

# Διαδίκτυο και Εφαρμογές

[www.cn.ntua.gr](http://www.cn.ntua.gr)

Γ. Ι. Στασινόπουλος  
ΕΜΠ, τηλ. 210 772 2531  
stassin@cs.ntua.gr

1. Προγραμματισμός για TCP - Sockets .....	3
Στοιχειώδης Server .....	3
Στοιχειώδης Client.....	6
2. Πρόσβαση σε Διαδικτυακές Εφαρμογές .....	10
HTTP (HyperText Transfer Protocol) .....	10
Παρακολούθηση Μηνυμάτων με TCP Monitor.....	14
SMTP (Simple Mail Transport Protocol - RFC 821) .....	15
Διάφορα .....	16
Half-close για Δήλωση Τερματισμού Αίτησης.....	16
MIME για Αποστολές πέραν του Απλού Κειμένου .....	16
3. Προγραμματισμός σε Επίπεδο URL.....	17
Διαφοροποίηση μεταξύ URIs, URLs και URNs.....	17
Άντληση Πληροφορίας από Web Server με URLConnection.....	18
Αποστολή Πληροφορίας σε Web Server με URLConnection .....	21
4. Η Πλευρά του Server.....	22
To Servlet.....	22
To Servlet μέσα στον Server .....	25
Χρήση Cookies και Sessions .....	28
Cookies .....	29
Session .....	34
Η Διαχείριση των Cookies στους Browsers.....	38
Server-Side Scripting και Σχέση Servlets με JSP (JavaServer Pages) .....	39
5. Εισαγωγή στην XML.....	42
Μεταβλητές Ιδιοτήτων – Attributes .....	44
Άλλα Περιεχόμενα Στοιχείου.....	44
Παράδειγμα .....	45

Namespaces.....	45
6. Σημασία και Χρήση της XML .....	47
XML και DOM (Document Object Model).....	47
XML και JDOM (Java Document Object Model) .....	58
XML και SAX (Simple API for XML) .....	64
7. XML και XSL .....	68
XSLT και XPath.....	72
Διαδικασία Εφαρμογής Templates.....	72
Παραδείγματα και Ειδικές Περιπτώσεις.....	75
XSLT Functions .....	78
XPath Functions .....	79
8. XML-RPC .....	82
Αναδιαταξιμότητα του Server.....	86
Παρακολούθηση Μηνυμάτων με TCP Monitor.....	86
9. SOAP .....	87
Παράδειγμα Υπηρεσίας SOAP-RPC.....	90
SOAP Server .....	90
SOAP Client.....	93
Ανακεφαλαίωση .....	95
Μεταφορά JavaBeans μέσω SOAP .....	96
SOAP Server (με JavaBeans).....	97
SOAP Client (με JavaBeans) .....	99
Serialization και Deserialization .....	102
Συνεισφορά του SOAP στην Αναφορά Σφαλμάτων .....	103
SOAP Messaging .....	103
10. Ανακεφαλαίωση Πρωτοκόλλων Εφαρμογών .....	111
Τα GET και POST του http.....	111
Το SOAP πάνω από http.....	111
11. Αντιπαράβολή Εφαρμογών και Υπηρεσιών .....	112
Περιβάλλοντα Εφαρμογών Ιστού.....	113
Συντελεστές Υπηρεσιών Ιστού .....	116
Εφαρμογές και Υπηρεσίες εκτός Ιστού .....	117
Οριζόντιες Υπηρεσίες.....	118
Web Interface και Internet of Things .....	119
12. Το Περιβάλλον .Net.....	120
Το SDK του Περιβάλλοντος .Net (MS Visual Studio) .....	120
Το Αρχείο myForm.aspx για Σελίδα Web .....	121
Το Αρχείο myForm.aspx.vb για Code Behind.....	125
Άλλα Controls και Πρόσθετες Δυνατότητες.....	126
Δημιουργία μίας Υπηρεσίας XML Web.....	127
Παραρτήματα, Βιβλιογραφία και Χρήσιμα Links .....	132
Παράτημα Α: Εγκατάσταση Java και Σχετικών Αρχείων (Jars).....	133
Παράτημα Β: Χρησιμοποιούμενες Command Line Εντολές .....	136
Βιβλιογραφία και Χρήσιμα Links .....	137

## 1. Προγραμματισμός για TCP - Sockets

Το στρώμα μεταφοράς αντιπροσωπεύεται στο Internet από τα πρωτόκολλα TCP (connection oriented) και UDP (connectionless, δηλ. με datagrams). Βάση λοιπόν κάθε δικτυακής εφαρμογής είναι το πώς χειριζόμαστε τα δύο αυτά πρωτόκολλα προγραμματιστικά μέσα στα τερματικά συστήματα. Η απάντηση εγκλείεται στη έννοια των sockets. Ένα socket (υποδοχή, θύρα, 'πρίζα') είναι μία οντότητα (αντικείμενο στην περίπτωση της Java) μέσω της οποίας το δικό μας πρόγραμμα ανταλλάσσει δεδομένα με τον διάυλο επικοινωνίας. Προγραμματιστικά δεν έχουμε παρά να διαβάζουμε από εκεί την ροή εισόδου στο σύστημά μας και αντίστροφα να γράφουμε εκεί την ροή εξόδου από το σύστημά μας. Ιδιαίτερα στην Java, υπάρχει πρόνοια οι δύο αυτές διαδικασίες να μην διαφέρουν από το γράψιμο / διάβασμα σε / από ένα αρχείο. Στην περίπτωση πολλών συνδέσεων TCP (πάντα αμφίδρομων), πρέπει σε κάθε μία να αντιστοιχείται στο δικό της socket. Για κάθε νέα επικοινωνιακή (TCP) σύνδεση δημιουργείται ένα socket. Η εξαφάνιση της σύνδεσης συνεπάγεται την εξαφάνιση του socket και το αντίστροφο. Όλα τα sockets τα οποία χειρίζεται μία συγκεκριμένη διαδικασία (η δική μας, ή μια από τις γνωστές εφαρμογές στο διαδίκτυο) έχουν βεβαίως το δικό τους όνομα για να ξεχωρίζονται προγραμματιστικά, αλλά είναι και το καθένα συνδεδεμένο στην 'άκρη' μίας σύνδεσης προς την εφαρμογή που εξυπηρετείται. Η άκρη αυτή αναγνωρίζεται με έναν απλό αριθμό, το port no ή τον αρ. θύρας. Αρχίζοντας από το δικό μας socket (συγκεκριμένο αντικείμενο), μπορούμε νοητικά (πρακτικά γίνεται αυτόματα) να βρούμε τον αριθμό της αντιστοιχούσης σύνδεσης TCP (τον οποίο φέρουν όλα τα πακέτα του πρωτοκόλλου αυτού), να καταλήξουμε στο άκρο του συνομιλητού μας και να βρούμε μέσα σε αυτόν το δικό του socket (συγκεκριμένο αντικείμενο), καθώς και το port no που αυτό εξυπηρετεί. Έτσι κατορθώνεται να συνεννοούνται το δικό μας με το δικό του πρόγραμμα. Αντίθετα στην περίπτωση του UDP δεν υπάρχει αυτή η αλληλουχία σχέσεων. Η πληροφορία ρίχνεται μέσα στο δίκτυο και καταφθάνει σε ανεξάρτητα πακέτα, όπως με το ταχυδρομείο. Στην περίπτωση αυτή αρκεί μία και μοναδική σταθερή θύρα, απ' όπου μαζεύουμε τα εισερχόμενα και όπου ρίχνουμε τα εξερχόμενα δεδομένα ανεξαρτήτως προέλευσης και προορισμού.

### Στοιχειώδης Server

Στην πλευρά του server πρέπει να αρχίσουμε από μία κατάσταση αναμονής. Δεν συμβαίνει τίποτε, μέχρι να δημιουργηθεί μία σύνδεση με πρωτοβουλία όμως του client. Στον server το αντικείμενο `s` της `ServerSocket` class (ένα 'μετα-socket' που θα φτιάξει το socket) φροντίζει να ακούει την θύρα με το αριθμό 8189. Αυτό επιτυγχάνεται με την μέθοδο `accept` του αντικειμένου αυτού, η οποία είναι blocking (κάθεται και περιμένει). Μόλις καταφθάσει TCP πακέτο αίτησης σύνδεσης στην θύρα αυτή, η `accept` δημιουργεί το αντικείμενο της class `Socket`, το αναφερόμενο εδώ ως 'incoming'. Οποιοσδήποτε θέλει να επικοινωνήσει με το πρόγραμμά μας (αντικείμενο της class `EchoServer` που δίδεται παρακάτω), πρέπει στην πλευρά του να δηλώσει το port 8189. Ο αριθμός αυτός θα μας μεταφερθεί μέσω του TCP πακέτου αίτησης σύνδεσης. Με την άφιξη του πακέτου αυτού, θα προκληθεί η δημιουργία του αντικειμένου 'incoming' socket από το αντικείμενο `s` της `ServerSocket` class. Συγκεκριμένα στο παράδειγμά μας, θα παίζουμε τον πελάτη (client), γράφοντας σε ένα νέο command window την εντολή

```
telnet localhost 8189
```

Η εντολή αυτή θα προκαλέσει την σύνδεση του command window με το port 8189 του συστήματος με την διδόμενη διεύθυνση (εδώ ο localhost, δηλ. loopback μέσα στο ίδιο μηχάνημα). Οτιδήποτε ακολουθεί εμφανίζεται στο 'incoming' socket της πλευράς του server.

Για να το διαβάσουμε (σαν server), ζητάμε από το αντικείμενο `incoming`, μέσω της μεθόδου `getInputStream()`, την ροή εισόδου για να την συνδέσουμε με ένα νέο αντικείμενο της class `InputStreamReader` και με αυτό ξεφεύγουμε από τα δικτυακά! Πράγματι η `InputStreamReader` ανήκει στο πακέτο `java.io`, το οποίο με `import` τίθεται στην διάθεσή μας. Συγκεκριμένα το νέο αντικείμενο που κατασκευάζεται με `new InputStreamReader()` μετατρέπει την ροή των bytes (όπως θα εξάγονται τα δεδομένα από τα εισερχόμενα πακέτα TCP) σε ροή χαρακτήρων. Δίδουμε αυτόν τον μετατροπέα στο νέο αντικείμενο `in` της class `BufferedReader`. Αυτό μπορεί να χρησιμοποιηθεί σαν ταμιευτής (buffer), απ' όπου διαβάζουμε απλούς χαρακτήρες ή και ολόκληρες γραμμές. Με την τεχνική αυτή της αλυσιδωτής σύνδεση ροών διαφορετικού τύπου (chaining), εξομοιώνουμε την λήψη των δεδομένων με το διάβασμα ενός οποιουδήποτε text αρχείου, που ευρίσκεται πάνω σε δίσκο. Για την έξοδο (του server με προορισμό πίσω στον client) γράφουμε στο αντικείμενο `out` της class `PrintWriter`, σαν να γράφαμε σε τερματικό. Στο αντικείμενο αυτό έχουμε σύνδεσει την ροή εξόδου, την οποία και αυτή λαμβάνουμε από το `socket` μέσω της `incoming.getOutputStream()`. Η παράμετρος `true` προκαλεί την δημιουργία και αποστολή πακέτου κάθε φορά που συμπληρώνεται μία γραμμή.

Εφόσον βρίσκουμε γραμμές κειμένου να έρχονται, τις στέλνουμε στην έξοδο. Όταν όμως διαπιστώσουμε το κείμενο "BYE" απολύουμε την σύνδεση. Τούτο συντελείται φυσικά πάλι μέσω του αντικειμένου `incoming` της `Socket` class. Καλούμε απλά την μέθοδό του `close()`. Μεταγλωττίζουμε το παρακάτω πρόγραμμα με `javac EchoServer.java`, το τρέχουμε με `java EchoServer`, και από άλλο παράθυρο το δοκιμάζουμε μέσω `telnet`.

```
import java.io.*;
import java.net.*;
/** Simple server that listens to port 8189; it echoes back all client input.
    TEST this with commandline: 'telnet localhost 8189' */
public class EchoServer
{ public static void main(String[] args )
  { try
    { // establish server socket
      ServerSocket s = new ServerSocket(8189);
      // wait for client connection – as soon as this comes,
      Socket incoming = s.accept ( ); //socket obj. incoming is constructed
      BufferedReader in = new BufferedReader (new InputStreamReader(incoming.getInputStream()));
      PrintWriter out = new PrintWriter (incoming.getOutputStream(),true/* autoFlush */);
      out.println( "Hello! Enter BYE to exit." );

      // echo client input
      boolean done = false;
      while (!done)
      { String line = in.readLine(); // wait here until a line arrives !
        if (line == null) done = true;
        else
        { out.println("Echo: " + line);
          if (line.trim().equals("BYE"))
            done = true; }
        }
      incoming.close();
    }
    catch (Exception e) { e.printStackTrace(); }
  }
}
```

**Κώδικας 1.1.** EchoServer - Δημιουργία και χρήση socket στον server

Ο παραπάνω απλούστερος δυνατός server (επιστρέφει απλά την ‘ηχώ’) δεν μπορεί να ανταποκριθεί σε παραπάνω από μία συνδέσεις. Τούτο το διαπιστώνουμε αν ανοίξουμε ένα δεύτερο παράθυρο και γράψουμε την ίδια εντολή telnet, ενόσω το αντικείμενό μας της class EchoServer, έχει ήδη δημιουργήσει το socket για την κλήση στην θύρα 8189 που ήρθε στο από το πρώτο παράθυρο. Σε αυτό δεν φταίει η μοναδικότης της θύρας 8189, στην οποία μπορούν πράγματι να απευθύνονται πολλές συνδέσεις. Ένας web server ξέρομε ότι είναι συνδεδεμένος (συνήθως) στην μοναδική θύρα 8080, και όμως ανταποκρίνεται συγχρόνως σε πάρα πολλούς clients. Το πρόβλημα είναι ότι στο παράδειγμά μας έχουμε ένα μοναδικό socket, το ‘incoming’. Πρέπει λοιπόν να μπορούμε να δημιουργούμε πολλά sockets, που στο επόμενο παράδειγμά μας θα είναι όλα συνδεδεμένα στην θύρα 8189. Ακόμα καλύτερα καταφεύγουμε σε κάποια τεχνική πολυπρογραμματισμού και αναθέτουμε κάθε νέα σύνδεση, σε μία νέα διαδικασία, με δικό της ξεχωριστό socket. Στον κόσμο της Java αυτό σημαίνει διαφορετικά threads, όπως βλέπομε στον παρακάτω ThreadedEchoServer.

```
import java.io.*;
import java.net.*;
/**Multithreaded server that listens to 8189; it echoes back input from each client.
TEST this with commandline: 'telnet localhost 8189' from several parallel command windows. */
public class ThreadedEchoServer // main class listening and building new threads and sockets
{ public static void main(String[] args )
  { try
    { int i = 1; ServerSocket s = new ServerSocket(8189);
      for (;;)
        { Socket incoming = s.accept( );
          System.out.println("Spawning " + i);
          Thread t = new ThreadedEchoHandler(incoming, i);
          t.start();
          i++; }
        } catch (Exception e) { e.printStackTrace(); }
    }
  }

  /** This class handles the client input for one server socket connection. */
  class ThreadedEchoHandler extends Thread
  { // Constructs a handler: i the incoming socket, c the counter for the handlers (used in prompt)
    public ThreadedEchoHandler(Socket i, int c) { incoming = i; counter = c; }
    public void run()
    { try
      { BufferedReader in = new BufferedReader (new InputStreamReader(incoming.getInputStream()));
        PrintWriter out = new PrintWriter (incoming.getOutputStream(), true /* autoFlush*/);
        out.println( "Hello! Enter BYE to exit." );
        boolean done = false;
        while (!done)
          { String str = in.readLine();
            if (str == null) done = true;
            else
              { out.println("Echo (" + counter + "): " + str);
                if (str.trim().equals("BYE")) done = true; }
          }
        incoming.close();
      } catch (Exception e){ e.printStackTrace(); }
    }
    private Socket incoming; private int counter;
  }
}
```

### Κώδικας 1.2. ThreadedEchoServer - Αντιμετώπιση πολλαπλών συνδέσεων

Έχουμε όπως πριν έναν `ServerSocket s`, ο οποίος φτιάχνει και αναθέτει σε ξεχωριστά `threads` από ένα `socket 'incoming'`, πάντα ακούοντας την θύρα 8189. Τώρα μπορούμε να ανταποκριθούμε παράλληλα και ανεξάρτητα σε πολλά παράθυρα (`clients`), απ'όπου τρέχουμε το `telnet` ακριβώς όπως πριν.

## Στοιχειώδης Client

Εδώ διαφοροποιούμεθα στο ότι αντί να περιμένουμε να ανταποκριθούμε σε μια αίτηση σύνδεσης (βλ. παραπάνω `s.accept()`) οπότε και κατασκευάζουμε ένα `socket`, δημιουργούμε τώρα την σύνδεση με δική μας πρωτοβουλία. Ο παρακάτω κώδικας είναι ουσιαστικά ένα δικό μας `telnet`, το οποίο θα μας φανεί χρήσιμο και στην συνέχεια στο να αρχικοποιούμε κατά τις επιθυμίες μας διαφόρους παραμέτρους της σύνδεσης. Θα μπορούμε να απευθυνόμαστε στους παραπάνω `servers`, αλλά και στους `servers` άλλων εφαρμογών που θα ακολουθήσουν. Θα επεμβαίνουμε μέσω ενός στοιχειώδους GUI για να καθορίζουμε παραμέτρους αλλά και το κείμενο / πληροφορία που θα θέλουμε να στείλουμε, όπως πριν με το `telnet`. Για καλύτερη κατανόηση τρέχουμε πρώτα το παρακάτω πρόγραμμα του `MyTelnet`, που από μόνο του δίνει την αίσθηση του τι συμβαίνει.

Ονομάζουμε τώρα το `socket` που θα φτιάξουμε `outgoing` σε αντίθεση με το `incoming` που είχαμε πριν (βλ. μέθοδο `connectToServer` που καλείται με το πάτημα του κουμπιού `CONNECT`). Το αντικείμενο (`class Socket`) `incoming` είχε δημιουργηθεί στο προηγούμενο παράδειγμα (που ήταν `server`) από ένα αντικείμενο της `class SocketServer`, την στιγμή που έφθασε η κλήση. Εδώ δημιουργούμε απ' ευθείας το αντικείμενο (`class Socket`) `outgoing` με έναν `constructor`,

```
Socket outgoing = new Socket(host, port);
```

Προσδιορίζουμε με `String` τον `host` και με `Integer` το `port` αυτού, δηλ. την πλήρη πληροφορία για το που θέλουμε να συνδεθούμε. Η σύνδεση συντελείται αμέσως - αν όλα πάνε καλά. Αν όχι θα περιμένουμε επ' άπειρον το πακέτο επιβεβαίωσης κλήσης. Πράγματι το πρόγραμμά μας θα κρεμάσει στην εντολή `in.readLine()`, όπου προσπαθούμε να διαβάσουμε το κείμενο που έχουμε προβλέψει να έρθει μέσα στο πρώτο πακέτο που θα επιστρέψει ο `server`, δηλ. το "Hello! Enter BYE to exit." του προηγούμενου `EchoServer`. Την περίπτωση αυτής της αναμονής θα την διορθώσουμε πιο κάτω.

Η αποστολή δεδομένων (μέθοδος `sendData` καλούμενη με πάτημα του κουμπιού `SEND`), καθώς και η λήψη (κώδικας μέσα στο κουμπί `SEE RESPONSE`) είναι όπως τις είδαμε στον `EchoServer`. Σε ανεξέλεγκτη αναμονή μπορούμε επίσης να περιπέσουμε και στην λήψη. Αρκεί να πατήσουμε `SEE RESPONSE`, όταν δεν έχουμε στείλει τίποτα με το `SEND` και άρα δεν έχουμε λάβει τίποτα από τον `EchoServer`.

Με αποστολή του κειμένου `BYE`, προκαλούμε τον `EchoServer` να κλείσει από την πλευρά του το `socket`, άρα και την σύνδεση. Όπως είδαμε και στην άλλες περιπτώσεις (άφιξη επιβεβαίωσης εγκατάστασης κλήσης, άφιξη δεδομένων) το προγραμματιστικό μοντέλο που ακολουθείται δεν βασίζεται σε κάποιου είδους `interrupts`. Πρέπει λοιπόν εμείς, στην πλευρά του `client`, να διαπιστώσουμε της κατάσταση της σύνδεσης με ένα `isClosed()` που επιστρέφει `boolean`. Δεν έχουμε λάβει πρόνοια για τούτο, ούτε για κατάργησης της κλήσης απ'ευθείας από την πλευρά του `MyTelnet`. Τούτο θα ήταν απλό με ένα `outgoing.close()`. Πιο

ενδιαφέρουσα είναι η περίπτωση του Half-Close που θα δούμε παρακάτω σε συνδυασμό με τις εφαρμογές πάνω από TCP.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.net.*;
import java.io.*;

public class MyTelnet extends JFrame {
    private JTextArea msgBody, rcvText;
    private JTextField toUrl,toPort;
    private JButton con,send,receive;
    private PrintWriter out;
    private BufferedReader in;

    public MyTelnet()
    {
        super( "MyTelnet Screen" );

        Box bHor = Box.createHorizontalBox();
        Box bVerIn = Box.createVerticalBox();
        Box bVer = Box.createVerticalBox();

        toUrl = new JTextField( "localhost", 5);
        bHor.add(toUrl );

        toPort = new JTextField( "8189", 5);
        bHor.add(toPort );

        con = new JButton( "CONNECT      ");
        con.addActionListener(
            new ActionListener() {
                public void actionPerformed( ActionEvent e )
                {connectToServer(toUrl.getText(), Integer.parseInt(toPort.getText()));}
            }
        );
        bVerIn.add(con);

        send = new JButton( "SEND      ");
        send.addActionListener(
            new ActionListener() {
                public void actionPerformed( ActionEvent e )
                {try {sendData(msgBody.getText()); }
                 catch(IOException exception) {}
                }
            }
        );
        bVerIn.add(send );

        receive = new JButton( "SEE RESPONSE");
        receive.addActionListener(
            new ActionListener() {
                public void actionPerformed( ActionEvent e )
                {try { //hangs' waiting for data reception
                     String rcvData = in.readLine();           //%%
                     rcvText.setText(rcvData);                 //%%
                     //%% Replace above 2 lines with next 3:   //%%
                }
            }
        );
    }
}
```

```

        //String rcvData;                                //%%
        //while ((rcvData = in.readLine()) != null) //%%
        //{rcvText.append(rcvData + "\r\n");} //%%
        //one more %% line below %%
    }
    catch(IOException exception) {}
}
);
bVerIn.add(receive);

bHor.add(bVerIn);

bVer.add(bHor);

msgBody = new JTextArea( "Body", 10, 15 );
bVer.add( new JScrollPane( msgBody ) );

rcvText = new JTextArea( 10, 15 );
rcvText.setEditable(false);
bVer.add( new JScrollPane( rcvText ) );

Container c = getContentPane();
c.add( bVer );
setSize( 425, 200 );
show();
}

private void connectToServer(String host, int port)
{ try {
    Socket outgoing = new Socket(host, port); //&&
    //&& Remove last line and insert next 2, for 10sec connection timeout
    // Socket outgoing = new Socket(); //&&
    // outgoing.connect(new InetSocketAddress(host, port), 10000); //&&
    //outgoing.setSoTimeout(4000); //%% 4sec timeout for data arrivals
    out = new PrintWriter(outgoing.getOutputStream());
    in = new BufferedReader(new InputStreamReader(outgoing.getInputStream()));
    String connResp = in.readLine(); //hangs' waiting for connection confirm
    rcvText.setText(connResp);
}
catch(IOException exception) {}
}

private void sendData (String textToSend) throws IOException
{ out.print(textToSend);
  out.print("\r\n"); // CR+LF to complete line
  out.flush(); }

public static void main( String args[] )
{ MyTelnet app = new MyTelnet();
  app.addWindowListener(
    new WindowAdapter() {public void windowClosing( WindowEvent e ) {System.exit( 0 );} } );
}
}

```

### Κώδικας 1.7. MyTelnet - Δημιουργία και χρήση socket στον client



Θα αντιμετωπίσουμε τώρα με timeouts τις δύο περιπτώσεις ανεξέλεγκτης αναμονής που εντοπίσαμε:

**(α) Περίπτωση αναμονής για επιβεβαίωση εγκατάστασης κλήσης.**

Κατασκευάζουμε πρώτα ένα ασύνδετο socket και μετά ζητάμε με την μέθοδο connect την σύνδεση στην συγκεκριμένη διεύθυνση μαζί όμως και με τον προσδιορισμό ενός timeout.

```
// first construction of an 'unconnected' socket, ...
```

```
Socket outgoing = new Socket();//...then connection with timeout in msec  
outgoing.connect(new InetSocketAddress(host, port), timeout);
```

Δίδουμε έτσι χρονικό περιθώριο για να έρθει η επιβεβαίωση της σύνδεσης. Αν αυτό δεν γίνει εγκαίρως δεν πέφτομε σε απροσδιόριστη αναμονή αλλά δημιουργείται ένα αντικείμενο της class SocketTimeoutException προς πληροφόρησή μας και το πρόγραμμα συνεχίζει. Αν ενεργοποιήσουμε τον κώδικα σύμφωνα με τις οδηγίες των σχολίων '&&', τότε προσπαθώντας να συνδεθούμε σε ανύπαρκτο server, δεν πέφτομε σε άπειρη αναμονή.

**(β) Περίπτωση αναμονής για άφιξη δεδομένων.**

Είναι η περίπτωση που πάμε να διαβάσουμε δεδομένα αφιχθέντα σε ένα αντικείμενο της class BufferedReader, με read (διάβασμα ενός χαρακτήρα) ή readLine (διάβασμα μίας γραμμής). Αν η μέθοδος

```
outgoing.setSoTimeout(4000); // waiting for 4sec maximum
```

έχει κληθεί πριν από όλα αυτά τα διαβάσματα μας εξασφαλίζει μία ειδοποίηση της εκποής του timeout χρόνου αναμονής (με SocketTimeoutException) και συνέχιση του προγράμματος. Εδώ τούτο γίνεται κατά την αρχική εγκατάστασή της σύνδεσης. Τώρα μπορούμε επίσης να διαβάζουμε διαδοχικές γραμμές στην λήψη, μέχρι να βρούμε την κενή. Στο κώδικα του MyTelnet ενεργοποιούμε τις γραμμές όπου υπάρχουν σχόλια '%%', ακολουθώντας τις σχετικές οδηγίες.

Συμπερασματικά παρατηρούμε τα εξής στα παραπάνω χαρακτηριστικά παραδείγματα προγραμματισμού sockets σε server και client:

- Με την Java, ο προγραμματισμός των sockets είναι θέμα 1-3 γραμμών κώδικα! Αποστολή και λήψη μηνυμάτων εξομοιώνεται με τις γενικές αρχές του I/O στην γλώσσα αυτή.
- Το αντικείμενο SocketServer αφορά τον server, δηλ. αυτόν που δέχεται κλήσεις και οφείλει να ανταποκριθεί σε αυτές. Αντίθετα ο client επενεργεί αποκλειστικά με αντικείμενο Socket.
- Υπάρχει πάντοτε θέμα 'πολυπρογραμματισμού' για τον server, το οποίο στην περίπτωση της Java επιλύεται με threads.
- Υπάρχει θέμα εισαγωγής timeouts σε όλες τις περιπτώσεις που αναμένουμε κάτι να αφιχθεί από την άλλη μεριά.

Δεν μας απασχολεί το port no του client. Ο client δεν αναμένει δημιουργία νέας σύνδεσης από την πλευρά του δικτύου ώστε να χρειάζεται να γνωρίζει που να την συνδέσει. Στην μεριά του client ασχολούμαστε μόνο με το αντικείμενο socket που εμείς δημιουργήσαμε και μέσω του οποίου χρησιμοποιούμε και διαχειριζόμαστε την σύνδεση.

## 2. Πρόσβαση σε Διαδικτυακές Εφαρμογές

Με στοιχειώδεις γνώσεις για το πρωτόκολλο με το οποίο ανταποκρίνεται κάθε γνωστή διαδικτυακή εφαρμογή μπορούμε να έχουμε άμεση πρόσβαση σε αυτή. Τα πρωτόκολλα αυτά είναι μάλλον απλά και σχεδόν πάντα ανταλλάσσουν ευκόλως κατανοητά μηνύματα κειμένου (text based). Θα δούμε το HTTP (HyperText Transfer Protocol) και το SMTP (Simple Mail Transport Protocol). Με το παραπάνω δικό μας πρόγραμμα MyTelnet διαμορφώνουμε τα text based μηνύματα σαν client, όπως κατά περίπτωση απαιτείται, και τα αποστέλλουμε σαν TCP data μέσω του κουμπιού SEND. Τα μηνύματα αυτά στέλνονται πάνω από μία σύνδεση TCP πού έχει ήδη εγκατασταθεί με το προηγούμενο πάτημα του κουμπιού CONNECT.

Ένα απλό πείραμα θα μας πείσει. Με το MyTelnet συνδεόμαστε στο port 80 οποιουδήποτε host (www.xyz.com, δηλ. σε οποιονδήποτε server), ή στο 8080 αν αυτός είναι ένας δεύτερος HTTP server μέσα στον δικόν μας υπολογιστή. Μετά στέλνομε σαν δεδομένα ακριβώς την γραμμή GET / HTTP/1.0 και δύο enter, χωρίς κανένα backspace – άρα πριν συνδεθούμε καθαρίζουμε το κείμενο στο παράθυρο αποστολής του μηνύματος. Λαμβάνουμε πίσω το .html κείμενο της εισαγωγικής σελίδας του site.

Στην γενικότερη περίπτωση (π.χ. στο μήνυμα πού θα κατασκευάζε ένας browser), αποστέλλεται ένα

```
GET /path_ιστοσελίδας_στον_server HTTP/1.0
```

αν ήδη υπάρχει σύνδεση στον server www.xyz.com, ή ακόμη ισοδύναμα, αν δεν υπάρχει

```
GET www.xyz.com/path_ιστοσελίδας_στον_server HTTP/1.0.
```

Πρέπει να ακολουθούν δύο enter, τα οποία αντιστοιχούν σε δύο κενές γραμμές: η πρώτη είναι το σημάδι μεταξύ επικεφαλίδας και σώματος του μηνύματος HTTP και η δεύτερη υποδηλώνει κενό σώμα μηνύματος. Όπως θα δούμε, στην περίπτωση του GET, οτιδήποτε είναι προς αποστολή προς τον server εντάσσεται στην επικεφαλίδα.

### HTTP (HyperText Transfer Protocol)

Ένας HTTP server είναι απλά ένα ‘μηχάνημα’ πού ανταποκρίνεται σε HTTP μηνύματα πού έρχονται από έναν ή πολλούς HTTP clients. Ο client, όπως ο δικός μας MyTelnet, ανοίγει μία σύνδεση σε αυτόν στο port 8080. Ο HTTP server οφείλει συνεχώς να ‘ακούει’ στο port 8080 και να δέχεται την κλήση, όπως ο δικός μας EchoServer. Στην πράξη είναι απαράδεκτο η αποδοχή της πρώτης κλήσης να αποκλείει τις εισερχόμενες αιτήσεις εγκατάστασης κλήσης επιπλέον clients, οπότε κάθε σοβαρός HTTP server πρέπει να θεωρείται σαν επέκταση μάλλον του ThreadedEchoServer. Ο client (συνήθως συγχρόνως με την αίτηση κλήσης) ζητά κάτι (request), π.χ. με το μήνυμα GET, όπως παραπάνω. Ο HTTP server απαντά θετικά ή αρνητικά (response) και απολύει από την μεριά του την TCP σύνδεση. Σε αυτό το επίπεδο, όλα τελειώνουν εδώ – δεν μένει τίποτα για το μέλλον, για τούτο και το πρωτόκολλο HTTP (σχεδόν αποκλειστικά η σήμερα χρησιμοποιούμενη version 1.1) αναφέρεται σαν stateless. Τούτο το κάνει και πολύ απλό, αν επιπλέον προσθέσουμε ότι όλα τα ανταλλάσσόμενα μηνύματα είναι σε μορφή κειμένου. Εφόσον δεν υπάρχουν καταστάσεις δεν υπάρχουν και εκκρεμότητες από το παρελθόν και κάθε μήνυμα απαιτεί μία, το πολύ, απάντηση. Τα κύρια HTTP μηνύματα είναι τα GET και POST (mηνύματα request) καθώς και το μήνυμα απόκρισης (response).

Με το μήνυμα GET ζητείται μία σελίδα. Αν ο host και το port έχουν ήδη καθορισθεί με την σύνδεση, απομένει ο εντοπισμός της σελίδας μέσα στον server καθώς και η ενδεχόμενη συναποστολή ορισμένων παραμέτρων που θα καθοδηγήσουν τον server στην επιλογή ή διαμόρφωσή της κατά τις επιθυμίες του client. Όλα αυτά διαπιστώνονται γράφοντας σε έναν browser την επιλογή

`http://www.cn.ntua.gr/index.php?option=com_content&task=blogcategory&id=71&Itemid=62&lang=el`  
η οποία εμπεριέχει όλα τα δεδομένα για να γίνει η σύνδεση TCP και να ορισθούν οι παράμετροι της αίτησης μέσω HTTP. Συγκεκριμένα

- εμείς (ο client) επιλέγουμε σαν host το `cn.ntua.gr` και έμμεσα το port 80 με το πρόθεμα `www` που υποδηλώνει το πρωτόκολλο
- εντοπίζουμε την σελίδα `index.php` (η `επέκταση.php` δηλώνει σελίδα τεχνολογίας PHP – αλλά αυτό είναι θέμα του host)
- συναποστέλλουμε με το GET τις παραμέτρους με τα ονόματα `option`, `task`, `id`, `Itemid`, `lang` και αντίστοιχα (μετά από '=' ) τις τιμές `com_content`, `blogcategory`, `71`, `62`, `el`

Εναλλακτικά, μπορούμε να συνδεθούμε με το MyTelnet (ή ακόμη και με telnet) στο `cn.ntua.gr`, port 80 και κατόπιν να στείλουμε (σε μία γραμμή, απλό κενό μετά το GET και πριν το HTTP/1.0, πάλι χωρίς κανένα άλλο χαρακτήρα, `backspace`, κλπ):

```
GET /index.php?option=com_content&task=blogcategory&id=71&Itemid=62&lang=el HTTP/1.0
ακολουθούμενο από δύο enter
```

Και τα δύο περιέχουν ακριβώς την ίδια πληροφορία και προκαλούν την αποστολή πίσω των ίδιων δεδομένων. Οι παράμετροι συντάσσονται, μετά ένα '?', σαν ζεύγη `name/value` διαχωριζόμενα μεταξύ τους με '&'. Αποστέλλονται μόνο αλφαριθμητικά σύμβολα. Το κενό συμβολίζεται με '+' και κάθε άλλος χαρακτήρας με τον κώδικα ASCII μετά από %, π.χ. η παράμετρος `FullName` με την τιμή της 'M. Smith' στέλνεται σαν (2e είναι το ASCII της τελείας σε hex)

`FullName=M%2e+Smith`

Είδαμε λοιπόν ότι στο σημείο αυτό η δουλειά του browser δεν διαφέρει από αυτήν που επιτελεί ο κώδικας του MyTelnet. Όλα καταλήγουν σε μία σύνδεση στο επίπεδο TCP, στην αποστολή του μηνύματος GET του HTTP και στην λήψη της απάντησης HTTP (HTTP response). Το μήνυμα GET αποτελείται μόνο από επικεφαλίδα, της οποίας το τέλος ορίζεται με μια υποχρεωτική κενή γραμμή. Σαν απόκριση λαμβάνεται στην περίπτωση επιτυχίας (όλα πήγαν καλά και ανευρέθη η ζητούμενη ιστοσελίδα)

HTTP/1.1 200 OK

Date: Tue, 28 Jun 2005 11:51:49 GMT

Server: Apache/1.7.33 (Unix) PHP/4.7.10 mod\_ssl/2.8.22 OpenSSL/0.9.7c

X-Powered-By: PHP/4.7.10

Set-Cookie: lang=deleted; expires=Mon, 28-Jun-2004 11:51:48 GMT; path=/  
Set-Cookie: mbfcookie=deleted; expires=Mon, 28-Jun-2004 11:51:48 GMT; path=/  
Set-Cookie: mbfcookie[lang]=el; expires=Wed, 29-Jun-2005 11:51:49 GMT; path=/  
Set-Cookie: sessioncookie=2803d11cfc3939a0ce58710fdcb36c07; expires=Tue, 28-Jun-2005 23:51:49 GMT; path=/  
Expires: Mon, 26 Jul 1997 05:00:00 GMT  
Last-Modified: Tue, 28 Jun 2005 11:51:50 GMT  
Cache-Control: no-store, no-cache, must-revalidate  
Cache-Control: post-check=0, pre-check=0  
Pragma: no-cache  
Connection: close

Content-Type: text/html

```
- κενή γραμμή και ακολουθεί η προς παρουσίαση σελίδα-
<?xml version="1.0" encoding="iso-8859-7"?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
TML 1.0 Transitional//EN" "http://www.w7.org/TR/xhtml1/DTD/xhtml1-transitional.d
td">
<html xmlns="http://www.w7.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-7" />
<title>Iae?iaoa - Computer Networks Laboratory</title>
<meta name="description" c
ontent=",,,,,,,,,CN - Computer Networks Laboratory" />
<meta name="keywords" cont
ent=",,,,,,,,,computer networks, lab, laboratory, ntua, iccs" />
..... κλπ, κλπ
```

### Σχήμα 2.1. Μήνυμα απόκρισης σε GET (επιτυχία – κωδ. 200 OK στην επικεφαλίδα)

Ο browser αντιλαμβάνεται την γλώσσα του σώματος του μηνύματος απόκρισης (συνήθως html ή xml όπως εδώ) και την παρουσιάζει ωραία, πράγμα το οποίο βεβαίως αδυνατεί να κάνει το MyTelnet, ή το command window. Στην περίπτωση αποτυχίας (στο GET, αντί /index.php γράφομε την ανύπαρκτη σελίδα /index1.php), η απόκριση είναι πάλι μία σελίδα html, με τυποποιημένη όμως την παρουσίαση της αποτυχίας ανεύρεσης με τον κωδικό της Error 404 (<TITLE>404 Not Found</TITLE>).

HTTP/1.1 404 Not Found

Date: Tue, 28 Jun 2005 12:04:28 GMT

Server: Apache/1.7.33 (Unix) PHP/4.7.10 mod\_ssl/2.8.22 OpenSSL/0.9.7c

Connection: close

Content-Type: text/html; charset=iso-8859-1

```
- κενή γραμμή και ακολουθεί καθαρή html-
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>404 Not Found</TITLE>
</HEAD><BODY>
<H1>Not Found</H1>
The requested URL /index1.php was not found on this server.<P>
<HR>
<ADDRESS>Apache/1.7.33 Server at
<A HREF="mailto:webmaster@cn.ntua.gr">www.cn.ntua.gr</A> Port 80</ADDRESS>
</BODY></HTML>
```

### Σχήμα 2.2. Μήνυμα αποτυχίας σε GET ('404 Not Found' στην επικεφαλίδα)

Τέλος ας δούμε και το GET πού αποστέλλεται από έναν πραγματικό browser (IE5). Πάλι έχουμε μόνο επικεφαλίδα ακολουθούμενη από μία κενή γραμμή, ή επικεφαλίδα όμως τώρα περιέχει και άλλες πληροφορίες πού έχουν να κάνουν με της δυνατότητες παρουσίασης / επεξεργασίας πού έχει ο συγκεκριμένος browser, κλπ.

GET /index.html HTTP/1.1

// καλούμε κάποιον server τοπικά (localhost)

Accept: image/gif, image/x-bitmap, image/jpeg, ..., application/msword, \*/\*

Accept-Language: el,en-gb;q=0.5

Accept-Encoding: gzip, deflate

```
If-Modified-Since: Mon, 13 Dec 2004 01:06:43 GMT // στείλε μου μόνον αν έχεις αλλαγή μετά από...
User-Agent: Mozilla/4.0 (compatible: MSIE 6.0; Windows NT 5.1; SV1)
Host: localhost: 8080
Connection: Keep-Alive // μετά την αποστολή απόκρισης, ΜΗΝ κλείσεις την σύνδεση TCP
- κενή γραμμή -
```

### Σχήμα 2.7. Μήνυμα GET αποστέλλόμενο από πραγματικό browser

Το παραπάνω μήνυμα το είδαμε μέσω του προγράμματος TCP Monitor, που εξηγείται στην επόμενη παράγραφο.

**Με το μήνυμα POST** ο client δεν στέλνει μόνο επικεφαλίδα, αλλά και σώμα (body) μηνύματος. Αυτό μπορεί να είναι οσονδήποτε μεγάλο. Έτσι μεταξύ των άλλων, το μήνυμα POST επιλύει περιορισμούς μεγίστου μήκους πού ενίοτε τίθενται για το μέγεθος μίας επικεφαλίδας του HTTP και που ανακύπτουν στην περίπτωση του GET.

Το σώμα του POST εκτείνεται σύμφωνα με το μήκος του, πού δηλώνεται με το υποχρεωτικό Content-Length: ... πεδίο της επικεφαλίδας. Η πρώτη γραμμή της επικεφαλίδας περιέχει υποχρεωτικά το πεδίο Content-Type: type, όπου type συνήθως text/plain ή text/html ή application/octet-stream ή application/x-www-form-urlencoded. Οποσδήποτε το μήνυμα POST, είναι το ενδεικνυόμενο για την αποστολή δεδομένων από τον client στον server. Έτσι ‘ανεβάζομε’ και ιστοσελίδες (γραμμένες σε html, δηλ. Content-Type: text/html) για ‘δημοσίευση’ σε έναν web server.

```
POST www.server.com/index.html HTTP/1.0
Referer: http://www.somewhere.com/links.html
Proxy-Connection: Keep-Alive
User-Agent: Mozilla/4.76 (Windows ME; U) Opera 5.11 [en]
Host: www.server.com
Accept: text/html, image/gif, image/jpeg, image/png, */*
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Content-Length: the counted number of characters in the body
Cookie: orangemilano=192218887821987
- κενή γραμμή -
```

*Μήνυμα (π.χ. περιεχόμενο ιστοσελίδας προς ανάρτηση)*

### Σχήμα 2.4. Μήνυμα POST

Πριν στείλει ο client οποιοδήποτε μήνυμα POST, πρέπει πρώτα το σώμα (body) του να έχει πλήρως αποθηκευθεί σε κάποιο buffer, ώστε να υπολογισθεί το μήκος του.

Είδαμε το πρωτόκολλο HTTP και τις διαδικασίες λήψης και εκπομπής των μηνυμάτων του GET και POST. Υπάρχουν επιπλέον μηνύματα μικρότερης σημασίας, όπως το HEAD (ο client ζητά μόνο τις επικεφαλίδες σελίδων στον server) και το DELETE (ο client ζητά την διαγραφή σελίδων στον server). Το τι κάνουν, ή μπορούν να κάνουν οι διάφορες αρχιτεκτονικές στον server με βάση το πρωτόκολλο HTTP, αποτελεί αντικείμενο ξεχωριστό πού σχετίζεται με τις διάφορες επικρατούσες τεχνολογίες (servlets, Active Server Pages – ASP, Java Server Pages - JSP, Common Gateway Interface - CGI, Hypertext Preprocessor -

PHP, κλπ). Μερικές θα γνωρίσουμε πιο κάτω. Επίσης μένει να εξηγηθεί πώς το HTTP μεταφέρει πληροφορία, η οποία κατ' αρχήν δεν αποτελείται από χαρακτήρες, όπως π.χ. μία εικόνα.

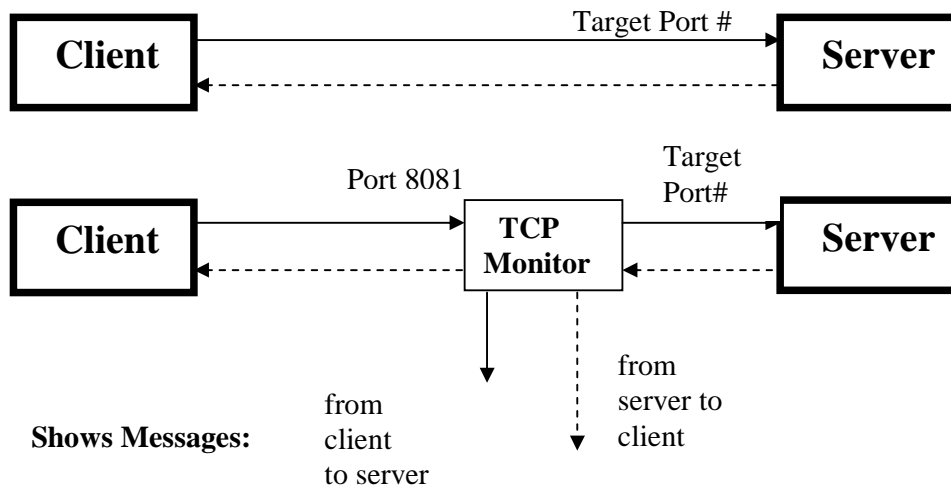
## **Παρακολούθηση Μηνυμάτων με TCP Monitor**

Στην προηγούμενη και στις επόμενες ενότητες μπορούμε να παρακολουθήσουμε τα ίδια τα μηνύματα που μεταφέρονται μεταξύ client και server. Στην πλευρά του client εγκαθιστούμε έναν 'TCP Monitor' σύμφωνα με το παρακάτω Σχήμα 2.5.

Η μόνη μετατροπή που χρειάζεται όταν παρεμβάλλεται ο TCP Monitor είναι ο client να μιλά σε αυτόν (στο 'Listen Port#' 8081) και αυτός να μιλά σε οποιονδήποτε server ('Target HostName', στα παραδείγματά μας θέτομε 'localhost') και θύρα αυτού ('Target Port#'). Στην αντίθετη διεύθυνση συμβαίνουν τα ανάλογα: ο server απαντά και η απάντηση, πριν επιδοθεί στον client, περνά μέσα από τον TCP Monitor. Ο server είναι τελείως ουδέτερος και ανυποψίαστος για την παρεμβολή του TCP Monitor. Στον client, η μόνη απαιτούμενη αλλαγή είναι να μην μιλά στον server αλλά στον TCP Monitor. Ο TCP Monitor ενεργοποιείται (από το directory του client) με την εντολή

```
java org.apache.axis.utils.tcpmon
```

Περνάμε τα παραπάνω αναφερθέντα στοιχεία στο πρώτο παράθυρο που ανοίγει (Admin), πατάμε Add και εμφανίζεται νέο tab με το επιλεγέν Port #. Αφήνοντας την διαδικασία του TCP Monitor να τρέχει, έχουμε τώρα την 'φωτογραφία' όλων των μηνυμάτων που διακινούνται.



Σχήμα 2.5. Παρεμβολή και διευθύνσεις TCP Monitor

Μπορούμε τώρα να επαναλάβουμε το παραπάνω παράδειγμα και να δούμε τα πραγματικά HTTP πακέτα που ανταλλάσσονται. Θα έχουμε τρία ανοικτά παράθυρα (command windows), για τον client, για τον TCP Monitor και για τον server.

## SMTP (Simple Mail Transport Protocol - RFC 821)

Ένας SMTP server είναι απλά ένα ‘μηχάνημα’ που ανταποκρίνεται σε SMTP μηνύματα που έρχονται από έναν ή πολλούς SMTP clients. Ο client, όπως ο δικός μας MyTelnet, ανοίγει μία σύνδεση σε αυτόν στο port 25. Ο SMTP server οφείλει συνεχώς να ‘ακούει’ στο port 25 και να δέχεται την κλήση, όπως ο δικός μας EchoServer. Συνδεόμαστε λοιπόν εδώ μέσω του MyTelnet στον SMTP server (π.χ. theseas.ntua.gr), γράφουμε το παρακάτω κείμενο και το αποστέλλουμε πατώντας SEND:

```
HELO <αποστέλλον host> (οτιδήποτε και αν γράψουμε, επιστρέφει την IP διεύθυνσή μας)
MAIL FROM: <email address αποστολέως>
RCPT TO: <email address παραλήπτη>
DATA
<το κείμενο του email – με enter (“\r\n”) διαχωρίζονται οι γραμμές>
.- υποχρεωτική τελεία>
QUIT
```

Διαβάζοντας γραμμή προς γραμμή το μήνυμα αυτό, ένας SMTP server (κατάλληλα ανεπτυγμένος με βάση τον δικό μας ThreadedEchoservert) αντιλαμβάνεται τα επιμέρους πεδία και πράττει αναλόγως. Ιδιαίτερως σημειώνουμε το τρόπο που βρίσκει το τέλος. Δεν βασίζεται στο μήκος των δεδομένων όπως είδαμε στο HTTP, αλλά περιμένει γραμμή με μία μεμονωμένη τελεία. Με το ‘QUIT’ διακόπτεται η σύνδεση με τον SMTP server. Το πρωτόκολλο SMTP βασίζεται λοιπόν στον διαχωρισμό γραμμών (line oriented). Θεωρητικά δεν ενδιαφέρει τον SMTP server από ποιόν client κατέφθασε το μήνυμα. Το ίδιο το μήνυμα SMTP έχει όλη την απαιτούμενη πληροφορία (μοντέλο ταχυδρομικής υπηρεσίας). Θα προωθήσει λοιπόν αυτό το μήνυμα email παραπέρα, με τελική κατάληξη στον παραλήπτη. Ενδιάμεσοι κόμβοι SMTP θέτουν στην επικεφαλίδα επιπλέον πληροφορία για την ακολουθείσασα διαδρομή. Εξηγήσαμε εδώ το πρωτόκολλο της αρχικής παράδοσης του

‘γράμματος’ στο ‘ταχυδρομείο’ και όχι το λίγο πιο πολύπλοκο πρωτόκολλο προώθησης των ‘γραμμάτων’ ανάμεσα στα ‘ταχυδρομικά γραφεία’. Όπως βλέπομε μπορούμε να στείλομε *email* σε οποιονδήποτε από οποιονδήποτε! Στην πράξη όμως σήμερα πλέον κατά την διαδρομή εφαρμόζονται οι όλο και εκτεινόμενες πολιτικές ασφάλειας (antiSPAM κλπ).

## Διάφορα

### Half-close για Δήλωση Τερματισμού Αίτησης

Είδαμε την αποκλειστική χρήση του request / response μοντέλου. Ο client αιτείται (request) και ο server αποκρίνεται (response). Τίθεται όμως το θέμα του πώς αντιλαμβάνεται ο server πού τελειώνει η αίτηση. Παρ’ όλο που επιδιώξαμε να ανάγουμε τα πάντα σε γενικό I/O, εδώ υπάρχει πρόβλημα: όταν γράφομε σε ένα αρχείο, γνωρίζομε που είναι το τέλος και έχομε εμείς την εξουσία να κλείσομε το αρχείο μετά την εγγραφή όλων των δεδομένων, εφόσον δεν περιμένουμε καμία απόκριση. Στην δικτυακή περίπτωση πρέπει ο server να γνωρίζει ότι ο client τελείωσε με την αίτηση, χωρίς όμως να έχει κλεισθεί η σύνδεση TCP, καθόσον αυτή χρειάζεται για την απόκριση. Στα HTTP και SMTP είδαμε τις σχετικές τεχνικές λύσεις (μήκος, το σημάδι τέλους αντίστοιχα). Στην γενική περίπτωση η λύση έγκειται στο λεγόμενο half-close: ο client ‘κλείνει’ την κατεύθυνση ‘client προς server’ και μένει ακόμη ανοικτή η ‘server προς client’ για να μπορέσει να στείλει ο server την απόκριση. Στον client έχομε

```
...
writer.print( . . );           //τελικό γράψιμο στον buffer εξόδου
writer.flush();               //τελική αποστολή - αναγκαστικό άδειασμα του buffer
socket.shutdownOutput();     //half-close
```

ενώ ο server χρησιμοποιεί κατόπιν την μέθοδο close(), όπως έχομε ήδη δει.

### MIME για Αποστολές πέραν του Απλού Κειμένου

Με το MIME (Multipurpose Internet Mail Extensions), το οποίο καταφεύγει στην κωδικοποίηση Base64, μπορούμε να μετατρέσομε σε ‘ASCII’ οποιαδήποτε ακολουθία ακεραίων οκτάδων (bytes). Τα bits μίας ακολουθία 3 τέτοιων οκτάδων ( $3*8=24$  bits) χωρίζονται σε 4 εξάδες ( $4*6=24$  bits). Κάθε εξάδα απεικονίζεται σε έναν από τους  $2^6=64$  χαρακτήρες ‘ASCII’ του παρακάτω πίνακα. Όταν η ακολουθία μας δεν έχει ακέραιη την τελευταία 24αδα των bits, τότε καταλήγει είτε με μία 8αδα είτε με μία 16αδα - είμαστε ικανοί για byte, όχι για bit alignment της πληροφορίας!. Τότε η κωδικοποίηση Base64 προσθέτει στο τέλος ‘=’ ή ‘==’ αντίστοιχα.

