ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ & ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

# Εναρμόνιση παραδειγμάτων δυναμικών σελίδων σε διάφορες τεχνολογίες

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΔΗΜΗΤΡΙΟΣ Ι. ΚΟΤΣΑΛΟΣ

**Επιβλέπων :** Γεώργιος Στασινόπουλος
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2012

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ & ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

# Εναρμόνιση παραδειγμάτων δυναμικών σελίδων σε διάφορες τεχνολογίες

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΔΗΜΗΤΡΙΟΣ Ι. ΚΟΤΣΑΛΟΣ

**Επιβλέπων :** Γεώργιος Στασινόπουλος
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 23 Οκτωβρίου 2012.

| ..... | ..... | ..... |
|---|---|---|
| Γεώργιος Στασινόπουλος<br>Καθηγητής Ε.Μ.Π. | Ευστάθιος Δ. Συκάς<br>Καθηγητής Ε.Μ.Π. | Μιχαήλ Ε. Θεολόγου<br>Καθηγητής Ε.Μ.Π. |

Αθήνα, Οκτώβριος 2012

.....
Δημήτριος Ι. Κότσαλος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

# ΠΕΡΙΛΗΨΗ

Τα τελευταία χρόνια,η τεχνολογία γύρω από τους ηλεκτρονικούς υπολογιστές αναπτύσσεται με τέτοιους ρυθμούς προκαλώντας ανάπτυξη σε όλους τους τομείς της ζωής μας.Για παράδειγμα ο Παγκόσμιος Ιστός αποτελεί πια αναπόσπαστο κομμάτι της κάθε επιχείρησης σήμερα και δεν χρησιμοποείται μόνο για προσωπική χρήση.Επιπλέον ολοένα και περισσότεροι χρήστες δικτυώνονται,περιηγούνται στο διαδίκτυο και συναναστρέφονται με άλλους χρήστες ,μέσω των ηλεκτρονικών κοινοτήτων κοινωνικής δικτύωσης.Όμως οι πιο πολλοί από εμάς έχουμε την ιδιότητα του χρήστη-πελάτη.Αναφερόμαστε μόνο στη πλευρά του πελάτη(client) και το μοναδικό μας εργαλείο είναι ο browser μπροστά στην οθόνη μας.Το τι γίνεται στη πλεύρα του server, το πως και από που έρχεται στην οθόνη μας η πληροφορία που ζητήσαμε, μας είναι σχεδόν άγνωστο.Η διπλωματική αυτή είναι μια ευκαιρία για τους χρήστες να γνωρίσουν τι περίπου βρίσκεται και συμβαίνει πίσω από μιά απλή ‘αναζήτηση’ στον ιστότοπο.Φυσικά το ρόλο του client και το ρόλο του server θα τον έχει ο χρήστης.Σημασία έχει να κατανοήσει τα όσα συμβαίνουν με ένα click στον browser.Απαραίτητο εργαλείο σε αυτή μας τη προσπάθεια είναι ένας web server που εδώ θα χρησιμοποιηθεί ο Apache Tomcat.Θα γνωρίσουμε πολλές από τις ικανότητες και ιδιότητες του.Επίσης θα γνωρίσουμε λίγο καλύτερα κάποιες ιδιότητες του χρήστη-πελάτη και σε αυτό θα μας βοηθήσει το client-side programming με τη χρήση της γλώσσας Javascript αλλά και της τεχνολογίας AJAX.Θα ασχοληθούμε τέλος με την ανταλλαγή μηνυμάτων SOAP.Για την κατανόηση των παραπάνω θα χρησιμοποιηθεί μία σειρά από παραδείγματα/εφαρμογές καθώς και οδηγίες ώστε κάποιος να μπορεί εύκολα να τις τρέξει στον προσωπικό του υπολογιστή.

Λέξεις κλειδιά: Tomcat, πλευρά του πελάτη,πλευρά του server,AJAX,javascript,SOAP

# ABSTRACT

In recent years, the technology surrounding computers is growing so rapidly causing growth in all areas of our life.For example, the Web is now an integral part of any business today and not used for personal use only.Additional,there are more and more networked users, browsing the web and interacting with other users through online social networking communities.But most of us have the status of user-client.We refer only on the client side and our only tool is a browser in front our screen. What happens on the side of the server, and how comes on the screen the information requested, is almost unknown. This diploma thesis is an opportunity for people to learn about what is going on behind a simple 'search' in World Wide Web.Of course ,user will have the role of the client and the role of the server. Our goal is to understand what is going on with one click on his browser.Our essential tool in this effort is a web server.Here we will use the Apache Tomcat.We are going to meet many of his skills and attributes.Additional, we will learn some properties of the user/client and on this purpose we will be helped by client-side programming,using the Javascript language and technology AJAX.Finally we will meet SOAP messages To understand the above we will use a series of examples / applications and instructions that anyone can easily run these on his personal computer.

