Πειραματική Μελέτη Μεθόδω
Σύγκρισης Υπηρεσιών Διαδικτι
στα Πλαίσια της Συντήρηση
Υπηρεσιοκεντρικών Εφαρμογ

Ευφροσύνη Κουρή

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕ

— ♦ —

Ιωάννινα, Ιούνιος 2009

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF IOANNINA

ΠΕΙΡΑΜΑΤΙΚΗ ΜΕΛΕΤΗ ΣΥΓΚΡΙΣΗΣ ΥΠΗΡΕΣΙΩΝ ΔΙΑΔΙΚΤΥΟΥ ΣΤΑ ΠΛΑΙΣΙΑ ΤΗΣ ΣΥΝΤΗΡΗΣΗΣ ΥΠΗΡΕΣΙΟΚΕΝΤΡΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

Η
ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Πληροφορικής
Εξεταστική Επιτροπή

από την

Κουρή Ευφροσύνη

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΟ ΛΟΓΙΣΜΙΚΟ

Ιούνιος 2009

ΑΦΙΕΡΩΣΗ

Στην αγαπημένη μου γιαγιά Ασημίνα και στον Αργύρη.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ. Απόστολο Ζάρρα για τη βοήθεια, την υπομονή του και τις πολύτιμες συμβουλές του κατά τη διάρκεια εκπόνησης της μεταπτυχιακής μου εργασίας.

Επίσης, θα ήθελα να ευχαριστήσω και τα υπόλοιπα μέλη της εξεταστικής μου επιτροπής, κ. Παναγιώτη Βασιλειάδη και κα. Ευαγγελία Πιτουρά, για τις σημαντικές υποδείξεις τους.

Ευχαριστώ τους ιδιοκτήτες της εταιρείας Comitech Α.Ε., κ. Κωνσταντίνο Γλάρο, κ. Χριστόφορο Μπάφα, κ. Κωνσταντίνο Παπανικολάου και κ. Πέτρο Παργανά για την κατανόηση τους σε όλα τα χρόνια των σπουδών μου κατά τα οποία εργαζόμουν εκεί. Επιπλέον, θα ήθελα να ευχαριστήσω τους φίλους και συνεργάτες μου Άγγελο Φιλίππου, Χρήστο Μαυρογιώργο και Ιωάννη Παπασταύρου για την ηθική τους υποστήριξη και συμπαράσταση κατά την περίοδο ολοκλήρωσης και συγγραφής της μεταπτυχιακής μου εργασίας.

Επιπλέον, θα ήθελα να ευχαριστήσω τους γονείς μου, Χάρη και Μαρία, και τις αδερφές μου, Ασημίνα και Αμαρυλλίς-Ιωάννα, αλλά και τη γιαγιά μου, Ασημίνα για την απλόχερη αγάπη τους και υποστήριξη κατά τη διάρκεια των σπουδών μου.

Τέλος, θα ήθελα να ευχαριστήσω θερμά τον φίλο και συνεργάτη μου, Αργύρη Γρίβα, για τις πολύτιμες συμβουλές του, τη συμπαράσταση, την υπομονή και επιμονή του αλλά και σε όλους τους φίλους μου που στάθηκαν δίπλα μου παρ' όλες τις δυσκολίες.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

ΠΕΡΙΛΗΨΗ

Ευφροσύνη Κουρή του Χαράλαμπου και της Μαρίας. MSc, Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Ιούνιος, 2009. Πειραματική μελέτη σύγκρισης υπηρεσιών διαδικτύου στα πλαίσια της συντήρησης υπηρεσιοκεντρικών εφαρμογών. Επιβλέποντας: Απόστολος Ζάρρας.

Στα πλαίσια της συντήρησης υπηρεσιοκεντρικών εφαρμογών, προκύπτει συχνά η ανάγκη αντικατάστασης μιας υπηρεσίας διαδικτύου με κάποια άλλη που είτε είναι ανανεωμένη έκδοσή της είτε έχει προκύψει από αυτήν μετά από μια σειρά τροποποιήσεων. Οι διαφορές μεταξύ της αρχικής υπηρεσίας και της νέας που θα πάρει τη θέση της, συνεπάγονται αλλαγές στον κώδικα μιας εφαρμογής που χρησιμοποιούσε την αρχική υπηρεσία. Στην παρούσα εργασία, θα μελετηθούν οι αλλαγές που συνήθως συμβαίνουν στις WSDL περιγραφές στα πλαίσια της συντήρησης και επέκτασής τους. Επιπλέον, θα μελετηθούν οι επιπτώσεις που έχουν οι αλλαγές αυτές στο επίπεδο του κώδικα των εφαρμογών που τις χρησιμοποιούν. Τέλος, προτείνεται μια μέθοδος ανίχνευσης αλλαγών και παραγωγής σεναρίων τροποποίησης για υπηρεσιοκεντρικές εφαρμογές. Σκοπός των σεναρίων τροποποίησης είναι να καθοδηγήσουν τον προγραμματιστή ως προς τις αλλαγές που πρέπει να πραγματοποιήσει στον κώδικα της εφαρμογής του.

EXTENDED ABSTRACT IN ENGLISH

Kouri, Efrosini, Initials. MSc, Computer Science Department, University of Ioannina, Greece. June, 2009. Experimental Study of Change Detection Techniques for the Maintenance of Service Oriented Applications. Thesis Supervisor: Zarras Apostolis.

Over the last few years, there has been an increasing popularity of Web Services used in web-based applications. The maintenance of these applications might include the replacement of the currently used Web Service with another Web Service which offers similar functionality. The new Web Service is either a new version of the initial Web Service or a Web Service derived from the initial after a sequence of changes made in order to meet the specific needs of the vendor.

The goal of this Thesis is to assist the programmer to maintain the existing source code with the minimum possible effort. Specifically, given two Web Services, an existing and a new one, we want to provide the programmer with an edit-script that contains a sequence of changes that transform the initial Web Service to the new one. This edit-script can be used to guide the programmer in order to achieve compatibility of his client application with the new Web Service.

In our approach, we study the changes that are usually made during the maintenance and extension of WSDL descriptions. Moreover, we study the impact of these changes in the source code of applications using them. Finally, we propose a method which detects the changes between the initial and the new WSDL descriptions and produces an edit-script. The edit-script contains a sequence of basic edit operations, such as insert, delete, move and update. We assign a cost to each operation, which represents the effort of the programmer when this operation is applied to the source code. The sum of the costs of every operation in an edit-script, is the total cost of the script.

Our proposed method is organized as a sequence of steps. First, given that a WSDL description is hierarchically structured, we model each input and output message of every operation in the two Web Services in a tree. We use known tree edit distance algorithms to detect changes between the operations of the two Web Services. These algorithms produce edit-scripts and calculate the total cost for every pair of operations in the two Web Services. In our implementation, we have used two different tree edit distance algorithms, MM-DIFF and MH-DIFF.

In the second step, we use the total costs calculated in the first step to map each operation of the initial to one operation of the new Web Service. The mapping is done with regard to the minimum cost and is solved with a bipartite weighted matching algorithm. The output of this step is a set of edit-scripts for each pair of the mapped operations.

We experimentally assess our method over a real set of Web Services. We have selected a subset of ten Web Services which differ in the number of operations they offer as well as the complexity of the structure of the messages they exchange. For each of them, we have generated approximately 200 variations. Each variation has been derived from the initial set of services after a sequence of changes has been applied. We call this sequence optimal.
Our findings indicate that the mapping of the operations calculated in the second step of our method has 100% success, which means that each operation of the variation is mapped to the correspondent operation of the initial Web Service. Moreover, the results produced by using the MH-DIFF algorithm in the first step are much closer to the optimal script in comparison to the usage of the MM-DIFF algorithm instead. On the other hand, the execution time for MH-DIFF was 6 times higher than the one of MM-DIFF.

ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ

1.1. Στόχοι

Η αυξανόμενη χρήση των υπηρεσιών διαδικτύου στην ανάπτυξη εφαρμογών κάνει ολοένα και πιο δύσκολο το πρόβλημα της συντήρησής τους. Κάποιες εταιρείες που υλοποιούν υπηρεσίες διαδικτύου (για παράδειγμα Google, Hotbot, Altavista) μπορεί να παρέχουν υπηρεσίες με παρόμοια λειτουργικότητα. Οι υπηρεσίες αυτές μπορεί να είναι αυτόνομες, δηλαδή η κάθε εταιρεία να τις αναπτύσσει ανεξάρτητα, ή να είναι παραγόμενες, δηλαδή να προέρχονται από την επέκταση υπηρεσιών που έχουν οριστεί από άλλες εταιρείες. Στα πλαίσια της συντήρησης υπηρεσιοκεντρικών εφαρμογών, προκύπτει συχνά η ανάγκη αντικατάστασης μιας υπηρεσίας διαδικτύου με μια τροποποιημένη έκδοσή της. Οι διαφορές μεταξύ της αρχικής υπηρεσίας και της νέας που θα πάρει τη θέση της, συνεπάγονται αλλαγές στον κώδικα της εφαρμογής. Στην παρούσα διατριβή, προτείνεται μια μέθοδος ανίχνευσης αλλαγών και παραγωγής σεναρίων τροποποίησης για υπηρεσιοκεντρικές εφαρμογές.

Σχετικές εργασίες έχουν μελετήσει το πρόβλημα της αναζήτησης υπηρεσιών σε σημασιολογικό επίπεδο, συγκρίνοντας τους όρους που χρησιμοποιούνται για την περιγραφή των υπηρεσιών και των λειτουργιών που παρέχονται από τις υπηρεσίες διαδικτύου [9]. Η σύγκριση σε επίπεδο όρων δεν δίνει τις απαραίτητες πληροφορίες σχετικά με το κόστος της συντήρησης της εφαρμογής και το είδος των αλλαγών που πρέπει να πραγματοποιηθούν στην εφαρμογή στα πλαίσια της συντήρησης.

Στην παρούσα διατριβή, μελετάμε τη δομή των υπηρεσιών διαδικτύου και συγκεκριμένα τη δομή των τύπων δεδομένων που ανταλλάσσονται με μηνύματα κατά

την εκτέλεση μιας λειτουργίας. Αναγνωρίζονται οι βασικοί τύποι των αλλαγών που συμβαίνουν στις δομές αυτές και εκτιμάται το προγραμματιστικό κόστος που συνεπάγονται οι αλλαγές αυτές για τις εφαρμογές που τις χρησιμοποιούν. Με βάση τα αποτελέσματα της μελέτης αυτής, καθορίζεται ένα σύνολο επιθυμητών χαρακτηριστικών για μια μέθοδο ανίχνευσης αλλαγών σε υπηρεσίες διαδικτύου. Εφόσον η δομή των μηνυμάτων είναι δενδρική, μελετάμε αλγορίθμους ανίχνευσης αλλαγών σε δένδρα και εξετάζουμε εάν μπορούν να εφαρμοστούν στο πρόβλημα της συντήρησης που θέτουμε. Λαμβάνοντας υπόψη τις πληροφορίες που παρέχονται από τους αλγόριθμους ανίχνευσης αλλαγών, προτείνουμε μια μέθοδο ανίχνευσης αλλαγών μεταξύ δύο υπηρεσιών διαδικτύου.

1.2. Δομή της Διατριβής

Η διατριβή περιέχει 4 κεφάλαια: Στο Κεφάλαιο 2, περιγράφεται η δομή των υπηρεσιών διαδικτύου και στο Κεφάλαιο 3 γίνεται μελέτη των αλγορίθμων ανίχνευσης αλλαγών σε δένδρα. Στο Κεφάλαιο 4 γίνεται μελέτη των αλλαγών που συνήθως συμβαίνουν σε WSDL περιγραφές στα πλαίσια της συντήρησης καθώς και των επιπτώσεων που έχουν οι αλλαγές αυτές στον κώδικα των εφαρμογών που τις χρησιμοποιούν. Επιπλέον, περιγράφεται αναλυτικά η προτεινόμενη μέθοδος. Στο Κεφάλαιο 5 παρουσιάζονται τα πειραματικά αποτελέσματα. Τέλος, στο Κεφάλαιο 6 γίνεται ανακεφαλαίωση της μεθόδου και προτείνονται μελλοντικές επεκτάσεις.

ΚΕΦΑΛΑΙΟ 2. ΥΠΟΒΑΘΡΟ

2.1. Γενικά

Τα τελευταία χρόνια, η χρήση του παγκόσμιου ιστού (Internet) έχει αυξηθεί δραματικά. Ολοένα και περισσότερες επιχειρήσεις παγκοσμίως, χρησιμοποιούν το διαδίκτυο για να προβληθούν και να δημοσιοποιήσουν τις δραστηριότητές τους. Με την εξέλιξη της τεχνολογίας, τους δόθηκε η δυνατότητα να παρέχουν υπηρεσίες μέσω του διαδικτύου σε χρήστες αλλά και σε άλλες επιχειρήσεις. Κάθε επιχείρηση δημιουργούσε τις υπηρεσίες της με βάση το σύστημα που χρησιμοποιούσε και έτσι ήταν δύσκολο, χρονοβόρο και με μεγάλο κόστος για μια άλλη επιχείρηση να τις χρησιμοποιήσει ακόμα και αν παρείχαν ακριβώς την ίδια λειτουργικότητα. Δημιουργήθηκε, λοιπόν, η ανάγκη για έναν τρόπο ώστε ήδη υπάρχουσες εφαρμογές, να μπορούν να ξαναχρησιμοποιηθούν εύκολα από διαφορετικές επιχειρήσεις.

Τη λύση στο πρόβλημα αυτό, ήρθαν να δώσουν οι υπηρεσίες διαδικτύου (Web Services). Οι υπηρεσίες διαδικτύου είναι αυτόνομες εφαρμογές που παρέχουν

διεπαφές μέσω του Ιστού. Η αρχιτεκτονική τους, που έχει θεσπιστεί από το World Wide Web Consortium (www.w3c.org), ορίζει ένα σύνολο από προδιαγραφές και κανόνες που πρέπει να ακολουθούνται για τη δημιουργία και παροχή τους μέσα από το διαδίκτυο. Η επικοινωνία με τις υπηρεσίες διαδικτύου γίνεται με τη χρήση ανοιχτών πρωτοκόλλων (HTTP – το πιο διαδεδομένο πρωτόκολλο διαδικτύου) και η XML χρησιμοποιείται σαν βάση για την περιγραφή τους. Η XML είναι μια γλώσσα που μπορεί να περιγράψει πολύπλοκα μηνύματα και συναρτήσεις και παράλληλα μπορεί να χρησιμοποιηθεί από οποιοδήποτε λειτουργικό σύστημα, πλατφόρμα και προγραμματιστική γλώσσα με ευκολία επειδή αποτελείται από κείμενο (text-based). Οι υπηρεσίες διαδικτύου μπορούν να δημοσιευτούν σε έναν κατάλογο UDDI (Universal Description, Discovery and Integration) μέσω του οποίου μπορεί κανείς να βρει την/τις υπηρεσίες που καλύπτουν τις ανάγκες του.

2.2. Μοντέλο χρήσης των Web Services

Το μοντέλο χρήσης μιας υπηρεσίας διαδικτύου ακολουθεί την παρακάτω διαδικασία:

- Διαχωρισμός των μηνυμάτων σε πακέτα XML.
- Μεταφορά των πακέτων από τον πελάτη (client) στον εξυπηρετητή (server) πάνω από ένα πρωτόκολλο διαδικτύου.
- Περιγραφή της διεπαφής που παρέχει μια υπηρεσία.
- Δημοσίευση της διεπαφής στο Internet.

Το πιο συνηθισμένο μοντέλο χρησιμοποιεί το πρωτόκολλο HTTP για την μεταφορά των πακέτων, αλλά μπορούν να χρησιμοποιηθούν και άλλα πρωτόκολλα όπως π.χ. FTP, SMTP κλπ. Το πακετάρισμα των δεδομένων γίνεται συνήθως με το πρωτόκολλο SOAP (Simple Object Access Protocol) ενώ η περιγραφή της διεπαφής γίνεται με WSDL (Web Service Description Language). Τέλος, χρησιμοποιείται το UDDI για την καταχώρηση των υπηρεσιών στο διαδίκτυο.

2.3. WSDL

Η WSDL [21] είναι μια γλώσσα περιγραφής για υπηρεσίες διαδικτύου σε μορφή XML. Ένα έγγραφο WSDL αποτελείται από 2 τμήματα: αφηρημένο (abstract) και συγκεκριμένο (concrete). Τα μηνύματα (**messages**) είναι αφηρημένες περιγραφές των

δεδομένων που ανταλλάσσονται ενώ οι επιμέρους διεπαφές (**port types**) είναι αφηρημένες συλλογές των λειτουργιών (**operations**) που προσφέρονται από την υπηρεσία διαδικτύου. Η συσχέτιση των επιμέρους διεπαφών με κάποια συγκεκριμένα πρωτόκολλα επικοινωνίας αποτελούν μια επαναχρησιμοποιήσιμη σύνδεση (**binding**). Το στιγμιότυπο μιας διεπαφής (**port**) καθορίζεται με τη συσχέτιση μιας διεύθυνσης δικτύου (URI) και ενός binding. Η συλλογή των ports καθορίζει μια υπηρεσία (**service**).

Ως εκ τούτου, ένα έγγραφο WSDL χρησιμοποιεί τα ακόλουθα στοιχεία για να καθορίσει τις υπηρεσίες διαδικτύου:

Αφηρημένο τμήμα

- Τύποι δεδομένων (types): Περιγραφή των τύπων δεδομένων που αποστέλλονται σε μια αίτηση προς την υπηρεσία διαδικτύου. Η περιγραφή γίνεται χρησιμοποιώντας κάποιο σύστημα τύπων, όπως XSD.
- Μήνυμα (message): Η δομή των μηνυμάτων που αποστέλλονται προς την υπηρεσία διαδικτύου.
- Λειτουργία (operation): Η αφηρημένη περιγραφή μιας λειτουργίας που υποστηρίζεται από την υπηρεσία διαδικτύου.
- Διεπαφή (port type): Το σύνολο των λειτουργιών που υποστηρίζονται από ένα ή περισσότερα ports.

Συγκεκριμένο τμήμα

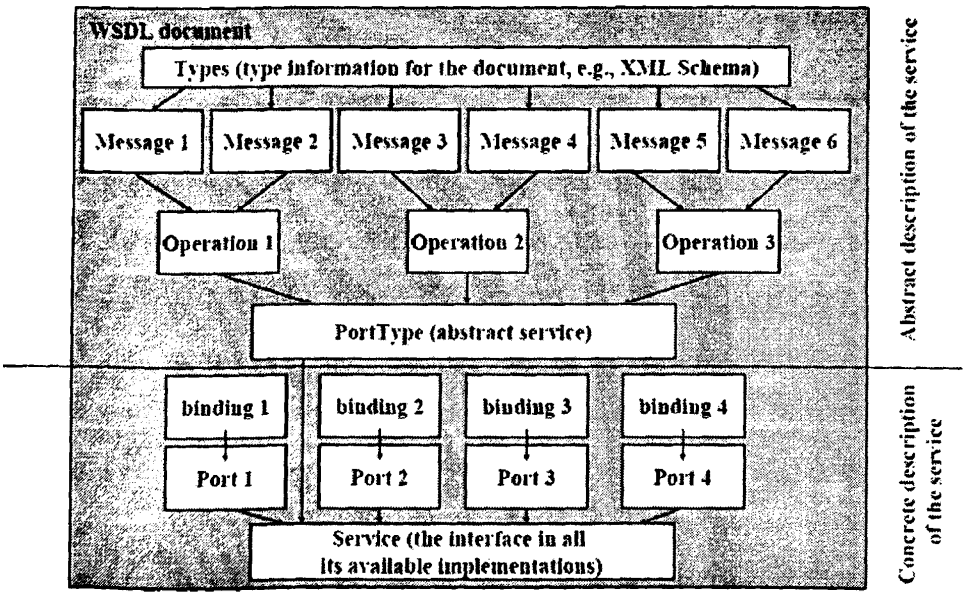
- Σύνδεση (binding): Συσχετίζει ένα συγκεκριμένο πρωτόκολλο επικοινωνίας και ένα συγκεκριμένο τύπο μηνυμάτων με μια συγκεκριμένη διεπαφή (port type).
- Στιγμιότυπο διεπαφής (port): Συσχετίζει μια σύνδεση και μια διεύθυνση δικτύου.
- Υπηρεσία (service): Συλλογές από συσχετισμένα στιγμιότυπα διεπαφής.

Η δομή μιας WSDL περιγραφής φαίνεται στο Σχήμα 2.1. Στο αφηρημένο τμήμα ορίζεται μια διεπαφή. Κάθε διεπαφή μπορεί να παρέχει μια ή περισσότερες λειτουργίες οι οποίες ανταλλάσουν μηνύματα που έχουν μια συγκεκριμένη δομή. Η υπηρεσία διαδικτύου μπορεί να παρέχει έναν ή περισσότερους τρόπους επικοινωνίας με κάθε διεπαφή. Οι τρόποι επικοινωνίας ορίζονται στο συγκεκριμένο τμήμα. Για παράδειγμα, έστω ότι θέλουμε να εκτελέσουμε τη λειτουργία Operation1.

Γνωρίζουμε ότι η λειτουργία αυτή λαμβάνει σαν είσοδο μηνύματα με τη μορφή που περιγράφεται στο Message1 και αποστέλλει σαν έξοδο μηνύματα με τη μορφή που περιγράφεται στο Message2. Η δομή των μηνυμάτων Message1 και Message2 περιγράφεται στο τμήμα Types. Την συγκεκριμένη λειτουργία, μπορούμε να την προσπελάσουμε χρησιμοποιώντας έναν από τους τέσσερεις συνδυασμούς binding και port. Το port παρέχει τη διεύθυνση διαδικτύου μέσω της οποίας μπορούμε να προσπελάσουμε τη λειτουργία ενώ το binding καθορίζει τα πρωτόκολλα επικοινωνίας που πρέπει να χρησιμοποιηθούν (π.χ. SOAP και HTTP, SOAP και JMS κλπ).

Η γενική μορφή ενός WSDL αρχείου παρουσιάζεται στον Πίνακα 2.1. Αρχικά, ορίζονται στο τμήμα types οι τύποι δεδομένων που χρησιμοποιούνται από τα μηνύματα. Στη συνέχεια, περιγράφεται η δομή κάθε μηνύματος σε ένα τμήμα message. Οι διεπαφές και οι λειτουργίες που παρέχει κάθε διεπαφή ορίζονται σε ένα τμήμα portType. Κάθε διαθέσιμη σύνδεση καθορίζεται σε ένα τμήμα binding ενώ η περιγραφή της υπηρεσίας περιγράφεται στο τμήμα service.

Στις επόμενες ενότητες, περιγράφουμε αναλυτικά το κάθε τμήμα.



Σχήμα 2.1 Δομή μιας WSDL Περιγραφής

Πίνακας 2.1 Γενική μορφή ενός WSDL αρχείου

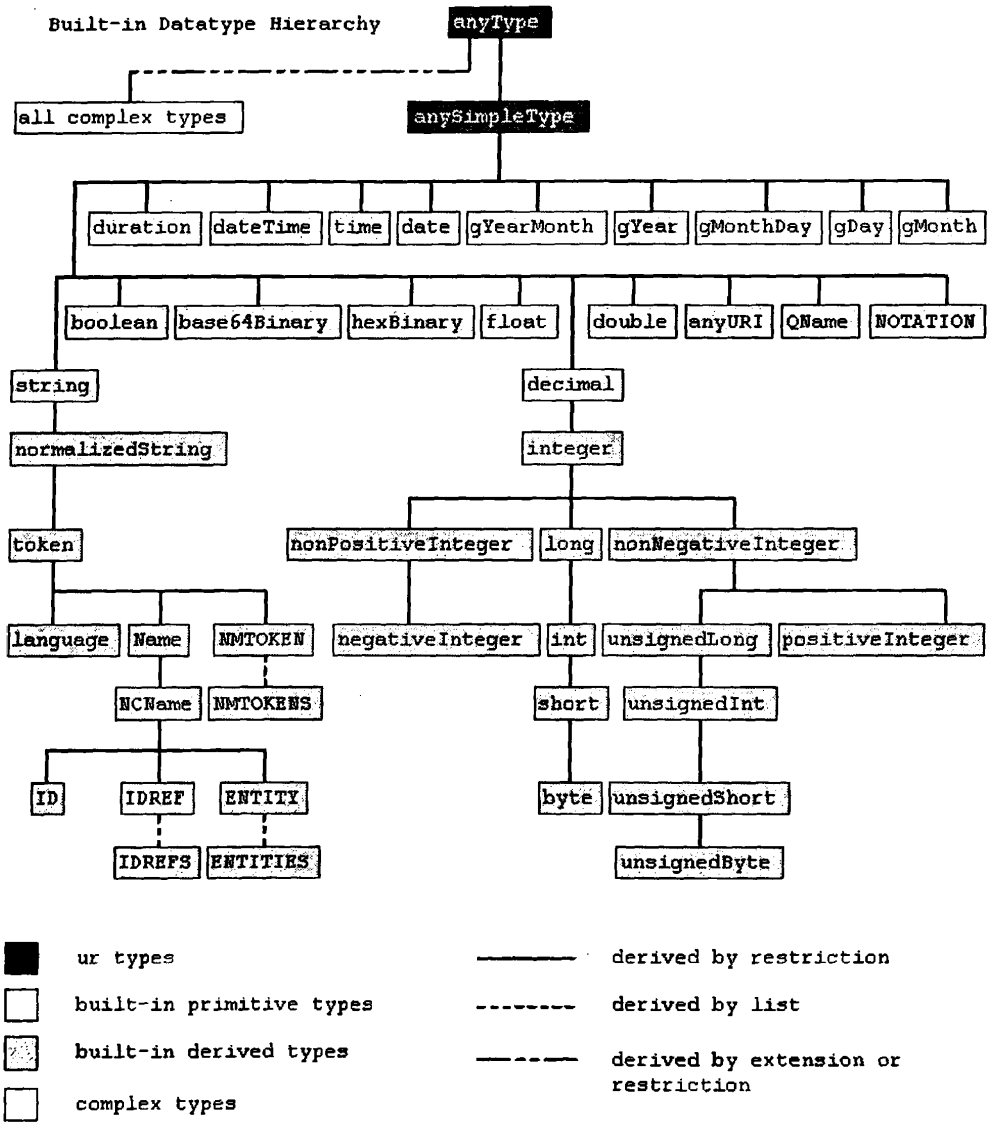
```
<definitions>
<types>
        Definition of types.....
</types>
<message>
        definition of a message....
</message>
<portType>
        definition of a port.....
</portType>
<binding>
        definition of a binding....
</binding>
<service>
        definition of a service....
</service>
</definitions>
```

2.3.1. Τύποι Δεδομένων (types)

Οι τύποι δεδομένων που περιγράφονται σε ένα WSDL αρχείο καθορίζουν την δομή του περιεχομένου των μηνυμάτων που ανταλλάσσονται με μια υπηρεσία διαδικτύου. Για μέγιστη διαλειτουργικότητα και ανεξαρτησία από την πλατφόρμα, προτιμάται η χρήση XSD ως κανονικοποιημένο σύστημα τύπων. Στο Σχήμα 2.2 φαίνονται οι προκαθορισμένοι τύποι δεδομένων που υπάρχουν στο XML Schema [22], αλλά σε ένα WSDL αρχείο μπορούν να επεκταθούν ώστε να υποστηρίζουν πιο πολύπλοκους τύπους δεδομένων.

Οι προκαθορισμένοι τύποι δεδομένων χωρίζονται σε απλούς (π.χ. string, decimal, datetime, double κλπ) και παραγόμενους (integer, long, normalizedString κλπ). Οι παραγόμενοι τύποι δεδομένων προκύπτουν θέτοντας περιορισμούς ή επεκτείνοντας έναν απλό ή ένα παραγόμενο τύπο. Για παράδειγμα, ο τύπος integer προκύπτει από τον απλό τύπο decimal, αφαιρώντας τα δεκαδικά ψηφία. Αντίστοιχα, ο τύπος NonNegativeInteger προκύπτει από τον τύπο integer περιορίζοντας τους αρνητικούς ακέραιους αριθμούς. Οι προκαθορισμένοι τύποι δεδομένων μπορούν να

χρησιμοποιηθούν σε πολύπλοκους τύπους (complex types) για να υποστηρίξουν πιο πολύπλοκες δομές. Η ιεραρχία αυτή παρουσιάζεται αναλυτικά στο Σχήμα 2.2.



Σχήμα 2.2 Προκαθορισμένοι Τύποι Δεδομένων

Μια περιγραφή τύπων δεδομένων ξεκινάει με το στοιχείο <schema>. Στη συνέχεια, μπορεί να ακολουθήσει μια σειρά από άλλα στοιχεία. Για παράδειγμα, για να ορίσουμε έναν πολύπλοκο τύπο δεδομένων *Address*, ο οποίος αποτελείται από τρία στοιχεία κειμένου (string): *country*, *city*, *street* και έναν ακέραιο αριθμό *streetnumber*, ακολουθούμε τη δομή που παρουσιάζεται στον Πίνακα 2.2.

Πίνακας 2.2 Παράδειγμα δομής για τον ορισμό ενός πολύπλοκου τύπου δεδομένων

```
<xs:element name="Address">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="country" type="xs:string"/>
   <xs:element name="city" type="xs:string"/>
   <xs:element name="street" type="xs:string"/>
   <xs:element name="streetnumber" type="xs:integer"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Τα πιο βασικά στοιχεία που χρησιμοποιούνται σε ένα XML schema παρουσιάζονται παρακάτω. Μια πλήρης λίστα των στοιχείων υπάρχει στο [23].

element: Καθορίζει ένα στοιχείο. Μπορεί να περικλείεται σε στοιχεία τύπου: schema, choice, all, sequence, group και μπορεί να περιέχει στοιχεία τύπου simpleType ή complexType κ.α. Αναλυτικά, η σύνταξή του παρουσιάζεται στον Πίνακα 2.3.

Πίνακας 2.3 Σύνταξη του τύπου δεδομένων element

```
<element id=ID name=NCName ref=QName type=QName
substitutionGroup=QName default=string fixed=string
form=qualified|unqualified maxOccurs=nonNegativeInteger|unbounded
minOccurs=nonNegativeInteger nillable=true|false
abstract=true|false block=(#all|list of (extension|restriction))
final=(#all|list of (extension|restriction)) any attributes>
annotation?,((simpleType|complexType)?,(unique|key|keyref)*))
</element>
```

simpleType: Χρησιμοποιεί έναν από τους προκαθορισμένους τύπους δεδομένων και θέτει περιορισμούς στο πεδίο τιμών του προκαθορισμένου τύπου. Για παράδειγμα, μπορεί να χρησιμοποιεί τον προκαθορισμένο τύπο κειμένου (string) με πεδίο τιμών: male, female. Η σύνταξή του παρουσιάζεται στον Πίνακα 2.4.

Πίνακας 2.4 Σύνταξη του τύπου δεδομένων simpleType

<simpleType id=ID name=NCName any attributes> *(annotation?,(restriction|list|union))* *</simpleType>*

complexType: Καθορίζει έναν πολύπλοκο τύπο δεδομένων. Μπορεί να χαρακτηρίζεται από ένα όνομα (name), ένα ID καθώς και από άλλα γνωρίσματα και μπορεί να περιέχει στοιχεία τύπου simpleContent ή complexContent κλπ. Η σύνταξή του παρουσιάζεται στον Πίνακα 2.5.

Πίνακας 2.5 Σύνταξη του τύπου δεδομένων complexType

<complexType id=ID name=NCName abstract=true|false *mixed=true|false block=(#all|list of (extension|restriction))* *final=(#all|list of (extension|restriction)) any attributes>* *(annotation?,(simpleContent|complexContent|((group|all|* *choice|sequence)?,((attribute|attributeGroup)*,anyAttribute?))))* *</complexType>*

simpleContent: Θέτει περιορισμούς ή επεκτάσεις πάνω σε έναν πολύπλοκο τύπο δεδομένων (complexType) που περιέχει μόνο στοιχεία κειμένου ή σε κάποιον από τους προκαθορισμένους τύπους δεδομένων. Η σύνταξή του παρουσιάζεται στον Πίνακα 2.6.

Πίνακας 2.6 Σύνταξη του τύπου δεδομένων simpleContent

<simpleContent id=ID any attributes> *(annotation?,(restriction|extension))* *</simpleContent>*

complexContent: Θέτει περιορισμούς ή επεκτάσεις πάνω σε έναν πολύπλοκο τύπο δεδομένων που περιέχει στοιχεία οποιουδήποτε τύπου (όχι μόνο στοιχεία κειμένου). Η σύνταξή του παρουσιάζεται στον Πίνακα 2.7.

Πίνακας 2.7 Σύνταξη του τύπου δεδομένων complexContent

<complexContent id=ID mixed=true|false any attributes> *(annotation?,(restriction|extension))* *</complexContent>*

sequence: Καθορίζει ότι τα στοιχεία που περιέχονται σε ένα στοιχείο sequence, πρέπει να εμφανίζονται σε μια ακολουθία. Κάθε στοιχείο μπορεί να εμφανίζεται μηδέν ή περισσότερες φορές. Η σύνταξή του παρουσιάζεται στον Πίνακα 2.8.

Πίνακας 2.8 Σύνταξη του τύπου δεδομένων sequence

<sequence id=ID maxOccurs=nonNegativeInteger|unbounded *minOccurs=nonNegativeInteger any attributes>* *(annotation?,(element|group|choice|sequence|any)*)* *</sequence>*

2.3.2. Μηνύματα (messages)

Τα μηνύματα καθορίζουν τη μορφή των δεδομένων που ανταλλάσσονται κατά την εκτέλεση μιας λειτουργίας. Κάθε μήνυμα μπορεί να αποτελείται από ένα ή περισσότερα τμήματα (parts). Σε σύγκριση με μια παραδοσιακή γλώσσα προγραμματισμού, τα τμήματα θα αποτελούσαν τις παραμέτρους μιας συνάρτησης. Κάθε μήνυμα χαρακτηρίζεται από ένα μοναδικό όνομα (name) ώστε να διαφοροποιείται από όλα τα μηνύματα στο συγκεκριμένο αρχείο. Κάθε τμήμα χαρακτηρίζεται από ένα όνομα μοναδικό μεταξύ των υπόλοιπων τμημάτων του μηνύματος στο οποίο ανήκει. Επιπλέον, κάθε μήνυμα συσχετίζεται με έναν τύπο δεδομένων ο οποίος είτε έχει οριστεί στο συγκεκριμένο αρχείο είτε αποτελεί βασικό τύπο δεδομένων. Η συσχέτιση αυτή γίνεται χρησιμοποιώντας τα γνωρίσματα element ή type. Η σύνταξη για τα μηνύματα παρουσιάζεται στον Πίνακα 2.9.

Πίνακας 2.9 Σύνταξη για την περιγραφή ενός μηνύματος

```
<definitions .... >
  <message name="nmtoken"> *
    <part name="nmtoken" element="qname"? type="qname"?/> *
  </message>
</definitions>
```

2.3.3. Λειτουργίες (Operations)

Μια λειτουργία είναι ένα σύνολο από στιγμιότυπα μηνυμάτων. Χαρακτηρίζεται από ένα όνομα και μπορεί να περιέχει το πολύ ένα στιγμιότυπο εισόδου (input), το πολύ ένα στιγμιότυπο εξόδου (output) και το πολύ ένα στιγμιότυπο μηνύματος λάθους (fault). Κάθε στιγμιότυπο χαρακτηρίζεται από ένα όνομα (name) και έναν τύπο μηνύματος (message) που πρέπει να αντιστοιχεί σε ένα από τα μηνύματα που είχαν οριστεί στο τμήμα μηνυμάτων. Υπάρχουν τέσσερα διαφορετικά στυλ λειτουργιών:

- One-way operation: η υπηρεσία δέχεται ένα μήνυμα εισόδου. Η γραμματική για αυτό το στυλ είναι: *<wsdl:operation name="nmtoken"><wsdl:input name="nmtoken"? message="qname"/></wsdl:operation>*
- Request-response operation: η υπηρεσία δέχεται ένα μήνυμα εισόδου και επιστρέφει ένα μήνυμα εξόδου. Τα προαιρετικά μηνύματα λάθους καθορίζουν τη μορφή που θα έχουν τα μηνύματα αν κατά την εκτέλεση της λειτουργίας προκύψει κάποιο σφάλμα. Η γραμματική είναι: *<wsdl:operation name="nmtoken" parameterOrder="nmtokens"> <wsdl:input name="nmtoken"? message="qname"/> <wsdl:output name="nmtoken"? message="qname"/> <wsdl:fault name="nmtoken" message="qname"/>* *</wsdl:operation>*
- Solicit-response operation: η υπηρεσία στέλνει ένα μήνυμα εξόδου και στη συνέχεια δέχεται ένα μήνυμα εισόδου. Η γραμματική είναι: *<wsdl:operation name="nmtoken" parameterOrder="nmtokens"> <wsdl:output name="nmtoken"? message="qname"/> <wsdl:input name="nmtoken"? message="qname"/> <wsdl:fault name="nmtoken" message="qname"/>* *</wsdl:operation>*

- Notification operation: η υπηρεσία στέλνει ένα μήνυμα εξόδου. Η γραμματική είναι: *<wsdl:operation name="nmtoken"> <wsdl:output name="nmtoken"? message="qname"/> </wsdl:operation>*

2.3.4. Διεπαφές (port types)

Οι διεπαφές είναι ένα σύνολο από λειτουργίες και από τα μηνύματα που εμπλέκονται σε αυτές. Χαρακτηρίζονται από ένα μοναδικό όνομα στο αρχείο WSDL. Η σύνταξη μιας διεπαφής παρουσιάζεται στον Πίνακα 2.10.

Πίνακας 2.10 Σύνταξη για την περιγραφή μιας διεπαφής

```
<wsdl:definitions .... >
  <wsdl:portType name="nmtoken">
    <wsdl:operation name="nmtoken" .... /> *
  </wsdl:portType>
</wsdl:definitions>
```

2.3.5. Συνδέσεις (Bindings)

Μια σύνδεση καθορίζει τη μορφή των μηνυμάτων και το πρωτόκολλο επικοινωνίας για τις λειτουργίες και τα μηνύματα μιας συγκεκριμένης διεπαφής. Για κάθε διεπαφή μπορεί να υπάρχουν μια ή περισσότερες συνδέσεις. Η σύνταξη για την περιγραφή μιας σύνδεσης παρουσιάζεται στον Πίνακα 2.11.

Το γνώρισμα name είναι μοναδικό μεταξύ των συνδέσεων του WSDL αρχείου. Το γνώρισμα type χαρακτηρίζει τη διεπαφή στην οποία αναφέρεται η σύνδεση. Για κάθε λειτουργία, δίνονται πληροφορίες σχετικές με το πακετάρισμα των μηνυμάτων για τη συγκεκριμένη σύνδεση.

Πίνακας 2.11 Σύνταξη για την περιγραφή μιας σύνδεσης

```
<wsdl:definitions .... >
  <wsdl:binding name="nmtoken" type="qname"> *
    <-- extensibility element (1) --> *
    <wsdl:operation name="nmtoken"> *
      <-- extensibility element (2) --> *
      <wsdl:input name="nmtoken"? > ?
        <-- extensibility element (3) -->
      </wsdl:input>
      <wsdl:output name="nmtoken"? > ?
        <-- extensibility element (4) --> *
      </wsdl:output>
      <wsdl:fault name="nmtoken"> *
        <-- extensibility element (5) --> *
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>
</wsdl:definitions>
```

2.3.6. Στιγμιότυπα διεπαφής (ports)

Ένα στιγμιότυπο διεπαφής καθορίζει μια συγκεκριμένη διεύθυνση δικτύου για μια συγκεκριμένη σύνδεση. Η σύνταξη του παρουσιάζεται στον Πίνακα 2.12.

Πίνακας 2.12 Σύνταξη για την περιγραφή ενός στιγμιότυπου διεπαφής

```
<wsdl:definitions .... >
  <wsdl:service .... > *
    <wsdl:port name="nmtoken" binding="qname"> *
      <-- extensibility element (1) -->
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Το γνώρισμα name είναι μοναδικό μεταξύ των στιγμιότυπων διεπαφής του WSDL αρχείου. Το γνώρισμα binding συσχετίζει τη συγκεκριμένη διεύθυνση δικτύου με μια συγκεκριμένη σύνδεση. Στο extensibility element (1) καθορίζεται η διεύθυνση δικτύου (URI) στην οποία βρίσκεται το στιγμιότυπο.