```
private static char[] toBase64 = // 64 chars: 'A' - 'Z', 'a' - 'z', '0' - '9', then '+', '/'
{
    'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
    'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
    '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '+', '/'
};
```

Σχήμα 2.6. Πίνακας για κωδικοποίηση Base64



Αφού γίνει η παραπάνω απεικόνιση στέλνονται πλέον (οι παραπάνω) χαρακτήρες 'ASCII', έκαστος με 7 bit. Στον δέκτη συμβαίνουν τα αντίστροφα. Το MIME είναι, πέραν από τις δικτυακές εφαρμογές, ευρέως διαδεδομένο, γενικής χρήσεως πρωτόκολλο. Καλύψαμε εδώ την δυνατότητα αποστολής οποιασδήποτε άλλης πρωτογενούς πληροφορίας πού ξεφεύγει από την ακολουθία χαρακτήρων, π.χ. μίας εικόνας.

### 3. Προγραμματισμός σε Επίπεδο URL

Είδαμε παραπάνω τα κοινά χαρακτηριστικά του προγραμματισμού εφαρμογών όπως το HTTP και SMTP. Αυτά επεκτείνονται και σε άλλες εφαρμογές που δεν γνωρίσαμε, π.χ. FTP (File Transport Protocol). Κεντρικά σημεία είναι το πρωτόκολλο, ο server προς τον οποίον απευθύνεται ο client, το port (θύρα) και ενδεχομένως ο συγκεκριμένος στόχος (τα αποκαλούμενα resources) μέσα στην εφαρμογή (π.χ. η σελίδα πού θέλουμε να κατεβάσουμε από τον server) – τούτο δεν αφορά το SMTP. Επιπλέον όλα συντελούνται με την ίδια χρήση των sockets, μέσω TCP. Όλα αυτά συντελούν στο να ομαδοποιηθούν τα παραπάνω σε κάτι ενιαίο γύρω από την έννοια του URL (Uniform Resource Locator). Οι σχετικές classes που προσφέρει η Java δεν περιορίζονται στην χρήση ενός εντοπιστή ενός στόχου μέσα στο διαδίκτυο (locator). Περιλαμβάνουν και την απόκτηση της πληροφορίας από / χρήση υπηρεσία του server με όλα όσα κάναμε παραπάνω με την διαχείριση και χρήση sockets. Άρα ουσιαστικά θα δούμε περίπου τα ίδια, αλλά σε ανώτερο και απλούστερο προγραμματιστικά επίπεδο.

Σαν τυπική λοιπόν περίπτωση, κατασκευάζουμε ένα αντικείμενο της class URL και μετά ανοίγουμε (μέθοδος openStream της class URL) μία ανωτέρου επιπέδου σύνδεση. Τούτο δεν μας επιστρέφει πλέον ένα socket, αλλά ένα αντικείμενο της class InputStream, απ' όπου απλά διαβάζουμε τα δεδομένα:

```
InputStream uin = url.openStream();
BufferedReader in = new BufferedReader(new InputStreamReader(uin));
String line;
while ((line = in.readLine()) != null)
{process line;}
```

Είναι φανερό ότι οι παραπάνω ευκολίες περιορίζονται κυρίως στην άντληση πληροφοριών από το διαδίκτυο και δεν αφορούν οποιαδήποτε διαδικτυακή εφαρμογή.

#### Διαφοροποίηση μεταξύ URIs, URLs και URNs

Το URI (Uniform Resource Identifier) σαν γενικότερη έννοια (όχι όμως σαν κλάση – πατέρας στην κληρονομικότητα της Java), περιλαμβάνει τις περιπτώσεις του URL (Uniform Resource Locator) πού είδαμε παραπάνω, αλλά και το URN (Uniform Resource Name). Επίσης το URI καθορίζει μόνον στο πώς αναφερόμαστε σε έναν πόρο (στόχο, resource - URI reference) και δεν έχει λειτουργικές δυνατότητες (δηλ. μεθόδους) για την πρόσβαση σε αυτόν, όπως είδαμε παραπάνω. Διατυπωμένο αλλιώς: η πολυπλοκότητα της σύνταξης των αναφορών σε πόρους δημιούργησε την ανάγκη η class URI να ασχολείται με την ονοματολογία την ίδια, ώστε να

διευκολύνεται η class URL στην καθ' αυτό πρόσβαση. Το URI είναι ένας γενικός όρος για ονόματα και διευθύνσεις διατυπωμένα σε μορφή text (και όχι π.χ. σε μορφή binary). Το URL είναι ένα σύνολο εκφράσεων URI που περιέχουν τον άμεσο τρόπο πρόσβασης μέσω Internet. Ξεκινώντας από μερικά παραδείγματα URL, όπως

```
http://maps.yahoo.com/py/maps.py?csz=Cupertino+CA
ftp://username:password@ftp.yourserver.com/pub/file.txt
mailto:cay@anybody.com
αλλά και file:///d:/somefolder/... (αρχείο στο localhost)
```

καταλαβαίνουμε την ανάγκη η class URI να ορίζει τον συντακτικό κανόνα

```
[scheme:]schemeSpecificPart[#fragment]
```

με προαιρετικά πεδία ([]) και το '#' να κατέχει την θέση προκαθορισμένων συμβόλων (π.χ. '?', '+', κλπ). Επιπλέον σε ένα προγραμματιστικό περιβάλλον μπορούμε για διευκόλυνση να ορίζουμε το URI σχετικά: όταν το πεδίο 'scheme:' υπάρχει, αποκαλούμε το URI απόλυτον (absolute), αλλιώς είναι σχετικό (relative). Όταν, όπως στην τρίτη πάνω περίπτωση, το schemeSpecificPart δεν περιλαμβάνει '/', τότε καλείται ημιδιαφανές (opaque). Ένα απόλυτο και μη-ημιδιαφανές (absolute non-opaque) URI, ή ένα σχετικό URI είναι πάντα ιεραρχικό, π.χ.

```
http://java.sun.com/index.html // absolute non-opaque
../java/net/Socket.html#Socket() // relative
```

πράγμα αναμενόμενο, εφόσον κινούμεθα προς συγκεκριμένη διαδικτυακή διεύθυνση, και κατόπιν στην ιεραρχία αρχειοθέτησης μέσα σε αυτήν. Τέλος το fragment έχει σημασία μόνο μέσα στα πλαίσια του συγκεκριμένου πόρου και δεν αφορά την διαδικασία εντοπισμού του πόρου αυτού.

Σε αναλογία με τα παραπάνω είναι και οι μέθοδοι που διατίθενται στην class URI. Από την μία μας επιτρέπεται η σύνθεση τέτοιων δομών από τα επί μέρους, ή δίδοντας έτοιμο το σχετικό string

```
URI(String scheme, String host, String path, String fragment)
URI(String str)
```

και από την άλλη μία πλούσια επιλογή getXYZ, μας επιτρέπει να πάρουμε τα επί μέρους (getPath, getPort, getFragment, getScheme) από το πλήρες URI. Τέλος, για να δούμε και την συσχέτιση, η myURI.toURL() επιστρέφει ένα αντικείμενο της class URL κατασκευασμένο από το αντικείμενο myURI της class URI.

## Άντληση Πληροφορίας από Web Server με URLConnection

Επανερχόμενοι αποκλειστικά στο HTTP, εξετάζουμε πάλι την περίπτωση άντλησης πληροφορίας από έναν web sever. Αυτή ως γνωστόν γίνεται μέσω του μηνύματος GET, το οποίο όμως εδώ θα κατασκευάζουμε προγραμματιστικά και όχι συντάσσοντας 'με το χέρι' ολόκληρο το μήνυμα, όπως πριν. Με ανάλογο προγραμματιστικά τρόπο, θα αναλύουμε και το

μήνυμα απόκρισης (HTTP response) που θα μας έρθει από τον (web) server. Προς τούτο διατίθεται η class `URLConnection` με περισσότερες δυνατότητες από την class `URL`. Καμία εκ των δύο δεν είναι απόγονος της άλλης, και οι δύο απορρέουν απ' ευθείας από την class `Object`. Εφόσον όλα τα απαιτούμενα για την σύνδεση στοιχεία είδαμε ότι περιλαμβάνονται στο `url`, τι περιμένουμε εδώ επί πλέον από τα παραδείγματα που δείξαμε παραπάνω; Η απάντηση είναι ότι τώρα διαμορφώνουμε προγραμματιστικά την επικεφαλίδα του μηνύματος `GET` καθώς και τις συνθήκες υπό τις οποίες αυτή θα **αποσταλεί** από τον client. Η διαμόρφωση αυτή κατευθύνει και τον web sever σε ιδιαίτερη για το εκάστοτε `GET` συμπεριφορά πριν αποκριθεί. Το ίδιο θα συμβεί και με την επικεφαλίδα της απόκρισης, από την οποία θα αντλούμε προγραμματιστικά, ότι στοιχείο ενδεχομένως μας ενδιαφέρει. Τέλος το σώμα της απόκρισης θα μας διατίθεται μέσω `InputStream`, δηλ. θα εξισούται με το διάβασμα ενός αρχείου. Τα βήματα μίας τέτοιας διαδικασίας στην πλευρά μας (client) είναι:

(α) Άνοιγμα (προετοιμασία) της σύνδεσης με

```
URLConnection mycon = url.openConnection();
```

απ' όπου αποκτώμε το αντικείμενο `mycon` της Class `URLConnection`. Από εδώ και πέρα θα απευθυνόμαστε σε αυτό για τα περαιτέρω.

(β) Καθορισμός των συνθηκών της σύνδεσης (request properties) με μεθόδους σαν τις

```
setDoInput(true); // αυτό true και το επόμενο false προκειμένου για GET
setDoOutput(false); // σημαίνει αποστολή GET και όχι POST !!
setIfModifiedSince(); // ενδιαφέρει πληροφορία μόνο να αλλάξε μετά από ..
setUseCaches(); // εξέτασε πρώτα πιθανή πληροφορία σε cache του browser
setAllowUserInteraction(); // ενδιαφέρει μόνον για applets (δικαιώματά τους)
setRequestProperty(); // εμφάνισε πρώτα διάλογο για username, password - ότι δοθεί τοποθετείται
// π.χ. στο ftp://username:password@ftp.yourserver.com/pub/file.txt
```

(κάθε μία από τις παραπάνω καλείται σαν `mycon.Xyz()` – οι παράμετροι δεν φαίνονται παραπάνω). Όλες αυτές οι μέθοδοι θα διαμορφώσουν κατάλληλα την επικεφαλίδα του `GET` που (ενδεχομένως!) θα αποσταλεί αλλά όχι μόνον αυτό. Προσδιορίζουν επίσης ιδιότητες στα δύο άκρα πριν γίνει η σύνδεση. Για παράδειγμα η σύνδεση (κλάση `URLConnection`, βλ. (γ) αμέσως παρακάτω) κατασκευάζει πάντα ένα `input stream` για να διαβάζουμε σαν client από τον server αλλά όχι και ένα `output stream`. Όταν λοιπόν θελήσουμε στη επόμενη παράγραφο να αναρτήσουμε πληροφορία στον server, πρέπει πρώτα να καλέσουμε την μέθοδο `mycon.setDoOutput(true)`, από τις παραπάνω.

(γ) Καθ' αυτό σύνδεση με

```
mycon.connect();
```

οπότε γίνεται η σύνδεση στο επίπεδο του socket αλλά και με τον συγκεκριμένο server και αποστέλλεται το `GET`. Αν όλα πάνε καλά, λαμβάνεται πίσω στον client το μήνυμα της `HTTP response`, το οποίο και εξετάζεται παρακάτω πάλι μέσω μεθόδων του αντικειμένου `mycon`.

(δ) Εξέταση της πληροφορίας στην επικεφαλίδα που επεστράφη (επικεφαλίδα του μηνύματος απόκρισης - `HTTP response`) με μεθόδους σαν

```
getContentType, getContentLength, getContentEncoding, getDate, getExpiration, getLastModified
```

Και πάλι, κάθε μία τις παραπάνω καλείται σαν `mycon.Xyz()`. Η σημασία τους είναι λίγο-πολύ προφανής, αν τις παραβάλαμε με την πληροφορία που επιστρέφεται στην επικεφαλίδα της `HTTP response`. Γενικότερα οι `getHeaderFieldKey(n)` και `getHeaderField(n)` επιστρέφουν το όνομα και την τιμή του `n`-στου πεδίου της επικεφαλίδας, πάντα σαν κείμενο (`String`). Αν τέτοια δεν υπάρχουν επιστρέφεται 'null'.

(ε) πρόσβαση στην επιστρεφόμενη πληροφορία (body της απόκρισης = 'requested resource data') με `getInputStream`.

```
        BufferedReader in = new BufferedReader(new
                                InputStreamReader(mycon.getInputStream()));

        String line;
        while ((line = in.readLine()) != null)
        {
            process line
        }
```

Στο τελικό βήμα (ε) το `InputStream` είναι ακριβώς αυτό που επιστρέφεται με την μέθοδο `openStream` της `URL` class. Πράγματι απλώς προσθέσαμε εδώ ιδιαίτερα για το HTTP ενδιάμεσα βήματα για καλύτερο έλεγχο του webserver, πάντα μέσα στο πρωτόκολλο HTTP. Η πλέον κοινότυπη διαδικασία για έναν client σαν απόρροια του βήματος (γ) είναι η παρουσίαση μίας επιστραφείσης σελίδας .html (περίπτωση web browsing). Θα δώσουμε μόνο ένα χαρακτηριστικό παράδειγμα, καθόσον οι δυνατότητες και περιπτώσεις είναι πολύ εκτεταμένες.

```
import java.io.*; // This program connects to an URL and displays the response header data
import java.net.*; // and the first 10 lines of the requested data. Supply on the command line the URL and
import java.util.*; // an optional username and password (HTTP basic authentication).
public class GETwithURL
{ public static void main(String[] args)
  { try
    { String urlName;
      urlName = args[0];
      URL url = new URL(urlName);
      URLConnection mycon = url.openConnection();
      mycon.connect();
      // print header fields, i.e. the 'keys' (names) with their string values
      // until a key returns a null value
      int n = 1; String key;
      while ((key = mycon.getHeaderFieldKey(n)) !=null)
        { String value = mycon.getHeaderField(n);
          System.out.println("The Key " + key + " has value " + value + ""); n++; }
      System.out.println("---- Showing the getMethods of the mycon obj ----");
      System.out.println("getContentType: " + mycon.getContentType());
      System.out.println("getContentLength: " + mycon.getContentLength());
      System.out.println("getContentEncoding: " + mycon.getContentEncoding());
      System.out.println("getDate: " + mycon.getDate());
      System.out.println("getExpiration: " + mycon.getExpiration());
      System.out.println("getLastModified: " + mycon.getLastModified());
      System.out.println("-----");
      BufferedReader in = new BufferedReader(new
        InputStreamReader(mycon.getInputStream()));
      String line; n = 1; // print first ten lines of contents
      while ((line = in.readLine()) != null && n <= 10)
        { System.out.println(line); n++; }
      if (line != null) System.out.println(". . .");
    } catch (IOException exception) { exception.printStackTrace(); }
  }
}
```

### Κώδικας 3.1. Διαχείριση του GET μέσω `URLConnection`

Δοκιμάζουμε τον παραπάνω κώδικα δίδοντας το επιθυμητό URL στην command line, π.χ.  
java GETwithURL "file:///d:/.../..." (περίπτωση τοπικού αρχείου)

Σαν πηγή πληροφορίας δοκιμάζουμε οποιαδήποτε από τις περιπτώσεις URL, που γνωρίσαμε πιά πάνω (http:, ftp:, file:). Με το παραπάνω πρόγραμμα τυπώνουμε πρώτα τα πεδία (όνομα και τιμή) της επικεφαλίδας του μηνύματος που επιστρέφεται, μετά εξάγουμε επιλεγμένη πληροφορία από την επικεφαλίδα και τέλος τυπώνουμε τις πρώτες δέκα γραμμές του σώματος. Οτιδήποτε είναι κείμενο (text) εμφανίζεται χωρίς πρόβλημα. Με το παραπάνω σημείο (β) δεν ασχολούμαστε. Δεν χρειάζεται εδώ (στο GET), καθόσον οι τιμές που θέτουν τα setDoInput (true) και setDoOutput(false) είναι ήδη πάντα βαλμένες στον client σαν default. Διαφορετική είναι η κατάσταση με το POST, το οποίο τώρα θα εξετάσουμε.

## Αποστολή Πληροφορίας σε Web Server με URLConnection

Ας δούμε πως διαφοροποιούνται τα παραπάνω, όταν δεν στέλνουμε στον server παραμέτρους μέσω GET προκειμένου να μας απαντήσει κατάλληλα, αλλά δική μας πληροφορία. Όταν η πληροφορία που πρέπει να αποσταλεί στον (web) server είναι εκτεταμένη, έστω και στην μορφή πολλών ζευγών name/value pairs, καταφεύγουμε στο μήνυμα POST. Η πληροφορία αυτή δεν ενσωματώνεται στην επικεφαλίδα σαν προέκταση του URL όπως στο GET, αλλά αποτελεί το σώμα του μηνύματος. Τα βήματα τώρα στην πλευρά μας (client) είναι:

(α) Άνοιγμα (προετοιμασία) της σύνδεσης με

```
URLConnection mycon = url.openConnection();
```

απ' όπου αποκτούμε το αντικείμενο mycon της Class URLConnection. Από εδώ και πέρα θα απευθυνόμαστε σε αυτό για τα περαιτέρω

(β) Εκφράζουμε μετά την επιθυμία μας για μία output connection.

```
mycon.setDoOutput(true);
```

Τούτο πρακτικά σημαίνει ότι θα σταλεί και σώμα από τον client, το οποίο θα πρέπει να διαβάσει ο server. Το σώμα αυτό πρέπει να γραφεί στην πλευρά του client σε ένα output stream. Η παραπάνω μέθοδος κάνει δυνατό το επόμενο βήμα.

(γ) Από το αντικείμενο mycon λαμβάνουμε μία ροή εξόδου για να αποστείλουμε τα δεδομένα μας. Προκειμένου να στείλουμε κείμενο, χειριζόμαστε αυτήν την ροή σαν PrintWriter:

```
PrintWriter out = new PrintWriter(mycon.getOutputStream());
```

(δ) Στέλνουμε τα δεδομένα σαν name1=value1&name2=value2& κωκ

```
out.print(name1 + "=" + URLEncoder.encode(value1) + "&");
```

```
out.print(name2 + "=" + URLEncoder.encode(value2));
```

Τα παραπάνω θα αποτελέσουν το σώμα του μηνύματος, μετά από μία κενή γραμμή κάτω από την επικεφαλίδα του POST. Επίσης, το μήνυμα δεν μπορεί να φύγει πριν συμπληρωθεί αποθηκευμένο στο αντικείμενο out, καθόσον πρέπει πρώτα να υπολογισθεί και τοποθετηθεί στην επικεφαλίδα το μήκος του σώματος.

(ε) Τέλος κλείνουμε το output stream (όχι την σύνδεση!) με

```
out.close();
```

και αναμένουμε την απόκριση, την οποία διαβάζουμε ακριβώς όπως πριν (βήμα ε)). Η απόκριση αυτή ουσιαστικά μας ενημερώνει αν η πληροφορία ελήφθη κανονικά από την

μεριά του server. Στο παραπάνω σημείο (δ) καλύψαμε την περίπτωση όπου το σώμα του μηνύματος είναι μία αλληλουχία ζευγών name/value. Αυτή είναι η τυπική περίπτωση όταν απαντάμε σε μία 'φόρμα' που έχει ληφθεί προηγουμένως, κατά πάσα πιθανότητα με GET. Όλες οι απαντήσεις/επιλογές μας συγκεντρώνονται σαν τέτοια ζεύγη (όνομα παραμέτρου και τιμή που πληκτρολογήσαμε για αυτή) και αποτελούν το σώμα του μηνύματος POST. Άλλη περίπτωση χρήσης του POST είναι όταν στέλνουμε περιεχόμενο (ιστοσελίδα για ανάρτηση) στον web server.

Για να δούμε και τις δύο περιπτώσεις, πρέπει τώρα να στραφούμε στον server.

## 4. Η Πλευρά του Server

Η Java όπως και άλλες γλώσσες προγραμματισμού για το διαδίκτυο υπεισέρχονται στο περιβάλλον web και στον client (web client) και στον server (web server). Ο web client αποκαλείται και thin client. Όλες οι βαριές εργασίες (αναζήτηση πληροφοριών σε βάσεις, εκτέλεση της λογικής μίας εφαρμογής, κλπ.) μετατοπίζονται στον server. Ο server είναι υπερσύνολο του web server. Ο web server είναι αυτός που επικοινωνεί με τους web clients κάτω από το περιβάλλον web που εξετάζουμε εδώ, ενώ ο server, ως υπερσύνολο, περιλαμβάνει με συγκεντρωτικό τρόπο την εκτέλεση των υπολοίπων αναγκαίων διαδικασιών που ενδεικτικά αναφέρθηκαν. Είναι αξιοπερίεργο ότι το διαδίκτυο προάγει τελικά μία συγκεντρωτική αντιμετώπιση. Αντί για κατανομή των διεργασιών, μετατοπίζει την πλειοψηφία τους στον server. Εν τούτοις το σημαντικό πλεονέκτημα που έχει πλέον εδραιωθεί, είναι η απλούστευση και γενίκευση του client. Ουσιαστικά έχουμε να κάνουμε κυρίως με τους δύο επικρατήσαντες browsers (Internet Explorer και Netscape), οι οποίοι μάλιστα έχουν και σχεδόν τα ίδια χαρακτηριστικά. Το πρωτόκολλο HTTP, ο browser και οι mark up γλώσσες HTML, XML αποτελούν τελικά την παγκοσμίως αποδεκτή βάση του δικτυακού προγραμματισμού. Με το HTTP μεταφέρουμε την πληροφορία και μάλιστα υπό την μορφή σελίδων αποτελούμενων από χαρακτήρες (ASCII). Μέσα σε αυτές περικλείονται τα πάντα (χαρακτήρες, κείμενα πολυμέσων, κώδικας προς εκτέλεση). Ο τελικός και μόνος αποδέκτης είναι ο browser ως παγκόσμια μηχανή παρουσίασης στον χρήστη καθώς και συλλογής των αντιδράσεών του. Για όλα αυτά διατίθενται διάφορες τεχνικές λύσεις στην πλευρά του server (στατικές σελίδες .html, servlets, δυναμικές σελίδες JSP, ASP, κλπ), αλλά η συγκεκριμένη συνεισφορά της Java είναι και για τον client (τα applets) και για τον server (τα servlets). Το applet είναι κώδικας που καταλήγει ενσωματωμένος σε HTML σελίδα για εκτέλεση στον client. Το servlet προσφέρει την δυνατότητα αλληλεπίδρασης των λειτουργιών στον server σύμφωνα με όσα ανταλλάσσονται με τον browser.

### To Servlet

Όλα τα servlets (που διατίθενται μέσω είτε της GenericServlet class είτε της HttpServlet class) πρέπει να πραγματοποιούν το Servlet interface (δηλ. πρέπει να πραγματοποιούν όλες της μεθόδους που ορίζονται σε αυτό). Όλα αυτά περιέχονται στα javax.servlet και javax.servlet.http packages της Java. Ιδιαίτερα το HttpServlet ορίζει εκείνες τις μεθόδους και interfaces που επιτρέπουν στον web server να ανταποκρίνεται στην αλληλεπίδραση με τον web client (browser) μέσω του πρωτοκόλλου http, δηλ. ουσιαστικά να μπορεί να αντιμετωπίζει την άφιξη των μηνυμάτων GET και POST. Εξ αυτού ονομάζεται και HTTP

server. Το Servlet interface του GenericServlet έχει τις εξής μεθόδους: Η void init(ServletConfig config) καλείται αυτόματα από τον server, ο οποίος δημιουργεί και το αντικείμενο ServletConfig που περιέχει παραμέτρους αρχικοποίησης. Με την μέθοδο getServletConfig() ανακτάται πρόσβαση στις παραμέτρους αυτές, που εντάσσονται στο επιστρεφόμενο αντικείμενο ServletConfig. Με την getServletInfo() επιστρέφεται ένα String με διάφορες πληροφορίες για το δημιουργηθέν servlet (εκδότης του, version, κλπ) και με την void destroy() καθαρίζεται το servlet από τον server. Το περισσότερο όμως ενδιαφέρον για την κατ' εξοχήν λειτουργία και χρήση του servlet, έχει η

void service (ServletRequest request, ServletResponse response)

Εξειδικεύοντας τα παραπάνω interfaces, η HttpServlet class εγγενώς διακρίνει τις αιτήσεις GET και POST του πρωτοκόλλου HTTP μέσω των δυνατοτήτων πολυμορφισμού της Java. Έτσι μόλις καταφθάνει στον server μία αίτηση GET ή POST του client, δημιουργούνται τα δύο αντικείμενα HttpServletRequest και HttpServletResponse και καλείται η μέθοδος service, η οποία με την σειρά της καλεί, ανάλογα με το αν κατέφθασε GET ή POST, την μέθοδο doGet ή doPost. Αυτές είναι μέθοδοι της HttpServlet class και συνυπάρχουν μαζί με άλλες σχετικές και παρόμοιες, με τις οποίες δεν θα ασχοληθούμε περαιτέρω (doDelete, doOptions, doPut, doTrace). Οι doGet και doPost λαμβάνουν σαν ορίσματα τα ορίσματα HttpServletRequest request και HttpServletResponse response της μητέρας τους, δηλ της μεθόδου service. Ο προγραμματίζων σε περιβάλλον servlets δεν έχει τελικά παρά να γράψει τις δικές του εκδόσεις για την doGet ή την doPost (κλασσική μεθοδολογία 'method overriding' της Java). Σε αυτές το αντικείμενο request μας διαθέτει οτιδήποτε πληροφορία περιείχε το μήνυμα (σε επικεφαλίδα και σώμα), ενώ το αντικείμενο response προσφέρεται σε εμάς για να διαμορφώσουμε την απόκρισή μας σαν server. Από αυτό το αντικείμενο response θα δημιουργηθεί το μήνυμα απόκρισης.

Μέσω των μεθόδων του αντικειμένου HttpServletRequest μπορούμε να έχουμε

- την τιμή μίας παραμέτρου που έχει στείλει ο client με GET ή POST. Δίδοντας το όνομα name (σαν String) της παραμέτρου, μας επιστρέφεται η τιμή της getParameter(name), πάλι σαν String. - το ίδιο γίνεται με array από String, συλλήβδην για πολλές παραμέτρους (μέθοδος String[] getParameterValues( String name)).
- την απαρίθμηση όλων των ονομάτων (χωρίς τιμές) των παραμέτρων που μετέφερε το POST. Τώρα η getParameterNames() επιστρέφει ένα Enumeration.
- όλα τα cookies που αποθηκεύει ο client, σαν array από Cookie objects (μέθοδος Cookie[] getCookies()).
- το HttpSession object, το οποίο κρατιέται στον server. Η μέθοδος HttpSession getSession(boolean create), επιστρέφει μία αναφορά σε ένα αντικείμενο session. Δίδουμε την παράμετρο create σαν false, όταν υπάρχει ήδη session, αλλιώς δίδοντας true, αιτούμεθα την δημιουργία μίας session.

Με τα cookie και session θα ασχοληθούμε ειδικότερα παρακάτω.

Μέσω των μεθόδων του αντικειμένου HttpServletResponse, μπορούμε

- να ανοίξουμε ένα stream εξόδου, οπότε γράφοντας κατόπιν σε αυτό, αναλαμβάνει ο server την αποστολή των γραφέντων στον client. Η ServerOutputStream getOutputStream() φτιάχνει ένα byte-based, ενώ η PrintWriter getWriter() ένα character-based stream εξόδου. Σχετικός είναι και ο ορισμός του τύπου κατά MIME, τον οποίον πρέπει να μάθει ο browser του client για να προσαρμόσει την εμφάνισή.

Έτσι με την μέθοδο `void setContentType (String type)` ορίζουμε τον τύπο, με πιο συχνή περίπτωση το `type = "text/html"`.

- να προσθέσουμε ένα cookie, το οποίο θα καταφθάσει στον client για αποθήκευση πάνω στην επικεφαλίδα του http response. Δίδουμε ένα Cookie object με την μέθοδο `void addCookie(Cookie cookie)`.

Η παρακάτω class επεκτείνει την `HttpServlet` προγραμματίζοντας την `doGet` σύμφωνα με τις επιθυμίες μας. Ενώ η `doGet` της αρχικής `HttpServlet` δεν κάνει τίποτε άλλο από το να ανταποκρίνεται στο GET request του http με την ένδειξη `BAD_REQUEST()` σαν απάντηση, εδώ την βλέπουμε να αποστέλλει χρήσιμη πληροφορία. Τούτο επιτυγχάνεται μέσω του αντικειμένου response που δηλώνεται ως `HttpServletResponse`, δηλ. ως αυτό που απαιτεί σαν παράμετρο η `doGet`.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class HTTPGetServlet extends HttpServlet {
    public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException
    {
        PrintWriter output;

        response.setContentType ("text/html");
        output = response.getWriter();

        StringBuffer buf = new StringBuffer() ;
        buf.append( "<HTML><HEAD><TITLE>\n" ); // write here line-by-line the html for the desired page
        buf.append( "A simple servlet example\n" );
        buf.append( "</TITLE></HEAD><BODY>\n" );
        buf.append( "<H1>Welcome to servlets !</H1>\n" );
        buf.append( "</BODY></HTML>" ); // end of desired page
        output.println (buf.toString());
        output.close (); // PrintWriter stream closed->buffer is flashed to client !!!
    }
}
```

#### **Κώδικας 4.1.** Το `HTTPGetServlet`, σαν στοιχειώδες servlet

Με την μέθοδο `setContentType` ορίζουμε σαν `text/html` τον τύπο της απάντησής μας (ο γνωστός τύπος του MIME). Κατόπιν η αναφορά `output` ορίζεται σαν μία αναφορά σε αντικείμενο `PrintWriter`, ένα αντικείμενο της Java (μέσα στο `java.io`), που συμπεριφέρεται σαν εκτυπωτής, δέχεται δηλαδή σειριακά διαδοχικούς χαρακτήρες. Ένα τέτοιο αντικείμενο όμως μας διαθέτει εκ κατασκευής και το `HttpServletResponse`: με την μέθοδο `getWriter` μας επιστρέφει μια αναφορά (reference) σε αυτό την οποία ταυτίζουμε με την δική μας `output`. Από κει και πέρα, μπορούμε να κάνουμε με το `output` όσα μας επιτρέπει η Java (`append` – προσθήκη κειμένου, `println` – ‘εκτύπωση γραμμής’, `close ()`). Όλα αντιστοιχούνται αυτόματα εδώ όχι βεβαίως σε πραγματική εκτύπωση, αλλά σε αποστολή προς τον web client του συρμού των χαρακτήρων μέσω μίας απόκρισης (response). Στην συγκεκριμένη περίπτωση ο συρμός είναι αυτός ο οποίος ως σελίδα `html` θα εμφανισθεί από τον browser. Τελικά η κλήση



της μεθόδου doGet ενός αντικειμένου της class HTTPGetServlet, θα προκαλέσει την εμφάνιση της συγκεκριμένης σελίδας html στην οθόνη του web client.

Συνοπτικά, τι πετύχαμε με τον παραπάνω κώδικα; Ορίσαμε το τι θα κάνει ο web server όταν λαμβάνει GET. Η δουλειά μέχρις εδώ έγινε χάρις (α) στην αρχική εγκατάσταση του servlet στον web server (δεν εξηγήθηκε) (β) στις προπρογραμματισμένες δυνατότητες όλων των αντικειμένων των class GenericServlet και HttpServlet και (γ) στις πρόσθετες δυνατότητες που προγραμματίσαμε κατά τον δικό μας σχεδιασμό της class HTTPGetServlet.

Η όλη χρήση του servlet είναι να στέλνει γραμμή προς γραμμή, με ενίοτε χιλιάδες διαδοχικές γραμμές println, την σελίδα .html στον browser. Βεβαίως σε επιλεγμένα σημεία μπορεί ο κώδικας να διαφοροποιεί την έξοδο αυτή, έτσι ώστε να έχουμε τις λεγόμενες δυναμικές σελίδες. Πάντως παραμένει σαν γενικό χαρακτηριστικό των servlets το γεγονός ότι πρόκειται για κώδικα στον οποίο σε μεγάλο βαθμό είναι απλά ενσωματωμένη η HTML.

Για να προκαλέσουμε την εμφάνιση της παραπάνω σελίδας, κτυπάμε τοπικά με τον browser `http://localhost:8080/<κάποιο όνομα που οδηγεί στο HTTPGetServlet>` που προσδιορίζει το πρωτόκολλο (http), το μηχάνημα (εδώ localhost) όπου είναι εγκατεστημένος ο server, την πύλη 8080 όπου ο web server αναμένει αιτήσεις (request) των client, και κάπως τον αποδέκτη HTTPGetServlet. Βεβαίως δεν θα γίνει τίποτε αν δεν φροντίσουμε τώρα να βάλουμε τον κώδικα του servlet που δημιουργήσαμε στον server, μαζί με την επιπλέον πληροφορία που σχετίζεται με αυτόν.

## **To Servlet μέσα στον Server**

Η παρακάτω διαδικασία αποσκοπεί στο να δώσει στον server την πληροφορία για το πώς θα κατευθύνει τις αιτήσεις του client στα κατάλληλα αρχεία ή διαδικασίες που είναι εγκατεστημένες μέσα του. Ειδικότερα για τα servlets, υπάρχει μία προκαθορισμένη αρχιτεκτονική που αποκαλείται servlet container. Αυτή συνυπάρχει μέσα στην γενική αρχιτεκτονική του εκάστοτε server και εδώ θα περιγράψουμε την περίπτωση του Tomcat. Ο Tomcat είναι, μεταξύ των άλλων, ένας servlet container, δηλαδή ένα περιβάλλον μέσα από το οποίο εκτίθενται στο διαδίκτυο όσα εκτελούν τα διάφορα από αυτόν φιλοξενούμενα servlets. Τούτο ακριβώς θα δούμε ευθύς αμέσως. Ο Tomcat τρέχει σε ένα περιβάλλον ενός κοινού web ή HTTP server, στην συγκεκριμένη περίπτωση του Apache. Αυτός συνυπάρχει αλλά δεν φαίνεται στα παρακάτω, καθόσον ασχολείται στο να δέχεται και να αποστέλλει HTTP μηνύματα, περίπου όπως γνωρίσαμε πιο πάνω. Ο Tomcat (και ο Apache) ξεκινά / σταματά με το batch file startup / shutdown μέσα στο C:\jakarta-tomcat-7.7.2\bin.

Όπως θα δούμε μία εφαρμογή web πέραν του κώδικα (class file) του servlet, περιλαμβάνει συνήθως και άλλα συμπληρωματικά αρχεία. Όλα αυτά αποτελούν έναν κατάλογο (dir) με προκαθορισμένη δομή, το οποίον στην περίπτωσή μας ονομάζουμε myServletDir. Έχει την παρακάτω δομή και τοποθετείται πάντα (όπως και κάθε άλλη παράλληλα εκτεθειμένη εφαρμογή) κάτω από το webapps του Tomcat. Τα πλαγιαστά είναι δικές μας ονομασίες, ενώ τα όρθια είναι ονόματα που αναγνωρίζονται έτσι από τον Tomcat. Κατ' αρχήν φορτώνουμε οποιοδήποτε στατικό αρχείο (π.χ. index.html, mytest.html ή ακόμη test.txt, κλπ.) κάτω από το folder myServletDir, δηλ. C:\jakarta-tomcat-7.7.2\webapps\myServletDir\test.txt και το βλέπομε από τον browser με `http://localhost:8080/myServletDir/test.txt`. Ειδικά το

index.html εμφανίζεται αν προσδιορίσουμε μόνο `http://localhost:8080/myServletDir` (η default σελίδα της εφαρμογής μας). Αυτή είναι η απλούστερη εργασία κάθε web server.

```
myServletDir
  index.html
  mytest.html
  WEB-INF
    web.xml
    classes
      HTTPGetServlet.class
    lib
      other.jar
  other_directories
```

**Σχήμα 4.1.** Ο Κατάλογος myServletDir της εφαρμογής μέσα στον server  
(dir του Tomcat, κάτω από το οποίο κρεμάμε αυτήν και άλλες εφαρμογές είναι το `C:\jakarta-tomcat-7.7.2\webapps`)

Τώρα τοποθετούμε την μεταγλωττισμένη class του servlet μας σύμφωνα με το παραπάνω μονοπάτι:

```
C:\jakarta-tomcat-7.7.2\webapps\myServletDir\WEB-INF\classes\ HTTPGetServlet.class
```

Χρειάζεται όμως και ένας λεγόμενος Deployment Descriptor (DD), ο οποίος καλείται πάντα `web.xml`, πάντα κάτω από το folder `WEB-INF`. Είναι σε μορφή `.xml` και δίνει όλες τις απαιτούμενες αντιστοιχίσεις μεταξύ του ονόματος της ‘εφαρμογής’, του ονόματος που η εφαρμογή αυτή είναι γνωστή και προσβάσιμη από τους browsers και του ονόματος του αρχείου (εδώ java class), το οποίο την εκτελεί. Αυτά είναι και τα ελάχιστα στοιχεία, τα οποία και θα δούμε εδώ. Υπάρχουν και άμπολλά άλλα, σχετικά με την έκθεση (deployment) της ‘εφαρμογής’ στο διαδίκτυο. Για να αναγνωρίζεται λοιπόν το servlet μας πάμε και διαμορφώνουμε το αρχείο `web.xml` ως εξής (το editing με Notepad και μετά αποθήκευση σαν ‘All Files’ και προέκταση `.xml`)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Copyright 1999-2004 The Apache Software Foundation ... -->

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<!-- web-app element was originally found empty - the following are additions
  for the demonstration of HTTPGetServlet -->
<web-app>
  <servlet> <!-- the following name will refer to the sepecified class -->
    <servlet-name>ServletDemo</servlet-name>
    <servlet-class>HTTPGetServlet</servlet-class>
  </servlet>
  <servlet-mapping><!-- the following (same) name will be called as the specified url patttern -->
    so hitting http://localhost:8080/myServletDir/firstServletDemo/* (any thing) -->
    will lead to HTTPGetServlet -->
    <servlet-name>ServletDemo</servlet-name>
    <url-pattern>/firstServletDemo/*</url-pattern>
  </servlet-mapping>
</web-app>
```

**Σχήμα 4.2.** Το `web.xml` σαν Deployment Descriptor της `HTTPGetServlet.class`

Η ενδιαφέρουσα πληροφορία είναι αυτή των στοιχείων `servlet-name`, `servlet-class` και `url-pattern`, το οποία και καθορίζουν τις παραπάνω αντιστοιχίσεις. Μέσα στο αρχείο `web.xml` ο συνδεδετικός κρίκος είναι το στοιχείο `servlet-name` πού του δώσαμε εμείς την αυθαίρετη τιμή `ServletDemo`. Αυτό μέσα στο στοιχείο `servlet` συνδέεται με τον κώδικα (το `servlet-class`), ενώ το ίδιο, χάριν του εξωτερικού κόσμου, συνδέεται μέσα στο `servlet-mapping` με το `url-pattern`. Με την πληροφορία αυτή μπορεί να εκτεθεί (deployed) η εφαρμογή πού εκτελεί το `servlet` είναι. Σηκώνουμε τον Tomcat, κτυπάμε από τον browser τον localhost, θύρα 8080, το γενικό αρχείο το οποίο περικλείει όλη την εφαρμογή μας (`myServletDir`), την ονομασία της εφαρμογής μας στο διαδίκτυο (`firstServletDemo`) και τίποτε (ή οτιδήποτε) μέσα σε αυτή, και μας εμφανίζεται η σελίδα πού δημιούργησε ο κώδικας του `servlet`, όπως τον γράψαμε πιο πάνω. Προφανώς ο Tomcat χρησιμοποίησε τις αντιστοιχίσεις που επισημάναμε για να γίνει η δουλειά. Ο εξωτερικός κόσμος (το διαδίκτυο) βλέπει όλη την δομή μέχρι το `WEB-INF`, αυτού μη συμπεριλαμβανομένου. Ειδικότερα ο κώδικας (`HTTPGetServlet.class`) δεν φαίνεται, παρά μόνον καλείται με το `alias` πού καθορίσαμε στο `web.xml` (`firstServletDemo`).

Τα όσα κάναμε παραπάνω μπορούν να γίνουν σύμφωνα με μία προτυποποιημένη γενίκευση. Η γενική ιδέα είναι ότι το folder `webapps` με το περιεχόμενό του είναι προσβάσιμο από το διαδίκτυο και περιέχει άλλους καταλόγους (folders) με εφαρμογές web. Μέσα σε καθεμία από αυτές έχουμε στατικές σελίδες, αλλά για το `WEB-INF`, που περιέχει όλα όσα είδαμε πιο πάνω, και επίσης τους υποκαταλόγους `lib` και `other_directories` σε πιο εξελεγχόμενες περιπτώσεις. Κάθε παιδί του `webapps` είναι λοιπόν αυτόνομο και η κανονική διαδικασία επιτρέπει την εισαγωγή του σαν αρχείο `.war` (Web Application Archive - WAR). Αυτό είναι κάτι ανάλογο με τα `.jar` της Java, δηλ. όλη η δομή του παραπάνω σχήματος 2.5 συμπίεσμένη σε ένα μοναδικό αρχείο. Τέτοιο `myServletDir.war` μπορούμε να κατασκευάσουμε με την εντολή

```
jar -cvfM MyServletDir.war WEB-INF index.html test.text
```

εκδιδόμενη μέσα από το `MyServletDir`. Το κατασκευασθέν `myServletDir.war` τοποθετούμε μέσα στο `webapps`. Ο Tomcat, καθώς σηκώνεται, επεκτείνει αυτόματα όσα τέτοια αρχεία `.war` βρει κάτω από το `webapps` και δημιουργεί το ίδιο `dir` που εμείς φτιάξαμε 'χειροκίνητα'. Μπορούμε να αντιπαραβάλουμε στο σημείο αυτό τις άλλες προεγκατεστημένες εφαρμογές του Tomcat, δηλ. τα `dir` αρχεία `ROOT`, `admin`, `examples`, `soap`, με τα αντίστοιχα `.war` αρχεία, από τα οποία προήλθαν. Για να δούμε την εσωτερική δομή των τελευταίων απλά τα κτυπάμε δύο φορές. Μπορούμε να θεωρήσουμε ένα τέτοιο αρχείο `.war` σαν αυτόνομη οντότητα που πραγματοποιεί μία εφαρμογή web και όταν τοποθετηθεί μέσα στον κατάλογο `webapps` κάθε `servlet container` και όχι μόνο του Tomcat.

Τα `servlets` φιλοξενούνται και εκτίθενται κατά στην αρχιτεκτονική του `servlet container` (εδώ Tomcat) μέσα σε έναν `server` (εδώ κρυμμένος Apache). Ο `container`, όπως είδαμε, είναι ένας τρόπος φορμαλισμού της σχέσης του ανεξαρτήτου κώδικα πού αντιμετωπίζει της αιτήσεις του `client` μέσα στο περιβάλλον του γενικότερου web server. Ο φορμαλισμός αυτός είναι γενικά αποδεκτός και έτσι το ίδιο αρχείο `WAR`, μπορούμε να το πάρουμε και να το τρέξουμε κάπου εκτός Tomcat. Τα `servlets` στην ουσία δεν είναι παρά `classes` της Java. Έχουν το προτέρημα να είναι σε ετοιμότητα στην μνήμη, ενόσω 'τρέχει' ο Tomcat. Δεν φορτώνονται δηλαδή την στιγμή πού καταφθάνει ένα μήνυμα από τον `client`. Αυτό δίνει καλή χρονική απόκριση. Επειδή μένουν συνεχώς ενεργά μπορούν να κρατηθεί μέσα τους την πληροφορία πού δημιουργήθηκε κατά την απόκριση προς ένα μήνυμα και μετά την συμπλήρωση του κύκλου `request` (με `GET` ή `POST`) – `response`. Αυτός είναι ένας τρόπος να αποκτήσουμε `state` (δηλ. δυνατότητα 'μνήμης') πάνω από το `stateless HTTP`. Άλλον τρόπο θα δούμε παρακάτω με τα `cookies`. Το βασικό μειονέκτημα των `servlets` είναι εύκολο να το δούμε ακόμη και στο απλοϊκό παράδειγμά μας. Αν μας ζητηθεί να αλλάξουμε το ελάχιστο μέσα στην ιστοσελίδα

που παράγει και αποστέλλει, χρειάζεται ξανά μεταγλώττιση της HTTPGetServlet.class, επαναδημιουργία του αρχείου .war (το οποίο δυνατόν να περιλαμβάνει και πάμπολλα άλλες classes), επανατοποθέτησή του στον server. Αυτό και άλλα σχετικά κάνουν δύσκολη την ανάπτυξη και έλεγχο πολύπλοκων εφαρμογών. Ο κύριος λόγος είναι ότι όλη η λογική της εφαρμογής και οι δομές των δεδομένων είναι περιπλεγμένα με όλες τις λεπτομέρειες της παρουσίασης, μέσα στον ίδιο κώδικα. Τα HTML statements ευρίσκονται σαν ορίσματα χιλιάδων εντολών εκτύπωσης, όπως οι μέθοδοι append της StringBuffer class, ή και η println της PrintWriter (βλ. Κώδικα 4.1).

Τέλος πολλά servlets, μέσα στον κοινό servlet container και το καθένα εξειδικευμένο για να απαντά τις διαφορετικές δυνατές αιτήσεις, αποτελούν μία εφαρμογή web (web application). Βασικό χαρακτηριστικό είναι, ότι χωρίς εμείς να κάνουμε τίποτα, το περιβάλλον αυτό αντιμετωπίζει πολλαπλές κλήσεις από πολλούς clients προς την ίδια υπηρεσία συγχρόνως. Στην περίπτωση μας εδώ, τούτο επιτυγχάνεται με το multi threading της Java.

## Χρήση Cookies και Sessions

Ας δούμε κατ' αρχήν τους τρόπους αποστολής αιτήσεων με τα εκάστοτε δεδομένα τους από τον client στον server. Αυτό γίνεται με την ενσωμάτωση στοιχείων GUI (τα λεγόμενα controls) σε οποιαδήποτε σελίδα .html. Δημιουργούμε την παρακάτω σελίδα με Notepad και την σώνουμε σαν myClientPage.html (επιλογή 'All Files', επέκταση .html). Μετά την κτυπάμε δύο φορές και βλέπουμε τι εμφανίζει.

```
<!-- A form for selection of user choices (inputted data to be sent by GET)
      two text boxes, three radio buttons; , submit and reset buttons. -->
<HTML>
<HEAD> <TITLE>A form to be filled in and submitted </TITLE> </HEAD>
<BODY>
  <FORM ACTION="http://localhost:8080/somewhere" METHOD="GET">
    <INPUT TYPE="text" NAME="firstName" > Put your name<BR>
    <INPUT TYPE="text" NAME="lastName" > ... and surname<BR>
    <STRONG>Select the Continent of your Destination:<br></STRONG>
    <PRE>
    <INPUT TYPE="radio" NAME="continent" VALUE="Europe">check here for Europe<BR>
    <INPUT TYPE="radio" NAME="continent" VALUE="Asia">check here for Asia<BR>
    <INPUT TYPE="radio" NAME="continent" VALUE="America" CHECKED>check here for
    America<BR>
    </PRE>
    <INPUT TYPE="submit" VALUE="Submit">
    <INPUT TYPE="reset"> </P>
  </FORM>
</BODY>
</HTML>
```

### Σχήμα 4.3. (Στατική) Σελίδα / Φόρμα στον browser

Αυτήν την σελίδα την αντιλαμβάνεται οποιοσδήποτε κοινός browser, ο οποίος εμφανίζει και τα σχετικά στοιχεία GUI. Επιπλέον ο browser, συμμορφούμενος με την παράμετρο METHOD="GET" ή "POST", αποστέλλει με πάτημα του 'Submit' τα δεδομένα της επιλογής μας αναλόγως, δηλ. με μήνυμα GET ή POST. Ο προορισμός είναι η τιμή σαν url της παραμέτρου ACTION. Πρόκειται για μία 'φόρμα', η οποία ουσιαστικά συλλέγει και

αποστέλλει τις επιλογές του καθήμενου μπροστά στον browser. Δεν έχουμε ακόμη φροντίσει για την υποδοχή της στη μεριά του server, αλλά εν τούτοις βλέπουμε να κωδικοποιούνται κατά το γνωστά στο GET οι τιμές που έδωσε ο χρήστης. Η σελίδα η ίδια είναι στατική ακόμη και αν έχει κατεβεί αρχικά από κάποιον server. Δεν έχει δημιουργηθεί δυναμικά από κάποιο πρόγραμμα και σαν φόρμα είναι πάντα η ίδια. Με αυτήν θα εξηγήσουμε τα cookies και sessions. Σε αντίθεση με αυτήν η στοιχειώδης 'φόρμα' που δημιουργήσαμε και επιδείξαμε με το HTTPGetServlet είναι δυναμική. Δημιουργήθηκε από κάποιο πρόγραμμα (αυτό του servlet) που έτρεξε στον server, το οποίο κάλλιστα θα μπορούσε να την διαφοροποιεί ανάλογα με τα δεδομένα του περιβάλλοντος (π.χ. προηγούμενες επιλογές του χρήστη, κλπ).

## Cookies

Τα cookies είναι πληροφορία που ο server μπορεί να αποθηκεύσει στον δίσκο του client. Η πληροφορία παραμένει προσβάσιμη από τον server για κάποιο χρόνο και κάτω από ορισμένες συνθήκες. Έτσι ένας server μπορεί, εξυπηρετώντας μεγάλο αριθμό clients, να συμπεριφέρεται στον καθένα με ελαφρά εξειδικευμένο τρόπο, χωρίς ο ίδιος ο server να είναι υποχρεωμένος να διατηρεί ένα τεράστιο όγκο πληροφορίας. Τυπική περίπτωση είναι όταν επισκεπτόμαστε μία ιστοσελίδα την επομένη ημέρα και βλέπουμε αυτήν να θυμάται τις επιλογές που κάναμε την προηγούμενη. Η πληροφορία αυτή δεν κρατήθηκε στον server, αλλά στον ίδιο τον δικό μας client. Είναι άλλωστε χρήσιμη μόνον όταν ο συγκεκριμένος client, επανασυνδεθή στον ίδιο server.

Παραπάνω είδαμε τις μεθόδους των αντικειμένων HttpServletRequest και HttpServletResponse. Όλη η σχετιζόμενη πληροφορία πάνω στην γραμμή μεταφέρεται στην επικεφαλίδα των μηνυμάτων HTTP όταν πρόκειται για GET και στο σώμα όταν πρόκειται για POST. Στον παρακάτω κώδικα CookieServlet, ο server δημιουργεί ένα cookie στον client, το οποίο μετά λαμβάνει υπ όψιν. Όλα τα cookies αποστέλλονται από τον client πίσω στον server μέσα σε κάθε GET προς τον συγκεκριμένο server, από τον οποίο έχουν αρχικά προέλθει. Πρακτικά δηλαδή υπενθυμίζει σε κάθε ευκαιρία στον server: σε εμένα, σαν client, είχες στείλει σε προηγούμενη σύνδεση αυτήν την πληροφορία. Βεβαίως τα cookies δεν μπορούν να συσσωρευτούν συνεχώς μέσα στον client. Συνήθως σβήνονται σύμφωνα με μία εκπνοή χρόνου (timeout) για το καθένα, αλλά και με άλλους τρόπους.

```
// Setting and Retrieving Cookies
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class CookieServlet extends HttpServlet {
    private String names[] = { "Europe", "Asia", "America", "Africa" };
    private String cities[] = { "Athens and Rome", "Peking and Amman", "New York", "Cairo" };

    public void doPost( HttpServletRequest request,           // reaction to the reception of POST
                      HttpServletResponse response ) throws ServletException, IOException
    {
        PrintWriter output;
        String conti = request.getParameter( "continent" ); // choice made will be sent back to client
        Cookie c = new Cookie( conti, getCities( conti ) ); // to be stored there as a cookie
        c.setMaxAge( 120 ); // seconds until cookie removed
        response.addCookie( c ); // must precede getWriter
        response.setContentType( "text/html" );
    }
}
```

```

output = response.getWriter();

// send HTML page to client
output.println( "<HTML><HEAD><TITLE>" );
output.println( "Cookies" );
output.println( "</TITLE></HEAD><BODY>" );
output.println( "<P>Welcome to Cookies!<BR>" );
output.println( "<P>" );
output.println( conti );
output.println( " is an interesting continent !" );
output.println( "</BODY></HTML>" );
output.close (); // close stream
}
public void doGet( HttpServletRequest request, // reaction to the reception of GET
                  HttpServletResponse response )
                  throws ServletException, IOException
{
    PrintWriter output;
    Cookie cookies[];

    cookies = request.getCookies(); // get client's cookies

    response.setContentType( "text/html" );
    output = response.getWriter();

    output.println( "<HTML><HEAD><TITLE>" );
    output.println( "Cookie with cities has been read !" );
    output.println( "</TITLE></HEAD><BODY>" );

    if ( cookies.length != 0 ) { // many cookies !!
        output.println( "<H1>Recommended Destinations</H1>" );
        // get the name of each cookie
        for ( int i = 0; i < cookies.length; i++ )
            output.println(
                "In " + cookies[ i ].getName() + " we recommend to visit "
                + cookies[ i ].getValue() + "<BR>" );
    }
    else {
        output.println( "<H1>Sorry, no recommendation possible !</H1>" );
        output.println( "You did not select a continent or " );
        output.println( "the cookies have expired." );
    }

    output.println( "</BODY></HTML>" );
    output.close(); // close stream
}

private String getCities( String conString)
{
    for ( int i = 0; i < names.length; ++i )
        if ( conString.equals( names[ i ] ) )
            return cities[ i ];

    return ""; // no matching string found
}
}

```

**Κώδικας 4.2.** Το CookieServlet για επίδειξη των cookies

Στον παραπάνω κώδικα

- η doPost δημιουργεί και αποστέλλει στον client ένα cookie, στο οποίο αποθηκεύει για το μέλλον την επιλογή μίας ηπείρου, που έγινε στον client και ελήφθη στον server με μήνυμα POST (βλ. φόρμα DestinationSelection.html παρακάτω). Έρχεται δηλαδή από τον client μία παράμετρος με το όνομα continent, η τιμή της διαβάζεται με την μέθοδο getParameter (βλ. μέθοδο doPost) του αντικειμένου request και φτιάχνεται σαν cookie το ζεύγος των strings: η ήπειρος (String conti) και το String των πόλεων σε αυτή (String που επιστρέφει η getCities). Κάθε cookie είναι ένα ζεύγος ονόματος παραμέτρου / τιμής, π.χ. εδώ Europe= Athens and Rome. Μετά τίθεται σαν χρόνος ζωής του cookie τα 120sec και το cookie προστίθεται (addCookie) στο αντικείμενο response, ορίζονται τα υπόλοιπα στοιχεία της επικεφαλίδας του μηνύματος και κατά τον συνήθη τρόπο (διαδοχικά println) διαμορφώνεται το σώμα της απόκρισης. Με το output.close κλείνει η ροή αυτή και αποστέλλεται η απόκριση (response). Μέσα στην επικεφαλίδα της περιλαμβάνεται το cookie. Η απόκριση επιβεβαιώνει οπτικά με την σελίδα html στον χρήστη την επιλογή του, αλλά και προκαλεί την αποθήκευση της επιλογής αυτής μέσα στον δίσκο του.
- στην doGet διαβάζεται το cookie μέσα από την επικεφαλίδα του μηνύματος GET που κατέφθασε από τον client. Είπαμε παραπάνω ότι ο browser οφείλει χωρίς άλλη δική μας ενέργεια να στείλει στον server, όλα τα cookies που έχει και έχουν προέλθει από αυτόν. Τα cookies ευρίσκονται στην επικεφαλίδα του GET και εξάγονται από αυτή μέσα στην doGet με ένα request.getCookies(). Από εκεί και πέρα δημιουργείται κατά τα γνωστά μία νέα σελίδα html, η οποία αποστέλλεται πίσω στον χρήστη.

Για την δοκιμή των παραπάνω φτιάχνουμε και δύο ιστοσελίδες/φόρμες html στον browser μας. Και οι δύο έχουν την ιδιότητα της φόρμας, δηλαδή περιέχουν στοιχεία GUI με τα οποία ο χρήστης καθορίζει τις επιλογές του. Στην παρακάτω DestinationSelection.html έχουμε αποκλειστική επιλογή με κουμπιά ="radio" και κουμπιά πατήματος για ="submit" και ="reset". Όλη η λειτουργικότητα αυτών είναι ευθύνη του (οποιουδήποτε !) browser. Αυτός είναι επίσης υπεύθυνος να αποστέλλει τα ζεύγη ονόματος /τιμής των παραμέτρων που επιλέγει ο χρήστης. Αυτές ορίζονται μέσα στον .html κώδικα της φόρμας, για παράδειγμα NAME="continent" VALUE="Europe". Με πάτημα του submit, οι επιλογές φεύγουν προς τον server. Ποίος είναι αυτός καθορίζεται με το FORM ACTION= ... . Ορίζεται επίσης ο τρόπος αποστολής. Αυτός καλείται 'μέθοδος', δηλ. METHOD="POST" ή στην άλλη φόρμα METHOD="GET". Η πρώτη θα προκαλέσει αποστολή μηνύματος POST με τις παραμέτρους στο σώμα του και κλήση της doPost() στον server, η άλλη αποστολή μηνύματος GET με τις παραμέτρους και τα cookies στην επικεφαλίδα του και κλήση της doGet() στον server.

