ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΗΜΟΣΙΟΠΟΙΗΣΗ ΠΛΗΡΟΦΟΡΙΩΝ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ ΜΕ ΧΡΗΣΗ ΣΥΝΔΥΑΣΜΕΝΩΝ ΥΠΗΡΕΣΙΩΝ WEB (ΤΟΜΟΣ Ι)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΚΩΝΣΤΑΝΤΙΝΟΥ Φ. ΛΑΙΜΟΥ

Επιβλέπων : ΤΙΜΟΛΕΩΝ Κ. ΣΕΛΛΗΣ
Καθηγητής

Αθήνα, Ιούλιος 2003

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΗΜΟΣΙΟΠΟΙΗΣΗ ΠΛΗΡΟΦΟΡΙΩΝ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ ΜΕ ΧΡΗΣΗ ΣΥΝΔΥΑΣΜΕΝΩΝ ΥΠΗΡΕΣΙΩΝ WEB (ΤΟΜΟΣ Ι)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΚΩΝΣΤΑΝΤΙΝΟΥ Φ. ΛΑΙΜΟΥ

Επιβλέπων : ΤΙΜΟΛΕΩΝ Κ. ΣΕΛΛΗΣ
Καθηγητής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 14η Ιουλίου 2003.

.....	.....	.....
Τιμολέων Σελλής	Ιωάννης Βασιλείου	Νεκτάριος Κοζύρης
Καθηγητής	Καθηγητής	Αναπληρωτής Καθηγητής

Αθήνα, Ιούλιος 2003

.....

ΚΩΝΣΤΑΝΤΙΝΟΣ Φ. ΛΑΙΜΟΣ
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Πρόλογος

Η παρούσα διπλωματική εργασία εκπονήθηκε κατά τη διάρκεια του έτους 2003 από τον Κωνσταντίνο Λαιμό. Ένα μέρος της ανάπτυξης έγινε στο Εργαστήριο Συστημάτων Βάσεων Γνώσεων και Δεδομένων του τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου.

Η υπόδειξη του θέματος έγινε από τον καθηγητή κ. Τίμο Σελλή, τον οποίο θα ήθελα να ευχαριστήσω θερμά, τόσο για το ενδιαφέρον που έδειξε, όσο και για τις προτάσεις και τις παρατηρήσεις, που έκανε, για τη βελτίωση της εργασίας.

Καθοριστική στην πραγματοποίηση αυτού του έργου ήταν και η συμμετοχή του διπλωματικού φοιτητή κ. Ευάγγελου Ζορμπά, τον οποίο είχα τη χαρά να έχω σαν επιβλέποντα βοηθό. Η καθοδήγησή του, η επιμονή και η υπομονή του, καθώς επίσης και το ενδιαφέρον που έδειξε για το αντικείμενο της διπλωματικής και για την πρόοδό της, όσον αφορά την υλοποίησή της, ήταν σημαντικοί παράγοντες που οδήγησαν στην επιτυχή ολοκλήρωσή της.

Σίγουρα δεν θα μπορούσα να παραλείψω να ευχαριστήσω όλους τους μεταπτυχιακούς φοιτητές και υπευθύνους του Εργαστηρίου Συστημάτων Βάσεων Γνώσεων και Δεδομένων, οι οποίοι με τον τρόπο τους συνετέλεσαν στην ύπαρξη ιδανικού κλίματος και περιβάλλοντος για την εκπόνηση της διπλωματικής εργασίας.

Ευχαριστώ, τέλος, την οικογένεια μου, που όλο αυτό το διάστημα στήριξε την προσπάθειά μου, σε όλες τις φάσεις της ανάπτυξης και υλοποίησης του έργου.

Ιούλιος 2003

Λαιμός Κωνσταντίνος

Περίληψη

Το αντικείμενο της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη μιας εφαρμογής, η οποία να επιτρέπει τη δημοσιοποίηση των δεδομένων μιας Βάσης Δεδομένων ως Web Service. Με τον όρο Web Service εννοούμε μια υπηρεσία, η οποία βρίσκεται σε έναν διακομιστή Web και την οποία μπορεί να χρησιμοποιήσει ο οποιοσδήποτε έχει πρόσβαση σε αυτόν. Μια τέτοια υπηρεσία θα μπορούσε να ήταν ένας πολύπλοκος υπολογισμός φόρου, τον οποίο θα θέλαμε να προσθέσουμε στην ιστοσελίδα μας. Έτσι αντί να προγραμματίσουμε εμείς τον κώδικα που απαιτείται για τον υπολογισμό αυτό, χρησιμοποιούμε στην ιστοσελίδα μας την υπηρεσία αυτή, πληρώνοντας φυσικά το αντίστοιχο τίμημα. Το μόνο που χρειάζεται για να χρησιμοποιήσουμε την υπηρεσία είναι να ξέρουμε τον τρόπο κλήσης και τη μορφή των μεταβλητών που απαιτούνται και να μπορούμε να επικοινωνήσουμε με τέτοιο τρόπο, ώστε η υπηρεσία να καταλάβει την κλήση μας και να επιστρέψει ένα κατανοητό, για εμάς, αποτέλεσμα.

Η εφαρμογή που αναπτύξαμε αποτελεί ακριβώς μια τέτοια υπηρεσία Web. Συγκεκριμένα κληθήκαμε να αναπτύξουμε μια εφαρμογή, η οποία να επιτρέπει στον κάτοχο μιας Βάσης Δεδομένων να κάνει publish κάποια από τα δεδομένα της βάσης του ως Web Service. Κύριο στόχο αποτέλεσε το να κρύψουμε τελείως από τον κάτοχο της βάσης τον τρόπο με τον οποίο δημιουργείται το Web Service, απαλλάσσοντας τον από προγραμματισμό υψηλού επιπέδου. Το μόνο που θα πρέπει να τον ενδιαφέρει είναι να επιλέξει τα δεδομένα που θέλει να δημοσιοποιήσει (γράφοντας τα κατάλληλα SQL Queries) και στη συνέχεια το σύστημα μας αναλαμβάνει αυτόματα τη δημιουργία του.

Η προσπάθειά μας τέλος επικεντρώθηκε στο να υλοποιήσουμε ένα αυτοματοποιημένο και πλήρες σύστημα, το οποίο θα παρέχει όσο το δυνατό πιο εύχρηστες και λειτουργικές υπηρεσίες.

Λέξεις Κλειδιά: Web Services, XML, SOAP, WSDL, Java, Java Web Services Developer Pack (JWSDP), SQL Server 2000.

Abstract

The scope of the current diploma thesis is the development of an application, which provides the capability of publishing the data of a Database as a Web Service. The term Web Service stands for a service which exists in a Web Server and can be used by anyone with access to that server. Such a service could be a complicated tax calculation that we would like to embed into our Web page. Instead of writing the required code ourselves, we can use the service in our page, by paying the corresponding price. The only thing we need to use this service is knowledge of the calling methods and the required variable types, in order to be able to communicate in a way that the service will understand our call and return an understandable - for us- result.

The application we developed consists of such a Web Service. Specifically our application enables the owner of a Database to publish some of the data contained in the Database as a Web Service. Our main goal was to hide the way the Web Service is created from the owner of the database, relieving him from high-level programming tasks. The only task he is required to complete is the selection of the data he wishes to publish (via appropriate SQL statements) and our system undertakes the automatic creation of the Web Service.

We made an effort to create an automated and complete system that will provide, as much as possible, functional and easy to use services.

Keywords: Web Services, XML, SOAP, WSDL, Java, Java Web Services Developer Pack (JWSDP), SQL Server 2000.

Πίνακας περιεχομένων

Αναλυτικός πίνακας περιεχομένων

1

Εισαγωγή

Στο παρόν κεφάλαιο, στην παράγραφο 1.1 αναλύεται το αντικείμενο της παρούσας διπλωματικής εργασίας και στην παράγραφο 1.2 γίνεται μια επισκόπηση της οργάνωσης του τόμου που κρατάτε στα χέρια σας.

1.1 Αντικείμενο της διπλωματικής

Το αντικείμενο της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη μιας εφαρμογής, η οποία να επιτρέπει τη δημοσιοποίηση των δεδομένων μιας Βάσης Δεδομένων ως Web Service. Με τον όρο Web Service εννοούμε μια υπηρεσία, η οποία βρίσκεται σε έναν διακομιστή Web και την οποία μπορεί να χρησιμοποιήσει ο οποιοσδήποτε έχει πρόσβαση σε αυτόν. Μια τέτοια υπηρεσία θα μπορούσε να ήταν ένας πολύπλοκος υπολογισμός φόρου, τον οποίο θα θέλαμε να προσθέσουμε στην ιστοσελίδα μας. Έτσι αντί να προγραμματίσουμε εμείς τον κώδικα που απαιτείται για τον υπολογισμό αυτό, χρησιμοποιούμε στην ιστοσελίδα μας την υπηρεσία αυτή, πληρώνοντας φυσικά το αντίστοιχο τίμημα. Το μόνο που χρειάζεται για να χρησιμοποιήσουμε την υπηρεσία είναι να ξέρουμε τον τρόπο κλήσης και τη μορφή των μεταβλητών που απαιτούνται και να μπορούμε να επικοινωνήσουμε με τέτοιο τρόπο, ώστε η υπηρεσία να καταλάβει την κλήση μας και να επιστρέψει ένα κατανοητό, για εμάς, αποτέλεσμα.

Η εφαρμογή που αναπτύξαμε αποτελεί ακριβώς μια τέτοια υπηρεσία Web. Συγκεκριμένα κληθήκαμε να αναπτύξουμε μια εφαρμογή, η οποία να επιτρέπει στον κάτοχο μιας Βάσης

Δεδομένων να κάνει publish κάποια από τα δεδομένα της βάσης του ως Web Service. Κύριο στόχο αποτέλεσε το να κρύψουμε τελείως από τον κάτοχο της βάσης τον τρόπο με τον οποίο δημιουργείται το Web Service, απαλλάσσοντας τον από προγραμματισμό υψηλού επιπέδου. Το μόνο που θα πρέπει να τον ενδιαφέρει είναι να επιλέξει τα δεδομένα που θέλει να δημοσιοποιήσει (γράφοντας τα κατάλληλα SQL Queries) και στη συνέχεια το σύστημα μας αναλαμβάνει αυτόματα τη δημιουργία του.

Η προσπάθειά μας τέλος επικεντρώθηκε στο να υλοποιήσουμε ένα αυτοματοποιημένο και πλήρες σύστημα, το οποίο θα παρέχει όσο το δυνατό πιο εύχρηστες και λειτουργικές υπηρεσίες.

1.2 Οργάνωση του τόμου

Η διπλωματική εργασία αποτελείται από δύο τόμους και ένα CD-ROM.

Ο πρώτος τόμος που κρατάτε στα χέρια σας αποτελείται από 6 κεφάλαια και αναλύει πλήρως την ανάπτυξη της εργασίας.

Το ***πρώτο κεφάλαιο*** είναι η εισαγωγή, η οποία δίνει μια σύντομη περιγραφή του συστήματος μας, εξηγεί τις ανάγκες που αυτό έρχεται να καλύψει και παρουσιάζει το γενικό πλάνο εργασίας του έργου.

Στο ***δεύτερο κεφάλαιο*** γίνεται μια εκτενέστερη περιγραφή του θέματος της διπλωματικής. Παρουσιάζονται με συντομία υπάρχοντα συστήματα σχετικά με το αντικείμενο της διπλωματικής εργασίας και στη συνέχεια αναλύεται ο στόχος και οι ανάγκες που αυτή έρχεται να καλύψει.

Στο ***τρίτο κεφάλαιο*** δίνεται το θεωρητικό υπόβαθρο που χρειάζεται να έχει κάποιος που διαβάζει την παρούσα εργασία. Περιγράφει όλα τα πρωτόκολλα και τα εργαλεία που χρησιμοποιήθηκαν και δίνει την ιστορία και τους στόχους των Web Services, με τα οποία ασχολείται κατά κύριο λόγο η εργασία αυτή.

Το ***τέταρτο κεφάλαιο*** ασχολείται με τη σχεδίαση και την ανάπτυξη του συστήματος μας. Αρχικά, ορίζονται πλήρως οι προδιαγραφές του συστήματος και οι απαιτήσεις των χρηστών από αυτό. Ακολουθεί η περιγραφή του σχεδιασμού της εφαρμογής, η ανάλυσή της σε επιμέρους στοιχεία και η διαδικασία ανάπτυξής της.

Στο ***πέμπτο κεφάλαιο*** γίνεται η περιγραφή της υλοποίησης του συστήματος. Τεκμηριώνεται η επιλογή της πλατφόρμας και των προγραμματιστικών εργαλείων που επιλέχθηκαν για την ανάπτυξη και παρατίθεται μια πιο λεπτομερής περιγραφή της υλοποίησης των επιμέρους στοιχείων της εφαρμογής.

Το ***έκτο κεφάλαιο*** είναι αφιερωμένο στον έλεγχο του συστήματος. Παρουσιάζεται αρχικά η μεθοδολογία του ελέγχου και κατόπιν παρατίθονται αναλυτικά τα αποτελέσματα του ελέγχου στο σύστημά μας.

Στο ***έβδομο κεφάλαιο***, που αποτελεί τον επίλογο της διπλωματικής, γίνεται επισκόπηση της εργασίας και παρουσιάζονται ορισμένες ιδέες που αφορούν βελτιώσεις και μελλοντικές επεκτάσεις του συστήματος.

Ο δεύτερος τόμος αποτελεί το εγχειρίδιο χρήσης του συστήματος μας. Υποδεικνύει τον τρόπο εγκατάστασης της εφαρμογής και τον τρόπο λειτουργίας αυτής.

Στο CD-ROM περιέχεται ο πηγαίος κώδικας της εφαρμογής, ο εκτελέσιμος κώδικας, τα αρχεία τεκμηρίωσης, οι δύο τόμοι σε ηλεκτρονική μορφή και οι διαφάνειες παρουσίασης της διπλωματικής εργασίας.

2

Περιγραφή Θέματος

2.1 Σχετικές εργασίες (εφαρμογές)

Ο κύριος ανταγωνιστής της εφαρμογής που φτιάξαμε με τη βοήθεια της Java είναι ο συνδυασμός του .NET και του SQL Server 2000 της Microsoft. Συγκεκριμένα, έχει κυκλοφορήσει μια αναβάθμιση του SQL Server 2000 με το όνομα SQLXML 3.0, που έχει ως σκοπό τη μετατροπή των δεδομένων σε XML για την ευκολότερη δημοσιοποίησή τους ως Web Service.

Ο τρόπος με τον οποίο δημοσιεύονται τα δεδομένα του SQL Server ως Web Service είναι με τη χρήση των λεγόμενων *stored procedures*. Συγκεκριμένα, δημιουργούμε *stored procedures* με τα αντίστοιχα queries που θέλουμε να εκτελούν και στη συνέχεια για να τα μετατρέψουμε σε Web Service ακολουθούμε τα εξής βήματα:

1. Δημιουργούμε έναν υποφάκελο στον Web Server για να αποθηκεύσουμε το αρχείο WSDL και κάποια configuration files του Web Service.
2. Ανοίγουμε το γραφικό περιβάλλον του SQLXML 3.0, δημιουργούμε ένα virtual directory και δίνουμε κάποια στοιχεία που μας ζητούνται, όπως τη βάση για την οποία θέλουμε να δημιουργήσουμε το Web Service και στοιχεία authentication για σύνδεση με τη βάση.

3. Δίνουμε ένα καινούργιο virtual name τύπου SOAP και δίνουμε το ίδιο όνομα στο Web Service.
4. Αποθηκεύουμε το virtual directory και είμαστε έτοιμοι να το τροποποιήσουμε με το να διαλέξουμε τις stored procedures, που είχαμε δημιουργήσει και τις οποίες θέλουμε να κάνουμε expose σαν μεθόδους Web Service.
5. Τέλος, δίνουμε ένα όνομα σε κάθε μέθοδο καθώς και τις μεταβλητές εισόδου και το format των δεδομένων εξόδου π.χ. XML.

Μόλις αποθηκεύσουμε το virtual directory, το SQLXML 3.0 δημιουργεί ένα static WSDL file με το όνομα ServiceName.wsdl και ένα configuration file με το όνομα ServiceName.scc. Το configuration file συνδέει τις stored procedures του SQL Server με την αντίστοιχη μέθοδο του Web Service, ορίζει τον αριθμό των δεδομένων που θα επιστρέφει κάθε query και το output style και αριθμεί τις παραμέτρους εισόδου και εξόδου καθώς και τους αντίστοιχους τύπους τους.

Στη συνέχεια για να δημιουργήσουμε τον κώδικα ενός client, το SQLXML 3.0 δημιουργεί αυτόματα κάποια .NET objects, τα οποία θέλουν μόνο μερικές μικρές προγραμματιστικές προσθήκες για να λειτουργήσουν κανονικά.

Όλη αυτή η ευκολία βέβαια οφείλεται στο ότι τόσο ο SQL Server 2000 όσο και το .NET είναι προϊόντα της Microsoft και είναι λογικό να δημιουργηθούν εργαλεία για τον ευκολότερο προγραμματισμό των Web Services.

2.2 Στόχος

Μερικοί από τους στόχους που θέλαμε να ικανοποιεί η εφαρμογή μας είναι οι εξής:

1. Να είναι όσο πιο εύχρηστη γίνεται για κάποιον που θέλει να κάνει publish κάποια δεδομένα της βάσης τους ως Web Service. Να αρκείται δηλαδή ο κάτοχος της βάσης στο να διαλέγει τα δεδομένα που θέλει και τα αντίστοιχα operations που θα εκτελούν και στη συνέχεια να μην τον απασχολεί καθόλου το πώς θα δημιουργηθεί το Web Service.
2. Να έχει ένα απλό αλλά κατανοητό interface.
3. Να είναι *cross platform*, να μπορεί δηλαδή να τρέξει σε διαφορετικά λειτουργικά συστήματα όπως Windows, Linux κ.λ.π. . Για το λόγο αυτό χρησιμοποιήσαμε και ως γλώσσα προγραμματισμού την Java της εταιρείας Sun Microsystems.
4. Να χρησιμοποιεί *open source* κώδικα, τον οποίο να μπορεί κάποιος που ενδιαφέρεται να τροποποιήσει ή να φτιάξει μια παρόμοια εφαρμογή να κατεβάσει δωρεάν από το

Internet. Πάλι η Sun Microsystems και τα πακέτα που προσφέρει δωρεάν στο Internet για την κατασκευή Web Services ικανοποίησαν το στόχο αυτό.

5. Να υποστηρίζει παραπάνω από μία βάσεις.
6. Να είναι δωρεάν! Να μπορεί δηλαδή μια εταιρεία, που χρειάζεται ένα λογισμικό για να κάνει publish κάποια δεδομένα της ως Web Service να το αποκτήσει αφιλοκερδώς.

3

Θεωρητικό υπόβαθρο

3.1 XML

3.1.1 Εισαγωγή

Σε ένα κόσμο όπου οι πληροφορίες παρέχονται μέσω του παγκόσμιου διαδικτύου, τα έγγραφα πρέπει να είναι εύκολα προσβάσιμα, μεταφέρσιμα και ευέλικτα. Πρέπει επίσης να είναι ανεξάρτητα οποιουδήποτε συστήματος και περιεχομένου. Οι γενικευμένες γλώσσες έχουν τέτοια χαρακτηριστικά , παρέχοντας στα έγγραφα αυτά μια δυνατότητα η οποία δεν υπάρχει σε άλλες γλώσσες περιγραφής εγγράφων. Η HTML είναι προβληματική και περιοριστική γλώσσα. Η XML έλυσε πολλά από τα προβλήματα που αντιμετώπισαν οι σχεδιαστές του web και είναι υπεύθυνη για την XHTML, μια ανασχεδιασμένη HTML. Θα χρησιμοποιείται για πολλά χρόνια επειδή προσφέρει αποτελεσματικές και δυναμικές πολυμεσικές λύσεις.

Η XML σχεδιάστηκε να ικανοποιήσει πολλές ανάγκες δίνοντας στα έγγραφα ένα μεγαλύτερο επίπεδο προσαρμοστικότητας στο στυλ και τη δομή από αυτό που υπήρχε παλαιότερα στην HTML. Η XML προσφέρει στους σχεδιαστές της HTML τη δυνατότητα να προσθέτουν περισσότερα στοιχεία στη γλώσσα. Δεν αναφέρεται μονάχα στους σχεδιαστές του web αλλά σε οποιονδήποτε ασχολείται με εκδόσεις.

Στην πραγματικότητα, η XML είναι markup γλώσσα για έγγραφα που περιέχουν δομημένες πληροφορίες. Markup γλώσσα είναι ένας μηχανισμός που καθορίζει δομές σε ένα έγγραφο. Οι δομημένες πληροφορίες περιλαμβάνουν περιεχόμενο και κάποιες διευκρινίσεις για το ρόλο που παίζει το περιεχόμενο. Σχεδόν όλα τα έγγραφα έχουν την ίδια δομή.

3.1.2 Τι είναι η XML;

- Η XML είναι συντομογραφία για το E**X**tensible **M**arkup **L**anguage
- Η XML είναι **markup** γλώσσα, η οποία μοιάζει πολύ με την HTML
- Η XML σχεδιάστηκε για να **περιγράφει δεδομένα**
- Τα XML tags δεν είναι προκαθορισμένα. Πρέπει να ορίσουμε τα **δικά μας tags**
- Η XML χρησιμοποιεί το **Document Type Definition** (DTD) ή το **XML Schema** για να περιγράψει τα δεδομένα
- Ένα XML κείμενο με ένα DTD ή ένα XML Schema σχεδιάστηκε για να **περιγράφει επαρκώς τον εαυτό του**

3.1.3 Διαφορές μεταξύ XML και HTML

- **Η XML σχεδιάστηκε για τη μεταφορά δεδομένων.**
- Η XML δεν είναι ένα αντικατάστατο της HTML.
- Η XML σχεδιάστηκε για να **περιγράφει** τα δεδομένα και εστιάζει στο **τι είναι** τα δεδομένα.
- Η HTML σχεδιάστηκε για να **παρουσιάζει** τα δεδομένα και εστιάζει στο **πώς φαίνονται** τα δεδομένα.
- Η HTML έχει να κάνει με την παρουσίαση των δεδομένων ενώ η XML έχει να κάνει με την περιγραφή των δεδομένων.

3.1.4 Η XML δεν κάνει ΤΙΠΟΤΑ

Μπορεί να είναι λίγο δύσκολο να το καταλάβει κανείς, αλλά η XML δεν κάνει ΤΙΠΟΤΑ. Η XML σχεδιάστηκε για τη δόμηση, την αποθήκευση και την αποστολή δεδομένων.