Key-words:Tomcat,client-side,server-side,AJAX,javascript,SOAP

# ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή κ. Γεώργιο Στασινόπουλο για τη δυνατότητα που μου έδωσε να ασχοληθώ με ένα τόσο ενδιαφέρον θέμα όπως είναι το Διαδίκτυο και οι εφαρμογές του και να εκπονήσω τη διπλωματική μου εργασία στον τομέα αυτό.
Τέλος, ευχαριστώ τους γονείς μου, Ιωάννη και Θεοδώρα καθώς και τα αδέρφια μου, Θωμά και Σταυρούλα που στάθηκαν βράχος πίσω μου όλα αυτά τα χρόνια και που πάντα με στηρίζουν και δείχνουν κατανόηση στις επιλογές μου ακόμα και αν δεν συμφωνούν πάντα μαζί μου.

# ΠΕΡΙΕΧΟΜΕΝΑ

# ΠΙΝΑΚΑΣ ΣΧΗΜΑΤΩΝ

# ΠΙΝΑΚΑΣ ΚΩΔΙΚΩΝ

# 1

# Εισαγωγή

Σε αυτό το κεφάλαιο,στη παράγραφο 1.1 γίνεται μια μικρή περιγραφή του αντικειμένου της διπλωματικής εργασίας ,ενώ στη παράγραφο 1.2 γίνεται μια αναφορά στη δομή και την οργάνωση της.

## 1.1 Περιγραφή

Η ραγδαία ανάπτυξη στην τεχνολογία των δικτύων με αποκορύφωμα τη δημιουργία του World Wide Web (www) η εύκολη χρήση του υπολογιστή του σήμερα και των διαδικτυακών εφαρμογών καθώς και η ανάπτυξη των social networks μας έχουν πραγματικά καθηλώσει μπροστά στην οθόνη του υπολογιστή μας.Η περιήγηση είναι πιο συναρπαστική από ποτέ και η κοινωνική δικτύωση αποκτά όλο και περισσότερους φίλους.Θα ήταν εξίσου συναρπαστικό να γνωρίσουμε λίγο περισσότερο το διαδίκτυο και τις εφαρμογές του μέσα απο μια σειρά παραδειγμάτων/εφαρμογών που σκοπό έχουν να μας δείξουν τι συμβαίνει στον browser μας,πως εμείς στέλνουμε και πως απαιτούμε πληροφοριές από το web server και τέλος πώς αυτός τις διαχειρίζεται και μας απαντά.Απαραίτητο εργαλείο είναι ο Tomcat (εδώ στην τελευταία του έκδοση 7.0.30) και αυτόν θα χρησιμοποιήσουμε ως server.Για την υλοποίηση αυτής της προσπάθειας θα γνωρίσουμε την πλευρά του χρήστη/πελάτη χρησιμοποιώντας javascript και την τεχνολογία AJAX αλλά και τη πλευρά του server χρησιμοποιώντας java servlets,jsp σελίδες ,το πρωτόκολλο SOAP και φυσικά τον Tomcat και τις ιδιότητες του.

## 1.2 Οργάνωση

Η διπλωματική εργασία αποτελείται από 7 κεφάλαια.

Στο πρώτο κεφάλαιο,στο οποίο βρισκόμαστε τώρα γίνεται μια περιγραφή του θέματος και του σκοπού αυτής της διπλωματικής εργασίας.Δίνεται επίσης η δομή και η οργάνωση της.

Στο δεύτερο κεφάλαιο συναντάμε τον Tomcat και αναλύουμε τους λόγους για τους οποίους τον επιλέξαμε να έχει το ρόλο του server στις εφαρμογές μας.

