ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΣΧΟΛΗ ΟΙΚΟΝΟΜΙΑΣ ΚΑΙ ΔΙΟΙΚΗΣΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ

Διπλωματική Εργασία

Σχεδιασμός και Υλοποίηση Διαδικτυακού Εργαλείου Εξόρυξης και Ταυτοποίησης Αδειών Προγραμμάτων Ελεύθερου Λογισμικού Ανοιχτού Κώδικα

Μπένος Γεώργιος
ΑΜ: 2022201802021

Επιβλέπων:
Νικόλαος Τσελίκας
Αναπληρωτής Καθηγητής

Τρίπολη, Δεκέμβριος 2020

Περίληψη

Τα τελευταία χρόνια έχει υπάρξει μια έκρηξη στη χρήση Ελεύθερου και Ανοιχτού-Κώδικα Λογισμικού (Free Open-Source Software - FOSS). Ενώ αυτή η εξέλιξη είναι κατά βάσιν καταπληκτικά νέα για την τεχνολογία και την αξιοποίησή της σε ποικίλους τομείς, έρχεται με τα δικά της προβλήματα και προκλήσεις.

Σε αυτή την εργασία θα οριστούν και αναλυθούν μερικές από αυτές τις προκλήσεις, όπως η δυσκολία εύρεσης συμβατότητας, η αύξηση του αριθμού των αδειών και οι «Άδειες από κηρομπογιά», άδειες κακής ποιότητας, φτιαγμένες χωρίς τη συμβολή ειδικών. Επίσης, θα αναφερθούν κοινοί τρόποι και σωστές πρακτικές σχετικά με την εύρεση, αναπαράσταση και αποθήκευση αδειών σε μεγαλύτερα projects λογισμικού.
Τέλος, θα προταθεί μια καινούρια λύση σε επίπεδο λογισμικού, η οποία χρησιμοποιεί λεκτική και σημασιολογική ανάλυση πάνω στα κείμενα των αδειών για την ταυτοποίηση με τα αντίστοιχα μοντέλα τους και πίνακες συμβατότητας, βασισμένους σε γράφους συμβατότητας για την πιστοποίηση συμβατών αδειών λογισμικού ή πρόταση καινούριων αδειών σε περίπτωση μη συμβατότητας.

Λέξεις - Κλειδιά

Άδεια λογισμικού, ανοιχτή/ελεύθερη άδεια λογισμικού, συμβατότητα αδειών λογισμικού.

Abstract

These last few years, the popularity of Free Open-Source Software (FOSS) has exploded dramatically. While it's generally Amazing news for technology and its adoption in many sectors, there are multiple problems and challenges associated with it.

This thesis will tackle some of these challenges like the difficulty in compatibility between licenses, compounded by the steady increase in the sheer number of licenses available, as well as "crayon licenses", bad quality licenses, drafted without experts' input. Also, some common ways and best practices regarding finding, representing and storing licenses in bigger software projects will be mentioned.

Finally, a new software solution will be proposed that leverages lexical and semantic analysis on the license text in order to identify the licenses with their respective models and compatibility matrices, based on compatibility graphs, in order to verify compatible software licenses or propose new ones in case of non-compatibility.

Keywords

Software License, Free Open-Source Software license, FOSS, FOSSology, Software license compatibility

Copyright

Περιεχόμενα

Πρόλογος

Το λογισμικό ανοιχτού κώδικα έχει διεισδύσει σε πάρα πολλούς τομείς της τεχνολογίας, αν όχι όλους, τα τελευταία χρόνια και τα πλεονεκτήματα που παρέχει είναι αδιαμφισβήτητα [1]. Πλέον το FOSS είναι αναπόσπαστο κομμάτι της καθημερινότητας μας, τόσο από την πλευρά των ιδιωτών χρηστών, όσο και από την πλευρά μεγάλων εταιρειών και οργανισμών.

Φαινομενικά, Η ιδέα του ανοιχτού λογισμικού φαίνεται να έρχεται σε αντιπαράθεση με το βασικό ανθρώπινο δικαίωμα της πνευματικής ιδιοκτησίας. Πως γίνεται ένα κομμάτι λογισμικού να ανήκει στον δημιουργό του και ταυτόχρονα να είναι ελεύθερο για χρήση, μετατροπή, αντιγραφή και διαμοιρασμό σε όλους; Υπάρχουν 2 τρόποι επίλυσης αυτού του θέματος. Ο πρώτος είναι με την αποποίηση οποιασδήποτε μορφής ιδιοκτησίας από τον δημιουργό και στην μεταβίβαση του στο κοινό κτήμα (public domain), κάτι που ίσως να μην είναι ιδανικό από την πλευρά του δημιουργού για πολλούς και διάφορους λόγους. Ο δεύτερος τρόπος είναι η παραχώρηση ενός υποσυνόλου δικαιωμάτων της συνολικής ιδιοκτησίας, όπως δικαιώματα χρήσης, αντιγραφής, διακίνησης και επικαρπίας κατόπιν συμφωνίας του δημιουργού με οποιονδήποτε ενδιαφερόμενο.

Μια τέτοια συμφωνία προσφέρει δικλίδες ασφαλείας και στις δυο πλευρές, αλλά η πρακτικότητα της είναι αντιστρόφως ανάλογη με τον αριθμό των ενδιαφερόμενων, καθώς ο δημιουργός θα πρέπει να διαπραγματευτεί τους όρους της συμφωνίας με τον κάθε ενδιαφερόμενο ξεχωριστά. Εναλλακτικά, ο δημιουργός μπορεί να γνωστοποιήσει μια λίστα με όρους και δικαιώματα την οποία έχει προεγκρίνει, και οι ενδιαφερόμενοι (συνήθως εμμέσως με την χρήση του λογισμικού) την αποδέχονται συνάπτοντας την συμφωνία, ή όχι.

Σε αυτή την βασική ιδέα βασίζεται ή έννοια της «Άδειας λογισμικού» και κατά βάσιν έχει την ισχύ συμβολαίου [2] μεταξύ του δημιουργού/ιδιοκτήτη και του ενδιαφερόμενου. Όπως και σε οποιονδήποτε άλλο τομέα, υπάρχει τεράστια ποικιλία τέτοιων εγγράφων για τεράστιο αριθμό χρήσεων, είτε συντεταγμένα a priori ώστε να καλύπτουν την πλειοψηφία των περιπτώσεων, είτε συντεταγμένα επί τόπου έχοντας υπ' όψιν τις ιδιαιτερότητες και τα λεπτά σημεία της κάθε περίπτωσης. Στην περίπτωση των αδειών FOSS, η δεύτερη επιλογή είναι εξαιρετικά επιβλαβής όπως θα αναλυθεί παρακάτω.

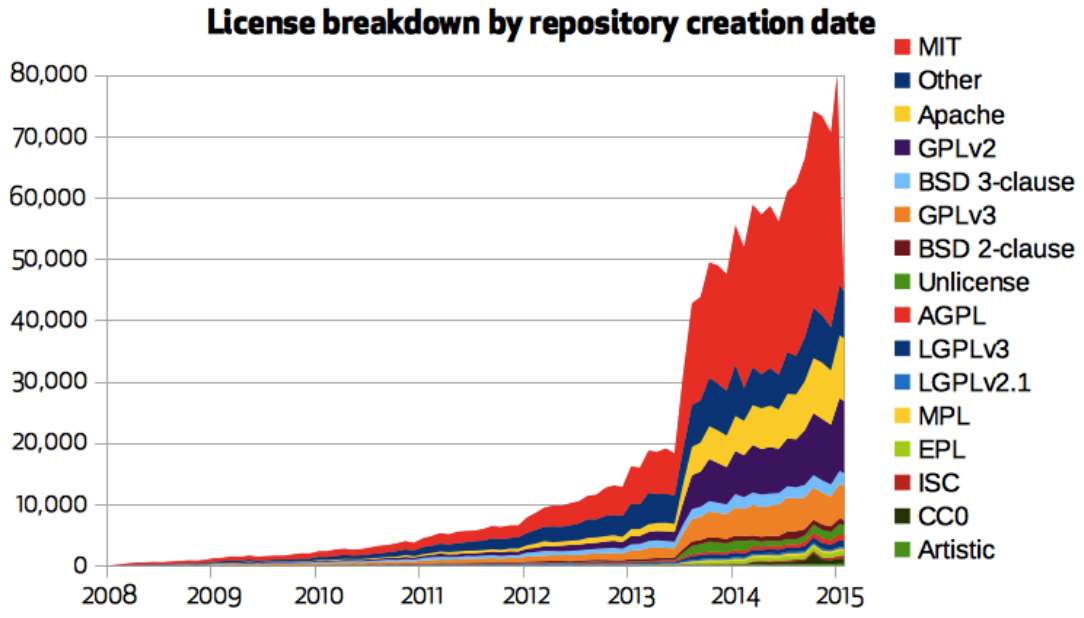
Προβλήματα & Προκλήσεις

Ένα από τα κυριότερα προβλήματα που παρατηρείται τα τελευταία χρόνια στο FOSS οικοσύστημα είναι η ταχεία εξάπλωση (proliferation) των αδειών λογισμικού [3]. Φαινομενικά η μεγαλύτερη ποικιλία αδειών μπορεί να δείχνει ως κάτι πάρα πολύ ευεργετικό για τον τομέα, παρέχοντας μεγάλη ελευθερία στους δημιουργούς και «μια άδεια για κάθε περίπτωση», και πράγματι ένας τέτοιος ισχυρισμός δε είναι πολύ μακριά από την αλήθεια, αλλά τα (συχνά όχι προφανή) μειονεκτήματα ενός τεράστιου αριθμού αδειών εκλείπουν τα πλεονεκτήματα του.

Μερικά από αυτά τα μειονεκτήματα είναι τα εξής:

- Αυξημένος χρόνος και προσπάθεια εύρεσης της κατάλληλης άδειας από τον δημιουργό.
- Εκθετικά αυξανόμενη πολυπλοκότητα στην εύρεση συμβατών αδειών σε περίπτωση που χρησιμοποιούνται πάνω από ένα κομμάτια κώδικα κάτω από διαφορετικές άδειες.
- Μεγάλος αριθμός από «άδειες κηρομπογιάς» (crayon licenses) που δεν καλύπτουν σωστά από νομικής άποψης τους δημιουργούς (ή τους χρήστες) του λογισμικού.
- Αυξημένα νομικά έξοδα και προσπάθεια σε περίπτωση προσφυγής στη δικαιοσύνη, καθώς κάθε άδεια/συμβόλαιο πρέπει να εξεταστεί ξεχωριστά.
- Αύξηση της πιθανότητας ασυμβατότητας αδειών ανάλογη με τον αριθμό αδειών που χρησιμοποιούνται σε ένα ολοκληρωμένο λογισμικό.

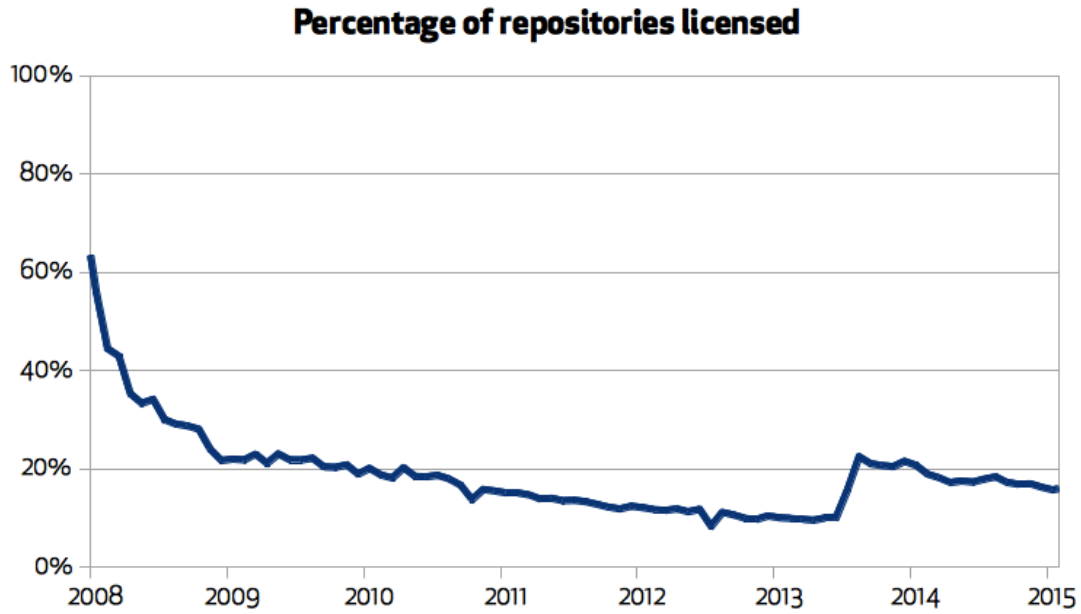
Ευτυχώς, τα τελευταία χρόνια έχει υπάρξει μια προσπάθεια μετρίασης, αν όχι αντιμετώπισης αυτού του προβλήματος από επιφανείς οργανισμούς του τομέα, όπως οι OSI, GNU, Intel, Google [4] και GitHub [5] η οποία φαίνεται να επιφέρει καρπούς.



Εικόνα 1: Το ποσοστό αδειών στα projects του GitHub τα έτη 2008- 2015

Στον αντίποδα, ένα επίσης σημαντικό πρόβλημα στον τομέα είναι η μη χρήση άδειας σε δημοσίως προσβάσιμο λογισμικό. Όπως αναφέρθηκε παραπάνω η πνευματική ιδιοκτησία είναι παγκοσμίως αναγνωρισμένο και αναφαίρετο ανθρώπινο δικαίωμα και κανένα κομμάτι κώδικα δε μπορεί να χρησιμοποιηθεί πουθενά χωρίς την ρητή συγκατάθεση του δημιουργού του, καθιστώντας το πρακτικά άχρηστο από νομικής άποψης.

Δυστυχώς, τα τελευταία χρόνια όλο και περισσότερα projects στην πλατφόρμα του GitHub παρέχονται χωρίς άδεια, παρ' όλες τις προσπάθειες που καταβάλλουν μεγάλοι και μικρότεροι οργανισμοί του τομέα [5] [4]. Συγκεκριμένα, το 2013 το GitHub έδωσε στο κοινό την πλατφόρμα choosealicense.com με σκοπό την μετατροπή της σχετικά περίπλοκης διαδικασίας επιλογής άδειας λογισμικού σε ένα απλό και φιλικό ερωτηματολόγιο και ενώ η αρχική ανταπόκριση ήταν θετικότατη, η πτωτική τάση συνεχίζεται σε βάθος χρόνου.



Εικόνα 2: Το ποσοστό projects στο GitHub που συμπεριλαμβάνουν άδεια

Μπορεί τα αίτια αυτού του προβλήματος να μην είναι παντελώς ξεκάθαρα, αλλά από τα δεδομένα της πρωτοβουλίας του github, η ευκολία επιλογής άδειας φαίνεται να είναι ένας σημαντικός παράγοντας.