2.3.7. Υπηρεσίες (services)

Μια υπηρεσία είναι ένα σύνολο από στιγμιότυπα διεπαφών. Το γνώρισμα name είναι μοναδικό μεταξύ των υπηρεσιών του WSDL αρχείου. Τα στιγμιότυπα διεπαφής που ανήκουν σε μια συγκεκριμένη υπηρεσία δεν μπορούν να επικοινωνούν μεταξύ τους (δεν μπορεί η έξοδος ενός στιγμιότυπου διεπαφής να είναι είσοδος σε κάποιο άλλο στιγμιότυπο). Αν υπάρχουν περισσότερα από ένα στιγμιότυπα διεπαφής για μια συγκεκριμένη διεπαφή αλλά υλοποιούν διαφορετικές συνδέσεις, τότε λειτουργούν σαν εναλλακτικές λύσεις δίνοντας στον χρήστη τη δυνατότητα να επιλέξει με ποιο τρόπο θα επικοινωνήσει με την υπηρεσία (βάσει του πρωτόκολλου επικοινωνίας, της απόστασης κλπ). Η σύνταξη για την περιγραφή μιας υπηρεσίας παρουσιάζεται στον Πίνακα 2.13.

Πίνακας 2.13 Σύνταξη για την περιγραφή μιας υπηρεσίας

```
<wsdl:definitions .... >
  <wsdl:service name="nmtoken"> *
    <wsdl:port .... /> *
  </wsdl:service>
</wsdl:definitions>
```

ΚΕΦΑΛΑΙΟ 3. ΣΧΕΤΙΚΗ ΔΟΥΛΕΙΑ

3.1. Γενικά

Στην παρούσα διατριβή, προτείνεται μια μέθοδος ανίχνευσης αλλαγών μεταξύ δύο υπηρεσιών διαδικτύου. Σχετικές εργασίες έχουν μελετήσει το πρόβλημα της αναζήτησης παρόμοιων υπηρεσιών και το έχουν προσεγγίσει σε σημασιολογικό επίπεδο, συγκρίνοντας τους όρους που χρησιμοποιούνται για την περιγραφή των υπηρεσιών και των λειτουργιών που παρέχονται από τις υπηρεσίες διαδικτύου [9]. Η σύγκριση σε επίπεδο όρων δεν δίνει τις απαραίτητες πληροφορίες σχετικά με το κόστος της συντήρησης της εφαρμογής και το είδος των αλλαγών που πρέπει να πραγματοποιηθούν στην εφαρμογή στα πλαίσια της συντήρησης. Η προτεινόμενη μέθοδος ανιχνεύει αλλαγές στη δομή των τύπων δεδομένων της εισόδου και της εξόδου της κάθε λειτουργίας. Λόγω της ιεραρχικής δομής τους, τα μηνύματα εισόδου και εξόδου μπορούν να μοντελοποιηθούν σε δένδρα. Στις επόμενες ενότητες, γίνεται μια συνοπτική περιγραφή των αλγορίθμων ανίχνευσης αλλαγών σε δένδρα.

Ένα παρόμοιο πρόβλημα που έχει μελετηθεί εκτενώς, είναι το πρόβλημα της ανίχνευσης αλλαγών στον πηγαίο κώδικα. Κάποια συστήματα διατήρησης εκδόσεων του πηγαίου κώδικα (όπως CVS και Subversion) δίνουν πληροφορίες σχετικά με το ποιες γραμμές έχουν προστεθεί, διαγραφεί ή τροποποιηθεί μεταξύ δύο εκδόσεων. Παρόμοιες πληροφορίες δίνονται και από το εργαλείο GNU diff [13].

Στην εργασία των Maletic και Collard [8], ανιχνεύονται συντακτικές αλλαγές μεταξύ αρχείων πηγαίου κώδικα, χρησιμοποιώντας μια ενδιάμεση αναπαράσταση του πηγαίου κώδικα σε XML αρχεία. Στη συνέχεια, χρησιμοποιείται το εργαλείο GNU diff για να εντοπίσει τις τροποποιημένες οντότητες.

Στην εργασία του Horwitz [12], ανιχνεύονται συντακτικές και λεκτικές αλλαγές μεταξύ δύο προγραμμάτων. Κάθε πρόγραμμα διαχωρίζεται σε μέρη, με βάση τη λειτουργικότητα που προσφέρει. Στη συνέχεια, τα παρόμοια μέρη του κάθε προγράμματος συνδυάζονται μεταξύ τους ανάλογα με τη λειτουργικότητά τους και εξάγεται ένα σύνολο αλλαγών. Στην εργασία των Fluri et al. [], παρέχεται ένα πιο ολοκληρωμένο σύνολο από λειτουργίες τροποποίησης και οι αλλαγές κατηγοριοποιούνται σε τύπους αλλαγών.

Στην εργασία των Raghavan et al., υλοποιήθηκε το εργαλείο Dex [3], που ανιχνεύει αλλαγές μεταξύ δύο προγραμμάτων σε κώδικα γραμμένο στη γλώσσα C. Οι Xing και Stoulia [16], πρότειναν το εργαλείο UMLDiff που ανιχνεύει αλλαγές σε UML κλάσεις. Οι Sager et al. [4], χρησιμοποίησαν διάφορους αλγορίθμους ανίχνευσης αλλαγών σε δένδρα για να ανιχνεύσουν αλλαγές σε κλάσεις Java.

3.2. Αλγόριθμοι Ανίχνευσης Αλλαγών σε Δένδρα

Έστω *T* ένα δένδρο με ρίζα. Το *T* λέγεται επονομαζόμενο δένδρο (labeled tree) εάν σε κάθε κόμβο έχει ανατεθεί ένα σύμβολο από ένα καθορισμένο πεπερασμένο αλφάβητο. Το *T* λέγεται ταξινομημένο δένδρο (ordered tree) εάν υπάρχει μια ταξινόμηση από αριστερά προς τα δεξιά μεταξύ των παιδιών κάθε κόμβου (siblings) και αταξινόμητο (unordered) εάν δεν υπάρχει τέτοια ταξινόμηση. Ορίζουμε τρεις λειτουργίες τροποποίησης (edit operations) για επονομαζόμενα δένδρα:

- Update: Αλλαγή του ονόματος (label) ενός κόμβου *v* στο δένδρο *T*.
- Delete: Διαγραφή ενός κόμβου *v* που δεν είναι ρίζα στο *T* με πατρικό κόμβο *v'* και μεταφορά των παιδιών του *v* στον κόμβο *v'*.
- Insert: Εισαγωγή ενός κόμβου *v* σαν παιδί του *v'* στο *T*.

Έστω δύο δένδρα *T1* και *T2* καθώς και μια συνάρτηση κόστους (cost function) για κάθε λειτουργία τροποποίησης. Ένα σενάριο τροποποίησης *S* (edit script) μεταξύ των δένδρων *T1* και *T2* ορίζεται ως μια ακολουθία από λειτουργίες τροποποίησης που μετατρέπουν το δένδρο *T1* στο δένδρο *T2*. Το κόστος του *S* είναι το άθροισμα από τα κόστη της κάθε λειτουργίας τροποποίησης στο *S*. Ένα βέλτιστο σενάριο τροποποίησης (optimal edit script) μεταξύ του *T1* και του *T2* είναι ένα σενάριο τροποποίησης μεταξύ των *T1* και *T2* με ελάχιστο κόστος. Το κόστος αυτό ονομάζεται απόσταση τροποποίησης δένδρων (tree edit distance). Το πρόβλημα της απόστασης

τροποποίησης δένδρων συνεπάγεται τον υπολογισμό της απόστασης και την εύρεση ενός βέλτιστου σεναρίου τροποποίησης.

Οι αλγόριθμοι που επιλύουν το πρόβλημα της απόστασης τροποποίησης δένδρων εφαρμόζονται είτε σε ταξινομημένα είτε σε αταξινόμητα δένδρα. Το πρόβλημα για αταξινόμητα δένδρα είναι NP-complete πρόβλημα. Έχει δειχθεί όμως [20] ότι σε συγκεκριμένες περιπτώσεις, μπορούν να αναπτυχθούν αλγόριθμοι με πολυωνυμικό χρόνο.

3.2.1. Αλγόριθμοι για ταξινομημένα δένδρα

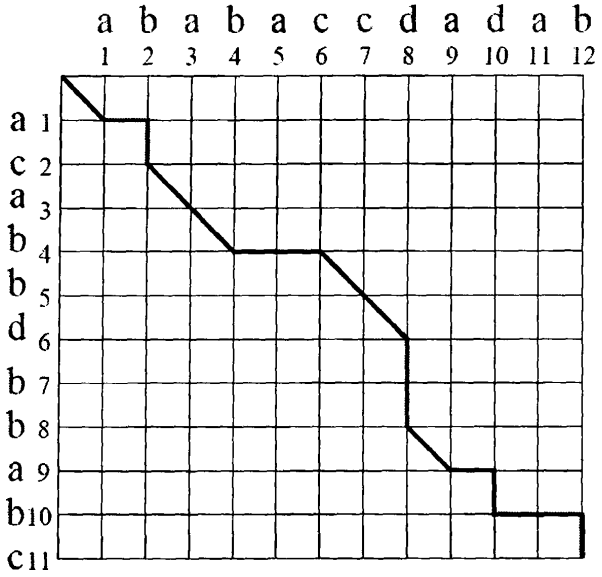
Το πρόβλημα της απόστασης τροποποίησης δένδρων για ταξινομημένα δένδρα αποτελεί μια γενίκευση του προβλήματος της απόστασης τροποποίησης χαρακτήρων [18]. Προτάθηκε το 1979 από τον Tai [17] και η χρονική και χωρική του πολυπλοκότητα ήταν $O(|T_1||T_2||L_1^2|L_2^2|)$. Στη συνέχεια, οι Zhang και Shasha [19] τροποποίησαν τον αλγόριθμο, βελτιώνοντας την χρονική πολυπλοκότητα σε $O(|T_1||T_2|\min(L_1,D_1)\min(L_2,D_2))$ και την χωρική πολυπλοκότητα σε $O(|T_1||T_2|)$. Ο αλγόριθμος τροποποιήθηκε από τον Klein [14] ο οποίος βελτίωσε την χρονική πολυπλοκότητα σε $O(|T_1|^2|T_2|\log|T_2|)$.

3.2.1.1. Αλγόριθμος MM-DIFF

Ο αλγόριθμος MM-DIFF που περιγράφεται στο [5], εφαρμόζεται σε δένδρα με ρίζα, επονομαζόμενα και ταξινομημένα (rooted, ordered, labeled trees). Οι λειτουργίες τροποποίησης που υποστηρίζονται είναι: εισαγωγή, διαγραφή και ανανέωση. Στον αλγόριθμο αυτό, ορίζεται μια βοηθητική δομή που καλείται γράφημα τροποποίησης. Στην οριζόντια στήλη του γραφήματος τοποθετούνται οι κόμβοι του δένδρου *T1* σε προδιατεταγμένη διάταξη και στην κάθετη στήλη, οι κόμβοι του *T2* σε προδιατεταγμένη διάταξη. Έστω ότι το δένδρο *T1* έχει *M* κόμβους και το δένδρο *T2*, *N* κόμβους. Τότε, το πρόβλημα της εύρεσης ενός σεναρίου τροποποίησης με ελάχιστο κόστος, επιλύεται με έναν αλγόριθμο εύρεσης της ελάχιστης διαδρομής από το σημείο (0,0) στο σημείο (M,N) του γραφήματος.

Έστω δύο ακολουθίες *A* = *ababaccdadab* και *B* = *acabbdbbabc*. Στο Σχήμα 3.1 παρουσιάζεται το γράφημα τροποποίησης. Με σκούρο χρώμα, φαίνεται μια διαδρομή

από το σημείο (0,0) στο σημείο (M,N). Οι οριζόντιες γραμμές αντιστοιχούν στη διαγραφή του στοιχείου A[x], οι κάθετες γραμμές στην εισαγωγή του στοιχείου B[y], ενώ οι διαγώνιες γραμμές επισημαίνουν το ταίριασμα του στοιχείου A[x] με το στοιχείο B[y]. Αν τα A[x] και B[y] έχουν διαφορετικά ονόματα, τότε οι διαγώνιες γραμμές αντιστοιχούν στην ανανέωση του ονόματος του A[x] στο όνομα του B[y]. Τελικά, το σενάριο τροποποίησης που παράγεται είναι: del(A[2]), ins(B[2]), del(A[5]), del(A[6]), upd(A[7], b), del(A[7]), del(A[8]), del(A[10]), ins(B[10]), del(A[11]), del(A[12]), ins(B[11]).



Σχήμα 3.1 Γράφημα Τροποποίησης της Ακολουθίας Α στην Ακολουθία Β

3.2.2. Αλγόριθμοι για αταξινόμητα δένδρα

Ο αλγόριθμος MH-Diff που περιγράφεται στα [6], [7] εφαρμόζεται σε μη-ταξινομημένα, επονομαζόμενα δένδρα με ρίζα (unordered, rooted, labeled trees). Εκτός από τις βασικές λειτουργίες τροποποίησης (insert, delete, update) που υποστηρίζονται από τους υπόλοιπους αλγορίθμους, ορίζονται τρεις επιπλέον λειτουργίες τροποποίησης: move, copy, glue. Η λειτουργία move υποδεικνύει την μετακίνηση ενός υποδένδρου με ρίζα τον κόμβο *n* στον κόμβο *p*. Η λειτουργία copy υποδεικνύει την αντιγραφή ενός υποδένδρου με ρίζα τον κόμβο *n* στον κόμβο *p*. Η λειτουργία glue υποδεικνύει την ένωση δύο υποδένδρων με ρίζες *n1* και *n2* αν τα υποδένδρα αυτά είναι ισομορφικά. Ουσιαστικά, αυτό σημαίνει την διαγραφή του υποδένδρου με ρίζα το *n1* αλλά ορίζεται για λόγους συμμετρίας αφού η λειτουργία glue είναι συμμετρική με τη λειτουργία copy. Έτσι, το παραγόμενο σενάριο

τροποποίησης από ένα δένδρο *T1* σε ένα δένδρο *T2* μπορεί να μετατραπεί παίρνοντας τις συμμετρικές λειτουργίες ώστε να μετατρέπει το δένδρο *T2* στο δένδρο *T1*. Το πρόβλημα αυτό είναι NP-hard. Χρησιμοποιώντας όμως μια ευρεστική λύση, το πρόβλημα επιλύεται στην χειρότερη περίπτωση σε χρόνο $O(n^3)$, όπου n είναι ο αριθμός των κόμβων. Ο αλγόριθμος αυτός, επειδή χρησιμοποιεί προσεγγιστικούς υπολογισμούς για να υπολογίσει το κόστος, δεν εγγυάται την βέλτιστη λύση.

ΚΕΦΑΛΑΙΟ 4. ΠΕΡΙΒΑΛΛΟΝ ΑΝΙΧΝΕΥΣΗΣ ΑΛΛΑΓΩΝ ΣΕ ΥΠΗΡΕΣΙΕΣ ΔΙΑΔΙΚΤΥΟΥ

4.1. Γενικά

Το πρώτο στάδιο για την υλοποίηση ενός περιβάλλοντος ανίχνευσης αλλαγών υπηρεσιών διαδικτύου, είναι η εκτενής μελέτη της δομής των υπηρεσιών διαδικτύου και των αλλαγών που συνήθως συμβαίνουν σε αυτές. Αφού καθοριστεί η σχέση μεταξύ των αλλαγών στις περιγραφές των WSDL και στις προγραμματιστικές αλλαγές που αυτές συνεπάγονται για τις εφαρμογές που τις χρησιμοποιούν, καθορίζουμε μια λίστα με τα επιθυμητά χαρακτηριστικά για το περιβάλλον ανίχνευσης αλλαγών. Στη συνέχεια, προτείνουμε μια μέθοδο ανίχνευσης αλλαγών σε WSDL περιγραφές.

4.1.1. Ορισμός του προβλήματος

Έστω η WSDL περιγραφή που παρουσιάζεται στον Πίνακα 4.1. Στην περιγραφή αυτή ορίζεται η λειτουργία *CreateIssue* που παίρνει σαν είσοδο το μήνυμα *CreateIssueSoapIn* και επιστρέφει το μήνυμα *CreateIssueSoapOut*. Το μήνυμα εισόδου έχει τη δομή που περιγράφεται στο στοιχείο *CreateIssue* ενώ το μήνυμα εξόδου έχει τη δομή που περιγράφεται στο στοιχείο *CreateIssueResponse*. Έστω η

PHP εφαρμογή που καλεί τη λειτουργία *CreateIssue* που παρουσιάζεται στον Πίνακα 4.2. Παρατηρούμε ότι στην κλήση *call()* της λειτουργίας *CreateIssue*, περνάμε σαν παράμετρο έναν πίνακα (array) με ένα στοιχείο *parameters*. Το στοιχείο αυτό αντιστοιχεί στο όνομα του μέρους (part) του μηνύματος *CreateIssueSoapIn*. Το στοιχείο *CreateIssue* που περιγράφει τη δομή του μηνύματος εισόδου, είναι ένας πολύπλοκος τύπος δεδομένων (complex type) που αποτελείται από τρία επιμέρους στοιχεία *ProjectID*, *ProblemDescr* και *FoundByName*. Παρατηρούμε ότι στην εφαρμογή PHP, εισάγουμε τιμές στα στοιχεία αυτά δημιουργώντας έναν ακόμα πίνακα *$arr_Issue_fields*. Το μήνυμα εξόδου θα αποθηκευτεί στην μεταβλητή *$result*, η οποία αποτελεί κείμενο (string) και εμφανίζεται με την εντολή *echo* στην οθόνη.

Πίνακας 4.1 WSDL περιγραφή της λειτουργίας CreateIssue

```
. . .
<s:element name="CreateIssue">
 <s:complexType>
  <s:sequence>
   <s:element minOccurs="1" maxOccurs="1" name="ProjectID" type="s:int" />
   <s:element minOccurs="0" maxOccurs="1" name="ProblemDescr" type="s:string" />
   <s:element minOccurs="0" maxOccurs="1" name="FoundByName" type="s:string" />
  </s:sequence>
 </s:complexType>
</s:element>
. . .
<wsdl:message name="CreateIssueSoapIn">
  <wsdl:part name="parameters" element="tns:CreateIssue" />
</wsdl:message>
<wsdl:message name="CreateIssueSoapOut">
  <wsdl:part name="return" element="s:string" />
</wsdl:message>
. . .
<wsdl:operation name="CreateIssue">
  <wsdl:input message="tns:CreateIssueSoapIn" />
  <wsdl:output message="tns:CreateIssueSoapOut" />
</wsdl:operation>
. . .
```

Πίνακας 4.2 Εφαρμογή PHP που καλεί την λειτουργία CreateIssue του Πίνακα 4.1

```
<?php

function CreateIssue(){
  $c = new nusoapclient('157739104.wsdl');

  $arr_Issue_fields = array('ProjectID' =>  79234875);
  $arr_Issue_fields += array('ProblemDescr' =>  'My Problem Description');
  $arr_Issue_fields += array('FoundByName' =>  'My Name');

  $result = $c->call('CreateIssue', array('parameters' => $arr_Issue_fields));

  echo $result;
}

?>
```

Έστω ότι η WSDL περιγραφή τροποποιείται και απαιτεί μια ακόμα παράμετρο κατά την κλήση της (*exception*) και επιστρέφει ένα ακόμα στοιχείο (*CreateIssueID*) στο μήνυμα εξόδου. Η τροποποιημένη WSDL περιγραφή παρουσιάζεται στον Πίνακα 4.3. Είναι προφανές ότι η εφαρμογή PHP που καλεί την λειτουργία *CreateIssue* θα πρέπει να υποστεί αλλαγές ώστε να διατηρήσει τη λειτουργικότητα της χρησιμοποιώντας την τροποποιημένη WSDL περιγραφή. Το ζητούμενο είναι να καθοδηγήσουμε τον προγραμματιστή με ένα σύνολο από προτεινόμενες αλλαγές ώστε η εφαρμογή του να είναι συμβατή με την τροποποιημένη έκδοση της WSDL περιγραφής. Στο συγκεκριμένο παράδειγμα, θα πρέπει να προταθεί στον προγραμματιστή η προσθήκη μιας νέας παραμέτρου εισόδου (complex type *IssueException*) στην κλήση της λειτουργίας καθώς και να ειδοποιηθεί για την αλλαγή της δομής της εξόδου σε σύνθετο τύπο με δύο πεδία, ένα εκ των οποίων ήταν αυτό που χρησιμοποιούσε στον κώδικά του. Η νέα έκδοση της εφαρμογής PHP, η οποία διατηρεί την ίδια λειτουργικότητα με την προηγούμενη έκδοση, αλλά χρησιμοποιεί την τροποποιημένη WSDL περιγραφή του Πίνακα 4.3, παρουσιάζεται στον Πίνακα 4.4.

Πίνακας 4.3 Τροποποιημένη έκδοση της WSDL περιγραφής του Πίνακα 4.1

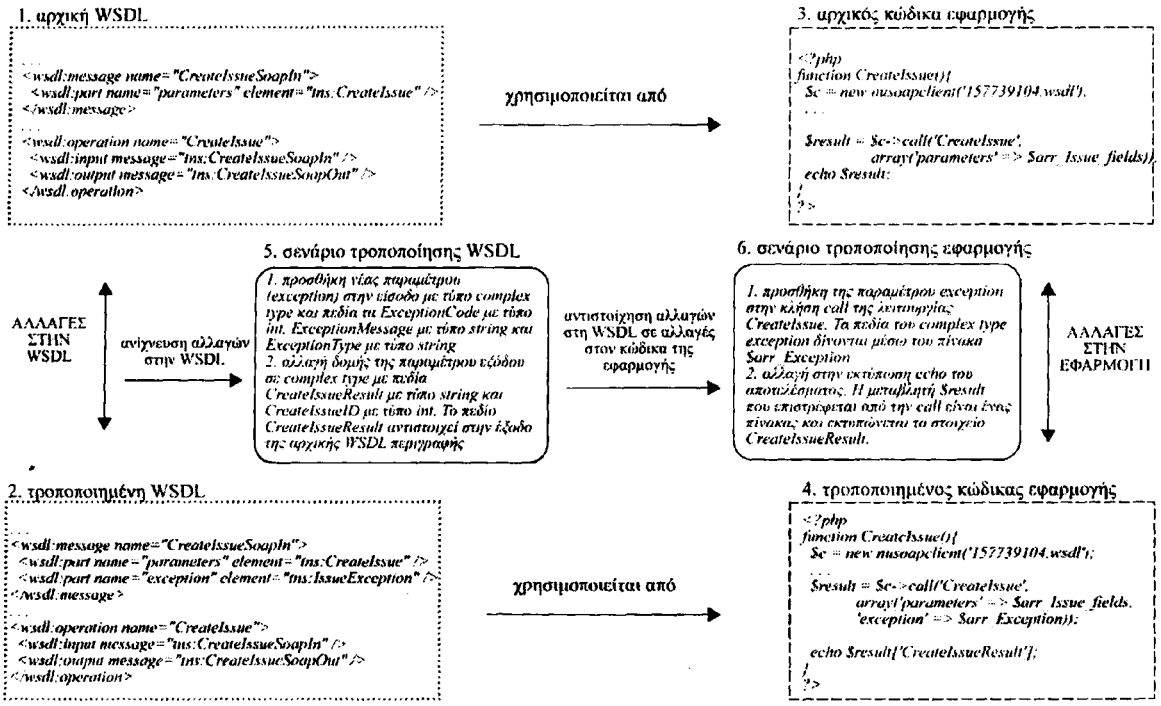
```
. . .
<s:element name="CreateIssue">
 <s:complexType>
  <s:sequence>
   <s:element minOccurs="1" maxOccurs="1" name="ProjectID" type="s:int" />
   <s:element minOccurs="0" maxOccurs="1" name="ProblemDescr" type="s:string" />
   <s:element minOccurs="0" maxOccurs="1" name="FoundByName" type="s:string" />
  </s:sequence>
 </s:complexType>
</s:element>
<s:element name="IssueException">
 <s:complexType>
  <s:sequence>
   <s:element minOccurs="1" maxOccurs="1" name="ExceptionCode" type="s:int" />
   <s:element minOccurs="0" maxOccurs="1" name="ExceptionMessage" type="s:string" />
   <s:element minOccurs="0" maxOccurs="1" name="ExceptionType" type="s:string" />
  </s:sequence>
 </s:complexType>
</s:element>
<s:element name="CreateIssueResponse">
 <s:complexType>
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="CreateIssueResult" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="CreateIssueID" type="s:int" />
  </s:sequence>
 </s:complexType>
</s:element>
. . .
<wsdl:message name="CreateIssueSoapIn">
  <wsdl:part name="parameters" element="tns:CreateIssue" />
  <wsdl:part name="exception" element="tns:IssueException" />
</wsdl:message>
<wsdl:message name="CreateIssueSoapOut">
  <wsdl:part name="return" element="tns:CreateIssueResponse" />
</wsdl:message>
. . .
<wsdl:operation name="CreateIssue">
  <wsdl:input message="tns:CreateIssueSoapIn" />
  <wsdl:output message="tns:CreateIssueSoapOut" />
</wsdl:operation>
...
```

Πίνακας 4.4 Τροποποιημένη έκδοση της εφαρμογής PHP του Πίνακα 4.2

```
<?php

function CreateIssue(){
  $c = new nusoapclient('157739104.wsdl');

  $arr_Issue_fields = array('ProjectID' =>  79234875);
  $arr_Issue_fields += array('ProblemDescr' =>  'My Problem Description');
  $arr_Issue_fields += array('FoundByName' =>  'My Name');

  $arr_Exception = array('ExceptionCode' =>  329687);
  $arr_Exception += array('ExceptionMessage' =>  'My Exception Message');
  $arr_Exception += array('ExceptionType' =>  'My Exception Type');

  $result = $c->call('CreateIssue',
            array('parameters' => $arr_Issue_fields, 'exception' => $arr_Exception));

  echo $result['CreateIssueResult'];
}

?>
```

Μια σχηματική αναπαράσταση της διαδικασίας ανίχνευσης αλλαγών στην WSDL περιγραφή και καθοδήγησης του προγραμματιστή για τις αντίστοιχες αλλαγές που

πρέπει να πραγματοποιηθούν στον κώδικα, απεικονίζεται στο Σχήμα 4.1. Στα πλαίσια 1 και 2, παρουσιάζονται τμήματα της αρχικής και της τροποποιημένης WSDL περιγραφής αντίστοιχα, ενώ στα πλαίσια 3 και 4 παρουσιάζονται τμήματα της εφαρμογής που χρησιμοποιεί την υπηρεσία πριν και μετά τις αλλαγές. Η διαδικασία ανίχνευσης αλλαγών γίνεται σε επίπεδο WSDL περιγραφών. Οι αλλαγές αυτές αποτυπώνονται σε ένα σενάριο τροποποίησης που παρουσιάζεται στο πλαίσιο 5 και στη συνέχεια, οι αλλαγές αυτές αντιστοιχίζονται σε αλλαγές που πρέπει να γίνουν στον κώδικα της εφαρμογής. Οι αλλαγές αυτές παρουσιάζονται στο πλαίσιο 6.



Σχήμα 4.1 Αναπαράσταση της διαδικασίας ανίχνευσης αλλαγών στην WSDL περιγραφή και αντιστοίχησης σε αλλαγές στην εφαρμογή

Συνοψίζοντας, στόχος είναι ο καθορισμός ενός σεναρίου τροποποίησης, δηλαδή μιας αλληλουχίας στοιχειωδών τροποποιήσεων που μπορούν να χρησιμοποιηθούν από τον προγραμματιστή για την αλλαγή του κώδικα της εφαρμογής του με τέτοιο τρόπο, ώστε να είναι συμβατή με τη νέα έκδοση της WSDL περιγραφής. Ως στοιχειώδεις τροποποιήσεις, ορίζονται οι εξής:

- Προσθήκη στοιχείου με απλό τύπο σε οποιοδήποτε σημείο της δομής της εισόδου ή της εξόδου (ακόμα και σε πολύπλοκο τύπο)
- Διαγραφή στοιχείου με απλό τύπο από οποιοδήποτε σημείο της δομής της εισόδου ή της εξόδου

- Μετακίνηση στοιχείου με απλό ή σύνθετο τύπο από ένα σημείο της δομής της εισόδου ή της εξόδου σε ένα άλλο σημείο
- Αλλαγή του ονόματος ενός στοιχείου σε οποιοδήποτε σημείο της δομής της εισόδου ή της εξόδου

Ιδανικά, το σενάριο τροποποίησης που προτείνεται στον προγραμματιστή θα πρέπει να είναι βέλτιστο από τη σκοπιά του προγραμματιστή. Με τον όρο βέλτιστο, εννοούμε ότι ελαχιστοποιεί τον «κόπο» που πρέπει να καταβάλλει ο προγραμματιστής για να προσαρμόσει κατάλληλα την εφαρμογή του. Για την ποσοτικοποίηση του «κόπου», σε κάθε λειτουργία στοιχειώδους τροποποίησης αντιστοιχίζεται ένα κόστος. Το άθροισμα του κόστους όλων των βημάτων ενός σεναρίου, ορίζεται ως το κόστος εφαρμογής του στον κώδικα. Επομένως, το πρόβλημα της βελτιστοποίησης του προτεινόμενου σεναρίου τροποποίησης, ανάγεται στην ελαχιστοποίηση του κόστους του.

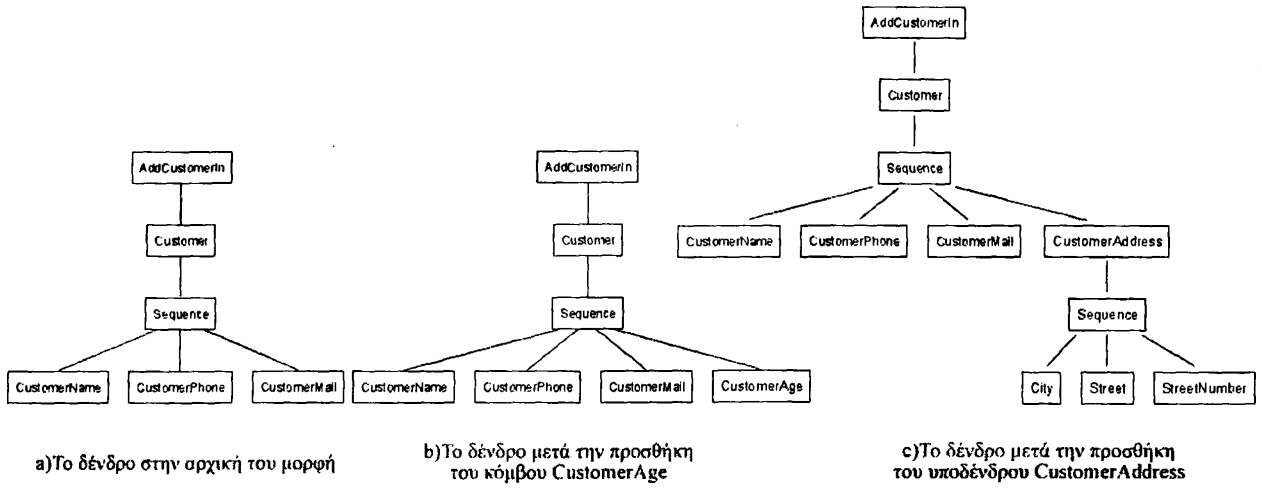
Είναι προφανές ότι για τη δημιουργία και βελτιστοποίηση του σεναρίου τροποποίησης, θα πρέπει να μελετηθούν τόσο οι αλλαγές που συνήθως συμβαίνουν στις WSDL περιγραφές που εξετάζουμε αλλά και η συσχέτισή τους με τις συνεπαγόμενες αλλαγές στον κώδικα των εφαρμογών που τις χρησιμοποιούν. Το είδος των αλλαγών που συμβαίνουν συνήθως στις WSDL περιγραφές θα καθορίσει και τις διαθέσιμες μεθόδους για ανίχνευσή τους, ενώ η συσχέτιση των αλλαγών αυτών με τις αντίστοιχες στον κώδικα των εφαρμογών θα καθορίσει τα κόστη που θα αντιστοιχηθούν στις αλλαγές που ανιχνεύθηκαν.

4.1.2. Μελέτη των αλλαγών στις υπηρεσίες διαδικτύου

Στην ενότητα αυτή μελετάμε αλλαγές που μπορεί να εφαρμοστούν σε μια υπηρεσία διαδικτύου στα πλαίσια της συντήρησης. Τέτοιες αλλαγές είναι η προσθήκη, διαγραφή ή τροποποίηση πεδίων από σύνθετους τύπους δεδομένων, η προσθήκη, διαγραφή ή τροποποίηση παραμέτρων εισόδου ή εξόδου από μια λειτουργία, η μετακίνηση ενός σύνθετου τύπου δεδομένων που συχνά γίνεται στα πλαίσια της συντήρησης και βελτιστοποίησης του κώδικα (refactoring).

