ΣΗΜΑΣΙΟΛΟΓΙΚΗ ΑΝΑΠΡΟΣΑΡΜΟΓΗ ΥΠΗΡΕΣΙΟ-ΚΕΝΤΡΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ ΔΙΑΧΥΤΟΥ ΥΠΟΛΟΓΙΣΜΟΎ

Η
ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

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

από τον

Διονύση Αθανασόπουλο

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΟ ΛΟΓΙΣΜΙΚΟ

Ιούνιος 2008

ΑΦΙΕΡΩΣΗ

Αφιερώνω την παρούσα εργασία στην οικογένειά μου και την Α. Κάτσενου.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον επιβλέποντά μου Απόστολο Ζάρρα για την πολύτιμη συμβολή του στην περάτωση της παρούσας εργασίας, καθώς και την οικογένειά μου για τη στήριξη της προσπάθειάς μου.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

ΠΕΡΙΛΗΨΗ

Διονύσης Αθανασόπουλος του Νικολάου και της Γεωργίας. MSc, Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Ιούλιος, 2008. Σημασιολογική αναπροσαρμογή υπηρεσιο-κεντρικών συστημάτων διάχυτου υπολογισμού. Επιβλέποντας: Απόστολος Ζάρρας.

Στην παρούσα εργασία αντιμετωπίζουμε το πρόβλημα της μεταβαλλόμενης διαθεσιμότητας υπηρεσιών διαδικτύου σε περιβάλλοντα διάχυτου υπολογισμού. Πιο συγκεκριμένα, μια υπηρεσία διαδικτύου μπορεί, ανά πάσα στιγμή, να πάψει να είναι διαθέσιμη. Σε ένα τέτοιο σενάριο, η εφαρμογή που παίζει το ρόλο του πελάτη προς τη μη διαθέσιμη υπηρεσία, θα πάψει να λειτουργεί. Παρόλα αυτά, στο περιβάλλον είναι πιθανόν να είναι διαθέσιμες άλλες αυτόνομες υπηρεσίες που προσφέρουν την ίδια λειτουργικότητα. Στην περίπτωση αυτή, θα ήταν ιδανικό να αναπροσαρμοστεί η εφαρμογή που παίζει το ρόλο του πελάτη με στόχο την χρήση κάποιας εξ αυτών των υπηρεσιών αντί της μη διαθέσιμης υπηρεσίας. Η αναπροσαρμογή της εφαρμογής θα ήταν εφικτή και εύκολη αν οι υπηρεσίες που μπορούν να αντικαταστήσουν τη μη διαθέσιμη υπηρεσία έχουν την ίδια προγραμματιστική διεπαφή με τη μη διαθέσιμη υπηρεσία. Κάτι τέτοιο όμως είναι γενικά δύσκολο να ισχύει στην πράξη σε συστήματα διάχυτου υπολογισμού, αποτελούμενα από αυτόνομες υπηρεσίες που έχουν υλοποιηθεί και εγκατασταθεί ανεξάρτητα.

Με βάση τα παραπάνω, στόχος της παρούσας εργασίας είναι μια προσέγγιση που επιτρέπει την αντικατάσταση μη διαθέσιμων υπηρεσιών από άλλες, οι οποίες προσφέρουν την ίδια λειτουργικότητα μέσω διαφορετικών διεπαφών, με τρόπο κλειστό ως προς τον κώδικα των εφαρμογών που χρησιμοποιούν αυτές τις υπηρεσίες, δηλ. χωρίς να απαιτούνται ριζικές αλλαγές στον κώδικα αυτών των εφαρμογών.

EXTENDED ABSTRACT IN ENGLISH

Dionisis Athanasopoulos. MSc, Computer Science Department, University of Ioannina, Greece. June, 2008. Semantic Reconfiguration of pervasive computing systems. Thesis Supervisor: Apostolos Zarras.

The present thesis deals with the dynamics of the web services' availability in pervasive computing environments. Specifically, from time to time a web service may not be available to the user. In such a scenario the client application will no longer execute correctly. However, some other autonomous services offering the same functionality may be available in the environment. In this case, the ideal solution would be the adaptation of the client service in such a manner that one of the other available web services will be used instead of the old non-available. The service adaptation is easy and effective, if the "candidate" services have the same WSDL interface with the non-available service. The difficulty of implementing this idea lies on the fact that the pervasive computing environments are composed by autonomous implemented and installed services that offer the same functionality through different interfaces.

The present thesis aims at a offering an innovative approach, which allows the non-available services replacement with other services that offer the same functionality through different interfaces and without requiring radical changes in the applications' code.

In order to achieve the set goal, we propose a system that has been implemented in this thesis frame and is based on the following ideas:

1. Organization of the services which provide the same functionality through different interfaces into categories that are characterized by abstract semantic descriptions of the services' functionalities. The semantic descriptions refer to the OWL-S markup language.

2. Search of services which may replace a non-available service based on the abstract semantic descriptions of the services' functionalities and the aforementioned available services organization.
3. Replacement of the non-available service without requiring radical changes in the applications' code according to the Adapter Design Pattern.

ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΗ

1.1. Ορισμός Προβλήματος - Στόχοι

Η εμφάνιση των φορητών υπολογιστών και των ασύρματων δικτύων (wireless LANs) οδήγησε στην ανάπτυξη κατανεμημένων συστημάτων που αποτελούνται από κινητούς πελάτες και εξυπηρετητές. Τα συστήματα αυτά αποκαλούνται συχνά και συστήματα κινητού υπολογισμού (mobile computing systems). Μία εφαρμογή των συστημάτων κινητού υπολογισμού είναι τα συστήματα διάχυτου υπολογισμού (pervasive computing). Βασικός στόχος των συστημάτων αυτών είναι η υποστήριξη των καθημερινών δραστηριοτήτων των χρηστών στα περιβάλλοντα στα οποία κινούνται οι χρήστες.

Τα συστήματα διάχυτου υπολογισμού βασίζονται συχνά σε αρχιτεκτονικές προσανατολισμένες στην προσφορά υπηρεσιών (Service Oriented Architecture – SOA). Πιο συγκεκριμένα, οι καθημερινές δραστηριότητες των χρηστών υποστηρίζονται από υπηρεσίες οι οποίες προσφέρονται από αυτόνομες εφαρμογές και έχουν εγκατασταθεί στο περιβάλλον των χρηστών. Οι εφαρμογές που είναι εγκατεστημένες στις κινητές συσκευές των χρηστών χρησιμοποιούν τις υπάρχουσες υπηρεσίες μέσω των προγραμματιστικών διεπαφών τους, με άλλα λόγια, παίζουν το ρόλο του πελάτη προς τις υπηρεσίες αυτές.

Με βάση τα παραπάνω, στην παρούσα εργασία αντιμετωπίζουμε το πρόβλημα της μεταβαλλόμενης διαθεσιμότητας υπηρεσιών. Πιο συγκεκριμένα, μια υπηρεσία στην οποία βασίζεται μια εφαρμογή, η οποία παίζει το ρόλο του πελάτη, μπορεί, ανά πάσα στιγμή, να πάψει να είναι διαθέσιμη για διάφορους λόγους, όπως η μετακίνηση του

χρήστη σε ένα άλλο περιβάλλον, η απεγκατάσταση της υπηρεσίας από τον ίδιο τον πάροχό της, σφάλματα στην εκτέλεση της υπηρεσίας, κλπ. Σε ένα τέτοιο σενάριο η εφαρμογή που παίζει το ρόλο του πελάτη προς τη μη διαθέσιμη υπηρεσία θα πάψει να λειτουργεί. Παρόλα αυτά, στο περιβάλλον είναι πιθανόν να είναι διαθέσιμες άλλες αυτόνομες υπηρεσίες που προσφέρουν την ίδια λειτουργικότητα. Στην περίπτωση αυτή, θα ήταν ιδανικό να αναπροσαρμοστεί η εφαρμογή που παίζει το ρόλο του πελάτη με στόχο την χρήση κάποιας εξ αυτών των υπηρεσιών αντί της μη διαθέσιμης υπηρεσίας. Η αναπροσαρμογή της εφαρμογής θα ήταν εφικτή και εύκολη αν οι υπηρεσίες που μπορούν να αντικαταστήσουν τη μη διαθέσιμη υπηρεσία έχουν την ίδια προγραμματιστική διεπαφή με τη μη διαθέσιμη υπηρεσία. Κάτι τέτοιο όμως είναι γενικά δύσκολο να ισχύει στην πράξη σε συστήματα διάχυτου υπολογισμού, αποτελούμενα από αυτόνομες υπηρεσίες που έχουν υλοποιηθεί και εγκατασταθεί ανεξάρτητα. Το πλέον σύνηθες σενάριο στην πράξη είναι η ύπαρξη υπηρεσιών που προσφέρουν ίδια ή παρόμοια λειτουργικότητα μέσω διαφορετικών προγραμματιστικών διεπαφών. Στην περίπτωση αυτή, η αναπροσαρμογή της εφαρμογής απαιτεί σημαντικές αλλαγές του κώδικα της, έτσι ώστε να γίνεται χρήση σε αυτόν της διεπαφής μιας νέας υπηρεσίας η οποία μπορεί να αντικαταστήσει μια υπηρεσία που πλέον δεν είναι διαθέσιμη. Η προσέγγιση αυτή γενικά έχει μεγάλο κόστος συντήρησης. Επίσης από πρακτική άποψη σε ένα περιβάλλον διάχυτου υπολογισμού δεν είναι ρεαλιστική. Δεν είναι δυνατόν να υπάρχει η απαίτηση ένας απλός χρήστης να έχει προγραμματιστικές δυνατότητες που θα του επιτρέπουν να αναπροσαρμόζει ο ίδιος τον κώδικα των εφαρμογών που χρησιμοποιεί.

Με βάση τα παραπάνω, στόχος της παρούσας εργασίας είναι μια προσέγγιση που επιτρέπει την αντικατάσταση μη διαθέσιμων υπηρεσιών από άλλες, οι οποίες προσφέρουν την ίδια λειτουργικότητα μέσω διαφορετικών διεπαφών, με τρόπο κλειστό ως προς τον κώδικα των εφαρμογών που χρησιμοποιούν αυτές τις υπηρεσίες, δηλ. χωρίς να απαιτούνται ριζικές αλλαγές στον κώδικα αυτών των εφαρμογών.

Για την επίτευξη αυτού του στόχου προτείνουμε ένα σύστημα, το οποίο υλοποιήσαμε στα πλαίσια αυτής της εργασίας και βασίζεται στις παρακάτω ιδέες:

1. Οργάνωση υπηρεσιών που παρέχουν παρόμοια λειτουργικότητα μέσω διαφορετικών διεπαφών σε κατηγορίες οι οποίες χαρακτηρίζονται από αφηρημένες σημασιολογικές περιγραφές της λειτουργικότητας των

υπηρεσιών. Οι σημασιολογικές περιγραφές βασίζονται στην πρότυπη γλώσσα OWL-S.

2. Αναζήτηση υπηρεσιών που μπορούν να αντικαταστήσουν μια μη διαθέσιμη υπηρεσία με βάση την αφηρημένη σημασιολογική περιγραφή της λειτουργικότητας που προσφέρουν αυτές οι υπηρεσίες και την προαναφερθείσα κατηγοριοποίηση διαθέσιμων υπηρεσιών.
3. Αντικατάσταση της μη διαθέσιμης υπηρεσίας με τρόπο κλειστό ως προς τον κώδικα των εφαρμογών που τη χρησιμοποιούν με βάση το Προσαρμοστικό Σχεδιαστικό Πρότυπο (Adapter Design Pattern [7]).

1.2. Δομή της Διατριβής

Η διατριβή περιέχει 6 κεφάλαια: Το Κεφάλαιο 1 ορίζει το πρόβλημα της σημασιολογικής αναπροσαρμογής συστημάτων λογισμικού που θα αντιμετωπιστεί στην παρούσα εργασία. Το Κεφάλαιο 2 αναλύει σχετικές προσεγγίσεις σε αυτό το πρόβλημα. Το Κεφάλαιο 3 παρουσιάζει την προτεινόμενη προσέγγιση της σημασιολογικής αναπροσαρμογής που θα υλοποιηθεί στην παρούσα εργασία. Το Κεφάλαιο 4 σχεδιάζει και αναλύει το σύστημα λογισμικού που αναπτύχθηκε. Το Κεφάλαιο 5 παρουσιάζει τις πειραματικές μετρήσεις που πραγματοποιήθηκαν. Το Κεφάλαιο 6 παρουσιάζει τα συμπεράσματα που προέκυψαν καθώς και ορισμένες προτάσεις για μελλοντικές επεκτάσεις.

ΚΕΦΑΛΑΙΟ 2. ΣΧΕΤΙΚΕΣ ΠΡΟΣΕΓΓΙΣΕΙΣ ΣΤΗΝ ΑΝΑΠΡΟΣΑΡΜΟΓΗ ΣΥΣΤΗΜΑΤΩΝ

2.1. Αναπροσαρμογή σταθερών κατανεμημένων συστημάτων

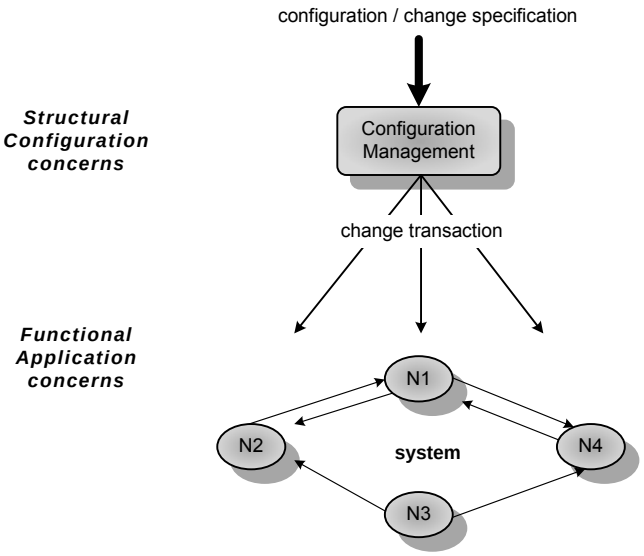
Η αρχιτεκτονική των κατανεμημένων συστημάτων μοντελοποιείται ως μία συλλογή από συνιστώντα μέρη (components) και από συνδέσμους (connectors). Ένα τέτοιο σύστημα μπορεί να αναπαρασταθεί ως ένα γράφημα, όπου οι κόμβοι είναι τα συνιστώντα μέρη και οι ακμές είναι οι σύνδεσμοι.

Ένα από τα βασικά χαρακτηριστικά των κατανεμημένων συστημάτων είναι οι αλλαγές που συμβαίνουν στο περιβάλλον του συστήματος αλλά και στις ανάγκες του εκάστοτε χρήστη. Παρόλο που το κατανεμημένο σύστημα θα έχει υλοποιηθεί, είναι απαραίτητο να προσαρμόζεται στις αλλαγές δυναμικά. Η αναπροσαρμογή του πρακτικά σημαίνει δυναμική προσθήκη, αντικατάσταση ή διαγραφή κάποιων κόμβων και κατά συνέπεια κάποιων συνδέσμων του συστήματος σύμφωνα με τον τρόπο μοντελοποίησης που αναφέρθηκε παραπάνω.

Η έννοια της αναπροσαρμογής (reconfiguration) ξεκίνησε αρχικά από τα σταθερά κατανεμημένα συστήματα (stationary distributed systems). Για να λυθεί το πρόβλημα ακολουθήθηκε η εξής προσέγγιση [1]: η αρχιτεκτονική του συστήματος μοντελοποιήθηκε σε δύο επίπεδα. Το πρώτο επίπεδο αφορούσε την λειτουργική περιγραφή των υποσυστημάτων του (software components) και το δεύτερο επίπεδο αφορούσε τις πληροφορίες για την κατάσταση του συστήματος (Σχήμα 2.1). Οι πληροφορίες για την κατάστασή του αποτελούν το τρέχων στιγμιότυπο του

συστήματος (system configuration). Αυτές οι πληροφορίες βοηθούν στην αναπροσαρμογή του συστήματος. Σε αυτά τα συστήματα υλοποιήθηκε μια επιπλέον οντότητα που ονομάστηκε διαχειριστής αλλαγών (reconfiguration manager) και συγκέντρωνε τα ακόλουθα βασικά χαρακτηριστικά:

1. Η αναπροσαρμογή του συστήματος είναι ανεξάρτητη από τους αλγορίθμους και τα πρωτόκολλα που χρησιμοποιεί το σύστημα.
2. Ο διαχειριστής των αλλαγών προσδιορίζει το ελάχιστο (minimal) σύνολο των κόμβων που επηρεάζεται από τις αλλαγές και διακόπτει την εκτέλεσή τους. Οι υπόλοιποι κόμβοι συνέχιζαν να εκτελούνται κανονικά.
3. Μετά την αναπροσαρμογή του, το σύστημα πρέπει να είναι σε συνεπή κατάσταση (consistent state).



Σχήμα 2.1 Στιγμιότυπο του συστήματος

Το επόμενο ερώτημα που έπρεπε να απαντηθεί ήταν το πώς θα εφαρμόζονταν οι αλλαγές στο σύστημα, ώστε να καταλήξει σε συνεπή κατάσταση. Μία τεχνική η οποία προτάθηκε [2] ήταν να απομονωθεί το τμήμα του συστήματος το οποίο επηρεαζόταν από τις αλλαγές και στην συνέχεια να μπλοκαριστούν όλες οι αιτήσεις προς αυτό το τμήμα (request blocking). Μια άλλη τεχνική [4] ήταν να μεταφερθούν όλες οι αιτήσεις προς άλλα τμήματα τα οποία μπορούν να ικανοποιήσουν αυτές τις αιτήσεις (request redirection).

2.2. Αναπροσαρμογή συστημάτων διάχυτου υπολογισμού

Η εμφάνιση των φορητών υπολογιστών και των ασύρματων τοπικών δικτύων (wireless LANs) οδήγησε στην ανάπτυξη των κινητών κατανεμημένων συστημάτων και των συστημάτων διάχυτου υπολογισμού. Οι βασικές αρχές των κλασικών κατανεμημένων συστημάτων συνεχίζουν να χρησιμοποιούνται. Όμως, η κινητικότητα των πελατών και των εξυπηρετητών πρόσθεσε νέα χαρακτηριστικά:

1. την απρόβλεπτη ύπαρξη κάποιου κόμβου στο δίκτυο
2. τους περιορισμούς σχετικά με τους πόρους των συσκευών που χρησιμοποιούν οι κινητοί κόμβοι (μνήμη, υπολογιστική ισχύς, μπαταρία κτλ)
3. τη συμβατότητα ετερογενών τεχνολογιών που χρησιμοποιούνται από τους κόμβους του συστήματος

Τα συστήματα τα οποία έχουν προταθεί και λαμβάνουν υπόψη τα παραπάνω χαρακτηριστικά των συστημάτων διάχυτου υπολογισμού, συνήθως ακολουθούν αρχιτεκτονικές προσανατολισμένες στην προσφορά υπηρεσιών (Service Oriented Architecture – SOA). Μια υπηρεσία είναι μία οντότητα η οποία προσφέρει ένα σύνολο λειτουργιών. Οι λειτουργίες της περιγράφονται από μία διεπαφή (interface). Για να πραγματοποιηθεί η κλήση της υπηρεσίας πρέπει να γνωρίζουμε την διεύθυνσή της. Οι υπηρεσίες διευθυνσιοδοτούνται με ένα URI (Uniform Resource Identifier).

Με δεδομένα τα παραπάνω χαρακτηριστικά, είναι πιθανόν κατά την εκτέλεση ενός συστήματος, ένας κινητός εξυπηρετητής να πάψει να είναι διαθέσιμος στο δίκτυο. Οι πελάτες που τον χρησιμοποιούσαν θα πρέπει να συνεχίσουν να εκτελούνται αφού επανασυνδεθούν σε κάποιον άλλον εξυπηρετητή ο οποίος προσφέρει την ίδια λειτουργικότητα. Αυτή είναι η έννοια της διαχείρισης αλλαγών στα κατανεμημένα συστήματα διάχυτου υπολογισμού. Η προσέγγιση αυτή θα πρέπει να διαφοροποιείται από την προηγούμενη για τα σταθερά κατανεμημένα συστήματα σε ένα βασικό σημείο: ο πελάτης δεν γνωρίζει εκ των προτέρων τους εξυπηρετητές που θα χρησιμοποιήσει. Πιο συγκεκριμένα, σε SOA συστήματα, ένας πελάτης δεν είναι σε θέση να γνωρίζει ούτε τα ονόματα αλλά ούτε και την διεπαφή των υπηρεσιών που θα χρησιμοποιήσει. Αντιθέτως, στα σταθερά συστήματα η διεπαφή του κόμβου που ήταν υποψήφια προς αντικατάσταση ήταν γνωστή εκ των προτέρων.

Σε SOA συστήματα, θα πρέπει να βρεθεί δυναμικά η διεπαφή της υποψήφιας προς αντικατάσταση υπηρεσίας. Οι γλώσσες που συνήθως χρησιμοποιούνται για να

περιγράψουν τις διεπαφές των υπηρεσιών είναι βασισμένες στην XML (WSDL, WSCL, WSCI, BPEL4WS, WS-CDL). Με δεδομένη μια XML περιγραφή, ένας κόμβος μπορεί να αναζητήσει μόνο τις συντακτικά συμβατές διεπαφές. Είναι όμως δυνατόν να υπάρχουν υπηρεσίες που έχουν την ίδια λειτουργικότητα αλλά δεν είναι συμβατές συντακτικά και επομένως θα πρέπει να περιγραφούν σε πιο αφηρημένο επίπεδο (abstract level).

Αυτό οδηγεί στο συμπέρασμα ότι για να μπορούν να ανακαλυφθούν οι υπηρεσίες από τα συστήματα διάχυτου υπολογισμού θα πρέπει να ομαδοποιηθούν σε οντολογίες. Έτσι δίνεται η δυνατότητα σε έναν πελάτη να κάνει σημασιολογική αναζήτηση της διεπαφής της υπηρεσίας που θα χρησιμοποιήσει.

Στην επόμενη παράγραφο αναλύονται 2 συστήματα τα οποία υλοποιούν μεθόδους διαχείρισης αλλαγών σε συστήματα διάχυτου υπολογισμού.

2.2.1. Το σύστημα RAPIDware

Το σύστημα RAPIDware [5] υλοποιεί διαχείριση αλλαγών βασισμένη σε συστήματα διάχυτου υπολογισμού που έχουν μοντελοποιηθεί ως μία συλλογή από συνιστώντα μέρη (components) και από συνδέσμους (βλ. ενότητα 2.1). Αυτό το σύστημα εφαρμόζεται στην ασύρματη μεταφορά βίντεο (wireless video streaming). Στόχος της διαχείρισης αλλαγών είναι η αντικατάσταση των συνιστώντων μερών του συστήματος από κάποια εναλλακτικά. Το σημαντικό είναι ότι όλα τα διαθέσιμα συνιστώντα μέρη είναι γνωστά εκ των προτέρων και έχουν αναπτυχθεί ώστε να ταιριάζουν μεταξύ τους ως προς την διεπαφή τους και ως προς την λειτουργική τους συμπεριφορά.

Το σύστημα RAPIDware χρησιμοποιεί έναν κεντρικό διαχειριστή αλλαγών (central reconfiguration manager - RM) ο οποίος έχει αποθηκευμένη την περιγραφή του τρέχοντος στιγμιότυπου του συστήματος. Σε αυτήν την περιγραφή κρατά πληροφορίες σχετικά με:

1. τις εξαρτήσεις των συνιστώντων μερών του συστήματος (dependency relationships)
2. τις επικοινωνιακές συνδέσεις μεταξύ των συνιστώντων μερών του συστήματος (communication segments).

Επιπλέον, η οντότητα RM έχει αποθηκευμένο το σύνολο όλων των ενεργειών (reconfiguration actions) που θα εφαρμοστούν για την διαχείριση των αλλαγών. Όταν συμβεί μία αλλαγή, η οντότητα RM πριν προχωρήσει σε κάποια ενέργεια, ελέγχει εάν ικανοποιούνται 2 προϋποθέσεις, δηλαδή ότι:

1. δεν παραβιάζονται οι εξαρτήσεις που των συνιστώντων μερών
2. δεν διακόπτονται επικοινωνιακές συνδέσεις.

Αφού ικανοποιούνται αυτές οι προϋποθέσεις, η οντότητα RM προσδιορίζει ένα σύνολο από ενέργειες οι οποίες μπορούν να εφαρμοστούν και παράγει ένα γράφημα ενεργειών. Τέλος, με τον αλγόριθμο του Dijkstra επιλέγει την συντομότερη ακολουθία αλλαγών.

2.2.2. Το σύστημα CASA

Στο σύστημα CASA [6] τα συνιστώντα μέρη είναι αντικείμενα (objects) μίας αντικειμενοστραφής γλώσσα προγραμματισμού. Αυτή η προσέγγιση εφαρμόζεται σε επίπεδο αντικειμένων. Ορίζεται ένα σύνολο από εναλλακτικές κλάσεις (alternative classes) οι οποίες χρησιμοποιούνται για την αντικατάσταση ενός αντικειμένου. Αυτό προϋποθέτει ότι οι εναλλακτικές κλάσεις:

1. έχουν κοινή διεπαφή
2. οι προ-συνθήκες (pre-conditions) και οι μετά-συνθήκες (post-conditions) των μεθόδων όλων των κλάσεων είναι ίδιες
3. και ότι η κατάσταση του αρχικού αντικειμένου μπορεί να αντιστοιχιστεί σε μία συνεπή κατάσταση ενός άλλου αντικειμένου.

Για να γίνει η αντικατάσταση του αντικειμένου χρησιμοποιείται το σχεδιαστικό πρότυπο Bridge [7]. Σύμφωνα με αυτό το σχεδιαστικό πρότυπο, κάθε σύνολο εναλλακτικών κλάσεων συσχετίζεται με μία μοναδική κλάση (Handle class) η οποία έχει την ίδια διεπαφή με τις κλάσεις του συνόλου. Στην συνέχεια χρησιμοποιούνται 2 τεχνικές αντικατάστασης:

1. το προς αντικατάσταση αντικείμενο ολοκληρώνει την εκτέλεσή του και μετά αντικαταστείται (lazy replacement).

2. το προς αντικατάσταση αντικείμενο σταματά την εκτέλεσή του και συνεχίζει την εκτέλεσή του μετά την αντικατάσταση από το σημείο όπου διακόπηκε (eager replacement).

Το ενδιαφέρον σημείο του συστήματος CASA είναι ότι χειρίζεται την κατάσταση ενός αντικειμένου σε χρόνο εκτέλεσης επιτρέποντας στο αντικείμενο να συνεχίσει την εκτέλεσή του από το σημείο όπου διακόπηκε. Το μειονέκτημά του είναι ότι οι διεπαφές των αντικειμένων είναι προκαθορίζονται κατά τον σχεδιασμό του συστήματος.

2.3. Σύνθεση υπηρεσιών

2.3.1. Αυτοματοποιημένη Σύνθεση υπηρεσιών βασιζόμενη στην λειτουργική συμπεριφορά των υπηρεσιών

Μία άλλη προσέγγιση [3] στην διαχείριση αλλαγών σε συστήματα υπηρεσιών διαδικτύου βασίζεται στην σύνθεση υπηρεσιών (service composition). Πιο συγκεκριμένα, όταν ένας πελάτης δεν μπορεί να ικανοποιηθεί από κάποια υπηρεσία, τότε το σύστημα κατασκευάζει μια νέα σύνθετη υπηρεσία (composite service). Δηλαδή, μια ακολουθία από υπηρεσίες που όταν εκτελεστούν επιστρέφουν το ίδιο αποτέλεσμα με την αρχική υπηρεσία. Αυτή η κατασκευή γίνεται είτε αυτόματα από ένα εργαλείο είτε χειρονακτικά από τον χρήστη. Στην αυτοματοποιημένη κατασκευή τίθενται 2 ερωτήματα:

1. η ύπαρξη ή μη μίας τέτοιας σύνθεσης υπηρεσιών που να ικανοποιεί τις ανάγκες του πελάτη (composition existence).
2. ο τρόπος κατασκευής της σύνθετης υπηρεσίας.

Στην κατασκευή της σύνθετης υπηρεσίας θα πρέπει να ελέγχεται η ροή των δεδομένων μεταξύ των επιμέρους υπηρεσιών ώστε να εκτελείται σωστά η σύνθετη υπηρεσία. Οι επιμέρους υπηρεσίες δεν μοντελοποιούνται ως ένα σύνολο από παραμέτρους και επιστρεφόμενες τιμές αλλά ως ένα σύνολο βημάτων-ενεργειών. Αυτή η μοντελοποίηση δίνει στον χρήστη την δυνατότητα να προσδιορίζει ο ίδιος το επόμενο βήμα στην εκτέλεση σε μία υπηρεσία. Γι’ αυτό το λόγο, οι υπηρεσίες μοντελοποιούνται ως ένα ντετερμινιστικό σύνολο καταστάσεων (Deterministic Finite

State Machines - FSMs). Η μετάβαση από μια κατάσταση σε μια άλλη είναι μία ενέργεια της υπηρεσίας. Τελικά, το σύστημα με δεδομένη μία υποψήφια υπηρεσία WS και ένα σύνολο από διαθέσιμες υπηρεσίες, κατασκευάζει την ζητούμενη ακολουθία υπηρεσιών.

2.3.2. Σύνθεση υπηρεσιών βασισμένη σε Aspect-Oriented προγραμματισμό

Η σύνθεση υπηρεσιών ανήκει στον τομέα της ανάπτυξης μίας χωρογραφίας για ένα σύνολο υπηρεσιών διαδικτύου (Web Services choreography). Η προσέγγιση για την σύνθεση υπηρεσιών που θα αναλύσουμε [10], χρησιμοποιεί την γλώσσα BPEL4WS [8, 9], την κατεξοχήν γλώσσα δημιουργίας χωρογραφιών. Η BPEL4WS είναι workflow-based γλώσσα για υπηρεσίες διαδικτύου.

Οι συνθέσεις στην BPEL4WS δημιουργούνται χρησιμοποιώντας την WSDL διεπαφή των συμμετεχόντων υπηρεσιών διαδικτύου έτσι ώστε κάθε υπηρεσία να μην συσχετίζεται με ένα συγκεκριμένο στιγμιότυπο υπηρεσίας (service endpoint). Για να συσχετιστεί με κάποιο στιγμιότυπο γίνεται αναζητείται συντακτικά (βλ. ενότητα 1.1) σε αποθήκες υπηρεσιών διαδικτύου.

Σε αυτήν την προσέγγιση προτείνεται λύση για δυναμική διαχείριση αλλαγών στην σύνθετη υπηρεσία που έχει κατασκευαστεί. Πιο συγκεκριμένα, προσδιορίζεται ποιες λειτουργίες πρέπει να κληθούν από κάθε υπηρεσία και με ποια σειρά. Υπάρχει περίπτωση να χρειαστεί να προστεθούν ή να αντικατασταθούν κλήσεις λειτουργιών σε αυτήν την σύνθετη υπηρεσία. Αυτήν την δυναμική τροποποίηση δεν την υποστηρίζει η γλώσσα BPEL4WS. Η λύση βασίζεται στον aspect-oriented προγραμματισμό. Σε χρόνο, προσδιορίζονται εντός του BPEL σεναρίου, τα σημεία όπου πρέπει να προστεθούν ή να αντικατασταθούν κλήσεις λειτουργιών. Αυτά τα σημεία ονομάζονται σημεία συνένωσης (join points).

Αυτή η προσέγγιση έχει ένα βασική μειονέκτημα. Στην ουσία, σε περίπτωση κάποιας αλλαγής, κατασκευάζει εκ νέου μία καινούρια ακολουθία υπηρεσιών. Υλοποιεί προσθήκη λειτουργιών από νέες ή ήδη υπάρχουσες υπηρεσίες και αντικατάσταση λειτουργιών από υπηρεσίες με συντακτικά ταυτόσημη WSDL διεπαφή. Δεν κάνει αντικατάσταση μίας λειτουργίας από μία ισοδύναμη λειτουργία η οποία ανήκει σε κάποια άλλη σημασιολογικά ισοδύναμη υπηρεσία διαδικτύου.

Επομένως, τα δύο βασικά σημεία όπου υστερούν συνολικά οι παραπάνω προσεγγίσεις είναι ότι:

1. δεν αναζητούν σημασιολογικά ισοδύναμες υπηρεσίες αλλά χρησιμοποιούν μόνο διαφορετικά στιγμιότυπα υπηρεσιών με ίδια WSDL διεπαφή
2. και κατά την αντικατάσταση υπηρεσιών, χρησιμοποιούν λειτουργίες από συντακτικά ταυτόσημες υπηρεσίες. Στιγμιότυπα τέτοιων υπηρεσιών είτε είναι γνωστά εκ των προτέρων είτε αναζητούνται δυναμικά.

ΚΕΦΑΛΑΙΟ 3. ΣΗΜΑΣΙΟΛΟΓΙΚΗ ΑΝΑΠΡΟΣΑΡΜΟΓΗ ΥΠΗΡΕΣΙΩΝ

3.1. Σημασιολογική περιγραφή υπηρεσιών

3.1.1. Βασικές ιδέες

Το σύστημα που υλοποιήθηκε στην παρούσα εργασία ακολουθεί αρχιτεκτονική προσανατολισμένη στην προσφορά υπηρεσιών διαδικτύου. Υποθέτουμε ότι η λειτουργικότητα κάθε διαθέσιμης υπηρεσίας περιγράφεται από την διεπαφή της σε γλώσσα WSDL [13]. Η γλώσσα WSDL περιγράφει μία υπηρεσία αυστηρά ως ένα σύνολο από λειτουργίες χωρίς να δίνει μία πιο αφηρημένη περιγραφή της πληροφορίας που προσφέρει η υπηρεσία. Στην πράξη, είναι πιθανόν, δύο υπηρεσίες να παρέχουν την ίδια λειτουργικότητα μέσω διαφορετικών διεπαφών. Αυτό οδηγεί στην ανάγκη ύπαρξης μίας λιγότερο αυστηρής περιγραφής της λειτουργικότητας των υπηρεσιών σε σημασιολογικό επίπεδο που θα επιτρέπει την οργάνωση των υπηρεσιών που έχουν την ίδια λειτουργικότητα αλλά διαφορετική διεπαφή. Αυτή η οργάνωση των υπηρεσιών σε σημασιολογικό επίπεδο περιλαμβάνει μία ομαδοποίηση των προγραμματιστικών διεπαφών των υπηρεσιών σε προφίλ υπηρεσιών. Τα προφίλ των υπηρεσιών οργανώνονται περαιτέρω σε κατηγορίες.

Ορισμός: Ένα προφίλ είναι ένας τρόπος ομαδοποίησης και περιγραφής υπηρεσιών σε σημασιολογικό επίπεδο οι οποίες θα έχουν:

1. είτε συντακτικά ταυτόσημες διεπαφές
2. είτε μη συντακτικά ταυτόσημες διεπαφές αλλά θα προσφέρουν ίδια ή παρόμοια λειτουργικότητα.

