) δείχνει ότι ένας χρόνος θα αναγνωριζόνταν μόνο αν εμφανιζόταν μετά από τις λέξεις “in” ή “by”. Ομοίως, ο ακόλουθος κανόνας (υποθέτοντας ότι μια κατάλληλη μακροεντολή για το “e-mail”) θα σήμαινε ότι μια διεύθυνση ηλεκτρονικού ταχυδρομείου, θα πρέπει να αναγνωρίζεται μόνο εφόσον θα εμφανιζόταν μέσα σε γωνιακές αγκύλες (οι οποίες δε θα αποτελούν μέρος της οντότητας):

```
Rule: Emailaddress1
({Token.string == ‘<’})
(
 (EMAIL)
)
:email
({Token.string == ‘>’})
```

```
-->
 :email.Address= {kind = "email", rule = "Emailaddress1"}
```

- **Άρνηση**

  Το Jape υποστηρίζει επίσης "αρνητικούς" περιορισμούς που ορίζουν την απουσία των annotations. Ένας αρνητικός περιορισμός σηματοδοτείται στην γραμματική με το χαρακτήρα "!".

```
Rule: PossibleName
(
 {Token.orth == "upperInitial", !Lookup}
):name
-->
 :name.PossibleName = {}
```

Ο κανόνας αυτός θα ταιριάζει με κεφαλαίο αρχικό γράμμα του Token, μόνο όταν δεν υπάρχει Lookup σχολιασμός που ξεκινάει από την ίδια θέση. Ο γενικός κανόνας είναι ότι ένας αρνητικός περιρισμός ταιριάζει σε οποιαδήποτε θέση δεν ταιριάζει ο αντίστοιχος θετικός. Οι αρνητικοί περιορισμοί δε δημιουργούν annotations - στο παραπάνω παράδειγμα, το :name θα περιέχει μόνο το annotation Token. Η εξαίρεση σε αυτό είναι η περίπτωση που ένας αρνητικός περιορισμός χρησιμοποιείται μόνος του, χωρίς να συνδυάζεται με θετικούς περιορισμούς. Τότε, δεσμεύει όλους τους σχολιασμούς που δεν ταιριάζουν με τον περιορισμό. Έτσι, το {!Lookup} θα δεσμεύει όλους τους σχολιασμούς που ξεκινούν από αυτήν την τοποθεσία, εκτός από τα Lookups. Σε γενικές γραμμές οι αρνητικοί περιορισμοί θα πρέπει να χρησιμοποιούνται μόνο σε συνδυασμό με θετικούς.

```
Rule: SurnameNotStartingWithDe
(
 {Surname, !Token.string ==~ "[Dd]e"}
):name
-->
 :name.NotDe = {}
```

Στο παραπάνω παράδειγμα, ο κανόνας θα ταιριάζει με κάθε Surname annotation που δεν ξεκινάει από την ίδια θέση με ένα Token, που θα έχει string "de" ή "De". Σημειώστε ότι αυτό είναι ελαφρώς διαφορετικό από το *{Surname, Token.string !=~ "[Dd]e"}*, καθώς η δεύτερη μορφή απαιτεί την ύπαρξη ενός Token σχολιασμού, ενώ η πρώτη μορφή (! Token ...) θα ταιριάζει κι αν δεν υπάρχει Token σχολιασμός σε αυτή τη θέση. Όπως και με τους θετικούς περιορισμούς, οι πολλαπλοί αρνητικοί περιορισμοί στο ίδιο είδος σχολιασμού πρέπει να ταιριάζουν όλοι με το ίδιο σχόλιο, ώστε να μπλοκαριστεί το συνολικό μοτίβο.

```
{Name, !Lookup.majorType == "person", !Lookup.minorType == "female"}
```

Στο παραπάνω παράδειγμα, ο κανόνας θα ταιριάξει με ένα "Name" σχολιασμό, αλλά μόνο αν δεν ξεκινάει στην ίδια θέση ως Lookup με majorType "person" και minorType "female". Ένα Lookup με majorType "person" και minorType "male" δεν θα εμπόδιζε το μοτίβο από το ταίριασμα. Ωστόσο, αρνητικοί περιορισμοί σε διαφορετικά είδη σχολιασμού είναι

ανεξάρτητοι. Έτσι, στο παρακάτω παράδειγμα, ο κανόνας θα ταιριάξει με ένα annotation Person, αλλά μόνο αν δεν υπάρχουν annotations ούτε Organization ούτε Location που ξεκινούν από το ίδιο μέρος.

```
{Person, !Organization, !Location}
```

Παρά το γεγονός ότι το Jape παρέχει στο χειριστή ένα τελεστή για την απουσία ενός ενιαίου είδους σχολιασμού, δεν υπάρχει υποστήριξη για ένα γενικό αρνητικό τελεστή, ο οποίος να προλαμβάνει έναν κανόνα από την ενεργοποίηση, αν εμφανιστεί μία συγκεκριμένη ακολουθία σχολιασμών. Μια λύση σε αυτό είναι η δημιουργία ενός "αρνητικού κανόνα", που θα έχει μεγαλύτερη προτεραιότητα από τον αντιστοίχο "θετικό κανόνα". Το στυλ του ταιριάσματος θα πρέπει να δηλωθεί ως Appelt σε αυτή την περίπτωση. Για να δημιουργηθεί ένας αρνητικός κανόνας, απλώς δηλώνεται στο LHS του κανόνα το μοτίβο που ΔΕΝ πρέπει να συνδυάζεται, ενώ στο RHS δε γίνεται τίποτα. Με αυτόν τον τρόπο, ο θετικός κανόνας δεν μπορεί να τροφοδοτηθεί εάν το αρνητικό μοτίβο ταιριάξει και αντιστρόφως, ο οποίος έχει το ίδιο τελικό αποτέλεσμα με τη χρήση ενός αρνητικού τελεστή. Μία χρήσιμη παραλλαγή για τους προγραμματιστές είναι να δημιουργηθεί ένα dummy annotation σχετικά με το RHS του αρνητικού κανόνα, παρά να μη γίνει τίποτα, και να δώσει στο dummy annotation ένα χαρακτηριστικό κανόνα. Με αυτόν τον τρόπο, είναι προφανές ότι ο αρνητικός κανόνας εκτελείται. Εναλλακτικά, μπορεί να γίνει χρήση κώδικα Java για το RHS εκτυπώνοντας ένα μήνυμα, όταν εκτελείται ο κανόνας.

```
Rule: NotPersonReverse
Priority: 20
// we don't want to match 'Jones, I'
(
 {Token.category == NNP}
 {Token.string == ","}
 {Token.category == PRP}
)
:foo
-->
{}

Rule:   PersonReverse
Priority: 5
// we want to match 'Jones, F.W.'

(
 {Token.category == NNP}
 {Token.string == ","}
 (INITIALS)?
)
:person -->
```

Στο παραπάνω παράδειγμα γίνεται ταίριασμα αρνητικών και θετικών κανόνων. Συγκεκριμένα, ο κανόνας αντιστοιχεί με ένα επώνυμο ακολουθούμενο από ένα κόμμα και ένα σετ από αρχικά (initials). Αλλά θέλουμε να διευκρινιστεί ότι τα αρχικά δεν θα πρέπει να έχουν PRP το POS category (προσωπική αντωνυμία). Γι 'αυτό, γίνεται ο ορισμός ενός αρνητικού

κανόνα που θα εκτελεστεί αν υπάρχει το PRP category, εμποδίζοντας έτσι το θετικό κανόνα από την εκτλελεση του.

3.2.2. Τελεστές

Παρακάτω φαίνονται πολλοί τελεστές που χρησιμοποιούνται συχνα σε Jape γραμματικές:

- Ισότητας :
  - Βασικοί τελεστλες ισότητας είναι "==" και "!="
  - Για strings: String.equals()
  - Για ακέραιους: Long.equals()
  - Για αριθμούς κινητής υποδιαστολής: Double.equals()
  - Για λογικές πράξεις: Boolean.equals()
- Σύγκρισης :
  - Βασικοί τελεστές σύγκρισης είναι οι '<', '<=', '>=' και'>'
  - Για strings: String. compareTo ()
  - Για ακέραιους: Long.compareTo ()
  - Για αριθμούς κινητής υποδιαστολής: Double.compareTo ()
- Κανονικών εκφράσεων :
  Οι τελεστές κανονικών εκφράσεων είναι "= ~ ", " == ~ ", "! ~" και "! = ~". Αυτοί οι τελεστές ταιριάζουν με τις συνήθεις εκφράσεις. Για παράδειγμα, το *{Token.string = ~ "[Dd] ogs"}* ταιριάζει με ένα Token σχόλιο του οποίου το χαρακτηριστικό string περιέχει μια συμβολοσειρά που ταιριάζει με την κανονική έκφραση "[Dd] ogs". Αν γινόταν χρήση του τελεστή "! ~ ", θα ταίριαζε αν η τιμή του χαρακτηριστικού δεν περιείχε ένα string που θα ταίριαζε με την κανονική έκφραση.
- Συμφραζόμενων :
  Οι τελεστές συμφραζόμενων είναι οι "contains" και "within" και τα συμπληρώματά τους "notContains" και "notWithin". Αυτοί οι τελεστές ταιριάζουν τα annotations στο πλαίσιο άλλων annotations.
  - Contains: Γράφεται ως εξής {X contains Y} και επιστρέφει true αν το annotation τύπου X περιέχει τελείως ένα annotation τύπου Y. Αντίθετα {X notContains Y}ταιριάζει αν ένας σχολιασμός του τύπου X δεν περιέχει ένα τύπου Y.
  - Within: Γράφεται ως εξής {X within Y} και επιστρέφει true αν ο σχολιασμός του τύπου X καλύπτεται πλήρως από το σχολιασμό του τύπου Y. Αντίθετα {X notWithin Y} ταιριάζει αν ένας σχολιασμός του τύπου X δεν καλύπτεται από σχολιασμό του τύπου Y.
- Custom :
  Είναι δυνατό να προστεθούν επιπλέον προσαρμοσμένοι τελεστές χωρίς να τροποποιηθεί η γλώσσα Jape. Υπάρχουν παράμετροι χρόνου αρχικοποίησης του Transducer, ώστε επιπλέον τελεστές μπορούν να αναφερθούν κατά το χρόνο εκτέλεσης. Για την πρόσθεση ενός προσαρμοσμένου τελεστή, πρέπει να γραφτεί μια κλάση που υλοποιεί το *gate.jape.constraint.ConstraintPredicate*, να γίνει η κλάση διαθέσιμη στο GATE (είτε βάζοντας την κλάση σε ένα αρχείο JAR στον κατάλογο lib ή βάζοντας την κλάση σε ένα plugin και τη φόρτωση του plugin), και στη συνέχεια βάζοντας σε λίστα αυτό το όνομα της κλάσης στην ιδιοκτησία "annotationAccessors" του Transducer.

3.3. Το δεξί μέρος (RHS)

Το RHS του κανόνα περιέχει πληροφορίες σχετικά με τον σχολιασμό που θα δημιουργηθεί. Οι πληροφορίες σχετικά με την έκταση του κειμένου που θα σχολιαστεί, μεταφέρονται από το LHS του κανόνα κάνοντας χρήση της ετικέτας που περιγράφτηκε, και γίνεται annotated με τον τύπο της οντότητας που ακολουθεί. Τέλος, τα χαρακτηριστικά και τις αντίστοιχες τιμές τους, προστίθενται στο σχολιασμό. Εναλλακτικά, το RHS του κανόνα μπορεί να περιέχει κώδικα της Java για τη δημιουργία και το χειρισμό των annotations.

```
Rule: GazLocation
(
{Lookup.majorType == location}
)
:location -->
 :location.Enamex = {kind="location", rule=GazLocation}
```

Στο παραπάνω παράδειγμα το μοτίβο που περιγράφεται θα προσδώσει ένα annotation τύπου "Enamex" με χαρακτηριστικά "kind" και "rule" με τιμές "location" και "GazLocation" αντίστοιχα, σε κάποιο annotation Lookup με majorType την τιμή location. Παρακάτω επισημαίνονται γενικές περιπτώσεις του RHS:

- **Αντιγραφή τιμών χαρακτηριστικών από το LHS στο RHS**
  Είναι δυνατή η αντιγραφή τιμών κάποιων χαρακτηριστικών από το αριστερό μέρος στο δεξί, γεγονός που θα φανεί με την ανάλυση της συμπεριφοράς του *newFeat = :bind.Type.oldFeat:*
  - Βρείτε όλα τα annotations του τύπου Type του LHS που συνδέονται με την ετικέτα bind
  - Βρείτε ένα από αυτά που έχει μια μη μηδενική τιμή για το χαρακτηριστικό oldFeat (αν υπάρχουν περισσότερες από μία, το ποιά θα επιλεγεί είναι θέμα του JAPE)
  - Αν υπάρχει μια τέτοια τιμή, ρυθμίστε το χαρακτηριστικό newFeat του νέου σχολιασμού σε αυτήν την τιμή
  - Αν δεν υπάρχει τέτοια μη μηδενική τιμή, δεν θα καθοριστεί καθόλου η λειτουργία του newFeat.

```
Rule: LocationType
(
 {Lookup.majorType == location}
):loc
-->
   :loc.Location = {rule = "LocationType", type = :loc.Lookup.minorType}
```

  Αυτό θα ρυθμίσει το "type" χαρακτηριστικό του παραγόμενου location με την τιμή του χαρακτηριστικού "minorType" από το Lookup annotation που θα συνδέεται με την ταμπέλα loc. Αν η αναζήτηση δεν έχει minorType, το location δε θα έχει κανένα χαρακτηριστικό "type".

- **Κενές ή προαιρετικές ταμπέλες**
  Γενικά, ο compiler του JAPE θα ρίξει μια εξαίρεση, αν ο κανόνας του RHS χρησιμοποιήσει μία ετικέτα που δεν υπάρχει στο LHS. Ωστόσο, μπορούν να χρησιμοποιηθούν ετικέτες από προαιρετικά μέρη της LHS.

```
Rule: NP
( (({Token.category == "DT"}):det)?
  (({Token.category ==~ "JJ.*"})*):adjs
  (({Token.category ==~ "NN.*"})+):noun
):np
-->
:det.Determiner = {},
:adjs.Adjectives = {},
:noun.Nouns = {},
:np.NP = {}
```

Εδώ παρατηρούμε ότι ο κανόνας μπορει να ταιριάξει με ένα Token με category την τιμή "NN". Σε αυτή την περίπτωση τα :det και :adjs είναι κενά annotation sets και γι' αυτό το λόγο θα αγνοηθούν στο RHS του κανόνα.

- **Μακροεντολές**

Όπως χρησιμοποιούνται οι μακροεντολές στο LHS, έτσι χρησιμοποιούνται και στο RHS με τη διαφορά ότι πρέπει να συμπεριληφθεί στη μακροεντολή η ετικέτα που ταιριάζει με τον κανόνα του LHS.

```
Macro: UNDERSCORES_OKAY        // separate
:match                                  // lines
{
   AnnotationSet matchedAnns = bindings.get("match");

   int begOffset = matchedAnns.firstNode().getOffset().intValue();
   int endOffset = matchedAnns.lastNode().getOffset().intValue();
   String mydocContent = doc.getContent().toString();
   String matchedString = mydocContent.substring(begOffset, endOffset);

   FeatureMap newFeatures = Factory.newFeatureMap();

   if(matchedString.equals("Spanish"))    {
    newFeatures.put("myrule",  "Lower");
   }
   else   {
    newFeatures.put("myrule",  "Upper");
   }

   newFeatures.put("quality",  "1");
   outputAS.add(matchedAnns.firstNode(), matchedAnns.lastNode(),
                        "Spanish_mark", newFeatures);
}
 Rule: Lower
(
   ({Token.string == "Spanish"})
:match)-->UNDERSCORES_OKAY   // no label here, only macro name

Rule: Upper
(
   ({Token.string == "SPANISH"})
:match)-->UNDERSCORES_OKAY   // no label here, only macro name
```

Παρατηρούμε ότι στη δεύτερη γραμμή υπάρχει η εντολή :match καθώς αυτή είναι η ετικέτα που ταιριάζει με το LHS των κανόνων μετέπειτα.

3.4. Προτεραιότητα

Κάθε γραμματική έχει μία από τις 5 πιθανές μορφές ελέγχου: “brill”, “all”, “first”, “once” και “appelt”. Αυτό καθορίζεται κατά την έναρξη της γραμματικής. Εάν δεν έχει καθοριστεί το στυλ ελέγχου, η προεπιλογή είναι brill, αλλά είναι προτιμότερο να καθορίζεται για λόγους σαφήνειας.

