ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΣΥΝΤΑΚΤΙΚΗ ΥΠΟΚΑΤΑΣΤΑΣΗ ΛΟΓΙΣΜΙΚΟΥ

Καλλιόπη Χαλκιά

Μεταπτυχιακή Εργασία

ΗΡΑΚΛΕΙΟ, ΚΡΗΤΗ
Μάιος 1993

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΣΥΝΤΑΚΤΙΚΗ ΥΠΟΚΑΤΑΣΤΑΣΗ ΛΟΓΙΣΜΙΚΟΥ

Εργασία που υποβλήθηκε από την
ΚΑΛΛΙΟΠΗ ΕΥΑΓΓΕΛΟΥ ΧΑΛΚΙΑ
ως μερική απαίτηση για την απόκτηση του
ΔΙΠΛΩΜΑΤΟΣ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΕΞΕΙΔΙΚΕΥΣΗΣ

Ηράκλειο, Μάιος 1993

Συγγραφέας : . . . .
Τμήμα Επιστ. Υπολογιστών, Μάιος 1993

Εισηγητική Επιτροπή

. . . .
Αναπληρωτής Καθηγητής Πάνος Κωνσταντόπουλος, Επόπτης

. . . .
Αναπληρωτής Καθηγητής Μανόλης Κατεβαίνης, Μέλος

. . . .
Αναπληρωτής Καθηγητής Γιώργος Τζιρίτας, Μέλος

Δεκτή : . . . .
Αναπληρωτής Καθηγητής Πάνος Κωνσταντόπουλος,
Πρόεδρος Επιτροπής Μεταπτυχιακών Σπουδών

Συντακτική Υποκατάσταση Λογισμικού

Καλλιόπη Χαλκιά
Μεταπτυχιακή Εργασια

Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

Περίληψη

Στην παρούσα εργασία στόχος μας είναι η περιγραφή σχέσεων συντακτικής υποκατάστασης ή ομοιότητας μεταξύ λογισμικών αντικειμένων, ώστε να γίνει η διαδικασία της αναχρησιμοποίησης όσο το δυνατόν πιο αποδοτική. Μεταξύ των λογισμικών αντικειμένων συγκαταλέγονται όχι μόνο ολοκληρωμένα προγράμματα αλλά και διαδικασίες που επιτελούν μία συγκεκριμένη λειτουργία καθώς και τύποι αντικειμένων, όπως ορίζονται σε οντοκεντρικές γλώσσες προγραμματισμού.

Αρχικά προτείνομε ένα μοντέλο για την περιγραφή και την αποθήκευση σχέσεων συντακτικής υποκατάστασης ή ομοιότητας σε Βάσεις Περιγραφής Λογισμικού (ΒΠΛ). Κατόπιν δίνομε ένα μοντέλο για την περιγραφή της συντακτικής δομής της γλώσσας CooL, με βάση το οποίο θα είμαστε σε θέση να περιγράψομε την συντακτική δομή λογισμικών αντικειμένων. Και τα δύο μοντέλα περιγράφονται στην γλώσσα TELOS, που είναι μία γλώσσα παράστασης γνώσεως.

Στη συνέχεια ορίζομε την σχέση υποκατάστασης για το σύνολο των βασικών και των δομημένων τύπων που ορίζονται και χρησιμοποιούνται σε γλώσσες προγραμματισμού, δίνοντας ταυτόχρονα και την συνάρτηση υπολογισμού του αντιστοίχου βαθμού υποκατάστασης. Κατόπιν δίνομε ένα σύνολο από γνωρίσματα που χαρακτηρίζουν την κάθε κατηγορία λογισμικού και με βάση αυτά τα γνωρίσματα προτείνομε μία συνάρτηση υπολογισμού του βαθμού υποκατάστασης για λογισμικά αντικείμενα. Η συνάρτηση αυτή χρησιμοποιεί επιμέρους αποτελέσματα για βαθμούς υποκατάστασης μεταξύ άλλων αντικειμένων, ανάλογα με το είδος τους (βασικοί ή δομημένοι τύποι ή λογισμικό). Τέλος προτείνομε και ένα μοντέλο για τον υπολογισμό του βαθμού

υποκατάστασης ή ομοιότητας μεταξύ λογισμικών αντικειμένων
με βάση σύνολα κριτηρίων.

Το μοντέλο συντακτικής υποκατάστασης λογισμικού δοκιμάστηκε σε δύο παραδείγματα εφαρμογής : προγράμματα ενοικίασης αυτοκινήτων και προγράμματα βελτιστοποίησης.

Επόπτης : Πάνος Κωνσταντόπουλος
Αναπληρωτής Καθηγητής Επιστήμης Υπολογιστών, Πανεπιστημίου Κρήτης

Syntactical Software Substitution

Calliope Halkia
Master of Science Thesis

Department of Computer Science
University of Crete

Abstract

In this paper we define and describe syntactical substitution and syntactical similarity relationships between software objects, in order to make the reuse process more effective. As software objects we consider well documented programs and files, procedures, implementing a certain function, as well as object types, as they are defined and used in object oriented languages.

We introduce a model for the description and storage of syntactical substitution or syntactical similarity relationships in a Software Information Base (SIB). We also define a model for the description of the syntactical structure of CooL, in order to describe the syntax of software objects in CooL. Both models are described using the TELOS, a knowledge representation language.

We define the syntactical substitution reletionship between basic and structured types used in the programming language. Then we introduce a function for the evaluation of the degree of syntactical substitution between two given types. After that, we present a set of attributes for the software objects we deal with, which characterizes them. Having defined those attributes we introduce the function for the evaluation of the degree of syntactical substitution between two given software objects. This function uses the results of the same function with other intermediate operants, such as basic or structured types or software objects, in order to evaluate the degree of syntactical substitution. In the end we introduce an extended model for the evaluation of of the degree of syntactical substitution between software objects with respect to sets of attributes and we introduce a similar function for the evaluation of the degree of syntactical substitution with respect to sets of attributes.

Our model was used in two different application examples : a car rental system and a library of optimization algorithms.

Supervisor : Panos Constantopoulos
Associate Professor of Computer Science, University of Crete

στους γονείς μου...

Ευχαριστίες

Θα ήθελα να ευχαριστήσω ειλικρινά τον επιβλέποντα καθηγητή μου κ. Πάνο Κωνσταντόπουλο, για τις πολύτιμες συμβουλές του, τη καθοδήγηση και τη συμπαράστασή του, όποτε είχα ανάγκη.

Επίσης θα ήθελα να ευχαριστήσω τον δρ. Martin Doerr για τις συζητήσεις και τις υποδείξεις του κατά τη διάρκεια της εργασίας μου, και τις πολύτιμες συμβουλές του στο προγραμματιστικό κομμάτι της εργασίας μου.

Θα ήθελα ακόμα να ευχαριστήσω τον συνάδελφο Θαλή Γεωργίου για την συμπαράστασή του στις δύσκολες στιγμές της ανάπτυξης της εργασίας μου και την υποστήριξή του. Ακόμη, τον ευχαριστώ για όλες τις ευχάριστες στιγμές και τα ενθαρρυντικά του λόγια κατά τη διάρκεια της συγγραφής της εργασίας μου.

Ακόμη θα ήθελα να ευχαριστήσω την αδελφή μου, Νατάσα, και την φίλη μου Ελευθερία Χαλκιαδάκη, για την υπομονή τους και τις συμβουλές τους, όλο το διάστημα των μεταπτυχιακών μου σπουδών.

Τέλος θα ήθελα να ευχαριστήσω το Ινστιτούτο Πληροφορικής του Ι.Τ.Ε. για την οικονομική ενίσχυση και την υλικοτεχνική υποστήριξη που μου παρείχε κατά την διάρκεια των μεταπτυχιακών σπουδών μου.

Περιεχόμενα

1. Εισαγωγή

Αναχρησιμοποίηση (re-use) εν γένει είναι η χρήση είτε ιδεών είτε αντικειμένων (objects), που έχουν ήδη δημιουργηθεί στο παρελθόν, σε νέες καταστάσεις. Μπορούμε να διαχωρίσομε δε την αναχρησιμοποίηση σε δύο διαφορετικά επίπεδα :

(α) των ιδεών και της γνώσης, και

(β) συγκεκριμένων αντικειμένων.

Ενα από τα βασικά σημεία ώστε να γίνει η διαδικασία της αναχρησιμοποίησης όσο το δυνατό πιο αποδοτική είναι και η ανεύρεση ενός συνόλου αντικειμένων που ικανοποιούν, άλλα λιγότερο και άλλα περισσότερο, τις απαιτήσεις για την ανάπτυξη της νέας εφαρμογής. Η διαδικασία αυτή, δεδομένης μιας βάσης γνώσης με αντικείμενα προσφερόμενα προς αναχρησιμοποίηση, περιλαμβάνει την διατύπωση των απαιτήσεών που πρέπει να πληρούν τα υποψήφια προς αναχρησιμοποίηση αντικείμενα και την επιλογή και διάταξή τους με βάση το σύνολο των απαιτήσεών αυτών.

Στην παρούσα εργασία θα μας απασχολήσει μια συγκεκριμένη κατηγορία αντικειμένων, τα λογισμικά αντικείμενα (software objects) ή πιο απλά το λογισμικό. Λέγοντας λογισμικό εννοούμε εφαρμογές είτε ως ολοκληρωμένα προγράμματα είτε ως ρουτίνες που επιτελούν καθορισμένες λειτουργίες. Σκοπός μας είναι η εύρεση του βαθμού ανταπόκρισης τέτοιων λογισμικών αντικειμένων σε κάποιο σύνολο απαιτήσεων/κριτηρίων. Δεδομένης δηλαδή μιας συγκεκριμένης <u>συντακτικής</u> <u>περιγραφής,</u> που προτείνομε για τα λογισμικά αντικείμενα, που περιγράφει τις απαιτήσεις μας, ανευρίσκομε ένα σύνολο λογισμικών αντικειμένων που μπορούν να χρησιμοποιηθούν στη νέα μας εφαρμογή ικανοποιώντας τις απαιτήσεις μας σε κάποιο <u>βαθμό</u> το καθένα.

Η χρήση ενός λογισμικού αντικειμένου στη θέση του πραγματικού ή του ιδεατου αντικειμένου που αποτυπώνεται από τη δεδομένη συντακτική περιγραφή λέγεται <u>υποκατάσταση.</u> Η συντακτική περιγραφή αποδίδει μορφολογικά χαρακτηριστικά (απαιτήσεις), τα οποία και συγκεντρώνουν το ενδιαφέρον μας στην παρούσα εργασία. Η σύνδεση αυτών με λειτουργικά χαρακτηριστικά είναι έμμεση και δεν θα μας απασχολήσει εδώ.

1.1. Αναχρησιμοποίηση Λογισμικού

Οπως είδαμε, διαχωρίσαμε ήδη απ' την αρχή την αναχρησιμοποίηση μεταξύ ιδεών και αντικειμένων. Εμάς θα μας απασχολήσει ειδικά η αναχρησιμοποίηση λογισμικού.

Με τον όρο αναχρησιμοποίηση λογισμικού, εννοούμε τη δυνατότητα να χρησιμοποιήσομε μικρά ή μεγάλα κομμάτια εφαρμογών (προγραμμάτων) που έχουν ήδη αναλυθεί, σχεδιαστεί, περιγραφεί ή/και υλοποιηθεί κατά τη διαδικασία ανάπτυξης άλλων εφαρμογών. Τέτοια τμήματα εφαρμογών μπορούν είτε να χρησιμοποιηθούν αυτούσια είτε να προσαρμοστούν ή και να μετατραπούν κατά τις απαιτήσεις μας, εφόσον είναι αρκετά κοντά σε αυτές και ο χρόνος της προσαρμογής τους είναι κατά το δυνατόν ελάχιστος. Ετσι αφενός μειώνομε το χρόνο ανάπτυξης της εφαρμογής που μας ενδιαφέρει και αφετέρου μειώνομε τις πιθανότητες λάθους που θα μπορούσαν να εμφανιστούν τουλάχιστον στα κομμάτια που αναχρησιμοποιήσαμε, αφού είναι ήδη δημιουργημένα και ελεγμένα.

Θέλοντας λοιπόν να αναχρησιμοποιήσομε ήδη υπάρχοντα αντικείμενα δύο είναι τα κύρια βήματα που πρέπει να ακολουθήσομε [PrDiaz-Fre] :

(α) να καθορίσομε τα κριτήρια, τις απαιτήσεις μας δηλαδή, που πρέπει να πληρούν τα υποψήφια για αναχρησιμοποίηση αντικείμενα, και

(β) να κατατάξομε με κάποιο τρόπο τα αντικείμενα που τελικά πληρούν τις απαιτήσεις μας.

Εχοντας καθορίσει τις απαιτήσεις μας στο βήμα (α), που αναφέραμε παραπάνω, καταλήγομε στο να ορίσομε έμμεσα ένα ιδεατό λογισμικό αντικείμενο το οποίο θα θέλαμε να υπήρχε στη βάση μας προς αναχρησιμοποίηση. Στην περίπτωση που αυτό το ιδεατό λογισμικό αντικείμενο δεν υπάρχει, αναζητούμε στην βάση μας ένα σύνολο λογισμικών αντικειμένων που να μπορούν να υποκαταστήσουν το παραπάνω αντικείμενο με κάποιο βαθμό επιτυχίας. Στο βήμα (β) απλώς κατατάσσομε τα λογισμικά αντικείμενα που ανευρέθησαν ως σχετικά ως προς τις απαιτήσεις μας σύμφωνα με τα κριτήριά μας.

Εμείς στη παρούσα εργασία ασχολούμαστε με την σχέση που περιγράφει την μονότροπη υποκατάσταση μεταξύ δύο λογισμικών αντικειμένων, σε αντίθεση με την σχέση ομοιότητας που περιγράφει την αμφίδρομη υποκατάστασή τους. Η σχέση αυτή ονομάζεται σχέση υποκατάστασης και η ουσιαστικότερη διαφορά της από τη σχέση ομοιότητας είναι ότι ο βαθμός υποκατάστασης δύο αντικειμένων εξαρτάται από τη διάταξή τους. Στην περίπτωση βέβαια που όλα τα στοιχεία του συνόλου ομοίων αντικειμένων έχουν βαθμό υποκατάστασης ανεξάρτητο διάταξης τότε η σχέση χαρακτηρίζεται ως σχέση ομοιότητας.

Εχοντας λοιπόν ορίσει τέτοιου είδους σχέσεις είμαστε σε θέση να βρίσκομε σύνολα αντικειμένων που πληρούν τις απαιτήσεις μας. Επιπλέον, με τον τρόπο αυτό επαυξάνομε την πληροφορία που μας παρέχει η βάση δεδομένων που έχομε στη διάθεσή μας με αυτόματο τρόπο, αν είμαστε σε θέση να εισάγομε την νέα πληροφορία που θα έχομε από τέτοιες σχέσεις υποκατάστασης ή/και ομοιότητας κάθε φορά που διατυπώνομε κάποιες νέες απαιτήσεις.

1.2. Ενα μοντέλο για την περιγραφή λογισμικών αντικειμένων

Στα πλαίσια του ερευνητικού έργου ITHACA [SIB] το πρόβλημα της αναχρησιμοποίησης κατέχει κεντρική θέση. Σκοπός του έργου είναι η δημιουργία ενός ολοκληρωμένου περιβάλλοντος ανάπτυξης εφαρμογών με οντοκεντρικές τεχνικές (object-oriented techniques). Τούτο περιλαμβάνει μια οντοκεντρική γλώσσα προγραμματισμού και βάση δεδομένων, εργαλεία ανάπτυξης εφαρμογών και υποστήριξης, και μία βάση λογισμικού. Ο μηχανικός εφαρμογής (application engineer) δημιουργεί νέες εφαρμογές από αναχρησιμοποιήσιμα αντικείμενα και φτιάχνει νέο κώδικα ή ανασχεδιάζει αντικείμενα,απλά ή συνθετότερα, ώστε να συνδέσει τα αναχρησιμοποιήσιμα αντικείμενα μεταξύ τους για τη νέα εφαρμογή.

Τα αναχρησιμοποιήσιμα λογισμικά αντικείμενα τεκμηριώνονται βάσει πληροφοριών λογισμικού που αποθηκεύονται στην Βάση Περιγραφής Λογισμικού, ΒΠΛ (Software Information Base, SIB) [SIB]. Η ΒΠΛ χρησιμεύει σαν ευρετήριο στο σύνολο των αντικειμένων λογισμικού με σκοπό να εξυπηρετήσει την αναχρησιμοποίησή τους. Η γλώσσα παράστασης γνώσεων που χρησιμοποιείται για την ΒΠΛ είναι η TELOS [MBJK]. Είναι μία γλώσσα που βασίζεται στο Entity-Relationship (E-R) μοντέλο, σχεδιασμένη για την ανάπτυξη πληροφοριακών συστημάτων.

Ολα τα είδη περιγραφών αποθηκεύονται στην ΒΠΛ με έναν ομοιόμορφο τρόπο. Αυτό επιτυγχάνεται ορίζοντας ένα μετα-μοντέλο (metamodel) στην ΒΠΛ.

Κάθε διαφορετικό μοντέλο που χρησιμοποιείται στα διαφορετικά στάδια ανάπτυξης των εφαρμογών ορίζεται ως περίπτωση (instance) του μετα-μοντέλου [Veze91].

Για την περιγραφή λογισμικών αντικειμένων σε επίπεδο υλοποίησης, έχει οριστεί το Μοντέλο Υλοποίησης, που είναι ουσιαστικά ένα μετα-μοντέλο. Το Μοντέλο Υλοποίησης έχει αποθηκευτεί στην ΒΠΛ. Κατά συνέπεια, κάθε μοντέλο για την συντακτική περιγραφή της δομής λογισμικών αντικειμένων ορίζεται ως περίπτωση του Implementation Model. Ετσι, αν για παράδειγμα θέλομε να περιγράψομε την συντακτική δομή λογισμικού γραμμένου σε CooL, C, C++ ή σε οποιαδήποτε άλλη γλώσσα δεν έχομε παρά να περιγράψομε πρώτα το μοντέλο της συγκεκριμένης γλώσσας, και κατόπιν να περιγράψομε τα λογισμικά αντικείμενα που μας ενδιαφέρουν με βάση το μοντέλο μας. Ενα τέτοιο μοντέλο είναι και το μοντέλο CooL [ITHACA.91.E2], [ITHACA.92.E2.#1] που περιγράφεται εκτενώς στο Παράρτημα Α.

Δομικοί λίθοι όπως στις περισσότερες γλώσσες είναι οι βασικοί τύποι δεδομένων, που δεν είναι άλλοι από τους τύπους αριθμών και χαρακτήρων. Με βάση την περιγραφή αυτών των τύπων περιγράφομε και τους γνωστούς δομημένους τύπους (πίνακες, εγγραφές, ενώσεις, σύνολα, λίστες, διαδικασίες).

Κατόπιν περιγράφομε τα αντικείμενα που εμείς χειριζόμαστε ως λογισμικά αντικείμενα. Για μας ένα λογισμικό αντικείμενο είναι μία οντότητα με σύνθετο χαρακτήρα. Μπορεί να είναι μία διαδικασία ή συνάρτηση (δηλαδή procedure, routine, method, function) μόνο, αλλά και ένα πρόγραμμα ή ένα αρχείο με κώδικα. Τέλος μπορεί να είναι και ένα αντικείμενο (object) με τη σημασία που τους προσδίδομε σε μία οντοκεντρική γλώσσα (δηλαδή μία κλάση), μια και από μόνο του ένα τέτοιο αντικείμενο επιτελεί ένα σύνολο από πράξεις άλλες διαφανείς σε μας και άλλες όχι (π.χ. public methods και private methods).

Μία διαδικασία αποτελείται από ένα σύνολο από παραμέτρους εισόδου, μεταφοράς και εξόδου, καθώς και ενός προαιρετικού τύπου επιστροφής. Αν λοιπόν έχομε τύπο επιστροφής αναφερόμαστε σε συναρτήσεις, αλλιώς σε διαδικασίες. Οι παράμετροι εισόδου μεταφέρουν πληροφορία που μπορεί να χρησιμοιποηθεί μόνο και όχι να αλλάξει για να χρησιμοποιηθεί έξω από τη διαδικασία. Οι παράμετροι μεταφοράς μεταφέρουν πληροφορία που μπορεί να χρησιμοιποηθεί και να αλλάξει για να χρησιμοποιηθεί και έξω από τη διαδικασία. Οι παράμετροι εξόδου μεταφέρουν πληροφορία έξω από τη διαδικασία και δεν χρησιμοποιούνται ως στοιχεία πληροφορίας μέσα στη διαδικασία. Τέλος, ο τύπος επιστροφής δηλώνει τον τύπο που επιστρέφει η συνάρτηση.

Ενα αρχείο είναι ένα σύνολο από σταθερές, τύπους δεδομένων, μεταβλητές, διαδικασίες, συνάρτησεις καθώς και αναφορές σε άλλα αρχεία (για κώδικα που χρησιμοποιεί από τα αρχεία αυτά). Βλέπομε δηλαδή, πως ένα αρχείο είναι ένα αντικείμενο λογισμικού σύνθετο, που αποτελείται από άλλα απλά ή και σύνθετα αντικείμενα λογισμικού.

Ενα ολοκληρωμένο πρόγραμμα είναι στην ουσία ένα σύνολο από διαδικασίες, συναρτήσεις, τύπους ή/και κλάσεις, καθώς και από μία λίστα από όλα τα άλλα αρχεία που αποτελούν τις πηγές (source) του προγράμματος.

Για παράδειγμα, η περιγραφή ενός λογισμικού αντικειμένου γραμμένου σε γλώσσα CooL γίνεται όπως ορίζεται στο Παράρτημα Α.

1.3. Ενα μοντέλο για την περιγραφή της σχέσης υποκατάστασης και της σχέσης ομοιότητας

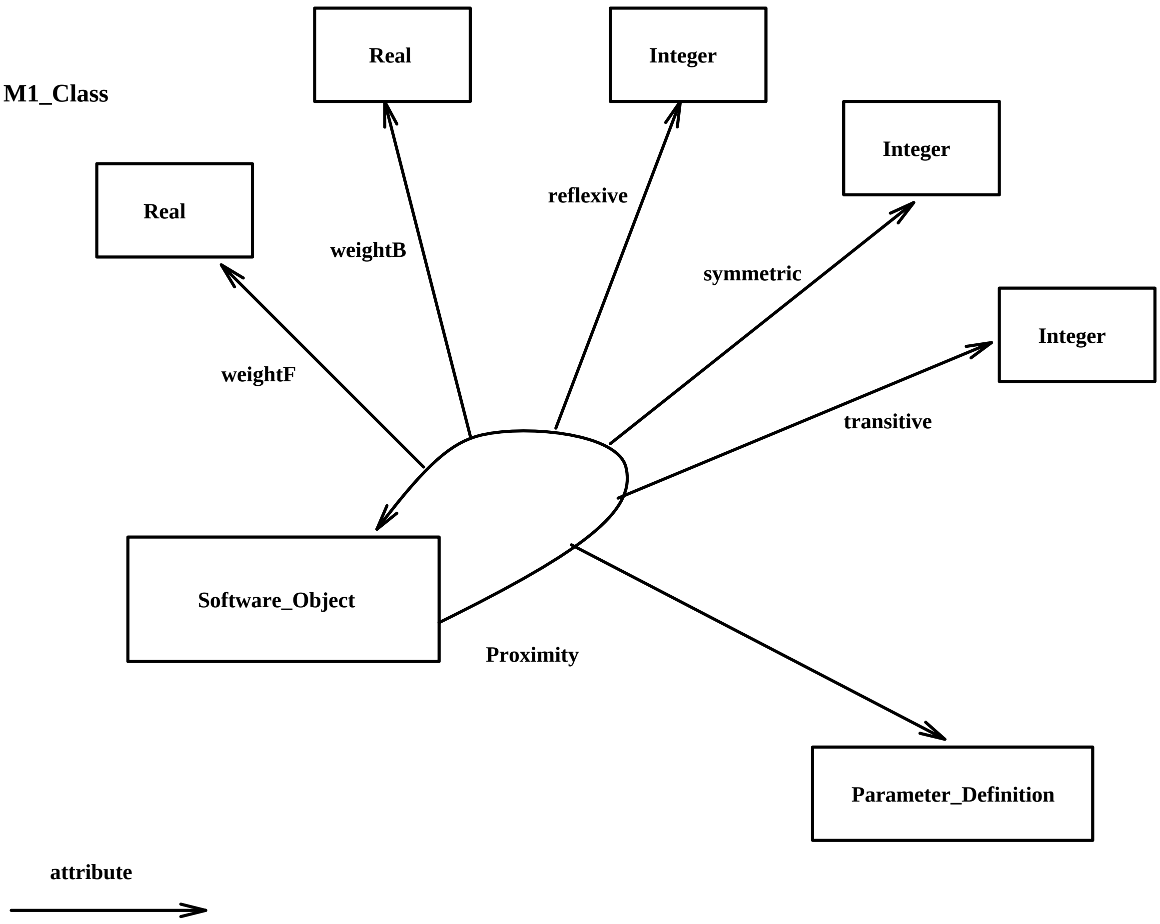
Ηδη έχομε αναφερθεί σε δύο είδη σχέσεων που θα χρησιμοποιήσομε : τη σχέση ομοιότητας και τη σχέση υποκατάστασης. Και οι δύο είναι διμελείς σχέσεις με πεδίο ορισμού το καρτεσιανό γινόμενο του συνόλου των αντικειμένων λογισμικού στην ΒΠΛ και πεδίο τιμών το κλειστό διάστημα [0, 1]. Επίσης ερωτήσεις για εύρεση ομοιότητας ή υποκατάστασης μεταξύ αντικειμένων λογισμικού μπορεί να αναφέρονται σε σύνολα κριτηρίων που καθορίζονται από τον χρήστη ενός περιβάλλοντας για αναχρησιμοποίηση και όχι απαραίτητα σε ένα και μόνο ρητά καθορισμένο σύνολο κριτηρίων. Επίσης, μπορούμε να περιγράψομε και πράξεις μεταξύ των συνόλων κριτηρίων που ορίζομε κάθε φορά.

Λόγω της διαφορετικής φύσης των δύο σχέσεων αρχικά ορίζομε μία γενικότερη κλάση για την περιγραφή μιας σχέσης προσέγγισης όπως είναι η ομοιότητα και η υποκατάσταση, δίνοντας επιπλέον τη δυνατότητα της περιγραφής της με βάση γνωστές ιδιότητες όπως ανακλαστική, συμμετρική/αντισυμμετρική και μεταβατική. Σε επόμενο κεφάλαιο θα εξετάσομε αναλυτικά ποιές ακριβώς ιδιότητες ισχύουν για κάθε μια σχέση που χειριζόμαστε.

Η ύπαρξη ενός τέτοιου μοντέλου είναι ιδιαίτερα χρήσιμη διότι μας δίνει τη δυνατότητα με αυτόματο τρόπο να επαυξήσομε τη βάση δεδομένων μας με γνώση που σχετίζεται με την ομοιότητα των λογισμικών αντικειμένων που περιέχει και που θα είναι διαθέσιμη ανά πάσα στιγμή μετά την δημιουργία της από την εφαρμογή του αλγορίθμου που προτείνομε. Αυτή η αυτόματη επαύξηση γίνεται με την εισαγωγή στην ΒΠΛ ή στην εκάστοτε βάση μας νέας πληροφορίας που αφορά λογισμικά αντικείμενα ως προς την δυνατότητα υποκατάστασης μεταξύ τους. Αυτή η πληροφορία προκύπτει από τον αλγόριθμο υπολογισμού του βαθμού υποκατάστασης μεταξύ λογισμικών αντικειμένων και εισάγεται στην ΒΠΛ τυπικά μέσω του μοντέλου που προτείνομε ευθύς αμέσως.

Στο σχήμα 1.1 βλέπομε μια σχηματική παράσταση του μοντέλου για την παράσταση και αποθήκευση σχέσεων προσέγγισης στην ΒΠΛ. Επίσης παρουσιάζεται σχηματικά και ο τρόπος με τον οποίο μπορούμε να ορίζομε περιβάλλοντα υπολογισμού τέτοιων σχέσεων (γλώσσες προγραμματισμού, κριτήρια).

Στο σχήμα 1.2 παραθέτομε το μοντέλο σε γλώσσα TELOS, ενώ στο Παράρτημα Γ δίνομε ένα παράδειγμα χρήσης του μοντέλου για την περιγραφή σχέσεων υποκατάστησης.



Σχήμα 1.1 : Παράσταση σχέσεων προσέγγισης στην ΒΠΛ

```
BEGINTRANSACTION

TELL Individual Software_Object in M1_Class
end Software_Object

TELL Individual  Parameter_Definition in M1_Class
end Parameter_Definition

TELL Attribute Proximity
	components
		from : Software_Object
		to : Software_Object
	in M1_Class
	with attribute

		weightF : Real;
		weightB : Real;

		parameters : Parameter_Definition;

		reflexive : Integer;
		symmetric : Integer;
		transitive : Integer

end Proximity

ENDTRANSACTION
```

Σχήμα 1.2 : Το μοντέλο για παράσταση σχέσεων προσέγγισης στην ΒΠΛ, σε γλώσσα TELOS.

Με την εισαγωγή των μοντέλων περιγραφής λογισμικών αντικειμένων και σχέσεων προσέγγισης επιτυγχάνομε μία τυπική περιγραφή της κλάσεως των λογισμικών αντικειμένων, ανεξάρτητη από το χρήστη ή το Μηχανικό Εφαρμογής. Ετσι, τα αποτελέσματά μας εξαρτώνται μόνο από τη συντακτική υφή των λογισμικών αντικειμένων και είναι ανεξάρτητα του πληθυσμού της ΒΠΛ.

Ανακεφαλαιώνοντας, στην παρούσα εργασία ασχολούμαστε με το πρόβλημα της εύρεσης σχέσεων υποκατάστασης μεταξύ λογισμικών αντικειμένων (και εν μέρει με την εύρεση σχέσεων ομοιότητας) με βάση την συντακτική τους περιγραφή. Προτείνομε ένα μοντέλο για την περιγραφή σχέσεων υποκατάστασης ή/και ομοιότητας μεταξύ λογισμικού, καθώς και ένα μοντέλο για την περιγραφή ερωτήσεων μεταξύ λογισμικού με βάση σύνολα κριτηρίων. Ακόμη προτείνομε και έναν αλγόριθμο για τον υπολογισμό του βαθμού υποκατάστασης μεταξύ λογισμικών αντικειμένων, του οποίου και βελτιώνομε τις επιδόσεις. Ο παραπάνω αλγόριθμος μπορεί να εφαρμοστεί σε οποιαδήποτε λογισμικά αντικείμενα δοσμένης της συντακτικής τους περιγραφής. Επίσης με βάση το μοντέλο περιγραφής της σχέσης υποκατάστασης μεταξύ λογισμικού και τις ιδιότητες της συνάρτησης υπολογισμού του βαθμού υποκατάστασης επιτυγχάνομε αυτόματη επαύξηση της βάσης μας. Οσον αφορά στα πειραματικά αποτελέσματα των δοκιμών που έγιναν, ο αλγόριθμος ανευρίσκει πάντα σχέση υποκατάστασης, εφόσον προκύπτει από το μοντέλο, με χρόνο απόκρισης πολύ ικανοποιητικό. Αυτό συμβαίνει επειδή το μοντέλο που προτείνομε είναι καθόλα τυπικό και δεν βασίζεται σε εικασίες του χρήστη ή σε πιθανές σχέσεις του λογισμικού.