Ορισμός: Μία κατηγορία είναι μία ομάδα από προφίλ. Οι υπηρεσίες που ομαδοποιούνται από αυτά τα προφίλ παρέχουν ίδιες ή παρόμοιες πληροφορίες στον πελάτη.

Η οργάνωση των υπηρεσιών σε προφίλ διευκολύνει τη διαδικασία αναπροσαρμογής των εφαρμογών που χρησιμοποιούν αυτές τις υπηρεσίες. Η γλώσσα που χρησιμοποιείται από την παρούσα εργασία για τις σημασιολογικές περιγραφές των υπηρεσιών είναι η OWL-S [11].
Για να περιγραφεί με πιο αφηρημένο τρόπο η κοινή λειτουργικότητα που προσφέρεται από υπηρεσίες μέσω διαφορετικών διεπαφών και να προκύψει τελικά μια σημασιολογική περιγραφή υπηρεσιών στη γλώσσα OWL-S, πρέπει να δοθούν απαντήσεις και στα τρία παρακάτω ερωτήματα:

1. Ποιες πληροφορίες παρέχουν οι υπηρεσίες στον πελάτη;
2. Πώς μπορεί να αντληθούν οι πληροφορίες από τις υπηρεσίες;
3. Πώς μπορεί να γίνει η επικοινωνία με τις υπηρεσίες;

Η απάντηση στο πρώτο ερώτημα οδηγεί στην κατασκευή του προφίλ (Profile) των υπηρεσιών. Ένα προφίλ υπηρεσιών περιλαμβάνει πληροφορίες σχετικά με:

1. τα δεδομένα εισόδου και τα δεδομένα εξόδου του,
2. την κατηγορία στην οποία ανήκει το προφίλ.

Η απάντηση στο δεύτερο ερώτημα αφορά τον τρόπο με τον οποίο μπορούν να αντληθούν οι πληροφορίες από τις υπηρεσίες. Αυτές πληροφορίες παρέχονται από λειτουργικές μονάδες των υπηρεσιών που ονομάζονται ατομικές διεργασίες. Μια ατομική διεργασία δέχεται έναν αριθμό δεδομένων εισόδου και επιστρέφει έναν αριθμό δεδομένων εξόδου τα οποία αποτελούν την ζητούμενη πληροφορία (σχήμα 3.3). Πρακτικά, ένα προφίλ υπηρεσιών υλοποιείται από το σύνολο των ατομικών

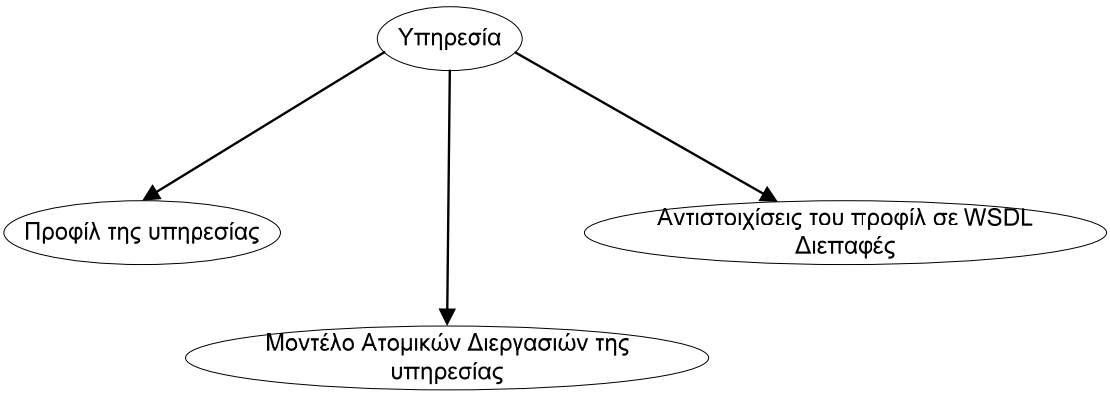
διεργασιών του που ονομάζεται μοντέλο ατομικών διεργασιών (Atomic Process Model).

Ορισμός: Οι ατομικές διεργασίες είναι ένα τρόπος περιγραφής της λειτουργικότητας των υπηρεσιών ενός προφίλ. Δηλαδή, μία ατομική διεργασία περιγράφει τα δεδομένα εισόδου που δέχεται μία υπηρεσία και τις πληροφορίες που επιστρέφει η υπηρεσία στον πελάτη.

Το τρίτο ερώτημα αφορά στην αντιστοίχηση (Grounding) του προφίλ υπηρεσιών στις διαφορετικές διεπαφές αυτών των υπηρεσιών. Μια τέτοια αντιστοίχηση απεικονίζει ατομικές διεργασίες σε λειτουργίες μίας διεπαφής. Λεπτομέρειες για τον τρόπο με τον οποίο γίνεται αυτή η αντιστοίχηση δίνονται στην ενότητα 3.1.2.

Συμπερασματικά, όπως φαίνεται και στο επόμενο σχήμα, μια σημασιολογική περιγραφή υπηρεσιών περιλαμβάνει:

1. ένα προφίλ,
2. ένα μοντέλο ατομικών διεργασιών,
3. ένα σύνολο αντιστοιχήσεων του προφίλ της σε διεπαφές υπηρεσιών.



Σχήμα 3.1 Η σημασιολογική περιγραφή μίας υπηρεσίας

Το παράδειγμα που ακολουθεί έχει σχέση με μία εφαρμογή ηλεκτρονικού εμπορίου. Έστω ότι ένας πελάτης ενδιαφέρεται για την αγορά ενός φορητού υπολογιστή και ενός βιβλίου. Για να προσδιοριστεί το είδος του φορητού υπολογιστή, είναι απαραίτητο να δοθούν κάποια χαρακτηριστικά του υπολογιστή. Τέτοια

χαρακτηριστικά θα μπορούσαν να είναι: η τιμή του, η επωνυμία του, η ημερομηνία παραγωγής του, η ταχύτητα του επεξεργαστή, η χωρητικότητα του σκληρού δίσκου, η χρονική διάρκεια της εγγύησής του κτλ. Παρόμοια χαρακτηριστικά μπορούν να δοθούν και για την αγορά του βιβλίου.

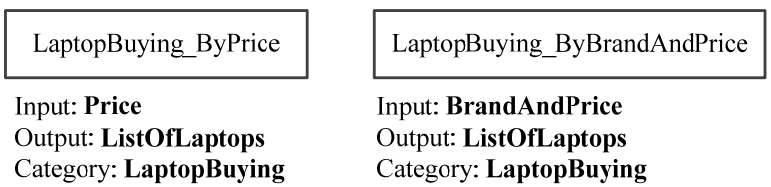
Έστω ότι υπάρχει ένας δικτυακός τόπος που δέχεται τα παραπάνω χαρακτηριστικά ως είσοδο και δίνει τη δυνατότητα της αναζήτησης των φορητών υπολογιστών που έχουν αυτά τα χαρακτηριστικά. Οι πληροφορίες για τους φορητούς υπολογιστές μπορεί να προσφέρονται – επιστρέφονται από υπηρεσίες διαδικτύου. Μία υπηρεσία, η οποία είναι διαθέσιμη στο διαδίκτυο για αυτόν τον σκοπό, μπορεί να προσφέρει πληροφορίες για φορητούς υπολογιστές αλλά να δέχεται ως είσοδο ένα υποσύνολο των παραπάνω χαρακτηριστικών. Επιπλέον, είναι πιθανόν δύο υπηρεσίες να δέχονται ως είσοδο όλα τα παραπάνω χαρακτηριστικά, να παρέχουν την ίδια λειτουργικότητα αλλά να μην έχουν την ίδια διεπαφή.

Επομένως, οι υπηρεσίες που παρέχουν την ίδια λειτουργικότητα θα πρέπει να ομαδοποιηθούν σημασιολογικά ώστε να έχουν το ίδιο προφίλ ανεξαρτήτως της διεπαφής τους. Όλα τα προφίλ των υπηρεσιών που προσφέρουν πληροφορίες για αγορά φορητών υπολογιστών σχηματίζουν μία κατηγορία, την κατηγορία *Αγορά Φορητών Υπολογιστών*. Αντίστοιχα, όλα τα προφίλ των υπηρεσιών που προσφέρουν πληροφορίες για αγορά βιβλίου σχηματίζουν την κατηγορία *Αγορά Βιβλίου*.

Έστω ότι δύο από τα προφίλ της κατηγορίας *Αγορά Φορητών Υπολογιστών (LaptopBuying)* είναι τα :

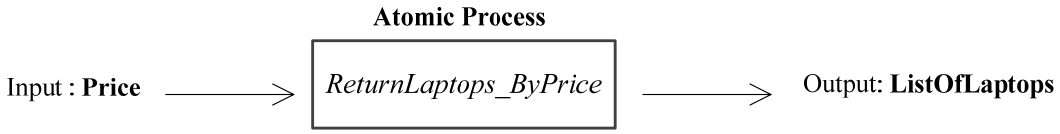
1. *LaptopBuying_ByPrice* και
2. *LaptopBuying_ByBrandAndPrice* (σχήμα 3.2).

Το πρώτο προφίλ έχει δεδομένο εισόδου την τιμή του φορητού υπολογιστή και δεδομένο εξόδου μία λίστα με στοιχεία φορητών υπολογιστών (*ListOfLaptops*). Το δεύτερο προφίλ έχει το ίδιο δεδομένο εξόδου αλλά έχει διαφορετικό δεδομένο εισόδου, την τιμή και την επωνυμία (*BrandAndPrice*) του φορητού υπολογιστή.



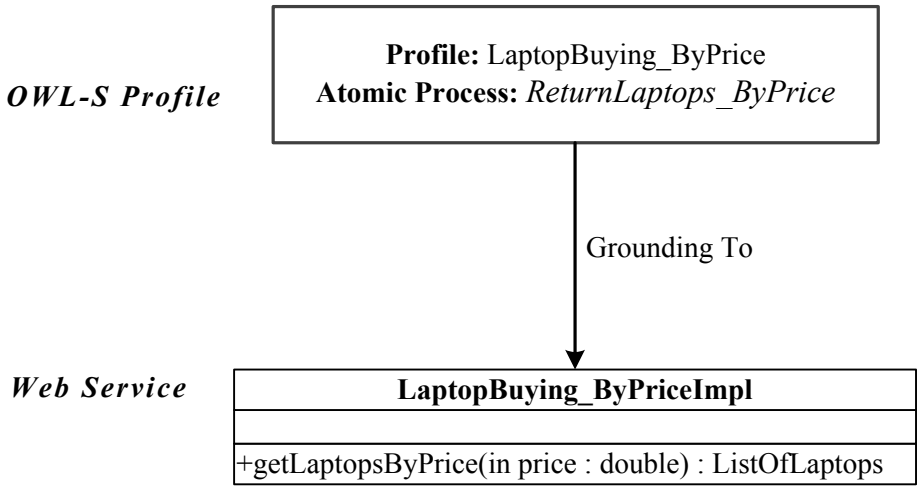
Σχήμα 3.2 Παραδείγματα προφίλ υπηρεσιών

Για να αντληθούν πληροφορίες από το προφίλ *LaptopBuying_ByPrice*, θα πρέπει να είναι γνωστό το όνομα μίας ατομικής διεργασίας του προφίλ. Για παράδειγμα, το προφίλ θα μπορούσε να έχει την ατομική διεργασία *ReturnLaptops_ByPrice* η οποία δέχεται ως είσοδο την τιμή ενός φορητού υπολογιστή και δίνει ως έξοδο μία λίστα με στοιχεία φορητών υπολογιστών.



Σχήμα 3.3 Η ατομική διεργασία *ReturnLaptops_ByPrice*

Η ατομική διεργασία *ReturnLaptops_ByPrice* δεν είναι πρόγραμμα που μπορεί να εκτελεστεί αλλά είναι περιγραφή μίας λειτουργίας την οποία προσφέρει το προφίλ της υπηρεσίας. Πρακτικά, πρέπει να βρεθεί μία διεπαφή που να αντιστοιχεί σημασιολογικά σε αυτό το προφίλ και να έχει μία μέθοδο που να επιστρέφει την ίδια πληροφορία που επιστρέφει η παραπάνω ατομική διεργασία. Αυτή η αντιστοίχηση απεικονίζεται στο επόμενο σχήμα:



Σχήμα 3.4 Η αντιστοίχηση ενός προφίλ σε μία διεπαφή

3.1.2. Η σημασιολογική περιγραφή υπηρεσιών στην γλώσσα OWL-S

Όπως αναφέρθηκε στην ενότητα 3.1.1, το σύστημα που υλοποιήθηκε στην παρούσα εργασία ακολουθεί αρχιτεκτονική προσανατολισμένη στην προσφορά υπηρεσιών διαδικτύου. Η κατάσταση του συστήματος προσδιορίζεται από το σύνολο των διαθέσιμων προς χρήση υπηρεσιών. Το σύστημα για να διαχειριστεί την κατάστασή του, αποθηκεύει πληροφορίες για τις σημασιολογικές περιγραφές των υπηρεσιών και τις ομαδοποιεί σε κατηγορίες. Πιο συγκεκριμένα, αποθηκεύει πληροφορίες σχετικά με:

1. Τις κατηγορίες.

   Αποθηκεύει το όνομα κάθε κατηγορίας αλλά και για κάθε κατηγορία τα ονόματα των προφίλ που περιέχει.

2. Τα προφίλ των υπηρεσιών.

   Αποθηκεύει το όνομα του κάθε προφίλ αλλά και τις πληροφορίες που αναφέρθηκαν στην ενότητα 3.1.1.

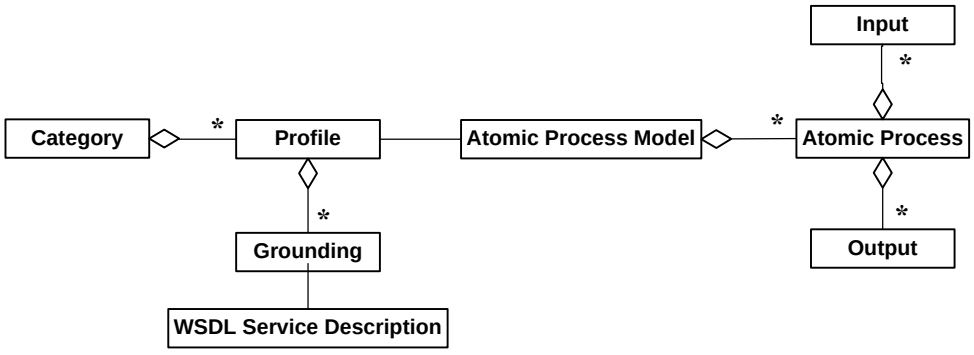
3. Το μοντέλο των ατομικών διεργασιών για κάθε προφίλ.

   Κάθε προφίλ περιέχει μία ή περισσότερες ατομικές διεργασίες. Αποθηκεύεται το όνομα κάθε ατομικής διεργασίας αλλά και τα δεδομένα εισόδου και εξόδου της.

4. Τις αντιστοιχίσεις των προφίλ σε διεπαφές.

   Κάθε προφίλ μπορεί να αντιστοιχίζεται σε μία ή περισσότερες διεπαφές υπηρεσιών.

Συμπερασματικά, οι πληροφορίες που αποθηκεύονται από το σύστημα έχουν την ακόλουθη δομή:



Σχήμα 3.5 Η δομή μίας κατηγορίας σημασιολογικών περιγραφών

Για να περιγραφεί μία κατηγορία από προφίλ στην γλώσσα OWL-S, κατασκευάζονται τέσσερα διαφορετικά αρχεία (.owl), ένα αρχείο για κάθε μία από τις τέσσερις πληροφορίες που αναφέρθηκαν παραπάνω. Πιο συγκεκριμένα, για μία κατηγορία κατασκευάζονται τα αρχεία:

1. Category.owl

Αυτό το αρχείο περιέχει γενικές πληροφορίες για μία κατηγορία υπηρεσιών. Δηλαδή, περιέχει το όνομα του κάθε προφίλ της κατηγορίας, τα ονόματα των ατομικών διεργασιών που περιλαμβάνει κάθε προφίλ και τα ονόματα των διεπαφών στις οποίες αντιστοιχεί κάθε προφίλ. Το επόμενο σχήμα δείχνει τις ετικέτες που χρησιμοποιούνται από το αρχείο για να αποθηκευτούν οι πληροφορίες για την κατηγορία *LaptopBuying* (βλ. παράδειγμα ενότητας 3.1.1), δηλαδή το όνομα του προφίλ (LaptopBuying_ByPrice), το όνομα της ατομικής διεργασίας του (ReturnLaptops_ByPrice) και το όνομα της αντιστοίχησής του σε μία διεπαφή (LaptopBuyingGrounding).

OWL-S Category

```
<service:Service rdf:ID="PriceOfLaptop">
    <service:presents rdf:resource="&LaptopBuying_profile;#LaptopBuying_ByPrice"/>
    <service:describedBy rdf:resource="&LaptopBuying_process;#ReturnLaptops_ByPrice"/>
    <service:supports rdf:resource="&LaptopBuying _grounding;#LaptopBuyingGrounding"/>
</service:Service>
```

Σχήμα 3.6 Τμήμα του αρχείου *LaptopBuyingCategory.owl*

2. Profile.owl

Αυτό το αρχείο περιέχει πληροφορίες για κάθε προφίλ της κατηγορίας. Δηλαδή, περιέχει το όνομα του κάθε προφίλ, τα ονόματα των ατομικών διεργασιών του και τα ονόματα των δεδομένων εισόδου και εξόδου του. Το επόμενο σχήμα δείχνει τις ετικέτες που χρησιμοποιούνται από το αρχείο για να αποθηκευτούν οι πληροφορίες για το προφίλ *LaptopBuying_ByPrice* (βλ. παράδειγμα ενότητας 3.1.1), δηλαδή το όνομα του προφίλ (LaptopBuying_ByPrice), το όνομα του δεδομένου εισόδου του (Price) και το όνομα του δεδομένου εξόδου του (ListOfLaptops).

OWL-S Profile

```
<profileHierarchy:Buying rdf:ID="BuyingHierarchy">
   <service:presentedBy rdf:resource="&LaptopBuyingService;#PriceOfLaptop"/>
   <profile:has_process rdf:resource="&LaptopBuyingProcess;#ReturnLaptops_ByPrice"/>
   <profile:hasInput rdf:resource="&LaptopBuyingProcess;#Price"/>
   <profile:hasOutput rdf:resource="&LaptopBuyingProcess;#ListOfLaptops"/>
</profileHierarchy:Buying>
```

Σχήμα 3.7 Τμήμα του αρχείου *LaptopBuyingProfile.owl*

3. AtomicProcess.owl

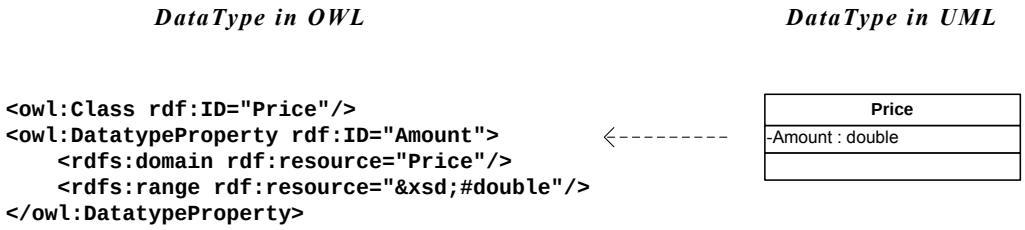
Αυτό το αρχείο περιέχει πληροφορίες για τις ατομικές διεργασίες που έχει κάθε προφίλ μίας κατηγορίας. Ορίζει το όνομα και τα δεδομένα εισόδου/εξόδου κάθε ατομικής διεργασίας του κάθε προφίλ. Στο προηγούμενο αρχείο (σχήμα 3.7) ορίστηκαν πάλι τα ονόματα των δεδομένων εισόδου/εξόδου αλλά δεν ορίστηκε σε ποια ατομική διεργασία ανήκει κάθε δεδομένο. Επιπλέον, σε αυτό το αρχείο ορίζονται αναλυτικά τα δεδομένα εισόδου/εξόδου ως κλάσεις στην γλώσσα OWL. Κάθε αφηρημένη οντότητα ή τύπος δεδομένων του πραγματικού κόσμου αναπαρίσταται ως OWL κλάση. Μία κλάση μπορεί να αποτελείται από ένα σύνολο ιδιοτήτων (owl:DatatypeProperty). Κάθε ιδιότητα χαρακτηρίζεται από ένα όνομα από τον τύπο της όπως αυτός ορίζεται στην γλώσσα OWL (σχήμα 3.8).

OWL Basic DataTypes

xsd:string	**xsd:normalizedString**	**xsd:boolean**
xsd:decimal	**xsd:float**	**xsd:double**
xsd:integer	**xsd:nonNegativeInteger**	**xsd:positiveInteger**
xsd:nonPositiveInteger	**xsd:negativeInteger**	**xsd:gMonth**
xsd:long	**xsd:int**	**xsd:short**
xsd:unsignedLong	**xsd:unsignedInt**	**xsd:unsignedShort**
xsd:hexBinary	**xsd:base64Binary**	**xsd:gYearMonth**
xsd:dateTime	**xsd:time**	**xsd:date**
xsd:gYear	**xsd:gMonthDay**	**xsd:gDay**
xsd:anyURI	**xsd:token**	**xsd:language**
xsd:NMTOKEN	**xsd:Name**	**xsd:NCName**
xsd:byte	**xsd:unsignedByte**	

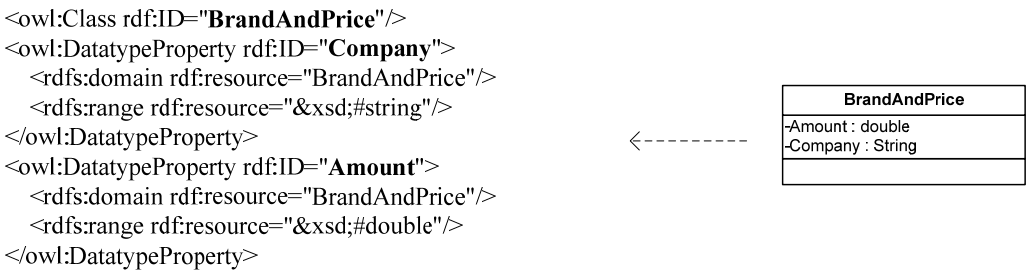
Σχήμα 3.8 Οι βασικοί τύποι δεδομένων στην γλώσσα OWL

Για παράδειγμα, για να δηλώσουμε στην γλώσσα OWL το δεδομένο εισόδου *Price* με μία ιδιότητα η οποία αντιστοιχεί σε τύπο δεδομένων πραγματικού αριθμού διπλής ακρίβειας (double), τότε ορίζουμε την κλάση *Price* ως εξής:



Σχήμα 3.9 Ορισμός της κλάσης *Price* στην OWL και την UML

Για να δηλώσουμε το δεδομένο εισόδου *BrandAndPrice* με δύο ιδιότητες οι οποίες αντιστοιχούν σε έναν τύπο δεδομένων ενός πραγματικού αριθμού διπλής ακρίβειας (double) και σε έναν τύπο δεδομένων ενός αλφαριθμητικού (string), τότε ορίζουμε την κλάση *BrandAndPrice* ως εξής:



Σχήμα 3.10 Ορισμός της κλάσης *BrandAndPrice* στην OWL και την UML

Όπως αναφέρθηκε στο παράδειγμα της ενότητας 3.1.1, το προφίλ *LaptopBuying_ByPrice* παρέχει την ατομική διεργασία *ReturnLaptops_ByPrice*. Το επόμενο σχήμα δείχνει τις ετικέτες που χρησιμοποιούνται στο αρχείο για να αποθηκευτούν οι πληροφορίες για αυτήν την ατομική διεργασία.

OWL-S Process

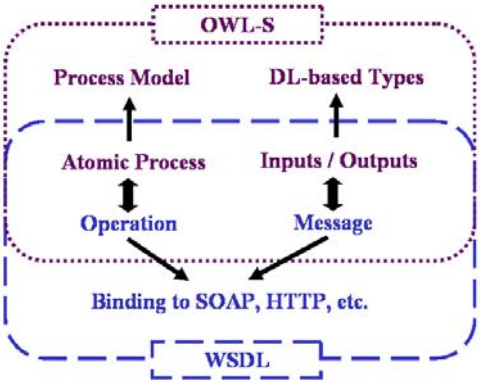
```
<owl:Class rdf:ID="Price"/>
<owl:DatatypeProperty rdf:ID="Amount">
    <rdfs:domain rdf:resource="Price"/>
    <rdfs:range rdf:resource="&xsd;#double"/>
</owl:DatatypeProperty>
                                        ...
<process:AtomicProcess rdf:ID="ReturnLaptops_ByPrice">
   <process:hasInput>
       <process:Input rdf:ID="OnlyPrice"/>
           <process:parameterType rdf:datatype="&xsd;#anyURI">&xsd;#Price</process:parameterType>
       </process:Input>
   </process:hasInput>
   <process:hasOutput>
       <process:Output rdf:ID="List"/>
           <process:parameterType rdf:datatype="&xsd;#anyURI">&xsd;#ListOfLaptops</process:parameterType>
   </process:Output>
  </process:hasOutput>
</process:AtomicProcess>
```

Σχήμα 3.11 Τμήμα του αρχείου *LaptopBuyingProcess.owl*

4. Grounding.owl

Αυτό το αρχείο περιέχει πληροφορίες για την αντιστοίχηση του κάθε προφίλ σε διεπαφές. Αυτή η αντιστοίχηση ακολουθεί τους εξής κανόνες (σχήμα 3.12):

1. κάθε ατομική διεργασία ενός προφίλ αντιστοιχίζεται σε μία λειτουργία της διεπαφής (WSDL operation),
2. τα δεδομένα εισόδου/εξόδου μίας ατομικής διεργασίας αντιστοιχίζονται ένα προς ένα στα τμήματα ενός μηνύματος (message parts) της αντίστοιχης λειτουργίας της διεπαφής,
3. ο τύπος των δεδομένων εισόδου ταυτίζεται με τον τύπο των τμημάτων του αντίστοιχου μηνύματος.



Σχήμα 3.12 Η αντιστοίχηση της OWL-S στην WSDL

Όπως αναφέρθηκε στο παράδειγμα της ενότητας 3.1.1, η ατομική διεργασία *ReturnLaptops_ByPrice* αντιστοιχίζεται στην λειτουργία *getLaptopByPrice*. Στο επόμενο σχήμα, φαίνεται με ποιον τρόπο μπορεί να αντιστοιχηθεί το δεδομένο εισόδου *Price* της ατομικής διεργασίας *ReturnLaptops_ByPrice* στο μήνυμα *wsdl_Price* της λειτουργίας *getLaptopByPrice*.

DataType in OWL ***Message in WSDL***

```
<owl:Class rdf:ID="Price"/>
<owl:DatatypeProperty rdf:ID="Amount">    <---------   <message name="wsdl_Price">
    <rdfs:domain rdf:resource="Price"/>                      <part name="wsdl_Amount">
    <rdfs:range rdf:resource="&xsd;#double"/>            </message>
</owl:DatatypeProperty>
```

Σχήμα 3.13 Η αντιστοίχηση του δεδομένου εισόδου *Price* στο μήνυμα *wsdl_Price*

Επιπλέον, το επόμενο σχήμα δείχνει τις ετικέτες που χρησιμοποιούνται από το αρχείο για την συνολική αντιστοίχηση της ατομικής διεργασίας *ReturnLaptops_ByPrice* στην λειτουργία *getLaptopByPrice* μίας διεπαφής.

OWL-S Grounding

```
<grounding:WsdlAtomicProcessGrounding rdf:ID="LaptopBuyingByPriceGrounding">
  <grounding:owlsProcess rdf:resource="&LaptopBuying_process;#ReturnLaptops_ByPrice"/>
  <grounding:wsdlOperation>
    <grounding:WsdlOperationRef>
      <grounding:portType rdf:datatype="&xsd;#anyURI">
       &LaptopBuying_wsdl_grounding;#LaptopBuying_PortType
      </grounding:portType>
      <grounding:operation rdf:datatype="&xsd;#anyURI">
       &LaptopBuying_wsdl_grounding;#getLaptopByPrice
      </grounding:operation>
      </grounding:WsdlOperationRef>
  </grounding:wsdlOperation>

  <grounding:wsdlInputMessage rdf:datatype="&xsd;#anyURI">
      &LaptopBuying_By_Price_wsdl_grounding;#wsdl_Price
  </grounding:wsdlInputMessage>
  <grounding:wsdlInput>
    <grounding:WsdlInputMessageMap>
      <grounding:owlsParameter rdf:resource="&LaptopBuying_process;#Amount
      <grounding:wsdlMessagePart rdf:datatype="&xsd;#anyURI">
       &LaptopBuying_wsdl_grounding;#wsdl_Amount
      </grounding:wsdlMessagePart>
    </grounding:WsdlInputMessageMap>
  </grounding:wsdlInput>
</grounding:WsdlAtomicProcessGrounding>
```

OWL Atomic Process

WSDL Operation

OWL DataTypeProperty

WSDL Messge Part

Σχήμα 3.14 Τμήμα του αρχείου *LaptopBuyingGrounding.owl*

3.2. Οργάνωση και αναζήτηση υπηρεσιών για την αντικατάσταση μίας μη διαθέσιμης υπηρεσίας

Γενικά, η αναζήτηση υπηρεσιών που μπορούν να αντικαταστήσουν μια μη διαθέσιμη υπηρεσία στοχεύει στην εύρεση υπηρεσιών που έχουν το ίδιο προφίλ με τη μη διαθέσιμη υπηρεσία ή υπηρεσιών που έχουν παρόμοια προφίλ τα οποία ανήκουν στην ίδια κατηγορία με αυτή της μη διαθέσιμης υπηρεσίας.

Ως συνέχεια του παραδείγματος της ενότητας 3.1.1, το προφίλ *LaptopBuying_ByPrice* έχει ως δεδομένο εισόδου μόνο την τιμή του φορητού υπολογιστή ενώ το προφίλ *LaptopBuying_ByBrandAndPrice* έχει ως δεδομένο εισόδου την τιμή και την επωνυμία. Δηλαδή, το προφίλ *LaptopBuying_ByPrice* ομαδοποιεί υπηρεσίες που δέχονται ως είσοδο λιγότερη πληροφορία από τις υπηρεσίες που ομαδοποιούνται από το προφίλ *LaptopBuying_ByBrandAndPrice* ενώ το αποτέλεσμα που παρέχουν οι υπηρεσίες που ομαδοποιούνται και από τα δύο προφίλ είναι ίδιου τύπου. Επομένως, οι υπηρεσίες που ομαδοποιούνται από το πρώτο προφίλ μπορούν να αντικαταστήσουν υπηρεσίες που ομαδοποιούνται από το δεύτερο προφίλ.

Η σχέση αντικατάστασης υπηρεσιών που ομαδοποιούνται από διαφορετικά προφίλ είναι συνάρτηση των δεδομένων εισόδου και εξόδου τους. Στην γενική περίπτωση, για να είναι δυνατόν υπηρεσίες ενός προφίλ Α να αντικαταστήσουν υπηρεσίες ενός προφίλ Β, θα πρέπει το Β να είναι είτε σημασιολογική επέκταση είτε σημασιολογική εξειδίκευση του Α. Στην επόμενη ενότητα ορίζονται πιο αυστηρά οι έννοιες αυτές.

3.2.1. Σημασιολογική επέκταση ή εξειδίκευση προφίλ υπηρεσιών

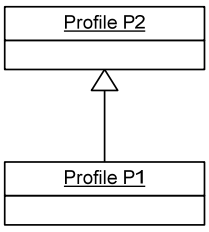
Ένα προφίλ υπηρεσίας περιλαμβάνει ένα σύνολο ατομικών διεργασιών (σχήμα 3.5). Ένα προφίλ P με N ατομικές διεργασίες μπορεί να συμβολιστεί ως εξής: $P = \{ap_{1,} ap_2, ... ap_N\}$. Κάθε ατομική διεργασία δέχεται έναν πλήθος δεδομένων εισόδου και δίνει ένα πλήθος δεδομένων εξόδου. Μία ατομική διεργασία ap με V δεδομένα εισόδου και W δεδομένα εξόδου μπορεί να συμβολιστεί ως εξής: $ap = \{i_{1,} i_2, ... i_V, o_1, o_2, ... o_W\}$.

Ορισμός 1: Ένα προφίλ $P_1 = \{ap_{11,} ap_{12}, ..., ap_{1x}, ... ap_{1N}\}$ είναι **σημασιολογική επέκταση ή εξειδίκευση** ενός προφίλ $P_2 = \{ap_{21,} ap_{22}, ..., ap_{2y}, ... ap_{2M}\}$ όταν για κάθε ατομική διεργασία ap_{1x} υπάρχει ατομική διεργασία ap_{2y} τέτοια ώστε:

1. για κάθε δεδομένο εισόδου της ατομικής διεργασίας ap_{2y} να υπάρχει κάποιο δεδομένο εισόδου της ατομικής διεργασίας ap_{1x} που να είναι επέκτασή του ή εξειδίκευσή του (βλ. Ορισμούς 2a, 2b) και
2. για κάθε κάθε δεδομένο εξόδου της ατομικής διεργασίας ap_{1x} να υπάρχει κάποιο δεδομένο εξόδου της ατομικής διεργασίας ap_{2y} που να είναι επέκτασή του ή εξειδίκευσή του.

Εάν δηλώσουμε τα δύο προφίλ P_1, P_2 ως OWL κλάσεις, τότε η σχέση της σημασιολογικής επέκτασης ή εξειδίκευσης που δόθηκε στον ορισμό 1, δηλώνει ότι η κλάση P_1 είναι υποκλάση της P_2. Αυτή η σχέση στην OWL γράφεται ως εξής:

```
<owl:Class rdf:ID="P1">
   <rdfs:subClassOf rdf:resource="#P2"/>
</owl:Class>
```



Σχήμα 3.15 Το προφίλ P1 είναι σημασιολογική επέκταση ή εξειδίκευση του P2

Ορισμός 2a: Ένα δεδομένο εισόδου/εξόδου T_1 ονομάζεται **εξειδίκευση** ενός δεδομένου εισόδου/εξόδου T_2 όταν οι OWL κλάσεις για το T_1 και το T_2 έχουν τον ίδιο αριθμό ιδιοτήτων (owl:DatatypeProperty) και μία ή περισσότερες από τις ιδιότητες του T_2 έχει ευρύτερο πεδίο τιμών από την αντίστοιχη του T_1.