- *Brill*: εξασφαλίζει ότι όταν περισσότεροι από έναν κανόνες αντιστοιχούν στην ίδια περιοχή του εγγράφου, τότε εκτελούνται όλοι. Το αποτέλεσμα αυτού είναι ότι ένα τμήμα κειμένου θα μπορούσε να δεσμέυσει περισσότερες από έναν τύπο οντότητες, και ότι δεν είναι απαραίτητη καμία προτεραιότητα σειράς. Το Brill θα εκτελέσει όλους τους κανόνες που ταιριάζουν ξεκινώντας από μια δεδομένη θέση και θα συνεχίσει τα ταιριάσματα μέχρι τη θέση του εγγράφου όπου το μεγαλύτερο ταίριασμα τελειώνει.
- *All*: Το στυλ “all” είναι παρόμοιο με το Brill, υπό την έννοια ότι θα εκτελέσει επίσης όλους τους κανόνες που ταιριάζουν, αλλά η αντιστοίχιση θα συνεχιστεί από το επόμενο offset μέχρι το τωρινό.
- *First*: ένας κανόνας εκτελείται για το πρώτο ταίριασμα που βρήκε. Αυτό το καθιστά ακατάλληλο για τους κανόνες που καταλήγουν σε “+” ή “?” ή “*”. Μόλις βρεθεί ένα ταίριασμα ο κανόνας εκτελείται και προφανώς δεν επιχειρεί να πάρει ένα μεγαλύτερο ταίριασμα (όπως οι άλλες δύο μορφές).
- *Once*: Με το “once” στυλ, όταν ένας κανόνας εκτελεστεί, μετά το πρώτο ταίριασμα όλη η JAPE φάση τερματίζει.
- *Applet:* μόνο ένας κανόνας μπορεί να εκτελεστεί για την ίδια περιοχή του κειμένου, σύμφωνα με ένα σύνολο κανόνων προτεραιότητας. Η προτεραιότητα λειτουργεί με τον ακόλουθο τρόπο:
    1. Από όλους τους κανόνες που ταιριάζουν με μια περιοχή του εγγράφου ξεκινώντας σε κάποιο σημείο X, θα εκτελεστεί αυτό το οποίο ταιριάζει με τη μακρύτερη περιοχή.
    2. Εάν περισσότεροι από ένας κανόνας αντιστοιχούν στην ίδια περιοχή, θα εκτελεστεί αυτός με την υψηλότερη τιμή προτεραιότητας.
    3. Εάν υπάρχουν περισσότεροι από ένας κανόνες με την ίδια προτεραιότητα, θα εκτελεστεί αυτός που έχει οριστεί σε προηγούμενο σημείο της γραμματικής (αυτός που ορίστηκε πρώτα).

```
Rule:  Location1
Priority: 25

(
 ({Lookup.majorType == loc_key, Lookup.minorType == pre}
  {SpaceToken})?
 {Lookup.majorType == location}
 ({SpaceToken}
  {Lookup.majorType == loc_key, Lookup.minorType == post})?
)
:locName -->
  :locName.Location = {kind = "location", rule = "Location1"}
```

```
Rule: GazLocation
Priority: 20
 (
 ({Lookup.majorType == location}):location
 )
 -->  :location.Name = {kind = "location", rule=GazLocation}
```

Στο παραπάνω παράδειγμα αν θεωρήσουμε ότι έχουμε το κείμενο “Athens mall”, όπου το “Athens” θα υπάρχει στις λίστες του gazetteer ως “location”, ενώ το “mall” θα ορίζεται ως “loc_key”, έχονας ως minorType την τιμή “post”. Τότε θα γινόταν εκτέλεση του πρώτου κανόνα, γιατί θα ταίαριαζε μεγαλύτερη περιοχή του κειμένου ξεκινόντας από την ίδια περιοχή. Τώρα, στην περίπτωση που θεωρούμε ότι έχουμε μόνο το κείμενο “Athens”, και οι δύο κανόνες θα μπορούσαν να εκτελεστούν, αλλά λόγω της μεγαλύτερης προτεραιότητας του πρώτου, θα εκτελεστεί αυτός.

Παρατηρούμε, λοιπόν, ότι η προτεραιότητα είναι μια πολύ σημαντική παράμετρος για τη σύνταξη κάθε γραμματικής. Ωστόσο, εξίσου σημαντικός είναι και ο ορισμός μιας παγκόσμιας προτεραιότητας, δηλαδή προτεραιότητας μεταξύ όλων των φάσεων, αφού η τοποθέτηση όλων των κανόνων σε μια ενιαία γραμματική είναι σχεδόν ανέφικτη λόγω διαφόρων παραγόντων. Έτσι, με τη χρήση ενός αρχείου (main.jape) ορίζονται οι φάσεις που θα χρησιοποιηθούν καθώς και η σειρά με την οποία θα εκτελούνται. Ένας από τους κύριους λόγους για τη χρήση μιας ακολουθίας φάσεων είναι ότι ένα μοτίβο μπορεί να χρησιμοποιηθεί μόνο μία φορά σε κάθε φάση, αλλά μπορεί να επαναχρησιμοποιηθεί σε μια μεταγενέστερη φάση. Σε συνδυασμό με το γεγονός ότι η προτεραιότητα μπορεί να λειτουργήσει μόνο σε μία γραμματική, μπορεί να αξιοποιηθεί για να συμβάλει στην αντιμετώπιση ζητημάτων ασάφειας. Μια λύση που έχει υιοθετηθεί είναι να γραφτεί μια φάση γραμματικής για κάθε είδος σχολιασμού, ή για κάθε συνδυασμό των παρόμοιων ειδών σχολιασμού, με σκοπό τη δημιουργία προσωρινών σχολιασμών. Αυτοί οι προσωρινοί σχολιασμοί γίνονται προσβάσιμοι από μεταγενέστερα στάδια της γραμματικής και μπορούν να τροποποιηθούν με τρόπο που θα επιλύει τις ασάφειες ή θα συγχωνεύει συνεχόμενους σχολιασμούς. Οι προσωρινοί σχολιασμοί μπορούν είτε να αφαιρεθούν αργότερα είτε απλά να αγνοηθούν.

3.5. Χρήση Java στο RHS

Το RHS του κανόνα Jape μπορεί να αποτελείται από οποιοδήποτε κώδικα Java. Αυτό είναι χρήσιμο για την αφαίρεση προσωρινών σχολιασμών και το φιλτράρισμα και το χειρισμό των χαρακτηριστικών προηγούμενων σχολιασμών. Στο παρακάτω παράδειγμα, ο κανόνας που ακολουθεί ταιριάζει με ένα όνομα προσώπου, π.χ. “Fred”, και προσθέτει ένα χαρακτηριστικό φύλου, ανάλογα με την τιμή του minorType από τη λίστα gazetteer στην οποία βρέθηκε το όνομα. Πρώτα, γίνεται λήψη των συνδέσεων που σχετίζονται με την ετικέτα του person (δηλαδή το σχολιασμό Lookup). Στη συνέχεια δημιουργείται ένα νέο annotation που ονομάζεται “personAnn”, το οποίο περιέχει το annotation, και δημιουργείται ένα νέο FeatureMap που δίνει τη δυνατότητα προσθήκης χαρακτηριστικών. Στη συνέχεια γίνεται λήψη του χαρακτηριστικού minorType (και η τιμή του) από τον σχολιασμό personAnn (σε

αυτή την περίπτωση, το χαρακτηριστικό θα είναι “gender” και η τιμή θα είναι “male”), και προστίθεται αυτή η τιμή σε ένα νέο χαρακτηριστικό που ονομάζεται “gender”. Δημιουργείται ακόμα μια ένα χαρακτηριστικό “rule” με την τιμή “FirstName”. Τέλος, όλα τα χαρακτηριστικά προστίθονται σε ένα νέο annotation με όνομα “FirstPerson”, το οποίο συνδέεται με τους ίδιους κόμβους του αρχικού “person”.

```
Rule: FirstName

(
 {Lookup.majorType == person_first}
):person
-->
{
AnnotationSet person = bindings.get("person");
Annotation personAnn = person.iterator().next();
FeatureMap features = Factory.newFeatureMap();
features.put("gender", personAnn.getFeatures().get("minorType"));
features.put("rule", "FirstName");
outputAS.add(person.firstNode(), person.lastNode(), "FirstPerson",
features);
}
```

Ο δεύτερος κανόνας (που περιέχεται στην επόμενη φάση γραμματικής) κάνει χρήση των σχολίων που παράγονται από τον πρώτο κανόνα που περιγράφεται παραπάνω. Αντί για την εύρεση του minorType από το σχολιασμό που παράγεται από το lookup gazetteer, αυτή τη φορά χρησιμοποιεί το χαρακτηριστικό από το σχολιασμό που παράγεται από τον προηγούμενο κανόνα γραμματικής. Έτσι, εδώ παίρνει την τιμή του “gender” χαρακτηριστικού από το σχολιασμό του “FirstPerson” και την προσθέτει σε ένα νέο χαρακτηριστικό (που πάλι ονομάζεται “gender” για λόγους ευκολίας), το οποίο προστίθεται στο νέο σχολιασμό (στο outputAS) “TempPerson”. Στο τέλος του κανόνα αυτού, τα υπάρχοντα annotations εισόδου (από το inputAS) αφαιρούνται, επειδή δεν χρειάζονται πλέον. Σημειώστε ότι στον προηγούμενο κανόνα, δεν αφαιρέθηκαν οι υπάρχουσες σημειώσεις, επειδή ενδέχεται να χρειαστούν αργότερα σε άλλη φάση της γραμματικής.

```
Rule: GazPersonFirst
(
 {FirstPerson}
)
:person
-->
{
AnnotationSet person = bindings.get("person");
Annotation personAnn = person.iterator().next();
FeatureMap features = Factory.newFeatureMap();
 features.put("gender", personAnn.getFeatures().get("gender"));
features.put("rule", "GazPersonFirst");
outputAS.add(person.firstNode(), person.lastNode(), "TempPerson",
features);
inputAS.removeAll(person);  }
```

Μπορεί, επίσης να συνδυαστούν τα μπλοκ της Java και οι κανονικές αναθέσεις (διαχωρίζοντας κάθε μπλοκ ή εκχώρηση από το επόμενο με κόμμα), οπότε το

παραπάνω RHS θα μπορούσε να εκφρασθεί ως εξής:

```
-->
:person.TempPerson = { gender = :person.FirstPerson.gender,
                rule = "GazPersonFirst" },
{
  inputAS.removeAll(bindings.get("person"));
}
```

3.5.1. Named Blocks

Για την συνηθισμένη περίπτωση όπου ένα μπλοκ Java αναφέρεται μόνο στους σχολιασμούς από ένα ενιαίο LHS σύνδεσμο, το Jape παρέχει ένα συμβολισμό για συντομογραφία.

```
Rule: RemoveDoneFlag

(
  {Instance.flag == "done"}
):inst
-->
:inst{
  Annotation theInstance = instAnnots.iterator().next();
  theInstance.getFeatures().remove("flag");
}
```

Αυτός ο κανόνας είναι παρόμοιος με τον παρακάτω:

```
Rule: RemoveDoneFlag
(
  {Instance.flag == "done"}
):inst
-->
{
  AnnotationSet instAnnots = bindings.get("inst");
  if(instAnnots != null && instAnnots.size() != 0) {
    Annotation theInstance = instAnnots.iterator().next();
    theInstance.getFeatures().remove("flag");
  }
}
```

Μια ετικέτα: *<label>* σε ένα μπλοκ Java δημιουργεί μια τοπική *μεταβλητή <label>Annots* εντός του μπλοκ Java, η οποία είναι το *AnnotationSet* συνδεδεμένο με την ετικέτα *<label>*. Επίσης, ο κώδικας Java στο μπλοκ εκτελείται μόνο αν υπάρχει τουλάχιστον ένα σχόλιο συνδεδεμένο με την ετικέτα. Φυσικά, για περισσότερη ευελιξία, π.χ. για εκτέλεση κάποιας δράσης σε περίπτωση που η ετικέτα δεν είναι συνδεδεμένη, θα χρειαστεί να γίνει χρήση ένός μη επισημασμένου μπλοκ και εκτέλεση του *bindings.get ()*.

3.5.2. RHS Embedded

Όταν μια γραμματική Jape αναλύεται, ένα πρόγραμμα ανάλυσης Jape δημιουργεί κλάσεις δράσης για όλα τα Java RHSs της γραμματικής (μια κλάση δράσης ανά RHS). Ο κώδικας Java για το RHS θα ενσωματωθεί ως ένα σώμα της μεθόδου *doit* και θα λειτουργήσει στο πλαίσιο αυτής της μεθόδου. Όταν ένας συγκεκριμένος κανόνας εκτελείται, η μέθοδος *doit* θα εκτελεστεί. Η μέθοδος doit καθορίζεται από τη διαπροσωπεία *gate.jape.RhsAction*. [2]

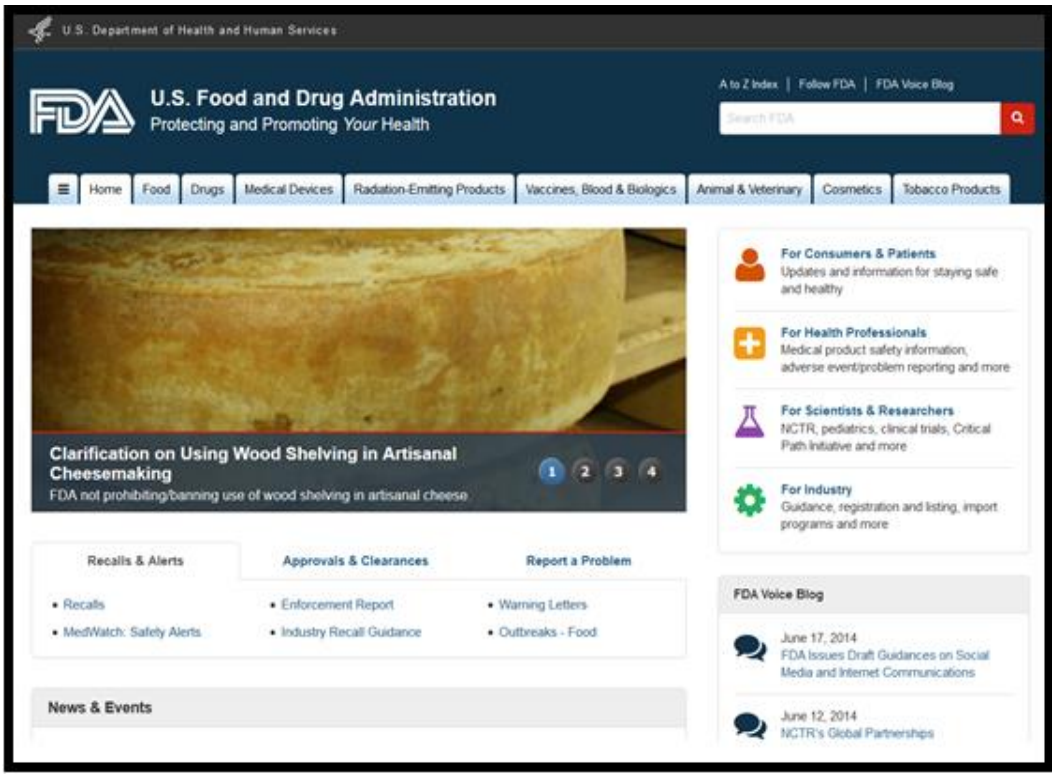
Κεφάλαιο 4

Εξαγωγή δεδομένων

4.1. Εισαγωγή

Στα παραπάνω κεφάλαια έγινε αρχικά μια εισαγωγή πάνω στο natural language processing, ενός από τους πιο σημαντικούς τομείς της τεχνητής νοημοσύνης που έχει συνεχή πρόοδο στις μέρες μας. Ακολούθησε λεπτομερής αναφορά στο πιο βασικό framework του NLP, το GATE, καθώς και σε μεθόδους προγραμματισμού με γραμματικές JAPE για επίτευξη των στόχων των παραπάνω επιστημονικών πεδίων. Ως συνέχεια αυτών, πλέον σε περισσότερο πρακτικό επίπεδο, στο κεφάλαιο αυτό θα γίνει εξονυχιστική ανάλυση πάνω σε εφαρμογές του Information Extraction αναφερόμενες σε συγκεκριμένα δεδομένα, καθώς και ορισμούς και χρήσεις νέων μεθόδων, με σκοπό την απόκτηση χρήσιμων δεδομένων για μετέπειτα επεξεργασία και ανάλυση.

Αντικείμενο μελέτης και έρευνας, καθώς και αφορμή έναρξης της εργασίας αποτέλεσε ένας μεγάλος όγκος δεδομένων, αναφερόμενος σε διατροφικές και ιατροφαρμακευτικές πληροφορίες. Για να είμαστε πιο ακριβείς, παρακάτω θα φανούν τρόποι απόκτησης και επεξεργασίας δεδομένων προερχόμενων από ένα μεγάλο, από άποψη πληροφοριών, site με θέμα την προστασία και προώθηση της ανθρώπινης υγείας. Το εν λόγω site, το www.fda.gov ή αναλυτικότερα U.S. Food and Drug Administration, περιέχει ένα σύνολο καρτέλων και σελίδων url αναφερόμενο πάνω σε τρόφιμα, φάρμακα, προϊόντα, ιατρικές συσκευές κ.α., πληροφοριών δηλαδή χρήσιμων για επεξεργασία από τομείς όπως η βιοιατρική. Στην παρακάτω εικόνα φαίνεται η αρχική του σελίδα με όλες τις πληροφορίες που προαναφέρθηκαν.