Για παράδειγμα, έστω μια λειτουργία *AddCustomer(Customer c)* που προσθέτει έναν πελάτη. Για κάθε πελάτη (*Customer*), αποθηκεύεται το όνομα του (*CustomerName*), ένα τηλέφωνο επικοινωνίας (*CustomerPhone*) και ένα e-mail (*CustomerMail*). Αν

θέλουμε να αποθηκεύουμε την ηλικία του πελάτη, αρκεί στον τύπο δεδομένων *Customer* να προσθέσουμε ένα πεδίο *CustomerAge* με τύπο int. Στο δένδρο, η αλλαγή αυτή αντιστοιχίζεται με την εισαγωγή ενός κόμβου φύλλου. Τα δένδρα πριν και μετά την εισαγωγή παρουσιάζονται στα Σχήματα 4.2(a) και 4.2(b) αντίστοιχα.



a)Το δένδρο στην αρχική του μορφή

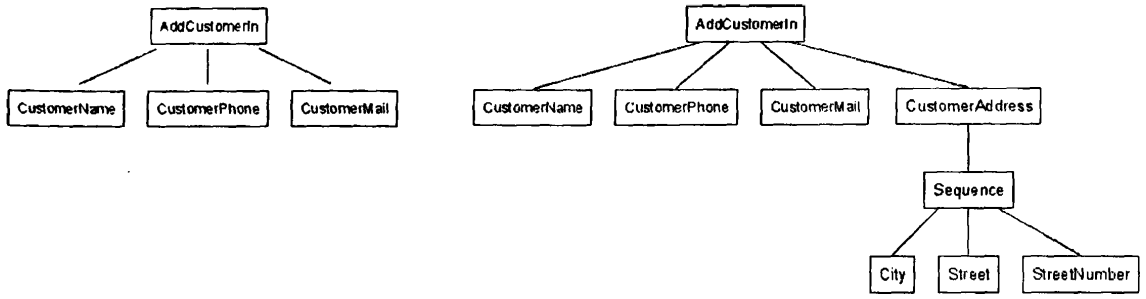
b)Το δένδρο μετά την προσθήκη του κόμβου CustomerAge

c)Το δένδρο μετά την προσθήκη του υποδένδρου CustomerAddress

Σχήμα 4.2 Παράδειγμα εισαγωγής ενός κόμβου και ενός υποδένδρου

Αν για κάθε πελάτη θέλουμε να αποθηκεύουμε και την διεύθυνσή του (*Address*), η οποία αποτελείται από την πόλη (*City*), την οδό (*Street*) και τον αριθμό (*StreetNumber*), τότε θα πρέπει να προσθέσουμε ένα σύνθετο τύπο δεδομένων *CustomerAddress* που θα έχει τρία πεδία, *City* με τύπο string, *Street* με τύπο string και *StreetNumber* με τύπο int. Στο δένδρο, η αλλαγή αυτή αντιστοιχίζεται με την εισαγωγή ενός υποδένδρου κάτω από έναν κόμβο φύλλο, όπως φαίνεται στο Σχήμα 4.2(c).

Έστω ότι η λειτουργία *AddCustomer()* παίρνει σαν παραμέτρους κάθε πεδίο που αφορά τον πελάτη ξεχωριστά. Τότε θα έχει τη μορφή *AddCustomer(CustomerName cm, CustomerPhone cp, CustomerMail cmail)*. Σε αυτή την περίπτωση, η αποθήκευση της διεύθυνσης σημαίνει την προσθήκη μια καινούργιας παραμέτρου, οπότε η μορφή της λειτουργίας θα είναι *AddCustomer(CustomerName cm, CustomerPhone cp, CustomerMail cmail, CustomerAddress ca)*. Στο δένδρο, η αλλαγή αυτή αντικατοπτρίζεται με την εισαγωγή ενός υποδένδρου κάτω από έναν κόμβο φύλλο, όπως ακριβώς και στην προηγούμενη περίπτωση. Τα δένδρα πριν και μετά την εισαγωγή της παραμέτρου *CustomerAddress*, φαίνονται στα Σχήματα 4.3(a) και 4.3(b).

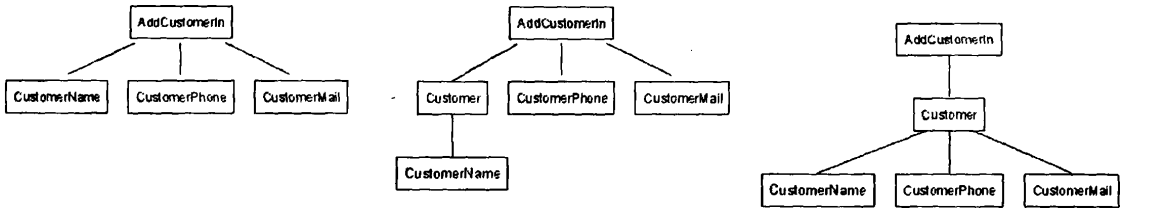


a)Το δένδρο στην αρχική του μορφή

b)Το δένδρο μετά την προσθήκη της παραμέτρου CustomerAddress

Σχήμα 4.3 Παράδειγμα εισαγωγής μιας παραμέτρου

Σε αυτό το παράδειγμα, έστω ότι στα πλαίσια της βελτιστοποίησης και συντήρησης του κώδικα (refactoring), θέλουμε να δημιουργήσουμε ένα σύνθετο τύπο δεδομένων *Customer* με πεδία *CustomerName*, *CustomerPhone* και *CustomerMail* έτσι ώστε η λειτουργία *AddCustomer()* να παίρνει σαν παράμετρο μόνο μια παράμετρο τύπου *Customer*. Η αλλαγή αυτή αντικατοπτρίζεται στο δένδρο με την εισαγωγή ενός κόμβου *Customer* στη θέση του κόμβου *CustomerName*, οπότε ο κόμβος *CustomerName* γίνεται παιδί του *Customer*, και στη συνέχεια γίνεται μετακίνηση των κόμβων *CustomerPhone* και *CustomerMail* κάτω από τον κόμβο *Customer*. Η διαδικασία της αλλαγής παρουσιάζεται στο Σχήμα 4.4.



a)Το δένδρο στην αρχική του μορφή

b)Το δένδρο μετά την προσθήκη του κόμβου Customer

c)Το δένδρο μετά την μετακίνηση των κόμβων CustomerPhone και CustomerMail στον κόμβο Customer

Σχήμα 4.4 Εισαγωγή του κόμβου Customer και μετακίνηση των κόμβων CustomerPhone και CustomerMail κάτω από τον κόμβο Customer

Αλλαγές που σχετίζονται με τη διαγραφή πεδίων ή σύνθετων τύπων δεδομένων, είναι οι αντίστροφες από τις αλλαγές που σχετίζονται με την προσθήκη. Τέτοια παραδείγματα είναι η διαγραφή του απλού πεδίου *CustomerAge* που αντιστοιχίζεται στη διαγραφή ενός κόμβου φύλλου, η διαγραφή του σύνθετου τύπου *CustomerAddress* είτε από τη θέση μιας παραμέτρου είτε από τον σύνθετο τύπο

δεδομένων *Customer* που αντιστοιχίζεται στη διαγραφή ενός υποδένδρου ή ο διαχωρισμός ενός σύνθετου τύπου δεδομένων σε πιο απλούς που είναι το αντίστροφο του Σχήματος 4.2.

Όπως είναι φανερό, λόγω της ιεραρχικής της δομής, μια WSDL περιγραφή μπορεί να μοντελοποιηθεί με μια δενδρική δομή. Με δεδομένο αυτό, η ανίχνευση αλλαγών σε WSDL περιγραφές μπορεί να αναχθεί στο πρόβλημα της ανίχνευσης αλλαγών σε δενδρικές δομές. Για τη λύση του προβλήματος αυτού έχει προταθεί μια σειρά αλγορίθμων, τόσο για ταξινομημένα όσο και αταξινόμητα δένδρα. Οι αλγόριθμοι αυτοί παίρνουν σαν είσοδο δύο δενδρικές δομές *T1* και *T2* (αρχική και τελική) και είναι σε θέση: a) να παράγουν ένα σενάριο τροποποίησης *S* που μετατρέπει το δένδρο *T1* στο δένδρο *T2* και b) υπολογίζουν την απόσταση μεταξύ των δύο δένδρων ως το άθροισμα του κόστους των επιμέρους λειτουργιών τροποποίησης που περιέχονται στο σενάριο *S*. Για να γίνει αυτό εφικτό, έχει ανατεθεί σε κάθε στοιχειώδη λειτουργία τροποποίησης μια συνάρτηση κόστους. Μια πιο λεπτομερής αναφορά στους αλγορίθμους ανίχνευσης αλλαγών σε δένδρα γίνεται στην ενότητα 3.2.

Στην εργασία αυτή, μοντελοποιούμε σε δενδρικές δομές μια WSDL περιγραφή ως εξής:

- Για κάθε λειτουργία O_i κατασκευάζονται δύο δένδρα T_i^{input} και T_i^{output}. Το δένδρο T_i^{input} αντιστοιχεί στην δενδρική δομή του μηνύματος εισόδου της O_i και το δένδρο T_i^{output} στην δενδρική δομή του μηνύματος εξόδου της.
- Η ρίζα του κάθε δένδρου αντιστοιχεί στο μήνυμα εισόδου ή εξόδου αντίστοιχα.
- Κάθε μέρος (part) ενός μηνύματος αντιστοιχίζεται σε παιδί της ρίζας.
- Ένας απλός τύπος δεδομένων (primitive type) αντιστοιχίζεται σε έναν κόμβο φύλλο.
- Ένας πολύπλοκος τύπος δεδομένων (complex type) αντιστοιχίζεται σε ένα υποδένδρο.

4.1.3. Αντιστοίχηση αλλαγών στις WSDL περιγραφές σε αλλαγές στις εφαρμογές

Οι αλλαγές στην WSDL περιγραφή μιας υπηρεσίας διαδικτύου συνεπάγονται αλλαγές στις εφαρμογές που την χρησιμοποιούν. Το πραγματικό κόστος των αλλαγών αυτών σε προγραμματιστικούς πόρους εξαρτάται και από την έκταση των αλλαγών αλλά και

από τη φύση τους. Με τον όρο έκταση εννοούμε το πλήθος των λειτουργιών τροποποίησης. Είναι προφανές, για παράδειγμα, ότι μεγαλύτερος αριθμός από εισαγωγές και διαγραφές στη δομή της εισόδου μιας λειτουργίας συνεπάγονται περισσότερες προγραμματιστικές αλλαγές για την εφαρμογή που καλεί τη συγκεκριμένη λειτουργία. Με τον όρο φύση, εννοούμε το είδος των λειτουργιών τροποποίησης, δηλαδή αν πρόκειται για εισαγωγή, διαγραφή ή κάποιο άλλο είδος λειτουργίας τροποποίησης. Για παράδειγμα, η εισαγωγή μιας επιπλέον παραμέτρου στην είσοδο που αντιστοιχεί σε μια λειτουργία εισαγωγής συνεπάγεται περισσότερο κόστος από την λειτουργία ανανέωσης του ονόματος μιας υπάρχουσας παραμέτρου.

Στην πραγματικότητα, το προγραμματιστικό κόστος που συνεπάγεται μια συγκεκριμένη λειτουργία τροποποίησης στη δομή δεν μπορεί να καθοριστεί με ακρίβεια διαθέτοντας μόνο την WSDL περιγραφή. Για να γίνει αυτό εφικτό, είναι απαραίτητο να γνωρίζουμε με ποιο τρόπο χρησιμοποιεί ο προγραμματιστής τη λειτουργία. Χαρακτηριστική περίπτωση είναι η διαφοροποίηση στο προγραμματιστικό κόστος ανάλογα με το αν μια παράμετρος εξόδου που διαγράφηκε χρησιμοποιούνταν για περαιτέρω υπολογισμούς στην εφαρμογή.

Για παράδειγμα, θεωρούμε την λειτουργία *getCustomer()* η οποία επιστρέφει ένα αντικείμενο *Customer* όμοιο με αυτό του Σχήματος 4.4(c). Εάν η WSDL περιγραφή τροποποιηθεί έτσι ώστε ο πολύπλοκος τύπος *Customer* να μην περιέχει πια το πεδίο *CustomerMail*, το προγραμματιστικό κόστος της διαγραφής εξαρτάται καθαρά από την εσωτερική λογική της εφαρμογής. Εάν το πεδίο *CustomerMail* δεν χρησιμοποιούνταν καθόλου στην εφαρμογή ή είχε ένα περιφερειακό ρόλο, τότε το κόστος είναι μικρό. Εάν το πεδίο αυτό συμμετείχε ενεργά στις διαδικασίες της εφαρμογής (εφαρμογή που ενημερώνει τους πελάτες μέσω e-mail), τότε η συγκεκριμένη υπηρεσία είναι ακόμα και ακατάλληλη για την εφαρμογή ή οι αλλαγές στη λογική της εφαρμογής θα πρέπει να είναι εκτεταμένες.

Δεδομένου ότι ο ακριβής καθορισμός των προγραμματιστικών αλλαγών δεν είναι εφικτός, η προσέγγιση που θα ακολουθήσουμε είναι η ανάθεση σε κάθε λειτουργία τροποποίησης μιας προκαθορισμένης τιμής για το κόστος. Κριτήριο για την επιλογή αυτών των τιμών είναι η προγραμματιστική συνέπεια των παραγόμενων σεναρίων τροποποίησης, δηλαδή το πόσο λογικές σε προγραμματιστικό επίπεδο είναι οι προτεινόμενες αλλαγές.

Για παράδειγμα, θεωρούμε το πολύπλοκο τύπο *Customer* του Σχήματος 4.2. Έστω ότι διαγράφεται το πεδίο *CustomerMail* και προστίθεται το πεδίο *CustomerPostCode* στον τύπο δεδομένων *CustomerAddress*. Το προγραμματιστικά ορθό σενάριο είναι η διαγραφή του *CustomerMail* και η προσθήκη του *CustomerAddress* δεδομένου ότι έχουν τελείως διαφορετικό ρόλο στην εφαρμογή. Στην περίπτωση αυτή, τα κόστη των λειτουργιών τροποποίησης θα πρέπει να είναι τέτοια, ώστε να αποφευχθεί το σενάριο μετακίνησης του *CustomerMail* στη θέση του *CustomerPostCode* και στη συνέχεια ανανέωσης της ετικέτας του (όνομα και τύπος). Αξίζει να σημειωθεί ότι το σενάριο μετακίνησης και ανανέωσης είναι απολύτως λογικό για έναν αλγόριθμο ανίχνευσης αλλαγών σε δένδρα εάν τα κόστη των λειτουργιών δεν το αποτρέπουν.

Με δεδομένα όλα τα παραπάνω, τα κόστη των λειτουργιών τροποποίησης για την ανίχνευση αλλαγών στα WSDL δένδρα παρουσιάζονται στον Πίνακα 4.5. Διαισθητικά, θεωρούμε ότι η μετακίνηση μιας δομής έχει το μικρότερο κόστος σε προγραμματιστικό επίπεδο, εφόσον στην πλειοψηφία των περιπτώσεων, οι μηχανισμοί που μεταχειρίζονται την αντίστοιχη δομή σε επίπεδο κώδικα έχουν ήδη υλοποιηθεί. Αυτό που συνήθως συμβαίνει είναι η αλλαγή του τρόπου που καλούνται. Η΄ εισαγωγή ενός πεδίου κοστίζει περισσότερο, δεδομένου ότι ο κώδικας που διαχειρίζεται τη δομή αυτή πρέπει να γραφτεί εξαρχής. Αντίστοιχα, η διαγραφή ενός πεδίου, μπορεί να προκαλέσει αρκετές αλλαγές στην λογική της εφαρμογής. Χαρακτηριστικό παράδειγμα είναι η διαγραφή του πεδίου *CustomerMail* σε μια εφαρμογή που ενημερώνει πελάτες μέσω e-mail.

Η ανανέωση δεν κοστίζει πολύ σε προγραμματιστικό επίπεδο. Επειδή, όμως, προτεραιότητα είναι τα προτεινόμενα σενάρια τροποποίησης να είναι προγραμματιστικά ορθά (όπως αναφέρθηκε σε προηγούμενη παράγραφο), επιλέξαμε το κόστος της ανανέωσης συν το κόστος της μετακίνησης να είναι ίσο με το κόστος της εισαγωγής συν το κόστος της διαγραφής και επιπλέον το κόστος της ανανέωσης να είναι μικρότερο από το κόστος της εισαγωγής συν το κόστος της διαγραφής. Με αυτόν τον τρόπο, η πιθανότητα να επιλεχθεί η ανανέωση ενός πεδίου είναι μεγαλύτερη από την πιθανότητα το πεδίο αυτό να διαγραφεί και να εισαχθεί στη συνέχεια στην ίδια θέση με διαφορετικό όνομα. Αντίστοιχα, η πιθανότητα να διαγραφεί ένα πεδίο και να εισαχθεί ένα καινούργιο πεδίο με διαφορετικό όνομα σε μια διαφορετική θέση αντί να μετακινηθεί στην καινούργια θέση και στη συνέχεια να ανανεωθεί, εξαρτάται και από τους υπόλοιπους κόμβους του δένδρου. Τέλος, επειδή

ορισμένοι αλγόριθμοι ανίχνευσης αλλαγών, διαθέτουν τις λειτουργίες αντιγραφής και συνένωσης και εφόσον οι λειτουργίες αυτές δεν έχουν άμεση αντιστοίχηση σε προγραμματιστικό επίπεδο, επιλέξαμε το κόστος τους να είναι μεγαλύτερο από τα υπόλοιπα, δίνοντας πολύ μικρή πιθανότητα να εμφανιστούν στο σενάριο τροποποίησης.

Πίνακας 4.5 Κόστη των λειτουργιών τροποποίησης

Λειτουργία	**Αναγνωριστικό**	**Κόστος**
Προσθήκη	ins	3
Διαγραφή	del	3
Ανανέωση	upd	0 αν έχουν ίδιο όνομα 4 αν έχουν διαφορετικό
Μετακίνηση	mov	2
Αντιγραφή	cpy	5
Συνένωση	glu	5

4.1.4. Καθορισμός των παραμέτρων του προβλήματος

Συνοψίζοντας τα αποτελέσματα της μελέτης που έγινε σχετικά με τις αλλαγές που συμβαίνουν στις υπηρεσίες διαδικτύου και το προγραμματιστικό κόστος που αυτές συνεπάγονται στις εφαρμογές που τη χρησιμοποιούν, ένας ακριβής ορισμός του προβλήματος είναι:

Δεδομένης μιας WSDL περιγραφής *W1* και μιας τροποποιημένης έκδοσης της, *W1'*, σκοπός είναι η εξαγωγή ενός σεναρίου τροποποίησης *S* αποτελούμενο από μια αλληλουχία *M* στοιχειωδών λειτουργιών τροποποίησης s_i με κόστος $c(s_i)$, που ελαχιστοποιεί το κόστος *C* του *S*. Το κόστος *C(S)* ορίζεται ως το άθροισμα του κόστους $c(s_i)$, για κάθε $0 \leq i \leq M$. Καθορίζοντας τα κόστη $c(s_i)$ με τέτοιο τρόπο ώστε να προσεγγίζουν σε ικανοποιητικό βαθμό τα πραγματικά προγραμματιστικά κόστη, το σενάριο τροποποίησης *S* μπορεί να καθοδηγήσει τον προγραμματιστή ως προς τις αλλαγές που πρέπει να κάνει στην εφαρμογή του με στόχο να χρησιμοποιήσει την *W1'* αντί της *W1*.

Συνεκτιμώντας όλες τις παραμέτρους του προβλήματος, καταλήγουμε στον καθορισμό ενός συνόλου επιθυμητών χαρακτηριστικών για τον αλγόριθμο ανίχνευσης αλλαγών δύο υπηρεσιών διαδικτύου. Τα χαρακτηριστικά αυτά είναι:

- Επιλεκτική σύγκριση συγκεκριμένων μερών της WSDL περιγραφής μεταξύ τους. Αν και ολόκληρες οι WSDL περιγραφές αποτελούν στην ουσία ένα δένδρο και επομένως θα μπορούσαν να συγκριθούν απευθείας μεταξύ τους, μια τέτοια προσέγγιση δεν θα παρήγαγε σενάρια τροποποίησης με ιδιαίτερη χρησιμότητα για τον προγραμματιστή. Αυτό που μας ενδιαφέρει είναι να παράγουμε σενάρια τροποποίησης που να υπαγορεύουν στον προγραμματιστή συγκεκριμένες αλλαγές για τον τρόπο που χρησιμοποιεί την υπηρεσία στην εφαρμογή του. Δεδομένου ότι από τη σκοπιά του προγραμματιστή μια υπηρεσία διαδικτύου είναι ένα σύνολο λειτουργιών, αυτό που μας ενδιαφέρει είναι να παράγουμε σενάρια για τη μετάβαση κάθε χρησιμοποιούμενης λειτουργίας στην αντίστοιχή της. Για παράδειγμα, έστω δύο υπηρεσίες διαδικτύου Δ και Δ' που θέλουμε να συγκρίνουμε. Στόχος είναι η δημιουργία σεναρίων τροποποίησης ελάχιστου κόστους για την είσοδο και έξοδο κάθε λειτουργίας Λ_i της Δ στην αντίστοιχή της Λ_i' της Δ'. Γίνεται, επομένως, κατανοητό ότι ο αλγόριθμος θα πρέπει να είναι σε θέση να καθορίζει την αντιστοιχία μεταξύ των λειτουργιών των δύο υπηρεσιών χωρίς να βασίζεται στην ονομασία τους. Ο τελευταίος αυτός περιορισμός σχετικά με την μη αποκλειστική χρήση της ονομασίας των λειτουργιών για την αντιστοίχηση, τίθεται με στόχο να καλύψει την περίπτωση μετάβασης μεταξύ παραγόμενων υπηρεσιών από διαφορετικούς κατασκευαστές. Στις περιπτώσεις αυτές, είναι κοινό ο κάθε κατασκευαστής να τροποποιεί τα ονόματα των λειτουργιών είτε εντελώς είτε προσθέτοντας κάποιο πρόθεμα.
- Χρήση ενός αλγορίθμου ανίχνευσης αλλαγών σε δένδρα που θα αποδίδει καλύτερα σε αλλαγές σαν και αυτές που συνήθως συμβαίνουν στις WSDL περιγραφές. Στόχος δεν είναι η αντιμετώπιση του προβλήματος σαν ένα γενικό πρόβλημα ανίχνευσης αλλαγών σε γράφους αλλά ο καθορισμός συγκεκριμένων προτύπων αλλαγών (change patterns) και η επιλογή αλγορίθμων που να αποδίδουν καλά σε τέτοιου είδους αλλαγές.
- Τα παραγόμενα σενάρια τροποποίησης θα πρέπει να αντικατοπτρίζουν ρεαλιστικές και εφικτές προγραμματιστικές αλλαγές. Ενδεικτικό παράδειγμα του επιθυμητού αυτού χαρακτηριστικού είναι η προτίμηση του σεναρίου τροποποίησης με λειτουργίες εισαγωγής και διαγραφής αντί του σεναρίου

τροποποίησης με λειτουργίες μετακίνησης και ανανέωσης που περιγράφτηκε στην παράγραφο 4.1.3.

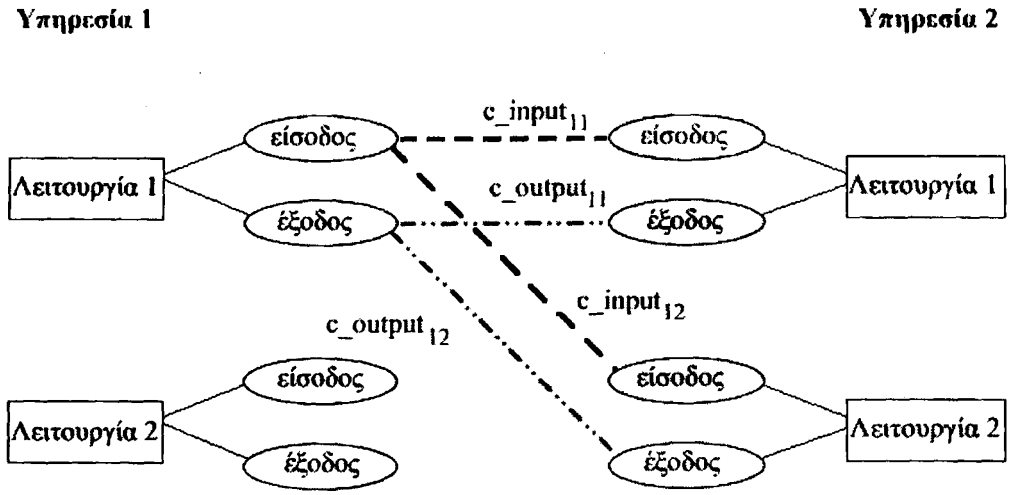
- Η ελαχιστοποίηση του εκτιμώμενου προγραμματιστικού κόστους που συνεπάγονται οι προτεινόμενες από το σενάριο τροποποίησης αλλαγές.
- Η διατήρηση της πολυπλοκότητας και επομένως των χρόνων εκτέλεσης σε όσο το δυνατόν χαμηλότερα επίπεδα. Δεδομένου ότι οι αλγόριθμοι ανίχνευσης αλλαγών σε δένδρα, ειδικά στην περίπτωση μη ταξινομημένων δένδρων, έχουν μεγάλη πολυπλοκότητα, τα αποτελέσματα ως προς το χαρακτηριστικό αυτό θα πρέπει να αξιολογηθούν σε αντίστοιχη βάση.

4.1.5. Προτεινόμενη μέθοδος

Με βάση τα επιθυμητά χαρακτηριστικά που καθορίστηκαν στην ενότητα 4.1.4, προτείνουμε μια μέθοδο ανίχνευσης αλλαγών και παραγωγής σεναρίου τροποποίησης μεταξύ δύο υπηρεσιών διαδικτύου *WS1* και *WS2* η οποία αποτελείται από δύο φάσεις. Η πρώτη φάση της μεθόδου, λαμβάνει σαν είσοδο ένα σύνολο από δενδρικές δομές για κάθε υπηρεσία. Συγκεκριμένα, για κάθε λειτουργία O_i^1 και O_j^2 των υπηρεσιών διαδικτύου *WS1* και *WS2* αντίστοιχα, λαμβάνει το πολύ δύο δενδρικές δομές, μια για την είσοδο (*T_input*) της λειτουργίας και μια για την έξοδο (*T_output*). Στη συνέχεια, συγκρίνει την είσοδο κάθε λειτουργίας O_i^1 με την είσοδο κάθε λειτουργίας O_j^2 και προκύπτει το κόστος c_input_{ij} και την έξοδο κάθε λειτουργίας O_i^1 με την έξοδο κάθε λειτουργίας O_j^2 και προκύπτει το κόστος c_output_{ij}. Το συνολικό κόστος c_{ij} της μετατροπής της λειτουργίας O_i^1 στην λειτουργία O_j^2 προκύπτει από το άθροισμα των c_input_{ij} και c_output_{ij}. Η πρώτη φάση ολοκληρώνεται όταν υπολογιστούν όλα τα κόστη c_{ij}, $0 \le i \le M$, $0 \le j \le N$, όπου M και N είναι ο αριθμός των λειτουργιών της υπηρεσίας *WS1* και *WS2* αντίστοιχα.

Για παράδειγμα, θεωρούμε τις δύο υπηρεσίες του Σχήματος 4.5. Η δομή της εισόδου της Λειτουργίας 1 της Υπηρεσίας 1 συγκρίνεται με την δομή της εισόδου της Λειτουργίας 1 της Υπηρεσίας 2. Αποτέλεσμα της σύγκρισης είναι το κόστος c_input_{11}. Στη συνέχεια, η δομή της εισόδου της Λειτουργίας 1 της Υπηρεσίας 1 συγκρίνεται με την δομή της εισόδου της Λειτουργίας 2 της Υπηρεσίας 2. Αποτέλεσμα της σύγκρισης είναι το κόστος c_input_{12}. Αντίστοιχα, συγκρίνεται και η έξοδος της Λειτουργίας 1 της Υπηρεσίας 1 με τις εξόδους των Λειτουργιών 1 και 2

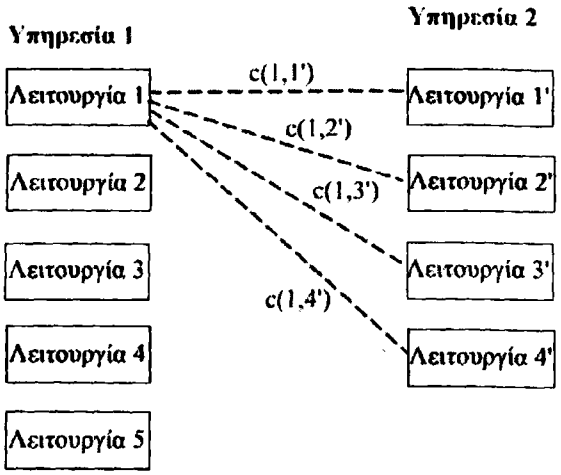
της Υπηρεσίας 2. Αποτελέσματα αυτών των συγκρίσεων είναι τα κόστη c_output_{11} και c_output_{12}. Τελικά, τα κόστη μετατροπής της Λειτουργίας 1 της Υπηρεσίας 1 στις Λειτουργίες 1 και 2 της Υπηρεσίας 2 είναι: $c_{11} = c\_input_{11} + c\_output_{11}$, $c_{12} = c\_input_{12} + c\_output_{12}$. Με τον ίδιο τρόπο υπολογίζονται τα κόστη c_{21} και c_{22}.



Σχήμα 4.5 Πρώτη φάση της προτεινόμενης μεθόδου

Η δεύτερη φάση της μεθόδου, λαμβάνει σαν είσοδο όλα τα κόστη c_{ij} που υπολογίστηκαν στην πρώτη φάση της μεθόδου και αντιστοιχίζει κάθε λειτουργία της υπηρεσίας *WS1* με μία μόνο λειτουργία της υπηρεσίας *WS2*. Κριτήριο για αυτή την αντιστοίχηση είναι το ελάχιστο κόστος. Ο αλγόριθμος που χρησιμοποιείται για την ένα-προς-ένα αντιστοίχηση είναι ο Ουγγρικός αλγόριθμος.

Για παράδειγμα, θεωρούμε τις δύο υπηρεσίες του Σχήματος 4.6. Τα κόστη μεταξύ των λειτουργιών της Υπηρεσίας 1 και της Υπηρεσίας 2 παρουσιάζονται στον Πίνακα 4.6. Κάθε στήλη του Πίνακα 4.6 αντιστοιχεί σε μια λειτουργία της Υπηρεσίας 1 ενώ κάθε γραμμή αντιστοιχεί σε μια λειτουργία της Υπηρεσίας 2. Κάθε κελί [i,j] έχει τιμή το κόστος μετατροπής της Λειτουργίας *i* στην Λειτουργία *j*. Στόχος της δεύτερης φάσης της μεθόδου είναι κάθε γραμμή να αντιστοιχίζεται σε μια μόνο στήλη. Αν η γραμμή i_1 αντιστοιχίζεται στη στήλη j_1, τότε η λειτουργία που αντιστοιχεί στην γραμμή i_1 αντιστοιχίζεται στη λειτουργία που αντιστοιχεί στη στήλη j_1. Στον Πίνακα 4.6, οι έντονες τιμές στα κελιά παρουσιάζουν την αντιστοίχηση του Σχήματος 4.6. Δηλαδή, η λειτουργία Λ1' αντιστοιχίζεται στη λειτουργία Λ5, η λειτουργία Λ2' αντιστοιχίζεται στη λειτουργία Λ1 κ.ο.κ.



Σχήμα 4.6 Δεύτερη φάση της προτεινόμενης μεθόδου

Πίνακας 4.6 Κόστη μεταξύ των λειτουργιών του Σχήματος 4.5

ΥΔ1\ΥΔ2	**Λ1**	**Λ2**	**Λ3**	**Λ4**	**Λ5**
Λ1'	2	8	14	10	**3**
Λ2'	**4**	7	9	10	12
Λ3'	5	11	**6**	7	8
Λ4'	8	22	7	**4**	11

Πίνακας 4.7 Ψευδοκώδικας της προτεινόμενης μεθόδου

```
for(i=0; i<s1.op.length; i++){
        for(j=0; j<s2.op.length; j++){
                c_input = tree_edit_distance(s1.op[i].input, s2.op[j].input);
                c_output = tree_edit_distance(s1.op[i].output, s2.op[j].output);
                cost[i,j] = c_input + c_output;
        }
}
minCost = hungarian(cost);
total_cost = 0;
for(i=0; i<minCost[0].length; i++){
        for(j=0; j<minCost[1].length; j++){
                if(minCost[i,j] == 0_star){
                        map(s1.op[i],s2.op[j]);
                        total_cost = total_cost + cost[i,j];
                }
        }
}
```

Στον Πίνακα 4.7 παρουσιάζεται ο ψευδοκώδικας για την προτεινόμενη μέθοδο. Στην πρώτη φάση της μεθόδου, καλείται για κάθε λειτουργία *i* της αρχικής υπηρεσίας *W1* και κάθε λειτουργία *j* της τροποποιημένης υπηρεσίας *W1'*, ο αλγόριθμος ανίχνευσης αλλαγών (tree edit distance) για τις εισόδους και τις εξόδους των λειτουργιών *i* και *j* και υπολογίζονται τα κόστη *c_input* και *c_output* αντίστοιχα. Το άθροισμα των

c_input και *c_output* αποθηκεύεται στην θέση [i,j] του πίνακα *cost* και αντιστοιχεί στο κόστος της μετατροπής της λειτουργίας *i* στην λειτουργία *j*. Στη δεύτερη φάση της μεθόδου, ο πίνακας *cost* δίνεται σαν είσοδος στον Ουγγρικό αλγόριθμο, ο οποίος επιστρέφει τον πίνακα *minCost*. Εάν ένα στοιχείο [i,j] του πίνακα *minCost* έχει την τιμή *0_star*, τότε η λειτουργία *i* αντιστοιχίζεται στη λειτουργία *j*. Η αντιστοίχηση αυτή υποδεικνύεται από την εντολή *map()*. Τέλος, το συνολικό κόστος της μετατροπής της υπηρεσίας *W1* στην υπηρεσία *W1'*, υπολογίζεται από το άθροισμα του επιμέρους κόστους της μετατροπής κάθε λειτουργίας *i* στην αντίστοιχή της λειτουργία *j*.

Στο περιβάλλον ανίχνευσης αλλαγών που αναπτύχθηκε, εκτός από τις μονάδες που υλοποιήθηκαν για τις δύο φάσεις της μεθόδου, υλοποιήθηκαν κάποιες επιπλέον μονάδες που σχετίζονται με την προεπεξεργασία των υπηρεσιών διαδικτύου. Συνολικά οι μονάδες που υλοποιήθηκαν είναι οι εξής:

<u>Προεπεξεργασία</u>

- Αναλυτής (Parser): Λαμβάνει σαν είσοδο μια περιγραφή WSDL και αναλύει την δομή της εισόδου και εξόδου κάθε λειτουργίας. Από την ανάλυση κάθε δομής προκύπτει ένα δένδρο. Το σύνολο των δένδρων αυτών αποτελεί την έξοδο του αναλυτή.
- Δημιουργός GML (GML Creator): Η μονάδα αυτή είναι βοηθητική και χρησιμοποιείται για την παραγωγή αρχείων GML. Η γραφική αναπαράσταση της δομής κάθε δένδρου αποτελεί μεγάλη βοήθεια κατά την αποσφαλμάτωση.

<u>Πρώτη φάση της μεθόδου</u>

- Αλγόριθμος 1 (MM-Diff): Η μονάδα αυτή λαμβάνει σαν είσοδο δύο δενδρικές δομές και υπολογίζει το κόστος της μετατροπής από τη μια στην άλλη χρησιμοποιώντας τον αλγόριθμο MM-Diff.
- Αλγόριθμος 2 (MH-Diff): Σε αντιστοιχία με τον Αλγόριθμο 1, ο Αλγόριθμος 2 συγκρίνει δύο δενδρικές δομές με τον αλγόριθμο MH-Diff.

<u>Δεύτερη φάση της μεθόδου</u>

- Ουγγρικός αλγόριθμος (Hungarian algorithm): Χρησιμοποιείται στην δεύτερη φάση της προτεινόμενης μεθόδου και υπολογίζει την αντιστοίχηση μεταξύ των λειτουργιών.