1.4. Ορολογία και συμβολισμοί

Στην παρούσα εργασία χρησιμοποιούνται οι παρακάτω οντότητες και επιπλέον σχέσεις. Σε όποιο σημείο της εργασίας χρησιμοποιηθεί κάποιος νέος συμβολισμός, αυτός θα ορισθεί ρητά στο συγκεκριμένο σημείο, και θα έχει τοπικό χαρακτήρα.

Οι νέες οντότητες που στο εξής θα αναφερόμαστε αφορούν σε αντικείμενα που είναι αποθηκευμένα στην ΒΠΛ και περιγράφονται σε TELOS.

Αυτές είναι :

- βασικοί τύποι, **BT**,
- δομημένοι τύποι, **ST**,
- τύποι object, **OT**,
- ρουτίνες, διαδικασίες ή συναρτήσεις, **Proc**,
- αρχεία, **F**,
- προγράμματα, **Prog** και
- λογισμικά αντικείμενα, **O**.

Επίσης ορίζομε δύο ακόμη οντότητες για την παράσταση των κριτηρίων μας :

- βασικό κριτήριο, **IP**, και
- σύνολο κριτηρίων, **C**.

Οι δύο βασικές σχέσεις είναι :

- η σχέση ομοιότητας, **S**, και
- η σχέση υποκατάστασης, **R**.

Στο εξής όταν θα αναφερόμαστε σε μία κλάση **X** στην ΒΠΛ, με τον συμβολισμό $X.name_i$ θα αναφερόμαστε στο i-στοιχείο από το σύνολο των γνωρισμάτων (attributes) της κλάσεως X, όπως έχει περιγραφεί στην TELOS. Επίσης, με $|X|$ θα συμβολίζομε τον αριθμό των γνωρισμάτων της που έχομε περιγράψει για την κλάση X, και με $|X.name|$ τον αριθμό των στοιχείων των γνωρισμάτων της κλάσεως X.

Τελος, παραθέτομε τρεις ακόμη συμβολισμούς που θα χρησιμοποιήσομε στην πορεία της εργασίας.

IC : ανάπτυγμα συνόλου κριτηρίων.

$\mathbf{S_C}$: σχέση ομοιότητας στο σύνολο κριτηρίων C.
$\mathbf{R_C}$: σχέση υποκατάστασης στο σύνολο κριτηρίων C.

2. Βιβλιογραφική ανασκόπηση

Ενα από τα βασικά προβλήματα στην αναχρησιμοποίηση λογισμικού είναι αυτό του εντοπισμού και της ανάκτησης αντικειμένων προς αναχρησιμοποίηση. Βέβαια, ανάλογο πρόβλημα συναντάμε και στον τομέα της Αντλησης Πληροφορίας (Information Retrieval), όπου εκεί τα αντικείμενα που μας ενδιαφέρουν είναι αποκλειστικά και μόνο έγγραφα (documents).

Ηδη από το 1965 περίπου [McIlroy], που τέθηκαν τα παραπάνω προβλήματα, έχουν γίνει και υλοποιηθεί αρκετές προτάσεις, τόσο στον ένα όσο και στον άλλο χώρο έρευνας. Το κλειδί είναι πάντα η σχέση μεταξύ δύο αντικειμένων και αντιπροσωπεύει την απόσταση των Ανακτηθέντων αντικειμένων από ένα συγκεκριμένο που θεωρούμε εστιακό ή στόχο. Η σχέση αυτή άλλοτε μπορεί να αντιπροσωπεύεται από μια απάντηση της μορφής ΝΑΙ/ΟΧΙ, οπότε και αναφερόμαστε σε αναζήτηση της δυαδικής λογικής (boolean retrieval) ή να αντιπροσωπεύεται από μια απάντηση που χαρακτηρίζεται από ένα βαθμό επιτυχίας, που φανερώνει την απόσταση ενός αντικειμένου από το σύνολο της απάντησης από το εστιακό αντικείμενό μας.

Στα υποκεφάλαια που θα ακολουθήσουν θα γίνει μία μικρή αναδρομή στο παρελθόν παρουσιάζοντας με συντομία μερικές εργασίες σχετικές με την παρούσα.

2.1. Γενική ανασκόπηση

Η πρώτη από τις εργασίες που θα εξετάσομε αναφέρεται σε μια πιό ειδική περίπτωση σχέσεων αντικειμένων λογισμικού και έχει προταθεί από την S. Horwitz [Hor]. Αναφέρεται στην εύρεση εννοιολογικών και λεκτικών διαφορών μεταξύ εκδόσεων (versions) του ιδίου προγράμματος. Αυτό επιτυγχάνεται με έναν αλγόριθμο για τον καθορισμό των διαφορετικών υπολογισμών μεταξύ των δύο προγραμμάτων, συγκρίνοντας τμήματά τους. Ετσι, καθορίζεται είτε η ομοιότητα είτε η διαφορά κατ’ επέκταση μεταξύ των κομματιών. Ενα πρόγραμμα θεωρείται ως το σύνολο των κομματιών του. Δύο κομμάτια ενός προγράμματος είναι όμοια αν και μόνο αν ταυτίζονται εννοιολογικά και λεκτικά. Αυτό που γίνεται είναι η δημιουργία ενός σημασιολογικού δικτύου που παριστάνει την ροή και τον έλεγχο των δεδομένων.

Το πρόβλημα είναι ότι η πρόταση αυτή είναι δύσκολο να εφαρμοστεί για γλώσσες πιό περίπλοκες και πιό εκφραστικές από την Pascal που να υποστηρίζουν και παραμέτρους διαδικασιών, αφού θα απαιτούσε τη χρήση ενός τεράστιου σημασιολογικού δικτύου. Επίσης, δεν είναι ακόμη γνωστό αν το NP-hard πρόβλημα για την σύγκριση των κομματιών των προγραμμάτων μπορεί να λυθεί αρκετά γρήγορα.

Με το πρόβλημα του εντοπισμού και της κατανόησης των αντικειμένων προς αναχρησιμοποίηση σε ένα οντοκεντρικό περιβάλλον λογισμικού ασχολείται η πρόταση που έγινε από τον X. Pintado [Pin-Tsic90], [Pin]. Κεντρικό εργαλείο είναι ο Affinity Browser, ένα εργαλείο για επιλογή και εξερεύνηση με βάση σχέσεις μεταξύ αντικειμένων. Ο χρήστης του εργαλείου μπορεί να δει διαφορετικές όψεις (views) των αντικειμένων με βάση τις σχέσεις τους. Κάθε όψη βασίζεται σε διαφορετική συνάρτηση υπολογισμού σχέσης (μετρική) και άρα αναδεικνύει διαφορετική σχέση.

Ενα μικρό παράδειγμα χρήσης/εφαρμογής του Affinity Browser είναι η κατασκευή μιας όψης για την λειτουργική συνάφεια μεταξύ κλάσεων. Μία κλάση κληρονομεί αναδρομικά τις μεθόδους της (methods) από τις υπερκλάσεις (supeclasses) της, που αποτελούν και τον μοναδικό τρόπο επικοινωνίας της με τις άλλες κλάσεις. Εστω ότι με $M(X)$ συμβολίζομε το σύνολο των μεθόδων της X. Το μέτρο της συνάφειας μπορεί να οριστεί ως το σύνολο των μεθόδων που είναι σχετικές με τον συνολικό αριθμό των μεθόδων που έχουν οριστεί και για τις δύο κλάσεις. Μία υποψήφια μετρική μπορεί να είναι η εξής :

$$A(X, Y) = \frac{| M(X \cap Y) |}{| M(X \cup Y) |},$$

όπου η ποσότητα $A(X, Y)$ εκφράζει τη συνάφεια μεταξύ των κλάσεων X και Y. Ετσι, αν θέλαμε να περιγράψομε συντακτικά λογισμικά αντικείμενα μέσα στα πλαίσια του Affinity Browser θα έπρεπε πρώτα απ’ όλα να δίναμε ένα τρόπο περιγραφής τους υπό την μορφή συνόλων από χαρακτηριστηκά στοιχεία τους και κατόπιν να δίναμε και μία μετρική πάνω σε αυτά.

Με το πρόβλημα της ανάκτησης πληροφορίας από μία βάση δεδομένων εμπλουτισμένη με έγγραφα σε ελεύθερο κείμενο (free text), ασχολούνται οι B.

Teufel και S. Schmidt [Teu-Sch]. Τέτοια έγγραφα μπορεί να είναι ελεύθερες σημειώσεις, περιγραφές προβλημάτων, άρθρα. Μία ερώτηση σε αυτές τις βάσεις, με σκοπό την αναζήτηση και ανάκτηση πληροφορίας, θα μπορεί να περιέχει κείμενο ή λέξη σε φυσική γλώσσα, σε αντίθεση με τα κλασσικά συστήματα ανάκτησης πληροφορίας που επιτρέπουν ερωτήσεις με τη χρήση όρων (terms) από ένα περιορισμένο ευρετήριο (vocabulary ή thesaurus).

Η εργασία βασίζεται στην ανάλυση λέξεων στο κείμενο με τη βοήθεια των n-grams, που δεν είναι τίποτε άλλο από το σύνολο των υπο-λέξεων μεγέθους n, της υπ' όψιν λέξεως. Η ομοιότητα μεταξύ κειμένων σε φυσική γλώσσα μετριέται αντιστοιχίζοντας τα κείμενα σε σύνολα υπο-λέξεων, με βάση την παραπάνω ανάλυση.

Μία ιδιαίτερα ενδιαφέρουσα πρόταση στην εργασία αυτή είναι η προσπάθεια συναγωγής ομοιότητας μεταξύ φαινομενικά ανομοίων κειμένων, που οφείλεται στο ότι η σχέση ομοιότητας, όπως ορίζεται, δεν είναι μεταβατική.

Η προσπάθεια των Teufel και Schmidt αν και έχει αρκετά ενδιαφέροντα σημεία, όπως αυτό της έμμεσης συναγωγής σχέσης ομοιότητας μεταξύ φαινομενικά ανομοίων κειμένων, δεν έχει καλά πειραματικά αποτελέσματα για ερωτήσεις με έναν όρο. Επίσης, μια και βασίζεται σε ελεύθερο κείμενο και όχι σε τυπικό ορισμό εννοιών δεν είναι δυνατή η χρήση της παραπάνω πρότασης των Teufel και Schmidt σε λογισμικό και ιδιαίτερα όσον αφορά την συντακτική τους δομή μια και μπορεί να οδηγήσει σε αυθαίρετες περιγραφές αλλά και σε αργές απαντήσεις λόγω της δομής των συνόλων των υπο-λέξεων.

Η επόμενη πρόταση είναι του Goyal [Goyal], που προτείνει συγκεκριμένες ιδέες για ένα Εξυπνο Πληροφοριακό Σύστημα. Ενα τέτοιο σύστημα επιτρέπει την ταυτόχρονη παρουσίαση διαφορετικών τμημάτων ενός εγγράφου ή και άλλων που αναφέρονται σε αυτό ή και εγγράφων που χαρακτηρίζονται ως σχετικά με αυτό. Επίσης βοηθητικές δυνατότητες όπως λεξικό και συνώνυμα θεωρούνται απαραίτητες. Ετσι, προτείνει ένα Εξυπνο Πληροφοριακό Σύστημα το οποίο υποστηρίζει μία σχέση ομοιότητας με συμμετρική και μεταβατική ιδιότητα χωρίς όμως τον υπολογισμό κάποιου αντίστοιχου βαθμού. Η ομοιότητα ορίζεται ως προς τις περιοχές ενδιαφέροντος (domains) και ως προς τα έγγραφα. Ολα αυτά υποστηρίζονται με τη βοήθεια μιας γλώσσας παρόμοιας με την PROLOG και σε επίπεδο ορισμών των σχέσεων και σε επίπεδο ερωτήσεων. Με τον τρόπον αυτό είμαστε σε θέση να διασχίζομε μη-γραμμικά τα έγγραφα της βάσης, σα να ηταν έγγραφα hypertext.

Ακόμη, ένα τέτοιο σύστημα επιτρέπει την αναζήτηση και την ανάκτηση πληροφορίας σε υψηλότερα ή χαμηλότερα νοητικά ή τεχνικά επίπεδα. Αν λοιπόν υποθέσομε πως ψάχνομε για μια λέξη, ίσως να μας ενδιέφερε η χρήση της, ή τα συνώνυμά της ή άλλες λέξεις όμοιες με αυτήν. Κάθε έγγραφο, D, περιγράφεται από ένα σύνολο από τμήματα, d_i, (παραγράφους, προτάσεις, λέξεις), από ένα σύνολο από έννοιες, C_i, (concepts) και από ένα σύνολο από δεδομένα (facts), F_i, που αναφέρονται στις έννοιες. Κάθε έννοια συνδέεται με τη σειρά της με ένα σύνολο τμημάτων, $\{ d_{ij} \}$. Με τον τρόπο αυτό δημιουργείται ένα σημασιολογικό δίκτυο. Ενα μονοπάτι μεταξύ δύο εννοιών ορίζει σημασιολογικές σχέσεις μεταξύ των εννοιών.

Το πρόβλημα με την πρόταση αυτή έγκειται στη δημιουργία μεθόδων για την περιγραφή των συνδέσμων μεταξύ των εγγράφων, ώστε να εμπεριέχουν και περιγραφικές πληροφορίες σχετικές με δεδομένα που χαρακτηρίζουν τα έγγραφα, με βάση τα οποία κρίθηκαν σχετικά. Μέχρι στιγμής το σημασιολογικό δίκτυο που δημιουργείται αναφέρεται μόνο στα έγγραφα, χωρίς να επιτρέπει την άμεση πρόσβαση στα στοιχεία των εγγράφων.

Τέλος, θα αναφερθούμε σε μία αρκετά σημαντική εργασία από τον χώρο και πάλι της Αντλησης Πληροφορίας που ανήκει στους G. Salton, E. Fox και H. Wu [SalFoxWu] που δημοσιεύθηκε το 1983, και ασχολείται με τον τρόπο αναζήτησης πληροφορίας σε μία βάση δεδομένων με έγγραφα. Κάθε στοιχείο της βάσης (έγγραφο) περιγράφεται από ένα άνυσμα που το i-στοιχείο του αντιπροσωπεύει το βάρος του i αντίστοιχου όρου (term).

Το μοντέλο που προτείνουν είναι γνωστό ως extended boolean retrieval, και με τη βοήθειά του μπορούμε να σχηματίσομε and και or ερωτήσεις ή συνδυασμούς τους και να ταξινομήσομε τα έγγραφα του συνόλου της απάντησης με βάση την μετρική που επιλέγομε, ως χρήστες. Ετσι, ανάλογα με τη μετρική που επιλέγομε, στις ακραίες περιπτώσεις μας δίνει τα τυπικά αποτελέσματα γνωστών μοντέλων άντλησης πληροφορίας (boolean retrieval κ.ά.). Ακόμη, τα βάρη των όρων αναδεικνύουν τη σχετική τους σημασία στην αναζήτηση των εγγράφων. Το extended boolean retrieval σύστημα μπορεί να χρησιμοποιηθεί και σε περιβάλλοντα για ερωτήσεις σε φυσική γλώσσα.

Η εργασία των Salton, Fox και Wu στάθηκε ουσιαστική στη διαμόρφωση ορισμένων κατευθύνσεων για την παρούσα εργασία που παρουσιάζομε, γεγονός που θα φανεί στα κεφάλαια που θα ακολουθήσουν.

2.2. Η πρόταση των Prieto-Diaz και Freeman

Μια από τις πιο σημαντικές εργασίες στον τομέα της αναχρησιμοποίησης είναι αυτή των R. Prieto-Diaz και R. Freeman [PrDiaz-Fre] που πρωτοπαρουσιάστηκε το 1987. Στην παραπάνω εργασία γίνεται μία προσπάθεια να δοθούν ορισμένες απαντήσεις στο πρόβλημα της ταξινόμησης (classification problem).

Ταξινόμηση, είναι η ομαδοποίηση "ομοίων" αντικειμένων. Ετσι, η ταξινόμηση αναδεικνύει τις σχέσεις μεταξύ αντικειμένων ή κλάσεών τους, και το αποτέλεσμά της είναι ένα δίκτυο σχέσεων (semantic network). Ενα σχήμα ταξινόμησης (classification scheme) είναι ένα εργαλείο για την δημιουργία μιας διάταξης με βάση ένα κατευθυνόμενο και δομημένο λεξιλόγιο. Οι Prieto-Diaz και Freeman προτείνουν το faceted classification scheme που βασίζεται στο ότι οι συλλογές των αντικειμένων προς αναχρησιμοποίηση είναι μεγάλες και αυξάνονται γρήγορα, και ότι υπάρχουν μεγάλες ομάδες από όμοια αντικείμενα, ακόμη και για πολύ ειδικές κλάσεις.

Ετσι, για την περιγραφή ενός λογισμικού αντικειμένου κατέληξαν στην χρήση ενός σταθερού ανύσματος, αποτελούμενου από έξι στοιχεία. Τα στοιχεία του ανύσματος αυτού σκοπό έχουν να περιγράψουν τα βασικά χαρακτηριστικά του εκάστοτε λογισμικού αντικειμένου που μας ενδιαφέρει να περιγράψομε. Τα βασικά χαρακτηριστικά κάθε λογισμικού αντικειμένου προδιαγράφονται με βάση :

(α) τη λειτουργικότητα (τί κάνει), και

(β) το περιβάλλον που λειτουργεί (που κάνει αυτό που εκτελεί).

Τελικά το άνυσμα που προκύπτει περιγράφεται από τα εξής στοιχεία : την λειτουργία που επιτελεί, τα αντικείμενα που διαχειρίζεται, τον τύπο στον οποίο επεμβαίνει, το είδος του συστήματος που απαιτεί για να δράσει, την περιοχή λειτουργικότητάς του και το είδος της εφαρμογής. Για τη διαχείριση των συνωνύμων που χρησιμοποιούνται για τις περιγραφές γίνεται χρήση ενός λεξικού συνωνύμων (thesaurus).

Για την κατάταξη και την εύρεση της απόστασης των αντικειμένων χρησιμοποιείται ένας μηχανισμός κανονικοποίησης και διάταξης. Διαφορετική μετρική χρησιμοποιείται για κάθε κριτήριο (στοιχείο του σταθερού ανύσματος που αναφέραμε παραπάνω). Λόγω του ότι τα κριτήρια που επιλέχθησαν δεν έχουν ακριβείς τιμές, εντάχθηκε στο μοντέλο και ένας μηχανισμός για τον καθορισμό του βαθμού της συμμετοχής ενός αντικειμένου σε κάθε ένα από τα κριτήρια. Ο μηχανισμός αυτός έχει τις ρίζες του στη θεωρία ασάφειας (fuzzy concepts), και συγκεκριμένα στη θεωρία των ασαφών συνόλων (fuzzy set theory).

Τέλος, όσον αφορά στον σχηματισμό και στην ανάδειξη ομοίων αντικειμένων χρησιμοποιείται και ένας μηχανισμός επέκτασης ερωτήσεως (query expansion mechanism). Με λίγα λόγια συμβαίνουν τα εξής : εάν μία ερώτηση δεν επιστρέψει σύνολο ομοίων αντικειμένων, τότε το σύστημα δημιουργεί, με εντολή του χρήστη, νέες περιγραφές για την ανάκτηση ομοίων αντικειμένων. Ο χρήστης μπορεί λοιπόν να επιλέξει μεταξύ των αντικειμένων που επιτελούν παρόμοιες λειτουργίες [PrDiaz].

Η πρόταση των Prieto-Diaz και Freeman αποδεικνύεται όλο και περισσότερο ικανή στο να εκφράσει τη πραγματική υπόσταση των κλάσεων και τις σχέσεις μεταξύ τους [PrDiaz]. Βέβαια το γεγονός ότι βασίζεται σε ένα ευρετήριο όρων που θα πρέπει να επαυξάνεται κάνει τα πράγματα πιό περίπλοκα και απαιτεί χώρο και έξυπνες και γρήγορες τεχνικές ανάκτησης.

Το ερευνητικό έργο REBOOT [REBOOT] ασχολείται με την υποστήριξη της αναχρησιμοιποίησης με σύνθεση , που σημαίνει ότι ο χρήστης φτιάχνει νέα συστήματα συνθέτοντάς τα λίγο-πολύ από στοιχειώδη συνιστώσες (atomic components). Για να επιτύχει μια τέτοια προσπάθεια χρησιμοποιείται ένα σύστημα διαχείρισης βιβλιοθήκης, που να αποθηκεύει τις ιδιότητες όλων των συνιστωσών. Κεντρικό τμήμα ενός τέτοιου συστήματος διαχείρισης είναι το σχήμα ταξινόμησης. Στο έργο REBOOT επιλέχθηκε ένα faceted σχήμα ταξινόμησης, με βάση την πρόταση των Prieto-Diaz και Freeman [PrDiaz-Fre]. Το σχήμα που τελικά χρησιμοποιούν αποτελείται από τέσσερα facets που αναφέρονται στα εξής χαρακτηριστικά :

αφαιρετικότητα
: αναφέρεται στον χαρακτηρισμό ενός αντικειμένου (π.χ. stack).

πράξεις
: αναφέρεται στο σύνολο των πράξεων των συνιστωσών.

αποδέκτες
: αναφέρεται στο είδος των αντικειμένων που οι συνιστώσες μπορούν να επηρεάσουν.

εξαρτήσεις
: αναφέρεται στο σύνολο των συνιστωσών που πρέπει να είναι παρούσες για

να δουλέψει μια άλλη συνιστώσα.

Κάθε ένα από τα facets περιγράφεται τελικά με έναν όρο. Ετσι όταν ο χρήστης επιθυμεί να ψάξει για μία συνιστώσα απλά γεμίζει τα facets με τους κατάλληλους όρους. Επιπλέον υποστηρίζεται η διατήρηση σχέσεων μεταξύ σχετικών όρων. Η δυνατότητα αυτή υποστηρίζει και την απαίτηση για ανάκτηση συνιστωσών που απαιτούν ελάχιστες αλλαγές. Η έκφραση της απόστασης μεταξύ όρων γίνεται με ανάθεση βαρών στις σχέσεις.

Η πειραματική χρήση του έργου REBOOT έδειξε ότι η καλή τεκμηρίωση των συνιστωσών των λογισμικών αντικειμένων διευκολύνει την ταξινόμησή κατά πολύ. Το κύριο μειονέκτημα του έργου είναι η αδυναμία του σχήματος ταξινόμησης να παραστήσει την συνεργασία μεταξύ των συνιστωσών.

2.3. Η πρόταση των Fugini και Faustle

Η εργασία των M. Fugini και S. Faustle [ITHACA.90.E3.6.#1], [ITHACA.90.E3.6.#2], ασχολείται με το πρόβλημα της περιγραφής λειτουργικών σχέσεων μεταξύ λογισμικού. Αναπτύχθηκε στα πλαίσια του ιδίου ερευνητικού έργου (ITHACA) και βασίζεται κατά πολύ στις ιδέες των Prieto-Diaz και Freeman που παρουσιάσαμε προηγουμένως.

Οι Fugini και Faustle προτείνουν για την περιγραφή των λογισμικών αντικειμένων ένα σχήμα βασισμένο σε λέξεις-κλειδιά. Με τα γνωρίσματα (attributes) στην ΒΠΛ, καθίσταται δυνατή η περιγραφή των λειτουργικών περιγραφών (functional descriptions) των αντικειμένων προς αναχρησιμοποίηση. Το σχήμα περιλαμβάνει και ένα σύνολο από λέξεις - κλειδιά με βάρη που περιγράφουν την συμπεριφορά, τον τρόπο χρήσης και την εμπειρία χρήσης για τα αντικείμενα στην ΒΠΛ. Βέβαια στην περίπτωση αυτή ξεφεύγομε από ένα σταθερό άνυσμα περιγραφής της λειτουργικότητας του λογισμικού και αναφερόμαστε πια σε μια λίστα από λειτουργικές περιγραφές του με την βοήθεια της TELOS.

Ετσι, μία λειτουργική περιγραφή είναι μία λίστα από λειτουργίες που περιγράφουν το τι κάνει μία κλάση. Κάθε πράξη περιγράφεται ως εξής :

< ρήμα, όνομα, βάρος >
Το πεδίο ρήμα περιγράφει τον τύπο της πράξης της κλάσης και ανήκει σε ένα λεξικό από ρήματα. Το πεδίο όνομα περιγράφει τον τύπο της κλάσης πάνω στον οποίο εφαρμόζεται η πράξη. Το πεδίο βάρος περιγράφει τη σχετική συνάφεια της πράξης στη λειτουργική περιγραφή. Το άθροισμα όλων των βαρών σε μία λειτουργική περιγραφή είναι 100.

Η ομοιότητα μεταξύ δύο κλάσεων ορίζεται ως η ένδειξη του πόσο εύκολα, με βάση την όλη εφαρμογή, μια κλάση μπορεί να αντικατασταθεί από μια άλλη κατά την δημιουργία μιας εφαρμογής. Η ομοιότητα ορίζεται λειτουργικά ως η πιθανότητα μία κλάση C_1 να αντικατασταθεί από την κλάση C_2, διατηρώντας τις απαιτήσεις ή τα κριτήριά μας. Η ομοιότητα διατηρεί την ανακλαστική ιδιότητα.

Ο βαθμός της ομοιότητας τελικά δίνεται από τον εξής τύπο :

$$Sim_{1-2} = \frac{CV_{1-2} + CV_{2-1}}{2},$$

όπου η ποσότητα CV_{X-Y} είναι ένας πραγματικός αριθμός στο διάστημα [0, 1] που εκφράζει το βαθμό εμπιστοσύνης στην αντικατάσταση της κλάσεως X από την κλάση Y.

Ενας μηχανισμός ερωτήσεων έχει αναπτυχθεί, δίνοντας τη δυνατότητα για ερωτήσεις ομοιότητας στο χρήστη. Οι ερωτήσεις που μπορούν να γίνουν έχουν την παρακάτω μορφή :

- πόσο κοντά είναι οι κλάσεις C_1 και C_2,
- ποιά αντικείμενα είναι πιο κοντά στη C_1 από τη C_2,
- ποιές άλλες κλάσεις είναι κοντά στη C_1, και
- γιατί μία κλάση C_2, για παράδειγμα, είναι κοντά στη C_1.

Στην εργασία των Fugini και Faustle η λειτουργική περιγραφή των κλάσεων των αντικειμένων λογισμικού δε γίνεται αυτόματα ούτε μέσω ενός ελεγχόμενου τρόπου, αλλά από τον Μηχανικό Εφαρμογής (application engineer). Το σύνολο των ρημάτων που θα χρησιμοποιήσει για την περιγραφή των κλάσεων άπτεται της κρίσης του ή του βαθμού λεπτομέρειας που θέλει να δώσει στην περιγραφή του κάθε λογισμικού αντικειμένου. Κατι ανάλογο συμβαίνει και με το λεξικό συνωνύμων, όπου το πλήθος των λειτουργικών περιγραφών που περιέχει πρέπει να αντανακλά και την πληροφορία που υπάρχει ήδη στις αντίστοιχες λειτουργικές περιγραφές αλλιώς η υπαρξή του είναι περιττή. Η ανάθεση βαρών στα ρήματα των λειτουργικών περιγραφών γίνεται είτε αυτόματα μέσω του συστήματος που κανονικοποιεί και ισοκατανέμει τα βάρη σε περίπτωση μη συμπλήρωσής τους (αυτόματη ανάθεση), είτε από τον Μηχανικό Εφαρμογής (υποκειμενική ανάθεση βαρών). Εχοντας λοιπόν απ’ τη μια μεριά έναν υποκειμενικό τρόπο ανάθεσης βαρών και από την άλλη μια μέθοδο που θεωρεί όλα τα ρήματα ισότιμα, οποιαδήποτε επιλογή μας δεν είναι σε καμμιά περίπτωση δίκαιη για τις περιγραφές με κριτήριο την υποψηφιότητά τους για αναχρησιμοποίηση. Η ανάθεση βαρών με κριτήριο δικαιοσύνης (ισοκατανομή) θα μπορούσε να εξυπηρετήσει μόνο στόχους περιήγησης στο σύνολο των αντικειμένων.

Οσον αφορά στις διαφορές που συναντάμε από την πρόταση των Prieto_Diaz και Freeman, υπάρχουν πολλά μειονεκτήματα, ειδικά στον τρόπο διαχείρισης των ερωτήσεων και στον υπολογισμό του βαθμού ομοιότητας. Δεν αναφέρεται κανένας μηχανισμός επέκτασης ερωτήσεων, που θα μπορούσε με την υποστήριξη του λεξικού των συνωνύμων να δώσει αποτέλεσμα. Ακόμη δεν υπάρχει πρόβλεψη για τον χειρισμό ασάφειας στις λειτουργικές περιγραφές.

Η πρόταση των Prieto_Diaz και Freeman κατά κύριο λόγο και λιγότερο των Fugini και Faustle, αντιμετωπίζουν το πρόβλημα της ομοιότητας λογισμικού με διαφορετική σκοπιά από ότι εμείς, μια και αναφερόμαστε μόνο σε συντακτικές περιγραφές, χωρίς να λαμβάνομε υπ’ όψιν μας και άλλες πιθανές σχέσεις ήδη τεκμηριωμένες που πηγάζουν από τον χώρο εφαρμογής ή και από την ίδια την λειτουργική περιγραφή του κάθε αντικειμένου.

3. Συντακτική υποκατάσταση λογισμικών αντικειμένων

Σκοπός μας στην παρούσα εργασία είναι η συναγωγή συντακτικής σχέσεως ομοιότητας μεταξύ λογισμικών αντικειμένων. Ενα λογισμικό αντικείμενο χαρακτηρίζεται συντακτικά από στοιχεία που αφορούν στην περιγραφή του ως υλοποίηση και κώδικα, και όχι την συμπεριφορά του. Σκοπός μας είναι να περιγράψομε την συντακτική υφή του λογισμικού όσο το δυνατό πιο τυπικά ώστε να περιορίσομε τυχόν υποκειμενικές περιγραφές του που θα δυσκόλευαν τα αποτελέσματά μας. Ετσι, καταλήξαμε στον ορισμό μιας πιο γενικής σχέσης που την ονομάζομε σχέση υποκατάστασης.

Στο κεφάλαιο αυτό θα ορίσομε λοιπόν τις δύο βασικές σχέσεις που θα χρησιμοποιήσομε και κατόπιν θα περιγράψομε αναλυτικά τις μετρικές που προτείνομε για τις παραπάνω σχέσεις, πάνω στο σύνολο των διαφορετικών οντοτήτων που χειριζόμαστε. Επίσης, θα ασχοληθούμε και με τις ιδιότητες που προκύπτουν για τις παραπάνω σχέσεις σύμφώνα με τις μετρικές που προτείναμε, και θα μελετήσομε τις συνθήκες για τις οποίες ισχύουν οι ιδιότητες αυτές.