Παράδειγμα για τον ορισμό 2a:

Έστω ότι υπάρχουν τα δεδομένα εισόδου/εξόδου *Person, AdultPerson* που έχουν μόνο μία ιδιότητα η οποία αντιστοιχεί στον βασικό τύπο *xsd:integer*. Εάν υποθέσουμε ότι η ιδιότητα του *AdultPerson* παίρνει τιμές από το διάστημα (0, 18), τότε είναι εξειδίκευση του δεδομένου *Person*. Η σχέση της εξειδίκευσης δηλώνει ότι η κλάση *AdultPerson* είναι υποκλάση της *Person*. Αυτή η σχέση στην OWL γράφεται ως εξής:

DataType Person

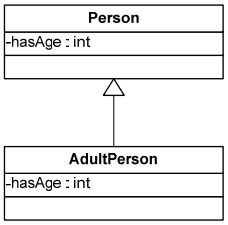
```
<owl:Class rdf:ID="Person"/>
<owl:DatatypeProperty rdf:ID="hasAge">
   <rdfs:domain rdf:resource="Person"/>
   <rdfs:range rdf:resource="&xsd;#int"/>
</owl:DatatypeProperty>
```

DataType AdultPerson

```
<owl:Class rdf:ID="AdultPerson">
   <rdfs:subClassOf rdf:resource="#Person"/>
   <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
           <owl:DatatypeProperty rdf:ID="hasAge"/>
        </owl:onProperty>
        <owl:allValuesFrom>
          <rdfs:Datatype>
            <xsp:base rdf:resource="&xsd;int"/>
            <xsp:minInclusive rdf:datatype="&xsd;int">0</xsp:minInclusive>
            <xsp:maxInclusive rdf:datatype="&xsd;int">18</xsp:maxInclusive>
          </rdfs:Datatype>
        </owl:allValuesFrom>
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

Σημασιολογική εξειδίκευση

```
<owl:Class rdf:ID="AdultPerson">
   <rdfs:subClassOf rdf:resource="#Person"/>
</owl:Class>
```



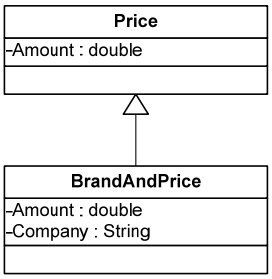
Σχήμα 3.16 Η σχέση της εξειδίκευσης στην OWL

Ορισμός 2b: Ένα δεδομένο εισόδου/εξόδου T_2 ονομάζεται **επέκταση** ενός δεδομένου εισόδου/εξόδου T_1 όταν οι ιδιότητες (owl:DatatypeProperty) της OWL κλάσης για το T_1 είναι υποσύνολο των ιδιοτήτων της OWL κλάσης για το T_2.

Παράδειγμα για τον ορισμό 2b:

Στην ενότητα 3.1.2 ορίστηκαν τα δεδομένα εισόδου/εξόδου *Price, BrandAndPrice* (σχήμα 3.9, 3.10 αντίστοιχα). Οι ιδιότητες της κλάσης *Price* είναι υποσύνολο των ιδιοτήτων της *BrandAndPrice* και επομένως η *BrandAndPrice* είναι επέκταση της *Price*. Η σχέση της επέκτασης δηλώνει ότι η κλάση *BrandAndPrice* είναι υποκλάση της *Price*. Αυτή η σχέση στην OWL γράφεται ως εξής:

```
<owl:Class rdf:ID="BrandAndPrice">
   <rdfs:subClassOf rdf:resource="#Price"/>
</owl:Class>
```

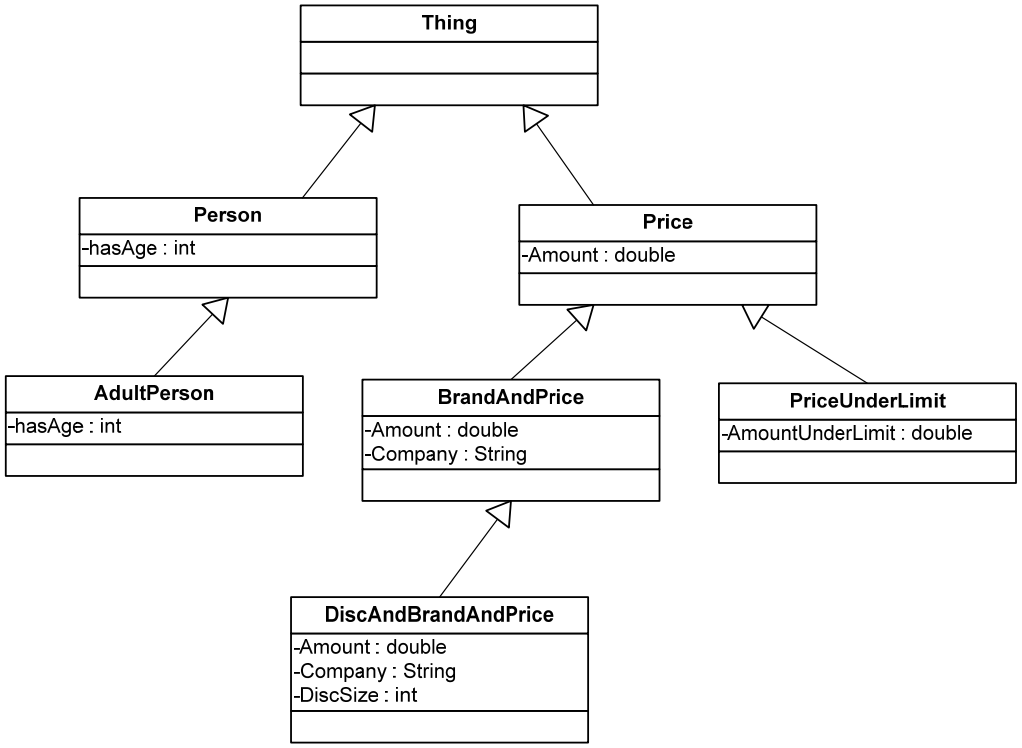


Σχήμα 3.17 Η σχέση της επέκτασης στην OWL

3.2.2. Οργάνωση των προφίλ υπηρεσιών

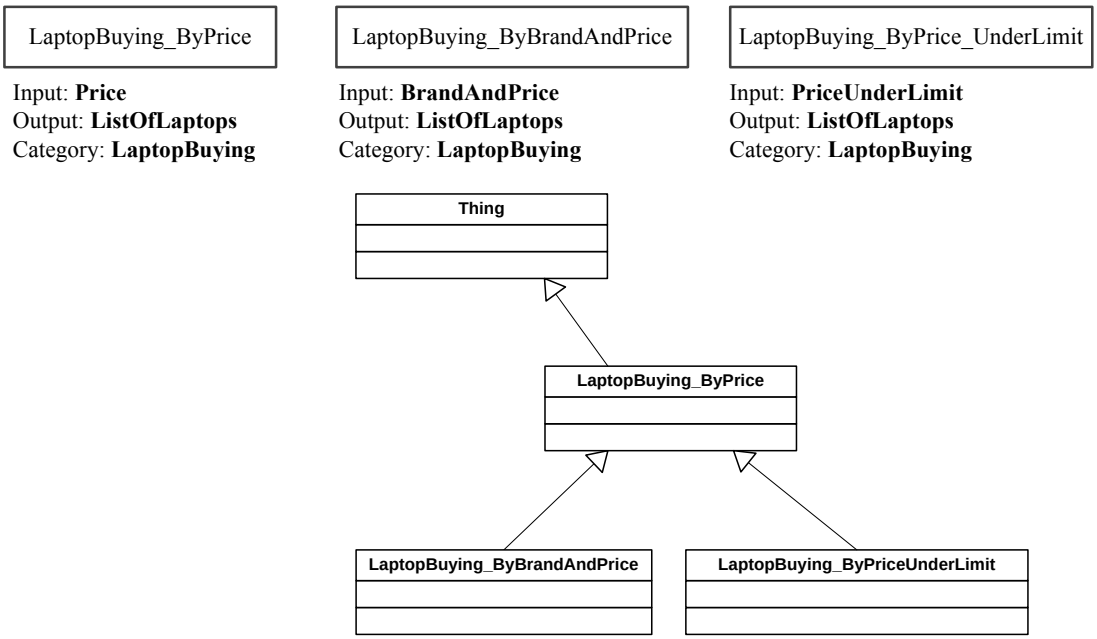
Στα πλαίσια του συστήματος που αναπτύχθηκε, λαμβάνουμε υπόψη σχέσεις σημασιολογικής επέκτασης ή εξειδίκευσης των δεδομένων εισόδου/εξόδου διαφορετικών προφίλ με σκοπό την δημιουργία αντίστοιχων συσχετίσεων μεταξύ των προφίλ αυτών. Πιο συγκεκριμένα, δεδομένης μιας δενδρικής δομής που συσχετίζει δεδομένα εισόδου/εξόδου διαθέσιμων υπηρεσιών προκύπτει μια αντίστοιχη δεντρική δομή για τα προφίλ αυτών των υπηρεσιών.

Για παράδειγμα, έστω τα ακόλουθα δεδομένα εισόδου/εξόδου: *Age*, *AdultAge*, *Price*, *BrandAndPrice*, *PriceUnderLimit*, *DiscAndBrandAndPrice*. Λόγω της σημασιολογικής σχέσης που υπάρχει μεταξύ τους (βλ. Ορισμούς 2a, 2b), σχηματίζεται συνολικά η δενδρική δομή του ακόλουθου σχήματος. Η ρίζα αυτής της δενδρικής δομής είναι η OWL κλάση *Thing*. Στην γλώσσα OWL κάθε κλάση θεωρείται υποκλάση της *OWL:Thing*.



Σχήμα 3.18 Στιγμιότυπο δενδρικής ιεράρχησης των δεδομένων εισόδου

Με δεδομένη την παραπάνω δενδρική δομή των δεδομένων εισόδου, προκύπτει μία ανάλογη δομή και για τα προφίλ των υπηρεσιών που βασίζονται σε αυτά τα δεδομένα. Για τα προφίλ του σχήματος 3.19, λαμβάνοντας υπόψη τον ορισμό 1, την σημασιολογική σχέση (επέκταση ή εξειδίκευση) των δεδομένων εισόδου τους και την ταύτιση των δεδομένων εξόδων τους, προκύπτει ότι το προφίλ *LaptopBuying_ByBrandAndPrice* είναι σημασιολογική επέκταση του *LaptopBuying_ByPrice* ενώ το προφίλ *LaptopBuying_ByPriceUnderLimit* είναι σημασιολογική εξειδίκευση του *LaptopBuying_ByPrice*.



Σχήμα 3.19 Δενδρική ιεράρχηση των προφίλ

3.2.3. Αναζήτηση υπηρεσιών για την αντικατάσταση μιας μη διαθέσιμης υπηρεσίας

Με βάση τη δενδρική ιεράρχηση των προφίλ διαθέσιμων υπηρεσιών, η διαδικασία αναζήτησης υπηρεσιών που μπορούν να αντικαταστήσουν μια μη διαθέσιμη υπηρεσία απλοποιείται σε μια διαδικασία αναζήτησης στη δενδρική ιεράρχηση των προφίλ. Πιο συγκεκριμένα, για δύο προφίλ P1, P2 ισχύει το ακόλουθο κριτήριο αντικατάστασης:

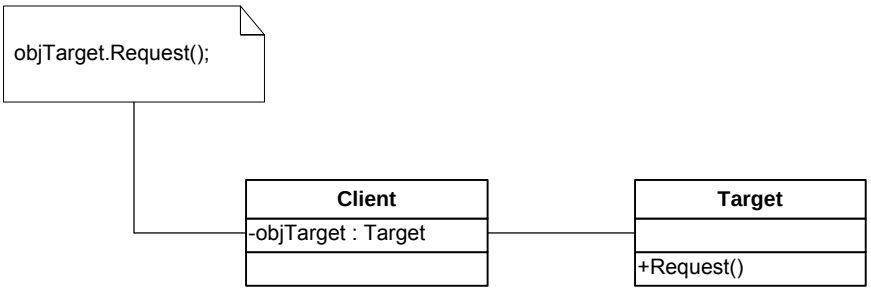
Κριτήριο αντικατάστασης υπηρεσιών: Εάν τα προφίλ P_1, P_2 ανήκουν στο ίδιο μονοπάτι της δενδρικής δομής των προφίλ και το P_1 βρίσκεται σε μικρότερο βάθος από το P_2, τότε οι υπηρεσίες που ομαδοποιούνται από το P_1 είναι σε θέση να αντικαταστήσουν υπηρεσίες που ομαδοποιούνται από το P_2.

3.3. Αντικατάσταση μη διαθέσιμης υπηρεσίας με τρόπο κλειστό ως προς τον κώδικα των εφαρμογών που τη χρησιμοποιούν

Στην περίπτωση που μία υπηρεσία *Target* χρησιμοποιείται στον κώδικα ενός πελάτη (σχήμα 3.20) και χρειαστεί να αντικατασταθεί από κάποια άλλη υπηρεσία, υπάρχουν οι ακόλουθες δύο εναλλακτικές προσεγγίσεις:

1. είτε να βρεθεί μία υπηρεσία που να έχει την ίδια διεπαφή με την αρχική,

2. είτε να βρεθεί μία υπηρεσία με διαφορετική διεπαφή η οποία όμως σημασιολογικά θα παρέχει την ίδια λειτουργικότητα με την αρχική υπηρεσία.



Σχήμα 3.20 Χρήση της υπηρεσίας Target από τον κώδικα ενός πελάτη

Για να βρεθεί η ζητούμενη για αντικατάσταση υπηρεσία στην πρώτη προσέγγιση, αρκεί να γίνει, στις αποθήκες του συστήματος, αναζήτηση υπηρεσιών οι οποίες ανήκουν στο ίδιο προφίλ και έχουν συντακτικά ταυτόσημες διεπαφές. Σε αυτήν την περίπτωση η μοναδική παρέμβαση που θα γίνει στον κώδικα του πελάτη είναι η αλλαγή του URI της υπηρεσίας.

Στην περίπτωση που δεν βρεθεί κάποια υπηρεσία που να έχει συντακτικά ταυτόσημη διεπαφή με την αρχική, ακολουθείται η δεύτερη προσέγγιση. Για να βρεθεί η ζητούμενη για αντικατάσταση υπηρεσία στην δεύτερη προσέγγιση, αρκεί να γίνει, στις αποθήκες του συστήματος, σημασιολογική αναζήτηση υπηρεσιών οι οποίες είτε θα έχουν κοινό προφίλ με την αρχική υπηρεσία είτε το προφίλ τους θα είναι σε θέση να αντικαταστήσει το προφίλ της αρχικής υπηρεσίας (βλ. Ενότητα 3.2). Λεπτομέρειες για τον αλγόριθμο της συντακτικής και της σημασιολογικής αναζήτησης δίνονται στην ενότητα 4.2.2.2.

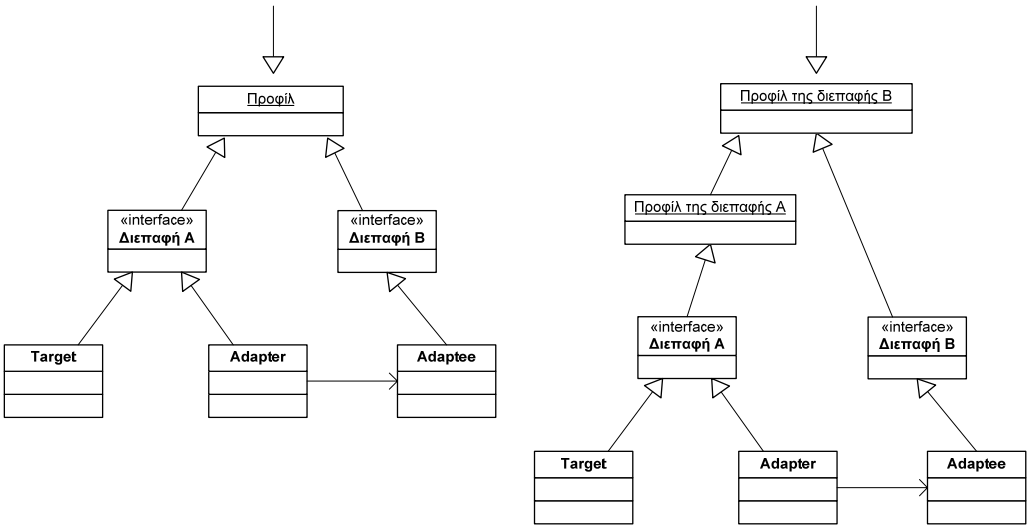
Εν γένει, στην δεύτερη προσέγγιση, επειδή η διεπαφή της υπηρεσίας θα είναι διαφορετική, στο κώδικα του πελάτη απαιτούνται πέραν της αλλαγής του URI της μη διαθέσιμης υπηρεσίας ριζικές τροποποιήσεις σε διάφορα σημεία του κώδικα. Το ζητούμενο είναι να βρεθεί ένας τρόπος εφαρμογής της δεύτερης προσέγγισης ώστε η αντικατάσταση της υπηρεσίας να αλλάξει μόνο το URI της στον κώδικα του πελάτη όπως ακριβώς συμβαίνει και στην πρώτη προσέγγιση.

Για να δοθεί απάντηση στον τρόπο εφαρμογής της δεύτερης προσέγγισης χρησιμοποιείται το Προσαρμοστικό Σχεδιαστικό Πρότυπο (Adapter Design Pattern [7]).

Έστω ότι η αρχική υπηρεσία είναι η *Target*, η οποία υλοποιεί μία διεπαφή Α, και η υπηρεσία που θα την αντικαταστήσει είναι η *Adaptee*, η οποία υλοποιεί μία διεπαφή Β. Η διεπαφή Α και η διεπαφή Β είναι πιθανόν:

1. να έχουν κοινό προφίλ ή
2. το προφίλ της διεπαφής Α να είναι σημασιολογική επέκταση ή εξειδίκευση του προφίλ της διεπαφής Β (σχήμα 3.21).

Στο Προσαρμοστικό Σχεδιαστικό Πρότυπο προτείνεται μία τεχνική, η οποία και στις δύο παραπάνω περιπτώσεις, προσαρμόζει την διεπαφή Β στην διεπαφή Α.



Σχήμα 3.21 Η υπηρεσία Adaptee θα αντικαταστήσει την υπηρεσία Target

Για να προσαρμοστεί η διεπαφή Β στην διεπαφή Α, κατασκευάζεται τεχνητά μία νέα υπηρεσία *Adapter* η οποία υλοποιεί την ίδια διεπαφή Α (σχήμα 3.21). Ο λόγος για τον οποίο κατασκευάζεται από το σύστημα η τεχνητή υπηρεσία *Adapter* είναι για να εμφωλεύσει την υπηρεσία *Adaptee*. Δηλαδή, στο εσωτερικό της τεχνητής υπηρεσίας

Adapter θα καλείται η υπηρεσία *Adaptee*. Έχοντας κατασκευάσει την υπηρεσία *Adapter*, ο πελάτης μπορεί να την χρησιμοποιήσει αντί της *Target* χωρίς αλλαγές στον κώδικά του ενώ εν αγνοία του θα καλείται η *Adaptee* (σχήμα 3.21). Ο πελάτης και η υπηρεσία *Target* δεν γνωρίζουν για την ύπαρξη της υπηρεσίας *Adaptee*. Ο συνδετικός κρίκος όλων των υπηρεσιών είναι η τεχνητή υπηρεσία *Adapter*.

ΚΕΦΑΛΑΙΟ 4. ΑΝΑΛΥΣΗ ΚΑΙ ΣΧΕΔΙΑΣΗ

4.1. Αρχιτεκτονική του συστήματος

Το σύστημα της παρούσας εργασίας υλοποιήθηκε πάνω από την πλατφόρμα *JDeveloper* [12]. Κεντρικό ρόλο στη λειτουργία του συστήματος, το οποίο υλοποιήθηκε στην παρούσα εργασία, κατέχει η οντότητα *Reconfiguration Manager* – RM. Η οντότητα RM χειρίζεται τα προφίλ των υπηρεσιών και τα χρησιμοποιεί για να πραγματοποιήσει την αντικατάσταση μίας υπηρεσίας. Πιο συγκεκριμένα, η RM αποθηκεύει πληροφορίες σχετικά με:

1. τις κατηγορίες των προφίλ,
2. τις ατομικές διεργασίες των προφίλ,
3. την δενδρική ιεράρχηση των προφίλ (βλ. ενότητα 3.4) και
4. την αντιστοίχηση των προφίλ σε διεπαφές.

Για να είναι σε θέση η οντότητα RM να χειρίζεται τις παραπάνω πληροφορίες, χρησιμοποιεί τα παρακάτω βασικά υποσυστήματα:

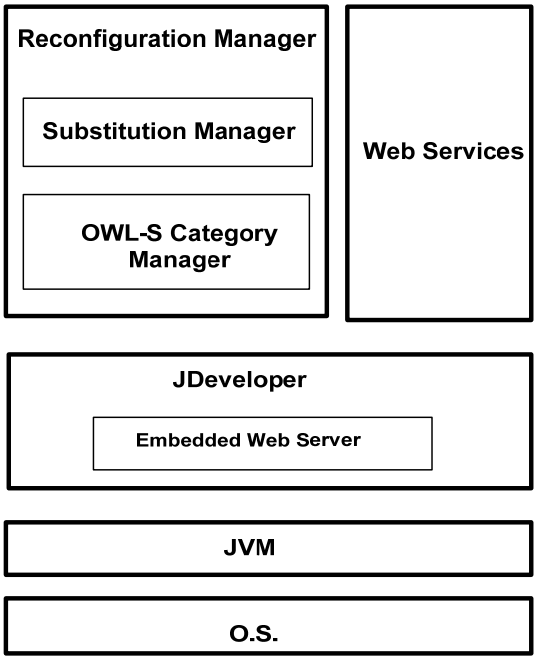
1. Το υποσύστημα *OWL-S Category Manager* (βλ. ενότητα 4.2.1)

   Αυτό το υποσύστημα χειρίζεται τις κατηγορίες των προφίλ (δημιουργία/τροποποίηση), τις ατομικές διεργασίες των προφίλ, τις αντιστοιχήσεις τους σε διεπαφές και την δενδρική ιεράρχησή τους. Επιπλέον, υλοποιεί την συντακτική και την σημασιολογική αναζήτηση υπηρεσιών.

2. Το υποσύστημα *Substitution Manager* (βλ. ενότητα 4.2.2)

   Αυτό το υποσύστημα υλοποιεί την αντικατάσταση μίας υπηρεσίας. Πιο αναλυτικά, με την βοήθεια του προηγούμενου υποσυστήματος *OWL-S Category Manager*, γίνεται συντακτική αναζήτηση ώστε να βρεθεί η κατάλληλη προς αντικατάσταση υπηρεσία. Στην περίπτωση που δεν βρεθεί υπηρεσία με συντακτικά ταυτόσημη διεπαφή, γίνεται σημασιολογική αναζήτηση. Η υπηρεσία που θα προκύψει από την σημασιολογική αναζήτηση θα παίξει τον ρόλο της *Adaptee* υπηρεσίας (σχήμα 3.22). Τέλος, κατασκευάζεται η τεχνητή υπηρεσία *Adapter*, στο εσωτερικό της οποίας, θα κληθεί η υπηρεσία *Adaptee* και γίνονται οι ελάχιστες δυνατές παρεμβάσεις στον κώδικα του πελάτη.

Το επόμενο σχήμα απεικονίζει την γενική αρχιτεκτονική του συστήματος:



Σχήμα 4.1 Η γενική αρχιτεκτονική του συστήματος

4.2. Σχεδίαση και ανάλυση του υποσυστήματος OWL-S Category Manager

4.2.1. Τα σενάρια εκτέλεσης του υποσυστήματος OWL-S Category Manager

Η λειτουργικότητα του υποσυστήματος *OWL-S Category Manager* συνοψίζεται στα ακόλουθα σενάρια εκτέλεσης (σχήμα 4.2):

1. Δημιουργία νέας κατηγορίας από προφίλ.

   Για να γίνει αποδεκτή η νέα κατηγορία από το σύστημα, θα πρέπει να περιέχει τουλάχιστον ένα προφίλ.

2. Προσθήκη ενός νέου προφίλ σε μία κατηγορία.

   Για να γίνει αποδεκτό το νέο προφίλ από το σύστημα, θα πρέπει να έχει τουλάχιστον μία ατομική διεργασία και μία αντιστοίχηση σε διεπαφή.

3. Προσθήκη αντιστοίχησης ενός προφίλ σε διεπαφή.

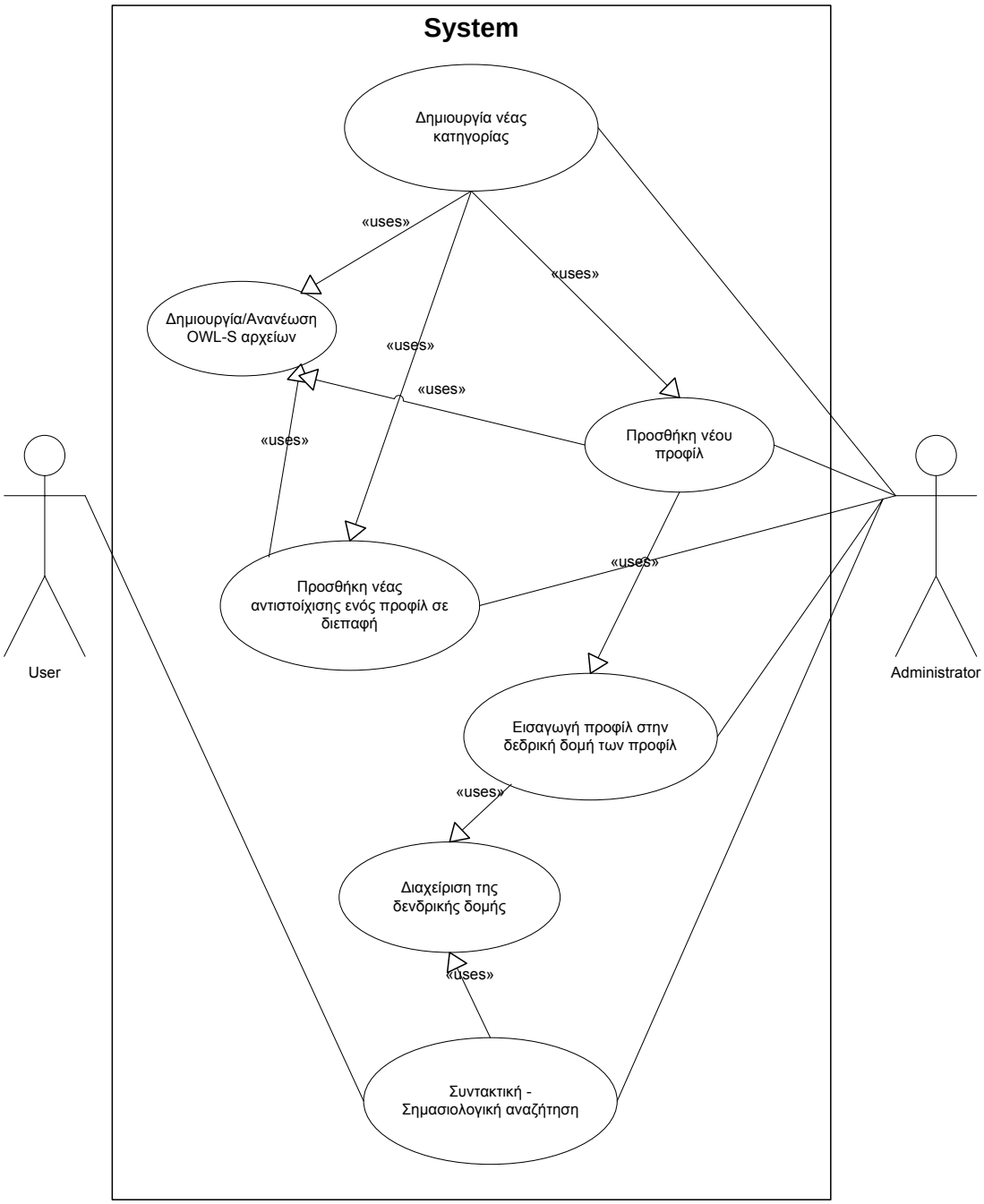
   Για να γίνει αποδεκτή η νέα αντιστοίχηση, είναι απαραίτητο να αντιστοιχίζεται κάθε λειτουργία της διεπαφής με μία ατομική διεργασία του προφίλ.

4. Εισαγωγή ενός προφίλ στην δενδρική δομή των προφίλ.

   Για να γίνει η εισαγωγή ενός νέου προφίλ στην δενδρική δομή, ακολουθείται ο αλγόριθμος που περιγράφεται στην παράγραφο 4.2.3.

5. Συντακτική ή σημασιολογική αναζήτηση ενός προφίλ.

   Η συντακτική και η σημασιολογική αναζήτηση των προφίλ χρησιμοποιεί την δενδρική δομή των προφίλ. Ο αλγόριθμος περιγράφεται αναλυτικά στην παράγραφο 4.2.4.



Σχήμα 4.2 Σενάρια εκτέλεσης του υποσυστήματος OWL-S Category Manager

4.2.2. Ανάλυση του υποσυστήματος OWL-S Category Manager

Το 1^{ο} σενάριο εκτέλεσης αφορά την δημιουργία μία νέας κατηγορίας. Η νέα κατηγορία θα περιέχει ένα τουλάχιστον προφίλ και ενδεχομένως μία αντιστοίχηση σε μία διεπαφή. Πρακτικά, αυτό σημαίνει ότι δημιουργούνται τα τέσσερα διαφορετικά OWL–S αρχεία (Category.owl, Profile.owl, Process.owl, Grounding.owl) που θα

περιγράφουν την νέα κατηγορία και τα προφίλ από τα οποία θα αποτελείται (βλ. ενότητα 3.3). Αυτά τα αρχεία δημιουργούνται από το εξαρτώμενο υποσύστημα *OWL-S Creator* (σχήμα 4.3) .

Το 2^{ο} σενάριο εκτέλεσης αφορά την προσθήκη ενός νέου προφίλ σε μία ήδη υπάρχουσα κατηγορία. Αυτό πρακτικά σημαίνει ότι θα ανανεωθούν τα περιεχόμενα των τεσσάρων διαφορετικών OWL–S αρχείων (Category.owl, Profile.owl, Process.owl, Grounding.owl) που περιγράφουν την υπάρχουσα κατηγορία. Αυτή η ανανέωση υλοποιείται από τα εξαρτώμενα υποσυστήματα *OWL-S Parser*, *OWL-S Creator* (σχήμα 4.3).

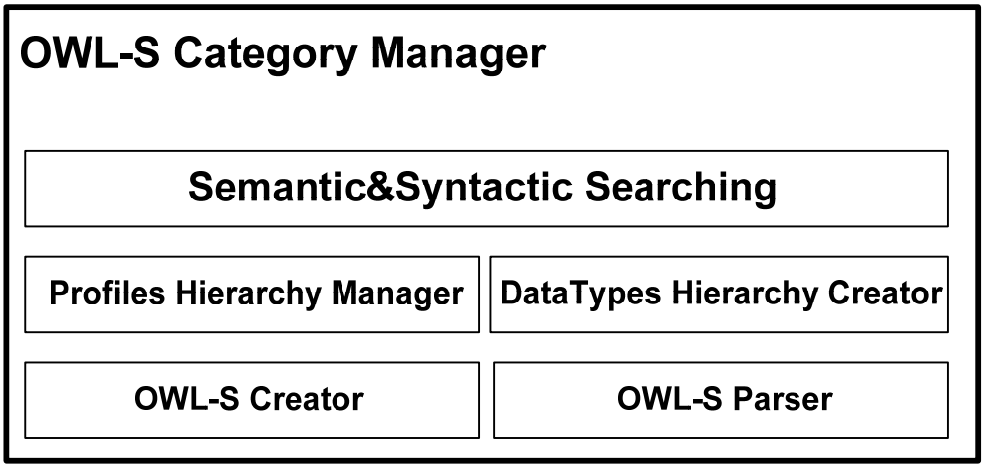
Τόσο στο πρώτο όσο και στο δεύτερο σενάριο εκτέλεσης δημιουργείται ένα νέο προφίλ. Εκτός από τις παραπάνω ενέργειες, εισάγεται το νέο προφίλ στην δενδρική δομή των προφίλ. Αυτή η ενέργεια υλοποιείται από το εξαρτώμενο υποσύστημα *Profiles Hierarchy Manager*. Ο αλγόριθμος της εισαγωγής ενός νέου προφίλ στην δενδρική δομή περιγράφεται στην επόμενη ενότητα.

Το 3^{ο} σενάριο εκτέλεσης αφορά την προσθήκη μία νέας αντιστοίχησης για ένα προφίλ σε μία διεπαφή. Θα πρέπει κάθε λειτουργία της διεπαφής να αντιστοιχηθεί σε μία ατομική διεργασία του προφίλ. Επιπλέον, θα ανανεωθούν τα περιεχόμενα του OWL–S αρχείου (Grounding.owl) που περιγράφει όλες τις αντιστοιχίσεις του συγκεκριμένου προφίλ σε διεπαφές. Αυτή η ανανέωση υλοποιείται από τα εξαρτώμενα υποσυστήματα *OWL-S Parser*, *OWL-S Creator*.

Συμπερασματικά, για τα παραπάνω σενάρια εκτέλεσης χρησιμοποιούνται τα ακόλουθα υποσυστήματα (σχήμα 4.3):

1. Το υποσύστημα *OWL-S Parser*.
   Υλοποιεί την συντακτική ανάλυση των OWL-S αρχείων.
2. Το υποσύστημα *OWL-S Creator*.
   Δημιουργεί/ανανεώνει τα *OWL-S* αρχεία.
3. Το υποσύστημα *DataTypes Hierarchy Creator*.
   Δημιουργεί την δενδρική δομή των τύπων των δεδομένων εισόδου/εξόδου. Οι τύποι των δεδομένων καθώς και η σχέση μεταξύ τους δίνονται εξαρχής ως είσοδος στο σύστημα. Αυτή η δενδρική δομή δημιουργείται κατά την έναρξη της λειτουργίας του συστήματος και δεν τροποποιείται κατά την εκτέλεσή του. Λεπτομέρειες σχετικά με τον τρόπο κατασκευής της δόθηκαν στην ενότητα 3.4.2.

4. Το υποσύστημα *Profiles Hierarchy Manager*.
   Διαχειρίζεται την δενδρική δομή των προφίλ των υπηρεσιών.
5. Το υποσύστημα *Semantic&Syntactic Searching*.
   Υλοποιεί τον αλγόριθμο της συντακτικής και της σημασιολογικής αναζήτησης προφίλ στην δενδρική δομή των προφίλ.



Σχήμα 4.3 Αρχιτεκτονική του υποσυστήματος OWL-S Category Manager

4.2.2.1. Ο αλγόριθμος εισαγωγής ενός προφίλ στην δενδρική δομή των προφίλ

Μεταξύ των τύπων των δεδομένων εισόδου/εξόδου των προφίλ μπορεί να υπάρχει σχέση σημασιολογικής επέκτασης ή εξειδίκευσης. Αυτό έχει ως αποτέλεσμα τα δεδομένα εισόδου/εξόδου να σχηματίζουν δενδρική δομή (βλ. ενότητα 3.4.2). Η δενδρική δομή, η οποία χρησιμοποιείται από το σύστημα για τους τύπους των δεδομένων εισόδου/εξόδου, κατασκευάζεται από το υποσύστημα *DataTypes Hierarchy Creator*, είναι κεντρικοποιημένη και ονομάζεται *DataTypesHierarchy*.

Με δεδομένη την δενδρική δομή *DataTypesHierarchy* των τύπων των δεδομένων εισόδου/εξόδου, προκύπτει μία ανάλογη δομή και για τα προφίλ που χρησιμοποιούν αυτά τα δεδομένα (βλ. ενότητα 3.4.2). Η δενδρική δομή των προφίλ ονομάζεται *ProfilesHierarchy*, είναι κεντρικοποιημένη και παράγεται δυναμικά.

Όταν δημιουργηθεί ένα νέο προφίλ (NewProfile), θα πρέπει να εισαχθεί στην δομή *ProfilesHierarchy*. Το σημείο όπου το νέο προφίλ θα εισαχθεί στην δομή δεν μπορεί να εντοπιστεί εξαρχής. Θα πρέπει να συγκριθεί με τα προφίλ της δομής *ProfilesHierarchy* ώστε να βρεθεί ένα υποψήφιο προφίλ (CandidateProfile) το οποίο

θα γίνει πατέρας του νέου προφίλ. Αυτή η σύγκριση έγκειται στο να ελέγξουμε εάν το νέο προφίλ είναι σημασιολογική επέκταση ή εξειδίκευση κάποιου άλλου προφίλ. Ο αλγόριθμος *CheckSubstitution* (σχήμα 4.4) που χρησιμοποιείται για την σύγκριση δύο προφίλ (NewProfile, CandidateProfile) έχει τα ακόλουθα χαρακτηριστικά:

1. Υλοποιεί τον ορισμό της σημασιολογικής επέκτασης ή εξειδίκευσης (βλ. ενότητα 3.4.1, Ορισμός 1). Πιο αναλυτικά, συγκρίνει κάθε ατομική διεργασία του νέου προφίλ με κάθε ατομική διεργασία του υποψήφιου προφίλ. Εάν για μία ατομική διεργασία του νέου προφίλ δεν βρεθεί κάποια αντίστοιχη ατομική διεργασία, τότε ο αλγόριθμος, για λόγους βελτιστοποίησης, τερματίζει ανεπιτυχώς χωρίς να ελέγξει τις υπόλοιπες ατομικές διεργασίες του νέου προφίλ. Αυτό φαίνεται από τα σημεία 3, 4 στον αλγόριθμο (σχήμα 4.4).
2. Συγκρίνει κάθε δεδομένο εισόδου/εξόδου μίας ατομικής διεργασίας του νέου προφίλ με κάθε δεδομένο εισόδου/εξόδου μίας ατομικής διεργασίας του υποψήφιου προφίλ. Για την σύγκριση των δεδομένων εισόδου/εξόδου δύο ατομικών διεργασιών, χρησιμοποιείται το κριτήριο αντικατάστασης των δεδομένων εισόδου/εξόδου, το οποίο αναφέρθηκε στην ενότητα 3.5. Εάν για έναν από τα δεδομένα εισόδου/εξόδου της ατομικής διεργασίας του νέου προφίλ δεν βρεθεί κάποιο αντίστοιχο δεδομένο, τότε, για λόγους βελτιστοποίησης, δεν ελέγχονται τα υπόλοιπα δεδομένα εισόδου/εξόδου της συγκεκριμένης ατομικής διεργασίας. Αντιθέτως, ο αλγόριθμος συγκρίνει την συγκεκριμένη ατομική διεργασία του νέου προφίλ με την επόμενη ατομική διεργασία του υποψήφιου προφίλ. Αυτό φαίνεται από τα σημεία 1, 2 στον αλγόριθμο (σχήμα 4.4).

CheckSubstitution

Είσοδος: NewProfile, CandidateProfile

Έξοδος: True : εάν το NewProfile είναι σημασιολογική επέκταση ή εξειδίκευση του CandidateProfile
False : εάν το NewProfile δεν είναι σημασιολογική επέκταση ή εξειδίκευση του CandidateProfile