Για την αποθήκευση της εξόδου της κάθε μονάδας, αναπτύχθηκε μια βάση δεδομένων. Το περιβάλλον ανίχνευσης αλλαγών έχει υλοποιηθεί σε Microsoft Visual Web Developer 2008 Express Edition, ενώ η βάση δεδομένων είναι Microsoft SQL Server 2008.

4.2. Προεπεξεργασία δεδομένων

Το στάδιο της προεπεξεργασίας αποτελείται από δύο μονάδες. Ο αναλυτής λαμβάνει σαν είσοδο μια περιγραφή WSDL και κατασκευάζει μια δενδρική δομή για την είσοδο και έξοδο κάθε λειτουργίας. Ο δημιουργός GML εξάγει ένα GML αρχείο για κάθε δενδρική δομή.

4.2.1. Αναλυτής (Parser)

Ο αναλυτής λαμβάνει σαν είσοδο ένα αρχείο WSDL και αναλύει το κάθε τμήμα του. Όπως περιγράφτηκε στο Κεφάλαιο 2, μια υπηρεσία διαδικτύου παρέχει ένα σύνολο από λειτουργίες και κάθε λειτουργία έχει το πολύ μια είσοδο και το πολύ μια έξοδο. Ο τύπος δεδομένων κάθε εισόδου και εξόδου περιγράφεται σε ένα μήνυμα το οποίο αποτελείται από ένα ή περισσότερα τμήματα. Η δομή του κάθε τμήματος αντιστοιχίζεται σε ένα απλό (primitive type) ή πολύπλοκο τύπο δεδομένων ο οποίος περιγράφεται στο τμήμα types του αρχείου WSDL.

Ο αναλυτής μελετάει την περιγραφή WSDL και για κάθε λειτουργία, κατασκευάζει μια δενδρική δομή για την είσοδο και μια για την έξοδο. Η δενδρική δομή για την είσοδο προκύπτει ως εξής: η ρίζα του δένδρου αντιστοιχεί στο μήνυμα της εισόδου και έχει σαν όνομα το όνομα του μηνύματος. Στη συνέχεια, κάθε τμήμα του μηνύματος αντιστοιχίζεται σε έναν κόμβο που εισάγεται στο δένδρο σαν παιδί της ρίζας. Κάθε τμήμα έχει ένα μοναδικό όνομα το οποίο και αποτελεί το όνομα του αντίστοιχου κόμβου. Έπειτα, κάθε τμήμα αντιστοιχίζεται με έναν τύπο δεδομένων μέσω του γνωρίσματος element. Αν ο τύπος δεδομένων είναι απλός, τότε εισάγεται στο δένδρο ένας κόμβος με όνομα το όνομα του στοιχείου στο οποίο αντιστοιχεί. Αν δεν έχει οριστεί όνομα, τότε ο κόμβος του δένδρου επονομάζεται με τον τύπο του στοιχείου (sequence, complexType κτλ) ακολουθούμενο από έναν αύξοντα αριθμό. Ο κόμβος αυτός, όπως και κάθε κόμβος που δεν έχει παιδιά, ονομάζεται κόμβος φύλλο.

Αν ο τύπος δεδομένων αντιστοιχεί σε έναν πολύπλοκο τύπο δεδομένων, τότε προστίθεται στο δένδρο ένα υποδένδρο με ρίζα τον κόμβο που αντιστοιχεί στο τμήμα του μηνύματος και παιδιά τους κόμβους που αντιστοιχούν στα στοιχεία του πολύπλοκου τύπου δεδομένων. Η δενδρική δομή για την έξοδο προκύπτει με παρόμοιο τρόπο.

Για παράδειγμα, θεωρούμε την περιγραφή WSDL που φαίνεται Σχήμα 4.7. Στο αριστερό μέρος του σχήματος, φαίνονται τα τμήματα του αρχείου. Η υπηρεσία υποστηρίζει δύο λειτουργίες: την *echoPerson* και την *echoDocument*. Και οι δύο λειτουργίες δέχονται μια είσοδο και επιστρέφουν μια έξοδο. Όπως φαίνεται και στο Σχήμα 4.7, κάθε είσοδος και έξοδος, αντιστοιχίζεται σε ένα μήνυμα δίνοντας στο γνώρισμα *message* το όνομα του μηνύματος στο οποίο αντιστοιχεί, το οποίο είναι μοναδικό. Αντίστοιχα, κάθε τμήμα του μηνύματος, αντιστοιχίζεται με έναν τύπο δεδομένων του τμήματος τύπων μέσω του γνωρίσματος *element*. Εσωτερικά στο τμήμα τύπων, ένα στοιχείο μπορεί να αναφέρεται σε κάποιον άλλο τύπο δίνοντας το όνομά του στο γνώρισμα *type*. Στο παράδειγμά μας, δηλώνονται δύο πολύπλοκοι τύποι δεδομένων, *Person* και *Document*, τα οποία αποτελούνται από άλλους, πιο απλούς τύπους δεδομένων (double, float, string, boolean). Έτσι, τα στοιχεία *x_Person* και *result_Person* αναφέρονται στον τύπο δεδομένων *Person* και τα στοιχεία *x_Document* και *result_Document* αναφέρονται στον τύπο δεδομένων *Document*.

Ο αναλυτής μελετάει αυτή την πολύπλοκη μορφή της περιγραφής WSDL, λαμβάνοντας υπόψη του όλα τα στοιχεία που μπορεί να εμφανιστούν στο τμήμα τύπων. Τα αρχεία WSDL θα πρέπει να ακολουθούν τους κανόνες και τις προδιαγραφές της W3C (W3C validated).

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="SoapInteropCompound" targetNamespace="http://soapinterop.org/"
xmlns:wsdlns="http://soapinterop.org/" xmlns:typens="http://soapinterop.org/xsd"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/">
    <types>
        <schema targetNamespace="http://soapinterop.org/xsd"
        xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
            <complexType name="Person">
                <sequence>
                    <element minOccurs="1" maxOccurs="1" name="Age" type="double"/>
                    <element minOccurs="1" maxOccurs="1" name="ID" type="xsd:float"/>
                </sequence>
                <attribute name="Name" type="string"/>
                <attribute name="Male" type="boolean"/>
            </complexType>
            <element name="x_Person" type="typens:Person"/>
            <element name="result_Person" type="typens:Person"/>

            <complexType name="Document">
            <simpleContent>
              <extension base="string">
                <xsd:attribute name ="ID" type="string"/>
              </extension>
            </simpleContent>
            </complexType>
            <element name="x_Document" type="typens:Document"/>
            <element name="result_Document" type="typens:Document"/>
        </schema>
    </types>
    <message name="echoPerson">
        <part name="x" element="typens:x_Person"/>
    </message>
    <message name="echoPersonResponse">
        <part name="Result" element="typens:result_Person"/>
    </message>
    <message name="echoDocument">
        <part name="x" element="typens:x_Document"/>
    </message>
    <message name="echoDocumentResponse">
        <part name="Result" element="typens:result_Document"/>
    </message>
    <portType name="SoapInteropCompound1PortType">
        <operation name="echoPerson" parameterOrder="x">
            <input message="wsdlns:echoPerson"/>
            <output message="wsdlns:echoPersonResponse"/>
        </operation>
        <operation name="echoDocument" parameterOrder="x">
            <input message="wsdlns:echoDocument"/>
            <output message="wsdlns:echoDocumentResponse"/>
        </operation>
    </portType>
    <binding name="SoapInteropCompound1Binding" type="wsdlns:SoapInteropCompound1PortType">
        <scap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="echoPerson">
            <soap:operation soapAction="http://soapinterop/echoPerson"/>
            <input>
                <scap:body use="literal"/>
            </input>
            <output>
                <soap:body use="literal"/>
            </output>
        </operation>
        <operation name="echoDocument">
            <soap:operation soapAction="http://soapinterop/echoDocument"/>
            <input>
                <scap:body use="literal"/>
            </input>
            <output>
                <scap:body use="literal"/>
            </output>
        </operation>

    </binding>
    <service name="Compound1">
        <port name="SoapInteropCompound1Port" binding="wsdlns:SoapInteropCompound1Binding">
            <soap:address location="http://www.whitemesa.net/interop/r3/compound1"/>
        </port>
    </service>
</definitions>
```

Τύποι

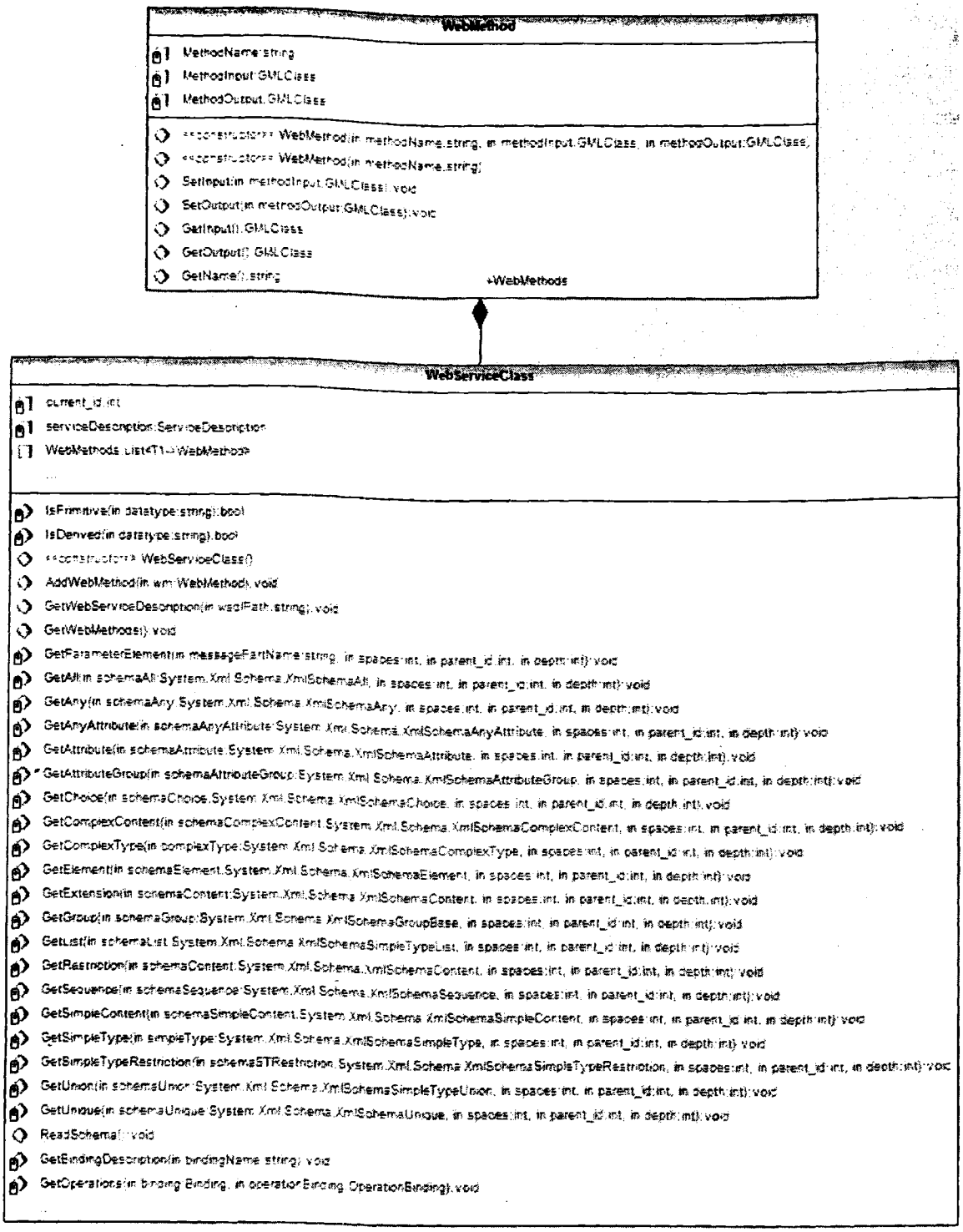
Μηνύματα

Λειτουργίες

Συνδέσεις

Υπηρεσίες

Σχήμα 4.7 Παράδειγμα WSDL Αρχείου



Σχήμα 4.8 Διάγραμμα των κλάσεων WebServiceClass και WebMethod

Για την υλοποίηση του αναλυτή, έχουν δημιουργηθεί δύο κλάσεις: η *WebServiceClass* και η *WebMethod*. Τα πεδία κάθε κλάσης και οι μέθοδοί τους, παρουσιάζονται στο Σχήμα 4.8. Η κλάση *WebServiceClass* αναλύει την WSDL περιγραφή ενός αρχείου. Αρχικά εντοπίζει το τμήμα λειτουργιών και για κάθε λειτουργία, αναλύει τα μηνύματα που χρησιμοποιεί. Κάθε μήνυμα αντιστοιχεί σε

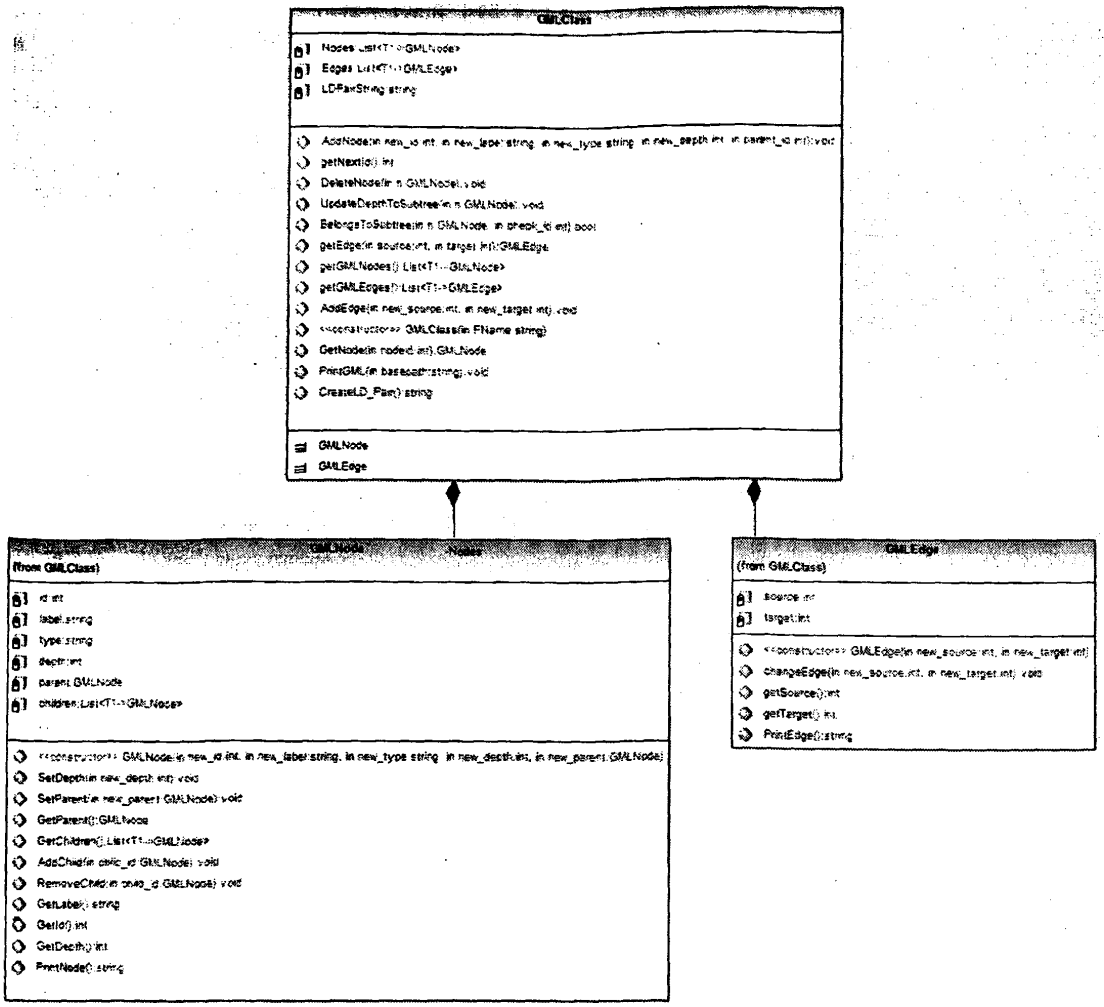
έναν τύπο δεδομένων από το τμήμα τύπων. Για κάθε στοιχείο της WSDL περιγραφής, έχει υλοποιηθεί αντίστοιχη μέθοδος που αναλύει το στοιχείο αυτό και αποθηκεύει τα απαραίτητα στοιχεία. Κάθε στιγμιότυπο της κλάσης *WebServiceClass* αποθηκεύει μια λίστα από στιγμιότυπα της κλάσης *WebMethod.* Κάθε στιγμιότυπο αντιστοιχεί σε μια λειτουργία της WSDL περιγραφής. Στην κλάση *WebMethod*, αποθηκεύονται τα απαραίτητα στοιχεία για κάθε λειτουργία.

4.2.2. Δημιουργός GML αρχείων

Για τη διευκόλυνση του χρήστη, έχει δημιουργηθεί μια βοηθητική μονάδα που αποθηκεύει τη δενδρική δομή σε GML αρχεία. Με αυτό τον τρόπο, η αναπαράσταση των δένδρων που προκύπτουν γίνεται με γραφικό τρόπο.

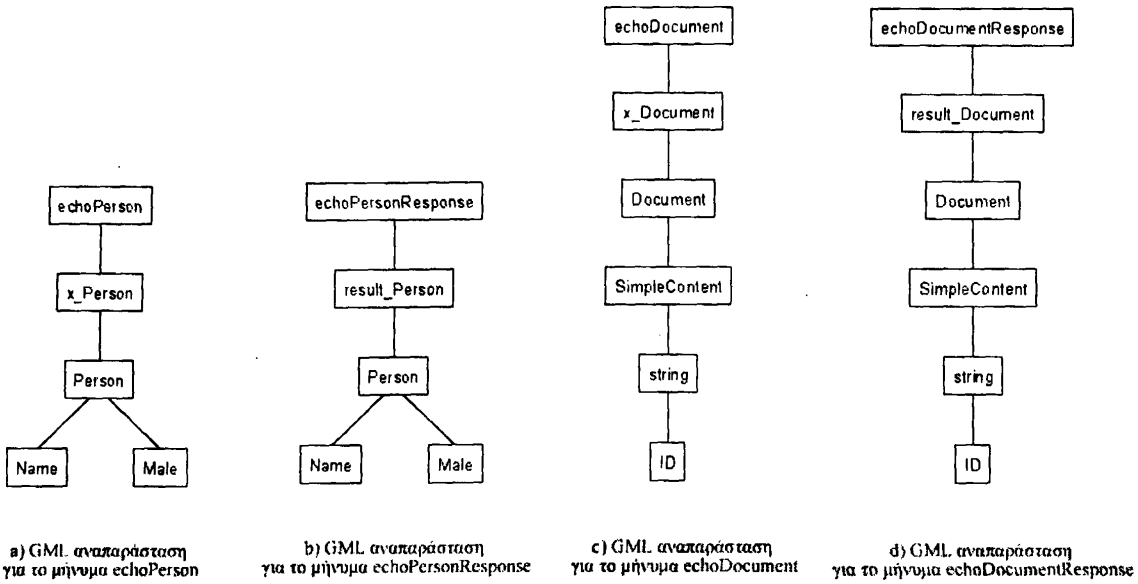
Η δομή ενός GML αρχείου είναι ιδιαίτερα απλή. Αποτελείται από ένα σύνολο αντικειμένων *node* και ένα σύνολο αντικειμένων *edge*. Το αντικείμενο *node* έχει γνωρίσματα *id*, *label* και *type*. Το *id* είναι ένας μοναδικός αριθμός που χαρακτηρίζει το *node*. Το *label* αντιπροσωπεύει το όνομα του *node* ενώ το *type* αφορά το σχήμα που θα έχει κάθε κόμβος στην γραφική αναπαράσταση. Το αντικείμενο *node* δέχεται και ένα σύνολο από άλλα γνωρίσματα, όπως τις διαστάσεις του σχήματος, τη θέση του σχήματος ως προς την αρχή των αξόνων κλπ. Το αντικείμενο *edge* αντιπροσωπεύει τις ακμές του δένδρου και έχει γνωρίσματα *source* και *target*. Το *source* αντιπροσωπεύει τον κόμβο από τον οποίο ξεκινάει η ακμή και το *target* τον κόμβο στον οποίο καταλήγει η ακμή. Οι τιμές των γνωρισμάτων αυτών αντιστοιχούν στο γνώρισμα *id* του αντίστοιχου κόμβου.

Με βάση τη δομή ενός GML αρχείου, για την υλοποίηση του δημιουργού GML αρχείων έχουν δημιουργηθεί τρεις κλάσεις: η *GMLClass* που περιέχει τις κλάσεις *GMLNode* και *GMLEdge*. Τα πεδία και οι μέθοδοι των παραπάνω κλάσεων παρουσιάζονται στο Σχήμα 4.9. Για κάθε GML αρχείο που δημιουργείται, προστίθεται ένα στιγμιότυπο της κλάσης *GMLClass*. Η κλάση αυτή περιέχει μια λίστα από στιγμιότυπα της κλάσης *GMLNode* και μια λίστα από στιγμιότυπα της κλάσης *GMLEdge*. Κάθε στιγμιότυπο της κλάσης *GMLNode* αντιστοιχεί σε έναν κόμβο του δένδρου ενώ κάθε στιγμιότυπο της κλάσης *GMLEdge* αντιστοιχεί σε μια ακμή του δένδρου.



Σχήμα 4.9 Διάγραμμα των κλάσεων GMLClass, GMLNode και GMLEdge

Συνεχίζοντας το παράδειγμα της ενότητας 4.2.1, μετά την εκτέλεση του δημιουργού GML, θα παραχθούν τέσσερα GML αρχεία. Οι γραφικές αναπαραστάσεις τους παρουσιάζονται στο Σχήμα 4.10. Παρατηρούμε ότι η ρίζα του κάθε δένδρου, αντιστοιχεί στο όνομα του μηνύματος. Τα παιδιά της ρίζας αντιστοιχούν στα τμήματα (parts) κάθε μηνύματος. Εάν τα μέρη αντιστοιχούν σε πολύπλοκους τύπους δεδομένων, τότε ακολουθεί η ιεραρχική δομή των τύπων αυτών.



a) GML αναπαράσταση για το μήνυμα echoPerson

b) GML αναπαράσταση για το μήνυμα echoPersonResponse

c) GML αναπαράσταση για το μήνυμα echoDocument

d) GML αναπαράσταση για το μήνυμα echoDocumentResponse

Σχήμα 4.10 GML αναπαραστάσεις για τα μηνύματα του παραδείγματος 4.4

4.3. Πρώτη φάση της μεθόδου

Η προτεινόμενη μέθοδος έχει σαν στόχο να ανιχνεύσει αλλαγές μεταξύ δύο υπηρεσιών διαδικτύου *WS1* και *WS2*. Για τη ανίχνευση αλλαγών μεταξύ των δενδρικών δομών, μπορεί να χρησιμοποιηθεί οποιοσδήποτε αλγόριθμος ανίχνευσης αλλαγών σε δένδρα. Στην παρούσα εργασία επιλέχθηκαν και αναπτύχθηκαν δύο αλγόριθμοι οι οποίοι περιγράφονται στις παραγράφους 4.3.1 και 4.3.2.

4.3.1. Αλγόριθμος MM-DIFF

Ο αλγόριθμος MM-DIFF [5] έχει σχεδιαστεί για επονομαζόμενα, ταξινομημένα δένδρα με ρίζα (rooted, ordered, labeled trees). Όπως περιγράφτηκε στην παράγραφο 4.1.2, τα δένδρα που προκύπτουν από τον αναλυτή είναι επονομαζόμενα δένδρα με ρίζα. Για τις ανάγκες του αλγορίθμου MM-DIFF, θεωρούμε ότι τα παιδιά ενός κόμβου ταξινομούνται με την ίδια σειρά που ταξινομούνται και τα αντίστοιχα στοιχεία στην περιγραφή WSDL.

Οι λειτουργίες τροποποίησης που μπορούν να εφαρμοστούν σε κάθε κόμβο ορίζονται ως εξής:

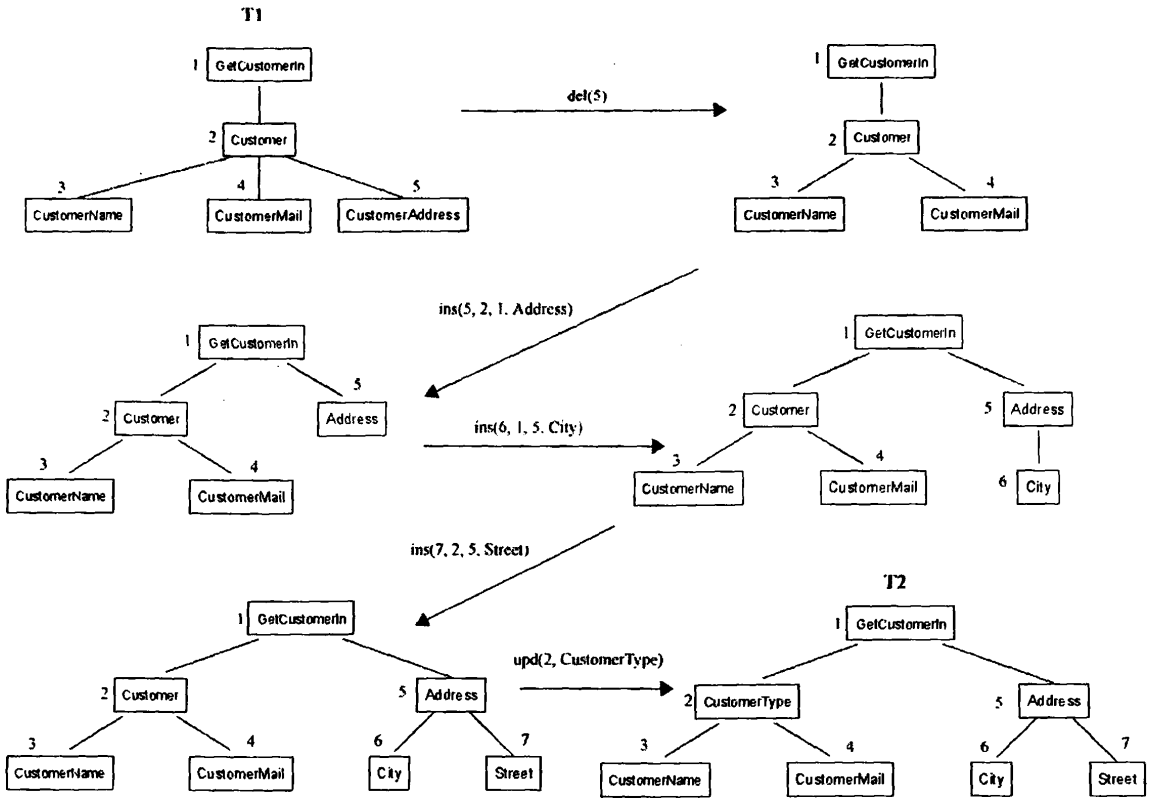
- Εισαγωγή: Έστω p ένας κόμβος στο δένδρο $T1$ και $T1_1 ... T1_k$ τα υποδένδρα του p. Έστω n ένας κόμβος που δεν ανήκει στο $T1$, l ένα αυθαίρετο όνομα και

$i \epsilon [1 \ldots k+1]$. Τότε η λειτουργία εισαγωγής *ins(n,i,p,l)* εισάγει τον κόμβο *n* σαν i-οστό παιδί του *p*. Ο κόμβος *n* είναι ένας κόμβος φύλλο με όνομα *l*.

- Διαγραφή: Έστω *n* ένας κόμβος φύλλο στο δένδρο *T1*. Η λειτουργία διαγραφής *del(n)* έχει σαν αποτέλεσμα την αφαίρεση του κόμβου *n* από το δένδρο. Πιο συγκεκριμένα, εάν ο κόμβος *n* είναι το i-οστό παιδί του κόμβου *p* με παιδιά $c_1, \ldots, c_k$, τότε στο μετασχηματισμένο δένδρο ο κόμβος *p* θα έχει παιδιά $c_1, \ldots, c_{i-1}, c_{i+1}, \ldots c_k$.
- Ανανέωση: Έστω *n* ένας κόμβος του T1 και *v* ένα όνομα. Η λειτουργία *upd(n,v)* έχει σαν αποτέλεσμα την ανανέωση του ονόματος του *n* σε *v*.

Κάθε λειτουργία τροποποίησης έχει ένα κόστος. Τα κόστη των λειτουργιών εισαγωγής, διαγραφής και ανανέωσης υπολογίζονται από τις αντίστοιχες συναρτήσεις c_i, c_d και c_u. Σημειώνουμε ότι σε αυτό τον αλγόριθμο δεν ορίζεται λειτουργία μετακίνησης. Αν κάποιο υποδένδρο μετακινηθεί στα πλαίσια της συντήρησης, ο αλγόριθμος θα υπολογίσει την αλλαγή αυτή σαν ένα σύνολο από διαγραφές όλων των κόμβων του μετακινούμενου υποδένδρου και ένα σύνολο από τις αντίστοιχες εισαγωγές στην νέα θέση του υποδένδρου.

Στο Σχήμα 4.11, παρουσιάζεται η διαδικασία μετατροπής ενός δένδρου *T1* στο δένδρο *T2*. Το δένδρο *T1* αντιστοιχεί στην είσοδο μιας λειτουργίας μιας υπηρεσίας διαδικτύου και το δένδρο *T2* αποτελεί μια παραλλαγή της εισόδου. Για κάθε βήμα της διαδικασίας μετατροπής, αναγράφεται η αντίστοιχη λειτουργία. Για παράδειγμα, στο πρώτο βήμα διαγράφεται ο κόμβος 5 και η λειτουργία που εκτελείται σημειώνεται ως *del(5)*. Στη συνέχεια, εισάγεται ένας κόμβος με αριθμό 5 σαν το δεύτερο παιδί του κόμβου 1 με όνομα *Address* και η λειτουργία σημειώνεται ως *ins(5,2,1,Address)*. Ακολουθούν δύο ακόμα εισαγωγές και τέλος το όνομα του κόμβου 2 ανανεώνεται από *Customer* σε *CustomerType*. Η λειτουργία ανανέωσης σημειώνεται ως *upd(2, CustomerType)*.



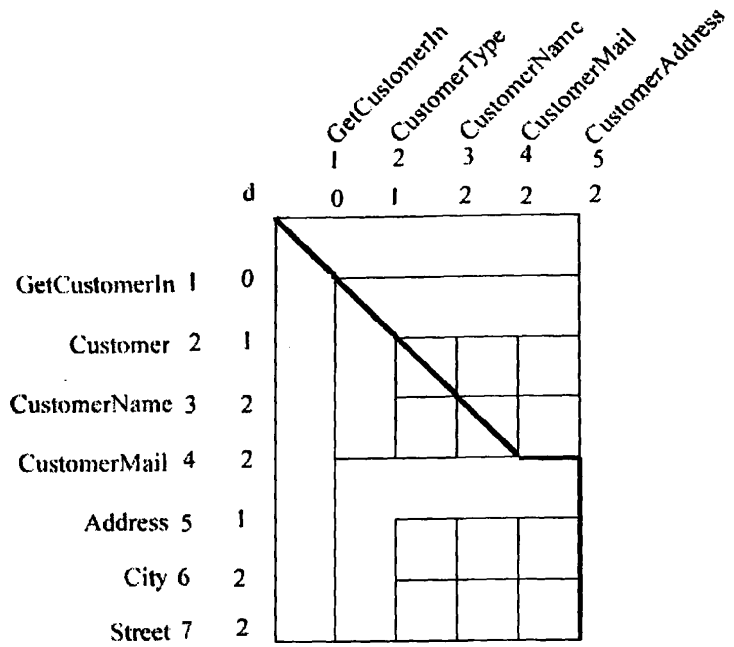
Σχήμα 4.11 Διαδικασία μετατροπής από το δένδρο T1 στο δένδρο T2

Ορίζουμε για κάθε κόμβο το ld-pair ως το ζεύγος (l,d), όπου *l* είναι το όνομα του κόμβου και *d* είναι το βάθος του κόμβου στο δένδρο. Η ld-pair αναπαράσταση ενός δένδρου, είναι η προδιατεταγμένη λίστα των ld-pair των κόμβων του δένδρου. Ο αλγόριθμος MM-DIFF υποθέτει ότι τα δένδρα βρίσκονται στη ld-pair αναπαράσταση. Για παράδειγμα στο Σχήμα 4.11, οι ld-pair αναπαραστάσεις των δένδρων *T1* και *T2* είναι:

T1 = {(1,0) (2,1) (3,2) (4,2) (5,2)}

T2 = {(1,0) (2,1) (3,2) (4,2) (5,1) (6,2) (7,2)}

Όπως περιγράφτηκε στην παράγραφο 3.2.1.1, μπορούμε να αναπαραστήσουμε τα δένδρα *T1* και *T2* σε ένα γράφημα τροποποίησης. Στην οριζόντια στήλη του γραφήματος τοποθετούνται οι κόμβοι του δένδρου *T1* σε προδιατεταγμένη διάταξη (ld-pair αναπαράσταση) και στην κάθετη στήλη, οι κόμβοι του *T2* σε προδιατεταγμένη διάταξη. Έστω ότι το δένδρο *T1* έχει *M* κόμβους και το δένδρο *T2* έχει *N* κόμβους. Τότε, το πρόβλημα της εύρεσης ενός σεναρίου τροποποίησης με ελάχιστο κόστος, επιλύεται με έναν αλγόριθμο εύρεσης της ελάχιστης διαδρομής από το σημείο (0,0) στο σημείο (M,N) του γραφήματος.



Σχήμα 4.12 Γράφημα τροποποίησης για τα δένδρα T1 και T2 του Σχήματος 4.11

Ορίζουμε το γράφημα τροποποίησης δύο δένδρων *T1* και *T2* ως ένα πλέγμα (M+1)x(N+1). Κάθε θέση (x,y) του πλέγματος αντιστοιχεί σε ένα κόμβο για xε[0...M+1] και yε[0...N+1]. Οι κόμβοι συνδέονται με ακμές ως εξής:

- Για xε[0...M-1] και yε[0...N-1], υπάρχει μια διαγώνια ακμή [(x,y), (x+1,y+1)] αν και μόνο αν $T1[x+1].d = T2[y+1].d$.
- Για xε[0...M-1] και yε[0...N], υπάρχει μια οριζόντια ακμή [(x,y), (x+1,y)] εκτός αν $y < N$ και $T2[y+1].d > T1[x+1].d$.
- Για xε[0...M] και yε[0...N-1], υπάρχει μια κάθετη ακμή [(x,y), (x,y+1)] εκτός αν $x < M$ και $T1[x+1].d > T2[y+1].d$.

Στο Σχήμα 4.12 παρουσιάζεται το γράφημα τροποποίησης των δένδρων *T1* και *T2* του Σχήματος 4.11. Οι διαγώνιες γραμμές δεν έχουν σχεδιαστεί για λόγους ευκρίνειας. Οι οριζόντιες γραμμές του γραφήματος αναπαριστούν τις διαγραφές, οι κάθετες γραμμές τις εισαγωγές και οι διαγώνιες το ταίριασμα (οι κόμβοι έχουν το ίδιο βάθος – αν έχουν διαφορετικό όνομα αναπαριστούν ανανέωση). Κάθε ακμή έχει ένα βάρος ίσο με το κόστος της αντίστοιχης λειτουργίας. Οι ακμές που απουσιάζουν από το γράφημα έχουν άπειρο κόστος. Ο αλγόριθμος MM-DIFF ψάχνει να βρει μια διαδρομή από το σημείο (0,0) στο σημείο (M,N) με το ελάχιστο κόστος. Η διαδρομή αυτή αντιστοιχίζεται στη συνέχεια σε ένα σενάριο τροποποίησης από το δένδρο *T1*

στο δένδρο *T2*. Η διαδρομή που προκύπτει από το γράφημα του Σχήματος 4.12 παρουσιάζεται με σκούρο χρώμα ενώ το σενάριο τροποποίησης που αντιστοιχεί σε αυτήν είναι: *del(5), ins(5,2,1,Address), ins(6,1,5,City), ins(7,2,5,Street), upd(2, CustomerType)*. Αν τα κόστη της διαγραφής και της εισαγωγής ενός κόμβου έχουν τιμή 3 και το κόστος της ανανέωσης είναι ίσο με 4, τότε το συνολικό κόστος του παραπάνω σεναρίου τροποποίησης είναι ίσο με 16.

Δεδομένου ενός γραφήματος τροποποίησης *G:(M+1)x(N+1)*, ορίζουμε έναν πίνακα *D* με διαστάσεις (M+1)x(N+1) τέτοιο ώστε το στοιχείο *D[x,y]* να αντικατοπτρίζει το μήκος της συντομότερης (με το ελάχιστο κόστος) διαδρομής από το σημείο (0,0) έως το σημείο (x,y) στο γράφημα τροποποίησης. Ο πίνακας *D* ονομάζεται πίνακας απόστασης του *G*. Έστω μια διαδρομή από το σημείο (0,0) στο σημείο (x,y) που αντιπροσωπεύει τον κόμβο *n*. Ο προηγούμενος κόμβος του *n* σε αυτή τη διαδρομή είναι είτε ο κόμβος στα αριστερά του *n* (σημείο (x-1,y)), είτε ο κόμβος πάνω από τον *n* (σημείο (x,y-1)) είτε ο κόμβος που βρίσκεται διαγώνια πάνω και αριστερά από τον *n* (σημείο (x-1,y-1)). Επομένως, η απόσταση του *n* από το σημείο (0,0) δεν μπορεί να είναι μεγαλύτερη από την απόσταση του προηγούμενου κόμβου από το σημείο (0,0) συν το βάρος της ακμής που συνδέει τον *n* με τον γειτονικό του κόμβο στα αριστερά στο γράφημα *G*. Αντίστοιχες σχέσεις ισχύουν και για τον γειτονικό κόμβο επάνω και τον γειτονικό κόμβο διαγώνια. Έτσι, για κάθε στοιχείο *D[x,y]*, $0<x\leq M$, $0<y\leq N$ ισχύει η παρακάτω σχέση:

D[x,y] = min{m1,m2,m3} (1) όπου,

m1 = D[x-1,y-1] + c_u*(A[x],B[y])* εάν ((x-1,y-1),(x,y))ϵG, διαφορετικά είναι άπειρο,

m2 = D[x-1,y] + c_d*(A[x])* εάν ((x-1,y),(x,y))ϵG, διαφορετικά είναι άπειρο και

m3 = D[x,y-1] + c_i*(B[y])* εάν ((x,y-1),(x,y))ϵG, διαφορετικά είναι άπειρο

Ο αλγόριθμος MM-DIFF ολοκληρώνεται σε δύο στάδια. Το πρώτο στάδιο παίρνει σαν είσοδο δύο δένδρα *T1* και *T2* σε ld-pair αναπαράσταση με *M* και *N* στοιχεία αντίστοιχα και επιστρέφει τον πίνακα απόστασης *D*. Η μέθοδος *mmDiff()*, αρχικοποιεί τη θέση [0,0] του πίνακα απόστασης *D* σε 0. Στη συνέχεια, υπολογίζει τις αποστάσεις της πρώτης γραμμής του πίνακα εφαρμόζοντας τη σχέση *m2* καθώς επίσης και τις αποστάσεις της πρώτης στήλης εφαρμόζοντας τη σχέση *m3*. Τέλος, εφαρμόζοντας τον ορισμό (1), υπολογίζει τις τιμές στα υπόλοιπα κελιά του πίνακα *D*. Στον Πίνακα 4.8 παρουσιάζεται ο κώδικας της μεθόδου *mmDiff()*.

Πίνακας 4.8 Κώδικας της μεθόδου mmDiff()

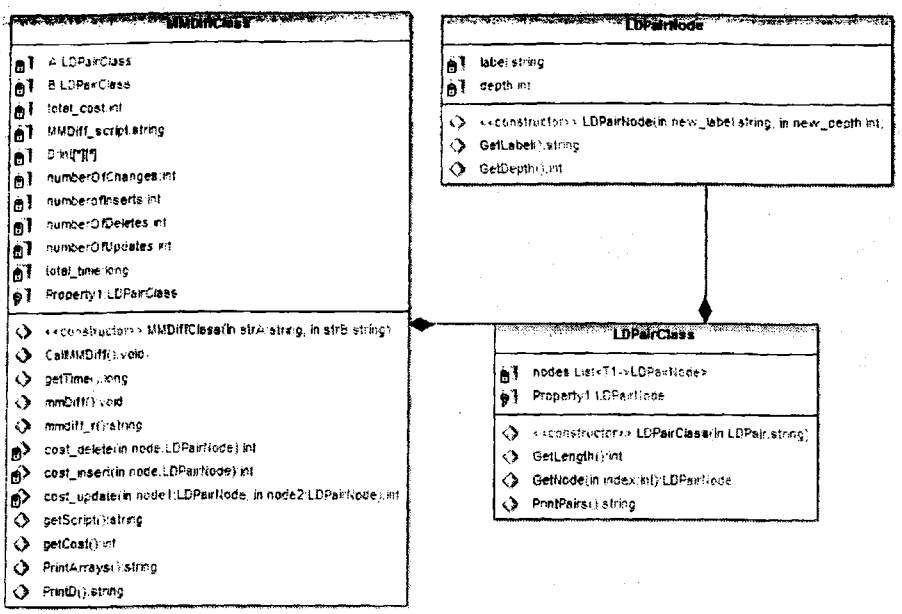
```
public void mmDiff()
{
 D[0][0] = 0;
 for (int i = 1; i <= M; i++)
 {
  D[i][0] = D[i - 1][0] + cost_delete(A.GetNode(i));
 }
 for (int j = 1; j <= N; j++)
 {
  D[0][j] = D[0][j - 1] + cost_insert(B.GetNode(j));
 }
 for (int i = 1; i <= M; i++)
 {
  for (int j = 1; j <= N; j++)
  {
     int m1 = int.MaxValue;
     int m2 = int.MaxValue;
     int m3 = int.MaxValue;