3.1. Σχέση υποκατάστασης

Η σχέση υποκατάστασης, R, δέχεται ως ορίσματα οποιοδήποτε ζεύγος ομοειδών οντοτήτων από το σύνολο αυτών που έχουν οριστεί στην παράγραφο 1.4 (τύποι, ρουτίνες, αρχεία ...). Οι τιμές της R ανήκουν στο διάστημα [0, 1]. Μία τέτοια σχέση ζητάμε να δίνει τιμή 1 όταν τα υπ' όψιν αντικείμενα ταυτίζονται ή όταν το δεύτερο όρισμα μπορεί να υποκαταστήσει πλήρως το πρώτο σύμφωνα με κάποιους κανόνες που δίνομε για κάθε μία από τις οντότητες ξεχωριστά. Οταν η υποκατάσταση είναι αδύνατη, τότε η σχέση παίρνει τιμή 0. Αρα έχομε :

$$R = R_{BT} \cup R_{ST} \cup R_{O},$$

όπου :

$$R_{BT} = BT \times BT \longrightarrow [0, 1]$$

$$R_{ST} = ST \times ST \longrightarrow [0, 1]$$

$$R_{O} = O \times O \longrightarrow [0, 1]$$

Για την R ισχύουν οι παρακάτω ιδιότητες, όπως προκύπτει από τα επόμενα υποκεφάλαια :

- για κάθε $X \in SObjs \ : \ R(X, X) = 1$,

- υπάρχουν $X, Y \in SObjs \ : \ R(X, Y) \neq R(Y, X)$.

Βλέπομε δηλαδή, πως η R δεν είναι ούτε συμμετρική ούτε αντισυμμετρική. Αυτό οφείλεται στον τρόπο ορισμού του βαθμού υποκατάστασης μεταξύ των βασικών τύπων που θα δώσομε αναλυτικά στο υποκεφάλαιο 3.3, και παρουσιάζονται στον Πίνακα 3.1 .

3.2. Σχέση ομοιότητας

Με βάση τον ορισμό της σχέσεως υποκατάστασης που είδαμε στο προηγούμενο κεφάλαιο θα ορίσομε και τη σχέση ομοιότητας. Ετσι, αν σε ένα σύνολο αντικειμένων λογισμικού A ισχύει :

$$\text{για κάθε } X, Y \in A \ : \ R(X, Y) = R(Y, X),$$

τότε λέμε ότι για τα στοιχεία του συνόλου A ισχύει η σχέση ομοιότητας, S, με βαθμό $S(X, Y) = R(X, Y)$.

Αρα για την S ισχύει :

- για κάθε $X \in A \ : \ S(X, X) = 1$,

- για κάθε $X \in A \ : \ S(X, Y) = S(Y, X)$.

3.3. Τύποι δεδομένων και σχέσεις μεταξύ αυτών

Εξετάζοντας το μοντέλο που περιγράφομε στο Παράρτημα Α για λογισμικά αντικείμενα παρατηρούμε ότι για την περιγραφή όλων των πιθανών λογισμικών αντικειμένων είναι απολύτως απαραίτητες οι κλάσεις των βασικών και των δομημένων τύπων, μια και βάσει αυτών περιγράφονται όλες οι παράμετροι και οι μεταβλητές των αντικειμένων λογισμικού. Πριν λοιπόν αρχίσομε να αναλύομε την συντακτική περιγραφή των λογισμικών αντικειμένων θα πούμε λίγα πράγματα για τα απλούστερα κομμάτια τους, τους τύπους.

Με βάση την εμπειρία μας και τη εκφραστικότητα καθενός από τους βασικούς τύπους μπορούμε να πούμε τα εξής :

- μερικοί βασικοί τύποι είναι από τη φύση τους πιό γενικοί (εκφραστικοί) από κάποιους άλλους, π.χ. οι ρητοί είναι πιο εκφραστικοί (υπερσύνολο) από τους φυσικούς. Ετσι, μία μεταβλητή που ο τύπος της είναι π.χ. double σίγουρα μπορεί να εκφράσει ό,τι και μία μεταβλητή integer.

- αν μία μεταβλητή X είναι διεύθυνση σε ένα τύπο Y, και μία μεταβλητή Z είναι τύπου Y, τότε η μεταβλητή Z μπορεί να αντικατασταθεί από την X, και αντίστροφα (παρατηρείστε πως εδώ γίνεται αναφορά σε μεταβλητές και όχι σε τύπους, γιατί χειριζόμαστε, όσον αφορά στη μεταβλητή X, διευθύνσεις μνήμης μεταβλητών, που έχουν κάποιο τύπο και όχι τον ίδιο τον τύπο αυτούσιο).

Με βάση τα παραπάνω ορίζομε για το σύνολο των βασικών τύπων τη σχέση υποκατάστασης, R ως :

$$R(X, Y) = Basic(X, Y), \quad \text{για} \quad X, Y \in BT$$

Για τις τιμές της Basic ισχύει :

$$Basic(i, j) = \begin{cases} 1 & \text{αν ο τυπος } j \text{ μπορεί να υποκαταστησει τον } i \\ 0 & \text{αλλιως} \end{cases}$$

όπου : $i, j \in BT$.

Οι τιμές της Basic δίνονται στον Πίνακα 3.1 .

Basic	INT	UINT	SINT	USINT	FLOAT	DOUBLE	BOOL	CHAR
INT	1	0	0	0	1	1	0	0
UINT	1	1	0	0	1	1	0	0
SINT	1	0	1	0	1	1	0	0
USINT	1	1	1	1	1	1	0	0
FLOAT	0	0	0	0	1	1	0	0
DOUBLE	0	0	0	0	0	1	0	0
BOOL	1	1	1	1	1	1	1	1
CHAR	1	1	1	1	1	1	0	1

Πίνακας 3.1

Η συνάρτηση Basic του βαθμού υποκατάστασης για βασικούς τύπους.

Μπορεί να αναρωτηθεί κανείς για το είδος των τιμών των στοιχείων του πίνακα Basic. Γιατί δηλαδή είναι διακριτές αντί να είναι συνεχείς, αφού οι τιμές της R ανήκουν στο [0, 1]. Η εξήγηση είναι αρκετά απλή και με πρακτική σημασία.

Εστω ότι θέλομε να ανακαθορίσομε τις τιμές του πίνακα Basic για τους βασικούς τύπους, ώστε οι τιμές του να είναι συνεχείς. Γνωρίζομε βέβαια ότι ένας double είναι πιο εκφραστικός από έναν float, για παράδειγμα, όμως και οι δύο υποκαθιστούν πάντα έναν integer. Πρέπει λοιπόν να καταφύγομε στα στοιχεία της αρχιτεκτονικής του υπολογιστή που χρησιμοποιούμε για την ανάπτυξη της εφαρμογής μας για να μάθομε πόσο περισσότερα bit χρησιμοποιούνται για την παράσταση ενός double από ενός ρητού; Αλλά και στην περίπτωση που ξέραμε κάτι τέτοιο τι νόημα θα είχε η χρησιμοποίηση ενός μέτρου σύγκρισης που αλλάζει από μηχανή σε μηχανή;

Τέλος, ορίζομε δύο ποσότητες που παρουσιάζουν περισσότερο στατιστικό ενδιαφέρον, για κάθε τύπο χωριστά :

- $E_i = \frac{\sum_{j \in BT-\{i\}} Basic(i, j)}{|BT-\{i\}|}$, και

- $G_j = \frac{\sum_{i \in BT-\{j\}} Basic(i, j)}{|BT-\{j\}|}$.

Η ποσότητα G_i εκφράζει το πόσο γενικός είναι ο τύπος i σε σχέση με το σύνολο των άλλων βασικών τύπων.

Η ποσότητα E_j εκφράζει την ευκολία παράστασης του τύπου j από το σύνολο των άλλων βασικών τύπων. Για παράδειγμα, το ότι έχομε $G_{DOUBLE} = 1$ και $E_{DOUBLE} = 0$, φανερώνει ότι μία μεταβλητή τύπου DOUBLE μπορεί να υποκαταστήσει οποιαδήποτε μεταβλητή άλλου βασικού τύπου, χωρίς να μπορεί να συμβεί το αντίστροφο.

Η κλάση που ακολουθεί είναι η κλάση των δομημένων τύπων. Με βάση την κλάση αυτή μπορούμε και ορίζομε τις μεταβλητές δομημένων τύπων. Στην κλάση αυτή ανήκουν οι παρακάτω τύποι :

- πίνακες μιας ή και περισσοτέρων διαστάσεων,
- εγγραφές (records),
- ενώσεις (unions),
- σύνολα (sets),
- λίστες (lists), και
- τύπος διαδικασίας (procedure type).

Πριν αναφερθούμε χωριστά στα στοιχεία που ορίζουν και περιγράφουν την καθεμιά από τις παραπάνω κατηγορίες δομημένων τύπων θα αναφέρομε πρώτα ορισμένες γενικές αρχές που καθορίζουν πότε ένας δομημένος τύπος (δηλαδή μία μεταβλητή αυτού) μπορεί να υποκαταστήσει έναν άλλο. Ετσι, αν οι X και Y είναι δομημένοι τύποι, τότε :

1. Τύποι Πινάκων

α. Αν ο τύπος X είναι πίνακας διάστασης 1, με αριθμό στοιχείων N_X και με τύπο στοιχείων T_X και ο Y πίνακας διάστασης 1, με αριθμό στοιχείων N_Y και με τύπο στοιχείων T_Y, τότε :

$$R(X,Y) = \begin{cases} 1 & N_Y \geq N_X \text{ και } R(T_Y, T_X) = 1 \\ 0 & \text{αλλιως} \end{cases}$$

β. αν ο X είναι πίνακας διάστασης D_X και ο Y πίνακας διάστασης D_Y, και με $X.elements_i$ συμβολίζομε τον τύπο των στοιχείων του πίνακα Y στην διάσταση i, και με $Y.elements_i$ συμβολίζομε τον τύπο των στοιχείων του πίνακα Y στην διάσταση i, τότε :

$$R(X,Y) = \begin{cases} 1 & D_X = D_Y \text{ και } R(X.element_i, Y.element_i) = 1 \text{ για καθε } i = 1,\ldots,D_X \\ 0 & \text{αλλιως} \end{cases}$$

2. Τύποι και διευθύνσεις

Αν ο X είναι NIL και ο Y είναι διεύθυνση, τότε :

$$R(X,Y) = 1.$$

3. Τύποι συνόλων

α. Αν ο X είναι κενό σύνολο (ή λίστα) και ο Y είναι οποιοδήποτε σύνολο (ή λίστα) τότε :

$$R(X,Y) = 1.$$

β. αν ο X είναι σύνολο (ή λίστα) με τύπο στοιχείων T_X και ο Y είναι σύνολο (ή λίστα) με τύπο στοιχείων T_Y, τότε :

$$R(X,Y) = \begin{cases} 1 & R(T_{X,}\, T_Y) = 1 \\ 0 & \text{αλλιως} \end{cases}$$

4. Τύποι εγγραφών

Αν ο X είναι τύπος εγγραφής με πλήθος πεδίων D_X και ο Y είναι τύπος εγγραφής με πλήθος πεδίων D_Y, και με $X.field_i$ συμβολίζομε τον τύπο του i πεδίου του X, και με $Y.field_i$ συμβολίζομε τον τύπο του i πεδίου του Y, τότε :

$$R(X,Y) = 1, \text{ αν } D_X \le D_Y$$

και υπαρχει μετάθεση των πεδίων του Y ωστε

$$R(X.field_i, Y.field_i) = 1 \text{ για καθε } i = 1,...,D_X$$

$$R(X,Y) = 0, \text{ αλλιως}$$

5. Τύποι ενώσεων

Αν ο X είναι τύπος ένωσης και ο Y είναι τύπος ένωσης, τότε μεταξύ τους ισχύει ό,τι και για τους τύπους εγγραφής, δηλαδή :

$$R(X,Y) = 1, \text{ αν } D_X \le D_Y$$

και υπαρχει μετάθεση των πεδίων του Y ωστε

$$R(X.field_i, Y.field_i) = 1 \text{ για καθε } i = 1,...,D_X$$

$$R(X,Y) = 0, \text{ αλλιως}$$

Τέλος, δίνομε τον πίνακα, Struct που δίνει τον βαθμό υπακατάστασης μεταξύ των διαφορετικών κατηγοριών των δομημένων τύπων. Η ένδειξη ?, σημαίνει ότι η τιμή του αντίστοιχου πεδίου μπορεί να γίνει 0 ή 1, ανάλογα με τη συγκεκριμένη περίπτωση των μεταβλητών των δύο τύπων που εξετάζομε. Ετσι έχομε :

$$R(X, Y) = Struct(X, Y), \quad \text{για} \quad X, Y \in ST$$

Για τις τιμές της Struct ισχύει :

$$Struct(i, j) = \begin{cases} 1 & \text{αν ο τυπος } j \text{ μπορεί να υποκαταστησει τον } i \\ 0 & \text{αλλιως} \end{cases}$$

όπου : $i, j \in ST$.

Οι τιμές της Struct δίνονται στον Πίνακα 3.2 .

Ολοι οι τύποι στους οποίους αναφερθήκαμε παραπάνω ορίζουν το σύνολο τύπων δεδομένων, DATATYPES, που συναντάμε στην περιγραφή της συντακτικής δομής των προγραμμάτων. Ο Πίνακας 3.2 μας δίνει αρκετά συνοπτικά τις περιπτώσεις που έχει νόημα να υπολογίσομε τον βαθμό υποκατάστασης δύο δομημένων τύπων σε περίπτωση που δεν τον γνωρίζομε ήδη ή τις περιπτώσεις που δεν έχει νόημα ο υπολογισμός ενός τέτοιου βαθμού (οι μεταβλητές της συνάρτησης Struct για τις οποίες παίρνει τιμή 0).

Η τιμή της συνάρτησης $R_{DATATYPES}$, δεδομένου ότι θέλομε να την υπολογίσομε για τα αντικέιμενα X και Y, υπολογίζεται ως εξής :

$$R_{DATATYPES}(X, Y) = \frac{\sum_{i=1}^{|X.datatypes|} R_i(X.datatypes_i, Y.datatypes_i)}{|X.datatypes|}$$

όπου X.datatypes είναι το σύνολο των τύπων δεδομένων που ορίσαμε για το αντικείμενο X.

Struct	Array	Record	Set	List	Union
Array	1	0	1	1	0
Record	?	1	?	?	1
Set	0	0	1	0	0
List	0	0	1	1	0
Union	?	0	?	?	1

Πίνακας 3.2

Η συνάρτηση Struct του βαθμού υποκατάστασης μεταξύ δομημένων τύπων.

3.4. Τύποι αντικειμένων

Στο σύνολο των τύπων για την περιγραφή των αντικειμένων λογισμικού ανήκουν και οι τύποι αντικειμένων (object types), οντότητες με δυναμική συμπεριφορά. Οι τύποι αντικειμένων χρησιμοποιούνται σε οντοκεντρικές γλώσσες προγραμματισμού, όπως στις γλώσσες C++, CooL, και η δομή τους έχει ορισμένα χαρακτηριστικά, κοινά μεταξύ γλωσσών προγραμματισμού, με βάση τα οποία θα θέσομε τα κριτήρια για τη σχέση υποκατάστασης μεταξύ τους.

Ενας τύπος αντικειμένων χαρακτηρίζεται από :

- τις υπερκλάσσεις (superclasses) του, P,
- το σύνολο των παραμέτρων του με τις οποίες επικοινωνεί με το εξωτερικό περιβάλλον, που ανήκουν στο σύνολο των public (δημοσίων) μελών του. Οι παράμετροι αυτές είναι τριών ειδών (εισόδου, IN, μεταφοράς, IO, και εξόδου, OUT) και αποτελούν τον τρόπο κάθε αντικειμένου για να αρχικοποιηθεί ή για να αλλάξει τιμές σε ιδιωτικές του μεταβλητές, και
- το σύνολο των public μεθόδων του, M, που αποτελούν τον τρόπο επικοινωνίας του με τα άλλα αντικείμενα (το interface του με το εξωτερικό

περιβάλλον). Μέσω των public μεθόδων μία μεταβλητή τύπου αντικειμένου εκτελεί κώδικα και έχει δυναμική συμπεριφορά.

Από τα παραπάνω γίνεται σαφές πως οποιαδήποτε προσπάθειά μας για σύγκριση τύπων αντικειμένων με βασικούς ή και δομημένους τύπους δεν έχει απολύτως κανένα νόημα, αντίθετα με τη σύγκριση τύπων αντικειμένων μεταξύ τους.

Αν ο X είναι τύπος αντικειμένων και ο Y είναι τύπος αντικειμένων, τότε ο βαθμός υποκατάστασης του X από το Y είναι :

$$R_{OBJTYPES}(X, Y) = \frac{\sum_{i \in P} R_i(X, Y)}{|P|} = \frac{\sum_{i \in P} R_i(X, Y)}{5},$$

όπου $P = \{SC, IN, IO, OUT, M\}$. Η ποσότητα $R_i(X, Y)$ εκφράζει την σχέση υποκατάστασης ως προς το i κριτήριο (superclasses, παραμέτροι εισόδου, μεταφοράς, εξόδου και μέθοδοι). Οι ποσότητες $R_i(X, Y),\ i \in P - \{SC\}$ υπολογίζονται ομοιόμορφα, ως εξής :

$$R_i(X, Y) = \frac{\sum_{j=1}^{|X.i|} R(X.i_j, Y.i_j)}{|X.i|}$$

Οσον αφορά στην ποσότητα R_{SC} έχομε :

$$R_{SC}(X, Y) = \frac{|path(X) \cap path(Y)|}{|path(X) \cup path(Y)|},$$

όπου το σύνολο $path(A)$ για έναν τύπο αντικείμενο A ορίζεται ως εξής :

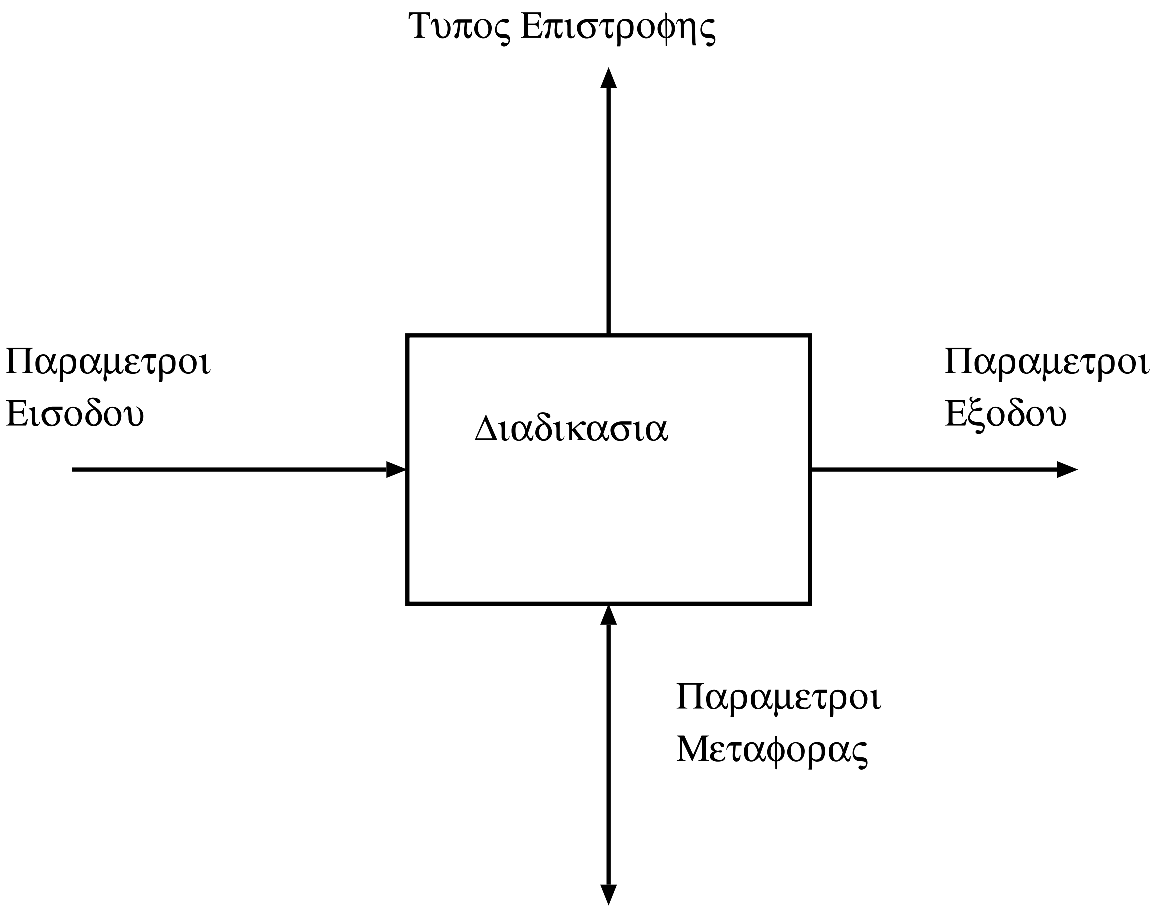
$path(A) =$ { το σύνολο των κόμβων που ανήκουν στο μονοπάτι από τη ρίζα της ιεραρχίας μέχρι το Α }.

Στους ορισμούς των παραπάνω συναρτήσεων παρατηρούμε ότι γίνεται χρήση και άλλων συναρτήσεων που δεν έχουν ήδη οριστεί. Για παράδειγμα, ενώ γίνεται χρήση της $R_M(X, Y)$, αυτή δεν έχει μέχρι στιγμής οριστεί. Η συνάτρηση αυτή υπολογίζει το βαθμό υποκατάστασης μεταξύ ρουτινών, και θα οριστεί στο υποκεφάλαιο 3.5 . Το γεγονός αυτό δίνει ήδη μία πρώτη υποψία για τον αναδρομικό χαρακτήρα της συνάρτησης υπολογισμού του βαθμού υποκατάστασης μεταξύ λογισμικών αντικειμένων(τύπων αντικειμένου, διαδικασιών, προγραμμάτων). Μια τέτοια συμπεριφορά της παραπάνω συνάρτησης είναι αναμενόμενη αφού ένα λογισμικό αντικείμενο μπορεί να αποτελείται από σύνολο άλλων με πιθανές αναφορές σε άλλα, κ.ο.κ. . Οσον αφορά δε στις υπόλοιπες συναρτήσεις, R_{IN}, R_{IO}, και R_{OUT} έχουν και αυτές αναδρομικό χαρακτήρα μια και μπορεί να αναφέρονται είτε σε βασικούς ή δομημένους τύπους, είτε σε τύπους αντικειμένων.

3.5. Συναρτήσεις, διαδικασίες και σχέσεις μεταξύ τους

Μία διαδικασία γενικά μπορούμε να τη φανταστούμε σαν ένα μαύρο κουτί, που δεν ξέρομε τι λειτουργίες κάνει και πώς τις κάνει, στην περίπτωση που ασχολούμαστε με τη συντακτική της δομή. Αυτό που είμαστε σε θέση να γνωρίζομε μόνο είναι το σύνολο των παραμέτρων (εισόδου, εξόδου, μεταφοράς και τύπος επιστραφής) με τις οποίες επικοινωνεί με το εξωτερικό περιβάλλον στην εφαρμογή μας (σχήμα 3.1). Αυτή η προσέγγιση αντικατοπτρίζει ακριβώς τον τρόπο με τον οποίο χειριζόμαστε τα αντικείμενα λογισμικού στο μοντέλο μας και στην παρούσα εργασία. Σκοπός μας δεν είναι η σημασιολογική ανάλυση τους αλλά η συντακτική, με βάση την οποία θα συνάγομε σχέσεις υποκατάστασης και ομοιότητας.

Ας πούμε όμως λίγα λόγια για κάθε είδος παραμέτρων μιας διαδικασίας, που στην περίπτωση που δηλώνεται τύπος επιστροφής λέγεται συνάρτηση :



Σχήμα 3.1 : Το σύνολο των παραμέτρων μιας διαδικασίας.

- Οι παράμετροι εισόδου, *IN*, χρησιμοποιούνται μόνο για να προσφέρουν τιμές που θα χρησιμοποιηθούν από τη διαδικασία.
- Οι παράμετροι μεταφοράς, *IO*, προσφέρουν τιμές που θα χρησιμοποιηθούν από τη ρουτίνα ως είσοδος αλλά και ως φορείς νέων τιμών έξω από τη διαδικασία.
- Οι παράμετροι εξόδου, *OUT*, χρησιμοποιούνται μόνο για να μεταφέρουν τιμές που θα χρησιμοποιηθούν έξω από τη διαδικασία.

- Ο τύπος επιστροφής, RT, αν υπάρχει, είναι μοναδικός και στην περίπτωση αυτή αναφερόμαστε σε συνάρτηση. Δηλαδή, η συνάρτηση είναι ειδική περίπτωση διαδικασίας. Ο τύπος επιστροφής μπορεί να χρησιμοποιηθεί έξω από τη συνάρτηση.

Εστω ότι X και Y είναι δύο διαδικασίες. Τότε η συνάρτηση που δίνει το βαθμό υποκατάστασης της X από την Y είναι :

$$R_{PROC}(X, Y) = \frac{\sum_{i \in P} R_i(X, Y)}{| P |} = \frac{\sum_{i \in P} R_i(X, Y)}{4},$$

όπου $P = \{IN, IO, OUT, RT\}$. Η ποσότητα $R_i(X, Y)$ εκφράζει την σχέση υποκατάστασης ως προς το i κριτήριο (παραμέτροι εισόδου, μεταφοράς, εξόδου και τύπος επιστροφής). Οι παραπάνω συναρτήσεις υπολογίζονται ομοιόμορφα, ως εξής :

$$R_i(X, Y) = \frac{\sum_{j=1}^{|X.i|} R(X.i_j, Y.i_j)}{| X.i |}$$

όπου, $i \in P$.

Ενα πρώτο ερώτημα που μας απασχολεί κοιτάζοντας του τύπους που δώσαμε παραπάνω, είναι ο τρόπος με τον οποίο θα συγκρίνομε τα στοιχεία των παραμέτρων σε κάθε λίστα. Εμεις προτείνομε παρακάτω τρεις πιθανούς τρόπους συσχέτισης των στοιχείων τους.

(α) την ένα προς ένα σύγκριση των στοιχείων για κάθε παράμετρο, όπως εμφανίζονται στη σειρά των δηλώσεων στην TELOS,

(β) όλους τους δυνατούς συνδυασμούς ανά ζεύγη των στοιχείων που ανήκουν στη συγκεκριμένη παράμετρο που κοιτάμε, ώστε το ένα στοιχείο του ζεύγους να ανήκει στο X και το άλλο στο Y, και

(γ) το ψάξιμο by-name, δηλαδή με κλειδί το όνομα του στοιχείου στην λίστα της παραμέτρου.

3.6. Αρχεία, προγράμματα και σχέσεις μεταξύ τους

Αφού περιγράψαμε τις απλούστερες οντότητες που χαρακτηρίσαμε ως αντικείμενα λογισμικού, τώρα θα δώσομε την συνάρτηση υπολογισμού του βαθμού υποκατάστασης μεταξύ προγραμμάτων, που είναι και ο τελικός στόχος μας. Δεν θα διαχωρίσομε τη συνάρτηση αυτή μεταξύ αρχείων και προγραμμάτων γιατί και ο χειρισμός τους από το μοντέλο μας είναι ομοιόμορφος, όπως είδαμε.

Ετσι, ένα πρόγραμμα χαρακτηρίζεται από :

- Το σύνολο των τύπων δεδομένων, $DATATYPES$, που ορίζονται για τις ανάγκες του συγκεκριμένου προγράμματος. Οι τύποι αυτοί ορίζονται μέσω των ήδη ορισμένων από το μοντέλο, βασικών και δομημένων τύπων.
- Το σύνολο των τύπων αντικειμένων, $OBJTYPES$, που σχεδιάστηκαν για το πρόγραμμα.
- Το σύνολο των διαδικασιών, $PROC$, που χρησιμοποιούνται από το πρόγραμμα και ανήκουν στο ίδιο αρχείο.
- Το σύνολο των αναφορών, $REFS$, σε άλλα αρχεία που περιέχουν χρήσιμα στο πρόγραμμα κομμάτια.

Εστω ότι X και Y είναι δύο προγράμματα. Τότε η συνάρτηση που δίνει το βαθμό υποκατάστασης του X από το Y είναι :

$$R(X, Y) = \frac{\sum_{i \in P} R_i(X, Y)}{| P |} = \frac{\sum_{i \in P} R_i(X, Y)}{5},$$

όπου $P = \{ DATATYPES, OBJTYPES, PROC, REFS, MAIN \}$, δηλαδή το σύνολο των κριτηρίων για τη σχέση υποκατάστασης, και όπου η ποσότητα $R_i(X, Y)$ εκφράζει την σχέση υποκατάστασης για το κάθε κριτήριο που έχομε αναφέρει παραπάνω.

Είναι λοιπόν φανερός ο **αναδρομικός χαρακτήρας** του υπολογισμού του βαθμού υποκατάστασης μεταξύ προγραμμάτων. Οσον αφορά δε τις υπόλοιπες συναρτήσεις, $R_{DATATYPES}$, $R_{OBJTYPES}$, R_{PROC}, R_{REFS} και R_{MAIN} έχουν και αυτές αναδρομικό χαρακτήρα όπως ήδη έχει αναφερθεί. Οσον αφορά το κριτήριο MAIN, αυτό αναφέρεται μόνο σε προγράμματα και όχι σε αρχεία που έχουν απλώς διαδικασίες, μια και υποκαθιστά τον ρόλο της μοναδικής main διαδικασίας στην αρχή του προγράμματος μια εφαρμογής. Οσον αφορά στον υπολογισμό της συνάρτησης R_{REFS} ισχύει :

$$R_{REFS}(X, Y) = \frac{\sum_{i=1}^{|X.references|} R(X.references_i, Y.references_i)}{|X.references|}$$

όπου $X.references_i$ και $Y.references_i$ συμβολίζομε το i στοιχείο του γνωρίσματος references για τα αντικείμενα X και Y αντίστοιχα. Η παραπάνω συνάρτηση δεν οδηγεί σε άπειρη αναδρομή όπως φαίνεται με μια πρώτη ματιά γιατί πάντα θα υπάρχει τουλάχιστον ένα πρόγραμμα χωρίς αναφορές στην εφαρμογή που εξετάζομε.

Οσον αφορά στον υπολογισμό της συνάρτησης R_{MAIN}, δεν υπάρχει καμμία ιδιαίτερη ιδιομορφία, ούτε και χρησιμοποιείται καμμία ιδιαίτερη συνάρτηση. Ετσι,ισχύει :

$$R_{MAIN}(X, Y) = R_{PROC}(X.main, Y.main)$$

3.7. Μεταβατικός υπολογισμός του βαθμού υποκατάστασης

Στα προηγούμενα κεφάλαια δώσαμε την πλήρη περιγραφή της σχέσης υποκατάστασης, R, και της συνάρτησης υπολογισμού της. Είδαμε επίσης και δύο ιδιότητές της : την ανακλαστική και τη μη-συμμετρική. Θα θέλαμε λοιπόν να εξετάσομε αν η R χαρακτηρίζεται και από κάποιες άλλες ιδιότητες, όχι τόσο προφανείς από τον ορισμό της. Μία πρώτη ιδιότητα είναι η **μεταβατική**, που η ύπαρξή της θα είχε μεγάλη σημασία για τον στόχο της αναχρησιμοποίησης λογισμικού.

