Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

PythonVoiceCodingPlugin: Syntactically aware μετακίνηση και τροποποίηση κώδικα Python μέσω φωνητικών εντολών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΙΤΣΙΟΣ ΠΑΝΑΓΙΩΤΗΣ

Επιβλέπων : Νικόλαος Σ. Παπασπύρου
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2022

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

PythonVoiceCodingPlugin: Syntactically aware μετακίνηση και τροποποίηση κώδικα Python μέσω φωνητικών εντολών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΙΤΣΙΟΣ ΠΑΝΑΓΙΩΤΗΣ

Επιβλέπων : Νικόλαος Σ. Παπασπύρου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 18η Ιουλίου 2022.

Νικόλαος Σ. Παπασπύρου Καθηγητής Ε.Μ.Π.	Κωνσταντίνος Σαγώνας Αν. Καθηγητής Ε.Μ.Π.	Βασίλειος Βεσκούκης Αν. Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2022

Κίτσιος Παναγιώτης

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

Περίληψη

Programming by voice ή η τέχνη της γραφής κώδικα χρησιμοποιώντας φωνητικές εντολές, μπορεί να είναι ένα ισχυρό εργαλείο για άτομα που υποφέρουν από τραυματισμό επαναλαμβανόμενης καταπόνησης (RSI) ή άλλες αναπηρίες που καθιστούν προβληματική τη χρήση του πληκτρολογίου ή του ποντικιού, καθώς και για τους κανονικούς χρήστες καθώς και για κανονικούς χρήστες που επιθυμούν να αυξήσουν τη ροή εργασίας τους. Ωστόσο, έρχεται επίσης με ένα μοναδικό σύνολο συμβιβασμού σε σύγκριση με τους παραδοσιακούς τρόπους εισαγωγής. Από τη μία πλευρά, η πιθανότητα εσφαλμένων αναγνωρίσεων και η καθυστέρηση που εισάγεται από το σύστημα αναγνώρισης ομιλίας μπορεί να μειώσει την αποτελεσματικότητα των παραδοσιακών εργαλείων ανάπτυξης λογισμικού, βελτιστοποιημένων για τη διαδραστικότητα των χρηστών. Από την άλλη πλευρά, οι φωνητικές εντολές μπορεί να απομνημονεύονται ευκολότερα από τις συντομεύσεις πληκτρολογίου, να αισθάνονται πιο φυσικές και να επιτρέπουν μεγαλύτερο εύρος επιλογών μεμονωμένα παρέχοντας μεγαλύτερη εκφραστικότητα. Η αντιμετώπιση αυτών των προκλήσεων απαιτεί συστήματα που ξεπερνούν την προσπάθεια μίμησης του κλασικού χρήστη του πληκτρολογίου και αντί αυτού να ενσωματωθούν βαθύτερα με τα ήδη υπάρχοντα εργαλεία.

Σε αυτή τη διπλωματική εργασία, ο συγγραφέας επανεξετάζει την προηγούμενη εργασία του από το 2019 σχετικά με την ενσωμάτωση του συντάκτη για το Sublime Text που επέτρεπε στους χρήστες του προγραμματισμού Caster μέσω φωνητικού πλαισίου να χρησιμοποιούν υψηλού επιπέδου συντακτική πλοήγηση και εντολές επεξεργασίας για τη γλώσσα προγραμματισμού Python. Αυτό το έργο επαναπροσεγγίζεται και ξαναγράφεται από την αρχή με έμφαση στην ικανότητα πολλαπλών πλατφορμών όσον αφορά τόσο τους συντάκτες όσο και τα προγράμματα που υποστηρίζονται από φωνητικά πλαίσια.

Λέξεις κλειδιά

Προγραμματισμός με φωνή, συντακτική πλοήγηση, γραμματική

Abstract

Programming by voice, or the art of writing code using voice commands, can be a powerful tool for people suffering from repetitive strain injury(RSI) or other disabilities that render the use of keyboard or mouse problematic ,as well as normal users wishing to augment their workflow. However, it also comes with a unique set of trade-offs when compared to traditional modes of input. On the one hand, the chance of misrecognitions and the latency introduced by the speech recognition system can reduce the effectiveness of traditional software development tooling optimized for user interactivity. On the other hand, voice commands can be easier to remember than keyboard shortcuts, feel more natural and and allow for a higher breadth of options in the single go providing for a higher expressibility. Addressing those challenges requires systems that go beyond trying to emulate classic keyboard user and instead integrate deeper with the already existing tooling.

In this diploma dissertation, the author revisits his previous work from 2019 on an editor integration with Sublime Text that allowed users of the Caster programming by voice framework to use high level syntactic navigation and editing commands for the Python programming language. That project gets reapproached and rewritten from scratch with an emphasis on cross-platform ability in terms of both editors as well as programs by voice frameworks supported.

Key words

Programming by voice, syntactic navigation, grammar, Python

Ευχαριστίες

Ευχαριστώ θερμά τον επιβλέποντα καθηγητή αυτής της διατριβής, κ. Νίκο Παπασπύρου, για τη συνεχή καθοδήγηση και εμπιστοσύνη του. Ευχαριστώ επίσης τα μέλη της συμβουλευτικής επιτροπής, κ.κ. Κωστή Σαγώνα και Βασίλη Βεσκούκη, καθως και την λογοθεραπεύτρια μου Δήμητρα Χαραλαμπίδου για την πρόθυμη και πάντα αποτελεσματική βοήθειά τους, τις πολύτιμες συμβουλές και τις χρήσιμες συζητήσεις που είχαμε. Θα ήθελα να ευχαριστήσω την οικογένειά μου και κυρίως τους γονείς μου, οι οποίοι με υποστήριξαν και έκαναν δυνατή την απερίσπαστη ενασχόλησή μου τόσο με την εκπόνηση της διπλωματικής μου, όσο και συνολικά με τις σπουδές μου. Τελευταίο αλλά εξίσου σημαντικό, θα ήθελα επίσης να εκφράσω την ευγνωμοσύνη μου για τους maintainers όλων των projects του dictation-toolbox, χωρίς την ανιδιοτελή εργασία των οποίων η αποφοίτησή μου θα ήταν αδύνατη

Κίτσιος Παναγιώτης,

Αθήνα, 18η Ιουλίου 2022

Η εργασία αυτή είναι επίσης διαθέσιμη ως Τεχνική Αναφορά CSD-SW-TR-1-22, Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών, Εργαστήριο Τεχνολογίας Λογισμικού, Ιούλιος 2022.

URL: `http://www.softlab.ntua.gr/techrep/`
FTP: `ftp://ftp.softlab.ntua.gr/pub/techrep/`

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Γιατί να ξαναγράψω από την αρχή;

Το αρχικό έργο mpourmpoulis/PythonVoiceCodingPlugin[1] ξεκίνησε γύρω από το πρόγραμμα επεξεργασίας SublimeText και την εγκατάσταση caster μου το καλοκαίρι του 2019, θα κυκλοφορούσε τελικά τον Νοέμβριο του ίδιου έτους και η ανάπτυξη θα συνεχιζόταν μέχρι τα τέλη του 2020. Ήταν αρκετά καλό για τις ανάγκες μου, αλλά υπέφερε από μια ποικιλία θεμάτων που ήθελα να αντιμετωπίσω εδώ και πολύ καιρό.

- Η απόκτηση λειτουργιών όσο το δυνατόν γρηγορότερα ήταν προτεραιότητα γιατί τα χρειαζόμουν απεγνωσμένα, οπότε έπρεπε να γίνουν αντισταθμίσεις στον αρχικό σχεδιασμό του μεταξύ της ταχύτητας ανάπτυξης και της ποιότητας/επεκτασιμότητας/δοκιμασίας κώδικα.
- Η ενσωματωμένη ενότητα `ast`[2] μου προκαλούσε ορισμένα ζητήματα που δεν θα έπρεπε να αντιμετωπίσω με μια σωστή υλοποίηση concrete syntax tree όπως licst[3].
- Έτρεχε μόνο μέσα στο ενσωματωμένο python του SublimeText, η οποία τότε ήταν η ξεπερασμένη 3.3, παρόλο που η νεότερη έκδοση του sublime text 4 προσφέρει έναν πιο σύγχρονο χρόνο εκτέλεσης 3.8.
- Εκ των υστέρων, διάφορα κομμάτια της προσέγγισης της γραμματικής αισθάνονται υπερβολικά και μη συστηματικά σχεδιασμένα.

Αλλά αναμφισβήτητα ο κύριος στόχος αυτής της προσπάθειας επανεγγραφής ήταν να γίνει το έργο cross-platform:

- υποστήριξη πολλών συντακτών,
- καθώς και πολλαπλός προγραμματισμός με φωνητικά πλαίσια, όχι μόνο dictation-toolbox/Caster[4].

Για να ικανοποιηθούν αυτές οι απαιτήσεις, χρειαζόταν μια δραστική επαναπροσέγγιση.

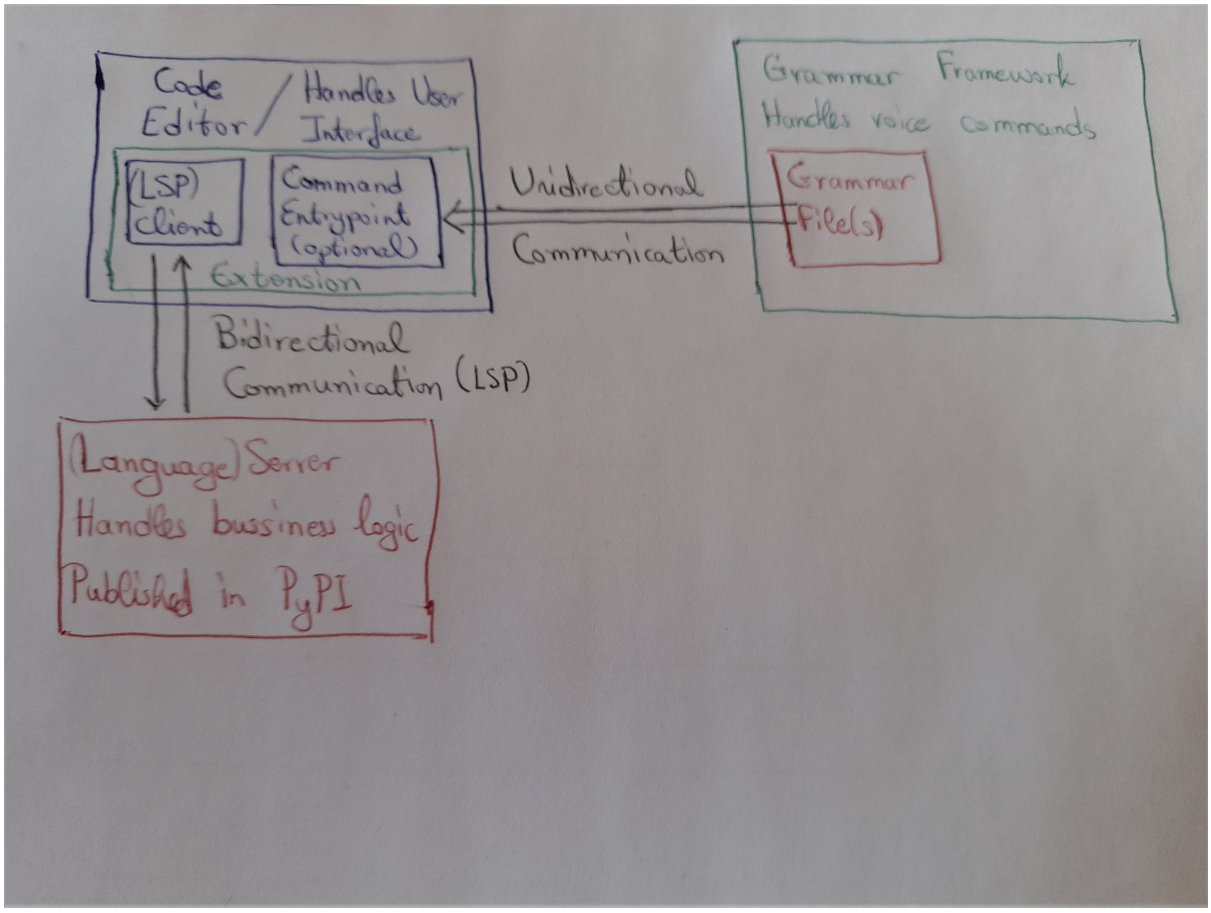
[1] https://github.com/mpourmpoulis/PythonVoiceCodingPlugin
[2] https://docs.python.org/3/library/ast.html#module-ast
[3] https://pypi.org/project/licst
[4] https://github.com/dictation-toolbox/Caster

1.2 Τα συστατικά του συστήματος

Η κεντρική ιδέα πίσω από αυτές τις επανεγγραφές είναι να τραβήξετε όλη την επιχειρηματική λογική έξω από το sublime plugin, να το διανείμετε ξεχωριστά και να το κάνετε διαθέσιμο για χρήση σε άλλους συντάκτες. Το έργο έχει πλέον αναδιοργανωθεί σε τρεις συνιστώσες, το καθένα εκτελείται σε διαφορετικό περιβάλλον.

- Πρώτον, έχουμε τα αρχεία γραμματικής που φορτώνονται από το πλαίσιο φωνητικού προγραμματισμού. Περιέχουν όλους τους ορισμούς φωνητικών εντολών καθώς και τον κωδικό που απαιτείται για την αποστολή ωφέλιμων φορτίων και την ενεργοποίηση ενεργειών στον τρέχοντα ενεργό επεξεργαστή, όταν αυτές οι εντολές αναγνωρίζονται.
- Δεύτερον, για κάθε υποστηριζόμενο πρόγραμμα επεξεργασίας θα πρέπει να υπάρχει μια επέκταση που πρόκειται να λάβει αυτά τα ωφέλιμα φορτία, να τα προωθήσει στη διαδικασία διακομιστή με τον κατάλληλο τρόπο και να φροντίσει όλα τα bits διεπαφής χρήστη που απαιτούνται όπως αλλαγή της επιλογής του δρομέα , επισήμανση κομματιών κώδικα κ.λπ. Ο κωδικός επέκτασης θα πρέπει να διατηρηθεί στο ελάχιστο, καθώς θα πρέπει να επαναδημιουργηθεί για κάθε πρόγραμμα επεξεργασίας χρησιμοποιώντας το εγγενές API και τη γλώσσα προγραμματισμού του και επομένως δεν θα πρέπει να περιέχει επιχειρηματική λογική.
- Τέλος, έχουμε τον διακομιστή, όπου βρίσκεται όλη η επιχειρησιακή λογική και έχουν γίνει όλοι οι καθαροί υπολογισμοί. Όπως θα δούμε, ο διακομιστής πρόκειται να βασιστεί στο πρωτόκολλο διακομιστή γλώσσας και θα εκτελείται στη δική του διαδικασία εκτός του προγράμματος επεξεργασίας, αλλά θα διαχειρίζεται από την επέκταση η οποία θα λειτουργεί ως πελάτης γλώσσας. Ο διακομιστής υλοποιείται ως πακέτο python και δημοσιεύεται στο PyPI ως python_voice_coding_plugin[5] .

[5] https://pypi.org/project/python_voice_coding_plugin



1.3 Σχετικά έργα

Για τον ενδιαφερόμενο αναγνώστη που επιθυμεί να μάθει περισσότερα για τον προγραμματισμό με φωνή γενικά, μια γρήγορη ματιά στο [1] , [2] , [3] , [4] μπορεί να είναι ενδιαφέρον. Εδώ θα μιλήσουμε μόνο για ενσωματώσεις εκδοτών.