     if(A.GetNode(i).GetDepth() == B.GetNode(j).GetDepth())
     {
         m1 = D[i-1][j-1] + cost_update(A.GetNode(i), B.GetNode(j));
     }
     if((j==N)||(B.GetNode(j+1).GetDepth()<=A.GetNode(i).GetDepth()))
     {
         m2 = D[i - 1][j] + cost_delete(A.GetNode(i));
     }
     if((i==M)||(A.GetNode(i+1).GetDepth()<=B.GetNode(j).GetDepth()))
     {
         m3 = D[i][j - 1] + cost_insert(B.GetNode(j));
     }
     D[i][j] = Math.Min(Math.Min(m1, m2), m3);
  }
 }
}
```

Το δεύτερο στάδιο παίρνει σαν είσοδο δύο δένδρα *T1* και *T2* σε ld-pair αναπαράσταση με *M* και *N* στοιχεία και τον πίνακα απόστασης *D* και επιστρέφει το σενάριο τροποποίησης από το δένδρο *T1* στο δένδρο *T2*. Η μέθοδος ξεκινάει από τη θέση (M,N) του πίνακα *D* και διασχίζει αντίστροφα τον πίνακα *D* αναθέτοντας σε κάθε οριζόντια, κάθετη και διαγώνια ακμή τις λειτουργίες εισαγωγής, διαγραφής και ανανέωσης αντίστοιχα. Αθροίζοντας το κόστος κάθε λειτουργίας του σεναρίου τροποποίησης, υπολογίζουμε το συνολικό κόστος της μετατροπής. Στον Πίνακα 4.9 παρουσιάζεται ο κώδικας της μεθόδου *mmDiff_r()*.

Πίνακας 4.9 Κώδικας της μεθόδου mmDiff_r()