Η ύπαρξη της μεταβατικής ιδιότητας σε σχέσεις προσέγγισης, όπως είναι οι σχέσεις υποκατάστασης και ομοιότητας, που ασχολούμαστε εδώ, έχει ιδιάζουσα σημασία μια και οδηγεί μέσω της συνάρτησης υπολογισμού της στην αυτόματη επαύξηση της γνώσης μας όσον αφορά το σύνολο των αντικειμένων που σχετίζονται μέσω τέτοιων σχέσεων προσέγγισης. Οντως, αν αποδεικνύαμε ότι για την σχέση υποκατάστασης, R, ίσχυε η μεταβατική ιδιότητα, αυτό αυτόματα θα σήμαινε ότι αν γνωρίζαμε τον βαθμό υποκατάστασης μεταξύ των αντικειμένων X και Y, $R(X,Y)$, και τον βαθμό υποκατάστασης μεταξύ των αντικειμένων Y και Z, $R(Y,Z)$, τότε θα μπορούσαμε να υπολογίσομε τον βαθμό υποκατάστασης μεταξύ των αντικειμένων X και Z, $R(X,Z)$, χωρίς να πυροδοτήσομε τον αλγόριθμο για τον υπολογισμό του μέσω των κανόνων που θέσαμε, όπως φαίνεται στο σχήμα 3.2 . Αυτό σημαίνει ότι θα είχαμε στη διάθεσή μας μία συνάρτηση, έστω f, που θα μπορούσε να υπολογίσει τήν ποσότητα $R(X,Z)$ μέσω σύνθεσης των δύο άλλων ήδη γνωστών. Μια τέτοια συνάρτηση, f, θα είχε την εξής ιδιότητα :

$$R(X,Z) = f(\,R(X,Y),\ R(Y,Z)\,)$$

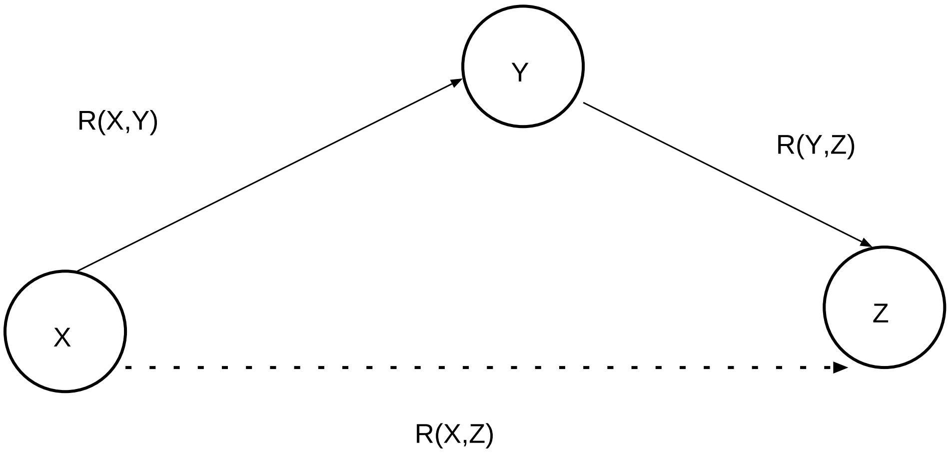
αν και μόνο αν $R(X,Y) > 0$ και $R(Y,Z) > 0$.

Με τον τρόπο αυτό θα είχαμε και λιγότερους υπολογισμούς για την εύρεση <u>όλων</u> των βαθμών υποκατάστασης μεταξύ όλων των αντικειμένων που υπάρχουν στην βάση μας.

Ας δούμε λοιπόν αν με βάση την συνάρτηση R που έχομε δώσει για το λογισμικό, μπορούμε να αποδείξομε ότι αν γνωρίζομε τον βαθμό υποκατάστασης μεταξύ των X, Y, και μεταξύ των Y, Z, τότε μπορούμε να συνάγομε και τον βαθμό υποκατάστασης μεταξύ των X, Z. Αν συμβαίνει κάτι τέτοιο τότε όντως θα μπορούμε να ισχυριστούμε ότι έχομε έναν τρόπο <u>μεταβατικού</u> <u>υπολογισμού</u> του βαθμού υποκατάστασης. Στις παραγράφους που ακολουθούν γίνεται μία αναφορά στα είδη των αντικειμένων που χειριζόμαστε με σκοπό την εξέταση του μεταβατικού υπολογισμού σε κάθε ένα από αυτά.

(α) Βασικοί Τύποι.

Ο μεταβατικός υπολογισμός ισχύει λόγω της μορφής της συνάρτησης, Basic, που δίνει τον βαθμό υποκατάστασης (Πίνακας 3.1) μεταξύ βασικών τύπων.



Σχήμα 3.2 : Περίπτωση συσχέτισης αντικειμένων λογισμικού.

(β) Δομημένοι Τύποι.

Αν οι X, Y, Z είναι δομημένοι τύποι, υπάρχει κάποια συνάρτηση που να υπολογίζει τον βαθμό υποκατάστασης $R(X, Z)$ από τους $R(X, Y)$, $R(Y, Z)$ αν αυτοί είναι γνωστοί και όχι ίσοι με μηδέν.

Πίνακες :

Η απόδειξη της ύπαρξης μιας τέτοιας συνάρτησης θα γίνει για μονοδιάστατους πίνακες. Για πίνακες μεγαλύτερης διάστασης η απόδειξη είναι εντελώς ανάλογη για την κάθε διάσταση.

Απόδειξη: Εστω X, Y, Z τύποι μονοδιάστατων πινάκων, με $R(X, Y) > 0$ και $R(Y, Z) > 0$. Εφόσον για κάθε $X, Y \in ST$ ισχύει : $R(X, Y) \in \{0, 1\}$, τότε για τους X, Y, Z έχομε :

$$R(X, Y) = 1 \quad \text{και} \quad R(Y, Z) = 1$$

Αυτό σημαίνει ότι :

$$R(X, Y) = 1 \quad <==> \quad \{\, N_X \le N_Y \quad \Lambda \quad R(T_{X,}\, T_Y) = 1 \,\} \qquad (1)$$

$$R(Y, Z) = 1 \quad <==> \quad \{\, N_Y \le N_Z \quad \Lambda \quad R(T_{Y,}\, T_Z) = 1 \,\} \qquad (2)$$

Από τις (1) και (2) έχομε :

$$N_X \le N_Z \quad \Lambda \quad R(T_{X,}\, T_Z) = 1$$

Αρα : $R(X, Z) = 1$.

Το ότι $R(T_{X,}\, T_Z) = 1$ δεν είναι τόσο προφανές από τις (1) και (2). Ομως αυτό συμβαίνει γιατί οι T_X, T_Y, και T_Z είναι τύποι βασικοί για να είμαστε σε θέση να υπολογίσομε την τιμή της R, σε πρώτο στάδιο. Οταν οι παραπάνω τύποι δεν είναι βασικοί και πάλι ισχύει η σχέση όπως θα αποδείξομε στο Θεώρημα 2, στο υποκεφάλαιο 3.8 . ●

Εγγραφές :

Απόδειξη: Εστω X, Y, Z τύποι εγγραφών με $R(X, Y) > 0$ και $R(Y, Z) > 0$. Με D_X, D_Y και D_Z συμβολίζομε το πλήθος των πεδίων των X, Y, και Z αντίστοιχα. Εφόσον για κάθε $X, Y \in ST$ ισχύει : $R(X, Y) \in \{0, 1\}$, τότε για τους X, Y, Z έχομε :

$$R(X, Y) = 1 \ \text{ και } \ R(Y, Z) = 1$$

Αυτό σημαίνει ότι :

$$R(X, Y) = 1 \quad <==> \quad \{D_X \le D_Y$$

$$\text{και υπαρχει μετάθεση των πεδίων του } Y \text{ ωστε}$$
$$R(X.field_i, Y.field_i) = 1 \text{ για καθε } i = 1,...,D_X\} \quad (3)$$

$$R(Y, Z) = 1 \quad <==> \quad \{D_Y \leq D_Z$$
$$\text{και υπαρχει μετάθεση των πεδίων του } Z \text{ ωστε}$$
$$R(Y.field_i, Z.field_i) = 1 \text{ για καθε } i = 1,...,D_Y\} \quad (4)$$

Από τις (3) και (4) έχομε :

$$D_X \leq D_Z$$
$$\text{και υπαρχει μετάθεση των πεδίων του } Y \text{ ωστε}$$
$$R(X.field_i, Z.field_i) = 1 \text{ για καθε } i = 1,...,D_X$$

Αρα : $R(X, Z) = 1$.

Το ότι $R(X.fields_i, Z.fields_i) = 1$ δεν είναι τόσο προφανές από τις (3) και (4). Ομως αυτό συμβαίνει γιατί οι $X.fields$, $Y.fields$, και $Z.fields$ είναι τύποι βασικοί για να είμαστε σε θέση να υπολογίσομε την τιμή της R, σε πρώτο στάδιο. Οταν οι παραπάνω τύποι δεν είναι βασικοί και πάλι ισχύει η σχέση όπως θα αποδείξομε στο Θεώρημα 2, στο υποκεφάλαιο 3.8 . ●

Σύνολα :

Απόδειξη: Εστω X, Y, και Z τύποι συνόλων με $R(X, Y) > 0$ και $R(Y, Z) > 0$. Εφόσον για κάθε $X, Y \in ST$ ισχύει : $R(X, Y) \in \{0, 1\}$, τότε για τους X, Y, Z έχομε :

$$R(X, Y) = 1 \quad \text{και} \quad R(Y, Z) = 1$$

Αυτό σημαίνει ότι :

$$R(X, Y) = 1 \quad <==> \quad \{ \quad R(T_X, T_Y) = 1 \} \quad (5)$$

$$R(Y, Z) = 1 \quad <==> \quad \{ \quad R(T_Y, T_Z) = 1 \} \quad (6)$$

Αρα : $R(X, Z) = 1$.

Το ότι $R(T_X, T_Z) = 1$ δεν είναι τόσο προφανές από τις (5) και (6). Ομως αυτό συμβαίνει γιατί οι T_X, T_Y και T_Z είναι βασικοί τύποι, για να είμαστε σε θέση να υπολογίσομε την τιμή της R, σε πρώτο στάδιο. Οταν οι παραπάνω τύποι δεν είναι βασικοί και πάλι ισχύει η σχέση όπως θα αποδείξομε στο Θεώρημα 2, στο υποκεφάλαιο 3.8 . ●

Οσον αφορά τους τύπους ένωσης (union) και λίστας, οι αποδείξεις είναι εντελώς ανάλογες με αυτές για εγγραφές και σύνολα αντίστοιχα.

(γ) Αντικείμενα Λογισμικού.

Δυστυχώς όσον αφορά τα αντικείμενα λογισμικού δεν μπορούμε να πούμε ανάλογα πράγματα εφόσον ο βαθμός υποκατάστασής τους ανά δύο είναι μη-μηδενικός. Πάνω σε αυτό δεν θα δώσομε απόδειξη αλλά ένα μικρό και απλό αντιπαράδειγμα όπου μεταξύ τριών συναρτήσεων X, Y, και Z, ισχύει $R(X, Y) > 0$ και $R(Y, Z) > 0$, ενώ $R(X, Z) = 0$ (σχήμα 3.2). Οι τύποι που περιγράφονται είναι τύποι μονοδιάστατων πινάκων από 9 στοιχεία ακεραίων, 10 στοιχεία ακεραίων, 9 στοιχεία double, 20 στοιχεία ρητών. Με βάση τους παραπάνω τύπους περιγράψαμε συντακτικά τις συναρτήσεις X, Y, και Z. Με το αντιπαράδειγμα αυτό αποδεικνύεται ότι δεν ισχύει η μεταβατική ιδιότητα για τη σχέση υποκατάστασης μεταξύ αντικειμένων λογισμικού και για τη συνάρτηση υπολογισμού της.

```
TELL IndividualClass Array9ofINT
        in S_Class, CooLArrayType with
        numEl       : 9
        element : CooLINT
end Array9ofINT

TELL IndividualClass Array10ofINT
        in S_Class, CooLArrayType with
        numEl       : 10
        element : CooLINT
end Array10ofINT

TELL IndividualClass Array9ofDouble
        in S_Class, CooLArrayType with
        numEl       : 9
        element : CooLDouble
end Array9ofDouble

TELL IndividualClass Array20ofFloat
        in S_Class, CooLArrayType with
        numEl       :20
        element : CooLFloat
end Array20ofFloat

TELL IndividualClass X
        in S_Class, CooLProcedure with
        inParameters
                : CooLINT;
                : CooLFloat
        inoutParameters
                : CooLDouble;
                : CooLDouble
        outParameters
                : CooLDouble;
                : Array20ofFloat
```

```
        returnType
                : Array10ofINT
end X

TELL IndividualClass Y
        in S_Class, CooLProcedure with
        inParameters
                : CooLDouble;
                : CooLFloat
        outParameters
                : CooLDouble;
                : CooLDouble
        returnType
                : Array9ofINT
end Y

TELL IndividualClass Z
        in S_Class, CooLProcedure with
        inParameters
                : CooLBool
        inoutParameters
                : CooLINT;
                : CooLINT
        outParameters
                : Array10ofINT;
                : CooLDouble
        returnType
                : Array9ofDouble
end Z
```

Σχήμα 3.2 : Ενα παράδειγμα όπου δεν ισχύει ο μεταβατικός υπολογισμός

Είναι φανερό λοιπόν ότι ο μεταβατικός υπολογισμός βαθμών υποκαταστασης ή ομοιότητας, όπως ορίσθηκαν, δεν είναι, εν γένει, δυνατός. Στο κεφάλαιο που ακολουθεί γίνεται μια προσπάθεια αντιστάθμισης αυτής της αδυναμίας.

3.8. Ενα κάτω φράγμα για τον αυτόματο υπολογισμό βαθμών υποκατάστασης

Στο προηγούμενο υποκεφάλαιο δείξαμε πως η σχέση υποκατάστασης που ορίσαμε δεν έχει συνάρτηση μεταβατικού υπολογισμού. Εντούτοις, αποδείξαμε πως ισχύει για μικρότερα σύνολα οντοτήτων που χειριζόμαστε, όπως βασικούς και δομημένους τύπους. Εδώ θα διατυπώσομε δύο θεωρήματα με στόχο να δώσουμε μια υπό συνθήκη μεταβατική ιδιότητα.

Ορίζω το σύνολο **correspond** ως εξής :

$$correspond(A, B, i) = \{\, j \in |A| \mid R_{i_j}(A, B) = 1 \,\},$$

δηλαδή το σύνολο correspond περιέχει τις θέσεις για όλα τα στοιχεία του i-γνωρίσματος για τα οποία $R_i(A, B) = 1$, για τα $A, B \in O$.

Θεώρημα 1 : Εστω $R(X, Y) > 0$ και $R(Y, Z) > 0$, για $X, Y, Z \in O$. Τότε ισχύει :

$$R_i(X, Z) \geq \frac{|\, correspond(X, Y, i) \cap correspond(Y, Z, i) \,|}{|\, X.i \,|} \qquad (7).$$

Απόδειξη : Το σύνολο $correspond(X, Y, i)$ έχει ως στοιχεία του όλα τα στοιχεία του i-γνωρίσματος για τα οποία $R_{i_k}(X, Y) = 1$. Ομοια, το σύνολο $correspond(Y, Z, i)$

έχει ως στοιχεία του ολα τα στοιχεία της i-ιδιότητας για τα οποία $R_{i_k}(Y, Z) = 1$. Με βαση όσα είπαμε στο προηγούμενο κεφάλαιο, αν τα γνωρίσματα των λογισμικών αντικειμένων X, Y, Z έχουν μόνο βασικούς και δομημένους τύπους, τότε για $j \in X.i$, όπου με $X.i_j$, $Y.i_j$ και $Z.i_j$ συμβολίζομε το j στοιχείο που αντιστοιχεί στο σύνολο των στοιχείων του i γνωρίσματος για κάθε ένα από τα αντικείμενα αντίστοιχα, :

$$R_i(X.i_j, Y.i_j) = 1 \ \Lambda \ R_i(Y.i_j, Z.i_j) = 1 \quad ===> \quad R_i(X.i_j, Z.i_j) = 1.$$

Αρα ισχύει η σχέση (7). Οταν όμως τα πεδία των λογισμικών αντικειμένων X, Y, Z έχουν κι αυτά με τη σειρά τους αναφορές σε άλλα λογισμικά αντικείμενα, τότε δεν ξέρομε αν η τιμή της R_{i_j} είναι 0 ή όχι. Για το λόγο αυτό σε κάθε τέτοια περίτωση που το j πεδίο δεν είναι βασικός ή δομημένος τύπος θεωρούμε ότι η τιμή της R_{i_j} είναι 1. ●

Με όμοιο τρόπο βρίσκουμε ότι η σχέση (7) ισχύει και για διαδικασίες, είτε αυτές αναφέρουν στις παραμέτρους τους δομημένους τύπους είτε όχι.

Εστω τώρα ότι οι X, Y, Z είναι τύποι αντικειμένων ή λογισμικά αντικείμενα. Τότε ξέρομε ότι ισχύει :

$$R(X, Z) = \frac{\sum_{i=1}^{|X|} R_i(X, Z)}{|X|} \quad (8)$$

Οι συναρτήσεις R_i μπορούν να αναφέρονται ανάλογα με το γνώρισμα i που εξετάζομε για τον υπολογισμό της R_i είτε σε βασικούς, είτε σε δομημένους τύπους, είτε σε λογισμικό με την ευρύτερη έννοια (διαδικασίες, τύποι αντικειμένων, προγράμματα). Η σχέση (7) ισχύει γενικά για τύπους αντικειμένων, ή για διαδικασίες. Με βάση αυτή την παρατήρηση, στη σχέση (8) αντικαθιστούμε την τιμή της συνάρτησης R_i, που εμφανίζεται μέσα στο άθροισμα, με το δεξιό μέλος της σχέσης (7). Ετσι έχομε :

$$R(X,Z) = \frac{\sum_{i=1}^{|X|} R_i(X,Z)}{|X|} \geq \frac{\sum_{i=1}^{|X|} \frac{|correspond(X,Y,i) \cap correspond(Y,Z,i)|}{|X_i|}}{|X|} \quad (9)$$

αν $R(X,Y) > 0$ και $R(Y,Z) > 0$.

Θεώρημα 2 : Εστω $R(X,Y) = 1$ και $R(Y,Z) = 1$, για $X, Y, Z \in O$. Τότε ισχύει :

$$R(X, Z) = 1.$$

Απόδειξη : Εφόσον $R(X,Y) = 1$, σημαίνει ότι $correspond(X,Y,i) = \{1, \ldots, |X_i|\}$, για κάθε $i = 1,\ldots,|X|$, και $correspond(Y,Z,i) = \{1, \ldots, |Y_i|\}$ για κάθε $i = 1,\ldots,|Y|$. Επίσης, επειδή $R_i(X,Y) = 1$, σημαίνει ότι και $|X_i| \leq Y_i|$, γιατί αν ίσχυε $|X_i| > Y_i|$ τότε το $|X_i|+1$ στοιχείο του γνωρίσματος i του X θα έδινε βαθμό υποκατάστασης 0 από το Y, αφού το Y δεν θα είχε $|X_i|+1$ στοιχείο για να τα συγκρίνει. Ομοια έχομε και $|Y_i| \leq |Z_i|$. Από τα παραπάνω συνάγομε ότι $X_i \subseteq Y_i$ και $Y_i \subseteq Z_i$. Από την σχέση (8) έχομε :

$$1 \geq R_i(X,Z) \geq \frac{|correspond(X,Y,i) \cap correspond(Y,Z,i)|}{|X_i|} =$$

$$\frac{|X_i \cap Y_i|}{|X_i|} = \frac{|X_i|}{|X_i|} = 1.$$

Αρα, $R(X,Z) = 1$.

Από το Θεώρημα 2 βλέπομε πως ο μεταβατικός υπολογισμός του βαθμού υποκατάστασης επιτυγχάνεται για αντικείμενα τα οποία υποκαθιστούν πλήρως το ένα το άλλο. Αυτή η διαπίστωση δεν έρχεται καθόλου σε αντίθεση με τα αποτελέσματά μας για βασικούς και δομημένους τύπους. Φυσικά τα παραπάνω αποτελέσματα και συμπεράσματα ισχύουν και για τη σχέση ομοιότητας που αναφέραμε.

Ειδικά αν θεωρήσομε ως σύνολο αναφοράς μας το σύνολο των οντοτήτων με $R(X, Y) = 1$, τότε στο σύνολο αυτό η σχέση ομοιότητας αποτελεί σχέση ισοδυναμίας. Κάθε στοιχείο αυτού του συνόλου είναι αντιπρόσωπος του συνόλου ως προς τη σχέση ομοιότητας. Αν αντί της σχέσης ομοιότητας θεωρήσομε τη σχέση υποκατάστασης τότε αποτελεί μερική διάταξη ως προς την σχέση $R(X, Y)$. Και στις δύο περιπτώσεις πάντως το Θεώρημα 2 μας παρέχει έναν υπό συνθήκη τρόπο για τον μεταβατικό υπολογισμό του βαθμού υποκατάστασης μεταξύ δύο αντικειμένων, εφόσον γνωρίζομε τους ενδιάμεσους βαθμούς υποκατάστασης σε σχέση με ένα τρίτο.

Η πρακτική αξία του Θεωρήματος 1 είναι λίγο διαφορετική από αυτή του Θεωρήματος 2, μια και δεν μας παρέχει κανένα τρόπο μεταβατικού υπολογισμού του βαθμού υποκατάστασης. Εντούτοις, η αξία του έγκειται στο ότι είναι σε θέση να μας γλυτώσει από άσκοπους υπολογισμούς του βαθμού υποκατάστασης μεταξύ δύο αντικειμένων στην περίπτωση που ψάχνομε για αντικείμενα με βαθμό υποκατάστασης μεγαλύτερο από ένα όριο, δίνοντας μας ταυτόχρονα και έναν προσεγγιστικό τρόπο αναζήτησης αντικειμένων. Ψάχνοντας μεταξύ αντικειμένων που από την εφαρμογή της σχέσης (7) προκύπτει ότι το δεξιό μέλος της σχέσης είναι μεγαλύτερο ή ίσο του ορίου που έχομε θέσει, είμαστε σίγουροι ότι θα βρούμε αντικείμενα που θα ικανοποιούν τις απαιτήσεις μας και με λιγότερους υπολογισμούς, παρά ψάχνοντας σε όλη την ΒΠΛ. Το σύνολο των αντικειμένων που βρίσκομε είναι υποσύνολο των αντικειμένων που ικανοποιούν την απαίτησή μας, σε όλη την ΒΠΛ. Επίσης, δεν είμαστε αναγκασμένοι να υπολογίσομε τον ακριβή βαθμό υποκατάστασης, μια και κάτι τέτοιο δεν ζητείται. Ακόμη, με τον τρόπο αυτό απαντούμε ταυτόχρονα μερικώς και στην ερώτηση για αναζήτηση αντικειμένων με βαθμό υποκατάστασης μικρότερο από το όριο αυτό. Στο σύνολο αυτών των αντικειμένων σίγουρα δεν θα ανήκουν αντικείμενα για τα οποία ισχύει η σχέση (7), με το δεξιό μέλος της μεγαλύτερο ή ίσο από το όριο αυτό.

4. Συντακτική υποκατάσταση λογισμικών αντικειμένων με βάση σύνολα κριτηρίων

Αφού περιγράψαμε το μοντέλο για τον ορισμό σχέσεων προσέγγισης και συγκεκριμένα σχέσεων υποκατάστασης και ομοιότητας μεταξύ αντικειμένων λογισμικού, το επόμενο βήμα μας είναι η περιγραφή ενός μοντέλου για τον σχηματισμό ερωτήσεων μεταξύ λογισμικών αντικειμένων ως προς τις προαναφερθείσες σχέσεις. Οι ερωτήσεις αυτές αναφέρονται σε λογισμικά αντικείμενα με σκοπό την ανάκτησή τους με βάση ένα σύνολο απαιτήσεων/κριτηρίων.

4.1. Υπολογισμός βαθμού υποκατάστασης με βάση σύνολα κριτηρίων

Το μοντέλο που περιγράψαμε στο κεφάλαιο 3 μας δίνει τη δυνατότητα για τον υπολογισμό του βαθμού υποκατάστασης μεταξύ οποιουδήποτε ζεύγους αντικειμένων λογισμικού, χωρίς όμως να μας επιτρέπει τον καθορισμό συνόλων κριτηρίων/απαιτήσεων με βάση τα οποία θα περιγράφαμε πιο πολύπλοκες απαιτήσεις πέρα από το αν ένα αντικείμενο μπορεί να χρησιμοποιηθεί στη θέση ενός άλλου.

Θα θέλαμε λοιπόν να επεκτείνομε το μοντέλο μας ώστε να μας δίνει τη δυνατότητα για ερωτήσεις περισσότερο σύνθετες με βάση σύνολα απαιτήσεων καθοριζόμενα από το χρήστη που στόχο έχουν την έκφραση πιο εξειδικευμένων απαιτήσεων. Ετσι, για παράδειγμα αν αναφερόμαστε στην κλάση των συναρτήσεων είναι πιθανό να μην μας ενδιαφέρει η ανεύρεση όλων των συναρτήσεων που υποκαθιστούν τον στόχο μας, αλλά η ανεύρεση συναρτήσεων που πληρούν ορισμένες και μόνο απαιτήσεις που στην πραγματικότητα είναι ένα υποσύνολο του συνόλου κριτηρίων για τον υπολογισμό του βαθμού υποκατάστασης μεταξύ συναρτήσεων. Πιο συγκεκριμένα, μπορούμε π.χ. να ζητήσομε όλες τις συναρτήσεις που έχουν έναν συγκεκριμένο τύπο επιστροφής ή να δέχονται ως παραμέτρους εισόδου μία συγκεκριμένη λίστα παραμέτρων. Πραγματικά, σε ένα περιβάλλον που αναφέρεται σε αντικείμενα λογισμικού, τα περιγράφει, τα χειρίζεται, τα ανευρίσκει ένα προς ένα, και όλα αυτά με βάση κριτήρια που αφορούν τις περιγραφές τους είναι φυσικό να επιθυμούμε να μπορούμε να τα διαχειριστούμε όχι μόνο με ένα και μοναδικό τρόπο αλλά να έχομε την δυνατότητα να ρωτάμε γι' αυτά με ελεύθερο τρόπο όσον αφορά τους δυνατούς συνδυασμούς των κριτηρίων που αρχικά θέσαμε για την σύγκρισή τους. Για το λόγο αυτό μιλάμε για σύνολα κριτηρίων που σχηματίζονται βάσει των γνωρισμάτων των οντοτήτων που λαμβάνουν μέρος στις σχέσεις που ορίσαμε, και αποτέλεσαν τα βασικά μας κριτήρια.

Εφόσον θα ασχοληθούμε με σχηματισμό ερωτήσεων μεταξύ αντικειμένων λογισμικού με βάση σύνολα κριτηρίων πρώτα - πρώτα θα ορίσομε τι είναι ένα σύνολο κριτηρίων και μετα θα ορίσομε τις πράξεις που μπορούμε να κάνομε μεταξύ τέτοιων συνόλων.

Ορισμός 1 : **Βασικό κριτήριο**, PC, ονομάζομε κάθε γνώρισμα (category) που εμφανίζεται στην περιγραφή των οντοτήτων που χειριζόμαστε (βασικοί και δομημένοι τύποι και αντικείμενα λογισμικού) και λαμβάνεται υπ' όψιν ως κριτήριο για τον υπολογισμό του βαθμού υποκατάστασης μεταξύ τους.

Ετσι τα είδη των κριτηρίων είναι :

Για δομημένους Τύπους :

- numEl, element, fields, elements

Για Αντικείμενα Λογισμικού :

- inParameter, inoutParameter, outParameter, returnType, method, superType, dataType, variable, procedure, objectType, reference, main.

Για Βασικούς Τύπους

Δεν ορίζομε κριτήρια γιατί η συνάρτηση υπολογισμού του βαθμού υποκατάστασης μεταξύ τους, Basic, είναι ορισμένη ανεξαρτήτως γνωρισμάτων. Οντως η συνάρτηση Basic είναι η μόνη που ορίζεται εξ' αρχής και δεν προκύπτει αναδρομικά από άλλες. Επίσης δεν θα είχε κανένα νόημα να θέσομε κριτήρια μεταξύ βασικών τύπων μια και είναι από τα δομικά στοιχεία κάθε γλώσσας και δεν φέρουν περισσότερη πληροφορία πέρα του τύπου τους.

Ορισμός 2 : **Σύνολο κριτηρίων**, C, ονομάζομε κάθε σύνολο από βασικά κριτήρια ή από σύνολα βασικών κριτηρίων. Ετσι, ένα σύνολο κριτηρίων έχει γενικά την εξής μορφή :

$$C = \{ C_1 , C_2 , \cdots , C_n , PC_1 , PC_2 , PC_k \} ,$$

όπου $C_i , i = 1,...,n$: σύνολο κριτηρίων και $PC_j , j = 1,...,k$: βασικό κριτήριο.

Ορισμός 3 : Ονομάζομε **ανάπτυγμα**, IC, ενός συνόλου κριτηρίων C, ένα σύνολο κριτηρίων ισοδύναμο του C, που περιέχει μόνο βασικά κριτήρια. Το σύνολο IC προέρχεται από την ανάπτυξη των C_i και από τα PC_j του C.

Ενα παράδειγμα.

Εστω, $C_1 = \{ a , b , c \}$, $C_2 = \{ a , c , l \}$ και $C_3 = \{ b , e , f \}$. Τότε αν $C = \{ C_1 , C_2 , C_3 \}$, το σύνολο IC είναι : $IC = \{ a , b , c , d , e , f , l \}$.

Πριν αρχίσομε να ασχολούμαστε με πράξεις μεταξύ των συνόλων κριτηρίων και τις συναρτήσεις τους για τον υπολογισμό του βαθμού υποκατάστασης μεταξύ αντικειμένων λογισμικού, θα δώσομε την συνάρτηση για τον υπολογισμό του βαθμού υποκατάστασης μεταξύ αντικεμένων λογισμικού με βάση ένα σύνολο κριτηρίων. Η συνάρτηση με βάση ένα σύνολο κριτηρίων C, είναι η εξής :

$$R_C(X , Y) = \frac{\sum_{PC \in IC} R_{PC}(X , Y)}{| IC |} \qquad (1).$$

Το σύνολο κριτηρίων C για δύο λογισμικά αντικείμενα X , Y μπορεί να είναι μέχρι και το σύνολο όλων των γνωρισμάτων που έχομε ορίσει για την κλάση στην οποία ανήκουν τα παραπάνω αντικείμενα.