Τέλος, υπάρχει το θέμα των «αδειών κηρομπογιάς». Ο όρος χρησιμοποιήθηκε πρώτη φορά από τον Bruce Perens, συνιδρυτή του OSI, σε αναφορά ενός sketch των Monty Python κατά το οποίο παρουσιάζεται μια άδεια σκύλου, ώς άδεια γάτας, έχοντας αντικαταστήσει τη λέξη «σκύλος» με τη λέξη «γάτα» με κηρομπογιά [6]. Παραθέτοντας το αρχικό email του Perens, η άποψη του για το θέμα είναι προφανής.

```
I've been calling these "crayon licenses", taking a line from an old
Monty Python sketch about a dog license with the word "dog" crossed
out and "cat" written in, in crayon.

Crayon, in this case, is a metaphor for the poor legal qualification
of the authors. Crayon licenses show a lack of understanding of
copyright law, license structure, and most important: what would
happen if the license were to be interpreted in court. We had an
excuse for writing such things in the early days of Open Source when
no lawyer would help us. /We no longer have that excuse./
```

> Crayon licenses harm Open Source developers because they don't do what the developer expects. My most poignant experience with one was working on the appeal of /Jacobsen v. Katzer. /Bob Jacobsen, an innocent Open Source developer, essentially lost his case in the lower court due to the poor drafting of the Artistic License 1.0, one of the initial Open Source licenses, when the judge found it to be tantamount to the public domain. This loss would also have been very damaging to Open Source in general, had it been allowed to stand. Bob suffered /very/ significant damage from this case. We are very fortunate that he persevered, and that we were able to overturn the decision on appeal.
>
> OSI should no longer approve crayon licenses, due to the potential they have to damage our own community.
>
> Thanks
>
> Bruce Perens

Όπως αποδεικνύεται, αυτές οι άδειες εγκυμονούν τεράστιους κινδύνους όχι μόνο για τους ίδιους τους προγραμματιστές που τις επιλέγουν, αλλά και για ολόκληρη την FOSS κοινότητα, ειδικά σε χώρες που ισχύει το «δίκαιό του εκδικασμένου».

Πρακτικά, υπάρχουν δύο τρόποι ώστε κάποιος να δημιουργήσει ένα crayon license, μπορεί να γράψει κάτι τέτοιο εξ αρχής (πχ beerware [7], ή WTFPL)[1]. Εναλλακτικά, και ίσως χειρότερα, κάποιος μπορεί να ξεκινήσει με μια ήδη υπάρχουσα άδεια και να προσθέσει, αφαιρέσει ή αλλάξει όρους, μερικές φορές χωρίς καν να αλλάξει το όνομα ή την έκδοση της άδειας.

[1] Οι Beerware και WTFPL δεν είναι ακριβώς crayon licenses, αφού είναι τουλάχιστον αναγνωρισμένες από το FSF (αν και όχι από το OSI) αλλά η χρήση τους αποθαρρύνεται για τους ίδιους ακριβώς λόγους.

Άδειες και συμβατότητα

Συμβατότητα αδειών

Στον πυρήνα αυτής της εργασία βρίσκεται η έννοια της συμβατότητας (ή ασυμβατότητας) αδειών, χωρίς να έχει δοθεί μέχρι τώρα κάποιος συμπαγής ορισμός.

Μια άδεια Α είναι **συμβατή** με μια άδεια Β αν και μόνο αν πληρούνται δύο προϋποθέσεις. [8]

- Οτιδήποτε επιτρέπεται από την άδεια Α επιτρέπεται και από την άδεια Β.
- Οποιοδήποτε συμμορφώνεται με την άδεια Α, μπορεί να συμμορφωθεί και με την άδεια Β χωρίς να παραβιάσει κάποιον όρο της Α.

Τότε και μόνο τότε ένα κομμάτι λογισμικού που περιέχει κώδικα κάτω από την άδεια Α και την άδεια Β (ή σε κάποιες περιπτώσεις μόνο την άδεια Α) μπορεί να δημοσιευθεί υπό την άδεια Β.

Αξίζει να σημειωθεί ότι η σχέση της συμβατότητας δεν είναι αντιστρέψιμη. Αν μια άδεια Α είναι συμβατή με μια άδεια Β, τότε οι πιθανότητες είναι ότι η Β δεν θα είναι συμβατή με την Α. Σε περίπτωση όπου η Β είναι και αυτή συμβατή με την Α, τότε οι 2 άδειες θεωρούνται **ισοδύναμες** και πιθανότατα προϊόν πολλαπλασιασμού αδειών.

Μια τέτοια περίπτωση είναι η περίπτωση των αδειών MIT και ISC, όπου η ISC αποτελεί μια απόπειρα εκσυγχρονισμού της MIT αφαιρώντας πράγματα που πλέον θεωρούνται αυτονόητα βάσει της συνθήκης της Βέρνης περί προστασίας καλλιτεχνικών και λογοτεχνικών έργων. [9]

Η σχέση της συμβατότητας όμως είναι μεταβατική. Αν μια άδεια Α είναι συμβατή με μια Β και ή Β είναι συμβατή με μια άδεια Γ. τότε η άδεια Α είναι συμβατή με την Γ, αφού πρέπει να υπάρξει κάποιος λόγος παραβίασης αυτής της συμβατότητας, κάτι που δεν ισχύει ανάμεσα στην Α και την Β , ούτε από την Β και την Γ.

Ο συνδυασμός των παραπάνω ιδιοτήτων της συμβατότητας αδειών έχει κάποιες εξαιρετικά ενδιαφέρουσες συνέπιες στην μελέτη τους. Συγκεκριμένα, οι άδειες διατάσσονται σε αλυσίδες συμβατότητας, μεταξύ αδειών και αυτές οι αλυσίδες αλληλοεπιδρούν μεταξύ τους, δημιουργώντας «δέντρα συμβατότητας».

Οι άδειες που βρίσκονται στην αρχή αυτών των αλυσίδων ονομάζονται επιτρεπτικές (permissive) και επιτρέπουν την έκδοση λογισμικού κάτω από μεγάλο αριθμό αδειών. Οι άδεις που βρίσκονται στη μέση ονομάζονται (weak-copyleft) και επιτρέπουν τον διαμοιρασμό λογισμικού μόνο κάτω από την ίδια άδεια αν έχει τροποποιηθεί ο αρχικός κώδικας, ή συμβατές άδειες αν δεν έχει τροποποιηθεί ο αρχικός κώδικας.

Τέλος, οι άδειες στο τέλος αλυσίδων ονομάζονται περιοριστικές (restrictive or strong-copyleft) και επιτρέπουν την δημοσιοποίηση λογισμικού μόνο κάτω από την ίδια άδεια.

Πίνακας 1: Κατηγορίες αδειών

Σύμβολο	Τύπος άδειας	Δημοσιοποίηση Λογισμικού	Παραδείγματα
	Public Domain	Κάτω από οτιδήποτε Χωρίς κανένα περιορισμό	CC0
	Permissive	Κάτω από πολλές άδειες σε όλες τις κατηγορίες	MIT BSD Apache v2.0
	Weak-Copyleft	Κάτω από την ίδια άδεια η συμβατές περιοριστικές	LGPL MPL
	Strong-Copyleft Restrictive	Μόνο κάτω από την ίδια άδεια	GPL OSL Sleepycat

Αυτές οι κατηγορίες όμως είναι εν τέλει ενδεικτικές και γραμμή που τις ξεχωρίζει είναι σε πολλές περιπτώσεις θολή. Μερικοί όροι μπορούν ερμηνευτούν διαφορετικά κάτω από διαφορετικές συνθήκες ή μπορεί να υπάρχει διαφορετική αντιμετώπιση ανάλογα με τον αν η σύνδεση υπάρχοντος και νέου κώδικα είναι στατική η δυναμική, γεγονός που κατά καιρούς έχει οδηγήσει ακόμα και σε διαφωνίες μεταξύ οργανισμών όπως FSF και GNU που θεωρούνται αυθεντίες του τομέα. [8] Εν τέλει η κάθε περίπτωση συμβατότητας μεταξύ αδειών είναι μοναδική και πρέπει να εξετάζεται ξεχωριστά από ειδικούς (ακόμα είναι λόγος κατά των Αδειών κηρομπογιάς»).

Π.χ. Οι Apache v2 και η GPLv2 είναι φαινομενικά συμβατές και ανήκουν σε συμβατές κατηγορίες (permissive προς strong copyleft) αλλά ύστερα από προσεκτική εξέταση, υπάρχουν διαφορές στον τρόπο διαχείρισης και τερματισμού πατεντών. Στην περίπτωση της Apachev2, αν ον ενάγων καταφύγει δικαστικώς λόγω καταπάτησης πατέντας σχετικά με το έργο, τότε όλες οι πατέντες που έχει κάτω από Apachev2 άδεια για το ίδιο έργο, τερματίζονται αυτόματα. Εναλλακτικά η GPLv2 ενώ έχει παρόμοιο σκεπτικό είναι πολύ πιο ελαστική και γενική στο θέμα. [8]

Πίνακας 2: Ο όρος σχετικά με τις δικαστικές διαμάχες περί πατέντας ανά άδεια

Apachev2	GPLv2
If You institute patent litigation against any entity (including a crossclaim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that	If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you

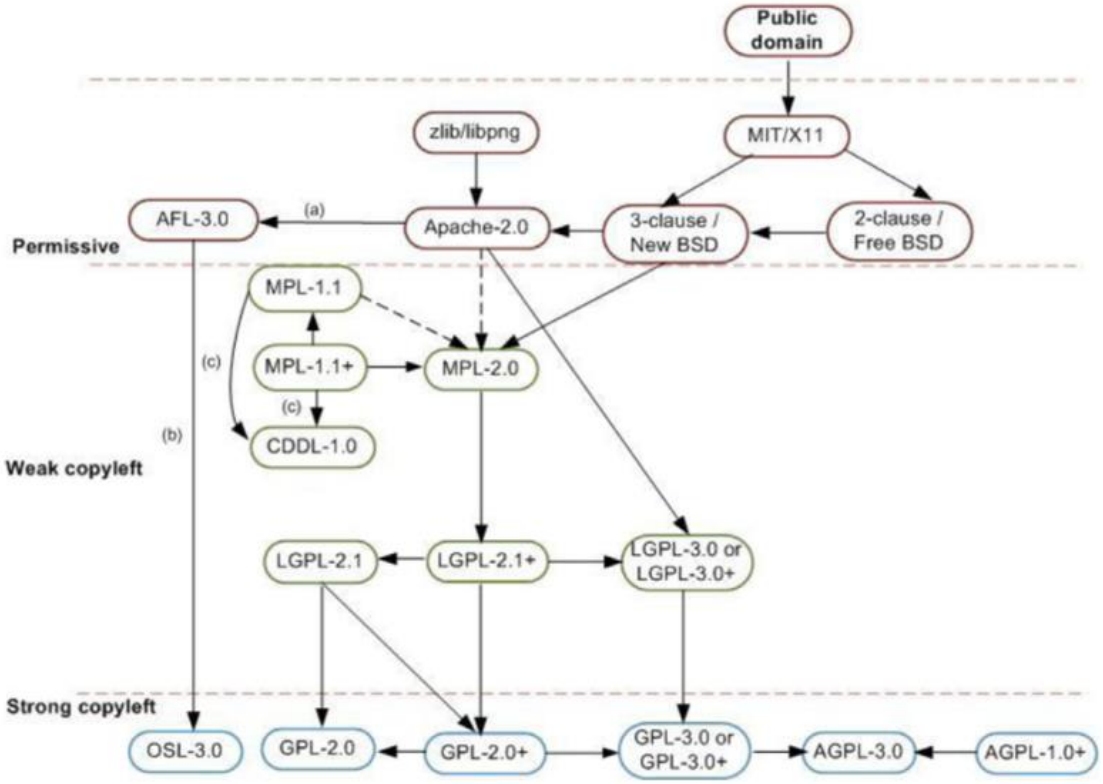
Work shall terminate as of the date such litigation is filed.	from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all.

Τέτοια προβλήματα ασυμβατότητας (και όχι μόνο) αδειών συναντώνται αρκετά συχνά στον πραγματικό κόσμο πολλές φορές με επιπτώσεις για την FOSS κοινότητα στο σύνολο της, όπως στην AMV vs Qybits [10] κατά την οποία η AMV, κατασκευαστής router χρησιμοποιούσε τον GPLv2 licensed πυρήνα linux ως βάση για το proprietary firmware των router τους και το λογισμικό της Qybits τροποποιούσε κομμάτια αυτού του πυρήνα σε προσπάθεια αποκλεισμού κακόβουλων σελίδων. Η AMV υπέβαλλε 2 αγωγές για παραβίαση copyright την περίοδο 2010-2011, αλλά ύστερα από την παρέμβαση ειδικών, το περιφερειακό δικαστήριο του Βερολίνου έκρινε ότι η AMV δεν έχει κανένα απολύτως copyright πάνω στον linux πυρήνα ενώ εξακολουθεί να δεσμεύεται από την GPLv2 άδεια του όπως αναφέρει ο επίμαχος όρος παραπάνω.

Τέλος, μια εναλλακτική προσέγγιση στο θέμα συμβατότητας αδειών είναι η χρήση διπλής άδειας και η επιλογή από τον ενδιαφερόμενο όποιας άδειας προτιμάει. Κάτι τέτοιο μπορεί σε πρώτο επίπεδο να ανοίγει αρκετά τις επιλογές σε πιθανώς συμβατές άδειες, αλλά δημιουργεί διαφορετικά προβλήματα σε βάθος χρόνου. Το πιο παράδοξο από αυτά, είναι η πιθανότητα δύο διαφορετικών διακλαδώσεων (fork) ενός project, τα οποία ύστερα από διάφορες συνεισφορές, έχουν φτάσει σε ασύμβατες μεταξύ τους άδειες και κανείς, ούτε καν ο αρχικός δημιουργός, δε μπορεί να τα ενώσει ξανά. [8] Λόγω της υπερβολικά αυξημένης πολυπλοκότητας (τόσο σε λεκτικό, όσο και σε επίπεδο συμβατότητας) με σχετικά μικρό όφελος, το ενδεχόμενο διπλής αδειοδότησης θα αγνοηθεί στην συνέχεια.

Λαμβάνοντας υπ’ όψιν τα παραπάνω, η συμβατότητα αδειών μπορεί να μοντελοποιηθεί ως ένα γράφημα, στο οποίο οι άδειες/κόμβοι ενώνονται με κατευθυνόμενες ακμές που υποδεικνύουν συμβατότητα και σε ειδικές περιπτώσεις, οι διακεκομμένες ακμές δείχνουν συμβατότητα που σταματάει να ισχύει στην άδεια-δέκτη. Προφανώς ένας τέτοιος γράφος είναι προϊόν ξεχωριστής μελέτης από ειδικούς [11], αλλά αποτελεί ισχυρότατο εργαλείο στην προσπάθεια αντιμετώπισης του προβλήματος εύρεσης συμβατότητας αδειών.

Όσον αφορά την εύρεση συμβατότητας μεταξύ υποστηριζόμενων αδειών, αρκεί να υπάρχει μονοπάτι από την άδεια Α στην άδεια Β, το οποίο αν περιέχει διακεκομμένη ακμή, βρίσκεται στο τέλος του μόνο. Η εισαγωγή νέων αδειών δεν είναι τόσο εύκολη υπόθεση, καθώς πρέπει να ελεγχθεί η συμβατότητα της νέας άδειας με όλες τις άδειες του γράφου καθώς και η συμβατότητα όλων των αδειών με τη καινούρια ώστε να τοποθετηθεί σωστά στο γράφημα.