```
public string mmDiff_r()
{
 while (i > 0 && j > 0)
 {
   if ((D[i][j] == D[i - 1][j] + cost_delete(A.GetNode(i))) &&
      ((j == N) || (B.GetNode(j + 1).GetDepth() <= A.GetNode(i).GetDepth())))
   {
       edit_script += "del(" + A.GetNode(i).GetLabel() + "), ";
       total_cost += cost_delete(A.GetNode(i));
       i = i - 1;
    }
    else if ((D[i][j] == D[i][j - 1] + cost_insert(B.GetNode(j))) &&
            ((i == M) || (A.GetNode(i + 1).GetDepth() <= B.GetNode(j).GetDepth())))
    {
        edit_script += "ins(" + B.GetNode(j).GetLabel() + "), ";
        total_cost += cost_insert(B.GetNode(j));
        j = j - 1;
    }
    else
    {
       if (A.GetNode(i).GetLabel().CompareTo(B.GetNode(j).GetLabel()) != 0)
       {
         edit_script+="upd("+A.GetNode(i).GetLabel()+","+B.GetNode(j).GetLabel()+"),";
         total_cost += cost_update(A.GetNode(i), B.GetNode(j));
       }
       i = i - 1;
       j = j - 1;
    }
 }
 while (i > 0)
 {
     edit_script += "del(" + A.GetNode(i).GetLabel() + "), ";
     total_cost += cost_delete(A.GetNode(i));
     i = i - 1;
 }
 while (j > 0)
 {
     edit_script += "ins(" + B.GetNode(j).GetLabel() + "), ";
     total_cost += cost_insert(B.GetNode(j));
     j = j - 1;
 }
 return edit_script;
}
```

Στο Σχήμα 4.13 παρουσιάζεται το διάγραμμα κλάσεων για τον αλγόριθμο MM-DIFF. Για κάθε σύγκριση μεταξύ δύο δένδρων, δημιουργείται ένα στιγμιότυπο της κλάσης *MMDiffClass*. Τα δένδρα βρίσκονται σε ld-pair αναπαράσταση και κάθε δένδρο που παίρνει σαν είσοδο η κλάση *MMDiffClass*, αποθηκεύεται σε ένα στιγμιότυπο της κλάσης *LDPairClass*. Κάθε στιγμιότυπο της κλάσης *LDPairNode* αντιστοιχεί σε έναν κόμβο του δένδρου σε ld-pair αναπαράσταση. Οι πιο βασικές μέθοδοι που διαθέτει είναι οι *mmDiff()* και η *mmDiff_r()* που υπολογίζουν το σενάριο τροποποίησης.



Σχήμα 4.13 Διάγραμμα των κλάσεων MMDiffClass, LDPairClass και LDPairNode

4.3.2. Αλγόριθμος MH-DIFF

Ο αλγόριθμος MH-DIFF έχει σχεδιαστεί για επονομαζόμενα, αταξινόμητα δένδρα με ρίζα (rooted, unordered, labeled trees). Οι λειτουργίες τροποποίησης που μπορούν να εφαρμοστούν σε κάθε κόμβο ορίζονται ως εξής:

- Εισαγωγή: Η διαδικασία εισαγωγής δημιουργεί έναν καινούργιο κόμβο *n* με όνομα *v*, ο οποίος τοποθετείται σαν παιδί ενός κόμβου *p*. Οι κόμβοι που ήταν παιδιά του *p*, γίνονται παιδιά του *n*.
- Διαγραφή: Κατά τη διαδικασία διαγραφής ενός κόμβου *n*, όλα τα παιδιά του *n* γίνονται παιδιά του *p*, πατέρα του *n* και ο κόμβος *n* διαγράφεται.
- Ανανέωση: Κατά τη διαδικασία ανανέωσης το όνομα του *n* αλλάζει σε *v*.
- Μετακίνηση: Η διαδικασία μετακίνησης ενός κόμβου *n* στον κόμβο *p* συνεπάγεται την μετακίνηση όλου του υποδένδρου με ρίζα το *n* στον κόμβο *p*. Ο κόμβος *p*, δηλαδή, γίνεται πατέρας του *n*. Η ρίζα δεν μπορεί να μετακινηθεί.
- Αντιγραφή: Η διαδικασία αντιγραφής ενός κόμβου *m* στον κόμβο *p* συνεπάγεται την αντιγραφή όλου του υποδένδρου με ρίζα το *m* στον κόμβο *p*. Η ρίζα δεν μπορεί να αντιγραφτεί.

- Συνένωση: Η διαδικασία συνένωσης είναι αντίθετη από την διαδικασία αντιγραφής. Δεδομένων δύο κόμβων *n1* και *n2* τέτοιων ώστε τα υποδένδρα με ρίζα τα *n1* και *n2* να είναι ισομορφικά, η διαδικασία συνένωσης συνεπάγεται την εξαφάνιση του υποδένδρου με ρίζα το *n1*. Διαισθητικά, ενώνεται με το υποδένδρο με ρίζα το *n2*. Η ρίζα δεν μπορεί να συνενωθεί. Η διαδικασία της συνένωσης ορίζεται για λόγους συμμετρίας εφόσον είναι αντίθετη της διαδικασίας αντιγραφής. Με αυτό τον τρόπο, ένα σενάριο τροποποίησης από ένα δένδρο *T1* σε ένα δένδρο *T2* μπορεί να αντιστραφεί ώστε να μετατρέπει το δένδρο *T2* στο δένδρο *T1*.

Κάθε λειτουργία τροποποίησης συνεπάγεται ένα κόστος κατά την μετατροπή ενός δένδρου *T1* σε ένα δένδρο *T2*. Τα κόστη εισαγωγής, διαγραφής, ανανέωσης, μετακίνησης, αντιγραφής και συνένωσης δίνονται από τις σταθερές c_i, c_d, c_u, c_m, c_c και c_g αντίστοιχα.

4.3.3. Γενική επισκόπηση

Ο αλγόριθμος ακολουθεί μια σειρά από βήματα ξεκινώντας από τα δύο δένδρα *T1* και *T2* και καταλήγοντας στην εξαγωγή ενός σεναρίου μετατροπής του δένδρου *T1* στο δένδρο *T2*. Το πρώτο βήμα, ο κατασκευαστής διμερή γράφου (induced graph builder), παίρνει σαν είσοδο τα δένδρα *T1* και *T2* και παράγει έναν διμερή γράφο *B* (bipartite graph) με τους κόμβους του *T1* στο ένα μέρος και τους κόμβους του *T2* στο άλλο μέρος και με δύο επιπλέον ειδικούς κόμβους + (στην πλευρά του *T1*) και – (στην πλευρά του *T2*). Το δεύτερο βήμα, ο κλαδευτής (pruner), παίρνει σαν είσοδο τον γράφο *B*, αφαιρεί (κλαδεύει) κάποιες ακμές εφαρμόζοντας κάποιους κανόνες κλαδέματος και επιστρέφει τον κλαδεμένο γράφο *B*. Για την εφαρμογή των κανόνων κλαδέματος, γίνεται ένας υπολογισμός του μέγιστου και ελάχιστου κόστους της κάθε ακμής. Έτσι, στον κλαδεμένο γράφο *B*, σε κάθε ακμή αντιστοιχεί ένα βάρος.

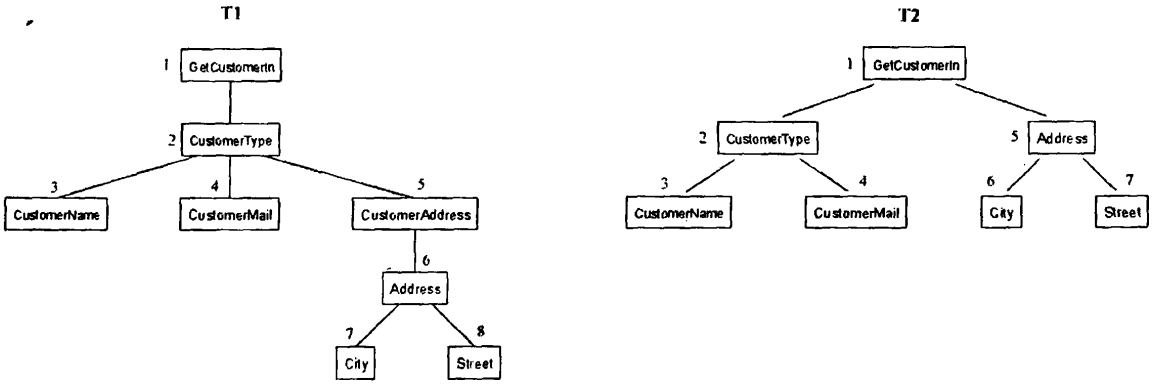
Το επόμενο βήμα του αλγορίθμου είναι η εύρεση των ακμών που καλύπτουν όλο το δένδρο έτσι ώστε κάθε κόμβος του δένδρου *T1* να συνδέεται με έναν μόνο κόμβο του *T2* και οι ακμές αυτές να έχουν το ελάχιστο δυνατό κόστος (minimum-cost edge cover). Το πρόβλημα αυτό ανάγεται στην επίλυση ενός προβλήματος ταιριάσματος βαρών (weighted matching problem) [15]. Ο αλγόριθμος που χρησιμοποιείται για την επίλυση αυτού του προβλήματος ονομάζεται Ουγγρικός αλγόριθμος. Το τελευταίο

βήμα του αλγορίθμου είναι η εξαγωγή του σεναρίου τροποποίησης, η οποία γίνεται με την επισημείωση των ακμών που παρέμειναν μετά την εκτέλεση του Ουγγρικού αλγόριθμου, με την κατάλληλη λειτουργία τροποποίησης.

Στις επόμενες ενότητες, θα γίνει ανάλυση του κάθε βήματος του αλγορίθμου MH-DIFF.

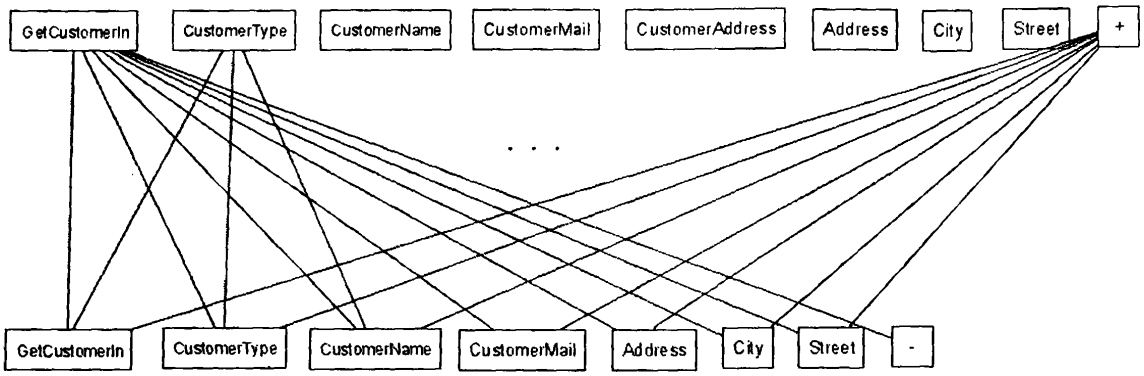
4.3.4. Κατασκευή Διμερή Γράφου

Έστω δύο δένδρα *T1* και *T2* και δύο ειδικοί τύποι κόμβων + και -. Ένας διμερής γράφος *B* απαρτίζεται από την συνένωση των δύο δένδρων με τέτοιο τρόπο ώστε κάθε κόμβος του δένδρου *T1* να συνδέεται με μια ακμή με κάθε κόμβο του δένδρου *T2*. Επιπλέον, κάθε κόμβος του *T1* συνδέεται με μια ακμή με τον ειδικό κόμβο – και κάθε κόμβος του δένδρου *T2* συνδέεται με τον ειδικό κόμβο +. Κατά την εκτέλεση όλων των βημάτων του αλγορίθμου MH-DIFF, η εσωτερική δομή του κάθε δένδρου (σχέση πατέρα – παιδιών) παραμένει αναλλοίωτη. Ο αλγόριθμος διαχειρίζεται μόνο τις ακμές που υπάρχουν μεταξύ των δύο δένδρων.



Σχήμα 4.14 Παράδειγμα εισόδου στον αλγόριθμο MH-DIFF

Θεωρούμε τα δένδρα *T1* και *T2* του Σχήματος 4.14. Ο διμερής γράφος που προκύπτει από την συνένωσή τους απεικονίζεται στο Σχήμα 4.15. Επειδή ο αριθμός των ακμών είναι εκθετικός ως προς τον αριθμό των κόμβων των δύο δένδρων, στο Σχήμα 4.15 δεν παρουσιάζονται όλες οι ακμές.



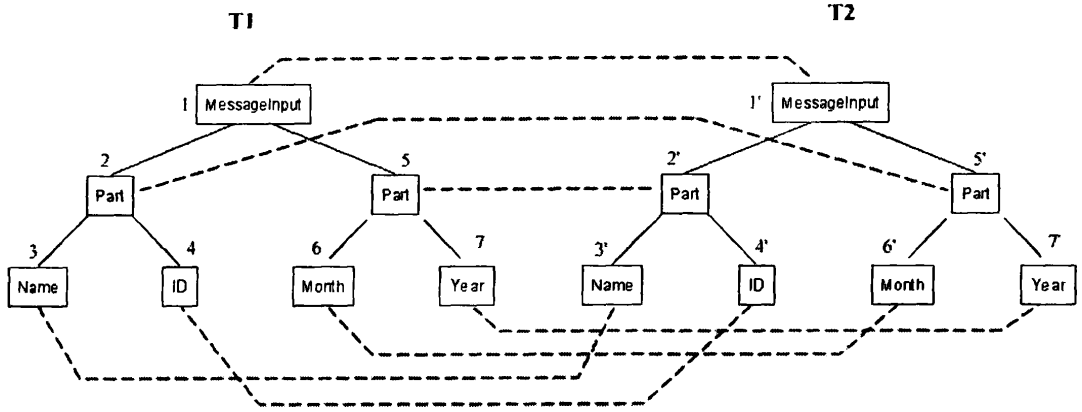
Σχήμα 4.15 Διμερής γράφος των δένδρων Τ1 και Τ2 του Σχήματος 4.14

4.3.5. Κλαδευτής

Από το Σχήμα 4.15, είναι φανερό ότι ο διμερής γράφος αποτελείται από έναν μεγάλο αριθμό ακμών ο οποίος αυξάνεται εκθετικά ως προς τον αριθμό των κόμβων. Πολλές από τις ακμές αυτές είναι ανεπιθύμητες. Για να αποφασίσουμε ποιες ακμές μπορούμε να αφαιρέσουμε από τον γράφο πρέπει να γνωρίζουμε το κόστος της κάθε ακμής και να αφαιρέσουμε εκείνες τις ακμές που έχουν το μεγαλύτερο κόστος. Όμως το κόστος της κάθε ακμής, εξαρτάται από το τελικό σύνολο των ακμών (σύνολο ακμών – στόχος). Μόνο τότε μπορούμε να γνωρίζουμε την λειτουργία που αντιστοιχήθηκε σε κάθε ακμή και επομένως να γνωρίζουμε το κόστος της. Το τελικό σύνολο των ακμών, όμως, ολοκληρώνεται στα επόμενα βήματα του αλγορίθμου. Για να βγούμε από αυτό το αδιέξοδο, γίνεται για κάθε ακμή ένας προσεγγιστικός υπολογισμός του άνω και κάτω ορίου του κόστους που μπορεί να έχει η ακμή αυτή.

Θεωρούμε τα δένδρα *T1* και *T2* του Σχήματος 4.16. Παρατηρούμε ότι οι κόμβοι 2 και 5 του δένδρου *T1* έχουν το ίδιο όνομα, ενώ και οι δύο κόμβοι έχουν δύο παιδιά, τους κόμβους 3, 4 και 6, 7 αντίστοιχα. Το δένδρο *T2* είναι ισομορφικό του δένδρου *T1*. Το ταίριασμα μεταξύ των δύο δένδρων υποδεικνύεται από τις διακεκομένες ακμές. Παρατηρούμε ότι ο κόμβος 2 του δένδρου *T1* έχει συνδυαστεί με τον κόμβο 5' του δένδρου *T2* και ο κόμβος 5 του δένδρου *T1* με τον κόμβο 2' του δένδρου *T2*. Κάτι τέτοιο είναι πιθανό αφού οι κόμβοι 2 και 5 έχουν το ίδιο όνομα. Το σενάριο τροποποίησης που θα παραχθεί με βάση τους παραπάνω συνδυασμούς, θα περιέχει μια λειτουργία μετακίνησης για κάθε έναν από τους κόμβους 3, 4, 6 και 7. Αυτό συνέβη επειδή οι κόμβοι 2 και 5 του δένδρου *T1* συνδυάστηκαν με τους κόμβους 5' και 2' του δένδρου *T2* αντίστοιχα και όχι με τους κόμβους 2' και 5'. Συμπεραίνουμε,

λοιπόν, ότι οι λειτουργίες μετακίνησης των κόμβων 3, 4, 6 και 7 παράχθηκαν εξαιτίας του αποτυχημένου συνδυασμού των κόμβων 2 και 5. Για να αποφύγουμε τέτοιους αποτυχημένους συνδυασμούς, θα «χρεώσουμε» τις λειτουργίες μετακίνησης των παιδιών 3, 4 και 6, 7 των κόμβων 2 και 5 αντίστοιχα στις ακμές [2,5'] και [5,2'] που είναι υπεύθυνες για τις επιπλέον λειτουργίες μετακίνησης. Με αυτό τον τρόπο, τα άνω και κάτω όρια των ακμών αυτών, θα λαμβάνουν υπόψη τους τις επιπλέον λειτουργίες τροποποίησης που μπορεί να δημιουργήσουν.



Σχήμα 4.16 Δίκαιος καταμερισμός στα άνω και κάτω όρια των ακμών

Καταλήγουμε, λοιπόν, στο εξής συμπέρασμα: αν μια ακμή *e* υποδεικνύει λειτουργία εισαγωγής, διαγραφής ή ενημέρωσης, «χρεώνουμε» την ακμή *e* με το κόστος της αντίστοιχης λειτουργίας. Αν μια ακμή *e* υποδεικνύει μετακίνηση, αντιγραφή ή συνένωση, «χρεώνουμε» την ακμή *e'* του πατρικού κόμβου και όχι την ακμή *e*.

Έστω *K* το σύνολο ακμών στόχος. Τότε, *E(m)* είναι το σύνολο των ακμών που περνούν από τον κόμβο *m* και ανήκουν στο σύνολο *K* και *C(m)* είναι το σύνολο των παιδιών του *m*. Ορίζουμε το κόστος της εξαναγκασμένης μετακίνησης (forced move) $c_{mf}(m, n)$ ενός κόμβου *m* που ανήκει στο *T1* και ενός κόμβου *n* που ανήκει στο *T2* ίσο με: $c_{mf}(m,n)=c_m$ εάν δεν υπάρχει κόμβος *n'* που ανήκει στο *C(n)* τέτοιος ώστε η ακμή [m,n'] να ανήκει στο *E*, διαφορετικά $c_{mf}(m,n) = 0$.

Λαμβάνοντας υπόψη ότι κάθε πατρικός κόμβος πρέπει να «χρεωθεί» με το κόστος της εξαναγκασμένης μετακίνησης των παιδιών του, ορίζουμε το κάτω όριο c_{lb} μιας ακμής ως εξής:

$$c_{lb}([m,n]) = c_u(m,n) + \frac{1}{2}\min\left\{\sum_{m'\in C(m)} c_{mf}(m',n), \sum_{n'\in C(n)} c_{mf}(m,n')\right\}$$

Ορίζουμε το κόστος της μετακίνησης υπό συνθήκη (conditional move) $c_{m?}(m, n)$ ενός κόμβου *m* που ανήκει στο *T1* και ενός κόμβου *n* που ανήκει στο *T2* ως εξής: $c_{m?}(m,n) = c_m$ εάν υπάρχει κόμβος *n'* που ανήκει στο *C(n)* τέτοιος ώστε η ακμή [m,n'] να ανήκει στο *E*, διαφορετικά $c_{m?}(m,n) = 0$. Ακόμα, ορίζουμε το κόστος $c_w(m,n)$ ως εξής: $c_w(m,n) = c_u(m,n)$ εάν οι κόμβοι *m* και *n* είναι κανονικοί, $c_w(m,n) = 0$ εάν ο κόμβος *m* είναι ο ειδικός κόμβος + και ο κόμβος *n* είναι ο ειδικός κόμβος -, $c_w(m,n) = c_i$ εάν ο κόμβος *m* είναι ο ειδικός κόμβος + και ο κόμβος *n* είναι κανονικός και $c_w(m,n) = c_d$ εάν ο κόμβος *m* είναι κανονικός και ο κόμβος *n* είναι ο ειδικός κόμβος -.

Το άνω όριο c_{ub} μιας ακμής ορίζεται ως εξής:

$$c_{ub}([m,n]) = c_w(m,n) + \frac{1}{2}\sum_{m' \in C(m)} (c_c(|E(m')|-1) + c_{m?}(m',n)) + \frac{1}{2}\sum_{n' \in C(n)} (c_g(|E(n')|-1) + c_{n?}(n',m))$$

Με βάση τους παραπάνω ορισμούς, παρατηρούμε ότι μια ακμή «χρεώνεται» τουλάχιστον (κάτω όριο) με το κόστος της ανανέωσης συν το κόστος της εξαναγκασμένης μετακίνησης των παιδιών του κόμβου στον οποίο αντιστοιχεί και το πολύ (άνω όριο) με το κόστος της εισαγωγής ή διαγραφής συν το κόστος της μετακίνησης υπό συνθήκη των παιδιών του κόμβου στον οποίο αντιστοιχεί.

Έχοντας ορίσει τα άνω και κάτω όρια του κόστους της κάθε ακμής, εφαρμόζουμε τους ακόλουθους κανόνες κλαδέματος:

Κανόνας 1: Έστω $e_1 = [m,n]$ μια ακμή του διμερή γράφου, e_2 μια οποιαδήποτε ακμή που περνάει από τον κόμβο *m* και e_3 μια οποιαδήποτε ακμή που περνάει από τον κόμβο *n*. Έστω $C_t = max\{c_m, c_c, c_g\}$. Εάν ισχύει η ανισότητα: $c_{lb}(e_1) >= c_{ub}(e_2) + c_{ub}(e_3) + 2C_t$, τότε η ακμή e_1 αφαιρείται. Εφαρμόζοντας, δηλαδή, τον κανόνα αυτό, αφαιρούνται ακμές που το ελάχιστο κόστος τους είναι τόσο μεγάλο, ώστε είναι προτιμότερο να χρησιμοποιηθούν δύο διαφορετικές ακμές που περνάνε από τους κόμβους *m* και *n* αντίστοιχα, αν έχουν μικρό άνω όριο κόστους.

Κανόνας 2: Έστω $e_1 = [m,n]$ μια ακμή του διμερή γράφου. Εάν ισχύει η ανισότητα: $c_{lb}(e_1) >= c_d(m) + c_i(n)$, τότε η ακμή e_1 αφαιρείται. Εάν, δηλαδή, το κόστος του «ταιριάσματος» δύο κόμβων είναι μεγαλύτερο από την διαγραφή ενός κόμβου και την εισαγωγή του άλλου, τότε δεν είναι προτιμότερο να γίνει το «ταίριασμα».

Η εφαρμογή των κανόνων κλαδέματος γίνεται επαναληπτικά μέχρι το σημείο που δεν αφαιρούνται άλλες ακμές. Αυτό γίνεται γιατί μετά από κάθε εφαρμογή, ξαναγίνεται υπολογισμός των άνω και κάτω ορίων του κόστους της κάθε ακμής. Λόγω της

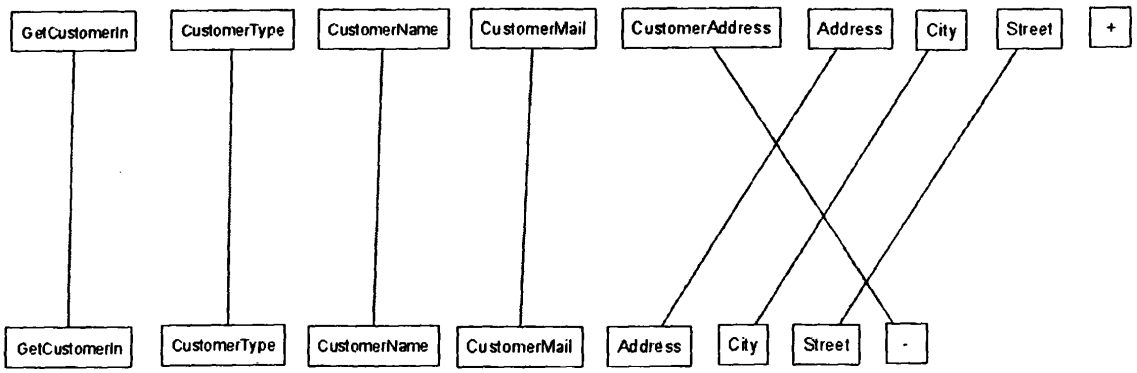
αφαίρεσης ορισμένων ακμών, είναι πιθανό τα κάτω όρια να αυξάνονται και τα άνω όρια να μειώνονται προκαλώντας την αφαίρεση επιπλέον ακμών.

4.3.6. Εύρεση ακμών με το ελάχιστο δυνατό κόστος

Μετά την εφαρμογή των κανόνων κλαδέματος, έχουμε έναν κλαδεμένο διμερή γράφο με λιγότερες ακμές από τον αρχικό. Η ιδανική περίπτωση είναι ένας γράφος με μόνο τις απαραίτητες ακμές με το ελάχιστο κόστος. Είναι πιθανό όμως να υπάρχουν περισσότερες ακμές. Για αυτό το λόγο, αφού ο γράφος κλαδευτεί, προσεγγίζουμε το κόστος της κάθε ακμής εξισώνοντας το με το κάτω όριο. Στη συνέχεια, το πρόβλημα της εύρεσης των ελάχιστων δυνατών ακμών με το ελάχιστο συνολικό κόστος ανάγεται στο πρόβλημα ταιριάσματος διμερή γράφου με βάρη (bipartite weighted matching problem). Η μέθοδος που χρησιμοποιείται για την επίλυση τέτοιων προβλημάτων, ονομάζεται Ουγγρική (Hungarian) [15] και περιγράφεται αναλυτικά στην παράγραφο 4.4.1.

Στο περιβάλλον ανίχνευσης αλλαγών που αναπτύχθηκε, η μέθοδος παίρνει σαν είσοδο έναν πίνακα nxn. Οι γραμμές του πίνακα αντιπροσωπεύουν τους κόμβους του δένδρου *T1* και οι στήλες τους κόμβους του *T2*. Κάθε στοιχείο [i,j] του πίνακα, έχει τιμή το κόστος της ακμής του κόμβου της θέσης *i* με τον κόμβο της θέσης *j*. Η έξοδος του αλγορίθμου, είναι ένας πίνακας όπου σε κάθε γραμμή υπάρχει ένα μόνο κελί με την τιμή του κόστους της ακμής και όλα τα υπόλοιπα κελιά έχουν τιμή το άπειρο. Έτσι, ο τελικός γράφος, περιέχει μόνο τις ακμές μεταξύ των κόμβων [i,j] που έχουν πραγματική τιμή. Οι υπόλοιπες ακμές αφαιρούνται από τον τελικό γράφο.

Μετά την εφαρμογή του Ουγγρικού αλγόριθμου στο διμερή γράφο του Σχήματος 4.15, παίρνουμε τον γράφο του Σχήματος 4.17. Παρατηρούμε ότι κάθε κόμβος του *T2*, συνδέεται μόνο με ένα κόμβο του *T1*, επομένως έχει βρεθεί το ελάχιστο σύνολο ακμών μεταξύ των *T1* και *T2* που καλύπτει όλο το γράφο.



Σχήμα 4.17 Γράφος που προκύπτει μετά την εφαρμογή του Ουγγρικού αλγορίθμου στο γράφο του Σχήματος 4.15

4.3.7. Εξαγωγή σεναρίου τροποποίησης

Μετά την εκτέλεση του Ουγγρικού αλγορίθμου, το σύνολο των ακμών μεταξύ των δένδρων *T1* και *T2* είναι ελάχιστο. Αν τα δένδρα έχουν τον ίδιο αριθμό κόμβων *N*, τότε υπάρχει μια ένα-προς-ένα αντιστοιχία μεταξύ των κόμβων των δύο δένδρων. Αν το δένδρο *T1* έχει περισσότερους κόμβους από το *T2*, τότε από κάθε κόμβο του *T2* περνάει μόνο μια ακμή, ενώ το αντίθετο συμβαίνει αν το δένδρο *T2* έχει περισσότερους κόμβους.

Έστω *K* το ελάχιστο σύνολο ακμών. Τότε, η εξαγωγή του σεναρίου τροποποίησης γίνεται με την ανάθεση μιας λειτουργίας τροποποίησης σε κάθε ακμή και με βάση κάποιους περιορισμούς, γίνεται σειριοποίηση των λειτουργιών και εξάγεται το σενάριο τροποποίησης. Το σύνολο A^+ = *{NIL, INS, DEL, UPD, MOV, CPY, GLU, MOV.UPD, CPY.UPD, GLU.UPD}* αναπαριστά τις λειτουργίες που μπορούν να ανατεθούν σε μια ακμή. Η ανάθεση *NIL* γίνεται όταν δύο κόμβοι «ταίριαξαν» τέλεια και δεν χρειάζεται να γίνει καμία αλλαγή, για παράδειγμα επειδή βρίσκονται στην ίδια θέση του γράφου και έχουν τα ίδια ονόματα. Οι διπλές λειτουργίες επισημαίνουν ότι εκτός από την λειτουργία *MOV*, *CPY* ή *GLU* που έχει ανατεθεί, είναι απαραίτητη και η ανανέωση του ονόματος του κόμβου (*UPD*).

Ο στόχος του σεναρίου τροποποίησης είναι να βοηθήσει τον χρήστη να μετατρέψει το δένδρο *T1* στο δένδρο *T2* χωρίς να χάσει κάποια σημαντική πληροφορία. Παρ' όλα αυτά, υπάρχουν κάποιες περιπτώσεις που κάτι τέτοιο δεν ισχύει. Έστω ένα σενάριο τροποποίησης *Σ1* που εισάγει έναν κόμβο *p* σαν πατέρα δέκα παιδιών, $n_1 ... n_{10}$. Στη συνέχεια ο κόμβος *p* μετακινείται σε κάποια άλλη θέση του δένδρου και στο τέλος

διαγράφεται. Το κόστος του *Σ1* θα είναι μικρότερο από κάποιο άλλο σενάριο *Σ2* στο οποίο οι κόμβοι $n_1 ... n_{10}$ μετακινήθηκαν ένας-ένας και όχι σαν ομάδα με μια λειτουργία μετακίνησης. Όμως, ο κόμβος *p* δεν υπάρχει ούτε στο δένδρο *T1* ούτε στο δένδρο *T2*, επομένως έχει χαθεί κάποια πληροφορία. Αυτό έχει σαν αποτέλεσμα το σενάριο τροποποίησης να μην έχει νόημα στον χρήστη, αφού θα έπρεπε να εισάγει έναν κόμβο και στη συνέχεια να τον διαγράψει. Για αυτό το λόγο, τα σενάρια τροποποίησης πρέπει να διατηρούν τις εξής ιδιότητες:

- Ιδιότητα 1: Σε κάθε κόμβο μπορεί να ανατεθεί το πολύ μια λειτουργία τροποποίησης που αλλάζει τη δομή του δένδρου (*INS, DEL, MOV, CPY, GLU*). Με αυτό τον τρόπο, εάν ένας κόμβος εισαχθεί, δεν μπορεί στη συνέχεια να διαγραφεί, να μετακινηθεί, να αντιγραφεί ή να ενωθεί. Αυτός ο περιορισμός ισχύει μόνο για τους κόμβους στους οποίους έχει ανατεθεί άμεσα μια τέτοια λειτουργία. Για παράδειγμα, εάν κάποιος κόμβος μετακινήθηκε έμμεσα λόγω της μετακίνησης κάποιου προγόνου του, μπορεί στη συνέχεια να μετακινηθεί, να διαγραφεί κλπ.
- Ιδιότητα 2: Ένας κόμβος που είναι είτε η πηγή είτε ο στόχος μιας έμμεσης αντιγραφής, δηλαδή συμμετέχει σε μια λειτουργία αντιγραφής λόγω κάποιου προγόνου του, δεν μπορεί στη συνέχεια να συμμετέχει σε μια λειτουργία ένωσης, άμεσα ή έμμεσα.

Μέθοδος Ανάθεσης: Έστω T_1^+ το σύνολο των κόμβων του δένδρου *T1* και ο ειδικός κόμβος + και T_2^+ το σύνολο των κόμβων του *T2* και ο ειδικός κόμβος -. Για κάθε ακμή $e = [m,n]$ που ανήκει στο σύνολο *K*, διακρίνουμε τις εξής περιπτώσεις:

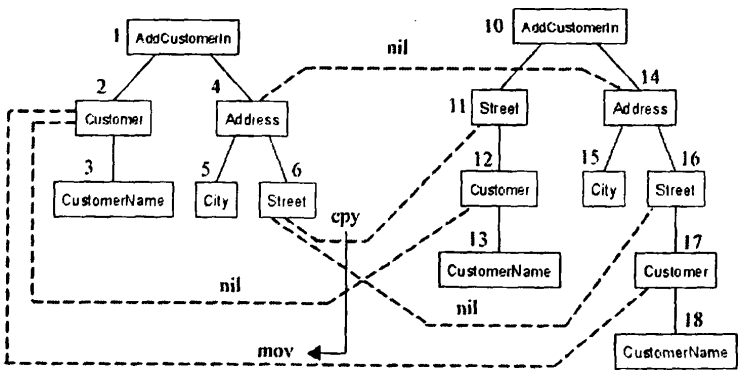
Περίπτωση 1: Ο βαθμός των κόμβων *m* και *n* είναι 1, άρα δεν υπάρχει άλλη ακμή που περνάει από αυτούς τους κόμβους. Τότε, αν *m* είναι ο ειδικός κόμβος +, ο κόμβος *n* πρέπει να εισαχθεί και στην ακμή *e* ανατίθεται η λειτουργία εισαγωγής. Αν ο *n* είναι ο ειδικός κόμβος -, ο κόμβος *n* διαγράφεται και στην ακμή *e* ανατίθεται η λειτουργία διαγραφής. Αν οι κόμβοι *m* και *n* ανήκουν στα δένδρα *T1* και *T2* αντίστοιχα, τότε εάν δεν υπάρχει ακμή στο *K* που συνδέει τους πατέρες των κόμβων *m* και *n*, ο κόμβος *m* πρέπει να μετακινηθεί στη σωστή θέση του δένδρου. Επομένως, ανατίθεται στην ακμή *e* η λειτουργία μετακίνησης. Διαφορετικά, δεν χρειάζεται καμία αλλαγή οπότε η ακμή σημειώνεται με *NIL*.

Περίπτωση 2: Ο βαθμός του *n* είναι 1 και ο βαθμός του *m* είναι μεγαλύτερος από 1, δηλαδή υπάρχουν περισσότερες από μια ακμές που περνάνε από τον κόμβο *m*.

Υποπερίπτωση 2.1: Ο κόμβος *m* δεν είναι ο ειδικός κόμβος +. Είναι προφανές ότι όλοι οι κόμβοι που ταιριάζουν με τον *m* εκτός από έναν έχουν προέλθει από λειτουργίες αντιγραφής. Επειδή κατά την αντιγραφή ενός υποδένδρου με ρίζα τον *n1*, αντιγράφονται χωρίς κόστος όλοι κόμβοι του υποδένδρου, πρέπει μόνο ο κόμβος *n1* να αντιγραφεί άμεσα (να του ανατεθεί η λειτουργία αντιγραφής) και οι υπόλοιποι κόμβοι του υποδένδρου να αντιγραφούν έμμεσα (να μην τους ανατεθεί καμία λειτουργία).

Μια άλλη παρατήρηση είναι η εξής: Έστω *n* και *n'* δύο κόμβοι του δένδρου *T2* που ταιριάζουν με τον κόμβο *m*. Εάν ούτε ο *n* ούτε ο *n'* αντιγράφεται άμεσα, τότε είτε ο πατέρας του *m*, *p(m)* ταιριάζει με δύο ή περισσότερους κόμβους του *T2*, είτε ο πατέρας του *n*, *p(n)* και ο πατέρας του *n'*, *p(n')* ταιριάζουν με τον ίδιο κόμβο *m'* του *T1*, είτε ισχύουν και τα δύο. Εάν m' = p(m), τότε ο κόμβος *m* αντιγράφεται έμμεσα γιατί κάποιος πρόγονος του *m* έχει αντιγραφεί σε κάποιο πρόγονο του *n* και κάποιο πρόγονο του *n'*. Εάν m' ≠ p(m), τότε μετακινούμε τον κόμβο *m* στον κόμβο *m'* και στη συνέχεια αντιγράφουμε τον κόμβο *m'*, αντιγράφοντας έτσι έμμεσα και τον κόμβο *m*. Γενικά, ομαδοποιούμε τα αντίγραφα n_i του κόμβου *m* με βάση τον κόμβο στον οποίο ταιριάζουν οι πατέρες των n_i. Οι ομάδες αυτές καλούνται αντίγραφα-λουλούδια.

Κάθε αντίγραφο-λουλούδι μπορεί να παράγει έναν αριθμό από έμμεσα αντίγραφα. Στην περίπτωση που m' ≠ p(m), πρέπει να μετακινήσουμε ένα αντίγραφο του *m* στον κόμβο *m'*. Μπορούμε να πάρουμε τέτοια έμμεσα αντίγραφα, αντιγράφοντας κάποιο πρόγονο του *m*. Εάν ο *m'* είναι ο ειδικός κόμβος +, τότε συνεχίζουμε να ψάχνουμε στους προγόνους του *m* για έναν πρόγονο που δεν είναι ο ειδικός κόμβος +, εφόσον κόμβοι που έχουν εισαχθεί, δεν μπορούν στη συνέχεια να αντιγραφούν.



Σχήμα 4.18 Παράδειγμα για την περίπτωση 2.1 του αλγορίθμου ανάθεσης

Ένα παράδειγμα της περίπτωσης 2.1, παρουσιάζεται στο Σχήμα 4.18. Έστω ότι εξετάζουμε την ακμή [2,12]. Εφόσον υπάρχει και η ακμή [2,17] που περνάει από τον κόμβο 2, εφαρμόζεται η περίπτωση 2.1. Παρατηρούμε ότι ο πατέρας του κόμβου 12, ο κόμβος 11, και ο πατέρας του κόμβου 17, ο κόμβος 16, ταιριάζουν με τον ίδιο κόμβο του δένδρου T1, τον κόμβο 6. Οι κόμβοι 12 και 17 ομαδοποιούνται και ψάχνουμε για ένα έμμεσο αντίγραφο του κόμβου 2 που μπορούμε να μετακινήσουμε στον κόμβο 6. Παρατηρούμε ότι το προφανές αντίγραφο του κόμβου 2, δηλαδή το αντίγραφο που θα ήταν παιδί του κόμβου 10, δεν υπάρχει. Επομένως ο κόμβος 2 μετακινείται στον κόμβο 6 και στη συνέχεια ο κόμβος 6 αντιγράφεται στον κόμβο 1. Η διαδικασία αυτή γίνεται αναθέτοντας στην ακμή [2,17] την λειτουργία μετακίνησης και στην ακμή [6,11] τη λειτουργία αντιγραφής. Στις ακμές [2,12] και [6,16] ανατίθεται η λειτουργία *NIL*. Με αυτό τον τρόπο, παίρνουμε ένα έμμεσο αντίγραφο του κόμβου 2. Η σειρά με την οποία θα γίνουν οι λειτουργίες, φαίνεται στο Σχήμα 4.18 με το βέλος, υποδεικνύοντας την εξάρτηση της αντιγραφής [6,11] από την μετακίνηση [2,17].

Υποπερίπτωση 2.2: Ο κόμβος *m* είναι ο ειδικός κόμβος +. Ο συνδυασμός του κόμβου *n* με τον ειδικό κόμβο +, υποδεικνύει μια λειτουργία εισαγωγής. Εφόσον ο βαθμός του *m* είναι μεγαλύτερος από 1, σημαίνει ότι περισσότεροι από ένας κόμβοι πρέπει να εισαχθούν στο δένδρο. Όπως και στην υποπερίπτωση 2.1, μια ή περισσότερες λειτουργίες εισαγωγής μπορούν να αποφευχθούν εάν σε κάποιο πρόγονο των κόμβων προς εισαγωγή ανατεθεί μια λειτουργία αντιγραφής. Σε μια τέτοια περίπτωση, θα παίρναμε έμμεσα αντίγραφα των κόμβων αυτών.

Αρχικά, ελέγχουμε το δένδρο *T2* για να βρούμε εάν υπάρχουν άλλα αντίγραφα του κόμβου *n*. Εάν δεν υπάρχουν αντίγραφα, αναθέτουμε στην ακμή *e* την λειτουργία εισαγωγής. Εάν υπάρχει ένα αντίγραφο του *n*, το *n1*, τότε εξετάζουμε αν υπάρχει αριθμός *k*, τέτοιος ώστε ο k-οστός πρόγονος του *n*, $p(n)^k$ και ο k-οστός πρόγονος του *n1*, $p(n1)^k$ να έχουν συνδυαστεί με τον ειδικό κόμβο +. Σε αυτή την περίπτωση, το υποδένδρο με ρίζα τον κόμβο $p(n1)^k$ είναι αντίγραφο του υποδένδρου $p(n)^k$, οπότε κατά την αντιγραφή αυτού του υποδένδρου, θα αντιγραφεί έμμεσα και ο κόμβος *n1*.

Γενικά, τοποθετούμε τα αντίγραφα n_i του κόμβου *n* στην ίδια ομάδα, εάν υπάρχει *k*, τέτοιο ώστε οι πρόγονοί τους στο επίπεδο *k* να έχουν συνδυαστεί με τον ειδικό κόμβο +. Από κάθε ομάδα, αναθέτουμε σε ένα από τα αντίγραφα την λειτουργία εισαγωγής

και στα υπόλοιπα αναθέτουμε την λειτουργία *NIL*. Τα αντίγραφα αυτά θα εισαχθούν στο δένδρο έμμεσα από την αντιγραφή κάποιου πρόγονού τους.

Περίπτωση 3: Ο βαθμός του *m* είναι 1 και ο βαθμός του *n* είναι μεγαλύτερος από 1, δηλαδή υπάρχουν περισσότερες από μια ακμές που περνάνε από τον κόμβο *n*. Η περίπτωση αυτή είναι τελείως ανάλογη με την περίπτωση 2.

4.4. Δεύτερη Φάση της Μεθόδου

4.4.1. Ουγγρική μέθοδος (Hungarian method)

Ο Ουγγρικός αλγόριθμος εφαρμόζεται σε διμερείς γράφους και βρίσκει ένα ταίριασμα μεταξύ των δύο μερών του γράφου τέτοιο ώστε κάθε κόμβος του αριστερού μέρους να ταιριάξει με έναν ακριβώς κόμβο του δεξιού μέρους (τέλειο ταίριασμα). Εάν κάθε ακμή που συνδέει έναν κόμβο του αριστερού μέρους με έναν κόμβο του δεξιού μέρους έχει ένα βάρος, τότε ο Ουγγρικός αλγόριθμος βρίσκει το τέλειο ταίριασμα με το ελάχιστο κόστος.

Έστω ότι το αριστερό μέρος του γράφου αντιπροσωπεύει τις λειτουργίες της υπηρεσίας διαδικτύου *Δ1* και το δεξί μέρος τις λειτουργίες της υπηρεσίας διαδικτύου *Δ2*. Από την πρώτη φάση της μεθόδου, έχουμε υπολογίσει το κόστος c_{ij} της μετατροπής κάθε λειτουργίας *i* της *Δ1* σε κάθε λειτουργία *j* της *Δ2*. Κάθε ακμή [i,j] έχει βάρος όσο το κόστος c_{ij}. Με δεδομένα τα βάρη αυτά, ο Ουγγρικός αλγόριθμος θα ταιριάξει τέλεια τις λειτουργίες της *Δ1* με τις λειτουργίες της *Δ2*. Με άλλα λόγια, κάθε λειτουργία της *Δ1* θα ταιριάξει με μία μόνο λειτουργία της *Δ2* και το ταίριασμα αυτό θα έχει το ελάχιστο κόστος.

Έστω *M* ο αριθμός των λειτουργιών της υπηρεσίας *Δ1* και *N* ο αριθμός των λειτουργιών της υπηρεσίας *Δ2*. Κατασκευάζουμε έναν πίνακα *Π* με διαστάσεις MxN, τέτοιο ώστε κάθε κελί [i,j] του πίνακα *Π* να έχει τιμή c_{ij}. Έστω $m=max\{M,N\}$. Μετατρέπουμε τον πίνακα *Π* σε τετραγωνικό με διαστάσεις mxm και θέτουμε σε κάθε νέο κελί την τιμή άπειρο. Δημιουργούμε έναν βοηθητικό πίνακα *M* με διαστάσεις mxm. Αρχικά, όλα τα στοιχεία του πίνακα *M* έχουν τιμή 0. Κάθε στοιχείο του *M* αποτελεί μια μάσκα για τον πίνακα *Π*. Συγκεκριμένα, αν το στοιχείο *M[i,j]* έχει τιμή 1, τότε θεωρούμε ότι το αντίστοιχο στοιχείο *Π[i,j]* έχει σημειωθεί με

αστερίσκο, ενώ ένα έχει τιμή 2 τότε έχει σημειωθεί με τόνο. Δημιουργούμε επίσης, δύο βοηθητικά διανύσματα *C* και *R* με μήκος *m*. Το διάνυσμα *C* αντιστοιχεί στις στήλες του πίνακα *Π* και το διάνυσμα *R* στις γραμμές του. Αρχικά, όλες οι θέσεις του διανύσματος έχουν τιμή 0. Εάν κάποια θέση του διανύσματος *C* έχει τιμή 1, τότε θεωρούμε ότι η αντίστοιχη στήλη έχει καλυφθεί. Αντίστοιχα, εάν κάποια θέση του διανύσματος *R* έχει τιμή 1, τότε θεωρούμε ότι η αντίστοιχη γραμμή του πίνακα *Π* έχει καλυφθεί. Παίρνοντας σαν είσοδο τον πίνακα *Π*, η Ουγγρική μέθοδος ακολουθεί τα παρακάτω βήματα:

Βήμα 1: Για κάθε γραμμή του πίνακα, βρίσκουμε το μικρότερο βάρος και το αφαιρούμε από όλα τα βάρη της αντίστοιχης γραμμής. Αποτέλεσμα αυτής της διαδικασίας είναι ότι κάθε γραμμή θα περιέχει τουλάχιστον ένα μηδενικό.

Βήμα 2: Διατρέχουμε τον πίνακα *Π* και για κάθε μηδενικό που συναντάμε έστω στη θέση [k,l], ελέγχουμε εάν υπάρχει κάποιο στοιχείο στην ίδια γραμμή ή στην ίδια στήλη που να έχει σημειωθεί με αστερίσκο. Εάν δεν υπάρχει τέτοιο στοιχείο, τότε στη θέση [k,l] του πίνακα *M* θέτουμε την τιμή 1 (σημειώνουμε το μηδενικό με αστερίσκο).

Βήμα 3: Καλύπτουμε κάθε στήλη *j* που περιέχει ένα στοιχείο με αστερίσκο, θέτοντας C(j)=1. Στη συνέχεια, αθροίζουμε τα στοιχεία του διανύσματος *C*. Εάν το άθροισμα αυτό είναι μεγαλύτερο ή ίσο από *m*, προχωράμε στο βήμα 7. Διαφορετικά, συνεχίζουμε στο βήμα 4.

Βήμα 4: Διατρέχουμε τον πίνακα *Π* έως ότου να βρούμε κάποιο μηδενικό που δεν έχει καλυφθεί. Εάν υπάρχει τέτοιο στοιχείο *Π[i,j]*, τότε το σημειώνουμε με τόνο θέτοντας M[i,j]=2. Εάν στην γραμμή *i* του πίνακα *M* δεν υπάρχει κανένα στοιχείο με τιμή 0, τότε προχωράμε στο βήμα 5. Διαφορετικά, καλύπτουμε την γραμμή *i* θέτοντας R(i)=1 και αφήνουμε ακάλυπτη τη στήλη *j* θέτοντας C(j)=0. Επαναλαμβάνουμε το βήμα 4 μέχρι να καλύψουμε όλες τις θέσεις του πίνακα *Π* με τιμή 0. Στη συνέχεια, αποθηκεύουμε την μικρότερη ακάλυπτη τιμή και προχωράμε στο βήμα 6.

Βήμα 5: Κατασκευάζουμε μια αλληλουχία από εναλλασσόμενα μηδενικά με τόνο και μηδενικά με αστερίσκο ως εξής: Έστω *Z0* το ακάλυπτο μηδενικό με τόνο που βρέθηκε στο βήμα 4. Τότε διατρέχουμε τη στήλη στην οποία βρίσκεται το *Z0* και βρίσκουμε αν υπάρχει μηδενικό με αστερίσκο, έστω *Z1*. Στη συνέχεια, διατρέχουμε την γραμμή στην οποία βρίσκεται το στοιχείο *Z1* και βρίσκουμε εάν υπάρχει μηδενικό με τόνο, έστω *Z2*. Συνεχίζουμε αυτή τη διαδικασία έως ότου να βρεθεί

κάποια στήλη που έχει μηδενικό με τόνο και δεν υπάρχει μηδενικό με αστερίσκο. Όταν βρεθεί μια τέτοια αλληλουχία, αφαιρούμε τον αστερίσκο από κάθε μηδενικό με αστερίσκο της αλληλουχίας και σημειώνουμε με αστερίσκο κάθε μηδενικό με τόνο. Τέλος, αφήνουμε ακάλυπτες όλες τις γραμμές του πίνακα και επιστρέφουμε στο βήμα 3.

Βήμα 6: Προσθέτουμε την τιμή που βρέθηκε στο βήμα 4 σε κάθε στοιχείο όλων των καλυπτόμενων γραμμών του πίνακα *Π* και την αφαιρούμε από κάθε στοιχείο όλων των ακάλυπτων στηλών. Επιστρέφουμε στο βήμα 4.

Βήμα 7: Ο αλγόριθμος τερματίζει και το ταίριασμα που βρέθηκε υπολογίζεται ως εξής: Εάν το στοιχείο *Π[i,j]* είναι μηδενικό με αστερίσκο, τότε η λειτουργία που αντιστοιχεί στην γραμμή *i* ταιριάζει με την λειτουργία που αντιστοιχεί στην στήλη *j*.

4.4.2. Αντιστοίχηση λειτουργιών

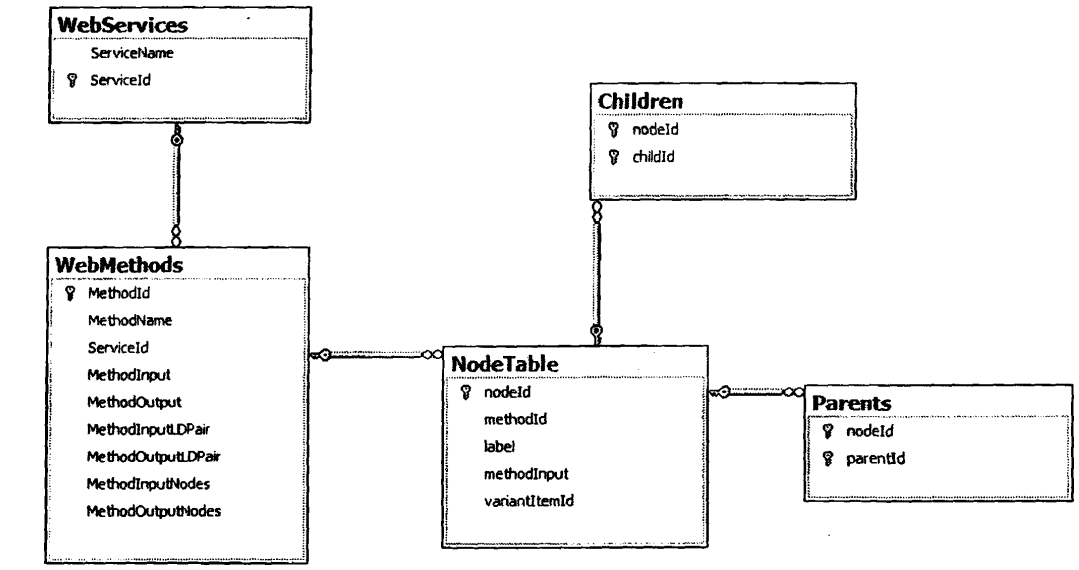
Στην πρώτη φάση της μεθόδου, υπολογίσαμε για κάθε ζεύγος λειτουργιών μεταξύ των υπηρεσιών διαδικτύου *Δ1* και *Δ2* που θέλουμε να συγκρίνουμε, ένα κόστος μετατροπής. Στη δεύτερη φάση της μεθόδου, τροφοδοτούμε την Ουγγρική μέθοδο με τα κόστη αυτά. Το αποτέλεσμα είναι μια αντιστοίχηση μεταξύ των λειτουργιών της υπηρεσίας *Δ1* και των λειτουργιών της υπηρεσίας *Δ2*.

Κατασκευάζουμε τον πίνακα *Π* που παίρνει σαν είσοδο η Ουγγρική μέθοδος ως εξής: Κάθε γραμμή του πίνακα *Π* αντιστοιχεί σε μια μέθοδο της υπηρεσίας *Δ1* και κάθε στήλη του πίνακα *Π* αντιστοιχεί σε μια μέθοδο της υπηρεσίας *Δ2*. Το στοιχείο [i,j] έχει τιμή όσο το κόστος της μετατροπής της μεθόδου που αντιστοιχεί στη θέση *i* του πίνακα στη μέθοδο που αντιστοιχεί στη θέση *j*.

Αφού εφαρμόσουμε την Ουγγρική μέθοδο, ο πίνακας *Π* θα έχει σε κάθε γραμμή και κάθε στήλη το πολύ ένα στοιχείο με τιμή μηδενικό με αστερίσκο. Τα στοιχεία αυτά αντιπροσωπεύουν την αντιστοίχηση μεταξύ των μεθόδων των δύο υπηρεσιών. Αν κάποια γραμμή ή κάποια στήλη δεν έχει καμία πραγματική τιμή, τότε η μέθοδος που αντιστοιχεί σε αυτή τη γραμμή ή στήλη δεν υπάρχει σε μια από τις δύο υπηρεσίες διαδικτύου και άρα πρέπει να προστεθεί ή να αφαιρεθεί αντίστοιχα. Υπολογίζοντας το άθροισμα των πραγματικών τιμών του πίνακα *Π*, βρίσκουμε το συνολικό ελάχιστο κόστος της μετατροπής από την υπηρεσία *Δ1* στην υπηρεσία *Δ2*.

4.5. Βάση Δεδομένων

Στο Σχήμα 4.19 παρουσιάζεται το σχήμα της βάσης δεδομένων που χρησιμοποιεί το περιβάλλον ανίχνευσης αλλαγών. Στον πίνακα *WebServices* αποθηκεύονται οι υπηρεσίες διαδικτύου που έχουν αναλυθεί. Στο πεδίο *ServiceName* αποθηκεύεται το όνομα του αρχείου WSDL που περιέχει την περιγραφή WSDL της υπηρεσίας ενώ το πεδίο *ServiceId* είναι ένας αύξων αριθμός που αποτελεί το κύριο κλειδί του πίνακα.



Σχήμα 4.19 Το σχήμα της βάσης δεδομένων του συστήματος

Για κάθε λειτουργία που υπάρχει στην WSDL περιγραφή μιας υπηρεσίας, εισάγεται μια εγγραφή στον πίνακα *WebMethods*. Το πεδίο *MethodId* είναι ένας αύξων αριθμός που αποτελεί το κύριο κλειδί του πίνακα και το πεδίο *ServiceId* είναι ξένο κλειδί στον πίνακα *WebServices* ώστε να γνωρίζουμε σε ποια υπηρεσία διαδικτύου ανήκει η λειτουργία. Το πεδίο *MethodName* αποθηκεύει το όνομα της λειτουργίας και τα πεδία *MethodInput* και *MethodOutput* αποθηκεύουν το όνομα του GML αρχείου της εισόδου και της εξόδου της λειτουργίας αντίστοιχα. Τα πεδία *MethodInputLDPair* και *MethodOutputLDPair* αποθηκεύουν την ld-pair αναπαράσταση των δένδρων που αντιστοιχούν στην είσοδο και την έξοδο της λειτουργίας και χρησιμοποιούνται κατά την σύγκριση δύο υπηρεσιών διαδικτύου όταν στην πρώτη φάση της μεθόδου χρησιμοποιείται ο αλγόριθμος MM-DIFF.

Οι πίνακες *NodeTable*, *Children* και *Parents* αποθηκεύουν τη δομή ενός δένδρου. Στον πίνακα *NodeTable* υπάρχει μια εγγραφή για κάθε κόμβο όλων των δένδρων που δημιουργούνται για κάθε υπηρεσία διαδικτύου. Το πεδίο *nodeId* είναι το κύριο κλειδί

του πίνακα *NodeTable*, το πεδίο *methodId* είναι ξένο κλειδί στον πίνακα *WebMethods*, το πεδίο *label* αποθηκεύει το όνομα του κόμβου και το πεδίο *methodInput* έχει την τιμή 1 εάν ο κόμβος που αντιστοιχεί στην εγγραφή περιέχεται στο δένδρο που αντιστοιχεί στην είσοδο της λειτουργίας και την τιμή 0 εάν περιέχεται στο δένδρο που αντιστοιχεί στην έξοδο της λειτουργίας. Στον πίνακα *Parents* υπάρχουν εγγραφές για κάθε ζεύγος πατέρα-παιδιού και από τον πίνακα *Children* μπορούμε εύκολα να ανακτήσουμε τα παιδιά ενός κόμβου.

4.6. Παράδειγμα χρήσης της μεθόδου

Έστω η WSDL περιγραφή *W1* που παρουσιάζεται στον Πίνακα 4.10. Η υπηρεσία παρέχει ένα σύνολο λειτουργιών, μια εκ των οποίων είναι η *Bank1_ConvertCurrency* που παίρνει σαν είσοδο έναν πολύπλοκο τύπο δεδομένων με πεδία το ποσό (*Amount*) μετρημένο στην νομισματική μονάδα *ConvertFromCurrency* και επιστρέφει το ποσό αυτό μετρημένο στην νομισματική μονάδα *ConvertToCurrency*. Η έξοδος επιστρέφει έναν αριθμό διπλής ακρίβειας, *ConvertedAmount*. Έστω ότι η λειτουργία αυτή καλείται από μια εφαρμογή της τράπεζας Bank1, η οποία χρησιμοποιεί το επιστρεφόμενο ποσό για μια συναλλαγή. Ένα τμήμα του κώδικα αυτής της εφαρμογής, παρουσιάζεται στον Πίνακα 4.11.

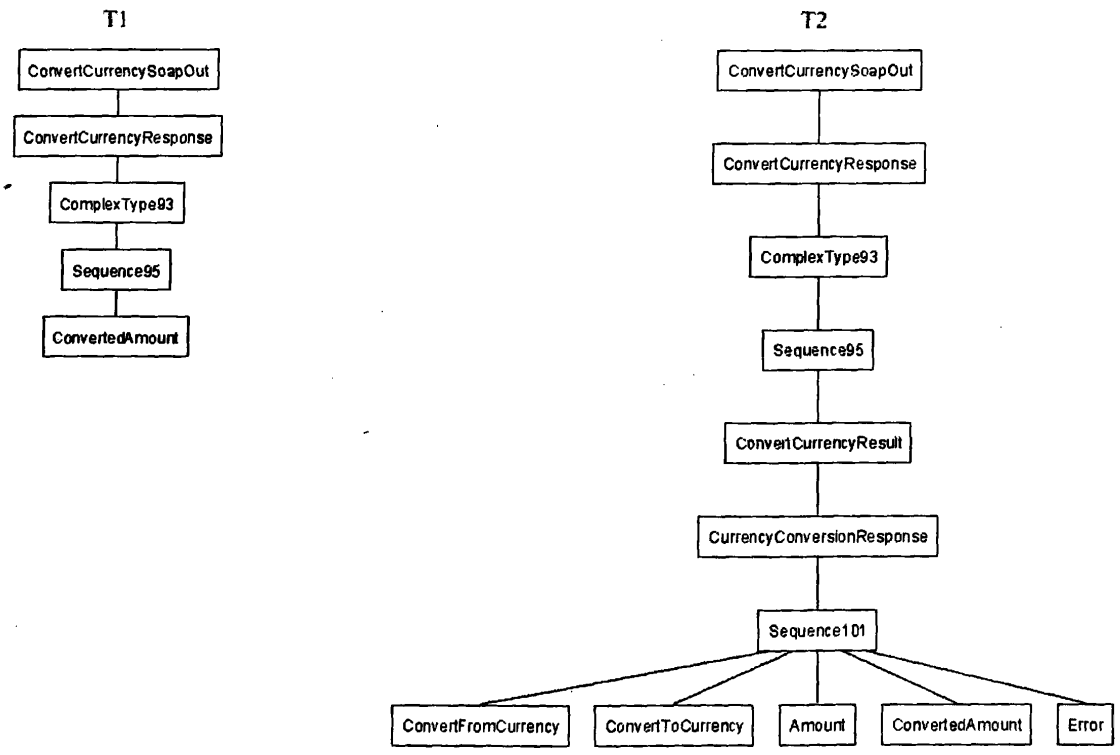
Έστω μια δεύτερη τράπεζα Bank2, η οποία χρησιμοποιεί σε μια εφαρμογή της, την υπηρεσία διαδικτύου της Bank1. Η Bank2 επιθυμεί να αναπτύξει τη δική της υπηρεσία διαδικτύου ώστε να καλύψει καλύτερα τις ανάγκες της και επεκτείνει την υπηρεσία της Bank1 δημιουργώντας ένα δικό της αντίγραφο *W2*. Η WSDL περιγραφή της *W2* παρουσιάζεται στον Πίνακα 4.12. Προφανώς, η εφαρμογή της Bank2 που χρησιμοποιούσε την *W1*, θα πρέπει να τροποποιηθεί.

Πίνακας 4.10 WSDL περιγραφή της W1

```
<wsdl:types>
 <s:schema elementFormDefault="qualified"
targetNamespace="http://www.serviceobjects.com/">
 . . .
<s:element name="ConvertCurrency">
 <s:complexType>
 <s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="Amount" type="s:string" />
  <s:element minOccurs="0" maxOccurs="1" name="ConvertFromCurrency" type="s:string" />
  <s:element minOccurs="0" maxOccurs="1" name="ConvertToCurrency" type="s:string" />
 </s:sequence>
 </s:complexType>
</s:element>
<s:element name="ConvertCurrencyResponse">
 <s:complexType>
   <s:sequence>
     <s:element minOccurs="0" maxOccurs="1" name="ConvertedAmount" type="s:double" />
   </s:sequence>
 </s:complexType>
</s:element>
</s:schema>
  </wsdl:types>
  <wsdl:message name="GetAllCurrenciesSoapIn">
    <wsdl:part name="parameters" element="tns:GetAllCurrencies" />
  </wsdl:message>
  <wsdl:message name="GetAllCurrenciesSoapOut">
    <wsdl:part name="parameters" element="tns:GetAllCurrenciesResponse" />
  </wsdl:message>
  <wsdl:message name="GetExchangeRateSoapIn">
    <wsdl:part name="parameters" element="tns:GetExchangeRate" />
  </wsdl:message>
  <wsdl:message name="GetExchangeRateSoapOut">
    <wsdl:part name="parameters" element="tns:GetExchangeRateResponse" />
  </wsdl:message>
  <wsdl:message name="ConvertCurrencySoapIn">
    <wsdl:part name="parameters" element="tns:ConvertCurrency" />
  </wsdl:message>
  <wsdl:message name="ConvertCurrencySoapOut">
    <wsdl:part name="parameters" element="tns:ConvertCurrencyResponse" />
  </wsdl:message>
  <wsdl:operation name="Bank1_GetAllCurrencies">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Gives a list of the
supported currency symbols.</documentation>
      <wsdl:input message="tns:GetAllCurrenciesSoapIn" />
      <wsdl:output message="tns:GetAllCurrenciesSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="Bank1_GetExchangeRate">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Gives the currency
exchange rate between the two currencies.</documentation>
      <wsdl:input message="tns:GetExchangeRateSoapIn" />
      <wsdl:output message="tns:GetExchangeRateSoapOut" />
    </wsdl:operation>
  <wsdl:operation name="Bank1_ConvertCurrency">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Gives the converted
amount of currency.</documentation>
      <wsdl:input message="tns:ConvertCurrencySoapIn" />
      <wsdl:output message="tns:ConvertCurrencySoapOut" />
  </wsdl:operation>
 . . .
```

Πίνακας 4.11 Εφαρμογή που χρησιμοποιεί την υπηρεσία W1

```
<?php

function getConvertedAmount($amount, $ConvertFrom, $ConvertTo){
  $c = new nusoapclient('Bank1.wsdl');

  $arr_params = array('Amount' =>  $amount);
  $arr_params += array('ConvertFromCurrency' =>  $ConvertFrom);
  $arr_params += array('ConvertToCurrency' =>  $ConvertTo);

  $result = $c->call('Bank1_ConvertCurrency', array('parameters' => $arr_params));

  return $result;
}

function completeTransaction($account1, $account2, $amount)
{
  $currency1 = getCurrencyOfAccount($account1);
  $currency2 = getCurrencyOfAccount($account2);
  $amountOfTransaction = getConvertedAmount($amount, $currency1, $currency2);
  $transactionId = executeTransaction($amountOfTransaction, $account1, $account2);
}

 . . .