Πάνω στα σύνολα κριτηρίων που περιγράψαμε ορίζομε δύο πράξεις, την ένωση και την τομή. Με βάση τις δύο αυτές πράξεις μπορούμε να σχηματίσομε δύο βασικά είδη ερωτήσεων. Τις ερωτήσεις διάζευξης, $Q_{\cup}$, ή διαζευκτικές ερωτήσεις και τις ερωτήσεις σύζευξης, $Q_{\cap}$, ή συζευκτικές ερωτήσεις.

Μία ερώτηση διάζευξης έχει την εξής δομή :

$$Q_{\cup} = [\, C_1 \cup C_2 \cup \ldots \cup C_n \,].$$

όπου τα C_i είναι σύνολα κριτηρίων.

Μία ερώτηση σύζευξης έχει την εξής δομή :

$$Q_{\cap} = [\, C_1 \cap C_2 \cap \ldots \cap C_n \,].$$

όπου τα C_i είναι σύνολα κριτηρίων.

Βέβαια όταν επιθυμούμε να κάνομε ερωτήσεις με συνδυασμό ένωσης και τομής τέτοιων συνόλων κριτηρίων λαμβάνεται πάντα υπ' όψιν η προτεραιότητα της τομής ως προς την ένωση. Επίσης μπορεί να γίνει και χρήση παρενθέσεων κατά τα γνωστά, όπως και στις πράξεις συνόλων.

4.2. Υπολογισμός βαθμού υποκατάστασης με βάση ερωτήσεις διάζευξης ή σύζευξης

Εχοντας ορίσει στο προηγούμενο υποκεφάλαιο τα επιτρεπόμενα είδη των πράξεων για σύνολα κριτηρίων στη συνέχεια θα δώσομε τις συναρτήσεις για τον υπολογισμό του βαθμού υποκατάστασης, με βάση σύνολα κριτηρίων, μεταξύ λογισμικών αντικειμένων.

Πρώτα όμως θα ορίσομε δύο νέα ειδικά σύνολα κριτηρίων που εκφράζουν τις απαιτήσεις των ερωτήσεων ένωσης και τομής. Ετσι για το σύνολο κριτηρίων $C = \{ C_1, C_2, \cdots, C_n \}$ έχομε :

$$C_{\cup} = \cup_{i=1}^{n} C_i$$

$$C_{\cap} = \cap_{i=1}^{n} C_i$$

Η συνάρτηση για τον υπολογισμό του βαθμού υποκατάστασης οντοτήτων με βάση την διάζευξη συνόλων κριτηρίων βασίζεται στην συνάρτηση (1) για τον υπολογισμό του βαθμού υποκατάστασης οντοτήτων με βάση οποιοδήποτε σύνολο κριτηρίων, και είναι :

$$R_{C_{\cup}}(X, Y) = \frac{\sum_{PC \in IC_{\cup}} R_{PC}(X, Y)}{|IC_{\cup}|} \quad (2).$$

Η συνάρτηση για τον υπολογισμό του βαθμού υποκατάστασης οντοτήτων με βάση την σύζευξη συνόλων κριτηρίων είναι :

$$R_{C_{\cap}}(X, Y) = \frac{\sum_{PC \in IC_{\cap}} R_{PC}(X, Y)}{|IC_{\cap}|} \quad (3).$$

Τέλος, ορίζομε : $R_{\varnothing} = 0$.

4.3. Ιδιότητες των ερωτήσεων διάζευξης και σύζευξης

Ας δούμε όμως ποιές είναι οι ιδιότητες που χαρακτηρίζουν τις ερωτήσεις συνόλων κριτηρίων, και που μπορούμε να τις χρησιμοποιήσομε είτε για να έχομε καλύτερα θεωρητικά αποτελέσματα είτε για να εξάγομε κάποια συμπεράσματα για τις ερωτήσεις με σύνολα κριτηρίων. Προς τούτο θα ανατρέξομε στις συναρτήσεις υπολογισμού του βαθμού υποκατάστασης μεταξύ λογισμικών αντικειμένων για σύνολα κριτηρίων, που δώσαμε στο υποκεφάλαιο

4.3 .

Εστω C_1, C_2, και C_3 σύνολα κριτηρίων. Τότε ισχύουν οι παρακάτω ιδιότητες.

- $$R_{C_1 \cup C_2}(X, Y) = R_{C_2 \cup C_1}(X, Y) \qquad (4)$$

Απόδειξη : Εστω C_1 και C_2 σύνολα κριτηρίων. Τότε : $C_1 \cup C_2 = C_2 \cup C_1$, λόγω της αντιμεταθετικής ιδιότητας της ένωσης των συνόλων. $I\ (C_1 \cup C_2) = I\ (C_2 \cup C_1)$ Αρα και :

$$R_{C_1 \cup C_2}(X, Y) = \frac{\sum_{PC \in I(C_1 \cup C_2)} R_{PC}(X, Y)}{|\ I\ (C_1 \cup C_2)\ |} =$$

$$\frac{\sum_{PC \in I(C_2 \cup C_1)} R_{PC}(X, Y)}{|\ I\ (C_2 \cup C_1)\ |} = R_{C_2 \cup C_1}(X, Y). \quad \bullet$$

- $$R_{C_1 \cap C_2}(X, Y) = R_{C_2 \cap C_1}(X, Y) \qquad (5)$$

- $$R_{(C_1 \cap C_2) \cup C_3}(X, Y) = R_{(C_1 \cap C_3) \cap C_2 \cup C_3}(X, Y) \qquad (6)$$

- $$R_{(C_1 \cup C_2) \cap C_3}(X, Y) = R_{(C_1 \cap C_3) \cup C_2 \cap C_3}(X, Y) \qquad (7)$$

Οι αποδείξεις για τις σχέσεις (5)-(7) είναι απόλυτα ανάλογες με την απόδειξη της σχέσης (4), και βασίζονται στις ιδιότητες των πράξεων συνόλων (αντιμεταθετική και προσεταιριστική).

- Αν για κάθε $PC_i, PC_j \in C$ ισχύει $R_{PC_i}(X, Y) = R_{PC_j}(X, Y)$, τότε :

$R_{C_\cup}(X, Y) = R_{C_\cap}(X, Y)$ (8)

Απόδειξη : Εστω X και Y δύο λογισμικά αντικείμενα. Εστω, $R_{PC_i}(X, Y) = R(X, Y)$, για κάθε i . Τότε :

$$R_{C_\cup}(X, Y) = \frac{\sum_{PC \in IC_\cup} R_{PC}(X, Y)}{|IC_\cup|} =$$

$$\frac{|IC_\cup| * R(X, Y)}{|IC_\cup|} = R(X, Y) \quad (8.1)$$

$$R_{C_\cap}(X, Y) = \frac{\sum_{PC \in IC_\cap} R_{PC}(X, Y)}{|IC_\cap|} =$$

$$\frac{|IC_\cap| * R(X, Y)}{|IC_\cap|} = R(X, Y) \quad (8.2)$$

Από τις (8.1) και (8.2) έχομε ότι : $R_{C_\cup}(X, Y) = R_{C_\cap}(X, Y)$. ●

- Εστω, $C_1 \subseteq C_2$, τότε :

$$\frac{|IC_1|}{|IC_2|} * R_{C_1}(X, Y) \leq R_{C_2}(X, Y) \leq \frac{|IC_1|}{|IC_2|} * R_{C_1}(X, Y) + \frac{|IC_2 - IC_1|}{|IC_1|}.$$

Απόδειξη : Εστω X και Y δύο λογισμικά αντικείμενα.

$$R_{C_2}(X,Y) = \frac{\sum_{PC \in IC_2} R_{PC}(X,Y)}{|IC_2|} \geq$$

$$\frac{\sum_{PC \in IC_1} R_{PC}(X,Y)}{|IC_2|} = \frac{|IC_1|}{|IC_2|} * R_{C_1}(X,Y).$$

$$R_{C_2}(X,Y) = \frac{\sum_{PC \in IC_2} R_{PC}(X,Y)}{|IC_2|} =$$

$$\frac{\sum_{PC \in IC_1} R_{PC}(X,Y)}{|IC_2|} + \frac{\sum_{PC \in I(C_2 - C_1)} R_{PC}(X,Y)}{|IC_2|} \leq$$

$$\frac{|IC_1|}{|IC_2|} * R_{C_1}(X,Y) + \frac{|IC_2 - IC_1|}{|IC_1|}.$$

4.4. Στάθμιση κριτηρίων

Με βάση το μοντέλο που προτείναμε για σύνθετες ερωτήσεις μεταξύ αντικειμένων λογισμικού με βάση σύνολα κριτηρίων, θα προτείνομε επιπλέον ένα πιο γενικευμένο μοντέλο για τέτοιου είδους ερωτήσεις με βάρη. Η ιδέα είναι να αντιστοιχίσομε σε κάθε υποσύνολο κριτηρίων C_i του συνόλου C ένα πραγματικό αριθμό που να αντικατοπτρίζει την φυσική προτίμηση του χρήστη ή την σπουδαιότητα κάθε συνόλου C_i σε σχέση με τα άλλα σύνολα.

Με βάση αυτά που είπαμε οι ερωτήσεις διάζευξης και σύζευξης παίρνουν τώρα την εξής μορφή :

$$Q_{\cup} = [\, w_1 * C_1 \cup w_2 * C_2 \cup \ldots \cup w_n * C_n \,].$$

$$Q_{\cap} = [\, w_1 * C_1 \cap w_2 * C_2 \cap \ldots \cap w_n * C_n \,].$$

Τα $w_i,\ i = 1,...,n$ είναι τα βαρη των συνόλων κριτηρίων.

Για τον υπολογισμό του βαθμού υποκατάστασης μεταξύ λογισμικών αντικειμένων με βάση σύνολα κριτηρίων με βάρη, χρησιμοποιούνται οι παρακάτω συναρτήσεις :

$$R_C^w(X, Y) = \frac{\sum_{PC \in IC} R_{PC}(X, Y) * w_{PC}}{|\, IC \,|}$$

όπου, $w_{PC} = \max\{\ w_i \ |\ PC \in C_i\ \}$

$$R_{C_{\cup}}^w(X, Y) = \frac{\sum_{PC \in IC_{\cup}} R_{PC}(X, Y) * w_{PC}}{|\, IC_{\cup} \,|}$$

$$R_{C_{\cap}}^w(X, Y) = \frac{\sum_{PC \in IC_{\cap}} R_{PC}(X, Y) * w_{PC}}{|\, IC_{\cap} \,|}$$

Δώσαμε λοιπόν ένα ορισμό για τον υπολογισμού της συνάρτησης $R_C^w(X, Y)$, καθώς και μία πρόταση για τον καθορισμό των βαρών στην παραπάνω συνάρτηση. Ο υπολογισμός της συνάρτησης $R_C^w(X, Y)$ όπως φαίνεται είναι καθ’ όλα ανάλογος της $R\,(X, Y)$, και στις ακραίαες περιπτώσεις οι δύο συναρτήσεις ταυτίζονται $R_C^I(X, Y) = R_C(X, Y)$, όπου $I = (1,...,1)$, το άνυσμα βαρών με στοιχεία μοναδιαία. Ανάλογες συναρτήσεις με βάρη συναντάμε στη βιβλιογραφία, σε περιπτώσεις όπου γίνεται χρήση βαρών. Παρακάτω θα προτείνομε τρεις ακόμη εναλλακτικούς τρόπους υπολογισμού της $R_C^w(X, Y)$, που κατά τη γνώμη μας έχουν περισσότερη φυσική σημασία ή μπορούν να χρησιμοποιηθούν πιο εύκολα.

- Βάρη καθοριζόμενα από το χρήστη. Αυτή η διαδικασία γίνεται διαλογικά, παράλληλα με την περιγραφή των C_i του C. Η μέθοδος αυτή αναδεικνύει και την πραγματική ουσία και υπόσταση των βαρών σε σχέση με τις επιθυμίες του χρήστη.

- Βάρη ισοκατανεμημένα μεταξύ όλων των C_i του C. Η μέθοδος αυτή είναι ακριβώς ότι κάναμε μέχρι τώρα για τον σχηματισμό και υπολογισμό ερωτήσεων με βάση σύνολα κριτηρίων χωρίς βάρη. Οντως αν θεωρήσομε ότι $w_i = 1$ για όλα τα C_i του C, τότε και $w_{PC} = 1$, οπότε ισχύει :

$$R_C^w(X, Y) = R_C(X, Y).$$

$$R_{C_{\cup}}^w(X, Y) = R_{C_{\cup}}(X, Y).$$

$$R_{C_{\cap}}^w(X, Y) = R_{C_{\cap}}(X, Y).$$

- Βάρη αυτόματα καθοριζόμενα από το σύστημα. Η μέθοδος αυτή θα πρέπει να σχηματίζει βάρη με κάποια φυσική ή και πρακτική σημασία. Μία πρόταση είναι η παρακάτω : κάθε w_i εκφράζει την περιεκτικότητα του αντιστοίχου C_i σε σχέση με τα υπόλοιπα C_j του C. Ετσι, έχομε :

$$w_i = \frac{|IC_i|}{|IC|}$$

4.5. Αλλαγή κλάσεως και επανυπολογισμός των βαθμών υποκατάστασης

Εστω S και T κλάσεις οντοτήτων. Με $\bar{S}$ ονομάζω το σύνολο των περιπτώσεων (instances) της S. Εστω λοιπόν ότι $X \in \bar{S}$. Ας υποθέσομε πως θέλομε να αλλάξομε κλάση και να δούμε ποιά είναι η σχέση του X με τις περιπτώσεις της T. Τότε θα μας ενδιέφερε να κάνομε όσο γίνεται λιγότερους υπολογισμούς του βαθμού υποκατάστασης του X με κάθε $Y \in \bar{T}$ (σχήμα 4.1). Ας δούμε λοιπόν κατά πόσο κάτι τέτοιο είναι εφικτό και τι κερδίζομε στη περίπτωση αυτή. Βασική προυπόθεση για να γίνει κάτι τέτοιο είναι ότι τα κριτήρια ως προς τα οποία υπολογίζεται η R καθορίζονται από την κλάση στην οποία τα X και Y ανήκουν καθώς επίσης ότι τα X και Y πρέπει να ανήκουν στην ίδια κλάση προκειμένου να συγκριθούν ή το X να είναι περίπτωση και των δύο κλάσεων.

Οι δυνατές περιπτώσεις είναι οι ακόλουθες :

(α) $\bar{S} \cap \bar{T} = \varnothing$.

Στην περίπτωση αυτή δεν έχει νόημα να γίνει κανείς υπολογισμός αφού το X δεν ανήκει καν στο $\bar{T}$.

(β) $\bar{S} \cap \bar{T} = \{ X \}$.

Δηλαδή, το μόνο κοινό στοιχείο των $\bar{S}$ και $\bar{T}$ είναι το X. Αυτό σημαίνει πως έχουν κάποιο σύνολο κριτηρίων κοινό αλλά δεν έχουν καμμία άλλη περίπτωση κοινή. Ετσι, θα πρέπει να γίνει από την αρχή ο υπολογισμός του βαθμού υποκατάστασης του X με κάθε $Y \in \bar{T}$.

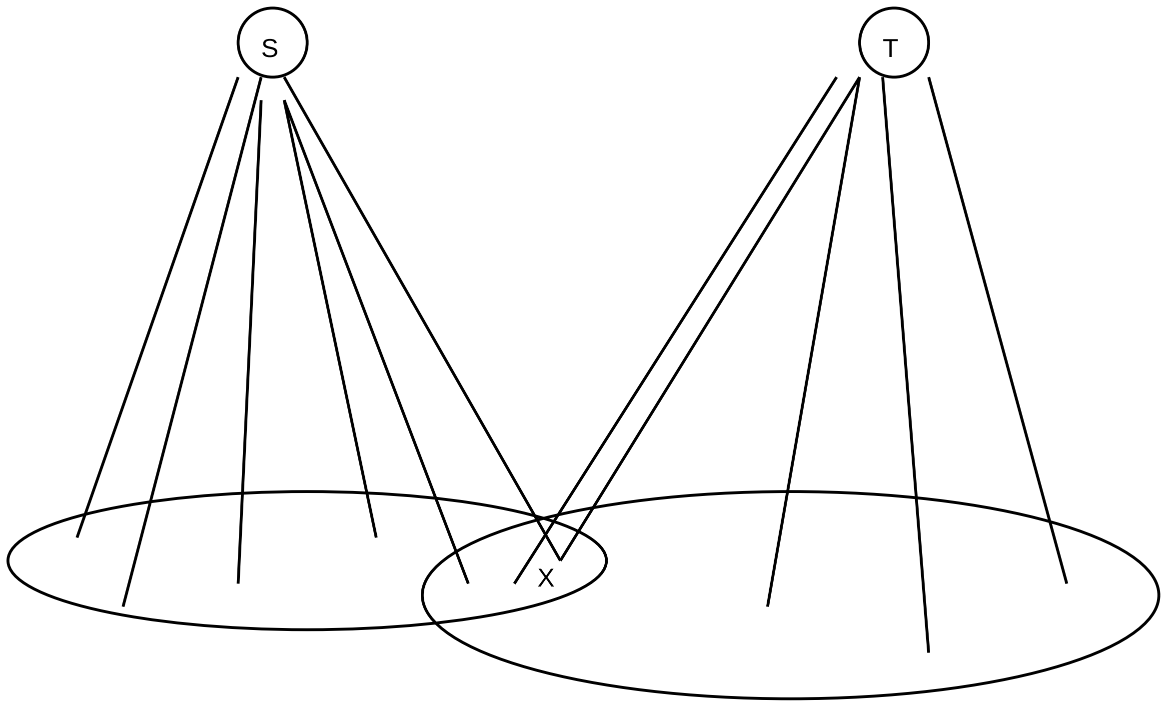
(γ) $\bar{S} \cap \bar{T} = \{ X \} \cup A$.

Εστω C_T το σύνολο των κριτηρίων της κλάσης T. Το σύνολο $A \subseteq \bar{T}$, και έχει την εξής μορφή :

$$A = \{ A_{1,} \cdots , A_k \}.$$

Για τα A_i λοιπόν ισχύει :

$$R_{C_T}(X, A_i) = \frac{\sum_{PC \in IC_T} R_{PC}(X, A_i)}{| IC_T |} =$$



Σχήμα 4.1 : Δύο κλάσεις και τα σύνολα των περιπτώσεών τους.

$$\frac{\sum_{PC \in I(C_T \cap C_S)} R_{PC}(X, A_i) + \sum_{PC \in I(C_T - C_S)} R_{PC}(X, A_i)}{|IC_T|} =$$

$$\frac{\sum_{PC \in I(C_T \cap C_S)} R_{PC}(X, A_i)}{|IC_T|} + \frac{\sum_{PC \in I(C_T - C_S)} R_{PC}(X, A_i)}{|IC_T|}$$

$$\frac{|I(C_T \cap C_S)|}{|IC_T|} * R_{C_T \cap C_S}(X, A_i) + \frac{|I(C_T - C_S)|}{|IC_T|} * R_{C_T - C_S}(X, A_i)$$

Βλέπομε δηλαδή πως κατά κάποιο τρόπο η αλλαγή στην κλάση αναφοράς ή ενδιαφέροντος μπορεί να απαιτήσει τόσο λιγότερους υπολογισμούς όσο πιο μεγάλο είναι το σύνολο των κοινών κριτηρίων. Επίσης αυτό που φαίνεται καθαρά είναι η άμεση σχέση της αλλαγής κλάσης με την αλλαγή συνόλου κριτηρίων, μια και κάθε κλάση περιγράφεται και αποτελείται από ένα σύνολο ιδιοτήτων (γνωρισμάτων) που την απαρτίζουν και που μερικές από αυτές είναι και βασικά κριτήρια για την σχέση υποκατάστασης και την σχέση ομοιότητας. Βέβαια το αντίθετο δεν συμβαίνει πάντα. Οι περιπτώσεις που αναλύσαμε παραπάνω ισχύουν, το πολύ μια από αυτές, κάθε φορά που αλλάζομε κλάση, δηλαδή έμμεσα αλλάζομε σύνολο κριτηρίων.

5. Ενας αλγόριθμος υπολογισμού του βαθμού υποκατάστασης λογισμικών αντικειμένων

Είδαμε ότι για κάθε ζεύγος οντοτήτων στη σχέση υποκατάστασης ή ομοιότητας υπάρχει ένας μοναδικός πραγματικός αριθμός, ο βαθμός υποκατάστασης ή ομοιότητάς τους, που τα χαρακτηρίζει.

Στο κεφάλαιο αυτό θα περιγράψομε έναν αλγόριθμο για τον υπολογισμό του βαθμού υποκατάστασης ή ομοιότητας μεταξύ αντικειμένων λογισμικού, καθώς και μία βελτίωσή του όσον αφορά την υπολογιστική του πλοκή.

5.1. Περιγραφή του αλγορίθμου

Ο αλγόριθμος που θα περιγράψομε παρακάτω υπολογίζει τον βαθμό υποκατάστασης μεταξύ δύο δοσμένων λογισμικών αντικειμένων. Με βάση τον αλγόριθμο αυτό μπορούμε κατά συνέπεια να υπολογίσομε τον βαθμό υποκατάστασης ή/και ομοιότητας ενός προκαθορισμένου λογισμικού αντικειμένου στην βάση μας, με όλα τα άλλα αντικείμενα που ανήκουν στην ίδια κλάση με αυτό.

<u>Ο αλγόριθμος.</u>

Δεδομένα :

Εστω X και Y δύο λογισμικά αντικείμενα των οποίων θέλομε να υπολογίσομε τον βαθμό υποκατάστασης, $R(X, Y)$.

Βήμα 1 :

Βρες για το σύνολο των γνωρισμάτων (categories) του αντικειμένου X τα στοιχεία του κάθε γνωρίσματος, από τη βάση. Ονόμασε το σύνολο αυτό S_X.

Βήμα 2 :

Βρες για το σύνολο των γνωρισμάτων (categories) του αντικειμένου Y τα στοιχεία του κάθε γνωρίσματος, από τη βάση. Ονόμασε το σύνολο αυτό S_Y.

Βήμα 3 :

Για κάθε γνώρισμα i των X και Y βρες τον βαθμό υποκατάστασης τους,

$R_i(X, Y)$, ως προς αυτό το γνώρισμα.

Βήμα 4 :

Υπολόγισε τον βαθμό υποκατάστασης των X και Y, $R(X, Y)$, όπως είδαμε στο κεφάλαιο 3.6 .

Στην περίπτωση που θέλομε να υπολογίσομε τον βαθμό υποκατάστασης ενός δοσμένου λογισμικού αντικειμένου, X, από το σύνολο των λογισμικών αντικειμένων, τότε δεν έχομε παρά να καλέσομε επαναληπτικά τον παραπάνω αλγόριθμο με σταθερό το πρώτο του όρισμα. Κάτι τέτοιο χρειάζεται πολύ συχνά στην αναχρησιμοποίηση λογισμικού,αφού στόχος μας συνήθως είναι η ανεύρεση ενός συνόλου αντικειμένων που πληρούν με κάποιο βαθμό τις απαιτήσεις μας, η κατάταξη των στοιχείων του και τελικά η επιλογή ενός αντικειμένου, συνήθως με δική μας ευθύνη.

5.2. Ανάλυση της πλοκής του αλγορίθμου

Η ανάλυση της πλοκής θα γίνει για την περίπτωση που ο αλγόριθμός μας καλείται επαναληπτικά N φορές, όπου το N παριστάνει τον αριθμό των λογισμικών αντικειμένων μεταξύ των οποίων θέλομε να δούμε αν υφίσταται η σχέση υποκατάστασης και σε ποιό βαθμό. Ο λόγος που γίνεται αυτό είναι γιατί μας ενδιαφέρει περισσότερο από όλα η υπολογιστική συμπεριφορά του αλγορίθμου σε σχέση με το πλήθος των αντικειμένων λογισμικού που υπάρχουν στην βάση δεδομένων μας. Ετσι, έστω :

N : το σύνολο των αντικειμένων λογισμικού που μας ενδιαφέρει, που μπορεί να είναι διαδικασίες ή προγράμματα.

Q : μέσος χρόνος εξυπηρέτησης μιας ερώτησης (θα αναφερθούμε στο κεφάλαιο 6).

OT : μέσος χρόνος σύγκρισης για τύπους αντικειμένων.

m : μέγιστος αριθμός στοιχείων ανά γνώρισμα.

Πριν δόσομε την έκφραση για την υπολογιστική πλοκή της επαναληπτικής κλήσης του αλγορίθμου, πρέπει πρώτα να αναφερθούμε σε κάποιες επιλογές που κάναμε κατά την υλοποίηση του αλγορίθμου και τελικά για τον υπολογισμό της πλοκής του. Πρώτα απ’ όλα πρέπει να αναφερθούμε στον τρόπο σύγκρισης των στοιχείων των κριτηρίων του λογισμικού. Στο κεφάλαιο 3.5 αναφερθήκαμε σε τρεις διαφορετικούς τρόπους σύγκρισής τους. Στην παρούσα υλοποίηση ακολουθήσαμε την ένα προς ένα κατά σειρά σύγκριση των στοιχείων των κριτηρίων (περίπτωση (α), κεφάλαιο 3.5).

Ο τύπος που περιγράφει την πλοκή του αλγορίθμου αναφέρεται στην περίπτωση αναζήτησης λογισμικού χωρίς σύνολα κριτηρίων. Ο τρόπος που θεωρούμε ότι γίνεται η σύγκριση των στοιχείων των γνωρισμάτων για κάθε ζεύγος αντικειμένων, είναι ο υπολογισμός του βαθμού υποκατάστασης μεταξύ όλων των δυνατών συνδυασμών των στοιχείων αυτών (περίπτωση (β). κεφάλαιο 3.5). Δηλαδή, αν X, Y είναι τα αντικείμενα των οποίων θέλομε να υπολογίσομε τον βαθμό υποκατάστασής τους, και $|X.i|$, $|Y.i|$ είναι ο αριθμός των στοιχείων των X, Y για το i γνώρισμα αντίστοιχα, τότε υπολογίζομε $|X.i|^2$ βαθμούς υποκατάστασης, R_i, και όχι $|X.i| * |Y.i|$, αφού στόχος μας είναι να υπολογίσομε την ποσότητα

$$R_i(X, Y) = \frac{\sum_{j=1}^{|X.i|} R(X.i_j, Y.i_j)}{|X.i|}$$

Ο παράγοντας Q εμφανίζεται παντού, και μάλιστα στο τετράγωνο, λόγω του ότι ο αλγόριθμος συνεργάζεται με ένα άλλο πρόγραμμα (το programmatic query interface), μέσω του οποίου ανακτάμε οποιαδήποτε πληροφορία είναι αποθηκευμένη στην ΒΠΛ, και λόγω της εκάστοτε ποσότητας $|X.i|^2$ που απαιτεί Q^2 ερωτήσεις στη ΒΠΛ, μία για κάθε στοιχείο.

Σύμφωνα λοιπόν με όσα αναφέραμε παραπάνω, και με βάση το ότι τα λογισμικά αντικείμενα έχουν τα γνωρίσματα dataTypes, objectTypes, procedures, references, main, πρέπει να γίνουν $|X.dataTypes|^2$, $|X.objectTypes|^2$, $|X.procedures|^2$, $|X.references|^2$, $|X.main|^2$ ψαξίματα και υπολογισμοί για κάθε γνώρισμα αντίστοιχα. Ομως το κάθε στοιχείο ενός γνωρίσματος μπορεί και αυτό να είναι γνώρισμα μιας άλλης κλάσης, οπότε θα χρειαστεί να γίνουν επιπλέον για καθένα τέτοιο στοιχείο m^2, αν m είναι ο μέσος αριθμός στοιχείων ανά στοιχείο γνωρίσματος.

Ετσι, για dataTypes απαιτούνται $|X.dataTypes|^2 * m^2$ υπολογισμοί. Για objectTypes απαιτούνται $|X.objectTypes|^2 * OT^2 * m^2$, θεωρώντας ότι υπάρχει και κάποιος επιπλέον υπολογισμός για τύπους αντικειμένων που πιθανώς να έχουν στις παραμέτρους τους αναφορές σε άλλους τύπους αντικεμένων. Για procedures απαιτούνται $|X.procedures|^2 * OT^2 * m^2$, θεωρώντας ότι υπάρχει και κάποιος επιπλέον υπολογισμός για τύπους αντικειμένων που πιθανώς να έχουν στις παραμέτρους τους αναφορές σε άλλους τύπους αντικεμένων. Τα γνωρίσματα references και main δεν αναφέρονται στον τύπο μια και μας οδηγούν από μόνα τους σε ερωτήσεις υπολογισμού του βαθμού υποκατάστασης μεταξύ προγραμμάτων. Επίσης, απαιτούνται $Q * N$ ερωτήσεις στην ΒΠΛ για την ανεύρεση όλων των προγραμμάτων.

Ετσι, ο τύπος που μας δίνει την πλοκή του αλγορίθμου που αναφέραμε στ προηγούμενο υποκεφάλαιο είναι :

$$O\,[\,Q * N + N * Q^2\,[\,|X.dataTypes|^2 * m^2 + |X.objectTypes|^2 * OT^2 * m^2 + |X.procedures|^2 * OT^2 * m^2\,]\,]$$

5.3. Βελτίωση της πλοκής του αλγορίθμου

Αν παρατηρήσομε προσεκτικά τον τρόπο υπολογισμού του βαθμού υποκατάστασης μεταξύ αντικειμένων λογισμικού βλέπομε πως τελικά καταλήγομε στην αναζήτηση ή/και στον υπολογισμό του βαθμού υποκατάστασης μεταξύ βασικών ή δομημένων τύπων. Για να είμαστε πιο ακριβείς δεν μπορούμε να υπολογίσομε τον βαθμό υποκατάστασης μεταξύ αντικειμένων λογισμικού αν **πρώτα** δεν υπολογίσομε τον βαθμό υποκατάστασης μεταξύ των δομημένων τύπων που τα συγκροτούν. Επίσης, είναι πολύ πιθανό για τον υπολογισμό ενός και μόνο βαθμού υποκατάστασης να χρειαστεί να υπολογίσομε ξανά και ξανά τον ίδιο βαθμό υποκατάστασης μεταξύ κάποιων άλλων τύπων.

Αυτή η απλή παρατήρηση μας οδήγησε σε μια προσπάθεια να αποφύγομε την χρήση του αλγορίθμου σε περιπτώσεις που απλά θα ξαναυπολογίζαμε ήδη γνωστά αποτελέσματα. Αυτό όμως δεν μας βοηθά στο να αποφύγομε επιπλέον υπολογισμούς από μεριάς δομής και λογικής του αλγορίθμου. Προτείνομε λοιπόν πριν να χρησιμοποιήσομε τον αλγόριθμό μας, να υπολογίζομε εκ των προτέρων τον βαθμό υποκατάστασης μεταξύ όλων των δομημένων τύπων που έχουν περιγραφεί στην ΒΠΛ. Κατόπιν, θα αποθηκεύομε τα αποτελέσματά μας και αυτά στην ΒΠΛ, με βάση το μοντέλο περιγραφής σχέσεων προσέγγισης. Με τον τρόπο αυτό ο βαθμός υποκατάστασης μεταξύ δομημένων τύπων ανακτάται σε σταθερό χρόνο όποτε χρειαστεί με μία απλή ερώτηση μέσω του programmatic interface.