Εικόνα 3¨: Γράφημα συμβατότητας αδειών

Ταυτοποίηση & εξαγωγή αδειών

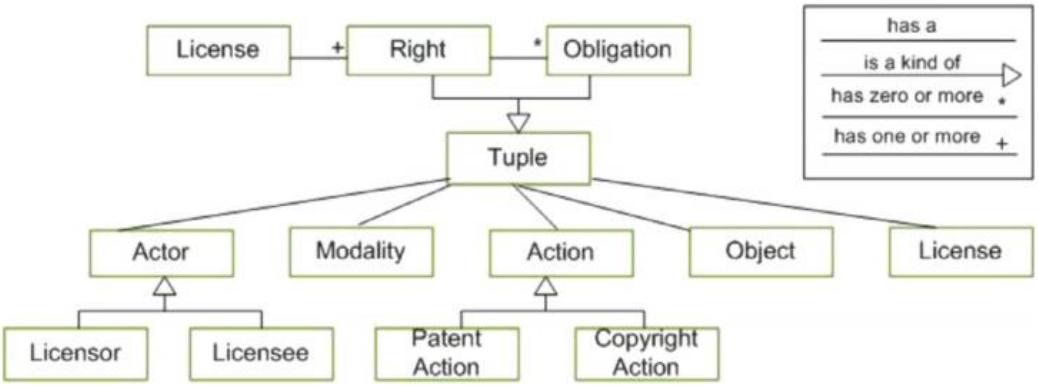
Εκτός από το κομμάτι της συμβατότητας αδειών που αναφέρθηκε παραπάνω, ένα επίσης σημαντικό ζήτημα είναι ο εντοπισμός, η μοντελοποίηση και η πιθανή ταυτοποίηση μιας υπάρχουσας άδειας σε ένα υπαρκτό κομμάτι λογισμικού.

Η άδεια, έχοντας κατά βάσιν την ισχύ συμφωνητικού, πρέπει να βρίσκεται σε εμφανές σημείο με απρόσκοπτη πρόσβαση στους χρήστες. Κάτι τέτοιο όμως είναι εξαιρετικά γενικό, καθώς το «εμφανές σημείο» είναι στην διακριτική ευχέρεια του ιδιοκτήτη του λογισμικού και ενώ υπάρχουν κάποιες καλές πρακτικές, δεν υπάρχει κάποιο πρότυπο. Μεγάλες πλατφόρμες φιλοξενίας λογισμικού όπως οι GitHub και google code συνιστούν την χρήση ενός αρχείου LICENSE(.md) ή COPYING(.md) στον βασικό φάκελο της εφαρμογής που περιέχει το πλήρες κείμενο της άδειας από μια επίσημη πηγή. Προφανώς όμως η άδεια μπορεί να βρίσκεται σε πολλά σημεία όπως:

- Σε άλλα αρχεία κειμένου, όπως README, ABOUT, INFO παρέα με επιπλέον πληροφορίες σχετικά με το λογισμικό.
- Σε σχόλια μέσα στα αρχεία του πηγαίου κώδικα.
- Ενσωματωμένη μέσα στα εκτελέσιμα αρχεία.
- Σε δική της σελίδα στην πλατφόρμα φιλοξενίας του λογισμικού, εφ‘ όσον υποστηρίζεται .
- Σε οποιονδήποτε συνδυασμό από τα παραπάνω.

Προχωρώντας στην μοντελοποίηση μιας άδεια, η συνηθέστερη προσέγγιση είναι η αντιγραφή του πλήρους και απαράλλαχτου κειμένου της άδειας, από μια επίσημη πηγή. Αυτή η προσέγγιση, σε συνδυασμό με την προηγούμενη, συνιστάται από τις μεγαλύτερες πλατφόρμες φιλοξενίας λογισμικού. Εναλλακτικές προσεγγίσεις περιλαμβάνουν:

- Την αναφορά της άδειας μόνο με το όνομα της, ή ακόμα και έναν σύνδεσμο προς εξωτερική σελίδα που περιλαμβάνει το πλήρες κείμενο της.
- Την χρήση μη λεκτικών μοντέλων, σε περίπτωση που η άδεια τείνει να υποστηρίζεται από συγκεκριμένα προγράμματα και όχι από ανθρώπους χωρίς την χρήση αυτών. [12]



Εικόνα 4: Η άδεια ως γράφημα κανόνων γραμματικής κατά Alspaugh

Τέλος, η ταυτοποίηση της άδειας περιλαμβάνει την σύγκριση του μοντέλου που χρησιμοποιήθηκε στην εκάστοτε άδεια με το αντίστοιχο μοντέλο μιας πιστοποιημένης από αξιόπιστη πηγή άδειας. Σε έναν ιδανικό κόσμο, η ομοιότητα θα ήταν 100% αν η άδεια ήταν ίδια με την πιστοποιημένη, και ένα οποιοδήποτε άλλο ποσοστό αλλιώς. Στην πράξη, όμως υπάρχει μια πληθώρα πιθανών λόγων που μπορούν να μειώσουν την ομοιότητα, χωρίς να επηρεάζουν το νόημα της άδειας, ιδιαίτερα αν η άδεια βρίσκεται σε λεκτικό μοντέλο όπως είναι μακράν πιο πιθανό. Μερικοί από αυτούς τους λόγους είναι, ορθογραφικά λάθη, ειδικές ευχαριστίες, συμπερίληψη σημειώσεων από τον ιδιοκτήτη λογισμικού κλπ. Κατά τον σχεδιασμό και την υλοποίηση θα προταθούν διάφοροι τρόποι αντιμετώπισης η έστω μετρίασης του ζητήματος.

Για λόγους απλότητας αλλά και κανονικότητας, καθώς όσο υποστηρίζονται οι χειρότερες πρακτικές τόσο θα αναπαράγονται και θα εξαπλώνονται, στον σχεδιασμό της προτεινόμενης λύσης θα ληφθούν υπόψιν μόνο οι καλύτερες και πιο κοινές πρακτικές στην διαχείριση της άδειας. Συγκεκριμένα, η άδεια θα βρίσκεται στον βασικό φάκελο του λογισμικού, σε αρχείο license(.md) ή copying(.md), οι βιβλιοθήκες θα έχουν τους δικούς τους υποφακέλους και μεταχειρίζονται σαν μικρότερα projects (Επίσης έχοντας μόνο μια άδεια στην μορφή αρχείου κειμένου με τίτλο license(.md) ή copying(.md)), και οι διπλές άδειες θα αγνοούνται.

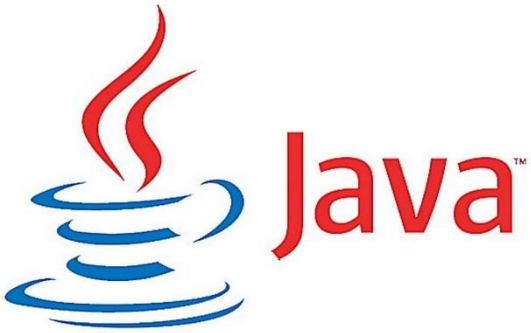
Τεχνολογίες που χρησιμοποιήθηκαν

Java

Γενικά

Η Java είναι μια από τις διαδεδομένες γλώσσες προγραμματισμού παγκοσμίως αυτή τη στιγμή. Είναι αντικειμενοστρεφής, πολυνηματική, και ανεξάρτητη υλοποίησης και υλικού, παράγοντες που ευθύνονται για την συντριπτική δημοτικότητα της.

Το σλόγκαν πίσω από την ανάπτυξη της java από την εταιρεία sun microsystems ήταν «Γράψε μια φορά, Τρέξε παντού» (Write Once, Run Anywhere, εν συντομία WORA) και σημαίνει ότι εφ' όσον μια εφαρμογή μεταγλωττιστεί, θα πρέπει να τρέχει σε όλες τις συσκευές που υποστηρίζουν Java, χωρίς την ανάγκη μεταγλώττισης από την αρχή, ανεξάρτητα από το λειτουργικό ή το υλικό της συσκευής.



Εικόνα 5: Το λογότυπο της java, Μια κούπα καφέ Ιάβας που δανείζει το όνομα του στη γλώσσα

Αυτό επιτυγχάνεται διότι ο κώδικας java μεταγλωττίζεται σε έναν ενδιάμεσο κώδικα που ονομάζεται bytecode και εκτελείται σε μια ψηφιακή μηχανή Java (Java Virtual Machine – JVM), χωρίς να υπάρχει καμία διάδραση του κώδικα bytecode με την υποβόσκουσα αρχιτεκτονική του κάθε συστήματος. Επίσης, ο παραπάνω τρόπος εκτέλεσης εκτός από την ανεξαρτητοποίηση από το υλικό προσθέτει (τουλάχιστον στη θεωρεία) μια παραπάνω βαθμίδα ασφάλειας, αφού μια δυνητικά επιβλαβής εφαρμογή περιορίζεται στα πλαίσια της ψηφιακής μηχανής και δεν επηρεάζει απ' ευθείας την πραγματική μηχανή. Δυστυχώς όμως όλα αυτά τα επιπλέον βήματα στην διαδικασία εκτέλεσης επηρεάζουν αρνητικά την ταχύτητα εκτέλεσης σε σχέση με μια εγγενή (native) γλώσσα όπως η C++, αν και έχουν υπάρξει σημαντικές πρόοδοι σε αυτόν τον τομέα ανά τα χρόνια.

Μια αρκετά κρυφή αλλά αξιόλογη διαφορά ης java με τις περισσότερες γλώσσες προγραμματισμού είναι ότι η java δεν διαθέτει κάποιο πρότυπο αναγνωρισμένο από οργανισμό προτυποποίησης, όπως ANSI η ISO και η υλοποίηση της ιδιοκτήτριας εταιρείας Oracle θεωρείται ως de facto πρότυπο. Η πιο πρόσφατη και μοναδική επίσημα υποστηριζόμενη από την oracle υλοποίηση της Java είναι η Java 8, αλλά υπάρχει πληθώρα άλλων υλοποιήσεων όπως οι OpenJDK – μια ανοιχτού κώδικα υλοποίηση, Apache Harmony – πολύ αξιόλογη προσπάθεια της Apache που δυστυχώς είχε άδοξο τέλος λόγω νομικών ζητημάτων, η υλοποίηση της google – παραλλαγή της Apache harmony που χρησιμοποιείται ευρέως στο Android. IcedTea – Η μόνη

υλοποίηση που έχει το πρόσθετο ελεύθερου λογισμικού Java για browsers, και πολλές άλλες.

Χαρακτηριστικά

JDK

Η Java, όντας γλώσσα προγραμματισμού που έστω και de facto το πρότυπο της ελέγχεται από εταιρεία λογισμικού, έχει το προνόμιο «επίσημης» σουίτας ανάπτυξης λογισμικού, το Java Development Kit. Το JDK περιέχει οτιδήποτε θα περίμενες κανείς από μια οποιαδήποτε σουίτα ανάπτυξης λογισμικού, όπως μεταγλωττιστή σε bytecode, εργαλείο εντοπισμού σφαλμάτων, εργαλεία profiling κώδικα, προσωπική Java virtual machine (προσωπική με την έννοια ότι είναι ξεχωριστή εγκατάσταση από την κύρια JVM του συστήματος), εργαλεία συμπίεσης και βελτιστοποίησης κώδικα, και πολλά ακόμα. Η μεγάλη διαφορά του JDK με ένα SDK οποιασδήποτε άλλης γλώσσας είναι ότι η Oracle, θέτει και σε αυτό τον τομέα το de facto standard, κάτι που οδηγεί σε μεγαλύτερο βαθμό τυποποίησης και συμβατότητας μεταξύ των άλλων διάφορων SDK της Java. (τεχνικά η Java είναι πλέον open source και υπάρχει μεγάλο πλήθος SDK για αυτή, όπως το πολύ δημοφιλές OpenJDK).

JVM

Σύμφωνα με τα de facto standard, μια ορθή υλοποίηση της Java Virtual machine, είναι οποιοδήποτε πρόγραμμα μπορεί να μετατρέψει κώδικα bytecode και να τον μετατρέψει σε κώδικα μηχανής για εκτέλεση στη μηχανή του χρήστη. Οι λεπτομέρειες όπως διαχείριση μνήμης, συλλογή σκουπιδιών και άλλες εσωτερικές διαδικασίες αφήνονται κενές εσκεμμένα ώστε να μη περιορίζουν τους προγραμματιστές και να ενισχύουν την διαλειτουργικότητα. Η διαλειτουργικότητα αυτή επεκτείνεται όχι μόνο στο σύστημα που θα τρέχει η JVM, αλλά και στις γλώσσες που δέχεται. Μπορεί να μην είναι εμφανές εκ πρώτης όψεως αλλά η JVM δέχεται σαν είσοδο java bytecode και όχι τον ίδιο τον κώδικα. Αυτό επιτρέπει την εκτέλεση οποιαδήποτε υλοποίησης γλώσσας είναι ικανή να μεταγλωττιστεί σε java bytecode όπως η JRuby και Jython, εκδοχές της Ruby και Python (Αν και πρακτικά οποιαδήποτε αναδρομική γλώσσα, όχι απαραίτητα προγραμματισμού μπορεί να μεταγλωττιστεί σε κώδικα μηχανής).

Garbage Collector

Μια ακόμα τεράστια καινοτομία της Java σε σχέση με αρκετές παραδοσιακές γλώσσες προγραμματισμού είναι η αυτόματη διαχείριση μνήμης. Η γενική ιδέα είναι ότι ο συλλέκτης σκουπιδιών εντοπίζει αντικείμενα που δεν χρησιμοποιούνται στο πρόγραμμα και απελευθερώνει την μνήμη τους. Η Java μπορεί να έφερε τον συλλέκτη σκουπιδιών στο προσκήνιο αλλά ο διαχειριστής σκουπιδιών προηγείται πολλά χρόνια, αφού εφευρέθηκε το 1959 από τον John McCarthy για χρήση στην γλώσσα Lisp. Στα πλαίσια της JVM, το πρότυπο δεν ορίζει τίποτα σχετικά με τον συλλέκτη και αυτό σημαίνει ότι η υλοποίηση του αφήνεται στον κάθε κατασκευαστή, κάτι που κάνει τον συλλέκτη άκρως ασαφές και συχνά αναξιόπιστο κομμάτι της ψηφιακής μηχανής. Τα προτερήματα μιας τέτοιας προσέγγισης στην διαχείριση μνήμης είναι εμφανή καθώς προλαμβάνει σφάλματα «ξεκρέμαστου» δείκτη και διπλής απελευθέρωσης, καθώς και

τις διαρροές μνήμης, τη μεγαλύτερη πληγή τεράστιου αριθμού προγραμμάτων και φόβο πολλών προγραμματιστών. Από την άλλη πλευρά τα αρνητικά αυτής της προσέγγισης περιλαμβάνουν την επιβάρυνση του συστήματος τόσο σε χρήση μνήμης όσο και σε υπολογισμούς καθώς ανά διαστήματα θα τρέχει ο συλλέκτης σκουπιδιών στο παρασκήνιο όσο τρέχει το πρόγραμμα στο προσκήνιο. Επίσης, αυτό το μοντέλο διαχείρισης μνήμης δεν είναι συμβατό με την χειροκίνητη διαχείριση μνήμης και ο προγραμματιστής δεν έχει κανέναν απολύτως έλεγχο στην αποδέσμευση μνήμης, ακόμα και αν ξέρει ακριβώς ποτέ και πού πρέπει να γίνει, σε τέτοιο βαθμό ώστε στη συντριπτική πλειοψηφία των υλοποιήσεων ο προγραμματιστής δεν μπορεί καν να καλέσει τον συλλέκτη κατά βούληση.