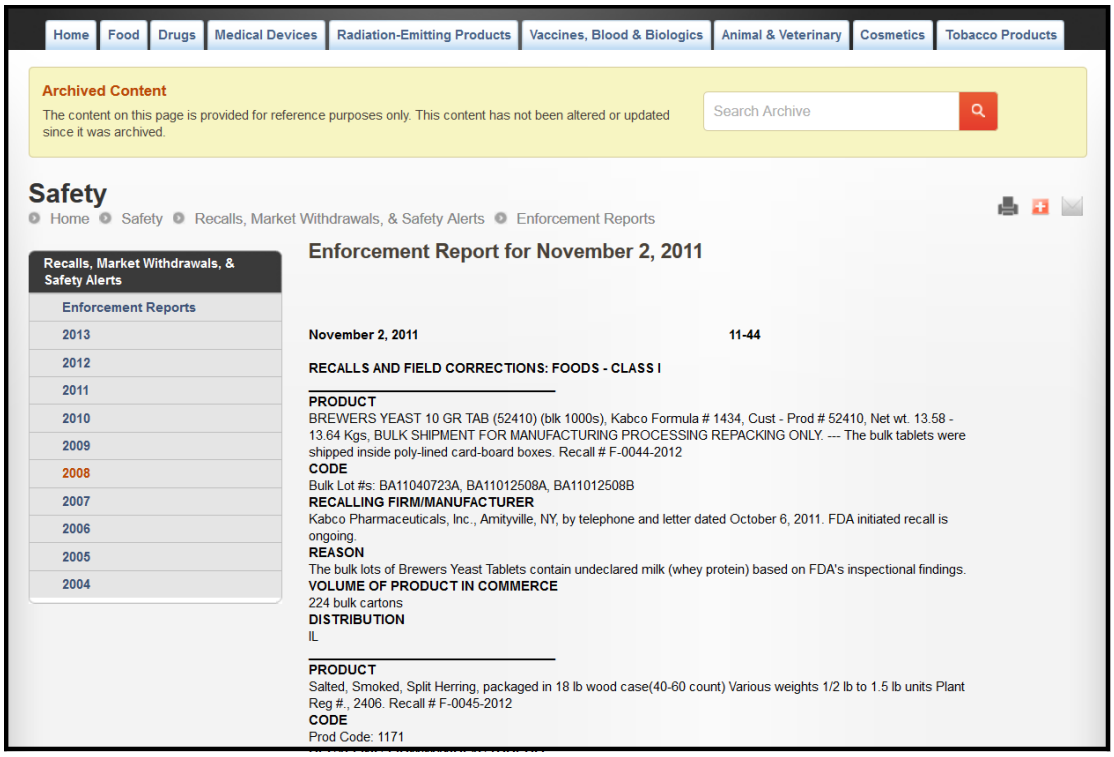
Εικόνα 4.1. Το "Food and Drug Administration" site

Εξερευνώντας αυτό το site, γίνεται εμφανές ότι περιλαμβάνει ένα σύνολο από

Reports - Enforcement Reports – τα οποία είναι αναφορές σε μηνιαία βάση πάνω σε διαφόρων ειδών προϊόντα. Συγκεκριμένα, υπάρχουν τέσσερα reports κάθε μήνα για όλες τις χρονιές από το 2004 μέχρι και το 2013, καθένα από τα οποία περιλαμβάνει ένα συνολο από products σε συνδυασμό με λεπτομερείς πληροφορίες πάνω σε αυτά. Υπάρχουν, λοιπόν, έξι πεδία με διαφορετικού τύπου δεδομένα για κάθε ξεχωριστή περίπτωση προϊόντος, τα οποία είναι τα εξής:

- PRODUCT
- CODE
- RECALLING FIRM/MANUFACTURER
- REASON
- VOLUME OF PRODUCT IN COMMERCE
- DISTRIBUTION

Συνεπώς, μιλάμε για ένα μεγάλο όγκο δεδομένων που η εκμετάλευση και η περαιτέρω ανάλυσή τους ενδέχεται να αποφέρει νέα αποτελέσματα στους τομείς της διατροφολογίας και της ιατρικής . Σε αυτό το σημείο, λοιπόν, εμφανίζεται η ανάγκη για εξαγωγή, απόκτηση και κατηγοριοποίηση των ανωτέρω δεδομένων με σκοπό τη διευκόλυνση της επεξεργασίας τους για δημιουργία νέων αποτελεσμάτων. Παρακάτω θα φανούν οι σκέψεις, οι μέθοδοι και η εκτέλεση αυτών για την επίτευξη του παραπάνω στόχου – ουσιαστικά του Information Extraction - προσαρμόζοντάς τα δεδομένα αυτά σε μια δομημένη μορφή από τη μέχρι τώρα αχανή εμφάνισή τους. Η εικόνα που ακολουθεί δείχνει την αρχή ενός Enforcement Report με όλα τα πεδία και τα δεδομένα που πρέπει να εξαχθούν.



Εικόνα 4.2. Το report των 2 Νοεμβρίου 2011

Η δομεμένη μορφή των δεδομένων, που έχει επετεύχθει στη συνέχεια και θα αναλυθεί διεξοδικά, είναι ένας **πίνακας html** με στήλες τα 6 αυτά πεδία περιλαμβάνοντας το σύνολο των πληροφοριών που εμφανίζεται σε ολόκληρο το report. Για να γίνει ωστόσο αυτό, πρέπει αφενώς να γίνει χρήση νέων κανόνων που αφορούν γραμματικές JAPE και αφεταίρου κατάλληλη χρήση του GATE Embedded

για επεξεργασία των δεδομένων αυτών μέσα από κάποιο γραφικό περιβάλλον για Java.

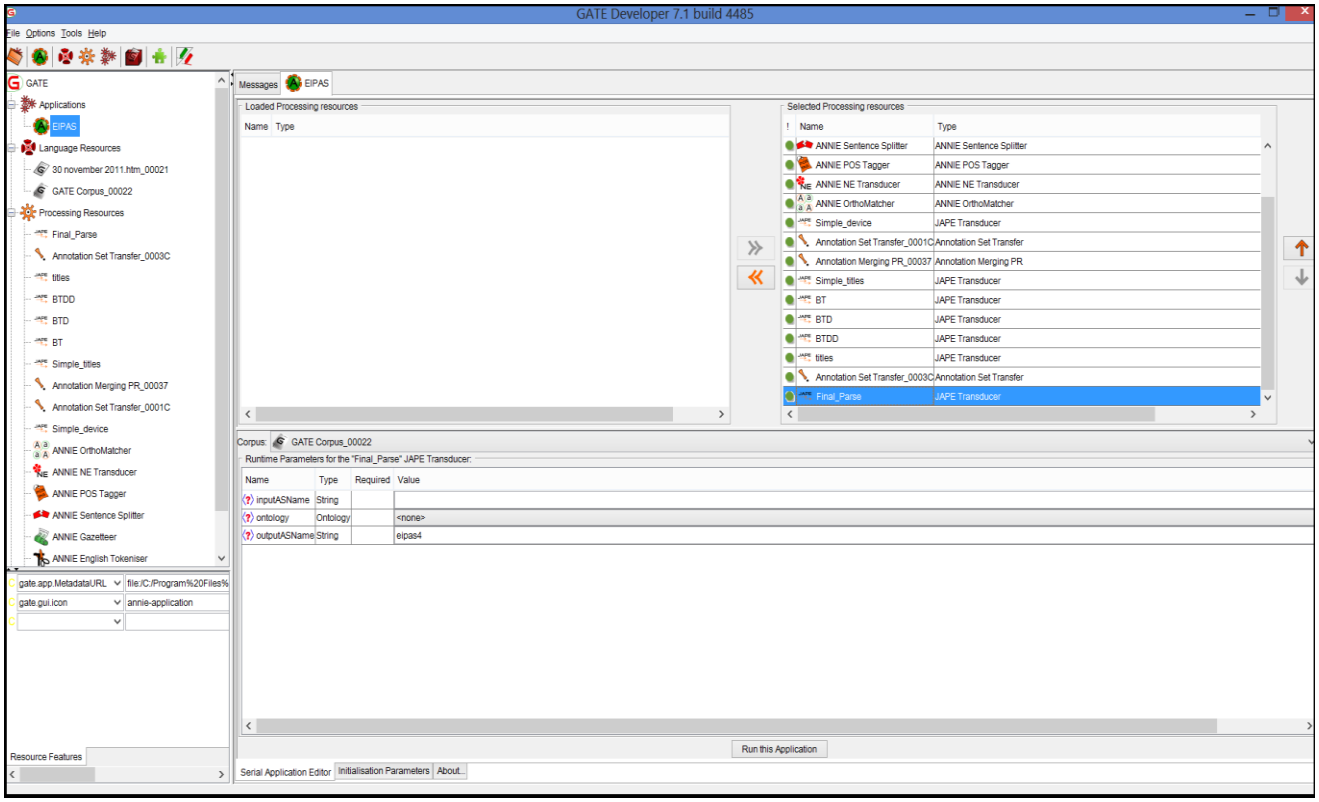
4.2. Το πρώτο βήμα – Δημιουργία νέων Annotations

4.2.1. Το EIPAS pipeline

Αρχικά, για να μπορέσει να γίνει η εξαγωγή των δεδομένων, απαραίτητη προϋπόθεση αποτελεί η εύρεσή τους και ο σχολιασμός τους. Και μιλώντας για σχολιασμό, εννοούμε τα απαιτούμενα annotations που θα πρέπει να δημιουργηθούν και να δεθούν με κάθε κομμάτι κειμένου υποδηλώνοντας τον τύπο του στο πλαίσιο του report. Ωστόσο, η δημιουργία των annotations αποτελεί αρκετά εξαντλητική διαδικασία, καθώς απαιτεί την εκ νέου δημιουργία μιας εφαρμογής (ή pipeline), συμπεριλαμβανομένων κατάλληλα ρυθμισμένων processing resources ικανών για σωστή αναγνώριση του κάθε εγγράφου. Για το σκοπό αυτό δημιουργήσαμε την εφαρμογή **EIPAS**, η οποία έχει τις βάσεις της στην εφαρμογή ANNIE που έχει σχολιαστεί διεξοδικά στο κεφάλαιο 2.2. Συγκεκριμένα, η νέα εφαρμογή περιλαμβάνει τα πρώτα 6 processing resources του ANNIE παρομοίως ορισμένα και στην ίδια σειρά, σε συνδυασμό με ένα σύνολο από άλλα 10 νέα, τα οποία θα δημιουργήσουν τα νέα annotations. Αυτά εμφανίζονται στη σειρά που είναι φορτωμένα για το pipeline EIPAS:

- Simple_device
- Annotation Set Transfer_0001C
- Annotation Merging PR_00037
- Simple_titles
- BT
- BTD
- BTDD
- titles
- Annotation Set Transfer_0003C
- Final_Parse

Όλα αυτά φαίνονται στην παρακάτω εικόνα, η οποία δείχνει το GATE Developer να έχει φορτώσει το EIPAS με το σύνολο των processing resources να είναι selected και έτοιμα να τρέξουν σε κάποιο ορισμένο corpus (τα πρώτα 2 resources, που είναι ίδια με του ANNIE, δεν φαίνονται).



Εικόνα 4.3. Η εφαρμογή EIPAS και τα processing resources της

4.2.2. Ο ρόλος των νέων processing resources

Καθώς ο ρόλος των πρώτων 6 processing resources είναι ήδη ορισμένος στο κεφάλαιο του ANNIE (2.2), παρακάτω φαίνεται η χρησιμότητα των υπόλοιπων πόρων που δημιουργήσαμε με σκοπό το IE των συγκεκριμένων reports. Για να είμαστε ακριβείς, έχει γίνει χρήση 7 Jape Transducers, 2 Annotation Set transfers και ενός Annotation Merging, τα οποία παρουσιάζονται στη σειρά που εκτελούνται. Να γίνει ωστόσο το σχόλιο, ότι για κάθε transducer γίνεται χρήση αρχείων .jape, τα οποία είναι αποθηκευμένα στο pc, με τη διαδικασία της φόρτωσή τους ως GrammarUrl στο πεδίο των Installation Parameters.

- **Simple_device**: περιέχει κανόνες για εύρεση οποιουδήποτε token που θα ξεκινάει με Dev (με “e” και “v” κεφαλαία ή πεζά), και δημιουργεί το annotation type “DTitle” στο custom1 annotation set.
- **Annotation Set Transfer_0001C**: μεταφέρει όσα annotations του custom1 επικαλύπτονται με τα strong annotations του Original Markups και τα τοποθετεί σε ένα νέο annotation set, το eipas. Στην ουσία μεταφέρει τον τύπο DTitle που δημιουργήθηκε από το simple_device στο νέο annotation set, το “eipas”
- **Annotation Merging PR_00037**: συγχωνεύει τα annotation sets “Original markups” και “eipas” σε ένα νέο που λέγεται “eipas2” και το οποίο περιέχει όλα τα types και των δύο sets.
- **Simple_titles**: Παίρνει κάποιο PDescription και ορίζει τους επόμενους strong τύπους με τα annotation types CD, PD, RC, Re, V, και D από τα Code, Product, Recall, Reason, Volume και Distribution αντίστοιχα. Ακόμα,

προσθέτει σε κάθε strong τύπο το annotation type “Tempstrong”.

- **BT**: Δημιουργεί το BT annotation type, το οποίο περιλαμβάνει όλα τα string “ RECALLS AND FIELD CORRECTIONS…”, τα οποία σηματοδοτούν την αρχή της περιγραφής ενός νέου προϊόντος. Επίσης, δημιουργεί το PDpDescription type για κάθε ακολουθία από PD, κενό χαρακτήρα και κάποια πρόταση.*(βλ. 4.3.2 παράδειγμα 1)*
- **BTD**: Δημιουργεί τα εξής annotation types:
    - *BTD*: παίρνει σαν είσοδο το BT annotation type από το προηγούμενο resource και επιλέγει τα BT που περιέχουν μετά το corrections τη λέξη DEVICES.
    - *ED*: Περιλαμβάνει το τελευταίο Token, το οποίο δεν είναι αριθμός, και το σύνολο των κενών χαρακτήρων ακριβώς πριν από κάθε BT ή το τέλος του report, δηλαδή τη φράση “END OF ENFORCEMENT REPORT..”
    - *testtttt*: Περιλαμβάνει όλες τις φράσεις “ RECALLING FIRM/MANUFACTURER”
- **BTDD**: Περιλαμβάνει το μεγαλύτερο τμήμα από JAPE γραμματικές και κανόνες και για το λόγο αυτό θεωρείται το ουσιαστικότερο τμήμα όλου του pipeline. Επίσης, αποτελεί το resource με τη δημιουργία των περισσότερων μεθόδων επεξεργασίας του report που, όπως θα φανεί στη συνέχεια, τα αποτελέσματά τους θα αποτελέσουν θεμελιώδη δεδομένα για την μετέπειτα πορεία του IE. Για τους παραπάνω λόγους, κρίθηκε απαραίτητη η δημιουργία κάποιων επιπλέον φάσεων (phases), οι οποίες έχουν ως στόχο να αφαιρούν χαρακτηριστικά από τα διάφορα Tokens με σκοπό τον εκ νέου ορισμό τους, διευκολύνοντας έτσι τη δημιουργία των νέων annotation types . Γενικά, αυτός ο συγκεκριμένος JAPE transducer είναι υπεύθυνος για τη δημιουργία πολλών νέων annotation types αρκετοί από τους οποίους δημιουργούνται για έμμεση χρήση, ενώ άλλοι καθαρά για λόγους debugging. Σε γενικές γραμμές λοιπόν, δημιουργούνται τα εξής annotation types:
    - *Section*: περιλαμβάνει όλες τις λέξεις που βρίσκονται μεταξύ δύο (long) τιμών και συγκεκριμένα μεταξύ του lastNode κάθε ED annotation και της τιμής του χαρακτηριστικού “temp-last-sentence-end”. Το αποτέλεσμα είναι η δημιουργία annotations καθένα από τα οποία περιλαμβάνει κομμάτι του report με το σύνολο των περιγραφών που υπάρχουν σε ένα “RECALLS AND FIELD CORRECTIONS…” πεδίο. Ουσιαστικά, περιλαμβάνει όλες τις περιγραφές χωρίζοντας τες σε τμήματα ανάλογα με την ύπαρξη των παραπάνων επικεφαλίδων. *(βλ. 4.3.2. παράδειγμα 2)*
    - *Split1*: Περιλαμβάνει δύο ειδών tokens, ανάλογα με την τιμή του χαρακτηριστικού kind, η οποία μπορεί να είναι “internal” ή “external”, σχετικά με το αποτέλεσμα δύο αντίστοιχων κανόνων – split1 και CR. Ο πρώτος κανόνας αναφέρεται σε σημεία στίξης, τελείες ή κενούς χαρακτήρες πρίν από κάποιο τίτλο (PRODUCT, CODE, κλπ.) ή το τέλος του report “END OF ENFORCEMENT REPORT..”. Η δεύτερη περίπτωση έχει μικρότερη προτεραιότητα, καθώς ο κανόνας που δημιουργεί το annotation βρίσκεται μετά τον πρώτο σε σειρά και ουσιαστικά εκτελείται στις περιπτώσεις που βρίσκει κάποιο BT.
    - *Description*: Παίρνει ως είσοδο όλα τα Split1 annotations και με παρόμοιο τρόπο με τη δημιουργία των annotations του Section, δημιουργεί αυτό το νέο annotation type το οποίο περιλαμβάνει κι αυτό

το σύνολο των περιγραφών του report, με διαφορετικό όμως χωρισμό. Συγκεκριμένα, κάθε annotation συνδέεται με τμήματα περιγραφών ανάλογα με τα έξι πεδία. Δηλαδή, ένα annotation περιλαμβάνει κάποιο PRODUCT και τις περιγραφές αυτού του πεδίου, άλλο annotation κάποιο CODE και τις περιγραφές του, ένα άλλο ενός άλλου PRODUCT ή REASON ή VOLUME ή RECALL κ.ο.κ, ξεχωρίζοντας έτσι τις περιγραφές ώστε να έρθει μετά το final_parse να τις κατηγοριοποιήσει στα έξι πεδία. *(βλ.4.3.2. παράδειγμα 3)*

- *BTDescription*: Συνδέεται με τα Section annotations τα οποία περιλαμβάνουν τα BTD annotations. Άρα μιλάμε για τα τμήματα του report που θα έχουν ως επικεφαλίδες αυτές που υποδηλώνει το BTD.

Ωστόσο, όσον αφορά την ορθή και εύκολη λειτουργία των παραπάνω κανόνων, ουσιαστικό ρόλο έχει ο ορισμός ενός συνόλου μακροεντολών. Αυτές κρίνονται απαραίτητες για την περεταίρω επεξεργασία του report, καθώς η χρήση τους θα διευκολύνει τη σύνθεση των υπόλοιπων JAPE γραμματικών και φάσεων για το συγκεκριμένο resource. Για την ακρίβεια, μιλάμε για τη φάση *find*, η οποία περιλαμβάνει τις ακόλουθες μακροεντολές:

- FULLSTOP: βρίσκει αν το token είναι τελεία.
- THREEDOTS: Βρίσκει αν υπάρχουν τρεις ή παραπάνω τελείες στη σερά.
- PUNCT: βρίσκει αν υπάρχει θαυμαστικό ή ερωτηματικό.
- NEWLINE: Βρίσκει αν υπάρχει χαρακτήρας αλλαγής γραμμής.
- TITLE2: Βρίσκει τους τίτλους "VOLUME OF PRODUCT IN COMMERCE" και "DISTRIBUTION".
- LINE: Βρίσκει 35 κάτω παύλες στη σειρά, καθώς αυτή η γραμμή παρουσιάζεται στα reports για να ξεχωρίσει τις αναφορές μεταξύ ξεχωριστών προϊόντων.
- TITLE: Βρίσκει τους έξι τίτλους των πεδίων αναφοράς των προϊόντων (PRODUCT, CODE, κλπ.).
- Eof: Βρίσκει το τέλος του report, για την ακρίβεια τη φράση "END OF ENFORCEMENT REPORT FOR".