Κάνοντας ένα βήμα παραπάνω από τον υπολογισμό εκ των προτέρων του βαθμού υποκατάστασης μεταξύ δομημένων τύπων, καταλήγομε στο να προυπολογίζομε και τον βαθμό υποκατάστασης μεταξύ όλων των τύπων αντικειμένων και την αποθήκευσή τους στην ΒΠΛ.

Εστω ότι το πλήθος των δομημένων τύπων της εφαρμογής μας είναι N_{ST} και το πλήθος των τύπων αντικειμένων είναι N_O. Σύμφωνα με το μοντέλο που περιγράψαμε για την αποθήκευση σχέσων προσέγγισης στο κεφάλαιο 1.2 και που παριστάνεται αναλυτικά στο Σχήμα 1.1, το πλήθος των συνδέσμων που απαιτούνται για την αποθήκευση των βαθμών υποκατάστασης μεταξύ δομημένων τύπων είναι A_{ST} επιπλέον αριθμοί (για την αποθήκευση των instanceOf σχέσεων) και $3 * A_{ST}$ αντικείμενα σε TELOS για την αποθήκευση των επιπλέον συνδέσμων μεταξύ τους (σύνδεσμος μεταξύ των δομημένων τύπων ανά δύο, για ευθύ και αντίστροφο βάρος), όπου $A_{ST} = \sum_{i=1}^{N_{ST}} (i-1) = \frac{N_{ST} * (N_{ST}-1)}{2}$ Αυτό συμβαίνει γιατί κάθε instanceOf σχέση αποθηκεύεται στη μνήμη σαν ένας αριθμός, ενώ κάθε σύνδεσμος μεταξύ αντικειμένων στην TELOS αποθηκεύεται στην μνήμη σαν τρία αντικείμενα, ένα για τον ίδιο τον σύνδεσμο και δύο για τα αντικείμενα τα οποία συνδέει [ITHACA.92.E2.#2]. Ανάλογα πράγματα ισχύουν και για τύπους αντικειμένων. Οσο αφορά στην ταχύτητα ανάκτησης της πληροφορίας που αποθηκεύομε δεν επηρεάζεται από μία τόσο μικρή προσαύξηση γιατί γίνεται χρήση B-Trees και κατά την αποθήκευση και κατά την αναζήτηση από τους αντίστοιχους αλγορίθμους. Πέρα από αυτά, όσον αφορά στις ερωτήσεις που γίνονται πολύ συχνά στην ΒΠΛ υπάρχει η πρόβλεψη για την τοποθέτηση στην κρυφή μνήμη (cache) των αντικειμένων και των συνδέσμων τους που χρησιμοποιούνται ευρύτατα, γεγονός που σε τελική ανάλυση ούτε και χρονικά θα επηρεάζει πλέον την αναζήτησή τους στο δίσκο. Αν επιλέγαμε να αποθηκεύαμε την πληροφορία αυτή σε πίνακα, που να είναι διαθέσιμος σ’ όλη τη διάρκεια της χρήσης του αλγορίθμου τότε θα απαιτούνταν ο πίνακας αυτός να είναι διάστασης N_{ST}^2, προκειμένου για δομημένους τύπους και N_O^2, προκειμένου για τύπους αντικειμένων.

Με βάση τα όσα αναφέραμε παραπάνω, η πλοκή του αλγορίθμου για N επαναλήψεις τώρα πια δεν απαιτεί το εξαντλητικό ψάξιμο για κάθε στοιχείο κάθε κριτηρίου γιατί είμαστε σε θέση να γνωρίζομε με ποιό στοιχείο συνδέεται με τη σχέση υποκατάστης κάθε στοιχείο κάνοντας μόνο μία απλή ερώτηση. Καταλήγομε λοιπόν να κάνομε μόνο ερωτήσεις για να βρούμε τα στοιχεία που αντιστοιχούν στις διαδικασίες ενός προγράμματος, για να υπολογίσομε την τιμή της R_{PROCS}, που συμμετέχει στον υπολογισμό της R, που δίνομε στο κεφάλαιο 3.6 . Ετσι η προηγούμενη σχέση γίνεται :

$$O\, [\; Q * N + N * Q\; [\; |\, X.datatypes\, | + |\, X.objectTypes\, | + |\, X.procedures\, |^2 * (\, m * Q\,)\;]\,]$$

6. Υλοποίηση

Στο κεφάλαιο αυτό αναφέρουμε το περιβάλλον στο οποίο ο αλγόριθμος υλοποιήθηκε, τα βοηθητικά εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξή του, καθώς και μερικές από τις βασικές επιλογές για την επικοινωνία του με το όλο περιβάλλον του έργου ITHACA.

6.1. Περιβάλλον υλοποίησης

Για την υλοποίηση του αλγορίθμου χρησιμοποιήθηκαν σταθμοί εργασίας Sun με λειτουργικό σύστημα Unix SunOS 4.1.1 και σύστημα παραθύρων Xwindows έκδοση 11 release 5. Ολες οι δοκιμές του αλγορίθμου προέρχονται από SPARCstation ELC με χρονισμό 40 MHz, απόδοση της τάξεως των 28 MIPS και μνήμη 18 MBytes.

Η γλώσσα που χρησιμοποιήθηκε είναι η C++. Η μετάφραση έγινε με την επιλογή βελτιστοποίησης -O και χρησιμοποιήθηκε η εκδοση 1.39.1 του μεταφραστού της. Οι βασικοί λόγοι για τους οποίους προτιμήθηκε η C++ είναι η πληρότητα των προγραμματιστικών δομών της, η ποιότητα του κώδικα που παράγει σε σχέση με τον μικρό χρόνο μεταγλώττισης που απαιτεί και τέλος το γεγονός ότι η συγκεκριμένη υλοποίηση του αλγορίθμου αποτελεί τμήμα μιας μεγαλύτερης εφορμογής που αναπτύσσεται σε C++.

Η παρούσα εργασία εντάσσεται στο ερευνητικό έργο ITHACA και συγκεκριμένα στο μέρος του έργου που ονομάζεται Selection Tool (ST). Το ST είναι το κύριο εργαλείο επικοινωνίας του εξωτερικού κόσμου με την ΒΠΛ. Αυτή επιτυγχάνεται με δύο κυρίως τρόπους. Εναν ερωτηματικό μηχανισμό για την ανεύρεση των αντικειμένων προς αναχρησιμοποίηση και ένα εργαλείο που επιτρέπει την περιήγηση (navigation) στην ΒΠΛ σε διαφορετικά επίπεδα λεπτομέρειας, που οναμάζεται Browser. Αν λοιπόν στόχος μας είναι η ανάδειξη ή η αναζήτηση ομοίων αντικειμένων ή αντικειμένων που μπορούν να υποκαταστήσουν ένα δοσμένο αντικείμενο, τότε και μόνο τότε τίθεται σε λειτουργία ο μηχανισμός υπολογισμού του βαθμού υποκατάστασης ή ομοιότητας που περιγράψαμε και τα αποτελέσματα αυτών των υπολογισμών τελικά επιδεικνύονται στον χρήστη μέσω του ST.

6.2. Υλοποίηση του αλγορίθμου

Για την υλοποίηση του αλγορίθμου δεν κρίθηκε αναγκαία η δημιουργία και χρησιμοποίηση ειδικών δομών δεδομένων για την αποθήκευση των ενδιάμεσων ή των τελικών αποτελεσμάτων. Οι δομές που χρησιμοποιήσαμε δεν είναι παρά απλές λίστες από τα αντικείμενα που χειριζόμαστε.

Οι λίστες αυτές χρησιμοποιούνται και για την αποθήκευση στην ΒΠΛ της νέας πληροφορίας που προκύπτει με την χρήση του αλγορίθμου, με τη δομή που περιγράψαμε για τις σχέσεις υποκατάστασης και ομοιότητας στο κεφάλαιο 1.3 . Η αποθήκευση γίνεται κάθε φορά που δημιουργείται ένα προκαθορισμένο πλήθος πληροφορίας από την χρήση του αλγορίθμου, με την χρήση του μηχανισμού αναδήλωσης της TELOS, του RETELL. Για την ενημέρωση της ΒΠΛ χρησιμοποιείται ένα βοηθητικό πρόγραμμα που εγγράφει σε ένα βοηθητικό αρχείο τα στοιχεία της λίστας που προαναφέραμε με τη μορφή μιάς αναδήλωσης (RETELL transaction) στην TELOS. Ετσι, σε τακτά χρονικά διαστήματα γίνεται προσαύξηση της ΒΠΛ με επικοινωνία μέσω pipes με βάση τα περιεχόμενα του βοηθητικού αρχείου.

Η όλη εργασία ανάκτησης της πληροφορίας για τα αντικείμενα που χειριζόματε από την ΒΠΛ γίνεται μέσω ενός εργαλείου που επιτρέπει στοιχειώδεις ερωτήσεις στην βάση μας, του programmatic query interface. Το παραπάνω εργαλείο επιτρέπει στον προγραμματιστή μιας εφαρμογής την ανάκτηση πληροφορίας από την ΒΠΛ, με ομοιόμορφο τρόπο, χωρίς να απαιτείται από αυτόν να γνωρίζει την δομή των κλάσεων που χρησιμοποιούνται για τον χειρισμό των αντικειμένων της TELOS ή για τον τρόπο αποθήκευσης και χειρισμού τους από τα φυσικά μέσα αποθήκευσης. Το programmatic query interface βασίζεται στο μοντέλο client-server, [ITHACA.92.E2.#2].

7. Πειραματική μελέτη

Στο κεφάλαιο αυτό δίνομε ένα παράδειγμα χρήσης του μοντέλου και του αλγορίθμου συντακτικής υποκατάστασης μεταξύ λογισμικών αντικεμένων. Στη συνέχεια παραθέτομε το σύνολο των πειραματικών δοκιμών του αλγορίθμου και κάνομε μία αξιολόγηση των αποτελεσμάτων του.

7.1. Παράδειγμα χρήσης του μοντέλου υποκατάστασης λογισμικών αντικειμένων

Στη συνέχεια δίνομε ένα παράδειγμα χρήσης του αλγορίθμου για τον υπολογισμό του βαθμού υποκατάστασης μεταξύ λογισμικών αντικειμένων. Πρώτα όμως θα αναφερθούμε στο μοντέλο που θα χρησιμοποιήσομε για την περιγραφή των λογισμικών αντικεμένων που θα αναφέρομε και θα περιγράψομε συντακτικά Η γλώσσα στην οποία θα βασιστούμε είναι η CooL και το μοντέλο της είναι το CooL Implementation Model, περίπτωση του Implementation Model της ΒΠΛ, και περιγράφεται αναλυτικά στο Παράρτημα Α.

Το παράδειγμα είναι παρμένο από τον χώρο της Διαχείρισης Πόρων και περιγράφει ένα σύνολο ρουτινών που επιλύουν προβλήματα από τον χώρο αυτό. Το παράδειγμα έχει διατυπωθεί σε TELOS και βρίσκεται στο Παράρτημα Β.

Ο αλγόριθμος που προτείναμε μπορεί να εφαρμοστεί σε αντικείμενα του παραδείγματος είτε δηλαδή σε διαδικασίες είτε σε ολόκληρα προγράμματα, απλά καλώντας την συνάρτηση sim_program(char *, char *) ή την συνάρτηση check_method(char *, char *) ανάλογα αν πρόκειται για προγράμματα ή για διαδικασίες. Ανάλογες συναρτήσεις υλοποιήθηκαν για τον υπολογισμό του βαθμού υποκατάστασης μεταξύ τύπων, είτε αυτοί είναι δομημένοι είτε τύποι αντικειμένων.

7.2. Πειραματική δοκιμή και αξιολόγηση

Το μοντέλο που προτείναμε και για τα αντικείμενα λογισμικού και για τις σχέσεις προσέγγισης μαζί με τον αλγόριθμο έχουν ήδη χρησιμοποιηθεί στα πλαίσια του ερευνητικού έργου ITHACA. Εχει ήδη περιγραφεί μία εφαρμογή για 'ενοικίαση αυτοκινήτων' (Παράδειγμα 1, Παράρτημα Β), στα πλαίσια μιας επίδειξης του έργου, η οποία σχεδιάστηκε και περιγράφηκε σε TELOS από μέλη της ερευνητικής ομάδας του έργου ITHACA [ITHACA.91.E2]. Ακόμη έχομε περιγράψει μία βιβλιοθήκη αλγορίθμων για την επίλυση προβλημάτων διαχείρισης πόρων (Παράδειγμα 2, Παράρτημα Β), με βάση ένα σύνολο αλγορίθμων όπως RANK, RELAX κ.ά. .

Ο αλγόριθμος που χρησιμοποιήθηκε για τον υπολογισμό του βαθμού υποκατάστασης ή/και ομοιότητας που περιγράψαμε στο κεφάλαιο 3, ήταν αυτός που περιγράφηκε στο κεφάλαιο 5. Τα αποτελέσματα αποθηκεύτηκαν στην ΒΠΛ με βάση το μοντέλο περιγραφής των σχέσεων υποκατάστασης και ομοιότητας που δώσαμε στο κεφάλαιο 1.3.

Ο χρόνος απόκρισης του συστήματος για ερώτηση για τον υπολογισμό του βαθμού υποκατάστασης μεταξύ λογισμικών αντικειμένων είναι πάρα πολύ ικανοποιητικός ακόμη και για την επαναληπτική χρήση του για την ανεύρεση του βαθμού υποκατάστασης μεταξύ ενός αντικειμένου με το σύνολο των αντικειμένων που είναι περιπτώσεις της ίδιας κλάσεως με αυτό.

Στη συνέχεια παραθέτομε δύο πίνακες με στατιστικά στοιχεία πάνω στο χρόνο απόκρισης του αλγορίθμου που υλοποιήσαμε. Τα στοιχεία προέρχονται από το παράδειγμα της περιγραφής της βιβλιοθήκης αλγορίθμων για την επίλυση προβλημάτων διαχείρισης πόρων.

Παράδειγμα	Δομημένοι Τύποι	Τύποι Αντικειμένων	Διαδικασίες	Προγράμματα
1	15	10	35	2
2	20	-	86	13

Πίνακας 7.1

Στοιχεία πληθυσμού ανά παράδειγμα

Ο Πίνακας 7.1 αναφέρεται σε στοιχεία πληθυσμού, πάνω στα οποία εφαρμόστηκε ο αλγόριθμος πολλές φορές. Οι αριθμοί που εμφανίζονται αντιστοιχούν στο μέσο πλήθος των αντικειμένων που εξετάστηκαν κάθε φορά που γινόταν μία ερώτηση για τον υπολογισμό του βαθμού υποκατάστασης μεταξύ λογισμικών αντικειμένων.

Παράδειγμα 1				
Χρόνοι (sec)	Δομημένοι Τύποι	Τύποι Αντικειμένων	Διαδικασίες	Προγράμματα
System	0.0	0.2	0.3	0.0
User	0.09	0.9	1.2	0.1
		Παράδειγμα 2		
Χρόνοι	Δομημένοι Τύποι	Τύποι Αντικειμένων	Διαδικασίες	Προγράμματα
System	0.0	-	0.4	0.1
User	0.1	-	2.0	0.4

Πίνακας 7.2

Μέσοι χρόνοι απόκρισης σε ερωτήσεις υπολογισμού βαθμού υποκατάστασης (sec)

Ο Πίνακας 7.2 αναφέρεται στους μέσους χρόνους απάντησης του αλγορίθμου σε ερωτήσεις υπολογισμού του βαθμού υποκατάστασης ενός αντικειμένου με το σύνολο των αντικειμένων που είναι περιπτώσεις της ίδιας κλάσης (επαναληπτική εκτέλεση).

Οσον αφορά στο είδος των αποτελεσμάτων πρέπει να αναφέρομε ότι η χρήση του αλγορίθμου έγινε μόνο για ερωτήσεις χωρίς σύνολα κριτηρίων, θεωρώντας δεδομένο ότι λαμβάνομε υπ’ όψιν όλα τα κριτήρια για τα κάθε είδος λογισμικού που ορίσαμε. Αυτό συνέβη λόγω της απουσίας ενός μηχανισμού υποστήριξης σχηματισμού ερωτήσεων και κατά συνέπεια και μιας ερωτηματικής γλώσσας που να υποστηρίζει τις ανάγκες του μοντέλου του κεφαλαίου 4.

Τα αποτελέσματα, δηλαδή ο υπολογιζόμενος βαθμός υποκατάστασης σε περιπτώσεις που αναφερόμαστε στον ίδιο χώρο (domain) και σε συναφείς διαδικασίες αντανακλούσαν και την σχέση υποκατάστασης ή ομοιότητας μεταξύ αντικειμένων. Λόγω του μικρού πλήθους των λογισμικών αντικειμένων μπορέσαμε να συνάγομε ορισμένα επιπλέον συμπεράσματα σχετικά με τα αποτελέσματα των δοκιμών. Στη συνέχεια παραθέτομε δύο πίνακες που παρουσιάζουν τους βαθμούς υποκατάστασης μεταξύ όλων των δυνατών συνδυασμών ομάδων λογισμικών αντικειμένων που περιγράφησαν στο Παράδειγμα 2, όπου το στοιχείο (i, j) του κάθε πίνακα δηλώνει τον βαθμό υποκατάστασης του αντικειμένου στη θέση i από το αντικείμενο στη θέση j. Δεν θεωρήθηκε απαραίτητο να παραθέσομε περισσότερα ή διαφορετικού χαρακτήρα στοιχεία, όπως για βαθμούς υποκατάστασης μεταξύ διαδικασιών ή τύπων λόγω του μεγάλου πλήθους τους ($\tilde{}\ 10^3$).

	lexincrement	increment	incrmnt	smincrement
lexincrement	1	0.963	0.992	0.958
increment	0.950	1	0.947	0.958
incrmnt	0.930	0.933	1	0.954
smincrement	0.826	0.826	0.843	1

Πίνακας 7.3

Βαθμοί υποκατάστασης Λογισμικού

Στον Πίνακα 7.3 παρουσιάζονται οι βαθμοί υποκατάστασης μεταξύ προγραμμάτων που επιλύουν προβλήματα κατανομής πόρων με ελαχιστοποίηση με συνάρτησης κόστους, για συνεχείς διαχωρίσιμες συναρτήσεις με ακέραιες μεταβλητές, με περιορισμούς αθροίσματος εκτός από την smincrement που επιλύει το γενικότερο πρόβλημα κατανομής πόρων σε υπομεριστική περιοχή. Οπως παρατηρούμε, τα στοιχεία της στήλης του smincrement δίνουν τους μεγαλύτερους βαθμούς υποκατάστασης σε σχέση με τα αντίστοιχα των άλλων στηλών, ενώ ταυτόχρονα τα στοιχεία της γραμμής του smincrement τους μικρότερους σε σχέση τα στοιχεία των άλλων γραμμών. Το γεγονός αυτό δηλώνει πως η smincrement έχει γενικότερη συντακτική περιγραφή από τα υπόλοιπα αντικείμενα της ομάδας αυτής, άρα μπορεί να τα υποκαταστήσει συντακτικά, αλλά όχι και αντίστροφα. Πράγματι, τα προβλήματα που επιλύονται από τα υπόλοιπα προγράμματα είναι ειδικές περιπτώσεις αυτών που επιλύει η smincrement, και κατά συνέπεια η συντακτική τους περιγραφή είναι αναμενόμενο να είναι πιο περιορισμένη από αυτή του smincrement. Επίσης, μία δεύτερη παρατήρηση είναι ότι οι υπόλοιποι βαθμοί υποκατάστασης βρίσκονται αρκετά κοντά μεταξύ τους. Και πάλι το γεγονός αυτό είναι αναμενόμενο αφού επιλύουν συναφή προβλήματα με διαφορά όχι στους τύπους των μεταβλητών τους (άρα και στην συντακτική τους περιγραφή) αλλά στους περιορισμούς τους.

	mr	cont	rank	relax	brelax1	brelax2
mr	1	0.925	0.956	0.971	0.910	0.934
cont	0.683	1	0.682	0.701	0.848	0.689
rank	0.930	0.922	1	0.967	0.892	0.940
relax	0.882	0.913	0.894	1	0.846	0.889
brelax1	0.710	0.900	0.740	0.740	1	0.673
bralx2	0.896	0.897	0.947	0.924	0.852	1

Πίνακας 7.4

Βαθμοί υποκατάστασης Λογισμικού

Στον Πίνακα 7.4 παρουσιάζονται οι βαθμοί υποκατάστασης μεταξύ προγραμμάτων που επιλύουν προβλήματα κατανομής πόρων με ελαχιστοποίηση της συνάρτησης κόστους για συνεχείς συναρτήσεις, με ακέραιες ή συνεχείς μεταβλητές. Για παράδειγμα, οι rank, brelax1, brelax2, relax επιλύουν το ίδιο πρόβλημα αλλά με διαφορετικές απαιτήσεις στην μορφή της συνάρτησης, απαιτώντας από τη μια ίδιο είδος μεταβλητών, και από την άλλη διαφορετικό είδος διαδικασιών για τον έλεγχο της μορφής της συνάρτησης κόστους ή για την εκμετάλλευση κάποιων παραπάνω ιδιοτήτων της (κυρτότητα, κ.ά). Αυτό που πρέπει να αναφέρομε είναι πως τα παραπάνω προγράμματα ακολουθούν συναφείς γενικές διαδικασίες αντιμετώπισης και επίλυσης του προβλήματος, γεγονός που μας οδηγεί στο συμπέρασμα ότι θα πρέπει να είναι και τα παραπάνω προγράμματα συντακτικά συναφή, με υψηλό βαθμό υποκατάστασης. Οι cont και mr επιλύουν το ίδιο πρόβλημα αλλά η μία για ακέραιες και η άλλη για συνεχείς μεταβλητές αντίστοιχα, ακολουθώντας όμως διαφορετική τακτική επίλυσης, άρα και διαφορετικές σημασιολογικά διαδικασίες από τις rank, brelax1, brelax2, relax. Το πρόβλημα που επιλύεται από την cont χρησιμοποιεί για την εύρεση της βέλτιστης λύσης την mr, άρα και κάθε λύση του διακριτού προβλήματος θα είναι και λύση του συνεχούς, όχι όμως και αντίστροφα. Είναι λοιπόν αναμενόμενο $R(cont, mr) \leq R(mr, cont)$, μια και οι οι τύποι των μεταβλητών στην μία θα είναι ακέραιοι κα στην άλλη πραγματικοί. Οι βαθμοί υποκατάστασης που δίνομε στους πίνακες 7.3 και 7.4 είναι όλοι μεγαλύτεροι του 0.6 . Αυτό συμβαίνει λόγω της ελλειπούς περιγραφής του λογισμικού σε TELOS, όπου δεν περιγράφησαν όλα τα γνωρίσματα των προγραμμάτων, παρά μόνο αυτά που αντιστοιχούσαν στις αναφορές τους σε άλλα προγράμματα και στις διαδικασίες που χρησιμοποιούν. Τα υπόλοιπα γνωρίσματα εφόσον δεν περιγράφησαν, ο αλγόριθμος θεώρησε ότι $R_i(X, Y) = 1$, όταν $|X.i| = |Y.i| = 0$ Η διαπίστωση αυτή μας οδηγεί στο συμπέρασμα πως είναι αναγκαία η πλήρης περιγραφή του λογισμικού για να έχομε αποτελέσματα που να ανταποκρίνονται στην συντακτική συνάφεια των λογισμικών αντικειμένων, ειδάλλως τα αποτελέσματα θα είναι παραπλανητικά.

Αυτό που τελικά φάνηκε είναι ότι για αντικείμενα που επιτελούν την ίδια λειτουργία ο αλγόριθμος με μεγάλη πιθανότητα μπορεί να δώσει μεγάλο βαθμό υποκατάστασης ή ομοιότητας που οφείλεται στο ότι πολλά μικρότερα τμήματά τους είναι κατ' ανάγκη αρκετά κοντά, άλλα σε επίπεδο τύπων και άλλα σε επίπεδο διαδικασιών. Αυτό φυσικά δεν πρέπει να μας παραπλανήσει και να παραμερίσομε την αξία ενός μοντέλου για τον υπολογισμό της σημασιολογικής συνάφειας προγραμμάτων, αφού χωρίς αυτό δεν μπορούμε να παραστήσομε και κατά συνέπεια και να υπολογίσμε την σημασιολογική συνάφεια μεταξύ λογισμικών αντικειμένων.

8. Συμπεράσματα, επεκτάσεις και εφαρμογή σε συγγενείς χώρους ενδιαφέροντος

Στο κεφάλαιο αυτό δίνομε τα συμπεράσματά μας από τη μελέτη και πειραματική χρήση του μοντέλου και του αλγορίθμου που προτείναμε για την συντακτική υποκατάσταση λογισμικών αντικειμένων. Επίσης προτείνομε πιθανούς τρόπους χρήσης του παραπάνω μοντέλου και αλγορίθμου για την εξυπηρέτηση άλλων εφαρμογών και στόχων.

8.1. Συμπεράσματα

Ο αλγόριθμος για τον υπολογισμό του βαθμού υποκατάστασης και ομοιότητας που περιγράψαμε στο κεφάλαιο 3 χρησιμοποιήθηκε και τα αποτελέσματα αποθηκεύτηκαν στην ΒΠΛ με βάση το μοντέλο περιγραφής των σχέσεων προσέγγισης. Τα αποτελέσματα του αλγορίθμου, όπως είδαμε και στο κεφάλαιο 7, είναι αρκετά ικανοποιητικά δεδομένου ότι για αντικείμενα με μεγάλη συντακτική συνάφεια υπολογίστηκε υψηλός βαθμός υποκατάστασης, και το αντίστροφο. Μερικές φορές μάλιστα τα αποτελέσματα μας έδιναν και την ψευδαίσθηση ότι αντικείμενα συντακτικά συναφή είναι και σημασιολογικά συναφή, και αυτό λόγω του μικρού πληθυσμού των δοκιμών μας σε συνδυασμό και με το σύνολο των αντικειμένων που περιγράφησαν που ανήκαν και στον ίδιο χώρο (προβλήματα κατανομής πόρων) αλλά και αντιμετώπιζαν συγγενή προβλήματα. Τέτοιας μορφής παρατηρήσεις δεν ξεπερνούν τον τοπικό χαρακτήρα τους, μια και το μοντέλο μας δεν έχει την δυνατότητα να συνάγει συμπεράσματα σχετικά με σημασιολογική συνάφεια λογισμικού. Στην περίπτωση που το μοντέλο μας είχε επαυξηθεί με ένα μοντέλο για τον καθορισμό σημασιολογικής συνάφειας μεταξύ αντικειμένων λογισμικού, τότε τα τελικό αποτέλεσμα, δηλαδή ο βαθμός υποκατάστασης, θα ήταν σύνθεση μιας συνάρτησης δύο μεταβλητών, μίας για τη συντακτική σχέση και μίας για τη σημασιολογική συνάφεια, και τα τελικά αποτελέσματα δεν θα ήταν τόσο μακριά στις ακραίες περιπτώσεις που αναφέραμε. Αυτό που πρέπει όμως να επισημάνομε είναι ότι ο αλγόριθμος αναδεικνύει πάντα την συντακτική σχέση των λογισμικών αντικειμένων εφόσον αυτή υπάρχει, βασιζόμενος στην σχέση μεταξύ των βασικών τύπων και των δομημένων τύπων που ορίσαμε στο κεφάλαιο 3.3 .

Οσον αφορά στην χρήση του μοντέλου που προτείναμε για την περιγραφή, αποθήκευση και ανάκτηση σχέσεων προσέγγισης, η ύπαρξή του στάθηκε ουσιαστική, αφού δεν θα χρειαστεί ποτέ να ξαναυπολογιστεί ο βαθμός υποκατάστασης ή ομοιότητας μεταξύ δύο αντικειμένων λογισμικού, αν έχει γίνει ήδη και αποθηκευτεί στην ΒΠΛ. Αυτό συμβαίνει γιατί το μοντέλο για τον καθορισμό τέτοιων σχέσεων, και κατά συνέπεια και ο υπολογιζόμενος βαθμός υποκατάστασης ή ομοιότητας, είναι ανεξάρτητοι του πληθυσμού της ΒΠΛ, μια και δεν λαμβάνουν υπ' όψιν στοιχεία που αφορούν στον πληθησμό της ΒΠΛ, αλλά μόνο στοιχεία που αφορούν στα γνωρίσματα των κλάσεων που περιγράφομε στο μοντέλο για την συντακτική δομή της κάθε γλώσσας υλοποίησης. Οντως, για να υπολογίσομε τον βαθμό υποκατάστασης, εξετάζομε ένα σύνολο γνωρισμάτων/κριτηρίων, που ορίσαμε στο κεφάλαιο 3. Εφόσον το σύνολο αυτό δεν αλλάξει και εφόσον οι συντακτικές περιγραφές των αντικειμένων που θέλομε να υπολογίσομε τον βαθμό υποκατάστασής τους δεν αλλάξουν, τότε το αποτέλεσμα θα παραμείνει το ίδιο είτε η ΒΠΛ επαυξηθεί με νέα λογισμικά αντιεκίμενα είτε όχι. Τούτο κάνει τον υπολογιζόμενο βαθμό υποκατάστασης ή/και ομοιότητας ανεξάρτητο του πληθυσμού της ΒΠΛ, για δοσμένο σύνολο κριτηρίων (κεφάλαιο 4). Είναι βέβαια κατανοητό ότι ο υπολογιζόμενος βαθμός υποκατάστασης ή/και ομοιότητας είναι άμεσα εξαρτώμενος από το σύνολο των κριτηρίων που καθορίζομε καθώς και από την περιγραφή των λογισμικών αντικειμένων σε επίπεδο συντακτικής δομής (τύποι, διαδικασίες,...).

Επίσης οι ιδιότητες των συναρτήσεων υπολογισμού του βαθμού υποκατάστασης λογισμικών αντικειμένων είναι χρήσιμες έχοντας και φυσική και πρακτική σημασία και εφαρμογή. Ενα ενδιαφέρον σημείο είναι αυτό που αναφέρεται στον μεταβατικό υπολογισμό του βαθμού υποκατάστασης, που αναφέραμε στο κεφάλαιο 3, και μας επιτρέπει την αυτόματη επαύξηση με συνδέσμους σχέσεως υποκατάστασης ή/και ομοιότητας, ενός συνόλου αντικειμένων της ΒΠΛ, που πληρούν τις συνθήκες που δίνομε στο κεφάλαιο 3.8, με τα αποτελέματα του αλγορίθμου για τις σχέσεις και τις οντότητες που χειριζόμαστε. Μέχρι στιγμής δεν έχει γίνει μια ανάλογη προσπάθεια για την αξιοποίηση αναλόγων αποτελεσμάτων μεταβατικών υπολογισμών βαθμών υποκατάστασης ή ομοιότητας σε εργασίες στον χώρο αυτό.