```
Algorithm CheckSubstitution
  For (each Atomic Process AP1 of NewProfile) do
    For (each Atomic Process AP2 of CandidateProfile) do
      For (each Input I1 of AP1) do
        answer_input = false
        For (each Input I2 of AP2) do
          If (I1 is semantic restriction or extension of I2) Then
            answer_input = true
          End If
        End For
(1)     If (answer_input is false) Then
          break
        End If
      End For
      For (each Output O1 of AP1) do
        answer_output = false
        For (each Output O2 of AP2) do
          If (O2 is semantic restriction or extension of O1) Then
(2)         answer_output = true
          End If
        End For
        If (answer_output is false) Then
          break
        End If
      End For
      If (All inputs of AP1 and All outputs of AP1 were checked) Then
        AP1 is semantic restriction or extension of AP2
        break
      End If
    End For
(3) If (All atomic processes of CandidateProfile were checked) Then
      AP1 isn't semantic restriction or extension of any atomic process of CandidateProfile
      break
    End If
  End For
  If (All atomic processes of NewProfile were checked) Then
    NewProfile is semantic restriction or extension of CandidateProfile
    return true
  Else
(4) NewProfile isn't semantic restriction or extension of CandidateProfile
    return false
  End If
End Algorithm
```

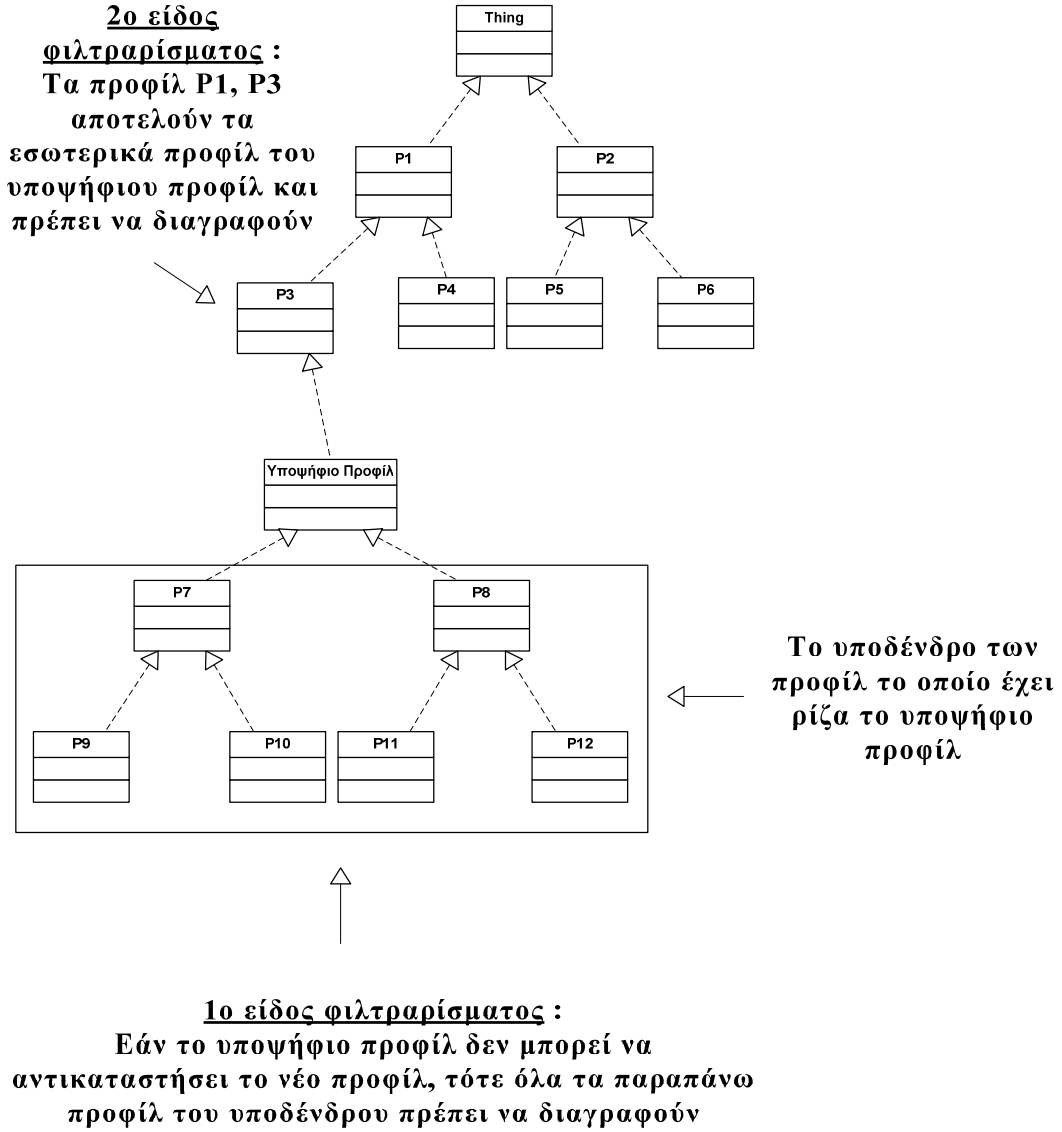
Σχήμα 4.4 Ο αλγόριθμος για την σύγκριση δύο προφίλ

Ο αλγόριθμος *CheckSubstitution*, ο οποίος υλοποιεί την σύγκριση δύο προφίλ, αποτελεί δομικό στοιχείο του αλγορίθμου *ProfilesHierarchyUpdate* για την εισαγωγή του νέου προφίλ στην δενδρική δομή *ProfilesHierarchy*. Ο αλγόριθμος *ProfilesHierarchyUpdate* (σχήμα 4.6), δέχεται ως είσοδο το νέο προφίλ (NewProfile), την δομή *ProfilesHierarchy*, την δομή *DataTypesHierarchy* και επιστρέφει την δομή *ProfilesHierarchy* ανανεωμένη. Ο αλγόριθμος περιλαμβάνει τα ακόλουθα βήματα:

1. Αρχικά, όλα τα προφίλ της δενδρικής δομής *ProfilesHierarchy*, θεωρούνται υποψήφια προς αντικατάσταση του νέου προφίλ. Αυτά τα προφίλ αποτελούν το σύνολο *CandidateSet*.
2. Ξεκινώντας από την ρίζα της δενδρικής δομής *ProfilesHierarchy*, γίνεται διάσχιση κατά πλάτος αυτής της δομής. Κάθε προφίλ που επιλέγεται από την διάσχιση κατά πλάτος, ελέγχεται εάν μπορεί να αντικαταστήσει το νέο προφίλ. Αυτός ο έλεγχος γίνεται με την βοήθεια του αλγορίθμου *CheckSubstitution*. Στην περίπτωση που το υποψήφιο προφίλ δεν μπορεί να αντικαταστήσει το νέο προφίλ, διαγράφεται από το σύνολο *CandidateSet*. Επιπλέον, διαγράφονται και όλα τα προφίλ του υποδένδρου το οποίο έχει ρίζα το υποψήφιο προφίλ. Αυτό είναι το πρώτο είδος φιλτραρίσματος που εφαρμόζεται στο σύνολο *CandidateSet* (σχήμα 4.5) και γίνεται για τον εξής λόγο: εάν ένα υποψήφιο προφίλ δεν μπορεί να αντικαταστήσει το νέο προφίλ, τότε ούτε και οι απόγονοί του στην δενδρική ιεράρχηση θα μπορούσαν.
3. Μετά την εκτέλεση του δεύτερου βήματος, το σύνολο *CandidateSet* θα περιέχει όλα τα προφίλ που μπορούν να αντικαταστήσουν το νέο προφίλ. Ωστόσο, από το σύνολο *CandidateSet*, πρέπει να διαγραφούν τα εσωτερικά προφίλ για κάθε υποψήφιο προφίλ. Για ένα υποψήφιο προφίλ, τα εσωτερικά του προφίλ στην δενδρική δομή, είναι όλα όσα σχηματίζουν το μονοπάτι από την ρίζα της δενδρικής δομής μέχρι το υποψήφιο προφίλ. Αυτό είναι το δεύτερο είδος του φιλτραρίσματος που εφαρμόζεται στο σύνολο *CandidateSet* (σχήμα 4.5) και γίνεται για τον εξής λόγο: εάν ένα υποψήφιο προφίλ μπορεί να αντικαταστήσει το νέο προφίλ, τότε και οι πρόγονοί του μπορούν να το αντικαταστήσουν. Οι πρόγονοί του είναι σημασιολογικά πιο αφηρημένα προφίλ. Όμως, ο αλγόριθμος αναζητά το λιγότερο αφηρημένο προφίλ ενός μονοπατιού ώστε να ταυτίζονται όσο το δυνατόν περισσότερο τα δεδομένα

εισόδου του και εξόδου του με αυτά του νέου προφίλ. Επομένως, οι πρόγονοί του πρέπει να διαγραφούν από το σύνολο *CandidateSet*.

4. Τελικά, το σύνολο *CandidateSet* θα έχει την μορφή που προέκυψε από το πρώτο και το δεύτερο είδος του φιλτραρίσματος. Όλα τα στοιχεία του συνόλου θα μπορούσαν να χρησιμοποιηθούν για την αντικατάσταση του νέου προφίλ. Όμως, ο αλγόριθμος επιλέγει μόνο ένα από αυτά τα προφίλ. Επιλέγεται εκείνο που έχει το μικρότερο βάθος στην δενδρική δομή. Αυτό γίνεται ώστε η δομή *ProfilesHierarchy* να διατηρείται κατά τον δυνατόν ισοζυγισμένη.



Σχήμα 4.5 Τα δύο είδη φιλτραρίσματος

ProfilesHierarchyUpdate

Είσοδος: NewProfile, DataTypesHierarchy, ProfilesHierarchy
Έξοδος: ProfilesHierarchy

```
Algorithm ProfilesHierarchyUpdate
    If (ProfilesHierarchy contains only one profile) Then
        NewProfile becomes child of Thing profile
    Else
        CandidateSet = {all profiles of ProfilesHierarchy}
        For (each CandidateProfile of ProfilesHierarchy BFS traversal) do
            result = CheckSubstitution(NewProfile, CandidateProfile)
            If (result is false) Then
                Remove CandidateProfile from CandidateSet
                Subtree = {all profiles of subtree with root the profile CandidateProfile}
                CandidateSet = CandidateSet - Subtree
            End If
        End For
        For (each CandidateProfile of CandidateSet) do
            Path = {all profiles of the path between Root and CandidateProfile}
            CandidateSet = CandidateSet - Path
        End For
        CandidateProfile = profile of CandidateSet with Minimum Depth into ProfilesHierarchy
        NewProfile becomes child of CandidateProfile
    End If
End Algorithm
```

Σχήμα 4.6 Ο αλγόριθμος ProfilesHierarchyUpdate

4.2.2.2. Ο αλγόριθμος της συντακτικής και της σημασιολογικής αναζήτησης

Στην περίπτωση που μία υπηρεσία *Target* χρησιμοποιείται στον κώδικα ενός πελάτη και χρειαστεί να αντικατασταθεί από κάποια άλλη υπηρεσία, τότε θα πρέπει να αναζητηθεί μία υπηρεσία που:

1. είτε θα έχει συντακτικά ταυτόσημη διεπαφή με την αρχική υπηρεσία *Target* (συντακτική αναζήτηση),
2. είτε θα έχει κοινό προφίλ με την αρχική υπηρεσία *Target* ($1^{η}$ περίπτωση σημασιολογικής αναζήτησης),
3. είτε θα έχει προφίλ, του οποίου, το προφίλ της αρχικής υπηρεσίας *Target* αποτελεί σημασιολογική επέκταση ή εξειδίκευση ($2^{η}$ περίπτωση σημασιολογικής αναζήτησης).

Στο σύστημα, το οποίο υλοποιήθηκε στην παρούσα εργασία, για να γίνει συντακτική και σημασιολογική αναζήτηση χρησιμοποιείται η δενδρική δομή *ProfilesHierarchy*. Ο αλγόριθμος της αναζήτησης *ServiceSearching* (σχήμα 4.7) υλοποιείται από το υποσύστημα *Semantic&Syntactic Searching*. Δέχεται ως είσοδο την αρχική υπηρεσία *Target*, την δενδρική δομή *ProfilesHierarchy* και την δομή *CategoryRepository,* η οποία έχει αποθηκευμένες όλες τις λεπτομέρειες για τα προφίλ όλων των κατηγοριών (βλ. ενότητα 4.2.5). Επιστρέφει μία υπηρεσία που μπορεί να αντικαταστήσει την αρχική υπηρεσία *Target*. Ο αλγόριθμος υλοποιεί τα εξής:

1. βρίσκει σε ποιο προφίλ (SubstitutedProfile) ανήκει η διεπαφή της αρχικής υπηρεσίας *Target*.
2. βρίσκει το σύνολο *GroundingSet* των αντιστοιχήσεων του προφίλ *SubstitutedProfile* σε διεπαφές. Εάν το σύνολο *GroundingSet* δεν είναι κενό, τότε ελέγχει:
    - εάν υπάρχει αντιστοίχηση σε υπηρεσίας, η οποία έχει κοινή διεπαφή με αυτήν της αρχικής υπηρεσίας, τότε το αποτέλεσμα της αναζήτησης είναι μόνο το URI της νέας υπηρεσίας. Αυτή είναι και η περίπτωση της συντακτικής αναζήτησης η οποία υλοποιείται στο σημείο 1 του αλγορίθμου (σχήμα 4.7).
    - αλλιώς, το αποτέλεσμα της αναζήτησης είναι μία τυχαία διεπαφή από αυτές του παραπάνω συνόλου *GroundingSet*. Αυτή είναι η περίπτωση της σημασιολογικής αναζήτησης γιατί η διεπαφή που βρέθηκε είναι σημασιολογικά ισοδύναμη με την διεπαφή της αρχικής υπηρεσίας. Η σημασιολογική αναζήτηση υλοποιείται από το σημείο 2 και κάτω του αλγορίθμου (σχήμα 4.7).
3. αλλιώς, βρίσκει το σύνολο *CandidateSet* των προφίλ, τα οποία μπορούν να αντικαταστήσουν το προφίλ *SubstitutedProfile*. Πρακτικά, το σύνολο *CandidateSet*, ταυτίζεται με το σύνολο *CandidateSet* που κατασκευάζει ο αλγόριθμος της εισαγωγής *ProfilesHierarchyUpdate* (σχήμα 4.6). Επομένως, αυτό το τμήμα του αλγορίθμου είναι ίδιο με το αντίστοιχο τμήμα του αλγορίθμου *ProfilesHierarchyUpdate*. Τελικά, επιλέγεται τυχαία ένα υποψήφιο προφίλ από αυτά του συνόλου *CandidateSet* και ο αλγόριθμος επιστρέφει στο βήμα 2 ώστε να ελέγξει τις αντιστοιχίσεις σε διεπαφές του τυχαία επιλεγόμενου προφίλ.

ServiceSearching

Είσοδος: Target, CategoryRepository, ProfilesHierarchy
Έξοδος: NewService : Η υπηρεσία που θα αντικαταστήσει την αρχική υπηρεσία Target

```
Algorithm ServiceSearching
    If (ProfilesHierarchy is Empty) Then
        return null
    Else
        SubstitutedProfile = the profile which is the semantic description of Target
        GroundingSet = {all groundings of SubstitutedProfile from CategoryRepository}
        If( GroundingSet isn't Empty) Then
            For (each CandidateGrounding of GroundingSet ) do
(1)             If (CandidateGrounding is syntactic equivalent with Service) Then
                    return CandidateGrounding
                End If
            End For
(2)         CandidateGrounding = a random grounding from GroundingSet
            return CandidateGrounding
        Else
            CandidateSet = {all profiles of ProfilesHierarchy}
            For (each CandidateProfile of CandidateSet) do
                result = CheckSubstitution(SubstitutedProfile , CandidateProfile)
                If (result is false) Then
                    Remove CandidateProfile from CandidateSet
                    Subtree = {all profiles of subtree with root the profile CandidateProfile}
                    CandidateSet = CandidateSet  - Subtree
                End If
            End For
            For (each CandidateProfile of CandidateSet) do
                Path = {all profiles of the path between Root and CandidateProfile}
                CandidateSet = CandidateSet  - Path
            End For
            SubstitutedProfile = a random profile from CandidateSet
            GroundingSet = {all groundings of SubstitutedProfile from CategoryRepository}
            If( GroundingSet isn't Empty) Then
                CandidateGrounding = a random grounding from GroundingSet
                return CandidateGrounding
            End If
            return null
    End If
End Algorithm
```

Σχήμα 4.7 Ο αλγόριθμος της συντακτικής και σημασιολογικής αναζήτησης

4.2.2.3. Οι κλάσεις υλοποίησης του υποσυστήματος OWL-S Category Manager

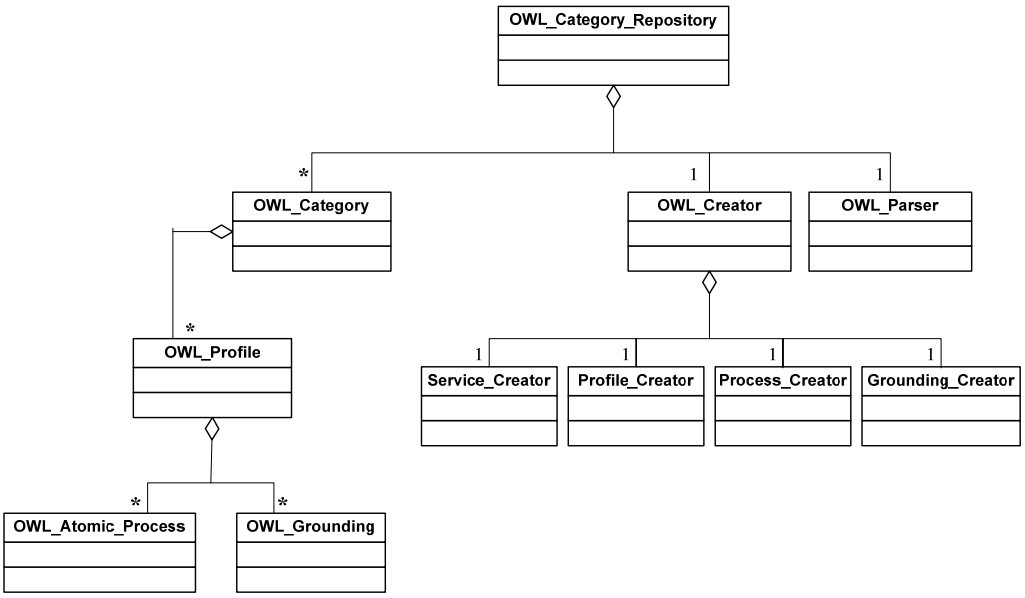
Το υποσύστημα *OWL-S Category Manager* διαχειρίζεται την αποθήκη των κατηγοριών του συστήματος. Μία από τις βασικές κλάσεις του είναι η *OWL_Category_Repository* (σχήματα 4.8 και 4.9).

Αυτή η κλάση αποθηκεύει όλες τις πληροφορίες για τις υπάρχουσες κατηγορίες. Πιο συγκεκριμένα, περιέχει ένα σύνολο αντικειμένων της κλάσης *OWL_category*. Κάθε αντικείμενο της κλάσης *OWL_category* αναπαραστά μία ξεχωριστή κατηγορία.

Η κλάση *OWL_category* περιέχει ένα σύνολο αντικειμένων της κλάσης *OWL_Profile* τα οποία αντιστοιχούν στα προφίλ μίας κατηγορίας. Κάθε αντικείμενο της κλάσης *OWL_Profile* περιέχει:

1. ένα σύνολο αναφορών της κλάσης *OWL_Atomic_Process* που αντιστοιχούν στις ατομικές διεργασίες ενός προφίλ.
2. ένα σύνολο αναφορών της κλάσης *OWL_Grounding* αναπαραστώντας τις αντιστοιχίσεις ενός προφίλ σε διεπαφές.

Επιπλέον, το υποσύστημα *OWL-S Category Manager* χρησιμοποιεί τις κλάσεις *OWL-S Creator, OWL-S Parser* (σχήμα 4.8). Η κλάση *OWL-S Creator* δημιουργεί/ανανεώνει τα περιεχόμενα των τεσσάρων διαφορετικών OWL–S αρχείων (Service.owl, Profile.owl, Process.owl, Grounding.owl) τα οποία περιγράφουν μία κατηγορία υπηρεσιών. Η κλάση *OWL-S Parser* υλοποιεί την συντακτική ανάλυση των περιεχομένων των παραπάνω OWL–S αρχείων.

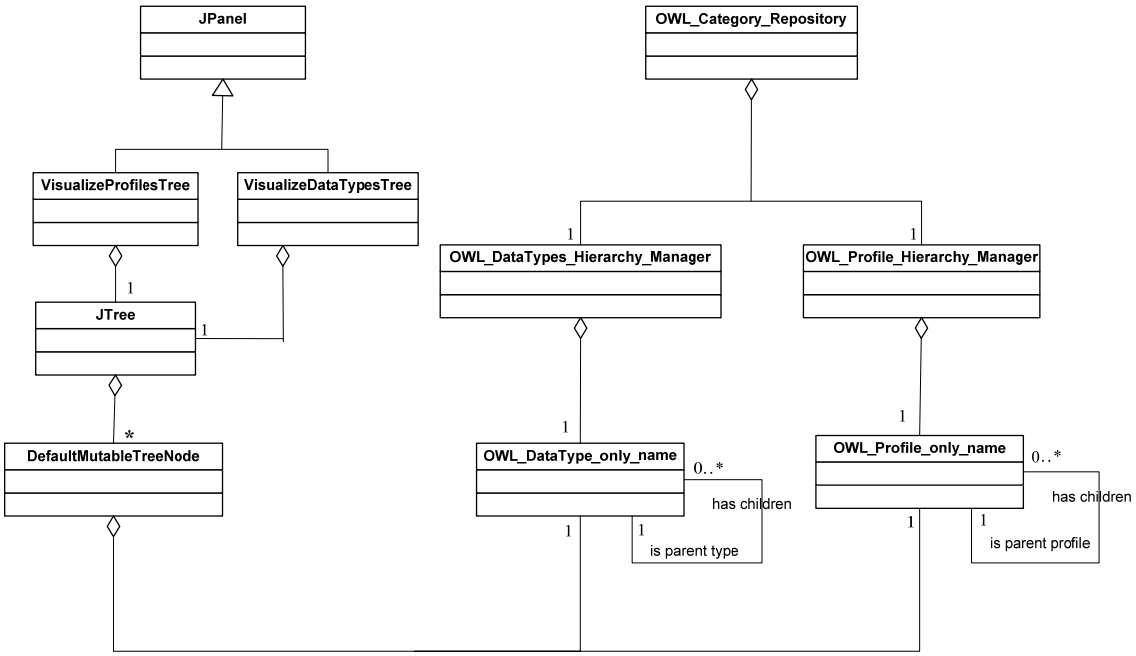


Σχήμα 4.8 Διάγραμμα κλάσεων για τον χειρισμό των κατηγοριών από προφίλ

Τέλος, δύο από τις βασικές κλάσεις του υποσυστήματος *OWL-S Category Manager* είναι η *OWL_Profile_Hierarchy_Manager* και η *OWL_DataTypes_Hierarchy_Manager* (σχήμα 4.9). Αυτές οι κλάσεις διαχειρίζονται τις δενδρικές δομές *ProfilesHierarchy* και *DataTypesHierarchy* αντίστοιχα (βλ. ενότητα 4.2.3).

Η κλάση *OWL_Profile_Hierarchy_Manager* περιλαμβάνει ένα σύνολο αντικειμένων της κλάσης *OWL_Profile_only_name*. Αυτά τα αντικείμενα αποθηκεύουν τα ονόματα των προφίλ της δενδρικής δομής *ProfilesHierarchy*. Ένα αντικείμενο της *OWL_Profile_only_name* περιέχει αναφορές σε αντικείμενα της ίδιας κλάσης *OWL_Profile_only_name*, όπου κάθε ένα από αυτά είναι μία σημασιολογική επέκτασή του ή εξειδίκευσή του. Με αυτά τα αντικείμενα και την σχέση μεταξύ τους προκύπτει η δενδρική δομή *ProfilesHierarchy*. Η δομή *ProfilesHierarchy* οπτικοποιείται με την κλάση *VisualizeProfilesTree* η οποία είναι υποκλάση (extends) της κλάσης *JPanel*.

Η κλάση *OWL_DataTypes_Hierarchy_Manager* περιλαμβάνει ένα σύνολο αντικειμένων της κλάσης *OWL_DataType_only_name*. Αυτά τα αντικείμενα αποθηκεύουν τα ονόματα των τύπων των δεδομένων εισόδου/εξόδου της δενδρικής δομής *DataTypesHierarchy*. Ένα αντικείμενο της *OWL_DataType_only_name* περιέχει αναφορές σε αντικείμενα της ίδιας κλάσης *OWL_DataType_only_name*, όπου κάθε ένα από αυτά είναι μία σημασιολογική επέκτασή του ή εξειδίκευσή του. Με αυτά τα αντικείμενα και την σχέση μεταξύ τους προκύπτει η δενδρική δομή *DataTypesHierarchy*. Η δομή *DataTypesHierarchy* οπτικοποιείται με την κλάση *VisualizeDataTypesTree* η οποία είναι υποκλάση (extends) της κλάσης *JPanel*.



Σχήμα 4.9 Διάγραμμα κλάσεων για τον χειρισμό και την οπτικοποίηση των δενδρικών δομών ProfilesHierarchy, DataTypesHierarchy

4.3. Σχεδίαση και ανάλυση του υποσυστήματος Substitution Manager

4.3.1. Τα σενάρια εκτέλεσης του υποσυστήματος Substitution Manager

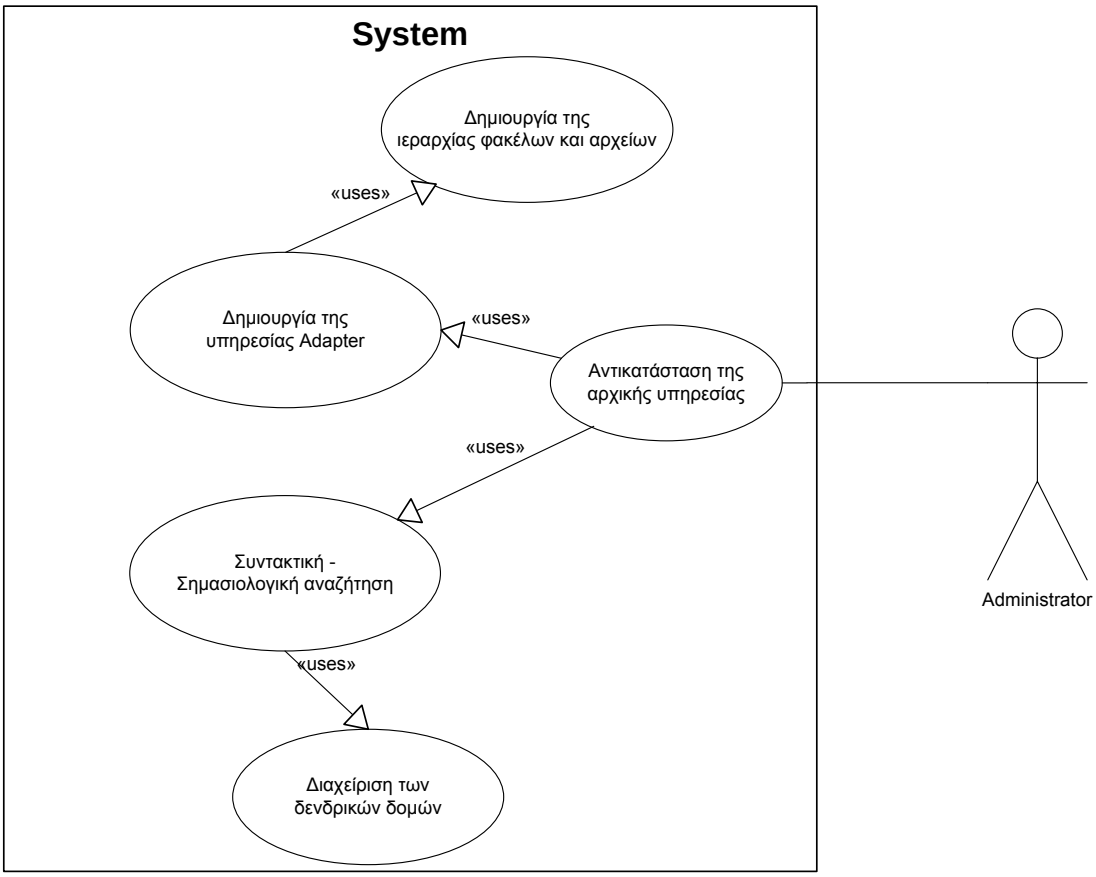
Έστω ότι ο κώδικας ενός πελάτη χρησιμοποιεί μία υπηρεσία *Target* η οποία πρέπει να αντικατασταθεί από κάποια άλλη υπηρεσία. Τον ρόλο της αντικατάστασης τον αναλαμβάνει το υποσύστημα *Substitution Manager*. Η λειτουργικότητα του υποσυστήματος *Substitution Manager* συνοψίζεται στα ακόλουθα σενάρια εκτέλεσης (σχήμα 4.10):

1. Δημιουργία της τεχνητής υπηρεσίας Adapter.

   Δημιουργείται η τεχνητή υπηρεσία *Adapter* μόνο όταν η υποψήφια προς αντικατάσταση υπηρεσία, η οποία θα επιστραφεί από την αναζήτηση, έχει διαφορετική διεπαφή από την αρχική υπηρεσία *Target* (σημασιολογική αναζήτηση). Σε αυτήν την περίπτωση, πρέπει να προσαρμοστεί η διεπαφή της υποψήφιας υπηρεσίας *Adaptee* στην διεπαφή της αρχικής υπηρεσίας *Target*. Αυτό γίνεται μέσω της υπηρεσίας *Adapter* (βλ. ενότητα 3.5.1).

2. Αντικατάσταση της υπηρεσίας στον κώδικα του πελάτη.

   Με βάση το αποτέλεσμα της αναζήτησης πρέπει να γίνουν οι κατάλληλες αλλαγές στον κώδικα του πελάτη. Στην περίπτωση της επιτυχούς συντακτικής αναζήτησης, η μοναδική αλλαγή που θα γίνει στον κώδικα θα είναι το URI της υπηρεσίας. Στην περίπτωση που θα χρησιμοποιηθεί η σημασιολογική αναζήτηση, θα παραχθεί η τεχνητή υπηρεσία *Adapter*, η οποία βοηθά ώστε και πάλι η μοναδική αλλαγή που θα γίνει στον κώδικα να είναι το URI της υπηρεσίας.



Σχήμα 4.10 Σενάρια εκτέλεσης του υποσυστήματος Substitution Manager

4.3.2. Ανάλυση του υποσυστήματος Substitution Manager

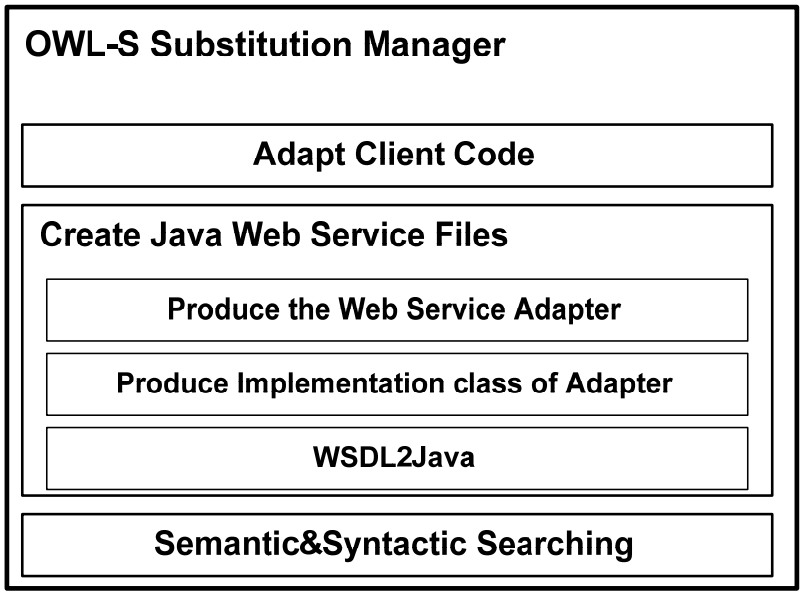
Στο πρώτο σενάριο εκτέλεσης γίνεται συντακτική και σημασιολογική αναζήτηση της υποψήφιας προς αντικατάσταση υπηρεσίας μέσω του αλγορίθμου *ServiceSearching*

(βλ. ενότητα 4.2.4). Όταν η υπηρεσία που θα επιστραφεί από την αναζήτηση έχει διαφορετική διεπαφή από την αρχική υπηρεσία, τότε δημιουργείται η τεχνητή υπηρεσία *Adapter*. Πρακτικά, δημιουργείται ολόκληρη η ιεραρχία φακέλων και αρχείων, η οποία είναι απαραίτητη για να εγκατασταθεί η τεχνητή υπηρεσία *Adapter* στον ενσωματωμένο *Web Server* της πλατφόρμας *JDeveloper*. Ένα από αυτά τα αρχεία είναι και το αρχείο υλοποίησης της υπηρεσίας *Adapter*. Το πρώτο σενάριο εκτέλεσης υλοποιείται από το εξαρτώμενο υποσύστημα *Create Java Web Service Files* και αναλύεται στην επόμενη ενότητα 4.2.8.

Το δεύτερο σενάριο εκτέλεσης αφορά την εφαρμογή των κατάλληλων αλλαγών στον κώδικα του πελάτη και υλοποιείται από το εξαρτώμενο υποσύστημα *Adapt Client Code*. Αυτό το υποσύστημα αναλύεται στην ενότητα 4.2.9.

Συμπερασματικά, για τα παραπάνω σενάρια εκτέλεσης χρησιμοποιούνται τα ακόλουθα υποσυστήματα (σχήμα 4.11):

1. Το υποσύστημα *Semantic&Syntactic Searching*.
   Υλοποιεί τον αλγόριθμο της συντακτικής και της σημασιολογικής αναζήτησης που αναλύθηκε στην παράγραφο 4.2.4.
2. Το υποσύστημα *Create Java Web Service Files*.
   Δημιουργεί την ιεραρχία φακέλων και αρχείων, η οποία είναι απαραίτητη για να γίνει *deploy* η τεχνητή υπηρεσία *Adapter* στην πλατφόρμα *JDeveloper*.
3. Το υποσύστημα *Adapt Client Code*.
   Υλοποιεί την αντικατάσταση της υπηρεσίας στον κώδικα του πελάτη.



Σχήμα 4.11 Αρχιτεκτονική του υποσυστήματος Substitution Manager

4.3.3. Οι ενέργειες για την δημιουργία της τεχνητής υπηρεσίας Adapter

Η τεχνητή υπηρεσία *Adapter* παράγεται μόνο όταν η υποψήφια προς αντικατάσταση υπηρεσία *Adaptee* έχει διαφορετική διεπαφή από την αρχική υπηρεσία *Target*. Ωστόσο, η υπηρεσία *Adapter*, θα έχει την ίδια διεπαφή με την αρχική υπηρεσία *Target* (βλ. ενότητα 3.3), αλλά θα διαφοροποιείται στην υλοποίησή της. Πρακτικά, για να μπορεί να χρησιμοποιηθεί ένα στιγμιότυπο της υπηρεσίας *Adapter* από τον κώδικα ενός πελάτη, θα πρέπει να γίνουν οι ακόλουθες ενέργειες:

1. να παραχθεί από το σύστημα ο σκελετός μίας Java κλάσης η οποία θα υλοποιεί την διεπαφή της υπηρεσίας *Adapter*,
2. να παραχθεί από το σύστημα το σώμα της προαναφερθήσας Java κλάσης όπου στο εσωτερικό του θα καλείται η υπηρεσία *Adaptee*,
3. να παραχθεί και να εγκατασταθεί η υπηρεσία διαδικτύου *Adapter* στην πλατφόρμα ενδιάμεσου λογισμικού *JDeveloper*.

4.3.3.1. Ανάλυση της διαδικασίας παραγωγής του σκελετού της κλάσης υλοποίησης

Αυτή η διαδικασία είναι τμήμα του αλγορίθμου *ProduceImplementationClass* (σχήμα 4.20) και υλοποιείται από το υποσύστημα *WSDL2Java*. Η διεπαφή της υπηρεσίας *Adapter* ταυτίζεται με την διεπαφή της αρχικής υπηρεσίας *Target*. Επομένως, από

αυτόν τον αλγόριθμο θα παραχθεί ο σκελετός για μία Java κλάση που θα υλοποιεί αυτήν την διεπαφή. Αυτός ο σκελετός θα περιλαμβάνει τα πρωτότυπα των μεθόδων της υπηρεσίας *Target*. Για κάθε μέθοδο θα παραχθούν τα ακόλουθα δεδομένα εισόδου/εξόδου:

1. Ένα δεδομένο εισόδου για κάθε τμήμα (part) του αντίστοιχου μηνύματος εισόδου της υπηρεσίας *Target*. Κάθε ένα από τα μηνύματα εισόδου είναι πιθανόν να περιέχει κάποιον από τους βασικούς XML τύπους ή έναν σύνθετο τύπο που προκύπτει από συνδυασμό των βασικών XML τύπων. Στην παρούσα εργασία γίνεται η υπόθεση ότι ένα μήνυμα εισόδου περιέχει μόνο κάποιον από τους βασικούς τύπους του σχήματος 4.12 ή έναν σύνθετο τύπο από συνδυασμό αυτών. Επιπλέον, στο σχήμα 4.12 δίνεται και η αντιστοίχησή τους σε Java τύπους, σύμφωνα με το JAX-RPC.
2. Ένα δεδομένο εξόδου για κάθε τμήμα (part) του αντίστοιχου μηνύματος εξόδου της υπηρεσίας *Target*. Το δεδομένο εξόδου θα είναι αντικείμενο της κλάσης *Holder*. Εάν το τμήμα του μηνύματος είναι σύνθετος XML τύπος, τότε κάθε ένας από τους επιμέρους τύπους που τον αποτελούν, αντιστοιχίζεται σε ένα αντικείμενο της κλάσης *Holder*.

Αντιστοίχιση βασικών XML τύπων σε Java τύπους

Simple XML type	Java type
xsd:string	java.lang.String
xsd:int	int
xsd:float	float
xsd:double	double
xsd:boolean	boolean

Σχήμα 4.12 Οι βασικοί XML τύποι που χρησιμοποιούνται στην παρούσα εργασία

Γενικά, η κλάση *Holder* χρησιμοποιείται στις μεθόδους των υπηρεσιών διαδικτύου. Σύμφωνα με το πρότυπο JAX-RPC, τα αντικείμενα αυτής της κλάσης λειτουργούν ως

δεδομένα εξόδου (inout/out parameters). Η αντιστοίχηση μερικών βασικών XML τύπων σε *Holder* κλάσεις δίνεται στο επόμενο σχήμα:

Αντιστοίχιση βασικών XML τύπων σε Holder κλάσεις

Simple XML type	Java Holder class
xsd:string	StringHolder
xsd:int	IntHolder
xsd:float	FloatHolder
xsd:double	DoubleHolder
xsd:boolean	BooleanHolder

Σχήμα 4.13 Η αντιστοίχηση μερικών βασικών XML τύπων σε Holder κλάσεις

Παράδειγμα:

Ας θεωρήσουμε, ως συνέχεια του παραδείγματος της ενότητας 3.1.1, ότι η διεπαφή της υπηρεσίας *Target* (σχήμα 4.14) παρέχει:

1. τη μέθοδο *getLaptopByBrandAndPrice*,
2. η οποία δέχεται ως είσοδο το μήνυμα *wsdl_BrandAndPrice* και
3. επιστρέφει ως έξοδο το μήνυμα *wsdl_NumberOfLaptops*.

Η διεπαφή της υπηρεσίας Target σε WSDL