JIT compilation

Όπως αναφέρθηκε παραπάνω, ένα από τα μεγαλύτερα προβλήματα της Java στο παρελθόν ήταν ο αργός χρόνος εκτέλεσης, αφού η JVM έπρεπε να μεταφράζει τον bytecode γραμμή - γραμμή σε κώδικα μηχανής και να εκτελεί κάθε γραμμή μεμονωμένα. Αυτό το πρόβλημα λύθηκε ή πιο σωστά ελαχιστοποιήθηκε το 1997 στην Java 1.1 με την εισαγωγή του just-in-time compiler. Παρόλου που οι κατά παραγγελία μεταγλωττιστές (on-demand compilers) υπήρχαν από την δεκαετία του 1960, η java είναι συνήθως το πιο διάσημο παράδειγμα της βελτίωσης που μπορούν να επιφέρουν στην ταχύτητα εκτέλεσης προγραμμάτων. Πιο παλιά υπήρχαν 2 βασικές προσεγγίσεις για την προσέγγιση κώδικα, η μεταγλώττιση (compilation), όπου ο κώδικας μετατρέπεται σε κώδικα μηχανής η αλλιώς εκτελέσιμο αρχείο μια φορά και στη συνέχεια εκτελείται απ’ ευθείας από την μηχανή, με υψηλό κόστος μεταγλώττισης και χαμηλό εκτέλεσης και η μετάφραση (interpretation) όπου αναλύθηκε παραπάνω με καθόλου κόστος μεταγλώττισης αλλά πολύ μεγάλο κόστος εκτέλεσης. Η βασική ιδέα της κατά παραγγελίας μεταγλώττισης είναι η ισορροπία μεταξύ των 2 παραπάνω, περιορίζοντας τα αρνητικά των δύο παραπάνω προσεγγίσεων. Ο κώδικας μεταγλωττίζεται κατά την εκτέλεση του προγράμματος όπως στην μετάφραση αλλά σε μεγαλύτερα μπλοκ και όχι ανά γραμμή, γεγονός που εξαλείφει την αρχική μεταγλώττιση αλλά με πολύ μικρότερο αντίκτυπο στον χρόνο εκτέλεσης από την μετάφραση για κάθε γραμμή ξεχωριστά. Βέβαια η κατά παραγγελία μεταγλώττιση δε γίνεται να είναι πιο γρήγορη από την εκτέλεση εγγενούς κώδικα, αλλά είναι σε αποδεκτά πλαίσια. Από το πιο γρήγορο στο πιο αργό είναι, native code, JIT compilation, interpretation, compilation (η διαδικασία μεταγλώττισης).

Σχεδιασμός

Βασικοί στόχοι

Πριν την εκκίνηση της διαδικασίας σχεδίασης της εφαρμογής και των επιμέρους στοιχείων της, θα ήταν δόκιμο να προσδιοριστούν κάποιοι βασικοί στόχοι και κατευθυντήριες γραμμές που θα καθορίζουν τόσο την διαδικασία σχεδίασης όσο και υλοποίησης.

- Η υλοποίηση της λειτουργίας των επιμέρους λειτουργειών (εντοπισμός, ανάλυση, μοντελοποίηση, σύγκριση άδειων και πρόταση νέας άδειας) θα πρέπει να είναι σχετικά ανεξάρτητη και η ανταλλαγή πληροφοριών μεταξύ αυτών των υποσυστημάτων θα πρέπει να γίνεται με δομημένο και ασφαλή τρόπο.
- Ο κώδικας θα πρέπει να είναι εύκολα αναγνώσιμος και σωστά τεκμηριωμένος για την περαιτέρω αναθεώρηση και συντήρηση.
- Ο κώδικας επίσης θα πρέπει να είναι εύκολα επεκτάσιμος για την διευκόλυνση ανάπτυξης μελλοντικών εκδόσεων της εφαρμογής με διορθωμένα σφάλματα ή περαιτέρω λειτουργικότητα.
- Η εισαγωγή νέων υποστηριζόμενων αδειών θα πρέπει να είναι όσο το δυνατό ευκολότερη.

Έχοντας υπόψιν τα παραπάνω οι απαιτήσεις του λογισμικού αναγνώρισης και πρότασης αδειών διαμορφώνονται ως εξής:

Πίνακας 3: Πίνακας απαιτήσεων λογισμικού

Αριθμός	Απαίτηση	Τύπος
#1	Εύρεση της κύριας άδειας του λογισμικού σε μορφή αρχείου κειμένου στον κύριο φάκελο ενός project.	Λειτουργική
#2	Εύρεση όλων των αδειών βιβλιοθηκών σε αρχεία κειμένου στους υποφακέλους ενός project	Λειτουργική
#3	Λεκτική και σημασιολογική ανάλυση στο κείμενο της άδειας για την εξαγωγή μοντέλου.	Λειτουργική
#4	Σύγκριση των αδειών με ένα πλήθος προεγκατεστημένων υποστηριζόμενων αδειών.	Λειτουργική
#5	Εύρεση της συμβατότητας μεταξύ των αναγνωρισμένων αδειών και της κύριας άδειας.	Λειτουργική
#6	Έγκριση της επιλογής του χρήστη αν η άδεια του project είναι συμβατή ή πρόταση μιας η παραπάνω αδειών	Λειτουργική
#7	(Μερική) ανεξαρτησία των υποσυστημάτων του λογισμικού.	Μη Λειτουργική
#8	Εύκολη εισαγωγή και διαγραφή αδειών από την λίστα υποστηριζόμενων αδειών	Μη Λειτουργική
#9	Υποστήριξη μόνο των ενδεικνυόμενων τεχνικών χρήσης άδειων λογισμικού.	Μη Λειτουργική
#10	Υποστήριξη αρχάριων χρηστών μέσω της διεπαφής	Μη Λειτουργική
#11	Αλλά και καταγραφή διάφορων στατιστικών για πιο έμπειρους χρήστες.	Μη Λειτουργική

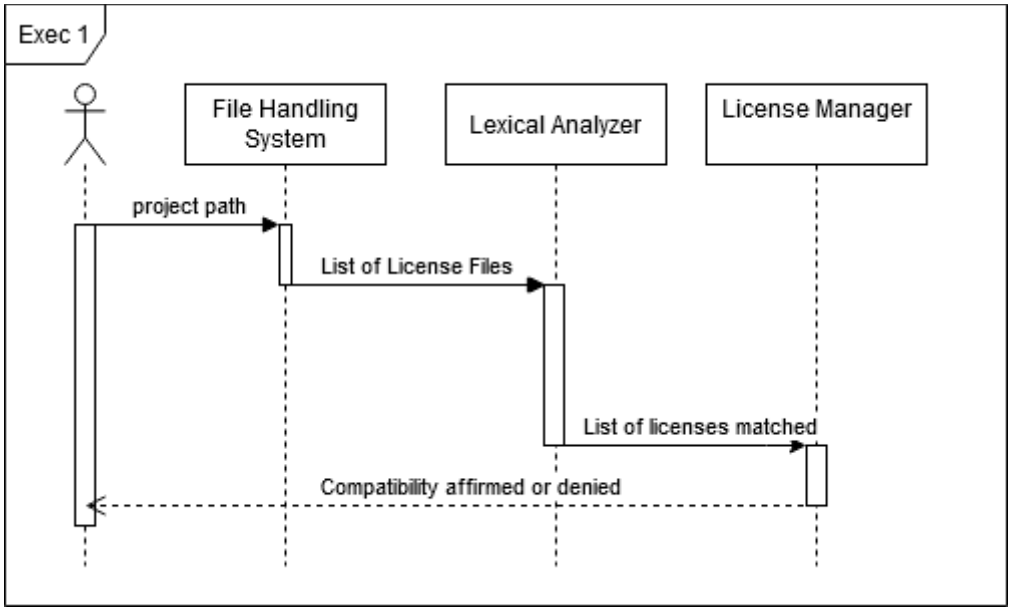
Σχεδιασμός Υποσυστημάτων

Έχοντας υπόψιν τους παραπάνω στόχους υλοποίησης και τις διαθέσιμες τεχνολογίες έφτασε η ώρα για τον σχεδιασμό της εφαρμογής. Η πρώτη μεγάλη απόφαση που πρέπει να παρθεί είναι ο διαχωρισμός της εφαρμογής σε επιμέρους υποσυστήματα, τι πρέπει να περιέχει το καθένα και το πώς αλληλοεπιδρούν μεταξύ τους. Οποιαδήποτε διαδικασία και να ακολουθήσει στη σχεδίαση ή την υλοποίηση, δεν θα είναι ιδιαίτερα αποδοτική αν βασίζεται σε λανθασμένη βάση.

Αρχικά, παρατηρώντας τις παραπάνω λειτουργικές απαιτήσεις, χωρίζονται σε 3 βασικές κατηγορίες.

- Οι #1 και #2 απαιτούν την εύρεση αρχείων σε μια δομή φακέλου.
- Οι #3 και #4 απαιτούν λεκτική ή/και σημασιολογική ανάλυση σε αρχείο κειμένου και σύγκριση των μοντέλων που προκύπτουν.
- Οι #5 και #6 είναι ανεξάρτητες από τα προηγούμενα και ασχολούνται με τις σχέσεις μεταξύ αδειών σαν οντότητες.

Από τα παραπάνω, προκύπτουν 3 κυρίως υποσυστήματα που θα απαρτίζουν τον πυρήνα του λογισμικού. Το υποσύστημα αρχείων, ο λεκτικός αναλυτής και το υποσύστημα αδειών.



Εικόνα 6: Το διάγραμμα ροής μεταξύ των υποσυστημάτων του προγράμματος

Υποσύστημα αρχείων

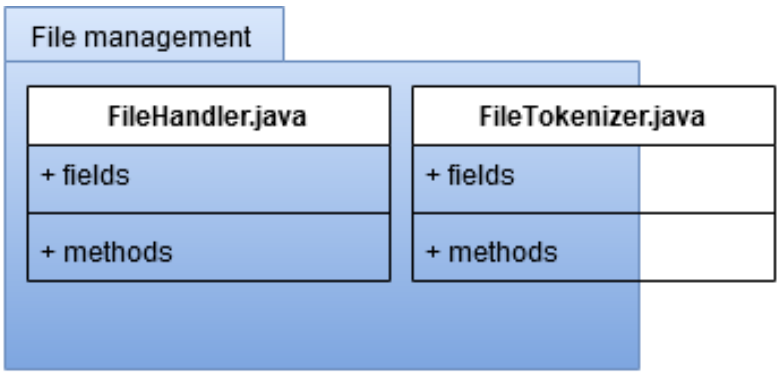
Το υποσύστημα αρχείων δέχεται σαν είσοδο το project που ενδιαφέρει τον χρήστη και θα είναι υπεύθυνο για την ανίχνευση της κύριας άδειας του λογισμικού (αν υπάρχει) και όσο το δυνατό περισσότερες άδειες στα πλαίσια που τίθενται από πλευράς σχεδιασμού.

Ο κυριότερος περιορισμός σε αυτό το υποσύστημα από σχεδιαστικής άποψης έχει αναφερθεί παραπάνω και είναι η υποστήριξη (μόνο) αδειών που βρίσκονται σαν πλήρη αρχεία κειμένου στον κύριο φάκελο του project, ή σε κάποιο υποφάκελο αν πρόκειται για άδεια βιβλιοθήκης με τίτλο LICENSE(.md) ή COPYING(.md). Επίσης, εδώ τίθεται το ζήτημα της διάδρασης με τον χρήστη. Μιας και μια πλήρης διεπαφή δεν είναι απαραίτητη στα πλαίσια αυτής της εργασίας, επαρκεί η χρήση ενός full ή relative path στον κυρίως φάκελο του αρχείου.

Μια τέτοια προσέγγιση εξυπηρετεί ιδιαίτερα, αλλά δε περιορίζεται σε, μεγάλα projects στα οποία χρησιμοποιείται κάποιο λογισμικό διαχείρισής εξαρτήσεων η βιβλιοθηκών όπως npm[2] ή Laravel-composer[3], καθώς κατά σύμβαση κατεβάζουν κάθε πακέτο σε δικό του, απομονωμένο φάκελο με τα αρχεία κώδικα καθώς και οποία συνοδευτικά αρχεία όπως το αρχείο της άδειας.

Οι κλάσεις του υποσυστήματος περιλαμβάνουν:

- **FileHandler.java** – Η κλάση υπεύθυνη για την λειτουργικότητα σε επίπεδο αρχείων (σαν αντικείμενα στον δίσκο). Εξερευνάει φακέλους, εντοπίζει τα αρχεία που εκπληρούν κάποιες προϋποθέσεις και διαχειρίζεται τα δικαιώματα τους.
- **FileTokenizer**.java – Η κλάση υπεύθυνη για την ανάγνωση των αρχείων σαν ροές εισόδου (input streams) και την εξαγωγή των νοηματικών μονάδων (tokens) από το απλό κείμενο. Επίσης λειτουργεί σαν κλάση προσωπείου (facade) ανάμεσα στο υποσύστημα αρχείων και τον λεκτικό αναλυτή.



Εικόνα 7: Το υποσύστημα διαχείρισης αρχείων

[2] https://www.npmjs.com/about
[3] https://getcomposer.org/

Υποσύστημα λεκτικού αναλυτή

Ο λεκτικός αναλυτής αποτελεί με μία έννοια την «καρδιά» του προγράμματος, αφού μετατρέπει την (για έναν υπολογιστή) σχετικά άχρηστη λεκτική πληροφορία των κειμένων αδειών στα αντίστοιχα διανυσματικά μοντέλα, τα οποία μπορούν να ευρετηριοποιηθούν, ζυγιστούν και συγκριθούν ώστε η ομοιότητα τους να χρησιμοποιηθεί για την ταυτοποίηση ή όχι των κειμένων με τις άδειες.

Οι πρώτες έννοιες που πρέπει να αναλυθούν εδώ, είναι οι έννοιες της λεκτικής και σημασιολογικής μονάδας. Αυτές οι έννοιες είναι άρρηκτα συνδεδεμένες με τις γλώσσες προγραμματισμού αλλά είναι εξαιρετικά χρήσιμες και εφαρμόσιμες σε φυσικές γλώσσες και γραμματικές.
Μια **λεκτική μονάδα** είναι μια αδιαίρετη μονάδα που χρησιμοποιείται στην κατασκευή λέξεων, φράσεων και εν τέλει κειμένων. Πχ. η λέξη αυτοκίνητο περιέχει 3 λεκτικές μονάδες, το πρόθεμα (αυτό-) που δίνει ένα επιπλέον νόημα στην ρίζα (στη προκειμένη περίπτωση «τον εαυτό του»), τη ρίζα (-κίνη-) που προσδιορίζει το κεντρικό νόημα (εδώ «η έννοια της κίνηση») και την κατάληξη (-το) που προσδιορίζει τη θέση και το ρόλο του (εδώ ουδέτερο ουσιαστικό). Μια παρατήρηση είναι ότι το νόημα μιας λέξης βρίσκεται στη ρίζα και το πρόθεμα ενώ το επίθεμα έχει πιο πολύ δομικό σκοπό μέσα στην γραμματική.
Η **νοηματική μονάδα** (semantic unit) είναι λίγο πιο αφηρημένη έννοια και περιλαμβάνει την αντιστοίχιση της λεκτικής μονάδας (ή της λέξης) σε μια συγκεκριμένη έννοια, πράξη ή αντικείμενο. Πχ. Σε επίπεδο σημασιολογικών μονάδων έχουμε: αυτοκίνητο = αμάξι = 🚗(Σε περίπτωση μη λεκτικού μοντέλου). Όπως αναφέρθηκε παραπάνω, το νόημα μιας λέξης εμπεριέχεται στο πρόθεμα και την ρίζα, αλλά δεν επαρκεί η απομόνωση τους για να έχουμε την πλήρη εικόνα. Το νόημα μιας πρότασης (και κατ' επέκταση όλων των λέξεων και λεκτικών μονάδων που περιέχει) αντιστρέφεται εντελώς αν υπάρχει μια άρνηση (οι κανόνες της κάθε γλώσσας καθορίζουν τι γίνεται με πάνω από μία αρνήσεις). Τέλος τα συνώνυμα, εξ ορισμού αντιστοιχούν στις ίδιες σημασιολογικές μονάδες.
Για τους σκοπούς του προγράμματος, ως σημασιολογική μονάδα ορίζεται το προϊόν της λεκτικής ανάλυσης που περιέχει το πρόθεμα και την ρίζα της λέξης, όπως δίνεται από τον αλγόριθμο Porter Stemmer [13] το οποίο τροποποιείται περεταίρω με το πρόθεμα «neg-» αν υπάρχει άρνηση στην πρόταση (περίοδο λόγου). Για λόγους ευκολίας και επειδή η γλώσσα που χρησιμοποιείται στις άδειες είναι σχετικά τυποποιημένη, θα αγνοούνται τα συνώνυμα.