```
<!-- A form for selection a of a continent. Selection to be sent by POST -->
<HTML>
<HEAD>
  <TITLE>Cookie will be written in our disc</TITLE>
</HEAD>
<BODY>
  <FORM ACTION="http://localhost:8080/myExamplesDir/firstCookieDemo"
                                     METHOD="POST">
    <STRONG>Select the Continent of your Destination:<br> </STRONG>
    <PRE>
    <INPUT TYPE="radio" NAME="continent" VALUE="Europe">check here for Europe<BR>
    <INPUT TYPE="radio" NAME="continent" VALUE="Asia">check here for Asia<BR>
```

```

    <INPUT TYPE="radio" NAME="continent" VALUE="America" CHECKED>check here for
America<BR>
    <INPUT TYPE="radio" NAME="continent" VALUE="Africa">check here for Africa<BR>
</PRE>
    <INPUT TYPE="submit" VALUE="Submit">
    <INPUT TYPE="reset"> </P>
</FORM>
</BODY>

</HTML>

```

### Κώδικας 4.3. Η φόρμα DestinationSelection.html με αποστολή μέσω POST

Στην παρακάτω DestinationSelection.html ο χρήστης απλά πατά το στοιχείο submit με τιμή "Recommend Cities" (ουσιαστικά εδώ το label που τυπώνεται πάνω στο κουμπί). Αυτό προκαλεί αποστολή μηνύματος GET. Το ενδιαφέρον εδώ είναι ότι μέσα στην επικεφαλίδα του αποστέλλονται και τα cookies και μάλιστα χωρίς εμείς να γράψουμε τίποτα ιδιαίτερο για αυτό.

```

<!-- Cities available from the chosen continent -->
<HTML>
<HEAD>
    <TITLE>Cookie taken into Account</TITLE>
</HEAD>
<BODY>
    <FORM ACTION="http://localhost:8080/myExamplesDir/firstCookieDemo"
        METHOD="GET">
        Press
        <INPUT TYPE=submit VALUE="Recommend Cities">
        for cities in your chosen continet
    </FORM>
</BODY>
</HTML>

```

### Κώδικας 4.4. Η φόρμα CitiesRecommendation.html με αποστολή μέσω GET

Θα μπορούσαμε να εκθέσουμε την νέα εφαρμογή σε ένα τελείως ανεξάρτητο κατάλογο (π.χ. MyCookiesDir) κάτω από το webapps σύμφωνα με την γνωστή δομή του Σχήματος 4.1. Αντ' αυτού προτιμάμε να εκθέσουμε και το HTTPGetServlet και το CookieServlet μαζί για να γνωρίσουμε και αυτήν την περίπτωση. Προς τούτο κατασκευάζουμε ένα MyExamplesDir και κάτω από αυτό το index.html και το WEB-INF. Μέσα στο τελευταίο βάζουμε το παρακάτω web.xml και το classes, όπου τίθενται και το HTTPGetServlet.class και το CookieServlet.class. Το νέο αρχείο WAR δημιουργείται μέσα από το MyExamplesDir με

```
jar -cvfM MyExamplesDir.war WEB-INF index.html
```

και τίθεται κάτω από το C:\jakarta-tomcat-7.7.2\webapps. Το παλαιό HTTPGetServlet ενεργοποιείται τώρα με δύο τρόπους, κτυπώντας

```

είτε το νέο          http://localhost:8080/myExamplesDir/firstServletDemo
είτε το παλαιό       http://localhost:8080/myServletDir/firstServletDemo

```

Στο παρακάτω web.xml βλέπουμε πάλι τον συνδετικό ρόλο του στοιχείου servlet-name μεταξύ των στοιχείων servlet και servlet-mapping ξεχωριστά για κάθε μία από τις δύο εφαρμογές (HTTPGetServlet και CookieServlet).



```

<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
  Copyright 1999-2004 The Apache Software Foundation
-->

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<!-- web-app element was originally found empty - the following are additions
  for the demonstration of HTTPGetServlet -->
<web-app>
  <servlet> <!-- the following name will refer to the sepecified class -->
    <servlet-name>ServletDemo</servlet-name>
    <servlet-class>HTTPGetServlet</servlet-class>
  </servlet>
  <servlet> <!-- the following name will refer to the sepecified class -->
    <servlet-name>CookieDemo</servlet-name>
    <servlet-class>CookieServlet</servlet-class>
  </servlet>

  <servlet-mapping><!-- the following (same) name will called as the specified url patttern
    so hitting http://localhost:8080/myExamplesDir/firstServletDemo/* (any thing)
    will lead to HTTPGetServlet -->
    <servlet-name>ServletDemo</servlet-name>
    <url-pattern>/firstServletDemo/*</url-pattern>
  </servlet-mapping>
  <servlet-mapping><!-- the following (same) name will called as the specified url patttern
    so hitting http://localhost:8080/myCookieDir/firstCookieDemo/* (any thing)
    will lead to CookieServlet -->
    <servlet-name>CookieDemo</servlet-name>
    <url-pattern>/firstCookieDemo/*</url-pattern>
  </servlet-mapping>
</web-app>

```

**Σχήμα 4.4.** Το web.xml για δύο εφαρμογές (HTTPGetServlet.class και CookieServlet.class)

Και τέλος η δοκιμή. Πατάμε στον browser την φόρμα DestinationSelection.html και επιλέγουμε μια ήπειρο. Με το submit η επιλογή μας φθάνει στον server, ο οποίος την επιβεβαιώνει. Κλείνουμε τον browser. Το ίδιο το HTTP είναι stateless. Ρωτήσαμε / ζητήσαμε κάτι και λάβαμε την απόκριση, χωρίς κανένα παραμένον σημάδι (state). Και όμως εδώ αυτό δεν ισχύει διότι υπάρχουν τα cookies! Πατάμε λοιπόν μετά στον browser την φόρμα CitiesRecommendation.html, η οποία μας προσκαλεί να πατήσουμε το κουμπί 'Recommend Cities'. Όταν το πατάμε, φαινομενικά δεν στέλνουμε τίποτα, αλλά στα κρυφά πάει το cookie πίσω στον server. Έτσι βλέπομε το 'μαγικό' να 'θυμάται' ο server την επιλογή μας για την ήπειρο. Η 'μνήμη' του όμως είναι στον client! Αν κάνομε πολλές διαδοχικές επιλογές μέσω DestinationSelection.html, θα δούμε πράγματι το αποτέλεσμα πολλών cookies. Όσο καθυστερούμε το πάτημα του 'Recommend Cities', τόσο θα 'ξεχνιούνται' οι επιλογές μας, καθώς θα σβήνονται στον client λόγω εκπνοής χρόνου τα αντίστοιχα cookies. Αν θέλομε να δούμε και την διαβίβαση των cookies, χρησιμοποιούμε τον TCP Monitor. Βεβαίως πρέπει τότε στις δύο ιστοσελίδες μας να κτυπάμε αυτόν και όχι κατευθείαν τον server. Βλέπομε λοιπόν, όταν π.χ. επιλέγουμε διαδοχικά America, αμέσως μετά Europe και μετά κτυπάμε το CitiesRecommendation.html,

- μέσα στην απόκριση προς το POST μία ιδιαίτερη γραμμή της επικεφαλίδας να φέρει το cookie ως:  
Set-Cookie: America=New York; Expires 'ακριβής χρόνος'
- μετά  
Set-Cookie: Europe=Athens and Rome; Expires 'ακριβής χρόνος'
- και μέσα στο GET μία ιδιαίτερη γραμμή της επικεφαλίδας να φέρει δύο cookies διαχωριζόμενα με ';':  
Cookie: America=New York; Europe=Athens and Rome

Είναι φανερό ότι η χρήση των cookies είναι απαραίτητη για να αποφευχθεί η κατάληψη πόρων στον server, όταν αυτός θέλει να θυμάται την προϋστορία της αλληλεπίδρασής του με μεγάλο αριθμό πελατών. Οι πόροι αυτοί του server συνίστανται σε μνήμη και σε κώδικα για την λογική διαχείρισης της πληροφορίας αυτής. Από την άλλη μεριά, τίθεται θέμα ασφάλειας όταν ο server, χωρίς καν να το γνωρίζουμε γράφει και διαβάζει πληροφορία στον/από τον δίσκο μας. Όλοι οι browsers δίδουν τις δυνατότητες διαγραφής των cookies ή και πλήρους απενεργοποίησης του μηχανισμού αυτού, όπως εξηγείται παρακάτω.

Τέλος και μία δευτερεύουσα παρατήρηση. Όταν πατάμε το 'Submit' της DestinationSelection.html, εμφανίζεται σαν 'Address' στον browser η επιθυμητή διεύθυνση <http://localhost:8080/myExamplesDir/firstCookieDemo>

χωρίς τίποτε άλλο (η φόρμα έχει POST και όχι GET). Όταν όμως πατάμε το 'Recommended Cities' της CitiesRecommendation.html εμφανίζεται

<http://localhost:8080/myExamplesDir/firstCookieDemo?>

Το τελευταίο '?' υπάρχει εδώ λόγω του GET της φόρμας, δηλ. για να εισάγει την ακολουθία ενδεχομένων παραμέτρων που επιλέγησαν στην φόρμα. Εδώ βεβαίως δεν έχουμε τίποτα, διότι στην περίπτωση αυτή τον ρόλο τους αναλαμβάνουν τα cookies.

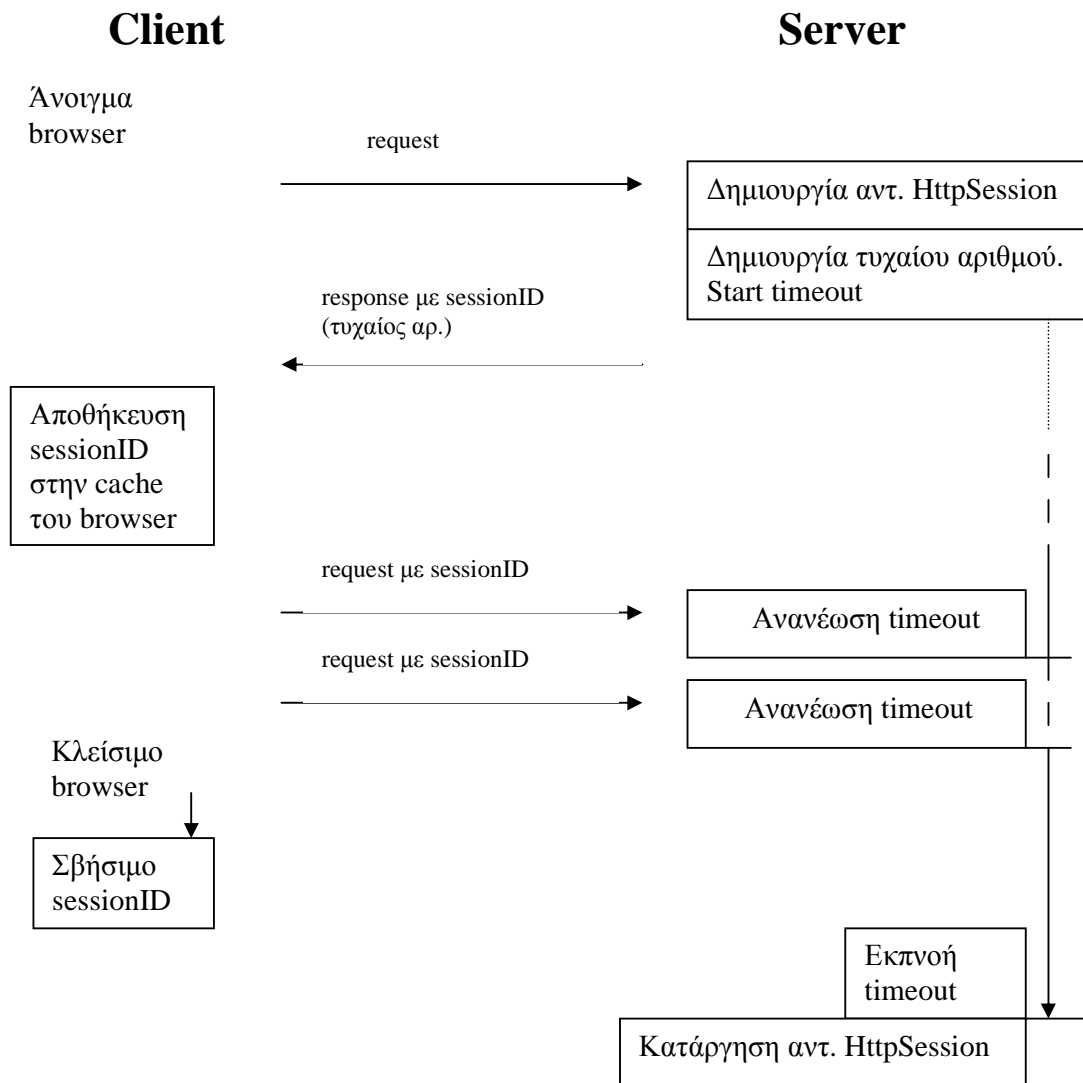
## Session

Η δημιουργία ενός αντικειμένου της class HttpSession στον web server, είναι ένας εναλλακτικός τρόπος να κρατιέται πληροφορία σχετική με την σύνοδο, αλλά την φορά αυτή στον web server. Με την πρώτη άφιξη ενός GET μπορούμε να δημιουργήσουμε αντικείμενο της HttpSession με την μέθοδο request.getSession(true) (του request object). Σε κάθε επόμενη άφιξη GET, η ίδια μέθοδος με παράμετρο false, μας επιστρέφει την αναφορά στο δημιουργηθέν αντικείμενο. Αν δηλαδή το mySession είναι ένα αντικείμενο HttpSession και το valueNames ένα array από String, με

```
mySession = request.getSession(false);
valueNames = mySession.getValueNames() ;
```

παίρνομε τα ονόματα των παραμέτρων που κρατιούνται στον web server στα πλαίσια αυτού του session. Το ενδιαφέρον είναι ότι η αναφορά στο αντικείμενο mySession διατηρείται μέσω

ενός και μοναδικού cookie, του sessionID (με τιμή έναν τυχαίο αριθμό δημιουργούμενο από τον server), σύμφωνα με το παρακάτω σχήμα. Όσα ευρίσκονται μέσα σε περίγραμμα



**Σχήμα 4.5.** Η εξέλιξη του session και του sessionID cookie

συντελούνται αυτόματα, εφόσον στο servlet χρησιμοποιείται προγραμματιστικά session. Το sessionID cookie διατηρείται όχι στον δίσκο του client αλλά στην cache του browser. Αυτό σημαίνει ότι αν κλείσουμε τον browser, χάνεται το αποθηκευμένο sessionID και μαζί με αυτό κάθε τρόπος αναφοράς από τον client στο αντικείμενο session του server. Στην μεριά του server, το αντικείμενο session θα σβησθεί με timeout (της τάξεως των 20 λεπτών). Άρα συμπερασματικά:

- τα cookies εν γένει είναι ζεύγη ονόματος/τιμής παραμέτρων που αποθηκεύονται στον δίσκο του client και διατηρούνται εκεί στην βάση ενός timeout

- ειδικά το sessionID cookie είναι ένα ζεύγος sessionID/τυχαίος αριθμός που αποθηκεύεται στην cache του client και διατηρείται εκεί ενόσω μένει ανοικτός ο browser. Το sessionID αποστέλλεται σαν παράμετρος σε κάθε αποστολή από τον client στον server για να μπορεί αυτός να συνδέσει τον browser με την συγκεκριμένη 'συνομιλία' (session). Αν, για ένα συγκεκριμένο διάστημα, δεν έρχεται και πάλιν η παράμετρος αυτή από τον client στον server, σβήνεται στον server το αντικείμενο session.

Το παρακάτω συγκεκριμένο παράδειγμα είναι το ίδιο με αυτό του Κώδικα 4.1., αλλά πραγματοποιημένο με session αντί cookies.

```
// Using Session
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class SessionServlet extends HttpServlet {
    private String names[] = { "Europe", "Asia", "America", "Africa" };
    private String cities[] = { "Athens and Rome", "Peking and Amman", "New York", "Cairo" };

    public void doPost( HttpServletRequest request,
                       HttpServletResponse response )
        throws ServletException, IOException
    {
        PrintWriter output;
        String conti = request.getParameter( "continent" );

        //          Replace the following 3 'cookie' lines
        //Cookie c = new Cookie( conti, getCities( conti ) );
        //c.setMaxAge( 120 ); // seconds until cookie removed
        //response.addCookie( c ); // must precede getWriter
        // BY
        //create a new session (with arg 'true')
        HttpSession session = request.getSession(true);
        session.putValue(conti, getCities( conti ));
        //

        response.setContentType( "text/html" );
        output = response.getWriter();

        // send HTML page to client
        output.println( "<HTML><HEAD><TITLE>" );
        output.println( "Session" );
        output.println( "</TITLE></HEAD><BODY>" );
        output.println( "<P>Welcome to Sessions!<BR>" );
        output.println( "<P>" );
        output.println( conti );
        output.println( " is an interesting continent !" );
        output.println( "</BODY></HTML>" );
        output.close(); // close stream
    }

    public void doGet( HttpServletRequest request,
                      HttpServletResponse response )
        throws ServletException, IOException
    {
        PrintWriter output;
```

```

//          Replace the following 2 'cookie' lines
//  Cookie cookies[];
//  cookies = request.getCookies(); // get client's cookies
// BY
//do not create a new session (with arg 'false')
HttpSession session = request.getSession(false);
String contiNames[];

if ( session != null )
    contiNames=session.getValueNames();
else
    contiNames = null;

response.setContentType( "text/html" );
output = response.getWriter();

output.println( "<HTML><HEAD><TITLE>" );
// output.println( "Cookie with cities has been read !" );
output.println( "</TITLE></HEAD><BODY>" );

// if ( cookies.length != 0 ) {

if ( contiNames != null && contiNames.length != 0 ) {
    output.println( "<H1>Recommended Destinations</H1>" );

// for ( int i = 0; i < contiNames.length; i++ )
//     output.println(
//         "In " + cookies[ i ].getName() + " we recommend to visit "
//         + cookies[ i ].getValue() + "<BR>" );
// }
for ( int i = 0; i < contiNames.length; i++ ) {
    String val = (String) session.getValue(contiNames[i]);
    output.println(
        "In " + contiNames[i] + " we recommend to visit "
        + val + "<BR>" );
    }
}

else {
    output.println( "<H1>Sorry, no recommendation possible !</H1>" );
    output.println( "You did not select a continent or " );
    output.println( "the session has been terminated." );
}

output.println( "</BODY></HTML>" );
output.close(); // close stream
}

private String getCities( String conString)
{
    for ( int i = 0; i < names.length; ++i )
        if ( conString.equals( names[ i ] ) )
            return cities[ i ];

    return ""; // no matching string found
}
}

```

#### Κώδικας 4.5. Το SessionServlet για επίδειξη της HttpSession

Φαίνονται σημειωμένες οι αλλαγές έναντι του αντίστοιχου servlet για cookies. Το αντικείμενο session της class HttpSession το αποκτώμε πάντα από το αντικείμενο request με

```
HttpSession session = request.getSession(true);
```

Σε αυτό αποθηκεύουμε τα ζεύγη όνομα παραμέτρου /τιμή που θέλουμε

```
session.putValue(conti, getCities( conti ));
```

Όταν μας ξαναμιλήσει ο client, αυτομάτως έχουμε την αναφορά στο σωστό session (μέσω του sessionID cookie) και ανακτώμε το αντικείμενο session απλά με

```
HttpSession session = request.getSession(false);
```

Από αυτό παίρνομαι το τι είχαμε αποθηκεύσει με

```
contiNames=session.getValueNames();
```

δηλ. κατ' αρχήν τα ονόματα των παραμέτρων (σαν vector). Ο σκοπός είναι προφανής. Ο server δεν χρειάζεται έξω από αυτήν την αντιμετώπιση του GET του συγκεκριμένου client, να θυμάται τίποτα, ενόσω ασχολείται με χιλιάδες άλλους. Όταν χρειασθεί, όπως εδώ, απλά ανατρέχει στο συγκεκριμένο αντικείμενο session. Αφού αποκτήσαμε τώρα τα ονόματα των παραμέτρων, ανακτώμε τις τιμές με

```
String val = (String) session.getValue(contiNames[i]);
```

Πρέπει πάντα να ελέγχουμε αν η αναφορά στο session (δηλ το sessionID cookie) έχει πραγματικά σταλεί από τον client, αλλιώς τίποτα από τα παραπάνω δεν είναι δυνατόν. Ο δε client πιθανόν να μην ανταποκρίνεται στην διακίνηση των cookies όπως εμείς λογαριάζομε (βλ. επομένη παράγραφο).

Ακολουθώντας την γνωστή διαδικασία με προφανείς αλλαγές, μπορούμε να δοκιμάσουμε το SessionServlet με τις δύο φόρμες του client, την DestinationSelection.html και την CitiesRecommendation.html. Εδώ όμως πρέπει να ανοίξομε την δεύτερη χωρίς να κλείσουμε τον browser, αλλιώς εξαφανίζεται το SessionID cookie από την cache του. Με το TCP Monitor μπορούμε να δούμε και τον ρόλο του SessionID cookie.

## Η Διαχείριση των Cookies στους Browsers

Ο χρήστης του browser έχει την δυνατότητα διαχείρισης των cookies. Συγκεκριμένα για τον

Firefox: Πατάμε Tools, Options και στην καρτέλα Privacy ανοίγουμε τα cookies. Μπορούμε να επιτρέψουμε την αποδοχή τους, να τα δούμε αποθηκευμένα, να σβήσουμε τα αποθηκευμένα και με 'ask me everytime' στο 'Keep Cookies' να τα βλέπομε και να τα εγκρίνομε/απορρίπτομε καθώς έρχονται.

IE5: Πατάμε Tools, Internet Options και στην καρτέλα Privacy πατάμε Advanced, Override Cookie Handling και επιλέγουμε prompt.

Netscape:

Τα παραπάνω είναι πολύ χρήσιμα για την κατανόηση και το debugging των παραδειγμάτων με cookies και session. Μπορούμε να δούμε αν τα cookies ενεγράφησαν πραγματικά στον client (ή τον browser), καθώς και αν ο server αντιμετωπίζει σωστά την περίπτωση να μην ανακτά τα cookies (ή το sessionID cookie) από τον δίσκο του client (ή την cache του browser), λόγω εκπνοής χρόνου, σβησίματος στον client, κλπ. Η cache του browser χάνει τα περιεχόμενά της όταν κλείσουμε τον browser.

## Server-Side Scripting και Σχέση Servlets με JSP (JavaServer Pages)

Όπως είδαμε στο παραπάνω παράδειγμα, η συγγραφή κώδικα και σελίδων HTML με servlets είναι αρκετά κοπιώδης με συνεχείς χρήσεις της εντολής `println`. Αντίθετα η JSP (JavaServer Pages), έχει σαν βάση την HTML σελίδα, μέσα στην οποία μόνο για την πραγματοποίηση της αλληλοδραστικότητας, παρεμβάλλεται κώδικας Java. Αυτή είναι μία σελίδα `.jsp`, όπου παρεμβάλλεται κώδικας μέσα σε HTML, αντί HTML μέσα σε κώδικα. Στην πραγματικότητα η JSP βασίζεται στα servlets. Η μηχανή της JSP παράγει από την `.jsp` σελίδα πηγαίο κώδικα servlet. Την πρώτη φορά που απαιτείται, μεταγλωττίζεται αυτός ο πηγαίος κώδικας σε ένα class file για την τρέχουσα και για κάθε μελλοντική χρήση. Έχουμε δηλαδή τα παρακάτω βήματα:

1. Ένας κοινός browser αναζητά (request) μία `.jsp` σελίδα
2. Ο server ανευρίσκει την αναζητηθείσα `.jsp` σελίδα και την παραδίδει στο 'JSP Servlet Engine'.
3. Την πρώτη φορά η `.jsp` σελίδα περνιέται από τον σχετικό parser, αλλιώς εκτελείται απ'ευθείας το βήμα 7.
4. Δημιουργείται ο πηγαίος κώδικας του servlet από το αρχείο `.jsp`. Αυτόματα δημιουργείται σειρά από `println` statements ισοδύναμα ως προς την τελική εμφάνιση με τα γραμμένα σε HTML.
5. Ο πηγαίος κώδικας μεταγλωττίζεται σε class.
6. Δημιουργείται το servlet με κλήση των μεθόδων `init` και `service`.
7. Τα παραγόμενα από το servlet δεδομένα αποστέλλονται από τον server στον client για παρουσίαση σαν κοινή HTML σελίδα

Το παραπάνω σενάριο αποτελεί server side scripting: ο πιο κοινός browser κτυπά `.jsp` σελίδες στον server, σαν να ήταν σελίδες `.html`. Ο browser δεν αντιλαμβάνεται τίποτα, διότι ο server δεν τις αποστέλλει πριν εκτελέσει τον ενσωματωμένο κώδικα Java και δημιουργήσει εκ του αποτελέσματος καθαρά `.html` σελίδες. Αυτές είναι που τελικά στέλνονται και τις οποίες εμφανίζει ο browser. Εφόσον εμπλέκεται κώδικας, οι σελίδες είναι δυναμικές, δηλ. αλλάζουν περιεχόμενο (ή και εμφάνιση) σαν αποτέλεσμα της εκτέλεσης του κώδικα στον server. Η αντίστοιχη τεχνολογία της Microsoft είναι η ASP (Active Server Pages), με την διαφορά ότι χρησιμοποιείται η VBScript αντί της Java. Υπάρχει και η περίπτωση της JavaScript μίας απλοποιημένης (και διαφορετικής) Java σε μορφή scripting γλώσσας. Υπάρχει και η περίπτωση του client-side scripting, όπου η σελίδα περιέχουσα JavaScript κατεβαίνει στον browser. Τότε αυτός, στην μεριά του (εφόσον έχει ενσωματωμένη δυνατότητα), κάνει την μετατροπή σε αμιγώς `.html` σελίδα και την παρουσιάζει. Το client-side scripting δεν είναι τόσο διαδεδομένο και δεν θα το εξετάσουμε εδώ περαιτέρω.

Θα φτιάξουμε ένα παράδειγμα server-side scripting περίπου ανάλογο των προηγούμενων (Κώδικες 4.3, 4.4). Γράφουμε σαν καθαρές `.html` εντολές, όσα ήταν πριν μέσα σε `println()`. Για το δυναμικό μέρος καταφεύγουμε στην Java. Έτσι έχουμε μέσα σε σελίδα `.html` τον κώδικα πάντα περικλειόμενο μεταξύ `<%...%>`. Στα σημεία αυτά πρέπει κανονικά μετά την εκτέλεση να δημιουργείται ένα τμήμα αποδεκτού κειμένου `.html`. Για πλήρη αντιστοιχία παίρνουμε την `SessionServlet.class` του παραδείγματος της παραγράφου Session και δημιουργούμε τις παρακάτω δύο `Continent.jsp` και `Cities.jsp` σελίδες.

```

..<%! private String names[] = { "Europe", "Asia", "America", "Africa" };
    private String cities[] = { "Athens and Rome", "Peking and Amman",
        "New York", "Cairo" };

    private String getCities( String conString)
    { for ( int i = 0; i < names.length; ++i )
        if ( conString.equals( names[ i ] ) ) return cities[ i ];
        return ""; } // no matching string found
%>
<HTML><HEAD><TITLE>JSP FIRST PAGE</TITLE></HEAD>
<BODY>
<% String conti = request.getParameter ( "continent" );
    session.putValue(conti, getCities( conti ));%>
<P>Welcome to JavaServer Pages - JSP !<BR>
<P>
    <%= "Good choice !" + conti %> is an interesting continent !
</BODY></HTML>

```

**Κώδικας 4.6.** Η Continent.jsp σελίδα δημιουργούσα session και ανταποκρινόμενη σε POST

```

<%!private String names[] = { "Europe", "Asia", "America", "Africa" };
    private String cities[] = { "Athens and Rome", "Peking and Amman",
        "New York", "Cairo" };

    private String getCities( String conString)
    {
        for ( int i = 0; i < names.length; ++i )
            if ( conString.equals( names[ i ] ) ) return cities[ i ];
        return ""; // no matching string found
    }
%>

<HTML><HEAD><TITLE> JSP Second Page </TITLE></HEAD>
<BODY>
<% String contiNames[];
    if ( session != null ) contiNames=session.getValueNames();
    else
        contiNames = null;

    if ( contiNames != null && contiNames.length != 0 ) { %>

        <H1>Recommended Destinations</H1>
        <%
            for ( int i = 0; i < contiNames.length; i++ ) {
                String val = (String) session.getValue(contiNames[i]);
                out.println(
                    "In " + contiNames[i] + " we recommend to visit "
                    + val + "<BR>" );
            }
        %>

        else { %>
            <H1>Sorry, no recommendation possible !</H1>
            You did not select a continent or the session has been terminated.
        %>
    }
</BODY></HTML>

```

**Κώδικας 4.7.** Η Cities.jsp σελίδα υποστηρίζουσα session και ανταποκρινόμενη σε GET



Ο κώδικας, ο οποίος κανονικά θα ήταν ενσωματωμένος μέσα στην μέθοδο service του servlet παρεμβάλλεται οπουδήποτε θέλουμε ανάμεσα σε '<%>' και '<%>'. Θυμίζουμε ότι αυτό σημαίνει κώδικας ο οποίος προηγουμένως ήταν μέσα στην doPost ή doGet, μέθοδοι οι οποίες προέρχονται από την service. Χρησιμοποιούμε ακόμη και το

```
<%= java expression giving String; %>
```

για να βγάλομε κατευθείαν κείμενο για .html. Γενικός κώδικας, έξω από το doPost ή doGet (π.χ. η βοηθητική μέθοδος getCities και δηλώσεις γενικών μεταβλητών), πρέπει να περικλείεται μεταξύ '<!' και '<%>'. Τα αντικείμενα request, response, session (και άλλα) είναι γνωστά και χρησιμοποιήσιμα χωρίς περαιτέρω ενέργειες από μέρους μας, όπως είχαμε στις παραγράφους περί cookies και session. Το αντικείμενο out (έναντι του output) είναι επίσης χρησιμοποιήσιμο για την έξοδο κειμένου για το html. Η out.println("some-string") που χρησιμοποιούμε είναι η προηγούμενη output.println("someString") και θα μπορούσε και αυτή να αντικατασταθεί με καθαρό html. Μένουν ανεξήγητες πολλές άλλες διευκολύνσεις και λεπτομέρειες, αλλά το τι συμβαίνει διαφαίνεται αρκετά καθαρά. Η κάθε σελίδα .jsp αυτόματα επιστρέφεται πίσω στα αντίστοιχα τμήματα της HttpSession.class από την οποία εμείς (χειροκίνητα) την δημιουργήσαμε. Τέλος για το deployment απλώς θέτομε τις Continent.jsp και Cities.jsp μέσα σε ένα folder, π.χ. JspExamples, κάτω από το webapps και αλλάζομε τις διευθύνσεις σε

```
http://localhost:8080/JspExamples/Continent.jsp και  
http://localhost:8080/JspExamples/Cities.jsp
```

μέσα στις δύο φόρμες. Κατά τα άλλα η συμπεριφορά του παραδείγματος δεν πρέπει να διαφέρει έναντι αυτού με session μέσα από servlet. Παρατηρούμε την αισθητή καθυστέρηση ανταπόκρισης, αλλά μόνον την πρώτη φορά που κτυπάμε την σελίδα .jsp. Όπως είπαμε, τότε γίνεται η δημιουργία της class ενός ισοδύναμου servlet. Από το σημείο αυτό, τούτο μένει στην μνήμη έτοιμο να ανταποκριθεί ταχύτατα όπως και στα προηγούμενα παραδείγματα. Σε επαγγελματικά περιβάλλοντα λαμβάνεται πρόνοια να μη υποστεί την καθυστέρηση αυτή ο 'πρώτος πελάτης' και προκαλείται από τον σύστημα μία εικονική κλήση για να επιτελεσθούν προκαταρκτικά τα ανωτέρω.

Αν φανταστούμε μία πραγματική εφαρμογή δυναμικών σελίδων με δεκάδες χιλιάδες γραμμές html για πολύπλοκες και (αισθητικά ωραίες) σελίδες με λίγες επιλογές και δυναμικά χαρακτηριστικά εδώ και εκεί, βλέπομε την κομψότητα που επιτυγχάνομε με την JSP. Θα είχαμε χιλιάδες println να κατακλύζουν τις λίγες γραμμές λογικής μέσα στο αντίστοιχο servlet. Από την άλλη, αν κάνομε το παραμικρό συντακτικό λάθος μέσα στην .jsp σελίδα (είτε στον κώδικα, είτε στην σωστή αλληλουχία των οριοθετήσεων με την καθαρή html) βλέπομε μηνύματα σφάλματος που ελάχιστη βοήθεια δίνουν. Επί πλέον δεν νοούνται δοκιμές και debugging έξω από το πλήρες περιβάλλον με την συμμετοχή του server. Ο λόγος βεβαίως είναι η πλήρης αυτοματοποίηση μέχρι την γλώσσα μηχανής και το deployment, στον οποίο δεν έχουμε καμία συμμετοχή και δυνατότητα επέμβασης.

Ανεξαρτήτως τεχνολογίας (cookies, session, JSP, κλπ) το πλήρες σενάριο θα ήταν, την πρώτη φορά που θα επισκεφθούμε το site να μας κατέβει η DestinationSelection σαν μία (στατική) σελίδα που παίζει τον ρόλο φόρμας για αυτόν που κάθετα στον browser. Αφού επιλέξομε την ήπειρο, κατόπιν να μας κατέβει μαζί με την επιβεβαίωση της επιλογής μας (δυναμικό μέρος) η CitiesRecommendation (άλλη στατική σελίδα σαν φόρμα), να πατήσομε το "Recommend Cities" και να μας επιστραφεί η τελική (δυναμική) σελίδα. Με την JSP είναι ζήτημα σχεδιασμού αν θα έχουμε μία ενιαία σελίδα, ή αν η μία θα μας οδηγεί σε άλλη ξεχωριστή, όπως περίπου κάναμε εδώ. Και πάλιν θα έπρεπε το κοινό μέρος να μην επαναλαμβάνεται, αλλά να διατίθεται σε πολλές σελίδες, είτε σαν κοινή μέθοδος ή ορισμοί είτε να εντάσσεται με κάποιο 'include'. Για όλα αυτά υπάρχουν απαντήσεις μέσα στα πλαίσια των ενεργών σελίδων (JSP, ASP, PHP, κλπ).

## 5. Εισαγωγή στην XML

Η XML (Extended Mark-up Language) είναι μια δηλωτική γλώσσα που ήρθε μετά την ASN1. Κύρια διαφορά είναι ότι στην XML χρησιμοποιούνται αποκλειστικά χαρακτήρες (character based, έναντι της ASN1 που είναι bit coded). Επιπλέον η δομή της είναι ταυτοχρόνως πιο απλή αλλά και πιο εύκαμπτη. Από άλλη οπτική γωνία, στην XML ένα κείμενο χρησιμοποιείται για την αναπαράσταση δομημένης πληροφορίας στο διαδίκτυο.

Όπως και στην HTML (Hyper Text Mark-up Language - και η XML - είναι γλώσσα που επισημαίνει επιπλέον πληροφορία αναφερόμενη στο βασικό κείμενο. Στην XML προσδιορίζουμε την πληροφορία αυτή με την βοήθεια ετικετών (tags), δηλ. με ένα αναγνωριστικό περικλειόμενο από <>. Με τις ετικέτες (tags) επιτυγχάνουμε το “markup”. Tags υπάρχουν και στην HTML, π.χ. το ζεύγος <B> </B> ή το μοναδικό <Br>. Αυτά όμως είναι προκαθορισμένα και σχετίζονται μόνο με την εμφάνιση του κειμένου. Τα tags της XML είναι προσδιορίσιμα ελεύθερα από τον χρήστη, ο οποίος μπορεί να τους δίδει και όποια σημασία αυτός ορίσει. Για τούτο πρέπει και πάντα να ζευγαρώνονται, δηλ. <text1>my car is here</text1>. Το αναγνωριστικό <text1> </text1> δεν προσδιορίζει αναγκαστικά το πώς θα εμφανισθεί το περικλειόμενο κείμενο (όπως θα έκανε το <B> </B> της HTML), αλλά μπορεί π.χ. να συνδυάζεται με μία ενέργεια, όπως θα καθορίζει δικός μας κώδικας που θα επεξεργασθεί το αρχείο XML. Το ζεύγος <text1> και </text1> ορίζει ένα **στοιχείο** (element), του οποίου το όνομα είναι το text1 και το περιεχόμενο οτιδήποτε περικλείεται μεταξύ του <text1> (opening tag) και του </text1> (closing tag).

Το επόμενο βήμα είναι η ενθυλάκωση (nesting) που δίδει την δύναμη να κατασκευασθούν στοιχεία με πολύπλοκη εσωτερική δενδρική δομή, π.χ.

```
<message>
  <to>you@yourAddress.com</to>
  <from>me@myAddress.com</from>
  <subject>XML Is Really Cool</subject>
  <text>
    How many ways is XML cool? Let me count the ways...
  </text>
  <another_text/>
</message>
```

Τα πυκνοτυπωμένα είναι tags (αναγνωριστικά), πάντα ζευγαρωμένα. Τόσο τα tags, όσο και η περικλειόμενη πληροφορία είναι πάντα κείμενο (text) (μία εξαίρεση παρακάτω). Η XML είναι εύκαμπτη, ευκόλως εννοούμενη, άλλα βεβαίως και σπάταλη σε κωδικοποίηση (text αντί bits). Στο παραπάνω παράδειγμα έχουμε στοιχεία με το όνομα message, to, from, subject, text, another\_text. Το στοιχείο another\_text είναι κενό (empty element) και αντί ζεύγους <another\_text> (ανοίγοντος) και </another\_text> (κλείνοντος tag), έχουμε την συντομογραφία <another\_text/>, που ανοίγει και κλείνει συγχρόνως. Συνήθως, όπως παρατηρούμε και στο παραπάνω παράδειγμα, το περιεχόμενο του στοιχείου είναι άλλο (άλλα) ενθυλακωμένα στοιχεία, ή / και κείμενο (text). Ένα στοιχείο με περιεχόμενο μόνο κείμενο λέγεται text element. Επίσης για το περικλειόμενο κείμενο χρησιμοποιείται και ο ορισμός PCDATA (Parsed Character Data), καθόσον το κείμενο αυτό λαμβάνεται υπόψιν σαν ενιαία και ξεχωριστή οντότητα από τους parsers (βλ. παρακάτω).

Το πρώτο παράδειγμα είναι ένας κατάλογος διευθύνσεων (address book). Η πρώτη σειρά θα εξηγηθεί παρακάτω. Με cut & paste αντιγράψτε το παρακάτω κείμενο στο Notepad και σώστε το με .xml. Μετά ανοίξτε το με τον browser (IE5). Το παρακάτω αποτελεί ένα πλήρες κείμενο XML (XML document).

```
<?xml version="1.0"?>
<addressbook>
  <!-- This is my good friend Bob. -->
  <contact>
    <name>Bob Rizzo</name>
    <city>New York</city>
    <address>1212 W 304th Street</address>
    <state>New York</state>
    <zip>10011</zip>
    <phone>
      <voice>212-555-1212</voice>
      <fax/>
    </phone>
    <email>frizzo@fruity.com</email>
    <web>http://www.fruity.com/rizzo</web>
    <company>Bob's Ratchet Service</company>
  </contact>

  <!-- This is my old college roommate Peter. -->
  <contact>
    <name>Peter Rosenberg</name>
    <address>1162 E 412th Street</address>
    <city>New York</city>
    <state>New York</state>
    <zip>10011</zip>
    <phone>
      <voice>212-555-1818</voice>
      <fax>212-555-1819</fax>
    </phone>
    <email>srosenberg@fruity.com</email>
    <web>http://www.fruity.com/rosenberg</web>
    <company>Rosenberg's Shoes & Glasses</company>
  </contact>
</addressbook>
```

### Παράδειγμα 5.1. Κείμενο addressbook.xml

Η δενδρική μορφή είναι προφανής και μπορούμε στον browser να ανοιγοκλείνουμε τα στοιχεία (φύλλα κάποιου βάλθους). Παρατηρήστε επίσης πώς διαχωρίζονται τα σχόλια, πώς κλείνει ένα στοιχείο (element) που δεν έχει πληροφορία, (υπάρχει παραπάνω τέτοιο παράδειγμα), πώς γράφονται τα δεσμευμένα σύμβολα &, ' (το “ γίνεται &quot; και τα <,> &lt; , &gt; αντίστοιχα – δοκιμάστε τα). Ο browser απλώς αντιλαμβάνεται και αποκωδικοποιεί το συντακτικό της δομής και την παρουσιάζει σαν δένδρο. Παραβιάζοντας την σύνταξη βλέπομε σχετικές παρατηρήσεις πού εκδίδει ο browser.

## Μεταβλητές Ιδιοτήτων – Attributes

Κάθε στοιχείο (element) δυνατόν να έχει και ιδιότητες (attributes) οι μεταβλητές των οποίων μαζί με τις τιμές τους συντάσσονται σύμφωνα με το παράδειγμα

```
<movie type="comedy" rating="for adults" year="1998"> .... </movie>
```

Το tag (movie) συνοδεύεται από ότι ο χρήστης θέλει να περιλάβει σαν ιδιότητες (comedy, rating, year). Κάθε μεταβλητή ιδιότητας φέρει σαν τιμή ένα text string. Η σημασία και χρήση των attributes είναι θέμα του χρήστη ή της εφαρμογής που επεξεργάζεται το XML. Κάθε μεταβλητή ιδιοτήτων θα μπορούσε να αποτελέσει υποστοιχείο (sub element), δηλ. το παραπάνω παράδειγμα θα μπορούσε να γραφεί σαν:

```
<movie>
  <type>comedy</type>
  <rating>for adults</rating>
  <year>1998</year>
</movie>
```

Το τι θα προτιμηθεί έγκειται συνήθως στην εμπειρία του προγραμματιστή (π.χ. ένα μακρύ κείμενο δεν θα ήταν κομψό ή πρακτικό να περιλαμβάνεται στις ιδιότητες).

Η πρώτη γραμμή `<?xml version="1.0"?>` αποτελεί εντολή προς τον parser (όλες οι εντολές αυτές περικλείονται από `<? ?>`), δηλ. το πρόγραμμα που αποκωδικοποιεί την δομή του XML. Μέσα σε αυτήν την εντολή το 'version' είναι μεταβλητή ιδιοτήτων. Το string xml είναι δεσμευμένη λέξη και δεν μπορείτε να χρησιμοποιηθεί για attribute, ούτε για tag.

## Άλλα Περιεχόμενα Στοιχείου

Πέραν του κειμένου, μπορούμε να περιλάβουμε οποιαδήποτε κωδικοποιημένη ή μη ψηφιακή πληροφορία (εικόνα, video), η οποία όμως περικλείεται από `<![CDATA[ ..]]>`. Ο όρος CDATA (Character Data) είναι σε αντίθεση με τον PCDATA (Parsed CDATA), δηλαδή η CDATA δεν υπόκειται σε parsing. Ο parser αγνοεί το περικλειόμενο, το οποίο διοχετεύεται όπου προβλέπει η εφαρμογή.

Ένα πλήρες κείμενο XML (XML document), είναι ένα αρχείο text (δημιουργημένο με Notepad, Word, κάποια εφαρμογή κλπ) το οποίο έχει την κατάληξη .xml. Περιέχει μία και μόνη δενδρική δομή, κάτω από μία και μοναδική ρίζα. Το στοιχείο ρίζας (root element) συμβολίζεται γενικά με / και δεν φαίνεται στο αρχείο κειμένου που αντιπροσωπεύει το .xml. Ακριβώς κάτω από την ρίζα υπάρχει το μοναδικό document element, στα παραπάνω παραδείγματα έχουμε /addressbook, /movie. Το document element (addressbook, movie) δεν χρειάζεται αναγκαστικά να συμπίπτει με το όνομα του αρχείου. Έτσι κάθε κείμενο XML παρομοιάζεται με ένα σύστημα αρχειοθέτησης με root directory το .xml αρχείο, folders, subfolders τα elements και files το περιεχόμενο των elements. Αυτή η αναλογία βοηθά μνημοτεχνικά την κατανόηση της δομής του XML, χωρίς όμως περαιτέρω προεκτάσεις.

## Παράδειγμα

Το παρακάτω vehicles.xml περιέχει όσα παρουσιάστηκαν παραπάνω και θα χρησιμεύσει στην κοινό κείμενο εφαρμογής μεθόδων και τεχνολογιών για την επεξεργασία XML κειμένων.

```
<?xml version="1.0"?>

<!--This is the way to write a comment-->
<vehicles>The root element (must be exactly one) is opened; this the first opening tag
  <cars type = "passenger car">
    <sedan no_doors="4"/><!--This was both an opening and a closing tag-->
    <jeep price="high">Another jeep defined below</jeep>
    <formula1 price= "very high">
      <![CDATA[<tag_to_be_ignored>Whatever placed here remains unparsed,
        </tag_to_be_ignored>, etc, qwerty, <, !, ""=, ASDD, etc,...]]>
    </formula1>
  </cars>

  <trolleys/><!--only a placeholder - to be expanded later-->

  <ambulances type ="public use" importance="1">
    <?targetdatabase SELECT * FROM Hospitals?>
  </ambulances>

  <jeeps type="military car" importance="2">
    <jeep price="3000" colour="brown" user="army"/>
    <jeep price="5200" user="navy" colour="gray"/>
    <jeep price="2800" colour="blue" user="air force"/>
  </jeeps>

  <lories>
    Some text belonging to 'lories'
    <lory1 no_of_axes="2" tons="three">lory one text</lory1>
    <lory2 no_of_axes="6" tons="14">
      <traction motor="400PS" no_of_axes="3"/>
      <tender lenght="15.53m" no_of_axes="3"/>
    </lory2>
    More text belonging to 'lories', placed between its subelements
    <military_lory/>
  </lories>

</vehicles>
<!--the root element has just been closed; this is necessarily the last closing tag-->
```

**Παράδειγμα 5.2.** Το κείμενο vehicles.xml (θα χρησιμοποιείται και παρακάτω)

## Namespaces

Με τα namespaces εξασφαλίζεται ελευθερία και ανεξαρτησία στην επιλογή του κειμένου των tags, δηλαδή στην ονοματοδότηση των επί μέρους στοιχείων (elements) ανάμεσα σε διαφορετικά κείμενα. Ας υποθέσουμε ότι ένα στοιχείο υπάρχει σε διαφορετικά κείμενα με το ίδιο όνομα.

Δηλαδή έχουμε `<any_tag>text1</any_tag>` στο `doc1.xml` και `<any_tag>text2</any_tag>` στο `doc2.xml`. Αν τώρα φέρομε μαζί τα δύο αυτά στοιχεία, για παράδειγμα σε ένα τρίτο `doc7.xml`, δεν θα μπορούν να διαφοροποιούνται με βάση το όνομά τους. Η λύση έρχεται με ένα πρόθεμα και μία σύμβαση πού υποδηλοί το πού έχει ορισθεί το κάθε στοιχείο. Έτσι γράφουμε στο `doc7.xml` το tag του πρώτου σαν `ns1:any_tag` και το tag του δεύτερου σαν `ns2:any_tag`. Όλα τα στοιχεία πού ορίζονται τοπικά μέσα στο `doc7.xml` δεν φέρουν κανένα πρόθεμα, δηλαδή έχουν το default namespace. Επομένως δυνατόν να συνυπάρχουν και τα τρία tags: `ns1:any_tag`, `ns2: any_tag` και `any_tag`. Πώς όμως εξασφαλίζεται η μοναδικότης των προθεμάτων πού επιλέγονται ανεξάρτητα σε διάφορα σημεία του κόσμου; Τα ίδια τα προθέματα `ns1`, `ns2` ορίζονται αποκλειστικά στο `doc7.xml` και διασυνδέονται με κάποια URIs, των οποίων η μοναδικότητα είναι έτσι ή αλλιώς εξησφαλισμένη. Έτσι μπορούμε να ορίσουμε στην αρχή του δικού μας `vehicles.xml` (μέσα στο opening tag αυτού) τα `ns1`, `ns2`

```
<vehicles xmlns:it= "http://www.italian_trolleys.com"
          xmlns:ru= "http://www.russian_trolleys.com">
```

και μετά να αναφερόμαστε στο κείμενο μας σε `it:trolleys`, `ru:trolleys`, τα οποία είναι διαφορετικά του δικού μας `trolleys`. Άρα τα namespaces δίδουν την δυνατότητα συσχέτισης στοιχείων ενός κειμένου XML με συγκεκριμένα URIs. Προσέξτε επίσης ότι αρκεί κάποιος ορισμός URIs στην αρχή του στοιχείου (συνήθως του root) όπου θα γίνει χρήση κάποιου namespace. Η χρήση κάποιου tag σαν `it:trolleys`, `ru:trolleys` μέσα στο κείμενο δεν απαιτεί σύνδεση με το αντίστοιχο URI, ούτε καν να είναι αυτό το URI υπαρκτό. Μπορούμε τώρα να αντικαταστήσουμε μέσα στο στοιχείο `<vehicles >` την γραμμή `<trolleys/><!--only a placeholder - to be expanded later-->` με

```
<it:trolleys design="elegant">Italian trolleys are very old and no longer in operation
  <it:capacity passenger_no="108"/>
</it:trolleys>

<ru:trolleys design="spartan">Russian trolleys are old and some are in operation
  <ru:capacity passenger_no="124"/>
</ru:trolleys>

<trolleys design="modern">Greek trolleys are new and many in operation
  <capacity passenger_no="132"/>
</trolleys>
```

### Παράδειγμα 5.3. Εισαγωγή των namespaces

Παραπάνω βλέπομε την χρήση των δύο ορισθέντων namespaces καθώς και του default, πού δεν φέρει κανένα πρόθεμα.

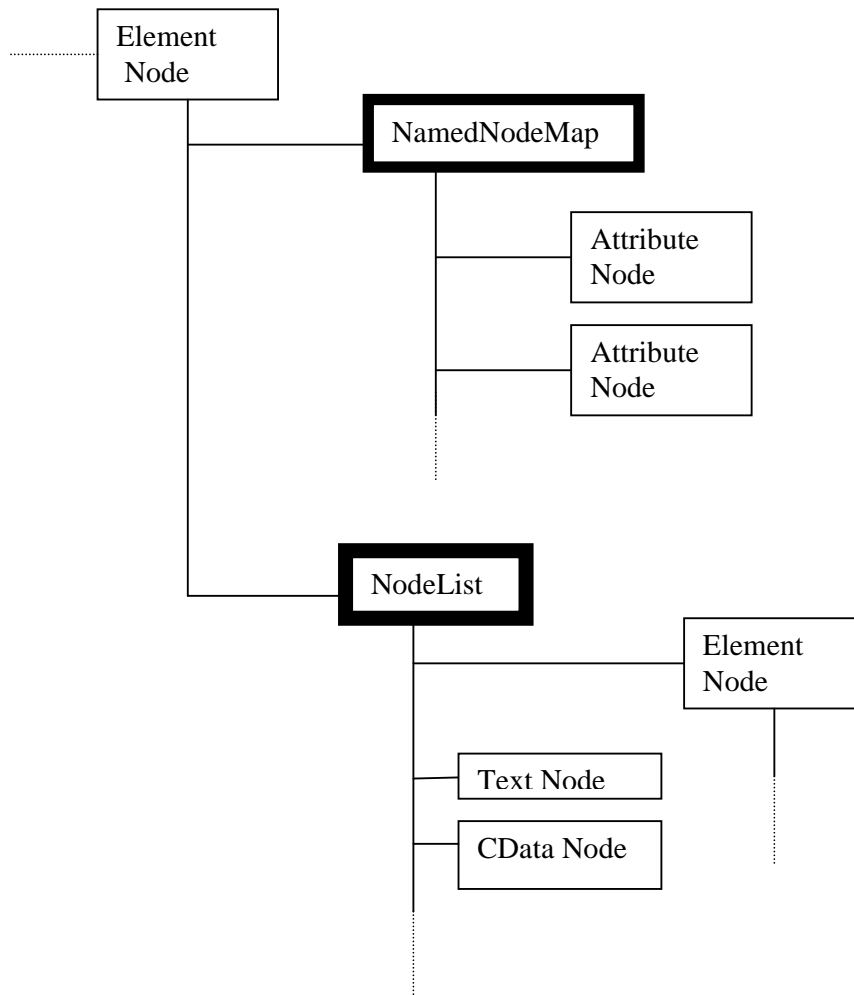
## 6. Σημασία και Χρήση της XML

Η σημασία της XML έγκειται στο γεγονός ότι διατίθενται έτοιμοι τρόποι σύνδεσης της δομής των κειμένων της με τα πλέον σύγχρονα προγραμματιστικά περιβάλλοντα. Τέτοια υπάρχουν για διάφορες γλώσσες, αλλά εμείς θα επικεντρωθούμε κυρίως στην Java. Στην διαδικασία αυτή εμπλέκεται πάντα ένας parser. Ο parser κατέχει κεντρικό ρόλο στην αντιστοίχιση και αλληλοσύνδεση της δενδρικής δομής του κειμένου XML με την αντίστοιχη (συνήθως αντικειμενοστραφή) δομή στην χρησιμοποιούμενη γλώσσα, με δεδομένο ότι το κείμενο XML είναι ένας συρμός χαρακτήρων ανταλλάσσιμος μέσω μίας επικοινωνιακής σύνδεσης, ενός αρχείου στον δίσκο, κλπ. Ο parser αποκτά τέτοια σημασία, που θεωρείται πλέον *core technology*. Οι παρακάτω παρουσιαζόμενοι parsers είναι οι πλέον διαδεδομένοι (και open source!), αλλά υπάρχει εκτεταμένο περαιτέρω αντικείμενο για ειδικά περιβάλλοντα, ιδιαίτερες απαιτήσεις απόδοσης, ελαχιστοποιημένο κώδικα (π.χ. για ενσωματωμένα συστήματα), κλπ.

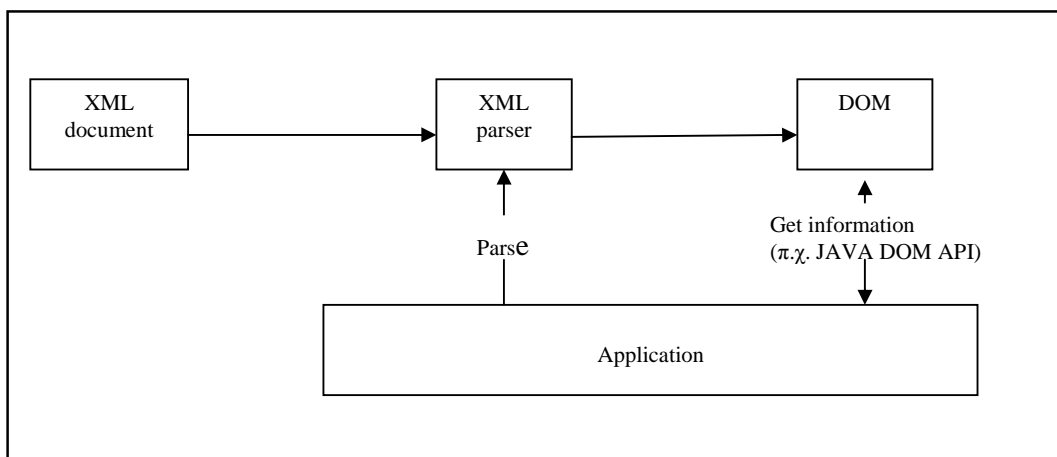
### XML και DOM (Document Object Model)

Το **Document Object Model (DOM)** μας βοηθά να θεωρήσουμε την δενδρική μορφή του XML κειμένου με αντικειμενοστραφή τρόπο και να επέμβουμε σε αυτό με συγκεκριμένα interfaces. Με το XML DOM μπορούμε να δημιουργήσουμε ένα XML κείμενο, να πλοηγηθούμε μέσα σε αυτό και να προσθέσουμε, μεταβάλουμε και αφαιρέσουμε στοιχεία του. Ένα XML κείμενο περνά από έναν parser, ο οποίος δημιουργεί και τα σχετικά αντικείμενα όπως προδιαγράφει το DOM. Με τα διατιθέμενα interfaces (μέσω JavaScript, VBScript, Java, κλπ) κάνουμε τις επιθυμητές επεξεργασίες και κατόπιν, αν θέλουμε, στην αντίθετη κατεύθυνση δημιουργούμε ένα νέο παρηλλαγμένο κατά τις επιθυμίες μας XML κείμενο. Κεντρική ιδέα του **DOM** είναι το **Node Object** με το αντίστοιχό του **Node Interface**, το οποίο θα δούμε παρακάτω σε λεπτομέρεια. Όμως εδώ σαν Node (κόμβος) νοείται όχι μόνον το κάθε στοιχείο (element) του XML, αλλά πέραν του Element Node, έχουμε το Attribute Node, το Text Node, το CDATA Node και άλλα περιφερειακής σημασίας. Η όλη δενδρική δομή, στην οποία εντάσσεται κατά DOM κάθε κείμενο XML, δίδεται το Σχήμα 6.1. Όλα τα μη μαυρισμένα τετράγωνα είναι **Node Objects**.

Σημαντικό χαρακτηριστικό του DOM είναι ότι η διαδικασία του parsing γίνεται δια μίας και χωρίς δική μας επέμβαση κατά την εξέλιξή της. Σύμφωνα με το Σχήμα 6.2, ο parser μετατρέπει το κείμενο XML στο αντίστοιχο δένδρο (κατά το Σχήμα 6.1), το οποίο διατίθεται στο προγραμματιστικό μας περιβάλλον.



Σχήμα 6. 1. Η ιεραρχία κατά το DOM



Σχήμα 6.2. Γενική αρχή DOM



Ερχόμαστε τώρα στα μαυρισμένα τετράγωνα του Σχήματος 6.1. Ο **NamedNodeMap** περιέχει τα attributes. Είναι μία μη διατεταγμένη συλλογή (collection), καθόσον κάθε attribute ανευρίσκεται με το όνομά του, χωρίς να ενδιαφέρει η σειρά. Αντίθετα η **NodeList** αποτελούμενη από Text Node, άλλα Element Nodes, κλπ. είναι μία διατεταγμένη λίστα. Συνεπώς, σύμφωνα με όσα γνωρίζουμε, ένα τυπικό XML element αποτελείται αφ'ενός από έναν NamedNodeMap (εντός του οποίου περικλείονται όλα τα attributes, σαν Attribute Nodes) και αφ'εταίρου από μία NodeList (η οποία περιλαμβάνει xyz Nodes, όπου xyz τα ενδεχόμενα Text, CDATA, κλπ, και εν τέλει κάποια (child) Element). Τούτο ακριβώς θέλει να δείξει το Σχήμα 6.1. Για παράδειγμα παρατηρούμε το **nodes\_example.xml**.