Το παρακάτω παράδειγμα είναι ένα σημείωμα της Mary για τον John, αποθηκευμένο σε XML:

```
<note>
<to>John</to>
<from>Mary</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Το σημείωμα έχει μια κεφαλή (heading) και ένα σώμα (body), το οποίο περιέχει το μήνυμα. Περιέχει επίσης πληροφορίες για τον αποστολέα και τον αποδέκτη. Παρ' όλα αυτά όμως το XML κείμενο εξακολουθεί να μην κάνει ΤΙΠΟΤΑ. Περιέχει απλά πληροφορίες, τοποθετημένες μέσα σε XML tags. Κάποιος πρέπει να γράψει ένα κομμάτι software για να το στείλει, να το λάβει ή να το παρουσιάσει.

3.1.5 Η XML είναι απλή και επεκτάσιμη

Τα tags που χρησιμοποιούνται στα HTML κείμενα είναι προκαθορισμένα. Ο συγγραφέας ενός HTML κειμένου μπορεί να χρησιμοποιήσει μόνο τα tags που ορίζονται στο HTML standard (όπως <p>, <h1> κ.λ.π.).

Η XML επιτρέπει στο συγγραφέα ενός XML κειμένου να ορίσει τα δικά του tags. Για παράδειγμα τα tags <to> και <from> στο παραπάνω παράδειγμα δεν είναι ορισμένα σε κανένα XML standard. Αυτά τα tags «επινοήθηκαν» από το συγγραφέα αυτού του XML κειμένου.

3.1.6 Η XML είναι ένα συμπλήρωμα της HTML

Είναι σημαντικό να καταλάβει κανείς ότι η XML δεν αποτελεί ένα αντικατάστατο της HTML. Είναι πολύ πιθανόν ότι οι μελλοντικές Web εφαρμογές θα χρησιμοποιούν την XML για να περιγράφουν τα δεδομένα, ενώ η HTML θα χρησιμοποιείται για την απεικόνιση των δεδομένων αυτών.

Για αυτό και η καλύτερη περιγραφή της XML είναι η εξής: **η XML είναι ένα δια-πλατφορμικό, ανεξάρτητο από software και hardware εργαλείο για την μετάδοση πληροφοριών**.

3.1.7 Η XML στην ανάπτυξη μελλοντικών Web εφαρμογών

Είναι συναρπαστικό το πόσο γρήγορα το XML standard αναπτύχθηκε και πόσο γρήγορα ένας μεγάλος αριθμός από σχεδιαστές λειτουργικών συστημάτων υιοθέτησαν το standard αυτό.

Πιστεύεται ότι η XML θα είναι τόσο σημαντική για το μέλλον του Web όσο ήταν η HTML για τη θεμελίωση του. Πιστεύεται επίσης ότι η XML θα είναι το πιο κοινό εργαλείο για την επεξεργασία των δεδομένων και τη μετάδοση τους.

3.2 SOAP

3.2.1 Εισαγωγή

Το SOAP είναι ένα πρωτόκολλο σχεδιασμένο για την ανταλλαγή δομημένων (XML) δεδομένων. Είναι φτιαγμένο για να λειτουργεί με τη βοήθεια άλλων πρωτοκόλλων μεταφοράς δεδομένων και συνήθως υλοποιείται η λειτουργία του πάνω από το HTTP.

Πιο απλά: *Το SOAP είναι ένα πρωτόκολλο σχεδιασμένο για την πρόσβαση σε μία Web Service*.

3.2.2 Τι είναι το SOAP;

- Το SOAP είναι συντομογραφία για το **S**imple **O**bject **A**ccess **P**rotocol
- Το SOAP είναι ένα **communication protocol**
- Το SOAP είναι **σχεδιασμένο για την επικοινωνία μεταξύ εφαρμογών**
- Το SOAP είναι **σχεδιασμένο για την αποστολή μηνυμάτων**
- Το SOAP είναι **σχεδιασμένο για την επικοινωνία μέσω του Internet**
- Το SOAP είναι **platform independent**
- Το SOAP είναι **language independent**
- Το SOAP βασίζεται **στην XML**
- Το SOAP είναι **απλό και επεκτάσιμο**
- Το SOAP επιτρέπει **την μεταφορά μηνυμάτων σε firewalls**
- Το SOAP θα αναπτυχθεί σαν ένα **W3C standard**

3.2.3 Γιατί να χρησιμοποιήσει κανείς το SOAP;

Είναι πολύ σημαντικό η ανάπτυξη εφαρμογών να επιτρέπει την επικοινωνία των προγραμμάτων μέσα από το Internet. Οι σημερινές εφαρμογές επικοινωνούν μεταξύ τους χρησιμοποιώντας απομακρυσμένες κλήσεις μεθόδων (**Remote Procedure Calls** ή **RPC**), αλλά το HTTP δεν σχεδιάστηκε για κάτι τέτοιο. Το RPC εμπεριέχει 2 προβλήματα: ένα πρόβλημα συμβατότητας και ένα πρόβλημα ασφάλειας, αφού λογικά τα firewalls και οι proxy servers πρέπει να μπλοκάρουν αυτού του είδους την κίνηση.

Ένας καλύτερος τρόπος για την επικοινωνία μεταξύ εφαρμογών είναι πάνω από το HTTP, επειδή το HTTP υποστηρίζεται από όλους τους Internet browsers και servers. Το SOAP σχεδιάστηκε για να επιτύχει κάτι τέτοιο.Το SOAP προσφέρει έναν τρόπο για την επικοινωνία

μεταξύ εφαρμογών που τρέχουν σε διαφορετικά λειτουργικά συστήματα, με διαφορετικές τεχνολογίες και διαφορετικές προγραμματιστικές γλώσσες.

3.2.4 Ανάλυση ενός SOAP μηνύματος

Ένα μήνυμα SOAP είναι ένα συνηθισμένο έγγραφο XML, το οποίο περιέχει τα ακόλουθα στοιχεία (elements):

- Ένα Envelope element, το οποίο προσδιορίζει ότι το έγγραφο XML είναι ένα μήνυμα SOAP
- Ένα προαιρετικό Header element, το οποίο περιέχει πληροφορίες επικεφαλίδας
- Ένα υποχρεωτικό Body element, το οποίο περιέχει πληροφορίες κλήσεων και απαντήσεων (call and response information)
- Ένα προαιρετικό Fault element, το οποίο περιέχει πληροφορίες για λάθη που τυχόν εμφανίστηκαν κατά την επεξεργασία του μηνύματος

3.2.5 Κανόνες σύνταξης ενός SOAP μηνύματος

Παρακάτω αναφέρουμε μερικούς βασικούς κανόνες σύνταξης ενός μηνύματος SOAP:

- Ένα SOAP μήνυμα ΠΡΕΠΕΙ να γράφεται χρησιμοποιώντας XML
- Ένα SOAP μήνυμα ΠΡΕΠΕΙ να χρησιμοποιεί το SOAP Envelope namespace
- Ένα SOAP μήνυμα ΠΡΕΠΕΙ να χρησιμοποιεί το SOAP Encoding namespace
- Ένα SOAP μήνυμα ΔΕΝ πρέπει να χρησιμοποιεί αναφορά σε DTD
- Ένα SOAP μήνυμα ΔΕΝ πρέπει να περιέχει XML Processing Instructions

3.2.6 Παράδειγμα ενός SOAP μηνύματος

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
  ...
  ...
</soap:Header>
<soap:Body>
  ...
  ...
  <soap:Fault>
    ...
    ...
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

3.3 WSDL

3.3.1 Εισαγωγή

Η WSDL είναι μια γλώσσα, βασισμένη στην XML, που περιγράφει υπηρεσίες που παρέχονται στο δίκτυο (Web Services) και τους τρόπους για να έχει κάποιος πρόσβαση σε αυτές.

3.3.2 Τι είναι η WSDL;

- Η WSDL είναι συντομογραφία για το **W**eb **S**ervices **D**escription **L**anguage
- Η WSDL είναι γραμμένη σε **XML**
- Η WSDL είναι ένα **έγγραφο XML**
- Η WSDL χρησιμοποιείται για να περιγράφει **Web Services**
- Η WSDL χρησιμοποιείται για τον εντοπισμό των **Web Services**
- Η WSDL δεν είναι ακόμα ένα **W3C standard**

3.3.3 Δομή ενός κειμένου WSDL

Ένα κείμενο WSDL περιγράφει μια Web Service χρησιμοποιώντας τα παρακάτω σημαντικά στοιχεία (elements):

Element	Περιγράφει
<portType>	Τις υπηρεσίες που παρέχονται από το Web Service
<message>	Τα μηνύματα που χρησιμοποιούνται από το Web Service
<types>	Τους τύπους δεδομένων που χρησιμοποιούνται από το Web Service
<binding>	Τα πρωτόκολλα επικοινωνίας που χρησιμοποιούνται από το Web Service

Η δομή ενός κειμένου WSDL είναι μοιάζει ως εξής:

```
<definitions>
<types>
   definition of types........
</types>

<message>
   definition of a message....
</message>

<portType>
   definition of a port.......
</portType>

<binding>
   definition of a binding....
</binding>

</definitions>
```

Ένα κείμενο WSDL μπορεί επίσης να περιέχει και άλλα στοιχεία, όπως extension elements και ένα service element που κάνει δυνατή τη σύνδεση μεταξύ των ορισμών πολλαπλών Web Services σε ένα και μόνο WSDL κείμενο.

3.3.4 WSDL Ports

Το **<portType>** element είναι το πιο σημαντικό στοιχείο ενός WSDL κειμένου.

Περιγράφει μια Web Service, τις λειτουργίες (operations) που μπορούν να εκτελεστούν, καθώς και τα μηνύματα που ανταλλάσσονται.

Το **<portType>** element μπορεί να συγκριθεί με μια βιβλιοθήκη συναρτήσεων ή ένα module ή μια κλάση στις απλές προγραμματιστικές γλώσσες.

3.3.4.1 Operation Types

Ο τύπος request-response είναι ο πιο συνηθισμένος τύπος λειτουργίας, αλλά ένα WSDL κείμενο ορίζει 4 τύπους:

Τύπος	Ορισμός
One-way	Η λειτουργία λαμβάνει ένα μήνυμα αλλά δεν επιστρέφει απάντηση
Request-response	Η λειτουργία λαμβάνει μια αίτηση και επιστρέφει μια απάντηση
Solicit-response	Η λειτουργία στέλνει μια αίτηση και περιμένει μια απάντηση
Notification	Η λειτουργία στέλνει ένα μήνυμα αλλά δεν περιμένει μια απάντηση

3.3.4.2 Παράδειγμα λειτουργίας Request – Response

Ένα παράδειγμα λειτουργίας request-response φαίνεται παρακάτω:

```
<message name="getTermRequest">
   <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
   <part name="value" type="xs:string"/>
</message>
<portType name="glossaryTerms">
  <operation name="getTerm">
      <input message="getTermRequest"/>
      <output message="getTermResponse"/>
  </operation>
</portType>
```

Στο παράδειγμα αυτό το port “glossaryTerms” ορίζει μια λειτουργία request-response που λέγεται “getTerm”. Η λειτουργία “getTerm” δέχεται ένα μήνυμα που λέγεται “getTermRequest” με μια παράμετρο “term” και επιστρέφει ένα μήνυμα που λέγεται “getTermResponse” με μια παράμετρο “value”.

3.3.5 WSDL Messages

Το **<message>** element περιγράφει τους τύπους δεδομένων που χρησιμοποιεί μια operation.

Κάθε μήνυμα αποτελείται από ένα ή περισσότερα κομμάτια. Τα κομμάτια αυτά μπορούν να συγκριθούν με τις παραμέτρους μιας συνάρτησης σε μια παραδοσιακή προγραμματιστική γλώσσα.

3.3.6 WSDL Types

Το **<types>** element ορίζει τους τύπους δεδομένων που χρησιμοποιεί το Web Service.

Για μέγιστη συμβατότητα μεταξύ διαφορετικών πλατφόρμων, το WSDL κείμενο χρησιμοποιεί το XML Schema για να ορίσει τους τύπους δεδομένων.

3.3.7 WSDL Bindings

Το **<binding>** element ορίζει το format του μηνύματος και πληροφορίες για τα πρωτόκολλα σε κάθε port.

Ας δούμε ένα παράδειγμα μιας λειτουργίας request-response για να καταλάβουμε τη χρήση του binding.

```
<message name="getTermRequest">
   <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
   <part name="value" type="xs:string"/>
</message>
<portType name="glossaryTerms">
  <operation name="getTerm">
      <input message="getTermRequest"/>
      <output message="getTermResponse"/>
  </operation>
</portType>
<binding type="glossaryTerms" name="b1">
<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http" />
  <operation>
    <soap:operation
     soapAction="http://example.com/getTerm"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
```

Το **binding** element έχει 2 attributes – το όνομα και τον τύπο.

Το name attribute (μπορούμε να χρησιμοποιήσουμε ό,τι όνομα θέλουμε) καθορίζει το όνομα του binding και το type attribute δείχνει στο port του binding, στην περίπτωσή μας στο “glossaryTerms” port.

Το **soap:binding** element έχει 2 attributes – το style attribute και το transport attribute.

Το style attribute μπορεί να είναι “rpc” ή “document”. Στην περίπτωσή μας χρησιμοποιούμε το document. Το transport attribute καθορίζει το SOAP πρωτόκολλο που θα χρησιμοποιηθεί. Στην περίπτωσή μας χρησιμοποιούμε το HTTP.

Το **operation** element ορίζει κάθε λειτουργία (operation) που εκθέτει το port.

Για κάθε λειτουργία η αντίστοιχη SOAP ενέργεια πρέπει να οριστεί. Πρέπει επίσης να οριστεί και ο τρόπος που θα κωδικοποιηθεί η είσοδος (input) και η έξοδος (output). Στην περίπτωση μας χρησιμοποιούμε το “literal”.

3.3.8 Πλήρης σύνταξη ενός WSDL κειμένου

```
<wsdl:definitions name="nmtoken"? targetNamespace="uri">
    <import namespace="uri" location="uri"/> *
    <wsdl:documentation .... /> ?
    <wsdl:types> ?
        <wsdl:documentation .... /> ?
        <xsd:schema .... /> *
    </wsdl:types>
    <wsdl:message name="ncname"> *
        <wsdl:documentation .... /> ?
        <part name="ncname" element="qname"? type="qname"?/> *
    </wsdl:message>
    <wsdl:portType name="ncname"> *
        <wsdl:documentation .... /> ?
        <wsdl:operation name="ncname"> *
            <wsdl:documentation .... /> ?
            <wsdl:input message="qname"> ?
                <wsdl:documentation .... /> ?
            </wsdl:input>
            <wsdl:output message="qname"> ?
                <wsdl:documentation .... /> ?
            </wsdl:output>
            <wsdl:fault name="ncname" message="qname"> *
                <wsdl:documentation .... /> ?
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:serviceType name="ncname"> *
        <wsdl:portType name="qname"/> +
    </wsdl:serviceType>
    <wsdl:binding name="ncname" type="qname"> *
        <wsdl:documentation .... /> ?
        <-- binding details --> *
        <wsdl:operation name="ncname"> *
            <wsdl:documentation .... /> ?
            <-- binding details --> *
            <wsdl:input> ?
                <wsdl:documentation .... /> ?
                <-- binding details -->
            </wsdl:input>
            <wsdl:output> ?
                <wsdl:documentation .... /> ?
                <-- binding details --> *
            </wsdl:output>
            <wsdl:fault name="ncname"> *
                <wsdl:documentation .... /> ?
                <-- binding details --> *
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="ncname" serviceType="qname"> *
        <wsdl:documentation .... /> ?
        <wsdl:port name="ncname" binding="qname"> *
            <wsdl:documentation .... /> ?
            <-- address details -->
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

3.4 JAVA

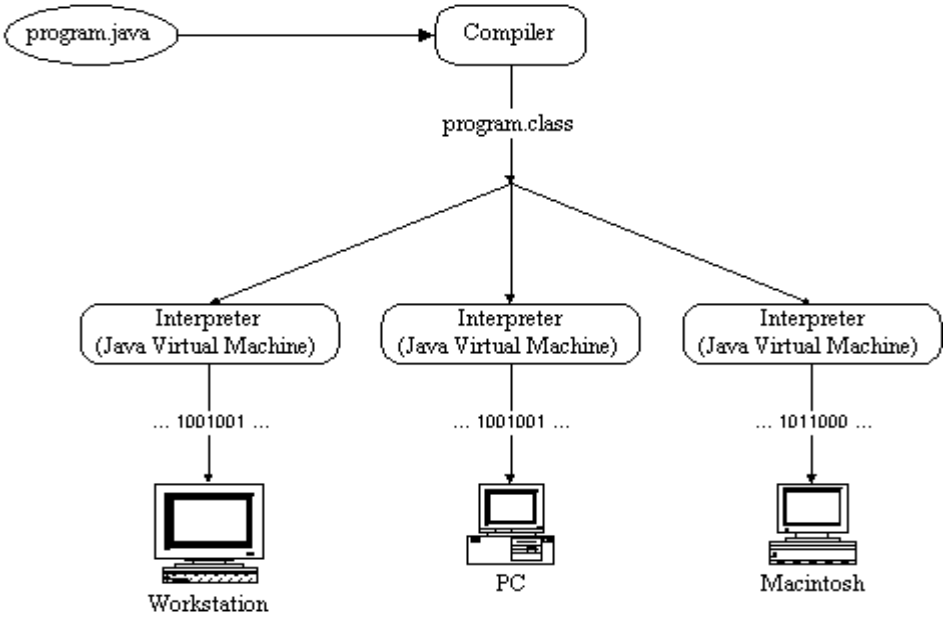
Η Java είναι μία γλώσσα προγραμματισμού υψηλού επιπέδου. Αναπτύχθηκε από την εταιρεία Sun microsystems και γρήγορα έλαβε μεγάλη δημοσιότητα λόγω της ανεπανάληπτης χρησιμότητάς της στην ανάπτυξη (κυρίως Web) εφαρμογών. Χαρακτηρίζεται από τα εξής : απλή (simple), αντικειμενοστραφής (object-oriented), κατανεμημένη (distributed), εύρωστη (robust), μεταφέρσιμη (portable), υψηλής απόδοσης (high performance), πολυνηματώδης (multithreaded), ασφαλής (secure) κ.ά.

Με την Java γράφονται δύο ειδών προγράμματα :

- Ανεξάρτητες εφαρμογές (applications) που τρέχουν σε οποιαδήποτε πλατφόρμα που περιέχει την JVM.

- Εφαρμογές που τρέχουν στους Web browsers και ονομάζονται applets. Καθώς ένας browser μεταφράζει μία HTML σελίδα, μόλις συναντήσει το tag της HTML <applet> ανακτά από τον server ένα πρόγραμμα Java (ήδη μεταφρασμένο) και μέσω του interpreter του browser το μετατρέπει σε bytecodes και κατόπιν το εκτελεί. Έτσι οι HTML σελίδες παύουν να είναι πλέον στατικές και αποκτούν δυναμικότητα στον υπολογιστή του browser, εκτελώντας σ' αυτόν πλήθος χρήσιμων λειτουργιών που θα ήταν αδύνατο να επιτευχθούν διαφορετικά.

Με τον compiler της Java το πρόγραμμα μετατρέπεται σε μια ενδιάμεση μορφή κώδικα που ονομάζεται Java bytecode. Έπειτα με τον interpreter της Java κάθε εντολή του προγράμματος σε Java bytecode μεταφράζεται σε γλώσσα μηχανής και κατόπιν τρέχει σε υπολογιστή οποιασδήποτε πλατφόρμας. Ο interpreter αυτός της Java ονομάζεται Java Virtual Machine.



Η JVM επιτρέπει στα προγράμματα Java το πολύτιμο χαρακτηριστικό “write once, run anywhere”. Αυτό είναι κι ένα από τα σημαντικότερα πλεονεκτήματα της Java έναντι των άλλων γλωσσών προγραμματισμού. Ο προγραμματιστής γράφει το πρόγραμμα και το μεταφράζει (compile) σε Java bytecodes σε οποιονδήποτε υπολογιστή που περιέχει Java compiler. Έπειτα οι Java bytecodes εκτελούνται σε οποιονδήποτε υπολογιστή που περιέχει την JVM (Java interpreter).

Επιπλέον η Java περιέχει τα λεγόμενα Java APIs (Application Programming Interfaces). Ένα Java API είναι μία συλλογή από έτοιμα προγράμματα. Το κάθε API παρέχει μία ολόκληρη βιβλιοθήκη με κλάσεις που καλύπτουν τις προγραμματιστικές ανάγκες για ένα συγκεκριμένο πεδίο εφαρμογών. Για παράδειγμα υπάρχουν τα παρακάτω APIs : AWT API (Abstract Window Toolkit, για graphical User Interfaces), JDBC API (Java DataBase Connectivity) για σύνδεση με βάσεις δεδομένων), Commerce API, Security API, Management API, RMI API (Remote Method Invocation), JavaBeans API, Servlet API (για κατασκευή εφαρμογών που τρέχουν αντί στον browser ,όπως κάνουν τα applets, στον Web Server) και πολλά άλλα.

3.5 Εισαγωγή στο JDBC API

Ο προγραμματισμός εφαρμογών για πρόσβαση σε βάσεις δεδομένων πάντα αποτελούσε για τον developer μία δύσκολη υπόθεση. Υπάρχουν εκατοντάδες προϊόντα (programming interfaces) που επιτρέπουν την πρόσβαση των εφαρμογών στις βάσεις δεδομένων. Το μειονέκτημά τους όμως είναι ότι το καθένα απ’ αυτά μιλάει στην εφαρμογή του προγραμματιστή τη δική του γλώσσα. Επομένως κάθε φορά που ο προγραμματιστής επιχειρεί τη σύνδεση της εφαρμογής του με μια βάση διαφορετικού τύπου (π.χ. SQL Server, Oracle) είναι αναγκασμένος να γράψει από την αρχή την εφαρμογή του έτσι ώστε αυτή να «μιλάει» το καινούριο interface προς την βάση. Κάτι τέτοιο όμως είναι αρκετά κοπιαστικό και αποσπά τον developer από το κυρίως έργο του. Την απάντηση σ’ αυτό το πρόβλημα έρχεται να δώσει η Java και μάλιστα με τρόπο επαναστατικό. Πρόκειται για το JDBC API που η Javasoft (της Sun Microsystems) περιέλαβε στα APIs της πλατφόρμας προγραμματισμού Java. Το JDBC API επαληθεύει τον ισχυρισμό της Java : “write once, compile once, run everywhere”.

Το JDBC API (Java DataBase Connectivity) είναι ένα από τα ισχυρότερα και πιο ολοκληρωμένα APIs της Java. Παρέχει στον προγραμματιστή τη δυνατότητα να συνδέσει την εφαρμογή του με βάσεις δεδομένων διαφόρων τύπων χωρίς να χρειαστεί να τροποποιήσει το πρόγραμμά του κάθε φορά που συνδέει την εφαρμογή του σε μια διαφορετική βάση. Παρεμβάλλει και ενεργεί ως middleware ανάμεσα στις Java εφαρμογές και τις σχεσιακές (relational) βάσεις. Το JDBC είναι ένα interface για πρόσβαση σε βάσεις δεδομένων (database access interface) που χρησιμοποιεί standard SQL ερωτήσεις. Προσφέρει στον

προγραμματιστή την άνεση να γράφει ένα πρόγραμμα που να στέλνει μια SQL ερώτηση σε μια σχεσιακή βάση δεδομένων είτε πρόκειται για SQL Server είτε για Oracle είτε για Access είτε για οποιαδήποτε άλλη πλατφόρμα. Η μόνη φροντίδα του προγραμματιστή είναι να επιλέγει κάθε φορά που συνδέεται σε μια βάση τον κατάλληλο driver για να καταλαβαίνει το JDBC τι είδους βάση πρόκειται να προσπελάσει ώστε να φορτώνει τις απαραίτητες ρουτίνες.

Με το JDBC API ο προγραμματιστής της εφαρμογής μπορεί να ρωτήσει την βάση οποιαδήποτε SQL ερώτηση επιθυμεί, να πάρει μεταπληροφορία από την βάση δεδομένων και όχι μόνον να παραλάβει τα αποτελέσματα από την βάση αλλά και να τα τροποποιήσει και να τα μορφοποιήσει όπως αυτός θέλει.

Επιπλέον το JDBC API περιέχει το JDBC–ODBC bridge (πρόκειται για κοινό προϊόν της Javasoft με την Intersolv). Αφορά ένα σύνολο προγραμμάτων που περιέχει το JDBC API τα οποία επιτρέπουν τη σύνδεση και την επικοινωνία των εφαρμογών που γράφει ο προγραμματιστής με οποιαδήποτε βάση δεδομένων που μπορεί να προσπελαστεί με τον οδηγό ODBC (Open DataBase Connectivity) της Microsoft. Η γέφυρα αυτή προσδίδει στο JDBC τη δυνατότητα να χρησιμοποιήσει ένα μέρος από τις λειτουργίες του ODBC προκειμένου η εφαρμογή να επικοινωνήσει με την βάση που χρησιμοποιεί τον οδηγό ODBC.

Η Java εφαρμογή μπορεί να επικοινωνήσει με οποιαδήποτε σχεσιακή βάση που έχει αναπτύξει για λογαριασμό της κάποιον JDBC driver. Επιπροσθέτως μπορεί να επικοινωνήσει και με βάσεις που δεν παρέχουν κάποιον JDBC driver, αρκεί μόνον να παρέχουν κάποιον ODBC driver. Τότε το JDBC χρησιμοποιεί τις ρουτίνες του JDBC- ODBC bridge και επιτυγχάνει τη σύνδεση. Φαίνεται έτσι η καταπληκτική ευελιξία που χαρακτηρίζει το JDBC API.

Η μεγάλη επιτυχία και λειτουργικότητα του JDBC οδηγεί καθημερινά ένα μεγάλο αριθμό κατασκευαστριών εταιριών συστημάτων client-server, database και middleware να υιοθετήσουν και να ενσωματώσουν την τεχνολογία JDBC στα προϊόντα τους.