- **titles**: Περιλαμβάνει 6 κανόνες, όσα και τα πεδία αναφοράς των προϊόντων, και δημιουργεί 2 annotation types:
  - *Title*: συνδέεται με όλους τους τίτλους των πεδίων, δηλαδή με τα product, code, reason, recall, volume και distribution
  - *fWord*: περιλαμβάνει όλες τις πρώτες λέξεις κάθενός από τα 6 πεδία που προαναφέρθηκαν.

  Δημιουργέι επίσης ένα νέο annotation set, το titles_temp, το οποίο θα περιλαμβάνει ως πεδία τα δύο ανώτερα annotation types (Title και fWord)
- **Annotation Set Transfer_0003C**: Μεταφέρει στο default annotation set τα annotations του titles_temp που δημιουργήθηκαν προηγουμένως από το titles resource, τα οποία επικαλύπτονται με το BTDescription. Ουσιαστικά, γίνεται επιλογή των τίτλων και των πρώτων λέξεων των πεδίων που ανήκουν σε ένα συγκεκριμένο τμήμα του report, αυτό που καθορίζει το BTDescription.
- **Final_Parse**: Είναι ένα από τα πιο βασικά κομμάτια όλης της εφαρμογής, καθώς αποτελεί το τελευταίο στάδιο του Information Extraction των reports. Συγκεκριμένα, περιλαμβάνει 6 κανόνες για το διαχωρισμό των επιμέρους πεδίων του report στα νέα 6 annotation types. Παίρνει, λοιπόν, ως είσοδο τα annotations του Descriptions και τα κατηγοριοποιεί, παράγοντας τα 6 annotation types – ένα για κάθε είδος περιγραφής - τα οποία θα τοποθετήσει

σε ένα νέο annotation set - το "eipas4". Τα annotation types που δημιουργούνται είναι τα εξής:

- CODE
- DISTRIBUTION
- PRODUCT
- REASON
- RECALL
- VOLUME

Καθένα από αυτά περιλαμβάνει το σύνολο όλων των περιγραφών κάθε πεδίου που περιλαμβάνονται σε όλο το report. Έτσι, ουσιαστικά χωρίζουμε το report σε έξι διαφορετικά τμήματα, ανάλογα με το είδος της περιγραφής των προϊόντων. *(βλ. 4.3.2.παράδειγμα 4)*

Αυτά είναι τα processing resources που χρησιμοποιούνται από το EIPAS για τη δημιουργία των annotations που απαιτούνται για την εξαγωγή των δεδομένων. Προφανώς, πολλοί από τους άνωθεν κανόνες και γραμματικές, όπως επίσης και η ύπαρξη κάποιων resources δεν χρησιμεύουν άμεσα στην επίτευξη αυτού του στόχου του συγκεκριμένου project. Ωστόσο, βοήθησαν αρκετά στη συλλογή ενδιάμεσων αποτελεσμάτων και ιδιαίτερα στην εύρεση λαθών (debugging) του συνολικού σχεδιασμού της νέας εφαρμογής. Για το λόγο αυτό κρίθηκε αναγκαία η αναφορά τους, δείχνοντας παράλληλα την πορεία εργασίας που επιχειρήθηκε.

4.2.3. Γραμματικές JAPE και αποτελέσματα αυτών

Παρακάτω φαίνονται κάποια ενδεικτικά τμήματα κώδικα από τις JAPE γραμματικές που χρησιμοποιήθηκαν, ενώ ακολουθούν εικόνες από τα αποτελέσματα αυτών. Γενικά, ως είσοδος έχει χρησιμοποιηθεί τυχαία το report με ημερομηνία 30 Νοεμβρίου του 2011 και τα αποτελέσματα των εικόνων αναφερονται στο ίδιο κομμάτι κειμένου για όλα τα επιλεγμένα πεδία, ώστε να φανεί η διαφορά των annotation types καθως και η σύνδεσή τους για τη δημιουργία του τελικού αποτελέσματος της εικόνας 4.

1. Η πρώτη φάση του BT που δημιουργεί το BT annotation type:

```
Phase:   BT
Input:  BT Sentence Token SpaceToken Lookup
Options: control = appelt debug=true

Rule: BT
({Token.string=="RECALLS"} {SpaceToken}
{Token.string=="AND"}{SpaceToken}{Token.string=="FIELD"}
{SpaceToken}{Token.string=="CORRECTIONS"} {Token.string==":"}
({SpaceToken})*({Token})*({SpaceToken})*({Token})*({SpaceToken})*({Token.string=="-
"})*
({SpaceToken})*({Token})*({SpaceToken})*({Token})*({Token})*({Token})*):bt
-->
:bt.BT={kind="Title",rule=Product}
```

2. Η δεύτερη φάση του resource BTDD που δημιουργεί το Section annotation type:

```
Phase:split
Input: Split TempNoSplitText BT ED Token SpaceToken
Options: control = first

Rule: int
({ED}):isplit3
-->
{
  Long endOffset = ((AnnotationSet)bindings.get("isplit3")).
      lastNode().getOffset();
    Long lastOffset = (Long)doc.getFeatures().get("temp-last-sentence-end");
  if(lastOffset == null) lastOffset = new Long(0);
  AnnotationSet tokens = inputAS.getContained(lastOffset, endOffset);
  if(tokens != null) tokens = tokens.get("Token");
  if(tokens != null && tokens.size() > 0){
    List<Annotation> tokList = new ArrayList<Annotation>(tokens);
    Collections.sort(tokList, new OffsetComparator());
    for(Annotation token : tokList){
      String tokenKind = (String)token.getFeatures().get("kind");
      if("word".equals(tokenKind)){
        Long startOffset = token.getStartNode().getOffset();
        if(startOffset.compareTo(endOffset) < 0){
          //create the new sentence
          try{
            outputAS.add(startOffset, endOffset, "Section",
                  Factory.newFeatureMap());
            //save the new end offset
            doc.getFeatures().put("temp-last-sentence-end", endOffset);
          }catch( InvalidOffsetException ioe){
            throw new GateRuntimeException(ioe);
          }
        }
        return;
      }
    }
  }
}
```

3. Η έκτη φάση του resource BTDD που δημιουργεί το Description annotation type:

```
Phase:split
Input: Split1 TempNoSplitText BT ED Token SpaceToken Lookup DEFAULT_TOKEN
Options: control = all

Rule: int
({Split1.kind == "internal"}):isplit4
-->
{
  Long endOffset = ((AnnotationSet)bindings.get("isplit4")).
      lastNode().getOffset();
    Long lastOffset = (Long)doc.getFeatures().get("temp-last-sentence-end2");
  if(lastOffset == null) lastOffset = new Long(0);
  AnnotationSet tokens = inputAS.getContained(lastOffset, endOffset);
```

```
  if(tokens != null) tokens = tokens.get("Token");
  if(tokens != null && tokens.size() > 0){
   List<Annotation> tokList = new ArrayList<Annotation>(tokens);
   Collections.sort(tokList, new OffsetComparator());
   for(Annotation token : tokList){
    String tokenKind = (String)token.getFeatures().get("kind");
    if("word".equals(tokenKind)){
     Long startOffset = token.getStartNode().getOffset();
     if(startOffset.compareTo(endOffset) < 0){
      //create the new sentence
      try{
       outputAS.add(startOffset, endOffset, "Descriptions",
            Factory.newFeatureMap());
       //save the new end offset
       doc.getFeatures().put("temp-last-sentence-end2", endOffset);
      }catch( InvalidOffsetException ioe){
       throw new GateRuntimeException(ioe);
      }
     }
     return;
    }
   }
  }
}
//sentence that doesn't consume a split
Rule: externalSplits
({Split1.kind == "external"}):esplit
-->
{
  Long endOffset = ((AnnotationSet)bindings.get("esplit")).
     firstNode().getOffset();

  Long lastOffset = (Long)doc.getFeatures().get("temp-last-sentence-end2");
  if(lastOffset == null) lastOffset = new Long(0);

  //get the start offset of the first token.kind==word
  AnnotationSet tokens = inputAS.getContained(lastOffset, endOffset);
  if(tokens != null) tokens = tokens.get("Token");
  if(tokens != null && tokens.size() > 0){
   //we have a more precise end offset
   endOffset = tokens.lastNode().getOffset();
   List<Annotation> tokList = new ArrayList<Annotation>(tokens);
   Collections.sort(tokList, new OffsetComparator());
   for(Annotation token : tokList){
    String tokenKind = (String)token.getFeatures().get("kind");
    if("word".equals(tokenKind)){
     Long startOffset = token.getStartNode().getOffset();
     if(startOffset.compareTo(endOffset) < 0){
      //create the new sentence
      try{
       outputAS.add(startOffset, endOffset, "Descriptions",
            Factory.newFeatureMap());
       doc.getFeatures().put("temp-last-sentence-end2", endOffset);
      }catch( InvalidOffsetException ioe){
```

```
          throw new GateRuntimeException(ioe);
        }
       }
       return;
     }
    }
  }
}
```

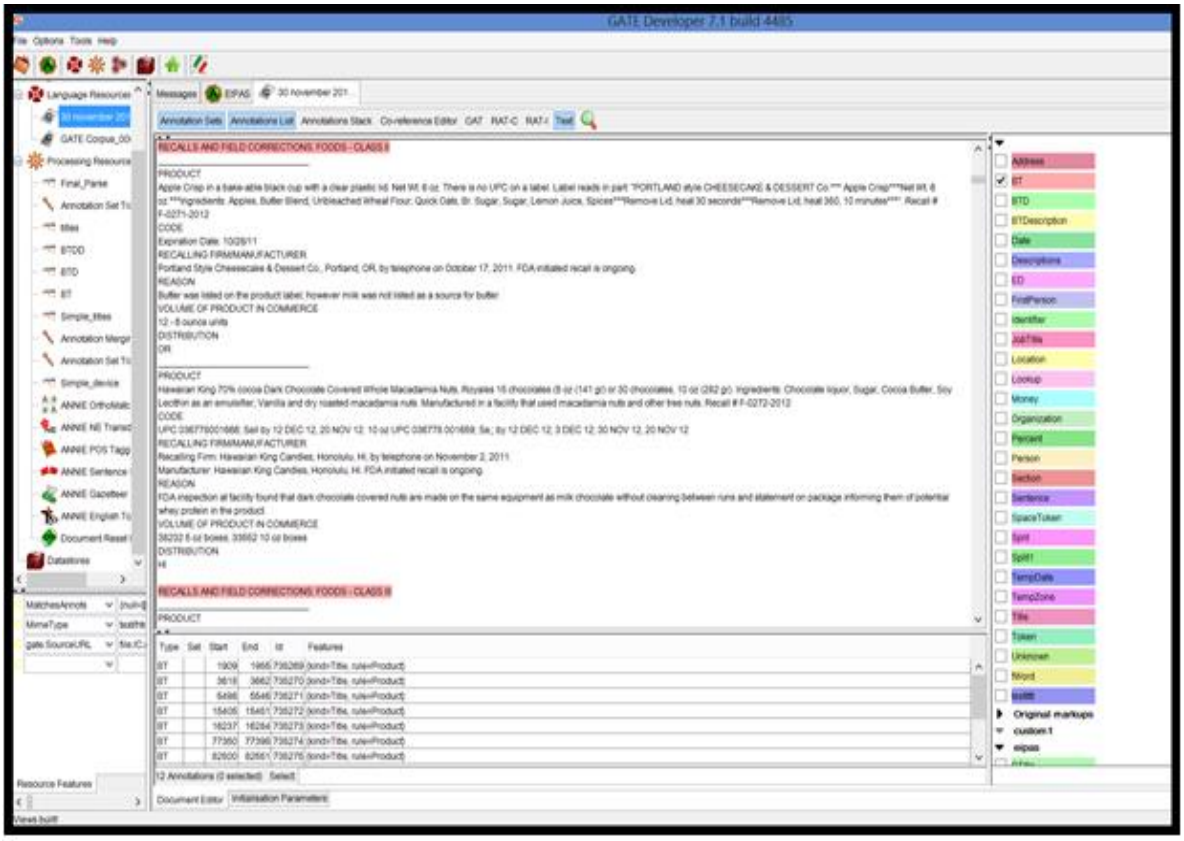
4. To final_parse δημιουργεί τα έξι annotation types για τα έξι είδη περιγραφών:

```
Phase:   SectionDescription
Input:  Descriptions BT Sentence Token SpaceToken Lookup Tempstrong
Options: control = all debug=true

MACRO: RECALLING
({Token.string=="RECALLING"}{SpaceToken}{Token.string=="FIRM"}{Token.string=="/"
}{Token.string=="MANUFACTURER"})
MACRO: VOLU
({Token.string=="VOLUME"}{SpaceToken}{Token.string=="OF"}{SpaceToken}{Token.stri
ng=="PRODUCT"}{SpaceToken}{Token.string=="IN"})

Rule: PDescription
({Descriptions contains Token.string=="PRODUCT",Descriptions notContains
Token.string=="VOLUME" })+:pcc
-->
:pcc.PRODUCT={kind="Parse",rule=PDescription}

Rule: CDescription
({Descriptions contains Token.string=="CODE"}):c
-->
:c.CODE={kind="Parse",rule=CDescription}

Rule: ReDescription
({Descriptions contains Token.string=="REASON"}):r
-->

:r.REASON={kind="Parse",rule=REASON}

Rule: RDescription
({Descriptions contains Token.string=="RECALLING",Descriptions contains
SpaceToken,Descriptions contains Token.string=="FIRM",Descriptions contains
Token.string=="/",Descriptions contains Token.string=="MANUFACTURER"}):re
-->

:re.RECALL={kind="Parse",rule=RDescription}

Rule: VDescription
({Descriptions contains Token.string=="VOLUME",Descriptions contains
SpaceToken,Descriptions contains Token.string=="OF",Descriptions contains
SpaceToken}):v
-->
:v.VOLUME={kind="Parse",rule=VDescription}
```

```
Rule: DDescription
({Descriptions contains Token.string=="DISTRIBUTION"}):d
-->
:d.DISTRIBUTION={kind="Parse",rule=DDescription}
```

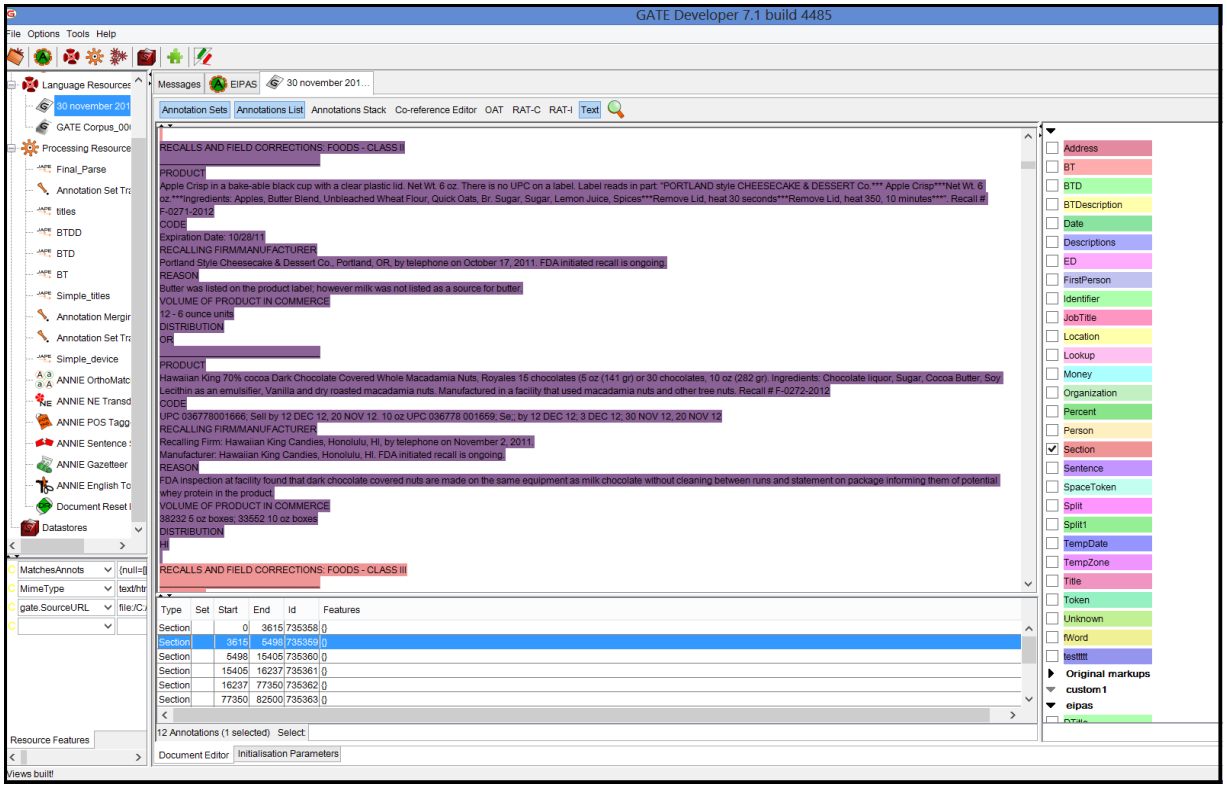
Παρακάτω φαίνονται τμήματα από τα αποτελέσματα των παραπάνω γραμματικών με χρήση του GATE Developer:

1. Φαίνονται δύο από το σύνολο των BT annotations.