```
<root>
<level1A a1="value_a1" a2="value_a2">This is level 1, Item A
  <level2>This is level 2, Item A_21</level2>
  <level2>This is level 2, Item A_22</level2>
  <level2></level2>
</level1A>

<level1B b1="value_b1" b2="value_b2" ></level1B>

<level1C>This is level 1, Item C
  <level2 >This is level 2, Item C_21
    <level3>This is level 3, Item C_21_31</level3>
    <level3>This is level 3, Item C_21_32</level3>
  </level2>
  <level2 c1="value_c1" c2="value_c2">This is level 2, Item C_22 </level2>

</level1C>

</root>
```

### Παράδειγμα 6.1. Το κείμενο nodes\_example.xml.

Τώρα θα χρησιμοποιήσουμε το MSXML (Microsoft XML) DOM του IE5. Δημιουργούμε την παρακάτω web σελίδα με ενσωματωμένη JavaScript (σε ίδιο directory με το παραπάνω κείμενο).

```
<HTML>
<HEAD><TITLE>DOM Demo</TITLE>

<SCRIPT language="Javascript">

var objDOM;
objDOM=new ActiveXObject("MSXML.DOMDocument");
objDOM.async=false;
objDOM.load("nodes_example.xml");
```

```

// begin of code for navigating within the XML document
var objCurrNode;           //points to current node
var objNamedNodeMap;      //points to collection of all attributes

// an initial output of the whole XML document...
alert(objDOM.xml);

objCurrNode= objDOM.documentElement;
alert(objCurrNode.nodeName);//the nodeName property of the objCurrNode
alert(objCurrNode.nodeValue);

objCurrNode= objDOM.documentElement.firstChild;
alert(objCurrNode.nodeName);
alert(objCurrNode.nodeValue);
alert(objCurrNode.getAttribute("a2"));//method of objCurrNode

objNamedNodeMap=objCurrNode.attributes;
    //we obtain the collection of all attributes
alert(objNamedNodeMap.item(0).name);
alert(objNamedNodeMap.item(0).value);
alert(objNamedNodeMap.item(1).name);
alert(objNamedNodeMap.item(1).value);

objCurrNode= objDOM.documentElement.firstChild.firstChild;
alert(objCurrNode.nodeName);
alert(objCurrNode.nodeValue);

objCurrNode= objDOM.documentElement.firstChild.firstChild.nextSibling;
alert(objCurrNode.nodeName);
alert(objCurrNode.nodeValue);

objCurrNode= objDOM.documentElement.lastChild;
alert(objCurrNode.nodeName);
alert(objCurrNode.nodeValue);

objCurrNode= objDOM.documentElement.lastChild.firstChild;
alert(objCurrNode.nodeName);
alert(objCurrNode.nodeValue);

alert("End of the xml document tour.. ...xml document modification follows..");

</SCRIPT>

</HEAD>

<SCRIPT language="Javascript">

```

```

// here some examples for modifying the given XML document

var objNewNode; //will refer to new tag
var objNewText; //will refer to new text

//use of objDOM methods for constructing the new ....
objNewNode = objDOM.createElement("NEW_level2");
//and assign value to the ext property of ...
objNewNode.text="welcome to our NEW_level2"

//, hang it to the existing tree (where?)
objDOM.documentElement.lastChild.appendChild(objNewNode);

//and again display everything in its new form
alert(objDOM.xml);

alert("End of the xml modifications.....");

</SCRIPT>

<BODY>
<P>.....bye !!!!.....</P>
</BODY>
</HTML>

```

### Κώδικας 6.1. Περιήγηση μέσα στο nodes\_example.xml με JavaScript

Σημειώνεται ότι η αντικατάσταση του “<” με τον ειδικό χαρακτήρα &lt των tags έγινε προκειμένου να εμφανίζονται τα tags στον browser ως κείμενο και επομένως ολόκληρο το XML κείμενο. Διαφορετικά ότι είναι μεταξύ των “<” “>” λαμβάνεται ως HTML tag από τον browser και δεν τυπώνεται. Η αντικατάσταση μόνο του “<” αρκεί προφανώς.

Το objDOM αναφέρεται σε ένα MSXML.DOMDocument, που δημιουργείται σαν ActiveXObject (τεχνολογία Microsoft). Μέσα σε αυτό, με την μέθοδό του load(), φορτώνουμε το XML. Πρώτα με το alert(objDOM.xml) δείχνουμε όλο το XML κείμενο.

Τώρα θα περιδιαβούμε μέσα σ’ αυτό με τις μεθόδους των αντικειμένων κατά DOM και την γνωστή μας (από τις σελίδες ASP) JavaScript. Η μέθοδος alert() θα μας παρουσιάζει σε Ok MessageBox το τι θέλουμε να δούμε, ουσιαστικά την θέση μας μέσα στην δενδρική μορφή του XML (η οποία όμως τώρα, κατά DOM, θεωρεί σαν κόμβους ξεχωριστά το tag, attributes, text, κλπ κατά την ιεραρχία κατά DOM). Το objCurrNode σαν μεταβλητή αναφέρεται διαδοχικά σε διαφορετικούς κόμβους (nodes) μέσω των ιδιοτήτων firstChild, lastChild, previousSibling, nextSibling, πού έχει (όχι πάντα - ανάλογα με την θέση του) ένας κόμβος. Όλες αυτές οι ιδιότητες επιστρέφουν άλλους κόμβους. Όταν πέσουμε σε attributes, χρησιμοποιούμε το NamedNodeMap interface. Αυτό αντιπροσωπεύει μία μη διατεταγμένη συλλογή (collection) κόμβων, δηλ. εδώ τα (ίσως) πλέον του ενός, αλλά πάντα ονοματισμένα attributes. Μπορούμε να αναφερθούμε σε κάθε attribute ξεχωριστά με το item (index), όπου το index αρχίζει από το 0 (0 το πρώτο attribute, 1 το δεύτερο, κοκ) . Τα ίδια ισχύουν και για το NodeList interface, αλλά εδώ έχουμε διατεταγμένη συλλογή. Τέλος τα nodeName,

nodeValue είναι οι πιο χρήσιμες ιδιότητες του εκάστοτε κόμβου. Ανάλογα με τον τύπο του κόμβου, επιστρέφεται και η αντιστοιχούσα τιμή κατά τον πίνακα :

NodeType	nodeName	nodeValue
Element Node	Tag name	NULL
Attribute Node	Name of attribute	Value of attribute
Text Node	#text	Content of the text node
CDATA Node	#cdata-section	Content of the CDATA section

**Πίνακας 6.1.** Όνομα και τιμή για κάθε τύπο κόμβου

Για να μετατρέψουμε το κείμενο XML μέσω του DOM, υπάρχουν οι μέθοδοι `removeChild(oldChild)`, `appendChild(newChild)`, `replace(newChild,oldChild)`, `insertBefore(newChild,refChild)`, με αναφορές στον τρέχοντα (`refChild`), παλαιό και νέο κόμβο. Με την `cloneNode(deep)` κλωνοποιούμε τον κόμβο, είτε μόνον του (με `deep=false`), είτε αυτόν και όλο το υποδένδρο που κρέμεται από κάτω του (με `deep=true`). Για να δημιουργήσουμε ένα κόμβο, άσχετο και ασύνδετο κατ' αρχήν με το κείμενο XML, χρησιμοποιούμε την μέθοδο `createElement(tagName)` ενός "document". Ανάλογες είναι και οι `createTextNode(data)`, `createAttribute(name)`, `createCDATASection(data)`. Ότι δημιουργήσουμε με αυτό τον τρόπο μπορεί μετά να κρεμασθεί στο δένδρο, όπως εξηγήθηκε.

Σημειώνεται ότι το παραπάνω παράδειγμα αποτελεί client side scripting (και όχι server side scripting όπως είχαμε με τις ενεργές σελίδες ASP). Ο browser του client ανοίγει μία σελίδα html στην οποία παρεμβάλλεται κώδικας μίας scripting γλώσσας. Τέτοιες υπάρχουν πολλές: JavaScript, VBScript, PerlScript, Python. Η πρώτη έρχεται μαζί με όλους τους browsers, η δεύτερη είναι ενσωματωμένη μόνον στον Explorer.

Σώστε τώρα την παραπάνω web σελίδα την αναφερόμενη στο σχετικό XML. Φέρτε την με τον browser και δείτε μέσω του IE5 πώς ο κώδικας μας περιάγει μέσα στο XML κείμενο κάνοντας αλλαγές σε αυτό.

Τα παραπάνω ήταν μία γρήγορη εισαγωγή με την βοήθεια της JavaScript. Θα δούμε τώρα πιο συστηματικά το DOM (Level2) API που δίδει η Java. Το API (Application Programming Interface) είναι ένα σύνολο classes τα οποία διατίθενται στην εφαρμογή που γράφουμε, προκειμένου αυτή να χρησιμοποιήσει τις δυνατότητες που δίδει (στην συγκεκριμένη περίπτωση) το DOM. Όταν λοιπόν κάποιος ισχυρίζεται ότι μας διαθέτει προς χρήσιν ένα DOM Java API θα πρέπει, μεταξύ άλλων, να έχει πραγματοποιήσει σε Java, την class Node, η οποία θα πρέπει να μας επιτρέψει να κάνουμε ορισμένα πράγματα με κάθε αντικείμενο του τύπου (της class) Node. Διατυπώνουμε τούτο πιο αυστηρά: Ένα αντικείμενο μίας class στον ορισμό της οποίας αναφέρεται `...implements Node`, διαθέτει οπωσδήποτε μία σειρά από μεθόδους (methods), όλες σχετιζόμενες με όσα ο χρήστης (η εφαρμογή μας) μπορεί να κάνει με κάθε DOM node.

Έχομε λοιπόν τις ακόλουθες μεθόδους

```

public interface Node {
    public String getNodeName();
    public String getNodeValue() throws DOMException;
    public void setNodeValue(String nodeValue) throws DOMException;
    public short getNodeType();
    public Node getParentNode();
    public NodeList getChildNodes();
    public Node getFirstChild();
    public Node getLastChild();
    public Node getPreviousSibling();
    public Node getNextSibling();
    public NamedNodeMap getAttributes();
    public Document getOwnerDocument();
    public Node insertBefore(Node newChild, Node refChild) throws DOMException;
    public Node replaceChild(Node newChild, Node oldChild) throws DOMException;
    public Node removeChild(Node oldChild) throws DOMException;
    public Node appendChild(Node newChild) throws DOMException;
    public boolean hasChildNodes();
    public Node cloneNode(boolean deep);
    public void normalize();
    public boolean isSupported(String feature, String version);
    public String getNamespaceURI();
    public String getPrefix();
    public void setPrefix(String prefix) throws DOMException;
    public String getLocalName();
    public boolean hasAttributes();

    // Node Type Constants
    public static final short ELEMENT_NODE;
    public static final short ATTRIBUTE_NODE;
    public static final short TEXT_NODE;
    public static final short CDATA_SECTION_NODE;
    public static final short ENTITY_REFERENCE_NODE;
    public static final short ENTITY_NODE;
    public static final short PROCESSING_INSTRUCTION_NODE;
    public static final short COMMENT_NODE;
    public static final short DOCUMENT_NODE;
    public static final short DOCUMENT_TYPE_NODE;
    public static final short DOCUMENT_FRAGMENT_NODE;
    public static final short NOTATION_NODE;
}

```

### Σχήμα 6.3. DOM (Level 2) API της Java - Node interface

Στην κάθε (πάντα public) μέθοδο του παραπάνω Node interface, φαίνεται καθαρά τι παράμετρο ζητά κάθε μέθοδος και τι επιστρέφει και τα περισσότερα είναι προφανή. Έστω `nodeobj` ένα αντικείμενο μίας class που πραγματοποιεί (implements) το Node interface. Τότε η `nodeobj.getNodeName()` επιστρέφει το όνομα του `nodeobj` σαν String. Το τι σημαίνει 'όνομα' καθορίζεται κατά περίπτωση σύμφωνα με τον Πίνακα 1 ανάλογα με τον τύπο του `nodeobj`. Ο τύπος λαμβάνεται με `nodeobj.getNodeType()`, οπότε το short που επιστρέφεται θα είναι μία από τις παραπάνω οριζόμενες σταθερές. Σύμφωνα με τον Πίνακα 1 εκτελείται στην αντίθετη κατεύθυνση και η `nodeobj.setNodeValue(nodeValue)`. Την περιδιάβασή μας την επιτυγχάνουμε, όπως και στο προηγούμενο παράδειγμα της JavaScript, με `nodeobj.getFirstChild()` και τα παρεμφερή, που όλα επιστρέφουν άλλο αντικείμενο της class Node. Ανάλογα ισχύουν για παρεμβολή (insert), αντικατάσταση (replace), εξαφάνιση

(remove), προσθήκη (append), κλωνοποίηση (clone) κόμβων. Εδώ χρειάζονται δεδομένα εισόδου της class Node κατά περίπτωση, μας επιστρέφεται δε πάλι ο κατά περίπτωση κόμβος. Η πεμπτουσία του DOM φαίνεται με τις `nodeobj.getChildNodes()` και `nodeobj.getAttributes()`, πού μας επιστρέφουν αντικείμενο της class `NodeList` και `NamedNodeMap` αντίστοιχα, πάντα κατά το Σχήμα 1.

Οι class `NodeList` και class `NamedNodeMap` είναι βεβαίως και αυτές interface του DOM, οριζόμενες ως εξής

```
public interface NodeList {
    public Node item(int index);
    public int getLength();
}

public interface NamedNodeMap {
    public Node getNamedItem(String name);
    public Node setNamedItem(Node arg) throws DOMException;
    public Node removeNamedItem(String name) throws DOMException;
    public Node item(int index);
    public int getLength();
    public Node getNamedItemNS(String namespaceURI, String localName);
    public Node setNamedItemNS(Node arg) throws DOMException;
    public Node removeNamedItemNS(String namespaceURI, String localName)
        throws DOMException;
}
```

#### Σχήμα 6.4. DOM (Level 2) API της Java - `NodeList` και `NamedNodeMap` interface

Φαίνεται καθαρά από τις διατιθέμενες μεθόδους η διαφορετική φύση των παραπάνω, δηλ. η διατεταγμένη λίστα `NodeList`, έναντι της μη διατεταγμένης συλλογής `NamedNodeMap`.

Όπως γνωρίζουμε κόμβος (node) είναι και κάθε attribute. Όπως βλέπομε παρακάτω το `Attr` extends `Node`, δηλαδή είναι ότι και το interface `Node`, αλλά με περαιτέρω ιδιότητες και μεθόδους. Αυτές οι πρόσθετες μέθοδοι δίδονται στο `Attr` interface.

```
public interface Attr extends Node {
    public String getName();
    public boolean getSpecified();
    public String getValue();
    public void setValue(String value) throws DOMException;
    public Element getOwnerElement();
}
```

#### Σχήμα 6.5. DOM (Level 2) API του Java - `Attr` interface

Το ίδιο το `Element Node`, σαν επέκταση του `Node` έχει ιδιαίτερο ενδιαφέρον, καθόσον περιλαμβάνει, μεταξύ άλλων, και την διαχείριση των attributes (εισαγωγή νέων, διαγραφή παλαιών, εξαγωγή ενός για επεξεργασία, κλπ). Βεβαίως το `Element` είναι το μόνον `Node` πού έχει attributes (ενσωματωμένα σε `NamedNodeMap`).

```
public interface Element extends Node {
    public String getTagName();
}
```

```

public String getAttribute(String Name);
public void setAttribute (String name, String value) throws DOMException;
public void removeAttribute (String name) throws DOMException;
public Attr getAttributeNode(String Name); //returns a whole Attr object
public Attr setAttributeNode(Attr newAttr) throws DOMException; //sets a new Attr & returns the same
public Attr removeAttributeNode(Attr oldAttr) throws DOMException;
public NodeList getElementsByTagName(String name);
//missing some, name space related
public boolean has Attribute(String name);
//missing one, name space related
}

```

### Σχήμα 6.6. DOM (Level 2) API του Java - Element interface

Ένα πλήρες κείμενο XML αποτελεί αντικείμενο της class Document. Όπως βλέπομε παρακάτω η Document extends Node, δηλαδή είναι ότι και η class Node, αλλά με περαιτέρω ιδιότητες και μεθόδους. Αυτές οι πρόσθετες μέθοδοι δίδονται στο Document interface.

```

public interface Document extends Node {
    public DocumentType getDocType();
    public DOMImplementation getImplementation();
    public Element getDocumentElement();
    public Element createElement(String tagName) throws DOMException;
    public DocumentFragment createDocumentFragment();
    public Text createTextNode(String data);
    public Comment createComment(String data);
    public CDATASection createCDATASection (String data) throws DOMException;
    public ProcessingInstruction
        createProcessingInstruction (String target, String data) throws DOMException;
    public Attr createAttribute(String name) throws DOMException;
    public EntityReference createEntityReference (String name) throws DOMException;
    public NodeList getElementsByTagName(String tagname);
    public Node importNode(Node importedNode, boolean deep) throws DOMException;
    public Element createElementNS(String namespaceURI, String qualifiedName)
        throws DOMException;
    public Attr ceateAttributeNS(String namespaceURI, String qualifiedName)
        throws DOMException;
    public NodeList
        getElementsByTagNameNS(String namespaceURI, String qualifiedName)
        throws DOMException;
    public Element getElementById(String elementId);
}

```

### Σχήμα 6.7. DOM (Level 2) API του Java - Document interface

Η getDocumentElement() επιστρέφει το root element. Τούτο είναι της class Element, η οποία είναι εξειδίκευση της Node (Element extends Node, πράγματι έχουμε κατά το Σχήμα 1 και 'Element Node'). Κάθε νέο Element (οποιασδήποτε class) το οποίο θέλουμε να

κατασκευάσαμε και να εντάξουμε στο υπάρχον μοντέλο XML κατά DOM, ανήκει αναγκαστικά σε κάποιο document. Γι' αυτό υπάρχουν τα διάφορα createXYZ (createNS λαμβάνοντας υπ' όψιν το namespace) που επιστρέφουν αντικείμενο τύπου XYZ. Το κάθε τέτοιο xyz είναι (εξειδίκευση του) node και μπορεί να αποτελέσει κατόπιν όρισμα εισόδου στις σχετικές μεθόδους του Node interface που δώσαμε παραπάνω.

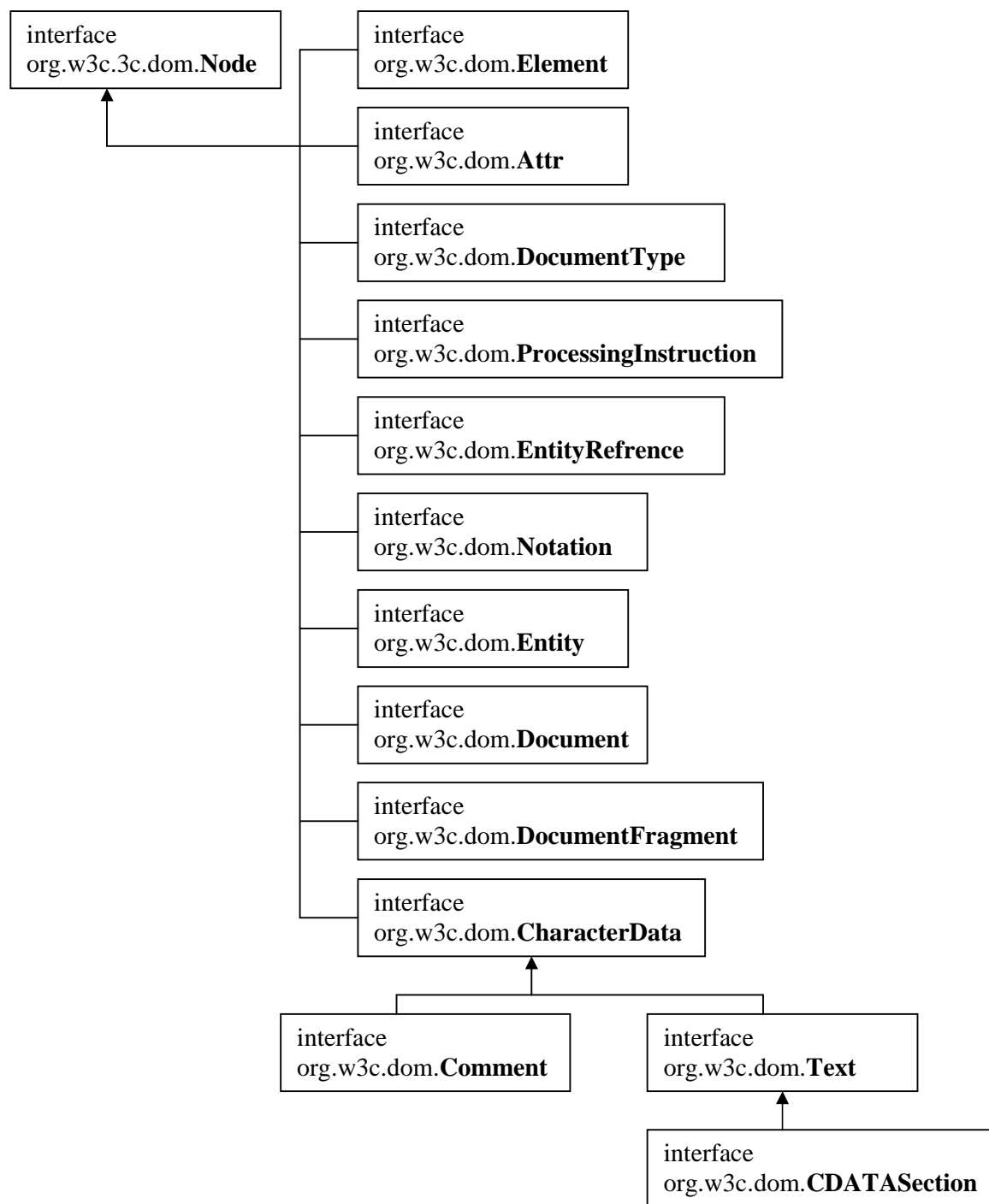
Σημειώνουμε επίσης ότι υπάρχουν και μία σειρά ορισμών

```
public interface Xyz extends Node { ... }
```

όπου το Xyz είναι Attr, CDATASection, CharacterData, Document (εδόθη ήδη), DocumentFragment, DocumentType, Element (σχολιάστηκε παραπάνω), Entity, EntityReference, Notation, ProcessingInstruction. Τα περισσότερα είναι γνωστά μας σαν περιπτώσεις (εξειδικεύσεις) του Node κατά DOM. Τέλος υπάρχουν και τα Comment και Text σαν εξειδικεύσεις της class CharacterData, ενώ το CDATASection είναι εξειδίκευση της Text.

Η πλήρης εικόνα των classes (interfaces) του DOM API και της ιεραρχίας των, δίδεται στο παρακάτω σχήμα. Παρατηρούμε ότι η ιεραρχία αυτή αντικατοπτρίζει την κληρονομικότητα της Java (ή του αντικειμενοστραφούς προγραμματισμού) και όχι την ιεραρχία ενός κειμένου XML. Το κάθε interface x προσθέτει μεθόδους επιπλέον αυτών που βρήκε στον πατέρα του y, δηλ. x extends y. Με την class x (κάθε interface είναι και class !) μπορούμε να κάνουμε περισσότερα πράγματα διότι η x είναι πιο εξειδικευμένη από την y. Χαρακτηριστική περίπτωση είναι η Document, η οποία στην έννοια του XML περιέχει ένα Node, αλλά εδώ σαν class (ή interface) είναι ιεραρχικά παιδί του Node. Για τούτο κληρονομεί όλες τις μεθόδους του και έχει και άλλες επιπλέον. Σε όλες τις περιπτώσεις δίδονται οι προς χρήσιν μέθοδοι υπό την μορφή των interfaces που φαίνονται στο Σχήμα 6.6. Οι classes ευρίσκονται στα packages org.w3c.dom.xyz.





**Σχήμα 6.8.** Ιεραρχία του DOM API

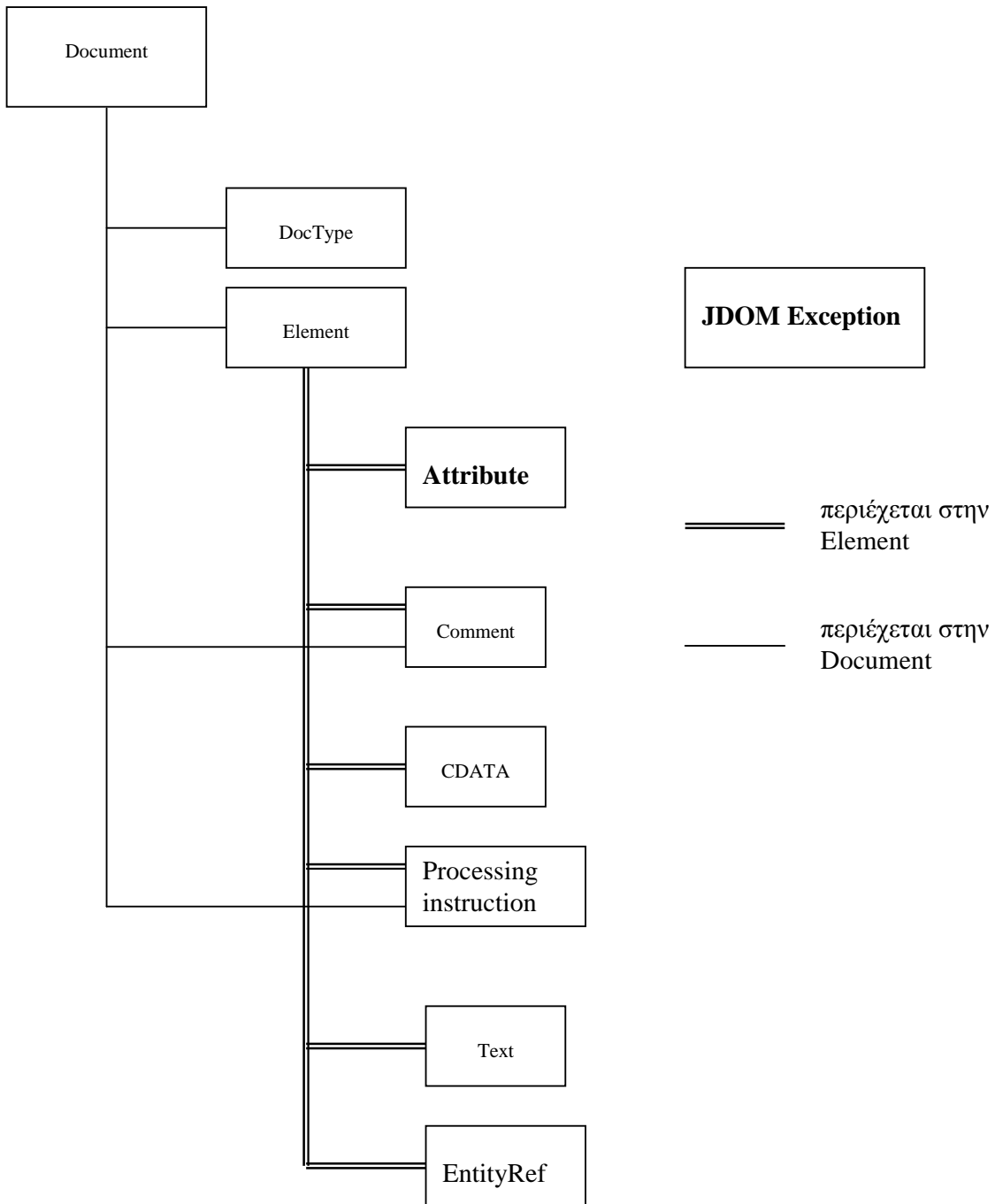
Δεν είναι παράξενο, ότι μελετώντας το Java API του DOM ανακαλύπτουμε όλη την θεωρία του XML. Για τούτο και δώσαμε παραπάνω μια κάπως λεπτομερέστερη περιγραφή χωρίς συγκεκριμένο παράδειγμα σχετικού προγράμματος σε Java.

## XML και JDOM (Java Document Object Model)

Το **Java Document Object Model (JDOM)** μας προσφέρει πρόσβαση σε ένα κείμενο XML μέσω μίας δενδρικής δομής (όπως και το DOM), αλλά εδώ αποκλειστικά μέσω ανοικτού Java API. Οι σχετικές classes σε UML μορφή δίδονται στο Σχήμα 6.9. **Το JDOM είναι διαφορετικό του Java API του DOM, που παρουσιάσαμε στο προηγούμενο κεφάλαιο.** Με τις μεθόδους του JDOM, που θα παρουσιασθούν παρακάτω, επεξεργαζόμαστε την δομή που δημιουργείται σαν αναπαράσταση του XML. Ένα αντικείμενο Document είναι η αναπαράσταση του όλου κειμένου XML και συγχρόνως περιέχει τα άλλα JDOM αντικείμενα που αντιστοιχούνται σύμφωνα με τις ονομασίες τους απ' ευθείας στις αντίστοιχες του XML. Το (XML) Element του JDOM δεν περιέχει τα μέρη του ομαδοποιημένα σε NodeList και NamedNodeMap όπως στο DOM. Επιπλέον μπορούμε να επηρεάσουμε (θέσουμε, αλλάξουμε, σβήσουμε) το περιεχόμενο ενός (XML) Element, χωρίς να μεταφερόμαστε και να δουλεύουμε σε κάθε σε επιμέρους 'κόμβους' (Attribute, Text, CDATA Node, κλπ ) ξεχωριστά. Για παράδειγμα η getAttributes() μέθοδος της Element class επιστρέφει μια (Java) List. Επιπλέον δεν υπάρχει εδώ TextNode. Το κείμενο (text) ενός Element απλά επιστρέφεται από την getText() μέθοδο του Element class. Κατά συνέπεια, αντικείμενα όπως Element, Attribute, ProcessingInstruction, Comment, κλπ μπορούν να δημιουργηθούν με το New. Τούτο χρησιμοποιείται για την κατασκευή ενός XML κειμένου εκ του μηδενός. Η ουσιαστική διαφορά έγκειται ότι ενώ το JDOM δεν νοείται έξω από την Java, οι δομές του DOM είναι ανεξάρτητες και πέραν από συγκεκριμένη γλώσσα προγραμματισμού. Άλλο αν εμείς παραπάνω τις αντιμετωπίσαμε με JavaScript, ή δώσαμε τα σχετικά interface σε περιβάλλον Java.

Το JDOM παρέχει τον τρόπο πρόσβασης στα στοιχεία του κειμένου XML, αλλά για την εισαγωγή / εξαγωγή του XML αναγκαστικά βασίζεται σε κάποιον parser (συγκεκριμένα τον SAX parser που θα δούμε παρακάτω) ή στην δενδρική δομή της ιεραρχίας του DOM (αν διατίθενται τέτοια).

Θα δούμε τώρα το μέρος της εισαγωγής ενός XML, στο παρακάτω παράδειγμα μέσω του SAX parser. Δημιουργούμε πρώτα έναν SAXBuilder. Αυτός χρησιμοποιώντας το SAX, μετατρέπει με την μέθοδό του build το κείμενο του αρχείου vehicles.xml σε ένα document, δηλ. στο αντικείμενο doc της γενικότερης class Document του Σχήματος 7. Τώρα, με το αντικείμενο jtp της class JDOMTreeProc, που εκτίθεται πιο κάτω, διερχόμαστε αναδρομικά όλα τα elements (στοιχεία) του doc.



**Σχήμα 6.9.** Τα αντικείμενα (object model) του JDOM με βάση την σχέση ‘περιέχεται’

```

import java.io.*;
import java.util.*;
import org.jdom.*;
import org.jdom.input.SAXBuilder;

public class XMLtoJDOMClasses {
public static void main(String args[]) throws FileNotFoundException,IOException,JDOMException
{
    System.out.println("hello from Vehicle classes demo" + "\n" + "\n");
    //Build the Document object doc instantiating the inputted
    //XML tree structure, via a JDOM tree (sxhna above)
    SAXBuilder builder = new SAXBuilder();
    Document doc = builder.build(new FileInputStream("vehicles.xml"));

    //call our only method of jtp (see processElement of JDOMTreeProc below).
    //This will process the root of the doc object,
    //(i.e. the root element of the XML) passed as an argument.
    //Method getRootElement of doc is found in the JDOM API
    JDOMTreeProc jtp = new JDOMTreeProc();
    jtp.processElement(doc.getRootElement());
}
}

```

### **Κώδικας 6.2.** Επίσκεψη με αναδρομή σε όλα τα elements μέσω JDOM (1/2)

Η JDOMTreeProc (βλ. επόμενο κώδικα) έχει την μέθοδο processElement, η οποία εξάγει το περιεχόμενο του κάθε στοιχείου σε μία list (mixedContent). Εδώ τοποθετούνται όλα τα αντικείμενα που αποτελούν το περιεχόμενο του τρέχοντος element, εκτός από τα attributes. Τα attributes δεν θεωρούνται περιεχόμενο, αλλά ιδιότητες του Element. Τα attributes εξάγονται μέσω μίας λίστας κατ' ευθείαν από το element (getAttributes()) για όλα, getAttribute(String name) για ένα προς ένα – αλλά τότε πρέπει να γνωρίζουμε εκ των προτέρων το όνομα). Επίσης το κείμενο μπορεί να εξαχθεί και αυτό απ' ευθείας από το Element με την getText(). Για αναφορά στα πραγματικά περιεχόμενα του Element, χρησιμοποιείται ένα γενικό αντικείμενο (object), το οποίο κατά περίπτωση γίνεται casted σαν Text, Comment, κλπ, Element (οπότε τώρα παιδί του τρέχοντος), βλ. Σχήμα 7. Τρέχοντας τις δύο παραπάνω Java classes, εμφανίζουμε τα Text και Attributes όλων των κόμβων. Προσέξτε ότι η processElement καλείται αναδρομικά. Τα στοιχεία της mixedContent είναι αντικείμενα με διάφορες ιδιότητες και χρησιμοποιούμε την getText για τα Text και Comment προκειμένου να τυπώσουμε το κείμενό τους σαν String. Το αντικείμενο CDATA τυπώνεται δύο φορές, μια σαν αντικείμενο Text (η CDATA είναι κατασκευασμένη σαν subclass της Text) και μία σαν CDATA. Το αντικείμενο ProcessingInstruction έχει κατά την θεωρία PITarget (η εφαρμογή στην οποίαν απευθύνεται η ακολουθούσα εντολή), δεδομένα (DATA, δηλ. η εκφώνηση της εντολής) και βεβαίως πατέρα στον οποίον ανήκει (Parent). Και τα τρία εξάγονται σαν κείμενα.

Όλα τα παραπάνω είναι πλέον κοινή Java και η διδόμενη παρακάτω μέθοδος επεξεργασίας μπορεί να χρησιμοποιηθεί γενικότερα επί οποιουδήποτε αρχικού XML.

```

import java.io.*;
import java.util.*;
import org.jdom.*;
import org.jdom.input.SAXBuilder;

public class JDOMTreeProc
{
public void processElement(Element el)
{
    // Print name and text of el
    System.out.println("\n-----");
    System.out.println("\nNEW ELEMENT named: "+ el.getName()+ ", with text:"+ el.getText());
    //Print attribute names and values:
    //(to do this first get all attributes via a list)
    List attributeList = el.getAttributes();
    printAttributes(attributeList);

    //mixedContent list will contains the contents of el
    // ---!!! Attributes are NOT considered contents of el ---!!
    List mixedContent = el.getContent();
    //Contents can be Text, Comment, ProcessingInstruction, ...
    //scan the content list, assigning first each of the list's
    //inhomogeneous elements to a general object o
    for (Iterator i = mixedContent.iterator(); i.hasNext();)
    {
        Object o = i.next();
        //if object o belongs to some particular class
        //(Text,etc, see JDOM classes),
        //we use casting and make o a pointer to an object
        //of this class, then place an appropriate call.
        if (o instanceof Text){processText((Text)o);};
        if (o instanceof Comment){processComment((Comment)o);};
        if (o instanceof ProcessingInstruction)
            {processProcessingInstruction((ProcessingInstruction)o);};
        if (o instanceof EntityRef){processEntityRef((EntityRef)o);};
        if (o instanceof CDATA){processCDATA((CDATA)o);};
        if (o instanceof Element)
            {//here a recursive call to this same method !!!!
                processElement((Element)o);}
    }
}

public void printAttributes(List attributeList)
{
    System.out.println("ATTRIBUTES (name/value pairs):");
    for (Iterator i = attributeList.iterator(); i.hasNext();)
    {
        Attribute att = (Attribute)i.next();
        System.out.print(att.getName()+ "="+ att.getValue() + " ");

    }
}

public void processText(Text text)
{
    //we output the (possibly void) element text once more
    System.out.println("\nANOTHER output of TEXT: " + text);
}

public void processProcessingInstruction(ProcessingInstruction pi)

```



```

import java.io.*;
import java.util.*;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.Attribute;
import org.jdom.output.*;
import org.jdom.Comment;

public class CreateXML
{

    public static void main(String args[]) throws FileNotFoundException,IOException
    {
        System.out.println("Creation of new XML");
        XMLOutputter outputter = new XMLOutputter(" ",true);
        FileOutputStream output = new FileOutputStream("newVehicle.xml");

        //new Document object, with element lory3 as root
        Element element0 = new Element("lory3");
        Document doc = new Document(element0);

        //add a comment
        Comment com = new Comment("This is a new entry to vehicles");
        doc.addContent(com);

        //new Element object to be used below
        Element element1 = new Element("tender");

        //One four_line statement!.... Creates
        //child, grandchild, grandchild's attribute at once!
        //setAttribute is method of Element
        //AND returns the (same) element object - therefore we continue
        //with setText, another method of Element !!!!!
        element0.addContent(new Element("traction")
            .addContent(new Element("tractionbody")
                .setAttribute(new Attribute("type", "extra heavy"))
                .setText("text for tractionbody")));

        //element1 will become child of element
        element0.addContent(element1);

        element1.setText("some text, ").addContent("... more text");

        //Method output will create an xml document as specified on top
        outputter.output(doc,output);

        //the JDOM objects (here an Element object) are usual java objects,
        //to demonstrate this, we apply the most common object method toString
        System.out.println(element0.toString());

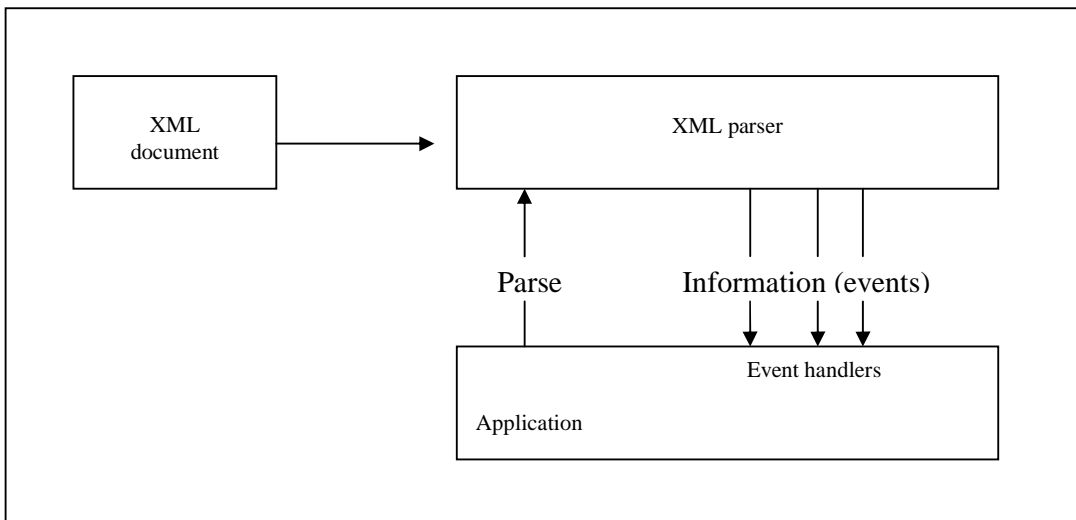
        System.exit(0);
    }
}

```

**Κώδικας 6.4.** Σειριακή εξαγωγή δομής κατά JDOM, σε αρχείο .xml

## XML και SAX (Simple API for XML)

Στη βάση όλων των παραπάνω ευρίσκεται το SAX (Simple API for XML), προσφερόμενο από τον SAX parser, τον οποίο μπορούμε να φανταστούμε σαν έναν σειριακό αναγνώστη (SAX reader) του κειμένου XML, χαρακτήρα προς χαρακτήρα. Ο SAX parser αναγνωρίζοντας τα διάφορα μέρη του XML αρχείου, δημιουργεί αντίστοιχα συμβάντα (events), στα οποία το δικό μας πρόγραμμα οφείλει να ανταποκριθεί σύμφωνα με τις επιθυμίες μας. Ο SAX parser διαβάζει την ταινία (συρμό χαρακτήρων) του XML κειμένου και ανιχνεύει και σταματά σε συγκεκριμένα σημεία. Σε κάθε σταμάτημα δημιουργείται ένα συμβάν (event) προς τον δικό μας event handler. Μόλις ο event handler αντιμετωπίσει το event, συνεχίζει ο parser μέχρι το επόμενο σημείο. Στο Σχήμα 4. φαίνεται η σειριακή είσοδος του κειμένου XML στον SAX parser. Αυτός ελέγχεται από την εφαρμογή στην οποία δηλώνει πίσω (callbacks) συμβάντα, δηλαδή την ανίχνευση των επί μέρους μερών του υπό ανάγνωση XML .



Σχήμα 6.10. Γενική αρχή SAX parser

Ο ContentHandler (ενσωματωμένος σε μορφή interface μέσα στον SAX parser) είναι ένας event handler κατά τα γνωστά της Java, ο οποίος δημιουργεί κλήσεις πίσω στην εφαρμογή κάθε φορά που ένα αξιοπρόσεκτο γεγονός (parsing event) διαπιστώνεται κατά την σειριακή ανάγνωση του κειμένου XML. Ένα interface είναι ο ορισμός μεθόδων (βλ. π.χ. startDocument, StartElement, κλπ). Στην περίπτωση μας η πλειοψηφία αυτών των μεθόδων θα καλείται όταν διαπιστώνονται οι ομώνυμες περιπτώσεις (αρχή του κειμένου, στοιχείου XML αντίστοιχα). Ο XMLEventsPresentor, παρακάτω σαν δική μας πραγματοποίηση του ContentHandler interface, οφείλει να προγραμματίσει όλες τις μεθόδους αυτές, έστω και με τετριμμένο τρόπο (άμεση επιστροφή χωρίς καμία ενέργεια). Σαν τον ContentHandler, υπάρχουν στον SAX και άλλα τρία παρόμοια interfaces ErrorHandler, DTDHandler, EntityResolver, με αντίστοιχη φιλοσοφία χρήσης, τα οποία όμως δεν ενδιαφέρουν εδώ και αφήνονται απραγματοποίητα (δηλ. δεν προχωράμε σε αντίστοιχη implementation). Δημιουργούμε πρώτα ένα αντικείμενο SAXParser και μετά μέσω αυτού ένα αντικείμενο XMLReader, το οποίο και θα διαβάσει το XML κείμενό μας. Στον XMLReader αντικείμενο προσκολλάται και το ContentHandler αντικείμενο που δημιουργήσαμε. Τέλος καλούμε την μέθοδο parse, η οποία αυτόματα δημιουργεί όλα τα callbacks, δηλαδή τις κλήσεις που



γίνονται κάθε φορά που στην διαδικασία της σειριακής ανάγνωσης ο parser ανακαλύπτει το διαπιστώνει το αντίστοιχο συμβάν. Ανάλογα με το τι βρίσκει, μας το διαθέτει μέσω αυτής της κλήσης και διάφορα σχετικά αντικείμενα που ενδιαφέρουν, π.χ. στην περίπτωση του element όλη την πληροφορία που σχετίζεται με το όνομά του. Στο συμβάν εμφάνισης νέου element, μας δίδεται η ευκαιρία να διαβάσουμε και τα attributes. Μέσω του interface Attributes (σαν Vector της Java) δίδουμε στο παράδειγμα ένα εκ των προτέρων γνωστό μας όνομα attribute (το colour), μας επιστρέφεται η θέση που βρίσκεται (μηδενική, πρώτη, δεύτερη κλπ μέσα στο element) και κατόπιν χρησιμοποιώντας αυτήν διαβάζουμε την τιμή.

Σημειωτέον, ότι χωρίς επιπλέον δική μας προσπάθεια δεν υπάρχει ουσιώδης πληροφορία για το πού ευρίσκεται μέσα στο XML η διαδικασία του parsing, π.χ. σε ποίο βάθος του δένδρου. Κρατώντας στην εφαρμογή μας μία προϊστορία δυνατόν να καταγράφουμε δομημένη την πληροφορία για το τι συνέβη / διαβάστηκε σε κάθε στιγμή. Δεν υπάρχει εν τούτοις κανένας τρόπος να γνωρίζουμε τι έπεται, δεδομένου ότι το δικό μας μέρος (της εφαρμογής) προχωρά παράλληλα με τον SAXParser και XMLReader, ο οποίος δεν έχει ακόμη διαβάσει πιο πέρα. Αν πάλι επιθυμούμε γνώση του σημείου, όπου ευρίσκεται ανά πάσα στιγμή ο parser χρησιμοποιούμε την class Locator. Δημιουργούμε το αντικείμενό της lc και το δηλώνουμε στον ContentHandler χρησιμοποιώντας την μέθοδο setDocumentLocator με την μοναδική εντολή this.lc = locator; Από όλες τις μεθόδους της δοκιμάζουμε παρακάτω την getLineNumber, που μας δίδει την θέση (αριθμό γραμμής) μέσα στο αρχείο, όπου διαπιστώθηκε από τον SAXParser το συγκεκριμένο συμβάν.

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import javax.xml.parsers.*;

//XMLEventsPresentor will be defined as an implementation
//of the ContentHandler interface. The ContentHandler interface
//contains the definition of a number of methods,
//which are automatically called, when the corresponding
//parsing event occurs. In our XMLEventsPresentor implementation
//we have to implement each and every one of these methods
//(even in a trivial way).
public class XMLEventsPresentor implements ContentHandler
{
    private Locator lc;           //will be registered below

    public static void main(String[] args) throws Exception
    {
        System.out.println("Here we go..");
        XMLEventsPresentor xmlep = new XMLEventsPresentor();
        xmlep.go();
    }

    public void go() throws Exception
    {
        SAXParserFactory spf = SAXParserFactory.newInstance();
        //We create first a parser object and then,
        //through a method of it, a reader object
        SAXParser sp = spf.newSAXParser();
        XMLReader sr = sp.getXMLReader();
        //We have to declare the ContentHandler to the reader
        //and then we can issue the 'parse' command to
        //(invoke the parse method of) the reader
        sr.setContentHandler(this);
        sr.parse("file:///D:/George/JavaClasses/Code/vehicles.xml");
    }
}
```

```

}

//Implementations of ALL ContentHandler interface methods:

public void setDocumentLocator(Locator locator)
{
    //Register the locator
    this.lc =locator ;
}

public void startDocument() throws SAXException
{
    System.out.println("Starting with the XML document...");
}

public void endDocument() throws SAXException
{
    System.out.println("..... XML document finished");
}

public void startPrefixMapping(String prefix, String uri)
    throws SAXException
{}

public void endPrefixMapping(String prefix)
    throws SAXException
{}

public void startElement(String namespaceURI, String localName,
    String qName, Attributes atts)
    throws SAXException
{
    //demonstrate use of locator: show the line of this event
    System.out.println("START of Element " + qName
        + " at line no "+ lc.getLineNumber());

    //get the index no of the attribute named 'colour' and
    //use this to find the colour value (wherever applicable)
    System.out.println("Attribute colour has index no"
        + atts.getIndex("colour") + " and value "
        + atts.getValue(atts.getIndex("colour")));
}

public void endElement(String namespaceURI, String localName,
    String qName) throws SAXException
{System.out.println("END of Element "+ qName); }

public void characters(char ch[], int start, int length)
    throws SAXException {}

public void ignorableWhitespace(char ch[], int start, int length)
    throws SAXException {}

public void processingInstruction(String target, String data)
    throws SAXException {}

public void skippedEntity(String name) throws SAXException {}
}

```

**Κώδικας 6.5.** Το ContentHandler interface και η ‘πραγματοποίηση’ των μεθόδων του

Με τον SAX parser ευρισκόμεθα στο κατώτερο προγραμματιστικά περιβάλλον που μας δίδει πλήρη πρόσβαση στο XML. Η ένταξη των δικών μας επιθυμιών (εφαρμογή μας) είναι πιο επίπονη από ότι με το DOM ή JDOM, αλλά οι διαθέσιμες δυνατότητες και η ταχύτητα εδώ υπερτερούν. Δεν περιμένουμε να δημιουργηθεί και να μας προσφερθεί η πλήρης δομή του XML σύμφωνα με το DOM ή JDOM μοντέλο, αλλά ειδοποιούμεθα και μπορούμε να ανταποκριθούμε στα σχετικά συμβάντα on the fly, καθώς διαβάζεται το XML κείμενο και χωρίς να περιμένουμε το τέλος αυτής της ανάγνωσης.