- anonfunc/intellij-voicecode[6] , ανοιχτού κώδικα, υποστήριξη για Jet Brains και talon. Βασίζεται στη διεπαφή δομής προγράμματος Jet brains, παρέχει βασική δομική πλοήγηση για πολλές γλώσσες και εκθέτει διάφορα API εγκεφάλου jet μέσω διακομιστή TCP.
- cursorless-dev/cursorless[7] , ανοιχτού κώδικα, υποστήριξη για VsCode και talon. Βασίζεται στο tree-sitter/tree-sitter[8] και παρέχει ισχυρή επεξεργασία για πολλές γλώσσες. Σε σύγκριση με τα έργα μας και με τα έργα αυτής της λίστας που εφαρμόζουν αποτελεσματικά μια προσέγγιση "από πάνω προς τα κάτω" που επιτρέπουν στον χρήστη να περιγράφει τι θέλει να επιλέξει ανάλογα με το είδος της περιοχής και ορισμένες πληροφορίες θέσης, χωρίς δρομέα αντ' αυτού ακολουθεί μια προσέγγιση από κάτω προς τα πάνω. Συγκεκριμένα, για κάθε διακριτικό πηγής σημειώνει ένα ή περισσότερα από τα γράμματά του με καπέλα διαφόρων χρωμάτων και σχημάτων , τα οποία μπορούν να χρησιμοποιηθούν για να το αναγνωρίσουν μοναδικά στις φωνητικές εντολές και στη συνέχεια επεκτείνει την επιλογή προς τα πάνω μέχρι να βρεθεί μια περιοχή του κατάλληλου είδους . Αυτά φυσικά έχουν το κόστος της εξάρτησης από την ισχυρή διεπαφή χρήστη του vscode, αλλά γίνονται προσπάθειες να το μεταφέρουν και σε άλλα προγράμματα επεξεργασίας.

[6] https://github.com/anonfunc/intellij-voicecode
[7] https://github.com/cursorless-dev/cursorless
[8] https://github.com/tree-sitter/tree-sitter

- serenadeai/serenade[9] , προηγουμένως κλειστή πηγή, αλλά η λίστα με το open τον Ιούνιο του 2022. Σε αντίθεση με τα άλλα έργα, είναι ένα ολόκληρο οικοσύστημα με τη δική του αναγνώριση ομιλίας. Υποστηρίζει πολλούς επεξεργαστές και γλώσσες προγραμματισμού και εκτός από τη συντακτική πλοήγηση επιτρέπει επίσης την εισαγωγή κώδικα μορφοποιημένου από μοντέλα μηχανικής εκμάθησης.

[9] https://github.com/serenadeai/serenade

Κεφάλαιο 2

Εγκατάσταση γραμματικής

Καθώς ο αριθμός των στοιχείων του συστήματος αυξάνει την πολυπλοκότητά του, η διατήρηση της διαδικασίας εγκατάστασης όσο το δυνατόν πιο απλή από την οπτική γωνία του χρήστη και η εύρεση ενός αποτελεσματικού τρόπου διανομής των απαραίτητων γραμματικών αρχείων ήταν ο κύριος στόχος της επανεγγραφής.

Σε γενικές γραμμές, υπάρχει μια ποικιλία απόλυτα λογικών τρόπων για την επίτευξη αυτού του στόχου, αλλά σε αυτό το στάδιο του έργου, η μέθοδος που θα επιλεγεί κατά προτίμηση δεν θα πρέπει να απαιτεί χειροκίνητη ενέργεια από τον χρήστη και θα πρέπει να κλιμακώνεται ώστε να υποστηρίζει πολλαπλά πλαίσια προγραμματισμού φωνής καθώς και πολλαπλούς επεξεργαστές κώδικα με όσο το δυνατόν μικρότερο αποτύπωμα.

Θα προχωρήσουμε με τις ιδέες που εξετάστηκαν και τελικά θα παρουσιάσουμε αυτήν που τελικά έγινε αποδεκτή.

2.1 Οι ιδέες που εξετάστηκαν

2.1.1 Git Repository

Ωστόσο

- αν και δεν είναι πραγματικά το τέλος του κόσμου, συνήθως εξακολουθεί να απαιτεί από τον χρήστη να προβεί σε κάποια χειροκίνητα βήματα για την εγκατάσταση ή την ενημέρωση της γραμματικής ή κάποιου επιπλέον κώδικα προκειμένου να τον αυτοματοποιήσει.
- Μπορεί να γίνει προβληματικό εάν θέλετε να υποστηρίξετε πολλαπλά frameworks. Για παράδειγμα, πιθανότατα χρειάζεστε πολλά τέτοια αποθετήρια, ένα για κάθε framework, προκειμένου να αποφύγετε τη λήψη γραμματικών αρχείων για διαφορετικό σύστημα από τους χρήστες στον κατάλογο χρηστών τους. Υπάρχουν φυσικά λύσεις, όπως το να έχετε ένα μόνο monorepo και στη συνέχεια να χρησιμοποιήσετε ένα σύστημα συνεχούς παράδοσης για να συγχρονίσετε τα υπόλοιπα, αλλά τα πράγματα γίνονται λίγο περίπλοκα.

2.1.2 Αυτόματη ανάπτυξη από την προσθήκη Editor

Μια άλλη φυσική ιδέα θα μπορούσε να είναι να συνδυάσετε τις γραμματικές μαζί με τις επεκτάσεις του επεξεργαστή και να τους αφήσετε να χειριστούν αυτόματα την εγκατάσταση/αναβάθμιση, κάτι που μοιάζει κάπως με την παλιά προσέγγιση που ακολουθούσα. Δυστυχώς, ένας σημαντικός περιορισμός είναι ότι περιλαμβάνει πολλή αντιγραφή κώδικα μεταξύ όλων των διαφόρων προσθηκών επεξεργασίας, κάτι που θέλω να αποφύγω.

2.1.3 Ξεχωριστό πακέτο PyPI

Η δημοσίευση της γραμματικής ως πακέτου στο μητρώο **pypi** με τον ίδιο τρόπο που είναι ο διακομιστής διακομιστή γλώσσας υποστήριξης , ήταν επίσης μια ιδέα που θεωρήθηκε φυσική και pythonic, ενώ η χρήση subpackages με εισαγωγές υπό όρους(conditional imports) θα μπορούσε να αντιμετωπίσει το ζήτημα των πολλαπλών frameworks.

Δυστυχώς:

- αυτή τη στιγμή, κανένα από τα πλαίσια που μας ενδιαφέρει δεν έχει εγγενή υποστήριξη για τη φόρτωση γραμματικών από πακέτα
- απαιτείται ακόμη πρόσθετη λογική ή μη αυτόματη ενέργεια χρήστη για την ενημέρωση των πακέτων

2.2 Τρέχουσα προσέγγιση: ανάπτυξη από τον διακομιστή γλώσσας

Η τρέχουσα προσέγγιση είναι να αφήσουμε την ανάπτυξη γραμματικής να χειρίζεται ο ίδιος ο εκτελέσιμος διακομιστής γλώσσας. Συγκεκριμένα, κατά την εκκίνηση του διακομιστή γλώσσας, θα (δημιουργήσει on-the-fly και) θα εγκαταστήσει αρχεία γραμματικής στις αντίστοιχες θέσεις του συστήματος αρχείων τους. Χρησιμοποιώντας αυτή την τεχνική

- Η εγκατάσταση/αναβαθμίσεις είναι όσο το δυνατόν πιο απρόσκοπτη από την πλευρά του χρήστη
- δεν χρειαζόμαστε πραγματικά κανένα επιπλέον κώδικα στους γλωσσικούς πελάτες μας, διατηρώντας τους λεπτούς και πιο γρήγορους για τη μεταφορά του έργου σε πολλούς επεξεργαστές
- Μπορούμε ακόμη και να εγκαταστήσουμε γραμματικές για πολλαπλά πλαίσια ταυτόχρονα

Η διαδικασία δημιουργίας και ανάπτυξης γραμματικής ελέγχεται από μια σειρά από επιλογές γραμμής εντολών/περιβαλλοντικές μεταβλητές και εάν η προεπιλεγμένη συμπεριφορά δεν είναι ικανοποιητική, ο χρήστης πρέπει απλώς να επεξεργαστεί τη διαμόρφωση της επέκτασης για να τις προσαρμόσει.

Κεφάλαιο 3

Δημιουργία της Γραμματικής

3.1 Κινήτρο

Η νέα έκδοση του έργου σηματοδοτεί μια σημαντική αλλαγή παραδείγματος όχι μόνο στον τρόπο με τον οποίο αναπτύσσονται τα αρχεία γραμματικής, αλλά και στον τρόπο με τον οποίο γράφονται, μεταβαίνοντας από χειρόγραφα σε αυτά που δημιουργούνται. υπάρχουν διάφοροι λόγοι για τους οποίους ελήφθησαν αυτές οι αποφάσεις

- Πρώτον, η μη αυτόματη έκδοση μπορεί να είναι χρονοβόρα, πιο επιρρεπής σε σφάλματα και δεν συμμορφώνεται με την αρχή του *DRY* (ή Μην επαναλάβετε τον εαυτό σας). Συγκεκριμένα, με τον έναν ή τον άλλον τρόπο ο διακομιστής γλώσσας θα περιέχει ήδη κώδικα που απαιτείται για την αναπαράσταση/επικύρωση όλων των ωφέλιμων φορτίων που μπορούμε να στείλουμε από τις εντολές που αναγνωρίζει το σύστημα. γράφοντας τη γραμματική με το χέρι σημαίνει ότι πρέπει να εφαρμόσουμε ξανά αυτήν την αναπαράσταση στο αρχείο γραμματικής, με ελάχιστη εγγύηση ότι συγχρονίζεται με αυτήν που υπάρχει στον διακομιστή γλώσσας. για να μην αναφέρουμε ότι το τελευταίο μπορεί επίσης να δημιουργηθεί εν μέρει
- Δεύτερον, η προσέγγιση έχει τη δυνατότητα να κλιμακωθεί καλύτερα εάν θέλετε να υποστηρίξετε πολλαπλά πλαίσια φωνητικού προγραμματισμού . Με μια χειροκίνητη προσέγγιση για να ορίσετε m ομιλούμενες κατασκευές για n πλαίσια, χρειάζεστε ένα σύνολο υλοποιήσεων O(mn) στα αρχεία γραμματικής. Αντί για μια αυτοματοποιημένη, χρειάζεστε υλοποιήσεις O(m) για τον καθορισμό των στοιχείων στον κώδικα υποστήριξης (που πιθανότατα δεν θα μπορούσατε να ξεφύγετε χωρίς κανέναν τρόπο) και υλοποιήσεις O(n) για τις γεννήτριες γραμματικής.

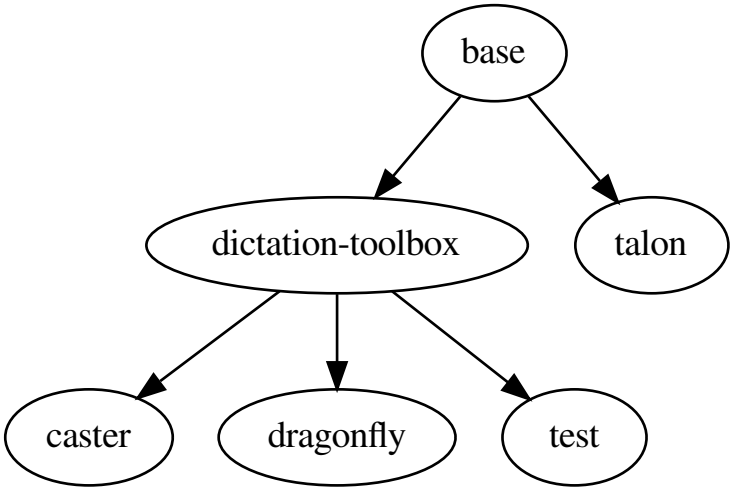
3.2 Jinja templates

Όπως αναφέρθηκε προηγουμένως, υπάρχει μια μη τετριμμένη δυνατότητα για επαναχρησιμοποίηση κώδικα σε διάφορα πλαίσια. Συγκεκριμένα

- είτε μιλάμε για γραμματικές που βασίζονται σε talon or dragonfly, θα χρειαστούμε κάποιον κώδικα για να χειριστούμε την επικοινωνία χαμηλού επιπέδου με τους επεξεργαστές κώδικα μας, ο οποίος μπορεί εύκολα να μοιραστεί
- Όλες οι γραμματικές για την dictation-toolbox είναι σε μεγάλο βαθμό παρόμοιες. Είτε μιλάμε για αυτόνομο dragonfly/caster/breathe, όλα ορίζουν τα προφορικά στοιχεία και τις ενέργειες σχεδόν με τον ίδιο τρόπο και διαφέρουν μόνο ως προς τον τρόπο κατασκευής και φόρτωσης της τελικής γραμματικής, καθώς και τις εισαγωγές που απαιτούνται

Προκειμένου να εκμεταλλευτούμε αυτή τη δυνατότητα επαναχρησιμοποίησης, θα βασιστούμε στο

jinja2[10] , το οποίο είναι ένα ισχυρό πλαίσιο απόδοσης προτύπων (templates) για python, και είναι οι δυνατότητες κληρονομικότητας προτύπων, προκειμένου να οικοδομήσουμε μια ιεραρχία προτύπων



3.2.1 Base

Στο βασικό πρότυπο ορίζουμε πέντε είδη μπλοκ που κάθε πλαίσιο μπορεί να παρακάμψει ή να κληρονομήσει από το μητρικό του και υλοποιούμε την ενότητα κοινής επικοινωνίας

- framework specific prologue
- communication
- actions definitions
- element definitions
- framework specific epilogue

```
{% block prologue %}
{% endblock prologue %}

{% block communication %}
import json
import os
import platform
import subprocess

def validate_subl():
    if platform.system() != 'Windows':
        return "subl"
    windows_candidates=[
        "subl",
        r"C:\Program Files\Sublime Text\subl",
```

(continues on next page)

[10] https://pypi.org/project/jinja2

(continued from previous page)

```
        r"C:\Program Files\Sublime Text 3\subl",
        r"C:\Program Files (x86)\Sublime Text 3\subl",
        r"C:\Program Files (x86)\Sublime Text\subl",
    ]
    for  candidate in windows_candidates:
        try:
            subprocess.check_call([candidate,"-h"],stdout=subprocess.PIPE,
↪stderr=subprocess.PIPE)
            return candidate
        except Exception as e:
            continue
    else:
        raise ValueError("subl not found. Please follow instructions https://www.
↪sublimetext.com/docs/command_line.html")

subl=validate_subl()


def send_sublime(c,data):
    print(c + " " + json.dumps(data))
    command=[subl,"-b", "--command",c + " " + json.dumps(data)]
    # Adopted from https://dragonfly2.readthedocs.io/en/latest/_modules/dragonfly/
↪actions/action_cmd.html#RunCommand
    import subprocess
    # Suppress showing the new CMD.exe window on Windows.
    startupinfo = None
    if os.name == 'nt':
        startupinfo = subprocess.STARTUPINFO()
        startupinfo.dwFlags |= subprocess.STARTF_USESHOWWINDOW


    ret_code = subprocess.call(command, startupinfo=startupinfo)
    return ret_code




{% endblock communication %}

{%block actions_definitions %}
{% endblock %}

{% block element_definitions %}
{% endblock %}

{% block epilogue %}
{% endblock %}
```

3.2.2 Dictation toolbox

Στο dictation tool box template χρειαζόμαστε τη βοήθεια των εξής jinja filters:

```
def generate_contents(generator: str) -> Mapping[Path, str]:
    env = Environment(
        loader=PackageLoader("python_voice_coding_plugin.grammar_generation"),
        autoescape=select_autoescape(),
    )
    env.filters["define_element_dragonfly"] = define_element_dragonfly
    env.filters["define_element_talon"] = define_element_talon
    return {
        k: env.get_template(v).render(elements=elements, generator=generator)
        for k, v in generators_to_templates[generator].items()
    }
```

```
{% extends "base.jinja" %}


{% block prologue %}
from dragonfly import *
{% endblock prologue %}

{% block actions_definitions %}
@Function
def navigate_ast_action(StandardCommand):
    print(StandardCommand)
    send_sublime("lsp_execute", {
            "command_name": "navigate_ast",
            "session_name": "PythonVoiceCodingPlugin",
            "command_args": [StandardCommand,"$file_uri"]
        }
    )
{% endblock actions_definitions %}


{% block element_definitions %}
{% for e in elements %}
{{ e[0] | define_element_dragonfly(e[1]) }}
{% endfor %}
{% endblock %}
```