Στο τρίτο κεφάλαιο θα γνωρίσουμε τη πλευρά του server.Θα δούμε τι είναι τα servlets και πως με μια εφαρμογή μπορούμε να στήσουμε έναν server μαζί με τη δομή του και τις ιδιότητες του.Θα γνωρίσουμε το server-side scripting

Στο τέταρτο κεφάλαιο αντιθέτως θα προσθέσουμε με τη βοήθεια της γλώσσας Javascript διάφορα scripts στην .html σελίδα μας για περισσότερη αλληλεπίδραση με το χρήστη αλλά και για να δούμε πως λειτουργεί client-side scripting.

Στο πέμπτο κεφάλαιο θα γνωρίσουμε την τεχνολογία AJAX, μια τεχνολογία που συνδυάζει το server-side και το client-side scripting.για περισσότερη αλληλεπίδραση και εφέ στον browser μας.

Στο έκτο κεφάλαιο θα δούμε πως ανταλάσσονται μηνύματα σύμφωνα με το πρωτόκολο SOAP και πολλές από τις ιδιότητες του.Απαραίτητο εδω εργαλείο πέραν του Tomcat είναι ο Tcp Monitor ώστε να βλέπουμε τη διακίνηση των μηνυμάτων.

Στο έβδομο κεφάλαιο γίνεται αναφορά στην υλοποίηση των όσων θα δούμε στα προηγούμενα κεφάλαια.Εγκατάστασεις αλλά και οδηγίες για την πραγματοποίηση των εφαρμογών.

# 2

# Γνωρίζοντας τον Tomcat

Στο κεφάλαιο αυτό θα γνωρίζουμε το βασικό μας εργαλείο για την διεκπεραίωση του στόχου μας ,τον Tomcat.

## 2.1 Web Server

Οι web servers αποτελούν ένα πολύ κοινό μέσο παρουσίασης δεδομένων. Βασίζονται σε ένα τυποποιημένο πρωτόκολλο που επιτρέπει την ανταλλαγή δεδομένων, που ονομάζεται HyperText Transfer Protocol (HTTP). Διαθέτουν επίσης πολύ καλή τεκμηρίωση και βοήθεια, η οποία επιτρέπει στους συντάκτες των σελίδων να εστιαστούν εξ ολοκλήρου στη δημιουργία της σελίδας τους και να μην ασχοληθούν με τις λεπτομέρειες της λειτουργίας του web server. Ένα από τα πλεονεκτήματα του προταθέντος συστήματος είναι ότι ο καθένας μπορεί να δημιουργήσει τη σελίδα του στην αφηρημένη γλώσσα που χρησιμοποιεί το συγκεκριμένο σύστημα (XForms) και να την ανεβάσει σε έναν web server έτσι ώστε να μπορέσει να τη δημοσιεύσει στον υπόλοιπο κόσμο. Ελάχιστη ή καμία επέμβαση δε χρειάζεται από το διαχειριστή του συστήματος. Για όλους αυτούς τους λόγους, η χρήση ενός web server καθιστά το σύστημα πιο ευπροσάρμοστο και εύχρηστο για το συντάκτη των σελίδων.
Ο web server που χρησιμοποιείται από το σύστημα, όπως προαναφέρθηκε, είναι ο Apache web server. Επιλέχτηκε εξαιτίας του γεγονότος ότι χρησιμοποιείται εκτενώς, διαθέτει άριστη τεκμηρίωση, είναι εύκολος στην εγκατάσταση και τη διαχείριση, και τέλος η χρήση του είναι δωρεάν.

## 2.2 Servlet container

Η χρήση ενός web server και μόνο δεν είναι αρκετή, εντούτοις. Το σύστημα είναι γραμμένο σε γλώσσα Java, υπό τη μορφή servlet. Ένα servlet container περιλαμβάνει το στοιχείο (component) ενός web server που φιλοξενεί και αλληλεπιδρά με εφαρμογές Java servlets. Ελέγχει τις εφαρμογές αυτές που αναπτύσσονται εντός του Web server και έχει την ευθύνη της προώθησης των αιτήσεων προς και των απαντήσεων από αυτά. Τα Servlets παρέχουν λειτουργικότητα που βασίζεται σε συστατικά («component-based»), και είναι ανεξάρτητη από την υπολογιστική πλατφόρμα που χρησιμοποιείται («platform-independent»). Υλοποιούν εφαρμογές βασισμένες στο Web, οι οποίες δεν έχει τους περιορισμούς στην απόδοση που έχουν τα προγράμματα CGI. Τα Servlets είναι ανεξάρτητα από τον web server και την πλατφόρμα στην οποία λειτουργεί: το μόνο που χρειάζονται είναι ένα servlet container, το πλαίσιο δηλαδή το οποίο τους επιτρέπει να

αλληλεπιδρούν με τον web server κατά τη διάρκεια της εκτέλεσης του προγράμματος. Αυτό το servlet container παρέχει τις μεθόδους που καλούνται για να λειτουργήσει το σύστημα. Ένας τέτοιος servlet container είναι ο Tomcat που θα δούμε παρακάτω και θα είναι το βασικό εργαλείο μας για την δημιουργία αλλά και την ανάπτυξη των εφαρμογών μας.