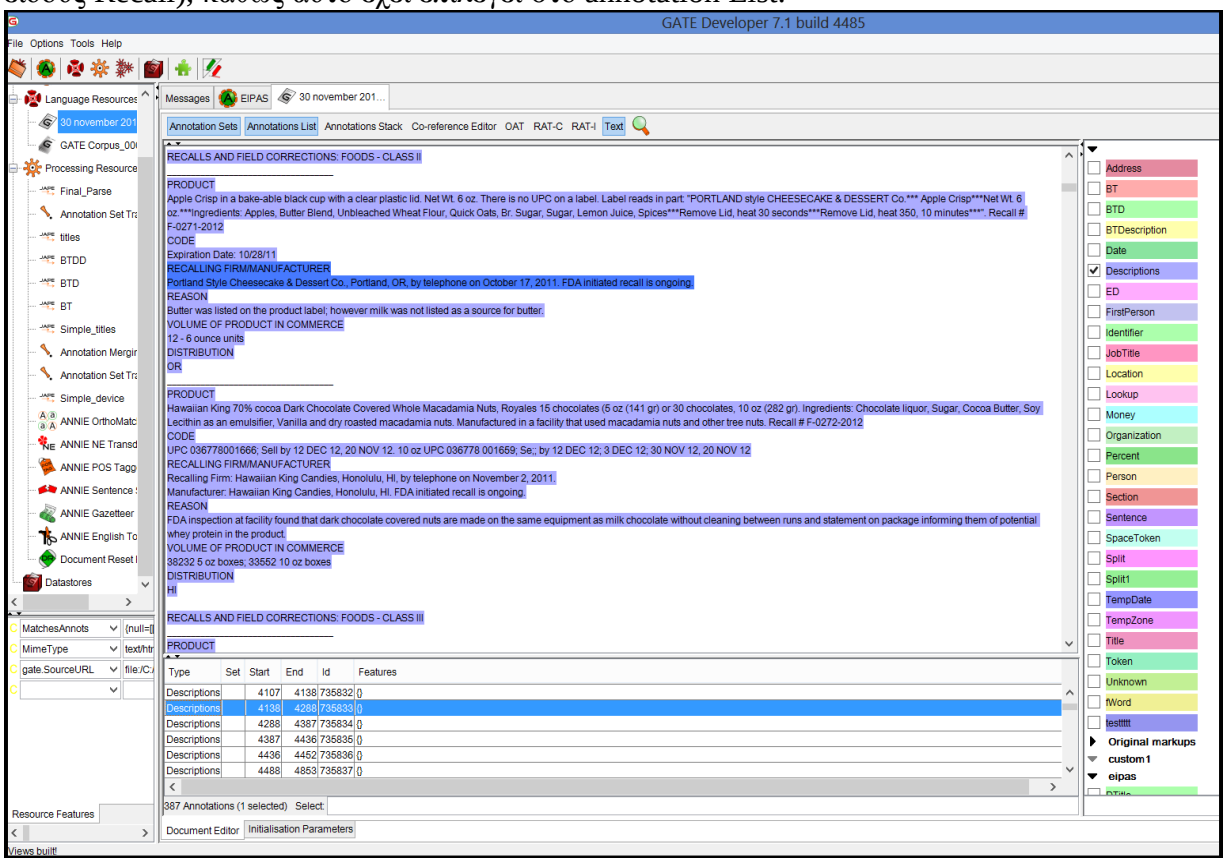
Εικόνα 4.4. Τα δύο πρώτα "BT" annotations

2. Το τμήμα που φαίνεται με το μοβ χρώμα έχει κανονικά το χρώμα του Section, απλά αναβοσβήνει, δηλώνοντας έτσι ότι αυτό είναι το annotation του Section το οποίο έχει επιλεγεί στο Annotation List.



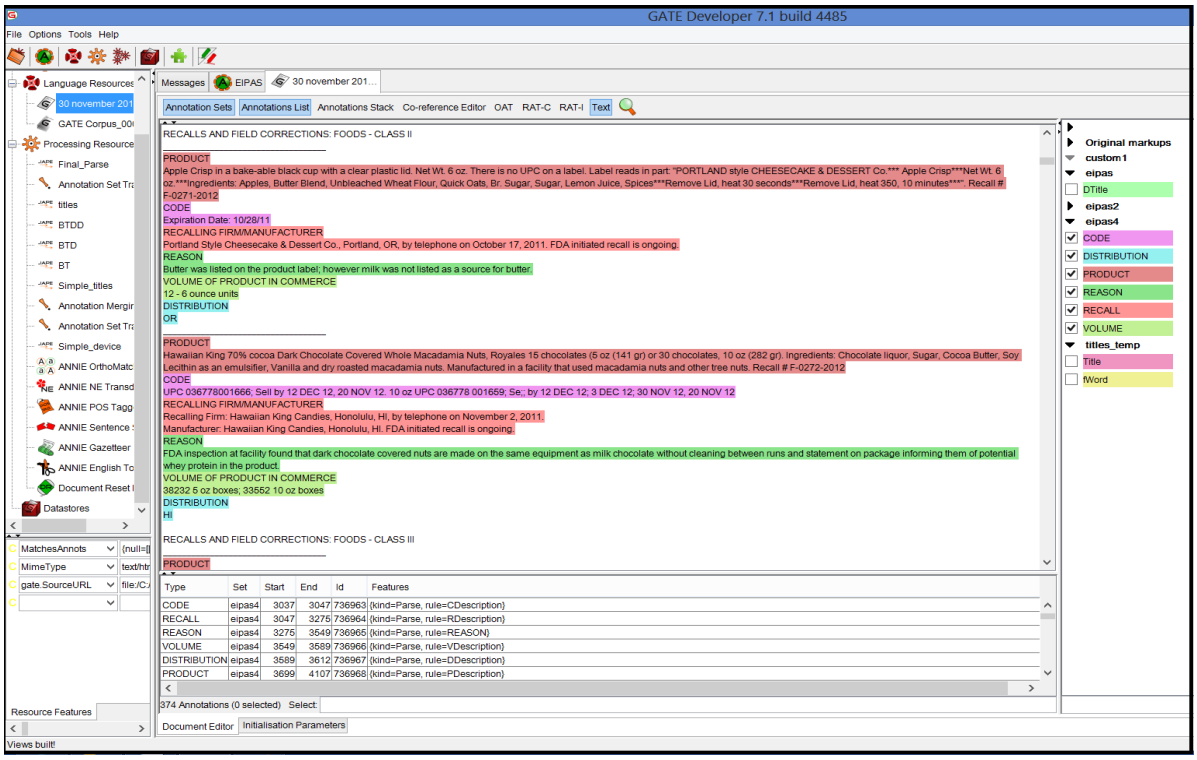
Εικόνα 4.5. Το πρώτο annotation του "Section"

3. Φαίνεται να ξεχωρίζει ένα annotation τύπου Description (και τυγχαίνει να είναι είδους Recall), καθώς αυτό έχει επιλεγεί στο annotation List.



Εικόνα 4.6. Τα "Description" annotations – Επισημασμένο το τρίτο (Recall)

4. Φαίνεται το τελικό αποτέλεσμα, δηλαδή η εκτέλεση του final_parse, το annotation set "eipas4", οι έξι τύποι για κάθε είδος περιγραφής και τα αποτελέσματα από την επιλογή όλων αυτών σε ένα τμήμα του report.



Εικόνα 4.7. Τα annotations των έξι ενδιαφερόμενων πεδίων

4.3. Η λειτουργία του Export – χρήση του GATE Embedded

Σε αυτή την ενότητα θα γίνει ανάλυση του τρόπου επεξεργασίας των δεδομένων των προηγούμενων παραγράφων με σκοπό την τελική εξαγωγή (**export**) τους στη διαχειρίσιμη μορφή ενός html πίνακα. Σε αυτό, λοιπόν, το στάδιο έπαιξε καταλυτικό ρόλο αυτό που ήδη σχολιάστηκε εκτενώς στο κεφάλαιο 2 (παράγραφος 2.3) ως GATE Embedded. Και αυτό γιατί μόνο έτσι μπορούν να φορτωθούν, να ρυθμιστούν και να επεξεργαστούν επιπλέον τα αποτελέματα από την εκτέλεση μιας εφαρμογής (pipeline) σε ένα σύνολο εγγράφων (corpus). Άρα, στο σημείο αυτό εμφανίζεται το GATE Embedded περιλαμβάνοντας ένα σύνολο από jar αρχεία, ικανά να φορτώσουν βιβλιοθήκες που θα επεξεργάζονται πλήρως τα annotations που παράγονται από το GATE Developer. Με αυτό τον τρόπο ανατίθεται η περεταίρω επίλυση του προβλήματος σε κάποιο virtual machine της Java, που στη συγκεκριμένη περίπτωση θα είναι το **Eclipse**.

4.3.1. Ο κώδικας σε JAVA

Το πρώτο στάδιο ήταν δημιουργία του νέου java project με όνομα "eipas" και η προσθήκη των επιπλέον αρχείων .jar, που αναφέρονται στο GATE, στο Java build path αυτού. Έτσι, για παράδειγμα έγινε προσθήκη του gate.jar αρχείου, του log4j.jar

αρχείου και άλλων πολλών για να μπορέσει να γίνει ο σχεδιασμός της εφαρμογής στο Eclipse. Εν συνεχεία, χρησιμοποιήσαμε τα imports που χρειάστηκαν ώστε να μην υπάρχουν προγραμματιστικά προβλήματα, όπως το *import gate.*;*.

Στο κυρίως πρόγραμμα, έγινε χρήση δύο public κλάσεων, του eipas και του SortedAnnotationList, καθώς και μία μέθοδος στην καθεμία, με τη *main()* μέθοδο να βρίσκεται στη βασική κλάση eipas. Τα έγγραφα από τα οποία θα γίνουν export οι πληροφορίες, δηλαδή τα σχετικά reports, δίνονται με μορφή url's στο πεδίο του Eclipse με τα Run Configurations και συγκεκριμένα ως Program Arguments χωριζόμενα με τον κενό χαρακτήρα μεταξύ τους. Συνεπώς, η *main(String args[])* θα έχει ορίσματα τα arguments που μόλις αναφέρθηκαν.

Στο κυρίως πρόγραμμα, αρχικά έπρεπε να γίνει αρχικοποίηση του Gate. Αυτό επετεύχθει με ένα σύνολο εντολών ρύθμισης των αρχείων, άλλου ορισμένου ως GateHome, άλλου ως αρχείου απ'όπου θα παρθούν τα απαιτούμενα plugins ή τα Creole και μερικά ακόμα. Βέβαια, μετά από αυτό, σειρά έχει η δημιουργία του pipeline που σχολιάσαμε διεξοδικά στην προηγούμενη παράγραφο, η οποία μπορεί να γίνει με δύο τρόπους. Ο πρώτος και λιγότερο εύχρηστος είναι η δημιουργία στο χέρι του συνόλου των processing resources που θα απαρτίσουν τη συγκεκριμένη εφαρμογή, καθώς και η ρύθμιση της ορθής σειράς τους. Για παράδειγμα, θα δημιουργούσαμε το Simple_titles με τον ακόλουθο τρόπο:

```
LanguageAnalyser Simple_titles = (LanguageAnalyser)Factory.createResource(
"gate.creole.Transducer", gate.Utils.featureMap("grammarURL", new
File("C:/Users/Geo/Desktop/Διπλωματική/eipas/NE1/main.jape").toURI().toURL(),
                        "encoding", "UTF-8"));
```

Παρατηρούμε, λοιπόν, ότι γίνεται χρήση ενός .jape αρχείου, το οποίο θα περιλαμβάνει τους κανόνες και τις JAPE γραμματικές που ήδη περιγράψαμε για το simple_titles στην παραπάνω παράγραφο – το ίδιο αρχείο που φορτώνεται ως GrammarUrl στα installation Parameters του GATE Developer. Με αντίστοιχο τροπο θα πρέπει να γίνει η δημιουργία όλων των resources για την εισαγωγή τους στο pipeline. Αντ'αυτού, όπως έχει αναφερθεί ξανά στο κεφάλαιο του gate embedded, υπάρχει η δυνατότητα να γίνει χρήση του GATE Developer και των δεδομένων του, χωρίς τη δημιουργία στο χέρι όλης της εφαρμογής στο περιβάλλον του Eclipse. Αυτός είναι και ο δεύτερος τρόπος δημιουργίας του pipeline και για ευνόητους λόγους αυτός που χρησιμοποιήθηκε και στην πράξη στο συγκεκριμένο κώδικα. Έτσι, με την αποθήκευση ενός .state αρχείου από την έτοιμη εφαρμογή EIPAS του Developer – το eipasAppgeorge.state -, φορτώνουμε στο Eclipse το pipeline αυτό, έχοντάς τα όλα έτοιμα:

```
File pluginsHome = new File("C:/Users/Geo/Desktop/Διπλωματική/eipas_app_states");
File annieGapp = new File(pluginsHome, "eipasAppgeorge.state");
CorpusController pipeline =(CorpusController)
        PersistenceManager.loadObjectFromFile(annieGapp);
```

Ακόμα, δημιουργήσαμε για κάθε όρισμα από τα arguments ένα document, ώστε να υπάρχουν όλα τα έγγραφα αποθηκευμένα και έτοιμα για περεταίρω επεξεργασία. Έτσι, δημιουργόντας ένα corpus με το σύνολο αυτών των εγγράφων και τρέχοντας την εφαρμογή πάνω σε αυτό θα έχουμε ακριβώς τα ίδια αποτελέσματα με αυτά του GATE Developer. Αυτό σημαίνει ότι τα annotations, τα annotation types, τα annotation sets και χαρακτηριστικά αυτών θα είναι ορισμένα και θα αναφέρονται επακριβώς σε καθένα από τα έγγραφα που έτρεξαν. Έτσι, χρειάζεται ακόμα ένα είδους φίλτρο επιλογής του annotation set "eipas4", το οποίο περιλαμβάνει τα πεδία

που μας ενδιαφέρουν, καθώς και η επιλογή των annotations μόνο αυτών των έξι πεδίων. Αυτό γίνεται ως εξής:

```
Iterator<Document> iter = corpus.iterator();
Document doc = (Document) iter.next();
AnnotationSet final_parse = doc.getAnnotations("eipas4");
Set<String> annotTypesRequired = new HashSet<String>();
annotTypesRequired.add("CODE");
annotTypesRequired.add("PRODUCT");
annotTypesRequired.add("DISTRIBUTION");
annotTypesRequired.add("REASON");
annotTypesRequired.add("RECALL");
annotTypesRequired.add("VOLUME");
Set<Annotation> peopleAndPlaces = new
            HashSet<Annotation>(final_parse.get(annotTypesRequired));
```

Τώρα ουσιαστικά έχει γίνει η επιλογή των annotations που χρειάζονται και μένει μόνο η εμφάνισή του κειμενου με το οποίο συνδέεται κάθε ένα. Παρόλ'αυτά, πριν την τελική επεξεργασία των annotations, γίνεται και κλήση της μεθόδου της δεύτερης κλάσης (*SortedAnnotationList*) με σκοπό την επίτευξη κάποιας μορφής ταξινόμησης των annotations διαχειρίζοντας και τις περιπτώσεις επικάληψης αυτών.

Γενικά, έχουν γίνει διάφορες προσεγγίσεις παρουσίασης των αποτελεσμάτων της παραπάνω εφαρμογης. Το πρόγραμμα που γράψαμε παρουσιάζει τα αποτελέσματα σε κάποια αρχεία .html τα οποία αποθηκεύονται στο project "eipas" του workspace του Eclipse. Δημιουργούμε, δηλαδή, κάποια αρχεία html καθένα από τα οποία παρουσιάζει με διαφορετικό τρόπο τα αποτελέσματα. Για την ακρίβεια, για κάθε report που εισάγεται στα ορίσματα δημιουργούνται:

- *StANNIE_toXML_1*: ένα αρχείο html με το report ακριβώς χωρίς να είναι αλλαγμένο ως είσοδο (input),
- *StANNIE _1*: ένα αρχείο html με χρωματισμένο όλο το background με διάφορα χρώματα, ανάλογα με τα έξι πεδία, όπως ακριβώς στο GATE Developer.
- *one_table_1*: ένας πίνακας html που έχει όλα τα δεδομένα κατηγοριοποιημένα σε στήλες, ανάλογα με τα πεδία, χωρίς να γίνεται σύμπτηξη αυτών
- *tables_in_row_1*: έξι πίνακες στη σειρά, καθένας από τους οποίους έχει μια στήλη και αποτελεί ένα από τα έξι πεδία προσδιορισμού των προϊόντων.

Οι αριθμοί που υπάρχουν στο τέλος κάθε αρχείου υποδηλώνουν το report - τη θέση του στο σύνολο των ορισμάτων – στο οποίο αναφέρονται. Δημιουργούνται, λοιπόν, τετράδες αρχείων html για κάθε όρισμα εισόδου.