## 7. XML και XSL

Με την **XSL (eXtensible Stylesheet Language)** μορφοποιούμε ένα κείμενο XML. Στη νέα του μορφή μπορεί πάλι να είναι XML (ή HTML), όπως το θέλει ένας άλλος αποδέκτης. Η μορφοποίηση, δηλαδή η μετατροπή ενός αρχείου .xml εισόδου κάτω από οδηγίες γραμμένες σε ένα κείμενο σε αρχείο .xsl, μπορεί να γίνει είτε μέσω του browser, είτε μέσω ενός XSLT (XSL Transformation) Processor. Η έξοδος είναι το μεταμορφωμένο αρχείο .xml ή .htm. Στην περίπτωση του IE5, υπάρχει ενσωματωμένος ο Microsoft XML (MSXML) parser, ο οποίος υποστηρίζει και μορφοποιήσεις σύμφωνα με τις περισσότερες προδιαγραφές της XSL. Θα περιγράψουμε παρακάτω πώς μπορεί ένας parser να κάνει μία τέτοια μορφοποίηση. Με τον IE5 μπορούμε να έχουμε μόνον έξοδο .htm, η οποία και εμφανίζεται. Με αυτό θα ασχοληθούμε πρώτα με βάση το παρακάτω αρχικό κείμενο XML. Το κείμενο Cars.xml πρέπει να είναι ευκατανόητο, με μοναδικό νέο στοιχείο την δεύτερη σειρά, όπου ο parser εντέλλεται να χρησιμοποιήσει το “stylesheet” CarPresenter.xsl. Άρα το παρακάτω XML κείμενο περιέχει με την (τονισμένη) αναφορά του στο stylesheet CarPresenter.xsl τις οδηγίες για το πώς θα πρέπει να εμφανισθεί από τον IE5.

```
<?xml version="1.0"?>

<!-- REFERENCE TO THE STYLESHEET: -->
<?xml-stylesheet href="CarPresenter.xsl" type="text/xsl"?>

<vehicles>
  <vehicle year="1996" make="Land Rover" model="Discovery">
    <mileage>36500</mileage>
    <color>black</color>
    <price>$22100</price>
  </vehicle>

  <vehicle year="1998" make="Land Rover" model="Discovery">
    <mileage>3550</mileage>
    <color>red</color>
    <price>$31995</price>
  </vehicle>

  <vehicle year="1996" make="Chevrolet" model="Suburban 2500">
    <mileage>49300</mileage>
    <color>green</color>
    <price>$25995</price>
  </vehicle>
</vehicles>
```

**Παράδειγμα 7.1.** Το κείμενο Cars.xml με την αναφορά του στο stylesheet CarPresenter.xsl.

Αν ‘κτυπήσουμε’ το Cars.xml με τον IE5, θα δούμε, αντί ενός κειμένου XML, έναν ωραίο πίνακα! Πώς έγινε αυτό;

Ας δούμε το CarPresenter.xsl (επόμενο listing), δηλ το κείμενο που θα δώσει τις οδηγίες στον IE5 για να δημιουργήσει το νέο HTML κείμενο, και το οποίο αναφέρεται στην δεύτερη

γραμμή του Cars.xml. Οι οδηγίες αυτές συνίστανται είτε σε νέα στοιχεία XML, οπότε γράφονται κατά τα ήδη γνωστά, είτε σαν μετατροπές που αναφέρονται σε στοιχεία του αρχικού XML, πράγμα που μας ενδιαφέρει τώρα πρωτίστως. Κάθε κείμενο .xsl είναι και αυτό γραμμένο συμβατά με την σύνταξη της XML και περιλαμβάνει ιδιαίτερες ετικέτες (tags με το πρόθεμα xsl:), οι οποίες και ορίζουν τις επιπλέον λειτουργικότητες. Κάθε αρχείο .xsl, σαν συντακτικά ισχύον κείμενο XML (valid XML), μπορεί να διαβασθεί αυτόνομα κατά τα γνωστά (π.χ. μέσω του browser). Η πρώτη γραμμή δηλώνει την έκδοση της χρησιμοποιούμενης XML (όπως πριν) και η δεύτερη χρειάζεται για δήλωση των χρησιμοποιούμενων στοιχείων του XSL. Συναντάμε εδώ την γνωστή μας έννοια του namespace: Η δεύτερη γραμμή δηλώνει, ότι οτιδήποτε έχει το πρόθεμα xsl: ('xmlns:xsl=...', δηλ. 'το xml namespace είναι το ...') αποκτά την σημασία που ορίζεται στο "http://www.w7.org/TR/WD-xsl". Αυτό μπορεί να φαίνεται σαν διεύθυνση δικτυακού τόπου, αλλά ουδέποτε θα πάμε σε αυτόν. Απλά σημαίνει ότι αυτός ο οποίος θα εμφανίσει το κείμενο (εδώ ο IE5) πρέπει να έχει προγραμματίσει όλα όσα έχουν συμφωνηθεί 'μέσα στον' τόπο αυτό, δηλ. ουσιαστικά στο αντίστοιχο πρότυπο. Προσέξτε επίσης ότι το xmlns:xsl δεν είναι τίποτε άλλο παρά attribute του στοιχείου xsl:stylesheet. Το xsl:stylesheet είναι και το root element του κειμένου CarPresentor.xsl, καθόσον ανοίγει στην δεύτερη σειρά και κλείνει στην τελευταία.

Ο σκοπός εδώ είναι να εμφανισθούν τα δεδομένα του XML σε μορφή πίνακα. Η εμφάνιση του πίνακα προετοιμάζεται με τα αρχικά στις αρχικές γραμμές του κειμένου XSL, ανεξάρτητα του Cars.xml. Το ενδιαφέρον είναι μόνον το πώς επιλέγουμε και γράφουμε μέσα στον πίνακα πραγματική πληροφορία 'τραβηγμένη' μέσα από το Cars.xml. Όλο το κείμενο XSL περικλείεται μεταξύ <xsl:stylesheet ...> και </xsl:stylesheet>. Μέσα σε αυτό ορίζονται templates (εκμαγεία). Το αρχικό φαίνεται μεταξύ <xsl:template match="/"> και το </xsl:template> της προτελευταίας σειράς. Κάθε εκμαγείο έχει το attribute match, με το οποίο δηλώνεται επί ποίου υποδένδρου του κειμένου XML θα εφαρμοσθεί το template αυτό. Μπορούμε αυτό να το διατυπώσουμε και αλλιώς: εφόσον υπάρχει κάποιο υποδένδρο στο αρχικό XML που να ταιριάζει με την τιμή του match (εδώ το root, δηλ. θα ταιριάζει όλο το Cars.xml), θα εφαρμοσθεί το template αυτό και μέσα σε αυτό το υποδένδρο. Κάθε αναφορά στο αρχικό XML περιορίζεται στο υποδένδρο που ορίστηκε με το match. Τούτο δεν σημαίνει πολλά εδώ που πρόκειται για το '/', αλλά παρακάτω θα δούμε templates καλούμενα μέσα από templates (ενθυλάκωση - nesting), οπότε κάθε template συνεπάγεται και ένα (σύνολο από) υποδένδρο(α), το(α) λεγόμενα context node(s). Ας δεχθούμε προς το παρόν εδώ ότι, λόγω του template, έχουμε πέσει μέσα σε ένα υποδένδρο, οριζόμενο από αυτό το context node.

Μέσα στο template μπορούν να παρατίθενται οποιαδήποτε στοιχεία XML (άσχετα με το αρχικό μας XML το οποίο κάλεσε το συγκεκριμένο XSL), τα οποία θα εμφανίζονται κατά τα γνωστά. Εκτός από αυτά, στοιχεία / εντολές του κατ' εξοχήν XSL (τα εισαγόμενα με xsl:) αναφέρονται και συλλέγουν στοιχεία από το αρχικό XML, πάντα όμως μέσα από το υποδένδρο όπου βρισκόμαστε σύμφωνα με τα παραπάνω. Έτσι το <xsl:for-each ....> ..... <xsl:for-each/>, επιλέγει όλα τα στοιχεία στο υποδένδρο (εδώ το root '/'), σύμφωνα και με τα attributes order-by (κριτήριο ταξινόμησης) και select (στοιχείο επί του οποίου εφαρμόζεται). Τα attributes αυτά έχουν μέσα στο XSL την γνωστή σύνταξη (attribute='xyz'). Κατόπιν με το στοιχείο XSL xsl:value-of παίρνεται τελικά η τιμή (δηλ. το text), ή με @ το attribute του επιλεγέντος XML στοιχείου. Ουσιαστικά τα στοιχεία / εντολές του XSL μας δίνουν την δυνατότητα να πλοηγηθούμε μέσα στον επιλεγέντα context node και να επιλέξουμε στην διαδρομή μας οτιδήποτε επιθυμούμε. Για παράδειγμα, το select (attribute του value-of) περιέχει μια λεγόμενη XPath expression, δηλαδή ένα μονοπάτι που αρχίζει στον context node. Επίσης μπορούμε να θεωρήσουμε ότι xsl:for-each είναι ένα mini template μέσα στο

αρχικό, διότι μας φέρνει (στην περιοχή της ισχύος του) σε κόμβο ή σύνολο κόμβων (όλα τα vehicle) κάτω από τον context node (root /). Αυτό είναι προσωρινό και μόλις βγούμε από το select element, επανέρχεται ο ισχύων context node, όπως ορίστηκε από το match του template το οποίο ‘εκτελούμε’.

Το πλήρες κείμενο CarPresentor.xsl είναι

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w7.org/TR/WD-xsl">

  <xsl:template match="/">
    <html>
      <head>
        <title>Used Vehicles</title>
      </head>
      <body background="Money.jpg">
        <h1 style="background-color: #446600;
          color: #FFFFFF; font-size: 20pt; text-align: center;
          letter-spacing: 1.0em">Used Vehicles</h1>
        <table align="center" border="2">
          <tr>
            <th>Year</th>
            <th>Make</th>
            <th>Model</th>
            <th>Mileage</th>
            <th>Color</th>
            <th>Price</th>
          </tr>
          <!-- This template has had so far only presentation elements, -->
          <!-- without any reference to the target xml document -->
          <!-- We now refer to the the target xml document -->
          <xsl:for-each order-by="+ price" select="vehicles/vehicle">
            <tr>
              <td><xsl:value-of select="@year"/></td>
              <td><xsl:value-of select="@make"/></td>
              <td><xsl:value-of select="@model"/></td>
              <td><xsl:value-of select="mileage"/></td>
              <td><xsl:value-of select="color"/></td>
              <td><xsl:value-of select="price"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

### Παράδειγμα 7.2. Το stylesheet CarPresentor.xsl.

Τώρα θα εφαρμόσουμε το αρχείο vehicles\_transformer.xsl επί του vehicles.xml καλώντας τον xalan XSLT Processor γραμμένο σε Java μέσω command line.

```

<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<!-- OUTER template: -->
<xsl:template match="/"><!-- this match makes "/" the context node -->

  <!-- create a new element with a specified name -->
  <xsl:element name="jeep_users">Organizations using jeeps are:
    <!-- the following will fill the new element -->
    <!-- the XPath expression starts from the context node -->
    <xsl:for-each select="vehicles/jeeps/jeep">
      <xsl:value-of select="@user"/> and .....
    </xsl:for-each>
  </xsl:element>

  <!-- same as above, but NO content will be generated because
    XPath expression does NOT start from context node !!!! -->
  <xsl:element name="jeep_not_found">

    <xsl:for-each select="jeeps/jeep">
      <xsl:value-of select="@user"/>
    </xsl:for-each>
  </xsl:element>

  <!-- The context node moves now down to ... -->
  <xsl:apply-templates select="vehicles/lories"/>

  <!-- The context node moves up again to .. -->
  <xsl:apply-templates select="vehicles/ambulances"/>

  <!-- The context node moves up again to .. -->
  <xsl:apply-templates select="vehicles/jeeps/jeep"/>
</xsl:template><!-- END of OUTER template -->

<!-- FIRST template called by outer template -->
<xsl:template match="lories">
  This text will be printed in the place of node lories
  <!-- The context node moves now down to lory1 -->
  <xsl:apply-templates select="lory1"/>
</xsl:template><!-- End of FIRST template called by outer template -->

<!-- SECOND template called by outer template -->
<xsl:template match="ambulances">
  This text will be printed in the place of ambulances
</xsl:template><!-- End of SECOND template called by outer template -->

<!-- THIRD template called by outer template -->
<xsl:template match="jeep">
  This text will be printed in the place of EACH jeep
</xsl:template><!-- End of THIRD template called by outer template -->

<!-- Template called by FIRST template called by outer template -->
<xsl:template match="lory1">
...and this is the text for lory1
</xsl:template><!-- End of ... -->

</xsl:stylesheet>

```

**Παράδειγμα 7.3.** Το vehicles\_transformer.xsl (προς εφαρμογήν επί του vehicles.xml)

Η αναφορά στο αρχείο .xsl γίνεται τώρα στο command line

```
java org.apache.xalan.xslt.Process -IN vehicles.xml -XSL vehicles_transformer.xsl
```

δηλ. εφάρμοσε την διαδικασία ... Xalan.xslt.Process του επί του vehicles.xml σύμφωνα με τις οδηγίες του vehicles\_transformer.xsl, το οποίο μπορεί να απομακρυνθεί από την δεύτερη γραμμή του vehicles.xml. Μετά το -IN έρχεται το .xml εισόδου, μετά το -XSL το .xsl του μετασχηματισμού και μετά ένα -OUT θα μπορούσαμε να ορίσουμε ένα .xml ή .html εγγραφής του αποτελέσματος. ΠΡΟΣΟΧΗ: Ο παραπάνω τρόπος πειραματισμού με τον xalan είναι πάντοτε προτιμητέος, καθόσον οι browsers δεν ανταποκρίνονται συχνά σε πολλά από τα υποσχόμενα εδώ, ή και σε οποιοδήποτε άλλο σύγγραμμα.

Επειδή το νέο .xml μέσα στο .xsl, δεν περικλείεται καθ' ολοκληρίαν σε στοιχείο <html>, δεν θα έχουμε έξοδο html, αλλά απλή έξοδο κειμένου στην κονσόλα χειρισμού. Με τον τρόπο αυτό είναι δυνατόν να δημιουργήσουμε ένα πραγματικό .xml στην έξοδο (θα μπορούσε να εξαχθεί και σαν αρχείο). Προς αυτήν την κατεύθυνση, το xsl στοιχείο <xsl:element name="xyz"> δημιουργεί πράγματι ένα νέο XML element ονοματισμένο xyz, όπως κάνουμε παρακάτω με το στοιχείο <xsl:element name="jeep\_users">. Παρ' όλον ότι δεν υπάρχει ουσιαστική διαφορά με το προηγούμενο παράδειγμα, βλέπουμε εδώ μια πραγματική περίπτωση XSLTransformation - XSLT: δεν εμφανίζουμε ένα XML με ωραίο τρόπο αλλά το μετατρέπουμε σε κάποιο άλλο (νέο) XML. Μπορούμε να απευθυνθούμε στο αρχικό .xml και να αντλήσουμε από εκεί πληροφορία, π.χ. με @user λαμβάνουμε την τιμή του attribute user.

## XSLT και XPath

Η XSLT χρησιμοποιεί το XPath για να ανεύρει συγκεκριμένα μέρη (κόμβους – nodes) μέσα στο κείμενο XML είτε πρόκειται για στοιχεία (element nodes), ιδιότητες (attribute nodes), κόμβους κειμένου (text nodes), κλπ όπως στην σύμβαση κατά DOM. Παρατηρούμε ότι στοιχεία της XSLT σαν τα xsl:template, xsl:apply-templates, xsl:for-each, xsl:value-of έχουν ιδιότητες (attributes) match ή select, των οποίων οι τιμές δίδονται με την προτυποποιημένη σύνταξη XPath. Έτσι ανευρίσκονται οι επιθυμητοί κόμβοι μέσα στο υπό επεξεργασία κείμενο XML. Το XPath δημιουργεί 'μονοπάτια' μέσα στην δομή του κειμένου XML με τρεις τρόπους: (α) σχετικά ως προς το υπό επεξεργασία στοιχείο, (β) απόλυτα (ως προς το root) και (γ) με την βοήθεια pattern matching, όπου π.χ. μπορεί να εντοπισθεί στοιχείο με πατέρα το στοιχείο A και απόγονο το στοιχείο B. Στο ταίριαγμα μπορούν και να υπεισέρχονται και attributes, π.χ. το @ μας οδηγεί στο attribute. Το αποτέλεσμα της ανεύρεσης μέσω των εκφράσεων του XPath είναι στην γενική περίπτωση όχι ένας, αλλά ένα σύνολο κόμβων (nodes-set).

Επιπλέον με τα XPath Functions μπορούμε να ανεύρουμε χρήσιμες τιμές που απορρέουν από τον ενδεχομένως άγνωστο κόμβο (ή κόμβους) στο οποίο μας έχει φέρει το 'μονοπάτι'.

## Διαδικασία Εφαρμογής Templates

Για να καταλάβουμε την διαδικασία εφαρμογής των templates πρέπει να διευκρινίσουμε την σχέση μεταξύ των xsl:apply-templates select="node-set-expression" και xsl:template match="pattern". Χονδρικά το πρώτο είναι η κλήση και το δεύτερο ο ορισμός του template



(της καλουμένης ρουτίνας). Εν τούτοις δεν είναι σωστό να θεωρήσουμε απλά μία εκτέλεση του αρχείου .xsl (stylesheet), με κλήσεις υπορουτινών, όπως στον κλασσική εκτέλεση ενός προγράμματος. Η εξήγηση πρέπει συνεχώς να αναφέρεται και να μεταπίπτει από το κείμενο .xml στο κείμενο .xsl και πάλι πίσω.

Υποθέτουμε ότι ο XSLT Processor ευρίσκεται σε κάποιο κόμβο από το ‘τρέχον σύνολο κόμβων’ (current node set). Αρχικά αυτό το σύνολο αυτό είναι η ρίζα “/” του κειμένου XML. Κατόπιν

(α) ο XSLT Processor, για κάθε κόμβο “X” του τρέχοντος συνόλου κόμβων ψάχνει για τα στοιχεία <xsl:template match="pattern"> του stylesheet (αρχείο .xsl), τα οποία δυνητικά ταιριάζουν στον κόμβο αυτό. Από αυτά επιλέγει εκείνο πού προσφέρει το πλησιέστερο ταίριαγμα.

(β) το επιλεγέν <xsl:template match="pattern"> εφαρμόζεται με “X” ως τον τρέχοντα κόμβο.

(γ) εάν κατά τη εκτέλεση του (β) ευρεθεί <xsl:apply-templates select="node-set-expression"> δημιουργείται ένα νέο ‘τρέχον σύνολο κόμβων’ και η διαδικασία συνεχίζεται αναδρομικά. Το παραπάνω "node-set-expression" ανευρίσκεται με το XPath σχετικά ως προς τον κόμβο “X” και όχι απόλυτα (ως προς την ρίζα).

Βλέπουμε, ότι όσο το ‘τρέχον σύνολο κόμβων’ τόσο και ο τρέχον κόμβος συνεχώς αλλάζουν και μάλιστα όχι πάντα στην κατεύθυνση κατάβασης από την ρίζα στα φύλλα του κειμένου XML επί του οποίου εφαρμόζεται ο μετασχηματισμός XSLT. Δεν αποκλείεται επίσης η επανειλημμένη επίσκεψη στον ίδιο κόμβο.

Ας δούμε τώρα την παραπάνω διαδικασία εφαρμοζόμενη επί του vehicles.xml σύμφωνα με το stylesheet vehicles\_transformer.xsl. Ξεκινάμε με την ρίζα του vehicles.xml (X="/") στην οποία ταιριάζει το “OUTER template..” του stylesheet. Η έκφραση XPath “vehicles/jeeps/jeep” πού χρησιμοποιείται, είναι σχετική με τον τρέχοντα κόμβο “/”. Πιο κάτω πέφτομε στην περίπτωση (γ) και το νέο ‘τρέχον σύνολο κόμβων’ μεταπίπτει στο μοναδικό στοιχείο “vehicles/lories”. Ψάχνονται πάλι όλα τα templates του stylesheet και ευρίσκεται να ταιριάζει το “FIRST template..”. Μέσα σε αυτό ξαναπέφτομε στην περίπτωση (γ) και το ‘τρέχον σύνολο κόμβων’ μεταπίπτει στο “lory1”. Από όλα τα templates ταιριάζει τώρα το “Template called by “FISRT template ..”. Μετά την εκτέλεση αυτού, το ‘τρέχον σύνολο κόμβων’ επιστρέφει στο “vehicles/lories”, μετά πίσω στο “/” και μετά στο “vehicles/ambulances”. Σε αυτό ευρίσκεται να ταιριάζει το “SECOND template ..”, εκτελείται, το ‘τρέχον σύνολο κόμβων’ επιστρέφει στο “/” και μετά στο “vehicles/jeeps/jeep” για το οποίο ταιριάζει το “THIRD Template..”. Κατόπιν επιστρέφομε στο “/” και η διαδικασία τερματίζεται.

Παρακολουθώντας τα παραπάνω βλέπομε ότι η προγραμματιστική εκτέλεση των παραπάνω εμπλέκει δύο διαδικασίες parsing πού τρέχουν αλληλοδιαδόχως:

- η μία εφαρμόζεται επί του .xml και ευρίσκει τους κόμβους πού θα αποτελέσουν το ‘τρέχον σύνολο κόμβων’, ορμώμενη από το <xsl:apply-templates select="node-set-expression"> του .xsl.
- η άλλη εφαρμόζεται επί του .xsl και ευρίσκει τον κόμβους xsl:template match="pattern"> (δηλ το template) πού ταιριάζει καλύτερα σε εκείνον τον κόμβο, μέσα στο ‘τρέχον σύνολο κόμβων’, με τον οποίον τώρα ασχολούμαστε.

Και οι δύο διαδικασίες χρησιμοποιούν το XPath, όπως παραπάνω αναφέρθηκε. Πρέπει όμως να υπάρχει και διασφάλιση έναντι της περίπτωσης η παραπάνω διαδικασία να διακοπεί ατενυάτιστη, όταν κανένα template δεν ευρίσκεται να ταιριάζει στον τρέχοντα κόμβο του κειμένου .xml. Εδώ υπεισέρχεται η έννοια ‘πλησιέστερο ταίριαγμα’ στο σημείο (α), καθώς και η ύπαρξη των ενσωματωμένων εκμαγείων. Πέραν των templates πού εμείς γράφομε στο κείμενο .xsl, υπάρχουν και τέσσερα ενσωματωμένα (built-in templates). Αυτά τα φανταζόμαστε στο παρασκήνιο και έτοιμα να εφαρμοστούν (με την σειρά προτεραιότητας όπως παρατίθενται παρακάτω), αν κανένα από τα γραμμένα στο κείμενο .xsl δεν προσφέρει πλησιέστερο ταίριαγμα. Τότε λοιπόν εφαρμόζεται όποιο από τα παρακάτω στην σειρά ταιριάζει πρώτο. Με ‘|’ εκφράζεται το OR.

```
-----  
<xsl:template match="*/">  
  <xsl:apply-templates/>  
</xsl:template>
```

Αυτό ταιριάζει σε οποιοδήποτε στοιχείο (element), όπως υπονοεί το σύμβολο “\*”, και στην ρίζα “/”. Δεν ταιριάζει σε attributes, text, PI (processing instructions), comment. Λόγω του <xsl:apply-templates>, το ‘τρέχον σύνολο κόμβων’ του βήματος (γ) αποτελείται τώρα από όλα τα παιδιά του τρέχοντος κόμβου (πού είναι αυστηρά elements – όχι attributes, text, PI).

```
-----  
<xsl:template match="*/" mode="m">  
  <xsl:apply-templates mode="m"/>  
</xsl:template>
```

Ισχύουν τα ίδια με το παραπάνω, αλλά τώρα εφαρμόζεται σε σχέση με το attribute mode, με το οποίο δεν θα ασχοληθούμε περαιτέρω. Επομένως εδώ αγνοείται.

```
-----  
<xsl:template match="text()|@"*>  
  <xsl:value-of select="."/>  
</xsl:template>
```

Σε αντίθεση με τα δύο παραπάνω, τώρα ταιριάζει το template σε οποιοδήποτε κόμβο attribute και text. Το αποτέλεσμα είναι να αντιγραφούν όλα τα attribute και text στο αποτέλεσμα.

```
-----  
<xsl:template match="processing-instruction()|comment()"/>
```

Τώρα ταιριάζει το template σε οποιαδήποτε PI ή comment (σχόλιο). Το αποτέλεσμα είναι τίποτα, οπότε τα PI ή comment δεν αντιγράφονται στο αποτέλεσμα.

Είναι τώρα προφανές, ότι η διαδικασία δεν μπορεί να διακοπεί χωρίς να ληφθεί υπ’ όψιν ολόκληρο το κείμενο .xsl.

## Παραδείγματα και Ειδικές Περιπτώσεις

Το παρακάτω, απλό στην εμφάνιση κείμενο identity.xsl παίρνει το αρχικό κείμενο και αποδίδει ένα με τα ίδια στοιχεία (elements) και ιδιότητες (attributes).

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

### Παράδειγμα 7.4. Το identity.xsl (‘μοναδιαίος μετασχηματισμός’)

Με το στοιχείο xsl:output καθορίζουμε τον τρόπο εμφάνισης του αποτελέσματος του μετασχηματισμού. Το attribute method="xml" δίδει πραγματικό κείμενο xml και όχι html, ή απλό κείμενο στην κονσόλα, όπως παραπάνω. Με encoding="UTF-8" ορίζουμε την κωδικοποίηση χαρακτήρων και με το indent="yes" θέτουμε σε εφαρμογή μία αυτόματη εισαγωγή κενών, ώστε τα υποστοιχεία να ‘μπαίνουν πιο μέσα’ και η τελική εμφάνιση να εμφανίζει πιο παραστατικά την δενδρική δομή. Το βασικό μας template ταιριάζει είτε σε οποιοδήποτε attribute (@\*) είτε σε οποιοδήποτε κόμβο - παιδί (element, text, comment, κλπ). Ο διαχωρισμός χρειάζεται διότι η συνάρτηση node() του XPath επιστρέφει κάθε κόμβο παιδί, εκτός όμως των attributes. Επομένως το βασικό template ταιριάζει και εφαρμόζεται στην αρχή στο root και κατόπιν σε κάθε attribute και κάθε κόμβο – παιδί. Το ότι συνεχίζουμε από το root στα παρακάτω, οφείλεται στην ύπαρξη του <xsl:apply-templates select="@\*|node()"/> μέσα στο βασικό template. Και τέλος με το xsl:copy αντιγράφεται ο τρέχον κόμβος (χωρίς τα παιδιά του) στο βαθμιαία σχηματιζόμενο δένδρο του αποτελέσματος. Εφαρμόζοντας τον xalan XSLT Processor κατά τα ήδη γνωστά, βλέπουμε το αποτέλεσμα του identity.xsl επί οποιουδήποτε κειμένου .xml.

Ένας τρόπος προσέγγισης των δυνατοτήτων της XSLT είναι να ξεκινήσουμε από το πλήρες αρχικό κείμενο .xml και στο παραπάνω identity.xsl να προσθέτουμε templates με πιο εξειδικευμένες τιμές στο attribute match. Τότε, στα σημεία πού επιθυμούμε, θα εφαρμόζονται αυτά, ως πλέον ταιριαστά. Το παρακάτω template μετατρέπει το attribute με το όνομα motor, οπουδήποτε ευρίσκεται, σε στοιχείο με το ίδιο όνομα. Προσθέτουμε το template αυτό μέσα στο stylesheet (identity.xsl) και το εφαρμόζουμε με τον xalan επί του vehicles.xml.

```
<xsl:template match="@motor">
  <xsl:element name="{name(.)}">
    <xsl:value-of select="."/>
  </xsl:element>
</xsl:template>
```

### Παράδειγμα 7.5. Πρόσθετο εξειδικευμένο template

Ανάλογα θα μπορούσαμε να προσθέσουμε / μετατρέψουμε σε συγκεκριμένα σημεία. Προσθέτουμε οποιοδήποτε πρόσθετο template στα προϋπάρχοντα του identity.xsl και δοκιμάζουμε.

Η αντίθετη μεθοδολογία είναι να αρχίσουμε από το τίποτε και διαδοχικά να κτίζουμε το .xml εξόδου επιλέγοντας / μετατρέποντας από το αρχικό (με εντατική χρήση του XPath), περίπου όπως ήταν και τα πρώτα παραδείγματα του κεφαλαίου αυτού. Η γενική δομή τώρα αποτελείται από την παράθεση μίας σειράς templates, εκ των οποίων το πρώτο ταιριάζει στην ρίζα, καθώς και από κλήσεις apply-templates μέσα από αυτά.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>

  <!-- OUTER TEMPLATE: the whole original document is matched **-->
  <xsl:template match="/">
    <html>
      <head>
        <title>Text for Title</title>
      </head>
      <body>
        <xsl:apply-templates select="[XPath Expression]"/>
      </body>
    </html>
  </xsl:template>

  <!-- ANOTHER TEMPLATE "xyx"*****-->
  <xsl:template match="xyx">
    [and so on ...]
    <xsl:apply-templates select="[another XPath Expression]"/>
    [more contents or multiple apply-templates ...]
  </xsl:template match="xyx">

  <!-- YET ANOTHER TEMPLATE "xyx"*****-->

</xsl:stylesheet>
```

### **Παράδειγμα 7.6.** Γενική δομή .xsl για την εξ αρχής κατασκευή νέου κειμένου

Το παραπάνω είναι γραμμένο για έξοδο πάλι σε .html.

Γνωρίσαμε μερικά στοιχεία της XSLT – (XSL Transformations) και πράγματι είδαμε πώς χρησιμοποιούνται ώστε ένα κείμενο XML μετατρέπεται σε ένα άλλο. Η γενική ιδέα ήταν κάθε στοιχείο του αρχικού κειμένου να απεικονίζεται σε ένα άλλο, μπορούμε όμως ανάλογα να αντιγράψουμε (xsl:copy), προσθέτουμε (xsl:text), αφαιρούμε και να αναδιατάσσουμε (xsl:sort) στοιχεία από το αρχικό στο τελικό κείμενο, και όλα αυτά μάλιστα υπό συνθήκες (xsl:choose, xsl:if, xsl:when). Τα παραπάνω μπορούν να συνδυασθούν και με συναρτήσεις, τις XSL Functions. Παρακάτω δίδονται όλα τα στοιχεία της XSLT και από ποιους browser υποστηρίζονται (Explorer η Netscape). Δυστυχώς τα στοιχεία που υποστηρίζονται είναι ένα υποσύνολο της πλήρους προδιαγραφής και μάλιστα ΔΕΝ περιέχουν το πολύ χρήσιμο node() στο οποίο καταφύγαμε παραπάνω. Γι' αυτό όσα παραδείγματα δεν τρέχουν με τους browsers, πολύ πιθανόν να είναι σωστά δίδοντας τα αναμενόμενα μέσω του xalan XSLT Processor.

## XSLT Elements

Element	Description	IE	NN
<a href="#">xsl:apply-imports</a>	Applies a template rule from an imported style sheet	6.0	
<a href="#">xsl:apply-templates</a>	Applies a template rule to the current element or to the current element's child nodes	5.0	6.0
<a href="#">xsl:attribute</a>	Adds an attribute	5.0	6.0
<a href="#">xsl:attribute-set</a>	Defines a named set of attributes	6.0	6.0
<a href="#">xsl:call-template</a>	Calls a named template	6.0	6.0
<a href="#">xsl:choose</a>	Used in conjunction with <xsl:when> and <xsl:otherwise> to express multiple conditional tests	5.0	6.0
<a href="#">xsl:comment</a>	Creates a comment node in the result tree	5.0	6.0
<a href="#">xsl:copy</a>	Creates a copy of the current node (without child nodes and attributes)	5.0	6.0
<a href="#">xsl:copy-of</a>	Creates a copy of the current node (with child nodes and attributes)	6.0	6.0
<a href="#">xsl:decimal-format</a>	Defines the characters and symbols to be used when converting numbers into strings, with the format-number() function	6.0	
<a href="#">xsl:element</a>	Creates an element node in the output document	5.0	6.0
<a href="#">xsl:fallback</a>	Specifies an alternate code to run if the XSL processor does not support an XSL element	6.0	
<a href="#">xsl:for-each</a>	Loops through each node in a specified node set	5.0	6.0
<a href="#">xsl:if</a>	Contains a template that will be applied only if a specified condition is true	5.0	6.0
<a href="#">xsl:import</a>	Imports the contents of one style sheet into another. <b>Note:</b> An imported style sheet has lower precedence than the importing style sheet	6.0	6.0
<a href="#">xsl:include</a>	Includes the contents of one style sheet into another. <b>Note:</b> An included style sheet has the same precedence as the including style sheet	6.0	6.0
<a href="#">xsl:key</a>	Declares a named key that can be used in the style sheet with the key() function	6.0	6.0
<a href="#">xsl:message</a>	Writes a message to the output (used to report errors)	6.0	6.0
<a href="#">xsl:namespace-alias</a>	Replaces a namespace in the style sheet to a different namespace in the output	6.0	
<a href="#">xsl:number</a>	Determines the integer position of the current node and formats a number	6.0	6.0
<a href="#">xsl:otherwise</a>	Specifies a default action for the <xsl:choose> element	5.0	6.0
<a href="#">xsl:output</a>	Defines the format of the output document	6.0	6.0

<a href="#">xsl:param</a>	Declares a local or global parameter	6.0	6.0
<a href="#">xsl:preserve-space</a>	Defines the elements for which white space should be preserved	6.0	6.0
<a href="#">xsl:processing-instruction</a>	Writes a processing instruction to the output	5.0	6.0
<a href="#">xsl:sort</a>	Sorts the output	6.0	6.0
<a href="#">xsl:strip-space</a>	Defines the elements for which white space should be removed	6.0	6.0
<a href="#">xsl:stylesheet</a>	Defines the root element of a style sheet	5.0	6.0
<a href="#">xsl:template</a>	Rules to apply when a specified node is matched	5.0	6.0
<a href="#">xsl:text</a>	Writes literal text to the output	5.0	6.0
<a href="#">xsl:transform</a>	Defines the root element of a style sheet	6.0	6.0
<a href="#">xsl:value-of</a>	Extracts the value of a selected node	5.0	6.0
<a href="#">xsl:variable</a>	Declares a local or global variable	6.0	6.0
<a href="#">xsl:when</a>	Specifies an action for the <xsl:choose> element	5.0	6.0
<a href="#">xsl:with-param</a>	Defines the value of a parameter to be passed into a template	6.0	6.0

## XSLT Functions

Name	Description
<a href="#">current()</a>	Returns the current node
<a href="#">document()</a>	Used to access the nodes in an external XML document
<a href="#">element-available()</a>	Tests whether the element specified is supported by the XSLT processor
<a href="#">format-number()</a>	Converts a number into a string
<a href="#">function-available()</a>	Tests whether the function specified is supported by the XSLT processor
<a href="#">generate-id()</a>	Returns a string value that uniquely identifies a specified node
<a href="#">key()</a>	Returns a node-set using the index specified by an <xsl:key> element
<a href="#">system-property()</a>	Returns the value of the system properties
<a href="#">unparsed-entity-uri()</a>	Returns the URI of an unparsed entity

## XPath Functions

Name	Description	Syntax
count()	Returns the number of nodes in a node-set	number=count(node-set)
id()	Selects elements by their unique ID	node-set=id(value)
last()	Returns the position number of the last node in the processed node list	number=last()
local-name()	Returns the local part of a node. A node usually consists of a prefix, a colon, followed by the local name	string=local-name(node)
name()	Returns the name of a node	string=name(node)
namespace-uri()	Returns the namespace URI of a specified node	uri=namespace-uri(node)
position()	Returns the position in the node list of the node that is currently being processed	number=position()

## String Functions

Name	Description	Syntax & Example
Concat()	Returns the concatenation of all its arguments	string=concat(val1, val2, ..)  <b>Example:</b> concat('The', ' ', 'XML') Result: 'The XML'
Contains()	Returns true if the second string is contained within the first string, otherwise it returns false	bool=contains(val, substr)  <b>Example:</b> contains('XML', 'X') Result: true
normalize-space()	Removes leading and trailing spaces from a string	string=normalize-space(string)  <b>Example:</b> normalize-space(' The XML ') Result: 'The XML'
starts-with()	Returns true if the first string starts with the second string, otherwise it returns false	bool=starts-with(string, substr)  <b>Example:</b> starts-with('XML', 'X')

		Result: true
string()	Converts the value argument to a string	string(value)  <b>Example:</b> string(314) Result: '314'
string-length()	Returns the number of characters in a string	number=string-length(string)  <b>Example:</b> string-length('Beatles') Result: 7
substring()	Returns a part of the string in the string argument	string=substring(string,start,length)  <b>Example:</b> substring('Beatles',1,4) Result: 'Beat'
substring-after()	Returns the part of the string in the string argument that occurs after the substring in the substr argument	string=substring-after(string,substr)  <b>Example:</b> substring-after('12/10','/') Result: '10'
substring-before()	Returns the part of the string in the string argument that occurs before the substring in the substr argument	string=substring-before(string,substr)  <b>Example:</b> substring-before('12/10','/') Result: '12'
translate()	Takes the value argument and replaces all occurrences of string1 with string2 and returns the modified string	string=translate(value,string1,string2)  <b>Example:</b> translate('12:30',':','!') Result: '12!30'

### Number Functions

Name	Description	Syntax & Example
ceiling()	Returns the smallest integer that is not less than the number argument	number=ceiling(number)  <b>Example:</b> ceiling(7.14) Result: 4
floor()	Returns the largest integer that is not greater than the number argument	number=floor(number)  <b>Example:</b> floor(7.14) Result: 3



number()	Converts the value argument to a number	number=number(value)  <b>Example:</b> number('100') Result: 100
Round()	Rounds the number argument to the nearest integer	integer=round(number)  <b>Example:</b> round(7.14) Result: 3
sum()	Returns the total value of a set of numeric values in a node-set	number=sum(nodeset)  <b>Example:</b> sum(/cd/price)

### Boolean Functions

Name	Description	Syntax & Example
boolean()	Converts the value argument to Boolean and returns true or false	bool=boolean(value)
false()	Returns false	false()  <b>Example:</b> number(false()) Result: 0
lang()	Returns true if the language argument matches the language of the the xsl:lang element, otherwise it returns false	bool=lang(language)
not()	Returns true if the condition argument is false, and false if the condition argument is true	bool=not(condition)  <b>Example:</b> not(false())
true()	Returns true	true()  <b>Example:</b> number(true()) Result: 1

## 8. XML-RPC

Η εξ αποστάσεως κλήση μίας διαδικασίας (Remote Procedure Call - RPC) αποτελεί την πεμπτούσια του μοντέλου πελάτου – εξυπηρετητή (client - server) στα κατακεμημένα συστήματα. Μία ή πολλές διαδικασίες (μέθοδοι στην ορολογία της Java) ευρίσκονται διαθέσιμες στον εξυπηρετητή (στον κοινό server) προς γενική χρήση από τον πελάτη (συνήθως έναν από πολλούς clients). Ο client πρέπει να αποστείλει πληροφορία πού καθορίζει (α) το όνομα της διαδικασίας (μεθόδου) και (β) τις παραμέτρους εισόδου. Αφού εκτελεστεί η διαδικασία στον server, πρέπει να επιστραφεί (γ) το αποτέλεσμα πίσω στον client. Οι παράμετροι εισόδου αλλά και το αποτέλεσμα δυνατόν να είναι πολύπλοκες δομές, στην περίπτωση μας αντικείμενα της Java. Πριν την ανάπτυξη της XML, η κλασσική κλήση RPC, προσπαθούσε να κωδικοποιήσει τα (β) και (γ) σε μορφή κειμένου (textual representation) με ειδικό και ιδιαίτερα συμφωνημένο τρόπο για κάθε γλώσσα, υπολογιστικό περιβάλλον, κλπ. Πάνω σε αυτή την προϊστορία, ήρθε ο συνδυασμός XML και Java να δώσει μια πολύ κομψή διέξοδο. Η XML προσφέρει μια απλή έκφραση σε μορφή κειμένου (textual representation), αλλά και ένα προτυποποιημένο τρόπο δόμησης του κειμένου αυτού. Προσθέτοντας τώρα την δυνατότητα μετάβασης από κείμενο XML σε δομές της Java (και αντίστροφα), με βάση τις τεχνολογίες των προηγούμενων κεφαλαίων, αποκτώμε ένα ολοκληρωμένο περιβάλλον γνωστό σαν XML-RPC.

Παρακάτω θα δημιουργήσουμε έναν XML-RPC server (class XmlRpcMyServer) και θα δηλώσουμε σε αυτόν έναν handler μέσω της addHandler, μεθόδου του αντικείμενου server της class WebServer. Κατόπιν με την XmlRpcMyClient θα δημιουργήσουμε έναν πελάτη (client). Σκοπός μας είναι να δούμε τον πελάτη να καλεί εξ αποστάσεως (Remote Procedure Call - RPC) μίαν μέθοδο του handler και να παίρνει απάντηση, και τα δύο με χρήση κωδικοποίησης κατά XML. Ο server θα ακούει στην υποδοχή (socket) 8585. Το αντικείμενο της Handler class είναι γνωστό στον Client με το όνομα “hello” και περιέχει μία μέθοδο (θα μπορούσε και περισσότερες), της οποίας το όνομα είναι γνωστό στον client (“sayHello”). Πρόκειται πράγματι για Remote Procedure Call (RPC), πού στο περιβάλλον της Java ανάγεται στην εξ αποστάσεως κλήση μίας μεθόδου (της sayHello). Η XML συνεισφέρει στο να στέλνει ο client στην μέθοδο αυτή τις παραμέτρους εισόδου και στο να του επιστρέφεται από τον server το αποτέλεσμα. Στην περίπτωση μας ο client θα στέλνει ένα String “onoma” και θα του επιστρέφεται το String “hello onoma”. Τα προτερήματα του XML-RPC, σε σχέση με τις παλαιότερες τεχνικές client server, θα εξηγηθούν μετά το παράδειγμα.

Κατ’ αρχήν δημιουργούμε τον handler (class HelloHandler). Κάθε τέτοιος handler περιέχει μίαν ή περισσότερες μεθόδους (εδώ την μοναδική sayHello). Ο σκοπός είναι μέσω της δήλωσης ενός ή περισσότερων τέτοιων handlers στον ένα και μοναδικό server να κάνουμε τις μεθόδους αυτές προσπελάσιμες εξ αποστάσεως (remotely). Η δική μας μέθοδος sayHello απλώς επικολλά και επιστρέφει το όνομα πού της δίνουμε στο String “Hello”.

```
public class HelloHandler {  
  
    public String sayHello(String name)  
    { return "Hello " + name; }  
}
```

**Κώδικας 8.1.** Η μέθοδος sayHello προς ενσωμάτωση στον server

Στον παρακάτω κώδικα του server δημιουργούμε το αντικείμενο server σαν νέον WebServer περνώντας και σαν args[0] τον αριθμό της υποδοχής (socket), όπου επιθυμούμε να ακούει.

Μετά δηλώνουμε στον server τον προκατασκευασμένο handler (μέθοδος addHandler του server). Μέσω της addHandler ορίζουμε επίσης το αναγνωριστικό (“hello”) με το οποίο ο handler αυτός θα είναι γνωστός.

```
import java.io.IOException;
import org.apache.xmlrpc.*;
import org.apache.xerces.parsers.*;

public class XmlRpcMyServer {

    public static void main(String args[]) throws ClassNotFoundException
    {
        //Specify driver for parsing/encoding from/to XML
        //(XmlRpc is static !)
        XmlRpc.setDriver("org.apache.xerces.parsers.SAXParser");

        //Instantiate & start the server
        System.out.println("Server getting up");
        WebServer server = new WebServer(Integer.parseInt(args[0]));
        server.start();

        //Register the handler
        server.addHandler("hello", new HelloHandler());
        System.out.println("Handler registered");
    }
}
```

### **Κώδικας 8.2.** Ο server πού φιλοξενεί την μέθοδο sayHello

Τέλος έχουμε τον client. Στο δημιουργούμενο αντικείμενο client της XmlRpcClient ορίζουμε την υποδοχή την οποίαν αυτός θα καλεί. Η προς κλήση μέθοδος στην μεριά του server καθορίζεται σύμφωνα με τα παραπάνω με hello.sayHello. Τούτο σαν String περνιέται στην μέθοδο execute του client. Επίσης έχουμε και τις παραμέτρους πού ζητά η sayHello. Αυτές οι παράμετροι (εδώ ένα μόνον String) πρέπει να περαστούν στην μορφή μίας Java Vector class. Το αποτέλεσμα πού επιστρέφει η μέθοδος του handler στον server προς αποστολή πίσω στον client ορίζεται σαν ένα γενικό αντικείμενο (Java object). Στην συγκεκριμένη περίπτωση γνωρίζουμε ότι είναι String, οπότε με casting αποκτώμε το αποτέλεσμα. Έτσι η έκφραση

```
(String)client.execute ("hello.sayHello", params);
```

αντιπροσωπεύει το Remote Procedure Call (RPC) και ισοδυναμεί με μία τοπική κλήση μεθόδου πού θα γινόταν μέσα στον client. Στην πραγματικότητα ο client κωδικοποιεί το όρισμα της execute, δηλαδή το ("hello.sayHello", params) και το στέλλει στον server. Ο server εντοπίζει την class πού έχει το αναγνωριστικό hello, μετά την μέθοδο sayHello αυτής της class και τέλος εξετάζει αν ο Vector των params πού ήλθε ταιριάζει με την δήλωση των arguments της μεθόδου. Τα βήματα αυτά είναι κλασσικά βήματα όπως και στη (τοπική) κλήση οποιασδήποτε μεθόδου στην Java. Επίσης ισχύει και το overloading της μεθόδου: δυνατόν να υπάρχουν στον handler πολλές μέθοδο με το ίδιο όνομα, αλλά με διαφορετικό signature, δηλαδή διαφορετική διάταξη παραμέτρων εισόδου. Στην περίπτωση αυτή θα κληθεί η μέθοδος εκείνη της οποίας το signature ταιριάζει με τον Vector των παραμέτρων πού έστειλε ο client.

```

import java.io.IOException;
import java.util.Vector;
import org.apache.xmlrpc.*;
import org.apache.xerces.parsers.*;

public class XmlRpcMyClient {

    public static void main(String args[])
        throws ClassNotFoundException,IOException, XmlRpcException
    {
        String serverPort = args[0];
        XmlRpc.setDriver("org.apache.xerces.parsers.SAXParser");
        //Specify the server and port
        XmlRpcClient client=new XmlRpcClient("http://localhost:" + serverPort + "/");

        //Create a request
        Vector params= new Vector();
        params.addElement(args[1]);

        //Make request and print result
        String result =
            (String)client.execute ("hello.sayHello",params);
        System.out.println("Response from Server: " + result);
    }
}

```

### **Κώδικας 8.3.** Ο client πού καλεί την μέθοδο sayHello

Αφού μεταγλωττίσουμε τα παραπάνω με τον javac, σηκώνουμε τον server με την εντολή στο command window (μέσα από το directory της XmlRpcMyServer.class)

```
start java XmlRpcMyServer 8085
```

Η start είναι εντολή του command window. Ανοίγεται ένα δεύτερο παράθυρο, όπου τώρα τρέχει ο server, ο οποίος τώρα ‘ακούει’ στην θύρα 8085. Τούτο μπορεί να διαπιστωθεί με Ctrl+Alt+Del, όπου ο Task Manager δείχνει μία java.exe να τρέχει (αυτή του σηκωμένου server). Τώρα πληκτρολογούμε στο αρχικό παράθυρο

```
java XmlRpcMyClient 8085 somename
```

Η πρώτη παράμετρος δίδει την θύρα (port) του server, όπου θέλουμε να στείλουμε την κλήση. Η δεύτερη είναι η παράμετρος της κλήσης. Όλα τα επιδιωκόμενα πραγματοποιούνται και πληροφορούμεθα, ότι ο client έλαβε πίσω ‘hello somename’.

Τελικά πού ευρίσκεται η συμμετοχή του XML; Στους server και client δημιουργήσαμε έναν SAX parser με την μέθοδο setDriver της class XmlRpc. Αυτή εισήχθη και στον server και στον client, ως ανήκουσα στο package org.apache.xmlrpc. Επειδή μάλιστα είναι static, δεν χρειάζεται instantiation (δημιουργία object με new). Η μέθοδος setDriver ορίζει την χρήση ενός SAX parser. Με αυτόν ο client κωδικοποιεί την αίτηση του σε XML και ο server αποκωδικοποιεί την αίτηση του client από XML σε αντικείμενα Java. Μετά ο server κωδικοποιεί τα επιστρεφόμενα από τις μεθόδους του handler αντικείμενα σε XML προς αποστολήν πίσω στον client. Ο υπόλοιπος WebServer δεν γνωρίζει από XML, απλώς λαμβάνει και στέλνει μέσω HTTP στην προκαθορισμένη υποδοχή. Τέλος στον client αποκωδικοποιείται το XML σε ένα αντικείμενο Java, από το οποίο λαμβάνομε με casting τον αναμενόμενο τύπο. Όλα επιτυγχάνονται με την συνεισφορά της XmlRpc class και του parser πού όρισε.

Στο παραπάνω παράδειγμα ο client απέστειλε και έλαβε πίσω ένα απλό String. Ποίες άλλες δομές θα μπορούσαν να διευθετηθούν από την class XmlRpc που χρησιμοποιήσαμε; Η απάντηση ευρίσκεται στον παρακάτω πίνακα με τους χρησιμοποιήσιμους τύπους και την αντιστοίχησή τους στα δύο περιβάλλοντα. Ο παρακάτω πίνακας συνεχώς εμπλουτίζεται σε νέες εκδόσεις (βλ. [www.xmlrpc.com](http://www.xmlrpc.com)).

XML-RPC data type	Java type
Int	int
boolean	boolean
string	String
double	double
DateTime.iso8601	Date
struct	Hashtable
array	Vector
base64	byte[]
Nil	null

**Πίνακας 8.1.** Τύποι που περιλαμβάνονται στο XML-RPC

Τέλος μπορούμε και να ‘φωτογραφίσουμε’ την ανταλλαγή των μηνυμάτων HTTP με τα ενσωματωμένα κείμενα XML, που αντιστοιχούν στο παραπάνω παράδειγμα. Αυτά μπορούμε πράγματι να τα καταγράψουμε με έναν TCP Monitor, όπως δίδεται παρακάτω. Ο client στέλνει το μήνυμα του Σχήματος 8.1, ο server στέλνει αυτό του 8.2 (προσέξτε πάντα την κενή γραμμή κάτω από το HTTP header)

```

POST /PRC2 HTTP/1.1
User-Agent: Apache Web Server /7.1 Beta
Host: newInstance.com
Content-Type: text/xml
Content-length: ....