Το επόμενο θέμα που πρέπει να αντιμετωπιστεί είναι η αποθήκευση και αναπαράσταση των λεκτικών/σημασιολογικών μοντέλων σε μορφή που διευκολύνει την σύγκριση μεταξύ τους. Ευτυχώς, αυτό είναι ένα παραδοσιακό και σχεδόν τετριμμένο ερώτημα στον τομέα ανάκτησης πληροφορίας και μια σχετικά απλή αλλά ισχυρή λύση είναι το ανεστραμμένο ευρετήριο πληροφορίας (inverted information index) στο οποίο κάθε έγγραφο, στην επικειμένη περίπτωση κάθε άδεια αποθηκεύεται σαν διάνυσμα με άξονες όλες τις μοναδικές λεκτικές (η νοηματικές) μονάδες και περιεχόμενο το βάρος κάθε μονάδας στο παρόν έγγραφο όπως αυτό ορίζεται. [14]

Η διαδικασία επιλογής βάρους στα διανύσματα ονομάζεται «ζύγισμα» του ευρετηρίου και στην περίπτωση που όλα τα βάρη είναι στο διάστημα [0-1] λέγεται «κανονικοποίηση». Στην πλειοψηφία των εφαρμογών στον τομέα ανάκτησης πληροφορίας, όπως μηχανές αναζήτησης κ.α. προτιμάται το σχήμα ζυγίσματος <tf-idf> (Term Frequency * Inverse Document Frequency) το οποίο υπολογίζει το βάρος ενός όρου ανάλογα με τον αριθμό των εμφανίσεων του στο έγγραφο και αντιστρόφως ανάλογα με τον αριθμό των εμφανίσεων του όρου σε όλα έγγραφα του ευρετηρίου.
Στα πλαίσια αυτού του προγράμματος όμως θα χρησιμοποιηθεί το σχήμα ζυγίσματος <tf> για δύο κυρίως λόγους.

1. Σε μια άδεια λογισμικού δεν υπάρχουν σημαντικοί και ασήμαντοι όροι (εκτός από τις λέξεις άνευ σημασίας που θα συζητηθούν αργότερα) και κατ' επέκταση δεν έχει καμία απολύτως σημασία σε ποια άλλη άδεια εμφανίζεται ένα όρος.
2. Μια τέτοια προσέγγιση κάνει τα ορθογραφικά λάθη σε μια άδεια μοναδικούς όρους με δυσανάλογα μεγάλο βάρος, ενώ στην πραγματικότητα, στην πλειοψηφία των περιπτώσεων ο οποιοσδήποτε, ακόμα και ένα δικαστήριο θα μπορούσε να καταλάβει το νόημα της λέξης που περιέχει το ορθογραφικό λάθος.

Στο σχήμα <tf> που θα χρησιμοποιηθεί, το βάρος μιας μονάδας σε μια άδεια θα ισούται με τον αριθμό των φορών που εμφανίζεται. Αξίζει να σημειωθεί ότι πολλές λέξεις μπορεί να αντιστοιχούν στην ίδια νοηματική μονάδα (semantic token) και στον αντίποδα δύο ίδιες λέξεις μπορεί να αντιστοιχούν σε διαφορετικές μονάδες λόγω της αλλαγής νοήματος από τα συμφραζόμενα (πχ άρνηση σε άλλο σημείο της πρότασης).

Το επόμενο ζήτημα αφορά το μέγεθος του τελικού ευρετηρίου και κατ' επέκταση τον χρόνο προσπέλασης του. Ένα σύγχρονο υπολογιστικό σύστημα μπορεί να κάνει λεκτική ανάλυση σε πολύ μεγάλα κείμενα σε ικανοποιητικό χρόνο, αλλά δεν υπάρχει λόγος να μη γίνουν περεταίρω βελτιώσεις.
Λίγο πιο πάνω αναφέρθηκε το σχήμα ζυγίσματος <tf-idf>. Πίσω από αυτό το σχήμα υπάρχει η βασική ιδέα ότι όσο σε περισσότερα έγγραφα βρίσκεται μια λεκτική η σημασιολογική μονάδα, τόσο λιγότερο περιγράφει ένα συγκεκριμένο έγγραφο. Οι δυο ακραίες εκφάνσεις αυτής της ιδέας είναι οι μοναδικοί όροι που περιγράφουν μοναδικά ένα έγγραφο και οι μονάδες τόσο κοινές που περιγράφουν όλα τα έγγραφα το ίδιο (δηλαδή δεν περιγράφουν κανένα έγγραφο). Σε μια φυσική γλώσσα που χρησιμοποιείται για την συγγραφή αδειών, ένα μεγάλο κομμάτι των λέξεων δεν περιέχουν δικό τους νόημα αλλά γεμίζουν δομικούς και βοηθητικούς ρόλους στη γραμματική. Τέτοιες λέξεις είναι άρθρα, αντωνυμίες, βοηθητικά ρήματα κλπ. Αυτές οι λέξεις ονομάζονται άνευ σημασίας (stopwords) και μπορούν να αγνοηθούν σχετικά ασφαλώς χωρίς να επηρεάσουν ιδιαίτερα το νόημα ενός κειμένου, εξοικονομώντας παράλληλα μεγάλο όγκο υπολογισμών και τελικό μέγεθος του ευρετηρίου.
Η εύρεση των stopwords είναι εκτός του πλαισίου αυτής της εργασίας και θα χρησιμοποιηθεί μια έτοιμη και ευρέως διαδεδομένη λίστα stopwords της αγγλικής γλώσσας.[4]

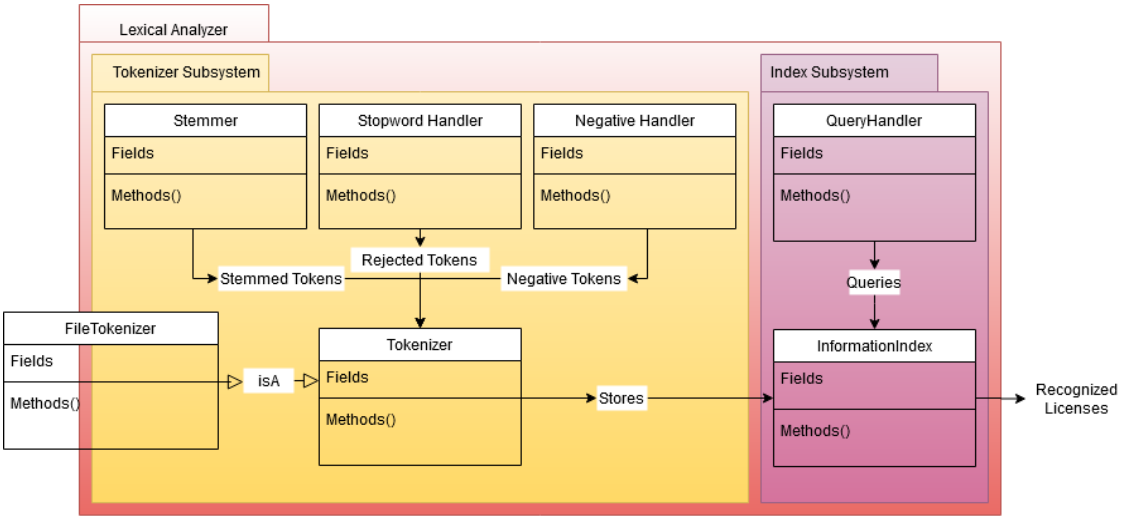
[4] https://www.ranks.nl/stopwords

Το τελευταίο ζήτημα που προκύπτει σχεδιαστικά είναι το η αλλαγή του νοήματος των νοηματικών μονάδων με βάση τα συμφραζόμενα. Ευτυχώς, λόγω της φύσης μιας άδειας λογισμικού σαν νομικό έγγραφο, αποφεύγεται ή χρήση διφορούμενων και νεφελωδών όρων (ακόμα ένα επιχείρημα κατά των αδειών κηρομπογιάς) και το νόημα κάθε περιόδου λόγου πρέπει να είναι όσο το δυνατόν πιο ξεκάθαρο και μη επιρρεπές σε παρερμηνείες. Παρ' όλα αυτά υπάρχει μια εξαιρετικά σημαντική περίπτωση αλλαγής νοήματος που είναι συνηθέστατη στην καθημερινότητα: η άρνηση.
Μια περίοδος λόγο μπορεί να περιέχει δεκάδες λέξεις και τις αντίστοιχες νοηματικές μονάδες, οι οποίες για τους σκοπούς εύρεσης ομοιότητας αντιστρέφουν το νόημα όλης της περιόδου λόγου. (Σε γραμματικό επίπεδο, αντιστρέφουν μόνο το νόημα του ρήματος μιας πρότασης, αλλά όλες οι λέξεις σε μια πρόταση αποτελούν χαρακτηρισμούς πάνω στο ρήμα). Έτσι δύο προτάσεις που σε μια πιο απλή λεκτική ανάλυση α είχαν μεγάλο βαθμό ομοιότητας, η σημασιολογική ομοιότητα τους είναι 0 (όντας ακριβώς αντίθετα). Μια τέτοια ανάλυση είναι εξαιρετικά σημαντική σε έγγραφα που περιέχουν δικαιώματα και υποχρεώσεις όπως οι άδειες αφού μια άρνηση αντιστρέφει το νόημα ενός ολόκληρου όρου.
Φυσικά αυτή η προσέγγιση έχει τις δικές τις προκλήσεις όπως η εύρεση μιας λίστας αρνήσεων που είναι περιέχει μεγάλο όγκο αρνήσεων χωρίς να περιέχει λέξεις αρνητικού συναισθήματός (negative sentiment) και τον προσδιορισμό του εύρους μιας πρότασης και κατά συνέπεια μιας άρνησης. Αυτά τα ζητήματα είναι επίσης εκτός του εύρους αυτής της εργασίας και ανήκουν στον τομέα την ανάλυσης συναισθήματος (sentiment analysis) και φιλολογίας αντίστοιχα.
Για την ανάπτυξη του λογισμικού χρησιμοποιήθηκε μια πολύ απλοποιημένη έκδοση της λίστας λέξεων που παρουσιάστηκε στο [15] και εκφράζουν αρνητικό συναίσθημα, από την οποία αφαιρέθηκαν οι περισσότερες λέξεις που δεν εκφράζουν αρκετά ισχυρή άρνηση ή δεν έχουν θέση σε έγγραφα νομικού τύπου.

Οι κλάσεις του υποσυστήματος περιλαμβάνουν:

- **Tokenizer.java** – Αυτή η abstract κλάση καθορίζει την συμπεριφορά και την πλειοψηφία των μεθόδων της διαδικασίας εξαγωγής νοηματικών μονάδων από απλό κείμενο.
- **FileTokenizer.java** – Όπως αναφέρθηκε παραπάνω, αυτή η υλοποίηση του Tokenizer εξειδικεύεται στην εξαγωγή tokens από αρχεία κειμένου συγκεκριμένα.
- **StopwordHandler.java** - Περιλαμβάνει την λίστα των stopwords και απορρίπτει λέξεις που ανήκουν σε αυτή.
- **Stemmer.java** – Η υλοποίηση του αλγόριθμου Porter stemmer για την εύρεση της ρίζας λέξεων. Χρησιμοποιείται σαν ένα βήμα στην διαδικασία μετατροπής μιας λέξης στην αντίστοιχη νοηματική μονάδα.
- **NegativeHandler.java** – Εδώ περιέρχεται η λίστα των αρνήσεων και η μέθοδος αναγνώρισης αν μια λέξη είναι άρνηση ή όχι.
- **InformationIndex.java** – Αυτή η κλάση μοντελοποιεί το ευρετήριο πληροφορίας και τις λειτουργίες που υποστηρίζει, όπως εισαγωγή εγγράφου, ζύγισμα του ευρετηρίου κλπ.
- **QueryHandler.java** – Αυτή η κλάση μοντελοποιεί τα ερωτήματα προς το ευρετήριο και καθορίζει τον υπολογισμό της ομοιότητας της κάθε ερώτησης

(άδειας που εντοπίστηκε στο project) με τις καταχωρήσεις το ευρετηρίου («επίσημες» υποστηριζόμενες άδειες).

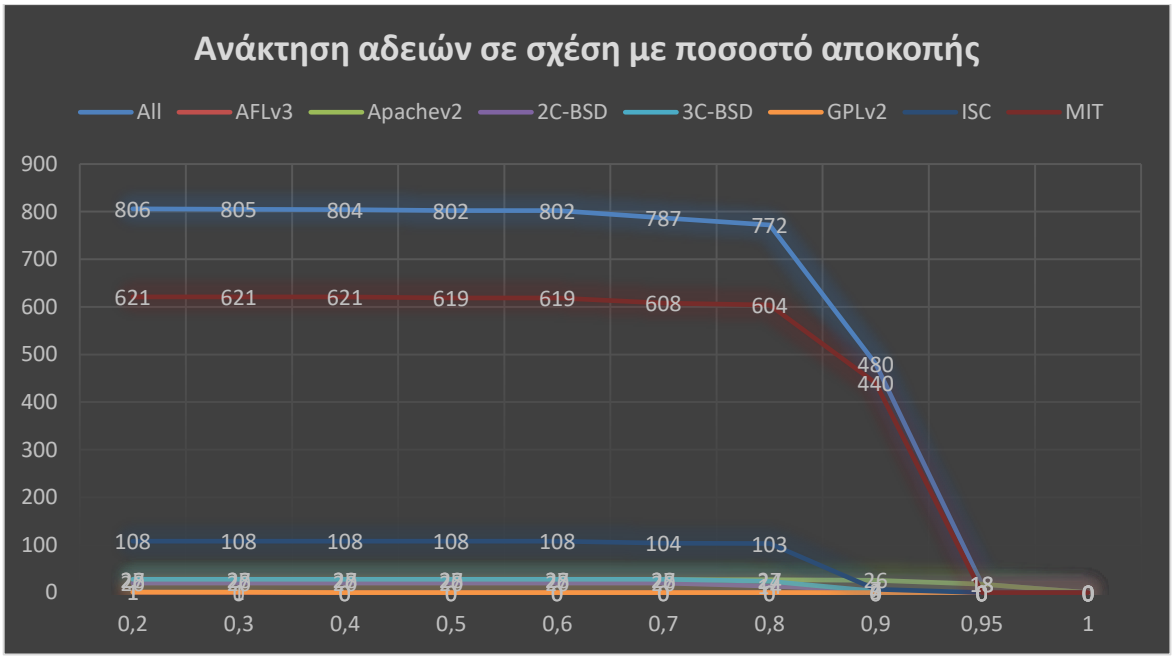


Εικόνα 8: Το υποσύστημα λεκτικού αναλυτή

Υποσύστημα διαχείρισης αδειών

Το υποσύστημα διαχείρισης αδειών είναι υπεύθυνο για την αναγνώριση αδειών και την αντιστοίχιση με τα πρότυπα τους καθώς και την εύρεση της συμβατότητας μεταξύ τους. Το ευρετήριο παρέχει ως έξοδο για κάθε άδεια που εντοπίστηκε στο project μια λίστα με την ομοιότητα (σαν ποσοστό επί τοις 100) με όλα τα πρότυπα αδειών. Το πρώτο σοβαρό ερώτημα είναι πάνω από ποιο ποσοστό ομοιότητας μια άδεια θεωρείται ότι είναι ίδια με το πρότυπο της;
Ιδανικά θα έπρεπε η ομοιότητα να είναι 100%, αλλά για λόγους που αναφέρθηκαν παραπάνω, κάτι τέτοιο είναι εξαιρετικά απίθανο. Με ένα υψηλό ποσοστό αποκοπής, κάποιες άδειες που θα έπρεπε να αναγνωριστούν απορρίπτονται, ενώ στον αντίποδα, κάποιες άδειες που θα έπρεπε να απορριφθούν αναγνωρίζονται. Στην επιστήμη πληροφορίας, αυτό είναι το αιώνιο ερώτημα ακρίβειας – ανάκλησης (precision – recall) [16].