## 2.3 Tomcat

Ο Tomcat είναι ένα open-source λογισμικό βασισμένο στην γλώσσα Java και την αρχιτεκτονική του servlet container,το οποίο χρησιμοποιείται για να φιλοξενήσει Java εφαρμογές.Είναι δηλαδή ένα περιβάλλον μέσα από το οποίο εκτίθενται στο διαδίκτυο όσα εκτελούν τα διάφορα από αυτόν φιλοξενούμενα servlets.Αρχικά αναπτύχθηκε ως Jakarta Tomcat.Εξαιτίας της αύξησης της ζήτησης αργότερα φιλοξενήθηκε ως ένα ξεχωριστό project, το Apache Tomcat ,το οποίο υποστηρίζεται από την The Apache Software Foundation.Ο Tomcat υλοποιεί τις Java Servlet και JavaServer Pages (JSP) προδιαγραφές και παρέχει ένα ιδανικό περιβάλλον για Java εφαρμογές.Περιλαμβάνει πολλές νέες δυνατότητες που τον ανάγουν σε μια χρήσιμη πλατφόρμα για την δημιουργία αλλά και την ανάπτυξη διαδικτυκών εφαρμογών και υπηρεσιών.Ο Tomcat τρέχει σε ένα περιβάλλον ενός κοινού web ή HTTP server,στη συγκεκριμένη περίπτωση του Apache.Αυτός συνυπάρχει αλλά δε φαίνεται στα παρακάτω,καθόσον ασχολείται στο να δέχεται και να αποστέλει HTTP μηνύματα.Επιλέγουμε τον Tomcat για πολλούς λόγους,μερικούς από τους οποίους είδαμε πιο πάνω ,αλλά και για δύο ακόμα σημαντικούς :

-Η διακίνηση του είναι δωρεάν.

-Υπάρχει πάρα πολύ υλικό(Documentation) τόσο σε βιβλία όσο και στο διαδίκτυο που περιγάφει τις λειτουργίες του.

Για να συνεχίσουμε στο επόμενο κεφάλαιο είναι προτεινόμενο να δούμε την υλοποίηση στο κεφάλαιο 7 έτσι ώστε να μπορούμε να τρέξουμε τις εφαρμογές που θα ακολουθήσουν στα επόμενα κεφάλαια.

# 3

# 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 πού επικοινωνεί με τους web clients κάτω από το περιβάλλον web πού εξετάζομε 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 έχομε να κάνομε 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 μεταφέρομε την πληροφορία και μάλιστα υπό την μορφή σελίδων αποτελουμένων 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.

## 3.1. Το 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 πού 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 διαμορφώσομε 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 πού 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 πού μετέφερε το POST. Τώρα η getParameterNames() επιστρέφει ένα Enumeration.
- όλα τα cookies πού 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. Δίδομε 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 εμφάνιση. Έτσι με την μέθοδο void setContentType (String type) ορίζομε 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. Δίδομε 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 βλέπομε 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 3.1.** Το HTTPGetServlet, σαν στοιχειώδες servlet

Με την μέθοδο setContentType ορίζομε 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 ταυτίζομε με την δική μας output . Από κει και πέρα, μπορούμε να κάνομε 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 πού 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 έχομε 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 προκαλέσομε την εμφάνιση της παραπάνω σελίδας, κτυπάμε τοπικά με τον browser

http://localhost:8080/<*κάποιο όνομα που οδηγεί στο HTTPGetServlet*>

πού 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 φροντίσομε τώρα να βάλομε τον κώδικα του servlet που δημιουργήσαμε στον server, μαζί με την επιπλέον πληροφορία που σχετίζεται με αυτόν.