Τέλος, αν προσπαθήσομε να αντιπαραβάλομε την παρούσα εργασία με τις αντίστοιχες που αναφερθήκαμε στο κεφάλαιο 2, παρατηρούμε τον αυστηρά τυπικό χαρακτήρα του ορισμού της σχέσης υποκατάστασης και ομοιότητας. Σε κανένα σημείο είτε της περιγραφής είτε του υπολογισμού του βαθμού των σχέσεων αυτών δεν επεμβαίνει ο Μηχανικός Εφαρμογής. Ο ορισμός των βασικών συναρτήσεων που χρησιμοποιούνται για τον υπολογισμό του βαθμού υποκατάστασης μεταξύ λογισμικού γίνεται σε πολύ χαμηλό επίπεδο (βασικοί τύποι) και κατόπιν όλα τα άλλα γίνονται αυτόματα. Αυτό μας επιτρέπει αφενός μεγάλη ανεξαρτησία από πλευράς κανόνων υπολογισμού και αφετέρου τον περιορισμό του υποκειμενικού χαρακτηρισμού των συντακτικών σχέσεων μόνο μεταξύ βασικών τύπων, καλύπτοντας κάθε τυπική περιγραφή των επιτρεπόμενων τύπων μιας γλώσσας προγραμματισμού. Τέλος, μέσω του τυπικού χαρακτήρα της συντακτικής περιγραφής του λογισμικού ξεφεύγομε από τον ανθρώπινο παράγοντα και από τον υποκειμενικότητα της περιγραφής της λειτουργικότητας του λογισμικού και των επιδόσεων των συναρτήσεων, χωρίς αυτό να σημαίνει πως δεν κρίνεται απαραίτητη η δημιουργία ενός μοντέλου για την εννοιολογική περιγραφή του λογισμικού, που θα συνεργαζόταν με τα αποτελέσματα της συντακτικής περιγραφής του.

8.2. Επεκτάσεις του μοντέλου υποκατάστασης λογισμικών αντικειμένων

Ανακεφαλαιώνοντας, είδαμε πώς με τη χρήση του μοντέλου υποκατάστασης λογισμικών αντικειμένων (κεφάλαιο 4) μπορούμε να χαρακτηρίσομε λογισμικά αντικείμενα σχετικά μεταξύ τους, με βάση σύνολα κριτηρίων που είτε τα καθορίζομε εμείς είτε το ίδιο το μοντέλο (εξ ορισμού). Με τον τρόπο αυτό ορίζοντας κάθε φορά και διαφορετικά σύνολα κριτηρίων περιγράφομε και διαφορετικά σύνολα λογισμικών αντικειμένων που είναι σχετικά με κάποιο κεντρικό αντικείμενο-στόχο.

Ετσι λοιπόν είναι φυσικό να θελήσομε να χρησιμοποιήσομε αυτή την πληροφορία που μας παρέχει το μοντέλο και ο αλγόριθμός μας ώστε να αναδείξομε σε ανώτερο επίπεδο (meta-level) κλάσεις αντικειμένων ως σχετικές ή να χαρακτηρίσομε κλάσεις αντικειμένων με βάση την πληροφορία των βαθμών υποκατάστασης που έχομε για τις περιπτώσεις (instances) τους. Θα μας ενδιέφερε λόγου χάριν να είχαμε πληροφορίες για την συνεκτικότητα των κλάσεων και με βάση τον βαθμό υποκατάστασης των περιπτώσεών της να αποφασίσομε για την βιωσιμότητα ή την αναδιοργάνωση της κλάσης, με τελικό στόχο την περιγραφή ενός καλύτερου και περισσότερο εκφραστικού μοντέλου παράστασης του χώρου ενδιαφέροντός μας. Στην περίπτωση που θα είμαστε σε θέση να προτείνομε με αυτόματο ή ημιαυτόματο τρόπο μία νέα πριγραφή για κάθε κλάση λογισμικών αντικειμένων θα είχαμε και μία μερική απάντηση στο πρόβλημα της ταξινόμησης (classification problem).

Στο σημείο αυτό θα αναφέρομε δύο ερωτήματα που είναι βασικά για τον χαρακτηρισμό κλάσεων σε ανώτερο επίπεδο με βάση την πληροφορία που έχομε για τις περιπτώσεις τους. Ετσι, τα εξής θέματα μας απασχολούν :

[1] Ποιό είναι το σύνολο των κριτηρίων μιας κλάσης για το οποίο όλες οι περιπτώσεις της είναι περισσότερο υποκαταστάσιμες από κάποια περίπτωσή της με όσο το δυνατό πιο υψηλό βαθμό και μικρότερη απόκλιση από αυτόν.

[2] Με ποιόν τρόπο μπορούμε να συγκρίνομε κλάσεις μεταξύ τους με βάση τους αντιπροσώπους τους.

Απαντώντας στο πρώτο από τα παραπάνω ερωτήματα είμαστε σε θέση να αναδείξομε αντιπροσώπο ή αντιπροσώπους κλάσεων. Το στοιχείο που χαρακτηρίζεται ως αντιπρόσωπος είναι κατά κάποιο τρόπο και το "γενικότερο" στοιχείο της κλάσης αφού υποκαθιστά όλα τα άλλα όσο το δυνατόν καλύτερα. Απαντώντας στο δεύτερο ερώτημα μπορούμε να χαρακτηρίζομε κλάσεις ως λιγότερο ή περισσότερο περιγραφικές από άλλες ή και να συγκρίνομε την εκφραστικότητα της ίδιας κλάσης αλλάζοντας την περιγραφή της.

Ετσι λοιπόν ορίζομε για μία κλάση s, τον αντιπρόσωπό της $[S_C] = X,\ X \in \bar{S}$, με βάση το σύνολο κριτηρίων C, ως εξής :

$$X \ :\ \delta_X = \max_{Z \in \bar{S}} \delta_Z\ , \quad \text{οπου} \ \ \delta_Z = \sum_{Y \in \bar{S}} R_C(Y, Z)\ .$$

Εχοντας ορίσει την ποσότητα δ_X, ορίζομε με βάση αυτή και την ποσότητα e_X, που υποδηλώνει την μέγιστη απόσταση του βαθμού υποκατάστασης των $Y \in \bar{S}$ από το X. Ετσι,

$$e_X = \max_{Y \in S} \{\ R_C(Y, X)\ \}$$

Στη συνέχεια, δίνομε δύο ορισμούς για τη σύγκριση κλάσεων με βάση τους αντιπροσώπους τους που ορίσαμε παραπάνω.

Ορισμός 1

Εστω C_S και C_T τα σύνολα κριτηρίων για τις κλάσεις S και T αντίστοιχα. Αν υπάρχουν C_1 και C_2, $C_1 \subseteq C_S$ και $C_2 \subseteq C_T$ ώστε :

1 $[S_{C_1}] = [T_{C_2}]$,

2 $\delta_{[S_{C_1}]} = \delta_{[T_{C_2}]}$,

3 $e_{[S_{C_1}]} \leq e_{[T_{C_2}]}$,

4 $C_1 \subseteq C_2$,

τότε λέμε ότι η κλάση S είναι πιο συγκεντρωμένη από την T ως προς το σύνολο κριτηρίων C_1, και το συμβολίζομε με $T \leq_{C_1} S$.

Το ότι απαιτούμε $\delta_{[S_{C_1}]} = \delta_{[T_{C_2}]}$ σημαίνει ότι ο αντιπρόσωπος των S και T εκπροσωπεί και τις δύο κλάσεις με τον ίδιο βαθμό, και αποτελεί κοινό σημείο αναφοράς. Η απαίτηση για $e_{[S_{C_1}]} \leq e_{[T_{C_2}]}$ είναι ουσιαστική μια και μας εξασφαλίζει ότι όλα τα <u>κοινά</u> <u>στοιχεία</u> των κλάσεων S και T είναι πιο κοντά στον αντιπρόσωπο $[S_{C_1}]$, στην κλάση S.

Ορισμός 2

Εστω C_S και C_T τα σύνολα κριτηρίων για τις κλάσεις S και T αντίστοιχα. Αν για κάθε C_1, $C_1 \subseteq C_S$ και $C_1 \subseteq C_T$ ώστε :

1 $[S_{C_1}] = [T_{C_1}]$,

2 $\delta_{[S_{C_1}]} = \delta_{[T_{C_1}]}$,

3 $e_{[S_{C_1}]} \leq e_{[T_{C_1}]}$,

τότε λέμε ότι η κλάση S είναι πιο συγκεντρωμένη από την T, και το συμβολίζομε με $T \leq S$.

Με βάση τους παραπάνω ορισμούς για την συγκέντρωση κλάσεων σε σχέση με άλλες με βάση σύνολα κριτηρίων και τους αντιπροσώπους των μπορούμε να κάνομε στατιστική μελέτη σε σχέση με τον πληθυσμό της κάθε κλάσης και να δούμε πώς αλλάζει ο αντιπρόσωπός της ανάλογα με το σύνολο κριτηρίων που θεωρούμε κάθε φορά. Πιθανώς για κλάσεις που οι αντιπρόσωποί τους δεν αλλάζουν με το σύνολο κριτηρίων να αντιστοιχούν σε βασικές κλάσης του μοντέλου που χρησιμοποιούμε για την περιγραφή του λογισμικού ή και σε κλάσεις ικανές να περιγράψουν λογισμικό που αντιστοιχεί σε γενικές κλάσεις (generic classes).

8.3. Εφαρμογή σε χώρους άλλου ενδιαφέροντος

Το μοντέλο καθώς και ο αλγόριθμος που προτείναμε αναφέρονται σε λογισμικά αντικείμενα, που μπορούν να περιγραφούν με ένα μοντέλο που παριστά το συντακτικό της οποιαδήποτε γλώσσας προγραμματισμού με βάση την οποία αυτά αναπτύσσονται. Ετσι, αν είμαστε σε θέση να δώσομε το μοντέλο της εκάστοτε γλώσσας προγραμματισμού που χρησιμοποιούμε τότε μπορούμε να εφαρμόσομε τον αλγόριθμο που προτείνομε. Φυσικά κάτι τέτοιο δεν αποκλείει και την συνεργασία και διασύνδεση περιγραφών λογισμικών αντικειμένων από διαφορετικές γλώσσες προγραμματισμού. Στην γενική αυτή περίπτωση βέβαια αυτό που πρέπει να γίνει επιπλέον είναι η επαύξηση του μοντέλου για την παράσταση και αποθήκευση σχέσεων προσέγγισης με επιπλέον περιγραφές που να διασυνδέουν τα κριτήρια των λογισμικών αντικειμένων από διαφορετικές γλώσσες προγραμματισμού. Φυσικά αυτή η εργασία πρέπει να γίνει από τον σχεδιαστή των μοντέλων των γλωσσών, γιατί προυποθέτει βαθύτερη γνώση της σημασίας των παραμέτρων σε κάθε γλώσσα προγραμματισμού.

Με τον τρόπο αυτό επιτυγχάνομε δύο στόχους. Πρώτον, έχομε την ευελιξία της αναζήτησης σχέσεων υποκατάστασης ή και ομοιότητας σε διαφορετικές γλώσσες και την συσχέτιση αποτελεσμάτων για σχέσεις υποκατάστασης ή ομοιότητας που προέρχονται από διαφορετικές γλώσσες. Κατά δεύτερο λόγο, θα μπορούσαμε να θέσομε ως μακροπρόθεσμο στόχο την σύγκριση διαφορετικών γλωσσών προγραμματισμού μεταξύ τους με βάση τον τρόπο που ένα ή περισσότερα λογισμικά αντικείμενα σχετίζονται συντακτικά μεταξύ τους σε καθεμία από αυτές.

Προτείνομε ακόμη την χρήση διαφορετικών μετρικών για τον υπολογισμό του βαθμού υποκατάστασης με στόχο την ανεύρεση διαφορετικών συνόλων υποκαταστάσιμων αντικειμένων, ανάλογα με κάθε μετρική. Μια τέτοια δυνατότητα παρουσιάζει αρκετό ενδιαφέρον από την άποψη ότι η κάθε μετρική έχει και διαφορετικές ιδιότητες, ορίζοντας και ένα διαφορετικό σύνολο υποκαταστάσιμων αντικειμένων. Ετσι, πιθανώς να μπορούμε να συνδέσομε μετρικές με συγκεκριμένους σκοπούς της χρήσης του μοντέλου μας, όπως να συνδέσομε την L_1 νόρμα αναζήτηση υποκαταστασίμων αντικειμένων, την L_f όταν απλά θέλομε να περιπλανηθούμε στην ΒΠΛ, και την $L_{-|(inf}$ όταν ψάχνομε για λογισμικό που πληρεί όλες τις απαιτήσεις μας με τον υψηλότερο βαθμό από όλα τα άλλα αντικείμενα στην ΒΠΛ.

Επίσης, αν ένα σύνολο αντικειμένων μπορεί να περιγραφεί με ένα μοντέλο όπου οι κλάσεις του χαρακτηρίζονται από γνωρίσματα, τότε μπορούμε να καθορίσομε για το μοντέλο αυτό τα γνωρίσματα που είναι βασικά για τα αντικείμενα αυτά. Τότε θα μπορούσαμε να εφαρμόσομε τον αλγόριθμο για τον υπολογισμό του βαθμού υποκατάστασης ή ομοιότητας μεταξύ των αντικειμένων αυτών με βάση τα νέα γνωρίσματα που θέσαμε και να βρίσκομε σχέσεις υποκατάστασης μεταξύ αντικειμένων, διαφορετικών από λογισμικό.

Ενα πολύ απλό παράδειγμα έρχεται από τον χώρο της ψηφιακής σχεδίασης. Ας υποθέσομε ότι τα αντικείμενα που μας ενδιαφέρουν είναι οι διάφορες πύλες. Στόχος μας είναι η ανεύρεση των πυλών που ταιριάζουν με κάποια συγκεκριμένη, με σκοπό να την αντικαταστήσουν αν χρειαστεί. Το πρώτο βήμα που έχομε να κάνομε είναι να περιγράψομε τον κόσμο των πυλών και να ορίσομε ποια είναι τα στοιχεία που μας ενδιαφέρουν. Στην περίπτωσή μας είναι οι ακροδέκτες (pins) τους. Κατόπιν δεν έχομε παρά να ορίσομε την μετρική που μας ενδιαφέρει για τον σκοπό μας. Στην περίπτωση αυτή μια πιθανή μετρική είναι η L_1 νόρμα. Κατόπιν δεν έχομε παρά να χρησιμοποιήσομε τον αλγόριθμο που χρησιμοποιούμε και για τα λογισμικά αντικείμενα. Το σύνολο απάντησης μας δίνει όλες τις πύλες που μπορούν να χρησιμοποιηθούν ικανοποιώντας τις απαιτήσεις μας.

Βιβλιογραφία

[PrDiaz-Fre]

Prieto-Diaz Ruben and Peter Freeman, "Classifying Software for Reusability," IEEE Software, pp. 6 - 16, Junary 1987.

[PrDiaz]

Prieto-Diaz Ruben, "Implementing Faceted Classification for Software Reuse", Communications of the ACM, vol. 34, no. 5, 1991

[Pin]

Xavier Pintado , "Selection and Exploration in an Object-Oriented Environment : The Affinity Browser," in Object Managemnt, pp. 79 - 87, 1990.

[SalFoxWu]

Salton Gerard, Edward Fox, and Harry Wu, "Extended Boolean Information Retrieval," Communication of the ACM, vol. 26, pp. 1022 - 1036, December 1983.

[Pin-Tsic91]

Xavier Pintado and D. Tsichritzis, "Fuzzy Relationships and Affinity Links," Report, 1991.

[Pin-Tsic90]

Xavier Pintado and D. Tsichritzis, "An Affinity Browser," in Object Management, 1990.

[SalWu]

Salton Gerard and Harry Wu, "A term weighting model Based on Utility Theory," in Information Retrieval Research, Butterworths & Co. Ltd, 1981.

[Goyal]

Panaj Goyal, "Intelligent Information Systems: The Concept of an Intelligent Document," Information Systems, vol. 14, no. 4, pp. 351 - 358, 1989.

[Teu-Sch]

Teufel Bernd and Stephanie Schmidt, "Full Text Retrieval Based On Syntactic Similarities," Information Systems, vol. 13, pp. 65 - 70, 1988.

[McIlroy]

M. D. McIlroy, "Mass-Produced Sortware Components", Software Engineering Concepts and Techniques, Pertocelli/Charter, Brussele, Belgium, pp. 88-98, 1976.

[ITHACA.91.E2]

Martin Weber, "The CooL Implementation Model," Institute of Computer Science, Fundation of Research and Technology - Hellas, Heraklio, Crete, 1991.

[ITHACA.92.E2.#1]

Yannis Haralabides, George Vlodakis, Martin Weber, " The CooL Implementation Model for the SIB," Institute of Computer Science, Fundation of Research and Technology - Hellas, Heraklio, Crete, 1992.

[MBJK]

John Mylopoulos, Alex Borgida, Matthias Jarke and Manolis Koubarakis, "TELOS : Representing Knowledge About Information Systems," ACM Transactions On Information Sysytems, vol. 8, no. 4, pp. 325 - 362, 1990.

[SIB]

Panos Constantopoulos, Martin Doerr, Yannis Vassiliou, "Repositories for Sortware Reuse : The Sortware Information Base," to appear in Proc. IFIP WG 8.1 Working Conference on Information Systems Development Process, 1993.

[Veze91]

Κωνσταντίνος Βεζερίδης, "The organization of an SIB for software reuse by a programming community," Master’s Thesis, University of Crete.

[Hor]

Susan Horwitz, "Identifying the Semantic and Textual Differences between Two Versions of a Programm," Proceedings of the ACM SIGPLAN’90, Conference on Programming Language Design and Implementation, White Plains, New York, June 20-22, 1990.

[ITHACA.90.E3.6.#1]

Maria Grazia Fugini, Stefano Faustle, "Functional Description of Classes for Similarity Queries of the Selection Tool," Politecnico di Milano, September, 1990.

[ITHACA.92.E2.#2]

Costas Dadouris, Martin Doerr, Ioanna Gardiki, Manolis Marakakis, Elena Pataki, Eleni Petra, George Spanoudakis, George Yeorgiannakis, "Implementation of the SIB", Institute of Computer Science, Fundation of Research and Technology - Hellas, Heraklio, Crete, January, 1992.

[REBOOT]

Lars Sivert Sorumgard, Guttorm Sindre and Frode Stokke, "Experiences in Reusable Components Classification," ERCIM Workshop on Methods and Tools for Software Reuse, Institute of Computer Science, Heraklion, Crete, October 29-30, 1992.

Παράρτημα Α.

Μοντέλο περιγραφής λογισμικών αντικειμένων

```
BEGINTRANSACTION

{*********************************************************
**      The System Model                         *
*********************************************************}
TELL IndividualClass System in M1_Class, ImplementationModel, DesignModel with
        entity
                : SystemUnit;
                : SystemProgram;
                : SystemLocation;
                : SystemClassification
        construct
                : SystemApplication;
                : SystemModule;
                : SystemSoftwareInfo;
                : SystemEnvironmentInfo
        description
                : SystemApplication;
                : SystemModule
end System
TELL IndividualClass SystemEntity in M2_Class isA Entity
end SystemEntity
TELL IndividualClass SystemConstruct in M2_Class isA Construct
end SystemConstruct
TELL IndividualClass SystemDescription in M1_Class, Description
    isA ID, DD with
      attribute
      {single}
            info : SystemSoftwareInfo
end SystemDescription

{*************************************************************
** The SystemSoftwareInfo describes the implementation aspects *
** of a software component. It consists of the following:      *
```

```
** - keyword       : Classes that can be used to classify this *
**                 component (can be more than one)         *
** - shortDescription: This is a one sentence description      *
**                   of the component                        *
** - sourceLanguages: The languages used (can be more than one)*
** - sourceLocation: The name of the file containing the source*
** - documentationLocation: file with documentation            *
** - testLogLocation: file with test data, etc.               *
** - testState     : a number between 0 and 100, 0 means not   *
**                 yet tested, 100 means proven to be correct*
** - version       : The version number                       *
** - releaseDate   : Date of release for this component       *
** - author        : The author of this component             *
**************************************************************}

TELL IndividualClass SystemSoftwareInfo in M1_Class, SystemConstruct with
        attribute
        {* necessary *}
                keywords: SystemClassification;
                shortDescription: String;
                sourceLanguages: String;
        {* necessary, single *}
                sourceLocation: SystemFile;
        {* single *}
                documentationLocation : SystemFile;
                testLogLocation  : SystemFile;
                testState: Integer;
                version: String;
                releaseDate: String;
                author: String
end SystemSoftwareInfo
TELL IndividualClass SystemClassification in M1_Class, SystemEntity
end SystemClassification
TELL IndividualClass SystemLocation in M1_Class, SystemEntity
end SystemLocation
TELL IndividualClass SystemFile in M1_Class, SystemConstruct isA SystemLocation
```

```
end SystemFile
TELL IndividualClass SystemDirectory in M1_Class, SystemConstruct isA SystemFile with
    attribute
        path: String;
        files: SystemFile
end SystemDirectory
TELL IndividualClass SystemSourceFile in M1_Class, SystemConstruct isA SystemFile
end SystemSourceFile
TELL IndividualClass SystemExecutableFile in M1_Class, SystemConstruct isA SystemFile
end SystemExecutableFile
TELL IndividualClass SystemDocumentationFile in M1_Class, SystemConstruct isA SystemFile
end SystemDocumentationFile
TELL IndividualClass SystemTestFile in M1_Class, SystemConstruct isA SystemFile
end SystemTestFile
TELL IndividualClass SystemEnvironmentInfo in M1_Class, SystemConstruct with
    attribute
        directories: SystemDirectory;
        executableFiles: SystemExecutableFile;
        envVariables: String;
        installationSteps: String;
        compilationSteps: String;
        start: SystemExecutableFile
end SystemEnvironmentInfo
TELL IndividualClass SystemApplication in M1_Class, SystemConstruct, Description
         isA SystemDescription with
    attribute
        modules     : SystemModule;
        neededApplications : SystemApplication;
        programs    : SystemProgram;
        environment : SystemEnvironmentInfo
end SystemApplication
TELL IndividualClass SystemModule
    in M1_Class, SystemConstruct, Description
    isA SystemDescription with
    attribute
        subModules    : SystemModule;
```

```
        neededModules : SystemModule;
        files         : SystemUnit
end SystemModule
TELL IndividualClass SystemUnit in M1_Class, SystemEntity
end SystemUnit
TELL IndividualClass SystemProgram in M1_Class, SystemEntity
end SystemProgram


{**********************************************************
**    The CooL Model                    *
**********************************************************}

TELL IndividualClass CooL in M1_Class, ImplementationModel with
    entity
        : CooLException
    construct
        : CooLConstant;
        : CooLVariable;
        : CooLExternalVariable;
        : CooLType;
        : CooLTransaction;
        : CooLProcedure;
        : CooLExternalProcedure;
        : CooLMethod;
        : CooLRedefinedMethod;
        : CooLPersistentObjectType;
        : CooLVolatileObjectType;
        : CooLFile;
        : CooLProgram
    description
        : CooLProcedure;
        : CooLMethod;
        : CooLRedefinedMethod;
        : CooLPersistentObjectType;
        : CooLVolatileObjectType;
```

```
        : CooLFile;
        : CooLProgram
end CooL

TELL IndividualClass CooLEntity in M2_Class isA Entity
end CooLEntity
TELL IndividualClass CooLConstruct in M2_Class isA Construct
end CooLConstruct
TELL IndividualClass CooLDescription in M1_Class, Description isA ID with
    attribute
    {single}
        info : SystemSoftwareInfo
end CooLDescription

{* exception *}

TELL IndividualClass CooLException in M1_Class, CooLEntity
end CooLException

{*********************************************************
** constants are described by their type (which must be a  *
** basic type) and by their value (which is represented as *
** a string).                                   *
*********************************************************}
TELL IndividualClass CooLConstant in M1_Class, CooLConstruct with
    attribute
    {necessary}
        type : CooLBasicType;
        value : String
end CooLConstant

{* types *}
TELL IndividualClass CooLType in M1_Class, CooLConstruct
end CooLType
TELL IndividualClass CooLDataType in M1_Class, CooLConstruct isA CooLType
end CooLDataType
```

```
{* basic types *}

TELL IndividualClass CooLBasicType in M1_Class, CooLConstruct isA CooLDataType
end CooLBasicType

{*****************************************************
** The basic CooL types are already defined in CooL, so *
** they are instances of BasicType. All other type      *
** constructs must be instanciated to get a real CooL   *
** type: e.g. an array is not a CooL type, but must be  *
** instantiated by giving the range and the element type*
*****************************************************}

TELL IndividualClass CooLINT in S_Class, CooLBasicType
end CooLINT
TELL IndividualClass CooLUnsignedINT in S_Class isA CooLINT
end CooLUnsignedINT
TELL IndividualClass CooLShortINT in S_Class isA CooLINT
end CooLShortINT
TELL IndividualClass CooLUnsignedShortINT in S_Class isA CooLUnsignedINT, CooLShortINT
end CooLUnsignedShortINT
TELL IndividualClass CooLFloat in S_Class, CooLBasicType
end CooLFloat
TELL IndividualClass CooLDouble in S_Class, CooLBasicType
end CooLDouble
TELL IndividualClass CooLBool in S_Class, CooLBasicType
end CooLBool
TELL IndividualClass CooLChar in S_Class, CooLBasicType
end CooLChar
TELL IndividualClass CooLAddress in S_Class, CooLBasicType
end CooLAddress

{**********************************************************
** Structured types in CooL are instantiations of one of the *
** following definitions                                 *
**********************************************************}
```

```
TELL IndividualClass CooLArrayType in M1_Class, CooLConstruct isA CooLDataType with
    attribute
    {necessary}
        numEl : Integer;
    {necessary, single}
        element : CooLType
end CooLArrayType
TELL IndividualClass CooLRecordType in M1_Class, CooLConstruct isA CooLDataType with
    attribute
    {necessary}
        fields : CooLType
end CooLRecordType
TELL IndividualClass CooLUnionType in M1_Class, CooLConstruct isA CooLDataType with
    attribute
    {necessary}
        fields : CooLType
end CooLUnionType
TELL IndividualClass CooLSetType in M1_Class, CooLConstruct isA CooLDataType with
    attribute
    {necessary, single}
        element : CooLType
end CooLSetType
TELL IndividualClass CooLProcedureType in M1_Class, CooLConstruct isA CooLDataType with
    attribute
        inParameters : CooLType;
        inoutParameters : CooLType;
        outParameters : CooLType;
    {single}
        returnType : CooLType
end CooLProcedureType
TELL IndividualClass CooLRefType in M1_Class, CooLConstruct isA CooLDataType with
    attribute
    {necessary, single}
        referencedType : CooLType
end CooLRefType
TELL IndividualClass CooLVariable in M1_Class, CooLConstruct with
```

```
        attribute
        {necessary, single}
                type  : CooLType
end CooLVariable
TELL IndividualClass CooLExternalVariable in M1_Class, CooLConstruct
                                isA CooLVariable
end CooLExternalVariable

{************************************************************
** the behavioural part of CooL: procedures, methods and       *
** transactions                                          *
************************************************************}

TELL IndividualClass CooLExecutableBlock in M1_Class, CooLConstruct with
        attribute
                executes: CooLExecutableBlock
end CooLExecutableBlock
TELL IndividualClass CooLTransaction in M1_Class, CooLConstruct
                            isA CooLExecutableBlock
end CooLTransaction
TELL IndividualClass CooLRoutine in M1_Class, CooLConstruct isA CooLExecutableBlock with
        attribute
                inParameters : CooLType;
                inoutParameters : CooLType;
                outParameters : CooLType;
                exceptions : CooLException;
        {single}
                returnType : CooLType
end CooLRoutine
TELL IndividualClass CooLProcedure in M1_Class, CooLConstruct, Description
                           isA CooLRoutine, CooLDescription
end CooLProcedure
TELL IndividualClass CooLExternalProcedure in M1_Class, CooLConstruct isA CooLRoutine
end CooLExternalProcedure
TELL IndividualClass CooLMethod in M1_Class, CooLConstruct, Description
        isA CooLRoutine, CooLDescription
```

```
end CooLMethod
TELL IndividualClass CooLRedefinedMethod in M1_Class, CooLConstruct, Description
                isA CooLMethod with
        attribute
        {necessary, single}
                redefines : CooLMethod
end CooLRedefinedMethod

{*************************************************************
** Objecttypes are described by their supertype (if any),   *
** the INITIALLY sequence, their parameters for the instan- *
** tiation, their instance variables, and their exported and*
** internal methods. If methods are redefined these are     *
** instantiated  under 'RedefinedMethod'.                   *
** Nonfunctional aspects of an object type are described    *
** with attribute 'info'.                                   *
*************************************************************}

TELL IndividualClass CooLObjectType in M1_Class isA CooLType with
        attribute
        {single}
                supertype : CooLObjectType;
                initially : CooLExecutableBlock;
        {attribute}
                inParameters : CooLVariable;
                inoutParameters : CooLVariable;
                outParameters : CooLVariable;
                stateVariables : CooLVariable;
                methods : CooLMethod;
                internalMethods : CooLMethod
end CooLObjectType
TELL IndividualClass CooLPersistentObjectType
        in M1_Class, CooLConstruct, Description
        isA CooLObjectType, CooLDescription with
        attribute
        {single}
```

```
        supertype : CooLPersistentObjectType;
        keyParameters : CooLVariable
end CooLPersistentObjectType
TELL IndividualClass CooLVolatileObjectType
    in M1_Class, CooLConstruct, Description
    isA CooLObjectType, CooLDescription with
    attribute
    {single}
        supertype : CooLVolatileObjectType
end CooLVolatileObjectType

TELL IndividualClass CooLFile in M1_Class, CooLConstruct, Description
    isA CooLDescription, SystemUnit with
    attribute
        exceptions  : CooLException;
        constants   : CooLConstant;
        dataTypes   : CooLDataType;
        variables   : CooLType;
        procedures  : CooLProcedure;
        objectTypes : CooLObjectType;
        references  : CooLFile
end CooLFile

{* here now the description of programs as a file with a *
** main procedure                                        *}

TELL IndividualClass CooLProgram in M1_Class, CooLConstruct, Description
    isA CooLFile, CooLDescription, SystemProgram with
    attribute
    {necessary, single}
        main        : CooLProcedure
end CooLProgram
ENDTRANSACTION
```

Παράρτημα Β.