<?xml version="1.0"?>
<methodCall>

<methodName>hello.sayHello</methodName>
<params>
  <param>
    <value><string>George</string></value>
  </param>
</params>
</methodCall>

```

```

HTTP/1.1 200 OK
Connection: close
Content-Type: text/xml
Content-length: ....
Date : .....
Server: Apache Web Server /7.1 Beta

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value><string>HelloGeorge</string>
    </value >
    </param>
  </params>
</methodResponse>

```

**Σχήμα 8.1.** (αριστερά) Το μήνυμα HTTP του client με την κλήση της μεθόδου sayHello

**Σχήμα 8.2.** (δεξιά) Το μήνυμα HTTP του server με την απάντηση της κλήσης της sayHello

Σημειώνεται ότι ο τρόπος αποτύπωσης της αίτησης και της απόκρισης στο XML είναι προκαθορισμένος στις προδιαγραφές του XML-RPC χωρίς δυνατότητα δικής μας επέμβασης. Για παράδειγμα τα χρησιμοποιούμενα tags των στοιχείων έχουν προκαθορισμένα ονόματα (βλ. και την XML-RPC στήλη του παραπάνω πίνακα). Προχωρώντας περαιτέρω θα υπερβούμε και αυτόν τον περιορισμό στο κεφάλαιο περί SOAP.

## Αναδιαταξιμότητα του Server

Μεγάλη σημασία έχει η δυνατότητα διαταξιμότητας (configurability) πού μπορούμε να επιτύχουμε για τον Server στο περιβάλλον XML-RPC. Ο παραπάνω κώδικας του server, δεν προσφέρει αυτήν την ιδιότητα. Η προσθήκη ενός πρόσθετου handler απαιτεί recompilation για να ληφθεί υπ' όψιν ο νέος κώδικας. Στην θέση της απολύτου αναφοράς

```
server.addHandler("hello", new HelloHandler())
```

πρέπει να χρησιμοποιήσουμε κάτι σαν

```
server.addHandler(handlerName, Class.forName(handlerClass).newInstance())
```

Τώρα τα handlerName και handlerClass διαβάζονται από εξωτερικό (configuration) file. Χρησιμοποιείται επίσης η τεχνική του 'dynamic class loading': η Class.forName(handlerClass) επιστρέφει την class με το ζητούμενο όνομα, από την οποία η new Instance() κατασκευάζει το αντικείμενο, το οποίο προσδένεται στον Server. Με τον τρόπο αυτό, με απλό reboot, ο server ενημερώνεται για το νέο σύνολο των handlers και των μεθόδων πού θέλουμε να προσφέρουμε στους χρήστες. Βεβαίως οι χρήστες (clients) εξακολουθούν να πρέπει να είναι ενημερωμένοι για τις νέες υπηρεσίες που διατίθενται (για το όνομά τους και τις απαιτούμενες παραμέτρους εισόδου). Επιτύχαμε όμως να μην απαιτείται διακοπή λειτουργίας για recompilation του server.

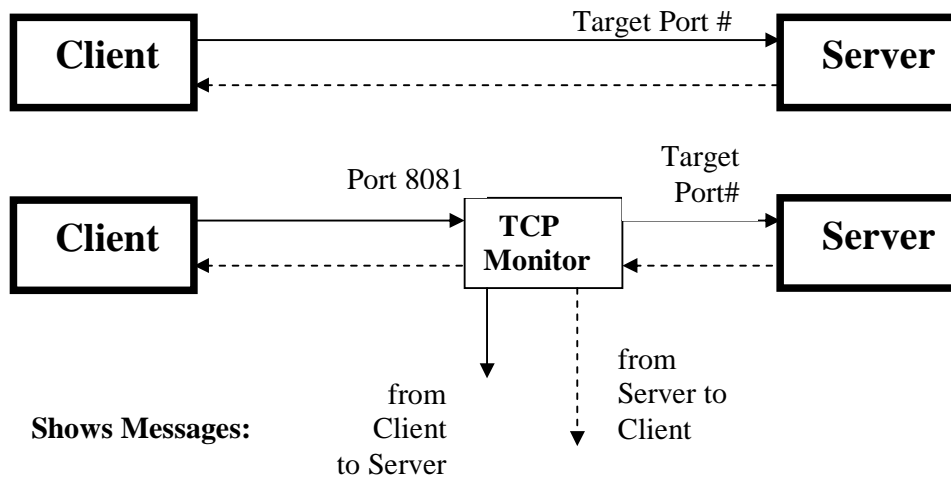
## Παρακολούθηση Μηνυμάτων με TCP Monitor

Στην προηγούμενη και επόμενη ενότητα μπορούμε να παρακολουθήσουμε τα ίδια τα μηνύματα πού μεταφέρονται μεταξύ server και client. Στην πλευρά του client εγκαθιστούμε έναν 'TCP Monitor' σύμφωνα με το παρακάτω Σχήμα 8.7.

Η μόνη μετατροπή πού χρειάζεται όταν παρεμβάλλεται ο TCP Monitor είναι ο client να μιλά σε αυτόν (στο 'Listen Port#' 8081) και αυτός να μιλά σε οποιονδήποτε server ('Target HostName', στα παραδείγματά μας θέτομε 'localhost') και θύρα αυτού ('Target Port#'). Στην αντίθετη διεύθυνση συμβαίνουν τα ανάλογα: ο server απαντά και η απάντηση, πριν επιδοθεί στον client, περνά μέσα από τον TCP Monitor. Ο server είναι τελείως ουδέτερος και ανυποψίαστος για την παρεμβολή του TCP Monitor. Στον client, η μόνη απαιτούμενη αλλαγή είναι να μην μιλά στον server αλλά στον TCP Monitor. Ο TCP Monitor ενεργοποιείται (από το directory του client) με την εντολή

```
java org.apache.axis.utils.tcpmon
```

Περνάμε τα παραπάνω αναφερθέντα στοιχεία στο πρώτο παράθυρο πού ανοίγει (Admin), πατάμε Add και εμφανίζεται νέο tab με το επιλεγέν Port #. Αφήνοντας την διαδικασία του TCP Monitor να τρέχει, έχουμε τώρα την 'φωτογραφία' όλων των μηνυμάτων πού διακινούνται.

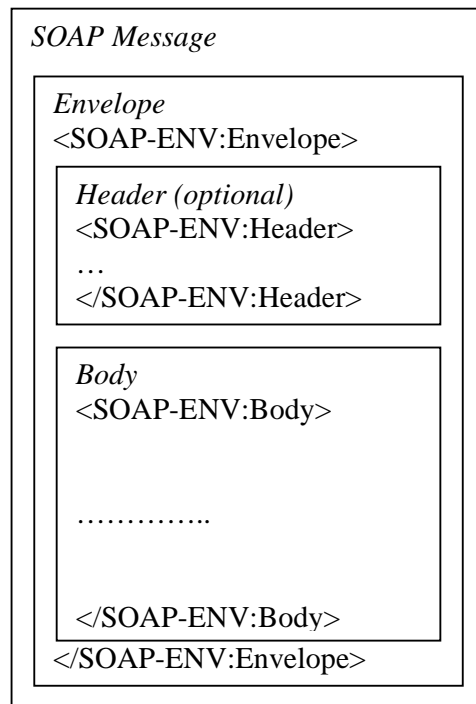


**Σχήμα 8.7.** Παρεμβολή και διευθύνσεις με τον TCP Monitor

Μπορούμε τώρα να επαναλάβουμε το παραπάνω παράδειγμα του XML-RPC και να δούμε τα πραγματικά HTTP πακέτα που ανταλλάσσονται. Θα έχουμε τρία ανοικτά παράθυρα (command windows), για τον server, για τον TCP Monitor και για τον client.

## 9. SOAP

Το Simple Object Access Protocol (SOAP) επιτρέπει την κατά παραγγελίαν διάρθρωση του κειμένου XML που μεταφέρει την 'πληροφορία' μεταξύ client και server πάνω από HTTP, χωρίς αυτή να υπόκειται στα προκαθορισμένα πρότυπα που χρησιμοποιεί το XML-RPC. Η 'πληροφορία' μπορεί τώρα να είναι ένα αντικείμενο (object) κάποιας class ορισμένης από την εφαρμογή, έτσι ώστε το SOAP να δρά σαν serializer: η δομή του αντικειμένου μετατρέπεται προς και από μία σειριακή ροή δεδομένων πάνω στο 'σύρμα' μετάδοσης. Η μετατροπή γίνεται σύμφωνα με κανόνες και πρότυπα βασισμένα στην XML και από κοινού συμφωνημένα σε client και server (το πρωτόκολλο SOAP). Το SOAP προωθείται και μέσα στα πλαίσια της Java ([www.w3.org/TR/SOAP](http://www.w3.org/TR/SOAP)) και από την Microsoft ([msdn.microsoft.com](http://msdn.microsoft.com)). Μία αναφορά (μέσω της ιδιότητας encodingStyle) στην διάρθρωση της κωδικοποίησης σε XML περιλαμβάνεται στο μήνυμα, έτσι ώστε ο δέκτης προκαταρκτικά να γνωρίζει και αποφασίζει αν και πώς θα ασχοληθεί με το αυτό. Γενικά το SOAP διαχωρίζει σαφώς σώμα (body - περιεχόμενο) και το φάκελο (envelope) που το περιλαμβάνει, και τα δύο εκφρασμένα σε XML. Ένα envelope μπορεί να έχει προαιρετικά μία επικεφαλίδα (header) και body με πολλά άμεσα στοιχεία – παιδιά, τα οποία αποτελούν τα λεγόμενα body blocks. Το SOAP μπορεί να χρησιμοποιηθεί και μέσα στα πλαίσια του RPC (SOAP-RPC.) σαν πιο εξελιγμένο και δυνατότερο του XML-RPC και για γενική μεταφορά πληροφορίας άσχετης με τις εξ αποστάσεως κλήσεις (SOAP Messaging).



**Σχήμα 9.1.** Γενική δομή μηνύματος SOAP και σχέση μεταξύ envelope, (header) και body

## SOAP-RPC

Με το Simple Object Access Protocol (SOAP) η προεργασία της κλήσης μίας RPC όπως στο προηγούμενο παράδειγμα ακολουθεί μία πιο ελέγξιμη διαδικασία. Πρέπει κατ' αρχήν να δημιουργηθεί στην πλευρά του client ένα Call object και σε αυτό να δηλωθούν ξεχωριστά τα στοιχεία της κλήσης. Όπως φαίνεται στο παρακάτω παράδειγμα, τούτο πρέπει να γίνει για το URI του server, για το όνομα της μεθόδου, για τον τρόπο κωδικοποίησης (είναι προς το παρόν σχεδόν πάντα ο ίδιος) και για τις παραμέτρους (πάλι μέσω Vector). 'Υπάρχουν ενδεχομένως και άλλα στοιχεία τα οποία εδώ παραλείπονται. Έχοντας τώρα αυτό το Call object στη μνήμη, μπορούμε δυναμικά να διαμορφώνουμε τις κλήσεις μας αντί του παγιωμένου στο πρόγραμμα τρόπου που είχαμε πριν.

```
// Creating and determining the Call Object
Call call = new Call();
// With different 'set...' methods we configure this call object for further use
call.setTargetObjectURI("...");
call.setMethodNameURI("hello");
call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
call.setParams(params); // params always is an object of class Vector
```

**Σχήμα 9.2.** Δημιουργία και χρήση του Call object



Η κλήση γίνεται πάλι με μέθοδο του Call object

```
// Invocation of the Remote Procedure Call  
Response res = call.invoke(new URL("..."), "");
```

### Σχήμα 9.3. Απόκριση μέσω του Call object

Οπότε τώρα πλέον ο client δεν έχει παρά να συνεχίσει με την επεξεργασία της απόκρισης, η οποία έχει δημιουργηθεί μέσα στον client σαν res (Response object).

Το μήνυμα πού θα μεταδοθεί έχει τώρα την μορφή πού ακολουθεί το SOAP αποτελούμενο από Envelope (φάκελο) και Body (σώμα, περιεχόμενο), βεβαίως πάντα κωδικοποιημένο κατά XML. Πολλοί φάκελοι με περιεχόμενο μπορούν να αποσταλούν το ένα κάτω από το άλλο μέσα στο ίδιο μήνυμα (SOAP Message). Στο κάλυμμα τίθεται όλη εκείνη η πληροφορία πού προσδιορίζει αποστολέα και παραλήπτη, τρόπο κωδικοποίησης (encodingStyle attribute) και αναμενόμενης επεξεργασίας του σώματος, κλπ. Σκοπός είναι ο παραλήπτης να μπορέσει να κρίνει αν έχει την δυνατότητα και αν τον ενδιαφέρει να επεξεργασθεί το σώμα. Αν συγκρίνομε το παρακάτω παράδειγμα με το κοινό RPC (εδώ πλέον έχομε SOAP-RPC), θα δούμε ότι τέτοια πληροφορία δεν παρείχεται πριν εντός του μεταφερομένου XML μηνύματος.

```
<soap:Envelope  
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"  
  soap:encodingStyle="http://myHOST.com/encodings/seecureEncoding"  
>  
  <soap:Body>  
    <article> ....  
  </article>  
  ....  
</soap:Body>  
</soap:Envelope>
```

### Σχήμα 9.4. Βασικά στοιχεία του μηνύματος SOAP

Επιπλέον έχομε απελευθερωθεί από τους περιορισμούς των τύπων που μπορούν να ανταλλαγούν, είτε σαν παράμετροι εισόδου είτε σαν απόκριση της κλήσης (παραπάνω 'Πίνακας Τύπων' πού ισχύει για το κοινό RPC). Εδώ κάθε δομή εκφρασμένη σε XML και συμβατή με ένα προσυμφωνημένο XML Schema είναι αποδεκτή σαν περιεχόμενο του σώματος.

Το SOAP διαχωρίζει σαφώς σώμα (body - περιεχόμενο) και το κάλυμμα (envelope) πού το περιλαμβάνει. Και τα δύο καθώς και όλο το μήνυμα είναι εκφρασμένα σε XML.

## Παράδειγμα Υπηρεσίας SOAP-RPC

Εδώ θα χρειασθούμε ένα 'servlet engine'. Τα servlets είναι memory-resident προγράμματα που τρέχουν μέσα σε ένα servlet engine (ή 'servlet container'). Επειδή ευρίσκονται ανά πάσα στιγμή στην μνήμη ανταποκρίνονται ταχύτατα σε αιτήσεις που καταφθάνουν στον (web) server από τους clients, χωρίς τις επιβαρύνσεις δημιουργίας μίας διαδικασίας (process creation) και κατοπινού καθαρισμού της (process cleanup). Τα servlets μπορούν να νοηθούν σαν applets, που τρέχουν όμως στον server και δεν δείχνουν GUI στον χρήστη. Με το τρόπο αυτό επεκτείνονται οι δυνατότητες ενός web server. Για παράδειγμα στην τεχνολογία JSP (Java Server Pages) έχουμε δυναμικές σελίδες web χρησιμοποιώντας περαιτέρω εξειδικευμένα servlets.

Ένας web server μπορεί να ανταποκρίνεται σε πακέτα HTTP. Τούτο σημαίνει ότι μπορεί να αντιμετωπίζει αιτήσεις που καταφθάνουν σύμφωνα με το πρωτόκολλο HTTP, και συγκεκριμένα τα πακέτα GET και POST. Ο web (ή HTTP) server 'ακούει' την θύρα (π.χ. 8080) μίας σύνδεσης TCP και όταν καταφθάνει μία τέτοια αίτηση από ένα πελάτη (client) την προωθεί στο κατάλληλο πρόγραμμα (καλύτερα software component) που είναι υπεύθυνο για την επεξεργασία της. Επιπλέον αναλαμβάνει να διεκπεραιώσει την (ενδεχόμενη) απάντηση του software component και να την αποστείλει μέσω HTTP πίσω στον πελάτη. Τέτοιος web server είναι ο Tomcat και στις περιπτώσεις που θα δούμε στην ενότητα αυτή το περιεχόμενο κάθε μηνύματος που είτε λαμβάνει, είτε εντέλλεται να αποστείλει πίσω στον client είναι κωδικοποιημένο με την μορφή του πρωτοκόλλου SOAP.

## SOAP Server

Αφού εγκαταστήσουμε τον Tomcat Server από το <http://jakarta.apache.org> στο ...\\jakarta-tomcat-7.7.2 μπαίνουμε στο folder ...\\jakarta-tomcat-7.7.2\\bin. Εκεί μέσα βλέπουμε όσα μπορούμε να κάνουμε μέσω command line. Με την εντολή startup, βλέπουμε τον Tomcat να σηκώνεται σε διαφορετικό παράθυρο. Με την shutdown εξαφανίζεται μαζί με το παράθυρό του. Ο Tomcat 'ακούει' στην θύρα 8080. Κτυπώντας με το browser την <http://localhost:8080>, την <http://localhost:8080/soap> καθώς και την <http://localhost:8080/soap/servlet/tpcrouter> βλέπουμε ότι ανταποκρίνεται (στην τελευταία έστω και αρνητικά). Επίσης ότι xyz.html βάλουμε στο ...\\jakarta-tomcat-7.7.2\\webapps\\ROOT, εμφανίζεται με επίσκεψη στο <http://localhost:8080/xyz.html>. Πρέπει βέβαια να καταλάβει ο server την νέα προσθήκη (shutdown και πάλι startup).

Τώρα θα δημιουργήσουμε μία υπηρεσία, την οποία και θα καλέσουμε με SOAP από έναν client. Η παρακάτω class VCatalog (vehicle catalog), υλοποιημένη γύρω από τον Hashtable catalog, επιτρέπει μέσω των μεθόδων της να προσθέσουμε ένα νέο όχημα carModel με τον κατασκευαστή του carMaker (μέθοδος addV), να εύρομε τον κατασκευαστή ενός οχήματος (getMaker) και να πάρουμε όλον τον κατάλογο οχημάτων / κατασκευαστών (listV).

```
package VShop; /*if present this MUST be the first statement,*/
               /* followed by 'imports'*/
import java.util.Hashtable;

public class VCatalog {
    /**The vehicles, by carModel */
    private Hashtable catalog;
```

```

public VCatalog(){
    catalog=new Hashtable();

    //Some content
    catalog.put("Buick","General Motors");
    catalog.put("Mustang","Ford");
    catalog.put("4CV","Citroen");
    catalog.put("Jeep","General Motors");
    catalog.put("Beagle","Volkswagen");
}

public void addV(String carModel, String carMaker) {
    if ((carModel==null)||carMaker==null){
        throw new IllegalArgumentException("carModel and carMaker cannot be null.");
    }
    catalog.put(carModel,carMaker);
}

public String getMaker(String carModel) {
    if (carModel==null){
        throw new IllegalArgumentException("carModel cannot be null.");
    }

    //Return the requested vehicle
    return (String)catalog.get(carModel);
}

public Hashtable listV(){return catalog;}
}

```

### **Κώδικας 9.1.** Η Vcatalog που πραγματοποιεί την υπηρεσία VehicleCatalog

Μεταγλωττίζοντας τον παραπάνω κώδικα με javac, έχουμε μία πλήρη, διαθέσιμη class, άσχετη με RPC και SOAP. Ως εκ τούτου και η παρακάτω διαδικασία εκθέσεως Java classes στο διαδίκτυο μέσω SOAP είναι γενική και εφαρμόσιμη σε ότι προϋπάρχει. Μετά την μεταγλώττιση δημιουργούμε ένα αρχείο jar (Java Archive File) κατά την πρακτική της Java, ως εξής. Τοποθετούμε την VCatalog.class κάτω από ένα folder (VShop) και από τον parent αυτού του folder γράφουμε την εντολή

```
jar cvf V.jar VShop/VCatalog.class
```

Το όνομα του αρχείου V.jar είναι αυθαίρετο. Το .jar θα ανοιχθεί και μέσα του θα ευρεθεί η VCatalog.class, όταν αυτό χρειασθεί. Προσέξτε όμως ότι το folder VShop πρέπει να είναι αυτό που έχει αναφερθεί στην εντολή package VShop (αναγκαστικά η πρώτη εντολή μέσα στο VCatalog.java).

Επόμενο βήμα είναι η κατασκευή του λεγομένου Deployment Descriptor (DD). Ο DD, σε μορφή XML, λαμβάνεται υπ' όψιν από τον SOAP server, την ώρα που αυτός αναλαμβάνει να διαθέσει την υπηρεσία (βλ. παρακάτω 'deploy'). Ο DD ορίζει τα εξής: (α) την URN της υπηρεσίας στο διαδίκτυο, (β) τις εκτιθέμενες μεθόδους της υπηρεσίας (γ) τους handlers οι οποίοι θα αναλάβουν τις διαδικασίες σειριοποίησης και αποσειριοποίησης (serialization / deserialization). Στο δικό μας VCatalogDD.xml που ακολουθεί, το (α) ορίζεται με το attributes id του εξωτερικού στοιχείου service, το (β) με τα methods και class (η class VCatalog μέσα στο package VShop) του στοιχείου provider, ενώ σχετικό με το (γ) θα το δούμε αργότερα. Το τελευταίο εσωτερικό στοιχείο faultListener σχετίζεται και με τις ισχυρές

δυνατότητες του SOAP στην διαχείριση λαθών, τις οποίες και θα δούμε παρακάτω. Όλη η υπηρεσία θα είναι στο διαδίκτυο γνωστή σαν 'VehicleCatalog' (με το URN της). Προσέξτε ότι όλα τα ονόματα των στοιχείων αποκτούν σημασία πού ορίζεται κάτω από το namespace "http://xml.apache.org/xml-soap/deployment", συντομευμένο μέσα στο κάτω κείμενο σαν isd.

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
  id="urn:VehicleCatalog">
  <isd:provider
    type="java" scope="Application" methods="addV getMaker listV">
    <isd:java class="VShop.VCatalog" static="false" />
  </isd:provider>
  <isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>
</isd:service>
```

### Σχήμα 9.5. Το VCatalogDD.xml, Deployment Descriptor της υπηρεσίας VehicleCatalog

Παρατηρούμε επίσης ότι ο Deployment Descriptor έχει ιδιαίτερη μορφή για κάθε διαφορετική (π.χ. Tomcat όπως εδώ) SOAP engine. Τούτο φαίνεται από το εξωτερικό στοιχείο service του οποίου το namespace πρέπει να αναφέρεται στο "http://xml.apache.org/xml-soap/deployment". Στην περίπτωση π.χ. μίας άλλης SOAP engine, όπως η Axis, θα είχαμε διαφορετικό εξωτερικό στοιχείο π.χ. με άλλο namespace (π.χ. "http://xml.apache.org/axis/wsdd"). Τούτο δεν εμποδίζει την διαλειτουργικότητα: ο Deployment Descriptor είναι κάτι εσωτερικό στον server, χωρίς επιρροή στην δομή και σύνταξη των μηνυμάτων SOAP.

Τώρα δε απομένει παρά να θέσουμε την δημιουργηθείσα υπηρεσία στην διάθεση του server. Προς τούτο ρίχνουμε το παραπάνω αρχείο V.jar (πού περιέχει τον κώδικα της υπηρεσίας), μέσα στο folder ...\\jakarta-tomcat-7.7.2\\lib\\common, όπου ο Tomcat αναμένει να ευρίσκει κάθε ζητούμενη υπηρεσία.. Τέλος με την παρακάτω command line (μέσα από το ...\\jakarta-tomcat-7.7.2\\bin) διατάζουμε τον server να εκθέσει την υπηρεσία

```
java org.apache.soap.server.ServiceManagerClient
  http://localhost:8080/soap/servlet/rpcrouter deploy ...\\VCatalogDD.xml
```

Με αυτήν την command line εντολή τρέχουμε την Java class ServiceManagerClient (την ευρισκόμενη στο soap.jar) με παραμέτρους εισόδου (α) το πού θα ενεργήσει, (β) τι θα κάνει (εδώ έκθεση υπηρεσίας – deploy) και (γ) σύμφωνα με ποίον Deployment Descriptor θα εκθέσει την υπηρεσία.

Αν αντί τα (β) και (γ) γράψουμε μόνο την παράμετρο list βλέπουμε ότι τώρα ο server διαθέτει 'στο κοινό' την υπηρεσία urn:VehicleCatalog, όπως ακριβώς του υπέδειξε ο Deployment Descriptor. Με τα (β) και (γ) να είναι query urn:BvehicleCatalog, μας απαντά ο server στο command window με όλον τον Deployment Descriptor. Με τα (β) και (γ) να είναι undeploy urn:VehicleCatalog, μπορούμε να αποσύρομε (undeploy) την υπηρεσία, πράγμα πού μπορεί να πιστοποιηθεί με ένα νέο list. Προσέξτε ότι το soap.jar είναι και αυτό τοποθετημένο στο ίδιο folder με το V.jar της υπηρεσίας μας (τούτο εξηγείται στις διαδικασίες εγκατάστασης – βλ. τέλος σημειώσεων).

Συνεχίζουμε με την υπηρεσία εκτεθειμένη. Αυτή η ίδια, δηλ. η VShop.VCatalog, στο διαδίκτυο γνωστή ως VehicleCatalog, ευρίσκεται έτοιμη να ανταποκριθεί, στο folder ...\jakarta-tomcat-7.7.2\lib\common, χωρίς να έχει ακόμη κληθεί από κανένα.

## SOAP Client

Μεταγλωττίζουμε την παρακάτω VAdderLister.java

```
import java.net.URL;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Vector;
import org.apache.soap.Constants;
import org.apache.soap.Fault;
import org.apache.soap.SOAPException;
import org.apache.soap.rpc.Call;
import org.apache.soap.rpc.Parameter;
import org.apache.soap.rpc.Response;

public class VAdderLister {

    public void addlist(URL url, String model, String manufacturer)
        throws SOAPException{

        //Build the Call object
        Call call = new Call();
        call.setTargetObjectURI("urn:VehicleCatalog");
        call.setMethodName("addV");
        call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);

//----- A D D I N G -----
        System.out.println("Adding vehicle model " + model + " by " + manufacturer);

        // Set up the parameters of the call
        Vector params = new Vector();
        params.addElement(new Parameter("model", String.class, model, null));
        params.addElement(new Parameter("manufacturer", String.class, manufacturer, null));
        call.setParams(params);

        // Invoke the call
        Response response;
        response = call.invoke(url, "");
        if (!response.generatedFault())
            { System.out.println("Successful Vehicle addition");}
        else { Fault fault = response.getFault();
            System.out.println("Error: " + fault.getFaultString());}

//----- L I S T I N G -----
        System.out.println("Listing the new updated catalog");

        // The Call object already in place,
        // updating only the relevant properties of this object
        call.setMethodName("listV");
        call.setParams(null); //the listV method has no input arguments
```

```

// Invoke the call (using the same Response object)

response = call.invoke(url, "");
if (!response.generatedFault())
{
    System.out.println("Successful invocation of the listV method");
    Parameter returnValue = response.getReturnValue();
    Hashtable catalog = (Hashtable)returnValue.getValue();
    Enumeration e = catalog.keys();
    while (e.hasMoreElements()){
        model =(String)e.nextElement();
        manufacturer= (String) catalog.get(model);
        System.out.println(" " + model + " by " + manufacturer);}
    } else {
    Fault fault = response.getFault();
    System.out.println("Error reported: " + fault.getFaultString());
    }
}

public static void main(String[] args) {
    if (args.length !=3)
        {System.out.println("Put url, model and manufacturer as arguments !!");
        return;}

    try {
        // URL for SOAP server to connect to
        URL url = new URL(args[0]);

        // Get values for the new vehicle
        String model = args[1];
        String manufacturer= args[2];

        //Add the new vehicle, then list catalog
        VAdderLister adderlister = new VAdderLister();
        adderlister.addlist(url, model, manufacturer);
    } catch (Exception e) {e.printStackTrace();}
}
}

```

### **Κώδικας 9.2.** Η VAdderLister σαν πελάτης της υπηρεσίας VehicleCatalog

Η VAdderLister.class τρέχεται με (τρεις) παραμέτρους εισόδου. Η πρώτη ορίζει την διεύθυνση, όπου διατίθεται η VehicleCatalog, της οποίας καλούνται εξ αποστάσεως οι μέθοδοι addV και listV. Με

```
java ...\VAdderLister http://localhost:8080/soap/servlet/rpcrouter "Cortina" "Ford"
```

προστίθεται στον κατάλογο που διατηρείται στον server ένα νέο μοντέλο (Cortina) με τον κατασκευαστή του (Ford), πράγμα που διαπιστώνεται ευθύς μετά με τον πλήρη Hashtable που επιστρέφεται από την listV. Ο Hashtable αυτός είναι και η πιο πολύπλοκη δομή που έχουμε μέχρι τώρα δει να διακινείται με SOAP. Για να την δούμε πραγματικά χρησιμοποιούμε τον TCP Monitor.

Στον παραπάνω κώδικα βλέπομε την δημιουργία του Call object, το οποίο μάλιστα χρησιμοποιείται το ίδιο δύο φορές για κλήση δύο διαφορετικών μεθόδων, δηλ. call.setMethodName("addV") και μετά call.setMethodName("listV"). Οι μέθοδοι αυτοί καθορίζονται με τα ονόματα που ορίζει ο DD και βεβαίως είναι ευθύνη του γράφοντος τον client να θέσει στην κλήση τις σωστές παραμέτρους (μέθοδος call.setParams του Call object). Η ίδια η κλήση συντελείται με την μέθοδο invoke του Call object, η οποία επιστρέφει πάντα

ένα αντικείμενο της class Response. Μέσα του ευρίσκεται η τιμή της απόκρισης της υπηρεσίας που εκτελείται στον server, αν βεβαίως τέτοια τιμή υπάρχει (η μέθοδος της υπηρεσίας δεν είναι void). Η επιστρεφόμενη τιμή είναι της class Parameter και εξάγεται από το Response object μέσω της μεθόδου του getReturnValue(). Κατόπιν, με casting, αποτυπώνουμε το αντικείμενο της γενικής class Parameter, σε αντικείμενο της class που γνωρίζουμε ότι είναι, π.χ.

```
Hashtable catalog = (Hashtable)returnValue.getValue();
```

Η returnValue είναι αντικείμενο της class Parameter, εξήχθη με την getValue() από το Response object, και επειδή αναμένουμε αντικείμενο της class Hashtable, την αποτυπώνουμε σε τέτοιο με το πρόθεμα (Hashtable). Το Response object έχει και άλλα χρήσιμα μέσα του, π.χ. αντικείμενο της class Fault, εξαγόμενο με getFault(), κλπ.

## Ανακεφαλαίωση

Τα παραπάνω βήματα συνοψίζονται ως ακολούθως:

- (α) Σηκώσαμε έναν (Tomcat) server.
- (β) Θέσαμε σε συγκεκριμένο subfolder του server την class (σε μορφή jar) που αντιπροσωπεύει την επιθυμητή υπηρεσία με τις διακριτές μεθόδους της.
- (γ) Γράψαμε έναν Deployment Descriptor (σε μορφή XML), ο οποίος μεταξύ άλλων περιέχει το όνομα της υπηρεσίας και τις διακριτές μεθόδους της που θα διατεθούν.
- (δ) Στην πλευρά του server προβήκαμε στην διαδικασία έκθεσης (deploy) της υπηρεσίας, αναφερόμενοι στο αρχείο του Deployment Descriptor. Είδαμε κατόπιν ότι το όνομα της υπηρεσίας αυτής πράγματι δίδεται από τον server (με list).
- (ε) Στην πλευρά του client, κάθε class, μέσω ενός Call object, μπορεί να χρησιμοποιήσει εξ αποστάσεως την υπηρεσία που εξετέθη από τον server. Αυτό πραγματοποιείται με κλήσεις των μεθόδων της, όπως αναφέρονται στον Deployment Descriptor της. Επιστρέφονται στον client όχι μόνον οι τιμές των αποκρίσεων (response) των μεθόδων της υπηρεσίας, αλλά και πιθανές αναφορές σφαλμάτων που πηγάζουν από την εκτέλεσή τους στον server. Η όλη επικοινωνία γίνεται μέσω SOAP πάνω από HTTP.

Παρατηρούμε ότι στην πλευρά του server, η υπηρεσία είναι τελείως γενική με καμία γνώση του ότι θα εξυπηρετήσει κλήσεις εξ αποστάσεως. Επίσης ούτε ο server την γνωρίζει την στιγμή που σηκώνεται και συμβουλευτεί τον Deployment Descriptor ποιους τύπους δεδομένων θα έχει να αποκωδικοποιήσει (κατά την λήψη) και κατόπιν να κωδικοποιήσει (κατά την αποστολή). Αυτό γίνεται δυναμικά. Στην λήψη λαμβάνει την σχετική πληροφορία μέσα από το SOAP μήνυμα. Αφού παραδώσει τις παραμέτρους εισόδου στις ζητούμενες μεθόδους, πληροφορείται από τον byte code (μέσα στο αρχείο .jar) της υπηρεσίας ποιού τύπου θα είναι η απόκριση για να την κωδικοποιήσει και την αποστέλλει πίσω στον client. Στο παράδειγμά μας μάλιστα υπήρχε και η περίπτωση ενός Hashtable. Όλα γίνονται από το SOAP engine που είναι ενσωματωμένο στον server. Στην πλευρά του client έχουμε πράγματι ρητά ζητήσει την συνδρομή της SOAP engine εφόσον στον κώδικά μας έχουμε εισάγει τον απαιτούμενο byte code με τα import org.apache.soap.xyz.

Η σύνταξη των SOAP μηνυμάτων και η μεταφορά τους μέσω HTTP μπορεί να εξετασθεί με τον TCP Monitor.

Είναι προφανές ότι τα παραπάνω μπορούν να γίνουν με τύπους (classes) δεδομένων, τους οποίους πρέπει εκ των προτέρων να γνωρίζει το SOAP engine πώς να αντιμετωπίσει (αποκωδικοποιήσει). Με άλλα λόγια δεν έχουμε ακόμη ξεφύγει ουσιαστικά από τις δυνατότητες του απλού RPC. Είναι επίσης προφανές ότι για την μεταφορά (πάντα μέσω SOAP) πιο πολύπλοκων ή πιο εξειδικευμένων δομών, χρειάζονται να επιστρατευθεί επιπλέον κατάλληλος κώδικας. Αυτό θα το δούμε τώρα για την μεταφορά 'σειριακά μέσω του σύρματος' αντικειμένων της Java.

## Μεταφορά JavaBeans μέσω SOAP

Θα εκτελέσουμε τώρα SOAP RPC όπου θα ανταλλάσσονται αντικείμενα της Java, όχι όμως οποιασδήποτε αυθαίρετης μορφής. Θα πρέπει τα αντικείμενα αυτά να έχουν την δομή των JavaBeans. Τούτο σημαίνει πρακτικά ότι θα πρέπει να υπάρχει πάντα ένας non argument constructor και για κάθε property Xyz του αντικειμένου να διατίθενται οι μέθοδοι getXyz και setXyz. Πιο συγκεκριμένα η παρακάτω VehicleBean class δίνει αντικείμενα JavaBeans σχετικά με το προηγούμενο παράδειγμά μας (μοντέλα αυτοκινήτων, κατασκευαστές, τώρα πρόσθετα και έτος). Όλες οι αναφερθείσες απαιτήσεις εκπληρώνονται στο παρακάτω κώδικα / ορισμό του δικού μας JavaBean, το οποίο ονομάζουμε VehicleBean.

Η πρώτη γραμμή του παρακάτω κώδικα δεν ανήκει στον ορισμό της VehicleBean class. Απαιτείται, κατ' αρχήν, για λόγους που θα εξηγηθούν παρακάτω.

```
package BVShop;
/*if present 'package' MUST be the first statement,*/
/* (possibly) followed by 'import'*/

// The following class follows the structure of a 'Java Bean'

public class VehicleBean {
    /**The properties of the VehicleBean*/
    String VModel;
    String VManufacturer;
    String VYear;

    /** The following non-argument constructor MUST be always present*/
    /** for this class to be a Java Bean*/
    public VehicleBean(){ }

    /** Another constructor CAN also be always present*/
    /** - the following initializes all properties whenever*/
    /** a new object is instantiated (with 'new') - */
    public VehicleBean(String model,String manu,String year){
        this.VModel= model; this.VManufacturer = manu; this.VYear= year; }

    /** get & set methods for all properties MUST be present*/
    public String getVModel(){return VModel;}
    public void setVModel(String model){this.VModel= model;}
    public String getVManufacturer(){return VManufacturer;}
    public void setVManufacturer(String manu){this.VManufacturer= manu;}
    public String getVYear(){return VYear;}
    public void setVYear(String year){this.VYear= year;}

    /** toString is an in-built method for every Java object */
    /** used for outputing an appropriate informative string */
```



```

public String toString(){
    return "" + VModel + " by " + VManufacturer + " (" + VYear + ")";
}

```

### Κώδικας 9.3. Ορισμός VehicleBean

## SOAP Server (με JavaBeans)

Ας υποθέσουμε ότι τώρα θέλουμε να διαθέσουμε μία υπηρεσία, παρόμοια με την προηγούμενη VehicleCatalog. Η παρακάτω παρουσιαζόμενη BVehicleCatalog θα είναι γραμμένη για το παραπάνω VehicleBean και οπωσδήποτε πιο κομψή καθώς θα προσθέτει, αποθηκεύει και επιστρέφει ολόκληρα αντικείμενα (VehicleBean) και θα δεν ασχολείται μεμονωμένα (και 'χύμα') με τις επιμέρους παραμέτρους των. Το σημαντικότερο είναι ότι τώρα ολόκληρα αντικείμενα θα πρέπει να 'σειροποιούνται' πριν την μεταφορά τους 'πάνω από το σύρμα'. Για αυτό θα καταφύγουμε στην βοήθεια της BeanSerializer class, πού διατίθεται μαζί με το Apache SOAP. Έτσι ο κώδικας BVCatalog πού δίδεται παρακάτω είναι αυτός της VCatalog ελαφρά τροποποιημένος. Ο Hashtable αποθηκεύει τώρα αντικείμενα VehicleBean με το όνομα του μοντέλου (Vmodel) σαν κλειδί. Η addV αναμένει σαν παράμετρο εισόδου ένα VehicleBean και δεν επιστρέφει τίποτα – απλώς προσθέτει μία σχετική εγγραφή στο Hashtable. Η getVehicleBean αναμένει σαν παράμετρο εισόδου ένα String και επιστρέφει ένα αντικείμενο VehicleBean. Τέλος η listV() δεν έχει παράμετρο εισόδου και επιστρέφει Hashtable, το οποίο όμως περιέχει και αυτό αντικείμενα VehicleBean.

```

package BVShop;
/*if present 'package MUST be the first statement,*/
/* (possibly) followed by 'import'*/
import java.util.Hashtable;

public class BVCatalog {
    /**The vehicles as Hashtable*/
    /** the car Model is the key for finding entries */
    private Hashtable catalog;

    public BVCatalog(){
        /** the old catalog remains*/
        /** but now the catalog (Hashtable) will contain also beans !!!*/
        catalog=new Hashtable();

        /**Some content - we NOW polulate with some objects!*/
        addV(new VehicleBean("Buick","General Motors","1948"));
        addV(new VehicleBean("Mustang","Ford","1960"));
        addV(new VehicleBean("4CV","Citroen","1950"));
        addV(new VehicleBean("Jeep","General Motors", "1942"));
        addV(new VehicleBean("Beatle","Volkswagen", "1938"));
    }
    /** FIRST METHOD TO BE EXPOSED - addV */
    /** input argument is now a single ('complex') object !*/
    /** nothing is returned*/
    public void addV(VehicleBean vObj) {
        if (vObj == null){
            throw new IllegalArgumentException("The object provided cannot be null.");
        }
        /** entry format for the Hashtable: key,data */
        /** vObj.getVModel gets the 'key' out of the bean object*/

```

```

        catalog.put(vObj.getVModel(),vObj);
        System.out.println("Addition at server side: " + vObj.getVModel());
    }

    /** SECOND METHOD TO BE EXPOSED - getVehicleBean */
    /** input argument is a String */
    /** a ('complex') object is returned */
    public VehicleBean getVehicleBean(String model) {
        if (model==null){
            throw new IllegalArgumentException("carModel cannot be null.");
        }

        //Return the requested vehicle - NOW AS AN OBJECT !!!!
        return (VehicleBean)catalog.get(model);
    }

    /** THIRD METHOD TO BE EXPOSED - listV */
    /** NO input argument - a whole table is returned */
    /** ... , and that with ('complex') objects as entries !!!!*/
    public Hashtable listV(){return catalog;}
}

```

#### Κώδικας 9.4. Η BVCatalog πού πραγματοποιεί την υπηρεσία BVehicleCatalog

Επειδή η BVCatalog.java πρέπει να βρει κατά την μεταγλώττισή της την BVShop/VehicleBean.class (λόγω του package BVShop), πρέπει με την σειρά: (α) να μεταγλωττίσουμε την VehicleBean.java, (β) να ρίξουμε την VehicleBean.class μέσα σε ένα folder ονόματι BVShop (subfolder του folder όπου ευρισκόμαστε), (γ) να μεταγλωττίσουμε την BVCatalog.java, (δ) να ρίξουμε την BVCatalog.class μέσα στο BVShop και (ε) να δημιουργήσουμε το BV.jar με

```
jar cvf BV.jar BVShop/VehicleBean.class BVShop/BVCatalog.class
```

Τώρα το BV.jar περιέχει δύο classes: την VehicleBean.class και την BVCatalog.class. Πάλι ρίχνουμε όπως πριν το BV.jar μέσα στο ...\\jakarta-tomcat-7.7.2\\lib\\common.

Ο νέος Deployment Descriptor είναι ο BVCatalogDD.xml (σε όλα τα νέα έχουμε προσθέσει το 'B' – λόγω 'Beans'). Δεν ενοχλεί το ότι μερικές από τις μεθόδους έχουν τα ίδια ονόματα, καθώς όλες ανήκουν σε υπηρεσία διαφορετικά ονοματισμένη (BVehicleCatalog αντί VehicleCatalog).

```

<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
    id="urn:BVehicleCatalog">

    <isd:provider
        type="java" scope="Application" methods="addV getVehicleBean listV">
        <isd:java class="BVShop.BVCatalog" static="false" />
    </isd:provider>

    <isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>

    <isd:mappings>
        <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            xmlns:x="urn:VBean_xmlns" qname="x:vObj"
            javaType="BVShop.VehicleBean"
            java2XMLClassName="org.apache.soap.encoding.soapenc.BeanSerializer"
            xml2JavaClassName="org.apache.soap.encoding.soapenc.BeanSerializer"/>
    </isd:mappings>

```

</isd:service>

### **Σχήμα 9.6.** Το BVCatalogDD.xml, Deployment Descriptor της υπηρεσίας BVehicleCatalog

Το πραγματικά καινούργιο είναι η ότι τώρα οι μέθοδοι δέχονται (η addV) ή επιστρέφουν (άμεσα η getVehicleBean, έμμεσα η listV) Java αντικείμενα (JavaBeans)! Ως εκ τούτου πρέπει να δοθεί στον server η πληροφορία του πώς θα γίνει η απεικόνιση (map) του ειδικού τύπου (εδώ bean) στην διαδικασία σειριοποίησης / αποσειριοποίησης. Το νέο στοιχείο map μέσα στο mappings κάνει ακριβώς αυτό. Εδώ το mappings περιέχει μία μοναδική 'απεικόνιση', το στοιχείο map με τις ιδιότητές του (attributes). Σαν attributes δίδονται (α) το encodingStyle – πάντα όπως αναγράφεται, (β) ένα αυθαίρετο namespace πού εδώ συντομεύεται απλά με x καθώς και το όνομα πού θα χρησιμοποιούμε για τον τύπο αυτό (μόνο για τους σκοπούς της μεταφοράς του πάνω στο 'σύρμα'), (γ) το πώς ορίζεται ο τύπος αυτός και (δ) το πού θα βρεθούν οι classes πού θα κάνουν τις μετατροπές java-xml και xml-java. Πρόκειται φυσικά για την class BeanSerializer, εφόσον καθώς είπαμε, όταν ανταλλάσσουμε αντικείμενα, αυτά θα είναι υποχρεωτικά της δομής των (Java)Beans. Η BeanSerializer καλύπτει με τις μεθόδους της και τις μετατροπές και για τις δύο διευθύνσεις, δηλ. επιτελεί serialization και deserialization. Για την μεταφορά αντικειμένων πού δεν εμπίπτουν στην δομή των JavaBeans πρέπει να γράψουμε (ή να βρούμε) επί τούτου ιδιαίτερους serializer. Τούτο είναι αποφευκτέο και αξίζει αντ' αυτού να φροντίσουμε να πέσουμε στα χνάρια πού δόθηκαν παραπάνω μετατρέποντας τις δομές μας ώστε να είναι συμβατές με τα JavaBeans. Ο server, συμβουλευόμενος το παραπάνω <isd:mappings>, γνωρίζει τώρα πώς θα χειρισθεί τα συγκεκριμένα JavaBeans, επιπλέον όλων των άλλων δομών του κοινού SOAP-RPC (δηλ. αυτές του Πίνακα 4.1.). Θα δούμε επίσης πώς τα όσα συμφωνούνται στο <isd:mappings> θα αναφερθούν στον client κατά την δημιουργία του Call object. Παρατηρούμε επίσης, ότι ο κώδικας της υπηρεσίας του server (BVCatalog παραπάνω) δεν ασχολείται με το <isd:mappings>. Τούτο βεβαίως ενδιαφέρει την διαδικασία λήψης / αποστολής μέσω SOAP, πού αποτελεί μέρος της γενικής υποδομής του (Tomcat) server προικισμένης με SOAP (λόγω του soap.jar), και όχι της κάθε επιμέρους υπηρεσίας που αυτός εκθέτει.

### **SOAP Client (με JavaBeans)**

Τώρα πρέπει και ο client να χρησιμοποιήσει την ίδια BeanSerializer class. Τούτο διαπιστώνεται στην παρακάτω BVAdderLister.java με τα επιπρόσθετα imports (έναντι της προηγούμενης VAdderLister.java). Ιδιαίτερως αναφέρουμε τις classes SOAPMappingRegistry και BeanSerializer, πού θα ασχοληθούν με τα beans. Το αντικείμενο reg της class SOAPMappingRegistry ('μητρώων απεικονίσεων') θα απεικονίζει οτιδήποτε αναφέρεται μέσα στην υπηρεσία BVehicleCatalog σαν vObj με την βοήθεια του αντικειμένου serializer της class BeanSerializer και σύμφωνα με τον τρόπο κωδικοποίησης πού εμείς προσδιορίζουμε. Έτσι με την μέθοδο mapTypes, καθορίζουμε (α) μία απαιτούμενη σταθερά, (β) ένα QualifiedName πού έχει το πρόθεμα VBean\_xmlns και όνομα vObj, (γ) την ίδια την class πού ορίζει το bean, και τέλος (δ,ε) τους serializer, (de)serializer. Πρόκειται βασικά για πληροφορία στην πλευρά του client, πού πρέπει φυσικά να ταυτίζεται με αυτήν του Deployment Descriptor στην πλευρά του server. Παρατηρούμε εδώ ότι αντιστοιχίσεις πού γίνονται στον server πρωτυποποιημένα μέσω του Deployment Descriptor, πρέπει στον client να επαναληφθούν σε πλήρη αντιστοιχία, αλλά με 'χειροκίνητο' τρόπο. Έτσι η

SOAPMappingRegistry προσφέρει κατά κάποιον τρόπο τον αποδοχέα της πληροφορίας που στον client γράφεται σαν isd:mappings.

Η δημιουργία και χρήση του Call object καθώς και η επεξεργασία του Response object παραμένουν όπως πριν.

```
import java.net.URL;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Vector;
import org.apache.soap.Constants;
import org.apache.soap.Fault;
import org.apache.soap.SOAPException;

import org.apache.soap.encoding.SOAPMappingRegistry;
import org.apache.soap.encoding.soapenc.BeanSerializer;

import org.apache.soap.rpc.Call;
import org.apache.soap.rpc.Parameter;
import org.apache.soap.rpc.Response;

import org.apache.soap.util.xml.QName;

public class BVAdderLister {

    public void addlist(URL url, String model, String manufacturer, String year)
        throws SOAPException{

        // Build the object with the data given by user
        VehicleBean vObj = new VehicleBean(model, manufacturer, year);

        //VehicleBean must now be mapped ... so SOAP can use it
        SOAPMappingRegistry reg = new SOAPMappingRegistry();
        BeanSerializer serializer = new BeanSerializer();
        reg.mapTypes(Constants.NS_URI_SOAP_ENC,
            new QName("urn:VBean_xmlns", "vObj"),
            VehicleBean.class, serializer, serializer);

        //Build the Call object
        Call call = new Call();
        //How to map, where to send, method to call, encoding "style"
        call.setSOAPMappingRegistry(reg);
        call.setTargetObjectURI("urn:BVehicleCatalog");
        call.setMethodName("addV");
        call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);

        //----- A D D I N G -----
        System.out.println("Adding vehicle model " + model + " by " + manufacturer);

        // Set up the parameters of the call
        Vector params = new Vector();

        //in the instructions given to the 'Serializer' - see Depl. Descr.
        params.addElement(new Parameter("vObj", VehicleBean.class, vObj, null));
        call.setParams(params);
    }
}
```

```

// Invoke the call
Response response;
response = call.invoke(url, "");
//We do not expect something back, unless there is a fault!!
if (!response.generatedFault())
    { System.out.println("Server reported NO FAULT while adding vehicle");}
else { Fault fault = response.getFault();
      System.out.println("Server reported FAULT while adding:");
      System.out.println(fault.getFaultString());}

//----- L I S T I N G -----
//We use the same Call Object and change this as appropriate
/* Another method is now called*/
call.setMethodName("listV");
/* NO parameters here !*/
/*(we cannot have a call with arguments as before)*/
call.setParams(null);
// Invoke the call; here we expect something back !!
response = call.invoke(url, "");

/*Extract the value returned in the form of a 'Parameter' Object*/
Parameter returnValue = response.getReturnValue();
/*Cast the 'Parameter' Object onto a Hashtable Object*/
Hashtable catalog = (Hashtable)returnValue.getValue();
Enumeration e = catalog.keys();
while (e.hasMoreElements()) {
    String VModel = (String)e.nextElement();
    VehicleBean vo = (VehicleBean)catalog.get(VModel);
    System.out.println(" " + vo.getVModel() + " by " + vo.getVManufacturer() +
        ", year " + vo.getVYear());
}
}

public static void main(String[] args) {
    if (args.length != 4)
        {System.out.println("Put url, model, manufacturer & year as arguments !!");
        return;}

    try {
        // URL for SOAP server to connect to
        URL urlink = new URL(args[0]);

        // Get values for the new vehicle
        String model = args[1];
        String manufacturer = args[2];
        String year = args[3];
        //Add the new vehicle,
        BVAdderLister adderlister = new BVAdderLister();
        adderlister.addlist(urlink, model, manufacturer, year);
    } catch (Exception e) {e.printStackTrace();}
}
}

```

**Κώδικας 9.5.** Η BVAdderLister πελάτης της υπηρεσίας BVehicleCatalog

Στον παραπάνω κώδικα γίνονται δύο κλήσεις στον server, η πρώτη με στόχο την μέθοδο addV και η δεύτερη την μέθοδο listV. Χρησιμοποιούμε το ίδιο Call object και για τις δύο. Την δεύτερη φορά αλλάζουμε στο αντικείμενο call μόνο όσα διαφέρουν. Εδώ σημειώνουμε, ότι χωρίς το call.setParams(null), θα καλέσουμε πάλι με τις παλαιές παραμέτρους, και θα έχουμε λάθος (signature) στον server, καθόσον η listV δεν έχει παραμέτρους εισόδου. Μετά την επιστροφή από την δεύτερη κλήση, κάνουμε casting σε Hashtable το επιστρεφόμενο, το οποίο είναι πάντα (και στο παράδειγμα χωρίς αντικείμενα) της class Parameter. Τα όσα ακολουθούν μέσω της Enumeration, αποσκοπούν μόνο στο να τυπώσουν τα στοιχεία του πίνακος των οχημάτων και δεν έχουν ιδιαίτερη σημασία.

Στην πλευρά του client δεν χρειάζεται να εμπλακούμε σε διαδικασία δημιουργίας .jar. Μεταγλωττίζουμε το VehicleBean.java στο ίδιο folder (αυτήν την φορά χωρίς την πρώτη εντολή package) και μετά μεταγλωττίζουμε το BVAdderLister.java, όπου βεβαίως χρησιμοποιείται η VehicleBean.class. Μετά τρέχουμε τον client από το ίδιο folder με

```
java BVShop.BVAdderLister http://localhost:8080/soap/servlet/rpcrouter "Smart" "Swatch"
"2001"
```

(αντί 8080, θέτουμε το port#, στο οποίο ακούει TCP Monitor - αν ευρίσκεται σε λειτουργία) και παρατηρούμε το παράθυρο του client, του server, αλλά και του TCP Monitor. Στο τελευταίο διαπιστώνουμε τον όγκο και σημασία της δουλειάς που έγινε από το SOAP, σχεδόν μαγικά !

## Serialization και Deserialization

Εξετάζοντας το folder

...jakarta-tomcat-7.7.2\webapps\soap\WEB-INF\classes\org\apache\soap\encoding\soapenc διαπιστώνουμε ότι υπάρχουν classes για serialization / deserialization όλων των δομών που συναντήσαμε στο απλό και στο SOAP-RPC, π.χ. array, boolean, date, hashtable, bean, .... Ιδιαίτερο ενδιαφέρον έχει η BeanSerializer class. Αυτή εμπεριέχει και τον deserializer. Στις παρακάτω περιγραφόμενες διαδικασίες φαίνεται καθαρά πως χρησιμεύει η δομή που υποθέσαμε (JavaBean) και η οποία απαιτεί την ύπαρξη getXyz, setXYZ και default constructor. Ουσιαστικά το XML μεταφέρει την αναγκαία και ικανή πληροφορία για να επανακατασκευαστεί το αντικείμενο στην άλλη πλευρά.

Ο bean serializer δημιουργεί το XML από το JavaBean ονομάζοντας το εδώ με την βοήθεια του προθέματος "urn:VBean\_xmlns" και του ονόματος "vObj". Μετά περνά από serialization κάθε μεταβλητή (property) που υποστηρίζεται από ένα getXyz (είτε απλή είτε αναδρομικά σαν ένα εσωτερικό JavaBean). Στην άλλη άκρη ο bean deserializer δημιουργεί από το XML το αντικείμενο (JavaBean). Εδώ χρησιμεύει ο default constructor, που είναι πάντα στην Java πλήρως προσδιορισμένος (ίδιο όνομα με την class, κανένα όρισμα). Μετά αποκαθίσταται κάθε μεταβλητή (property) με κλήση της setXYZ. Το τι ακριβώς μεταφέρεται σύμφωνα με τις υποδείξεις μέσα στον κώδικα client και server, αλλά και του Deployment Descriptor, φαίνεται ωραία με την χρήση του TCP Monitor.

## Συνεισφορά του SOAP στην Αναφορά Σφαλμάτων

Εδώ θα δούμε πώς το SOAP μπορεί ακόμη και να επιστρέψει στον client μηνύματα σφάλματος που δημιουργήθηκαν στον server. Ήδη στον client είδαμε ότι από το αντικείμενο response εξάγουμε με `fault.getFaultString()` μία περιγραφή του αντικειμένου fault. Αυτό το fault έρχεται πίσω, εφόσον δημιουργηθεί στον server, και προέρχεται από το διαγνωστικό που είναι σε θέση να παράγει ο server στην προσπάθειά του να ανταποκριθεί στην ζητούμενη υπηρεσία. Μεταφέρεται χάρις στο SOAP. Το SOAP είναι σε θέση να μετατρέπει κάθε αντικείμενο Fault της Java σε XML δομή κατά DOM. Στον παραπάνω Deployment Descriptor έχουμε ήδη επικαλεσθεί στο στοιχείο `<isd:faultListener>` την class `org.apache.soap.server.DOMFaultListener` και η επιθυμητή αναφορά σφάλματος ήδη επιστρέφεται στον client. Δεν έχουμε λοιπόν παρά να την δούμε να μεταφέρεται μέσω του TCP Monitor. Προκαλούμε ένα σφάλμα μέσα στην μέθοδο `addV` του server (που εμφανίζεται στο runtime): απλά απομακρύνουμε την γραμμή `catalog = New Hashtable`. Η `addV` δεν έχει που να εγγράψει και επιστρέφεται μία αναφορά σφάλματος που δημιουργήθηκε στον server.

Στο επιστρεφόμενο SOAP μήνυμα υπάρχουν στον φάκελο `<SOAP-ENV:Fault>` τα `<faultcode>` και `<faultstring>` με περιεχόμενο που δημιουργεί το ίδιο το πρωτόκολλο SOAP, αλλά και το στοιχείο XML `<stacktrace>` με περιεχόμενο την πλήρη αναφορά σφάλματος (stack trace) της Java. Παρατηρούμε επίσης ότι τα παραπάνω αποτελούν και άλλο παράδειγμα ειδικής κωδικοποίησης και αποστολής 'πάνω στο σύρμα' (με serialization / deserialization) αντικειμένου Java. Πριν είχαμε οποιοδήποτε JavaBean και τώρα οποιοδήποτε Fault object. Αν στην πλευρά του client, με περαιτέρω κώδικα, επεξεργαζόμαστε και εκτυπώνουμε το περιεχόμενο text του `<stacktrace>` element, θα βλέπαμε την πλήρη αναφορά σφαλμάτων που θα εξέδιδε η ίδια η γλώσσα (εδώ η Java) αν η εκτέλεση της ζητηθείσης υπηρεσίας γινόταν τοπικά. Έτσι εξομοιώνουμε και σε αυτό το σημείο την τοπική κλήση με την εξ αποστάσεως (RPC).