3.2.3 Dragonfly

```
{% extends "dictation-toolbox.jinja" %}




{% block epilogue %}

class PythonVoiceCodingPluginRule(MappingRule):
    mapping = {
        "<StandardCommand>": navigate_ast_action,
```

(continues on next page)

(continued from previous page)

```
    }
    extras =  [
        StandardCommand,

    ]

context = AppContext(executable="sublime_text")
grammar = Grammar("PythonVoiceCodingPlugin", context=context)
rule = PythonVoiceCodingPluginRule(name="python voice coding plugin")
grammar.add_rule(rule)
grammar.load()

{% endblock epilogue %}
```

3.2.4 Caster

```
{% extends "dictation-toolbox.jinja" %}

{% block prologue %}
{{ super() }}
from castervoice.lib.merge.state.short import R
from castervoice.lib.ctrl.mgr.rule_details import RuleDetails
{% endblock prologue %}


{% block epilogue %}

class PythonVoiceCodingPluginRule(MappingRule):
    mapping = {
        "<StandardCommand>": navigate_ast_action,
    }
    extras =  [
        StandardCommand,

    ]

def get_rule():
    return PythonVoiceCodingPluginRule, RuleDetails(name="python voice coding plugin",
↪executable="sublime_text", title="Sublime Text")

{% endblock epilogue %}
```

3.2.5 Talon

Το talon είναι ιδιαίτερη περίπτωση γιατί χρειαζόμαστε δύο ξεχωριστά αρχεία:

.py

Στο οποίο θα βρείτε όλες τις ενέργειες κ.λπ.

```
{% extends "base.jinja" %}

{% block prologue %}
from talon import Context, Module, actions, speech_system

mod = Module()

ctx = Context()

{% endblock prologue %}

{% block actions_definitions %}


@mod.action
def python_voice_coding_plugin_navigate_ast(data:dict) -> None:
        ''' navigate the AST'''
        send_sublime("lsp_execute", {
            "command_name": "navigate_ast",
            "session_name": "PythonVoiceCodingPlugin",
            "command_args": [data,"$file_uri"]
        }
    )
{% endblock actions_definitions %}

{% block element_definitions %}

{% for e in elements %}
{{ e[0] | define_element_talon(e[1]) }}
{% endfor %}

{% endblock %}
```

.talon

Στο οποίο εγγράφεται η εντολή φωνής.

```
app.exe: /sublime_text/
-

^<user.python_voice_coding_plugin_StandardCommand>$:
        user.python_voice_coding_plugin_navigate_ast(python_voice_coding_plugin_
↪StandardCommand)
```

3.3 Testing μετ' επιστροφής με hypothesis

3.3.1 Πρότυπο δοκιμής

Δημιουργούμε πρώτα ένα πρότυπο όπου αφαιρούμε όλο τον περιττό κώδικα για την επικοινωνία με τους συντάκτες, φορτώνοντας τις γραμματικές και όλες τις άλλες παρενέργειες.

```
{% extends "dictation-toolbox.jinja" %}



{% block communication %}
{% endblock communication %}
```

3.3.2 pytest fixture

Στη συνέχεια, προχωρήσαμε στη δημιουργία μιας περιόδου σύνδεσης σε ευρεία κλίμακα `pytest.fixture()`[11] που θα μας επιτρέψει να αναλύσουμε εντολές, που δίνονται σε κείμενο αντί για ήχο, στα αντίστοιχα ωφέλιμα φορτία τους. Για να γίνει αυτό

- θα δημιουργήσει τον κώδικα που δημιουργείται από το πρότυπο δοκιμής
- αξιολογείται δυναμικά χρησιμοποιώντας `exec()`[12] και ανάκτηση του ριζικού στοιχείου
- μεταφόρτωση στο `dragonfly.test.ElementTester` με τη μηχανή κειμένου

```
@pytest.fixture(scope="session")
def dragonfly_tester():
    from dragonfly.language.loader import language

    language._language = "en"
    test_code = generate_contents("test")[Path("test.py")]
    data = {}
    c = compile(test_code, "<str>", "exec")
    exec(c, data)

    return ElementTester(data["StandardCommand"], get_engine("text"))
```

Danger: Πρέπει να ζητήσουμε ρητά μια μηχανή "κειμένου", διαφορετικά μπορεί να καταλήξετε να κολλήσετε στο sapi5. το ίδιο μπορεί αν δεν ορίσουμε ρητά τη γλώσσα στα Αγγλικά λόγω κρυφής κλήσης στον κατασκευαστή του *IntegerRef*.

[11] https://docs.pytest.org/en/latest/reference/reference.html#pytest.fixture
[12] https://docs.python.org/3/library/functions.html#exec

3.3.3 Δοκιμή Υποθέσεων

- Δημιουργούμε στιγμιότυπα μοντέλων χρησιμοποιώντας `hypothesis.strategies.from_type()`[13]
- Μετατρέψτε τα σε προφορική μορφή
- Αναλύστε τα χρησιμοποιώντας τον ελεγκτή στοιχείων για να αποκτήσετε ένα λεξικό
- Αναλύουμε αυτό το λεξικό σε ένα pydantic model
- Συγκρίνουμε τις δύο περιπτώσεις μοντέλου είναι ίσες

```
@settings(max_examples=1000, deadline=500)
@given(c=strategies.from_type(StandardCommand))
def test_bidirectional(c, dragonfly_tester):
    spoken_form = into_spoken(c)
    r = StandardCommand.parse_obj(dragonfly_tester.recognize(spoken_form))
    assert r == c, f"{spoken_form} {r} {c}"
```

[13] https://hypothesis.readthedocs.io/en/latest/data.html#hypothesis.strategies.from_type

Κεφάλαιο 4

Πρωτόκολλο διακομιστή γλώσσας (LSP)

4.1 Εισαγωγή

Προκειμένου να επιτραπεί η επικοινωνία μεταξύ του πελάτη μας που τρέχει μέσα στο πρόγραμμα επεξεργασίας και του διακομιστή που εφαρμόζει την επιχειρηματική λογική, ενώ θα μπορούσαμε να βασιστούμε σε ένα προσαρμοσμένο πρωτόκολλο, επιλέξαμε να αξιοποιήσουμε τη γλώσσα τους για τέτοιες εργασίες, δηλαδή το language server protocol.

το language server protocol είναι χτισμένο πάνω από το **json-RPC v2** και επεκτείνεται με **αμφίδρομη επικοινωνία**. Χρησιμοποιώντας ένα σύνολο **τυποποιημένων και αγνωστικών προς την γλώσσα προγραμματισμού μηνυμάτων** σε μια ποικιλία πιθανών μεταφορών όπως stdio ή TCP, ένας **πελάτης γλώσσας** μπορεί να μιλήσει σε έναν ή περισσότερους **διακομιστές γλώσσας που εκτελούνται σε ξεχωριστές διεργασίες**, συγχρονισμός των περιεχομένων των αρχείων που επεξεργάζονται και έκδοση αιτημάτων για **υπηρεσίες IntelliSense** όπως αυτόματη συμπλήρωση, μετάβαση στον ορισμό, βοήθεια υπογραφής κ.λπ. υπό τον όρο φυσικά ότι ο διακομιστής διαφημίζει την **αντίστοιχη ικανότητα** στην αρχική χειραψία.

Φυσικά, δεν μας ενδιαφέρουν οι γλωσσικές υπηρεσίες όπως η αυτόματη συμπλήρωση, επομένως θα πρέπει να επεκτείνουμε το Πρωτόκολλο.

4.2 Προκλήσεις και Αποφάσεις Εφαρμογής

4.2.1 Επιλεγμένη Υλοποίηση LSP

Μία από τις πρώτες αποφάσεις σχεδιασμού που έπρεπε να ληφθούν ήταν ο τρόπος με τον οποίο το έργο μου θα βασιστεί στο πρωτόκολλο διακομιστή γλώσσας σε επίπεδο κώδικα. Αρχικά είχα σκεφτεί τη δημοσίευση κώδικα ως πρόσθετο(plug-in) για το pylsp[14] αλλά επειδή δεν είναι ο μόνος ,ήδη υπάρχων,None δημοφιλής διακομιστής γλώσσας για την python, προτιμώ να δημιουργήσω έναν αυτόνομο χρησιμοποιώντας το pygls[15] , το οποίο παρέχει μια γενική υλοποίηση/SDK LSP.

[14] https://pypi.org/project/pylsp
[15] https://pypi.org/project/pygls

4.2.2 Εντολή Custom Method vs Workspace Execute

Προχωρώντας, πρέπει να αποφασίσουμε ποιον από τους δύο τρόπους θα χρησιμοποιήσουμε για να επεκτείνουμε το πρωτόκολλο διακομιστή γλώσσας

- δημιουργία προσαρμοσμένων μεθόδων στον διακομιστή
- δημιουργήστε μια μέθοδο workspace_executeCommand[16]

Ελήφθη η απόφαση να ακολουθήσει την τελευταία επιλογή για δύο λόγους.

- pygls[17] της στιγμής αυτής της γραφής δεν υποστηρίζει εγγενώς custom μεθόδους διακομιστή. Είναι δυνατό να το αντιμετωπίσετε χρησιμοποιώντας το pygls-tagls-custom[18] το οποίο αναπτύχθηκε για το tagls[19] αλλά φαίνεται ότι έχουμε να κάνουμε μόνο με μία εντολή διακομιστή αυτή τη στιγμή είναι πιο περίπλοκη.
- workspace_executeCommand[20] θα πρέπει να υποστηρίζεται ήδη εγγενώς από τους περισσότερους πελάτες, κάνοντας το έργο ευκολότερο να γίνει σταδιακά υιοθετήσιμο σε περισσότερους συντάκτες

Τούτου λεχθέντος, η τρέχουσα υλοποίηση του pygls[21] μπορεί να είναι λίγο περίεργη, ειδικά στον τρόπο με τον οποίο προσπαθεί να επεξεργαστεί τα ορίσματα για να περάσουμε workspace_executeCommand[22]. Για να παρακάμψουμε αυτό το ζήτημα, δημιουργούμε μια προσαρμοσμένη μέθοδο για την καταχώρηση εντολών που θα διασφαλίζει ότι τα ορίσματα έχουν επικυρωθεί και αποσυμπιεστεί.

```
    def custom_command(
        self, command_name: str
    ) -> Callable[[F], Callable[["PythonVoiceCodingPluginLanguageServer", Any], Any]]:
        """Decorator used to register custom commands.

        Example:
            @ls.command('myCustomCommand')
            def my_cmd(ls, a, b, c):
                pass
        """

        def wrapper(f: F):
            f = validate_arguments(config=dict(arbitrary_types_allowed=True))(f)

            async def function(server: PythonVoiceCodingPluginLanguageServer, args):
                return await f(server, *args)

            self.lsp.fm.command(command_name)(function)
            return f

        return wrapper
```

[16] https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/#workspace_executeCommand
[17] https://pypi.org/project/pygls
[18] https://pypi.org/project/pygls-tagls-custom
[19] https://pypi.org/project/tagls
[20] https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/#workspace_executeCommand
[21] https://pypi.org/project/pygls
[22] https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/#workspace_executeCommand

4.2.3 Διαλογικό εναντίον μονόδρομου

Τώρα όλες αυτές οι πληροφορίες κατάστασης μπορούν να ρέουν μεταξύ του πελάτη και του διακομιστή με δύο τρόπους:

- Αποστολή και λήψη των δεδομένων με μία μόνο κίνηση Μέσω των παραμέτρων αιτήματός τους και του μηνύματος απάντησης.
- Επιτρέψτε έναν μικρό διάλογο, όπου ο διακομιστής ρωτά τον πελάτη για τους τύπους πληροφοριών για τους οποίους έχει δηλώσει υποστήριξη.

Σε γενικές γραμμές, η πρώτη προσέγγιση είναι πιο συνεπής, αλλά για λόγους που έχουν να κάνουν με το να κάνω όσο το δυνατόν χαμηλότερο τον απαραίτητο αριθμό κώδικα για την υποστήριξη των βασικών χαρακτηριστικών και έτσι να επιταχύνω τη διαδικασία μεταφοράς του έργου σε περισσότερους συντάκτες, επέλεξα προσέγγιση που βασίζεται κυρίως στο δεύτερο, με προαιρετική μερική επιστροφή.

Παράδειγμα:

```
:: --> PythonVoiceCodingPlugin workspace/executeCommand(2): {'command': 'navigate_ast',
↪'workDoneToken': 'wd2', 'arguments': [{'op': 'select', 'query': {'description': 'second
↪', 'target': 'argument'}}, 'file:///C:/Users/Username/Documents/GitHub/
↪PythonVoiceCodingPlugin/python_voice_coding_plugin/lsp/server.py']}
:: <-  PythonVoiceCodingPlugin window/showMessage: {'type': 3, 'message': 'Typedocument'}
:: <-- PythonVoiceCodingPlugin selection/getCursors(e168caa7-3ae8-4337-9d61-
↪901d361eb0d8): {'uri': 'file:///C:/Users/Username/Documents/GitHub/
↪PythonVoiceCodingPlugin/python_voice_coding_plugin/lsp/server.py'}
:: >>> PythonVoiceCodingPlugin e168caa7-3ae8-4337-9d61-901d361eb0d8: [{'start': {'line':
↪12, 'character': 18}, 'end': {'line': 12, 'character': 18}}]
:: <-- PythonVoiceCodingPlugin regions/get(a1ea18fc-e98f-49b9-807f-794e2d03de92): {'uri
↪': 'file:///C:/Users/Username/Documents/GitHub/PythonVoiceCodingPlugin/python_voice_
↪coding_plugin/lsp/server.py'}
:: >>> PythonVoiceCodingPlugin a1ea18fc-e98f-49b9-807f-794e2d03de92: {'Green': [],
↪'Origin': [], 'Yellow': [], 'Red': [{'start': {'line': 1, 'character': 1}, 'end': {
↪'line': 2, 'character': 4}}], 'Blue': [], 'Main': [], 'Orange': []}
:: <-- PythonVoiceCodingPlugin workspace/applyEdit(cec3d19e-0488-4e29-b389-
↪40d0afa9f384): {'label': None, 'edit': {'changes': {'file:///C:/Users/Username/
↪Documents/GitHub/PythonVoiceCodingPlugin/python_voice_coding_plugin/lsp/server.py': [{
↪'newText': '#\n', 'range': {'start': {'line': 0, 'character': 0}, 'end': {'line': 0,
↪'character': 0}}}]}}}
:: <-  PythonVoiceCodingPlugin selection/setCursors: {'uri': 'file:///C:/Users/Username/
↪Documents/GitHub/PythonVoiceCodingPlugin/python_voice_coding_plugin/lsp/server.py',
↪'selection': [{'start': {'line': 0, 'character': 2}, 'end': {'line': 1, 'character': 4}
↪}]}
:: <-  PythonVoiceCodingPlugin clipboard/setContents: {'contents': 'something'}
:: <-  PythonVoiceCodingPlugin regions/set: {'regions': {'Red': [{'start': {'line': 0,
↪'character': 2}, 'end': {'line': 1, 'character': 4}}]}, 'uri': 'file:///C:/Users/
↪Username/Documents/GitHub/PythonVoiceCodingPlugin/python_voice_coding_plugin/lsp/
↪server.py'}
:: <<< PythonVoiceCodingPlugin 2: None
```

4.2.4 UTF 16 έναντι UTF-8

Εσωτερικά για την επιχειρηματική μας λογική χρησιμοποιούμε τα `CodeRange`[23] και `CodePosition`[24] που έχει 2 μεγάλες διαφορές με τα range[25] και position[26]

- Οι γραμμές έχουν ευρετήριο 1 έναντι 0
- οι μετατοπίσεις στηλών χρησιμοποιούν σημεία κωδικού Unicode ενώ στο πρωτόκολλο διακομιστή γλώσσας βασίζονται σε UTF 16

οπότε πρέπει να φροντίσουμε να τα μετατρέψουμε κατάλληλα. Η τελευταία έκδοση του πρωτοκόλλου διακομιστή γλώσσας, δηλαδή η 3.17, επιτρέπει στους clients να διαφημίζουν την υποστήριξη UTF-8, αλλά support του UTF 16 εξακολουθεί να είναι υποχρεωτική.