Η ακρίβεια είναι αρκετά δύσκολο να προσδιοριστεί, καθώς το αν μια μετατροπή σε μια άδεια είναι ισοδύναμη με την αρχική άδεια είναι εγγενώς νομικό ζήτημα (πέρα από τα προφανή, όπως ορθογραφικά λάθη), αλλά η ανάκτηση μπορεί να λειτουργήσει ως μέτρο προσδιορισμού της ποιότητας του αποτελέσματος. Σε ένα ενδεικτικό project με περίπου 800 βιβλιοθήκες και άδειες, το ποσοστό ανάκτησης ανά άδεια και συνολικά σε σχέση με το ποσοστό αποκοπής δίνεται από το παρακάτω σχήμα..



Η μεγάλη καμπή, γνωστή ως «γόνατο» βρίσκεται σε ποσοστό αποκοπής 0,8 σε άδειες όπως οι BSD, MIT και ISC, ενώ αρκετά υψηλότερα, περίπου στο 0.95 σε άδειες όπως οι Apachev2 και GPLv2. Κάτι τέτοιο είναι αναμενόμενο, αφού σε μεγαλύτερες άδειες (σε αριθμό λέξεων) ο κάθε όρος που αλλάζει έχει μικρότερη σημαντικότητα στο σύνολο της. Αυτό σημαίνει ότι ένα ποσοστό αποκοπής δεν είναι επαρκές και κάθε άδεια πρέπει να έχει το δικό της ποσοστό αποκοπής, ανάλογο με το μέγεθος της, το οποίο τοποθετείται στο «γόνατο» της ανάλογης σειράς. Ώστε να μετριαστεί η απώλεια ανάκλησης και κατ' επέκταση το κέρδος της ακρίβειας.

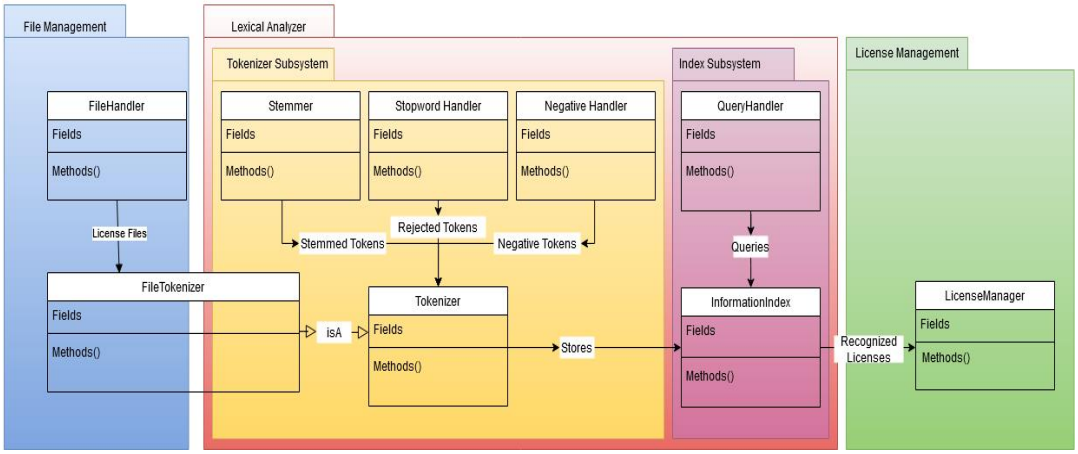
Το επόμενο θέμα που καλείται να λύσει ο διαχειριστής αδειών είναι η συμβατότητα μεταξύ αδειών. Για τους σκοπούς της εργασίας θα χρησιμοποιηθεί ένα υπογράφημα της εικόνας 3 σαν βάση στο οποίο προστίθενται οι άδειες ISC (Internet Systems Consortium) σαν ισοδύναμη της MIT [17] λόγω της δημοτικότητας της και η EUPLv1.2 μιας και αποτελεί πρωτοφανή προσπάθεια προτυποποίησης στον τομέα από φορέα με προ υπάρχουσα νομική ισχύ [18]. Επίσης η συμβατότητα της άδειας ορίζεται σε ένα από τα παραρτήματα της ονομαστικά και είναι επίσημα μεταφρασμένη στις γλώσσες των κρατών μελών της ΕΕ (και πλέον και στα Αγγλικά).
Εσωτερικά, θα χρησιμοποιηθεί μια παραλλαγή του πίνακα γειτνίασης, ο πίνακας μονοπατιού. Ένας πίνακας NxN όπου κάθε γραμμή αποτελείται από το διάνυσμα μιας άδειας «βιβλιοθήκης» και περιέχει true η false ανάλογα με το αν η άδεια είναι συμβατή (αν υπάρχει μονοπάτι με αφετηρία αυτή την άδεια)με όλες τις άλλες βιβλιοθήκες που υποστηρίζονται. Αρχικά μια υλοποίηση μνήμης N^2 μπορεί να φαίνεται αντιπαραγωγική, αλλά έχει δυο μεγάλα πλεονεκτήματα.

- Μπορεί να περιέχει μόνο ένα bit σε κάθε κελί, ελαχιστοποιώντας σε πρακτικό επίπεδο την χρήση μνήμης για σχετικά μικρό υποστηριζόμενο αριθμό αδειών.
- Έχοντας ένα μέρος των υπολογισμών έτοιμο στη μνήμη έχει κέρδη σε επίπεδο χρόνου, κάτι που χρησιμοποιεί πολύ ο λεκτικοσημασιολογικός αναλυτής.

Τέλος, όπως αναφέρθηκε στον πρόλογο, μια πολύ χρήσιμη λειτουργικότητα σε προγράμματα τέτοιας χρήσης είναι η πρόταση μιας άδειας στον χρήστη αν η αρχική άδεια δεν είναι συμβατή ή δεν έχει επιλεχθεί άδεια λογισμικού εξ' αρχής.. Η εις βάθος ανάλυση των απαιτήσεων του χρήστη ή των αναγκών του project κρίνεται εκτός του σκοπού αυτής της εργασίας, αλλά σε αυτή την περίπτωση θα προτείνεται μια λίστα με όλες τις συμβατές άδειες λογισμικού.

Λόγω του σχετικά μικρού μεγέθους των υπολογισμών, αυτό το υποσύστημα περιλαμβάνει μόνο την κλάση LicenseManager.java πού είναι υπεύθυνη για τον ορισμό σημείων αποκοπής στις άδειες, την εύρεση συμβατότητας των αναγνωρισμένων αδειών βιβλιοθηκών με την αρχική καθώς και την εύρεση συμβατότητας όλων των αδειών μεταξύ τους (για την πρόταση νέας άδειας).

Έχοντας υπ' όψιν τα παραπάνω, το τελικό διάγραμμα κλάσεων του λογισμικού διαμορφώνεται ως εξής:



Εικόνα 9: Το διάγραμμα κλάσεων του προγράμματος

Προφανώς ένα μεγάλο κομμάτι του σχεδιασμού προηγείται της υλοποίησης, αλλά η ανάπτυξη λογισμικού είναι μια κυκλική διαδικασία στον πυρήνα της και κάποια κομμάτια του σχεδιασμού, όπως η επιλογή ποσοστού αποκοπής βασίζεται σε αποτελέσματα και μαθήματα από αποτελέσματα εκτέλεσης του λογισμικού. Για λόγους ευκολίας στον αναγνώστη όλα τα κομμάτια του σχεδιασμού, ανεξάρτητα από το τον χρόνο της σύλληψης τους ομαδοποιούνται παραπάνω.

Υλοποίηση

Προκαταρκτικά

Για την ανάπτυξη του λογισμικού χρησιμοποιήθηκε η έκδοση Java 1.8.0_251 όντας η τελευταία stable έκδοση κατά την ημερομηνία δημιουργίας του project, αλλά λόγω της της ισχυρής συμβατότητάς της JRE (Java Runtime Environment) είναι στην φιλοσοφία της γλώσσας η υποστήριξη πολλών παλαιότερων εκδόσεων και κατ' επέκταση δεν τίθενται θέματα συμβατότητας. Συγκεκριμένα, η Java 8 είναι προγραμματισμένη να λαμβάνει ενημερώσεις ασφαλείας μέχρι το 2030 και να υπάρχει γενική συμβατότητα για απεριόριστο χρόνο. [19] Θεωρητικά, το πρόγραμμα θα πρέπει να μπορεί να τρέξει σε οποιαδήποτε έκδοση Java μεγαλύτερη από την 1.8 σε απεριόριστο βάθος χρόνου.

Oracle Java SE Support Roadmap*†				
Release	**GA Date**	**Premier Support Until**	**Extended Support Until**	**Sustaining Support**
7	July 2011	July 2019	July 2022*****	Indefinite
8**	March 2014	March 2022	December 2030	Indefinite
9 (non-LTS)	September 2017	March 2018	Not Available	Indefinite
10 (non-LTS)	March 2018	September 2018	Not Available	Indefinite
11 (LTS)	September 2018	September 2023	September 2026	Indefinite
12 (non-LTS)	March 2019	September 2019	Not Available	Indefinite
13 (non-LTS)	September 2019	March 2020	Not Available	Indefinite
14 (non-LTS)	March 2020***	September 2020	Not Available	Indefinite
15 (non-LTS)	September 2020***	March 2021	Not Available	Indefinite

Εικόνα 10: Το πρόγραμμα υποστήριξης εκδόσεων Java

Σε επίπεδο IDE, χρησιμοποιήθηκε το Apache NetBeans 12.0 λόγω του μεγάλου βαθμού ενσωμάτωσης με την γλώσσα Java, καθώς και για λόγους ευκολίας και εμπειρίας. Για το χτίσιμο του project επιλέχθηκε το εργαλείο ANT, το οποίο είναι απλούστερο και ελαφρύτερο από πιο σύγχρονες εναλλακτικές όπως τα Gradle και Maven, κυρίως γιατί περιορίζει το scope του στο χτίσιμο και την μεταγλώττιση του project και δεν ελέγχει όλο τον κύκλο ζωής του λογισμικού.

Μετά την δημιουργία ενός νέου ANT project, το Ide δημιουργεί μια main κλάση LicComp η οποία εκτελείται με την εκκίνηση του προγράμματος. Αυτή η κλάση θα είναι υπεύθυνη για την διάδραση με τον χρήστη, την μεταφορά μηνυμάτων μεταξύ των επιμέρους κλάσεων και την παραμετροποίηση διαφόρων κομματιών που το υποστηρίζουν (όπως ο τρόπος ζυγίσματος του ευρετηρίου).

Στη συνέχεια, εισήχθησαν οι έτοιμες κλάσεις βιβλιοθηκών, stemmer.java και κομμάτι των NegativeHandler.java και StopWordHandler.java ώστε η λειτουργικότητα τους να είναι διαθέσιμη όταν χρειαστεί.

Υλοποίηση Υποσυστημάτων

Λόγω της σχετικής ανεξαρτησίας των υποσυστημάτων, κάθε υποσύστημα υλοποιείται ξεχωριστά, με την σειρά που αναφέρεται στο σχήμα 6. Συγκεκριμένα, Υποσύστημα αρχείων –> Λεκτικός Αναλυτής –> Υποσύστημα αδειών, με τα αντίστοιχα κομμάτια της main κλάσης να γεμίζουν τα λογικά κενά μεταξύ των υποσυστημάτων.

Επίσης το API κάθε κλάσης και των μεθόδων της υπάρχει στα αρχεία πηγαίου κώδικα, κάτω από τον φάκελο Javadoc.

Υποσύστημα διαχείρισης αρχείων

Η ανάπτυξη της εφαρμογής ξεκινάει με την υλοποίηση του υποσυστήματος λόγω της ανεξαρτησίας του από το υπόλοιπο σύστημα και με σκοπό την χρήση του ως ακρογωνιαίο λίθο για την ανοικοδόμηση της υπόλοιπης εφαρμογής.

Η κλάση δεν περιέχει κάποιες μεταβλητές, καθώς παρέχει μόνο λειτουργικότητα σε επίπεδο αρχείων.
Αφού δεν υπάρχει κίνδυνος διαρροής πληροφορίας από πεδία της κλάσης, μιας και δεν υπάρχουν, οι μέθοδοι είναι public για την επαναχρησιμοποίηση τους σε άλλες εφαρμογές στο μέλλον.

Επίσης, μιας και δεν υπάρχουν πεδία προς αρχικοποίηση, δεν είναι απαραίτητη η δημιουργία νέας μεθόδου `constructor`.

Στη συνέχεια ακολουθούν οι μέθοδοι της κλάσης `FileHandler` που υλοποιούν τη λειτουργικότητα που πρέπει να έχει η κλάση. Ο κώδικας δεν θα παρατεθεί ολόκληρος λόγω χώρου, εκτός από κάποια σημεία που χρήζουν περαιτέρω ανάλυσης.

Η μέθοδος `File openDir(String dirPath)` ελέγχει τα δικαιώματα ανάγνωσης και αν είναι φάκελος στο αρχείο που δείχνει η μεταβλητή `dirPath` και το ανοίγει σαν `File` αν πληροί τα παραπάνω κριτήρια.

Η μέθοδος `SortedSet<String> getAllFileNames(File dir)` δέχεται σαν όρισμα ένα αρχείο φακέλου `dir` και επιστρέφει ένα αλφαβητικά ταξινομημένο σετ με τα σχετικά paths όλων των αρχείων που περιέχει ο φάκελος. Σε αυτή την μέθοδο δεν περιέχονται οι έλεγχοι που έχουν ήδη γίνει στην `openDir()`.

Η μέθοδος `SortedSet<String> getFilteredFileNames(File dir, String filter)` παρέχει την ίδια λειτουργία με την παραπάνω, με την διαφορά ότι δέχεται ένα επιπλέον όρισμα `pattern` και επιστρέφει μόνο τα σχετικά paths των αρχείων που το όνομα τους ταιριάζει στο regular expression pattern που δόθηκε.

Τέλος υπάρχει η εσωτερική `private class PatternFilter` η οποία υλοποιεί το interface `FilenameFilter` και καθορίζει το πότε ένα αρχείο ταιριάζει στο pattern. Στην προκειμένη περίπτωση καλείται η ενσωματωμένη μέθοδος `matches` τον αντικειμένων string η οποία δεν κάνει κάποια αλλαγή στον μηχανισμό των regular expression της java.

Υποσύστημα λεκτικού αναλυτή

Ακολουθεί το υποσύστημα λεκτικού αναλυτή και η βασική του κλάση Tokenizer που εξάγει έναν αριθμό λεκτικών/νοηματικών tokens από μια απροσδιόριστη είσοδο και εφαρμόζει επάνω τους έναν αριθμό οξοποιήσεων ώστε να είναι έτοιμα για ευρετηριόποιηση.