?>
```



a) Μήνυμα εξόδου της Bank1_ConvertCurrency

b) Μήνυμα εξόδου της Bank2_ConvertCurrency

Σχήμα 4.20 Σχηματική αναπαράσταση της δομής των μηνυμάτων εξόδου των λειτουργιών Bank1_ConvertCurrency και Bank2_ConvertCurrency

Πίνακας 4.12 WSDL περιγραφή της W2

```
<wsdl:types>
 <s:schema elementFormDefault="qualified"
targetNamespace="http://www.serviceobjects.com/">
 . . .
<s:element name="ConvertCurrency">
 <s:complexType>
 <s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="Amount" type="s:string" />
  <s:element minOccurs="0" maxOccurs="1" name="ConvertFromCurrency" type="s:string" />
  <s:element minOccurs="0" maxOccurs="1" name="ConvertToCurrency" type="s:string" />
 </s:sequence>
 </s:complexType>
</s:element>
<s:element name="ConvertCurrencyResponse">
 <s:complexType>
  <s:sequence>
   <s:element minOccurs="0" maxOccurs="1" name="ConvertCurrencyResult"
type="tns:CurrencyConversionResponse" />
  </s:sequence>
 </s:complexType>
</s:element>
<s:complexType name="CurrencyConversionResponse">
 <s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="ConvertFromCurrency" type="s:string" />
  <s:element minOccurs="0" maxOccurs="1" name="ConvertToCurrency" type="s:string" />
  <s:element minOccurs="0" maxOccurs="1" name="Amount" type="s:string" />
  <s:element minOccurs="1" maxOccurs="1" name="ConvertedAmount" type="s:double" />
  <s:element minOccurs="0" maxOccurs="1" name="Error" type="s:string" />
 </s:sequence>
</s:complexType>
</s:schema>
  </wsdl:types>
  <wsdl:message name="GetAllCurrenciesSoapIn">
    <wsdl:part name="parameters" element="tns:GetAllCurrencies" />
  </wsdl:message>
  <wsdl:message name="GetAllCurrenciesSoapOut">
    <wsdl:part name="parameters" element="tns:GetAllCurrenciesResponse" />
  </wsdl:message>
  <wsdl:message name="GetExchangeRateSoapIn">
    <wsdl:part name="parameters" element="tns:GetExchangeRate" />
  </wsdl:message>
  <wsdl:message name="GetExchangeRateSoapOut">
    <wsdl:part name="parameters" element="tns:GetExchangeRateResponse" />
  </wsdl:message>
  <wsdl:message name="ConvertCurrencySoapIn">
    <wsdl:part name="parameters" element="tns:ConvertCurrency" />
  </wsdl:message>
  <wsdl:message name="ConvertCurrencySoapOut">
    <wsdl:part name="parameters" element="tns:ConvertCurrencyResponse" />
  </wsdl:message>
  <wsdl:operation name="Bank2_GetAllCurrencies">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Gives a list of the
supported currency symbols.</documentation>
      <wsdl:input message="tns:GetAllCurrenciesSoapIn" />
      <wsdl:output message="tns:GetAllCurrenciesSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="Bank2_GetExchangeRate">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Gives the currency
exchange rate between the two currencies.</documentation>
      <wsdl:input message="tns:GetExchangeRateSoapIn" />
      <wsdl:output message="tns:GetExchangeRateSoapOut" />
    </wsdl:operation>
  <wsdl:operation name="Bank2_ConvertCurrency">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Gives the converted
amount of currency.</documentation>
      <wsdl:input message="tns:ConvertCurrencySoapIn" />
      <wsdl:output message="tns:ConvertCurrencySoapOut" />
  </wsdl:operation>
...
```

Η μέθοδος ανίχνευσης αλλαγών που αναπτύχθηκε σε αυτή την εργασία, εξάγει τα παρακάτω αποτελέσματα:

Η λειτουργία *Bank1_GetAllCurrencies* αντιστοιχήθηκε με τη λειτουργία *Bank2_GetAllCurrencies*, η λειτουργία *Bank1_GetExchangeRate* αντιστοιχήθηκε με τη λειτουργία *Bank2_GetExchangeRate* ενώ η λειτουργία *Bank1_ConvertCurrency* αντιστοιχήθηκε με τη λειτουργία *Bank2_ConvertCurrency*. Τα σενάρια τροποποίησης μεταξύ των δύο πρώτων αντιστοιχήσεων δεν περιείχαν καμία αλλαγή, εφόσον η Bank2 δεν τροποποίησε τη δομή των μηνυμάτων εισόδου και εξόδου. Το σενάριο τροποποίησης μεταξύ των μηνυμάτων εισόδου των λειτουργιών *Bank1_ConvertCurrency* και *Bank2_ConvertCurrency* ήταν επίσης κενά. Τα δένδρα των μηνυμάτων εξόδου των παραπάνω λειτουργιών παρουσιάζονται στο Σχήμα 4.20, ενώ το σενάριο τροποποίησης μεταξύ τους παρουσιάζεται στον Πίνακα 4.13.

Πίνακας 4.13 Σενάριο τροποποίησης των μηνυμάτων εξόδου των λειτουργιών Bank1_ConvertCurrency και Bank2_ConvertCurrency

```
INS(ConvertCurrencyResult,Sequence95),
INS(CurrencyConversionResponse,ConvertCurrencyResult),
INS(Sequence101,CurrencyConversionResponse), INS(ConvertFromCurrency,Sequence101),
INS(ConvertToCurrency,Sequence101), INS(Amount,Sequence101),
MOV(ConvertedAmount,Sequence101), INS(Error,Sequence101)
```

Με οδηγό το σενάριο τροποποίησης του Πίνακα 4.13 και με την βοήθεια του Σχήματος 4.20, ο προγραμματιστής της εφαρμογής παρατηρεί ότι το πεδίο *ConvertedAmount* που χρησιμοποιεί στην εφαρμογή του δεν επιστρέφεται απευθείας από την λειτουργία *Bank2_ConvertCurrency*. Αντίθετα, η λειτουργία αυτή επιστρέφει έναν πολύπλοκο τύπο δεδομένων με περισσότερα πεδία. Ένα από τα πεδία αυτά είναι το *ConvertedAmount*. Η αλλαγή αυτή γίνεται αντιληπτή από την λειτουργία τροποποίησης *MOV(ConvertedAmount,Sequence101)*, που ενημερώνει τον προγραμματιστή ότι το πεδίο που τον ενδιαφέρει μεταφέρθηκε σε κάποια άλλη θέση. Τελικά, θα πρέπει στην εφαρμογή του, στην συνάρτηση *getConvertedAmount()* που επιστρέφει το ποσό να αλλάξει την εντολή *return* ώστε να επιστρέφει το σωστό πεδίο. Είναι προφανές ότι η αλλαγή αυτή, επηρεάζει όλες τις κλήσεις της λειτουργίας *Bank2_ConvertCurrency* αλλά και όλες τις συναρτήσεις που καλούν τη συνάρτηση *getConvertedAmount()* όπως η *completeTransaction()*. Ο κώδικας της εφαρμογής που χρησιμοποιεί την υπηρεσία *W2*, παρουσιάζεται στον Πίνακα 4.14. Είναι επίσης πιθανό, ο προγραμματιστής να επιθυμεί να χρησιμοποιήσει και ένα από τα υπόλοιπα

επιστρεφόμενα πεδία ώστε να επεκτείνει την εφαρμογή του. Ένα τέτοιο παράδειγμα είναι η χρήση του πεδίου *Error* το οποίο, στην περίπτωση που παρουσιάστηκε κάποιο σφάλμα κατά την κλήση της *Bank2_ConvertCurrency*, περιέχει το μήνυμα λάθους.

Πίνακας 4.14 Εφαρμογή που χρησιμοποιεί την υπηρεσία W2

```
<?php

function getConvertedAmount($amount, $ConvertFrom, $ConvertTo){
  $c = new nusoapclient('Bank2.wsdl');

  $arr_params = array('Amount' =>  $amount);
  $arr_params += array('ConvertFromCurrency' =>  $ConvertFrom);
  $arr_params += array('ConvertToCurrency' =>  $ConvertTo);

  $result = $c->call('Bank2_ConvertCurrency', array('parameters' => $arr_params));

  return $result['ConvertedAmount'];
}

function completeTransaction($account1, $account2, $amount)
{
  $currency1 = getCurrencyOfAccount($account1);
  $currency2 = getCurrencyOfAccount($account2);
  $amountOfTransaction = getConvertedAmount($amount, $currency1, $currency2);
  $transactionId = executeTransaction($amountOfTransaction, $account1, $account2);
}