[23] https://libcst.readthedocs.io/en/latest/metadata.html#libcst.metadata.CodeRange
[24] https://libcst.readthedocs.io/en/latest/metadata.html#libcst.metadata.CodePosition
[25] https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/#range
[26] https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/#position

Κεφάλαιο 5

Δαμάζοντας τη Σφίγγα

Η παλιά προσέγγιση για την documentation του έργου βασιζόταν αποκλειστικά στη manuel εγγραφή ζωντανών παραδειγμάτων φωνητικών εντολών, καταγράφοντας την έξοδο οθόνης με το NickeManarin/ScreenToGif[27]. Αυτή η προσέγγιση μπορεί να είναι πολύ χρονοβόρα εάν θέλετε να καταγράψετε περισσότερα από 100 τέτοια παραδείγματα , παρέχοντας παράλληλα ελάχιστη εγγύηση ότι η συμπεριφορά που εμφανίζεται θα ταιριάζει με την τρέχουσα έκδοση του κώδικα. Χρειαζόταν μια πιο αυτοματοποιημένη προσέγγιση και ιδανικά μια που θα μπορούσε επίσης να ενσωματωθεί στις προσπάθειές μου για την αντιμετώπιση της έλλειψης testing infrastructure.

5.1 Με ένα σμπάρο τρία τρυγόνια

Το `doctest`[28] είναι μια ισχυρή standard βιβλιοθήκη που επιτρέπει στους προγραμματιστές να ενσωματώνουν αποσπάσματα κώδικα παρόμοια με τις διαδραστικές συνεδρίες Python (REPL) ακριβώς μέσα στις συμβολοσειρές εγγράφων (docstrings) των συναρτήσεων/κλάσεων/μονάδων τους. Κατά την εκτέλεση, πρόκειται να επαληθεύσει ότι κάθε εντολή/είσοδος θα παράγει την αντίστοιχη έξοδο. Για παράδειγμα:

```
def contains(outer: CodeRange, inner: CodeRange) -> bool:
    r"""
    >>> a=CodeRange(CodePosition(1,2),CodePosition(1,3))
    >>> b=CodeRange(CodePosition(1,2),CodePosition(2,3))
    >>> c=CodeRange(CodePosition(1,2),CodePosition(2,0))
    >>> d=CodeRange(CodePosition(1,1),CodePosition(2,1))
    >>> contains(a,b)
    False
    >>> contains(b,a)
    True
    >>> contains(c,a)
    True
    >>> contains(d,a)
    True
    >>> contains(c,d)
    False
    >>> contains(d,c)
    True
    >>> contains(c,b)
    False
    >>> contains(b,c)
```

(continues on next page)

[27] https://github.com/NickeManarin/ScreenToGif
[28] https://docs.python.org/3/library/doctest.html#module-doctest

(continued from previous page)

```
    True
    >>> contains(b,c)
    True

    """
    second = astuple(outer)
    first = astuple(inner)
    return second[0] <= first[0] <= second[1] and second[0] <= first[1] <= second[1]
```

Αυτά τα παραδείγματα εισόδου-εξόδου εξυπηρετούν

- ως **δοκιμές ακριβώς δίπλα στον κωδικό τους**
- ως **τεκμηρίωση για άλλους προγραμματιστές** που διαβάζουν ή θέλουν να χρησιμοποιήσουν το πακέτο τους

Σε αυτό το έργο επρόκειτο να προχωρήσουμε την ιδέα ένα βήμα παραπέρα και να στοχεύσουμε σε έναν **τρίτο στόχο**: θα χρησιμοποιήσουμε αυτές τις δοκιμές για να δημιουργήσουμε **τεκμηρίωση για τους τελικούς μας χρήστες**. Αρχικά ήθελα αυτά να έχουν τη μορφή κινούμενων gif αλλά θα μπορούσαν να κάνουν στατικές εικόνες ή html αντικείμενα σαν εικόνες .

Εφόσον το έργο μας έχει ήδη την έννοια ενός αντικειμένου `State` που αντιπροσωπεύει όλα όσα γνωρίζει ο διακομιστής γλώσσας για τη διεπαφή χρήστη, θα το αναπτύξουμε σε δύο φάσεις.

- Δημιουργήστε τα κατάλληλα αντικείμενα κατάστασης από τα παραδείγματα doctest.
- Επισημάνετε τα με έναν ιδιαίτερο τρόπο, θυμίζοντας τον τρόπο που θα τα εμφάνιζε ένας επεξεργαστής.

5.2 Απεικονίζοντας το State

5.2.1 Έννοιες Pygments

Το pygments[29] είναι ένα ευρέως χρησιμοποιούμενο και ισχυρό γενικό εργαλείο επισήμανσης σύνταξης που χρησιμοποιείται από το sphinx[30] για να βελτιώσει τα παραδείγματα πηγαίου κώδικα. Η επεξεργασία επισήμανσης αποτελείται από τέσσερα στοιχεία.

- Πρώτον, ένα lexer χωρίζει την πηγή σε tokens, τα οποία είναι κομμάτια πηγαίου κώδικα μαζί με τον τύπο διακριτικού που καθορίζει τι αντιπροσωπεύουν σημασιολογικά.

```
>>> from pygments import lex
>>> import pp
>>> _example_code="def function(x:int)->int\n    return x+1\n"
>>> lexer = PythonLexer()
>>> pp(list(lex(_example_code, lexer)))
[
    (Token.Keyword, 'def'),
    (Token.Text, ' '),
    (Token.Name.Function, 'function'),
    (Token.Punctuation, '('),
    (Token.Name, 'x'),
```

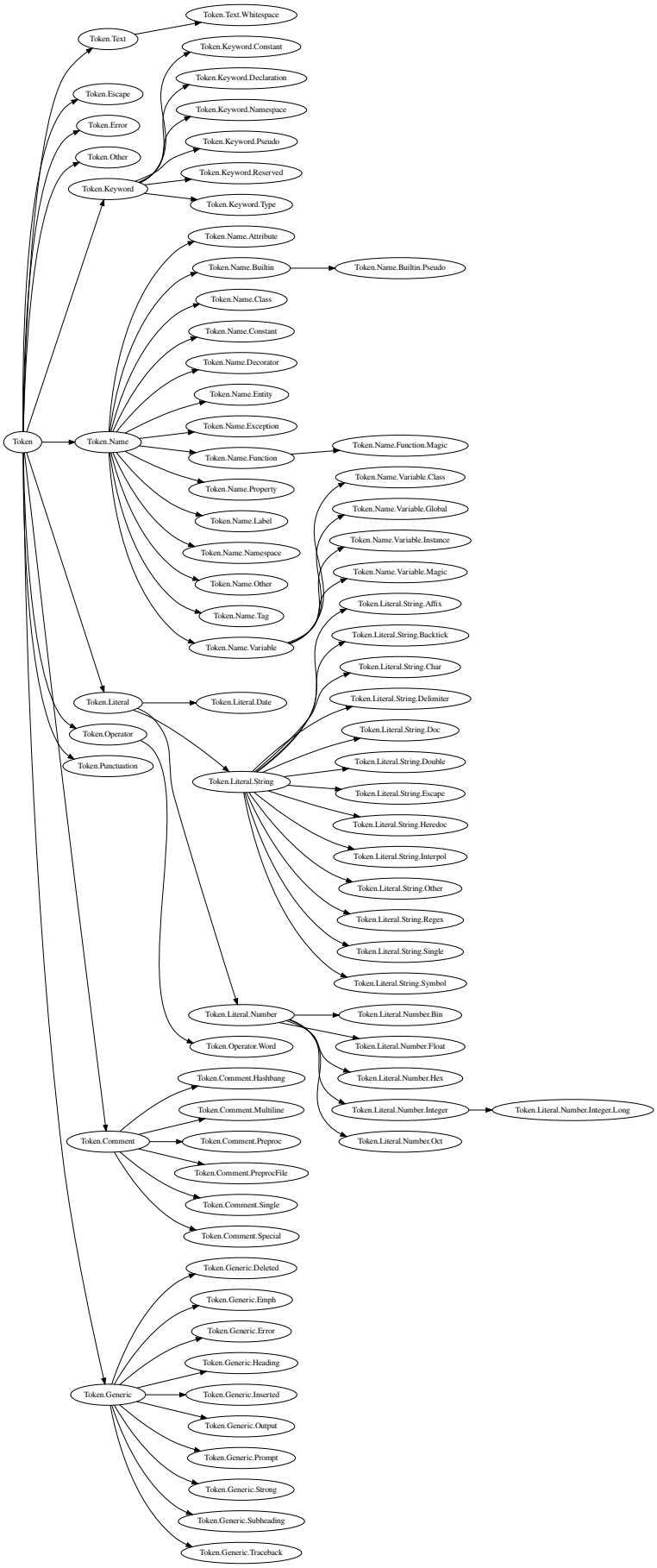
(continues on next page)

[29] https://pypi.org/project/pygments
[30] https://pypi.org/project/sphinx

(continued from previous page)

```
    (Token.Punctuation, ':'),
    (Token.Name.Builtin, 'int'),
    (Token.Punctuation, ')'),
    (Token.Operator, '-'),
    (Token.Operator, '>'),
    (Token.Name.Builtin, 'int'),
    (Token.Text, '\n'),
    (Token.Text, '    '),
    (Token.Keyword, 'return'),
    (Token.Text, ' '),
    (Token.Name, 'x'),
    (Token.Operator, '+'),
    (Token.Literal.Number.Integer, '1'),
    (Token.Text, '\n'),
]
```

- Η ροή tokens μπορεί στη συνέχεια να υποβληθεί σε επεξεργασία από φίλτρα που θα παράγουν μια τροποποιημένη ροή.
- Ένας μορφοποιητής τελικά γράφει αυτή τη ροή σε ένα αρχείο εξόδου, σε μορφή όπως HTML, LaTeX.
- Η διαδικασία μορφοποίησης υπαγορεύεται από ένα στυλ που αντιστοιχίζει κάθε τύπο διακριτικού σε ένα σύνολο ιδιοτήτων που καθορίζουν πώς θα πρέπει να τονιστεί.


Token
Token.Text
Token.Text.Whitespace
Token.Escape
Token.Error
Token.Other
Token.Keyword
Token.Keyword.Constant
Token.Keyword.Declaration
Token.Keyword.Namespace
Token.Keyword.Pseudo
Token.Keyword.Reserved
Token.Keyword.Type
Token.Name
Token.Name.Attribute
Token.Name.Builtin
Token.Name.Builtin.Pseudo
Token.Name.Class
Token.Name.Constant
Token.Name.Decorator
Token.Name.Entity
Token.Name.Exception
Token.Name.Function
Token.Name.Function.Magic
Token.Name.Property
Token.Name.Label
Token.Name.Namespace
Token.Name.Other
Token.Name.Tag
Token.Name.Variable
Token.Name.Variable.Class
Token.Name.Variable.Global
Token.Name.Variable.Instance
Token.Name.Variable.Magic
Token.Literal
Token.Literal.Date
Token.Literal.String
Token.Literal.String.Affix
Token.Literal.String.Backtick
Token.Literal.String.Char
Token.Literal.String.Delimiter
Token.Literal.String.Doc
Token.Literal.String.Double
Token.Literal.String.Escape
Token.Literal.String.Heredoc
Token.Literal.String.Interpol
Token.Literal.String.Other
Token.Literal.String.Regex
Token.Literal.String.Single
Token.Literal.String.Symbol
Token.Literal.Number
Token.Literal.Number.Bin
Token.Literal.Number.Float
Token.Literal.Number.Hex
Token.Literal.Number.Integer
Token.Literal.Number.Integer.Long
Token.Literal.Number.Oct
Token.Operator
Token.Operator.Word
Token.Punctuation
Token.Comment
Token.Comment.Hashbang
Token.Comment.Multiline
Token.Comment.Preproc
Token.Comment.PreprocFile
Token.Comment.Single
Token.Comment.Special
Token.Generic
Token.Generic.Deleted
Token.Generic.Emph
Token.Generic.Error
Token.Generic.Heading
Token.Generic.Inserted
Token.Generic.Output
Token.Generic.Prompt
Token.Generic.Strong
Token.Generic.Subheading
Token.Generic.Traceback

5.2.2 Τροποποίηση του στυλ

Αρχικά, πρέπει να τροποποιήσουμε το στυλ ώστε να επιτρέπονται περιοχές με προσαρμοσμένο φόντο και περίγραμμα χρωματιστά. Στην ιδανική περίπτωση, για αυτές τις επισημασμένες περιοχές θα θέλαμε να αλλάξουμε μόνο αυτές τις συγκεκριμένες ιδιότητες χωρίς να παρεμβαίνει στα υπόλοιπα όπως το χρώμα των γραμμάτων. Ένας εύκολος τρόπος για να το πετύχετε είναι απλά επεκτείνετε τις τυπικές ιεραρχίες token που φαίνονται παραπάνω. Συγκεκριμένα, πρόκειται να προσαρμόσουμε το επιλεγμένο στυλ έτσι ώστε για κάθε τύπο διακριτικού όπως *Token.Text* θα υπάρχουν πρόσθετοι τύποι token *Token.Text.Red*, *Token.Text.Green* κ.λπ. τις ιδιότητες βάσης αλλά υπερισχύουν του χρώματος φόντου και του περιγράμματος.

5.2.3 Αλλαγή των Tokens

Στη συνέχεια, για να δημιουργήσουμε αυτά τα ειδικά tokens, μπορούμε απλά να δημιουργήσουμε ένα φίλτρο ότι αυτό θα αλλάξει τον τύπο token όλων των κουπονιών που βρέθηκαν εντός των επιθυμητών περιοχών.

Επιγραμματικά, το παρακάτω

```
.. state_code_block::
    :selection:
        - [[1,4],[1,12]]
        - [[4,0],[4,5]]
    :regions:
        Red:
            - [[1,4],[1,12]]
            - [[4,6],[4,10]]


    from typing import Iterator

    # This is an example
    class Math:
        @staticmethod
        def fib(n: int) -> Iterator[int]:
            """ Fibonacci series up to n """
            a, b = 0, 1
            while a < n:
                yield a
                a, b = b, a + b

    result = sum(Math.fib(42))
    print("The answer is {}".format(result))
```

παράγει

5.3 Συνοψίζοντας τα DocTests

Ενώ αρχικά εξετάστηκαν μερικές πιο περίπλοκες ιδέες, τελικά συμβιβάστηκα με κάτι σχετικά απλό: εκτελέστε τα doctests, επιθεωρήστε τον καθορισμένο χώρο ονομάτων (namespace) για ειδικές μεταβλητές όπως *results*, *origins* που συνήθως θα είναι λίστες/λίστες λιστών/λεξικά λιστών κ.λπ. που περιέχουν αντικείμενα `libcst.CSTNode`[31]. Χάρη στην ευελιξία και τις δυνατότητες μετ' επιστροφής του `libcst` μπορούμε εύκολα να δημιουργήσουμε πηγαίο κώδικα που περιέχει αυτούς τους κόμβους *origins* ανώτατου επιπέδου καθώς και να υπολογίσουμε θέσεις για αυτούς τους κόμβους *αποτελέσματα* εντός αυτού.

Note: Για λόγους ευκολίας, θα χρησιμοποιήσουμε το xdoctest[32] αντί του ενσωματωμένου `doctest`[33] για αυτήν την εργασία

---

[31] https://libcst.readthedocs.io/en/latest/nodes.html#libcst.CSTNode
[32] https://pypi.org/project/xdoctest
[33] https://docs.python.org/3/library/doctest.html#module-doctest

Κείμενο στα αγγλικά

Chapter 6

Project Reorganization And Architecture

6.1 Why A Rewrite From Scratch?

The original mpourmpoulis/PythonVoiceCodingPlugin[34] project begun with me hacking around my SublimeText editor and my caster installation during the summer of 2019 , would eventually be released in November of that year and development would continue until late 2020. While it was more than good enough for my needs but it was suffering from a variety of issues that I to have wanted to address for a long time.