```
...
<message name="wsdl_BrandAndPrice">
        <part name="wsdl_company" element="xsd:string"/>
        <part name="wsdl_amount" element="xsd:double"/>
</message>

<message name="wsdl_NumberOfLaptops">
        <part name="wsdl_Laptops" element="xsd:int"/>
</message>

<portType name="LaptopBuyingByBrandAndPricePortType">
        <operation name="getLaptopByBrandAndPrice">
                <input message="tns:wsdl_BrandAndPrice"/>
                <output message="tns:wsdl_NumberOfLaptop"/>
        </operation>
</portType>
                                        ...
```

Σχήμα 4.14 Παράδειγμα για την διεπαφή της υπηρεσίας Target

Η τεχνητή υπηρεσία *Adapter* θα υλοποιεί την παραπάνω διεπαφή της υπηρεσίας *Target* (σχήμα 4.14). Ο σκελετός για την Java κλάση υλοποίησής της, ο οποίος θα παραχθεί από το υποσύστημα *WSDL2Java*, είναι ο ακόλουθος:

Ο παραγόμενος σκελετός της Java κλάσης υλοποίησης

```
package Targetpackage;

import org.omg.CORBA.DoubleHolder;
import org.omg.CORBA.StringHolder;

public class Target{
    public Target(){
    }

    public void getLaptopByBrandAndPrice(IntHolder wsdl_Laptops, String wsdl_company, double wsdl_amount){
                ...
    }
}
```

Σχήμα 4.15 Ο σκελετός της Java κλάσης υλοποίησης της υπηρεσίας Adapter

4.3.3.2. Ανάλυση της διαδικασίας παραγωγής του σώματος της κλάσης υλοποίησης

Αυτή η διαδικασία είναι τμήμα του αλγορίθμου *ProduceImplementationClass* (σχήμα 4.20) και υλοποιείται από το υποσύστημα *Produce Implementation class of Adapter*. Με δεδομένο τον σκελετό της κλάσης υλοποίησης της υπηρεσίας *Adapter*, θα παραχθεί το σώμα της Java κλάσης, το οποίο θα καλεί την υπηρεσία *Adaptee*. Για αυτήν την κλήση, θα πρέπει να παραχθεί Java κώδικας ο οποίος θα υλοποιεί τις ακόλουθες ενέργειες:

1. Δυναμική προσαρμογή των δεδομένων εισόδου/εξόδου της μεθόδου της υπηρεσίας *Target* στα δεδομένα εισόδου/εξόδου της μεθόδου της υπηρεσίας *Adaptee*.
2. Δυναμική κλήση της μεθόδου της υπηρεσίας *Adaptee*.

Η πρώτη ενέργεια έχει σχέση με τον συσχετισμό των δεδομένων εισόδου/εξόδου της μεθόδου της υπηρεσίας *Target* με τα δεδομένα εισόδου/εξόδου της μεθόδου της υπηρεσίας *Adaptee*. Αυτό πρακτικά σημαίνει ότι πρέπει να αντιστοιχηθεί ένα δεδομένο εισόδου/εξόδου της μεθόδου της υπηρεσίας *Adaptee* σε ένα δεδομένο εισόδου/εξόδου της μεθόδου της υπηρεσίας *Target*. Η πληροφορία αυτής της αντιστοίχησης έχει ήδη βρεθεί και αποθηκευτεί από τον αλγόριθμο της συντακτικής

και σημασιολογικής αναζήτησης *ServiceSearching* (βλ. ενότητα 4.2.2.2). Πιο αναλυτικά, από αυτόν τον αλγόριθμο προέκυψε ποια μέθοδος της υπηρεσίας *Adaptee* μπορεί να αντικαταστήσει κάθε μία από τις μεθόδους της υπηρεσίας *Target*. Επομένως, για τα δεδομένα εισόδου τους και εξόδου τους ισχύουν οι εξής παρατηρήσεις:

1. Για κάθε δεδομένο εισόδου T_2 της μεθόδου της υπηρεσίας *Adaptee* υπάρχει ένα δεδομένο εισόδου T_1 της μεθόδου της υπηρεσίας *Target* το οποίο είναι σημασιολογική επέκτασή του ή εξειδίκευσή του. Στην περίπτωση που το T_1 είναι εξειδίκευση, το πεδίο τιμών κάθε ιδιότητας του T_2 είναι υπερσύνολο του πεδίου τιμών για την αντίστοιχη ιδιότητα του T_1 (βλ. ενότητα 3.4.1 – Ορισμός 2a). Επομένως, λόγω της σχέσης του υπερσυνόλου, οι τιμές για κάθε ιδιότητα του T_1 μπορεί να χρησιμοποιηθούν απευθείας, χωρίς καμία τροποποίηση, ως τιμές για τις αντίστοιχες ιδιότητες του T_2. Στην περίπτωση που το T_1 είναι επέκταση, κάθε ιδιότητα του T_1 ταυτίζεται με μία ιδιότητα του T_2 αλλά το πλήθος των ιδιοτήτων του T_2 είναι υποσύνολο του πλήθος των ιδιοτήτων του T_1 (βλ. ενότητα 3.4.1 – Ορισμός 2b). Επομένως, λόγω της ταύτισης των ιδιοτήτων, ως τιμές για κάθε ιδιότητα του T_2 μπορεί να χρησιμοποιηθούν απευθείας, χωρίς καμία τροποποίηση, οι τιμές των αντίστοιχων ιδιοτήτων του T_1.
2. Για κάθε δεδομένο εξόδου T_1 της μεθόδου της υπηρεσίας *Target* υπάρχει ένα δεδομένο εξόδου T_2 της μεθόδου της υπηρεσίας *Adaptee* το οποίο είναι σημασιολογική επέκτασή του ή εξειδίκευσή του. Ισχύουν αντίστοιχες παρατηρήσεις με αυτές των δεδομένων εισόδου.

Από τις παραπάνω παρατηρήσεις, προκύπτει το συμπέρασμα ότι, για την πρώτη ενέργεια, λόγω της σχέσης της σημασιολογικής επέκτασης ή εξειδίκευσης που έχουν τα δεδομένα εισόδου/εξόδου, αρκεί να χρησιμοποιηθούν, χωρίς ριζικές τροποποιήσεις, οι τιμές για κάποιες από τις ιδιότητες του T_2 ως τιμές στις αντίστοιχες ιδιότητες του T_1. Τελικά, αλγόριθμος *ProduceBody* θα πρέπει να διατάξει τα δεδομένα εισόδου της μεθόδου της υπηρεσίας *Target* ώστε να χρησιμοποιηθούν ως δεδομένα εισόδου στην κλήση της μεθόδου της υπηρεσίας *Adaptee* (σχήμα 4.19). Όπως αναφέρθηκε παραπάνω, αυτή η διάταξη έχει ήδη προσδιοριστεί από τον αλγόριθμο *ServiceSearching*.

Σχετικά, με την δεύτερη ενέργεια, θα πρέπει να παραχθεί δυναμικά κώδικας για την κλήση της μεθόδου της υπηρεσίας *Adaptee*. Αυτός ο κώδικας θα τοποθετηθεί εντός του σώματος της μεθόδου της υπηρεσίας *Adapter* και πρέπει να ακολουθεί την διεπαφή της δυναμικής κλήσης των υπηρεσιών διαδικτύου σύμφωνα με το πρότυπο JAX-RPC (Dynamic Invocation Interface – DII). Αυτή η διεπαφή δίνει την δυνατότητα να καλείται δυναμικά η μέθοδος της υπηρεσίας *Adaptee* (σχήμα 4.20), καθορίζοντας τις ακόλουθες παραμέτρους:

1. το όνομα της θύρας της υπηρεσίας διαδικτύου (portName),
2. την διεύθυνση της υπηρεσίας διαδικτύου (setTargetEndpointAddress),
3. το όνομα της μεθόδου που θα κληθεί (setOperation),
4. το όνομα και τον τύπο των δεδομένων εισόδου/εξόδου της μεθόδου (addParameter) και
5. τις τιμές των δεδομένων εισόδου/εξόδου στην κλήση της μεθόδου (invoke).

Παράδειγμα:

Ας θεωρήσουμε, ως συνέχεια του παραδείγματος της ενότητας 3.1.1, ότι η διεπαφή της υπηρεσίας *Adaptee* (σχήμα 4.17) παρέχει:

1. τη μέθοδο *getLaptopByPrice*,
2. η οποία δέχεται ως είσοδο το μήνυμα *wsdl_Price* και
3. επιστρέφει ως έξοδο το μήνυμα *wsdl_NumberOfLaptops*.

Η διεπαφή της υπηρεσίας Adaptee σε WSDL

```
...
<message name="wsdl_price">
        <part name="wsdl_amount" element="xsd:double"/>
</message>

<message name="wsdl_NumberOfLaptops">
        <part name="wsdl_Laptops" element="xsd:int"/>
</message>

<portType name="LaptopBuyingByPricePortType">
        <operation name="getLaptopByPrice">
                <input message="tns:wsdl_price"/>
                <output message="tns:wsdl_NumberOfLaptops"/>
        </operation>
</portType>
...
```

Σχήμα 4.16 Παράδειγμα για την διεπαφή της υπηρεσίας Adaptee

Ένα παράδειγμα κλήσης της μεθόδου *getLaptopByPrice* της υπηρεσίας *Adaptee* σύμφωνα με το πρότυπο JAX-RPC είναι το ακόλουθο:

Dynamic Invocation Interface - DII

```
try{
    String portName = "AdapteeSoapHttpPort";
    QName serviceName = new QName("Adaptee");
    ServiceFactory factory = ServiceFactory.newInstance();
    Service service = factory.createService(serviceName);

    Call call = service.createCall(new QName(portName));
    call.setTargetEndpointAddress("http://localhost:8888/Adaptee_WebService-Adaptee-context-root/AdapteeSoapHttpPort");

    QName operationName = new QName("http://Adapteepackage/","getLaptopByPrice");
    call.setOperationName(operationName);
    call.addParameter("wsdl_Laptops", new QName("http://www.w3.org/2001/XMLSchema", "double"), ParameterMode.OUT);
    call.addParameter("wsdl_amount", new QName("http://www.w3.org/2001/XMLSchema","int"), ParameterMode.IN);
    call.setReturnType(XMLType.XSD_STRING);
    call.setProperty(Call.OPERATION_STYLE_PROPERTY, "rpc");
    call.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY, "http://schemas.xmlsoap.org/soap/encoding/");

    double tmp_amount = 1000;
    Object[] actualArgs = {tmp_amount};
    response = (String) call.invoke(actualArgs);

}catch(Exception ex){ex.printStackTrace();}
```

Σχήμα 4.17 Παράδειγμα δυναμικής κλήσης μίας μεθόδου υπηρεσίας σύμφωνα με το JAX-RPC

Σύμφωνα με τα παραδείγματα διεπαφών για τις υπηρεσίες *Target* και *Adaptee* που αναφέρθηκαν στις ενότητες 4.3.3.1 και 4.3.3.2 αντίστοιχα, η κλήση υλοποίησης που θα προκύψει για την υπηρεσία *Adapter* είναι η ακόλουθη:

Η κλάση υλοποίησης της υπηρεσίας Adapter

```
package Adapterpackage;

import org.omg.CORBA.DoubleHolder;
import org.omg.CORBA.StringHolder;

public class Adapter{
 public Adapter(){}

 public void getLaptopByBrandAndPrice(intHolder wsdl_Laptops, String wsdl_company, double wsdl_amount){
    try{
       String portName = "AdapteeSoapHttpPort";
       QName serviceName = new QName("Adaptee");
       ServiceFactory factory = ServiceFactory.newInstance();
       Service service = factory.createService(serviceName);

       Call call = service.createCall(new QName(portName));
       call.setTargetEndpointAddress("http://localhost:8888/Adaptee_WebService-Adaptee-context-root/AdapteeSoapHttpPort");

       QName operationName = new QName("http://Adapteepackage/","getLaptopByPrice");
       call.setOperationName(operationName);
       call.addParameter("wsdl_Laptops", new QName("http://www.w3.org/2001/XMLSchema", "int"), ParameterMode.OUT);
       call.addParameter("wsdl_amount", new QName("http://www.w3.org/2001/XMLSchema","double"), ParameterMode.IN);
       call.setReturnType(XMLType.XSD_STRING);
       call.setProperty(Call.OPERATION_STYLE_PROPERTY, "rpc");
       call.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY, "http://schemas.xmlsoap.org/soap/encoding/");

      Object[] actualArgs = {wsdl_amount};
      call.invoke(actualArgs);

    }catch(Exception ex){ex.printStackTrace();}
 }
}
```

Σχήμα 4.18 Παράδειγμα για την παραγόμενη Java κλάση υλοποίησης της υπηρεσίας Adapter

ProduceImplementationClass

Είσοδος: Οι διεπαφές των υπηρεσιών Target, Adaptee

Έξοδος: Δημιουργείται το σώμα κάθε μεθόδου TargetOperation της υπηρεσίας Target και αποθηκεύεται εντός του αρχείου Adapter.java στον δίσκο

```
Algorithm ProduceImplementationClass
    Define name of Java package as Adapterpackage
    Import Java Holder classes
    Define the Default constructor
    For (each TargetOperation) do
        For (each part of Input Message) do
            Define the corresponding parameter
        End For
        For (each part of Output Message) do
            Define the corresponding Holder object
        End For
        Set the parameters of the DII call
        For (each AdapteeParameter of AdapteeOperation) do
            Find the corresponding TargetParameter of TargetOperation
            Set TargetParameter as parameter of AdapteeOperation for the DII call
        End For
    End For
    Create a Java file with the name Adapter.java and with the above contents
End Algorithm
```

Σχήμα 4.19 Ο αλγόριθμος παραγωγής της κλάσης υλοποίησης

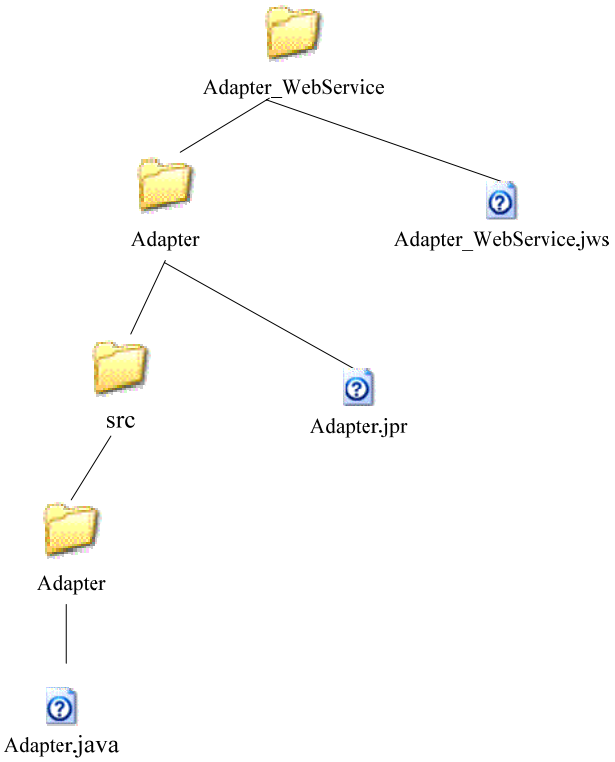
4.3.3.3. Ανάλυση της διαδικασίας εγκατάστασης της τεχνητής υπηρεσίας Adapter

Πρακτικά, για να εγκατασταθεί η τεχνητή υπηρεσία *Adapter* στην πλατφόρμα *JDeveloper*, πρέπει προηγουμένως να δημιουργηθεί δυναμικά από το σύστημα μία ιεραρχία φακέλων και αρχείων με συγκεκριμένη δομή (σχήμα 4.20). Η δυναμική παραγωγή αυτής της ιεραρχίας φακέλων και αρχείων γίνεται από το υποσύστημα *Produce the Web Service Adapter*.

Παράδειγμα:

Ως συνέχεια του παραδείγματος της ενότητας 3.1.1, έστω ότι η υπηρεσία *Target* έχει το προφίλ *LaptopBuying_ByBrandAndPrice* ενώ η υπηρεσία *Adaptee* έχει το προφίλ *LaptopBuying_ByPrice*. Η κλάση υλοποίησης της υπηρεσίας *Adapter* περιέχεται στο αρχείο *Adapter.java* και αποθηκεύεται εντός του φακέλου *Adapterpackage*. Επιπλέον,

παράγονται και τα αρχεία *Adapter_WebService.jws*, *Adapter.jpr* (σχήματα 4.21 και 4.22) τα οποία είναι απαραίτητα για να εγκατασταθεί η υπηρεσία *Adapter* στην πλατφόρμα *JDeveloper*.



Σχήμα 4.20 Η ιεραρχία φακέλων και αρχείων της τεχνητής υπηρεσίας Adapter

Το αρχείο Adapter_WebService.jws

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<jws:workspace xmlns:jws="http://xmlns.oracle.com/ide/project">
  <hash n="component-versions">
    <value n="oracle.ide.model.Project" v="10.1.3.3.0"/>
    <value n="oracle.jdevimpl.xml.oc4j.ds.DataSourcesMigratorHelper" v="10.1.3.3.0"/>
  </hash>
  <list n="listOfChildren">
    <hash>
      <value n="nodeClass" v="oracle.ide.model.Project"/>
      <url n="URL" path="Adapter/Adapter.jpr"/>
    </hash>
  </list>
</jws:workspace>
```

Σχήμα 4.21 Τα περιεχόμενα του αρχείου Adapter_WebService.jws

Το αρχείο Adapter.jpr

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<jpr:project xmlns:jpr="http://xmlns.oracle.com/ide/project">
  <hash n="component-versions">
    <value n="oracle.adf.dt.migration.ProjectMigrator" v="10.1.3.3.0"/>
  </hash>
  <value n="defaultPackage" v="Adapter"/>
  <hash n="oracle.bm.commonIde.data.project.ModelerProjectSettings">
    <hash n="modelersContentSet">
      <list n="url-path">
        <url path="model/"/>
      </list>
    </hash>
  </hash>
  <hash n="oracle.ide.model.ResourcePaths">
    <hash n="resourcesContentSet">
      <list n="pattern-filters">
        <string v="+*"/>
      </list>
      <list n="url-path">
        <url path="."/>
      </list>
    </hash>
  </hash>
  <hash n="oracle.jdeveloper.compiler.OjcConfiguration">
    <value n="internalEncoding" v="Cp1252"/>
  </hash>
  <hash n="oracle.jdeveloper.model.J2eeSettings">
    <value n="j2eeWebAppName" v="Adapter_WebService-Adapter-webapp"/>
    <value n="j2eeWebContextRoot" v="Adapter_WebService-Adapter-context-root"/>
    <hash n="webContentSet">
      <list n="url-path">
        <url path="public_html/"/>
      </list>
    </hash>
  </hash>
  <hash n="oracle.jdeveloper.model.PathsConfiguration">
    <hash n="javaContentSet">
      <list n="pattern-filters">
        <string v="+**"/>
      </list>
      <list n="url-path">
        <url path="src/"/>
      </list>
    </hash>
  </hash>
  <hash n="oracle.jdeveloper.runner.RunConfigurations">
    <hash n="runConfigurationDefinitions">
      <hash n="Default">
        <value n="custom" v="false"/>
        <value n="name" v="Default"/>
      </hash>
    </hash>
    <list n="runConfigurationList">
      <string v="Default"/>
    </list>
  </hash>
  <hash n="oracle.jdevimpl.config.JProjectPaths">
    <url n="outputDirectory" path="classes/"/>
  </hash>
  <hash n="oracle.tip.tools.ide.pm.addin.PMProjectSettings">
    <hash n="Integration_Content">
      <list n="pattern-filters">
        <string v="+**"/>
      </list>
    </hash>
  </hash>
</jpr:project>
```

Σχήμα 4.22 Τα περιεχόμενα του αρχείου Adapter.jpr

4.3.3.4. Οι αλλαγές στον κώδικα του πελάτη

Στην περίπτωση που μία υπηρεσία *Target* χρησιμοποιείται στον κώδικα ενός πελάτη και χρειαστεί να αντικατασταθεί από κάποια άλλη υπηρεσία, τότε θα πρέπει με την βοήθεια του αλγορίθμου *ServiceSearching* (βλ. ενότητα 4.2.2.2) να αναζητηθεί μία υπηρεσία *Adaptee*. Για το αποτέλεσμα της αναζήτησης υπάρχουν δύο περιπτώσεις:

1. Εάν η συντακτική αναζήτηση έχει αποτέλεσμα, τότε η υπηρεσία *Adaptee* θα έχει κοινή διεπαφή με την υπηρεσία *Target*. Σε αυτήν την περίπτωση, η μοναδική αλλαγή που θα γίνει στον κώδικα του πελάτη θα είναι το όνομα και το URI της υπηρεσίας *Target* (σημεία 1-3 στα σχήματα 4.23, 4.24).
2. Αλλιώς, η υπηρεσία *Adaptee* θα έχει διαφορετική διεπαφή από την υπηρεσία *Target*. Ωστόσο, σε αυτήν την περίπτωση, η υπηρεσία *Adaptee* καλείται μέσω της τεχνητής υπηρεσίας *Adapter*. Ο κώδικας του πελάτη θα χρησιμοποιεί την *Adapter*, η οποία κατασκευάζεται με τέτοιο τρόπο από το σύστημα, ώστε να έχει την ίδια διεπαφή με την αρχική υπηρεσία *Target*. Λόγω της ίδιας διεπαφής, η μοναδική αλλαγή που θα γίνει και πάλι στον κώδικα του πελάτη θα είναι το όνομα και το URI της υπηρεσίας *Adapter* (σημεία 1-3 στα σχήματα 4.23, 4.24).

Παράδειγμα:

Ως συνέχεια του παραδείγματος της ενότητας 4.3.3.1, έστω ότι ο κώδικας του πελάτη χρησιμοποιεί μία υπηρεσία *Target* η οποία έχει την διεπαφή του σχήματος 4.13. Η δυναμική JAX-RPC κλήση της είναι η ακόλουθη:

Κλήση της υπηρεσίας Target

```
try{
(1) String portName = "TargetSoapHttpPort";
(2) QName serviceName = new QName("Target");
    ServiceFactory factory = ServiceFactory.newInstance();
    Service service = factory.createService(serviceName);

    Call call = service.createCall(new QName(portName));
(3) call.setTargetEndpointAddress("http://localhost:8888/Target_WebService-Target-context-root/TargetSoapHttpPort");

    QName operationName = new QName("http://Targetpackage/","getLaptopByBrandAndPrice");
    call.setOperationName(operationName);
    call.addParameter("wsdl_Laptops", new QName("http://www.w3.org/2001/XMLSchema", "double"), ParameterMode.OUT);
    call.addParameter("wsdl_company", new QName("http://www.w3.org/2001/XMLSchema","String"), ParameterMode.IN);
    call.addParameter("wsdl_amount", new QName("http://www.w3.org/2001/XMLSchema","double"), ParameterMode.IN);
    call.setReturnType(XMLType.XSD_STRING);
    call.setProperty(Call.OPERATION_STYLE_PROPERTY, "rpc");
    call.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY, "http://schemas.xmlsoap.org/soap/encoding/");

    String tmp_company = "TOSHIBA";
    double tmp_amount = 1000;
    Object[] actualArgs = {tmp_company, tmp_amount};
    response = (String) call.invoke(actualArgs);

}catch(Exception ex){ex.printStackTrace();}
```

Σχήμα 4.23 Τμήμα του κώδικα του πελάτη που περιέχει την κλήση της υπηρεσίας Target

Στην περίπτωση που το σύστημα κατασκευάσει την υπηρεσία *Adapter*, τότε στον κώδικα του πελάτη θα αλλάξει μόνο το όνομα και το URI της υπηρεσίας *Target* (σημεία 1-3 στα σχήματα 4.23, 4.24):

Κλήση της υπηρεσίας Adapter

```
try{
(1)  String portName = "AdapterSoapHttpPort";
(2)  QName serviceName = new QName("Adapter");
     ServiceFactory factory = ServiceFactory.newInstance();
     Service service = factory.createService(serviceName);

     Call call = service.createCall(new QName(portName));
(3)  call.setTargetEndpointAddress("http://localhost:8888/Adapter_WebService-Adapter-context-root/AdapterSoapHttpPort");

     QName operationName = new QName("http://Adapterpackage/","getLaptopByBrandAndPrice");
     call.setOperationName(operationName);
     call.addParameter("wsdl_Laptops", new QName("http://www.w3.org/2001/XMLSchema", "int"), ParameterMode.OUT);
     call.addParameter("wsdl_company", new QName("http://www.w3.org/2001/XMLSchema","String"), ParameterMode.IN);
     call.addParameter("wsdl_amount", new QName("http://www.w3.org/2001/XMLSchema","double"), ParameterMode.IN);
     call.setReturnType(XMLType.XSD_STRING);
     call.setProperty(Call.OPERATION_STYLE_PROPERTY, "rpc");
     call.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY, "http://schemas.xmlsoap.org/soap/encoding/");

     String tmp_company = "TOSHIBA";
     double tmp_amount = 1000;
     Object[] actualArgs = {tmp_company, tmp_amount};
     response = (String) call.invoke(actualArgs);