?>
```

ΚΕΦΑΛΑΙΟ 5. ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

5.1. Γενικά

Ο στόχος των πειραμάτων είναι να ελεγχθεί η ποιότητα των αποτελεσμάτων των δύο προσεγγίσεων πάνω σε πραγματικά δεδομένα. Για το λόγο αυτό, χρησιμοποιήσαμε ένα υποσύνολο από πραγματικές υπηρεσίες διαδικτύου [1],[2]. Το υποσύνολο αποτελείται από δέκα υπηρεσίες διαδικτύου οι οποίες διαφέρουν ως προς τον αριθμό των λειτουργιών που προσφέρουν αλλά και ως προς τον αριθμό των στοιχείων που έχουν τα μηνύματα εισόδου και εξόδου που ανταλλάσσονται. Τα στοιχεία αναγράφονται αναλυτικά στους Πίνακες του Παραρτήματος. Για κάθε υπηρεσία διαδικτύου, παράγουμε ένα σύνολο από πιθανές παραλλαγές με κριτήριο το είδος και την ποσότητα των αλλαγών. Κατά την δημιουργία της κάθε παραλλαγής, αποθηκεύεται το σενάριο τροποποίησης που χρησιμοποιήθηκε για να την παράγει και υπολογίζεται το κόστος του (άθροισμα του κόστους των επιμέρους λειτουργιών τροποποίησης). Το κόστος αυτό θεωρείται βέλτιστο.

Για να μετρήσουμε την ποιότητα των αποτελεσμάτων της κάθε προσέγγισης, υπολογίζουμε για κάθε παραλλαγή με κάθε προσέγγιση, το τελικό κόστος της μετατροπής από την αρχική υπηρεσία διαδικτύου στην παραλλαγή. Έτσι, το πρώτο κριτήριο σύγκρισης είναι η σύγκριση του βέλτιστου κόστος με το κόστος που υπολογίστηκε από κάθε προσέγγιση. Το δεύτερο κριτήριο σύγκρισης, είναι ο χρόνος που δαπάνησε κάθε αλγόριθμος για να ολοκληρώσει τη σύγκριση.

5.2. Γεννήτορας παραλλαγών

Για να παράγουμε πιθανές παραλλαγές μιας υπηρεσίας διαδικτύου που αντικατοπτρίζουν πραγματικές αλλαγές κατά την συντήρηση ή επέκταση μιας υπηρεσίας διαδικτύου, θέτουμε ορισμένες παραμέτρους. Η πρώτη παράμετρος ορίζει το ποσοστό της εισαγωγής νέων κόμβων. Εφόσον οι αλλαγές έχουν να κάνουν με την εισαγωγή κόμβων φύλλων ή την εισαγωγή υποδένδρων σε κόμβους φύλλα, ο γεννήτορας επιλέγει με κάποια τυχαιότητα τον πατέρα ενός καινούργιου κόμβου. Αν για παράδειγμα το ποσοστό εισαγωγής είναι 10% και το δένδρο έχει 100 κόμβους, ο γεννήτορας θα εισάγει 10 νέους κόμβους. Οι 5 από αυτούς θα ανήκουν σε ένα καινούργιο υποδένδρο, ενώ οι υπόλοιποι θα εισαχθούν στο δένδρο σαν κόμβοι φύλλα.

Η δεύτερη παράμετρος ορίζει το ποσοστό της διαγραφής κόμβων. Διαγραφές μπορούν να γίνουν είτε σε ενδιάμεσους κόμβους είτε σε κόμβους φύλλα. Ο γεννήτορας επιλέγει με κάποια τυχαιότητα τον κόμβο που θα διαγραφεί εξασφαλίζοντας ότι θα γίνουν διαγραφές από όλο το δένδρο. Όπως και στην περίπτωση της εισαγωγής, αν το ποσοστό διαγραφής είναι 10% και το δένδρο έχει 100 κόμβους, τότε θα επιλεγούν 10 τυχαίοι κόμβοι για διαγραφή.

Η τρίτη παράμετρος ορίζει το ποσοστό της μετακίνησης. Ο γεννήτορας επιλέγει δύο τυχαίους κόμβους *n1* και *n2* για κάθε μετακίνηση. Η μετακίνηση ορίζει ότι ο κόμβος *n1* και όλο το υποδένδρο του θα μετακινηθεί κάτω από τον κόμβο *n2*. Όπως και στις προηγούμενες περιπτώσεις, αν το ποσοστό μετακίνησης είναι 10% και το δένδρο έχει 100 κόμβους, τότε θα μετακινηθούν 10 υποδένδρα.

Για να αποφευχθούν αλλαγές που δεν έχουν νόημα στην πραγματικότητα, λαμβάνουμε ορισμένα πράγματα υπόψη μας:

- Η ρίζα του δένδρου δεν μπορεί να διαγραφεί ή να μετακινηθεί. Μπορεί φυσικά να συμμετέχει σε μια διαδικασία μετακίνησης ως ο νέος πατέρας ενός κόμβου.
- Κατά τη διαγραφή ενός κόμβου, αποθηκεύουμε τον πατέρα *p* του κόμβου που διαγράφτηκε και ο *p* δεν μπορεί να συμμετέχει στη διαδικασία εισαγωγής ενός νέου κόμβου. Αν κάτι τέτοιο συνέβαινε, θα είχαμε το πιθανό σενάριο της διαγραφής ενός κόμβου και της εισαγωγής ενός νέου κόμβου στη θέση του. Κάτι τέτοιο, αντιστοιχεί σε διαδικασία ανανέωσης και αυξάνει χωρίς λόγο το βέλτιστο κόστος.

- Κατά την διαδικασία μετακίνησης ενός κόμβου *n1* σε ένα κόμβο *n2*, ελέγχουμε καταρχήν οι δύο κόμβοι να είναι διαφορετικοί, δεύτερον ελέγχουμε ο κόμβος *n2* να μην είναι ήδη πατέρας του κόμβου *n1* εφόσον η σειρά των πεδίων σε ένα σύνθετο τύπο δεδομένων δεν παίζει ρόλο και τέλος ελέγχουμε αν ο κόμβος *n2* ανήκει στο υποδένδρο του *n1*. Κάτι τέτοιο θα δημιουργούσε κύκλους και δεν θα συνέβαινε ποτέ σε πραγματικές συνθήκες.

5.3. Πειραματική μελέτη

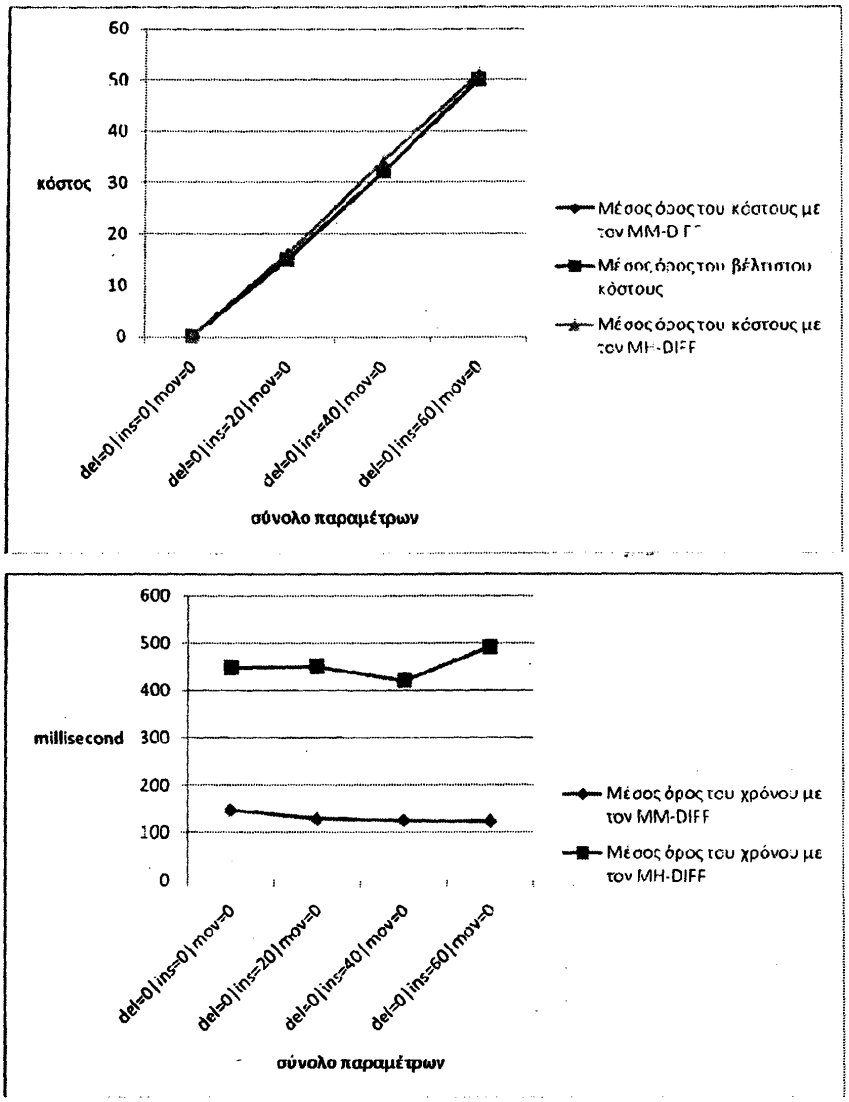
Για κάθε πραγματική υπηρεσία διαδικτύου από το σύνολο που διαθέτουμε, δημιουργήσαμε συνολικά περίπου 200 παραλλαγές. Συγκεκριμένα, τα ποσοστά εισαγωγής, διαγραφής και μετακίνησης κυμαίνονται μεταξύ 0% και 60% με βήμα 20 και έχουν δημιουργηθεί τρεις παραλλαγές για κάθε σύνολο παραμέτρων.

Μετά την δημιουργία των παραλλαγών, συγκρίνουμε κάθε υπηρεσία διαδικτύου με τις 200 παραλλαγές της. Τα επιμέρους κόστη μεταξύ των μεθόδων καθώς και τα προτεινόμενα από κάθε προσέγγιση σενάρια τροποποίησης αποθηκεύονται στη βάση δεδομένων. Επιπλέον, αποθηκεύουμε το συνολικό κόστος που υπολογίστηκε από κάθε προσέγγιση αλλά και το χρόνο που χρειάστηκε για τον υπολογισμό για την σύγκριση μιας υπηρεσίας διαδικτύου σε μια παραλλαγή της.

Στα διαγράμματα που παρουσιάζονται στη συνέχεια, στον άξονα των X αναπαρίσταται το σύνολο παραμέτρων με το οποίο δημιουργήθηκαν οι παραλλαγές ενώ στον άξονα των Y αναπαρίσταται ο μέσος όρος του κόστους ή του χρόνου των παραλλαγών από όλες τις υπηρεσίες διαδικτύου που επιλέξαμε. Σε αυτά τα διαγράμματα, οι παράμετροι κωδικοποιούνται ως εξής: με ins ορίζεται το ποσοστό εισαγωγής, με del το ποσοστό διαγραφής και με mov το ποσοστό μετακίνησης. Το σύνολο παραμέτρων del=20|ins=40|mov=20 σημαίνει ότι στην συγκεκριμένη παραλλαγή το ποσοστό διαγραφής είναι 20%, το ποσοστό εισαγωγής είναι 40% και το ποσοστό μετακίνησης είναι 20%.

Η γραμμή με τετράγωνα σε κάθε σημείο αναπαριστά το μέσο όρο του βέλτιστου κόστους. Όπως αναφέρθηκε στην παράγραφο 5.1, το βέλτιστο κόστος υπολογίζεται από το σενάριο τροποποίησης που παρήγαγε ο γεννήτορας για κάθε παραλλαγή. Η γραμμή με τρίγωνα σε κάθε σημείο αναπαριστά το μέσο όρο του κόστους όταν στην πρώτη φάση της μεθόδου χρησιμοποιήθηκε ο αλγόριθμος MH-DIFF (προσέγγιση 1)

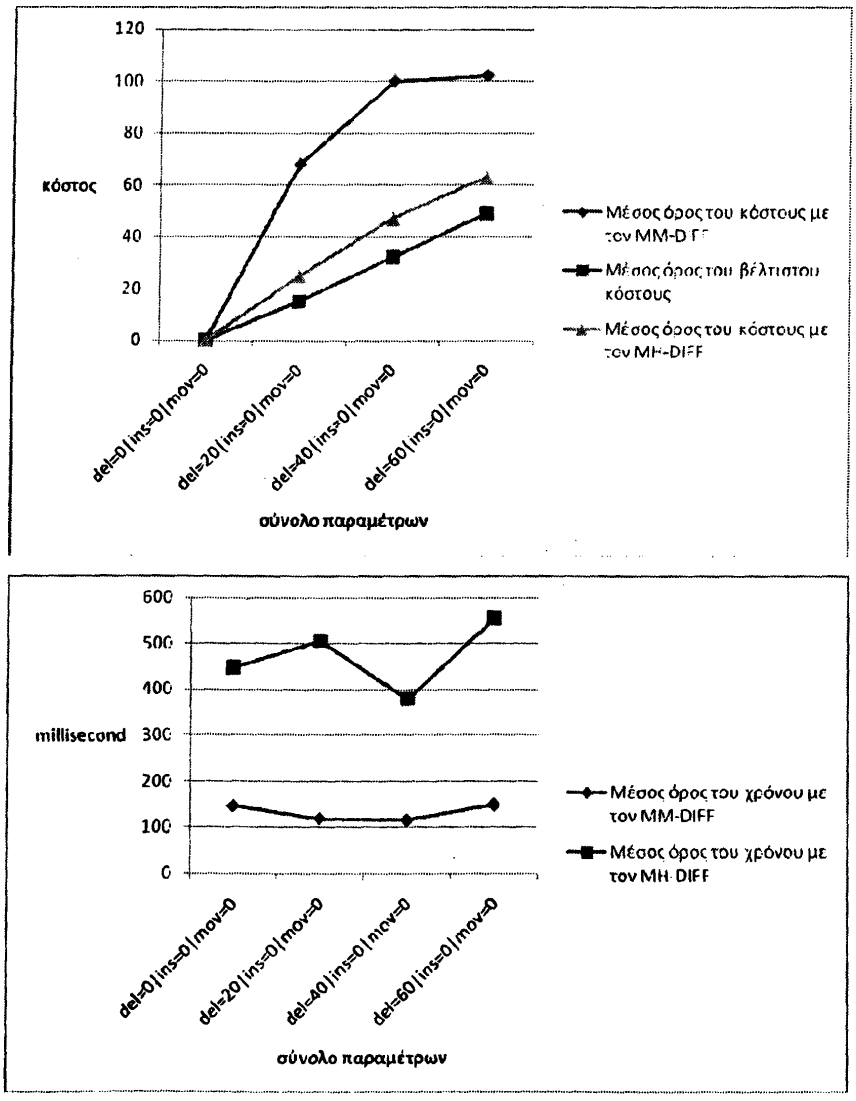
ενώ αντίστοιχα η γραμμή με ρόμβους σε κάθε σημείο αναπαριστά τον μέσο όρο του κόστους με χρήση του αλγορίθμου MM-DIFF (προσέγγιση 2).



Σχήμα 5.1 Μέσος όρος χρόνου και κόστους στις παραλλαγές που είχαν μόνο εισαγωγές

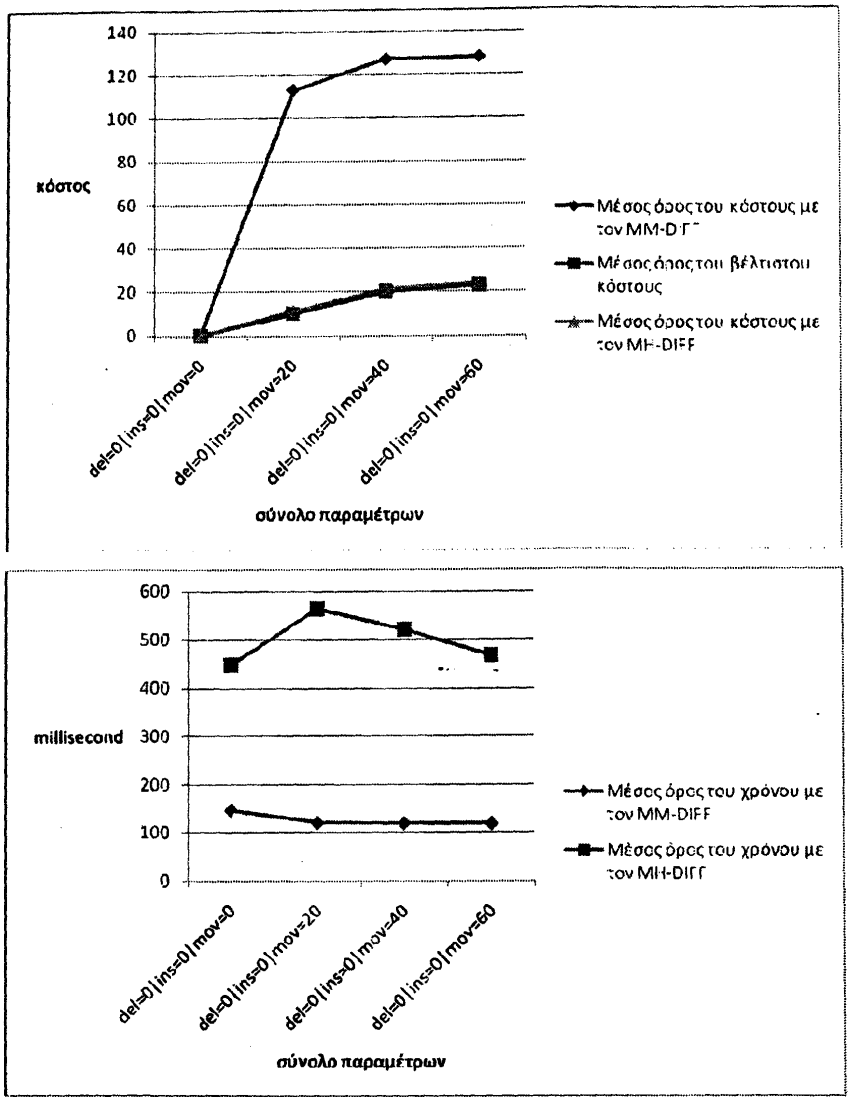
Στο Σχήμα 5.1 παρουσιάζεται ο μέσος όρος του κόστους και του χρόνου αντίστοιχα σε παραλλαγές που διέφεραν από την αρχική λόγω της εισαγωγής πεδίων (κόμβοι φύλλα) ή πολύπλοκων τύπων δεδομένων (υποδένδρα). Στο διάγραμμα κόστους παρατηρούμε ότι τα αποτελέσματα της πρώτης προσέγγισης της μεθόδου ταυτίζονται με το βέλτιστο ενώ τα αποτελέσματα της δεύτερης προσέγγισης είναι σχεδόν ταυτόσημα. Κάτι τέτοιο είναι αναμενόμενο, αφού κατά την εισαγωγή ενός πεδίου σε μια παραλλαγή, προστίθεται ένας κόμβος φύλλο στο δένδρο. Ο αλγόριθμος MM-DIFF επιστρέφει το βέλτιστο αποτέλεσμα όταν έχουν γίνει εισαγωγές κόμβων

φύλλων ενώ ο MH-DIFF βρίσκει μια λύση που είναι πολύ κοντά στη βέλτιστη. Δεν εγγυάται όμως την εύρεση της βέλτιστης λύσης. Είναι προφανές ότι τέτοιου είδους αλλαγές βρίσκουν άμεση εφαρμογή στην περίπτωση των περιγραφών WSDL κατά την εισαγωγή νέων πεδίων ή και ολόκληρων τύπων δεδομένων στην περιγραφή. Η διαφορά των προσεγγίσεων που αποτυπώνεται στο διάγραμμα είναι σε συμφωνία με την διαφορά τους σε προγραμματιστικό κόστος δεδομένου ότι τα σενάρια τροποποίησής τους είναι σχεδόν ίδια. Στο διάγραμμα χρόνου, παρατηρούμε ότι η δεύτερη προσέγγιση απαιτεί τουλάχιστον 300 millisecond παραπάνω για βρει το τελικό αποτέλεσμα. Κάτι τέτοιο είναι αναμενόμενο εφόσον ο αλγόριθμος MM-DIFF έχει πολυπλοκότητα $O(n^2)$ ενώ ο MH-DIFF $O(n^3)$.



Σχήμα 5.2 Μέσος όρος χρόνου και κόστους στις παραλλαγές που είχαν μόνο διαγραφές

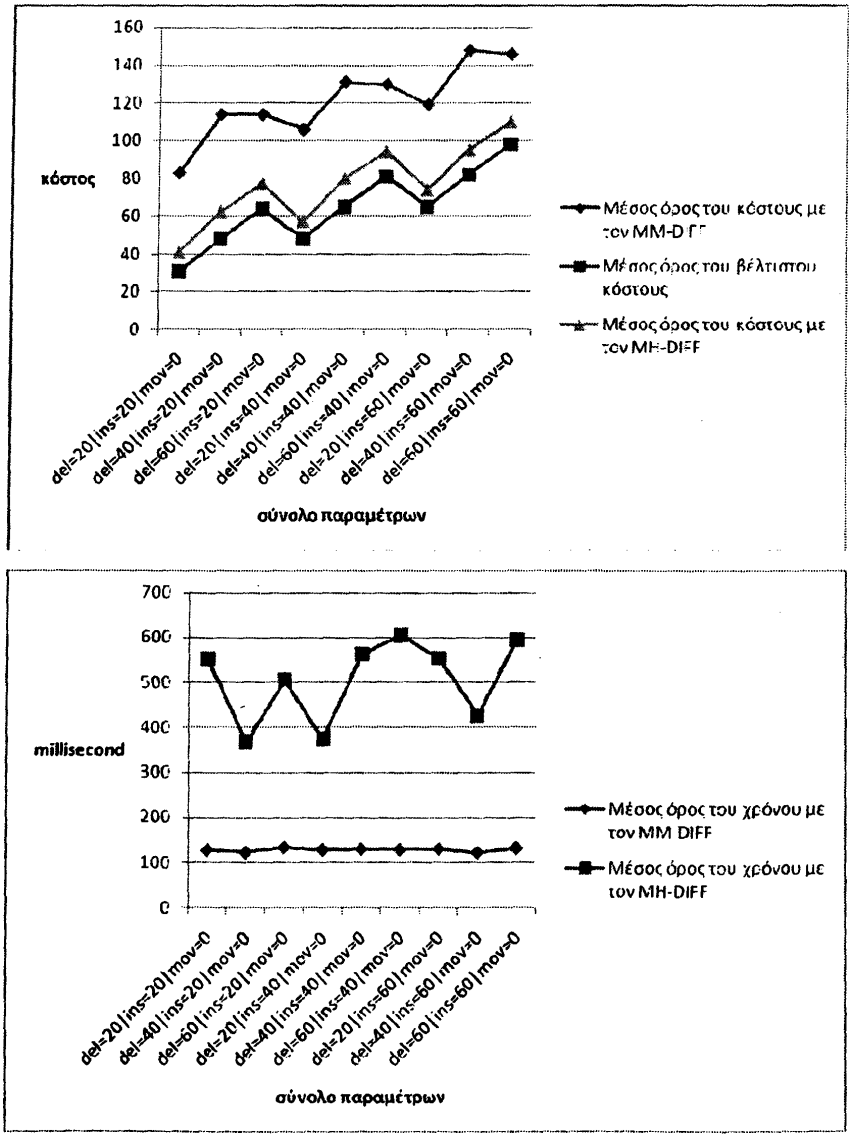
Στο Σχήμα 5.2 παρουσιάζεται ο μέσος όρος του κόστους και του χρόνου σε παραλλαγές που διέφεραν από την αρχική λόγω της διαγραφής κόμβων φύλλων ή ενδιάμεσων κόμβων ή υποδένδρων. Εάν στις παραλλαγές δεν δινόταν η δυνατότητα διαγραφής ενδιάμεσων κόμβων, τα αποτελέσματα της πρώτης προσέγγισης θα έπρεπε να είναι βέλτιστα. Ο αλγόριθμος MM-DIFF όμως, στην περίπτωση της διαγραφής ενός ενδιάμεσου κόμβου *n*, επιστρέφει ένα σενάριο τροποποίησης στο οποίο διαγράφεται όλο το υποδένδρο με ρίζα τον *n*, διαγράφεται ο *n* και εισάγεται πάλι όλο το υποδένδρο από τα παιδιά του *n* και κάτω. Για αυτό το λόγο, όσο αυξάνεται το ποσοστό των διαγραφών, τόσο αυξάνεται και η απόσταση της βέλτιστης λύσης από την λύση της πρώτης προσέγγισης. Ο αλγόριθμος MH-DIFF, υποστηρίζει την διαγραφή ενδιάμεσων κόμβων. Για αυτό το λόγο, τα αποτελέσματα της δεύτερης προσέγγισης είναι πολύ πιο κοντά στο βέλτιστο. Το σενάριο διαγραφής ενός ενδιάμεσου κόμβου αποτελεί ένα πραγματικό σενάριο κατά τη συντήρηση μιας περιγραφής WSDL. Ένα χαρακτηριστικό παράδειγμα του σεναρίου διαγραφής ενός ενδιάμεσου κόμβου, θα ήταν η διαγραφή του κόμβου *CustomerAddress* και η τοποθέτηση των *City*, *Street*, *StreetNumber* ως απευθείας παιδιά του πολύπλοκου τύπου Customer του Σχήματος 4.2(c). Με δεδομένο αυτό, η διαφορά στο κόστος των προσεγγίσεων που αποτυπώνεται στο διάγραμμα αναπαριστά και τη διαφορά σε προγραμματιστικό κόστος της υλοποίησης των δύο σεναρίων. Από το διάγραμμα χρόνου, παρατηρούμε ότι στη δεύτερη προσέγγιση της μεθόδου απαιτούνται κατά μέσο όρο 300 millisecond παραπάνω σε σχέση με την πρώτη προσέγγιση.



Σχήμα 5.3 Μέσος όρος χρόνου και κόστους στις παραλλαγές που είχαν μόνο μετακινήσεις

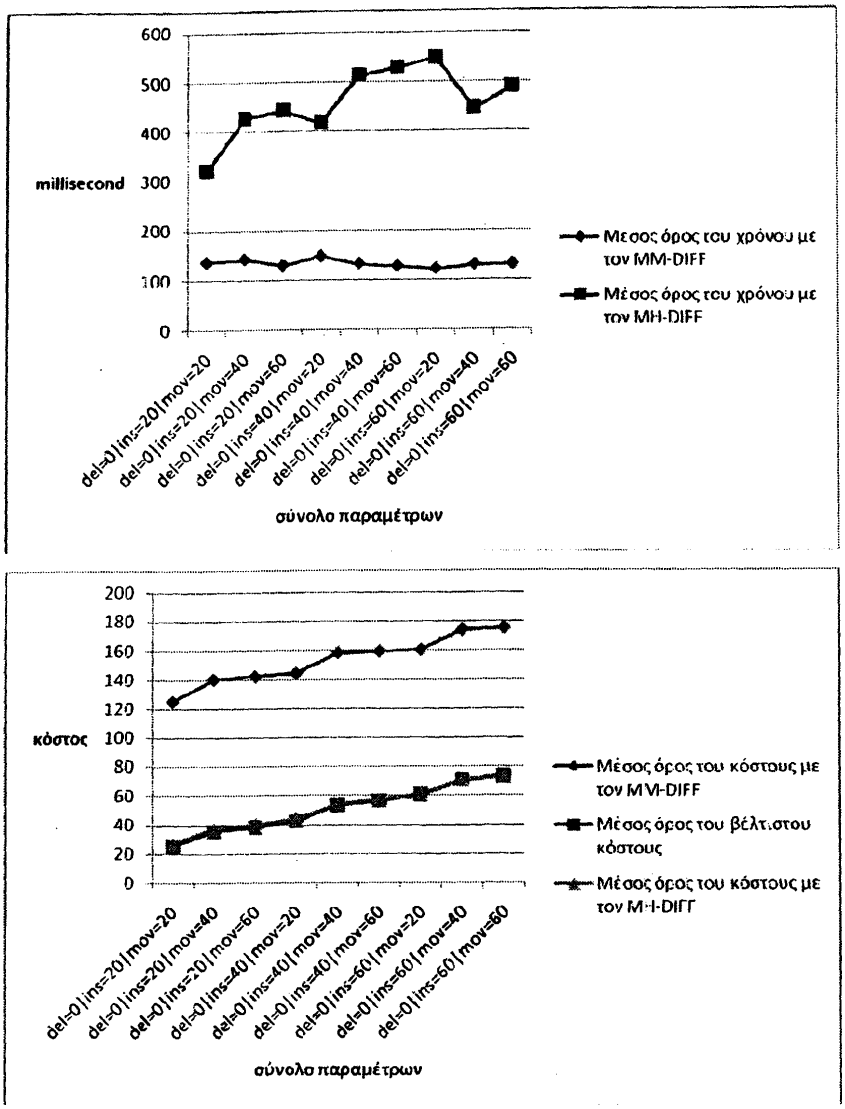
Στο Σχήμα 5.3 παρουσιάζεται ο μέσος όρος του κόστους και του χρόνου σε παραλλαγές που διέφεραν από την αρχική λόγω της μετακίνησης υποδένδρων. Όπως είναι αναμενόμενο, τα αποτελέσματα της μεθόδου με χρήση του αλγορίθμου MM-DIFF, διαφέρουν πολύ από το βέλτιστο. Θυμίζουμε ότι ο MM-DIFF προσομοιώνει μια μετακίνηση ενός υποδένδρου με την διαγραφή κάθε κόμβου του υποδένδρου από την τρέχουσα θέση του και την εισαγωγή όλων των κόμβων του υποδένδρου στη νέα θέση. Αντίθετα, ο MH-DIFF μπορεί να εντοπίσει τις μετακινήσεις υποδένδρων και να τις σημειώσει με μια μόνο λειτουργία τροποποίησης *mov*. Είναι λοιπόν αναμενόμενο τα αποτελέσματα της μεθόδου με χρήση του MH-DIFF να προσεγγίζουν το βέλτιστο. Τα κόστη αυτά αποτυπώνουν το πραγματικό προγραμματιστικό κόστος της

συντήρησης μιας εφαρμογής που θα χρησιμοποιούσε την υπηρεσία μιας και αυτό που θα γινόταν στην πράξη, θα ήταν η διατήρηση των μηχανισμών χειρισμού ενός πολύπλοκου τύπου (που αναπαρίσταται από το μετακινούμενο υποδένδρο) και η μετακίνηση της θέσης του μέσα στη δομή της εισόδου ή της εξόδου και όχι η διαγραφή και η εκ νέου υλοποίησή του σε ένα διαφορετικό σημείο. Από το διάγραμμα χρόνου, παρατηρούμε ότι η πρώτη προσέγγιση της μεθόδου απαιτεί περίπου 110 millisecond για να συγκρίνει δύο υπηρεσίες διαδικτύου, ενώ η δεύτερη προσέγγιση απαιτεί από 450 έως 650 millisecond για να ολοκληρώσει τη σύγκριση.



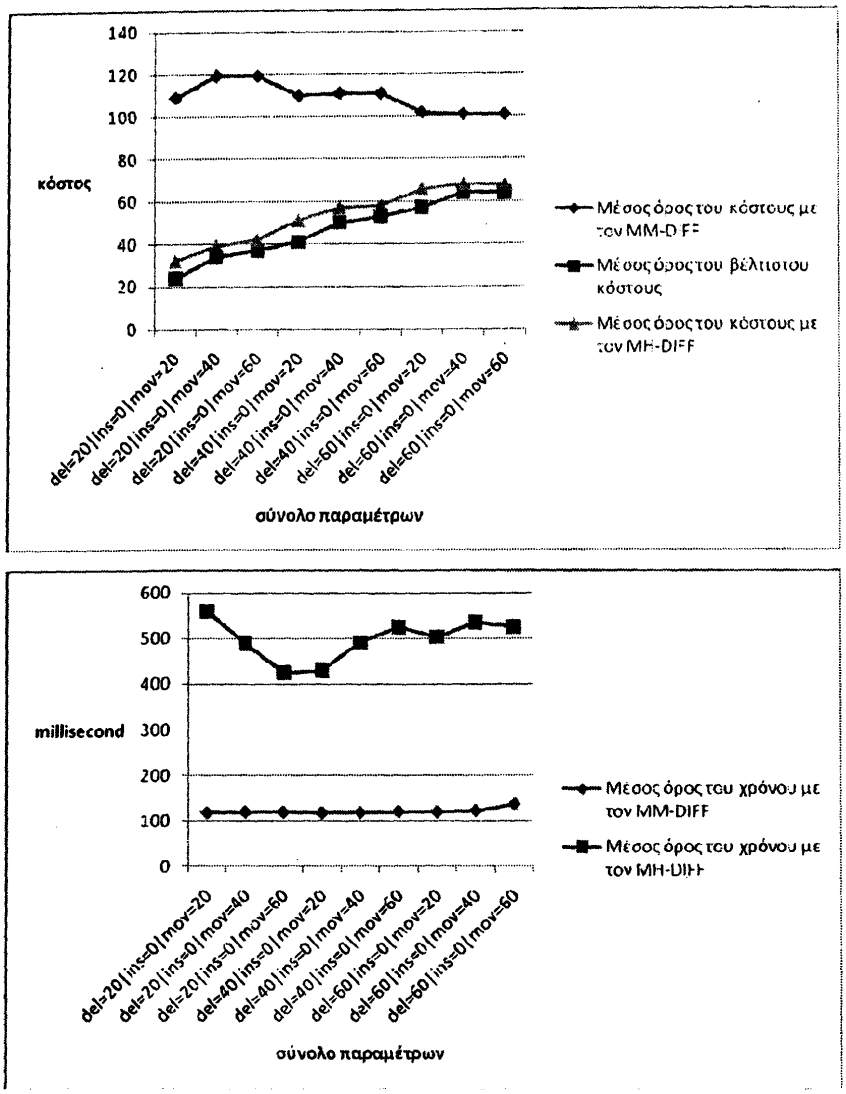
Σχήμα 5.4 Μέσος όρος χρόνου και κόστους στις παραλλαγές που είχαν εισαγωγές και διαγραφές

Στο Σχήμα 5.4 παρουσιάζεται ο μέσος όρος του κόστους και του χρόνου σε παραλλαγές που διέφεραν από την αρχική λόγω της εισαγωγής και διαγραφής πεδίων. Τα αποτελέσματα της μεθόδου με χρήση του αλγορίθμου MM-DIFF είναι περίπου 50 μονάδες πάνω από το βέλτιστο κόστος. Συγκρίνοντάς το με τα Σχήματα 5.1 και 5.2, είναι προφανές ότι οι διαγραφές ενδιάμεσων κόμβων είναι αυτές που αυξάνουν το κόστος σε τόσο μεγάλο βαθμό αφού από το Σχήμα 5.1 συμπεραίνουμε ότι οι εισαγωγές δεν επηρεάζουν το αποτέλεσμα. Αντίθετα, τα αποτελέσματα της μεθόδου με χρήση του αλγορίθμου MH-DIFF προσεγγίζουν πολύ καλύτερα το βέλτιστο αποτέλεσμα, διαφέροντας κατά μέσο όρο 10 μονάδες. Από το διάγραμμα χρόνου παρατηρούμε ότι η δεύτερη προσέγγιση απαιτεί περίπου 350 έως 600 millisecond για να συγκρίνει δύο υπηρεσίες διαδικτύου ενώ η πρώτη προσέγγιση απαιτεί περίπου 110 millisecond.



Σχήμα 5.5 Μέσος όρος χρόνου και κόστους στις παραλλαγές που είχαν εισαγωγές και μετακινήσεις

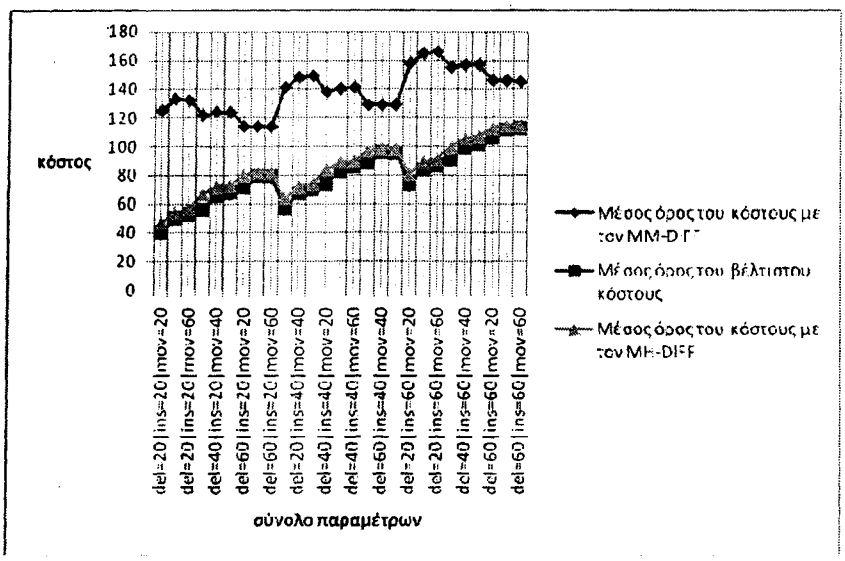
Στο Σχήμα 5.5 παρουσιάζεται ο μέσος όρος του κόστους σε παραλλαγές που διέφεραν από την αρχική λόγω της εισαγωγής και μετακίνησης κόμβων. Όπως και στο Σχήμα 5.3 όπου οι παραλλαγές είχαν μόνο μετακινήσεις και η διαφορά μεταξύ των αποτελεσμάτων της πρώτης προσέγγισης της μεθόδου είχαν διαφορά 100 περίπου μονάδων, έτσι και στο διάγραμμα του Σχήματος 5.5, η διαφορά αυτή διατηρείται. Αντίθετα, τα αποτελέσματα της δεύτερης προσέγγισης είναι σχεδόν ταυτόσημα με το βέλτιστο. Από το διάγραμμα χρόνου, παρατηρούμε ότι η δεύτερη προσέγγιση απαιτεί περίπου 300 έως 550 millisecond για να ολοκληρώσει μια σύγκριση μεταξύ δύο υπηρεσιών διαδικτύου ενώ η πρώτη προσέγγιση απαιτεί περίπου 110 millisecond.



Σχήμα 5.6 Μέσος όρος χρόνου και κόστους στις παραλλαγές που είχαν διαγραφές και μετακινήσεις

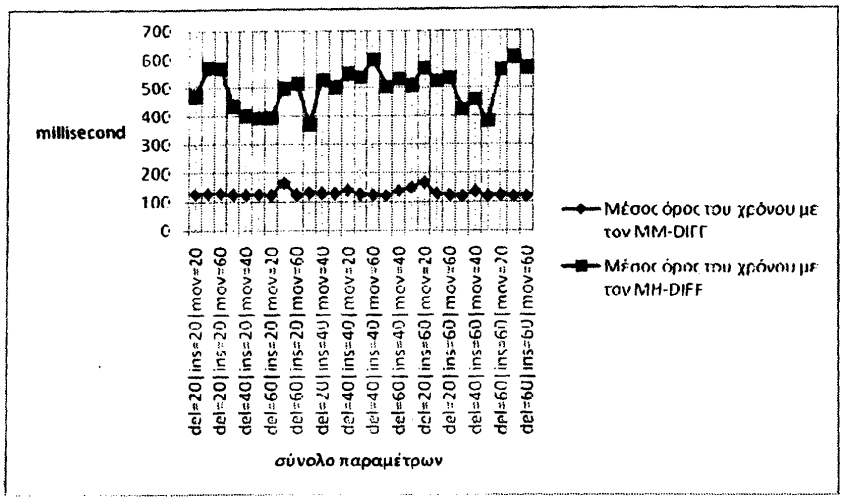
Στο Σχήμα 5.6 παρουσιάζεται ο μέσος όρος του κόστους και του χρόνου σε παραλλαγές που διέφεραν από την αρχική λόγω της διαγραφής και μετακίνησης κόμβων. Παρατηρούμε ότι η πρώτη προσέγγιση υπολογίζει ότι το συνολικό κόστος κατά μέσο όρο της μετατροπής μιας υπηρεσίας διαδικτύου στις παραλλαγές της είναι 40 έως 100 μονάδες περισσότερες από το βέλτιστο κόστος. Αξίζει να σημειώσουμε ότι όσο αυξάνεται το ποσοστό των διαγραφών, τόσο μειώνεται η διαφορά στο μέσο όρο του κόστους. Αυτό οφείλεται στην εξής παρατήρηση: ο αλγόριθμος MM-DIFF προσομοιώνει την μετακίνηση ενός υποδένδρου με τη διαγραφή και εισαγωγή στη νέα θέση των κόμβων του υποδένδρου. Εφόσον το ποσοστό των διαγραφών αυξάνεται, κατά την μετακίνηση ενός υποδένδρου απαιτούνται λιγότερες εισαγωγές

κόμβων στη νέα θέση του υποδένδρου εάν από το υποδένδρο αυτό έχουν διαγραφεί κάποιοι κόμβοι. Αντίθετα, τα αποτελέσματα της μεθόδου με χρήση του αλγόριθμου MH-DIFF που ορίζει τη λειτουργία μετακίνησης, διαφέρουν ελάχιστα από το βέλτιστο. Ο χρόνος που απαιτείται από την πρώτη προσέγγιση είναι περίπου 110 millisecond, ενώ ο χρόνος που απαιτείται από την δεύτερη προσέγγιση κυμαίνεται μεταξύ 400 και 600 millisecond.



Σχήμα 5.7 Μέσος όρος κόστους στις παραλλαγές που είχαν εισαγωγές, διαγραφές και μετακινήσεις

Στο Σχήμα 5.7 παρουσιάζεται ο μέσος όρος του κόστους σε παραλλαγές που συνδυάζουν όλα τα είδη αλλαγών με κάποιο ποσοστό. Παρατηρούμε ότι η πρώτη προσέγγιση υπολογίζει ότι το συνολικό κόστος κατά μέσο όρο της μετατροπής μιας υπηρεσίας διαδικτύου στις παραλλαγές της είναι 30 έως 80 μονάδες περισσότερες από το βέλτιστο κόστος, ενώ η δεύτερη προσέγγιση υπολογίζει το ίδιο περίπου κόστος με το βέλτιστο.



Σχήμα 5.8 Μέσος όρος χρόνου στις παραλλαγές που είχαν εισαγωγές, διαγραφές και μετακινήσεις

Στο Σχήμα 5.8 παρουσιάζεται ένα διάγραμμα με το μέσο όρο του χρόνου που δαπανήθηκε για τη σύγκριση δύο υπηρεσιών διαδικτύου στις παραλλαγές που είχαν εισαγωγές, διαγραφές και μετακινήσεις. Παρατηρούμε ότι ο χρόνος που απαιτείται από τη μέθοδο με χρήση του αλγορίθμου MM-DIFF παραμένει σταθερός γύρω στα 110 millisecond. Αντίθετα, η μέθοδος με χρήση του αλγορίθμου MH-DIFF έχει αρκετές διακυμάνσεις στο χρόνο αλλά κατά μέσο όρο απαιτεί 350 έως 600 millisecond για έναν υπολογισμό.

5.4. Συμπεράσματα

Από τα διαγράμματα κόστους της ενότητας 5.3, συμπεραίνουμε ότι η μέθοδος που προτείνουμε για την ανίχνευση αλλαγών μεταξύ δύο υπηρεσιών διαδικτύου, είναι πολύ πιο ακριβής όταν στην πρώτη φάση της μεθόδου χρησιμοποιείται ο αλγόριθμος MH-DIFF. Αυτό οφείλεται κυρίως στο γεγονός ότι ο αλγόριθμος MH-DIFF ορίζει την λειτουργία μετακίνησης, ενώ ο αλγόριθμος MM-DIFF προσομοιώνει μια μετακίνηση με εισαγωγές και διαγραφές κόμβων.

Όσον αφορά το χρόνο που απαιτεί η μέθοδος για τη σύγκριση δύο υπηρεσιών διαδικτύου, παρατηρούμε ότι όταν χρησιμοποιείται στην πρώτη φάση της μεθόδου ο αλγόριθμος MM-DIFF, η μέθοδος απαιτεί από 110 έως 150 millisecond, ενώ όταν χρησιμοποιείται ο αλγόριθμος MH-DIFF απαιτούνται 300 έως 600 millisecond κατά

μέσο όρο. Με άλλα λόγια, ο αλγόριθμος MH-DIFF μπορεί να απαιτήσει έως και 6 φορές περισσότερο χρόνο από ότι ο MM-DIFF.

Συνολικά, η δεύτερη προσέγγιση της μεθόδου είναι αυτή που φαίνεται να ικανοποιεί σε μεγαλύτερο βαθμό τα περισσότερα από τα επιθυμητά χαρακτηριστικά που καθορίστηκαν στην παράγραφο 4.1.4. Πιο συγκεκριμένα, μια συνοπτική σύγκριση μεταξύ των δύο προσεγγίσεων σε σχέση με τα χαρακτηριστικά αυτά παρουσιάζεται στον Πίνακα 5.1.

Πίνακας 5.1 Συγκριτικός πίνακας των δύο προσεγγίσεων της προτεινόμενης μεθόδου

Επιθυμητά χαρακτηριστικά	**Πρώτη προσέγγιση (MM-DIFF)**	**Δεύτερη προσέγγιση (MH-DIFF)**
Αντιστοίχηση μεθόδων	100% επιτυχία	100% επιτυχία
Καταλληλότητα αλγορίθμου σύγκρισης για τη φύση των συγκεκριμένων αλλαγών	Ικανοποιητική απόδοση των αλλαγών σχετικά με εισαγωγή μεμονωμένων κόμβων και υποδένδρων αλλά η απόδοση υποβαθμίζεται σημαντικά στην περίπτωση που οι αλλαγές περιλαμβάνουν μετακινήσεις ή διαγραφές ενδιάμεσων κόμβων	Σταθερά ικανοποιητική απόδοση σε όλα τα πρότυπα αλλαγών
Ικανότητα εφαρμογής παραγόμενων σεναρίων τροποποίησης	Τα παραγόμενα σενάρια στις περιπτώσεις της διαγραφής ενδιάμεσων κόμβων και μετακινήσεων είναι πολύ μακριά από τις πραγματικές προγραμματιστικές αλλαγές που απαιτούνται	Τα παραγόμενα σενάρια στην συντριπτική τους πλειοψηφία δίνουν στον προγραμματιστή ρεαλιστικές κατευθυντήριες γραμμές
Ελαχιστοποίηση του κόστους	Βέλτιστο κόστος στην περίπτωση των εισαγωγών αλλά σημαντική απόκλιση στην περίπτωση των ενδιάμεσων διαγραφών και των μετακινήσεων	Πολύ κοντά στο βέλτιστο σενάριο ακόμα και στις περιπτώσεις που οι αλλαγές ήταν ιδιαίτερα εκτεταμένες και συνδυαστικές
Ελαχιστοποίηση χρόνου εκτέλεσης	Σταθερός σε σχέση με την έκταση των αλλαγών και σημαντικά μικρότερος	Διακυμάνσεις ανάλογα με τη φύση των αλλαγών και κατά μέσο όρο 5 φορές πιο αργός

Συνοψίζοντας, είναι φανερό ότι η δεύτερη προσέγγιση αποτελεί πληρέστερη λύση του προβλήματος. Τόσο η ποσοτική (κόστη) όσο και η ποιοτική (ρεαλιστικότητα σεναρίων τροποποίησης) ανάλυση των αποτελεσμάτων αποδεικνύουν ότι αποτελεί την καταλληλότερη προσέγγιση για το πρόβλημα. Ωστόσο, η μεγάλη διαφορά στο χρόνο εκτέλεσης και η άμβλυνση της διαφοράς για συγκεκριμένους τύπους αλλαγών (εισαγωγές μεμονωμένων παραμέτρων ή πολύπλοκων τύπων) είναι πιθανό να καθιστούν την πρώτη προσέγγιση προτιμότερη για πολύ συγκεκριμένες περιπτώσεις, κυρίως όταν ο χρόνος εκτέλεσης αποτελεί σημαντικό περιορισμό.

ΚΕΦΑΛΑΙΟ 6. ΕΠΙΛΟΓΟΣ

6.1. Ανακεφαλαίωση

Στην εργασία αυτή, μελετήθηκε το θέμα της συντήρησης υπηρεσιών διαδικτύου. Πιο συγκεκριμένα, αναπτύχθηκε μια μέθοδος για την ανίχνευση αλλαγών μεταξύ υπηρεσιών διαδικτύου, όπου η μια υπηρεσία διαδικτύου αποτελεί τροποποιημένη έκδοση της άλλης («παραγόμενη» υπηρεσία διαδικτύου). Σχετικές εργασίες, που έχουν μελετήσει το θέμα της αναζήτησης παρόμοιων υπηρεσιών, το έχουν προσεγγίσει σε σημασιολογικό επίπεδο. Σε αυτή την εργασία, έγινε ανίχνευση αλλαγών ως προς τη δομή των μηνυμάτων που ανταλλάσσονται κατά την κλήση μιας υπηρεσίας διαδικτύου. Η μελέτη αυτή είχε σαν στόχο να αντιστοιχήσει τις αλλαγές στη δομή των μηνυμάτων σε αλλαγές που πρέπει να γίνουν σε προγραμματιστικό επίπεδο. Αποτέλεσμα της μελέτης είναι η παραγωγή εφικτών και ρεαλιστικών σεναρίων τροποποίησης που μπορούν να καθοδηγήσουν τον προγραμματιστή που χρησιμοποιεί την αρχική υπηρεσία στην εφαρμογή του ώστε να χρησιμοποιήσει την τροποποιημένη έκδοση της υπηρεσίας αυτής.

Όπως αναφέρθηκε στο Κεφάλαιο 4, μια WSDL περιγραφή είναι ιεραρχικά δομημένη. Συγκεκριμένα, μια WSDL περιγραφή παρέχει ένα σύνολο λειτουργιών και κάθε λειτουργία λαμβάνει ένα μήνυμα εισόδου και αποστέλλει ένα μήνυμα εξόδου. Λόγω της ιεραρχικής δομής, τα μηνύματα εισόδου και εξόδου της κάθε λειτουργίας, μοντελοποιήθηκαν σε δένδρα.

Η προτεινόμενη μέθοδος αποτελείται από δύο φάσεις. Στην πρώτη φάση της μεθόδου, ανιχνεύθηκαν οι αλλαγές μεταξύ της κάθε λειτουργίας της αρχικής υπηρεσίας και της κάθε λειτουργίας της τροποποιημένης έκδοσής της. Για την

ανίχνευση αλλαγών μεταξύ δύο δένδρων, χρησιμοποιήθηκαν γνωστοί αλγόριθμοι ανίχνευσης αλλαγών. Για κάθε ζεύγος μηνύματος εισόδου μιας λειτουργίας της αρχικής υπηρεσίας και μηνύματος εισόδου μιας λειτουργίας της τροποποιημένης έκδοσής της, παράχθηκε ένα σενάριο τροποποίησης. Το άθροισμα του κόστους κάθε λειτουργίας τροποποίησης στο σενάριο, αποτελεί το κόστος μετατροπής του αρχικού μηνύματος εισόδου στο τροποποιημένο. Η ίδια διαδικασία ακολουθήθηκε και για κάθε ζεύγος μηνυμάτων εξόδου. Το κόστος μεταξύ δύο λειτουργιών ορίστηκε ως το άθροισμα του κόστους μεταξύ των μηνυμάτων εισόδου και των μηνυμάτων εξόδου.

Στην δεύτερη φάση της μεθόδου, έγινε μια ένα-προς-ένα αντιστοίχηση μεταξύ των λειτουργιών της αρχικής και της τροποποιημένης υπηρεσίας. Από την πρώτη φάση της μεθόδου έχει υπολογιστεί το κόστος τροποποίησης μεταξύ κάθε δύο λειτουργιών που ανήκουν στην αρχική και τροποποιημένη υπηρεσία αντίστοιχα. Η ένα-προς-ένα αντιστοίχηση έγινε με κριτήριο το ελάχιστο κόστος. Το άθροισμα του κόστους μεταξύ των λειτουργιών που αντιστοιχήθηκαν, αποτελεί το συνολικό κόστος της μετατροπής της αρχικής υπηρεσίας στην τροποποιημένη έκδοσή της.

Το αποτέλεσμα της μεθόδου ανίχνευσης αλλαγών, είναι η παραγωγή των σεναρίων τροποποίησης, τα οποία μπορούν να χρησιμοποιηθούν σαν καθοδήγηση από τον προγραμματιστή, που χρησιμοποιεί την αρχική υπηρεσία διαδικτύου στην εφαρμογή του και επιθυμεί να χρησιμοποιήσει την τροποποιημένη έκδοσή της.

6.2. Μελλοντικές επεκτάσεις

Όπως περιγράφτηκε στο Κεφάλαιο 4, διαθέτοντας μόνο τις WSDL περιγραφές δύο υπηρεσιών διαδικτύου, είναι αδύνατον να υπολογίσουμε το πραγματικό προγραμματιστικό κόστος μιας αλλαγής σε επίπεδο WSDL σε μια εφαρμογή που χρησιμοποιεί την υπηρεσία διαδικτύου. Για το λόγο αυτό, το κόστος κάθε λειτουργίας τροποποίησης, έχει προσεγγιστεί με μια τιμή. Μια προτεινόμενη επέκταση, είναι η σύνδεση της μεθόδου ανίχνευσης αλλαγών, με ένα πρόγραμμα που θα δέχεται σαν είσοδο τις WSDL περιγραφές δύο υπηρεσιών διαδικτύου καθώς και τον κώδικα της εφαρμογής που χρησιμοποιεί την αρχική υπηρεσία διαδικτύου. Το πρόγραμμα αυτό, θα υπολογίζει το πραγματικό προγραμματιστικό κόστος κάθε αλλαγής και θα το δίνει σαν είσοδο στη μέθοδο ανίχνευσης αλλαγών. Με αυτό τον τρόπο, τα παραγόμενα σενάρια τροποποίησης θα μπορούν να καθοδηγήσουν καλύτερα τον προγραμματιστή

ως προς τις αλλαγές που πρέπει να κάνει στην εφαρμογή του ώστε να χρησιμοποιήσει την τροποποιημένη έκδοση της αρχικής υπηρεσίας διαδικτύου.

ΑΝΑΦΟΡΕΣ

[1] Al-Masri, E., and Mahmoud, Q. H. Discovering the best web service, (poster) 16th International Conference on World Wide Web (WWW), 2007, pp. 1257-1258.

[2] Al-Masri, E., and Mahmoud, Q. H. QoS-based Discovery and Ranking of Web Services, IEEE 16th International Conference on Computer Communications and Networks (ICCCN), 2007, pp. 529-534.

[3] Augustine, V., Leon, D., Podgurski, A., Raghavan, S. and Rohana, R. Dex: A Semantic-Graph Differencing Tool for Studying Changes in Large Code Base. Proc. Int'l Conf. Software Maintenance, pp. 188-197, Sept. 2004.

[4] Bernstein, A., Kiefer, C., Pinzger, M. and Sager, T. Detecting Similar Java Classes Using Tree Algorithms, Proc. Int'l Workshop Mining Software Repositories, pp. 65-71, May 2006.

[5] Chawathe, S. Comparing Hierarchical Data in External Memory, Proceedings of the 25th International Conference on Very Large Data Bases, Sept. 1999.

[6] Chawathe, S., Garcia-Molina, H. Meaningful change detection in structured data, Proceedings of the 1997 ACM SIGMOD international conference on Management of data, p. 26-37, May 1997.

[7] Chawathe, S., Rajaraman, A., Garcia-Molina, H., Widom, J. Change Detection in Hierarchically Structured Information. Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, p. 493-504, June 1996.

[8] Collard, M.L. and Maletic, J.I. Supporting Source Code Difference Analysis, Proc. Int'l Conf. Software Maintenance, pp. 210-219, Sept. 2004.

[9] Dong, X., Halevy, A., Madhavan, J., Nemes, W. and Zhang, J. Similarity Search for Web Services, Proceedings of the 30th International Conference on Very Large Data Bases, 2004.

[10] Fluri, B., Gall, H.C., Pinzger, M. and Wursch, M. Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction, IEEE Transactions on Software Engineering, vol. 33, no. 11, pp. 725-743, Nov. 2007.

[11] Fremantle, P., Weerawarana, S., & Khalaf, R. Enterprise Services, Communications of the ACM, Vol. 45.No 10, pp. 77-82, October 2002.

[12] Horwitz, S. Identifying the Semantic and Textual Differences between Two Versions of a Program. Proc. ACM SIGPLAN Conf. Programming Language Design and Implementation, pp. 234-245, June 1990.

[13] Hunt, J.W. and Szymanski, T.G. A Fast Algorithm for Computing Longest Common Subsequences, Comm. ACM, vol. 20, no. 5, pp. 350-353, May 1977.

[14] Klein, P.N. Computing the edit-distance between unrooted ordered trees. Proceedings of the 6th annual European Symposium on Algorithms (ESA) 1998, p. 91-102, Springer-Verlag, 1998.

[15] Papadimitriou, C. and Steiglitz, K. Combinatorial Optimization. Prentice-Hall, 1982.

[16] Stoulia, E. and Xing, Z. UMLDiff: An Algorithm for Object-Oriented Design Differencing. Proc. Int'l Conf. Automated Software Eng., pp. 54-65, Nov. 2005.

[17] Tai, K. The tree-to-tree correction problem. Journal of the Association for Computing Machinery (JACM), 26:422-422, 1979.

[18] Wagner, R., and Fischer, M. The string-to-string correction problem. Journal of the ACM (JACM), 21:168-173, 1974.

[19] Zhang, K., and Shasha, D. Simple fast algorithms for the editing distance between trees and related problems. SIAM Journal of Computing, 18:1245-1262, 1989.

[20] Zhang, K., Statman, R. and Shasha, D. On the editing distance between unordered labeled trees, Information Processing Letters, 42:133-19, 1992.

[21] World Wide Web Consortium, Web Service Description Language (WSDL) 1.1, available at http://www.w3.org/TR/wsdl, January 2006.

[22] World Wide Web Consortium, XML Schema Part 2: Datatypes Second Edition, available at http://www.w3.org/TR/xmlschema-2/, October 2004.

[23] W3Schools, XML Schema Reference, available at http://www.w3schools.com/schema/schema_elements_ref.asp

ΠΑΡΑΡΤΗΜΑ

Πίνακας 6.1 Λειτουργίες της υπηρεσίας 103907719.wsdl

Όνομα Μεθόδου	Αριθμός στοιχείων μηνύματος εισόδου	Αριθμός στοιχείων μηνύματος εξόδου
runAndWaitFor	28	10
createAndRun	28	5
waitFor	5	3
getStatus	5	14
getResults	5	10

Πίνακας 6.2 Λειτουργίες της υπηρεσίας 105393659.wsdl

Όνομα Μεθόδου	Αριθμός στοιχείων μηνύματος εισόδου	Αριθμός στοιχείων μηνύματος εξόδου
GetCallPermssion	12	7
GetACNielsenCitySizes	10	13
GetSrLinkCitySizes	10	13
GetCountryFlag	11	5
GetCountryInfo	11	51
GetCountryList	10	12
GetSRLinkCountryList	10	12
GetContinentList	10	12
GetCurrencyList	10	12
GetLanguageList	10	12
GetTimeZoneList	10	13
GetCountryListByCurrency	11	12
GetCountryListByContinent	11	12
GetCountryLanguageList	5	12

Πίνακας 6.3 Λειτουργίες της υπηρεσίας 101648622.wsdl

Όνομα Μεθόδου	Αριθμός στοιχείων μηνύματος εισόδου	Αριθμός στοιχείων μηνύματος εξόδου
runAndWaitFor	67	10
createAndRun	67	5
waitFor	5	3

getStatus	5	14
getResults	5	10

Πίνακας 6.4 Λειτουργίες της υπηρεσίας 112110367.wsdl

Όνομα Μεθόδου	Αριθμός στοιχείων μηνύματος εισόδου	Αριθμός στοιχείων μηνύματος εξόδου
runAndWaitFor	47	15
createAndRun	47	5
waitFor	5	3
getStatus	5	14
getResults	5	15

Πίνακας 6.5 Λειτουργίες της υπηρεσίας 125052688.wsdl

Όνομα Μεθόδου	Αριθμός στοιχείων μηνύματος εισόδου	Αριθμός στοιχείων μηνύματος εξόδου
getRegExpDetails	5	17
ListAllAsXml	5	25
listRegExp	8	9

Πίνακας 6.6 Λειτουργίες της υπηρεσίας 139707792.wsdl

Όνομα Μεθόδου	Αριθμός στοιχείων μηνύματος εισόδου	Αριθμός στοιχείων μηνύματος εξόδου
createCmap	16	5
createFolder	25	5
createResource	16	5
delete	14	4
getCmap	14	5
getResource	14	5
getResourceMeta	14	5
getResourceMetaList	14	5
getRootResourceMeta	13	5
getRootResourceMetaList	13	5
saveResource	15	5
saveCmap	15	5
setResourceMeta	14	5

Πίνακας 6.7 Λειτουργίες της υπηρεσίας 157739104.wsdl

Όνομα Μεθόδου	Αριθμός στοιχείων μηνύματος εισόδου	Αριθμός στοιχείων μηνύματος εξόδου
CreateIssue	7	5
CreateNewIssue	8	5
FindIssues	12	9
GetIssue	7	9
GetIssueLookups	7	9
UpdateIssue	12	3
GetReport	14	5
StartBriqTimer	10	3
StopBriqTimer	7	3
BriqTimerRunning	10	5
GetRunningTimers	7	9
LogException	18	3

Πίνακας 6.8 Λειτουργίες της υπηρεσίας 185543971.wsdl

Όνομα Μεθόδου	Αριθμός στοιχείων μηνύματος εισόδου	Αριθμός στοιχείων μηνύματος εξόδου
findFeatures	57	51
deleteFeatures	10	3
geocodeFeatures	17	57
updateDataFile	44	3
addFeatures	42	12
getAvailableDataFileAttributes	5	16
updateFeatures	42	3
createDataFile	43	3
getUniqueValues	12	12
getVersion	3	5
getDataFilesInfo	9	78
renameField	8	3
deleteDataFile	6	3

Πίνακας 6.9 Λειτουργίες της υπηρεσίας 157871984.wsdl

Όνομα Μεθόδου	Αριθμός στοιχείων μηνύματος εισόδου	Αριθμός στοιχείων μηνύματος εξόδου
Send	43	35
SendWithNoReplyCode	35	35
SendMultiple	49	43
GetMessagesByGroupKey	8	38
GetMessageByMessageKey	5	35
GetAllMessages	7	38

GetPollStartPoint	3	5
GetMessagesPoll	8	42
GetMessagesInRange	9	38
GetMessagesByCategory	15	38
GetMessagesByDestination	10	38
CancelReplies	5	3
CancelMultipleReplies	8	13
GetPrice	35	5
GetPrices	38	13

Πίνακας 6.10 Λειτουργίες της υπηρεσίας 109361683.wsdl

Όνομα Μεθόδου	Αριθμός στοιχείων μηνύματος εισόδου	Αριθμός στοιχείων μηνύματος εξόδου
runAndWaitFor	29	11
createAndRun	29	5
waitFor	5	3
getStatus	5	14
getResults	5	11

ΣΥΝΤΟΜΟ ΒΙΟΓΡΑΦΙΚΟ

Η Ευφροσύνη Κουρή γεννήθηκε το 1983 στην Αθήνα. Αποφοίτησε από το σχολείο Ι.Μ. Παναγιωτόπουλος το 2001 και την ίδια χρονιά εισήχθη στο προπτυχιακό πρόγραμμα σπουδών του Τμήματος Πληροφορικής του Πανεπιστημίου Ιωαννίνων. Ολοκλήρωσε τις προπτυχιακές της σπουδές τον Σεπτέμβριο του 2006 και το ίδιο ακαδημαϊκό έτος ξεκίνησε τις μεταπτυχιακές της σπουδές στο ίδιο τμήμα.
Από τον Ιούλιο του 2005 εργάζεται στην εταιρεία Comitech Α.Ε. στα Ιωάννινα, ενώ από τον Ιούνιο του 2009 είναι μέτοχος και ιδρυτικό στέλεχος της "eFusion - Ανάπτυξη Λογισμικού" Ε.Π.Ε.