3.6 WEB SERVICES

3.6.1 Τι είναι οι Web Services;

Ένα από τα παράξενα με τις Web Services (συνδυασμένες υπηρεσίες Ιστού) είναι η ίδια η ονομασία τους. Η λέξη services μας φέρνει στο μυαλό την εικόνα ανθρώπων που πληρώνουν για κάτι και παίρνουν κάτι πίσω. Αλλά στο πιο βασικό επίπεδο, οι Web Services είναι απλά μια νέα γεύση από τεχνολογίες λογισμικού βασισμένες σε κάποια standards, που επιτρέπει στους προγραμματιστές να συνδυάσουν υπάρχουσα συστήματα υπολογιστών με νέους τρόπους μέσω του Internet, μέσα σε μια επιχείρηση ή μεταξύ πολλών.

Οι Web Services επιτρέπουν στις επιχειρήσεις να γεφυρώσουν επικοινωνιακά χάσματα – μεταξύ λογισμικών γραμμένων σε διαφορετικές προγραμματιστικές γλώσσες, από διαφορετικούς προμηθευτές που τρέχουν σε διαφορετικά λειτουργικά συστήματα. Και εδώ είναι ο μεγάλος στόχος: οι Web Services στην τελειοποιημένη μελλοντική τους μορφή θα επιτρέπουν τέτοιου είδους επικοινωνίες να συνεχιστούν *χωρίς ανθρώπους*. Κάτι σαν υπερ-αυτοματοποίηση!

3.6.2 Μιλάμε δηλαδή για υπολογιστές που καταλαμβάνουν τον κόσμο;

Ε όχι ακριβώς! Μιλάμε για υπολογιστές, οι οποίοι θα μπορούν να κάνουν συναλλαγές και επιχειρησιακές συνεργασίες μεταξύ τους χωρίς να χρειάζεται μεσολάβηση από κάποιον άνθρωπο. Ας υποθέσουμε ότι έχουμε ένα κατάστημα που σερβίρει πρωινό και έχουμε παράλληλα μια ιστοσελίδα, στην οποία περιέχεται και ο τιμοκατάλογος του καταστήματος αυτού. Έστω τώρα ότι θέλουμε η ιστοσελίδα μας να ελέγχει την τιμή της μοτσαρέλας σε μια συνεχή και καθημερινή βάση. Αν υποθέσουμε τώρα ότι αυτό ήταν κάτι σημαντικό για μια επιχείρηση, τότε σίγουρα θα υπήρχαν προμηθευτές μοτσαρέλας, οι οποίοι θα προσλάμβαναν ανθρώπους για να στήσουν Web Services που θα παρέχουν on-line τις τιμές τους. Εν τω μεταξύ, και εμείς σαν επιχείρηση θα προσλαμβάναμε ανθρώπους, οι οποίοι θα έφτιαχναν μια Web Service εφαρμογή για την ιστοσελίδα μας, η οποία θα είχε ως λειτουργία να βρίσκει αυτόματα όχι μόνο εκείνες τις επιχειρήσεις μοτσαρέλας που παρέχουν on-line τις τιμές των προϊόντων τους, αλλά και εκείνη την επιχείρηση με τις φτηνότερες τιμές. Έτσι χωρίς εμείς να χρειαστεί να παρέμβουμε καθόλου, πετυχαίνουμε η ιστοσελίδα μας να ανανεώνεται κατάλληλα κάθε μέρα!

3.6.3 Ποιες είναι οι τεχνολογίες που χρησιμοποιούν οι Web Services;

Οι Web Services δεν χρησιμοποιούν κάποια συγκεκριμένη τεχνολογία, αλλά ένα σύνολο από καθιερωμένα πρωτόκολλα επικοινωνίας, που περιλαμβάνουν το HTTP, την XML, το SOAP, το UDDI και το WSDL. Μια Web service μπορεί να αναπτυχθεί σε κάθε υπολογιστική πλατφόρμα και σε κάθε αναπτυξιακό περιβάλλον, αρκεί να επικοινωνεί με τις υπόλοιπες web services χρησιμοποιώντας τα παραπάνω πρωτόκολλα.

3.6.4 Σε τι διαφέρουν οι Web Services από παλαιότερα μοντέλα κατανεμημένου προγραμματισμού;

Οι web services έχουν πάρει πολλά στοιχεία από παλιότερα μοντέλα κατανεμημένου προγραμματισμού, όπως η CORBA και άλλες. Οι βασικές όμως διαφορές των web services είναι 2:

1. Είναι ελαφρά ορισμένες και συνδεδεμένες μεταξύ τους
2. Είναι χτισμένες πάνω σε υπάρχοντα πρωτόκολλα όπως το HTTP και η XML.

Και ένας βασικός λόγος για τον οποίο οι Web Services αναμένεται να επιτύχουν είναι γιατί οι εμπνευστές τους έχουν επικεντρωθεί στο να τις κάνουν όσο πιο απλές και φιλικές γίνεται.

3.6.5 Είναι οι Web Services μια παραλλαγή του Application Service Provider (ASP) μοντέλου;

Παρότι και οι Web Services και τα ASPs εφαρμόζουν την έννοια του «λογισμικό ως υπηρεσία», οι ομοιότητες τελειώνουν εκεί. Τα ASPs παραδίδουν ολόκληρες εφαρμογές από μια κεντρική θέση φιλοξενίας (cental hosting location), ενώ οι web services περιέχουν κατανεμημένα συστατικά. Τα ASPs είναι κάτι σαν «μαύρα κουτιά» ενώ οι Web Services είναι εγγενώς επεκτάσιμες. Τα ASPs είναι τόσο επιχειρησιακό πρότυπο όσο και λύση τεχνολογίας. Οι Web Services μπορούν από τη μια να επιτρέψουν νέες μορφές επιχειρησιακών προτύπων, αλλά από την άλλη είναι πλήρως μια λύση τεχνολογίας.

3.6.6 Ποιος ανέπτυξε τα πρότυπα για τις Web Services;

Μερικά από τα πρωτόκολλα που χρησιμοποιούν οι web services έχουν τυποποιηθεί από ανεξάρτητες οργανώσεις, όπως η W3C. Το HTTP και η XML αποτελούν τη βάση πάνω στην οποία οι DevelopMentor, UserLand Software και η Microsoft ανέπτυξαν το SOAP. Μετά τον αρχικό ορισμό των προδιαγραφών του SOAP, η IBM και η Ariba συνεργάστηκαν με την Microsoft για να αναπτύξουν το UDDI. Η IBM επίσης εργάστηκε πάνω στην ανάπτυξη και άλλων προδιαγραφών για τις web services, όπως των WSDL, WSFL και άλλων.

4

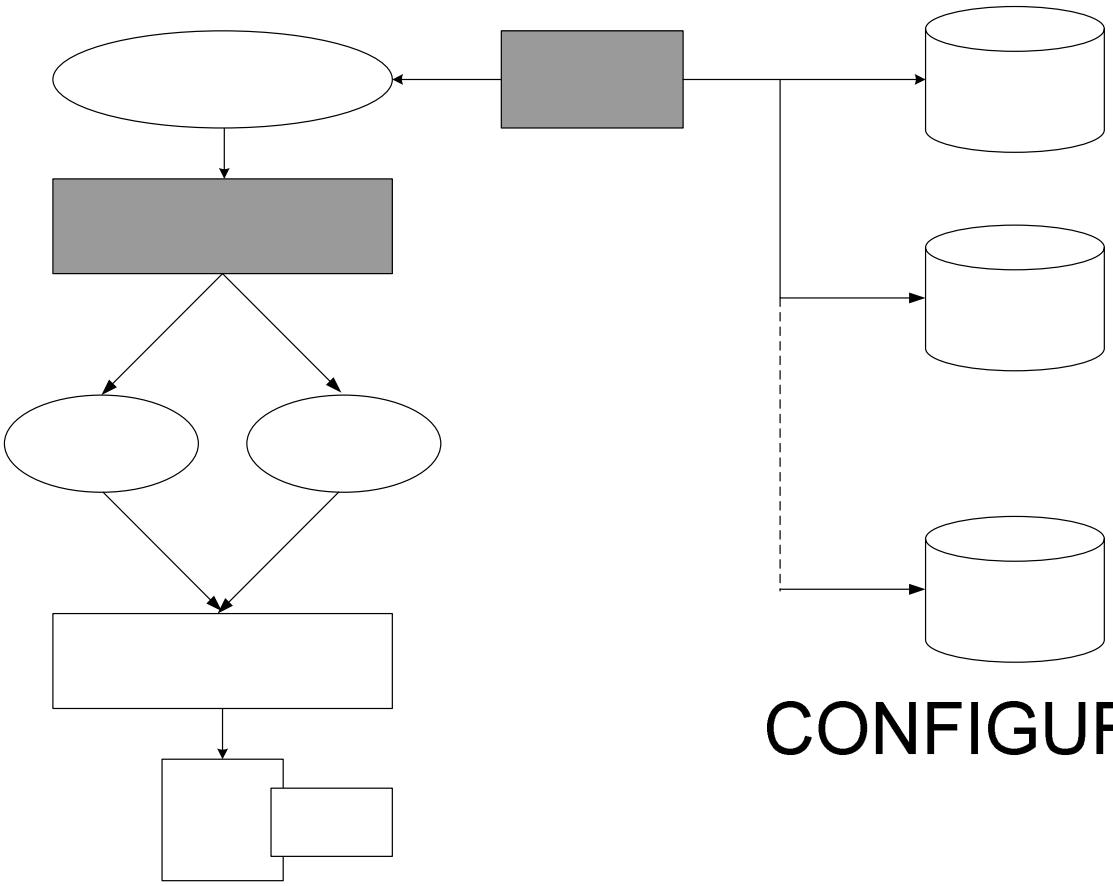
Ανάλυση και σχεδίαση

Στο παρόν κεφάλαιο θα παρουσιαστεί η ανάλυση και ο σχεδιασμός του λογισμικού του συστήματος μας. Στην παράγραφο 4.1 γίνεται η αρχιτεκτονική περιγραφή του συστήματος, όπου θα προσδιοριστούν και θα περιγραφούν τα λογικά επιμέρους «κομμάτια» που το αποτελούν. Στη συνέχεια, στην παράγραφο 4.2, στην περιγραφή των λειτουργιών του συστήματος, θα αναλυθούν και θα περιγραφούν με λεπτομέρεια οι δυνατότητες τις οποίες παρέχει στους χρήστες του.

4.1 Περιγραφή Αρχιτεκτονικής

Όπως είχαμε πει στο Κεφάλαιο 2, μερικοί από τους στόχους που είχαμε θέσει για την εφαρμογή μας είναι να μπορεί να συνδέεται με πολλές βάσεις, να χρησιμοποιεί open source κώδικα (συγκεκριμένα το πακέτο JWSDP της Sun Microsystems) και τελευταίο και κυριότερο το να μην απασχολεί καθόλου τον κάτοχο της βάσης το πώς θα δημιουργηθεί το Web Service. Αυτός θα πρέπει να αρκείται στο να διαλέγει με κάποιο τρόπο (π.χ. με SQL statements) τα δεδομένα της βάσης που θέλει να κάνει publish και στη συνέχεια η Web Service να κατασκευάζεται αυτόματα.

Έχοντας αυτά στο μυαλό μας σχεδιάσαμε την εφαρμογή, το design κομμάτι της οποίας φαίνεται στο παρακάτω σχήμα:

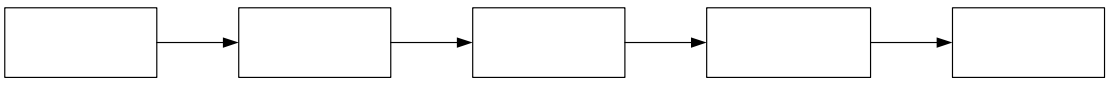


Τα κουτάκια με το γκρι χρώμα αποτελούν το δικό μας λογισμικό ενώ τα κουτάκια με το άσπρο χρώμα αποτελούν έτοιμο λογισμικό (συγκεκριμένα πρόκειται για το JWSDP). Τα κυλινδρικά κουτάκια αποτελούν τα αρχεία που παράγονται σε κάποια στάδια της εκτέλεσης του προγράμματος και τα οποία χρησιμοποιούνται σαν είσοδο σε κάποια άλλα στάδια.

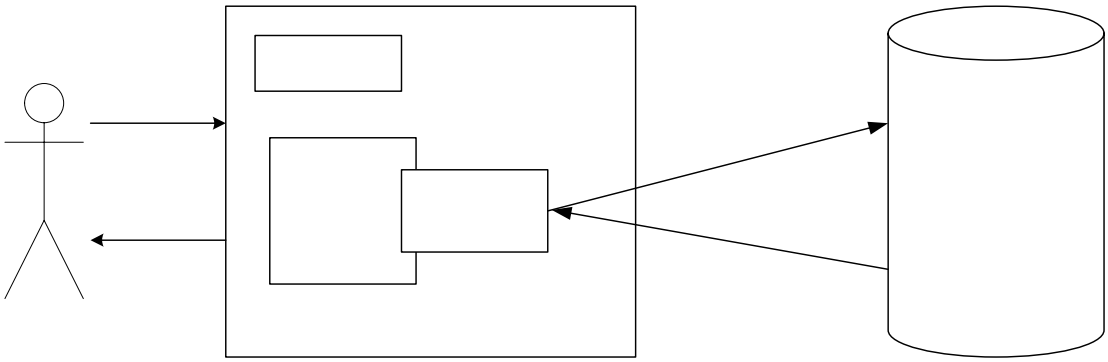
Πιο αναλυτικά, η εφαρμογή μας κάνει τα εξής:

1. Ο χρήστης συνδέεται με το Σύστημα Διαχείρισης Βάσεων Δεδομένων και επιλέγει τη βάση που θέλει από τη λίστα με τις υπάρχουσες βάσεις.
2. Αφού έχει επιλέξει την κατάλληλη βάση, μπορεί να δει όλους τους πίνακες της βάσης αυτής και επιλέγοντας κάποιο πίνακα μπορεί να δει όλα τα πεδία και τους αντίστοιχους τύπους του πίνακα αυτού. Έχοντας λοιπόν την εικόνα της βάσης καλείται να δώσει τα συστατικά του κάθε operation του Web Service. Συγκεκριμένα καλείται να δώσει τα queries που θα εκτελεί το operation, τις μεταβλητές εισόδου του operation, καθώς και το όνομα του operation.
3. Αφού ο χρήστης περιγράψει και το τελευταίο operation, καλείται να δώσει κάποιες επιπλέον πληροφορίες που περιγράφουν το web service, όπως τη διεύθυνση που θέλει να γίνει deploy το service, το όνομα του service και άλλες.
4. Μόλις δώσει τις πληροφορίες αυτές παράγεται ένα configuration file (σε XML), το οποίο περιγράφει πλήρως το Web Service περιέχοντας όλες τις πληροφορίες που έδωσε ο χρήστης στα παραπάνω βήματα.
5. Ο χρήστης τώρα είναι έτοιμος να πατήσει το κουμπί deploy για να κάνει deploy το Web Service. Συγκεκριμένα, μόλις ο χρήστης πατήσει το κουμπί deploy η εφαρμογή μας διαβάζει το configuration file και παράγει όλα τα απαραίτητα αρχεία java (όπως τα beans που θα χρειαστούν και τον wrapper) καθώς και κάποια properties files. Τα αρχεία αυτά τα χρησιμοποιεί στη συνέχεια το JSWDP για να κάνει deploy το Web Service.

Αν θελήσουμε τώρα να δούμε και γραφικά τη ροή της εκτέλεσης της εφαρμογής μας έχουμε το παρακάτω flow chart:



Σχεδιάζουμε τώρα το runtime κομμάτι της εφαρμογής μας:



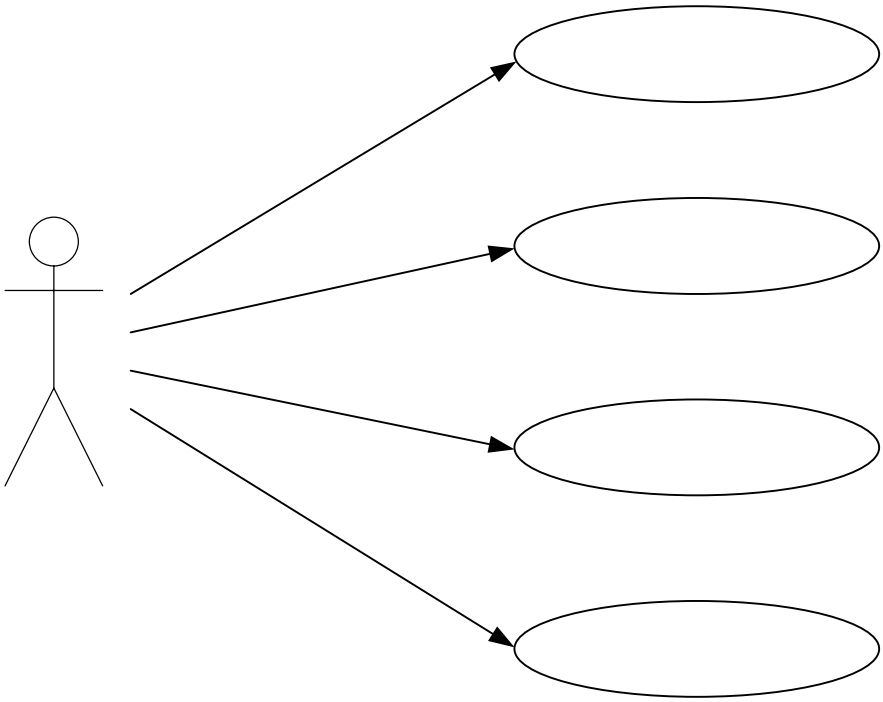
Συγκεκριμένα, ένας χρήστης που θέλει να χρησιμοποιήσει το Web Service μας ακολουθεί τα παρακάτω βήματα:

1. Επιλέγει κάποιο operation και δίνει τις μεταβλητές εισόδου που του ζητάει το operation αυτό.
2. Ο wrapper του web service παίρνει τις μεταβλητές αυτές και εκτελεί την κατάλληλη λειτουργία που ορίζει το operation (π.χ. μια λειτουργία θα ήταν ένα select από τη βάση ή ένα update).
3. Το αποτέλεσμα της λειτουργίας αυτής επιστρέφεται από τη βάση. Το αποτέλεσμα αυτό σε περίπτωση ενός select query θα είναι ένας πίνακας με τα στοιχεία που ζητάει το query ενώ σε οποιαδήποτε άλλη περίπτωση (update, insert, delete) θα είναι ένας integer που θα δηλώνει αν η ενέργεια ήταν επιτυχής ή όχι.
4. Ο χρήστης παίρνει την απάντηση.

4.2 Περιγραφή Λειτουργιών

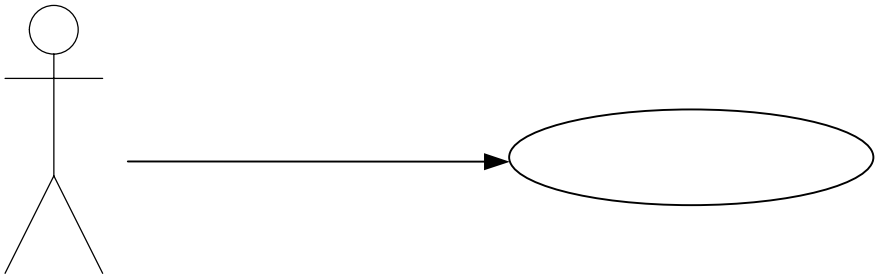
Οι λειτουργίες της εφαρμογής μας χωρίζονται σε δύο κατηγορίες. Στις λειτουργίες που μπορεί να εκτελέσει ο σχεδιαστής του Web Service και στις λειτουργίες που μπορεί να εκτελέσει ένας χρήστης που θέλει να χρησιμοποιήσει το Web Service.

Οι λειτουργίες που μπορεί να εκτελέσει ο σχεδιαστής του Web Service φαίνονται στο παρακάτω διάγραμμα:



Όπως βλέπουμε, κατά τη σχεδίαση του Web Service ο χρήστης συνδέεται αρχικά με βάση (λειτουργία 1), κατόπιν δίνει τα queries για κάθε operation του web service (λειτουργία 2), δίνει κάποιες επιπλέον πληροφορίες που περιγράφουν το Web Service (λειτουργία 3) και τέλος κάνει deploy το Web Service (λειτουργία 4).

Από την άλλη πλευρά, ένας χρήστης που θέλει να χρησιμοποιήσει το Web Service έχει μόνο μία λειτουργία, η οποία φαίνεται στο παρακάτω διάγραμμα:



Ο χρήστης λοιπόν το μόνο που έχει να κάνει είναι να καλέσει το Web Service δίνοντας τις μεταβλητές εισόδου που του ζητούνται και στη συνέχεια παίρνει τα αποτελέσματα που ζήτησε.

5

Υλοποίηση

Στην ενότητα αυτή περιγράφονται τα χαρακτηριστικά της συγκεκριμένης υλοποίησης, όπως η πλατφόρμα ανάπτυξης και εκτέλεσης, τα προγραμματιστικά εργαλεία που επιλέχθηκαν, και παρατίθενται οι λεπομέρειες της υλοποίησης.

5.1 Πλατφόρμες και προγραμματιστικά εργαλεία

5.1.1 Επιλογή γλώσσας προγραμματισμού

Η υλοποίηση του συστήματος έγινε αποκλειστικά σε JAVA 2. Συγκεκριμένα, χρησιμοποιήθηκε το Java 2 SDK Standard Edition v1.4.1. Η επιλογή της συγκεκριμένης γλώσσας στηρίχθηκε στα παρακάτω χαρακτηριστικά της:

- Είναι αντικειμενοστραφής. Η Java – σε αντίθεση με τη C++ που περιέχει και μη αντικειμενοστραφή χαρακτηριστικά – είναι μια καθαρά αντικειμενοστραφής γλώσσα προγραμματισμού. Το γεγονός αυτό διευκολύνει την ανάπτυξη επαναχρησιμοποιήσιμου κώδικα. Η ίδια η Java παρέχει ένα σημαντικό αριθμό κλάσεων που επιταχύνουν την ανάπτυξη κώδικα.

- Παράγει κώδικα μεταφέρσιμο σε οποιαδήποτε αρχιτεκτονική. Ο μεταγλωττιστής της Java δεν παράγει κώδικα για την αρχιτεκτονική ενός συγκεκριμένου συστήματος,

αλλά για την εικονική μηχανή της Java (Java Virtual Machine). Μια εφαρμογή λοιπόν αυτής της γλώσσας μπορεί να τρέξει σε οποιαδήποτε πλατφόρμα, χωρίς καμιά μετατροπή. Επομένως, ο κώδικας δημιουργείται αλλά και μεταγλωττίζεται μονάχα μια φορά.

- Παράγει σταθερό αλλά και αξιόπιστο κώδικα. Η Java περιορίζει τα προγραμματιστικά λάθη που μπορεί να οδηγήσουν σε εφαρμογές ασταθείς και αναξιόπιστες. Παρέχει μηχανισμούς ελέγχου λαθών και παραβάσεων μνήμης κατά την εκτέλεση (run-time errors), που θα μπορούσαν να οδηγήσουν σε αστάθεια το σύστημα, φροντίζει για την αποδέσμευση της μνήμης (ύπαρξη garbage collector), αφαιρώντας από τον προγραμματιστή αυτή την ευθύνη, ενώ τέλος ο χειρισμός των εξαιρέσεων απλοποιεί τη διαχείριση λαθών και την ανάνηψη από αυτά. Η έλλειψη δεικτών, χωρίς να μειώνει τη δύναμή της, απλοποιεί την ανάπτυξη του κώδικα και απαλλάσει τις εφαρμογές από τις απρόβλεπτες καταστάσεις που εμφανίζεται σε προγράμματα C και C++.

- Είναι δυναμική και κατανεμημένη. Χαρακτηρίζοντας την Java δυναμική, αναφερόμαστε στη δυνατότητα δυναμικής φόρτωσης μιας κλάσης στον Java μεταφραστή οποιαδήποτε χρονική στιγμή κατά την εκτέλεση. Επίσης, μας δίνει τη δυνατότητα να αποκτήσουμε δυναμικά πληροφορίες για μια κλάση στο χρόνο εκτέλεσης. Η Java χαρακτηρίζεται, επίσης, κατανεμημένη, χάρη στην υποστήριξη που παρέχει σε επίπεδο δικτύου. Εδώ πρέπει να αναφερθούμε στο Remote Method Invocation (RMI) Application Interface (API), το οποίο επιτρέπει σε ένα Java πρόγραμμα να καλέσει μεθόδους απομακρυσμένων αντικειμένων σαν να ήταν τοπικά.