Τα πεδία της κλάσης είναι `protected`, καθώς είναι απαραίτητη η χρήση τους από τις κλάσης που θα υλοποιούν την `Tokenizer` στον τομέα εισόδου.

```
protected Scanner input;//The scanner to extract raw tokens from the in
put
protected final Stack<String> tokenBuffer; //the buffer that stores
tokens for reversal if a negation is found
```

Ο constructor ορίζεται ως εξής:

```
    public Tokenizer(InputStream inputFile)
    {
        this.input=new Scanner (inputFile);
        this.tokenBuffer = new Stack<>();
        input.useDelimiter("[ \t\n\\x0B\\f\\r-]+");
        //separate tokens at any whitespace or -
    }
```

Ο `constructor` δέχεται το ένα InputStream για το οποίο είναι υπεύθυνη η υλοποίηση του και χωρίζει τις λέξεις με λευκά κενά ή παύλες (τα υπόλοιπα σημεία στίξης παίζουν ρόλο στη δομή της πρότασης και καθαρίζονται αργότερα, όχι κατά την είσοδο)

Η αφηρημένη μέθοδος `protected void pushNextToken()` είναι υπεύθυνη για την ανάκτηση μιας λέξης από την είσοδο και την αποθήκευση της στην στοίβα του buffer.

Η μέθοδος `String getNextToken()` επιστρέφει το πρώτο token του buffer η `null` αν είναι άδειος.

Η μέθοδος `String getNextNonTrivialToken(boolean ignoreNums)` καλεί την `getNextToken()` μέχρι να εντοπίσει μια λέξη που δεν είναι stopword. Το όρισμα `ignoreNums` χρησιμοποιείται στην επιλογή αν οι λέξεις που περιέχουν αριθμούς θεωρούνται stopwords.

Η μέθοδος `String getNextStemmedToken(boolean ignoreNums)` καλεί την `getNextNonTrivialToken()` με όρισμα το `ignoreNums`, μετατρέπει το token σε πίνακα χαρακτήρων, ώστε να μπορεί να αναγνωριστεί από τον `Stemmer` και επιστρέφει το αποτέλεσμα.

Η αφηρημένη μέθοδος `public boolean isDone()` καθορίζει αν δεν υπάρχει άλλο κείμενο στην είσοδο και κατ' επέκταση μπορεί να κλείσει.

Η μέθοδος `String cleanupToken(String token)` καθαρίζει τα tokens από σύμβολα που δεν είναι αλφαριθμητικά και μετατρέπει τα πάντα σε πεζά. Τέλος απορρίπτει τα tokens που έχουν μείνει κενά μετά την αντικατάσταση (πχ σειρές από σημεία στίξης)

Η `boolean hasToken()` ελέγχει αν ο token buffer είναι άδειος η περιέχει επιπλέον tokens.

Η μέθοδος `boolean tokenEndsSentence(String token)` ελέγχει αν το `token` κλείνει μια πρόταση, εντοπίζοντας αν το token τελειώνει με τελεία, κόμμα, την αγγλική άνω τελεία(;) η αν βρίσκεται σε μια προεπιλεγμένη λίστα λέξεων που τελειώνουν προτάσεις. Αυτή η μέθοδος είναι απαραίτητη στο σημασιολογικό κομμάτι του αναλυτή καθορίζοντας το εύρος μιας άρνησης.

Τέλος ο εσωτερικός πίνακας `private static final String[] ENDING_TOKENS` περιέχει τις λέξεις κλειδιά που κλείνουν μια πρόταση, όπως σύνδεσμοι, αντωνυμίες κλπ.

Η επόμενη κλάση προς ανάλυση είναι η `FileTokenizer` η οποία είναι υλοποίηση της `Tokenizer` και χρησιμοποιεί μια προσέγγιση τύπου FSM (finite state machine) στη οποία προσομοιώνει μια μηχανή πεπερασμένων καταστάσεων για την ανάγνωση tokens από αρχεία αδειών.

Το πεδίο `private TokenizerState state` αποθηκεύει την εσωτερική FSM κατάσταση του `FileTokenizer` σαν μια από τις πιθανές τιμές που ορίζονται στο enumeration `TokenizerState {STANDBY, READY, READING, NEGATIVE, DONE}`

Ο `constructor` της κλάσης αρχικοποιεί το πεδίο `state` και καλεί τον `super` constructor της κλάσης `Tokenizer`.

```
public FileTokenizer(FileInputStream inputFile)
{
    super(inputFile);
    state= TokenizerState.STANDBY;
}
```

Η μέθοδος **pushNextToken()** είναι υπεύθυνη για την εξαγωγή tokens από το αρχείο εισόδου και την εισαγωγή στον buffer. Στην αρχή του κάθε αρχείου απορρίπτει τις γραμμές που ταιριάζουν στο regular expression ((?m)^[cC]opyright.+[0-9]{4}.+$) καθώς οι κάτοχοι των copyrights αλλά και τα έτη αλλάζουν από το πρότυπο της άδειας στην κάθε υλοποίηση. Στη συνέχεια διαβάζει κάθε λέξη με την μέθοδο `next()` και περνάει κάθε λέξη από μια σειρά μεθόδων για τον μετασχηματισμό του σε token.

Αρχικά κοιτάει αν το token τελειώνει μια πρόταση και αν ναι εισάγει στον buffer το ειδικό `«END-OF-SENTENCE»` token. Στη συνέχεια, καλεί την `cleanupToken()` και ελέγχει αν είναι άρνηση. Αν είναι άρνηση, εξάγει τα tokens από τον buffer, τους δίνει το πρόθεμα `«neg-»`, τα εισάγει ξανά στον buffer και μεταβαίνει στην κατάσταση `negative`. Στην κατάσταση `negative`, όλα τα tokens εισάγονται στον buffer με το πρόθεμα `«neg-»`.

Οι κλάσεις `NegativeHandler` και `StopwordHandler` περιέχουν εσωτερικά έναν πίνακα με λέξεις που ταιριάζουν στην κλάση (αρνήσεις και stopwords) και την αντίστοιχη μέθοδο is για να ελέγχουν αν μια λέξη βρίσκεται στον εσωτερικό πίνακα.

Η κλάση `InformationIndex` μοντελοποιεί ένα αντεστραμμένο ευρετήριο πληροφορίας στο οποίο εισάγονται τα semantic tokens μετά την εξαγωγή τους από το κείμενο της άδειας.
Εσωτερικά χρησιμοποιεί `TreeMaps` για την προσομοίωση associative πινάκων (πίνακες με δείκτη αντικείμενο) που κρατάνε πληροφορίες σχετικά με τα έγγραφα και τα tokens που ευρετηριοποίουνται. Συγκεκριμένα περιέχει τα πεδία:

```
private final SortedMap<String, SortedMap<String, Double>> body
   //the main index, as a 2d array represented by a Map of Maps
private final SortedMap<String, Integer> countList
   //contains the total occurences of each token
private final SortedMap<String, Integer> wordsInDoc
   //contains the total number of tokens in a document
private final SortedMap<String, Integer> docsInWord
   //contains the number of ducuments that each token is present
```

Ο `constructor` απλά αρχικοποιεί τα πεδία με κενά TreeMaps ώστε να είναι έτοιμα για χρήση στη συνέχεια.

Η μέθοδος `void insertDocument(String docName)` εισάγει το έγγραφο με όνομα `docName` στο ευρετήριο περιέχοντας 0 tokens.

Η μέθοδος `boolean containsDocument(String docName)` ελέγχει αν το έγγραφο με όνομα `docName` υπάρχει στο ευρετήριο.

Η μέθοδος `boolean insertToken(String docName, String token)` αρχικά ελέγχει αν το έγγραφο `docName` υπάρχει και το `token` προϋπάρχει ή είναι καινούριο. Αν είναι καινούριο το εισάγει σε όλα τα έγγραφα με αριθμό εμφανίσεων 0 και στις βοηθητικές δομές. Τέλος εισάγει το `token` στο έγγραφο `docName.`

Η μέθοδος `boolean containsToken(String token)` ελέγχει αν υπάρχει το `token` σε οποιοδήποτε έγγραφο του ευρετηρίου. Για λόγους ταχύτητας ελέγχει τη βοηθητική δομή `docsInWord` αντί να διατρέχει το ευρετήριο.

Η `void printIndex(OutputStream out, boolean verbose)` εκτυπώνει το ευρετήριο σε μια (σχετικά) αναγνώσιμη μορφή για σκοπούς αποσφαλμάτωσης στο stream εξόδου `out` με κάποια επιπλέον μηνύματα αν το flag `verbose` είναι ενεργοποιημένο.

Οι μέθοδοι `int getDictionarySize()` και `int getDocumentNumber()` επιστρέφουν τον συνολικό αριθμό μοναδικών tokens και εγγράφων στο ευρετήριο αντίστοιχα.

Η μέθοδος `void applyWeightsScheme(String weightScheme)` είναι υπεύθυνη για το «ζύγισμα» του ευρετηρίου και πρέπει να καλείται υποχρεωτικά μόνο και μόνο μια φορά μετά την εισαγωγή του τελευταίου token. Τα ορίσματα που δέχεται είναι είτε «tf» όπου δεν κάνει τίποτα, καθώς το ευρετήριο είναι ήδη σε βάρος "tf", είτε «tf-idf» στο οποίο συνυπολογίζεται η συχνότητα εμφάνησης του κάθε token σε άλλα έγγραφα. Για τους σκοπούς του προγράμματος, χρησιμοποιείται η τιμή «tf».

Η μέθοδος `getDocDominantTerms(int n, String docName)` επιστρέφει τα `n tokens` με μεγαλύτερο βάρος στο έγγραφο `docName`. Χρησιμοποιείται επίσης μόνο για σκοπούς debugging και η `SortedMap<String, SortedMap<String, Double>> getDominantTermsFromAll(int n)` παρέχει την ίδια λειτουργικότητα για όλα τα έγγραφα.

Στη συνέχεια η μέθοδος `SortedMap<String, Double> similarityWithAll(Map<String, Double> query, boolean normalizedSim)` υπολογίζει την ομοιότητα μεταξύ ου διανύσματος `query` με όλα τα διανύσματα εγγράφων του ευρετηρίου. Εσωτερικά καλέι την μέθοδο `similarity` και επιστρέφει τα αποτελέσματα σε ένα Map ταξινομημένο κατά φθίνουσα σειρά αξίας. Υπάρχει και αντίστοιχη μέθοδος που χρησιμοποιεί `QueryHandler` αντί για διάνυσμα ερώτησης.

Η μέθοδος `Double similarity(String docName, Map<String, Double> query, boolean normalizedSim)` υπολογίζει την ομοιότητα μεταξύ ενός εγγράφου και ενός διανύσματος ερώτησης. Αρχικά η ομοιότητα υπολογίζεται ως εσωτερικό γινόμενο μεταξύ των δυο διανυσμάτων, και αν το flag normalizedSum είναι ενεργό, κανονικοποίει το αποτέλεσμα, χρησιμοποιώντας ουσιαστικά ομοιότητα συνημίτονου. Τέλος μετατρέπει την ομοιότητα από συνημίτονο σε μοίρες και ύστερα σε ποσοστό (υποθέτοντας ότι γωνία 90 μοιρών είναι 0% ομοιότητα). Υπάρχει και αντίστοιχη μέθοδος που χρησιμοποιεί `QueryHandler` αντί για διάνυσμα ερώτησης.

Τέλος η εσωτερική κλάση `reverseValueComparator` χρησιμοποιείται για την επιστροφή των αποτελεσμάτων ταξινομημένα κατά φθίνουσα σειρά αξίας, κάτι που πάει αντίστροφα στην φυσική ταξινόμηση των Maps.

Η τελευταία κλάση του υποσυστήματος λεκτικού αναλυτή είναι η κλάση που είναι υπεύθυνη για την μοντελοποίηση των ερωτήσεων προς το ευρετήριο. Περιέχει τα πεδία:

```
    private final String name;
    private final SortedMap<String,Double> body;
         //the vector of the query
    private final FileInputStream inputStream;
         //the input stream to read the query from
```

Ο `constructor` δέχεται ένα ρεύμα εισόδου και αρχικοποιεί τα πεδία `in` και `body` με ένα άδειο διάνυσμα.

Η μέθοδος `nextQuery()` χρησιμοπιεί εσωτερικά έναν FileTokenizer και φτιάχνει ένα νέο διάνυσμα ερώτησης, ζυγισμένο κατά «tf» καθώς σε μια απομονωμένη ερώτηση δεν έχει νόημα το «idf»

Τέλος, η μέθοδος `getQueryVector()` επιστρέφει το εσωτερικό διάνυσμα `body` και χρησιμοποιείται σε μεθόδους ομοιότητας που χρησιμοποιούν `QueryHandler` σαν όρισμα.

Υποσύστημα Αδειών

Το υποσύστημα αδειών περιέχει μόνο την κλάση `LicenseManager` που είναι υπεύθυνη για το ταίριασμα των αδειών που αναγνωρίστηκαν από τον λεκτικό αναλυτή με τα μοντέλα τους, την εύρεση συμβατότητας μεταξύ των αδειών αυτών και την πρόταση καινούριας άδειας αν δεν εντοπίστηκε κύρια άδεια ή δεν είναι συμβατή με κάποια από τις άδειες βιβλιοθηκών.

Αρχικά περιέχει την εσωτερική κλάση compatibilityManager που κατά βάσιν περιέχει την hardcoded πληροφορία που χρειάζεται για κάθε άδεια, όπως ένα enumeration με της υποστηριζόμενες άδειες, το ποσοστό αποκοπής κάθε άδειας, τον πίνακα συμβατότητας όλων των αδειών και μια βοηθητική μέθοδος που διατρέχει τον πίνακα και αποφασίζει μια άδεια A είναι συμβατή με μια άδεια B.
Επίσης περιέχει τα πεδία:

```
    private final File baseDir
      //the root dir of the project as a file
    private final File mainLicenseFile
      //the main license file
    private compatibilityManager.licenseTypes mainLicense
      //the main license enum or null if not found
    private final List<File> libLicenses
      //a list of the library license files
    private final String[] ACCEPTED_LIC_FILES = {"LICENSE", "LICENSE.md
", "COPYING", "COPYING.md"};  //the accepted filenames
```

Ο `constructor` χρησιμοποιεί το String `basePath` για να βρει τον base φάκελο `baseDir` του project και καλεί 2 φορές την συνάρτηση `checkDir`, μια κοιτώντας μόνο το `baseDir` για την εύρεση του κυρίους αρχείου της άδειας και μια με την αναδρομική της λειτουργεία, κοιτώντας κάθε υποφάκελο του project για να βρεί τις άδειες βιβλιοθηκών και να αρχικοποιήσει τα πεδία `mainLicenseFile` και `libLicenses` αντίστοιχα.