}catch(Exception ex){ex.printStackTrace();}
```

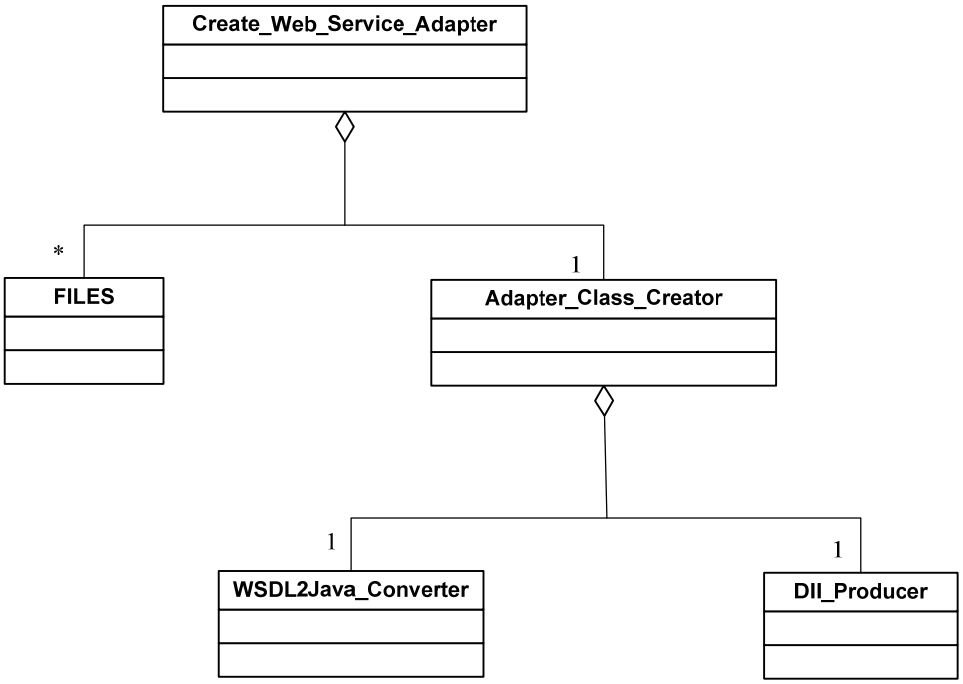
Σχήμα 4.24 Ο κώδικας του πελάτη μετά την αντικατάσταση της υπηρεσίας Target από την υπηρεσία Adapter

4.3.3.5. Οι κλάσεις υλοποίησης του υποσυστήματος Substitution Manager

Το υποσύστημα *Substitution Manager* διαχειρίζεται τις αντικαταστάσεις των υπηρεσιών διαδικτύου. Μία από τις βασικές κλάσεις του είναι η *Create_Web_Service_Adapter* (σχήμα 4.25). Αυτή η κλάση:

1. Δημιουργεί στον δίσκο το αρχείο *Adapter.java*, το οποίο περιέχει την κλάση υλοποίησης της τεχνητής υπηρεσίας *Adapter*. Γι' αυτόν τον σκοπό χρησιμοποιεί αντικείμενα δύο επιπλέον κλάσεων: της *Adapter_Class_Creator* και της *FILES*. Η κλάση *FILES* δημιουργεί στον δίσκο το αρχείο *Adapter.java*. Η κλάση *Adapter_Class_Creator* χρησιμοποιεί αντικείμενα δύο επιπλέον κλάσεων: της *WSDL2Java_Converter* και της *DII_Producer*. Η κλάση *WSDL2Java_Converter* δημιουργεί τον σκελετό της κλάσης υλοποίησης της υπηρεσίας *Adapter* ενώ η κλάση *DII_Producer* δημιουργεί το σώμα της κλάσης υλοποίησης της υπηρεσίας *Adapter*.

2. Δημιουργεί την ιεραρχία των φακέλων και των αρχείων για την τεχνητή υπηρεσία *Adapter* (σχήμα 4.20). Γι' αυτόν τον σκοπό χρησιμοποιεί και πάλι αντικείμενα της κλάσης *FILES*.



Σχήμα 4.25 Διάγραμμα κλάσεων για το υποσύστημα Substitution Manager

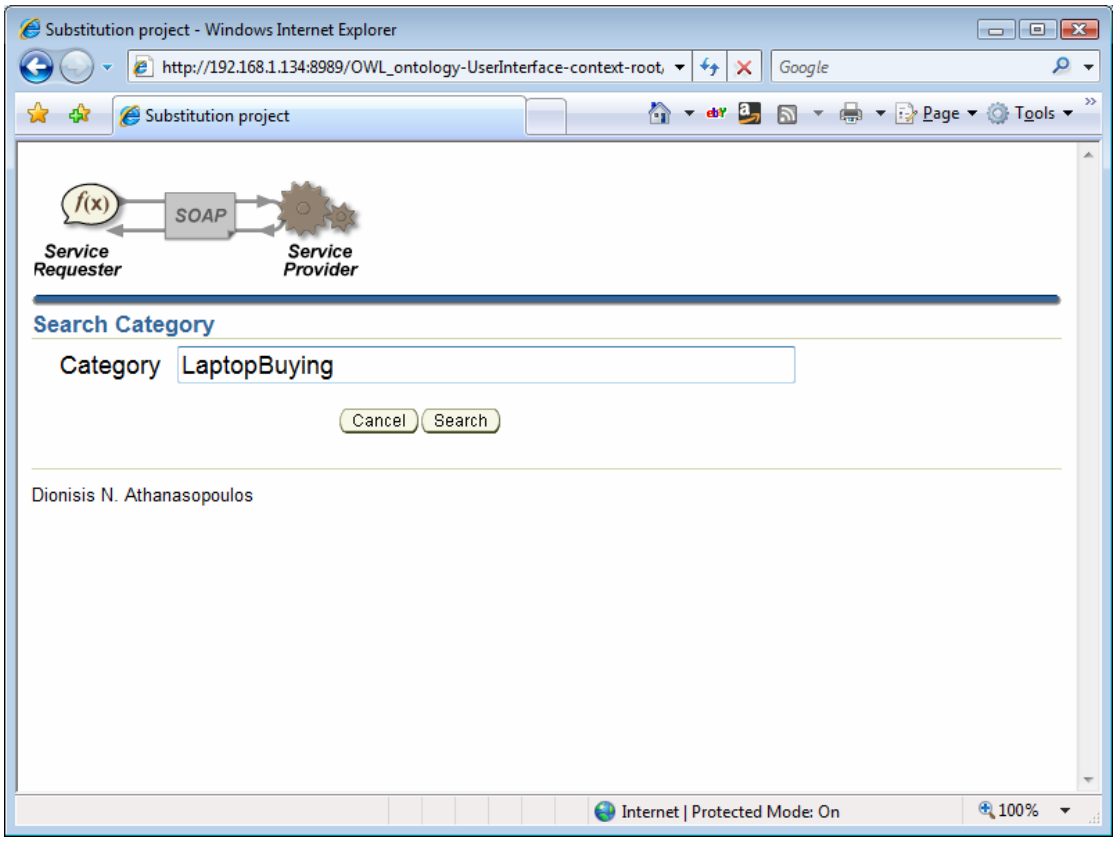
4.4. Η διεπαφή του συστήματος

4.4.1. Η διεπαφή του υποσυστήματος OWL-S Category Manager

Σε αυτήν την ενότητα δίνεται η διεπαφή του υποσυστήματος *OWL-S Category Manager* με βάση τα σενάρια εκτέλεσής του (βλ. ενότητα 4.2.1.1):

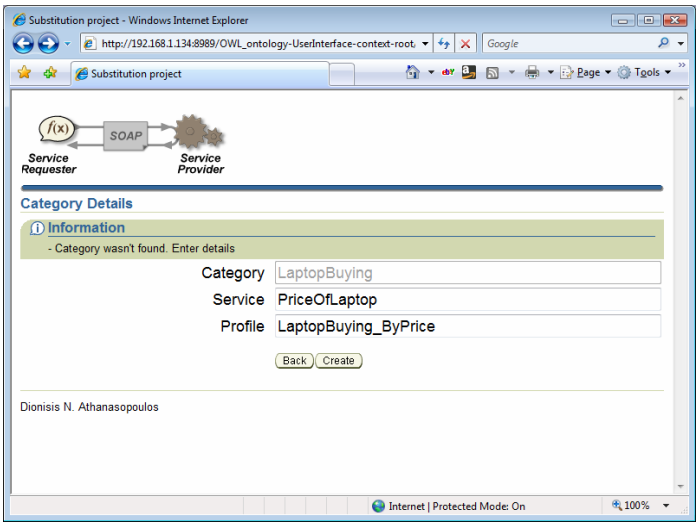
1. Δημιουργία νέας κατηγορίας από προφίλ.

   Για να δημιουργηθεί μία νέα κατηγορία υπηρεσιών, θα πρέπει να δοθεί το όνομα της κατηγορίας (σχήμα 4.26).



Σχήμα 4.26 Καταχώρηση του ονόματος της νέας κατηγορίας

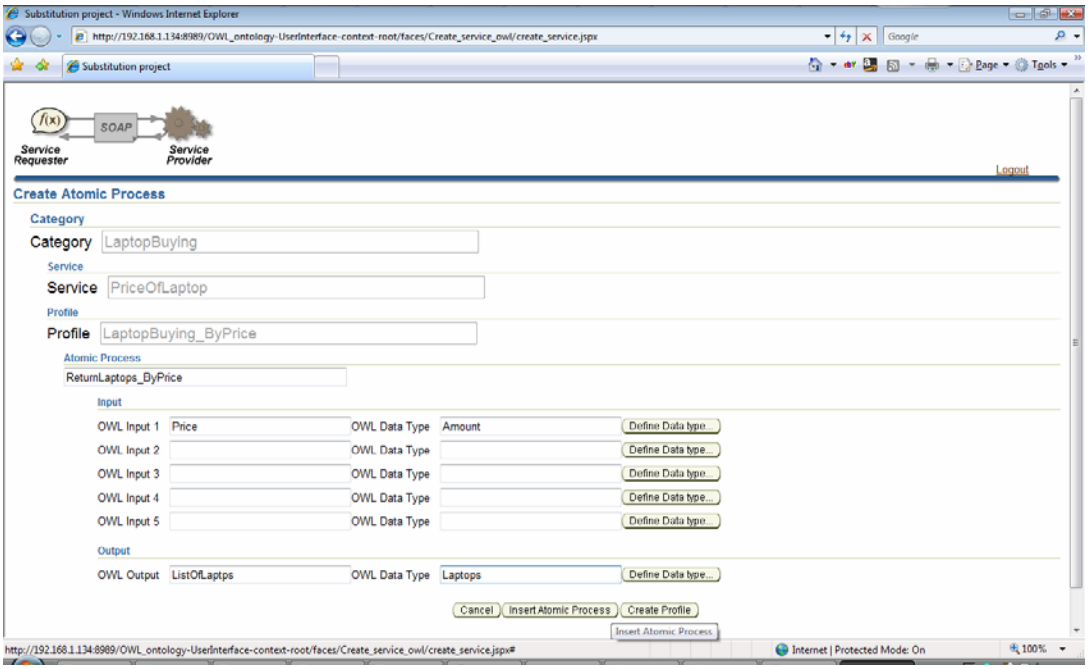
Το σύστημα πριν δημιουργήσει την κατηγορία, ελέγχει εάν υπάρχει, ώστε να αποφευχθεί η καταχώρηση κατηγορίας με ίδιο όνομα (διπλότυπο). Σε κάθε περίπτωση, το σύστημα οδηγείται στο δεύτερο σενάριο εκτέλεσης για να εισαχθεί ένα νέο προφίλ στην κατηγορία (σχήμα 4.27).



Σχήμα 4.27 Έλεγχος ύπαρξης κατηγορίας και καταχώρηση ενός νέου προφίλ

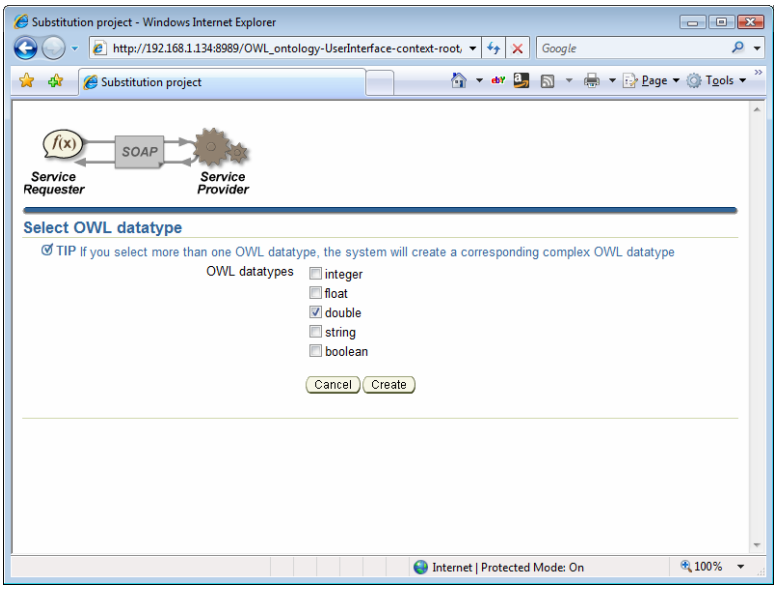
2. Προσθήκη ενός νέου προφίλ σε μία κατηγορία.

   Για να δημιουργηθεί ένα νέο προφίλ, θα πρέπει να εισαχθεί το όνομα μίας τουλάχιστον ατομικής διεργασίας του, το όνομα των δεδομένων εισόδου της και το όνομα των δεδομένων εξόδου της (σχήμα 4.28).



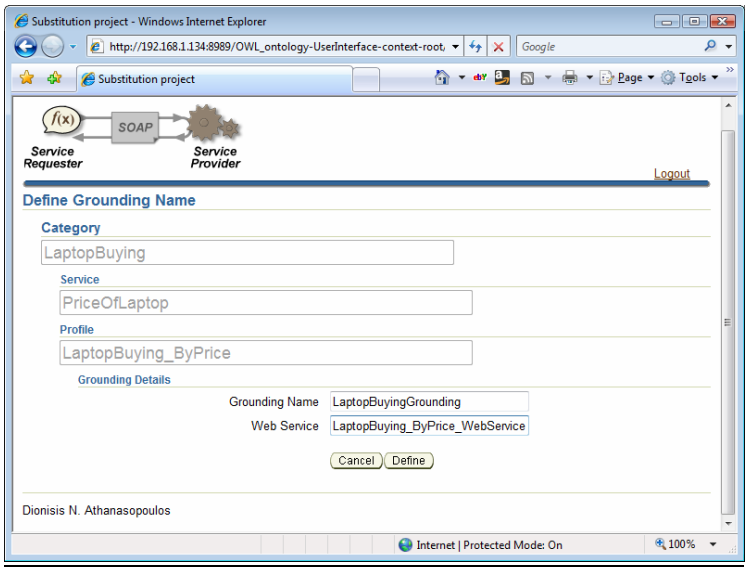
Σχήμα 4.28 Καταχώρηση των στοιχείων μίας ατομικής διεργασίας

Επιπλέον, στην διεπαφή του σχήματος 4.28 υπάρχει η επιλογή *Define Data type* με την βοήθεια της οποίας οδηγούμαστε στην διεπαφή του σχήματος 4.29 όπου μπορεί να οριστεί ο τύπος κάθε δεδομένο εισόδου/εξόδου.



Σχήμα 4.29 Ορισμός του τύπου ενός δεδομένου εισόδου/εξόδου

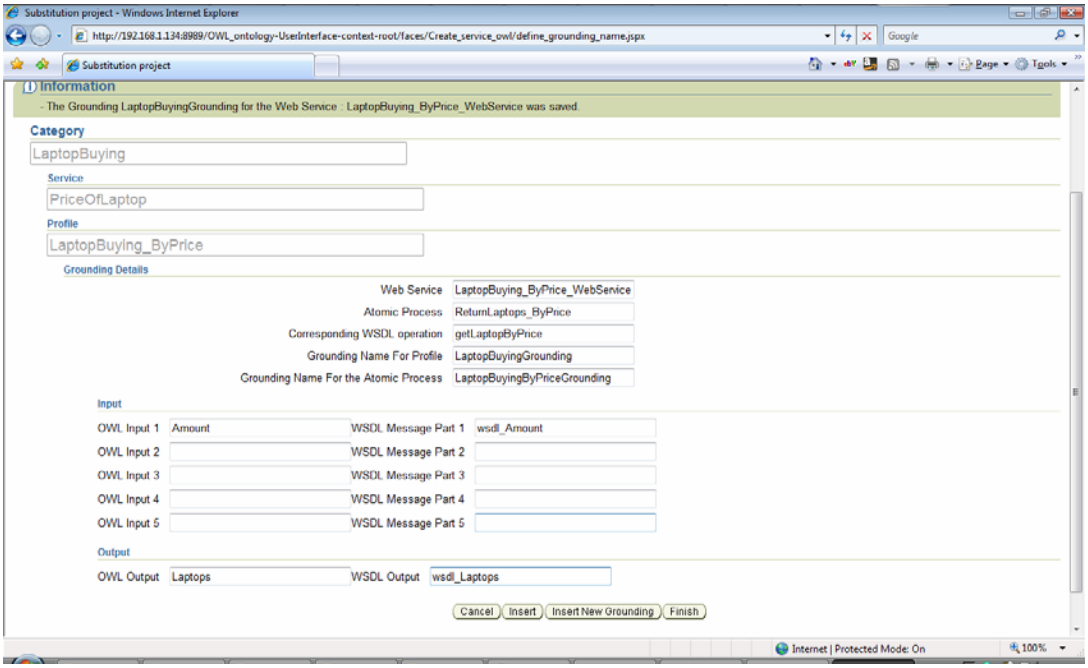
Αφού εισαχθούν όλες οι παραπάνω πληροφορίες στην διεπαφή του σχήματος 4.28, με την επιλογή *Insert Atomic Process* καταχωρείται η νέα ατομική διεργασία στο τρέχων προφίλ. Η παραπάνω διαδικασία μπορεί να επαναληφθεί ώστε να καταχωρηθούν και επιπλέον ατομικές διεργασίες στο τρέχων προφίλ. Με την επιλογή *Create Profile* της ίδιας διεπαφής τερματίζει η διαδικασία δημιουργίας ατομικών διεργασιών και το σύστημα οδηγείται στο τρίτο σενάριο εκτέλεσης (σχήμα 4.30).



Σχήμα 4.30 Καταχώρηση του ονόματος της αντιστοίχησης ενός προφίλ σε μία διεπαφή

3. <u>Προσθήκη αντιστοίχησης ενός προφίλ σε διεπαφή</u>.

   Για να δημιουργηθεί μία νέα αντιστοίχηση ενός προφίλ σε μία διεπαφή, θα πρέπει να εισαχθούν το όνομα της μεθόδου της διεπαφής με την οποία θα αντιστοιχηθεί μία ατομική διεργασία του προφίλ, καθώς και το όνομα κάθε δεδομένου εισόδου/εξόδου της μεθόδου με το οποίο θα αντιστοιχηθεί κάθε δεδομένου εισόδου/εξόδου της ατομικής διεργασίας (σχήμα 4.31).



Σχήμα 4.31 Καταχώρηση των στοιχείων της αντιστοίχησης ενός προφίλ σε μία διεπαφή

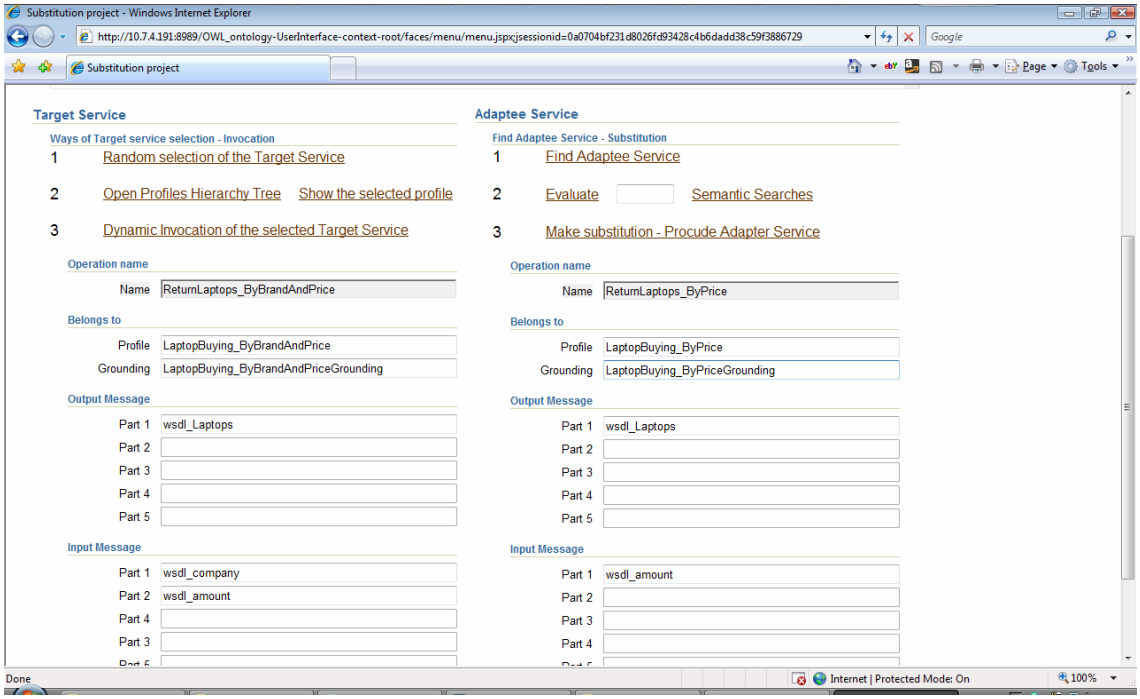
Αφού εισαχθούν όλες οι παραπάνω πληροφορίες στην διεπαφή του σχήματος 4.31, με την επιλογή *Insert* καταχωρείται η αντιστοίχηση στο τρέχων προφίλ. Με την επιλογή *Insert New Grounding* του σχήματος 4.31, η παραπάνω διαδικασία μπορεί να επαναληφθεί ώστε να καταχωρηθούν και επιπλέον αντιστοιχίσεις του τρέχοντος προφίλ σε διεπαφές. Με την επιλογή *Finish* τερματίζει η διαδικασία δημιουργίας αντιστοιχήσεων του τρέχοντος προφίλ σε διεπαφές.

4.4.2. Η διεπαφή του υποσυστήματος Substitution Manager

Σε αυτήν την ενότητα δίνεται η διεπαφή του υποσυστήματος *Substitution Manager* (βλ. ενότητα 4.2.2.1). Για να δημιουργηθεί η τεχνητή υπηρεσία *Adapter*, θα πρέπει να επιλεχθεί η υπηρεσία *Target* την οποία θα αντικαταστήσει. Η υπηρεσία *Target* μπορεί είτε να επιλεχθεί τυχαία από το σύστημα είτε από τον ίδιο τον χρήστη. Σε κάθε περίπτωση εμφανίζονται στην οθόνη οι λεπτομέρειες για μία μέθοδο της υπηρεσίας *Target*.

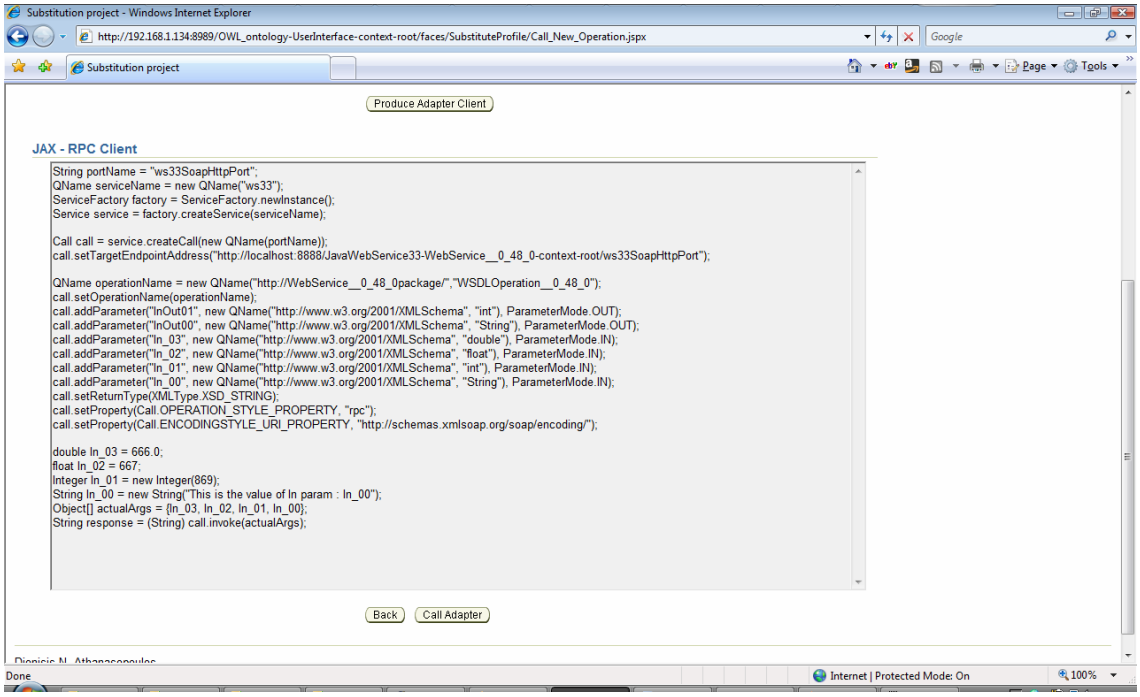
Παράδειγμα:

Θεωρούμε ότι έχει επιλεγεί μία υπηρεσία *Target* η οποία έχει το προφίλ *LaptopBuying_ByBrandAndPrice*. Στην συνέχεια, με την επιλογή *Find Adaptee Service* του σχήματος 4.32, το σύστημα κάνει συντακτική και σημασιολογική αναζήτηση της υπηρεσίας *Adaptee* η οποία θα αντικαταστήσει την υπηρεσία *Target* και εμφανίζει τις λεπτομέρειες της στην οθόνη (σχήμα 4.32). Η υπηρεσία *Adaptee* έχει το προφίλ *LaptopBuying_ByPrice*.



Σχήμα 4.32 Οι λεπτομέρειες για μία μέθοδο της υπηρεσίας Target και για την υποψήφια προς αντικατάστασής της μέθοδο της υπηρεσίας Adaptee

Μετά την εύρεση της υπηρεσίας *Adaptee*, κατασκευάζεται δυναμικά από το σύστημα η υπηρεσία *Adapter* καθώς και ο κώδικας για την JAX-RPC κλήση της. Το σχήμα 4.33 απεικονίζει τον δυναμικά παραγόμενο κώδικα.



Σχήμα 4.33 Ο δυναμικά παραγόμενος κώδικας για την JAX-RPC κλήση της υπηρεσίας Adapter

ΚΕΦΑΛΑΙΟ 5. ΕΛΕΓΧΟΣ ΚΑΙ ΜΕΤΡΗΣΕΙΣ

5.1. Μεθοδολογία ελέγχου

Οι πειραματικές μετρήσεις που πραγματοποιήθηκαν επικεντρώνονται στο χρόνο εκτέλεσης των βασικών αλγορίθμων τους οποίους που χρησιμοποιεί το σύστημα, καθώς και στο επιπλέον κόστος που εισάγει το Προσαρμοστικό Σχεδιαστικό Πρότυπο στην χρήση υπηρεσιών.

Οι αλγόριθμοι που χρησιμοποιεί το σύστημα εφαρμόζονται επί της δενδρικής δομής *ProfilesHierarchy* (βλ. ενότητα 4.2.2.1). Για να πραγματοποιηθούν οι μετρήσεις, κατασκευάζεται η δομή *ProfilesHierarchy* χρησιμοποιώντας τον αλγόριθμο *ProfilesHierarchyGeneration* (σχήμα 5.1). Αυτός ο αλγόριθμος παράγει προφίλ και τα εισάγει στη δενδρική δομή *ProfilesHierarchy*. Για την εισαγωγή τους χρησιμοποιούνται οι προαναφερθέντες αλγόριθμοι *CheckSubstitution* και *ProfilesHierarchyUpdate*. Πιο συγκεκριμένα, ο αλγόριθμος *ProfilesHierarchyGeneration*:

1. δέχεται ως είσοδο την δενδρική δομή *DataTypesHierarchy* (βλ. ενότητα 3.2.2) και τις παραμέτρους του πειράματος, δηλαδή το πλήθος των προφίλ, το πλήθος των ατομικών διεργασιών τους και το πλήθος των δεδομένων εισόδου/εξόδου τους και επιστρέφει
2. τη δομή *ProfilesHierarchy*, η οποία θα περιέχει τα παραγόμενα προφίλ, και
3. τη δομή *CategoryRepository,* η οποία θα περιέχει τις λεπτομέρειες για τα προφίλ.

Ο αλγόριθμος έχει τα ακόλουθα χαρακτηριστικά:

1. Αρχικά, παράγει έναν δοσμένο αριθμό από προφίλ πρώτου επιπέδου και χρησιμοποιεί τον αλγόριθμο *ProfilesHierarchyUpdate* ώστε να τα εισάγει στην αρχικά άδεια δομή *ProfilesHierarchy*. Ως προφίλ πρώτου επιπέδου ορίζονται εκείνα που μετά την εκτέλεση του αλγορίθμου, θα εισαχθούν, με μεγάλη πιθανότητα, στο πρώτο επίπεδο της δενδρικής δομής *ProfilesHierarchy*. Για να επιτευχθεί αυτό, ο τύπος των δεδομένων εισόδου τους επιλέγεται με τυχαίο τρόπο από το πρώτο επίπεδο της δενδρικής δομής *DataTypesHierarchy* ενώ ο τύπος των δεδομένων εξόδου τους επιλέγεται με τυχαίο τρόπο από το τελευταίο επίπεδο της δενδρικής δομής *DataTypesHierarchy*. Με αυτήν επιλογή εξασφαλίζεται ότι τα προφίλ πρώτου επιπέδου θα είναι πιο γενικά από τα προφίλ των επόμενων επιπέδων. Επομένως, τα προφίλ των επόμενων επιπέδων θα αποτελούν σημασιολογική επέκταση ή εξειδίκευση των προφίλ του πρώτου επιπέδου.
2. Παράγει τα προφίλ των επόμενων επιπέδων με αντίστοιχο τρόπο μέχρι να καλυφθεί ο δοσμένος αριθμός προφίλ του εκάστοτε πειράματος. Το πλήθος των προφίλ κάθε επιπέδου θα είναι διπλάσιο από αυτό του προηγούμενου επιπέδου. Το πλήθος των επιπέδων (το βάθος της δομής *ProfilesHierarchy*), το οποίο θα προκύψει από την εκτέλεση του αλγορίθμου, εξαρτάται τόσο από το βάθος της δομής *DataTypesHierarchy* όσο και από το συνολικό πλήθος των προφίλ που θα παραχθούν.

Οι μετρήσεις αφορούν τον υπολογισμό, για κάθε προφίλ, του μέσου χρόνου εκτέλεσης των αλγορίθμων *CheckSubstitution* και *ProfilesHierarchyUpdate*. Επαναλήφθηκαν αυτές οι μετρήσεις, με διάφορες εκτελέσεις του αλγορίθμου *ProfilesHierarchyGeneration*. Σε κάθε εκτέλεσή του, κατασκευάστηκε εκ νέου η δομή *ProfilesHierarchy*. Τα προφίλ που παράχθησαν:

1. είχαν διαφορετικό αριθμό ατομικών διεργασιών αλλά ίδιο αριθμό δεδομένων εισόδου/εξόδου σε σχέση με την προηγούμενες εκτελέσεις, ή
2. είχαν ίδιο αριθμό ατομικών διεργασιών αλλά διαφορετικό αριθμό δεδομένων εισόδου/εξόδου σε σχέση με την προηγούμενες εκτελέσεις.

Επιπλέον, έγιναν μετρήσεις, για τον υπολογισμό του μέσου χρόνου εκτέλεσης του αλγορίθμου *ServiceSearching*. Συγκεκριμένα, μετρήθηκε ο μέσος χρόνος της σημασιολογικής αναζήτησης ενός προφίλ στην δομή *ProfilesHierarchy*.

Τέλος, έγιναν μετρήσεις, για τον χρόνο που χρειάζεται ο κώδικας του πελάτη να εκτελεστεί για την περίπτωση που δεν έχει γίνει η αντικατάσταση της αρχικής υπηρεσίας *Target* από την υπηρεσία *Adapter* αλλά και για την περίπτωση που έχει γίνει η αντικατάσταση. Οι μετρήσεις έγιναν σε σχέση με το πλήθος των δεδομένων εισόδου της μεθόδου της αρχικής υπηρεσίας *Target*.

ProfilesHierarchyGeneration

Είσοδος: DataTypesHierarchy, NumProfiles, NumAtomicProcesses, NumInputs, NumOutputs
Έξοδος: ProfilesHierarchy, CategoryRepository

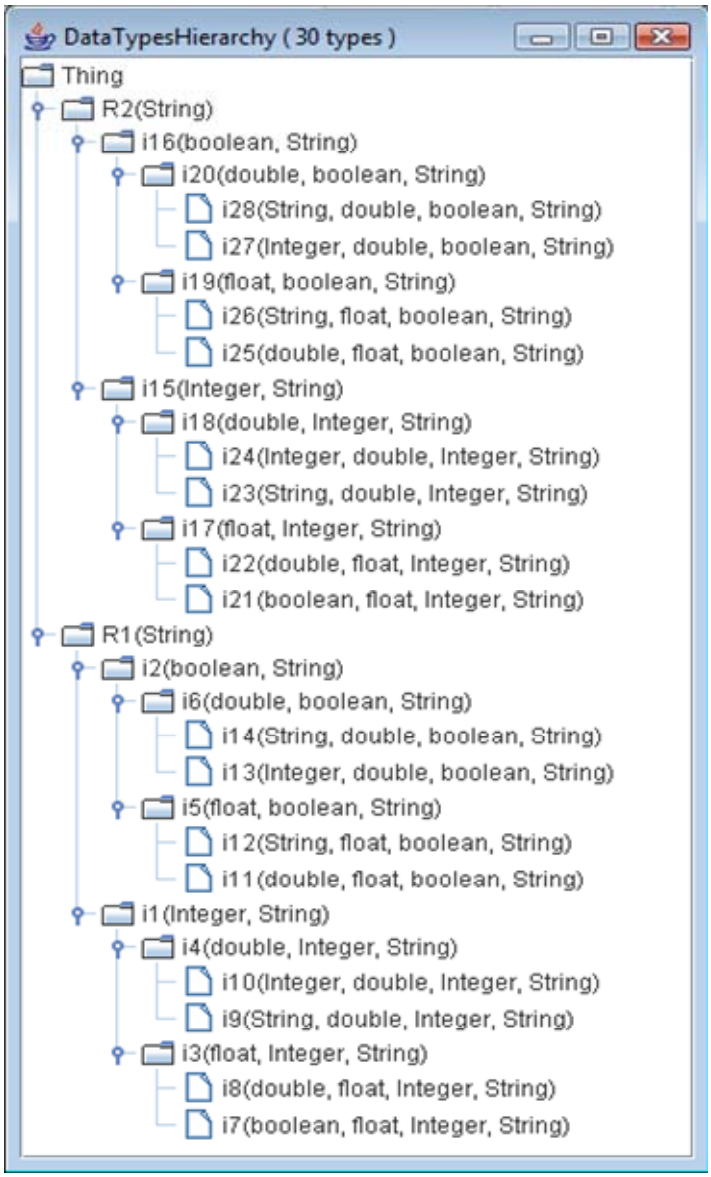
```
Algorithm ProfilesHierarchyGeneration
    For ( i from 1 to NumProfiles) do
        Generate a random String as name of current profile i
        For ( j from 1 to NumAtomicProcesses) do
            Generate a random String as name of current atomic process j
            For ( k from 1 to NumInputs) do
                Generate a random String as name of current input k
                If ( i >= 0 && i <= 1 ) Then
                    Type = a Random datatype from 1st Level of DataTypesHierarchy
                    Set Type as datatype of current input k
                Else If ( i >= 2 && i <= 9 ) Then
                    Type = a Random datatype from 2nd Level of DataTypesHierarchy
                    Set Type as datatype of current input k
                Else If ( i >= 10 && i <= 26 ) Then
                    Type = a Random datatype from 3rd Level of DataTypesHierarchy
                    Set Type as datatype of current input k
                Else If ( i >= 27 && i <= 59 ) Then
                    Type = a Random datatype from 4th Level of DataTypesHierarchy
                    Set Type as datatype of current input k
                Else
                    Type = a Random datatype from 4th Level of DataTypesHierarchy
                    Set Type as datatype of current input k
                End If
            End For
            For ( k from 1 to NumOutputs) do
                Generate a random String as name of current output k
                If ( i >= 0 && i <= 1 ) Then