- getting features out as quickly as possible was a priority because I desperately needed them so trade-offs had to be made in its initial design between speed of development and code quality/extensibility/testability
- the builtin `ast`[35] module was causing me some issues that I would not have to face with a proper concrete syntax tree implementation like licst[36]
- it was running only inside SublimeText's embedded python , which at the time was an obsolete 3.3 , even though the newest version of sublime text 4 does offer a more modern 3.8 runtime.
- with hindsight various bits of the approach to the grammar feel overengineered and unsystematic

But arguably the main goal of this rewrite attempt was to make the project cross-platform

- supporting multiple editors
- as well as multiple programming by voice frameworks, not just dictation-toolbox/Caster[37]

In order to meet these demands, a drastic reapproach was required.

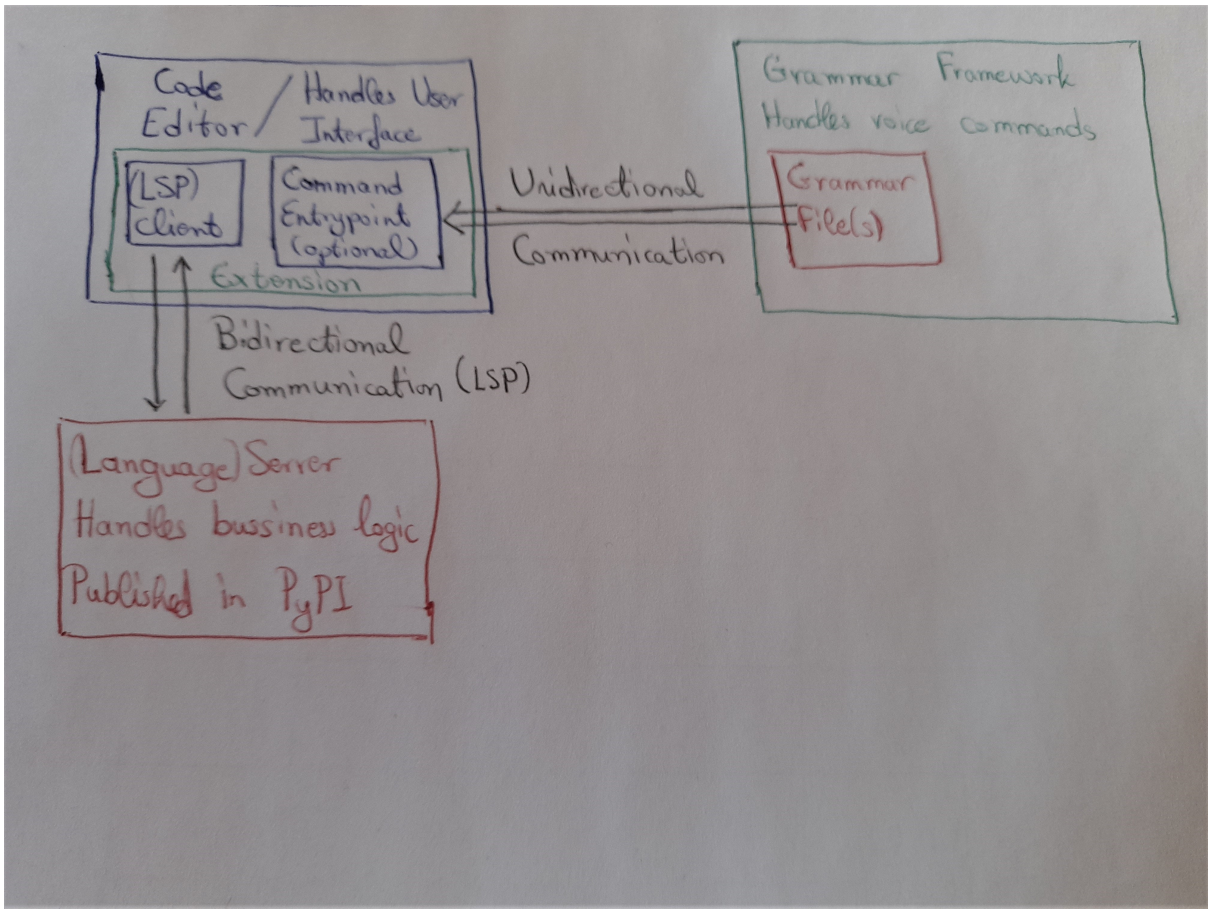
6.2 System Components

The central idea behind these rewrite is to pull all the business logic outside of the sublime plugin, distribute it separately and make it available for use in other editors. The project is now reorganized in three components,each running in a different environment.

- Firstly, we have the grammar files which are loaded by the voice programming framework. They contain all of the voice command definitions as well as code needed for sending payloads and triggering actions on the currently active editor, when these commands are recognized.

[34] https://github.com/mpourmpoulis/PythonVoiceCodingPlugin
[35] https://docs.python.org/3/library/ast.html#module-ast
[36] https://pypi.org/project/licst
[37] https://github.com/dictation-toolbox/Caster

- Secondly, for each supported editor there should be an extension that is going to receive those payloads, forward them to the server process in an appropriate manner and take care of all the user interface bits needed like changing the cursor selection , highlighting pieces of code etc. The extension code should be kept at a minimum as it will have to be reimplemented for each editor using its native API and programming language and should thus contain no business logic.
- Finally, we have the server, where all the business logic is located and all the pure computations are preformed. As we shall see, the server is going to build upon the language server protocol and would be run in its own process outside of the editor but managed by the extension which itself is going to act as a language client. The server is implemented as a python package and published in PyPI as python_voice_coding_plugin[38].



6.3 Related projects

For the interested reader wishing to learn more about programming by voice in general a quick look at [1] , [2] , [3] , [4] might be interesting. Here we are going to only talk about editor integrations.

- anonfunc/intellij-voicecode[39] , open source, support for Jet Brains and talon. It is based on Jet brains Program Structure Interface, provides basic structural navigation for multiple languages and exposes various jet brains APIs via TCP server.
- cursorless-dev/cursorless[40] , open source, support for VsCode and talon. It is based on tree-

[38] https://pypi.org/project/python_voice_coding_plugin
[39] https://github.com/anonfunc/intellij-voicecode
[40] https://github.com/cursorless-dev/cursorless

sitter/tree-sitter[41] and provides powerful editing for multiple languages. Compared to our and to the projects in this list that effectively take a "top-down" approach letting the user describe what he wants to select by the kind of the region and some positional information, cursorless instead takes a bottom up approach. In particular, for each source token it annotates one or more of its letters with hats of various colors and shapes , which can be used to uniquely identify it in voice commands, and then expands the selection upwards until a region of the proper kind is found. These of course comes at the cost of being reliant on vscode's powerful user interface but there are efforts to port it in other editors as well.

- serenadeai/serenade[42] , previously closed source but relisting the open in June 2022. Unlike the other projects, it is an entire ecosystem with its own speech recognition. It supports several editors and programming languages and apart from syntactical navigation also allows for inserting code formatted by machine learning models.

[41] https://github.com/tree-sitter/tree-sitter
[42] https://github.com/serenadeai/serenade

Chapter 7

Grammar Deployment

As the number of components in the system is driving up its complexity, keeping the installation process as simple as possible from a user perspective and finding an effective way to distribute the grammar files needed was a major goal of rewrite.

In general, there exists a variety of perfectly reasonable ways to achieve this goal but at this stage of the project, the method chosen preferably should not require manual user action and should scale to support multiple voice programming frameworks as well as multiple code editors with as little footprint as possible.

We are going to go through with the ideas considered and finally present the one eventually accepted.

7.1 Ideas Considered

7.1.1 Git Repository

This is arguably the most straightforward approach and for good reasons:

- Firstly, people are already familiar with it as it is very widely used in the ecosystem by project such as dictation-toolbox/Caster[43], knausj85/knausj_talon[44], cursorless-dev/cursorless-talon[45], mrob95/mathfly-talon[46] and mrob95/mathfly[47]
- Secondly,because it is a version control system, it makes it easier for people to customize to their own needs or even contribute back to the project, allowing for collaboration with community

However

- while not really the end of the world, it typically still requires the user to undertake some manual steps for installing or updating the grammar or some extra code in order to automate it for them
- It can get problematic if you want to support multiple frameworks. For Example, you probably need multiple such repositories, one for each framework, in order to avoid users downloading grammar files for a different system in their user directory. There are of course workarounds such as having a single monorepo and then using a continuous delivery system in order to synchronize the rest but things are getting a little bit complex.

[43] https://github.com/dictation-toolbox/Caster
[44] https://github.com/knausj85/knausj_talon
[45] https://github.com/cursorless-dev/cursorless-talon
[46] https://github.com/mrob95/mathfly-talon
[47] https://github.com/mrob95/mathfly

7.1.2 Automatic Deployment From Editor Plugin

Another natural idea might be to sheave the grammars along with the editor extensions and let them handle the installation/upgrade automatically, which is somewhat similar to the old approach I was taking. Unfortunately, one major limitation is that it involves a lot of code duplication between all the various editor plug-ins,which is something I want to avoid.

7.1.3 Separate PyPI Package

Publishing the grammar as package in the **pypi** registry the same way the the backend language server server is ,was also an idea considered it felt natural and pythonic and using sub packages with conditional imports could address the multiple frameworks issue.

Unfortunately:

- at the moment, none of the frameworks we are interested in has native support for loading grammars from packages,
- additional logic or manual user action is still needed for updating the packages.

7.2 Current Approach: deploying from the language server

The current approach is to let grammar deployment get handled by the language server executable itself. In particular, upon launching the language server, it will (generate on-the-fly and) install grammar files to their corresponding file system locations. Using this technique

- Installation/upgrades are as seamless as possible from a user perspective
- we do not really need any extra code in our language clients, keeping them thin and faster to port the project to multiple editors
- We can even install grammars for multiple frameworks simultaneously

The grammar generation and deployment process is controlled by a series of commandline options/environmental variables and if the default behavior is not satisfactory, the user simply has to edit the extension configuration to adjust them.

7.2.1 python_voice_coding_plugin

python_voice_coding_plugin.

```
python_voice_coding_plugin [OPTIONS]
```

Options

--version
: Show the version and exit.

--caster-user-dir <caster_user_dir>

--dragonfly-user-dir <dragonfly_user_dir>

--talon-user-dir <talon_user_dir>

--navigation-grammar-generators <navigation_grammar_generators>

Options
: caster | dragonfly | talon

--stdio

--ws

--tcp

--host <host>

Default
: 127.0.0.1

--port <port>

Environment variables

CASTER_USER_DIR

: Provide a default for *--caster-user-dir*

DRAGONFLY_USER_DIR

: Provide a default for *--dragonfly-user-dir*

TALON_USER_DIR

: Provide a default for *--talon-user-dir*

NAVIGATION_GRAMMAR_GENERATORS

: Provide a default for *--navigation-grammar-generators*

Chapter 8

Generating The Grammar

8.1 Motivation

The new version of the project marks a major paradigm shift not only in their way the grammar files are getting deployed but also how they are written,switching from handwritten to generated one's. there are a variety of reasons why these decisions was made

- firstly the manual version can be **time-consuming, more error-prone and does not adhere to the principle of *DRY* (or Do not repeat yourself)** . In particular, one way or another the language server is already going to contain code needed to represent/validate all the payloads we can send from the commands the system recognizes. writing the grammar by hand means we have to re-implement this representation in the grammar file, with little guarantee that it is synchronized to the one present in the language server.

Note: not to mention that the latter might also be partially generated as well

---

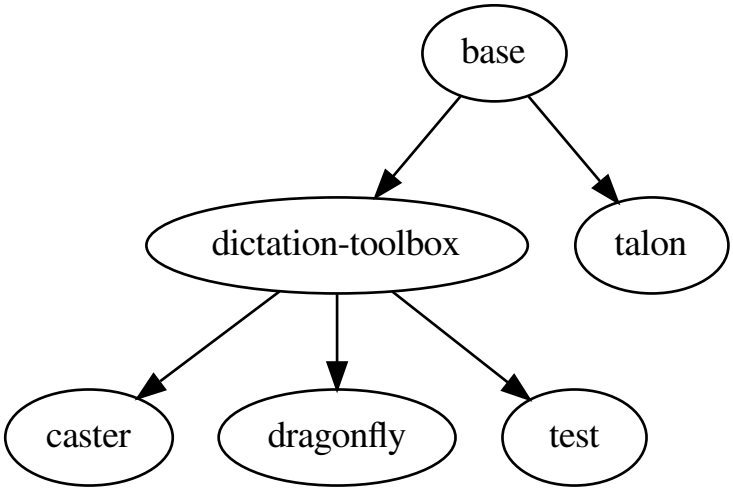
- Secondly, the approach has the potential to **scale better if you want to support multiple voice programming frameworks** . With a manual approach in order to define **m** spoken constructs for **n** frameworks you need a total of **O(mn)** implementations in the grammar files. instead with an automated one, you need **O(m)** implementations for defining the elements in your backend code (which you probably couldn't get away without either way) and **O(n)** implementations for the grammar generators. Furthermore, it makes reusing code bits across frameworks much easier, as we shall see
- Finally, generating the grammar has the additional advantage of making it relatively easy to switch implementation. For example, if you are now using `Dictlist` you want to change it into `Choice` or you want to wrap it in a `RuleWrap` it becomes trivial

8.2 Jinja templates

As hinted upon earlier, there is a nontrivial potential for code reusability across frameworks. in particular

- whether we are talking about talon or dragonfly based grammars, we are going to some code in order to handle the low level communication with our editors,which can easily be shared
- All for dictation-toolbox grammars are largely similar. Whether we are talking about standalone dragonfly/caster/breathe the old define spoken elements and actions pretty much the same way only differ in their way the final grammar is constructed and loaded, as well as the imports needed

in order to take advantage of these reusability we are going to rely on jinja2[48] , which is a powerful template rendering framework for python, and it's template inheritance features, in order to build a template hierarchy



8.2.1 Base

In the base template we define five kinds of blocks that each framework can override or inherit from its parent and we implement the common communication section

- framework specific prologue
- communication
- actions definitions
- element definitions
- framework specific epilogue

```
{% block prologue %}
{% endblock prologue %}

{% block communication %}
import json
import os
import platform
import subprocess

def validate_subl():
    if platform.system() != 'Windows':
        return "subl"
    windows_candidates=[
        "subl",
```

(continues on next page)

[48] https://pypi.org/project/jinja2

(continued from previous page)

```
        r"C:\Program Files\Sublime Text\subl",
        r"C:\Program Files\Sublime Text 3\subl",
        r"C:\Program Files (x86)\Sublime Text 3\subl",
        r"C:\Program Files (x86)\Sublime Text\subl",
    ]
    for  candidate in windows_candidates:
        try:
            subprocess.check_call([candidate,"-h"],stdout=subprocess.PIPE,
↪stderr=subprocess.PIPE)
            return candidate
        except Exception as e:
            continue
    else:
        raise ValueError("subl not found. Please follow instructions https://www.
↪sublimetext.com/docs/command_line.html")

subl=validate_subl()


def send_sublime(c,data):
    print(c + " " + json.dumps(data))
    command=[subl,"-b", "--command",c + " " + json.dumps(data)]
    # Adopted from https://dragonfly2.readthedocs.io/en/latest/_modules/dragonfly/
↪actions/action_cmd.html#RunCommand
    import subprocess
    # Suppress showing the new CMD.exe window on Windows.
    startupinfo = None
    if os.name == 'nt':
        startupinfo = subprocess.STARTUPINFO()
        startupinfo.dwFlags |= subprocess.STARTF_USESHOWWINDOW


    ret_code = subprocess.call(command, startupinfo=startupinfo)
    return ret_code




{% endblock communication %}

{%block actions_definitions %}
{% endblock %}

{% block element_definitions %}
{% endblock %}

{% block epilogue %}
{% endblock %}
```

8.2.2 Dictation toolbox

In the dictation tool box template

```
{% extends "base.jinja" %}


{% block prologue %}
from dragonfly import *
{% endblock prologue %}

{% block actions_definitions %}
@Function
def navigate_ast_action(StandardCommand):
    print(StandardCommand)
    send_sublime("lsp_execute", {
            "command_name": "navigate_ast",
            "session_name": "PythonVoiceCodingPlugin",
            "command_args": [StandardCommand,"$file_uri"]
        }
    )
{% endblock actions_definitions %}


{% block element_definitions %}
{% for e in elements %}
{{ e[0] | define_element_dragonfly(e[1]) }}
{% endfor %}
{% endblock %}
```