## 3.2. Το 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 πού αποκαλείται servlet container. Αυτή συνυπάρχει μέσα στην γενική αρχιτεκτονική του εκάστοτε server και εδώ θα περιγράψομε 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:\ apache-tomcat-7.0.30 \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 πού αναγνωρίζονται έτσι από τον Tomcat. Κατ' αρχήν φορτώνομε οποιοδήποτε στατικό αρχείο (π.χ. index.html, mytest.html ή ακόμη test.txt, κλπ.) κάτω από το folder *myServletDir*, δηλ. C:\ apache-tomcat-7.0.30\webapps*myServletDir*\test.txt και το βλέπομε από τον browser με http://localhost:8080/*myServletDir*/test.txt. Ειδικά το index.html εμφανίζεται αν προσδιορίσομε 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 3.1.** Ο κατάλογος myServletDir της εφαρμογής μέσα στον server
(dir του Tomcat, κάτω από το οποίο κρεμάμε αυτήν και άλλες εφαρμογές είναι το C:\ apache-tomcat-7.0.30*webapps*)

Τώρα τοποθετούμε την μεταγλωττισμένη class του servlet μας σύμφωνα με το παραπάνω μονοπάτι:
C:\apache-tomcat-7.0.30\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 αντιστοιχήσεις μεταξύ του ονόματος της 'εφαρμογής', του ονόματος πού 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 πάμπολλά άλλα, σχετικά με την έκθεση (deployment) της ‘εφαρμογής’ στο διαδίκτυο. Για να αναγνωρίζεται λοιπόν το servlet μας πάμε και διαμορφώνομε 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 3.2.** To 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 αντιστοιχήσεις. Μέσα στο αρχείο web.xml ο συνδετικός 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 αντιστοιχήσεις 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 έχομε στατικές σελίδες, αλλά για το WEB-INF, που περιέχει όλα όσα είδαμε πιο πάνω, και επίσης τους υποκαταλόγους lib και other_directories σε πιο εξελιγμένες 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 3.1 συμπιεσμένη σε ένα μοναδικό αρχείο. Τέτοιο myServletDir.war μπορούμε να κατασκευάσομε 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 θεωρήσομε 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 πάρομε και να το τρέξομε 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 αποκτήσομε 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 αλλάξομε 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 classses), 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 3.1).

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 κάνομε τίποτα, το περιβάλλον αυτό αντιμετωπίζει πολλαπλές κλήσεις από πολλούς clients προς την ίδια υπηρεσία συγχρόνως. Στην περίπτωσή μας εδώ, τούτο επιτυγχάνεται με το multi threading της Java.

## 3.3. 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 3.3.** (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 έχομε ακόμη

φροντίσει για την υποδοχή της στη μεριά του server, αλλά εν τούτοις βλέπομε να κωδικοποιούνται κατά το γνωστά στο GET οι τιμές πού 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΄ πού 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).

## 3.3.1. 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 βλέπομε αυτήν να θυμάται τις επιλογές που κάναμε την προηγουμένη. 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 συσσωρεύονται 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 theree as a cookie
  c.setMaxAge( 120 );  // seconds until cookie removed
  response.addCookie( c );  // must preceede 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 3.2.** 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, πού 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 πού 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 φτιάχνομε 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 έχομε 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 πού 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 3.3.** 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 πού 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 γράψομε 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 3.4.** 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 3.1. Αντ' αυτού προτιμάμε να εκθέσουμε και το HTTPGetServlet και το CookieServlet μαζί για να γνωρίσομε και αυτήν την περίπτωση. Προς τούτο κατασκευάζομε ένα myExamplesDir και κάτω από αυτό το index.html και το WEB-INF. Μέσα στο τελευταίο βάζομε 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:\ apache-tomcat-7.0.30\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