Η μέθοδος `checkDir(File dir, boolean recur)` είναι μια αναδρομική μέθοδος η οποία ανοίγει τον φάκελο `dir` και αφού ελέγξει τα δικαιώματα ανάγνωσης, ψάχνει να βρει αν υπάρχουν αρχεία που το όνομα τους βρίσκεται στον πίνακα `ACCEPTED_LIC_FILES`. Αν το flag `recur` είναι αληθές, για κάθε αρχείο του φακέλου που είναι επίσης φάκελος, καλεί τον εαυτό της αναδρομικά και εισάγει τα αποτελέσματα στην λίστα `libLicenses` αντί να τα επιστέφει όπως μια κύρια άδεια.

```
private static class compatibilityManager{
    static enum licenseTypes {
        aflv3("aflv3",0,0.90),
        apachev2("apachev2",1, 0.90),
        bsd2("bsd2",2, 0.80),
        bsd3("bsd3",3, 0.80),
        cddl("cddl",4, 0.85),
        euplv1_2("euplv1_2",5, 0.90),
        gplv2("gplv2",6,0.90),
        gplv3("gplv3",7, 0.90),
        isc("isc",8, 0.80),
        lgplv2_1("lgplv2_1",9, 0.90),
        lgplv3("lgplv3",10, 0.90),
        mit("mit",11, 0.80),
        mplv2("mplv2",12, 0.85);

        private final int pos;
        private final String name;
        private final double cutoff;

        private licenseTypes(String name, int pos, double cutoff){
            this.pos=pos;
            this.name=name;
            this.cutoff=cutoff;
        }
    }

    private static boolean iscompatible(String libLicence, String mainLicense){
        return matrix[licenseTypes.valueOf(libLicence).pos][licenseTypes.valueOf(mainLicense).pos];
    }
    private static final boolean[][] matrix={
       //                aflv3 apav2  bsd2  bsd3  cddl eupl+ gpl21 gplv3  isc  lgpl+ lgpl3  mit  mplv2
       /* aflv3     */  {true ,false,false,false,false, true,false,false,false,false,false,false,false},
       /* apachev2  */  { true, true,false,false,false, true,false, true,false,false, true,false, true},
       /* bsd2      */  { true, true, true, true,false, true, true, true,false, true, true,false, true},
       /* bsd3      */  { true, true,false, true,false, true, true, true,false, true, true,false, true},
       /* cddl      */  {false,false,false,false, true, true,false,false,false,false,false,false,false},
       /* euplv1_2  */  {false,false,false,false,false, true, true,false,false, true, true,false, true},
       /* gplv2_1   */  {false,false,false,false,false, true, true, true,false, true,false,false,false},
       /* gplv3     */  {false,false,false,false,false, true,false, true,false, true, true,false,false},
       /* isc       */  { true, true, true, true,false, true, true, true, true, true, true, true, true},
       /* lgplv2_1  */  {false,false,false,false,false, true, true, true,false, true, true,false,false},
       /* lgplv3    */  {false,false,false,false,false, true,false, true,false,false, true,false,false},
       /* mit       */  { true, true, true, true,false, true, true, true, true, true, true, true, true},
       /* mplv2     */  {false,false,false,false,false, true, true, true,false, true, true,false, true}
    };
}
```

Εικόνα 11: Τα περιεχόμενα της κλάσης CompatibilityManager

Η μέθοδος `Boolean isMainLicencePermitted(Map<String,Integer> licensesFound)` ελέγχει την συμβατότητα μεταξύ της κύριας `mainLicense` άδειας και όλων των αδειών που εντοπίσηκαν στο project `licensesFound` και επιστρέφει `false` αν εντοπιστεί έστω και μια ασυμβατότητα μεταξύ τους.

Η `μέθοδος recommendLicenses(Map<String, Integer> licensesFound)` ελέγχει τη συμβατότητα όλων των υποστηριζόμενων αδειών με τις άδειες που βρέθηκαν `licensesFound` και επιστρέφει όσες είναι συμβατές με όλες τις άδειες στην λίστα. Χρησιμοποιείται για την πρόταση νέας άδειας στον χρήστη.

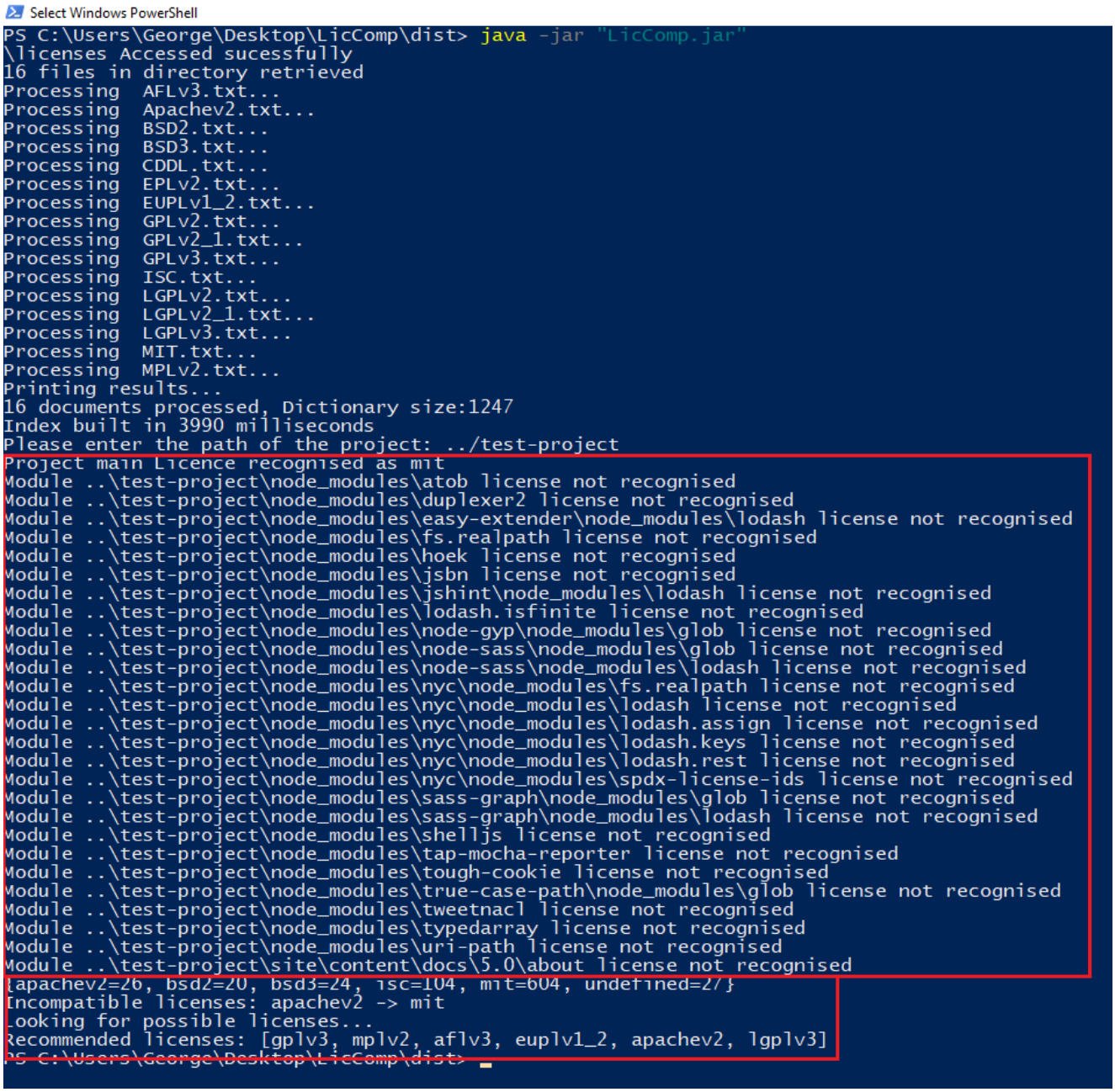
Τέλος, η μέθοδος `Double getLicenceCutoff(String license)` επιστρέφει το ποσοστό αποκοπής για την άδεια `license` και χρησιμοποιείται από τον λεκτικό αναλυτή για την απόρριψη αδειών που έχουν μικρότερη ομοιότητα.

Αξίζει να σημειωθεί, ότι όλες οι κλάσεις σε όλα τα υποσυστήματα περιέχουν μεθόδους get και set όπου χρειάζονται, οι οποίες δεν αναφέρονται εδώ για λόγους χώρου

Οδηγίες χρήσης

Αυτό το λογισμικό έχει 2 κύριες λειτουργικότητες, την πιστοποίηση συμβατότητας υπάρχουσας άδειας ή την πρόταση νέας άδειας αν η πιστοποίηση αποτύχει ή δεν υπάρχει άδεια. Και στις 2 περιπτώσεις, ο χρήστης πρέπει να κάνει τα ίδια βήματα, με την έξοδο του προγράμματος να αλλάζει ανάλογα.

1. Λήψη και εγκατάσταση της java Runtime Environment (JRE) 1.8+
2. Νέο παράθυρο terminal/command line.
3. Πλοήγηση στην τοποθεσία που βρίσκεται το «LicComp.jar» (Στον φάκελο Liccomp/dist στα αρχεία που προμηθεύονται).
4. Εκτέλεση της εντολής `java -jar "LicComp.jar"`
5. Μικρή αναμονή μέχρι το πρόγραμμα να φορτώσει και επεξεργαστεί τις υποστηριζόμενες άδειες.
6. Εισαγωγή του απόλυτου η σχετικού path του base φακέλου του project προς εξέταση. (Η κενό για την χρήση ενός ενσωματωμένου demo project).
7. Το πρόγραμμά θα αναφέρει μια λίστα από αρχεία αδειών που δεν αναγνωρίστηκαν αυτόματα προς ενημέρωση του χρήστη.
8. Το πρόγραμμά δείχνει μια λίστα με τον αριθμό και το είδος των αριθμών που αναγνωρίστηκαν και τα τελικά του αποτελέσματα.
    a. Είτε την επιβεβαίωση ότι η δεν υπάρχει πρόβλημα συμβατότητας
    b. Η το πρόβλημα που υπήρξε με την άδεια του project (δεν υπάρχει ή δεν είναι συμβατή με κάποια συγκεκριμένη άδεια) και μια λίστα αδειών που μπορεί να έχει το project.
    c. Η σε σπάνιες περιπτώσεις ειδοποίηση ότι το project δε μπορεί να δημοσιευτεί καν γιατί δεν υπάρχει άδεια που είναι συμβατή με όλες τις άδειες βιβλιοθηκών.



Εικόνα 12: Μια τυπική εκτέλεση του προγράμματος με χρήση του test-project

Μελλοντικά σχέδια

Μπορεί η εφαρμογή να θεωρείται πλήρης για τα πλαίσια αυτής της εργασίας όμως αυτό δε σημαίνει ότι είναι τέλεια. Υπάρχει ακόμα πληθώρα βελτιώσεων και επιπλέον χαρακτηριστικών που μπορούν να υλοποιηθούν πριν η εφαρμογή μπορέσει να πιάσει την πλήρη δυναμικότητα της, που είναι όμως εκτός των πλαισίων αυτής της εργασίας.

Διορθώσεις

Ο λεκτικός αναλυτής έχει περιθώρια βελτίωσης στην εξαγωγή αρνήσεων, την αναγνώριση περιόδων λόγου (προτάσεων) και στην απόρριψη κομματιών που πρέπει να αλλάζουν (όπως έτη, ονόματα κλπ.).

Επεκτάσεις

- Υποστήριξη περισσότερων αδειών
- Δυναμική εισαγωγή αδειών από τον χρήστη.
- Υποστήριξη διπλών αδειών (dual license).
- Καλύτερο UI και UX για αρχάριους χρήστες,

Συμπεράσματα

Ο Σκοπός αυτής της εργασίας είναι η ανάπτυξη μιας πρακτικής προσέγγισης που αναπτύχθηκε κυρίως στις [8] και [11] ώστε να μπορεί να χρησιμοποιηθεί σε πρακτικά projects από αρχάριους προς μέτριους χρήστες και να βοηθήσει στην αντιμετώπιση του προβλήματος του πολλαπλασιασμού και της εύρεσης συμβατότητας αδειών.

Η συγκεκριμένη προσέγγιση προσπαθεί να ενοποιήσει τα λεκτικά μοντέλα των αδειών, αφού αυτά χρησιμοποιούνται περισσότερο από τους χρήστες, αλλά και σε νομικό επίπεδο, με πιο περίπλοκα νοηματικά μοντέλα που χρησιμοποιούνται από την ακαδημαϊκή κοινότητα και τους ειδικούς για την ανάλυση των αδειών λογισμικού σε μεγαλύτερο βαθμό. Αυτή η αρχική προσπάθεια από μόνη της δεν δείχνει κάποια εξαιρετικά αποτελέσματα αλλά με την έλευση της μηχανικής μάθησης (machine learning) και των νευρωνικών δικτύων ένας πολύ πιο ισχυρός σημασιολογικός αναλυτής θα μπορούσε να οδηγήσει στην σχετικά πλήρη κατανόηση των όρων των άδειών και την εύστοχη εύρεση ασυμβατοτήτων με άλλες άδειες.

Bibliography

[1] T. S.-Y. Lee, H.-W. Kim and S. Gupta, "Measuring open source software success,," *Omega, Volume 37, Issue 2,*, pp. 426-438, 2009.

[2] L. McKnight, "IT Lawyer series: What clauses should I look out for when reviewing a software license agreement?," LegalVision, 2019 September 2019. [Online]. Available: https://legalvision.com.au/lawyer-series-clauses-look-reviewing-software-license-agreement/. [Accessed 4 12 2020].

[3] R. W. Gomulkiewicz, "Open Sour Open Source License Pr ce License Proliferation: Helpful Div ation: Helpful Diversity or Hopeless ersity or Hopeless," *Washington University Journal of Law & Policy*, vol. 30, 2009.

[4] G. Stein, Interviewee, *Google says no to license proliferation.* [Interview]. 2 November 2006.

[5] Ben Balter, "Open source license usage on GitHub.com," GitHub, 9 3 2015. [Online]. Available: https://github.blog/2015-03-09-open-source-license-usage-on-github-com/. [Accessed 4 12 2020].

[6] B. Perens, "[License-discuss] a Free Island Public License?," 16 12 2011. [Online]. Available: https://lists.opensource.org/pipermail/license-discuss_lists.opensource.org/2011-December/017416.html. [Accessed 6 12 2020].

[7] P.-H. Kamp, "Poul-Henning Kamp's Personal page," 24 10 2004. [Online]. Available: https://people.freebsd.org/~phk/. [Accessed 6 12 2020].

[8] G. M. Kapitsaki, N. D. Tselikas and I. E. Foukarakis, "An insight into license tools for open source software systems," *The Journal of Systems and Software*, vol. 102, pp. 72-87, 2015.

[9] GitHub inc., "ISC license," GitHub inc., [Online]. Available: https://choosealicense.com/licenses/isc/. [Accessed 6 12 2020].

[10] H. Welte, "AVM violating license of the Linux kernel," gpl-violations.org, 20 6 2011. [Online]. Available: https://gpl-violations.org/news/20110620-avm-cybits/. [Accessed 6 12 2020].

[11] G. M. Kapitsaki, F. Kramer. and N. D. Tselikas, "Automating the license compatibility process in open source software with SPDX," *The Journal of Systems and Software*, vol. 131, pp. 386-401, 2017.

[12] T. Alspaugh, H.U. Asuncion and W. Scacchi, "Intellectual property rights requirements for heterogeneously-licensed systems.," in *Proceedings of 17th IEEE International Requirements Engineering Conference*, 2009.

[13] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130-137, 1980.

[14] C. David, "Reverse indexing," *The Indexer*, vol. 26, no. 1, 2008.

[15] B. Liu, M. Hu and J. Cheng, "Opinion Observer: Analyzing," in *14th International World Wide Web conference (WWW-2005), May 10-14,*, Chiba, Japan, 2005.

[16] M. Buckland and F. Gey, "The relationship between Recall and Precision," *Journal of the Association for Information Science and Technology*, no. 45, pp. 12-19, 1994.

[17] OpenBSD, " OpenBSD Copyright Policy," OpenBSD, 04 12 2016. [Online]. Available: https://www.openbsd.org/policy.html. [Accessed 20 12 2020].

[18] Comission, The European, "COMMISSION IMPLEMENTING DECISION (EU) 2017/863," 18 05 2017. [Online]. Available: https://joinup.ec.europa.eu/sites/default/files/custom-page/attachment/eupl_v1.2_en.pdf. [Accessed 13 12 20202].

[19] Oracle, "Oracle Java SE Support Roadmap," oracle, 13 5 2020. [Online]. Available: https://www.oracle.com/java/technologies/java-se-support-roadmap.html. [Accessed 20 12 2020].

[20] D. Wheeler, "Why Open Source Software? Look at thhe Numbers!," 18 July 2015. [Online]. Available: https://dwheeler.com/oss_fs_why.html.