8.2.3 Dragonfly

```
{% extends "dictation-toolbox.jinja" %}




{% block epilogue %}

class PythonVoiceCodingPluginRule(MappingRule):
    mapping = {
        "<StandardCommand>": navigate_ast_action,
    }
    extras =  [
        StandardCommand,

    ]

context = AppContext(executable="sublime_text")
grammar = Grammar("PythonVoiceCodingPlugin", context=context)
rule = PythonVoiceCodingPluginRule(name="python voice coding plugin")
grammar.add_rule(rule)
grammar.load()

{% endblock epilogue %}
```

8.2.4 Caster

```
{% extends "dictation-toolbox.jinja" %}

{% block prologue %}
{{ super() }}
from castervoice.lib.merge.state.short import R
from castervoice.lib.ctrl.mgr.rule_details import RuleDetails
{% endblock prologue %}


{% block epilogue %}

class PythonVoiceCodingPluginRule(MappingRule):
    mapping = {
        "<StandardCommand>": navigate_ast_action,
    }
    extras =  [
        StandardCommand,

    ]

def get_rule():
    return PythonVoiceCodingPluginRule, RuleDetails(name="python voice coding plugin",
↪executable="sublime_text", title="Sublime Text")

{% endblock epilogue %}
```

8.2.5 Talon

talon is a little bit of a special case because we need two separate files

.py

where would you find all of the elements actions etc.

```
{% extends "base.jinja" %}

{% block prologue %}
from talon import Context, Module, actions, speech_system

mod = Module()

ctx = Context()

{% endblock prologue %}

{% block actions_definitions %}


@mod.action
def python_voice_coding_plugin_navigate_ast(data:dict) -> None:
        ''' navigate the AST'''
```

(continues on next page)

(continued from previous page)

```
        send_sublime("lsp_execute", {
            "command_name": "navigate_ast",
            "session_name": "PythonVoiceCodingPlugin",
            "command_args": [data,"$file_uri"]
        }
    )
{% endblock actions_definitions %}

{% block element_definitions %}

{% for e in elements %}
{{ e[0] | define_element_talon(e[1]) }}
{% endfor %}

{% endblock %}
```

.talon

where we actually register the voice command

```
app.exe: /sublime_text/
-

^<user.python_voice_coding_plugin_StandardCommand>$:
        user.python_voice_coding_plugin_navigate_ast(python_voice_coding_plugin_
↪StandardCommand)
```

8.3 Element Definitions

8.3.1 Run-time Element Definition vs Static Code Generation

Regarding whether this grammar generation process should be happening at runtime or statically ahead of time, or in other words whether the grammar file should contain the code needed to produce all of the elements in a dynamic fashion or whether these elements should be already statically rolled out, I have gone with the latter option because it makes things more to debugable.

In particular, for every desirable spoken construct we define the corresponding native Python type in the backend. Then for each generator, we define functions that can spit out grammar code for those native types. We bridge our approach with the aforementioned Jinja templates with the help of the following jinja filters

```
def generate_contents(generator: str) -> Mapping[Path, str]:
    env = Environment(
        loader=PackageLoader("python_voice_coding_plugin.grammar_generation"),
        autoescape=select_autoescape(),
    )
    env.filters["define_element_dragonfly"] = define_element_dragonfly
    env.filters["define_element_talon"] = define_element_talon
    return {
        k: env.get_template(v).render(elements=elements, generator=generator)
```

(continues on next page)

(continued from previous page)

```
        for k, v in generators_to_templates[generator].items()
    }
```

8.3.2 Native Representation

We will define our elements in such a way that we get json we want.

For the purposes of modeling elements that are simply a *choice between phrases* we can very conveniently create a special subclass of `enum.Enum`[49] and `str`[50]

class SpokenEnumeration

A base class for all spoken enumerations

For example

```
class Ordinal(SpokenEnumeration):
    FIRST = auto()
    """ the first one """
    SECOND = auto()
    THIRD = auto()
    FOURTH = auto()
    FIFTH = auto()
    SIXTH = auto()
    LAST = auto()
```

in order to allow for more complicated structures that can reference other elements to allow for some sort of hierarchy,dictionaries come in handy in the json output and we simply resort to using pydantic models

class Model

the base model we are going to use for all our models.

By default it is going to be immutable

class Config

faux_immutability = True

arbitrary_types_allowed = True

underscore_attrs_are_private = True

taking advantage of Python's type hints to reuse elements

Warning: Excluding trivial things like `int`[51] we avoid in-line definitions because it can make life significantly more complicated. Anything appearing as a type unit should correspond to an already defined and referenceable element

[49] https://docs.python.org/3/library/enum.html#enum.Enum
[50] https://docs.python.org/3/library/stdtypes.html#str
[51] https://docs.python.org/3/library/functions.html#int

8.3.3 Dragonfly

define_enumeration(*enumeration*, *name=None*)

in order to model choices we convert enumerations using `DictList`[52]

Examples

```
>>> print(define_element(types.Ordinal))
Ordinal = DictList("Ordinal",{
    "first": "first",
    "second": "second",
    "third": "third",
    "fourth": "fourth",
    "fifth": "fifth",
    "sixth": "sixth",
    "last": "last"
})
>>> print(define_element(types.ArgumentTargets))
ArgumentTargets = DictList("ArgumentTargets",{
    "argument": "argument"
})
```

Parameters

- **enumeration** (*Type*[53]*[*`SpokenEnumeration`*]*) –
- **name** (*Optional*[54]*[*`str`[55]*]*) –

Return type

str[56]

define_model(*element*, *name=None*)

in order to represent dictionaries, we can very handily use `Compound`

```
>>> print(define_element(types.VerticalDescription))
VerticalDescription = RuleWrap(
    name="VerticalDescription",
    element=Compound(
        spec="<direction> [<step>]",
        extras=[
            DictListRef("direction",VerticalDirection),
            IntegerRef("step",0,10,default=1)
        ],
        value_func=lambda node, extras: {k:v for k,v in extras.items() if k !="_
↪node"}
    )
)
```

[52] https://dragonfly2.readthedocs.io/en/latest/lists.html#dragonfly.grammar.list.DictList
[53] https://docs.python.org/3/library/typing.html#typing.Type
[54] https://docs.python.org/3/library/typing.html#typing.Optional
[55] https://docs.python.org/3/library/stdtypes.html#str
[56] https://docs.python.org/3/library/stdtypes.html#str

```
>>> print(define_element(types.FullRange))
FullRange = RuleWrap(
    name="FullRange",
    element=Compound(
        spec="<lower> until <upper>",
        extras=[
            IntegerRef("lower",0,10),
            IntegerRef("upper",0,10)
        ],
        value_func=lambda node, extras: {k:v for k,v in extras.items() if k !="_
↪node"}
    )
)
```

Command

```
>>> import inspect
>>> import pp
>>> print(inspect.getsource(types.StandardCommand))
class StandardCommand(Model):
    op: Operation = Operation.SELECT
    query: Query

>>> print(define_element(types.StandardCommand))
StandardCommand = RuleWrap(
    name="StandardCommand",
    element=Compound(
        spec="[<op>] <query>",
        extras=[
            DictListRef("op",Operation,default="select"),
            RuleRef(Query.rule,"query")
        ],
        value_func=lambda node, extras: {k:v for k,v in extras.items() if k !=
↪"_node"}
    )
)

>>> dragonfly_tester = getfixture("dragonfly_tester")
>>> pp(dragonfly_tester.recognize("up attribute root two until three"))
{
    'op': 'select',
    'query': {
        'description': {'direction': 'up', 'step': 1},
        'target': 'attribute_root',
        'index': {'lower': 2, 'upper': 3},
    },
}
```

Parameters

- **element** (*Type*[57]*[*Model*]*) –
- **name** (*Optional*[58]*[*str*[59]*]*) –

Return type

str[60]

define_alternative(*element*, *name=None*)

In order to define a `Union` we naturally use `Alternative`[61] which we then wrap inside `RuleWrap`[62]

```
>>> types.Target
typing.Union[python_voice_coding_plugin.types.spoken.targets.ArgumentTargets,
↪python_voice_coding_plugin.types.spoken.targets.AttributeTargets]
>>> print(define_element(types.Target,"Target"))
Target = RuleWrap("Target",Alternative(children=[
            DictListRef("ArgumentTargets",ArgumentTargets),
            DictListRef("AttributeTargets",AttributeTargets)
        ]
    )
)

>>> types.Description
typing.Union[python_voice_coding_plugin.types.spoken.descriptions.Ordinal, python_
↪voice_coding_plugin.types.spoken.descriptions.VerticalDescription]
>>> print(define_element(types.Description,"Description"))
Description = RuleWrap("Description",Alternative(children=[
            DictListRef("Ordinal",Ordinal),
            RuleRef(VerticalDescription.rule,"VerticalDescription")
        ]
    )
)

>>> types.SubIndex
typing.Union[int, typing.Sequence[int], python_voice_coding_plugin.types.spoken.
↪subindexing.FullRange, python_voice_coding_plugin.types.spoken.subindexing.
↪AfterRange, python_voice_coding_plugin.types.spoken.subindexing.UntilRange]
>>> print(define_element(types.SubIndex,"SubIndex"))
SubIndex = RuleWrap("SubIndex",Alternative(children=[
            IntegerRef("SingleIndex",0,10),
            MultipleIndeces,
            RuleRef(FullRange.rule,"FullRange"),
            RuleRef(AfterRange.rule,"AfterRange"),
            RuleRef(UntilRange.rule,"UntilRange")
        ]
    )
)
```

Parameters

name (*Optional*[63]*[str*[64]*]*) –

Return type

str[65]

[57] https://docs.python.org/3/library/typing.html#typing.Type
[58] https://docs.python.org/3/library/typing.html#typing.Optional
[59] https://docs.python.org/3/library/stdtypes.html#str
[60] https://docs.python.org/3/library/stdtypes.html#str
[61] https://dragonfly2.readthedocs.io/en/latest/elements.html#dragonfly.grammar.elements_basic.Alternative
[62] https://dragonfly2.readthedocs.io/en/latest/elements.html#dragonfly.grammar.elements_basic.RuleWrap
[63] https://docs.python.org/3/library/typing.html#typing.Optional
[64] https://docs.python.org/3/library/stdtypes.html#str
[65] https://docs.python.org/3/library/stdtypes.html#str

8.3.4 Talon

define_enumeration(*element*, *name=None*)

in order to model choices we must first forward declare them via `talon.Module.list()`[66] and then provide for values using `talon.Context.lists`[67]

Examples

```
>>> print(define_element(types.Ordinal))
mod.list("python_voice_coding_plugin_Ordinal")
ctx.lists["user.python_voice_coding_plugin_Ordinal"] ={
    "first": "first",
    "second": "second",
    "third": "third",
    "fourth": "fourth",
    "fifth": "fifth",
    "sixth": "sixth",
    "last": "last"
}
>>> print(define_element(types.ArgumentTargets))
mod.list("python_voice_coding_plugin_ArgumentTargets")
ctx.lists["user.python_voice_coding_plugin_ArgumentTargets"] ={
    "argument": "argument"
}
```

Parameters

- **element** (*Type*[68]*[SpokenEnumeration]*) –
- **name** (*Optional*[69]*[str*[70]*]*) –

Return type

str[71]

define_model(*element*, *name=None*)

in order to represent dictionaries, we can very handily use `talon.Module.capture()`[72]

```
>>> print(define_element(types.VerticalDescription))
@mod.capture(
    rule="{user.python_voice_coding_plugin_VerticalDirection} [<number>]"
)
def python_voice_coding_plugin_VerticalDescription(m) ->str:
    return {
        "direction" : m.python_voice_coding_plugin_VerticalDirection ,
        "step" : getattr(m,"number",1)
    }
```

we need to make sure when multiple appearances of the same extra

[66] https://talonvoice.com/docs/index.html#document-api#talon.Module.list
[67] https://talonvoice.com/docs/index.html#document-api#talon.Context.lists
[68] https://docs.python.org/3/library/typing.html#typing.Type
[69] https://docs.python.org/3/library/typing.html#typing.Optional
[70] https://docs.python.org/3/library/stdtypes.html#str
[71] https://docs.python.org/3/library/stdtypes.html#str

```
>>> print(define_element(types.FullRange))
@mod.capture(
    rule="<number> until <number>"
)
def python_voice_coding_plugin_FullRange(m) ->str:
    return {
        "lower" : m.number_1 ,
        "upper" : m.number_2
    }

>>> print(define_element(types.NestedQuery))
@mod.capture(
    rule="<user.python_voice_coding_plugin_SimpleQuery> <user.python_voice_coding_
↪plugin_SimpleQuery>"
)
def python_voice_coding_plugin_NestedQuery(m) ->str:
    return {
        "outer" : m.python_voice_coding_plugin_SimpleQuery_1 ,
        "inner" : m.python_voice_coding_plugin_SimpleQuery_2
    }
```

Command

```
>>> import inspect
>>> import pp
>>> print(inspect.getsource(types.StandardCommand))
class StandardCommand(Model):
    op: Operation = Operation.SELECT
    query: Query

>>> print(define_element(types.StandardCommand))
@mod.capture(
    rule="[{user.python_voice_coding_plugin_Operation}] <user.python_voice_
↪coding_plugin_Query>"
)
def python_voice_coding_plugin_StandardCommand(m) ->str:
    return {
        "op" : getattr(m,"python_voice_coding_plugin_Operation","select") ,
        "query" : m.python_voice_coding_plugin_Query
    }
```

Parameters

- **element** (*Type*[72][Model]) –
- **name** (*Optional*[73][*str*[74]]) –

Return type

str[75]

[72] https://talonvoice.com/docs/index.html#document-api#talon.Module.capture
[73] https://docs.python.org/3/library/typing.html#typing.Type
[74] https://docs.python.org/3/library/typing.html#typing.Optional
[75] https://docs.python.org/3/library/stdtypes.html#str
[76] https://docs.python.org/3/library/stdtypes.html#str

define_alternative(*element*, *name=None*)

In order to define a `Union` we can simply use a `talon.Module.capture()`[77] where we simply make a choice of full of the possibilities and return the first (and only) value captured

```
>>> types.Target
typing.Union[python_voice_coding_plugin.types.spoken.targets.ArgumentTargets,
↪python_voice_coding_plugin.types.spoken.targets.AttributeTargets]
>>> print(define_element(types.Target,"Target"))
@mod.capture(
    rule="{user.python_voice_coding_plugin_ArgumentTargets} | {user.python_voice_
↪coding_plugin_AttributeTargets}"
)
def python_voice_coding_plugin_Target(m) ->str:
    return m[0]

>>> types.Description
typing.Union[python_voice_coding_plugin.types.spoken.descriptions.Ordinal, python_
↪voice_coding_plugin.types.spoken.descriptions.VerticalDescription]
>>> print(define_element(types.Description,"Description"))
@mod.capture(
    rule="{user.python_voice_coding_plugin_Ordinal} | <user.python_voice_coding_
↪plugin_VerticalDescription>"
)
def python_voice_coding_plugin_Description(m) ->str:
    return m[0]

>>> types.SubIndex
typing.Union[int, typing.Sequence[int], python_voice_coding_plugin.types.spoken.
↪subindexing.FullRange, python_voice_coding_plugin.types.spoken.subindexing.
↪AfterRange, python_voice_coding_plugin.types.spoken.subindexing.UntilRange]
>>> print(define_element(types.SubIndex,"SubIndex"))
@mod.capture(
    rule="<number> | <user.python_voice_coding_plugin_MultipleIndeces> | <user.
↪python_voice_coding_plugin_FullRange> | <user.python_voice_coding_plugin_
↪AfterRange> | <user.python_voice_coding_plugin_UntilRange>"
)
def python_voice_coding_plugin_SubIndex(m) ->str:
    return m[0]
```

Parameters

name (*Optional*[78]*[str*[79]*]*) –

Return type

str[80]