                    Type = a Random datatype from 4th Level of DataTypesHierarchy
                    Set Type as datatype of current output k
                Else If ( i >= 2 && i <= 9 ) Then
                    Type = a Random datatype from 3rd Level of DataTypesHierarchy
                    Set Type as datatype of current output k
                Else If ( i >= 10 && i <= 26 ) Then
                    Type = a Random datatype from 2nd Level of DataTypesHierarchy
                    Set Type as datatype of current output k
                Else If ( i >= 27 && i <= 59 ) Then
                    Type = a Random datatype from 1st Level of DataTypesHierarchy
                    Set Type as datatype of current output k
                Else
                    Type = a Random datatype from 1st Level of DataTypesHierarchy
                    Set Type as datatype of current output k
                End If
            End For
        End For
        Insert current profile i into Category Repository
        ProfilesHierarchyUpdate(NewProfile, DataTypesHierarchy, ProfilesHierarchy)
    End For
End Algorithm
```

Σχήμα 5.1 Ο αλγόριθμος ProfilesHierarchyGeneration

5.2. Αναλυτική παρουσίαση ελέγχου

5.2.1. Μετρήσεις για τον αλγόριθμο CheckSubstitution

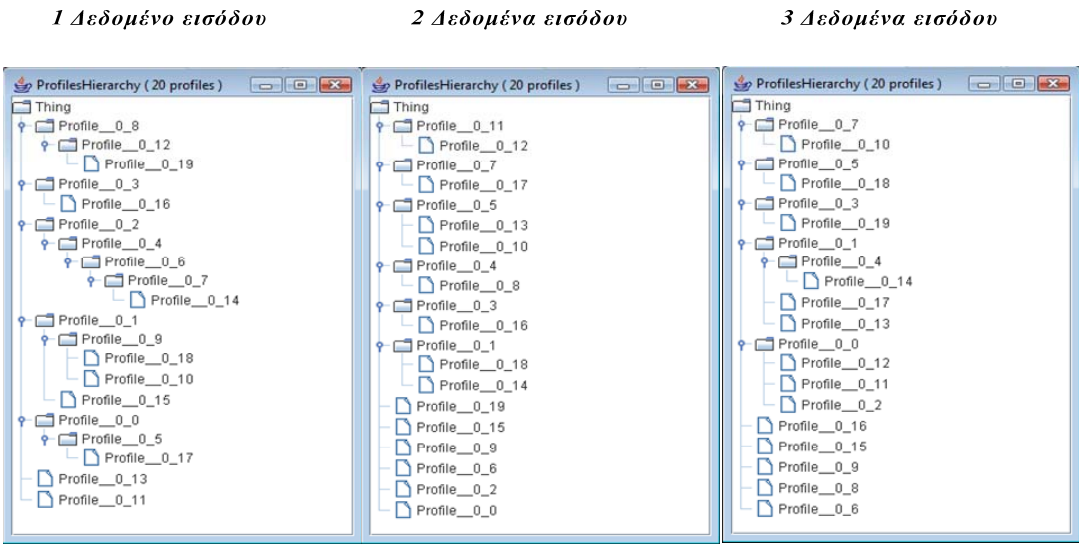
Έστω ότι ο αλγόριθμος *ProfilesHierarchyGeneration* δέχεται ως είσοδο την δενδρική δομή *DataTypesHierarchy* του σχήματος 5.2, η οποία έχει βάθος τέσσερα. Ο αλγόριθμος *CheckSubstitution* ελέγχει εάν ένα προφίλ είναι σημασιολογική επέκταση κάποιου άλλου προφίλ.



Σχήμα 5.2 Η δομή DataTypesHierarchy με βάθος τέσσερα.

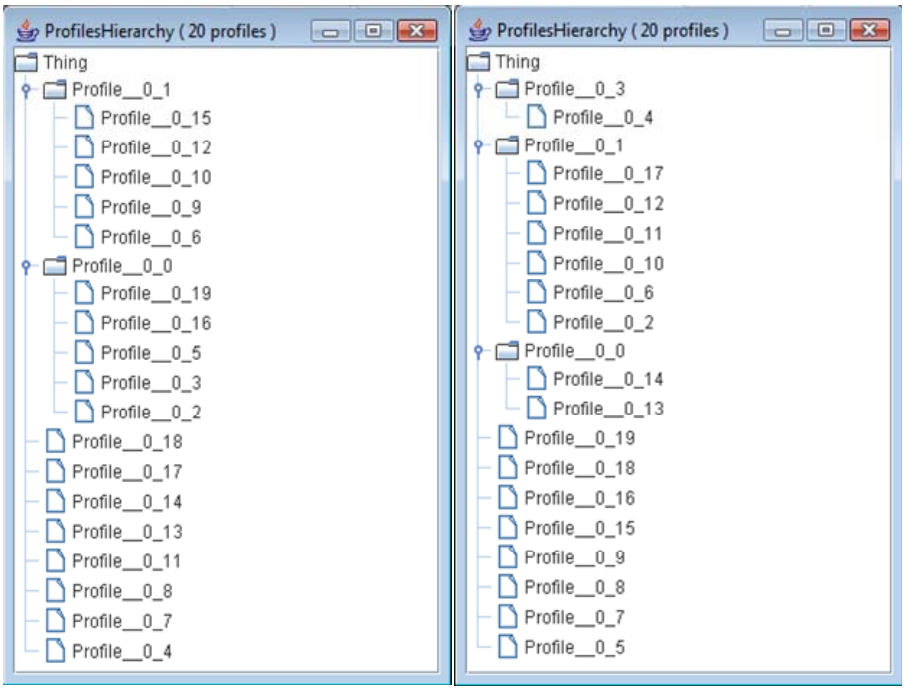
5.2.1.1. Πρώτη μέτρηση του αλγορίθμου CheckSubstitution

Για την πρώτη μέτρηση έγιναν πέντε εκτελέσεις του αλγορίθμου *ProfilesHierarchyGeneration*. Σε κάθε μία από αυτές τις εκτελέσεις, παράγονται 20 προφίλ, που το καθένα έχει μόνο μία ατομική διεργασία. Σε κάθε εκτέλεση μεταβάλλεται το πλήθος των δεδομένων εισόδου της ατομικής διεργασίας. Στα δύο επόμενα σχήματα απεικονίζεται η δομή *ProfilesHierarchy* που παράγεται σε κάθε μία από αυτές τις εκτελέσεις. Παρατηρούμε ότι όσο αυξάνεται το πλήθος των δεδομένων εισόδου, μειώνεται και το βάθος του δένδρου. Αυτό γίνεται γιατί είναι πιο δύσκολο να βρεθεί σημασιολογική σχέση (επέκταση/εξειδίκευση) ανάμεσα σε προφίλ με πολλά δεδομένα εισόδου.



Σχήμα 5.3 Η δομή ProfilesHierarchy για 20 παραγόμενα προφίλ από διαφορετικές εκτελέσεις του αλγορίθμου ProfilesHierarchyGeneration

4 Δεδομένα εισόδου ***5 Δεδομένα εισόδου***



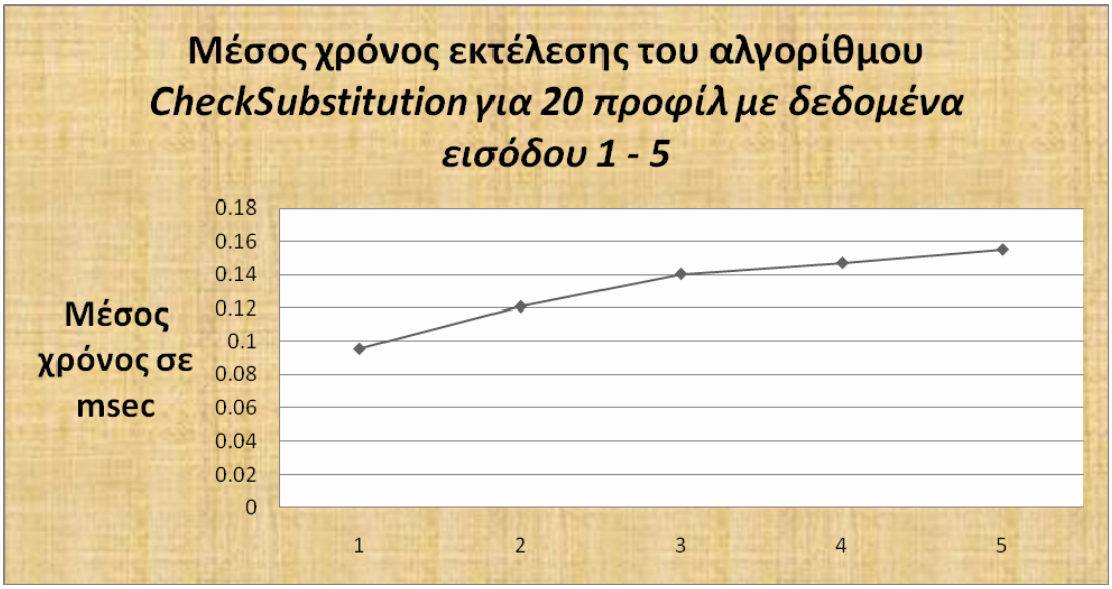
Σχήμα 5.4 Η δομή ProfilesHierarchy για 20 παραγόμενα προφίλ από διαφορετικές εκτελέσεις του αλγορίθμου ProfilesHierarchyGeneration

Για κάθε μία από τις παραπάνω εκτελέσεις, μετρήθηκε συνολικά και για τα 20 προφίλ, ο μέσος χρόνος εκτέλεσης του αλγορίθμου *CheckSubstitution* (σχήμα 5.5) και το μέσο πλήθος των εκτελέσεων του αλγορίθμου *CheckSubstitution*, δηλαδή το μέσο πλήθος των συγκρίσεων που εκτελούνται και για τα 20 προφίλ (σχήμα 5.6).

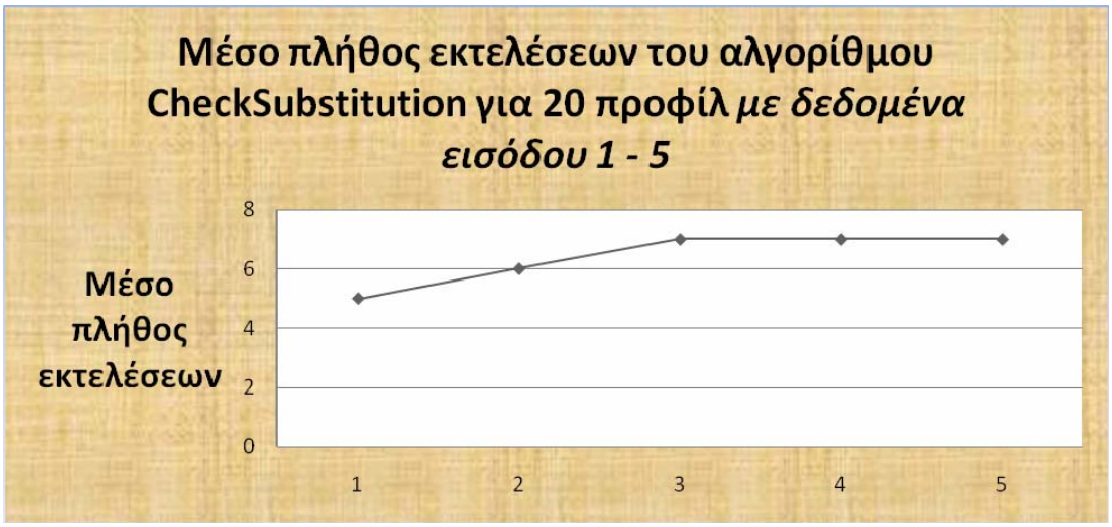
Συμπεράσματα:

Από το σχήμα 5.5 συμπεραίνουμε ότι ο χρόνος εκτέλεσης του αλγορίθμου *CheckSubstitution* και το πλήθος των δεδομένων εισόδου του κάθε προφίλ είναι ανάλογα ποσά. Από το σχήμα 5.6, παρατηρούμε ότι το μέσο πλήθος των εκτελέσεων του αλγορίθμου *CheckSubstitution* παρουσιάζει μία μικρή αύξηση καθώς αυξάνεται το πλήθος των δεδομένων εισόδου, αλλά για αριθμό δεδομένων εισόδου μεγαλύτερο από 3 έχει σταθερή τιμή. Αυτό συμβαίνει γιατί το πλήθος των εκτελέσεων του αλγορίθμου δεν εξαρτάται μόνο από το πλήθος των δεδομένων εισόδου αλλά από το βάθος της δομής *ProfilesHierarchy*. Όπως φαίνεται από το σχήμα 5.4, η μορφή που

έχει η δομή για 4 και 5 δεδομένα εισόδου είναι σχεδόν ίδια. Επομένως, και το πλήθος των εκτελέσεων του αλγορίθμου θα είναι σχεδόν ίδιο.



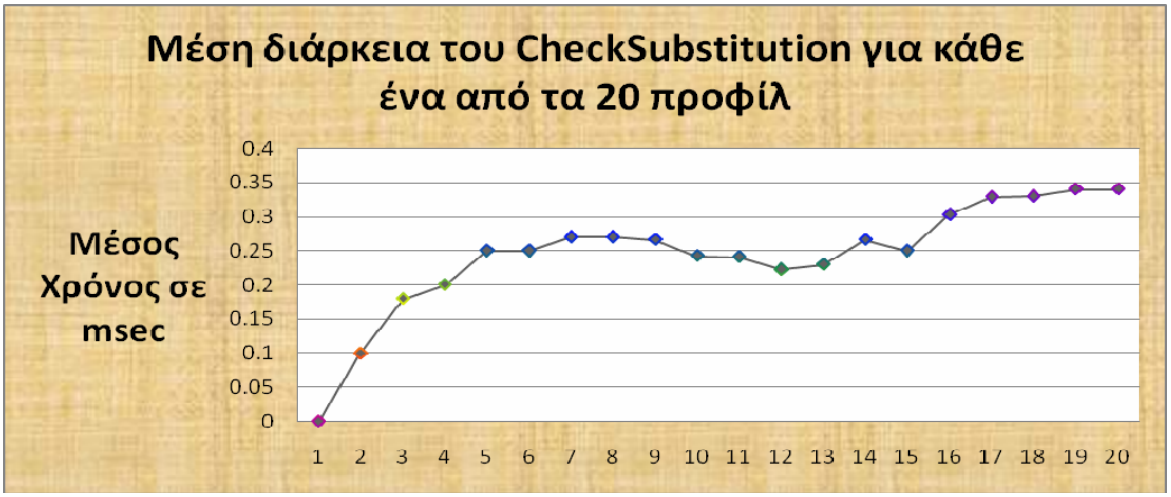
Σχήμα 5.5 Μέσος χρόνος εκτέλεσης του CheckSubstitution για τα 20 προφίλ συνολικά



Σχήμα 5.6 Μέσο πλήθος εκτελέσεων του CheckSubstitution για τα 20 προφίλ συνολικά

Τα επόμενα σχήματα αναφέρονται σε μετρήσεις που έγιναν στα προφίλ της πρώτης δομής *ProfilesHierarchy* του σχήματος 5.3, τα οποία έχουν μόνο ένα δεδομένο εισόδου. Πιο αναλυτικά, το σχήμα 5.7 απεικονίζει το ακριβές πλήθος των εκτελέσεων του αλγορίθμου *CheckSubstitution* (πλήθος συγκρίσεων) οι οποίες απαιτούνται για να εισαχθεί στην δομή κάθε ένα από αυτά τα 20 προφίλ. Το σχήμα 5.8 απεικονίζει το μέσο χρόνο εκτέλεσης του αλγορίθμου *CheckSubstitution* που απαιτείται για να εισαχθεί στην δομή κάθε ένα από αυτά τα 20 προφίλ.

Σχήμα 5.7 Το ακριβές πλήθος εκτελέσεων του CheckSubstitution για κάθε ένα από τα 20 προφίλ

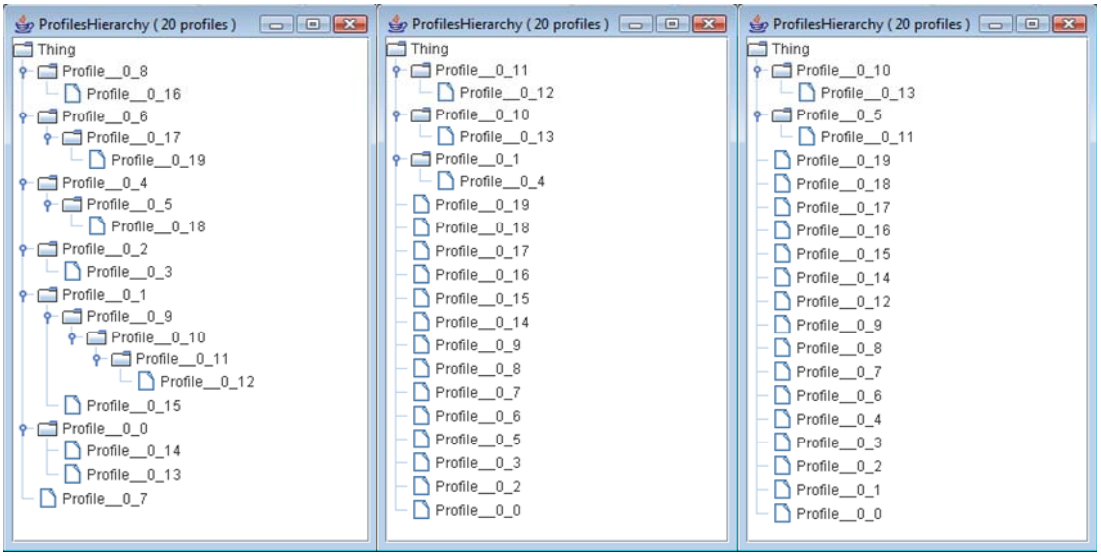


Σχήμα 5.8 Μέσος χρόνος εκτέλεσης του CheckSubstitution για κάθε ένα από τα 20 προφίλ

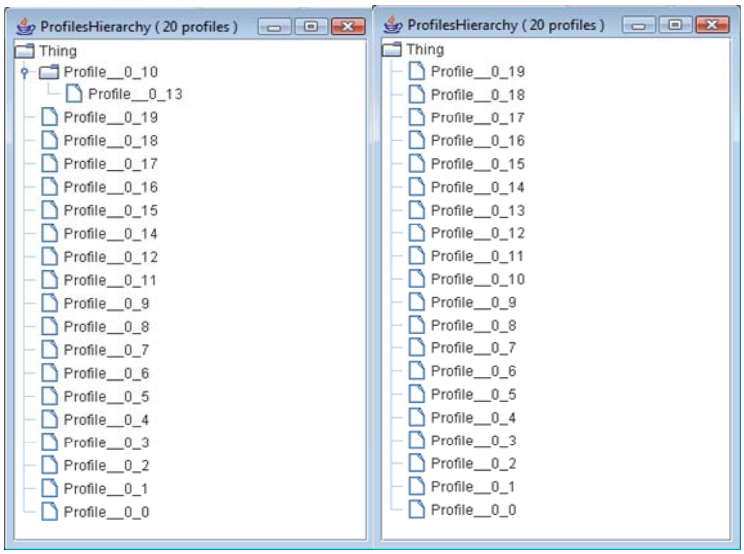
5.2.1.2. Δεύτερη μέτρηση του αλγορίθμου CheckSubstitution

Για την δεύτερη μέτρηση έγιναν πέντε εκτελέσεις του αλγορίθμου *ProfilesHierarchyGeneration*. Σε κάθε μία από αυτές τις εκτελέσεις, παράγονται 20 προφίλ, που το καθένα έχει μόνο μία ατομική διεργασία. Σε κάθε εκτέλεση μεταβάλλεται το πλήθος των δεδομένων εξόδου της ατομικής διεργασίας. Στα δύο επόμενα σχήματα απεικονίζεται η δομή *ProfilesHierarchy* που παράγεται σε κάθε μία από αυτές τις εκτελέσεις. Παρατηρούμε ότι όσο αυξάνεται το πλήθος των δεδομένων εξόδου, μειώνεται και το βάθος του δένδρου. Αυτό γίνεται γιατί είναι πιο δύσκολο να βρεθεί σημασιολογική σχέση (επέκταση/εξειδίκευση) ανάμεσα σε προφίλ με πολλά δεδομένα εξόδου.

1 Δεδομένο εξόδου *2 Δεδομένα εξόδου* *3 Δεδομένα εξόδου*



4 Δεδομένα εξόδου *5 Δεδομένα εξόδου*

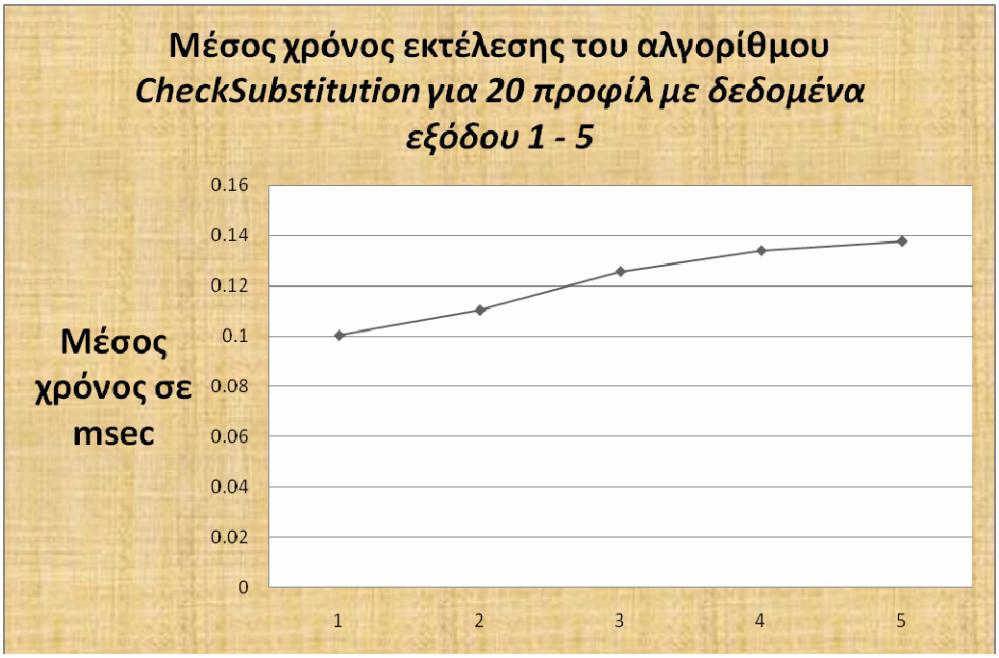


Σχήμα 5.9 Η δομή ProfilesHierarchy για 20 παραγόμενα προφίλ από διαφορετικές εκτελέσεις του αλγορίθμου ProfilesHierarchyGeneration

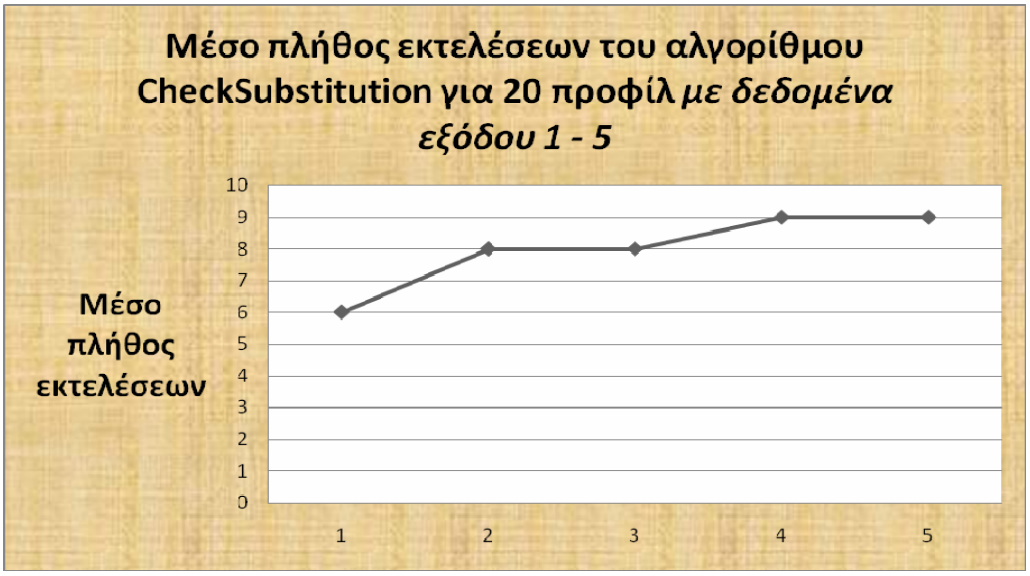
Για κάθε μία από τις παραπάνω εκτελέσεις, μετρήθηκε συνολικά και για τα 20 προφίλ, ο μέσος χρόνος εκτέλεσης του αλγορίθμου *CheckSubstitution* (σχήμα 5.10) και το μέσο πλήθος των εκτελέσεων του αλγορίθμου *CheckSubstitution*, δηλαδή το μέσο πλήθος των συγκρίσεων που εκτελούνται και για τα 20 προφίλ (σχήμα 5.11).

Συμπεράσματα:

Από το σχήμα 5.10 συμπεραίνουμε ότι ο χρόνος εκτέλεσης του αλγορίθμου *CheckSubstitution* και το πλήθος των δεδομένων εξόδου του κάθε προφίλ είναι ανάλογα ποσά. Από το σχήμα 5.11, παρατηρούμε ότι το μέσο πλήθος των εκτελέσεων του αλγορίθμου *CheckSubstitution* παρουσιάζει μία μικρή αύξηση καθώς αυξάνεται το πλήθος των δεδομένων εξόδου, αλλά για αριθμό δεδομένων εξόδου μεγαλύτερο από 3 έχει σταθερή τιμή. Αυτό συμβαίνει γιατί το πλήθος των εκτελέσεων του αλγορίθμου δεν εξαρτάται μόνο από το πλήθος των δεδομένων εξόδου αλλά από το βάθος της δομής *ProfilesHierarchy*. Όπως φαίνεται από το σχήμα 5.9, η μορφή που έχει η δομή για 4 και 5 δεδομένα εξόδου είναι σχεδόν ίδια. Επομένως, και το πλήθος των εκτελέσεων του αλγορίθμου θα είναι σχεδόν ίδιο.



Σχήμα 5.10 Μέσος χρόνος εκτέλεσης του CheckSubstitution για τα 20 προφίλ συνολικά

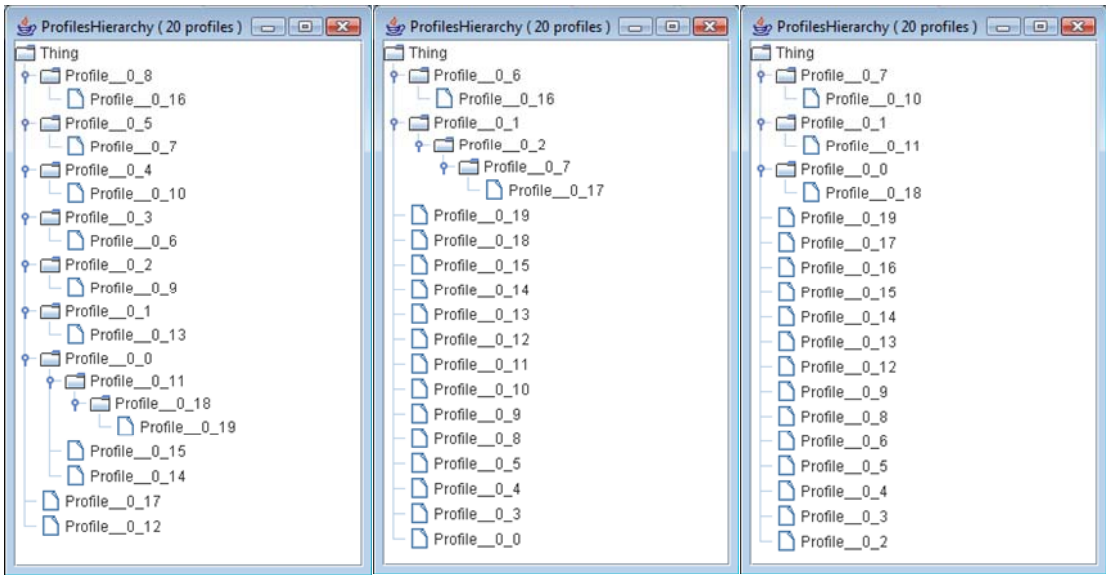


Σχήμα 5.11 Μέσο πλήθος εκτελέσεων του CheckSubstitution για τα 20 προφίλ συνολικά

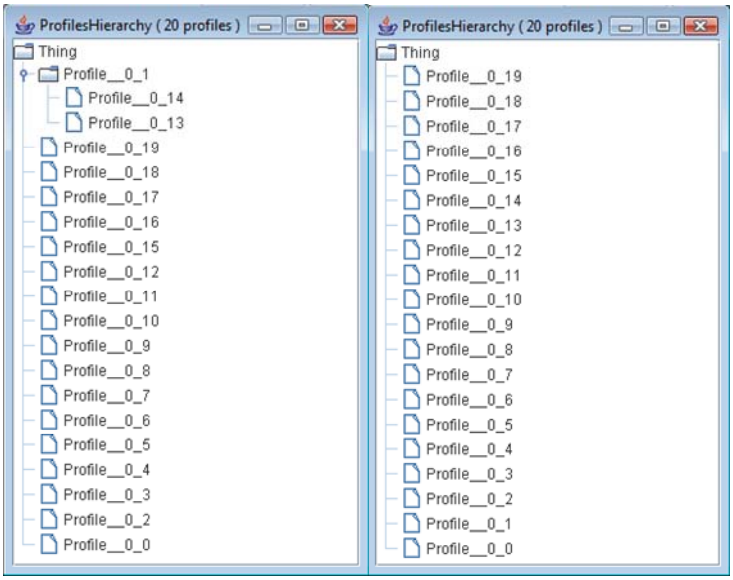
5.2.1.3. Τρίτη μέτρηση του αλγορίθμου CheckSubstitution

Για την τρίτη μέτρηση έγιναν πέντε εκτελέσεις του αλγορίθμου *ProfilesHierarchyGeneration*. Σε κάθε μία από αυτές τις εκτελέσεις, παράγονται 20 προφίλ, που το καθένα έχει μόνο ένα δεδομένο εισόδου και ένα δεδομένο εξόδου. Σε κάθε εκτέλεση μεταβάλλεται το πλήθος των ατομικών διεργασιών των προφίλ. Στα δύο επόμενα σχήματα απεικονίζεται η δομή *ProfilesHierarchy* που παράγεται σε κάθε μία από αυτές τις εκτελέσεις. Παρατηρούμε ότι όσο αυξάνεται το πλήθος των ατομικών διεργασιών, μειώνεται και το βάθος του δένδρου. Αυτό γίνεται γιατί είναι πιο δύσκολο να βρεθεί σημασιολογική σχέση (επέκταση/εξειδίκευση) ανάμεσα σε προφίλ με πολλές ατομικές διεργασίες.

1 Ατομική διεργασία *2 Ατομικές διεργασίες* *3 Ατομικές διεργασίες*



4 Ατομικές διεργασίες *5 Ατομικές διεργασίες*

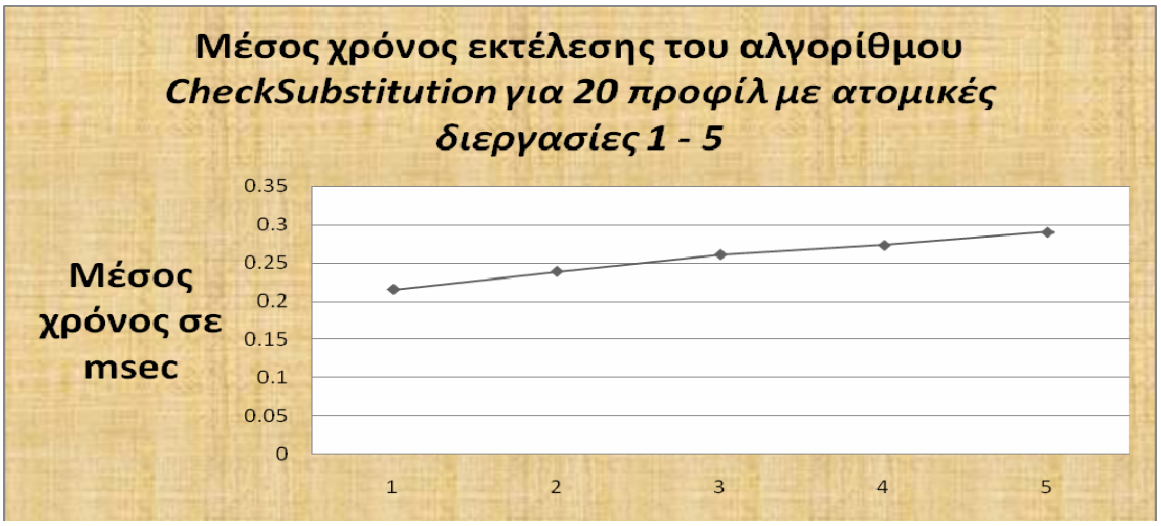


Σχήμα 5.12 Η δομή ProfilesHierarchy για 20 παραγόμενα προφίλ από διαφορετικές εκτελέσεις του αλγορίθμου ProfilesHierarchyGeneration

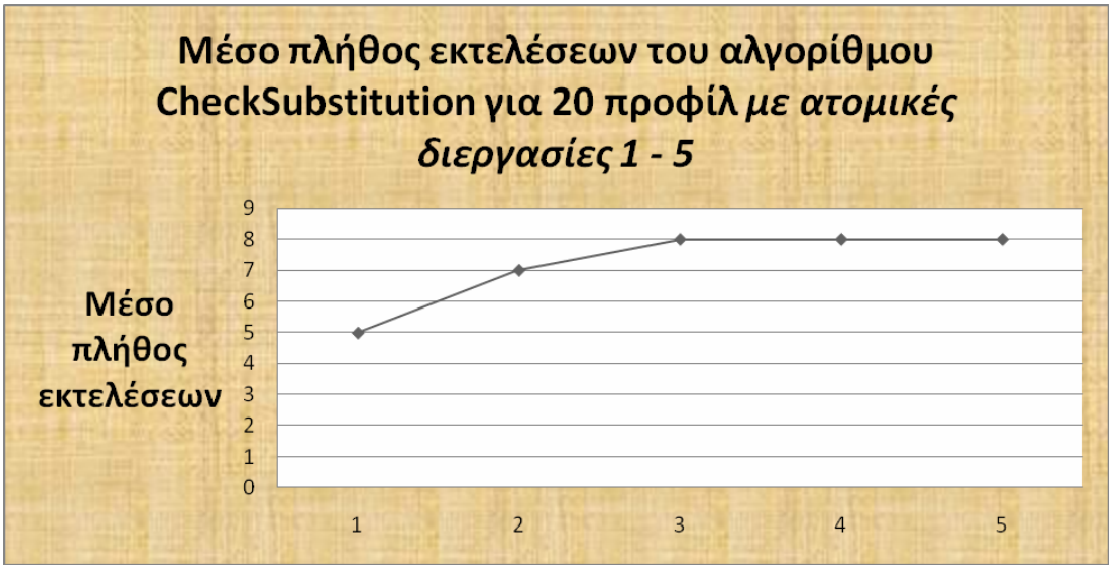
Για κάθε μία από τις παραπάνω εκτελέσεις, μετρήθηκε συνολικά και για τα 20 προφίλ, ο μέσος χρόνος εκτέλεσης του αλγορίθμου *CheckSubstitution* (σχήμα 5.13) και το μέσο πλήθος των εκτελέσεων του αλγορίθμου *CheckSubstitution*, δηλαδή το μέσο πλήθος των συγκρίσεων που εκτελούνται και για τα 20 προφίλ (σχήμα 5.14).

Συμπεράσματα:

Από το σχήμα 5.13 συμπεραίνουμε ότι ο χρόνος εκτέλεσης του αλγορίθμου *CheckSubstitution* και το πλήθος των ατομικών διεργασιών του κάθε προφίλ είναι ανάλογα ποσά. Από το σχήμα 5.14, παρατηρούμε ότι το μέσο πλήθος των εκτελέσεων του αλγορίθμου *CheckSubstitution* παρουσιάζει μία μικρή αύξηση καθώς αυξάνεται το πλήθος των ατομικών διεργασιών, αλλά για αριθμό ατομικών διεργασιών μεγαλύτερο από 3 έχει σταθερή τιμή. Αυτό συμβαίνει γιατί το πλήθος των εκτελέσεων του αλγορίθμου δεν εξαρτάται μόνο από το πλήθος των ατομικών διεργασιών αλλά και από το βάθος της δομής *ProfilesHierarchy*. Όπως φαίνεται από το σχήμα 5.14, η μορφή που έχει η δομή για 4 και 5 ατομικές διεργασίες είναι σχεδόν ίδια. Επομένως, και το πλήθος των εκτελέσεων του αλγορίθμου θα είναι σχεδόν ίδιο.



Σχήμα 5.13 Μέσος χρόνος εκτέλεσης του CheckSubstitution για τα 20 προφίλ συνολικά



Σχήμα 5.14 Μέσο πλήθος εκτελέσεων του CheckSubstitution για τα 20 προφίλ συνολικά

5.2.2. Μετρήσεις για τον αλγόριθμο ProfilesHierarchyUpdate

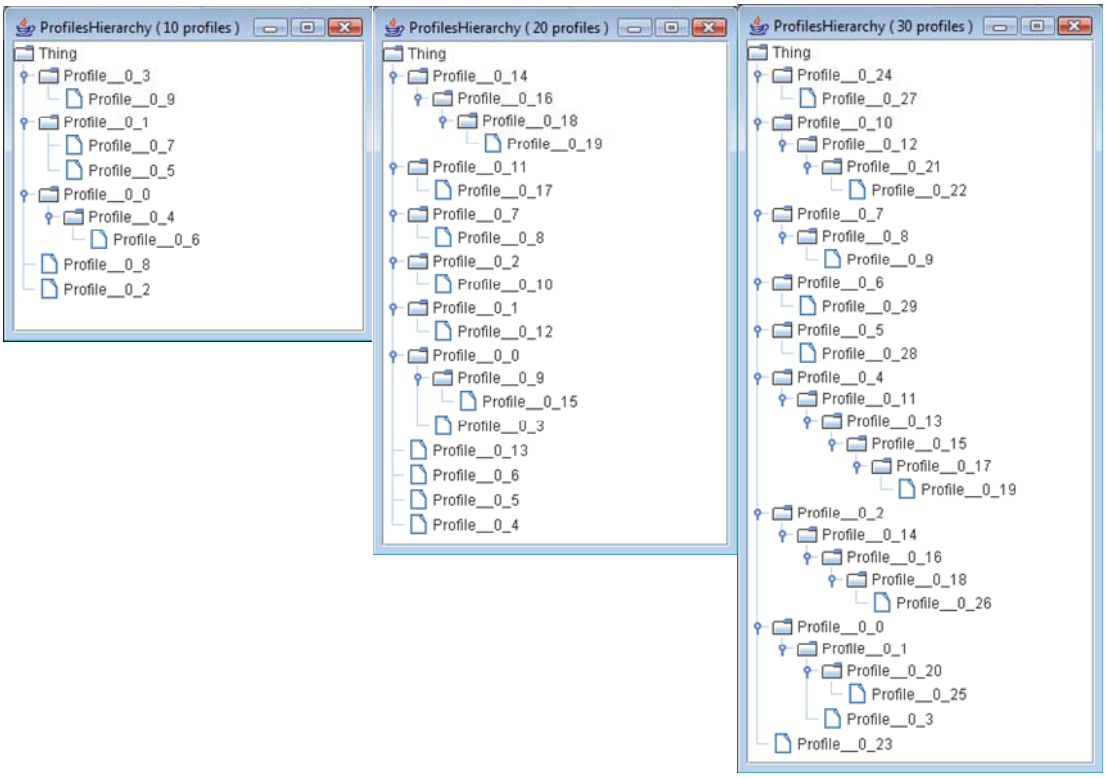
Έστω ότι ο αλγόριθμος *ProfilesHierarchyGeneration* δέχεται ως είσοδο την δενδρική δομή *DataTypesHierarchy* του σχήματος 5.2, η οποία έχει βάθος τέσσερα. Ο αλγόριθμος *ProfilesHierarchyUpdate* εισάγει ένα νέο προφίλ στη δομή *ProfilesHierarchy*.

Σε αυτήν την μέτρηση γίνονται 10 εκτελέσεις του αλγορίθμου *ProfilesHierarchyGeneration*. Στην πρώτη εκτέλεση παράγονται 10 προφίλ και με αυτά κατασκευάζεται η δομή *ProfilesHierarchy*. Το πλήθος των προφίλ που παράγονται σε κάθε εκτέλεση είναι κατά 10 περισσότερα από την προηγούμενη εκτέλεση. Σε κάθε επόμενη εκτέλεση παράγεται εκ νέου η δομή *ProfilesHierarchy*. Σε όλες τις εκτελέσεις, τα προφίλ που παράγονται έχουν μόνο μία ατομική διεργασία, ένα δεδομένο εισόδου και ένα δεδομένο εξόδου. Στα επόμενα σχήματα απεικονίζεται ενδεικτικά η δομή *ProfilesHierarchy* που παράγεται σε μερικές από αυτές εκτελέσεις. Παρατηρούμε ότι όσο αυξάνεται το πλήθος των προφίλ, αυξάνεται και το βάθος του δένδρου. Σε μία εκτέλεση του αλγορίθμου *ProfilesHierarchyGeneration* υπογίζεται ο χρόνος που απαιτείται για να εισαχθεί ένα προφίλ στην δομή. Αυτός είναι και ο χρόνος εκτέλεσης του αλγορίθμου *ProfilesHierarchyUpdate* για ένα προφίλ. Στην συνέχεια, υπολογίζεται ο μέσος χρόνος εκτέλεσης του αλγορίθμου

ProfilesHierarchyUpdate για τα πρόφιλ που θα εισαχθούν τελικά στην δομή. Αυτές οι μετρήσεις επαναλαμβάνονται σε κάθε μία από τις 10 εκτελέσεις του αλγορίθμου *ProfilesHierarchyGeneration* (σχήμα 5.17) και αποδίδονται γραφικά στο σχήμα 5.18.

Συμπεράσματα:

Από το σχήμα 5.18 συμπεραίνουμε ότι ο χρόνος εκτέλεσης του αλγορίθμου *ProfilesHierarchyUpdate* εξαρτάται από το πλήθος των προφίλ της δομής. Πιο αναλυτικά, όσο αυξάνεται το πλήθος των προφίλ, αυξάνεται και το βάθος της δομής. Όταν η δομή έχει μεγάλο βάθος τότε ο αλγόριθμος εκτελεί περισσότερες συγκρίσεις και επομένως μεγαλώνει και ο χρόνος εκτέλεσής του. Επιπλέον, από το σχήμα γίνεται φανερή μία απότομη αύξηση του χρόνου εκτέλεσης του αλγορίθμου όταν το πλήθος των προφίλ αυξάνεται από 50 σε 60. Αυτό συμβαίνει γιατί το βάθος του δένδρου αυξήθηκε απότομα.