Επιπλέον η αναφορά σφάλματος ξεφεύγει από την δομή του SOAP που γνωρίσαμε να ασχολείται με την κωδικοποίηση (serialization / deserialization) μεταβλητών, είτε εισόδου σε, είτε εξόδου από κλήσεις μεθόδων. Αν θεωρήσουμε το ίδιο το σφάλμα σαν ένα γενικό κείμενο XML, μπορούμε να πούμε ότι στην περίπτωση αυτή έχουμε μαζί με το SOAP-RPC (Remote Procedure Call) και SOAP Messaging, δηλαδή την μεταφορά αυτούσιων μηνυμάτων (πάντα κειμένων XML) μέσω SOAP. Με την περίπτωση αυτή στην γενικότητά της ασχολείται η επομένη παράγραφος.

## SOAP Messaging

Είδαμε πώς το SOAP μπορεί να χρησιμοποιηθεί μέσα στα πλαίσια του RPC (SOAP-RPC) εξελιγμένο και δυνατότερο του απλού XML-RPC. Σε αυτό της τον ρόλο η λεγόμενη 'SOAP engine' δρα και στις δύο πλευρές και:

- σειριοποιεί κλήσεις μεθόδων σε πακέτα SOAP
- αποσειριοποιεί πακέτα SOAP σε κλήσεις κατά τις συμβάσεις της Java

Τώρα, στα πλαίσια του SOAP Messaging θα δούμε πώς η ίδια ‘SOAP engine’ μπορεί να χρησιμεύσει και για την γενική μεταφορά πληροφορίας σε XML άσχετης (κατ’ αρχήν) με τις εξ αποστάσεως κλήσεις. Δρώντας πάλι και στις δύο πλευρές (αποστολέας και λήπτης) η SOAP engine θα:

- περιτυλίγει κείμενα XML μέσα σε πακέτα SOAP (κατά την αποστολή)
- εξάγει κείμενα XML μέσα από πακέτα SOAP (κατά την λήψη)
- θέτει αιτήσεις (SOAP requests) και διεκπεραιώνει τις αποκρίσεις (SOAP responses)
- δέχεται αιτήσεις (SOAP requests) και επιστρέφει αποκρίσεις (SOAP responses)

Πέραν του ρόλου της SOAP engine παρατηρούμε ότι μία εφαρμογή στον δέκτη ενός μηνύματος (SOAP Message) θα πρέπει να μπαίνει στην δομή ενός κειμένου XML, περιτυλιγμένου εντός του SOAP μηνύματος. Τούτο έρχεται σε αντίθεση με την ευκολία που είχαμε στην συγγραφή, για παράδειγμα, της υπηρεσίας BVCatalog για τον προηγούμενο SOAP-RPC server. Η BVCatalog ήταν τελείως ουδέτερη ως προς το RPC και το SOAP, πιο συγκεκριμένα ως προς την διαχείριση του μηνύματος που μετέφερε τις παραμέτρους είτε εισόδου είτε απόκρισης. Τώρα μία τυπική μέθοδος στα πλαίσια του SOAP Messaging θα πρέπει να επιδρά επί ενός αντικειμένου request και response έχοντας την μορφή:

```
public void methodName(Envelope env, SOAPContext req, SOAPContext res)
    throws java.io.IOException, javax.mail.MessagingException
```

Τα δύο παραπάνω αντικείμενα, req και res αντίστοιχα, ανήκουν στην class SOAPContext. Εδώ μπορούμε να παρατηρήσουμε και τις αναλογίες με τον μηχανισμό ερωταποκρίσεων στο απλό http και τις classes HttpServletRequest, HttpServletResponse response. Επιπλέον η παραπάνω μέθοδος θα (μπορεί να) αναφέρεται αμέσως στον φάκελο (envelope) του SOAP και να αντιμετωπίζει η ίδια τα ενδεχόμενα σφάλματα είτε επικοινωνίας (IOException), είτε του ίδιου του πρωτοκόλλου ή μηνύματος SOAP (MessagingException). Επιπλέον υπάρχει ο περιορισμός να φέρει το ίδιο όνομα με την ρίζα (root element) του περιεχομένου (SOAP message content). Παρατηρούμε επίσης ότι, σε αντίθεση με τα παραδείγματα του RPC, η ίδια μέθοδος δεν επιστρέφει τίποτα (void). Το αποτέλεσμα της επεξεργασίας του κειμένου που έρχεται με το αντικείμενο req, μπορεί να καταλήγει στο αντικείμενο res. Στην περίπτωση αυτή έχουμε πάλι την έννοια του server. Εδώ ο server δέχεται ένα κείμενο XML, το μετατρέπει και, πάλι σαν XML, το επαναστέλλει στον αρχικό αποστολέα ή σε κάποιον τρίτο.

Ας υποθέσουμε ότι το σταλμένο από τον client και εισερχόμενο στον server μήνυμα είναι ουσιαστικά το γνωστό μας vehicles, σαν κείμενο XML. Για απλούστευση του κώδικα του client, το παρακάτω vehiclesSOAP.xml είναι ήδη φτιαγμένο όπως πρέπει να αποσταλεί στον server, δηλ. πλήρες μήνυμα SOAP και βεβαίως εμπεριέχει το vehicles.xml. Τούτο αποτελεί αυτούσιο το Body της SOAP Envelope με την σημαντική προσθήκη ενός απαιτούμενου namespace, που αναφέρεται στο όνομα που θα δώσουμε στην υπηρεσία SOAP Messaging, που κατασκευάζουμε (urn:Messaging\_Service).

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
<s:Body>
<vehiclesSOAP xmlns="urn:Messaging_Service">
  <cars ...
```

----- το περιεχόμενο του vehicles.xml όπως πριν -----



```
</lories>  
</vehiclesSOAP>  
</s:Body>  
</s:Envelope>
```

### Σχήμα 9.7. Το vehicles.xml μέσα στο SOAP Message

Ο παρακάτω κώδικας είναι του client. Διαβάζεται ένα αρχείο .xml όπως το παραπάνω, και ένα url από το command line. Από το αρχείο σε μορφή κειμένου η unmarshall φτιάχνει ένα αντικείμενο ενν της class Envelope. Γενικά ο όρος ‘marshall’ υπονοεί την μετατροπή ενός αντικείμενου σε κάποια συμφωνημένη σειριακή μορφή και ο ‘unmarshall’ το αντίθετο. Εδώ έχουμε το περίεργο ότι ενώ είχαμε ένα κείμενο, το μετατρέψαμε σε μία δομή .xml, για να σταλεί πάλι σαν κείμενο ‘πάνω στο σύρμα’. Βεβαίως τούτο έγινε εδώ, απλά και μόνο για να δειχθεί, ότι πράγματι το SOAP θα κάνει αυτή την δουλειά για να μεταφέρει το μήνυμα από τον client στον server: πράγματι το αντικείμενο msg της org.apache.soap.messaging.Message θα φροντίσει να σταλεί το αντικείμενο ενν στο σωστό url και εκεί μέσα να απευθυνθεί στην σωστή υπηρεσία.

```

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.net.URL;
import javax.xml.parsers.DocumentBuilder;
// for SAX & DOM
import org.w3c.dom.Document;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
/// for SOAP
import org.apache.soap.Constants;
import org.apache.soap.Envelope;
import org.apache.soap.SOAPException;
import org.apache.soap.messaging.Message;
import org.apache.soap.transport.SOAPTransport;
import org.apache.soap.util.xml.XMLParserUtils;

public class MessagingClient {
    // the following method covers the .xml file into a SOAP message
    // and the sends this to serviceURL
    public void sender (URL serviceURL, String msgFilename)
        throws IOException, SAXException, SOAPException {

        // Handle the XML document supplied in the input file
        FileReader readerObj = new FileReader(msgFilename);
        DocumentBuilder builder = XMLParserUtils.getXMLDocBuilder();
        Document doc = builder.parse(new InputSource(readerObj));

        if (doc == null) {
            throw new SOAPException(Constants.FAULT_CODE_CLIENT, "Error in XML parsing");
        }

        // Create Message Envelope
        Envelope env = Envelope.unmarshall(doc.getDocumentElement());

        // Send the Message
        Message msg = new Message();
        msg.send(serviceURL, "urn:Messaging_Service", env);

        // Receive response (e.g. to the previous message sent)
    }

    public static void main(String args[]) {
        if (args.length !=2) {
            System.out.println("Please give the url and the .xml message file");
            return; }

        try {
            URL serviceURL = new URL(args[0]);

            MessagingClient messagingClientObj = new MessagingClient();
            messagingClientObj.sender(serviceURL, args[1]);

        } catch (Exception e) {e.printStackTrace();}
    }
}

```

**Κώδικας 9.6.** Δημιουργία μηνύματος SOAP και αποστολή του από τον client

Ο κώδικας του server αποτελείται ουσιαστικά από την μέθοδο vehiclesSOAP, η οποία πρέπει να έχει το ίδιο όνομα με το όνομα της ρίζας του μηνύματος που έφθασε. Ο κώδικας θέλει να δείξει ότι πράγματι εισάγεται το vehicles σε μορφή DOM στην μνήμη του δέκτη (server), καθόσον θα μπορούμε να περιηγηθούμε μέσα σε αυτό, να τυπώσουμε την τιμή συγκεκριμένων attributes, να αλλάξουμε την τιμή άλλων και να επαναστείλομε πίσω το μέρος του μεταβληθέντος vehicles. Η λήψη και προετοιμασία για επεξεργασία επιτελείται με τον παρακάτω κώδικα.

```
public void vehiclesSOAP(Envelope env, SOAPContext req, SOAPContext res)
    throws java.io.IOException, javax.mail.MessagingException {
    // Get the vehicles element as the first body entry of the SOAP message;
    // here there is one body element - in general could be many!
    // so in general we put these in a vector,
    // then iterate though it and final get the XML in DOM into memory!
    Vector bodyEntries = env.getBody().getBodyEntries();
    Element vehicles_in_DOM = (Element)bodyEntries.iterator().next();
    // .....}
```

### **Κώδικας 9.7.** Κώδικας για την εξαγωγή σε DOM, του κειμένου XML που έφθασε με SOAP

Με το συμβάν της έλευσης του μηνύματος καλείται αυτόματα η vehiclesSOAP, η οποία και μας δίδει το αφιχθέν SOAP Envelope μέσω του αντικειμένου env. Αυτή δεν επιστρέφει τίποτα (void), καθόσον έχουμε μήνυμα και όχι RPC. Με env.getBody().getBodyEntries() αποκτώμε όλα τα BodyEntries elements. Αυτά θα μπορούσαν να ήταν πολλά: είδαμε ότι ένα Envelope μπορεί να έχει περισσότερα του ενός block. Ένας πιο πολύπλοκος server, θα μπορούσε να ασχοληθεί με πολλά κείμενα XML λαμβανόμενα μέσα στο ίδιο μήνυμα SOAP. Η env είναι αντικείμενο της class Envelope (έχουμε κάνει import org.apache.soap.Envelope), η μέθοδος getBody() του αντικειμένου env επιστρέφει ένα αντικείμενο της class Body (ΔΕΝ έχουμε κάνει import org.apache.soap.Body), ενώ τελικά η getBodyEntries() class επιστρέφει από το αντικείμενο αυτό ένα της class Vector. Χρησιμοποιείται Vector, για την γενική περίπτωση των πολλών στοιχείων Body. Εδώ όμως έχουμε ένα, άρα και 'πρώτο', το οποίο μας το βγάζει από το bodyEntries το iterator σε ένα και μοναδικό βήμα. Μετά το casting, από γενικό αντικείμενο / στοιχείο της Vector αποκτώμε αντικείμενο (XML) Element. Το vehicles είναι πλέον στην μνήμη σε μορφή DOM με το όνομα vehicles\_in\_DOM. Από εδώ και πέρα ακολουθεί η επεξεργασία του, όπως μάθαμε στη ενότητα για το DOM, με την διαφορά ότι εδώ εργαζόμαστε σε Java και όχι σε JavaScript. Το τι γίνεται, εξηγείται εύκολα με αναφορά στο DOM Java API.

Επιλέγουμε να στείλουμε πίσω μόνο το μέρος του κειμένου, όπου έγιναν αλλαγές. Χρησιμοποιούμε το ίδιο αντικείμενο bodyEntries που φτιάξαμε για την υποδοχή του μηνύματος, και αφού το καθαρίσουμε, του προσθέτουμε το κομμάτι προς αποστολή, δηλ. bodyEntries.add(jeeps\_in\_DOM). Και το αντικείμενο env εξακολουθεί να υπάρχει (παππούς του bodyEntries με νέο εγγονάκι!). Στην μέθοδό του marshall ορίζουμε ποιος θα γράψει, δηλ. το writerObj. Θα έπρεπε να ορίσουμε και ένα XMLJavaMappingRegistry, αν είχαμε ειδικούς τύπους να μετατραπούν σε XML όπως στο XMP-SOAP. Το null σημαίνει ότι τέτοια περίπτωση δεν υπάρχει στο SOAP Messaging (η marshall / unmarshall είναι κοινή και στα δύο περιβάλλοντα). Τέλος η setRootPart του αντικειμένου res (για response), που μας δόθηκε με την κλήση της vehiclesSOAP, καθορίζει το τι θα φύγει και τι μορφής είναι αυτό που θα σταλεί (writerObj.toString(),"text/xml"). Δίδεται τώρα ο πλήρης κώδικας του server.

```

package MSS;
import java.io.IOException;
import java.io.StringWriter;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Enumeration;
import java.util.Vector;
import java.util.List;
import java.util.Iterator;
import javax.mail.MessagingException;

//for SOAP
import org.apache.soap.Constants;
import org.apache.soap.Envelope; /*New!!! for messaging */
import org.apache.soap.Fault;
import org.apache.soap.SOAPException;
import org.apache.soap.encoding.SOAPMappingRegistry;
import org.apache.soap.encoding.soapenc.BeanSerializer;
import org.apache.soap.rpc.Call;
import org.apache.soap.rpc.Parameter;
import org.apache.soap.rpc.Response;
import org.apache.soap.rpc.SOAPContext;
import org.apache.soap.util.xml.QName;

// DOM related
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

public class MessagingServer {
    public void vehiclesSOAP(Envelope env, SOAPContext req, SOAPContext res)
        throws java.io.IOException, javax.mail.MessagingException {

        System.out.println(".... inside MessagingServer's vehiclesSOAP method");

        // Get the vehicle element as the first body entry of the SOAP message;
        // here there is one body element - in general could be many!
        // so in general we put these in a vector,
        // then iterate though it and finally get the XML in DOM form into memory!

        Vector bodyEntries = env.getBody().getBodyEntries();
        Element vehicles_in_DOM = (Element)bodyEntries.iterator().next();
        System.out.println(vehicles_in_DOM.getTagName() +
            ".... first body entry extracted out of SOAP");

        // DOM processing in Java for transforming the XML

        // A test that we actually have access to vehicles through DOM:
        // Go to all vehicles>jeeps>jeep and print all colour attributes
        Element jeeps_in_DOM =
            (Element)vehicles_in_DOM.getElementsByTagName("jeeps").item(0);
        NodeList jeepList = jeeps_in_DOM.getElementsByTagName("jeep");

        for (int i=0, len=jeepList.getLength(); i<len; i++){
            Element jeep = (Element)jeepList.item(i);
            System.out.println("Printing jeep colour from within DOM: "
                + jeep.getAttribute("colour"));
            jeep.setAttributeNode("user").setValue("military");
            System.out.println("Jeep 'user' changed - check that in the response");
        }
    }
}

```

```

        // Build the response SOAP message
        bodyEntries.clear(); // reuse of same Vector object
        bodyEntries.add(jeeps_in_DOM);
        StringWriter writerObj = new StringWriter();
        env.marshall(writerObj,null);

        // Send envelope back to client
        res.setRootPart(writerObj.toString(),"text/xml");
    }
}
}

```

### Κώδικας 9.8. Ο server δέχεται, μετατρέπει και επαναστέλλει το μήνυμα SOAP

Και τέλος μένει το MessagingServiceDD.xml, δηλ. ο Deployment Descriptor. Ορίζουμε μία μέθοδο και μόνο, την vehiclesSOAP, η οποία και θα ασχοληθεί με το μήνυμα. Η java class δίδεται πάλι μέσω .jar (βλ. παρακάτω) και το id είναι εκείνο, που αναφέρθηκε στο namespace πού εισήχθη στο κορυφαίο στοιχείο vehiclesSOAP. Φυσικά δεν υπάρχει πληροφορία για κωδικοποίηση ειδικών τύπων.

```

<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
    id="urn:Messaging_Service">
    <isd:provider
        type="java" scope="Application" methods="vehiclesSOAP">
        <isd:java class="MSS.MessagingServer" static="false" />
    </isd:provider>
    <isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>
</isd:service>

```

### Σχήμα 9.8. Το MessagingServiceDD.xml, Deployment Descriptor της υπηρεσίας πού παρέχει ο MessagingServer

Τώρα είμαστε έτοιμοι για δοκιμή. Χρειάζεται και πάλι .jar με την γνωστή διαδικασία,  
 jar cvf MServer.jar MSS/MessagingServer.class

Για deployment της υπηρεσίας έχουμε κατά τα γνωστά  
 java org.apache.soap.server.ServiceManagerClient  
 http://localhost:8080/soap/servlet/rpcrouter **deploy** C:\MessagingServiceDD.xml

Σηκώνουμε και τον TCP Monitor, καθόσον είναι απαραίτητος για να δούμε το μήνυμα client προς server και πίσω. Για να τρέξουμε τον client, πρέπει τώρα να του δώσουμε την διεύθυνση του messengerouter και όχι του rpcrouter, όπως πριν!

```

java MessagingClient http://localhost:8081/soap/servlet/messengerouter
C:\vehiclesSOAP.xml

```

Δεν προβλέψαμε να δούμε τίποτα στη μεριά του client, ούτε αυτός επεξεργάζεται το μήνυμα πού επιστρέφεται. Βλέπομε όμως όσα γίνονται στο παράθυρο του server, και βεβαίως στον TCP Monitor το αποστέλλόμενο και επιστρεφόμενο από τον server μήνυμα.

Το παραπάνω παράδειγμα, με επαναποστολή όχι αναγκαστικά στον αρχικό αποστολέα, θα μπορούσε να αποτελέσει την βάση ενός ενδιαμέσου πράκτορα λήψης, επεξεργασίας και

προώθησης εντολών / παραγγελιών από πελάτες (clients, σαν αρχικοί αιτούντες) σε ένα τελικό αποδέκτη (ο προμηθευτής). Μία επέκταση της class MessagingServer, ως ενδιαμέσος, θα επενεργεί μόνο σε μηνύματα SOAP XML αποτελώντας κατ' εξοχήν περίπτωση SOAP Messaging και όχι SOAP-RPC.

## 10. Ανακεφαλαίωση Πρωτοκόλλων Εφαρμογών

Τα ευρέως αποδεκτά και προτυποποιημένα πρωτόκολλα είναι πάντα text based, δηλαδή βασίζονται στην μετάδοση (εκτυπώσιμων) χαρακτήρων. Σε αντίθεση binary πρωτόκολλα, αν και οικονομικότερα για την μετάδοση αφορούν παρωχημένες επιλογές ή και λύσεις που αφορούν κλειστά περιβάλλοντα και επί μέρους προϊόντα. Πάντως η ενσωμάτωση τέτοιων μερικών λύσεων μέσα σε κείμενα XML δεν αποκλείεται (χρήση κόμβων με CDATA). Εξ' άλλου οι βασικές λειτουργίες του browser αφορούν την εμφάνιση της html που είναι και αυτή text based.

Οι δομές XML, είτε χρησιμοποιούμενες για RPC είτε για XML Messaging είναι και αυτές ειδικότερης δομής text based περιεχόμενα ή και αυτόνομα μηνύματα πρωτοκόλλων, όπως το SOAP. Άρα το XML είναι η βάση πρόσβασης σε προσφερόμενες λειτουργίες (RPC), είτε για κωδικοποίηση και μετάδοση δομημένης πληροφορίας, όπως π.χ. μία προσφορά ή ένα τιμολόγιο.

### Τα GET και POST του http

Τα GET και POST είναι τα μηνύματα του πρωτοκόλλου http (Hypertext Transfer Protocol) για την κωδικοποίηση και μετάδοση παραμέτρων, όπως τα ζεύγη ονομασίας/τιμής (name/value pairs), μαζί με την συνακόλουθη σημαντική πληροφορία κατά την αίτηση (request) πληροφορίας ή υπηρεσίας. Μία σειρά γραμμών από 'http request headers' στην επικεφαλίδα τους ορίζει, μεταξύ άλλων, τι αναζητεί ο πελάτης (client) από τον εξυπηρετητή (server). Ο τελευταίος αποκρίνεται, με μία σειρά γραμμών από 'http response headers' ακολουθούμενες, σε περίπτωση επιτυχίας, από την ζητούμενη πληροφορία.

Το http GET περνά τις παραμέτρους σαν 'url-encoded text', το οποίο επικολλάται πίσω από URL του server, ο οποίος εντέλλεται να επεξεργασθεί την αίτηση (request). Αυτό το 'url-encoded text' είναι γνωστό και σαν 'query string'.

Το http POST κωδικοποιεί τις παραμέτρους με ίδιο τρόπο όπως το GET (url-encoded text). Αντί όμως να μεταφέρονται σαν μέρος του URL, τα ζεύγη ονομασίας/τιμής (name/value pairs) περνούν στον server σαν σώμα του μηνύματος HTTP request.

### Το SOAP πάνω από http

Το SOAP είναι ένα απλό και ελαφρύ πρωτόκολλο βασισμένο στο XML για την ανταλλαγή δομημένης πληροφορίας πάνω στο web. Ο σχεδιασμός του SOAP έχει γίνει με την απλότητα ως πρωταρχική, παρέχοντας έτσι το ελάχιστο σε λειτουργικότητα. Το πρωτόκολλο ορίζει ένα πλαίσιο μηνύματος χωρίς πληροφορία σημαντική για τα επίπεδα μεταφοράς και εφαρμογής. Για τούτο και είναι δομημένο (modular) και εύκολα επεκτάσιμο.

Με το να ταξιδεύει πάνω από τα κοινά, προτυποποιημένα πρωτόκολλα μεταφοράς, το SOAP χρησιμοποιεί την ανοικτή αρχιτεκτονική του Internet και αποκτά αξία για χρήση από οποιοδήποτε σύστημα είναι ικανό να υποστηρίξει τα πλέον βασικά πρότυπα του Internet. Η υποδομή που απαιτείται για την υποστήριξη μιας υπηρεσίας Web XML και συμβατής με το SOAP, είναι μάλλον απλοϊκή. Είναι όμως και αρκετά ισχυρή καθώς, προσθέτοντας σχετικώς ολίγα στην υπάρχουσα υποδομή, επιτρέπει την πρόσβαση στην υπηρεσία σε

παγκόσμια κλίμακα. Οι προδιαγραφές του πρωτοκόλλου SOAP (<http://www.w7.org/TR/soap>) αποτελούνται από τέσσερα κύρια μέρη, που περιέχουν:

(α) έναν υποχρεωτικό επεκτάσιμο φάκελο (envelope) για την ενθυλάκωση των δεδομένων. Ο φάκελος SOAP ορίζει ένα μήνυμα SOAP και είναι η βασική ανταλλασσόμενη μονάδα μεταξύ επεξεργαστών μηνυμάτων SOAP. Τούτο είναι και το μόνο υποχρεωτικό μέρος της προδιαγραφής.

(β) προαιρετικούς κανόνες κωδικοποίησης των δεδομένων. Με αυτούς παρουσιάζονται τύποι δεδομένων όπως αυτοί ορίζονται στις εφαρμογές καθώς και ένα ενιαίο μοντέλο για σειριοποίηση μοντέλα δεδομένων χωρίς σύνταξη.

(γ) μία μορφή ανταλλαγής μηνυμάτων του τύπου RPC-style (request/response). Κάθε μήνυμα SOAP είναι μία μετάδοση μονής κατεύθυνσης. Αν και οι ρίζες του SOAP ευρίσκονται στο RPC, το SOAP δεν περιορίζεται στο να είναι ένας μηχανισμός request/response. Όπως θα δούμε στην συνέχεια, οι υπηρεσίες XML web συχνά συνδυάζουν μηνύματα SOAP για να πραγματοποιήσουν την μορφή αυτή, αλλά το ίδιο το SOAP δεν επιβάλλει αυτήν την συγκεκριμένη μορφή ανταλλαγής μηνυμάτων.

(δ) ένα δέσιμο μεταξύ SOAP και HTTP. Και αυτό το μέρος είναι προαιρετικό. Μπορούμε να χρησιμοποιήσουμε το SOAP σε συνδυασμό με κάθε μηχανισμό μεταφοράς (π.χ. SMTP, FTP, ή και ένα floppy disk), αρκεί αυτός να είναι ικανός να μεταφέρει την SOAP envelope.

## 11. Αντιπαραβολή Εφαρμογών και Υπηρεσιών

Ένας web server δέχεται μηνύματα / αιτήσεις (request, συνήθως GET ή POST) και είναι εις θέσιν να απαντήσει με μηνύματα / αποκρίσεις (response). Η σύνταξη των μηνυμάτων και οι κανόνες της απλής αυτής αλληλεπίδρασης αποτελούν το πρωτόκολλο http. Πάνω σε αυτή την κοινή και απλούστατη βάση κτίζονται για διαφορετικούς σκοπούς διαφοροποιημένοι τρόποι αλληλεπίδρασης. Διακρίνομε λοιπόν περιπτώσεις, όπου ένας εξυπηρετητής (server)

(α) εξυπηρετεί τους 'πελάτες' του (clients) αποστέλλοντάς τους σελίδες για παρουσίαση σε html. Απαιτείται αναγκαστικά η χρήση του πρωτοκόλλου http, και ο server ανταποκρίνεται με μήνυμα response στα μηνύματα request (GET ή POST) του client. Στο εσωτερικό του, ο server είτε έχει στατικές σελίδες αποθηκευμένες, είτε τις κατασκευάζει την ώρα που ζητούνται (με χρήση servlet) είτε χρησιμοποιεί μία μεικτή τεχνολογία (δυναμικές σελίδες). Αποθηκεύει σελίδες (στατικές ή δυναμικές) τις οποίες αποστέλλει στον πελάτη. **Ο τελικός 'πελάτης' είναι ο (άνθρωπος καθήμενος μπροστά στον) browser.** Από τον server φεύγει πάντοτε κάτι παρουσιάσιμο, άρα κωδικοποιημένο σε html. Το 'παρουσιάσιμο' περιλαμβάνει και 'στοιχεία ελέγχου' GUI (Graphical User Interface) elements ή controls που επιτρέπουν 'έλεγχο' από τον χρήστη. Όπως είδαμε, παραδείγματα τέτοιων 'GUI controls', με τις ενέργειες τους, είναι τα

<INPUT TYPE="text"...>	δηλ. πλαίσιο κειμένου,
<INPUT TYPE="radio"...>	δηλ. κουμπιά αποκλειστικής επιλογής
<INPUT TYPE="submit"...>	δηλ. αποστολή των δεδομένων πίσω στον server,
<INPUT TYPE="reset"...>	δηλ. κατάργηση όλων των επιλογών

και πολλά άλλα που βασίζονται και εμπλέκουν γνωστές και προϋποτιθέμενες δυνατότητες του browser να τα παρουσιάσει στον ανθρώπινο χρήστη και να ανταποκριθεί αναλόγως, ώστε οι επιλογές του να αποσταλούν πίσω στον server.



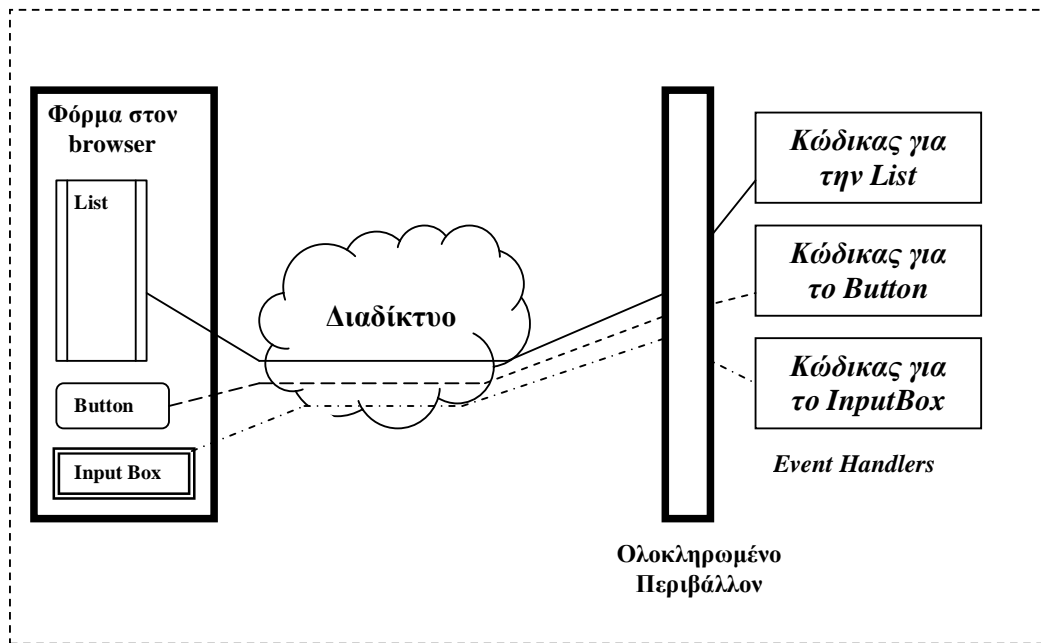
Πάνω στην βάση αυτή κτίζονται οι λεγόμενες **εφαρμογές web (web applications)**. Λέγονται εφαρμογές διότι εμπλέκουν και περιλαμβάνουν και τον τελικό χρήστη και χαρακτηρίζονται με 'web', καθόσον κτίζονται πάνω στο http. Πραγματοποιούν το λεγόμενο **interactive web**, παρέχοντας την αλληλοδραστικότητα με τον χρήστη.

(β) εξυπηρετεί τους 'πελάτες' του (clients) αποστέλλοντάς τους αποτελέσματα κλήσεων σε συναρτήσεις που αυτός διαθέτει για ευρεία χρήση με πρόσβαση μέσω του διαδικτύου. Χρησιμοποιείται πάλι το πρωτόκολλο http, το οποίο μεταφέρει το 'ερώτημα' και την 'απάντηση'. Το 'ερώτημα' προσδιορίζει την επιθυμητή υπηρεσία, μέθοδο μέσα σε αυτή, και τις παραμέτρους κλήσης. Η 'απάντηση' είναι το αποτέλεσμα που επιστρέφει η εκτέλεση της μεθόδου στο εσωτερικό του server. Ερώτημα και απάντηση κωδικοποιούνται όπως είδαμε σε XML, μέσα σε SOAP, μέσα σε σώμα (body) του http. Το μεταφερόμενο μέσω http **δεν** είναι κωδικοποιημένο σε html, εφόσον **δεν** προορίζεται για παρουσίαση μέσω browser. **Ο τελικός 'πελάτης' (client) είναι μία διαδικασία** κατάλληλα σχεδιασμένη (συνήθως από άλλους) σε κάποιο άλλο απομακρυσμένο μηχάνημα και όχι ένας γενικός browser. Βρισκόμαστε εδώ στην περίπτωση μίας **υπηρεσίας ιστού (web service)**, καθόσον οι δυνατότητες που προσφέρει ο server είναι γενικής χρήσεως και φτιαγμένες χωρίς την γνώση του πόσοι, ποίοι και από που (σύστημα, λειτουργικό, γλώσσα, συγκεκριμένο πρόγραμμα) θα τις χρησιμοποιήσουν μέσω του διαδικτύου. Οι υπηρεσίες αυτές πάλι χαρακτηρίζονται με 'web', καθόσον κτίζονται και εδώ πάνω σε http. Πραγματοποιούν το λεγόμενο **programmable web**, επιτρέποντας την ανάπτυξη χαλαρά συνδεδεμένων καταναμημένων εφαρμογών. Στο πλαίσιο αυτό, οι προσφερόμενες υπηρεσίες και μέθοδοι αυτών πρέπει να διαφημίζονται / ανακαλύπτονται (service advertising) και ο τρόπος χρήσης τους να τεκμηριώνεται (service description) με κάποιο τρόπο. Έχουμε εδώ σχετικά πρότυπα το UDDI (Universal Description, Discovery and Integration) και το WSDL (Web Services Description Language). Επειδή η αλληλεπίδραση βασίζεται σε μηνύματα κωδικοποιημένα σε XML, και μάλιστα μέσα σε φακέλους SOAP, μιλάμε κυρίως για **XML web services**.

Ακολούθως θα δοθεί εντελώς σχηματικά τι περιλαμβάνουν οι εφαρμογές και τι οι υπηρεσίες ιστού σαν κύρια χαρακτηριστικά και σαν απαραίτητα συμπληρώματα για τον σχεδιασμό και την λειτουργία τους πάνω στο διαδίκτυο.

## Περιβάλλοντα Εφαρμογών Ιστού

Η γενική ιδέα του κάθε ολοκληρωμένου περιβάλλοντος για την ανάπτυξη μίας εφαρμογής web φαίνεται στο παρακάτω Σχήμα 11.1. Αριστερά φανταζόμαστε μία φόρμα φορτωμένη στον browser, με τα στοιχεία ελέγχου (GUI controls) πάνω σε αυτή να δέχονται τις ενέργειες του ανθρώπινου χρήστη. Η αλληλοδραστικότητα (interaction) απαιτεί ότι σαν απόρροια κάθε τέτοιας ενέργειας κάτι πρέπει να γίνεται. Αυτό το κάτι εκτελείται δεξιά, στον server. Έχουμε server side scripting ή server side programming, καθόσον υποθέτουμε ότι ο client έχει περιορισμένες δυνατότητες, είναι δηλ. μόνον ένας browser.



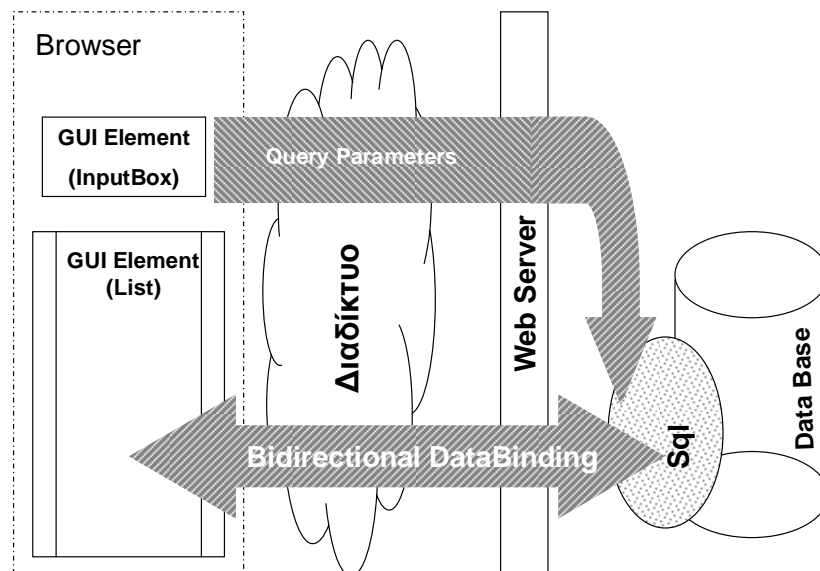
**Σχήμα 11. 1.** Γενική ιδέα ολοκλήρωσης εφαρμογών web

Αν ανακεφαλαιώσουμε, αναλύσουμε και ομαδοποιήσουμε σε κάποιο αρχιτεκτονικό μοντέλο βασισμένο στις αρχές του αντικειμενοστραφούς προγραμματισμού τους τρόπους ανταλλαγής μηνυμάτων του http, μπορούμε να απομονώσουμε όλα τα κοινά και επαναλαμβανόμενα στοιχεία και να τα προσφέρουμε έτοιμα στην μορφή ενός ολοκληρωμένου περιβάλλοντος ανάπτυξης. Αυτό που διαφοροποιείται σε κάθε συγκεκριμένη εφαρμογή είναι οι φόρμες οι ίδιες με τα στοιχεία ελέγχου που θέλουμε να έχουν επάνω τους και ο τρόπος αντίδρασης του τελικού συστήματος σε ενέργειες του ανθρώπινου χρήστη. Τις τελευταίες τις μοντελοποιούμε κάτω από την γενική έννοια του συμβάντος (event). Τότε αρκεί να δώσουμε την δυνατότητα συγγραφής ρουτινών κώδικα που θα εκτελούνται (στον server) μετά από κάθε τέτοιο συμβάν. Κάθε στοιχείο ελέγχου έχει έναν (συνήθως μεγάλο) αριθμό προκαθορισμένων συμβάντων. Σε κάθε τέτοιο συμβάν μπορεί να συνδεθεί μία διαδικασία συμβάντος (event handler). Οι αλληλουχία ενεργειών του χρήστη και εκτέλεσης των σχετικών διαδικασιών συμβάντων αποτελούν ουσιαστικά την 'εφαρμογή'. Όλες οι άλλες υποστηρικτικές διαδικασίες είναι κριμένες και πραγματοποιούνται χωρίς καμία επιπλέον προγραμματιστική μας προσπάθεια. Αυτές μπορούν να συνοψισθούν ως εξής:

- το πώς τα συμβάντα και η πληροφορία που εισάγεται από τον ανθρώπινο χρήστη λαμβάνονται υπ' όψιν και στέλλονται από αριστερά προς τα δεξιά είναι ούτως η άλλως διαδικασία ενσωματωμένη στον browser.
- το πώς ο server αποκωδικοποιεί την παραπάνω πληροφορία από τα λαμβανόμενα μηνύματα http και μας την διαθέτει για τον προγραμματισμό των event handlers είναι η συνεισφορά του ολοκληρωμένου περιβάλλοντος.
- το πώς ο κώδικας μας μπορεί να καθορίζει τις επόμενες ενέργειες προσφέρεται πάλι από τον ολοκληρωμένο περιβάλλον (π.χ. ο τονισμός κάποιου στοιχείου ελέγχου στην τρέχουσα φόρμα, ή η παρουσίαση της επομένης φόρμας). Οι μηχανισμοί με τους οποίους αυτά πραγματοποιούνται είναι και αυτοί γενικοί και έτοιμοι βασιζόμενοι βεβαίως στην μεταφορά της κατάλληλης πληροφορίας μέσω http.

Κάθε λοιπόν περιβάλλον ανάπτυξης έχει ένα γραφικό τρόπο για να σχεδιάζουμε τις φόρμες (κατά την ανάπτυξη) και μας επιτρέπει την συγγραφή του κώδικα των event handlers. Από αυτά συντάσσεται αυτόματα η html που αποδίδει τις σχεδιασθείσες φόρμες. Στο run time οι φόρμες στέλνονται προς τον browser και τα αποτελέσματα των ενεργειών του χρήστη πίσω στον server. Επιπλέον, στον server, τρέχει και κώδικας πίσω από κάθε στοιχείο ελέγχου (code behind). Εκεί λαμβάνονται υπ' όψιν τα συμβάντα και η μεταβλητή πληροφορία πάνω στις φόρμες σύμφωνα με το δικό μας προγραμματισμό των event handlers.

Ένα μεγάλο μέρος των απαιτήσεων και των σχετικών τεχνολογιών που τις εξυπηρετούν αφορά την βάση δεδομένων, που σχεδόν πάντα υπάρχει πίσω από κάθε server και αποτελεί ουσιαστικό μέρος της εφαρμογής. Από όλα αυτά δεν έχει ειπωθεί τίποτα ως τώρα, αλλά αντιμετωπίζονται με το ίδιο σκεπτικό όπως τα παραπάνω. Συνήθως αυτό που τελικά ο χρήστης ζητά και βλέπει μέσω του browser ευρίσκεται σαν εγγραφή και αποκτάται από την βάση με κάποιο ερώτημα. Μια τυπική πλήρης αλληλεπίδραση αναλύεται στην τυπική αλυσίδα ενεργειών που δείχνει το παράδειγμα στο παρακάτω σχήμα:



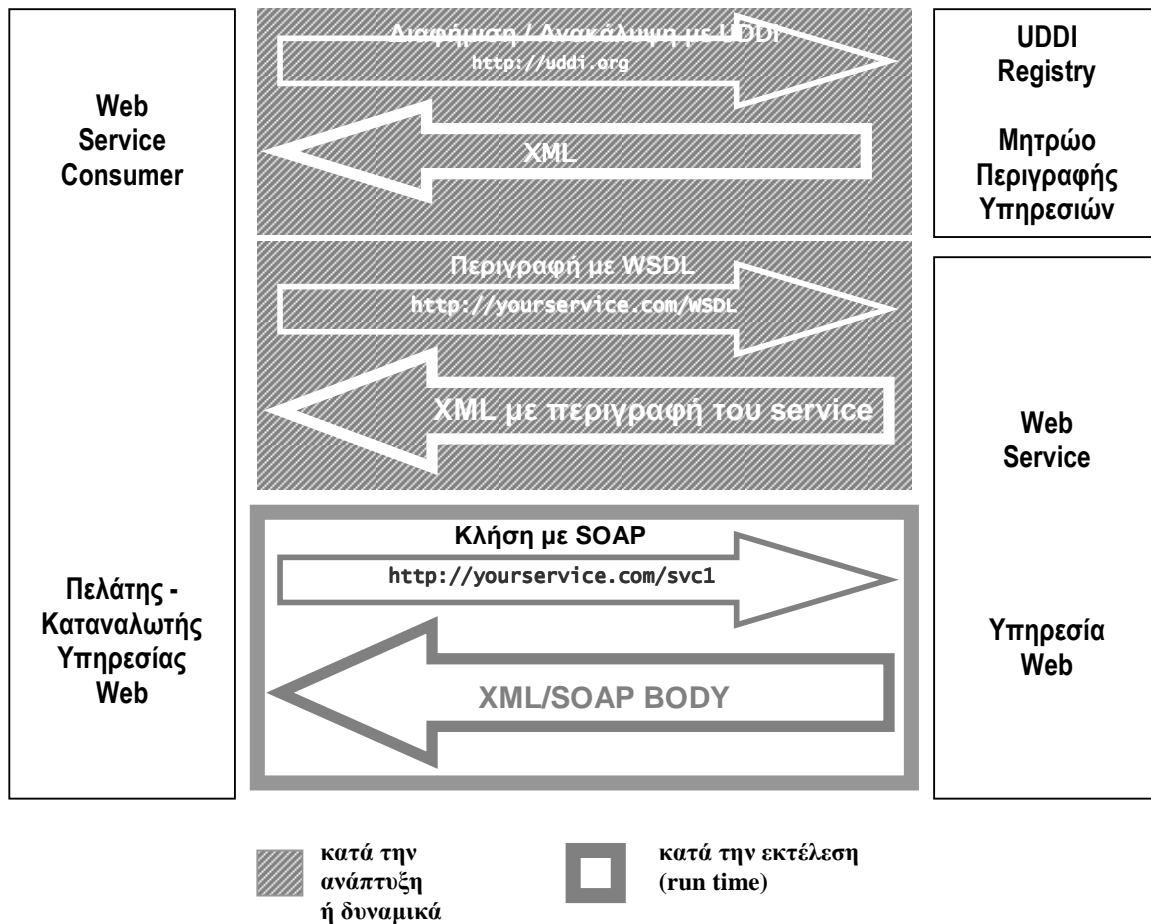
**Σχήμα 11. 2.** Διαδικτυακή εφαρμογή συμπεριλαμβάνουσα βάση δεδομένων

Το ολοκληρωμένο περιβάλλον μας δίνει την δυνατότητα να δέσουμε και στις δύο κατευθύνσεις αυτό που φαίνεται στο GUI Element 'List' του browser με εγγραφές ενός ερωτήματος (query) στην βάση. Τι ίδιο το ερώτημα δυνατόν να μην είναι μία στατική sql έκφραση, αλλά να παραμετροποιείται σύμφωνα με κατάλληλη εισαγωγή δεδομένων /

παραμέτρου σε άλλο GUI Element της φόρμας. Το όλο δέσιμο ονομάζεται (bidirectional) data binding και προσφέρεται τυποποιημένο στα περιβάλλοντα ανάπτυξης.

## **Συντελεστές Υπηρεσιών Ιστού**

Το σενάριο χρήση των υπηρεσιών web σε ένα εντελώς ανοικτό περιβάλλον φαίνεται στο παρακάτω Σχήμα 11.7. Στο πλήρες σενάριο, κατ'αρχήν πρέπει να αναζητηθεί η κατάλληλη υπηρεσία με το πρωτόκολλο UDDI (Universal Description, Discovery and Integration). Ο 'πελάτης', σαν καταναλωτής, απευθύνεται σε ένα μητρώο υπηρεσιών, όπου αυτές είναι καταχωρημένες και υποτίθεται ότι διαθέτει (ίσως προγραμματισμένα) την δυνατότητα της επιλογής του ποία του είναι χρήσιμη. Ο τρόπος αναζήτησης (URL) της επιλεγείσας του έρχεται πίσω συνταγμένος σε XML. Το 'μητρώο υπηρεσιών' είναι ουσιαστικά μία βάση δεδομένων (UDDI registration database for Web Services). Ο πελάτης απευθύνεται τώρα στον πάροχο της υπηρεσίας, ο οποίος του επιστρέφει τον τρόπο χρήσης των μεθόδων αυτής συντεταγμένο κατά το πρότυπο WSDL (Web Services Description Language). Ουσιαστικά του περιγράφονται όσα αναφέρθηκαν για την απομακρυσμένη κλήση με SOAP - RPC. (methods, arguments, return values). Ακολουθεί η πραγματική χρήση της υπηρεσίας σε run time και η απάντηση μέσω XML και SOAP.



**Σχήμα 11.3.** Ανακάλυψη και χρήση υπηρεσίας web

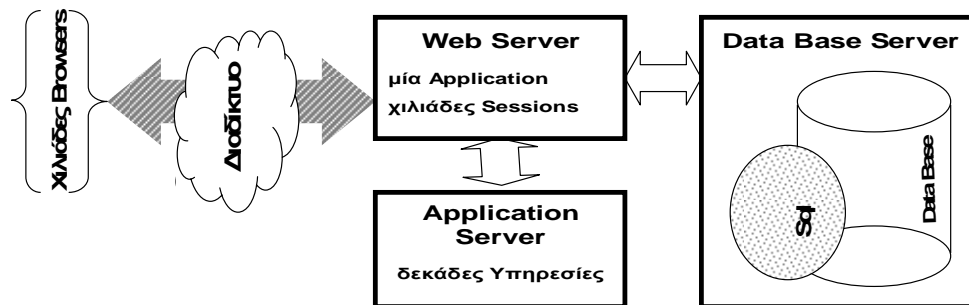
Μεγάλο βάρος τίθεται στο ορισμό και ομαδοποίηση των υπηρεσιών ιστού, έτσι ώστε να είναι χρήσιμες και εύχρηστες στον μεγαλύτερο δυνατό αριθμό πελατών. Τέτοιοι είναι, στην περίπτωση αυτή, τα άλλα προγράμματα / καταναλωτές πολύ, λίγο, ελάχιστα ή και καθόλου γνωστά ή προκαθορισμένα.

## Εφαρμογές και Υπηρεσίες εκτός Ιστού

Χάρis στον αντικειμενοστραφή προγραμματισμό, η τυποποίηση της ανάπτυξης και η ομαδοποίηση των σχετικών λειτουργιών με χρήση των κοινών χαρακτηριστικών τους έχει θέσει το σκέλος του διαδικτύου στο παρασκήνιο. Αν παρατηρήσει κανείς αρχιτεκτονικές ιδέες όπως στα Σχήματα 11.1, 11.2, 11.3, εύκολα περιλαμβάνει με μικρές τροποποιήσεις την περίπτωση, όπου δεν υπάρχει browser και διαδίκτυο, αλλά αριστερά μία φόρμα στον ίδιο μηχανήμα (για τα Σχήματα 11.1 και 11.2) ή μία άλλη εσωτερική διαδικασία (για τα Σχήματα 11.7.). Για παράδειγμα στο περιβάλλον της MS έχουμε αντίστοιχα αριστερά μια Win Form (και όχι μία Web form) ή μία Windows service(και όχι μία Web service). Σημειώνεται ότι ιστορικά η μετάβαση που συζητείται εδώ έγινε στην αντίθετη διεύθυνση, δηλαδή το περιβάλλον που περιλαμβάνει την απομακρυσμένη πρόσβαση μέσω του διαδικτύου πραγματοποιήθηκε κατόπιν και με χαρακτηριστικά που κατά το δυνατόν να ομοιάζαν με αυτά

που είχαν γίνει τρέχοντα και συνηθισμένα σε 'τοπικά' περιβάλλοντα. Σημειώνεται επίσης ότι η πολυπλοκότητα και η δυνατότητες του 'τοπικού' περιβάλλοντος είναι συνήθως ένα υπερσύνολο του διαδικτυακού. Μία τοπική φόρμα έχει πολλές περισσότερες δυνατότητες ενσωματωμένες στα στοιχεία ελέγχου της (GUI elements ή controls) παρά όσα είναι δυνατά με ένα browser (γιατί !?!).

Οι έννοια της υπηρεσίας με τοπική και όχι εξ αποστάσεως κλήση (με SOAP – RPC) μέσω διαδικτύου ευρίσκεται και στο εσωτερικό του 'server'. Σε σοβαρές διαδικτυακές εφαρμογές ο 'server' 'σπάει' σε δύο (ή και σε τρία). Αφ' ενός έχουμε μπροστά το web server που έχει όλο το βάρος της εισροής των ερωτήσεων (GET ή POST requests) και εκροής των απαντήσεων (responses) από / προς χιλιάδες browsers και αφ' εταίρου, δίπλα του, τον 'application server', ο οποίος προσφέρει υπό μορφήν τοπικών υπηρεσιών όλα εκείνα που απαιτούν εκτεταμένους υπολογισμούς, αναζητήσεις σε βάσεις (μέσω ίδιου ή άλλου data base server), κλπ, βλ. Σχήμα 11.4. Έτσι ο web server αναθέτει δύσκολες και χρονοβόρες εργασίες σε άλλους και αφοσιώνεται στο μοναδικό, αλλά εξαιρετικά βαρύ, έργο να αποστέλλει σχεδόν έτοιμες πλέον ιστοσελίδες στους πελάτες του.



**Σχήμα 11.4.** Κατανομή εξυπηρέτησης πελατών σε περισσότερους servers

Στο σχήμα φαίνονται 'χιλιάδες' sessions, η μία για τον κάθε ενεργό browser που εξυπηρετείται από μία εφαρμογή (application). Υπάρχουν ισάριθμα Session objects και ένα Application object. Η λογική της εφαρμογής εξασφαλίζεται από μερικές 'δεκάδες' υπηρεσίες μέσα στον Application, που προσφέρουν μερικές 'εκατοντάδες' διακριτές μεθόδους. Τέλος η εφαρμογή βασίζεται σε και προσφέρει δεδομένα μέσα από μία ή περισσότερες βάσεις δεδομένων.

## Οριζόντιες Υπηρεσίες

Εφαρμογές και υπηρεσίες, τοπικές ή διαδικτυακές έχουν και κατά περίπτωση διάφορα ή κοινά οριζόντια χαρακτηριστικά. Αυτά προσδίδονται από τις λεγόμενες οριζόντιες υπηρεσίες. Έχουμε συνηθίσει σε πολλές από αυτές ακόμη και στο επίπεδο κοινού χρήστη προσωπικού υπολογιστού: login, undo, κλπ. Το undo έχει την ίδια σημασία και αποτέλεσμα ανεξαρτήτως προγράμματος, είναι δηλαδή οριζόντιο και κατά προτίμησιν μία φορά προγραμματισμένο για όλες τις περιπτώσεις. Το απλούστερο παράδειγμα στις διαδικτυακές εφαρμογές είναι η καταμέτρηση των επισκέψεων των πελατών μας. Επίσης μαζί με ένα σύνολο δικτυακών υπηρεσιών θα θέλαμε να έχουμε ένα κοινό μηχανισμό για το έλεγχο πρόσβασης, άλλον έναν για την χρέωση, άλλον ένα για την διατήρηση log files κλπ. Οι οριζόντιες υπηρεσίες μπορούν να πραγματοποιηθούν σαν τοπικές υπηρεσίες, καλούμενες από το εσωτερικό της εφαρμογής ή από άλλες υπηρεσίες στον ρόλο, αυτή την φορά, του πελάτη.

## **Web Interface και Internet of Things**

Η απλότητα και γενική παραδοχή του πρωτοκόλλου http επιτρέπει την φθηνή πρόσβαση μέσω web σε συσκευές πολύ απλούστερες και λειτουργικά πτωχότερες από έναν πλήρη server, όπως ο Apache που γνωρίσαμε. Έτσι σε κάθε απλή ψηφιακή συσκευή που κατασκευάζεται ή πωλείται σαν αυτόνομο σύστημα είναι πλέον σχεδόν απαίτηση να εντάσσεται και ένα 'web interface'. Αυτό σημαίνει ότι πρέπει η συσκευή να δρα και ως απλός web server. Όταν ο χρήστης της, σαν client, απευθυνθεί σε αυτή, πρέπει να του παρουσιάζεται μία έστω και στοιχειώδης web σελίδα. Η σελίδα αυτή χρησιμεύει σαν οδηγία χρήσεως, επιτρέπει την αρχικοποίηση (configuration) ή 'διοίκηση' (management) της συσκευής, ή αποτελεί αναπόσπαστο μέρος της ίδιας της λειτουργίας δίνοντας δεδομένα (π.χ. ένα 'web θερμόμετρο').

Η ιδέα επεκτείνεται σε μεγάλη κλίμακα, έτσι που στο άμεσο μέλλον να μιλάμε για internet πραγμάτων. Η κάθε συσκευή ή αισθητήριο, ακόμη και ευτελούς αξίας και κατά προτίμησιν ασύρματη, θα έχει την IP διεύθυνσή της και θα 'καταλαβαίνει' http. Δεν έχουμε όμως φθάσει μέχρι εκεί. Νέοι τρόποι εξαιρετικά δυναμικής διευθυνσιοδότησης απαιτούνται για δισεκατομμύρια τέτοιων συσκευών και η αυτονομία τους δεν επιτρέπει σπατάλη ισχύος με επανεκπομπές και πρωτόκολλο μεταφοράς τόσο πολύπλοκο όσο το TCP.

## 12. Το Περιβάλλον .Net

Έχει ενδιαφέρον να δούμε τις σχέσεις ανάμεσα στις βασικές τεχνολογίες που ανακεφαλαιώθηκαν στα προηγούμενα κεφάλαια καθώς και το πώς αυτές αλληλοσυμπληρώνονται. Ο πιο άμεσος τρόπος είναι να γνωρίσουμε ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE, Integrated Development Environment). Εκεί θα διαπιστώσουμε κατ' αρχήν αυτές τις στενές σχέσεις και αλληλοσυσχετίσεις από το γεγονός ότι ένα τέτοιο περιβάλλον είναι ικανό να αντιμετωπίσει με σχεδόν ενιαίο τρόπο όλες τις περιπτώσεις. Θα δούμε επίσης ότι οι βασικές τεχνολογίες (μέχρι και το επίπεδο http, την ανταλλαγή μέσω SOAP, κλπ) είναι κατ' αρχήν αόρατες από τον χρήστη - σχεδιαστή. Η ανάπτυξη σε ένα τέτοιο IDE αρχίζει πάνω από αυτές και το IDE εξασφαλίζει όλες τους επικοινωνιακούς μηχανισμούς και αλληλοσυσχετίσεις αυτόματα.

Το περιβάλλον **.Net** της Microsoft επιτρέπει την συνύπαρξη και συνεργασία προγραμμάτων γραμμένων σε διαφορετικές αντικειμενοστραφείς γλώσσες, Visual Basic, C#, C++, J#, JScript. Τούτο επιτυγχάνεται διότι κάθε πρόγραμμα, γραμμένο σε οποιαδήποτε από τις γλώσσες αυτές, μεταγλωττίζεται αρχικά στην **Common Language Runtime - CLR** που είναι η πεμπτούσια του περιβάλλοντος **.Net**. Το CLR αντιστοιχεί στο Byte Code, σαν ενδιάμεσος κώδικας πριν την τελική εκτέλεση από κάποιον επεξεργαστή. Η Java είναι portable σε όλους τους επεξεργαστές που μπορούν να τρέξουν τον Java Byte Code. Αντίστοιχα κάθε ανάπτυξη στις παραπάνω γλώσσες μπορεί να ενταχθεί στο **.Net** και είναι portable σε επεξεργαστές που μπορούν να τρέξουν CLR.

Το **ASP.NET**, διάδοχος του περιβάλλοντος ASP, είναι η τεχνολογία της MS για δυναμικές σελίδες. Το προγραμματιστικό τους μέρος μπορεί να είναι στις παραπάνω γλώσσες. Οι σελίδες ASP (ASP pages) με επίθεμα **.asp** φιλοξενούνται στον **Internet Information Server - IIS**. Ο IIS είναι αντίστοιχος του Apache που γνωρίσαμε. Οι σελίδες **.asp** που κατασκευάζονται μέσα στο Visual Studio μπορούν να τρέξουν και απ' ευθείας μέσα από αυτό στην διάρκεια της ανάπτυξής τους (βλ. παρακάτω). Τα παρακάτω αναφέρονται στο περιβάλλον ανάπτυξης Microsoft Visual Studio 2005.

### Το SDK του Περιβάλλοντος .Net (MS Visual Studio)

Με το ίδιο SDK (S/w Development Kit) του περιβάλλοντος **.Net** (Microsoft Visual; Studio) μπορούμε να αναπτύσσουμε κώδικα σε διάφορα από τα πλαίσια που γνωρίσαμε παραπάνω. Επιλέγουμε

- File -> Open (New) Web Site ASP.NET Web Site για **εφαρμογή ιστού**.
- File -> Open (New) Web Site ASP.NET Web Service για **υπηρεσία ιστού**.
- File -> Open (New) Windows Application για **τοπική εφαρμογή στο PC** μας.
- File -> Open (New) Windows Service για **τοπική υπηρεσία στο PC** μας.

Τα παραπάνω πρέπει να υπάρχουν σαν installed templates δηλ. εγκατεστημένα (μέσα στο Visual Studio) εκμαγαεία που δίδουν την αρχική δομή και αρχικές κλάσεις όσων θέλομε να κάνομε.



Οι εφαρμογές με φόρμες Windows (Win Forms) και οι εφαρμογές με φόρμες για browsers (Web Forms) δεν διαφέρουν ουσιαστικά στο υψηλό επίπεδο του σχεδιασμού μέσα από το ολοκληρωμένο περιβάλλον. Στην πραγματική βέβαια λειτουργία (run time) η διαφορά είναι αβυσσαλέα, καθόσον στην πρώτη περίπτωση οι φόρμες είναι στο ίδιο μηχάνημα με τον (υποτιθέμενο) 'server', ενώ στην δεύτερη οι φόρμες εμφανίζονται στον browser μακριά από τον πραγματικό server (δηλ. τον ΠΣ), οποίος φιλοξενεί τις σελίδες .asp. Επειδή το σχεδιαστικό περιβάλλον έχει κρυμμένες μέσα τους όλες τις λεπτομέρειες και τα κατώτερα στρώματα που μάθαμε, οι υποστηρίζοντες μηχανισμοί είναι σε εμάς κρυμμένοι και η ανάπτυξη και των δύο περιπτώσεων φαίνεται σχεδόν σαν ίδια.

Στην περίπτωση εφαρμογών ιστού, που αποκλειστικά θα εξετάσουμε, μπορούμε πολύ εύκολα να επεκταθούμε σε πλήρεις 3-tier applications (εφαρμογές τριών συμμετεχόντων). Σε αυτές αλληλεπιδρούν (α) ο browser, (β) ο server και (γ) μία βάση. Το (γ) αγνοήθηκε ως τώρα, ως ξένο προς τα δικτυακά, δεν μπορεί όμως να λείπει από οποιαδήποτε σοβαρή εφαρμογή. Το .Net περιβάλλον μας επιτρέπει επιπλέον να επεκταθούμε πέραν από την έννοια της κλασικής (σχεσιακής - relational) βάσης συμπεριλαμβάνοντας και διάβασμα από / εγγραφή σε (ιεραρχικές) δομές XML. Άρα το (γ) γενικεύεται σε 'data store', μία γενική έννοια που σήμερα περιλαμβάνει βάσεις ή / και κείμενα XML. Μπορούμε λοιπόν να 'δέσουμε ένα GUI element που θα εμφανίζονται στον browser είτε με έγγραφη σε μία βάση (προσβάσιμη με έκφραση sql) είτε με κλάδο ενός κειμένου XML (προσβάσιμο με έκφραση XPath).

## Το Αρχείο myForm.aspx για Σελίδα Web

Η φόρμα myForm.aspx μπορεί να εμφανισθεί σε 'Design' και σε 'Source View'. Αρχικά με drag and drop από ένα tool box φέρομε τα στοιχεία GUI πάνω στον καμβά της υπό κατασκευήν δυναμικής web σελίδας (Design View). Ο 'Κώδικας' 12.1 δημιουργείται μετά αυτόματα (Source View). Δεν πρόκειται όμως για κώδικα παρά μόνο για μία σειρά δηλώσεων. Με την σειρά που εμφανίζονται οι δηλώσεις για το κάθε στοιχείο XML, θα εμφανίζονται από πάνω αριστερά προς δεξιά και κάτω τα αντίστοιχα στοιχεία GUI πάνω στον browser. Τα περισσότερα από τα παραπάνω στοιχεία XML θα αποτυπωθούν αυτόματα σε κομμάτι κώδικα html και το τελευταίο θα είναι αυτό που θα αποστέλλεται στο run time στον browser. Εξετάζοντας επομένως τον παρακάτω Κώδικα 12.1, αποκαλύπτομε την γενική ιδέα και μερικές από τις πάμπολλες λεπτομέρειες και δυνατότητες του περιβάλλοντος ανάπτυξης.

Είναι σημαντικό να αποσαφηνισθεί εξ αρχής ο ρόλος του αρχείου myForm.aspx:

- (α) τα <% @ ..... />, έχουν αποκλειστικά εσωτερική χρήση και είναι συνήθως compiler directives,
- (β) ο γενικός σκελετός σε html αφορά την ίδια την σελίδα,
- (γ) στην θέση των δηλώσεων σε μορφή XML θα επέμβει ο server και, ακολουθώντας το (α), θα τις αντικαταστήσει με κανονική html, δημιουργώντας έτσι τα στοιχεία GUI μέσα στο (β), δηλαδή την πλήρη σελίδα που θα αποσταλεί για εμφάνιση στον browser και
- (δ) όλες οι ενέργειες που συνδέονται με τα στοιχεία τα GUI αποτελούν το κώδικα που θα τρέχει πάντα στον server και ο οποίος, αν υπάρχει, ευρίσκεται σε άλλο αρχείο, συνδεδεμένο με το RunForm.aspx (βλέπε παρακάτω).

Κάθε control (παραπάνω περίπτωση (γ)) αντιπροσωπεύεται από ένα στοιχείο (element) ονοματιζόμενο asp:nameOfControl. Τα ονόματα των διατιθεμένων από την MS controls είναι

συγκεκριμένα (εδώ χρησιμοποιούνται τα Button, Label, TreeView,,), ενώ των controls, που φτιάξαμε εμείς, είναι δικά μας (cc1). Ακολουθούν διάφορα attributes με τις τιμές τους (όπως γνωρίζουμε, πάντοτε σαν Strings). Μερικά attributes (άλλα προαιρετικά και άλλα υποχρεωτικά) αφορούν την class πατέρα ... όλων των controls. Με (το υποχρεωτικό) runat που εδώ έχει τιμή 'server' δηλώνεται ότι, κατά την δημιουργία του, το συγκεκριμένο control θα συνδεθεί με κώδικα που θα τρέχει στον server (παραπάνω περίπτωση (δ)). Εδώ πάρα πολλά συντελούνται από το SDK χωρίς να φαίνονται πουθενά. Πολλά από τα attributes (προαιρετικά ή υποχρεωτικά) συνοδεύουν μόνο το συγκεκριμένο τύπο control.

Προφανώς το compiler directive

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="myForm.aspx.vb" Inherits="_Default" %>
```