- Περιλαμβάνει ή υποστηρίζει όλα τα απαραίτητα APIs για την πραγματοποίηση των λειτουργιών που χρειάστηκαν (JDBC, Swing, Servlets, JAXP). Το γεγονός αυτό αφενός διευκολύνει τον προγραμματιστή, αφού του παρέχει ένα βασικό υπόβαθρο, πάνω στο οποίο θα δημιουργήσει τις δικές του εφαρμογές, αφετέρου ελαχιστοποιεί τις πιθανότητες κακής επικοινωνίας μεταξύ των τμημάτων του κώδικα. Ταυτόχρονα, οδηγεί σε ευανάγνωστο κώδικα και σίγουρα στην καλύτερη και βαθύτερη εκμάθηση του προγραμματιστικού περιβάλλοντος.

- Συνοδεύεται από πλήρη και κατατοπιστική τεκμηρίωση (documentation), η οποία παρέχεται δωρεάν μέσω του World Wide Web.

5.1.2 Επιλογή λειτουργικού συστήματος

Έχοντας να δουλέψουμε με τη γλώσσα Java είχαμε την επιλογή να αναπτύξουμε τον κώδικά μας σε οποιοδήποτε περιβάλλον και λειτουργικό σύστημα θέλαμε. Επιλέχθηκαν τα Windows 2000 της εταιρίας Microsoft λόγω της συμβατότητας που προσφέρουν, αφού αποτελούν μακράν την πιο διαδεδομένη πλατφόρμα, αλλά και λόγω της πληθώρας των προγραμμάτων και εργαλείων για ανάπτυξη κώδικα που διατίθενται σε αυτά.

Ο παραγόμενος κώδικας βέβαια είναι πολύ εύκολο να χρησιμοποιηθεί και σε άλλα λειτουργικά συστήματα, αρκεί βέβαια να είναι εγκατεστημένο το Java Runtime Environment 1.4.1.

5.1.3 Επιλογή πακέτου για τη δημιουργία Web Services

Ως πακέτο για τη δημιουργία Web Services χρησιμοποιήσαμε το Java Web Services Developer Pack (JWSDP) της εταιρεία Sun Microsystems. Περιλαμβάνει ένα πλήθος τεχνολογιών, οι οποίες κάνουν πολύ πιο εύκολη και γρήγορη την κατασκευή Web Services, χρησιμοποιώντας την Java.

Το βασικό API του JWSDP είναι το JAX-RPC, το οποίο είναι μια συντομογραφία του Java API for XML-based RPC. Το API αυτό χρησιμοποιείται για την κατασκευή τόσο των Web Services όσο και των clients που τις καλούν χρησιμοποιώντας *remote procedure calls* (RPC) και XML.

Το JAX-RPC χρησιμοποιεί το SOAP σαν το πρωτόκολλο μεταφοράς μηνυμάτων. Παρόλο που το JAX-RPC στηρίζεται σε αρκετά πολύπλοκα πρωτόκολλα, το API του κρύβει αυτή την πολυπλοκότητα από τον σχεδιαστή μιας εφαρμογής. Από τη μεριά του server, ο σχεδιαστής της εφαρμογής αρκεί να ορίσει τις remote procedures με το να ορίσει τις μεθόδους σε ένα interface γραμμένο σε Java. Ο σχεδιαστής επίσης γράφει τον κώδικα μίας ή και περισσότερων κλάσεων που υλοποιούν αυτές τις μεθόδους. Αλλά και η σχεδίαση του client είναι αρκετά εύκολη. Ένας client δημιουργεί ένα proxy, ένα τοπικό object που παριστάνει την service, και μετά απλά καλεί τις μεθόδους στον proxy.

Με το JAX-RPC οι clients και οι Web Services έχουν ένα μεγάλο πλεονέκτημα – η Java είναι μια platform independent προγραμματιστική γλώσσα. Επιπλέον, το JAX-RPC δεν είναι περιοριστικό : ένας JAX-RPC client μπορεί να έχει πρόσβαση σε μια Web Service, η οποία δεν τρέχει σε πλατφόρμα που χρησιμοποιεί Java και αντίστροφα. Αυτό οφείλεται στο ότι το JAX-RPC χρησιμοποιεί τεχνολογίες ορισμένες από το World Wide Consortium (W3C), όπως του HTTP, του SOAP και του WSDL.

5.1.4 Επιλογή τρόπου δημιουργίας δυναμικών ιστοσελίδων

Για τη δημιουργία του client για την επίδειξη της εφαρμογής χρησιμοποιήθηκε η τεχνολογία των JSP (Java Server Pages), που περιλαμβάνει ένα σύνολο πλεονεκτημάτων σε σχέση με άλλες παρόμοιες τεχνολογίες και επιτρέπει στους προγραμματιστές να δημιουργήσουν δυναμικές ιστοσελίδες, οι οποίες μεταφράζονται κατά τη διάρκεια της εκτέλεσής τους. Οι JSPs έχουν τα παρακάτω πλεονεκτήματα:

- Είναι μέρος του κώδικα που συνοδεύει την πλατφόρμα της Java.
- Είναι ανεξάρτητες της αρχιτεκτονικής του συστήματος στο οποίο εκτελούνται.
- Μπορουν να «επικοινωνήσουν» πολύ εύκολα με άλλες κλάσεις ή οντότητες της Java.
- Μπορούν να εξυπηρετούν ταυτόχρονα πολλαπλές αιτήσεις διαφορετικών χρήστών και επιπλέον έχουν τη δυνατότητα να τις συγχρονίζουν.
- Έχουν τη δυνατότητα να διαβιβάζουν τις αιτήσεις των χρηστών σε άλλους servers ή JSPs.
- Παρέχουν όλες τις δυνατότητες των φαινομενικά ισχυρότερων servlets, αφού αυτόματα μεταφράζονται σε servlets που αναλαμβάνουν να φέρουν σε πέρας την απαίτηση.
- Δίνουν τη δυνατότητα στον προγραμματιστή να δημιουργήσει πολύ εύκολα τόσο στατικές όσο και δυναμικές οντότητες, με μια πιο φυσική προσέγγιση απο άλλες τεχνολογίες.
- Εκτός από έξοδο σε HTML μορφή, τα JSPs μπορούν να έχουν και έξοδο σε XML, ή ακόμα και σε WML μορφή, η οποία είναι απαραίτητη για την επικοινωνία με τους χρήστες μέσω του WAP.
- Συνοδεύεται από πλήρη και κατατοπιστική τεκμηρίωση (documentation), η οποία παρέχεται δωρεάν μέσω του World Wide Web.
- Υποστηρίζονται από τη μεγάλη πλειονότητα των servers που κυκλοφορουν στην αγορά λογισμικού.

5.1.5 Επιλογή Συστήματος Διαχείρισης Βάσεων Δεδομένων

Επειδή η πλατφόρμα ανάπτυξης του προγράμματος ήταν τα Windows επιλέξαμε να χρησιμοποιήσουμε ως ΣΔΒΔ τον SQL Server 2000, ο οποίος μαζί με την Oracle αποτελούν τα πιο εμπορικά Συστήματα Διαχείρισης Βάσεων Δεδομένων που χρησιμοποιούνται σήμερα.

Το σύστημά μας βεβαίως, λόγω της ανάπτυξης με τη Java και το JDBC μπορεί να έχει πρόσβαση σε μεγάλο πλήθος από διαφορετικές Β.Δ. με τη βοήθεια των κατάλληλων drivers.

5.1.6 Επιλογή τρόπου επικοινωνίας με τη Βάση Δεδομένων

Για την επικοινωνία με το Σύστημα Διαχείρισης Βάσεων Δεδομένων χρησιμοποιήθηκε το πρωτόκολλο JDBC (Java DataBase Connectivity). Το JDBC είναι ιδιαίτερα απλό, ευέλικτο και αποτελείται από κλάσεις και διαπροσωπίες (interfaces) που επιτρέπουν την επεξεργασία των δεδομένων μιας οποιασδήποτε Βάσης Δεδομένων με τη χρήση ενσωματωμένης SQL, καθώς επίσης και στην επισκόπηση των μετα-δεδομένων της. Για να μπορέσει μια συγκεκριμένη Βάση Δεδομένων να επικοινωνήσει μέσω JDBC θα πρέπει να υλοποιηθούν οι κατάληλοι οδηγοί (drivers), κλάσεις δηλαδή, που υλοποιούν τα interfaces και τις αφηρημένες κλάσεις που δηλώνονται από το JDBC API. Στην παρούσα διπλωματική επιλέχθηκαν οδηγοί τύπου 4. Ένας οδηγός τύπου 4 χαρακτηρίζεται από το γεγονός ότι μετατρέπει τις καλούμενες μεθόδους στο πρωτόκολλο δικτύου (network protocol) το οποίο χρησιμοποιείται άμεσα από το εκάστοτε ΣΔΒΔ. Αυτό επιτρέπει την πραγματοποίηση μιας άμεσης κλήσης από το μηχάνημα-πελάτη (client machine) στον εξυπηρετητή του ΣΔΒΔ και είναι μια πρακτική λύση για πρόσβαση μέσω κάποιου δικτύου ή ακόμα και μέσω του Internet, κάτι που συμβαίνει στην πλειοψηφία των περιπτώσεων χρησιμοποίησης του συστήματος. Καθώς πολλά από αυτά τα πρωτόκολλα είναι εξειδικευμένα ανά σύστημα, η πρωταρχική πηγή για αυτούς τους οδηγούς ήταν ο κατασκευαστής του συγκεκριμένου ΣΔΒΔ, δηλαδή η Microsoft.

5.1.7 Επιλογή εργαλείου ανάπτυξης κώδικα

Για την ανάπτυξη του κώδικα χρησιμοποιήθηκε το JCreator ένα shareware πρόγραμμα για ανάπτυξη Java προγραμμάτων που δουλεύει πάνω στο JDK και προσφέρει ένα γραφικό παραθυρικό περιβάλλον αρκετά εύχρηστο και βοηθητικό. Προτιμήθηκε από άλλα προγράμματα που κατασκευάζουν κώδικα, για να υπάρχει άμεση επαφή του κατασκευαστή με τον κώδικα.

5.1.8 Επιλογή Web Server

Ως Web Server χρησιμοποιήθηκε ο Tomcat 4.0.1 (The Jakarta Project). Τα πλεονεκτήματά του σε σχέση με άλλους Web Server που κυκλοφορούν στην αγορά λογισμικού είναι τα ακόλουθα:

- Παρέχει υποστήριξη για εφαρμογές που κάνουν χρήση της τεχνολογίας των Java Server Pages (JSP).

- Αποτελεί έναν αρκετά σταθερό Web Server.

- Συνοδεύεται από πλήρη και κατατοπιστική τεκμηρίωση (documentation), η οποία παρέχεται δωρεάν μέσω του World Wide Web.

- Είναι ένας από τους πιο διαδεδομένους Web Servers που κυκλοφορούν στην αγορά λογισμικού.

- Διανέμεται δωρεάν μέσω του World Wide Web.

Ο παραγόμενος κώδικας βέβαια είναι πολύ εύκολο να χρησιμοποιηθεί και σε άλλους Web Servers, αρκεί βέβαια υποστηρίζουν την τεχνολογία των Java Server Pages.

5.2 Λεπτομέρειες υλοποίησης

5.2.1 Οργάνωση του κώδικα

Ο κώδικας της εφαρμογής μας είναι οργανωμένος σε 3 packages, τα οποία περιέχουν κλάσεις που σχετίζονται μεταξύ τους. Τα packages αυτά περιέχονται σε ένα κύριο package το WebServiceDesigner, που περιέχει όλα τα άλλα packages.

5.2.1.1 Package GUI

Περιέχει τις κλάσεις για την ανάπτυξη και διαχείριση του GUI (Grafical User Interface) και έχει την ευθύνη της επικοινωνίας με το χρήστη.

5.2.1.2 Package UtilityFiles

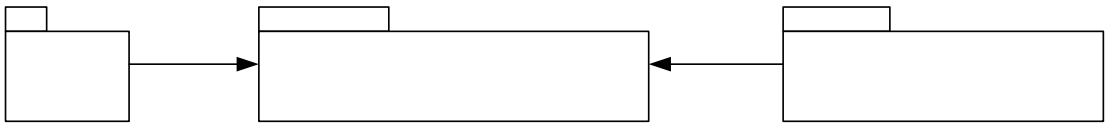
Περιέχει βοηθητικές κλάσεις. Συγκεκριμένα περιέχει:

- Μια κλάση, η οποία περιγράφει τη βάση στην οποία συνδέθηκε ο χρήστης, δηλαδή όλους τους πίνακες που περιέχει η βάση αυτή καθώς και όλα τα πεδία και τους αντίστοιχους τύπους των πινάκων αυτών.
- Μια κλάση, η οποία παράγει ένα Configuration File, το οποίο περιγράφει πλήρως το Web Service αφότου ο χρήστης έχει δώσει όλα τα δεδομένα που του ζητάει η εφαρμογή.
- Μια κλάση, η οποία παράγει τις απαραίτητες java classes καθώς και κάποια property files, τα οποία χρησιμοποιεί το JWSDP για να κάνει deploy το Web Service.

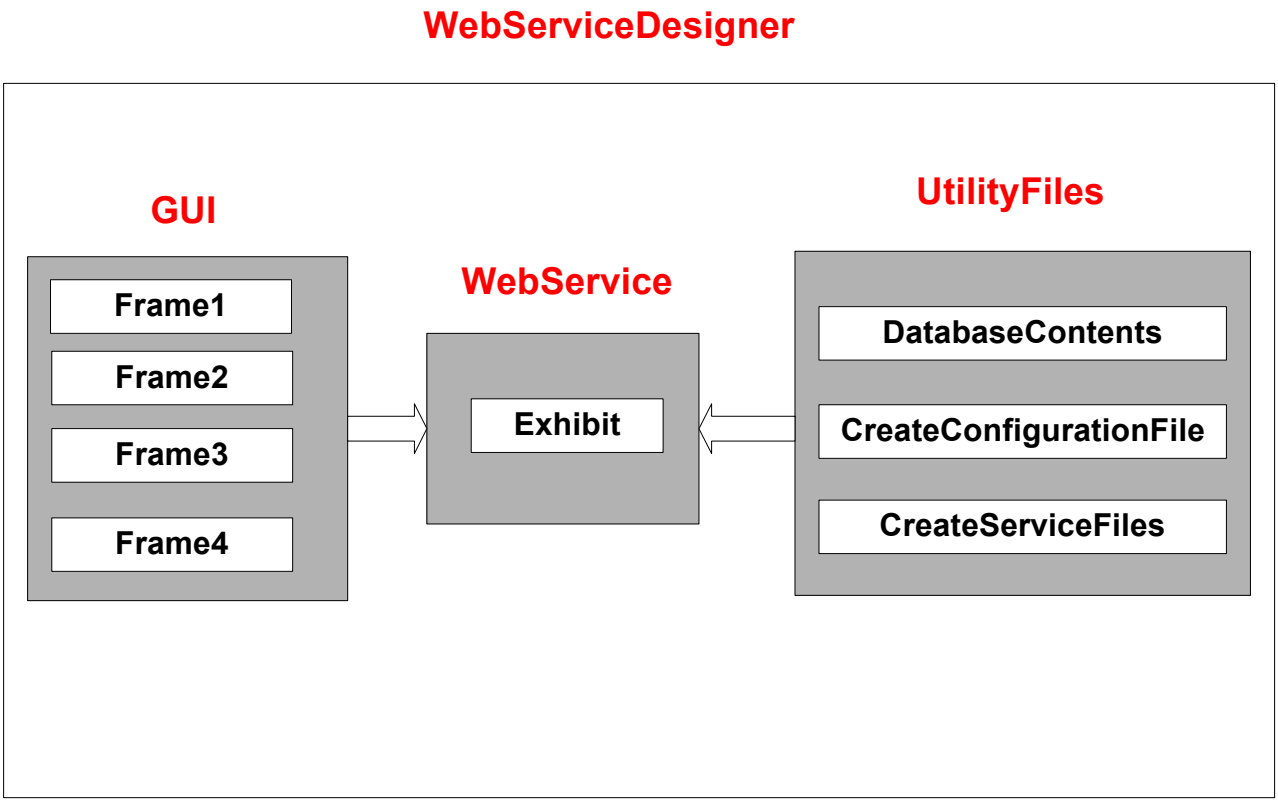
5.2.1.3 Package Web_Service

Περιέχει τις κλάσεις που πραγματοποιούν όλη τη λειτουργία του προγράμματος και χρησιμοποιούν τις κλάσεις από τα παραπάνω packages.

Σχηματικά, τα packages που κατασκευάσαμε καθώς και η εξάρτηση μεταξύ τους φαίνεται στο παρακάτω σχήμα:



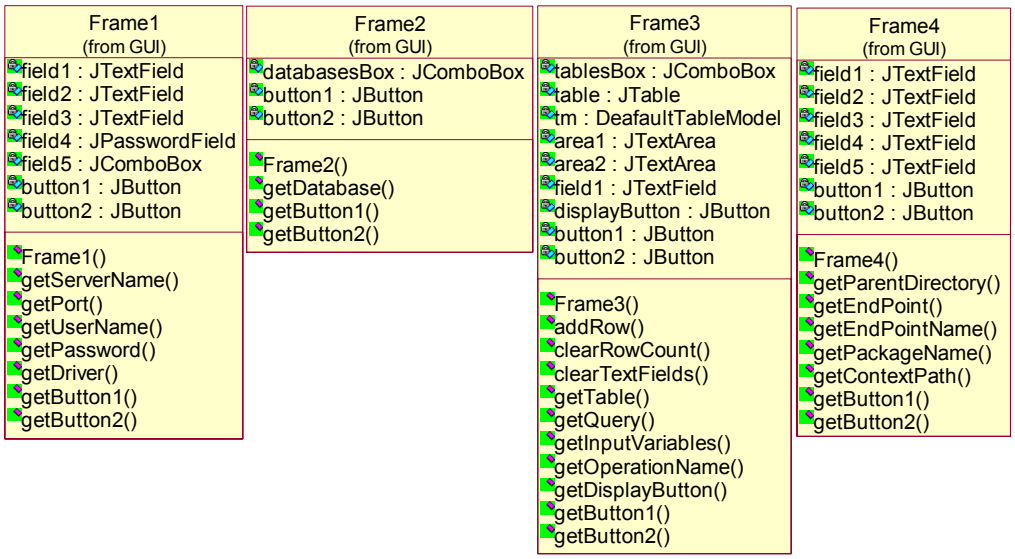
Αν θελήσουμε τώρα να δούμε αναλυτικά τα περιεχόμενα των packages έχουμε το παρακάτω σχήμα. Τα packages έχουν σημειωθεί ως κουτιά που περιέχουν τις κλάσεις, ενώ οι βασικές διαχειριστικές κλάσεις έχουν τονισμένα γράμματα. Από το σχήμα φαίνεται επίσης ότι οι βασικές διαχειριστικές κλάσεις βρίσκονται στο package WebService, το οποίο συνεργάζεται άμεσα τόσο με το package GUI όσο και με το package UtilityFiles.



5.2.2 Συνοπτική παρουσίαση των κλάσεων

5.2.2.1 Package GUI

Το λογικό διάγραμμα του package GUI φαίνεται παρακάτω:



Class Frame1

Εμφανίζει το αρχικό παράθυρο της εφαρμογής, στο οποίο ο χρήστης καλείται να συνδεθεί με κάποιο ΣΔΒΔ εισάγοντας:

1. Το όνομα του server
2. Το port στο οποίο "ακούει" το ΣΔΒΔ
3. Το όνομα του χρήστη
4. Τον κωδικό του χρήστη
5. Τον κατάλληλο driver ανάλογα με το είδος του ΣΔΒΔ (π.χ. αν πρόκειται για τον SQL Server διαλέγει τον αντίστοιχο SQL Server Driver από μια drop-down list)

Class Frame2

Εμφανίζει το δεύτερο παράθυρο της εφαρμογής, στο οποίο ο χρήστης καλείται να επιλέξει κάποια από τις υπάρχουσες βάσεις του ΣΔΒΔ από μια drop-down list.

Class Frame3

Εμφανίζει το τρίτο παράθυρο της εφαρμογής, στο οποίο ο χρήστης έχει τη δυνατότητα να δει τους πίνακες που περιέχει η βάση την οποία επέλεξε καθώς και τα πεδία και τους αντίστοιχους τύπους των πινάκων αυτών. Έχοντας λοιπόν την εικόνα της βάσης καλείται να δώσει τα συστατικά του κάθε operation του Web Service. Συγκεκριμένα καλείται να δώσει:

1. Το query που θα εκτελεί το operation
2. Τις μεταβλητές εισόδου του operation
3. Το όνομα του operation

Class Frame4

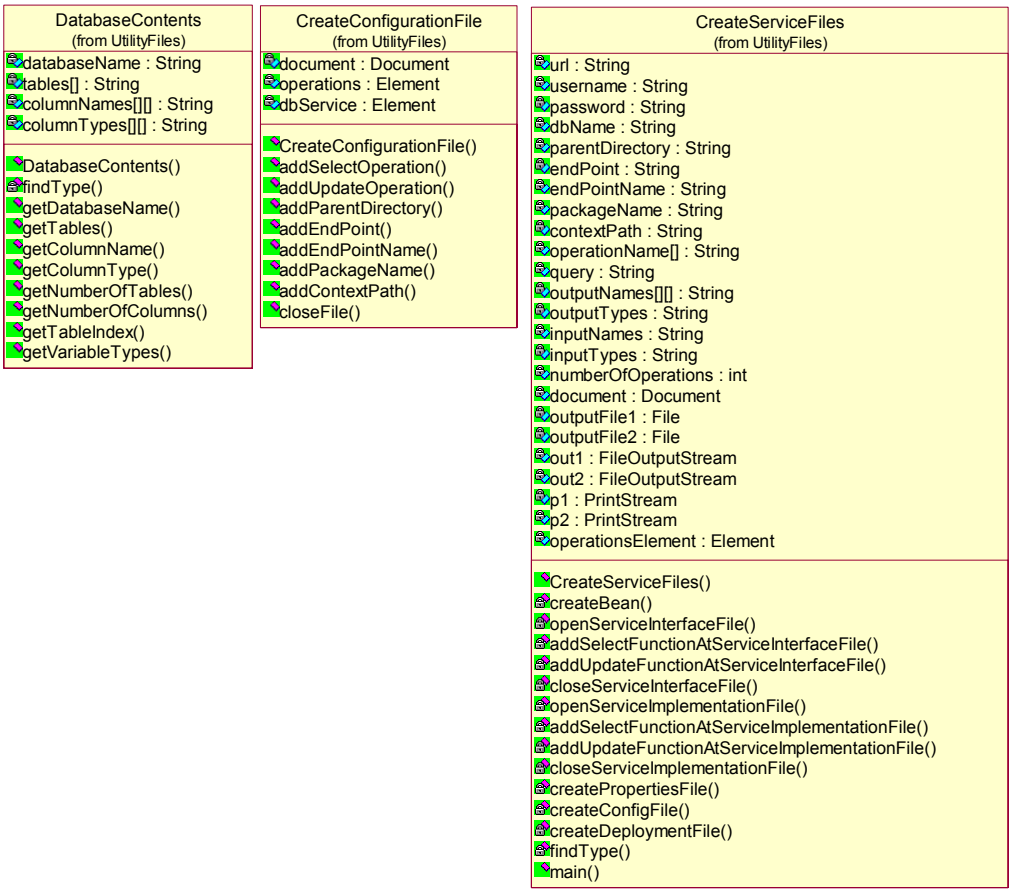
Εμφανίζει το τελευταίο παράθυρο της εφαρμογής, στο οποίο ο χρήστης καλείται να δώσει κάποιες επιπλέον πληροφορίες που περιγράφουν το Web Service και συγκεκριμένα:

1. Το directory, στο οποίο έκανε unzip την εφαρμογή
2. Το όνομα του Web Service
3. Το url στο οποίο θέλει να γίνει deploy το Web Service
4. Το όνομα του package, στο οποίο θα αποθηκευτούν οι κλάσεις οι οποίες θα προκύψουν από τη μεταγλώττιση του κώδικα κατά την παραγωγή του Web Service

Μόλις δώσει τις πληροφορίες αυτές, ο χρήστης μπορεί να πατήσει το κουμπί deploy και να κάνει deploy το Web Service. Το αν έγινε σωστά deploy μπορεί να το ελέγξει πηγαίνοντας στο url στο οποίο είναι εγκατεστημένο το Web Service, το οποίο και έδωσε παραπάνω.

5.2.2.2 Package UtilityFiles

Το λογικό διάγραμμα του package UtilityFiles φαίνεται παρακάτω:



Class DatabaseContents

Ανάλογα με τη βάση που επέλεξε ο χρήστης, η κλάση DatabaseContents περιέχει τη μορφή της βάσης αυτής, δηλαδή όλους τους πίνακες της βάσης καθώς και τα πεδία και τους αντίστοιχους τύπους των πινάκων αυτών.