[77] https://talonvoice.com/docs/index.html#document-api#talon.Module.capture
[78] https://docs.python.org/3/library/typing.html#typing.Optional
[79] https://docs.python.org/3/library/stdtypes.html#str
[80] https://docs.python.org/3/library/stdtypes.html#str

8.4 Roundtrip testing with hypothesis

8.4.1 Test Template

We first create a template where we remove all of unneeded code for the communication with the editors, loading the grammars and all the other side effects.

```
{% extends "dictation-toolbox.jinja" %}



{% block communication %}
{% endblock communication %}
```

8.4.2 pytest fixture

We then proceeded to create a session wide `pytest.fixture()`[81] that is going to allow us to parse commands ,given in Text instead of sound, into their respective payloads. In order to do that

- will generate the code generated from the test template
- evaluated dynamically using `exec()`[82] and retrieving the root element
- uploading to `dragonfly.test.ElementTester` with the text engine

```
@pytest.fixture(scope="session")
def dragonfly_tester():
    from dragonfly.language.loader import language

    language._language = "en"
    test_code = generate_contents("test")[Path("test.py")]
    data = {}
    c = compile(test_code, "<str>", "exec")
    exec(c, data)

    return ElementTester(data["StandardCommand"], get_engine("text"))
```

Danger: We must explicitly request a "text" engine, otherwise you might end up hooking on sapi5. the same can happen if we do not explicitly set the language to English due to hidden call in the constructor of the *IntegerRef*

[81] https://docs.pytest.org/en/latest/reference/reference.html#pytest.fixture
[82] https://docs.python.org/3/library/functions.html#exec

8.4.3 Hypothesis Test

- We create model instances using `hypothesis.strategies.from_type()`[83]
- convert them into the spoken form
- parse them using the element tester in order to obtain a dictionary
- we parse that dictionary into a pydantic model
- we compare the two model instances are equal

```
@settings(max_examples=1000, deadline=500)
@given(c=strategies.from_type(StandardCommand))
def test_bidirectional(c, dragonfly_tester):
    spoken_form = into_spoken(c)
    r = StandardCommand.parse_obj(dragonfly_tester.recognize(spoken_form))
    assert r == c, f"{spoken_form} {r} {c}"
```

Note: we must also register custom strategies with `hypothesis.strategies.register_type_strategy()`[84] to avoid generating large integers and lists

---

[83] https://hypothesis.readthedocs.io/en/latest/data.html#hypothesis.strategies.from_type
[84] https://hypothesis.readthedocs.io/en/latest/data.html#hypothesis.strategies.register_type_strategy

Chapter 9

LSP

9.1 Introduction

In order to allow for communication between our client running inside the editor and the server implementing the business logic, while we could build upon a custom protocol, we have instead chosen to leverage their lingua franca for such tasks,namely the language server protocol

The language server protocol is built on top of **json-RPC v2** and is extended with **bidirectional communication**. Using a set of **standardized and programming language agnostic messages** over a variety of possible transports such as stdio or TCP, a **language client** can talk to one or more **language servers running on separate processes**,synchronizing the contents of the files being edited and issuing requests for **IntelliSense services** such as auto completion, go to definition, signature help etc. provided of course that the server advertises the **corresponding capability** in the initial handshake

Of course, we are not interested in language services like auto completion so we are going to have to extend the Protocol.

9.2 Challenges And Implementation Decisions

9.2.1 Chosen LSP Implementation

One of the early design decisions need to be made was how my project is going to build upon the language server protocol at the code level. initially I had considered publishing code as a plug-in for pylsp[85] but since it is not the only already existing popular language server for python, I prefer to build a standalone one using pygls[86] , which provide a generic LSP implementation/SDK

9.2.2 Custom Method vs Workspace Execute command

Moving on, we must decide which of the two ways we're going to use in order to extend the language server protocol

- create custom methods on the server
- create an workspace_executeCommand[87] method

The decision was made to go with the latter option for two reasons

[85] https://pypi.org/project/pylsp
[86] https://pypi.org/project/pygls
[87] https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/#workspace_executeCommand

- pygls[88] of the moment of this writing does not natively support custom server methods. It is possible to to work around this using pygls-tagls-custom[89] which was developed for tagls[90] but given we are only dealing with one single server command at the moment it is more complex.
- workspace_executeCommand[91] should be already supported natively by most clients,making the project easier to becoming incrementally adoptable in more editors

That being said, the current implementation of pygls[92] can be a little bit surprising specifically in the way it tries to process the arguments for we pass workspace_executeCommand[93] . To circumvent this issue we create a custom method for registering commands which going to make sure the arguments are validated and unpacked.

```
    def custom_command(
        self, command_name: str
    ) -> Callable[[F], Callable[["PythonVoiceCodingPluginLanguageServer", Any], Any]]:
        """Decorator used to register custom commands.

        Example:
            @ls.command('myCustomCommand')
            def my_cmd(ls, a, b, c):
                pass
        """

        def wrapper(f: F):
            f = validate_arguments(config=dict(arbitrary_types_allowed=True))(f)

            async def function(server: PythonVoiceCodingPluginLanguageServer, args):
                return await f(server, *args)

            self.lsp.fm.command(command_name)(function)
            return f

        return wrapper
```

9.2.3 Stateful user interface

- Language server protocol is designed to be user interface agnostic
- no user interface synchronization
- multiple selections are not a first-class citizen

[88] https://pypi.org/project/pygls
[89] https://pypi.org/project/pygls-tagls-custom
[90] https://pypi.org/project/tagls
[91] https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/#workspace_executeCommand
[92] https://pypi.org/project/pygls
[93] https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/#workspace_executeCommand

9.2.4 Dialogical versus one-way

Now all of this stateful information can flow between the client and the server in two ways:

- send and receive the data in a single go Via their request parameters and the response message
- allow a small dialogue, where the server queries the client for the types of information the latter has declared support for

In general the first approach is more consistent, but for reasons that have to do with making the amount of code necessary to support the basic features as low as possible and thus speed up the process of porting the project to more editors, I have chosen approach that mostly relies on the second one, with an optional partial fall back

Example:

```
:: --> PythonVoiceCodingPlugin workspace/executeCommand(2): {'command': 'navigate_ast',
↪'workDoneToken': 'wd2', 'arguments': [{'op': 'select', 'query': {'description': 'second
↪', 'target': 'argument'}}, 'file:///C:/Users/Username/Documents/GitHub/
↪PythonVoiceCodingPlugin/python_voice_coding_plugin/lsp/server.py']}
:: <-  PythonVoiceCodingPlugin window/showMessage: {'type': 3, 'message': 'Typedocument'}
:: <-- PythonVoiceCodingPlugin selection/getCursors(e168caa7-3ae8-4337-9d61-
↪901d361eb0d8): {'uri': 'file:///C:/Users/Username/Documents/GitHub/
↪PythonVoiceCodingPlugin/python_voice_coding_plugin/lsp/server.py'}
:: >>> PythonVoiceCodingPlugin e168caa7-3ae8-4337-9d61-901d361eb0d8: [{'start': {'line':
↪12, 'character': 18}, 'end': {'line': 12, 'character': 18}}]
:: <-- PythonVoiceCodingPlugin regions/get(a1ea18fc-e98f-49b9-807f-794e2d03de92): {'uri
↪': 'file:///C:/Users/Username/Documents/GitHub/PythonVoiceCodingPlugin/python_voice_
↪coding_plugin/lsp/server.py'}
:: >>> PythonVoiceCodingPlugin a1ea18fc-e98f-49b9-807f-794e2d03de92: {'Green': [],
↪'Origin': [], 'Yellow': [], 'Red': [{'start': {'line': 1, 'character': 1}, 'end': {
↪'line': 2, 'character': 4}}], 'Blue': [], 'Main': [], 'Orange': []}
:: <-- PythonVoiceCodingPlugin workspace/applyEdit(cec3d19e-0488-4e29-b389-
↪40d0afa9f384): {'label': None, 'edit': {'changes': {'file:///C:/Users/Username/
↪Documents/GitHub/PythonVoiceCodingPlugin/python_voice_coding_plugin/lsp/server.py': [{
↪'newText': '#\n', 'range': {'start': {'line': 0, 'character': 0}, 'end': {'line': 0,
↪'character': 0}}}]}}}
:: <-  PythonVoiceCodingPlugin selection/setCursors: {'uri': 'file:///C:/Users/Username/
↪Documents/GitHub/PythonVoiceCodingPlugin/python_voice_coding_plugin/lsp/server.py',
↪'selection': [{'start': {'line': 0, 'character': 2}, 'end': {'line': 1, 'character': 4}
↪}]}
:: <-  PythonVoiceCodingPlugin clipboard/setContents: {'contents': 'something'}
:: <-  PythonVoiceCodingPlugin regions/set: {'regions': {'Red': [{'start': {'line': 0,
↪'character': 2}, 'end': {'line': 1, 'character': 4}}]}, 'uri': 'file:///C:/Users/
↪Username/Documents/GitHub/PythonVoiceCodingPlugin/python_voice_coding_plugin/lsp/
↪server.py'}
:: <<< PythonVoiceCodingPlugin 2: None
```

9.2.5 UTF 16 versus UTF-8

Internally for our business logic we are using `CodeRange`[94] and `CodePosition`[95] which has 2 big differences with range[96] and position[97]

- Lines are 1-indexed vs 0-indexed
- the column offsets are using Unicode code points whereas in the language server protocol they are UTF 16 based

So we must make sure to convert them appropriately. The latest version of the language server protocol, namely 3.17, does allow clients to advertise UTF-8 support but implementing UTF 16 is still mandatory.

9.3 Extensions

In order to model the user interface state in the editor we use the following representation

class State

State(code: str, selection: Sequence[libcst._position.CodeRange] = <factory>, clipboard: Optional[str] = None, regions: Mapping[str, Sequence[libcst._position.CodeRange]] = <factory>)

code: `str`[98]

selection: `Sequence`[99]**[CodeRange]**

clipboard: `Optional`[100]**[**`str`[101]**] = None**

regions: `Mapping`[102]**[**`str`[103]**,** `Sequence`[104]**[CodeRange]]**

__init__(*code*, *selection=<factory>*, *clipboard=None*, *regions=<factory>*)

Parameters

- **code** (*str*[105]) –
- **selection** (*Sequence*[106]*[CodeRange]*) –
- **clipboard** (*Optional*[107]*[str*[108]*]*) –
- **regions** (*Mapping*[109]*[str*[110]*, Sequence*[111]*[CodeRange]]*) –

Return type
None

[94] https://libcst.readthedocs.io/en/latest/metadata.html#libcst.metadata.CodeRange
[95] https://libcst.readthedocs.io/en/latest/metadata.html#libcst.metadata.CodePosition
[96] https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/#range
[97] https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/#position

Then we define a series of server initiated CRUD-Like requests for reading and writing those properties, for which the client can opt in via the corresponding capabilities

9.3.1 Client Experimental Capabilities

```
class ExperimentalCapabilities
    selection: SelectionCapabilities
    clipboard: ClipboardCapabilities
    regions: RegionsCapabilities
class SelectionCapabilities
    get: bool[112]
    set: bool[113]
    multiple: bool[114]
class ClipboardCapabilities
    set: bool[115]
class RegionsCapabilities
    get: bool[116]
    set: bool[117]
```

[98] https://docs.python.org/3/library/stdtypes.html#str
[99] https://docs.python.org/3/library/typing.html#typing.Sequence
[100] https://docs.python.org/3/library/typing.html#typing.Optional
[101] https://docs.python.org/3/library/stdtypes.html#str
[102] https://docs.python.org/3/library/typing.html#typing.Mapping
[103] https://docs.python.org/3/library/stdtypes.html#str
[104] https://docs.python.org/3/library/typing.html#typing.Sequence
[105] https://docs.python.org/3/library/stdtypes.html#str
[106] https://docs.python.org/3/library/typing.html#typing.Sequence
[107] https://docs.python.org/3/library/typing.html#typing.Optional
[108] https://docs.python.org/3/library/stdtypes.html#str
[109] https://docs.python.org/3/library/typing.html#typing.Mapping
[110] https://docs.python.org/3/library/stdtypes.html#str
[111] https://docs.python.org/3/library/typing.html#typing.Sequence
[112] https://docs.python.org/3/library/functions.html#bool
[113] https://docs.python.org/3/library/functions.html#bool
[114] https://docs.python.org/3/library/functions.html#bool
[115] https://docs.python.org/3/library/functions.html#bool
[116] https://docs.python.org/3/library/functions.html#bool
[117] https://docs.python.org/3/library/functions.html#bool

9.3.2 Requests

Selection

SELECTION_GET_CURSORS = 'selection/getCursors'

class GetSelectionRequest

GetSelectionResponse
alias of Sequence[118][Range]

Regions

REGIONS_GET = 'regions/get'

class GetRegionsRequest

GetRegionsResponse
alias of Mapping[119][str[120], Sequence[121][Range]]

9.3.3 Notification

Code Edits

Code Edits can be handled natively by workspace/applyEdit[122]

Selection

SELECTION_SET_CURSORS = 'selection/setCursors'

class SetSelectionRequest

selection: Sequence[123][Range]

Alternatively, it can be handled natively , but only for a single selection window/showDocument[124]

[118] https://docs.python.org/3/library/typing.html#typing.Sequence
[119] https://docs.python.org/3/library/typing.html#typing.Mapping
[120] https://docs.python.org/3/library/stdtypes.html#str
[121] https://docs.python.org/3/library/typing.html#typing.Sequence
[122] https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/#workspace_applyEdit
[123] https://docs.python.org/3/library/typing.html#typing.Sequence
[124] https://microsoft.github.io/language-server-protocol/specifications/lsp/3.17/specification/#window_showDocument

Clipboard

```
CLIPBOARD_SET_CONTENTS = 'clipboard/setContents'

class ClipboardSetContentsRequest

     contents: str[125]
```

Regions

```
REGIONS_SET = 'regions/set'

class SetRegionsRequest

     regions: Mapping[126][str[127], Sequence[128][Range]]
```

[125] https://docs.python.org/3/library/stdtypes.html#str
[126] https://docs.python.org/3/library/typing.html#typing.Mapping
[127] https://docs.python.org/3/library/stdtypes.html#str
[128] https://docs.python.org/3/library/typing.html#typing.Sequence

Chapter 10

Taming The Sphinx

The old approach for documenting the project was solely based on manually recording live examples of voice commands, by capturing screen output using NickeManarin/ScreenToGif[129] . This approach can be very time-consuming if you want to record 100+ of such examples , while giving little guaranty that the behavior displayed is going to match the current version of the code base. A more automated approach was needed and ideally one that could also integrate with my efforts to address the lack of testing infrastructure.

10.1 Killing Three Birds With One Stone

`doctest`[130] is a powerful standard library module that allows developers to embed snippets of code similar to interactive Python sessions (REPL) right inside their functions/classes/modules docstrings. Upon execution, it is going to verify that each command/input is going to produce the corresponding output. For Example:

```
def contains(outer: CodeRange, inner: CodeRange) -> bool:
    r"""
    >>> a=CodeRange(CodePosition(1,2),CodePosition(1,3))
    >>> b=CodeRange(CodePosition(1,2),CodePosition(2,3))
    >>> c=CodeRange(CodePosition(1,2),CodePosition(2,0))
    >>> d=CodeRange(CodePosition(1,1),CodePosition(2,1))
    >>> contains(a,b)
    False
    >>> contains(b,a)
    True
    >>> contains(c,a)
    True
    >>> contains(d,a)
    True
    >>> contains(c,d)
    False
    >>> contains(d,c)
    True
    >>> contains(c,b)
    False
    >>> contains(b,c)
    True
    >>> contains(b,c)
```

(continues on next page)

[129] https://github.com/NickeManarin/ScreenToGif
[130] https://docs.python.org/3/library/doctest.html#module-doctest

(continued from previous page)

```
    True

    """
    second = astuple(outer)
    first = astuple(inner)
    return second[0] <= first[0] <= second[1] and second[0] <= first[1] <= second[1]
```

this technique allows developers to kill two birds with one stone as those input-output examples serve both

- as **tests right next to their code**
- as **documentation for other developers** reading or wanting to use their package

In this project were going to take the idea one step further and aim for a **third goal** : we are going to use those tests in order to generate **documentation for our end-users** . Initially I wanted these to be in the form of animated gifs but static images or image like objects could do.