Όπως είναι εύκολα αντιληπτό, η δημιουργία του πρώτου αρχείου αποτελεί εύκολη υπόθεση, καθώς δε γίνεται απολύτως καμία αλλαγή. Η κατασκευή, όμως, των υπόλοιπον τριών απαιτεί ένα σύνολο από strings που περιέχουν εντολές δημιουργίας html αρχείων. Αυτά που χρησιμοποιήθηκαν στο προκείμενο παρουσιάζονται παρακάτω:

//---------- HTML STRINGS----------//

```
String startTagPart_1 = "<span GateID=\"";
String startTagPart_2 = "\" title=\"";
String startTagPart_Product = "\" style=\"background:Red;\">";
String startTagPart_Code = "\" style=\"background:Pink;\">";
String startTagPart_Distribution = "\" style=\"background:Blue;\">";
```

```
String startTagPart_Volume = "\" style=\"background:Yellow;\">";
String startTagPart_Reason = "\" style=\"background:Green;\">";
String startTagPart_Recall = "\" style=\"background:Brown;\">";
String endTag = "</span>";
String str_Product = "<span style=\"background:Red;\">";
String str_Code = "<span style=\"background:Pink;\">";
String str_Distribution = "<span style=\"background:Blue;\">";
String str_Volume = "<span style=\"background:Yellow;\">";
String str_Reason = "<span style=\"background:Green;\">";
String str_Recall = "<span style=\"background:Brown;\">";
// table style
String first = "<div style=\"vertical-align:top;\">";
String last = "</div>";
String table_Product = "<table border=\"1\" style=\"width:20\" align=\"left\">";
String table_Code = "<table border=\"1\" style=\"width:20\" align=\"left\">";
String table_Distribution = "<table border=\"1\" style=\"width:20\" align=\"left\">";
String table_Volume = "<table border=\"1\" style=\"width:20\" align=\"left\">";
String table_Reason = "<table border=\"1\" style=\"width:20\" align=\"left\">";
String table_Recall = "<table border=\"1\" style=\"width:20\" align=\"left\">";
// table headers
String head_Code = "<tr><th> CODE </th></tr>";
String head_Distribution = "<tr><th> DISTRIBUTION </th></tr>";
String head_Product = "<tr><th> PRODUCT </th></tr>";
String head_Reason = "<tr><th> REASON </th></tr>";
String head_Recall = "<tr><th> RECALL </th></tr>";
String head_Volume = "<tr><th> VOLUME </th></tr>";

String row = "<tr>";
String col = "<td>";
String endrow = "</tr>";
String endcol = "</td>";
String endtable = "</table>";

String table = "<table border=\"1\" >";
String head = "<tr><th> PRODUCT </th><th> CODE </th><th> RECALL
</th><th> REASON </th><th> VOLUME </th><th> DISTRIBUTION </th></tr>";
```

Σε γενικές γραμμές, η δημιουργία των τριών άλλων html αρχείων αποτέλεσε μια απλή σχετικά διαδικασία, χωρίς όμως να σημαίνει ότι δεν ανέκυψαν προβλήματα. Το κυριότερο πρόβλημα ήταν η δημιουργία του δεύτερου αρχείου, με τα αποτελέσματα, δηλαδή, να εμφανίζονται όπως ακριβώς στο GATE Developer – διαφορετικά χρωματισμένο background για κάθε πεδίο πληροφορίας. Συγκεκριμένα, το ουσιαστικότερο εμπόδιο ήταν η διατήρηση του ίδιου στιλ. Για παράδειγμα να είναι το χρώμα του background κόκκινο, για κάποιο σύνολο γραμμών που αναφέροναν σε κάποια περιγραφή. Αυτή η αλλαγή γραμμής με το ήταν υπαίτια για τη μη διατήρηση του χρώματος σε όλο το μήκος των περιγραφών, αλλά και για το σειριακό ψάξιμο του αρχείου, με σκοπό την εισαγωγή εντολών ορισμού εκ νέου αυτού του ίδιου στιλ. Η προαναφερθείσα διαδικασία φαίνεται παρακάτω:

```
editableContent.insert((int)insertPositionStart, startTagPart_Product);
editableContent.insert((int)insertPositionStart, currAnnot.getType());
editableContent.insert((int)insertPositionStart, startTagPart_2);
editableContent.insert((int)insertPositionStart,currAnnot.getId().toString());
```

```
editableContent.insert((int)insertPositionStart, startTagPart_1);
int f1 = currAnnot.getType().toString().length();
int f2 = currAnnot.getId().toString().length();
int pop = f1+f2+49;
long j = insertPositionStart+pop;
int flag=0;
long diaf=insertPositionEnd-insertPositionStart;
long bres=0;
String sub = editableContent.substring((int)j,(int)j+1);
while (j<insertPositionEnd+pop)
{
        Out.prln(sub);
        if (sub.equals("<") &&
                editableContent.substring((int)j+1,(int)j+2).equals("d") &&
                editableContent.substring((int)j+2,(int)j+3).equals("i") &&
                editableContent.substring((int)j+3,(int)j+4).equals("v"))
        {
                m=j+5;
                j=j+4;
                bres=bres+4;
        }
        if (sub.equals("<") &&
                editableContent.substring((int)j+1,(int)j+2).equals("/") &&
                editableContent.substring((int)j+2,(int)j+3).equals("d") &&
                editableContent.substring((int)j+3,(int)j+4).equals("i") &&
                editableContent.substring((int)j+4,(int)j+5).equals("v"))
        {
                if(flag==0)
                {
                        editableContent.insert((int)j, endTag);
                        insertPositionEnd = insertPositionEnd + 7;
                        j=j+7;
                }
                else if (flag>0)
                {
                        editableContent.insert((int)m, str_Product);
                        insertPositionEnd = insertPositionEnd + 30;
                        j = j+30;
                        editableContent.insert((int)j, endTag);
                        insertPositionEnd = insertPositionEnd + 7;
                        j=j+7;
                }
                j=j+5;
                bres=bres+5;
                flag++;
        }
        j++;
        bres++;
        if ( bres==diaf && !(sub.equals(">")) )
        {
                editableContent.insert((int)m , str_Product);
                j=j+30;
                editableContent.insert((int)j, endTag);
                j=insertPositionEnd+pop+1;
```

```
        }
        sub = editableContent.substring((int)j,(int)j+1);
}
```

Το τμήμα αυτό του κώδικα αναφέρεται στα annotations τύπου PRODUCT και γι'αυτό παίρνει ως παραμέτρους τα συγκεκριμένα strings και integers που αναφέρονται σε αυτό το πεδίο. Αντίστοιχη διαδικασία εφαρμόζεται σε κάθε πεδίο ξεχωριστά με μικροδιαφορές (μπορεί να είναι λέξεις, μήκη των strings κ.α.).

Όσον αφορά τώρα το τελικό στάδιο της διπλωματικής, την εξαγωγή των δεδομένων σε ένα πίνακα html, παρατηρούμε ότι δημιουργήθηκαν 2 διαφορετικά αρχεία. Το *one_table_1* ελέγχει το είδος του annotation και τοποθετεί το κείμενο με το οποίο είναι αυτό συνδεδεμένο στην αντίστοιχη στήλη. Έτσι, δημιουργείται στο τέλος ένας πίνακας έξι στηλών που όμως σε κάθε σειρά υπάρχει μόνο μία στήλη με πληροφορίες από το report. Παροτρινόμενοι από αυτή τη μη εύκολα διαχειρίσιμη μορφή δεδομένων, δημιουργήσαμε το τελευταίο αρχείο *tables_in_row_1*, το οποίο θα παρουσιάσει τα δεδομένα σε μια εύκολα αναγνώσιμη και επεξεγάσιμη μορφή. Αυτό, ωστόσο, απαιτεί περισσότερο υπολογιστικό χρόνο, που όμως στο ποσοστό του σε σχέση με το συνολικό χρόνο εκτέλεσης του project είναι ελάχιστος. Άρα, στο τελευταίο αρχείο δημιουργούμε έξι διαφορετικούς πίνακες, καθένας από τους οποίους εχει μία στήλη για κάθε ξεχωριστό πεδίο και τους τοποθετούμε στη σειρά, ώστε να είναι εύκολα αναγνώσιμοι. Συνεπώς, ενώ για το one_table μπορούμε με ένα πέρασμα όλων των annotations να το κατασκευάσουμε, για το tables_in_row θα πρέπει να κάνουμε τόσα περάσματα, όσοι και οι πίνακες, δηλαδή έξι. Αυτά φαίνονται πρακτικά στα παρακάτω ενδεικτικά τμήματα κώδικα.

*//---------- **one_table** ----------//*

```
        String geo =
            editableContent.substring((int)insertPositionStart,
                (int)insertPositionEnd);
        if (currAnnot.getType().equals("PRODUCT"))
        {
                gpant77.write(row);
                gpant77.write(col);
                gpant77.write(geo);
                gpant77.write(endcol);
                gpant77.write(col);
                gpant77.write(endcol);
                gpant77.write(col);
                gpant77.write(endcol);
                gpant77.write(col);
                gpant77.write(endcol);
                gpant77.write(col);
                gpant77.write(endcol);
                gpant77.write(col);
                gpant77.write(endcol);
                gpant77.write(endrow);
        }
        else if (currAnnot.getType().equals("CODE"))
        {
                gpant77.write(row);
                gpant77.write(col);
```

```
                gpant77.write(endcol);
                gpant77.write(col);
                gpant77.write(geo);
                gpant77.write(endcol);
                gpant77.write(col);
                gpant77.write(endcol);
                gpant77.write(col);
                gpant77.write(endcol);
                gpant77.write(col);
                gpant77.write(endcol);
                gpant77.write(col);
                gpant77.write(endcol);
                gpant77.write(endrow);
        }
        else if(currAnnot.getType().equals("RECALL"))
                        .
                        .
                        .
```

```
//--------------------------- tables_in_row ---------------------------------------------//
```

```
String geo =    editableContent.substring((int)insertPositionStart,
                (int)insertPositi onEnd);
if (currAnnot.getType().equals("DISTRIBUTION"))
{
        gpant.write(row);
        gpant.write(col);
        gpant.write(geo);
        gpant.write(endcol);
        gpant.write(endrow);
```

4.3.2. Τελικά αποτελέσματα – Exported Data

Παρακάτω, θα φανούν τα αποτελέσματα από το τρέξιμο του τελικού κώδικα Java πάνω στο report των 30 Νοεμβρίου του 2011. Επιλέχθηκε το ίδιο report με αυτό του GATE Developer για λόγους ευκολότερης σύγκρισης αποτελεσμάτων και δημιουργίας σχολιασμών. Τα αποτελέσματα έχουν ως εξής:

- Στην πρώτη εικόνα (4.8), φαίνεται τμήμα του report ακριβώς όπως είναι στο site.
- Στη δεύτερη εικόνα (4.9), φαίνεται το ίδιο τμήμα του report σε html αρχείο (τμήμα του *StANNIE_toXML*).
- Στην τρίτη εικόνα (4.10), φαίνεται το ίδιο τμήμα του report με χρωματισμένα τα διάφορα πεδία (τμήμα του *StANNIE*).
- Στην τέταρτη εικόνα (4.11), φαίνεται τμήμα του πίνακα one_table.
- Στην Πέμπτη εικόνα (4.12), φαίνεται τμήμα του tables_in_row.

1)

DISTRIBUTION
Nationwide

RECALLS AND FIELD CORRECTIONS: FOODS - CLASS II

PRODUCT
Apple Crisp in a bake-able black cup with a clear plastic lid. Net Wt. 6 oz. There is no UPC on a label. Label reads in part: "PORTLAND style CHEESECAKE & DESSERT Co.*** Apple Crisp***Net Wt. 6 oz.***Ingredients: Apples, Butter Blend, Unbleached Wheat Flour, Quick Oats, Br. Sugar, Sugar, Lemon Juice, Spices***Remove Lid, heat 30 seconds***Remove Lid, heat 350, 10 minutes***". Recall # F-0271-2012
CODE
Expiration Date: 10/28/11
RECALLING FIRM/MANUFACTURER
Portland Style Cheesecake & Dessert Co., Portland, OR, by telephone on October 17, 2011. FDA initiated recall is ongoing.
REASON
Butter was listed on the product label; however milk was not listed as a source for butter.
VOLUME OF PRODUCT IN COMMERCE
12 - 6 ounce units
DISTRIBUTION
OR

PRODUCT
Hawaiian King 70% cocoa Dark Chocolate Covered Whole Macadamia Nuts, Royales 15 chocolates (5 oz (141 gr) or 30 chocolates, 10 oz (282 gr). Ingredients: Chocolate liquor, Sugar, Cocoa Butter, Soy Lecithin as an emulsifier, Vanilla and dry roasted macadamia nuts. Manufactured in a facility that used macadamia nuts and other tree nuts. Recall # F-0272-2012
CODE
UPC 036778001666; Sell by 12 DEC 12, 20 NOV 12. 10 oz UPC 036778 001659; Se;; by 12 DEC 12; 3 DEC 12; 30 NOV 12, 20 NOV 12
RECALLING FIRM/MANUFACTURER
Recalling Firm: Hawaiian King Candies, Honolulu, HI, by telephone on November 2, 2011.
Manufacturer: Hawaiian King Candies, Honolulu, HI. FDA initiated recall is ongoing.
REASON
FDA inspection at facility found that dark chocolate covered nuts are made on the same equipment as milk chocolate without cleaning between runs and statement on package informing them of potential whey protein in the product.
VOLUME OF PRODUCT IN COMMERCE
38232 5 oz boxes; 33552 10 oz boxes
DISTRIBUTION
HI

RECALLS AND FIELD CORRECTIONS: FOODS - CLASS III

PRODUCT

Εικόνα 4.8. Τμήμα του report των 2 Νοεμβρίου 2011

2)

DISTRIBUTION
Nationwide

RECALLS AND FIELD CORRECTIONS: FOODS - CLASS II

PRODUCT
Apple Crisp in a bake-able black cup with a clear plastic lid. Net Wt. 6 oz. There is no UPC on a label. Label reads in part: "PORTLAND style CHEESECAKE & DESSERT Co.*** Apple Crisp***Net Wt. 6 oz.***Ingredients: Apples, Butter Blend, Unbleached Wheat Flour, Quick Oats, Br. Sugar, Sugar, Lemon Juice, Spices***Remove Lid, heat 30 seconds***Remove Lid, heat 350, 10 minutes***". Recall # F-0271-2012
CODE
Expiration Date: 10/28/11
RECALLING FIRM/MANUFACTURER
Portland Style Cheesecake & Dessert Co., Portland, OR, by telephone on October 17, 2011. FDA initiated recall is ongoing.
REASON
Butter was listed on the product label; however milk was not listed as a source for butter.
VOLUME OF PRODUCT IN COMMERCE
12 - 6 ounce units
DISTRIBUTION
OR

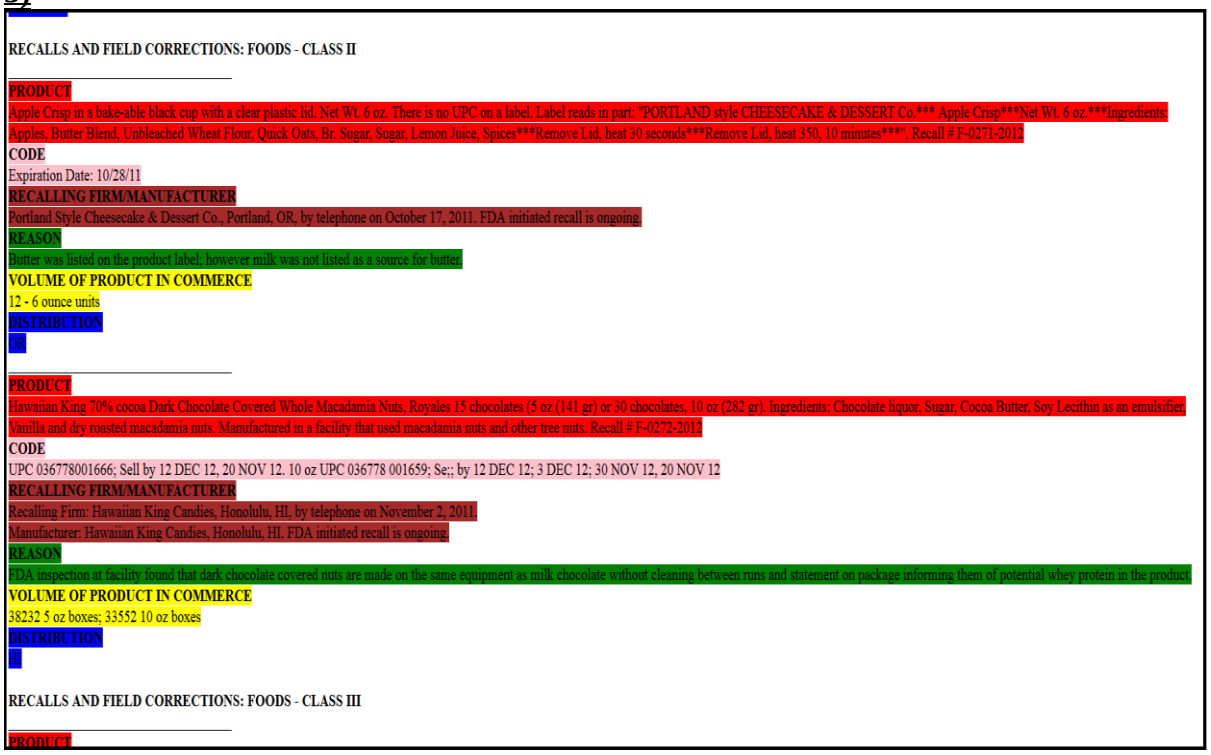
PRODUCT
Hawaiian King 70% cocoa Dark Chocolate Covered Whole Macadamia Nuts, Royales 15 chocolates (5 oz (141 gr) or 30 chocolates, 10 oz (282 gr). Ingredients: Chocolate liquor, Sugar, Cocoa Butter, Soy Lecithin as an emulsifier, Vanilla and dry roasted macadamia nuts. Manufactured in a facility that used macadamia nuts and other tree nuts. Recall # F-0272-2012
CODE
UPC 036778001666; Sell by 12 DEC 12, 20 NOV 12. 10 oz UPC 036778 001659; Se;; by 12 DEC 12; 3 DEC 12; 30 NOV 12, 20 NOV 12
RECALLING FIRM/MANUFACTURER
Recalling Firm: Hawaiian King Candies, Honolulu, HI, by telephone on November 2, 2011.
Manufacturer: Hawaiian King Candies, Honolulu, HI. FDA initiated recall is ongoing.
REASON
FDA inspection at facility found that dark chocolate covered nuts are made on the same equipment as milk chocolate without cleaning between runs and statement on package informing them of potential whey protein in the product.
VOLUME OF PRODUCT IN COMMERCE
38232 5 oz boxes; 33552 10 oz boxes
DISTRIBUTION
HI