Class CreateConfigurationFile

Η κλάση CreateConfigurationFile αναλαμβάνει την κατασκευή ενός αρχείου xml, συγκεκριμένα του *configurationFile.xml* το οποίο περιγράφει πλήρως το Web Service περιέχοντας όλα τα στοιχεία που έδωσε ο χρήστης κατά το τρέξιμο της εφαρμογής. Περιέχει δηλαδή, τα στοιχεία που έδωσε ο χρήστης στα Frames 1-4, όπως τα στοιχεία που χρειάζονται για να συνδεθεί με το ΣΔΒΔ, τις λεπτομέρειες των operations με τα αντίστοιχα queries τους και τις πληροφορίες για το deployment του Web Service, όπως το όνομα του Web Service, το url στο οποίο θα γίνει deploy κ.λ.π.

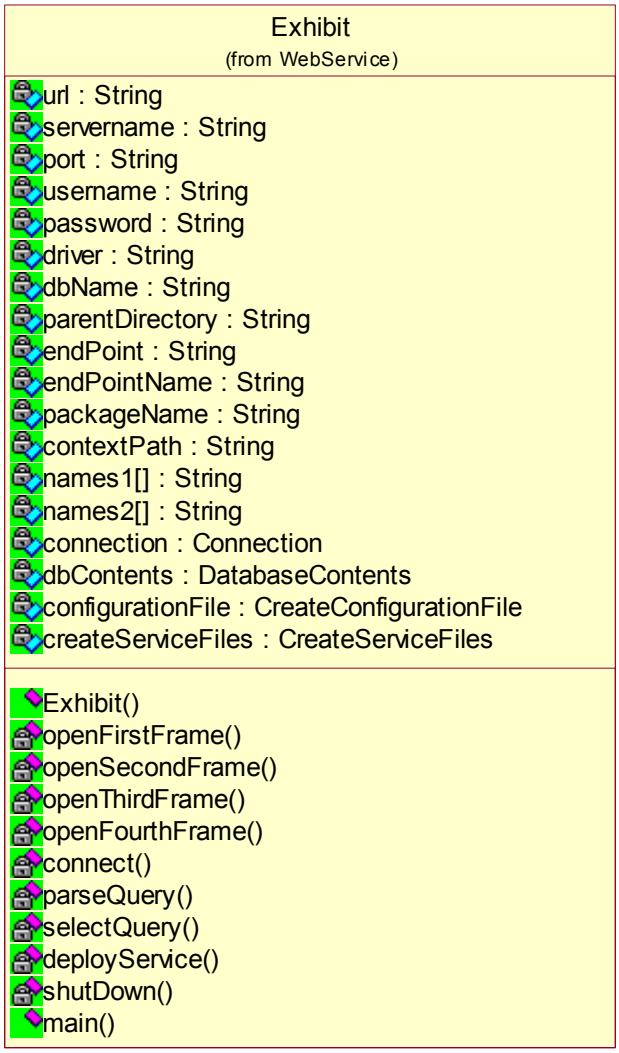
Class CreateServiceFiles

Η κλάση CreateServiceFiles αναλαμβάνει την κατασκευή όλων εκείνων των .java αρχείων καθώς και κάποιον property files, τα οποία ζητάει το JWSDP για να κάνει deploy το Web Service. Συγκεκριμένα παράγει τα εξής αρχεία:

1. Για εκείνα τα operations που κάνουν select κάποια δεδομένα από τη βάση φτιάχνει κατάλληλα serviceBeans, τα οποία έχουν ως μεταβλητές τα πεδία των selects. Π.χ. αν έχουμε ένα operation που εκτελεί το query "*SELECT Professors.FirstName, Professors.LastName FROM Professors WHERE Professors.ID = ?*", τότε το serviceBean θα έχει ως μεταβλητές τα *Professors_FirstName, Professors_LastName*.
2. Παράγει το αρχείο serviceIF.java, το οποίο αποτελεί το interface του Web Service.
3. Παράγει το αρχείο serviceImpl.java, το οποίο αποτελεί στην ουσία τον wrapper του Web Service και περιέχει ως μεθόδους του τα operations του Web Service.
4. Παράγει κάποια property files τα οποία ζητάει το JWSDP και συγκεκριμένα τα αρχεία build.properties και jaxrpc-ri.xml, τα οποία είναι απαραίτητα για το deployment του Web Service.

5.2.2.3 Package WebService

Το λογικό διάγραμμα του package WebService φαίνεται παρακάτω:



Class Exhibit

Η κλαση Exhibit είναι η κεντρική κλάση της εφαρμογής, η οποία χρησιμοποιεί τις κλάσεις από τα υπόλοιπα packages για να πετύχει την επικοινωνία με το χρήστη και να κάνει deploy το Web Service. Συγκεκριμένα καλεί τις κλάσεις Frame 1-4 για να εμφανίσει τα 4 παράθυρα της εφαρμογής και μόλις ο χρήστης δώσει όλα τα απαραίτητα στοιχεία καλεί την κλάση CreateConfigurationFile, για να φτιάξει το αρχείο που περιγράφει το Web Service. Κατόπιν καλεί την κλάση CreateServiceFiles για να φτιάξει όλα τα απαραίτητα αρχεία που ζητάει το JWSDP και τέλος κάνει deploy το Web Service.

Αν θέλαμε τώρα να δούμε τη σειρά με την οποία καλούνται οι μέθοδοι κάθε κλάσης έχουμε τα εξής:

1. Καλείται η μέθοδος *openFirstFrame()* της κλάσης *Exhibit*. Η μέθοδος αυτή αρχικοποιεί την κλάση *Frame1*, που βρίσκεται στο package GUI, η οποία εμφανίζει το πρώτο παράθυρο της εφαρμογής. Στο παράθυρο αυτό ζητείται από το χρήστη να δώσει τα απαραίτητα στοιχεία για τη σύνδεση με το ΣΔΒΔ.
2. Αφού ο χρήστης δώσει τα στοιχεία αυτά, καλείται η μέθοδος *connect()* της κλάσης *Exhibit*, με την οποία συνδεόμαστε με το ΣΔΒΔ.
3. Καλείται η μέθοδος *openSecondFrame()* της κλάσης *Exhibit*. Η μέθοδος αυτή αρχικοποιεί την κλάση *Frame2*, που βρίσκεται στο package GUI, η οποία εμφανίζει το δεύτερο παράθυρο της εφαρμογής. Στο παράθυρο αυτό ζητείται από το χρήστη να επιλέξει τη βάση με την οποία θέλει να συνδεθεί.
4. Αφού ο χρήστης διαλέξει τη βάση αυτή αρχικοποιείται η κλάση *CreateConfigurationFile*, που βρίσκεται στο package UtilityFiles και η οποία δημιουργεί το αρχείο *ConfigurationFile.xml* το οποίο περιγράφει πλήρως το Web Service περιέχοντας όλα τα στοιχεία που έδωσε ο χρήστης κατά το τρέξιμο της εφαρμογής. Αρχικοποιείται επίσης η κλάση *DatabaseContents*, που βρίσκεται στο package UtilityFiles και η οποία περιέχει τη μορφή της βάσης που επέλεξε ο χρήστης, δηλαδή όλους τους πίνακες της βάσης καθώς και τα πεδία και τους αντίστοιχους τύπους των πινάκων αυτών
5. Καλείται η μέθοδος *openThirdFrame()* της κλάσης *Exhibit*. Η μέθοδος αυτή αρχικοποιεί την κλάση *Frame3*, που βρίσκεται στο package GUI, η οποία εμφανίζει το τρίτο παράθυρο της εφαρμογής. Στο παράθυρο αυτό ζητείται από το χρήστη να δώσει τα operations του Web Service. Ό,τι στοιχεία δώσει προστίθενται στο *ConfigurationFile.xml* με την κλήση των αντίστοιχων μεθόδων της κλάσης *ConfigurationFile* και συγκεκριμένα των *addSelectOperation()* και *addUpdateOperation()*, ανάλογα με το είδος του operation.
6. Καλείται η μέθοδος *openFourthFrame()* της κλάσης *Exhibit*. Η μέθοδος αυτή αρχικοποιεί την κλάση *Frame4*, που βρίσκεται στο package GUI, η οποία εμφανίζει το τέταρτο παράθυρο της εφαρμογής. Στο παράθυρο αυτό ζητείται από το χρήστη να δώσει κάποιες παραμέτρους, οι οποίες είναι απαραίτητες για το deployment του Web Service. Οι παράμετροι αυτοί προστίθενται στο *ConfigurationFile.xml* με την κλήση των κατάλληλων μεθόδων της κλάσης *CreateConfigurationFile*.
7. Καλείται η μέθοδος *closeFile()* της κλάσης *CreateConfigurationFile*, με την οποία «κλείνει» το αρχείο *ConfigurationFile.xml*.

8. Αρχικοποιείται η κλάση *CreateServiceFiles*, που βρίσκεται στο package UtilityFiles παίρνοντας ως είσοδο το *ConfigurationFile.xml* και η οποία αναλαμβάνει την κατασκευή όλων εκείνων των .java αρχείων καθώς και κάποιον property files, τα οποία ζητάει το JWSDP για να κάνει deploy το Web Service.
9. Καλείται η μέθοδος *deployService()* της κλάσης *Exhibit* οπότε και γίνεται deploy το Web Service με τη βοήθεια του JWSDP.
10. Τέλος, καλείται η μέθοδος *shutDown()* της κλάσης *Exhibit* οπότε και «κλείνει» η σύνδεση με το ΣΔΒΔ.

6

Έλεγχος

Στην ενότητα αυτή θα περιγραφεί ο έλεγχος του συστήματος.

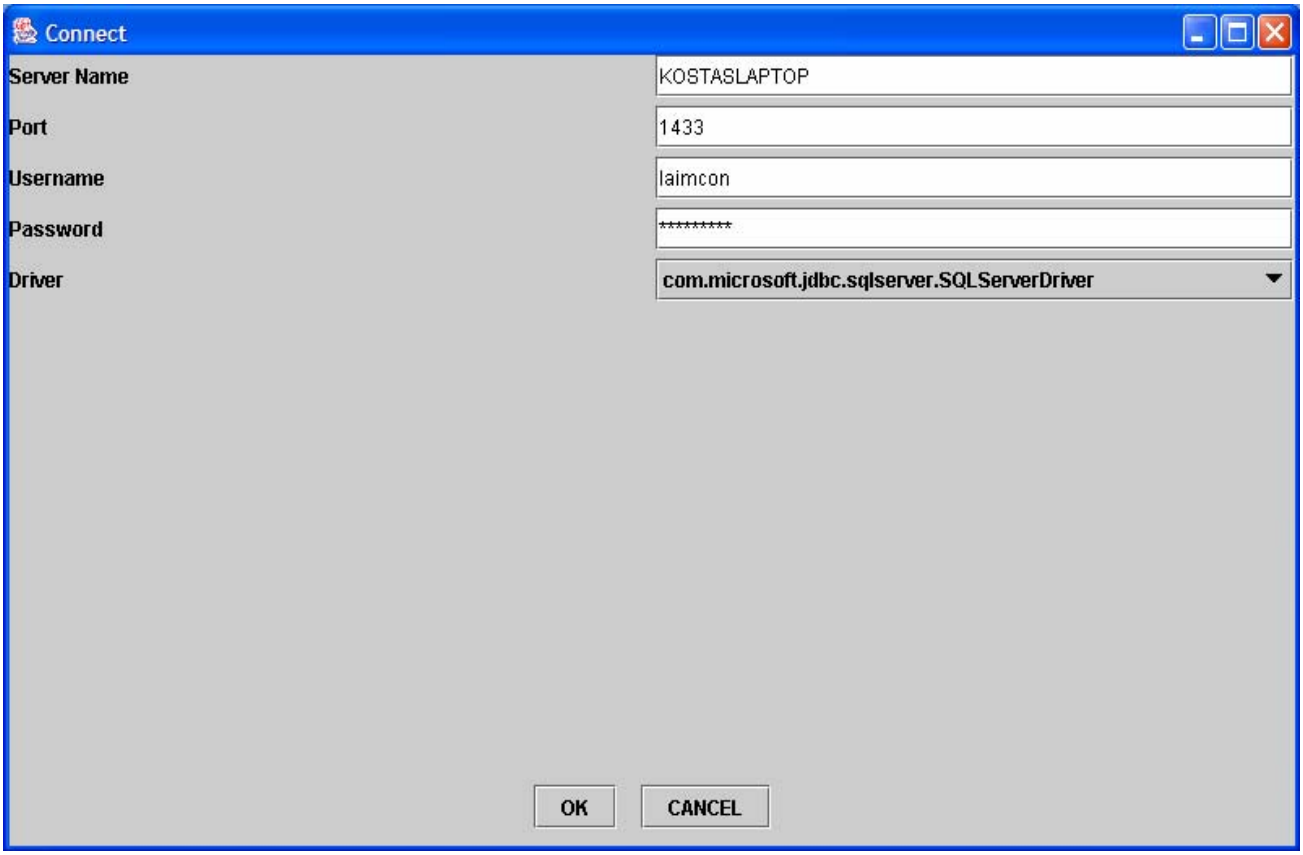
6.1 Μεθοδολογία Ελέγχου

Η μεθοδολογία που έχει χρησιμοποιηθεί είναι η μέθοδος του μαύρου κουτιού. Θα παρουσιαστεί ένα συγκεκριμένο παραδείγματα εκτέλεσης της εφαρμογής για τη δημιουργία ενός Web Service, ώστε να αποδειχτεί η ορθότητα των λειτουργιών που υλοποιήθηκαν.

6.2 Αναλυτική παρουσίαση έλεγχου

Για να φανούν στην πράξη τόσο οι δυνατότητες του συστήματος όσο και ο τρόπος λειτουργίας του ακολουθεί μια επίδειξη του συστήματος.

Τρέχοντας το αρχείο runWebServiceApplication.bat ξεκινά την εφαρμογή μας. Το πρώτο παράθυρό της φαίνεται στο Σχήμα 1:

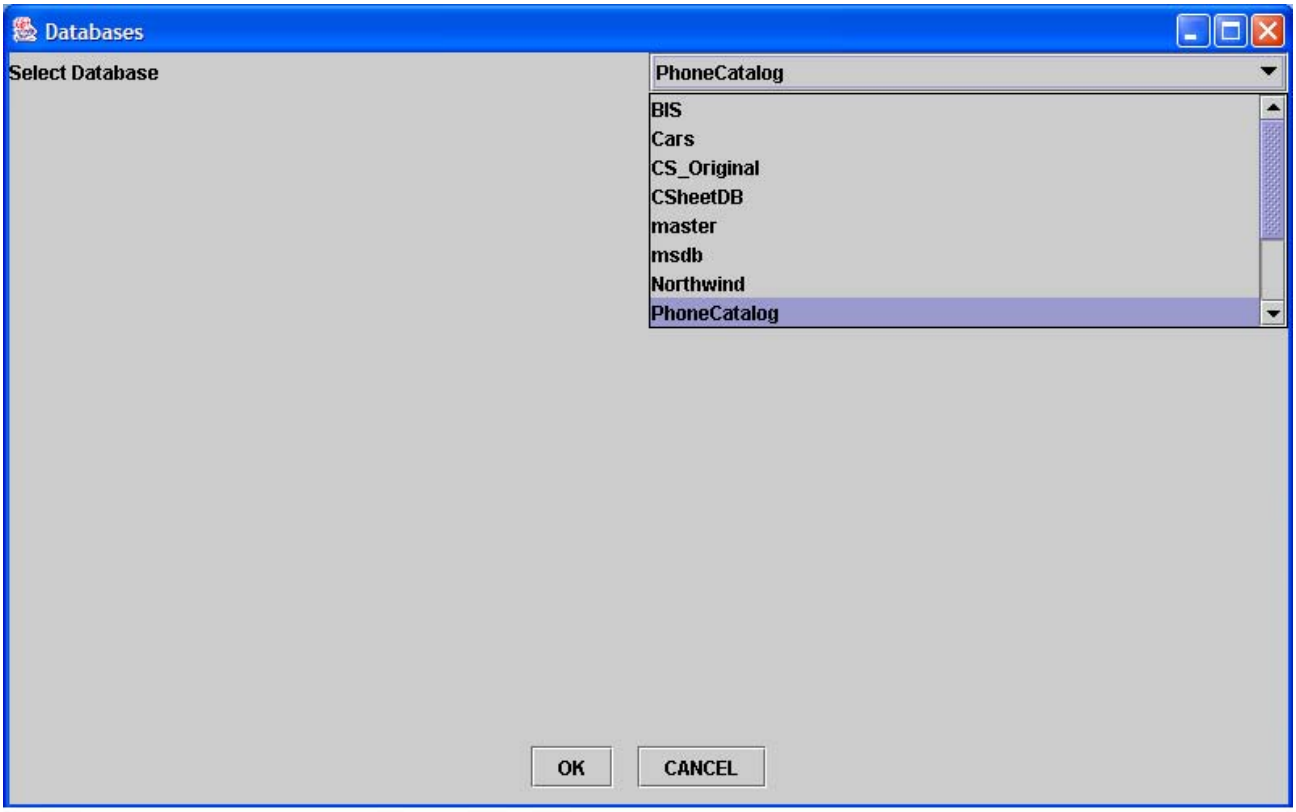


Σχήμα 1

Στο παράθυρο αυτό ζητείται από το χρήστη να δώσει τα εξής στοιχεία:

1. Το όνομα του server, στον οποίο είναι εγκατεστημένο το ΣΔΒΔ (στην περίπτωση μας ο SQL Server 2000).
2. Το port του server (στην περίπτωση του SQL Server το default port είναι το 1433).
3. Το όνομα ενός χρήστη που έχει πρόσβαση στο ΣΔΒΔ.
4. Τον κωδικό του χρήστη.
5. Τον κατάλληλο driver για τη σύνδεση με το ΣΔΒΔ (στην περίπτωση μας το com.microsoft.jdbc.sqlserver.SQLServerDriver).

Μόλις δώσουμε τα στοιχεία αυτά πατάμε το κουμπί OK. Η εφαρμογή χρησιμοποιεί τα στοιχεία που δώσαμε για να συνδεθεί με το ΣΔΒΔ οπότε και μεταφερόμαστε στο δεύτερο παράθυρο, το οποίο φαίνεται στο Σχήμα 2:

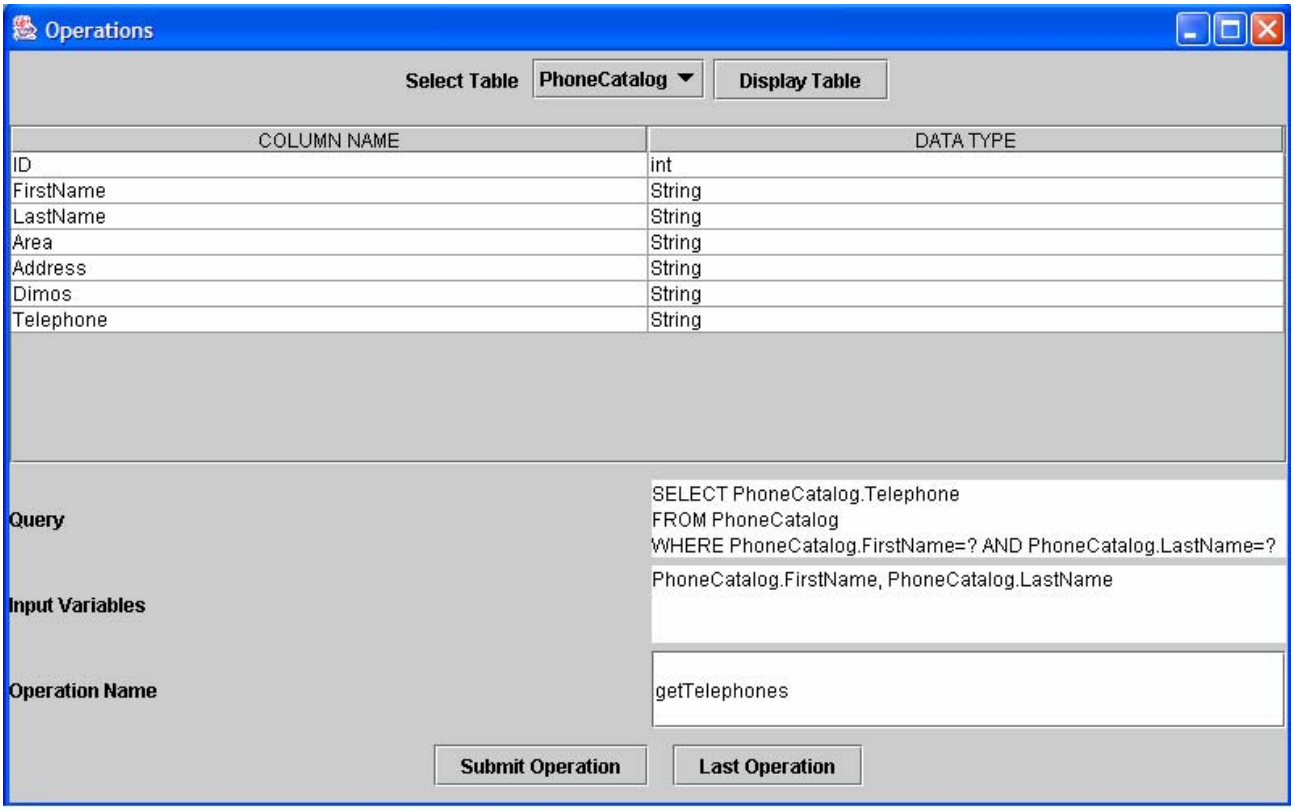


Σχήμα 2

Στο παράθυρο αυτό ο χρήστης μπορεί να δει τις υπάρχουσες βάσεις και να επιλέξει αυτή που θα χρησιμοποιήσει το Web Service. Μόλις πατήσει το κουμπί OK αυτό που κάνει εσωτερικά ο κώδικας της εφαρμογής είναι οι εξής 2 λειτουργίες:

1. Ανοίγει το αρχείο ConfigurationFile.xml, το οποίο όπως έχουμε πει περιγράφει το Web Service περιέχοντας όλα τα στοιχεία που έδωσε ο χρήστης κατά το τρέξιμο της εφαρμογής. Συγκεκριμένα, στο σημείο αυτό της εκτέλεσης της εφαρμογής τα στοιχεία που περιέχει το αρχείο αυτό είναι τα στοιχεία που έδωσε ο χρήστης στην πρώτη οθόνη (δηλαδή το όνομα του server, το port, το όνομα και τον κωδικό του χρήστη και τον κατάλληλο driver) και στην δεύτερη οθόνη (δηλαδή το όνομα της βάσης).
2. Δημιουργεί την κλάση DatabaseContents, η οποία είναι η κλάση που περιγράφει τη βάση που επέλεξε ο χρήστης, δηλαδή όλους τους πίνακες της βάσης καθώς και τα πεδία και τους αντίστοιχους τύπους των πινάκων αυτών.

Μόλις τελειώσουν οι 2 αυτές λειτουργίες μεταφερόμαστε στο τρίτο παράθυρο της εφαρμογής, το οποίο φαίνεται στο Σχήμα 3. Υποθέτουμε ότι επιλέξαμε τη βάση PhoneCatalog, η οποία είναι μια βάση με τα τηλέφωνα όλων των κατοίκων της Αθήνας.



Σχήμα 3

Στο παράθυρο αυτό, ο χρήστης μπορεί αρχικά να επιλέξει κάποιον από τους πίνακες της βάσης και πατώντας το κουμπί **Display Table** να δει τα πεδία του πίνακα αυτού και τους αντίστοιχους τύπους τους. Η βάση PhoneCatalog περιέχει μόνο έναν πίνακα με όνομα επίσης PhoneCatalog οπότε πατώντας το κουμπί **Display Table** έχουμε τα αποτελέσματα που φαίνονται στο Σχήμα 3.

Αφού λοιπόν ο χρήστης έχει αποκτήσει την εικόνα της βάσης, καλείται να δώσει τα operations του Web Service συμπληρώνοντας τα πεδία:

1. **Query**: Εδώ καλείται να δώσει το query που θα εκτελεί το operation.
2. **Input Variables**: Εδώ καλείται να δώσει τις μεταβλητές εισόδου του query, δηλαδή εκείνα τα στοιχεία που θα δίνει ένας client σαν είσοδο στο αντίστοιχο operation του Web Service.
3. **Operation Name**: Εδώ καλείται να δώσει το όνομα του operation.