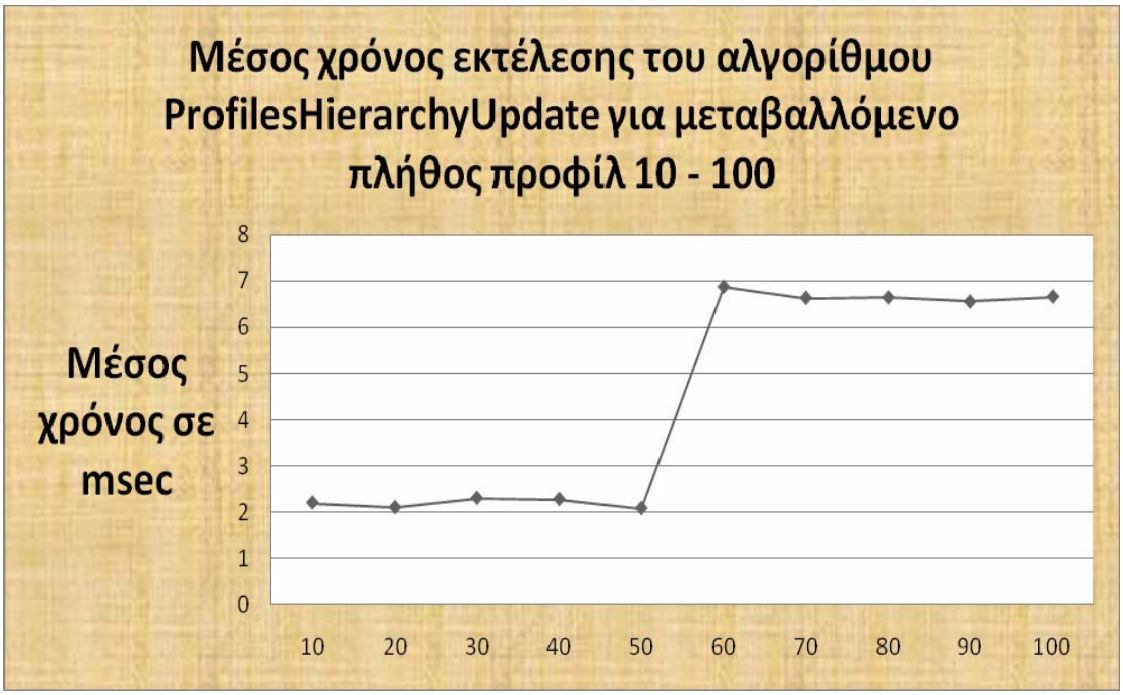


Σχήμα 5.15 Η δομή ProfilesHierarchy για 10, 20 και 30 προφίλ αντίστοιχα από διαφορετικές εκτελέσεις του αλγορίθμου ProfilesHierarchyGeneration

Σχήμα 5.16 Η δομή ProfilesHierarchy για 40, 70 και 100 προφίλ αντίστοιχα από διαφορετικές εκτελέσεις του αλγορίθμου ProfilesHierarchyGeneration

Το πλήθος των προφίλ της δομής ProfilesHierarchy	**Μέσος Χρόνος σε msec**
10	**2.2**
20	**2.1**
30	**2.3**
40	**2.275**
50	**2.08**
60	**6.866**
70	**6.628**
80	**6.637**
90	**6.548**
100	**6.655**

Σχήμα 5.17 Οι μετρήσεις του μέσου χρόνου εκτέλεσης του ProfilesHierarchyUpdate για μεταβαλλόμενο πλήθος από προφίλ 10 – 100



Σχήμα 5.18 Μέσος χρόνος εκτέλεσης του ProfilesHierarchyUpdate για μεταβαλλόμενο πλήθος από προφίλ 10 – 100

5.2.3. Μετρήσεις για τον αλγόριθμο ServiceSearching

Έστω ότι ο αλγόριθμος *ProfilesHierarchyGeneration* δέχεται ως είσοδο την δενδρική δομή *DataTypesHierarchy* του σχήματος 5.2, η οποία έχει βάθος τέσσερα. Ο αλγόριθμος *ServiceSearching* υλοποιεί την σημασιολογική αναζήτηση ενός προφίλ στη δομή *ProfilesHierarchy*.

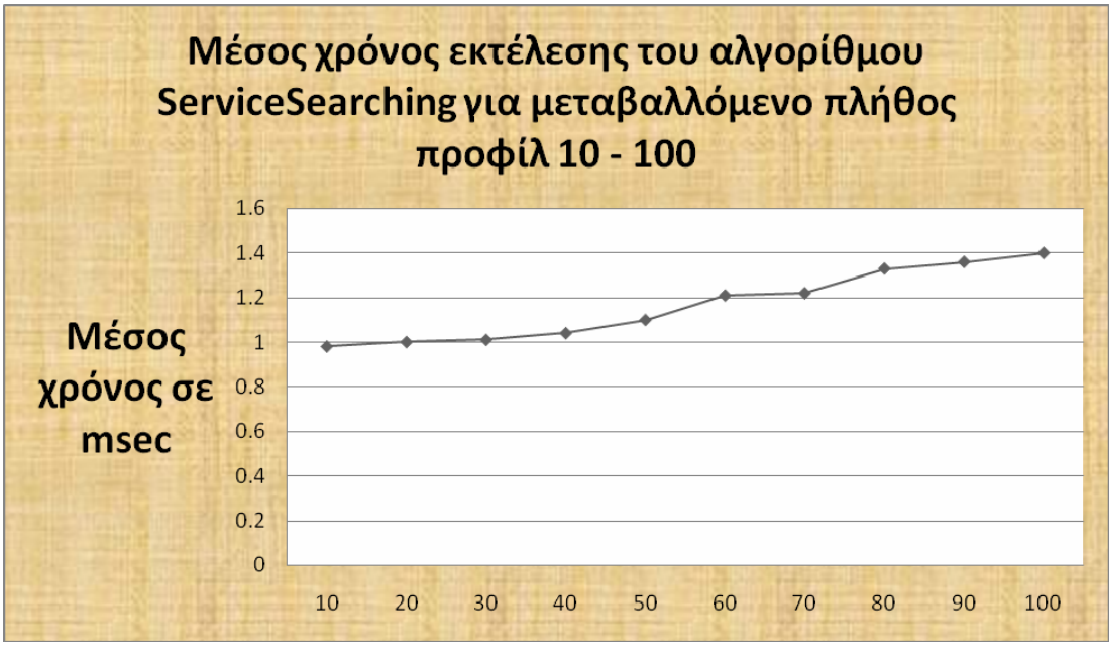
Σε αυτήν την μέτρηση γίνονται 10 εκτελέσεις του αλγορίθμου *ProfilesHierarchyGeneration*. Στην πρώτη εκτέλεση παράγονται 10 προφίλ και με αυτά κατασκευάζεται η δομή *ProfilesHierarchy*. Το πλήθος των προφίλ που παράγονται σε κάθε εκτέλεση είναι κατά 10 περισσότερα από την προηγούμενη εκτέλεση. Σε κάθε επόμενη εκτέλεση παράγεται εκ νέου η δομή *ProfilesHierarchy*. Σε όλες τις εκτελέσεις, τα προφίλ που παράγονται έχουν μόνο μία ατομική διεργασία, ένα δεδομένο εισόδου και ένα δεδομένο εξόδου. Στα επόμενα σχήματα απεικονίζεται ενδεικτικά η δομή *ProfilesHierarchy* που παράγεται σε μερικές από αυτές εκτελέσεις. Σε μία εκτέλεση του αλγορίθμου *ProfilesHierarchyGeneration* γίνεται σημασιολογική αναζήτηση ενός τυχαίου προφίλ και υπογίζεται ο χρόνος που απαιτείται για την αναζήτηση. Αυτός είναι και ο χρόνος εκτέλεσης του αλγορίθμου

ServiceSearching για το συγκεκριμένο προφίλ. Για την ίδια δομή *ProfilesHierarchy*, γίνονται συνολικά 50 σημασιολογικές αναζητήσεις τυχαία επιλεγμένων προφίλ και υπολογίζεται ο μέσος χρόνος αναζήτησης. Αυτές οι μετρήσεις επαναλαμβάνονται σε κάθε μία από τις 10 εκτελέσεις του αλγορίθμου *ProfilesHierarchyGeneration* και αποδίδονται γραφικά στο σχήμα 5.19. Τα δένδρα που προέκυψαν από αυτές τις 10 εκτελέσεις έχουν την ίδια μορφή και το ίδιο βάθος με αυτά που απεικονίζονται ενδεικτικά στα σχήματα 5.15, 5.16.

Στην συνέχεια, επαναλήφθηκαν δύο ακόμα φορές όλες οι παραπάνω μετρήσεις για παραγόμενα προφίλ τα οποία είχαν δύο και τρία δεδομένα εισόδου αντίστοιχα. Αυτές οι μετρήσεις αποδίδονται γραφικά στα σχήματα 5.20 και 5.21 αντίστοιχα.

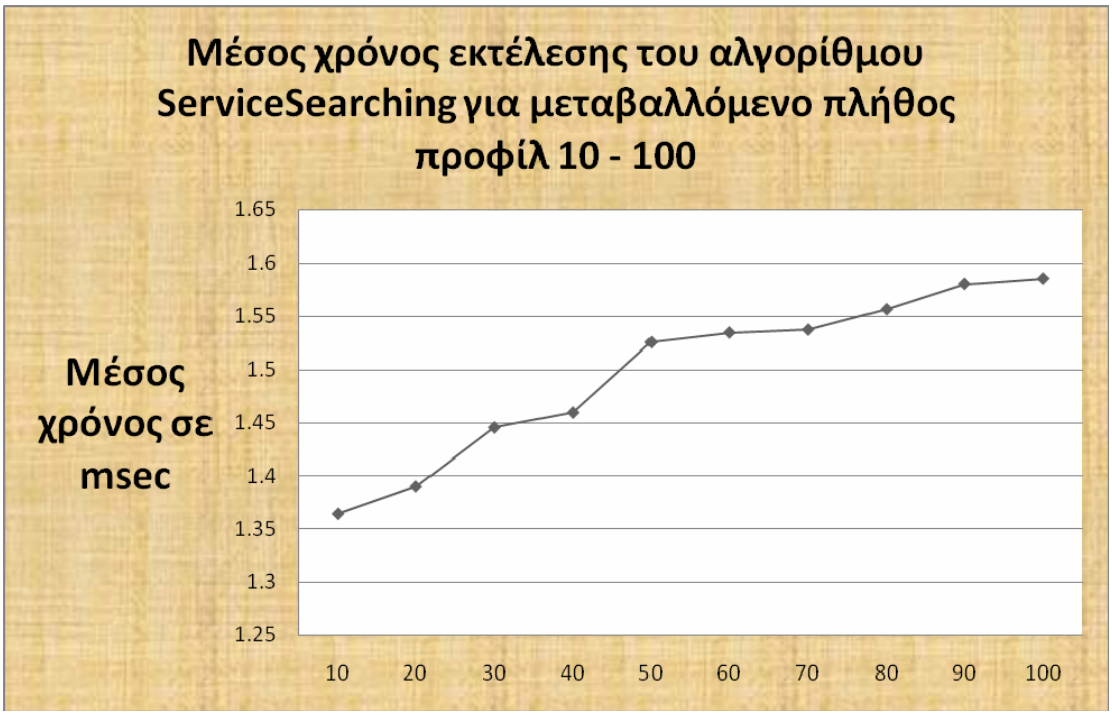
Συμπεράσματα:

Από το σχήμα 5.19 συμπεραίνουμε ότι ο χρόνος εκτέλεσης του αλγορίθμου *ServiceSearching* εξαρτάται από το πλήθος των προφίλ της δομής. Όσο αυξάνεται το πλήθος των προφίλ της δομής, αυξάνεται και ο μέσος χρόνος εκτέλεσης του αλγορίθμου. Ωστόσο, αυτή η αύξηση είναι μικρή αναλογικά με την αύξηση του πλήθους των προφίλ. Την ίδια συμπεριφορά παρουσιάζει ο χρόνος εκτέλεσης του αλγορίθμου και στην περίπτωση που τα προφίλ έχουν δύο ή τρία δεδομένα εισόδου.



Σχήμα 5.19 Μέσος χρόνος εκτέλεσης του ServiceSearching για μεταβαλλόμενο πλήθος από προφίλ 10 – 100 με ένα δεδομένο εισόδου

Σχήμα 5.20 Μέσος χρόνος εκτέλεσης του ServiceSearching για μεταβαλλόμενο πλήθος από προφίλ 10 – 100 με δύο δεδομένα εισόδου



Σχήμα 5.21 Μέσος χρόνος εκτέλεσης του ServiceSearching για μεταβαλλόμενο πλήθος από προφίλ 10 – 100 με τρία δεδομένα εισόδου

5.2.4. Μετρήσεις για την JAX-RPC κλήση μίας μεθόδου

Ο κώδικας ενός πελάτη που χρησιμοποιείται για την κλήση μία μεθόδου της αρχικής υπηρεσίας *Target* (σχήμα 4.23), βασίζεται στη διεπαφή της δυναμικής κλήσης των υπηρεσιών διαδικτύου σύμφωνα με το πρότυπο JAX-RPC (Dynamic Invocation Interface – DII). Έγιναν μετρήσεις για την διάρκεια αυτής της κλήσης σε συνάρτηση με το πλήθος των δεδομένων εισόδου της μεθόδου. Η μέθοδος είχε μόνο ένα δεδομένο εξόδου. Στο σχήμα 5.22 απεικονίζονται οι μετρήσεις της μέσης διάρκειας μίας JAX-RPC κλήσης για μεταβαλλόμενο πλήθος δεδομένων εισόδου και στο σχήμα 5.23 αποδίδονται γραφικά.

Πλήθος Δεδομένων Εισόδου	**Μέσος Χρόνος σε msec**
1	**7.0**
2	**7.2**
3	**7.3**
4	**7.1**
5	**7.3**
6	**7.2**
7	**7.4**
8	**7.7**
9	**7.6**
10	**7.7**

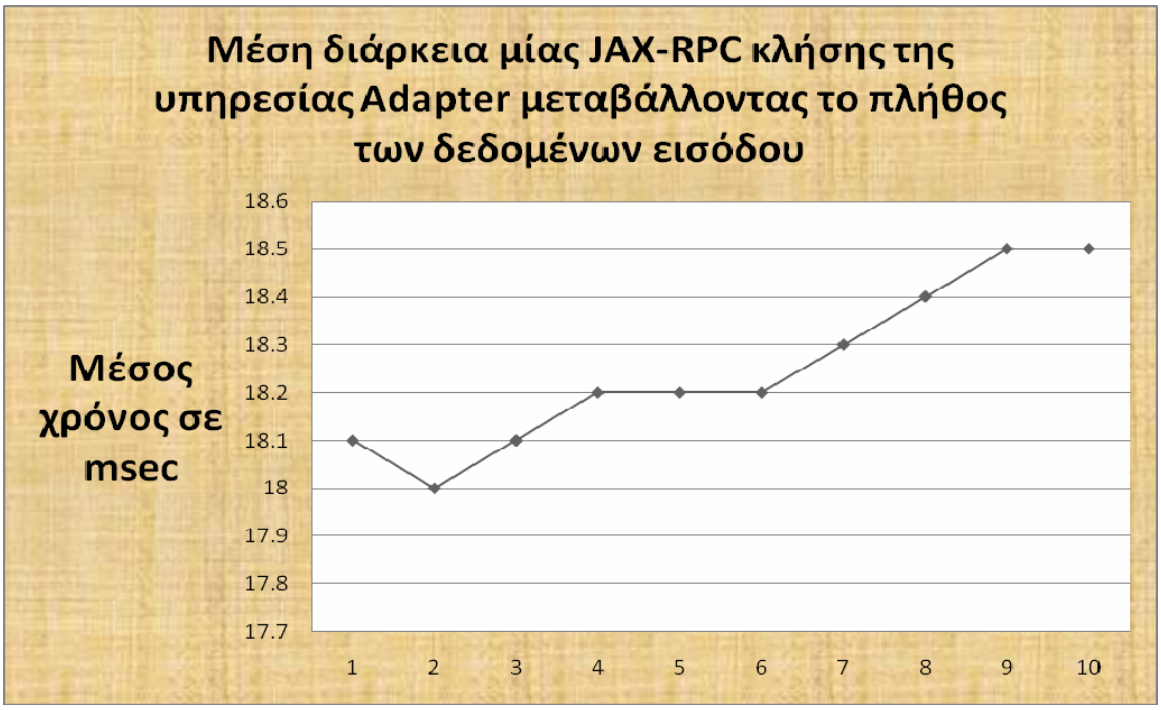
Σχήμα 5.22 Οι μετρήσεις της μέσης διάρκειας μίας JAX-RPC κλήσης για μεταβαλλόμενο πλήθος δεδομένων εισόδου

Σχήμα 5.23 Μέση διάρκεια μίας JAX-RPC κλήσης για μεταβαλλόμενο πλήθος δεδομένων εισόδου

Στην περίπτωση που η υπηρεσία *Target* αντικατασταθεί από την υπηρεσία *Adapter* (σχήμα 4.24), τότε η διάρκεια εκτέλεσης της JAX-RPC κλήσης είναι μεγαλύτερη. Αυτό συμβαίνει γιατί εντός της μεθόδου της υπηρεσίας *Adapter* γίνεται και μία επιπλέον κλήση σε μία μεθόδου της υπηρεσίας *Adaptee*. Έγιναν μετρήσεις για την διάρκεια αυτής της κλήσης σε συνάρτηση με το πλήθος των δεδομένων εισόδου της μεθόδου της υπηρεσίας *Adapter*. Στο σχήμα 5.24 απεικονίζονται οι μετρήσεις της μέσης διάρκειας μίας τέτοιας JAX-RPC κλήσης για μεταβαλλόμενο πλήθος δεδομένων εισόδου και στο σχήμα 5.25 αποδίδονται γραφικά.

Πλήθος Δεδομένων Εισόδου για τις υπηρεσίες Target και Adaptee	**Μέσος Χρόνος σε msec**
1	**18.1**
2	**18.0**
3	**18.1**
4	**18.2**
5	**18.2**
6	**18.2**
7	**18.3**
8	**18.4**
9	**18.5**
10	**18.5**

Σχήμα 5.24 Οι μετρήσεις της μέσης διάρκειας μίας JAX-RPC κλήσης για την υπηρεσία Adapter μεταβάλλοντας το πλήθος των δεδομένων εισόδου



Σχήμα 5.25 Μέση διάρκεια μίας JAX-RPC κλήσης για την υπηρεσία Adapter μεταβάλλοντας το πλήθος των δεδομένων εισόδου

ΚΕΦΑΛΑΙΟ 6. ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΜΕΛΛΟΝΤΙΚΕΣ ΕΠΕΚΤΑΣΕΙΣ

6.1. Συμπεράσματα

Στην παρούσα εργασία αντιμετωπίσαμε το πρόβλημα της μεταβαλλόμενης διαθεσιμότητας υπηρεσιών διαδικτύου σε περιβάλλοντα διάχυτου υπολογισμού. Πιο συγκεκριμένα, υλοποιήθηκε μια προσέγγιση που να επιτρέπει την αντικατάσταση μη διαθέσιμων υπηρεσιών από άλλες, οι οποίες προσφέρουν την ίδια λειτουργικότητα μέσω διαφορετικών διεπαφών, με τρόπο κλειστό ως προς τον κώδικα των εφαρμογών που χρησιμοποιούν αυτές τις υπηρεσίες, δηλαδή χωρίς να απαιτούνται ριζικές αλλαγές στον κώδικα αυτών των εφαρμογών.

Για να υλοποιηθούν τα παραπάνω, προτάθηκε ένα σύστημα λογισμικού το οποίο:

1. Οργανώνει τις υπηρεσίες με παρόμοια λειτουργικότητα μέσω διαφορετικών διεπαφών σε κατηγορίες οι οποίες χαρακτηρίζονται από αφηρημένες σημασιολογικές περιγραφές.
2. Αναζητά υπηρεσίες που μπορούν να αντικαταστήσουν μια μη διαθέσιμη υπηρεσία με βάση την αφηρημένη σημασιολογική περιγραφή της λειτουργικότητάς τους.
3. Αντικαθιστά τη μη διαθέσιμη υπηρεσία με τρόπο κλειστό ως προς τον κώδικα των εφαρμογών που τη χρησιμοποιούν με βάση το Προσαρμοστικό Σχεδιαστικό Πρότυπο (Adapter Design Pattern).

Έγιναν πειραματικές μετρήσεις σε όλους τους αλγορίθμους που χρησιμοποιεί το σύστημα. Από αυτές τις μετρήσεις προέκυψε ότι το χρονικό κόστος που απαιτείται

για την οργάνωση και συντήρηση των δεδομένων του προτεινόμενου συστήματος είναι σε λογικά πλαίσια. Η αναζήτηση υπηρεσιών που μπορούν να αντικαταστήσουν μία μη διαθέσιμη υπηρεσία επίσης γίνεται με μικρό χρονικό κόστος. Επιπλέον, έγιναν μετρήσεις σχετικά με την αντικατάσταση μίας υπηρεσίας από κάποια άλλη και προέκυψε το συμπέρασμα ότι το προτεινόμενο σύστημα πραγματοποιεί την αντικατάσταση με ελάχιστες αλλαγές στην εφαρμογή του πελάτη, εισάγοντας όμως μια αναμενόμενη χρονική καθυστέρηση στην εκτέλεση της εφαρμογής.

6.2. Μελλοντικές επεκτάσεις

Στα πλαίσια του συστήματος που αναπτύχθηκε, ορίστηκε η σχέση της σημασιολογικής επέκτασης/εξειδίκευσης των δεδομένων εισόδου/εξόδου διαφορετικών προφίλ υπηρεσιών με σκοπό την δημιουργία αντίστοιχων συσχετίσεων μεταξύ των προφίλ αυτών. Το σύστημα που υλοποιήθηκε δέχεται ως είσοδο μία στατική δενδρική δομή που συσχετίζει δεδομένα εισόδου/εξόδου διαθέσιμων υπηρεσιών. Δεδομένης αυτής της δομής, προέκυψε μια αντίστοιχη δενδρική δομή για τα προφίλ αυτών των υπηρεσιών. Μία επέκταση της παρούσας εργασίας θα μπορούσε να είναι η δυναμική παραγωγή της δενδρικής δομής που συσχετίζει τα δεδομένα εισόδου/εξόδου.

Επιπλέον, στο σύστημα που υλοποιήθηκε, δεν γίνεται η δυναμική αντικατάσταση μη διαθέσιμων υπηρεσιών. Μία επέκταση της παρούσας εργασίας θα μπορούσε να είναι η αντικατάσταση μη διαθέσιμων υπηρεσιών σε χρόνο εκτέλεσης ώστε η εφαρμογή του πελάτη να συνεχίσει να λειτουργεί κανονικά μετά την αντικατάσταση.

ΑΝΑΦΟΡΕΣ

[1] J. Kramer and J. Magee. 'The evolving Philoshophers Problem: Dynamic Change Management', IEEE Transactions on Software Engineering, Vol. 16, No. 11, November 1990.

[2] C. Bidan, V. Issanry, T. Saridakis, A. Zarras. 'A dynamic reconfiguration service for corba', ICCDS '98, Proceedings of the 4th IEEE International Conference on Configurable Systems, pp 35-42, 1998.

[3] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella. 'Automatic Service Composition Based on Behavioral Descriptions', International Journal of Cooperative Information Systems Vol. 14, pp 333–376, No. 4, 2005.

[4] N. Minsky, V. Ungureanu, W. Wang, J. Zhang. 'Building reconfiguration primitives into the law of a system', ICCDS '96, Proceedings of the 3rd IEEE International Conference on Configurable Distributed Systems, pp 62-69, 1996.

[5] P. K. McKinley, R. E. Kurt Stirewalt, B. H. C. Cheng, L. K. Dillon, S. Kulkarni. 'RAPIDware: Component-Based Development of Adaptive and Dependable Middleware', August 2005.

[6] A. Mukhija, M. Glinz. 'CASA – A Contract-based Adaptive Software Architecture Framework', Proceedings of the 3rd IEEE Workshop on Applications and Services in Wireless Networks (ASWN 2003), Berne, Switzerland, July 2003, pp. 275-286.

[7] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. 'Design Patterns: Elements of Reusable Object-Oriented Software' (Addison-Wesley Professional Computing Series), Hardcover - Nov 10, 1994.

[8] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, S. Weerawarana. 'Business Process Execution Language for Web Services', Version 1.0, 31 July 2002.

[9] R. Khalaf, N. Mukhi, S. Weerawarana. 'Service–Oriented Composition in BPEL4WS', *WWW2003*, May 20–24, 2003, Budapest, Hungary, ACM 1581136803/ 03/0005.xxx.

[10] Y. Xu, S. Tang, Y. Xu, Z. Tang. 'Towards Aspect Oriented Web Service Composition with UML', 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007), 0-7695-2841-4/07 2007 IEEE.

[11] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, K. Sycara. 'OWL-S: Semantic Markup for Web Services', W3C Member Submission 22 November 2004.

[12] http://www.oracle.com/technology/products/jdev/index.html.

[13] E.Christensen, F. Curbera, G.Meredith, S.Weerawarana, 'Web Services Description Language (WSDL) 1.1', W3C Note, 15 March 2001.

ΔΗΜΟΣΙΕΥΣΕΙΣ ΣΥΓΓΡΑΦΕΑ

D. Athanasopoulos, A. V. Zarras, V. Issarny, E. Pitoura, P. Vassiliadis. «CoWSAMI: Interface-aware context gathering in ambient intelligence environments», Pervasive and Mobile Computing, Volume 4, Issue 3, June 2008, Pages 360-389.

ΣΥΝΤΟΜΟ ΒΙΟΓΡΑΦΙΚΟ

Ο Διονύσης Αθανασόπουλος γεννήθηκε στην Πάτρα Αχαΐας, αποφοίτησε από το 1^{ο} Λύκειο Αμαλιάδας και είναι πτυχιούχος του τμήματος Πληροφορικής του Πανεπιστημίου Ιωαννίνων.