RECALLS AND FIELD CORRECTIONS: FOODS - CLASS III

PRODUCT

Εικόνα 4.9. Το ανώτερο τμήμα του report σε html αρχείο δημιουργημένο από το EIPAS

3)



RECALLS AND FIELD CORRECTIONS: FOODS - CLASS II

PRODUCT
Apple Crisp in a bake-able black cup with a clear plastic lid. Net Wt. 6 oz. There is no UPC on a label. Label reads in part: "PORTLAND style CHEESECAKE & DESSERT Co.*** Apple Crisp***Net Wt. 6 oz.***Ingredients: Apples, Butter Blend, Unbleached Wheat Flour, Quick Oats, Br. Sugar, Sugar, Lemon Juice, Spices***Remove Lid, heat 30 seconds***Remove Lid, heat 350, 10 minutes***". Recall # F-0271-2012
CODE
Expiration Date: 10/28/11
RECALLING FIRM/MANUFACTURER
Portland Style Cheesecake & Dessert Co., Portland, OR, by telephone on October 17, 2011. FDA initiated recall is ongoing.
REASON
Butter was listed on the product label; however milk was not listed as a source for butter.
VOLUME OF PRODUCT IN COMMERCE
12 - 6 ounce units
DISTRIBUTION
OR

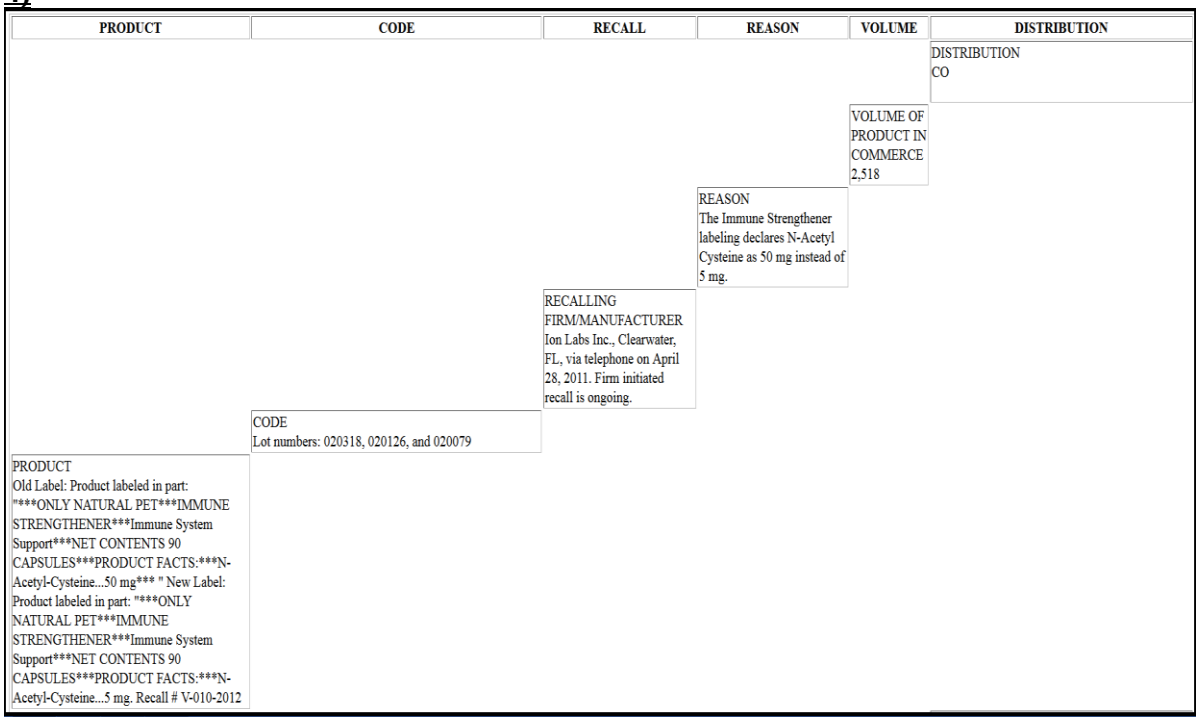
PRODUCT
Hawaiian King 70% cocoa Dark Chocolate Covered Whole Macadamia Nuts, Royales 15 chocolates (5 oz (141 gr) or 30 chocolates, 10 oz (282 gr). Ingredients: Chocolate liquor, Sugar, Cocoa Butter, Soy Lecithin as an emulsifier, Vanilla and dry roasted macadamia nuts. Manufactured in a facility that used macadamia nuts and other tree nuts. Recall # F-0272-2012
CODE
UPC 036778001666; Sell by 12 DEC 12, 20 NOV 12. 10 oz UPC 036778 001659; Se;; by 12 DEC 12; 3 DEC 12; 30 NOV 12, 20 NOV 12
RECALLING FIRM/MANUFACTURER
Recalling Firm: Hawaiian King Candies, Honolulu, HI, by telephone on November 2, 2011.
Manufacturer: Hawaiian King Candies, Honolulu, HI. FDA initiated recall is ongoing.
REASON
FDA inspection at facility found that dark chocolate covered nuts are made on the same equipment as milk chocolate without cleaning between runs and statement on package informing them of potential whey protein in the product.
VOLUME OF PRODUCT IN COMMERCE
38232 5 oz boxes; 33552 10 oz boxes
DISTRIBUTION
HI

RECALLS AND FIELD CORRECTIONS: FOODS - CLASS III

PRODUCT

Εικόνα 4.10. Το report με επισημασμένα τα έξι διαφορετικά πεδία

4)



PRODUCT	CODE	RECALL	REASON	VOLUME	DISTRIBUTION
					DISTRIBUTION CO
				VOLUME OF PRODUCT IN COMMERCE 2,518	
			REASON The Immune Strengthener labeling declares N-Acetyl Cysteine as 50 mg instead of 5 mg.		
		RECALLING FIRM/MANUFACTURER Ion Labs Inc., Clearwater, FL, via telephone on April 28, 2011. Firm initiated recall is ongoing.			
	CODE Lot numbers: 020318, 020126, and 020079				
PRODUCT Old Label: Product labeled in part: "***ONLY NATURAL PET***IMMUNE STRENGTHENER***Immune System Support***NET CONTENTS 90 CAPSULES***PRODUCT FACTS:***N-Acetyl-Cysteine...50 mg*** " New Label: Product labeled in part: "***ONLY NATURAL PET***IMMUNE STRENGTHENER***Immune System Support***NET CONTENTS 90 CAPSULES***PRODUCT FACTS:***N-Acetyl-Cysteine...5 mg. Recall # V-010-2012					

Εικόνα 4.11. Τμήμα του html αρχείου "one_table"

5)

PRODUCT	CODE	RECALL	REASON	VOLUME	DISTRIBUTION
PRODUCT Old Label: Product labeled in part: "***ONLY NATURAL PET***IMMUNE STRENGTHENER***Immune System Support***NET CONTENTS 90 CAPSULES***PRODUCT FACTS:***N-Acetyl-Cysteine...50 mg*** " New Label: Product labeled in part: "***ONLY NATURAL PET***IMMUNE STRENGTHENER***Immune System Support***NET CONTENTS 90 CAPSULES***PRODUCT FACTS:***N-Acetyl-Cysteine...5 mg. Recall # V-010-2012	CODE Lot numbers: 020318, 020126, and 020079	RECALLING FIRM/MANUFACTURER Ion Labs Inc., Clearwater, FL, via telephone on April 28, 2011. Firm initiated recall is ongoing.	REASON The Immune Strengthener labeling declares N-Acetyl Cysteine as 50 mg instead of 5 mg.	VOLUME OF PRODUCT IN COMMERCE 2,518	DISTRIBUTION CO
PRODUCT 28% Catfish Feed. Recall # V-009-2012	CODE none, bulk product	RECALLING FIRM/MANUFACTURER Recalling Firm: Melick Aquafeed, Catawissa, PA, by letter on 10/26/2011. Manufacturer: Melick Aquafeed LLC, Greenville, MI. Firm initiated recall is ongoing.	REASON Catfish feed was manufactured using corn that was found to be above the legal limit for Aflatoxin.	VOLUME OF PRODUCT IN COMMERCE 300 tons	DISTRIBUTION TX, MO, AR
PRODUCT 6" - 8" Merrick Doggie Wishbones All Natural Beef Chew For Dogs, UPC 0 22808 29050 6. Recall # V-008-2012	CODE Lot 11031, Best By 30 Jan 2013	RECALLING FIRM/MANUFACTURER Recalling Firm: Tejas Industries Inc., Hereford, TX, by press release on August 8, 2011. Manufacturer: Tejas Industries Inc., Plainview, TX. FDA initiated recall is ongoing.	REASON The product has the potential to be contaminated with *Salmonella*.	VOLUME OF PRODUCT IN COMMERCE 248 cases	DISTRIBUTION NY, WA, KS, CA, IL, MD, RI, FL, and GA
PRODUCT STERIS 5085 and 5085SRT Surgical Tables, Catalog Numbers: 5085-ST01-410-1; 5085SRT- ST0-1420-6. Recall # Z-0279-2012	CODE Tables identified with the following Codes are subject to correction: 0401510077 to 0421411110. (Note: not all tables manufactured in this range are affected by this voluntary field correction).	RECALLING FIRM/MANUFACTURER Recalling Firm: Steris Corp. Mentor, OH, by letter on October 17, 2011. Manufacturer: Steris Corp, Montgomery, AL. Firm initiated recall is ongoing.	REASON Some hydraulic column cylinders installed in certain 5085 and 5085SRT Surgical Tables were assembled by their supplier with incorrect snap rings and an anomaly was present in the tilt cylinder. This error may affect the user's ability to move the table top out of the full right tilt position.	VOLUME OF PRODUCT IN COMMERCE 167 units. 5085 = 108 & 5085SRT = 59	DISTRIBUTION Nationwide
PRODUCT Stryker Endoscopy 3.55 MM Super 90 S-SERFAS Energy Probe; electrosurgical device. Model number: 279-351-300. Recall # Z-0265-2012	CODE Serial Numbers: 1P00100 - 1P00XXX	RECALLING FIRM/MANUFACTURER	REASON	VOLUME OF PRODUCT IN COMMERCE 6307 probes	DISTRIBUTION Nationwide, Germany, South America, Italy, Saudi Arabia, United Kingdom, and Canada
PRODUCT Spectra Optia Apheresis System, REF:	CODE All lots				DISTRIBUTION Nationwide and Internationally
	CODE Product Code Lot # 81-2093 G03411108 G03680409 81-2094 G03421108 G03690409 81-3573 1000 1002 1527 G03251108 G03630409 G03920709				DISTRIBUTION Nationwide and Internationally
					DISTRIBUTION NY
					DISTRIBUTION CA
					DISTRIBUTION CA
					DISTRIBUTION CA
					DISTRIBUTION NC
					DISTRIBUTION

Εικόνα 4.12. Τμήμα του τελικού html αρχείου “tables_in_row”

Κεφάλαιο 5

Future Work

5.1. Σύνοψη

Το γενικό πλαίσιο της παρούσας εφαρμογής αναφέρεται στην εξαγωγή δεδομένων από συγκεκριμένες ιατρικοφαρμακευτικές και διατροφικές σελίδες με σκοπό τη διευκόλυνση της περεταίρω επεξεργασίας τους από άλλα συστήματα. Οι σελίδες που δόθηκαν ως είσοδος, ήταν ένα σύνολο από reports που περιλαμβάνουν περιγραφές διαφόρων ιατρικών και διατροφικών προϊόντων, χωρισμένες σε έξι διαφορετικά είδη σχολιασμών. Η υλοποίηση αυτή έκανε χρήση του γενικού open source software του GATE με σκοπό τη δημιουργία εξειδικευμένων, στα δεδομένα που απαιτούνται, αλγορίθμων. Η επίτευξη του τελικού αποτελέσματος προέκυψε από μια σταδιακή πορεία εμβάθυνσης στον τομέα του text engineering. Συγκεκριμένα, το αρχικό στάδιο ήταν η ενασχόληση με το GATE Developer, το βασικό IDE του GATE, και η κατανόηση της λειτουργίας του κυρίως στο πλαίσιο εφαρμογών σχετικών με το ANNIE, το βασικό IE του GATE . Έπειτα, σειρά είχε το GATE Embedded και η χρήση της υπάρχουσας βιβλιοθήκης Java που διαθέτει, για την επεξεργασία εγγράφων. Το ουσιαστικότερο, ωστόσο, στάδιο ήταν η δημιουργία της εφαρμογής EIPAS, μιας εφαρμογής βασισμένης στο ANNIE, αλλά επιπλέον προγραμματισμένης για την παραγωγή συγκεκριμένων αποτελεσμάτων. Σχετικά με αυτή την υλοποίηση, απαραίτητη προϋπόθεση αποτέλεσε η γνώση της γλώσσας JAPE, με σκοπό τη δημιουργία annotations συνδεδεμένων με το κείμενο ικανών για την ανάλυση που χρειάζεται. Έτσι, δημιουργήσαμε ένα σύνολο κανόνων στηριζόμενων σε τέτοιες γραμματικές και με την προσθήκη κάποιων resources ολοκληρώσαμε το σχεδιασμό της εφαρμογής. Το τελευταίο τμήμα της διπλωματικής στηρίχθηκε στο Eclipse και στην τελική επεξεργασία των αποτελεσμάτων του EIPAS, κάνοντας χρήση του GATE Embedded. Τα δεδομένα από τα reports στα οποία έτρεξε η συγκεκριμένη εφαρμογή, θα δομηθούν σε μια εύκολα επεξεργάσιμη μορφή έξι πινάκων html στη σειρά. Άρα, τα αποτελέσματα έχουν εξαχθεί και πλέον είναι έτοιμα για ανάλυση και συλλογή συμπερασμάτων.

5.2. Συμπεράσματα

Γενικά, η εξόρυξη δεδομένων (**data mining**) – ως υπερσύνολο του IE - χρησιμοποιείται κυρίως σήμερα από εταιρείες με ισχυρή εστίαση στον πελάτη – δηλαδή οργανισμούς λιανικής πώλησης, οικονομικών, επικοινωνίας και μάρκετινγκ. Η εξόρυξη δεδομένων έχει μεγάλη σημασία λόγω της τεράστιας εφαρμογής της. Χρησιμοποιείται όλο και περισσότερο σε επιχειρηματικές εφαρμογές για την κατανόηση και εν συνεχεία πρόβλεψη πολύτιμων δεδομένων, όπως οι αγοραστικές δράσεις των καταναλωτών και η τάση της αγοράς, τα προφίλ των πελατών, η ανάλυση της βιομηχανίας Έτσι, τομείς ενδιαφέροντος του data mining εκτός των άλλων είναι και το άμεσο μάρκετινγκ, η βιοπληροφορική, η γενετική, η ανάλυση κειμένου, η διαχείριση πελατειακών σχέσεων και οι χρηματοπιστωτικές υπηρεσίες. [41]

Σχετικά με το **Information Extraction**, υπάρχουν πολλοί λόγοι που το καθιστούν ένα πολύ σημαντικό επιστημονικό πεδίο. Αρχικά, χρησιμοποιείται για να συνοψίσουμε μελέτες σε μια κοινή μορφή, για τη διευκόλυνση της σύνθεσης και συνεκτικής παρουσίασης των δεδομένων. Ακόμα, βοηθάει στον προσδιορισμό αριθμητικών δεδομένων για μετα-αναλύσεις και στη λήψη πληροφορίων για την αντικειμενική αξιολόγηση του κινδύνου μεροληψίας και εφαρμογής των μελετών.

Τέλος, αποτελεί χρήσιμο εργαλείο για τον συστηματικό εντοπισμό δεδομένων που λείπουν ή που έχουν εκτιμηθεί εσφαλμένα, ακόμα και αποτελεσμάτων που δεν έχουν μελετηθεί μέχρι τώρα. [42]

Όσον αφορά τη μελέτη του συνόλου των δεδομένων που εξήχθησαν, υπάρχουν πολλές διαφορετικές προσεγγίσεις. Μία συνήθης προσέγγιση είναι η χρήση τους από άλλα, πιο εξειδικευμένα συστήματα, που επιτελούν συγκεκριμένες λειτουργίες. Ωστόσο, η πιο συνηθισμένη εφαρμογή είναι η εκπόνηση στατιστικών αναλύσεων. Με απλά λόγια, τα δεδομένα αυτά μπορούν να αποτελέσουν ένα πολύ καλό αρχείο εισόδου για αλγόριθμους κατασκευασμένους με στόχο την ανάλυση πληροφοριών για εξαγωγή στατιστικών στοιχείων. Έτσι για παράδειγμα, οι πληροφορίες των προϊόντων μπορούν με κατάλληλο χειρισμό να κατηγοριοποιήσουν τα προϊόντα ανάλογα με ποσοστά επιτυχίας, με ημερομηνίες λήψεών τους, με σύνολα μονάδων/κομματιών που χρησιμοποιήθηκαν και με πολλούς άλλους τρόπους. Το ουσιαστικό αποτέλεσμα από μια τέτοια διαδικασία είναι η εξαγωγή συμπερασμάτων σχετικά με το decision-making[37]. Με αυτόν τον τρόπο, λοιπόν, μπορεί να οδηγηθούν συστήματα σε νέες προσεγγίσεις και επεκτάσεις στο πλαίσιο κάλυψης ανθρωπίνων αναγκών, βασισμένων πάντα σε αποφάσεις προερχόμενες από δεδομένα του Information Extraction.