Συγκεκριμένα ας υποθέσουμε ότι θέλουμε να ορίσουμε ένα operation στο οποίο ένας χρήστης να δίνει σαν είσοδο το Όνομα και το Επώνυμο ενός κατοίκου της Αθήνας και το Web Service να του επιστρέφει το τηλέφωνό του. Τότε τα πεδία θα πρέπει να συμπληρωθούν με τον τρόπο που φαίνεται στο Σχήμα 3.

Μόλις ο χρήστης συμπληρώσει τα πεδία του σχήματος 3 έχει 2 επιλογές:

1. Αν θέλει να ορίσει κι άλλα operations για το Web Service, αρκεί να πατήσει το κουμπί **Submit Operation**, οπότε και τα πεδία καθαρίζονται από τα προηγούμενα δεδομένα και καλείται να δώσει τα καινούργια στοιχεία.
2. Αν αυτό ήταν το τελευταίο operation που θέλει να ορίσει, αρκεί να πατήσει το κουμπί **Last Operation**, οπότε και μεταφέρεται στο τέταρτο και τελευταίο παράθυρο της εφαρμογής.

Σε κάθε περίπτωση αυτά που κάνει εσωτερικά ο κώδικας κάθε φορά που ο χρήστης κάνει submit κάποιο operation είναι τα εξής:

Αρχικά γίνεται parsing του query, οπότε και έχουμε 2 περιπτώσεις – να έχουμε ένα Select query ή ένα Update, Insert, Delete query. Ανάλογα τώρα με την περίπτωση γίνονται τα εξής:

SELECT QUERY

Αν το query είναι τύπου Select, γίνεται κατάλληλο parsing για να πάρουμε τα ονόματα των πεδίων που επιστρέφει το query. Π.χ. στο παράδειγμά μας έχουμε το query:

SELECT PhoneCatalog.Telephone

FROM PhoneCatalog

WHERE PhoneCatalog.FirstName=? AND PhoneCatalog.LastName=?

Με το parsing που κάνει η εφαρμογή βλέπει αρχικά ότι πρόκειται για ένα query τύπου Select, γιατί η πρώτη λέξη είναι *SELECT*. Αφού λοιπόν πρόκειται για Select query, η συνέχεια του parsing είναι να πάρει τα ονόματα των πεδίων που επιστρέφει το query, και τον αντίστοιχο πίνακα στον οποίο περιέχεται το πεδίο αυτό. Συγκεκριμένα στο παράδειγμα μας, το query επιστρέφει το *PhoneCatalog.Telephone*, δηλαδή το πεδίο *Telephone* που περιέχεται στον πίνακα *PhoneCatalog*.

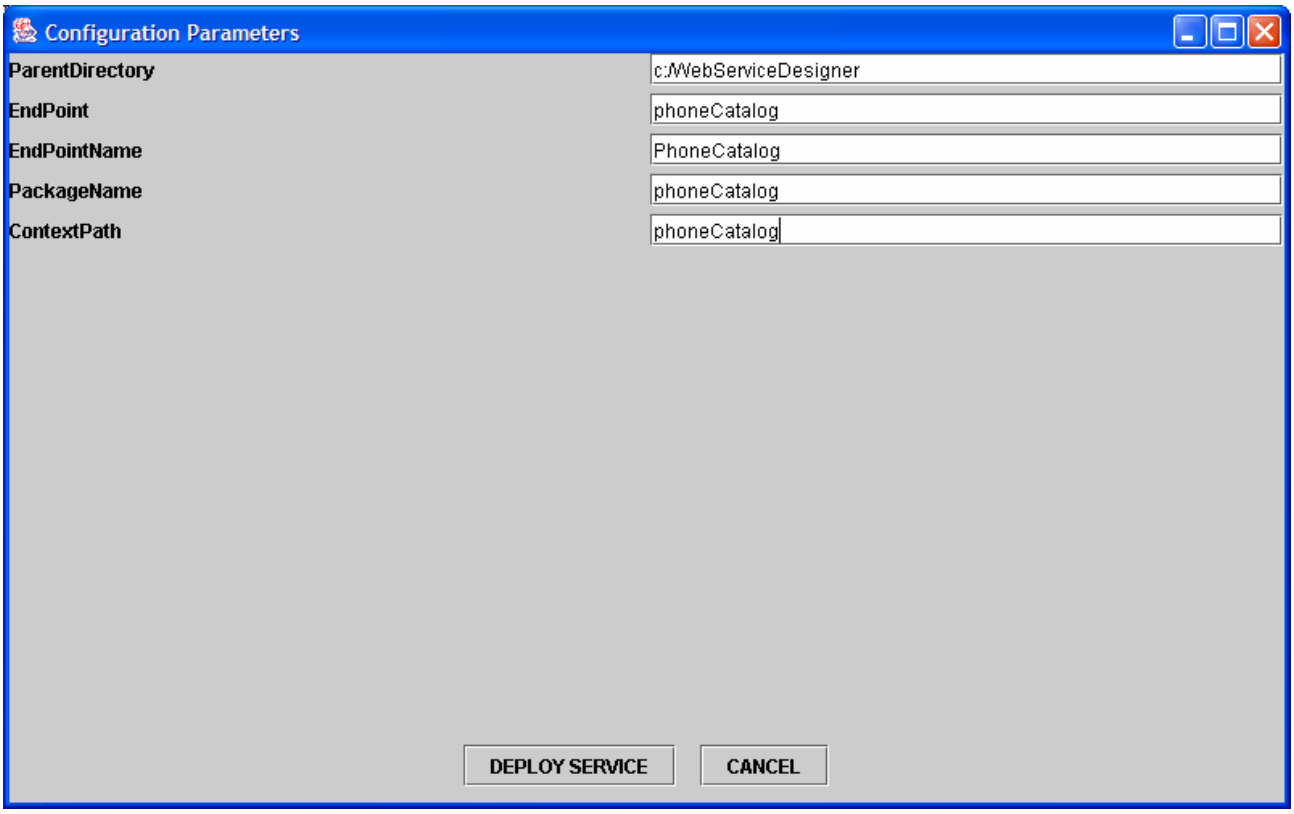
Επόμενο βήμα είναι να πάρουμε τους τύπους τόσο των πεδίων που επιστρέφει το query όσο και των πεδίων που θα δέχεται ως είσοδο το query. Συγκεκριμένα στο παράδειγμά μας έχουμε ως είσοδο στο query τα πεδία *PhoneCatalog.FirstName* και *PhoneCatalog.LastName* που είναι τύπου String και το query επιστρέφει το *PhoneCatalog.Telephone* που είναι επίσης τύπου String. Για να βρει τους τύπους των πεδίων αυτών, η εφαρμογή χρησιμοποιεί την κλάση DatabaseContents την οποία αρχικοποιήσαμε σε παραπάνω βήμα και η οποία όπως είπαμε περιγράφει τη βάση που επέλεξε ο χρήστης, δηλαδή όλους τους πίνακες της βάσης καθώς και τα πεδία και τους αντίστοιχους τύπους των πινάκων αυτών.

UPDATE, INSERT, DELETE QUERY

Αν το query είναι τύπου Update, Insert ή Delete το parsing που γίνεται είναι ακριβώς ίδιο με την περίπτωση του Select, απλά δε μας ενδιαφέρει να βρούμε το όνομα και τον τύπο που θα επιστρέφει το query, αφού στην περίπτωση μη Select query το μόνο που επιστρέφεται είναι μια μεταβλητή result τύπου int, που δηλώνει αν η εκτέλεση του query ήταν επιτυχημένη ή όχι.

Αφού λοιπόν πάρουμε μετά το parsing τα ονόματα και τους τύπους των πεδίων εισόδου και εξόδου στην ουσία έχουμε έτοιμα τα δομικά στοιχεία του operation. Επόμενο βήμα είναι να προσθέσουμε το operation αυτό στο αρχείο ConfigurationFile.xml. Τον τρόπο με τον οποίο το προσθέτουμε θα τον δούμε παρακάτω, όταν και θα δείξουμε ακριβώς τη μορφή του ConfigurationFile.xml μετά το τέλος της εφαρμογής.

Όταν ο χρήστης τελειώσει και με το τελευταίο operation πατάει το κουμπί **Last Operation** οπότε και μεταφέρεται στο τέταρτο και τελευταίο παράθυρο της εφαρμογής, το οποίο φαίνεται στο Σχήμα 4:



Σχήμα 4

Στο παράθυρο αυτό ο χρήστης καλείται να δώσει κάποιες παραμέτρους, οι οποίες είναι απαραίτητες για το deployment του Web Service. Συγκεκριμένα, οι παράμετροι αυτοί είναι οι εξής:

1. **ParentDirectory** : Το directory στο οποίο ο χρήστης εγκατέστησε την εφαρμογή.
2. **EndPoint** : Μαζί με το ContextPath φτιάχνουν το url στο οποίο θέλει ο χρήστης να γίνει deploy το Web Service.
3. **EndPointName** : Το όνομα του Web Service.
4. **PackageName** : Το όνομα του package στο οποίο τοποθετούνται οι κλάσεις τις οποίες κάνει compile το JWSDP κατά την εγκατάσταση του Web Service.
5. **ContextPath** : Μαζί με το EndPoint φτιάχνουν το url στο οποίο θέλει ο χρήστης να γίνει deploy το Web Service. Συγκεκριμένα, το url αυτό είναι το http://localhost:8080/ContextPath/EndPoint. Το url αυτό θα πρέπει να επισκεφτεί ο σχεδιαστής του Web Service μετά την εκτέλεση της εφαρμογής για να δει ότι το Web Service εγκαταστάθηκε κανονικά.

Μόλις ο χρήστης δώσει όλες τις παραμέτρους στο παράθυρο αυτό μπορεί να πατήσει το κουμπί **DEPLOY SERVICE** για να κάνει deploy το Web Service. Αυτά που γίνονται εσωτερικά στον κώδικα της εφαρμογής όταν ο χρήστης πατήσει το κουμπί **DEPLOY SERVICE** είναι τα εξής:

Πρώτα απ' όλα, προστίθενται και οι τελευταίες πληροφορίες στο αρχείο *ConfigurationFile.*xml και το αρχείο αυτό κλείνει. Η τελική μορφή του για το το Web Service που φτιάξαμε φαίνεται στην επόμενη σελίδα:

```
<?xml version="1.0" encoding="UTF-8"?>
<Web_Service>
<!--This is the configuration file for the Web Service-->
   <DatabaseService>
      <URL>jdbc:microsoft:sqlserver://KOSTASLAPTOP:1433</URL>
      <Username>laimcon</Username>
      <Password>tzamaria7</Password>
      <DatabaseName>PhoneCatalog</DatabaseName>
      <Operations>
         <Operation>
            <OperationName>getTelephones</OperationName>
            <Query>SELECT PhoneCatalog.Telephone AS PhoneCatalog_Telephone
                   FROM  PhoneCatalog WHERE PhoneCatalog.FirstName=? AND
                   PhoneCatalog.LastName=?</Query>
            <Output>
               <Variable Name="PhoneCatalog_Telephone" Type="String"/>
            </Output>
            <Input>
               <Variable Name="input1" Type="String"/>
               <Variable Name="input2" Type="String"/>
            </Input>
         </Operation>
         <Operation>
            <OperationName>getAddresses</OperationName>
            <Query>SELECT PhoneCatalog.Address AS PhoneCatalog_Address
                   FROM  PhoneCatalog WHERE PhoneCatalog.FirstName=? AND
                   PhoneCatalog.LastName=?</Query>
            <Output>
               <Variable Name="PhoneCatalog_Address" Type="String"/>
            </Output>
            <Input>
               <Variable Name="input1" Type="String"/>
               <Variable Name="input2" Type="String"/>
            </Input>
         </Operation>
         <Operation>
            <OperationName>getDetails</OperationName>
            <Query>SELECT PhoneCatalog.Area AS PhoneCatalog_Area,
                   PhoneCatalog.Address AS PhoneCatalog_Address,
                   PhoneCatalog.Dimos AS PhoneCatalog_Dimos,
                   PhoneCatalog.Telephone AS PhoneCatalog_Telephone
                   FROM PhoneCatalog WHERE PhoneCatalog.FirstName=? AND
                   PhoneCatalog.LastName=?</Query>
            <Output>
               <Variable Name="PhoneCatalog_Area" Type="String"/>
               <Variable Name="PhoneCatalog_Address" Type="String"/>
               <Variable Name="PhoneCatalog_Dimos" Type="String"/>
               <Variable Name="PhoneCatalog_Telephone" Type="String"/>
            </Output>
            <Input>
               <Variable Name="input1" Type="String"/>
               <Variable Name="input2" Type="String"/>
            </Input>
         </Operation>
      </Operations>
      <ParentDirectory>c:/WebServiceDesigner</ParentDirectory>
      <EndPoint>phoneCatalog</EndPoint>
      <EndPointName>PhoneCatalog</EndPointName>
      <PackageName>phoneCatalog</PackageName>
      <ContextPath>phoneCatalog</ContextPath>
   </DatabaseService>
</Web_Service>
```

Το αρχείο αυτό περιέχει πλήρως όλα τα στοιχεία που δώσαμε κατά την εκτέλεση της εφαρμογής. Όπως βλέπουμε, περιέχει αρχικά τα απαραίτητα στοιχεία για τη σύνδεση με το ΣΔΒΔ καθώς και τη βάση την οποία χρησιμοποιεί το Web Service. Στη συνέχεια, περιέχει όλα τα operations που ορίσαμε και για κάθε operation περιέχει το όνομα του operation, το query που εκτελεί και τις μεταβλητές εισόδου και εξόδου με τους αντίστοιχους τύπους τους. Συγκεκριμένα, βλέπουμε ότι έχουμε 3 operations. Και τα 3 δέχονται ως είσοδο 2 strings (το όνομα και το επώνυμο κάποιου κατοίκου των Αθηνών) και επιστρέφουν το μεν πρώτο το τηλέφωνό του, το δε δεύτερο τη διεύθυνση του και το δε τρίτο την περιοχή, τη διεύθυνση, το δήμο και το τηλέφωνό του. Τέλος το *ConfigurationFile.xml*, περιέχει το όνομα του Web Service, το που είναι εγκατεστημένο (ContextPath – EndPoint) και το package στο οποίο τοποθετούνται οι κλάσεις τις οποίες κάνει compile το JWSDP κατά την εγκατάσταση του Web Service.

Αφού λοιπόν έχει δημιουργηθεί το αρχείο *ConfigurationFile.xml*, η εφαρμογή το στέλνει ως είσοδο στην κλάση *CreateServiceFiles* για να παραχθούν όλες οι απαραίτητες java classes και κάποια property files, τα οποία χρειάζεται το JWSDP για να κάνει deploy το Web Service. Τα αρχεία αυτά πρέπει να είναι το interface, ο implementor του interface και τα αρχεία jaxrpc-ri.xml και config.xml, τα οποία διαβάζουν το wsdeploy tool και wscompile tool του JWSDP. Παράγονται επίσης ό,τι java beans είναι απαραίτητα στα οποία αποθηκεύονται τα αποτελέσματα από την εκτέλεση των select queries.

Τα αρχεία λοιπόν που παράγονται στο συγκεκριμένο παράδειγμα εκτέλεσης της εφαρμογής είναι τα εξής:

ServiceBean1.java

Αφορά το πρώτο operation, το οποίο επιστρέφει το τηλέφωνο ανάλογα με το όνομα και το επώνυμο που δίνουμε ως είσοδο. Ο κώδικας του φαίνεται παρακάτω:

```
package phoneCatalog;

public class ServiceBean1 {
   private String PhoneCatalog_Telephone;

   public ServiceBean1()
   {
   }

   public void setPhoneCatalog_Telephone( String input )
   {
      PhoneCatalog_Telephone = input;
   }

   public String getPhoneCatalog_Telephone()
   {
      return PhoneCatalog_Telephone;
   }
}
```

ServiceBean2.java

Αφορά το δεύτερο operation, το οποίο επιστρέφει τη διεύθυνση ανάλογα με το όνομα και το επώνυμο που δίνουμε ως είσοδο. Ο κώδικας του φαίνεται παρακάτω:

```
package phoneCatalog;

public class ServiceBean2 {
    private String PhoneCatalog_Address;

    public ServiceBean2()
    {
    }

    public void setPhoneCatalog_Address( String input )
    {
        PhoneCatalog_Address = input;
    }

    public String getPhoneCatalog_Address()
    {
        return PhoneCatalog_Address;
    }
}
```

ServiceBean3.java

Αφορά το τρίτο operation, το οποίο επιστρέφει την περιοχή, τη διεύθυνση, το δήμο και το τηλέφωνο ανάλογα με το όνομα και το επώνυμο που δίνουμε ως είσοδο. Ο κώδικας του φαίνεται παρακάτω:

```
package phoneCatalog;

public class ServiceBean3 {
    private String PhoneCatalog_Area;
    private String PhoneCatalog_Address;
    private String PhoneCatalog_Dimos;
    private String PhoneCatalog_Telephone;

    public ServiceBean3()
    {
    }

    public void setPhoneCatalog_Area( String input )
    {
        PhoneCatalog_Area = input;
    }

    public void setPhoneCatalog_Address( String input )
    {
        PhoneCatalog_Address = input;
    }

    public void setPhoneCatalog_Dimos( String input )
    {
        PhoneCatalog_Dimos = input;
    }

    public void setPhoneCatalog_Telephone( String input )
    {
        PhoneCatalog_Telephone = input;
    }

    public String getPhoneCatalog_Area()
    {
        return PhoneCatalog_Area;
    }

    public String getPhoneCatalog_Address()
    {
        return PhoneCatalog_Address;
    }

    public String getPhoneCatalog_Dimos()
    {
        return PhoneCatalog_Dimos;
    }

    public String getPhoneCatalog_Telephone()
    {
        return PhoneCatalog_Telephone;
    }
}
```

ServiceIF.java

Αποτελεί το service definition interface. Ο κώδικας του φαίνεται παρακάτω:

```
package phoneCatalog;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ServiceIF extends Remote
{
   public ServiceBean1[] getTelephones( String input1, String
input2, int numberOfHundreds ) throws RemoteException;

   public ServiceBean2[] getAddresses( String input1, String input2,
int numberOfHundreds ) throws RemoteException;

   public ServiceBean3[] getDetails( String input1, String input2, int
numberOfHundreds ) throws RemoteException;
}
```

Το interface δείχνει στην ουσία ότι έχουμε 3 operations με όνοματα getTelephones, getAddresses και getDetails, κάθε μία από τις οποίες δέχεται ως είσοδο 2 strings (το όνομα και το επώνυμο) καθώς και έναν αριθμό που δηλώνει ποια εκατοντάδα θέλουμε να επιστρέψει το πρόγραμμα. Π.χ. αν έχουμε ένα query που κανονικά θα επέστρεφε 1.000 εγγραφές, εμείς δίνοντας σαν μεταβλητή για το numberOfHundreds το 1, δηλώνουμε ότι θέλουμε το πρόγραμμα να μας επιστρέψει τις 100 πρώτες εγγραφές. Αυτό γίνεται για να μην μεταφέρεται από τον server στον client υπερβολικά μεγάλος αριθμός εγγραφών, κάτι που θα μπορούσε σε περιπτώσεις βάσεων με εκατομμύρια δεδομένων να υπερφορτώσει το δίκτυο.

ServiceImpl.java

Το αρχείο αυτό είναι στην ουσία και ο wrapper του Web Service. Περιέχει ακριβώς όλες τις μεθόδους που ορίζει το ServiceIF.java με τον κώδικα που εκτελεί η κάθε μία, δηλαδή το query στη βάση και την αποθήκευση των δεδομένων που επιστρέφονται στο κατάλληλο java bean. Ο κώδικας του φαίνεται παρακάτω:

```
package phoneCatalog;

import java.sql.*;
import java.util.*;
import java.rmi.*;
import javax.xml.rpc.server.*;
import javax.xml.rpc.JAXRPCException;

public class ServiceImpl implements ServiceIF {
    private Connection connection;
    private PreparedStatement query;

    public ServiceImpl()
    {
        try
        {
            String url =
"jdbc:microsoft:sqlserver://KOSTASLAPTOP:1433;DatabaseName=PhoneCatalog";
            String userName = "laimcon";
            String password = "tzamaria7";

            Class.forName( "com.microsoft.jdbc.sqlserver.SQLServerDriver" );

            connection = DriverManager.getConnection( url, userName, password );
        }
        catch ( ClassNotFoundException e1 )
        {
            e1.printStackTrace();
            throw new JAXRPCException( e1.getMessage() );
        }
        catch ( SQLException e2 )
        {
            e2.printStackTrace();
            throw new JAXRPCException( e2.getMessage() );
        }
    }
```

```
    public ServiceBean1[] getTelephones( String input1, String input2, int
numberOfHundreds )
    {
        try
        {
            query = connection.prepareStatement( "SELECT PhoneCatalog.Telephone AS
PhoneCatalog_Telephone FROM  PhoneCatalog WHERE PhoneCatalog.FirstName=? AND
PhoneCatalog.LastName=?" );
            query.setString( 1, input1 );
            query.setString( 2, input2 );

            ResultSet resultSet = query.executeQuery();

            List serviceInformation = new ArrayList();

            int start = ( numberOfHundreds - 1 ) * 100;
            int end = numberOfHundreds * 100;
            int count = 0;

            while ( resultSet.next() )
            {
                count++;

                if ( ( count > start ) && ( count <= end ) )
                {
                    ServiceBean1 serviceBean = new ServiceBean1();
                    serviceBean.setPhoneCatalog_Telephone( resultSet.getString(
"PhoneCatalog_Telephone" ) );
                    serviceInformation.add( serviceBean );
                }
            }

            ServiceBean1[] serviceBeans = new ServiceBean1[ serviceInformation.size() ];

            serviceInformation.toArray( serviceBeans );

            return serviceBeans;
        }
        catch ( SQLException e )
        {
            e.printStackTrace();
            return null;
        }
    }
```

```
    public ServiceBean2[] getAddresses( String input1, String input2, int
numberOfHundreds )
    {
        try
        {
            query = connection.prepareStatement( "SELECT PhoneCatalog.Address AS
PhoneCatalog_Address FROM  PhoneCatalog WHERE PhoneCatalog.FirstName=? AND
PhoneCatalog.LastName=?" );
            query.setString( 1, input1 );
            query.setString( 2, input2 );

            ResultSet resultSet = query.executeQuery();

            List serviceInformation = new ArrayList();

            int start = ( numberOfHundreds - 1 ) * 100;
            int end = numberOfHundreds * 100;
            int count = 0;

            while ( resultSet.next() )
            {
                count++;

                if ( ( count > start ) && ( count <= end ) )
                {
                    ServiceBean2 serviceBean = new ServiceBean2();
                    serviceBean.setPhoneCatalog_Address( resultSet.getString(
"PhoneCatalog_Address" ) );
                    serviceInformation.add( serviceBean );
                }
            }

            ServiceBean2[] serviceBeans = new ServiceBean2[ serviceInformation.size() ];

            serviceInformation.toArray( serviceBeans );

            return serviceBeans;
        }
        catch ( SQLException e )
        {
            e.printStackTrace();
            return null;
        }
    }
```

```
    public ServiceBean3[] getDetails( String input1, String input2, int numberOfHundreds )
    {
        try
        {
            query = connection.prepareStatement( "SELECT PhoneCatalog.Area AS
PhoneCatalog_Area, PhoneCatalog.Address AS PhoneCatalog_Address, PhoneCatalog.Dimos AS
PhoneCatalog_Dimos, PhoneCatalog.Telephone AS PhoneCatalog_Telephone FROM  PhoneCatalog
WHERE PhoneCatalog.FirstName=? AND PhoneCatalog.LastName=?" );
            query.setString( 1, input1 );
            query.setString( 2, input2 );

            ResultSet resultSet = query.executeQuery();

            List serviceInformation = new ArrayList();

            int start = ( numberOfHundreds - 1 ) * 100;
            int end = numberOfHundreds * 100;
            int count = 0;

            while ( resultSet.next() )
            {
                count++;

                if ( ( count > start ) && ( count <= end ) )
                {
                    ServiceBean3 serviceBean = new ServiceBean3();
                    serviceBean.setPhoneCatalog_Area( resultSet.getString(
"PhoneCatalog_Area" ) );
                    serviceBean.setPhoneCatalog_Address( resultSet.getString(
"PhoneCatalog_Address" ) );
                    serviceBean.setPhoneCatalog_Dimos( resultSet.getString(
"PhoneCatalog_Dimos" ) );
                    serviceBean.setPhoneCatalog_Telephone( resultSet.getString(
"PhoneCatalog_Telephone" ) );
                    serviceInformation.add( serviceBean );
                }
            }

            ServiceBean3[] serviceBeans = new ServiceBean3[
serviceInformation.size() ];

            serviceInformation.toArray( serviceBeans );

            return serviceBeans;
        }
        catch ( SQLException e )
        {
            e.printStackTrace();
            return null;
        }
    }
}
```

jaxrpc-ri.xml

Αποτελεί ένα configuration file το οποίο διαβάζει το wsdeploy tool του JWSDP. Περιέχει βασικές πληροφορίες απαραίτητες για την εγκατάσταση του Web Service. Το αρχείο αυτό φαίνεται παρακάτω:

```
<?xml version="1.0" encoding="UTF-8"?>
<webServices
    xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/dd"
    version="1.0"
    targetNamespaceBase="http://com.test/wsdl"
    typeNamespaceBase="http://com.test/types"
    urlPatternBase="/ws">

    <endpoint
        name="PhoneCatalog"
        displayName="Web Service"
        description="A simple web service"
        interface="phoneCatalog.ServiceIF"
        implementation="phoneCatalog.ServiceImpl"/>

    <endpointMapping
        endpointName="PhoneCatalog"
        urlPattern="/phoneCatalog"/>

    </webServices>
```

Όπως παρατηρούμε το αρχείο αυτό περιέχει πληροφορίες όπως το url στο οποίο θα γίνει deploy το Web Service, το όνομα του interface και του interface implementor καθώς και το όνομα του Web Service.