Since our project already has a notion of a `State` object representing everything the language server knows about the user interface, we are going to build around that in two phases

- Create the appropriate state objects from the doctest examples
- Highlight them in a special manner, reminiscent of the way an editor would display them

10.2 Highlighting The State

10.2.1 Pygments Concepts

pygments[131] is a widely used and powerful generic syntax highlighter employed by sphinx[132] in order to prettify source code examples. The highlighting processing consists of four components

- Firstly,a lexer splits the source into tokens, which are pieces of source code along with token type determining what they semantically represent

```
>>> from pygments import lex
>>> import pp
>>> _example_code="def function(x:int)->int\n    return x+1\n"
>>> lexer = PythonLexer()
>>> pp(list(lex(_example_code, lexer)))
[
    (Token.Keyword, 'def'),
    (Token.Text, ' '),
    (Token.Name.Function, 'function'),
    (Token.Punctuation, '('),
    (Token.Name, 'x'),
    (Token.Punctuation, ':'),
    (Token.Name.Builtin, 'int'),
    (Token.Punctuation, ')'),
    (Token.Operator, '-'),
    (Token.Operator, '>'),
    (Token.Name.Builtin, 'int'),
```

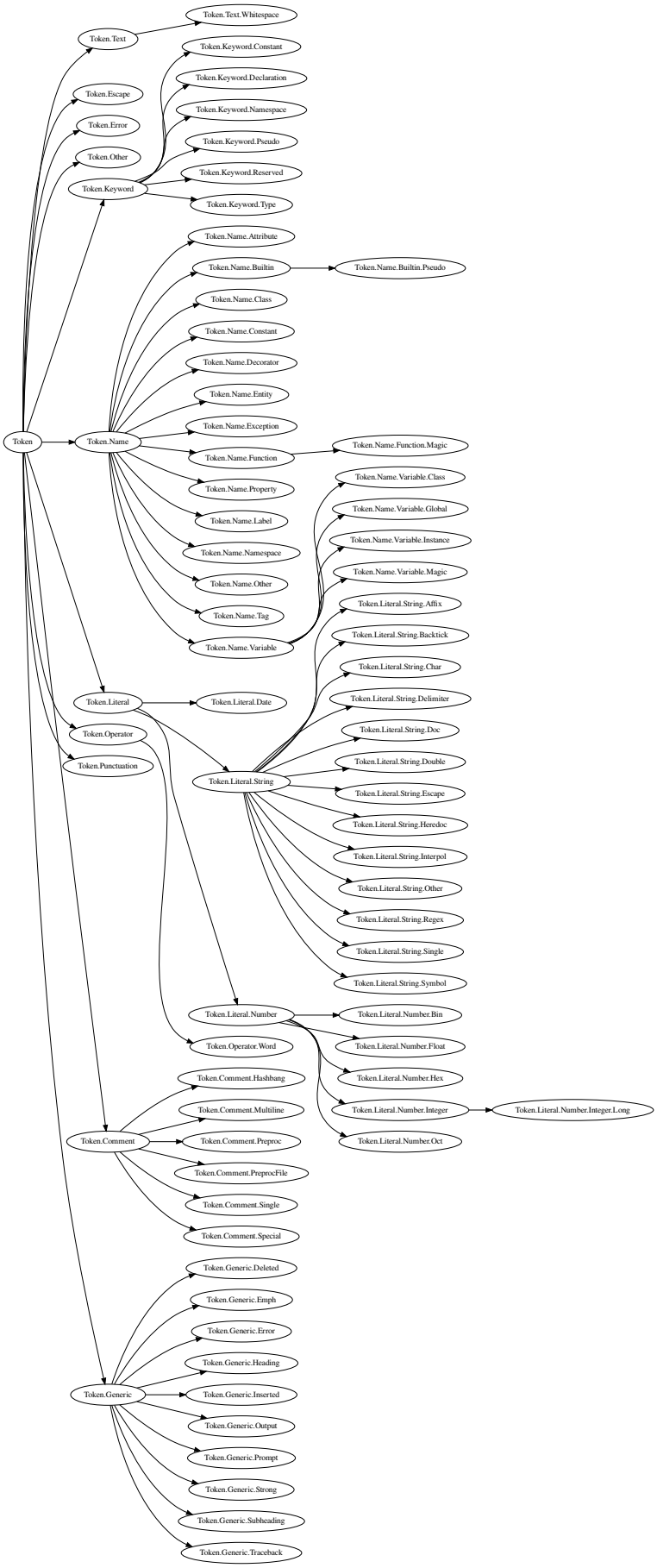
(continues on next page)

[131] https://pypi.org/project/pygments
[132] https://pypi.org/project/sphinx

(continued from previous page)

```
    (Token.Text, '\n'),
    (Token.Text, '    '),
    (Token.Keyword, 'return'),
    (Token.Text, ' '),
    (Token.Name, 'x'),
    (Token.Operator, '+'),
    (Token.Literal.Number.Integer, '1'),
    (Token.Text, '\n'),
]
```

The standard token types form a hierarchy shown below


Token
Token.Text
Token.Text.Whitespace
Token.Escape
Token.Error
Token.Other
Token.Keyword
Token.Keyword.Constant
Token.Keyword.Declaration
Token.Keyword.Namespace
Token.Keyword.Pseudo
Token.Keyword.Reserved
Token.Keyword.Type
Token.Name
Token.Name.Attribute
Token.Name.Builtin
Token.Name.Builtin.Pseudo
Token.Name.Class
Token.Name.Constant
Token.Name.Decorator
Token.Name.Entity
Token.Name.Exception
Token.Name.Function
Token.Name.Function.Magic
Token.Name.Property
Token.Name.Label
Token.Name.Namespace
Token.Name.Other
Token.Name.Tag
Token.Name.Variable
Token.Name.Variable.Class
Token.Name.Variable.Global
Token.Name.Variable.Instance
Token.Name.Variable.Magic
Token.Literal
Token.Literal.Date
Token.Literal.String
Token.Literal.String.Affix
Token.Literal.String.Backtick
Token.Literal.String.Char
Token.Literal.String.Delimiter
Token.Literal.String.Doc
Token.Literal.String.Double
Token.Literal.String.Escape
Token.Literal.String.Heredoc
Token.Literal.String.Interpol
Token.Literal.String.Other
Token.Literal.String.Regex
Token.Literal.String.Single
Token.Literal.String.Symbol
Token.Literal.Number
Token.Literal.Number.Bin
Token.Literal.Number.Float
Token.Literal.Number.Hex
Token.Literal.Number.Integer
Token.Literal.Number.Integer.Long
Token.Literal.Number.Oct
Token.Operator
Token.Operator.Word
Token.Punctuation
Token.Comment
Token.Comment.Hashbang
Token.Comment.Multiline
Token.Comment.Preproc
Token.Comment.PreprocFile
Token.Comment.Single
Token.Comment.Special
Token.Generic
Token.Generic.Deleted
Token.Generic.Emph
Token.Generic.Error
Token.Generic.Heading
Token.Generic.Inserted
Token.Generic.Output
Token.Generic.Prompt
Token.Generic.Strong
Token.Generic.Subheading
Token.Generic.Traceback

- The token stream can be then processed by filters that will produce a modified stream
- A formatter finally writes this stream to an output file, in a format such as HTML, LaTeX
- the formatting process is dictated by a style mapping each token type to a set of properties properties that determine how it should be highlighted

10.2.2 Modifying The Style

Firstly, we must notify the style in order to allow for regions with custom background and border colors. Ideally, for those highlighted regions we would like to change only those specific properties without interfering with the rest such as letter color. An easy way to achieve that is to simply extend the standard hierarchies shown above. In particular, we are going to adapt the chosen style so that for each token type like *Token.Text* there will be additional token types *Token.Text.Red* , *Token.Text.Green* etc all of them inheriting the base properties but overriding the background color and the border. To illustrate:

Examples

```
>>> import pp
>>> pp(style_modifiers)
{
    '': '',
    'Blue': '#008000',
    'Green': '#000080',
    'Red': '#800000',
    'Selection': '#808080',
    'Yellow': '#808000',
}
>>> pp(len(MaterialStyle.styles))
78
>>> pp(len(CustomStyle.styles))
468
>>> pp(list(MaterialStyle.styles.items())[:3])
[
    (Token.Text, '#EEFFFF'),
    (Token.Escape, '#89DDFF'),
    (Token.Error, '#FF5370'),
]
>>> pp(list(CustomStyle.styles.items())[:16])
[
    (Token.Text, '#EEFFFF bg: border:'),
    (Token.Text.Selection, '#EEFFFF bg:#808080 border:#808080'),
    (Token.Text.Red, '#EEFFFF bg:#800000 border:#800000'),
    (Token.Text.Blue, '#EEFFFF bg:#008000 border:#008000'),
    (Token.Text.Green, '#EEFFFF bg:#000080 border:#000080'),
    (Token.Text.Yellow, '#EEFFFF bg:#808000 border:#808000'),
    (Token.Escape, '#89DDFF bg: border:'),
    (Token.Escape.Selection, '#89DDFF bg:#808080 border:#808080'),
    (Token.Escape.Red, '#89DDFF bg:#800000 border:#800000'),
    (Token.Escape.Blue, '#89DDFF bg:#008000 border:#008000'),
    (Token.Escape.Green, '#89DDFF bg:#000080 border:#000080'),
    (Token.Escape.Yellow, '#89DDFF bg:#808000 border:#808000'),
    (Token.Error, '#FF5370 bg: border:'),
    (Token.Error.Selection, '#FF5370 bg:#808080 border:#808080'),
```

(continues on next page)

(continued from previous page)

```
    (Token.Error.Red, '#FF5370 bg:#800000 border:#800000'),
    (Token.Error.Blue, '#FF5370 bg:#008000 border:#008000'),
]
```

10.2.3 Annotating The Tokens

Then in order to produce those special tokens we can simply create a filter that this going to change the token type of all tokens found within the desirable regions.

class RegionsFilter

function(*lexer*, *stream*, *options*)

Note: if necessary, it might end up splitting tokens. For instance, in the following snippet suppose we want to convert the words *function* and from the return statement **only** *retu* . This is the output we get

---

Examples

```
>>> from pygments import lex
>>> import pp
>>> _example_code="def function(x:int)->int\n    return x+1\n"
>>> lexer = PythonLexer()
>>> pp(list(lex(_example_code, lexer)))
[
    (Token.Keyword, 'def'),
    (Token.Text, ' '),
    (Token.Name.Function, 'function'),
    (Token.Punctuation, '('),
    (Token.Name, 'x'),
    (Token.Punctuation, ':'),
    (Token.Name.Builtin, 'int'),
    (Token.Punctuation, ')'),
    (Token.Operator, '-'),
    (Token.Operator, '>'),
    (Token.Name.Builtin, 'int'),
    (Token.Text, '\n'),
    (Token.Text, '    '),
    (Token.Keyword, 'return'),
    (Token.Text, ' '),
    (Token.Name, 'x'),
    (Token.Operator, '+'),
    (Token.Literal.Number.Integer, '1'),
    (Token.Text, '\n'),
]
>>> regions={"red":translating_to_coderange("[[[2,4],[2,8]],[[1,4],[1,12]]]")}
>>> regions
{'red': [CodeRange(start=CodePosition(line=2, column=4), end=CodePosition(line=2,
↪column=8)), CodeRange(start=CodePosition(line=1, column=4), end=CodePosition(line=1,
↪column=12))]}
>>> _example_state = State(_example_code,regions=regions)
```

(continues on next page)

(continued from previous page)

```
>>> lexer.add_filter(RegionsFilter(regions=regions_from_state(_example_state)))
>>> pp(list(lex(_example_code, lexer)))
[
    (Token.Keyword, 'def'),
    (Token.Text, ' '),
    (Token.Name.Function.Red, 'function'),
    (Token.Punctuation, '('),
    (Token.Name, 'x'),
    (Token.Punctuation, ':'),
    (Token.Name.Builtin, 'int'),
    (Token.Punctuation, ')'),
    (Token.Operator, '-'),
    (Token.Operator, '>'),
    (Token.Name.Builtin, 'int'),
    (Token.Text, '\n'),
    (Token.Text, '    '),
    (Token.Keyword.Red, 'retu'),
    (Token.Keyword, 'rn'),
    (Token.Text, ' '),
    (Token.Name, 'x'),
    (Token.Operator, '+'),
    (Token.Literal.Number.Integer, '1'),
    (Token.Text, '\n'),
]
```

10.2.4 Sphinx node and directive

Unfortunately getting our custom filters to work with the native `docutils.nodes.doctest_block` is a little bit tricky and requires some monkey patching, so I simply ended up creating a new type of node

class state_code_block

> **__init__**(*state=None*, **args*, ***kwargs*)
>
> **state:** ***State***

and registering the corresponding visitor functions for the sphinx writers:

```
def setup(app: Sphinx):
    app.add_node(
        state_code_block,
        html=(visit_state_code_block_html, None),
        latex=(visit_state_code_block_latex, None),
    )
    app.add_directive("state_code_block", StateCodeBlockDirective)
```

Along with it, we also registered a custom directive that can accept parameters in yaml format

class StateCodeBlockDirective

> Directive for a code block with special highlighting or line numbering settings.
>
> **has_content = True**
>
> > May the directive have content?

required_arguments = 0

Number of required directive arguments.

optional_arguments = 1

Number of optional arguments after the required arguments.

final_argument_whitespace = False

May the final argument contain whitespace?

option_spec: Dict[133][str[134], Callable[135][[str[136]], Any[137]]] = {'clipboard': <function translating_to_coderange>, 'regions': <function translating_to_coderange>, 'selection': <function translating_to_coderange>}

Mapping of option names to validator functions.

run()

To illustrate

```
.. state_code_block::
    :selection:
        - [[1,4],[1,12]]
        - [[4,0],[4,5]]
    :regions:
        Red:
            - [[1,4],[1,12]]
            - [[4,6],[4,10]]


    from typing import Iterator

    # This is an example
    class Math:
        @staticmethod
        def fib(n: int) -> Iterator[int]:
            """ Fibonacci series up to n """
            a, b = 0, 1
            while a < n:
                yield a
                a, b = b, a + b

    result = sum(Math.fib(42))
    print("The answer is {}".format(result))
```

produces

[133] https://docs.python.org/3/library/typing.html#typing.Dict
[134] https://docs.python.org/3/library/stdtypes.html#str
[135] https://docs.python.org/3/library/typing.html#typing.Callable
[136] https://docs.python.org/3/library/stdtypes.html#str
[137] https://docs.python.org/3/library/typing.html#typing.Any

10.3 Summarizing The DocTests Or Generating The State

While initially some more complex ideas were considered, I eventually settled for something relatively simple: run the doctests, inspect the namespace defined for special variables like *results* , *origins* which will typically be lists/lists of lists/dictionaries of lists etc containing `libcst.CSTNode`[138] objects. Thanks to the flexibility and round-trip capabilities of `libcst` we can easily generate source code containing those top level *origins* nodes as well as compute locations for those *results* nodes within it.

Note: For convenience reasons, we are going to use the xdoctest[139] instead of the built-in `doctest`[140] for this task

---

10.4 Generating The Doc Tests

A big thanks to pytest_accept[141].

[138] https://libcst.readthedocs.io/en/latest/nodes.html#libcst.CSTNode
[139] https://pypi.org/project/xdoctest
[140] https://docs.python.org/3/library/doctest.html#module-doctest
[141] https://pypi.org/project/pytest_accept

Βιβλιογραφία

[1] Hands-free coding. https://handsfreecoding.org/.

[2] Dictation-toolbox. https://dictation-toolbox.github.io/dictation-toolbox.org/.

[3] Programming by voice may be the next frontier in software development. https://spectrum.ieee.org/programming-by-voice-may-be-the-next-frontier-in-software-development.

[4] On voice coding. https://dusty.phillips.codes/2020/02/15/on-voice-coding/.