υποδηλώνει ότι εδώ πρόκειται για (δυναμική) σελίδα ASP.NET, η οποία θα αποθηκεύεται στον server και θα αποστέλλεται (μετά από απεικόνισή της σε html) στον client. Το αρχείο myForm.aspx δεν περιέχει κώδικα σε οποιαδήποτε γλώσσα εκτός από τον γενικό σκελετό σε html και δηλώσεις σε μορφή XML, όπως τις προδιαγράφει το συγκεκριμένο SDK. Τούτο σε πολλές περιπτώσεις αρκεί. Κώδικας, σχετιζόμενος με την σελίδα αυτή, μπορεί εν τούτοις να υπάρχει σε άλλο αρχείο, όπως θα εξηγήσουμε στην επόμενη παράγραφο. Έτσι με τα attributes CodeFile και Language δηλώνουμε το αρχείο αυτό και την χρησιμοποιούμενη γλώσσα (VB, δηλ. Visual Basic).

Άλλο compiler directive είναι το

```
<%@ Register Assembly="HelloWeb" Namespace="WebControlLibrary1" TagPrefix="cc1" %>
```

Με το TagPrefix δηλώνεται στον μεταγλωττιστή ότι το cc1 είναι ένα GUI Element (control) φτιαγμένο από εμάς. Παρακάτω θα δούμε την κατασκευή του cc1. Εδώ γνωρίζουμε στον compiler, ότι μπορούμε να αναφερόμαστε σε αυτό το cc1 στην παρούσα σελίδα. Ουσιαστικά πρόκειται για μία δήλωση (declaration).

Το στοιχείο

```
<%@ Register Assembly="HelloWeb" Namespace="WebControlLibrary1" TagPrefix="cc1" %>
```

αφορά

Παρακάτω το τμήμα

```
<html xmlns="http://www.w7.org/1999/xhtml" >
<head runat="server"> <title>myFrom Experimenta Page</title></head>
<body> <form id="form1" runat="server"><div> &nbsp;
είναι τα κλασσικά εγναγωγικά μίας φόρμας, όπως θα γραφόντουσαν και 'με το χέρι'. Φυσικά τα στοιχεία αυτά κλείνουν στο τέλος του αρχείου.
```

Ερχόμαστε τώρα στην παράθεση των συγκεκριμένων GUI Elements που επιλέχθηκαν από τον σχεδιαστή αυτής της φόρμας. Αρχίζουμε με ένα button και ένα label

```
<asp:Button ID="runButton" runat="server" Text="Run" />
<asp:Label ID="Label3" runat="server" Text="Clients"></asp:Label>
```

τα οποία έχουν 'ID' για να αναφερόμαστε προγραμματιστικά σε αυτά, 'Text' το οποίο θα εμφανισθή επ' αυτών. Ο ενδεχόμενος κώδικας που τα συνοδεύει είναι πάντοτε runat="server".

Με τις δύο μόνον γραμμές του στοιχείου asp:Xml εμφανίζουμε τον ωραιότατο πίνακα με τα αυτοκίνητα (βλ. αρχή σημειώσεων της XML).

```
<asp:Xml ID="Xm11" runat="server"
DocumentSource="~/App_Data/CarsForNewXSLTNamespace.xml"
TransformSource="~/App_Data/CarPresentorNEWnamespace.xslt"></asp:Xml>
```

Αρκεί να δώσουμε την πηγή (DocumentSource) και το κείμενο XSLT (TransformSource) που θέλουμε να χρησιμοποιηθεί για την εμφάνιση! Τα ίδια τα αρχεία αποτελούν φυσικά μέρος του Web Site.

Και τώρα με το asp:AccessDataSource η πιο κλασσική περίπτωση ενός πίνακα που τροφοδοτείται από εγγραφές σε πίνακα βάση (εδώ ένας πίνακας ονόματι L3 στην Access TaHi.mdb).

```
<asp:AccessDataSource ID="TaHiDataSource" runat="server"
DataFile="~/App_Data/TaHi.mdb" SelectCommand="SELECT [Id], [Name], [CostAcc],
[nFlow], [Active] FROM [L3]"></asp:AccessDataSource>
```

Παρατηρούμε ομοιότητες και διαφορές με την περίπτωση του XML. Εδώ προσδιορίζουμε το αρχείο της βάσης και την έκφραση sql. Το ίδιο το AccessDataSource δεν πρόκειται να δειχθεί στον browser και το στοιχείο asp:AccessDataSource αφορά αποκλειστικά το πως ο server θα αντλεί τα δεδομένα από την βάση. Για την εμφάνισή των απαιτείται το GUI Element DataGrid, ή GridView που διατίθεται στο toolbox και έχει συρθεί πάνω στο Design View της φόρμας.

Το GridView εμφανίζει δεδομένα σε μορφή πίνακα και πρέπει να δεθεί με το DataSource (βλ. σύνδεση μέσω του DataSourceID)

```
<asp:GridView ID="L3DGrid" runat="server" Width="352px"
DataSourceID="TaHiDataSource" AutoGenerateColumns="False" DataKeyNames="Id">
  <Columns>
    <asp:BoundField DataField="Id" HeaderText="Id" InsertVisible="False"
      ReadOnly="True" SortExpression="Id" />
    <asp:BoundField DataField="Name" HeaderText="Name" SortExpression="Name" />
    <asp:BoundField DataField="CostAcc" HeaderText="CostAcc"
      SortExpression="CostAcc" />
    <asp:BoundField DataField="nFlow" HeaderText="NodeFlow"
      SortExpression="nFlow" />
    <asp:BoundField DataField="Active" HeaderText="Active"
      SortExpression="Active" />
  </Columns>
  <RowStyle HorizontalAlign="Right" />
  <EditRowStyle HorizontalAlign="Right" />
</asp:GridView>
```

Το HeaderText δίδει τις λεζάντες που εμφανίζονται πάνω από κάθε στήλη, κλπ.

Και τώρα θα στείλουμε στον browser ένα 'γυμνό' XML, για το οποίο δεν έχουμε εμείς φροντίσει να παρέξομε κείμενο XSLT για την εμφάνισή του. Εδώ καταφεύγομε στο TreeView, άλλο ένα έτοιμο GUI Element, ή control, ικανό βεβαίως να δείχνει οποιοδήποτε XML, καθόσον όλα τα XML είναι δένδρα. Η ιδέα είναι η ίδια: πρώτα το (εδώ XML)DataSource και κατόπιν το (εδώ Tree)View, όπου και προσδιορίζεται πως αυτό δένεται με τα δεδομένα

```
<asp:XmlDataSource id="PeopleDataSource" runat="server" DataFile="~/App_Data/peopleXml.xml" />
<asp:TreeView id="PeopleTreeView" runat="server" DataSourceID="PeopleDataSource">
  <DataBindings>
    <asp:TreeNodeBinding DataMember="LastName" TextField="#InnerText" />
    <asp:TreeNodeBinding DataMember="FirstName" TextField="#InnerText" />
    <asp:TreeNodeBinding DataMember="Street" TextField="#InnerText" />
    <asp:TreeNodeBinding DataMember="City" TextField="#InnerText" />
    <asp:TreeNodeBinding DataMember="Region" TextField="#InnerText" />
    <asp:TreeNodeBinding DataMember="ZipCode" TextField="#InnerText" />
    <asp:TreeNodeBinding DataMember="Title" TextField="#InnerText" />
    <asp:TreeNodeBinding DataMember="Description" TextField="#InnerText" />
  </DataBindings>
</asp:TreeView>
```

Τώρα, αυτό το XML, θα ανοιγοκλείνει τα φύλλα του στον browser και έτσι θα βλέπομε οποιοδήποτε μεγέθους XML.. Πώς γίνεται αυτό, αφού εμείς δε γράψαμε πουθενά σχετικό

κώδικα; Το μυστικό έγκειται ότι αυτό ειδικά το GUI Element, περιέχει και κώδικα σε JavaScript που κάνει την δουλειά. Ο κώδικας αυτός αποτελεί αναπόσπαστο μέρος του GUI Element, δεν εμφανίζεται πουθενά σε εμάς. Είναι σαν το παράδειγμα client side scripting που γνωρίσαμε στην ενότητα περί XML με τον κώδικα να είναι ενσωματωμένος στο control και να στέλλεται από τον server στον browser. Τούτος πρέπει να είναι για την περίπτωση αυτή 'JavaScript enabled', αλλιώς δεν θα λειτουργεί το ανοιγόκλεισμα των φύλλων. Προσέξτε ότι και εδώ εξακολουθούμε να εχομε runat="server". Τα όσα είπαμε περί τρεξίματος της JavaScript στον client, αποτελούν εσωτερική υπόθεση του control και όχι το πως ενδεχομένως εμείς θέλομε προγραμματιστικά να το χρησιμοποιήσομε.

Και τέλος μπαίνοντας στα ενδότερα φτιάχνομε και ένα δικό μας control ή GUI Element. Το namespace δεν είναι πιά asp, αλλά cc1 (δικό μας).

```
<cc1:WebCustomControl1 ID="WebCustomControl1_1" runat="server" Height="12px"
Text=" Hello from Server Control made by me " Width="312px"/>
```

Είναι ένα απλό μήνυμα σε ένα Text Box αλλά κατασκευάστηκε με ιδιαίτερο τρόπο. Η περίπτωση αυτή θα αναλυθεί ιδιαίτερος πιά κάτω.

Δίνομε τώρα την πλήρη σελίδα.

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="myForm.aspx.vb" Inherits="_Default"
%>

<%@ Register Assembly="HelloWeb" Namespace="WebControlLibrary1" TagPrefix="cc1" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w7.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w7.org/1999/xhtml" >
<head runat="server">
<title>myFrom Experimenta Page</title>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:Button ID="runButton" runat="server" Text="Run" />
<asp:Label ID="Label3" runat="server" Text="Clients"></asp:Label>
<asp:Xml ID="Xm1" runat="server"
DocumentSource="~/App_Data/CarsForNewXSLTNamespace.xml"
TransformSource="~/App_Data/CarPresentorNEWnamespace.xslt"></asp:Xml>
<input id="Reset1" type="reset" value="reset" />&nbsp;&nbsp;&nbsp;
<asp:AccessDataSource ID="TaHiDataSource" runat="server"
DataFile="~/App_Data/TaHi.mdb"
SelectCommand="SELECT [Id], [Name], [CostAcc], [nFlow], [Active] FROM
[L3]"></asp:AccessDataSource>
<asp:TextBox ID="TextBox1" runat="server" Text="<%$ Resources:RunForm, String1
%>"></asp:TextBox>
<asp:GridView ID="L3DGrid" runat="server" Width="352px" DataSourceID="TaHiDataSource"
AutoGenerateColumns="False" DataKeyNames="Id">
<Columns>
<asp:BoundField DataField="Id" HeaderText="Id" InsertVisible="False"
ReadOnly="True"
SortExpression="Id" />
<asp:BoundField DataField="Name" HeaderText="Name" SortExpression="Name" />
<asp:BoundField DataField="CostAcc" HeaderText="CostAcc"
SortExpression="CostAcc" />
<asp:BoundField DataField="nFlow" HeaderText="NodeFlow" SortExpression="nFlow"
/ >
```

```

        <asp:BoundField DataField="Active" HeaderText="Active" SortExpression="Active"
/>
    </Columns>
    <RowStyle HorizontalAlign="Right" />
    <EditRowStyle HorizontalAlign="Right" />
</asp:GridView>

    <asp:XmlDataSource
id="PeopleDataSource"
runat="server"
DataFile="~/App_Data/peopleXml.xml" />

<asp:TreeView
id="PeopleTreeView"
runat="server"
DataSourceID="PeopleDataSource">
    <DataBindings>
        <asp:TreeNodeBinding DataMember="LastName" TextField="#InnerText" />
        <asp:TreeNodeBinding DataMember="FirstName" TextField="#InnerText" />
        <asp:TreeNodeBinding DataMember="Street" TextField="#InnerText" />
        <asp:TreeNodeBinding DataMember="City" TextField="#InnerText" />
        <asp:TreeNodeBinding DataMember="Region" TextField="#InnerText" />
        <asp:TreeNodeBinding DataMember="ZipCode" TextField="#InnerText" />
        <asp:TreeNodeBinding DataMember="Title" TextField="#InnerText" />
        <asp:TreeNodeBinding DataMember="Description" TextField="#InnerText" />
    </DataBindings>
</asp:TreeView>
    <asp:TextBox ID="TextBox2" runat="server" BackColor=Azure></asp:TextBox>
    <ccl:WebCustomControl1 ID="WebCustomControl1_2" runat="server" />
    &nbsp;&nbsp;&nbsp;<br />
runat="server" Height="12px"
    Text="Hellow from Server Control made by me -
    See C:\Documents and Settings\Administrator\Desktop\DotNetCode\HelloWeb !!"
Width="312px" />
</div>
</form>
</body>
</html>

```

## Κώδικας 12.1. Δήλωση των στοιχείων της σελίδας στο myForm.aspx

### Το Αρχείο myForm.aspx.vb για Code Behind

Το επίθεμα .vb του αρχείου myForm.aspx.vb υποδηλώνει, ότι εδώ υπάρχει κώδικας σε γλώσσα Visual Basic, που συνοδεύει την προηγούμενη σελίδα και εξειδικεύει / εμπλουτίζει την συμπεριφορά είτε όλης της φόρμας, είτε των επιμέρους controls που αυτή φιλοξενεί. Είδαμε εξ' άλλου στην προηγούμενη παράγραφο, πώς επιτυγχάνεται η σχετική ενημέρωση του compiler. Συμπεριλαμβάνομε εδώ (θα μπορούσε να ήταν και σε ξεχωριστό αρχείο) αρχικοποιήσεις και γενικότερες ενέργειες. Τέτοιες είναι π.χ. η σύνδεση και άνοιγμα της βάσης, η αλλαγή κειμένου σε ένα text box με το πάτημα ενός button κλπ. Πρόκειται πράγματι για κώδικα που υπάρχει στο παρασκήνιο (Code Behind) και σχετίζεται άμεσα με όσα θέλομε να συντελούνται στην όλη εφαρμογή. Ο κώδικας μπαίνει σε συγκεκριμένες θέσεις που προτείνονται από το περιβάλλον ανάπτυξης. Πέραν από δηλώσεις τύπων και μεταβλητών, δεν μπορεί να υπάρχει άλλος κώδικας παρά η κωδικοποίηση συγκεκριμένων event handlers. Η σελίδα σαν σύνολο και τα στοιχεία που την αποτελούν αντιδρούν σε χειρισμούς του ανθρώπινου χρήστη. Με αυτούς είναι συνδεδεμένα συγκεκριμένα συμβάντα (events) για τα οποία, όπου θέλομε, εξειδικεύομε ή συμπληρώνομε την συμπεριφορά της εφαρμογής με την πλήρωση των σχετικών ρουτινών. Αυτές αποτελούν και τους event handlers. Άλλος κώδικας, γραμμένος αλλού, δεν έχει κανένα νόημα και φυσικά δεν

επιτρέπεται να υπάρχει. Στο επίπεδο αυτό ο προγραμματισμός δεν διαφέρει από την ανάπτυξη μίας εφαρμογής WinForms σε τοπικό περιβάλλον. Και εκεί οφείλουμε να προδιαγράψουμε τι θα συμβαίνει σε κάθε ενέργεια στην οθόνη του PC. Εδώ όμως πίσω μας και χωρίς να μας απασχολεί με αυτό το περιβάλλον ανάπτυξης, κάθε ενέργεια του χρήστη, κάθε click του, συμβαίνει μακριά, στον browser, και η κλήση στον αντίστοιχο event handler μέσα στον server γίνεται μετά από λήψη σχετικού μηνύματος http.

## Άλλα Controls και Πρόσθετες Δυνατότητες

Στην περίπτωση που επιθυμούμε λειτουργικότητα που δεν προσφέρεται από κάποιο από τα controls (GUI Elements) του ASP.NET, μπορούμε να κατασκευάσουμε δικά μας. Υπάρχουν δύο ειδών, τα user και τα custom controls.

Τα user controls είναι τα απλούστερα και φτιάχνονται σαν containers στους οποίους βάζουμε κώδικα html για όμορφη παρουσίαση και επί μέρους άλλα προϋπάρχοντα controls. Το σύνολο εμφανίζεται σαν κάτι το ενιαίο με τις δικές του ιδιότητες και μεθόδους. Είναι κατά βάση η σύνδεση απλούστερων controls σε κάτι δικό μας που περιέχει την ένωση των επί μέρους λειτουργιών. Από την στιγμή που σχεδιάστηκε, το user control αποτελεί ένα αυθύπαρκτο control, χρησιμοποιήσιμο σαν κάθε άλλο.

Ένα custom control είναι μία class, την οποία γράφουμε εξ αρχής σαν εξειδίκευση της class Control ή WebControl του ASP.NET.

## User Controls

Παράδειγμα ενός user control δίδεται στον παρακάτω Κώδικα 12.2. που περιέχεται σε ένα αρχείο .ascx στην προκειμένη περίπτωση στο MoneyField.ascx. Πρόκειται για ένα TextBox και μία DropDownList, αμφότερα κανονικά controls του ASP.Net. Το user control δείχνει νομίσματα πίσω από τα οποία κρύβονται οι συντελεστές μετατροπής των μεταξύ των αξιών. Φαίνεται καθαρά ότι πρόκειται για την ομαδοποίηση πολλών controls σε ένα νέο ενιαίο που προσφέρει μία ιδιαίτερη, από εμάς ενσωματωμένη, λειτουργικότητα.. Φυσικά η πρώτη γραμμή είναι το compiler directive, που δηλώνει ότι ο κώδικας αφορά Control και όχι (web) Page

```
<%@ Control Inherits="MoneyFieldBase" Src="MoneyField.ascx.cs" %>
<asp:TextBox ID="amount" Runat="server" />
<asp:DropDownList ID="currency" AutoPostBack="true"
    OnSelectedIndexChanged="Select" Runat="server">
    <asp:ListItem Text="Euro" Value="1.0" Selected="true" />
    <asp:ListItem Text="Dollars" Value="0.88" />
    <asp:ListItem Text="Francs" Value="1.47" />
    <asp:ListItem Text="Pounds" Value="0.62" />
</asp:DropDownList>
```

### Κώδικα 12.2. Το αρχείο MoneyField.ascx. που ορίζει ένα user control

Ενώ αρχεία με επίθεμα .aspx αφορούν, όπως είδαμε παραπάνω, σελίδες, το .ascx υποδηλώνει τον ορισμό ενός 'user control'. Ως εκ τούτου έχουμε εδώ το compiler directive <%@ Control attribute="value" [attribute="value" ... ] %>. Τα user controls δεν μπορούν να τρέξουν σαν αυτόνομα αρχεία. Το αρχείο τους το προσθέτουμε στις σελίδες ASP.NET, όπως θα κάναμε με οποιοδήποτε άλλο control. Επίσης δεν εμπεριέχει τα στοιχεία html, body ή form. Αυτά

ευρίσκονται όπως είδαμε πιο πάνω στην φιλοξενούσα δυναμική σελίδα. Επιπλέον ένα αρχείο .ascx.vb μπορεί και εδώ να συνεισφέρει ‘VB κώδικα στο παρασκήνιο’ (Code Behind) σύμφωνα με όσα είδαμε και με την σελίδα web.

Δεν πρέπει να ξεχνάμε ότι το παραπάνω user control, όπως και όλα τα άλλα, τελικά όλα αυτόματα θα αναλυθούν και διατυπωθούν σε html για να σταλούν σε αυτήν την ‘κουτή’ μορφή στον αδαή browser.

## Custom Controls σε αντιπαράθεση με User Controls

Τα custom controls πρέπει να μεταγλωττίζονται εκ των προτέρων σε ένα DLL, το οποίο τίθεται στον φάκελο \bin. Με ένα custom control επιτυγχάνουμε μία τελείως νέα λειτουργικότητα που δεν περιορίζεται πλέον στο να είναι η ένωση προϋπαρχόντων στοιχείων. Κάθε νέο, δικό μας custom control πρέπει να προέρχεται από την class Control

## Δημιουργία μίας Υπηρεσίας XML Web

Στην περίπτωση των εφαρμογών όπου πρωτοστατούσαν οι (δυναμικές) σελίδες web, είχαμε το αρχείο myForm.aspx όπου καθορίζονταν με δηλωτικό τρόπο τα στοιχεία πάνω στην φόρμα. Επικουρικά και όπου θέλομε παραπάνω λειτουργικότητα υπήρχε και ο κώδικας στο παρασκήνιο γραμμένος στο αρχείο myForm.aspx.vb, με το .vb να υποδηλώνει την γλώσσα. Στην περίπτωση των υπηρεσιών web η έμφαση αναστρέφεται. Δεν έχουμε καθόλου το μέρος εμφάνισης, εφόσον δεν υπάρχει φόρμα να παρουσιασθεί στον χρήστη. Αντίθετα πρωταρχικό ρόλο αποκτά ο κώδικας. Άρα η myForm.aspx αντιστοιχείται με το παρακάτω αρχείο που τώρα ονομάζομε myService.asmx (επίθεμα .asmx αντί .aspx). Περιέχει μόνον το compiler directive <%@ WebService .... %> αντί του <%@ Page .... %> που γνωρίσαμε στην σελίδα

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/myService.cs" Class="myService"
%>
//The particulars of this service, including its wsdl description
//can be seen at
//http://localhost:1050/WebService/myService.asmx
```

### Κώδικας 12.7. Δήλωση Υπηρεσίας στο myService.asmx

Όπως πριν, δηλώνεται με το attribute CodeBehind, το αρχείο, όπου ο compiler θα βρει τον κώδικα. Εδώ για διαφοροποίηση, προτιμάμε την γλώσσα Visual Visual C#, (επίθεμα .cs αντί .vb). Βλέπομε επίσης στο σχόλιο, ότι παρ’ όλον ότι εμείς δεν κατασκευάζομε σελίδα, θα μπορέσομε να ‘δούμε’, σαν ανθρώπινος ‘χρήστης’ την υπηρεσία μέσω browser. Στο τοπικό περιβάλλον ανάπτυξης αρκεί στο τέλος να τρέξομε τον server και να ‘κτυπήσομε’ την ‘σελίδα’ myService.asmx. Θα παρουσιασθεί μία σελίδα, προκατασκευασμένη από το περιβάλλον ανάπτυξης, που θα μας επιτρέπει να δοκιμάσομε (μέσω web) την υπηρεσία.

Ο παρακάτω Κώδικας 12.4 στο αναφερόμενο αρχείο myService.vs πραγματοποιεί (σε γλώσσα Visual Visual C#, επίθεμα .cs) την υπηρεσία XML Web εκθέτοντας τις μεθόδους της μέσω web. Στις πρώτες γραμμές φέρονται με το ‘using’ κλάσεις από τις προϋπάρχουσες στο

SDK και που χρειάζονται, με πιο ενδιαφέρον το 'package' System.Web.Services. Αυτό περιέχει το WebService. Η δική μας Class θα γίνει παιδί αυτής με την παρακάτω δήλωση `public class myService : System.Web.Services.WebService` και θα κληρονομήσει όλες τις ιδιότητες και δυνατότητες στις οποίες αποσκοπούμε. Θα πρέπει απλώς να την επεκτείνουμε και να την εξειδικεύσουμε να κάνει τα δικά μας. Η Class `myService` αποτελεί και την υπηρεσία μας, η οποία και θα εκθέσει τις μεθόδους της. Για τούτο είναι και `public`. Σημειώνουμε ότι εδώ το αρχείο που την περιέχει (`myService.vs`) δεν έχει κατ' ανάγκην το ίδιο όνομα.

Έπονται οι μέθοδοι: Η `public myService()` είναι ο constructor. Εδώ είναι κενός, καλείται λοιπόν ο default constructor. Μετά έχουμε τις `HelloWorld`, `Add` και `Mult`, που όχι μόνον είναι `public` αλλά έχουν και το attribute `[WebMethod]`. Σε αντίθεση η `myService` (δηλ. ο constructor) και η `sub` είναι `public`, αλλά χωρίς αυτό το attribute. Η διαφορά έγκειται στο ότι μόνον οι `HelloWorld`, `Add` και `Mult` εκτίθενται στο διαδίκτυο και μπορούν να κληθούν εξ αποστάσεως. Αυτές έχουν και λεκτική περιγραφή που είναι εμφανίσιμη στο διαδίκτυο.

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://tempuri.org/",
            Description = "Hello from myService")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class myService : System.Web.Services.WebService
{
    public myService () {} // empty constructor - default is used

    [WebMethod(Description="Returns a hello message")]
    public String HelloWorld()
    {return "MyHelloWorld !";}

    [WebMethod(Description = "Returns the sum")]
    public int Add(int a1, int a2)
    {return a1 + a2;}

    [WebMethod] //here we give no description to be exposed
    public int Mult(int a1, int a2)
    {return a1 * a2;}

    // Above methods HelloWorld, Add & Mult
    // have WebMethodAttribute '[WebMethod]'
    // and thus can be remotely called across the Web

    public int Sub(int a1, int a2)
    {return a1 - a2;}
    // Above method Sub does NOT have WebMethodAttribute '[WebMethod]'
    // Although public, it cannot be remotely called across the Web
}
```

#### **Κώδικας 12.4.** Η Υπηρεσία Web XML κωδικοποιημένη στο `myService.asmx`.

Σε πρώτη φάση, μπορούμε να θεωρήσουμε το αρχείο `myService.asmx` σαν μία σελίδα που θέλουμε να κτυπήσουμε από το web. Πηγαίνοντας εκεί μέσω ενός browser, θα μπορέσουμε να δοκιμάσουμε την υπηρεσία σαν ανθρώπινος χρήστης. Όσα δούμε εξάγονται από όσα εμείς γράψαμε για τη υπηρεσία μας και προβάλλονται με προκατασκευασμένο τρόπο: Θα δούμε τις μεθόδους (μόνον όσες επιλέξαμε προς έκθεση στο web) και τις λεκτικές περιγραφές, όπου



τέτοιες υπάρχουν. Επιλέγοντας μία από τις μεθόδους, θα κληθούμε να δώσουμε τις παραμέτρους εισόδου. Μας παρατίθεται επίσης το μήνυμα POST με το οποίο θα σταλεί η παράμετρος εισόδου και το 'HTTP/1.1 200 OK' μήνυμα response με το οποίο θα μας επιστραφεί η απάντηση, στην περίπτωση που θα ήμασταν ένας μηχανικός client.

Το ενδιαφέρον είναι ότι παρήχθη αυτόματα και η περιγραφή σε wsdl, την οποία και βλέπουμε σαν ανθρώπινος χρήστης! Αυτή παρατίθεται παρακάτω, όπου είναι προφανές ότι παρέχεται με δηλωτικό τρόπο, απόλυτα επεξεργάσιμο μηχανικά, οποιοδήποτε στοιχείο ενδιαφέρει τον επίδοξο client, που θα ήθελε να μας καλέσει. Επιπλέον παρατηρούμε ότι τίποτε από τα εσωτερικά του server μας δεν φανερώνεται, διότι δεν ενδιαφέρει. Πληροφορία του που και πως έχει αυτός οργανώσει τα σχετικά αρχεία, η γλώσσα και ο κώδικας της υπηρεσίας είναι εντελώς εσωτερικά θέματα που δεν ενδιαφέρουν αλλά και δεν περιορίζουν τον πελάτη.

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:tns="http://tempuri.org/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" targetNamespace="http://tempuri.org/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Hello from
myService</wsdl:documentation>
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
      <s:element name="HelloWorld">
        <s:complexType />
      </s:element>
      <s:element name="HelloWorldResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="HelloWorldResult" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="Add">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="a1" type="s:int" />
            <s:element minOccurs="1" maxOccurs="1" name="a2" type="s:int" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="AddResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="AddResult" type="s:int" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="Mult">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="a1" type="s:int" />
            <s:element minOccurs="1" maxOccurs="1" name="a2" type="s:int" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="MultResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="MultResult" type="s:int" />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="HelloWorldSoapIn">
    <wsdl:part name="parameters" element="tns:HelloWorld" />
  </wsdl:message>
</wsdl:definitions>
```

```

</wsdl:message>
<wsdl:message name="HelloWorldSoapOut">
  <wsdl:part name="parameters" element="tns:HelloWorldResponse" />
</wsdl:message>
<wsdl:message name="AddSoapIn">
  <wsdl:part name="parameters" element="tns:Add" />
</wsdl:message>
<wsdl:message name="AddSoapOut">
  <wsdl:part name="parameters" element="tns:AddResponse" />
</wsdl:message>
<wsdl:message name="MultSoapIn">
  <wsdl:part name="parameters" element="tns:Mult" />
</wsdl:message>
<wsdl:message name="MultSoapOut">
  <wsdl:part name="parameters" element="tns:MultResponse" />
</wsdl:message>
<wsdl:portType name="myServiceSoap">
  <wsdl:operation name="HelloWorld">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Returns a hello
message</wsdl:documentation>
    <wsdl:input message="tns:HelloWorldSoapIn" />
    <wsdl:output message="tns:HelloWorldSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="Add">
    <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Returns the
sum</wsdl:documentation>
    <wsdl:input message="tns:AddSoapIn" />
    <wsdl:output message="tns:AddSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="Mult">
    <wsdl:input message="tns:MultSoapIn" />
    <wsdl:output message="tns:MultSoapOut" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="myServiceSoap" type="tns:myServiceSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="HelloWorld">
    <soap:operation soapAction="http://tempuri.org/HelloWorld" style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Add">
    <soap:operation soapAction="http://tempuri.org/Add" style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Mult">
    <soap:operation soapAction="http://tempuri.org/Mult" style="document" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:binding name="myServiceSoap12" type="tns:myServiceSoap">
  <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="HelloWorld">
    <soap12:operation soapAction="http://tempuri.org/HelloWorld" style="document" />
    <wsdl:input>
      <soap12:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap12:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="Add">
    <soap12:operation soapAction="http://tempuri.org/Add" style="document" />

```

```

<wsdl:input>
  <soap12:body use="literal" />
</wsdl:input>
<wsdl:output>
  <soap12:body use="literal" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="Mult">
  <soap12:operation soapAction="http://tempuri.org/Mult" style="document" />
  <wsdl:input>
    <soap12:body use="literal" />
  </wsdl:input>
  <wsdl:output>
    <soap12:body use="literal" />
  </wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="myService">
  <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Hello from
myService</wsdl:documentation>
  <wsdl:port name="myServiceSoap" binding="tns:myServiceSoap">
    <soap:address location="http://localhost:1046/WebService/myService.asmx" />
  </wsdl:port>
  <wsdl:port name="myServiceSoap12" binding="tns:myServiceSoap12">
    <soap12:address location="http://localhost:1046/WebService/myService.asmx" />
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

**Σχήμα 12.1.** Περιγραφή σε WSDL της myservice

## **Παραρτήματα, Βιβλιογραφία και Χρήσιμα Links**

## Παράρτημα Α: Εγκατάσταση Java και Σχετικών Αρχείων (Jars)

Θα δούμε στο Παράρτημα αυτό την εγκατάσταση (Windows 2000 ή XP) της Java και των σχετικών αρχείων (jar files) που αναφέρθηκαν και χρησιμοποιούνται στα παραδείγματα αυτών των σημειώσεων.

### Εγκατάσταση Java

Στο site της Java (<http://java.sun.com>) βρίσκουμε και κατεβάζουμε το [J2SE 1.4.2 SDK](#). Διαλέγουμε το SDK και όχι το JRE, επίσης καλό είναι να κατεβάσουμε και το documentation το οποίο περιέχει όλες τις περιγραφές των μεθόδων.

Στην εγκατάσταση που γίνεται με διπλό κλικ στο αρχείο διαλέγουμε να εγκαταστήσουμε το SDK (j2sdk1.4.2\_03) και το Java Runtime Environment (jre). Δεν είναι υποχρεωτικό να εγκαταστήσουμε το Code Samples και Source code. Καλό είναι η εγκατάσταση να γίνει σε κάποιο root directory (C:\ ή D:\). Εδώ υποθέτουμε το C:\

Για να είναι ποιο εύκολη η χρήση του μεταγλωττιστή της Java (Java Compiler – javac) καλό είναι να θέσουμε κάποια environment variables ως εξής:

Πατάμε δεξί κλικ στο My Computer, Control Panel, System, Advanced, Environment Variables όπου, φτιάχνουμε μια μεταβλητή με όνομα JAVA\_HOME και τιμή το φάκελο της εγκατάστασης μας (π.χ. C:\j2sdk1.4.2\_03) και μια μεταβλητή (αν δεν υπάρχει!) με όνομα Path και τιμή το φάκελο bin της εγκατάστασης της Java ακολουθούμενο από ';' (π.χ. C:\j2sdk1.4.2\_03\bin;).

### Εγκατάσταση Tomcat

Στο site του μαθήματος δίδεται το jakarta-tomcat-7.7.2.zip. Εξάγοντας αυτό σε οποιοδήποτε folder (έστω στο C:\), δημιουργείται subfolder με όνομα jakarta-tomcat-7.7.2. Για να τρέξει ο Tomcat αρκεί να μπούμε στο ...\\bin και να γράψουμε startup. Ο Tomcat ευρίσκεται στο <http://jakarta.apache.org>, απ' όπου κατεβάσαμε το binary αρχείο στο site του μαθήματος.

### Εγκατάσταση επί πλέον αρχείων

Για να εγκαταστήσουμε τα επιπλέον αρχεία - βιβλιοθήκες (Java Archive – επέκταση .jar) πρέπει να αντιγράψουμε τα αρχεία jdom.jar, xalan.jar, xmlrpc-1.2-b1.jar, xerces.jar, soap.jar, mail.jar, activation.jar, axis.jar, servlet.jar (που βρίσκονται στο site του μαθήματος) στους παρακάτω δύο φακέλους:

(JAVA\_HOME)\jre\lib\ext (... και C ή D:\Program Files\Java\j2re1.4.2\_03\lib\ext)

Με αυτό καλύπτουμε τις ανάγκες του client. Στην πλευρά του βλέπει μόνο τις classes της Java. Για τον server τα ίδια αρχεία πρέπει να ριχθούν και στο ...\\jakarta-tomcat-7.7.2\lib\common (αν υπάρχει πρόβλημα και στο ...\\jakarta-tomcat-7.7.2\lib).

Μέσα στο jdom.jar, κατεβασμένο από <http://www.jdom.org> ευρίσκεται ο jdom.jar, καθώς και ο Xerces parser.

Το xmlrpc-1.2-b1.jar είναι κατεβασμένο από <http://classic.helma.at/hannes/xmlrpc/>, πού μας παραπέμπει στο <http://www.jdom.org>.

Το xerces.jar από <http://xml.apache.org>.

Το soap.jar από <http://xml.apache.org/dist/soap>.

Το mail.jar από <http://java.sun.com/products/javamail>.

Το activation.jar από <http://java.sun.com/products/beans/glasgow/jaf.html>.

Μέσα στο axis.jar, κατεβασμένο από <http://xml.apache.org/axis> ευρίσκεται ο TCP Monitor.

### Μεταγλώττιση και τρέξιμο των παραδειγμάτων

Κάθε παράδειγμα που υπάρχει στις σημειώσεις τρέχεται ως εξής: Αντιγράφουμε τον κώδικά σε ένα txt αρχείο και κατόπιν σώζουμε το αρχείο με όνομα, το όνομα της class και κατάληξη .java. Ανοίγουμε ένα DOS Prompt και μπαίνουμε στο directory που σώσαμε το αρχείο μας.

Κατόπιν μεταγλωττίζουμε το αρχείο με

```
javac <όνομα αρχείου>.java.
```

οπότε δημιουργείται το αρχείο <όνομα αρχείου>.class που περιέχει το bytecode. Για να εκτελέσουμε τον κώδικα γράφουμε java <όνομα αρχείου> (χωρίς επίθεμα), εφόσον είμαστε στο ίδιο folder με το αρχείο αυτό.

Εάν μία class μέσα στο αρχείο program1.java αναφέρεται σε άλλη πού ευρίσκεται στο αρχείο program2.java, τότε πρέπει τα program1.java και program2.java να είναι στο ίδιο folder. Τότε η μεταγλώττιση, javac program1.java, του πρώτου αρχείου, προκαλεί και την αυτόματη μεταγλώττιση του δεύτερου και βλέπουμε να δημιουργούνται στο ίδιο folder τα αρχεία program1.class και program2.class. Για να τρέξει το program1.class χρησιμοποιεί βεβαίως το program2.class, το οποίο πρέπει να παραμένει στο ίδιο folder. Για ένα πιο εξελιγμένο σενάριο, βλ. την επόμενη παράγραφο.

### Δημιουργία αρχείου αρχείου .jar (πού περιλαμβάνει πολλές classes)

Στην πλευρά του server, ο bytecode πού εκτελεί την υπηρεσία πού δημιουργήσαμε χρειάζεται στις (περιπτώσεις πού περιγράφονται) να δίδεται στην μορφή αρχείου .jar. Αυτό είναι ένα συμπιεσμένο αρχείο (Java Archive, κάτι σαν .zip), το οποίο εμπεριέχει τον κώδικα μίας ή περισσοτέρων classes. Έστω ότι θέλουμε να δημιουργήσουμε program12.jar, πού να περιέχει το κώδικα των program1 και program2 όπως πάνω. Έστω subfolder το όνομα ενός subfolder (πού θα δημιουργήσουμε παρακάτω), subfolder του folder όπου εργαζόμαστε

Σαν πρώτη γραμμή των program1.java και program2.java πού ευρίσκονται στο folder γράφουμε

```
package subfolder;
```

Μεταγλωττίζουμε πρώτα το program2.java, εφόσον αυτό χρησιμοποιείται από το program1.java. Μετά δημιουργούμε το subfolder, στο οποίο ρίχνουμε το program2.class. Επαναλαμβάνουμε τι ίδιο με το program1.java. Με τα program1.class και program2.class μέσα στο folder/subfolder εκτελούμε από το folder την εντολή

```
jar cvf program12.jar subfolder/program1.class subfolder/program2.class
```

Το program12.jar εμφανίζεται δημιουργημένο μέσα στο folder. Το 'manifest' πού αναφέρεται στην απόκριση της εντολής jar cvf ... είναι μέτα-πληροφορία, δηλ. πληροφορία μέσα στο .jar για το τι διατίθεται μέσα σε αυτό. Αν ανοίξουμε το program12.jar με κάποιο πρόγραμμα πού εμφανίζει δεκαεξαδικά (π.χ. το UltraEdit), θα δούμε να υπάρχουν και τα ASCII strings subfolder/program1.class και subfolder/program2.class. Παρατηρούμε επίσης ότι το όνομα program12 του .jar είναι αυθαίρετο και από μας καθορισμένο, άσχετο από το όνομα του subfolder. Σε οποιονδήποτε δοθεί το program12.jar, αν είναι ικανός να το 'ανοίξει' (όπως ο Tomcat), θα βρει στη διάθεσή του τις program1.class και program2.class. Ο Tomcat δεν ενδιαφέρεται για το όνομα του οποιουδήποτε .jar. Απλά τα ανοίγει όλα (π.χ. αυτά πού έχουν ριχθεί στο ...\\jakarta-tomcat-7.7.2\\lib\\common) και βλέπει τα ονόματα όλων των classes (εδώ σαν subfolder/program1.class και subfolder/program2.class).

## Παράτημα Β: Χρησιμοποιούμενες Command Line Εντολές

Οι πιο πολύπλοκες χρησιμοποιούμενες Command Line εντολές παρατίθενται προς διευκόλυνση (για cut & paste). Θα χρειασθεί πιθανόν να αλλαχθούν μερικά *ορίσματα* ή κάτι από τα *path segments*.

### Μετασχηματισμός XSLT μέσω xalan

```
java org.apache.xalan.xslt.Process -IN vehicles.xml -XSL vehicles_transformer.xml  
(και επιπλέον -OUT result.xml, αν το επιθυμούμε)
```

### Δημιουργία .jar

```
jar cvf BV.jar BVShop/VehicleBean.class BVShop/BVCatalog.class  
jar cvf MServer.jar MSS/MessagingServer.class  
(μετά τοποθέτηση των .jar στο ...\jakarta-tomcat-7.7.2\lib\common του Tomcat)
```

### Directory για startup, shutdown και εντολές ελέγχου του Tomcat

```
cd C:\JavaDownloads\jakarta-tomcat-7.7.2\bin
```

### Για σήκωμα, κατάργηση, κλπ των διαφόρων webservices στον Tomcat

```
java org.apache.soap.server.ServiceManagerClient  
http://localhost:8080/soap/servlet/rpcrouter deploy C:\BVCatalogDD.xml
```

```
java org.apache.soap.server.ServiceManagerClient  
http://localhost:8080/soap/servlet/rpcrouter undeploy urn:BVehicleCatalog
```

```
java org.apache.soap.server.ServiceManagerClient  
http://localhost:8080/soap/servlet/rpcrouter query urn:BVehicleCatalog
```

```
java org.apache.soap.server.ServiceManagerClient  
http://localhost:8080/soap/servlet/rpcrouter deploy C:\MessagingServiceDD.xml
```

### Για εμφάνιση των εγκατεστημένων στον Tomcat webservices

```
java org.apache.soap.server.ServiceManagerClient  
http://localhost:8080/soap/servlet/rpcrouter list
```

### Για να τρέξει ο TCP Monitor

```
java org.apache.axis.utils.tcpmon
```

### Για να τρέξει ο client (μέσα από το folder όπου υπάρχει η σχετική .class)

... προκειμένου για SOAP **RPC**

```
java BVAdderLister http://localhost:8080/soap/servlet/rpcrouter "Smart" "Swatch" "2001"
```

... προκειμένου για SOAP **Messaging** (το ... 8081, λόγω TCP Monitor!)

```
java MessagingClient http://localhost:8081/soap/servlet/messagerouter  
C:\vehiclesSOAP.xml
```

### Ενδιαφέρον κείμενο για configuration και λειτουργία Tomcat

[file:///C:/JavaDownloads/jakarta-tomcat-7.7.2/doc/tomcat-ug.html#what\\_servlets\\_jsps](file:///C:/JavaDownloads/jakarta-tomcat-7.7.2/doc/tomcat-ug.html#what_servlets_jsps)



## Βιβλιογραφία και Χρήσιμα Links

### Γενικά για Internet και web

Το **Internet Engineering Task Force (IETF)**, [www.ietf.org](http://www.ietf.org), ιδρύθηκε το 1986 και είναι ο οργανισμός προώθησης των σχετικών προτύπων. Για το web έχουμε το **World Wide Web Consortium (W3C)**, <http://www.w7.org>. το οποίο ιδρύθηκε το 1994 από τον Tim Berners-Lee (‘εφευρέτη’ του web στα πλαίσια της εργασίας του στο CERN για την ανάπτυξη υποδομής ανταλλαγής δεδομένων πειραμάτων φυσικής των σωματιδίων).

**Τεχνολογίες XML**, [www.w7.org/xml](http://www.w7.org/xml)

Οι παραπάνω σημειώσεις περιέχουν απλουστευμένη ύλη από τα  
**Beginning XML**, David Hunter, Wrox  
**Java & XML**, Brett Mc Laughlin, **O’Reilly**  
**Java and XSLT**, Eric Burke, **O’Reilly**  
**XSLT**, Doug Tidwell, **O’Reilly**

Για έτοιμες λύσεις (και πολύπλοκων) προβλημάτων XSLT  
**XSLT Cookbook**, Sal Mangano, **O’Reilly**

**Beginning Java Web Services**, H.Bequet, M.M. Kunnumpurath, S. Rhody, A. Tost, Wrox

Για το πώς μπορούν να φανούν άμεσα χρήσιμες οι γνώσεις σε XML  
**Office 2003 XML**, Evan Lenz, Mary McRae, Simon St.laurent, **O’Reilly**  
(Περιγράφει το πώς η Microsoft εξάγει πλέον την εσωτερική αναπαράσταση αρχείων .doc, .xls, .mdb σε μορφή XML, για περαιτέρω εξωτερική επεξεργασία σε οποιοδήποτε περιβάλλον, χρήση και επανεισαγωγή σε αυτά τα αρχεία με τα αντίστοιχα προγράμματα του Office).

### Για Java

**Java, how to program, Deitel & Deitel, Prentice Hall.** Όλα τα εισαγωγικά και επιπλέον τα βασικά των Multithreading, Database Connectivity, Servlets, RMI, JavaBeans με καλά παραδείγματα. Σε πολύ μεγάλη πληρότητα -ίσως υπερβολική.

**Core Java 2, Volume I Fundamentals Cay Horstman, Gary Cornell [Prentice Hall, the Sun Microsystems Press, Java Series].** Το καλύτερο. Δυσκολότερο του παραπάνω αλλά πραγματικά επικεντρωμένο στην ουσία.

**Java 2 Programmer, Bill Brogden & Marcus Green, QUW Certification.** Για certification exam . Πολύ περιεκτικό αλλά πληρέστατο και χρήσιμο με πολλά quiz μεγάλης διδακτικής αξίας. Δεν αρκεί μόνο του.

**Java Cookbook, Ian Darwin, O’ Reilly.** Όχι για αρχή, αλλά πολύ χρήσιμο για δεύτερο στάδιο.

Στο [www.w3schools.com](http://www.w3schools.com) tutorials για JavaScript, DOM, XSLT.