Config.xml

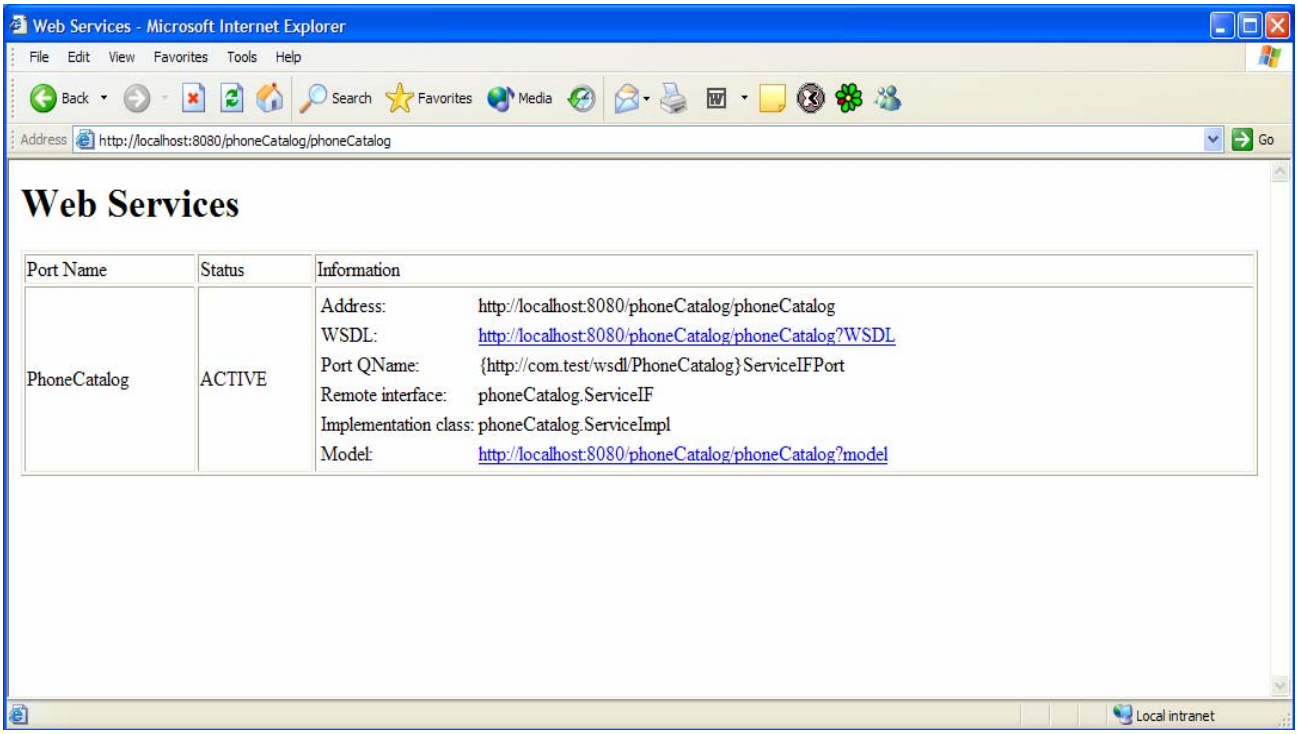
Αποτελεί ένα configuration file το οποίο διαβάζει το wscompile tool του JWSDP. Περιέχει βασικές πληροφορίες απαραίτητες κατά την δημιουργία των stubs που χρησιμοποιεί κάποιος client. Το αρχείο αυτό φαίνεται παρακάτω:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
    xmlns = "http://java.sun.com/xml/ns/jax-rpc/ri/config">
    <wsdl location =
"http://localhost:8080/phoneCatalog/phoneCatalog?WSDL"
        packageName = "phoneCatalog">
    </wsdl>
    </configuration>
```

Όπως παρατηρούμε το αρχείο αυτό περιέχει το url στο οποίο βρίσκεται το WSDL αρχείο του Web Service καθώς και το package στο οποίο πρέπει να αποθηκεύσει τις κλάσεις που θα προκύψουν από τη μεταγλώττιση.

Όλα αυτά τα αρχεία παράγονται δυναμικά μόλις ο χρήστης δώσει όλα τα στοιχεία που του ζητάει η εφαρμογή. Αυτό είναι και το μεγάλο πλεονέκτημα της εφαρμογής μας αφού ο χρήστης δεν ασχολείται καν με την παραγωγή των αρχείων αυτών (τα οποία πρέπει να έχουν συγκεκριμένη μορφή) αλλά αρκείται στο να δώσει μόνο τα απαραίτητα στοιχεία για την παραγωγή τους.

Μετά λοιπόν την παραγωγή των αρχείων αυτών, αναλαμβάνει δουλειά το JWSDP και κάνει deploy το Web Service.Για να δούμε αν όλα έγιναν σωστά πάμε στο url που είχαμε ορίσει δηλαδή στο http://localhost:8080/phoneCatalog/phoneCatalog οπότε και βλέπουμε τα εξής:



Βλέπουμε λοιπόν ότι το Web Service εγκαταστάθηκε κανονικά και κάνοντας click στο http://localhost:8080/phoneCatalog/phoneCatalog?WSDL μπορούμε να δούμε το WSDL αρχείο του Web Service. Το αρχείο αυτό φαίνεται στην επόμενη σελίδα:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://com.test/wsdl/PhoneCatalog"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:ns2="http://com.test/types/PhoneCatalog" name="PhoneCatalog"
targetNamespace="http://com.test/wsdl/PhoneCatalog">
  <types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://com.test/types/PhoneCatalog"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:soap-
enc="http://schemas.xmlsoap.org/soap/encoding/"
targetNamespace="http://com.test/types/PhoneCatalog">
      <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
      <complexType name="ArrayOfServiceBean2">
        <complexContent>
          <restriction base="soap-enc:Array">
            <attribute ref="soap-enc:arrayType"
wsdl:arrayType="tns:ServiceBean2[]"/></restriction></complexContent></complexType>
      <complexType name="ServiceBean2">
        <sequence>
          <element name="phoneCatalog_Address"
type="string"/></sequence></complexType>
      <complexType name="ArrayOfServiceBean3">
        <complexContent>
          <restriction base="soap-enc:Array">
            <attribute ref="soap-enc:arrayType"
wsdl:arrayType="tns:ServiceBean3[]"/></restriction></complexContent></complexType>
      <complexType name="ServiceBean3">
        <sequence>
          <element name="phoneCatalog_Address" type="string"/>
          <element name="phoneCatalog_Area" type="string"/>
          <element name="phoneCatalog_Dimos" type="string"/>
          <element name="phoneCatalog_Telephone"
type="string"/></sequence></complexType>
      <complexType name="ArrayOfServiceBean1">
        <complexContent>
          <restriction base="soap-enc:Array">
            <attribute ref="soap-enc:arrayType"
wsdl:arrayType="tns:ServiceBean1[]"/></restriction></complexContent></complexType>
      <complexType name="ServiceBean1">
        <sequence>
          <element name="phoneCatalog_Telephone"
type="string"/></sequence></complexType></schema></types>
  <message name="ServiceIF_getAddresses">
    <part name="String_1" type="xsd:string"/>
    <part name="String_2" type="xsd:string"/>
    <part name="int_3" type="xsd:int"/></message>
  <message name="ServiceIF_getAddressesResponse">
    <part name="result" type="ns2:ArrayOfServiceBean2"/></message>
  <message name="ServiceIF_getDetails">
    <part name="String_1" type="xsd:string"/>
    <part name="String_2" type="xsd:string"/>
    <part name="int_3" type="xsd:int"/></message>
  <message name="ServiceIF_getDetailsResponse">
    <part name="result" type="ns2:ArrayOfServiceBean3"/></message>
  <message name="ServiceIF_getTelephones">
    <part name="String_1" type="xsd:string"/>
    <part name="String_2" type="xsd:string"/>
    <part name="int_3" type="xsd:int"/></message>
  <message name="ServiceIF_getTelephonesResponse">
    <part name="result" type="ns2:ArrayOfServiceBean1"/></message>
```

```
<portType name="ServiceIF">
    <operation name="getAddresses" parameterOrder="String_1 String_2 int_3">
      <input message="tns:ServiceIF_getAddresses"/>
      <output message="tns:ServiceIF_getAddressesResponse"/></operation>
    <operation name="getDetails" parameterOrder="String_1 String_2 int_3">
      <input message="tns:ServiceIF_getDetails"/>
      <output message="tns:ServiceIF_getDetailsResponse"/></operation>
    <operation name="getTelephones" parameterOrder="String_1 String_2 int_3">
      <input message="tns:ServiceIF_getTelephones"/>
      <output message="tns:ServiceIF_getTelephonesResponse"/></operation></portType>
  <binding name="ServiceIFBinding" type="tns:ServiceIF">
    <operation name="getAddresses">
      <input>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="http://com.test/wsdl/PhoneCatalog"/></input>
      <output>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="http://com.test/wsdl/PhoneCatalog"/></output>
      <soap:operation soapAction=""/></operation>
    <operation name="getDetails">
      <input>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="http://com.test/wsdl/PhoneCatalog"/></input>
      <output>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="http://com.test/wsdl/PhoneCatalog"/></output>
      <soap:operation soapAction=""/></operation>
    <operation name="getTelephones">
      <input>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="http://com.test/wsdl/PhoneCatalog"/></input>
      <output>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="http://com.test/wsdl/PhoneCatalog"/></output>
      <soap:operation soapAction=""/></operation>
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="rpc"/></binding>
  <service name="PhoneCatalog">
    <port name="ServiceIFPort" binding="tns:ServiceIFBinding">
      <soap:address xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
location="http://localhost:8080/phoneCatalog/phoneCatalog"/></port></service></defin
itions>
```

Για να δούμε τώρα ότι το Web Service που σχεδιάσαμε δουλεύει όντως σωστά θα προγραμματίσουμε έναν client. Υπάρχουν πολλοί τρόποι για να προγραμματίσει κάποιος έναν client, ο οποίος να έχει πρόσβαση σε ένα Web Service, από την στιγμή που έχει στα χέρια του το WSDL αρχείο που περιγράφει το service. Εμείς θα χρησιμοποιήσουμε και πάλι το JWSDP.

Ας δούμε τώρα το σενάριο του Web Service. Με τον τρόπο που περιγράψαμε παραπάνω, εγκαταστήσαμε ένα Web Service (το *PhoneCatalog* Web Service), το οποίο έχει πρόσβαση σε μια βάση με τα τηλέφωνα όλων των κατοίκων της Αθήνας και το οποίο υποστηρίζει τα εξής 3 operations:

1. Την εύρεση του τηλεφώνου κάποιου κατοίκου των Αθηνών, δίνοντας ως είσοδο το Όνομα και το Επώνυμο του.
2. Την εύρεση της διεύθυνσης κάποιου κατοίκου των Αθηνών, δίνοντας ως είσοδο το Όνομα και το Επώνυμο του.
3. Την προβολή όλων των σημαντικών στοιχείων κάποιου κατοίκου των Αθηνών και συγκεκριμένα της περιοχής στην οποία μένει, το δήμο, τη διεύθυνση και το τηλέφωνό του.

Υποθέτουμε τώρα ότι έχουμε και μια δεύτερη βάση από το Υπουργείο Συγκοινωνιών με όλα τα αυτοκίνητα της Ελλάδας. Θεωρούμε σημαντικό να φτιάξουμε ένα Web Service (το *CarOwners* Web Service) , το οποίο να υποστηρίζει τις εξής 2 operations:

1. Την εύρεση του ιδιοκτήτη ενός αυτοκινήτου, δίνοντας ως είσοδο τα γράμματα και τον αριθμό της πινακίδας.
2. Την τροποποίηση των στοιχείων του ιδιοκτήτη ενός αυτοκινήτου, δίνοντας πάλι ως είσοδο τα γράμματα και τον αριθμό της πινακίδας.

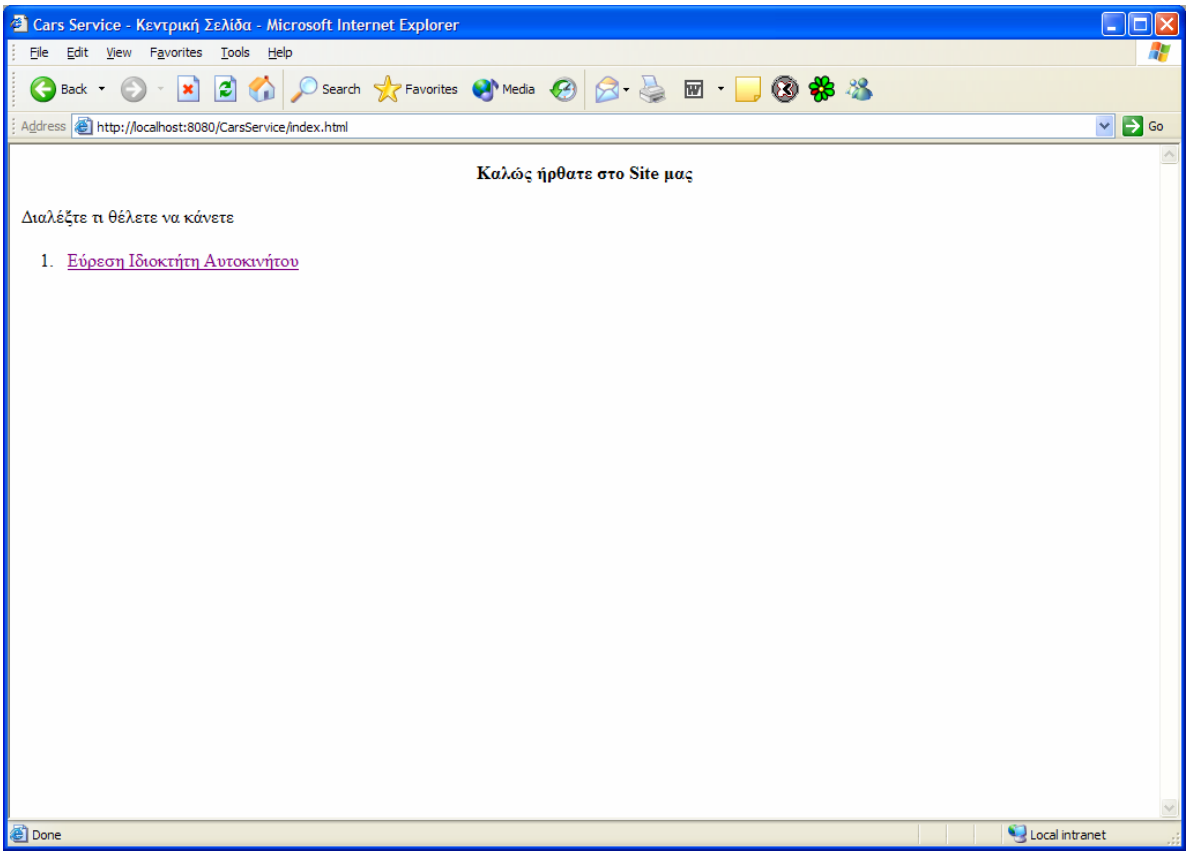
Το Web Service το κατασκευάζουμε ακριβώς με τον τρόπο που περιγράψαμε και στην περίπτωση του Web Service με τον τηλεφωνικό κατάλογο.

Κατασκευάζουμε λοιπόν έναν client, ο οποίος να έχει πρόσβαση και στα 2 Web Service και το σημαντικότερο να παίρνει κάποια στοιχεία από το πρώτο Web Service και να τα δίνει ως είσοδο σε κάποιο operation του δεύτερου Web Service! Ένα καλό σενάριο είναι το εξής:

1. Δίνουμε τα γράμματα και τον αριθμό πινακίδας ενός αυτοκινήτου ως είσοδο στο Web Service *CarOwners* οπότε και παίρνουμε το Όνομα και το Επώνυμο του ιδιοκτήτη.
2. Έχοντας τώρα το Όνομα και το Επώνυμο του ιδιοκτήτη, μπορούμε να το δώσουμε ως είσοδο στο Web Service *PhoneCatalog*, για να πάρουμε το τηλέφωνο του ή τη διεύθυνσή του ή και αναλυτικά τα στοιχεία του.

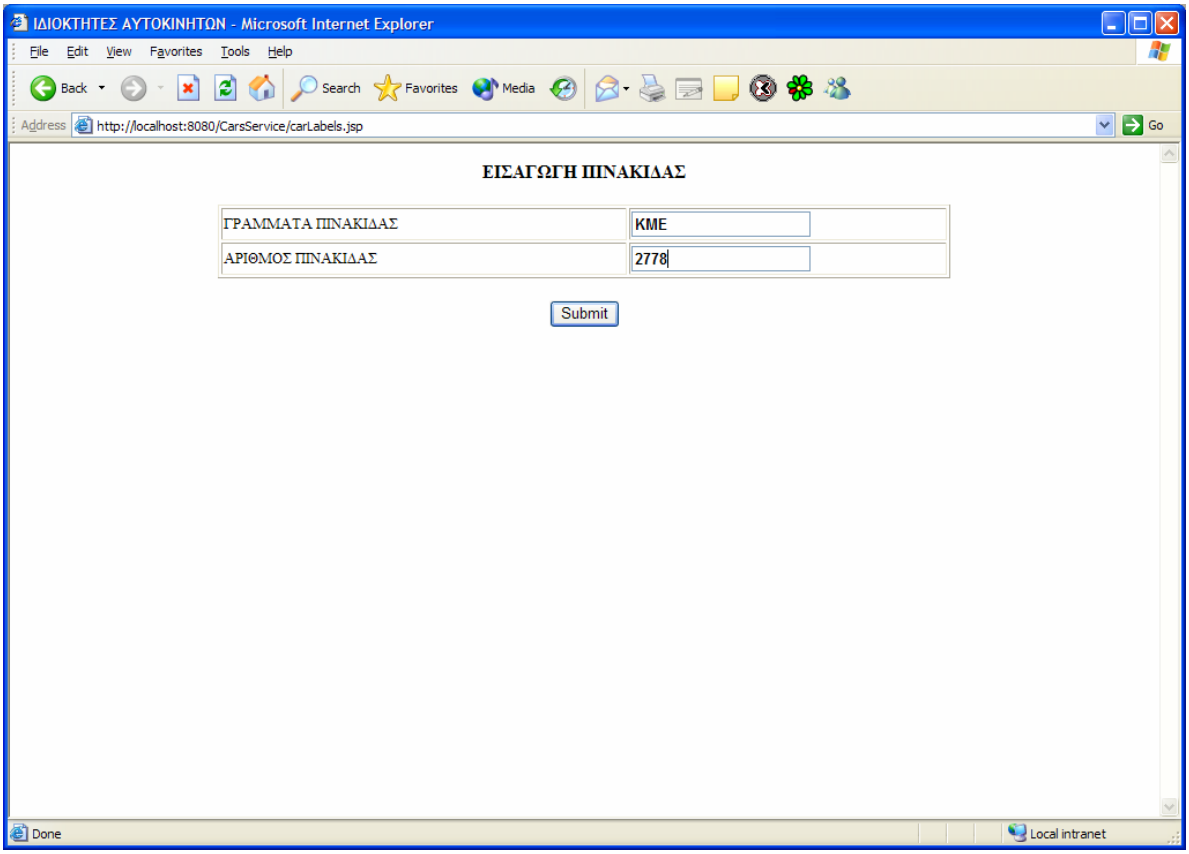
Παρακάτω θα δείξουμε και γραφικά το «τρέξιμο» αυτού του σεναρίου. Φτιάξαμε κατάλληλες jsp σελίδες έτσι ώστε να μπορεί ο οποιοσδήποτε που έχει πρόσβαση στο Internet να συνδεθεί με το site μας και να χρησιμοποιήσει το Web Service.