Μία αρκετά σημαντική παρατήρηση, που θα υποδηλώσει τη σημασία της παρούσας διπλωματικής, είναι η αλλαγή της εικόνας (format) των report. Υπάρχει, δηλαδή, μια νέα τάση δημιουργίας των report εδώ και ενάμιση χρόνο περίπου, εγκαταλείποντας την παλαιότερη αχανή εμφάνιση των περιγραφών ως ένα τεράστιο κείμενο – αυτό που έχει φανεί στα παραπάνω συμπεράσματα. Συγκεκριμένα, τα νέα reports έχουν υιοθετήσει μια πολύ καλή μορφή δόμησης των δεδομένων, παραπλήσια με αυτή που εκπονήσαμε στο πλαίσιο αυτής της διπλωματικής. Υπάρχει ένας μεγάλος πίνακας με κατηγοριοποιημένα δεδομένα ανάλογα με το είδος του προϊόντος, τον κωδικό του, την περιγραφή του, την κλάση του και άλλες πληροφορίες, που επιτρέπει με κατάλληλο φιλτράρισμα την επιλογή πεδίων για συγκεκριμενοποιημένη εμφάνιση των ενδιαφερόμενων πληροφοριών. Συνεπώς, η EIPAS εφαρμογή αποτελεί αδιαμφισβήτητα μία χρήσιμη ενέργεια, γεγονός που προκύπτει από την εμφανή ανάγκη κατηγοριοποίησης των δεδομένων των reports. Προφανώς, η χρησιμότητα αυτή εμφανίζεται στις παλαιότερες μορφές των report, καθιστώντας δυνατή την επεξεργασία πληροφοριών του παρελθόντος. Η νέα μορφή του report φαίνεται στην παρακάτω εικόνα.

[37] *decision-making*: Η λήψη αποφάσεων μπορεί να θεωρηθεί ως η γνωστική διαδικασία με αποτέλεσμα την επιλογή μιας πεποίθησης ή μιας πορείας δράσης μεταξύ διαφόρων εναλλακτικών δυνατοτήτων. Κάθε διαδικασία λήψης αποφάσεων παράγει μια τελική επιλογή, η οποία μπορεί ή όχι να ενεργήσει άμεσα. Η λήψη αποφάσεων είναι η μελέτη για τον προσδιορισμό και την επιλογή εναλλακτικών λύσεων με βάση τις αξίες και τις προτιμήσεις του αποφασίζοντα. Η λήψη αποφάσεων είναι μια από τις κεντρικές δραστηριότητες του management και αποτελεί ένα τεράστιο μέρος οποιασδήποτε διαδικασίας υλοποίησης. [40]

Εικόνα 5.1. Η νέα μορφή των reports

Συμπερασματικά, η εκπόνηση αυτής της διπλωματικής εργασίας αποτελεί μια καλή εισαγωγή στο Natural Language Processing και ιδίως στο Information Extraction, καθώς ταυτόχρονα και ένα ενδιάμεσο βήμα για την εξαγωγή αποτελεσμάτων από ένα μεγάλο όγκο ιατρικών και διατροφικών δεδομένων. Και λέγοντας ενδιάμεσο βήμα, εννοούμε το πρώτο στάδιο που θα ξεχωρίσει και θα κατηγοριοποιήσει τις πληροφοριές, των οποίων η εν συνεχεία ανάλυσή τους θα προσδόσει τα τελικά συμπεράσματα σχετικά με αυτά τα προϊόντα.

5.3. Μελλοντικές προκλήσεις

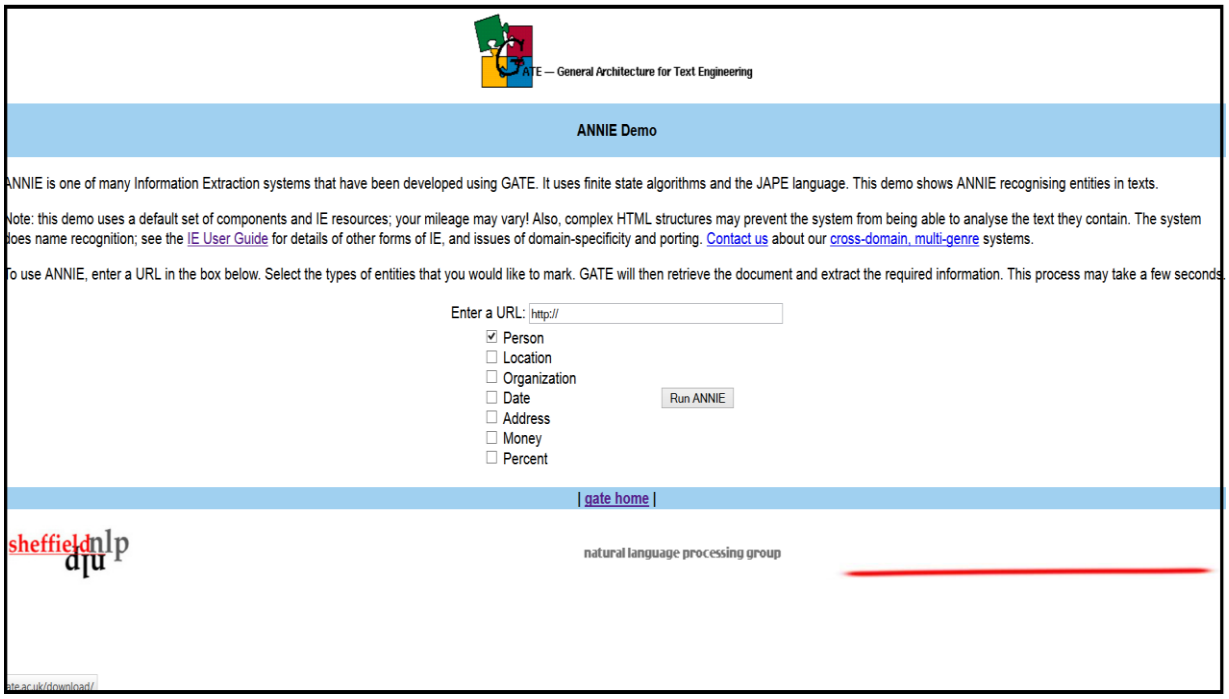
Με μια καλή ανασκόπηση των προηγούμενων προκύπτουν ιδέες που μπορούν να βασιστούν στα ήδη πεπραγμένα και να αποτελέσουν τη συνέχιση αυτής της διπλωματικής.

Έτσι, ίσως η πρώτη σκέψη είναι η δημιουργία της επόμενης βαθμίδας αυτής της εφαρμογής που δημιουργήθηκε. Μιλάμε, άρα, για τη δημιουργία ενός νέου συστήματος που θα λαμβάνει ως είσοδο την έξοδο του EIPAS και θα την επεξεργάζεται κατάλληλα. Βέβαια, αυτό το "κατάλληλα" μπορεί αν περιλαμβάνει διάφορες προεκτάσεις ανάλογα με τις εκάστοτε ανάγκες. Δηλαδή, το νέο σύστημα

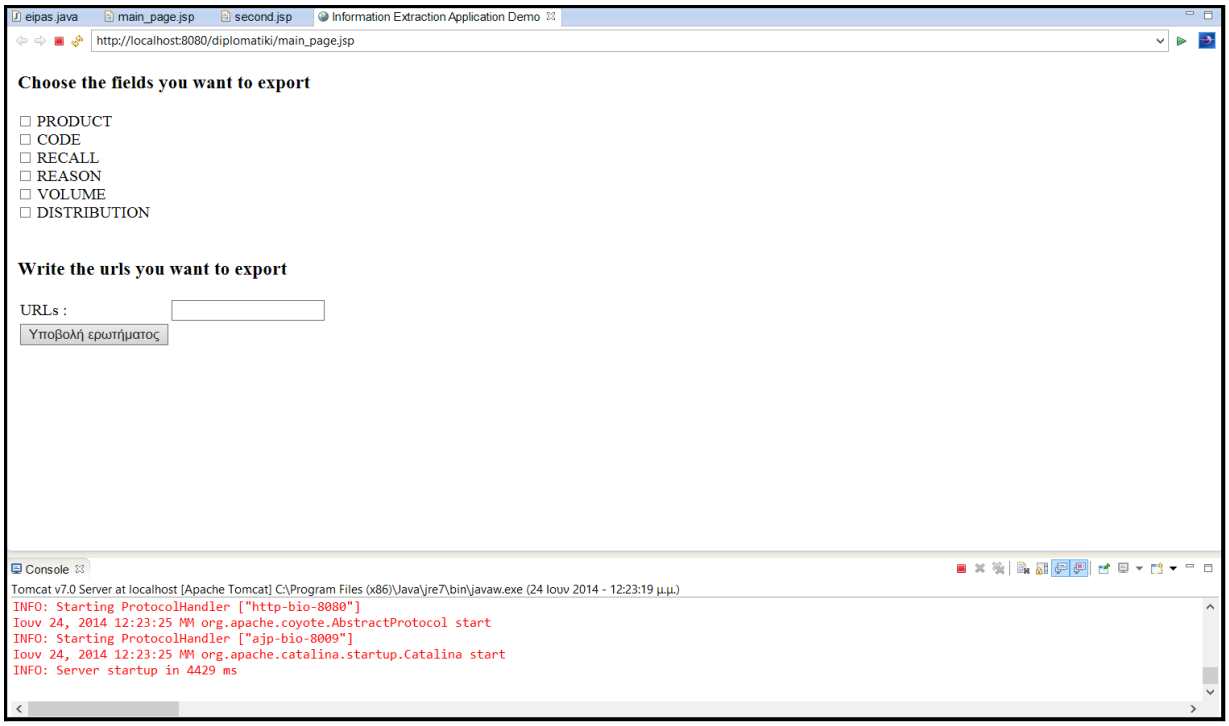
λογισμικού ενδέχεται να παράγει στατιστικά στοιχεία των προϊόντων, ή να προσδιορίζει προβλήματα που έχουν ήδη προκύψει, ή να συγκρίνει αποτελέσματα μεταξύ άλλων εισόδων, ή να παράγει αποφάσεις σχετικές με αυτά τα προϊόντα ή να έχει οποιαδήποτε άλλη χρήση στην κατεύθυνση κάλυψης συγκεκριμένων αναγκών.

Μία δεύτερη άποψη θα μπορούσε να υποστηρίζει μια νέας μορφής εξαγωγή πληροφοριών. Θα μπορούσε να υπάρχει ανάγκη για ένα ακόμα πιο εξειδικευμένο σύστημα λογισμικού πάνω στο Information Extraction, το οποίο να κάνει export δεδομένα που απαιτούν περισσότερο συνδυασμένες πληροφορίες. Μία τέτοια περίπτωση θα μπορούσε να λειτουργεί με άμεσο τρόπο στα reports (παλιάς ή ακόμα και νέας μορφής) ή και με έμμεσο, έχοντας ως είσοδο τα αποτελέσματα του EIPAS.

Τέλος, μία καλή ιδέα με περισσότερο διαδραστικό χαρακτήρα θα μπορούσε να αναφέρεται στη δημιουργία μιας σχετικής web-page σελίδας. Αναφέρομαι σε ένα περιβάλλον, όπου ο κάθε χρήστης θα μπορούσε να έχει κατά μία έννοια ενεργό ρόλο πάνω στην εξαγωγή των δεδομένων. Συγκεκριμένα, η σελίδα αυτή θα έχει ένα σύνολο από πεδία που μπορούν να επιλεγούν, καθώς και ένα χωρίο συμπλήρωσης των url’s από τα οποία θα γίνει το export. Έτσι, με χρήση ενός κουμπιού (button) θα γίνεται η μεταφορά σε μία νέα σελίδα που θα περιλαμβάνει το σύνολο των πινάκων με τα δεδομένα που έχουν ζητηθεί, ενώ από εκεί θα είναι δυνατή η επιστροφή στην αρχική σελίδα με χρήση ενός νέου κατάλληλου κουμπιού. Αυτή η υλοποίηση μοιάζει με τη σελίδα του ANNIE Demo με τη διαφορά των επιπλέον πεδίων αναφοράς, καθώς και την εμφάνιση των δεδομένων στη δομημένη μορφή πινάκων. Όσον αφορά την λειτουργικότητα της, σίγουρα δε θα προσδόσει κάτι επιπλέον από το ήδη δημιουργημένο EIPAS, στο πλαίσιο ενός ενδιάμεσου σταδίου για την περεταίρω επεξεργασία των exported δεδομένων. Ωστόσο, είναι μια πολύ καλή περίπτωση παρουσίασης της λειτουργικότητας του με ιδιαίτερα εύχρηστο τρόπο. Έτσι, η σελίδα αυτή δεν απαιτεί τη χρήση ενός Java virtual machine και με άμεσο τρόπο δίνεται η δυνατότητα εμφάνισης στο χρήστη, πληροφοριών που τον ενδιαφέρουν. Παρακάτω φαίνεται η σελίδα του ANNIE Demo, ενώ ακολουθεί μια ενδεικτική πρόχειρη εικόνα του Eclipse, που τρέχει σε ένα server για την ιδέα που μόλις αναφέρθηκε.



Εικόνα 5.2. Το site του ANNIE Demo



Εικόνα 5.3. Η μορφή ενός μελλοντικού EIPAS Demo

Ευρετήριο

A

C

D

E

F

G

I

J

L

M

N

O

P

R

S

T

V

W

Βιβλιογραφία

[1] D. Jurafsky and J. H.Martin, Speech and Language Processing, 2008.

[2] Univercity of Sheffield, Department of Computer Science, Developing Language Processing Components with GATE, Version 8 (a User Guide), Sheffield.

[3] "Natural language processing," [Online]. Available: http://en.wikipedia.org/wiki/Natural_language_processing.

[4] R. J. Mooney and Z. Zhou, "CS 388: Natural Language Processing," 2014. [Online]. Available: http://www.cs.utexas.edu/~mooney/cs388/.

[5] D. Jurafsky and C. Manning, "Natural Language Processing," Stanford, 2014. [Online]. Available: https://www.coursera.org/course/nlp.

[6] "The future of natural-language processing," 29 March 2001. [Online]. Available: http://www.itworld.com/UIR001229ontology.

[7] P. Koehn, MOSES-Statistical Machine Translation System, Edinburgh, 2014.

[8] "Blocks world," [Online]. Available: http://en.wikipedia.org/wiki/Blocks_world.

[9] "Chatterbot," [Online]. Available: http://en.wikipedia.org/wiki/Chatterbots.

[10] "Moore's law," [Online]. Available: http://en.wikipedia.org/wiki/Moore%27s_law.

[11] "Noam Chomsky," [Online]. Available: http://en.wikipedia.org/wiki/Noam_Chomsky.

[12] "Corpus linguistics," [Online]. Available: http://en.wikipedia.org/wiki/Corpus_linguistics.

[13] "Decision tree," [Online]. Available: http://en.wikipedia.org/wiki/Decision_tree.

[14] "Cache language model," [Online]. Available: http://en.wikipedia.org/wiki/Cache_language_model.

[15] "Probability distribution," [Online]. Available: http://en.wikipedia.org/wiki/Probability_distribution.

[16] "Speech recognition," [Online]. Available: http://en.wikipedia.org/wiki/Speech_recognition.

[17] "AI-complete," [Online]. Available: http://en.wikipedia.org/wiki/AI-complete.

[18] "Speech synthesis," [Online]. Available: http://en.wikipedia.org/wiki/Text-to-speech.

[19] "Stemming," [Online]. Available: http://en.wikipedia.org/wiki/Stemming#Examples.

[20] "Data mining," [Online]. Available: en.wikipedia.org/wiki/Data_mining.

[21] "Open-source software," [Online]. Available: http://en.wikipedia.org/wiki/Open-source_software.

[22] "Software framework," [Online]. Available: http://en.wikipedia.org/wiki/Software_framework.

[23] "Abstraction (computer science)," [Online]. Available: http://en.wikipedia.org/wiki/Abstraction_%28computer_science%29.

[24] "Integrated development environment," [Online]. Available: http://en.wikipedia.org/wiki/Integrated_development_environment.

[25] "Ontology (information science)," [Online]. Available: http://en.wikipedia.org/wiki/Ontology_%28information_science%29.

[26] "Machine learning," [Online]. Available: http://en.wikipedia.org/wiki/Machine_learning.

[27] "Parsing," [Online]. Available: http://en.wikipedia.org/wiki/Parsing.

[28] "Phase-type distribution," [Online]. Available: http://en.wikipedia.org/wiki/Phase-type_distribution.

[29] "Morphology," [Online]. Available: http://en.wikipedia.org/wiki/Morphology.

[30] "Tag (metadata)," [Online]. Available: http://en.wikipedia.org/wiki/Tag_%28metadata%29.

[31] "Information retrieval," [Online]. Available: http://en.wikipedia.org/wiki/Information_retrieval.

[32] "Graphical user interface," [Online]. Available: http://en.wikipedia.org/wiki/Graphical_user_interface.

[33] "Application programming interface," [Online]. Available: http://en.wikipedia.org/wiki/Application_programming_interface.

[34] "Serialization," [Online]. Available: http://en.wikipedia.org/wiki/Serialization.

[35] "Pipeline (computing)," [Online]. Available: http://en.wikipedia.org/wiki/Pipeline_%28computing%29.

[36] "n-gram," [Online]. Available: http://en.wikipedia.org/wiki/N-gram.

[37] "Finite-state machine," [Online]. Available: http://en.wikipedia.org/wiki/Finite-state_machine.

[38] "Thread safety," [Online]. Available: http://en.wikipedia.org/wiki/Thread_safety.

[39] "Pattern matching," [Online]. Available: http://en.wikipedia.org/wiki/Pattern_matching.

[40] "Decision-making," [Online]. Available: http://en.wikipedia.org/wiki/Decision_making.

[41] "Importance of Data Mining Services in Business," [Online]. Available: http://www.articlesbase.com/outsourcing-articles/importance-of-data-mining-services-in-business-1047892.html.

[42] E. H. C. Program, "Data Extraction," [Online]. Available: http://www.slideshare.net/AHRQEHCProgram/data-extraction.