Δύο παραδείγματα περιγραφής λογισμικών αντικειμένων με βάση το μοντέλο υλοποιησης CooL

Παράδειγμα 1

```
BEGINTRANSACTION

TELL IndividualClass AddressType
  in S_Class, CooLRecordType with
  fields
    Street:  String30;
    StreetNo:  CooLINT;
    PostalCode:  CooLINT;
    City:  String30;
    Country:  String30
end AddressType

TELL IndividualClass String30
  in S_Class, CooLArrayType with
  numEl  : 30
  element  : CooLChar
end String30

TELL IndividualClass CooLProcedure‘main
  in S_Class, CooLProcedure with
  info
    : main‘Info
  executes
    : choose_from_menu;
    : rentCar;
    : returnCar;
    : addCustomer;
    : payment;
    : addCar;
    : printStatistics
end CooLProcedure‘main

TELL IndividualClass main‘Info
  in S_Class, SystemSoftwareInfo with
  shortDescription
```

```
    : "This is the main for the program."
  sourceLanguages
    : "CooL"
  sourceLocation
    : SystemSourceFile‘car_main
  testState
    : 0
  version
    : "0.0/0"
  author
    : "Martin Weber"
end main‘Info

TELL IndividualClass choose_from_menu
       in S_Class, CooLProcedure with
       returnType
              : CooLINT
end choose_from_menu

TELL IndividualClass CT
  in S_Class, CooLPersistentObjectType with
  supertype
    : PersonType
  initially
    : CT‘INITIALLY
  stateVariables
    : CT‘preferedModel;
    : CT‘distance
  methods
    : CT‘setPreferedModel;
    : CT‘changeAddress
  internalMethods
    : CT‘setDistance
end CT

TELL IndividualClass CT‘preferedModel
```

```
  in S_Class, CooLVariable with
  type: ModelType
end CT‘preferedModel

TELL IndividualClass CT‘distance
  in S_Class, CooLVariable with
  type: CooLINT
end CT‘distance

TELL IndividualClass CT‘INITIALLY
  in S_Class, CooLExecutableBlock with
  executes
    : CT‘setDistance
end CT‘INITIALLY

TELL IndividualClass CT‘changeAddress
  in S_Class, CooLRedefinedMethod with
  redefines
    : PersonType‘changeAddress
  inParameters
    : CT‘changeAddress‘address
end CT_changeAddress

TELL IndividualClass CT‘changeAddress‘address
  in S_Class, CooLVariable with
  type: AddressType
end CT‘changeAddress‘address

TELL IndividualClass CT‘setDistance
  in S_Class, CooLMethod
end CT‘setDistance

TELL IndividualClass CT‘setPreferedModel
  in S_Class, CooLMethod with
  inParameters
    : CT‘setPreferedModel‘model
```

```
end CT‘setPreferedModel

TELL IndividualClass CT‘setPreferedModel‘model
  in S_Class, CooLVariable with
  type: ModelType
end CT‘setPreferedModel‘model

TELL IndividualClass rental_statistics
  in S_Class, CooLFile with
  exceptions
    : NoAmountBeforeCarReturn
  objectTypes
    : RentalType;
    : StatisticsType
  references
    : customer;
    : car;
    : date
  info
    : rental_statistics‘Info
end rental_statistics

TELL IndividualClass rental_statistics‘Info
  in S_Class, SystemSoftwareInfo with
  shortDescription
    : "This file implements rentals and statistics."
  sourceLanguages
    : "CooL"
  sourceLocation
    : SourceFile‘rental_statistics
  testState
    : 0
  version
    : "0.0/0"
  author
    : "Martin Weber"
```

```
  keywords
    : statistics;
    : rental
end rental_statistics‘Info

TELL IndividualClass statistics in S_Class, SystemClassification
end statistics

TELL IndividualClass rental in S_Class, SystemClassification
end rental

TELL IndividualClass car_main in S_Class, CooLProgram with
      procedures
            : choose_from_menu;
            : rentCar;
            : returnCar;
            : addCustomer;
            : payment;
            : addCar;
            : printStatistics
      main
            : CooLProcedure‘main
      references
            : car;
            : customer;
            : rental_statistics;
            : date
      info
            : car_main‘Info
end car_main

TELL IndividualClass popi_main in S_Class, CooLProgram with
      procedures
            : choose_from_menu;
          : rentCar;
          : returnCar;
```

```
            : addCustomer;
            : payment
      main
              : CooLProcedure‘main
      references
            : car;
            : customer;
            : rental_statistics
end popi_main

TELL IndividualClass car_main‘Info
   in S_Class, SystemSoftwareInfo with
      shortDescription
              : "This is an application for car rental offices."
      sourceLanguages
              : "CooL";
              : "C"
      sourceLocation
              : SystemSourceFile‘car_main
      testState
              : 0
      version
              : "0.0/0"
      author
              : "Martin Weber"
      keywords
              : car_rental;
              : rental
end car_main‘Info

TELL IndividualClass car_rental in S_Class, SystemClassification
end car_rental

TELL IndividualClass CarRental in S_Class, SystemApplication with
  programs
    : car_main
```

```
end CarRental

TELL IndividualClass AF‘CarRental in S_Class, AF with
  impDescr
    : CarRental
end AF‘CarRental

TELL IndividualClass rentCar
  in S_Class, CooLProcedure
end rentCar

TELL IndividualClass returnCar
  in S_Class, CooLProcedure
end returnCar

TELL IndividualClass addCustomer
  in S_Class, CooLProcedure
end addCustomer

TELL IndividualClass payment
  in S_Class, CooLProcedure
end payment

TELL IndividualClass addCar
  in S_Class, CooLProcedure
end addCar

TELL IndividualClass printStatistics
  in S_Class, CooLProcedure
end printStatistics

TELL IndividualClass SystemSourceFile‘car_main
  in S_Class, SystemSourceFile
end SystemSourceFile‘car_main

TELL IndividualClass car
```

```
  in S_Class, CooLFile
       with
       references
              : from_car_to_test
end car

TELL IndividualClass customer
  in S_Class, CooLFile
end customer

TELL IndividualClass date
  in S_Class, CooLFile
end date

TELL IndividualClass from_car_to_test
  in S_Class, CooLFile
end from_car_to_test

TELL IndividualClass PersonType
  in S_Class, CooLPersistentObjectType
end PersonType

TELL IndividualClass ModelType
  in S_Class, CooLPersistentObjectType
end ModelType

TELL IndividualClass PersonType‘changeAddress
  in S_Class, CooLMethod
end PersonType‘changeAddress

TELL IndividualClass NoAmountBeforeCarReturn
  in S_Class, CooLException
end NoAmountBeforeCarReturn

TELL IndividualClass RentalType
  in S_Class, CooLPersistentObjectType
```

```
end RentalType

TELL IndividualClass StatisticsType
  in S_Class, CooLPersistentObjectType
end StatisticsType

TELL IndividualClass SourceFile‘rental_statistics
  in S_Class, SystemSourceFile
end SourceFile‘rental_statistics
ENDTRANSACTION
```

Παράδειγμα 2

```
BEGINTRANSACTION

TELL IndividualClass INTPointer
        in S_Class, CooLRefType with
        referencedType    : CooLINT
end INTPointer

TELL IndividualClass DOUBLEPointer
        in S_Class, CooLRefType with
        referencedType  : CooLDouble
end DOUBLEPointer

TELL IndividualClass CHARPointer
        in S_Class, CooLRefType with
        referencedType    : CooLChar
end CHARPointer

TELL IndividualClass Array_Of_Functions
        in S_Class, CooLArrayType with
        numEl        : 10
        element : FUNCTION
end Array_Of_Functions

TELL IndividualClass FUNCTION
        in S_Class, CooLProcedureType with
        inParameters        : CooLDouble
        returnType  : CooLDouble
end FUNCTION

TELL IndividualClass MY_FUNCTION
        in S_Class, CooLRefType with
        referencedType    : Array_Of_Functions
end MY_FUNCTION
```

```
TELL IndividualClass LEMMA
        in S_Class, CooLRecordType with
        fields
                : CooLDouble;
                : CooLINT
end LEMMA

TELL IndividualClass ARR_LEMMA
        in S_Class, CooLRefType with
        referencedType : LEMMA
end ARR_LEMMA

TELL IndividualClass BoundPointer
        in S_Class, CooLRefType with
        referencedType : Bound
end BoundPointer

TELL IndividualClass Bound
        in S_Class, CooLRecordType with
        fields
                : CooLINT;
                : CooLINT;
                : BoundPointer
end Bound

TELL IndividualClass TERM
        in S_Class, CooLRecordType with
      fields
                : CooLINT;
                : CooLINT;
                : CooLINT
end TERM

TELL IndividualClass TERMPointer
        in S_Class, CooLRefType with
        referencedType : TERM
```

```
end TERMPointer

TELL IndividualClass NODE
        in S_Class, CooLRecordType with
        fields
                : CooLINT;
                : CooLINT;
                : CooLINT;
                : NODEPointer
end NODE

TELL IndividualClass NODEPointer
        in S_Class, CooLRefType with
        referencedType : NODE
end NODEPointer

TELL IndividualClass STACK
        in S_Class, CooLRecordType with
       fields
                : INTPointer;
                : CooLINT
end STACK

TELL IndividualClass STACKPointer
        in S_Class, CooLRefType with
        referencedType : STACK
end STACKPointer


{************************************************************}




TELL IndividualClass brelax1File
        in S_Class, CooLFile with
```

```
        procedures
                : BRELAX_ALG;
                : check_SCR_solution;
                : brelax1
        references
                : relaxFile
end brelax1File


TELL IndividualClass brelax2File
        in S_Class, CooLFile with
        procedures
                : find_Delta_plus;
                : find_Delta_minus;
                : check_if_empty;
                : define_Js;
                : define_newJ;
                : brelax2
end brelax2File

TELL IndividualClass rankFile
        in S_Class, CooLFile with
        procedures
                : find_deriv;
                : cont_binsearch1;
                : cont_binsearch2;
                : discr_binsearch;
                : comp;
                : rank
end rankFile

TELL IndividualClass incrementFile
        in S_Class, CooLFile with
        procedures
                : find_smaller;
                : increment
```

```
end incrementFile


TELL IndividualClass relaxFile
	in S_Class, CooLFile with
	procedures
		: find_deriv;
		: binsearch;
		: single_binsearch;
		: cont_binsearch;
		: single_cont_binsearch;
		: SCR_relax;
		: check_relax_solution;
		: relax
end relaxFile


TELL IndividualClass lexincFile
	in S_Class, CooLFile with
	procedures
		: find_min;
		: lexIncrement
end lexincFile

TELL IndividualClass contFile
	in S_Class, CooLFile with
	procedures
		: isEmpty;
		: isInteger;
		: CeilX;
		: find_newLambda;
		: computeJ;
		: PrimeX;
		: lemmaIsOk;
		: computeSolution;
		: resetSets;
```

```
                : resetFunctions;
                : contMINIMAX
        references
                : mrFile
end contFile

TELL IndividualClass mrFile
        in S_Class, CooLFile with
        procedures
                : inverse;
                : find_feasible;
                : find_lambda;
                : find_xopt;
                : solveMR
end mrFile

TELL IndividualClass incrmtFile
        in S_Class, CooLFile with
        procedures
                : newDval;
                : findminD;
                : increment
end incrmtFile

TELL IndividualClass smincrmtFile
        in S_Class, CooLFile with
        procedures
                : getbound;
                : get_data;
                : checkmin;
                : read_bounds;
                : problem_bounded;
                : fix_sat_vector;
                : sum;
                : check_sat;
                : built_P;
```

```
                : newDval;
                : findminD;
                : smincrement
end smincrmtFile

TELL IndividualClass gsmFile
        in S_Class, CooLFile with
        procedures
                : functZ;
                : functV;
                : equal;
                : functInc;
                : bubble;
                : gSMDR
end gsmFile

TELL IndividualClass ntwkdrFile
        in S_Class, CooLFile with
        procedures
                : show;
                : solveNtwkdr;
                : functIncNT;
                : increase;
                : decrease;
                : initStack;
                : push;
                : pop;
                : emptyStack;
                : getGraph;
                : ntwkDR
end ntwkdrFile

TELL IndividualClass smlpFile
        in S_Class, CooLFile with
        procedures
                : bubble;
```

```
                : smLP
end smlpFile



TELL IndividualClass BRELAX_ALG
        in S_Class, CooLProcedure with
        inParameters
                : CooLINT;
                : CooLDouble;
                : CHARPointer
        outParameters
                : DOUBLEPointer
end BRELAX_ALG

TELL IndividualClass check_SCR_solution
        in S_Class, CooLProcedure with
        inParameters
                : DOUBLEPointer;
                : CHARPointer;
                : CooLINT;
                : DOUBLEPointer
        outParameters
                : DOUBLEPointer
        returnType
                : CooLChar
end check_SCR_solution

TELL IndividualClass brelax1
        in S_Class, CooLProcedure with
        inParameters
                : CooLINT;
                : CooLDouble;
                : MY_FUNCTION;
                : DOUBLEPointer
        outParameters
```

```
                : DOUBLEPointer
        returnType
                : CooLDouble
end brelax1


TELL IndividualClass find_Delta_plus
        in S_Class, CooLProcedure with
        inParameters
                : CHARPointer;
                : DOUBLEPointer;
                : DOUBLEPointer;
                : CooLINT
        returnType
                : CooLDouble
end find_Delta_plus

TELL IndividualClass find_Delta_minus
        in S_Class, CooLProcedure with
        inParameters
                : CHARPointer;
                : DOUBLEPointer;
                : DOUBLEPointer;
                : CooLINT
        returnType
                : CooLDouble
end find_Delta_minus

TELL IndividualClass check_if_empty
        in S_Class, CooLProcedure with
        inParameters
                : CHARPointer;
                : CooLINT
        returnType
                : CooLChar
end check_if_empty
```

```
TELL IndividualClass define_Js
        in S_Class, CooLProcedure with
        inParameters
                : CHARPointer;
                : DOUBLEPointer;
                : DOUBLEPointer;
                : CooLINT
        outParameters
                : CHARPointer;
                : CHARPointer
end define_Js

TELL IndividualClass define_newJ
        in S_Class, CooLProcedure with
        inParameters
                : CooLINT;
        inoutParameters
                : CHARPointer;
                : CHARPointer
end define_newJ

TELL IndividualClass brelax2
        in S_Class, CooLProcedure with
        inParameters
                : CooLINT;
                : CooLDouble;
                : MY_FUNCTION;
             : DOUBLEPointer
        outParameters
             : DOUBLEPointer
        returnType
                : CooLDouble
end brelax2

TELL IndividualClass find_deriv
        in S_Class, CooLProcedure with
```

```
    inParameters
        : CooLINT;
        : CooLDouble
    returnType
        : CooLDouble
end find_deriv

TELL IndividualClass cont_binsearch1
    in S_Class, CooLProcedure with
    inParameters
        : CooLDouble;
        : CooLDouble;
        : CooLINT;
        : CooLDouble;
        : CooLDouble;
        : CooLDouble;
        : CooLDouble;
        : CooLDouble
    returnType
        : CooLDouble
end cont_binsearch1

TELL IndividualClass cont_binsearch2
    in S_Class, CooLProcedure with
    inParameters
      : CooLDouble;
      : CooLDouble;
        : CooLINT;
      : CooLDouble;
      : CooLDouble;
        : ARR_LEMMA;
      : CooLDouble
    returnType
        : CooLDouble
end cont_binsearch2
```

```
TELL IndividualClass discr_binsearch
        in S_Class, CooLProcedure with
        inParameters
                : CooLDouble;
                : CooLDouble;
                : CooLDouble;
                : ARR_LEMMA
        returnType
                : CooLINT
end discr_binsearch

TELL IndividualClass comp
        in S_Class, CooLProcedure with
        inParameters
                : ARR_LEMMA;
                : ARR_LEMMA
        returnType
                : CooLINT
end comp

TELL IndividualClass rank
        in S_Class, CooLProcedure with
        inParameters
                : CooLINT;
                : CooLDouble;
                : MY_FUNCTION;
                : DOUBLEPointer
        returnType
                : CooLDouble
end rank

TELL IndividualClass find_smaller
        in S_Class, CooLProcedure with
        inParameters
                : DOUBLEPointer;
                : INTPointer;
```

```
        : CooLINT;
        : CooLINT
    returnType
        : CooLINT
end find_smaller


TELL IndividualClass binsearch
    in S_Class, CooLProcedure with
    inParameters
        : CooLDouble;
        : CooLDouble;
        : CooLINT;
        : CooLDouble;
        : CooLChar;
        : CooLDouble;
        : CooLDouble
    returnType
        : CooLDouble
end binsearch

TELL IndividualClass single_binsearch
    in S_Class, CooLProcedure with
    inParameters
        : CooLINT;
        : CooLDouble;
        : CooLChar
    returnType
        : CooLDouble
end single_binsearch

TELL IndividualClass cont_binsearch
    in S_Class, CooLProcedure with
    inParameters
        : CooLDouble;
        : CooLDouble;
```

```
                : CooLDouble;
                : CooLINT;
                : DOUBLEPointer;
                : DOUBLEPointer;
                : CHARPointer;
                : CooLDouble;
                : CooLDouble
        returnType
                : CooLDouble
end cont_binsearch

TELL IndividualClass single_cont_binsearch
        in S_Class, CooLProcedure with
        inParameters
                : CooLDouble;
                : CooLINT;
                : DOUBLEPointer;
                : DOUBLEPointer;
                : CHARPointer
        returnType
                : CooLDouble
end single_cont_binsearch

TELL IndividualClass SCR_relax
        in S_Class, CooLProcedure with
        inParameters
                : CHARPointer;
                : CooLDouble;
                : CooLINT
        outParameters
                : DOUBLEPointer
end SCR_relax

TELL IndividualClass check_relax_solution
        in S_Class, CooLProcedure with
        inParameters
```

```
        : DOUBLEPointer;
        : CHARPointer;
        : CooLINT
    returnType
        : CooLChar
end check_relax_solution

TELL IndividualClass relax
    in S_Class, CooLProcedure with
    inParameters
        : CooLINT;
        : CooLDouble;
        : MY_FUNCTION
    outParameters
        : DOUBLEPointer
    returnType
        : CooLDouble
end relax

TELL IndividualClass find_min
    in S_Class, CooLProcedure with
    inParameters
        : CooLINT;
        : INTPointer;
        : MY_FUNCTION;
        : INTPointer
    returnType
        : CooLINT
end find_min

TELL IndividualClass lexIncrement
    in S_Class, CooLProcedure with
    inParameters
        : CooLINT;
        : CooLINT;
        : MY_FUNCTION;
```

```
                : INTPointer
        outParameters
                : INTPointer
        returnType
                : CooLINT
end lexIncrement

TELL IndividualClass isEmpty
        in S_Class, CooLProcedure with
        inParameters
                : INTPointer;
                : CooLINT
        returnType
                : CooLINT
end isEmpty

TELL IndividualClass isInteger
        in S_Class, CooLProcedure with
        inParameters
                : DOUBLEPointer;
                : CooLINT
        returnType
                : CooLINT
end isInteger

TELL IndividualClass CeilX
        in S_Class, CooLProcedure with
        inParameters
                : DOUBLEPointer;
                : INTPointer;
                : INTPointer
        outParameters
                : DOUBLEPointer
end CeilX

TELL IndividualClass find_newLambda
```

```
        in S_Class, CooLProcedure with
        inParameters
                : MY_FUNCTION;
                : INTPointer;
                : CooLINT;
                : INTPointer
        returnType
                : CooLDouble
end find_newLambda

TELL IndividualClass computeJ
        in S_Class, CooLProcedure with
        inParameters
                : MY_FUNCTION;
                : INTPointer;
                : CooLINT;
                : INTPointer;
                : CooLDouble;
        outParameters
                : INTPointer
end computeJ

TELL IndividualClass PrimeX
        in S_Class, CooLProcedure with
        inParameters
                : MY_FUNCTION;
                : INTPointer;
                : CooLINT;
                : INTPointer;
                : CooLDouble
        outParameters
                : INTPointer
end PrimeX

TELL IndividualClass lemmaIsOk
        in S_Class, CooLProcedure with
```

```
	inParameters
		: CooLINT;
		: CooLINT;
		: INTPointer;
		: INTPointer
	returnType
		: CooLINT
end lemmaIsOk

TELL IndividualClass computeSolution
	in S_Class, CooLProcedure with
	inParameters
		: INTPointer;
		: CooLINT;
		: CooLINT;
		: INTPointer
	outParameters
		: INTPointer
end computeSolution

TELL IndividualClass resetSets
	in S_Class, CooLProcedure with
	inParameters
		: INTPointer;
		: CooLINT;
		: INTPointer
	outParameters
		: INTPointer;
		: INTPointer;
		: INTPointer
end resetSets

TELL IndividualClass resetFunctions
	in S_Class, CooLProcedure with
	inParameters
		: INTPointer;
```

```
        : CooLINT
    outParameters
        : MY_FUNCTION
end resetFunctions

TELL IndividualClass contMINIMAX
    in S_Class, CooLProcedure with
    inParameters
        : CooLINT;
        : CooLINT;
        : MY_FUNCTION;
        : INTPointer
    outParameters
        : INTPointer
    returnType
        : CooLINT
end contMINIMAX

TELL IndividualClass inverse
    in S_Class, CooLProcedure with
    inParameters
        : MY_FUNCTION;
        : CooLDouble;
        : CooLINT;
        : CooLDouble;
        : CooLDouble;
        : CooLDouble
    returnType
        : CooLDouble
end inverse

TELL IndividualClass find_feasible
    in S_Class, CooLProcedure with
    inParameters
        : CooLDouble;
        : MY_FUNCTION;
```

```
                : DOUBLEPointer;
                : CooLDouble;
                : CooLDouble
        returnType
                : CooLINT
end find_feasible

TELL IndividualClass find_lambda
       in S_Class, CooLProcedure with
       inParameters
                : CooLDouble;
                : MY_FUNCTION;
                : CooLINT;
                : CooLDouble;
                : CooLDouble
        returnType
                : CooLDouble
end find_lambda

TELL IndividualClass find_xopt
        in S_Class, CooLProcedure with
        inParameters
                : MY_FUNCTION;
                : DOUBLEPointer;
                : CooLDouble;
                : CooLINT;
                : CooLDouble
        returnType
                : CooLDouble
end find_xopt

TELL IndividualClass solveMR
        in S_Class, CooLProcedure with
        inParameters
                : CooLINT;
                : CooLDouble;
```

```
        : MY_FUNCTION;
        : DOUBLEPointer
    outParameters
        : DOUBLEPointer
    returnType
        : CooLINT
end solveMR

TELL IndividualClass newDval
    in S_Class, CooLProcedure with
    inParameters
        : CooLINT;
        : INTPointer
    returnType
        : CooLDouble
end newDval

TELL IndividualClass findminD
    in S_Class, CooLProcedure with
    inParameters
        : CooLINT
    returnType
        : CooLINT
end findminD

TELL IndividualClass increment
    in S_Class, CooLProcedure with
    inParameters
        : CooLINT;
        : CooLINT;
        : MY_FUNCTION
    outParameters
        : INTPointer
    returnType
        : CooLDouble
end increment
```

```
TELL IndividualClass getbound
	in S_Class, CooLProcedure with
	inParameters
		: CooLINT
	returnType
		: CooLINT
end getbound

TELL IndividualClass get_data
	in S_Class, CooLProcedure with
	returnType
		: CooLINT
end get_data

TELL IndividualClass checkmin
	in S_Class, CooLProcedure with
	inParameters
		: CooLINT;
		: CooLINT
	returnType
		: CooLINT
end checkmin

TELL IndividualClass read_bounds
	in S_Class, CooLProcedure with
	returnType
		: BoundPointer
end read_bounds

TELL IndividualClass problem_bounded
	in S_Class, CooLProcedure with
      returnType
		: CooLINT
end problem_bounded

TELL IndividualClass fix_sat_vector
```

```
          in S_Class, CooLProcedure with
        returnType
                    : CooLINT
end fix_sat_vector

TELL IndividualClass sum
          in S_Class, CooLProcedure with
          inParameters
                    : CooLINT;
                    : INTPointer
          returnType
                    : CooLINT
end sum

TELL IndividualClass check_sat
          in S_Class, CooLProcedure with
          inParameters
                    : CooLINT;
                    : CooLINT
          returnType
                    : CooLINT
end check_sat

TELL IndividualClass built_P
          in S_Class, CooLProcedure with
          returnType
                    : CooLINT
end built_P

TELL IndividualClass smincrement
          in S_Class, CooLProcedure with
          inParameters
                    : CooLINT;
                    : MY_FUNCTION;
                    : CooLINT
          outParameters
```

```
                : INTPointer
        returnType
                : CooLDouble
end smincrement

TELL IndividualClass up_log
        in S_Class, CooLProcedure with
        inParameters
                : CooLINT;
                : CooLINT
        returnType
                : CooLINT
end up_log

TELL IndividualClass functZ
        in S_Class, CooLProcedure with
        inParameters
                : INTPointer;
                : CooLINT;
                : TERMPointer
        returnType
                : CooLDouble
end functZ

TELL IndividualClass functV
        in S_Class, CooLProcedure with
      inParameters
                : INTPointer;
                : CooLINT;
                : TERMPointer;
                : MY_FUNCTION
        returnType
                : CooLDouble
end functV

TELL IndividualClass equal
```

```
     in S_Class, CooLProcedure with
    inParameters
          : INTPointer;
          : CooLINT;
          : TERMPointer
     returnType
          : CooLINT
end equal

TELL IndividualClass functInc
     in S_Class, CooLProcedure with
    inParameters
          : CooLINT;
          : CooLINT;
          : TERMPointer;
          : MY_FUNCTION
     returnType
          : CooLDouble
end functInc

TELL IndividualClass bubble
     in S_Class, CooLProcedure with
    inParameters
          : CooLINT
     inoutParameters
          : TERMPointer
end bubble

TELL IndividualClass gSMDR
     in S_Class, CooLProcedure with
     inParameters
          : INTPointer;
          : INTPointer;
          : CooLINT;
          : MY_FUNCTION;
          : MY_FUNCTION
```

```
        returnType
                : CooLDouble
end gSMDR

TELL IndividualClass smLP
        in S_Class, CooLProcedure with
      inParameters
                : INTPointer;
                : CooLINT;
                : DOUBLEPointer;
                : MY_FUNCTION
        returnType
                : CooLDouble
end smLP

TELL IndividualClass ntwkDR
        in S_Class, CooLProcedure with
      inParameters
                : CooLINT;
                : CHARPointer;
                : CHARPointer;
                : MY_FUNCTION
        returnType
                : CooLDouble
end ntwkDR

TELL IndividualClass show
        in S_Class, CooLProcedure with
      inParameters
                : CHARPointer;
                : NODEPointer;
                : CooLINT;
                : CooLINT;
                : CooLINT;
                : INTPointer;
                : CooLDouble
```

```
end show

TELL IndividualClass solveNtwkdr
     in S_Class, CooLProcedure with
    inParameters
          : NODEPointer;
          : NODEPointer;
          : CooLINT;
          : CooLINT;
          : INTPointer;
          : MY_FUNCTION
     returnType
          : CooLDouble
end solveNtwkdr

TELL IndividualClass functIncNT
     in S_Class, CooLProcedure with
    inParameters
          : MY_FUNCTION;
          : INTPointer;
          : CooLINT
     returnType
        : CooLDouble
end functIncNT

TELL IndividualClass increase
     in S_Class, CooLProcedure with
    inParameters
          : CooLINT;
          : CooLINT
     inoutParameters
          : NODEPointer
end increase

TELL IndividualClass decrease
     in S_Class, CooLProcedure with
```

```
    inParameters
          : CooLINT;
          : CooLINT
     inoutParameters
          : NODEPointer
end decrease

TELL IndividualClass initStack
     in S_Class, CooLProcedure with
    inParameters
          : CooLINT
     inoutParameters
          : STACKPointer
end initStack

TELL IndividualClass push
     in S_Class, CooLProcedure with
    inParameters
          : CooLINT
     inoutParameters
        : STACKPointer
end push

TELL IndividualClass pop
     in S_Class, CooLProcedure with
     inoutParameters
        : STACKPointer
end pop

TELL IndividualClass emptyStack
     in S_Class, CooLProcedure with
    inoutParameters
        : STACKPointer
end emptyStack

TELL IndividualClass getGraph
```

```
        in S_Class, CooLProcedure with
        inParameters
                : CHARPointer;
                : INTPointer;
                : INTPointer;
                : INTPointer
        inoutParameters
                : NODEPointer;
                : NODEPointer
end getGraph

ENDTRANSACTION
```

Παράρτημα Γ.

Παράδειγμα περιγραφής σχέσεων υποκατάστασης στο μοντέλο περιγραφής σχέσεων προσέγγισης

Εδώ περιγράφομε σε γλώσσα TELOS τη σχέση υποκατάστασης MyProximity, που ορίζεται για λογισμικό που έχει περιγραφεί στο μοντέλο CooL. Οταν επιθυμούμε να ορίσομε σύνολα κριτηρίων, όπως τα περιγράψαμε στο κεφάλαιο 4, απλά ορίζομε το κάθε κριτήριο ως isA του ParametersDefinition, σε μετα-επίπεδο, ενώ σε επίπεδο απλών κλάσεων ορίζομε μία κλάση, για παράδειγμα την MyParameters_1, ως instanceOf όλων των κριτηρίων που επιθυμούμε να λάβομε υπόψιν μας για τον υπολογισμό του βαθμού υποκατάστασης. Κατόπιν ορίζομε σε επίπεδο απλών κλάσεων την MyProximity ως instanceOf της μετα-κλάσης Proximity, που περιγράψαμε στο κεφάλαιο 1, σχήμα 1.2 . Ολες οι ανά δύο σχέσεις λογισμικών αντικειμένων ορίζονται ως isA της MyProximity ή της οποιασδήποτε σχέσης υποκατάστασης θέλομε να περιγράψομε, και σε αυτή την περιγραφή δίνομε και τον υπολογιζόμενο βαθμό υποκατάστασης. Τελος, πρέπει να πούμε ότι για απλότητα κάνομε τη σύμβαση να μην δίνομε το γνώρισμα parameters όταν θέλομε να υπολογίσμομε τον βαθμό υποκατάστασης ως προς όλα τα κριτήρια που έχομε περιγράψει στο κεφάλαιο 3.

```
BEGINTRANSACTION

TELL Individual Procedures in M1_Class isA Parameter_Definition
end Procedures

TELL Individual References in M1_Class isA Parameter_Definition
end References

{ ορίζω ως κριτήρια, μόνο τις Procedures και References }
TELL Individual  MyParameters_1 in S_Class, Procedures,
References
end MyParameters_1

{ όταν περιγράφομε κάθε πρόγραμμα, δηλώνομε ότι είναι και isA Files }
TELL Individual Files in S_Class, Software_Object
end Files

{ η σχέση MyProximity θέλω να δίνει τον βαθμό υποκατάστασης με βάση }
{ τις InParameters και OutParameters που έχουν τα πραγράμματα      }
{ cont, mr.                                        }
TELL Attribute MyProximity
	components
		from : Files
		to : Files
	in S_Class, Proximity
	with attribute

		parameters : MyParameters_1;

		reflexive : 1;
		symmetric : 1;
		transitive : 0
end MyProximity

{ οι cont, mr έχουν περιγραφεί στο μοντέλο CooL ως προγράμματα, και }
```

```
{ επιπλέον ως isA CooL_File. Υπολογίστηκε ότι :                }
{ R( cont, mr) = 0.683,   R ( mr, cont) = 0.925   }
TELL Attribute cont_mr
        components
                from : cont
                to : mr
        in S_Class isA MyProximity
        with attribute
                weightF : 0.683;
                weightB : 0.925
end cont_mr

ENDTRANSACTION
```