Η αρχική σελίδα φαίνεται στο Σχήμα 5:



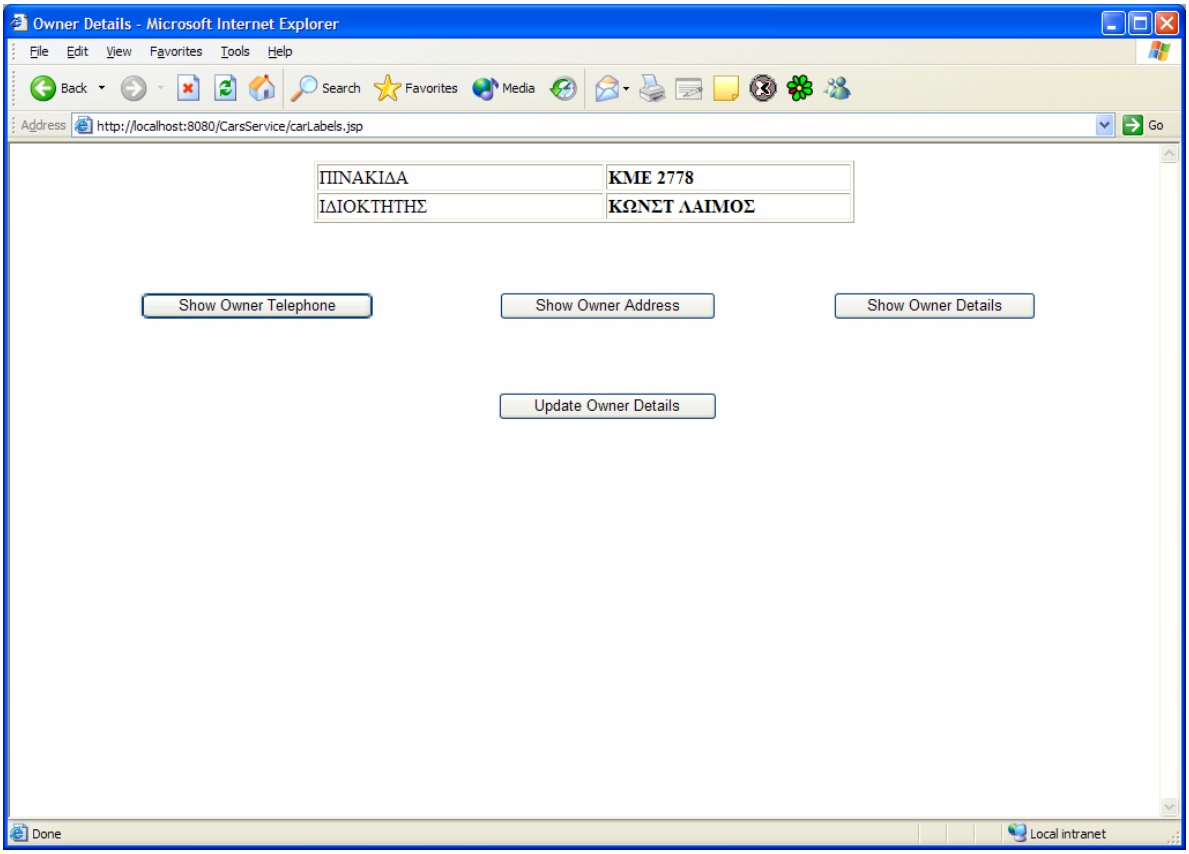
Σχήμα 5

Η σελίδα αυτή θα μπορούσε να ήταν η αρχική σελίδα μιας υπηρεσίας του Υπουργείου Συγκοινωνιών, που να υποστήριζε διάφορες λειτουργίες ανεύρεσης στοιχείων αυτοκινήτων. Στην περίπτωση μας υποστηρίζεται μόνο η λειτουργία της εύρεσης του ιδιοκτήτη ενός αυτοκινήτου. Κάνουμε click στο «Εύρεση Ιδιοκτήτη Αυτοκινήτου» οπότε και μεταφερόμαστε στη δεύτερη σελίδα, η οποία φαίνεται στο Σχήμα 6:



Σχήμα 6

Στη σελίδα αυτή δίνουμε ως δεδομένα τα γράμματα και τον αριθμό πινακίδας ενός αυτοκινήτου. Στη συνέχεια πατώντας το κουμπί **Submit** τα δεδομένα αυτά δίνονται ως είσοδο στο 1^{ο} operation του Web Service *CarOwners*, οπότε και παίρνουμε τον ιδιοκτήτη του συγκεκριμένου αυτοκινήτου. Το αποτέλεσμα της ενέργειας αυτής φαίνεται στο Σχήμα 7:



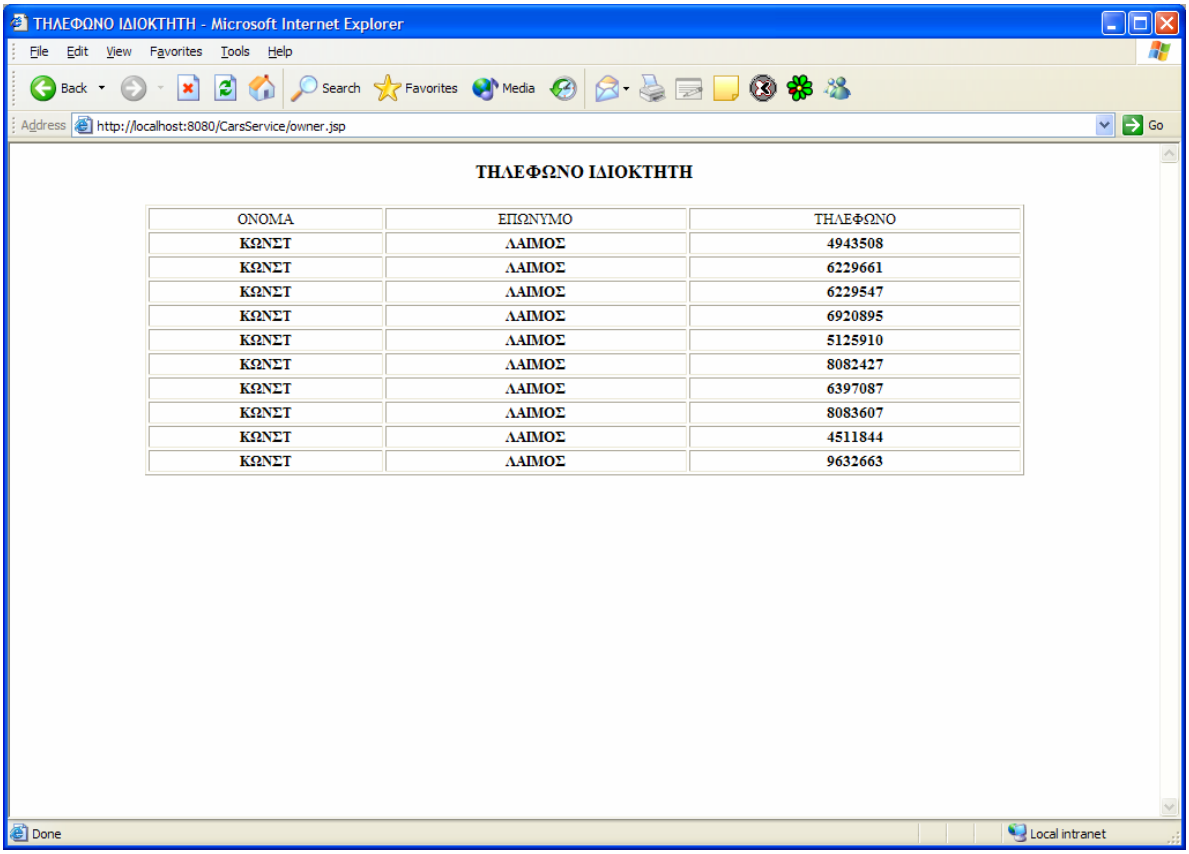
Σχήμα 7

Έχοντας τώρα το Όνομα και το Επώνυμο του ιδιοκτήτη έχουμε τις εξής επιλογές:

1. Να το δώσουμε ως είσοδο στο 1ο operation του Web Service *PhoneCatalog*, οπότε και να πάρουμε το τηλέφωνό του.
2. Να το δώσουμε ως είσοδο στο 2ο operation του Web Service *PhoneCatalog*, οπότε και να πάρουμε τη διεύθυνσή του.
3. Να το δώσουμε ως είσοδο στο 3ο operation του Web Service *PhoneCatalog*, οπότε και να πάρουμε τα αναλυτικά του στοιχεία και συγκεκριμένα την περιοχή στην οποία μένει, το δήμο, τη διεύθυνση και το τηλέφωνό του.

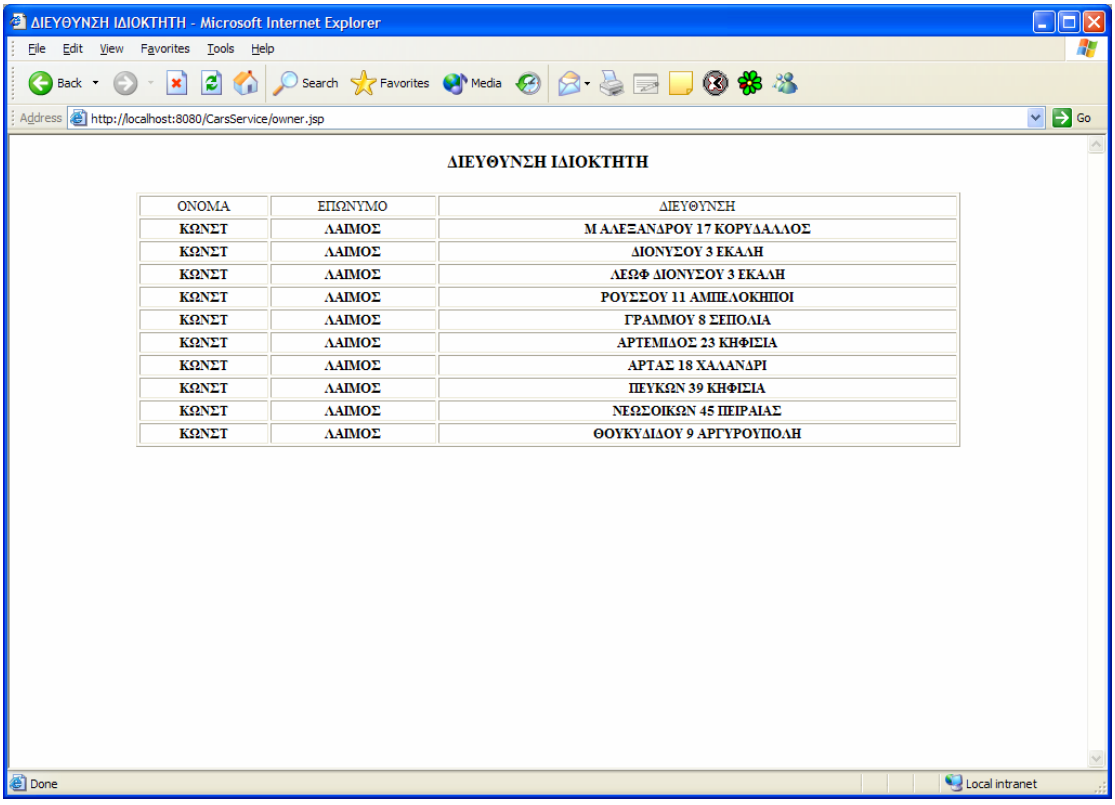
Κάθε μια από αυτές τις ενέργειες πραγματοποιείται πατώντας το αντίστοιχο κουμπί οπότε έχουμε τα εξής αποτελέσματα:

Πατώντας το κουμπί **Show Owner Telephone** μεταφερόμαστε στη σελίδα που φαίνεται στο σχήμα 8 (προφανώς επειδή δεν υπάρχει μόνο ένας κάτοικος Αθηναίων με το ονοματεπώνυμο ΚΩΝΣΤ ΛΑΙΜΟΣ παίρνουμε παραπάνω από ένα αποτελέσματα!) :



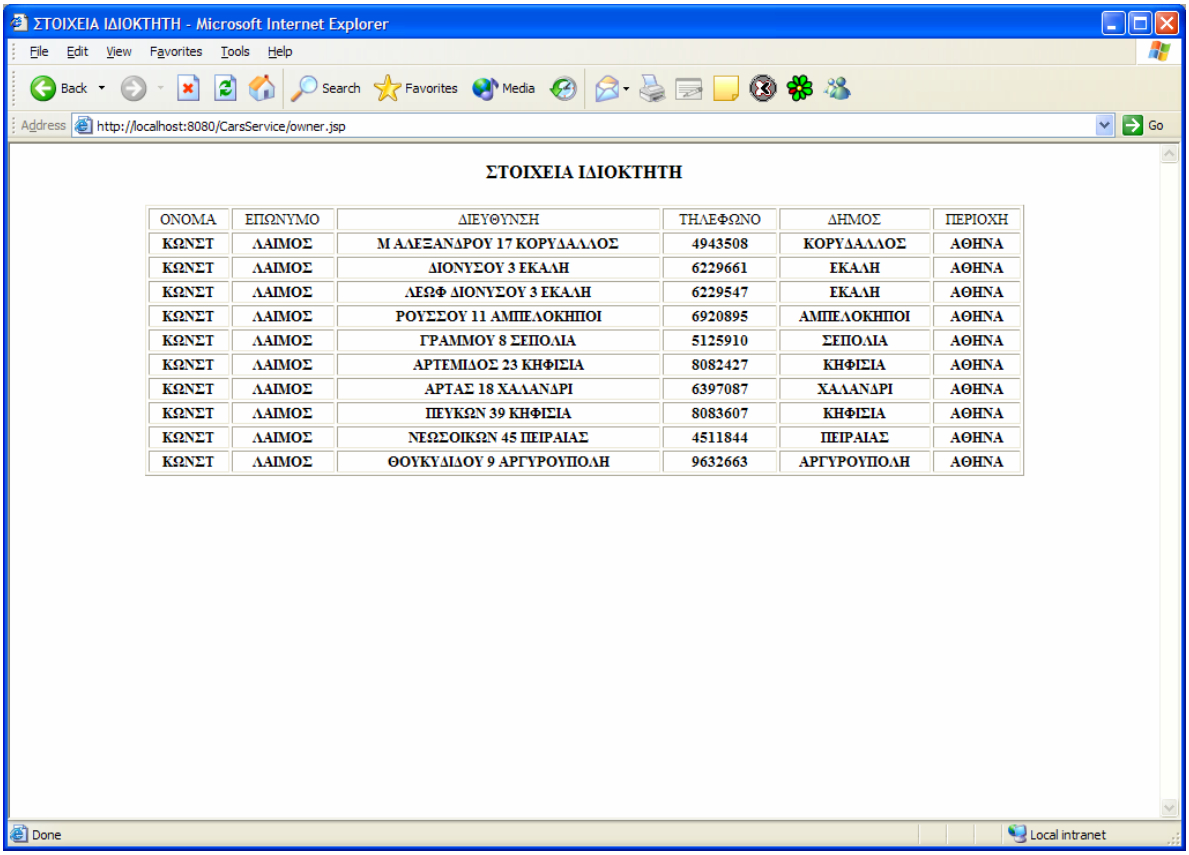
Σχήμα 8

Πατώντας το κουμπί **Show Owner Address** πάμε στη σελίδα που φαίνεται στο Σχήμα 9:



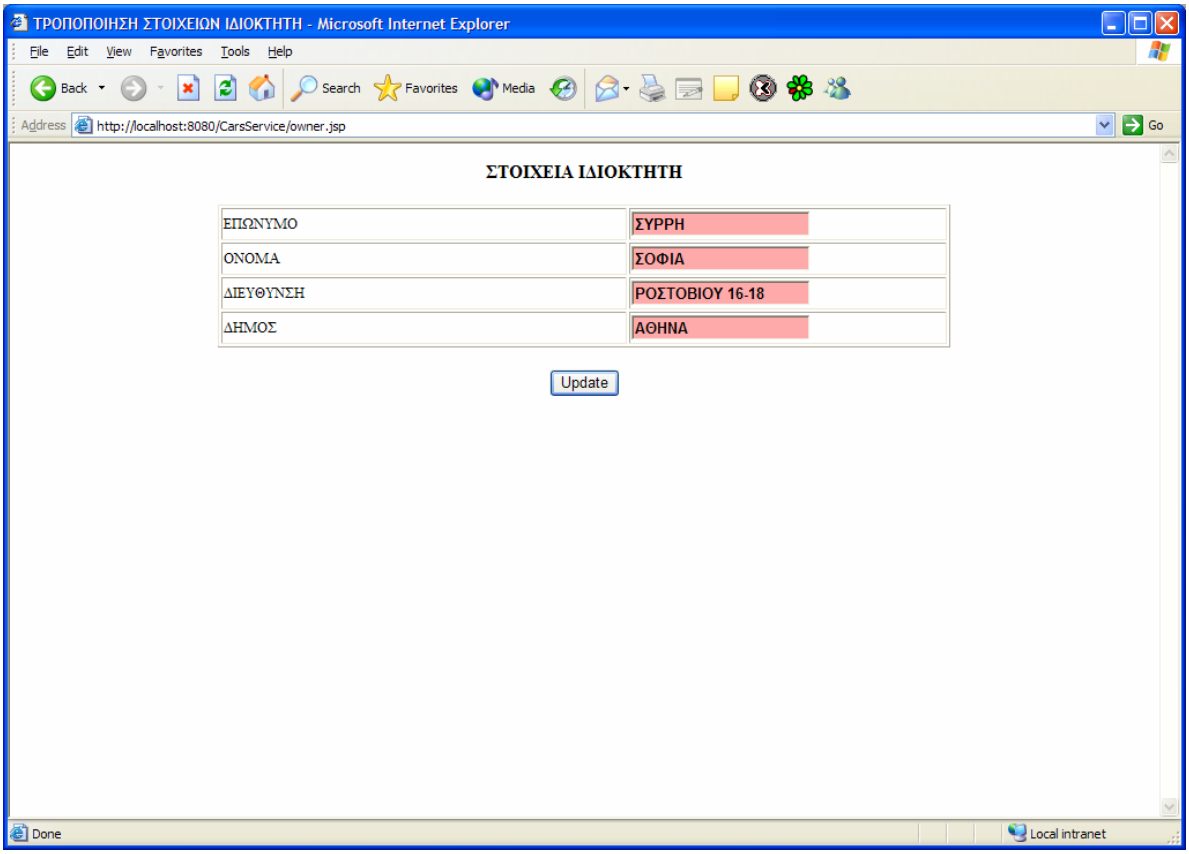
Σχήμα 9

Πατώντας το κουμπί **Show Owner Details** πάμε στη σελίδα που φαίνεται στο Σχήμα 10:



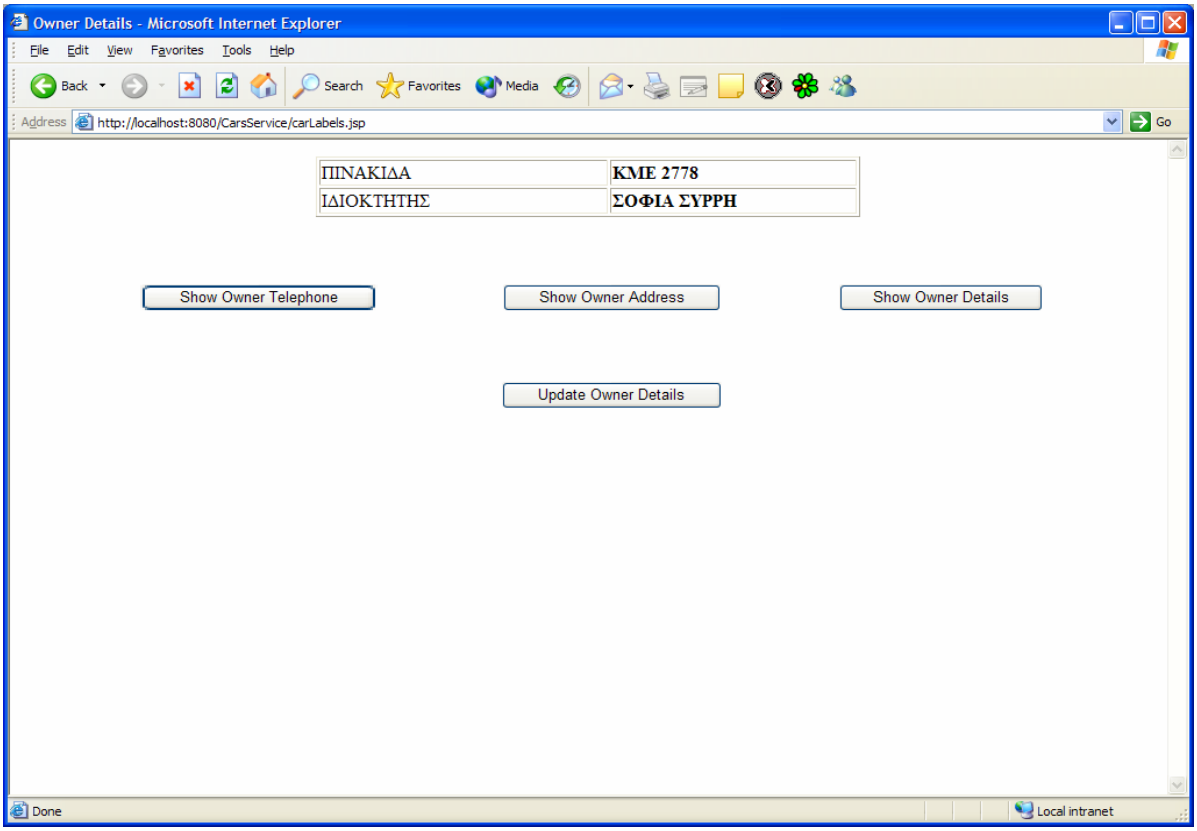
Σχήμα 10

Πατώντας τέλος το κουμπί **Update Owner Details** έχουμε τη δυνατότητα να χρησιμοποιήσουμε το 2^{ο} operation του Web Service *CarOwners*, να τροποποιήσουμε δηλαδή τα στοιχεία του ιδιοκτήτη του αυτοκινήτου. Αυτό το operation θα ήταν χρήσιμο για την περίπτωση που ο αρχικός ιδιοκτήτης του αυτοκινήτου αποφάσιζε να το πουλήσει, οπότε και θα έπρεπε να ενημερωθεί η βάση με τον καινούργιο ιδιοκτήτη. Η σελίδα στην οποία μεταφερόμαστε λοιπόν πατώντας κουμπί **Update Owner Details** φαίνεται στο Σχήμα 11:



Σχήμα 11

Αν τώρα κάναμε ξανά αναζήτηση για το αυτοκίνητο με αριθμό πινακίδας ΚΜΕ 2778 θα παίρναμε το παρακάτω αποτέλεσμα, το οποίο δείχνει ότι η ενημέρωση έγινε επιτυχώς:



Είδαμε λοιπόν ένα ολοκληρωμένο παράδειγμα στησίματος και χρήσης ενός Web Service. Είδαμε επίσης και ένα από τα σημαντικά στοιχεία των Web Services. Ότι μπορούν να συνδυαστούν μεταξύ τους! Το γεγονός ότι πήραμε κάποια δεδομένα από το πρώτο Web Service και τα χρησιμοποιήσαμε στο δεύτερο δείχνει ακριβώς αυτό.

7

Επίλογος

Στην ενότητα αυτή συνοψίζουμε τα αποτελέσματα τις διπλωματικής εργασίας και παραθέτουμε τις μελλοντικές επεκτάσεις που έχουν νόημα και ενδιαφέρον να υλοποιηθούν.

7.1 Σύνοψη και συμπεράσματα

Είδαμε λοιπόν ότι αναπτύχθηκε μια λογισμική εφαρμογή, η οποία επιτρέπει στον κάτοχο μιας Βάσης Δεδομένων να κάνει publish κάποια από τα δεδομένα της βάσης του ως Web Service. Είδαμε επίσης ότι καταφέραμε να κρύψουμε τελείως από τον κάτοχο της βάσης τον τρόπο με τον οποίο δημιουργείται το Web Service, απαλλάσσοντας τον από προγραμματισμό υψηλού επιπέδου. Το μόνο που αρκείται να κάνει είναι η επιλογή των δεδομένων που θέλει να δημοσιοποιήσει (γράφοντας τα κατάλληλα SQL Queries) και στη συνέχεια το σύστημα μας αναλαμβάνει αυτόματα τη δημιουργία του. Η ευκολία αυτή επιτείνεται με μια αρκετά εύχρηστη και λειτουργική διαπροσωπεία με το χρήστη.

Γενικά λοιπόν, όπως παρουσιάστηκε στα προηγούμενα κεφάλαια, το σύστημα μας αποτελεί μια ικανοποιητική λύση για τη δημοσιοποίηση των δεδομένων μιας Βάσης Δεδομένων ως Web Services.

7.2 Μελλοντικές επεκτάσεις

Οι επεκτάσεις και οι βελτιώσεις που μπορούν να γίνουν στο σύστημά μας, έτσι ώστε αυτό να γίνει ολοκληρωμένο και να μπορεί να ανταγωνιστεί άλλες υλοποιήσεις συστημάτων για την παρασκευή Web Services είναι πάρα πολλές και απαιτούν για την υλοποίησή τους αρκετές ώρες εργασίας και προσπάθειας. Εμείς, ωστόσο, αναφέρουμε στη συνέχεια μερικές από αυτές που τις θεωρούμε πιο σημαντικές.

- Το πιο σημαντικό αφορά θέματα security. Το θέμα security χωρίζεται σε 2 κατηγορίες
  - **Authentication**: Στη δική μας υλοποίηση από τη στιγμή που ο κάτοχος της βάσης κάνει publish τα δεδομένα του ως Web Service έχοντας ορίσει τις αντίστοιχες operations που μπορεί να τρέξει ένας client, η Web Service είναι πλέον δημόσια για όλους. Μπορεί δηλαδή ο οποιοσδήποτε, έχοντας κατεβάσει το WSDL αρχείο που περιγράφει τη Web Service, να φτιάξει έναν client και να έχει πρόσβαση σε όλες τις operations που ορίζει η Web Service. Θα ήταν σημαντικό λοιπόν το σύστημα μας να υποστηρίζει κάποιο μηχανισμό authentication με τον οποίο να επιτρέπει σε ορισμένους μόνο χρήστες να έχουν πρόσβαση στη Web Service, αλλά και σε αυτούς που έχουν πρόσβαση να μπορούν να χρησιμοποιήσουν κάποιες από τις operations που έχουν οριστεί ή όλες.
  - **Encryption**: Καλό θα ήταν επίσης να υπάρχει ένας μηχανισμός encryption των δεδομένων που έρχονται από τη βάση, ώστε να μην μπορεί κάποιος να τα υποκλέψει.
- Μια άλλη μελλοντική επέκταση θα ήταν το WSDL αρχείο που περιγράφει τη Web Service να γίνεται publish σε μια UDDI registry. Να ρωτάται δηλαδή ο χρήστης αν μετά το deployment της Web Service θέλει να την κάνει publish και σε μια UDDI registry.

8

Βιβλιογραφία

R. Elmasri – S.B. Navathe – Θεμελιώδεις αρχές συστημάτων Βάσεων Δεδομένων – Τόμος Α’ & Β’

Εμμ. Σκορδαλάκης – Εισαγωγή στην Τεχνολογία Λογισμικού

Στ. Ζάχος – Προγραμματιστικές Τεχνικές

Deitel Developer Series – Java™ How to Program

Deitel Developer Series – Advanced Java™ 2 Platform How to Program

Deitel Developer Series – Java Web Services for Experienced Programmers

Sun Microsystems – The Java Tutorial – available at http://java.sun.com

Sun Microsystems – The Java 2 Standard Edition Documentation – available at http://java.sun.com

Sun Microsystems – The Java 2 Enterprise Edition Documentation – available at http://java.sun.com

Microsoft - SQL Server 2000 Documentation – available at http://www.microsoft.com

Web Services Protocols Tutorial – available at http://www.w3schools.com

World Wide Web Consortium papers – available at http://www.w3c.com