(υποτίθεται ότι συνυπάρχουν κάτω από το webapps και το myServletDir (με το web.xml του Σχήματος 3.2) και το myExamplesDir (με το web.xml του παρακάτω Σχήματος 3.4)).
Στο παρακάτω web.xml βλέπομε 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>
```

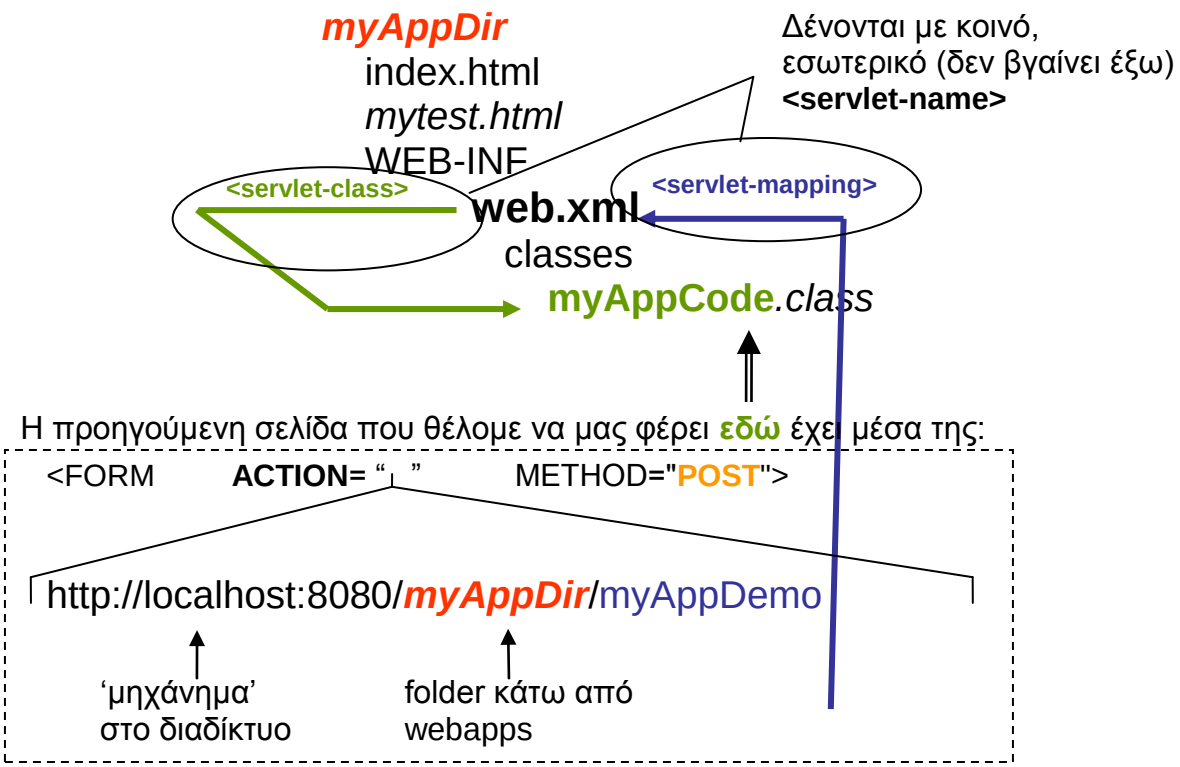
```
 </servlet>

  <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/myExamplesDir/firstCookieDemo/* (any thing)
                        will lead  to CookieServlet  -->
    <servlet-name>CookieDemo</servlet-name>
    <url-pattern>/firstCookieDemo/*</url-pattern>
    </servlet-mapping>
</web-app>
```

**Σχήμα 3.4.** Το web.xml για δύο εφαρμογές (HTTPGetServlet.class και CookieServlet.class)

Η χρήση της πληροφορίας του παραπάνω web.xml από τον server στην διάρκεια της εκτέλεσης της εφαρμογής web ανακεφαλαιώνεται στο παρακάτω Σχήμα 3.5. Για λόγους ταχύτητας η πληροφορία αυτή δεν εξάγεται από το web.xml την ώρα της εκτέλεσης. Ο server διαβάζει τα web.xml όλων των εφαρμογών web που του έχουν ανατεθεί (όλες σαν την myAppDir κάτω από το webapps) κατά το startup και ενημερώνει σχετικούς εσωτερικούς πίνακες.

Κάτω από webapps του Tomcat:




**Σχήμα 3.5.** Η χρήση web.xml κατά την εκτέλεση της εφαρμογής

Και τέλος η δοκιμή. Πατάμε στον browser την φόρμα DestinationSelection.html και επιλέγομε μια ήπειρο. Με το submit η επιλογή μας φθάνει στον server, ο οποίος την επιβεβαιώνει. Κλείνομε 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 πατήσομε το κουμπί 'Recommend Cities'. Όταν το πατάμε, φαινομενικά δεν στέλνομε 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. επιλέγομε 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 προϊστορία 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 γνωρίζομε 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 έχομε τίποτα, διότι στην περίπτωση αυτή τον ρόλο τους αναλαμβάνουν τα cookies.

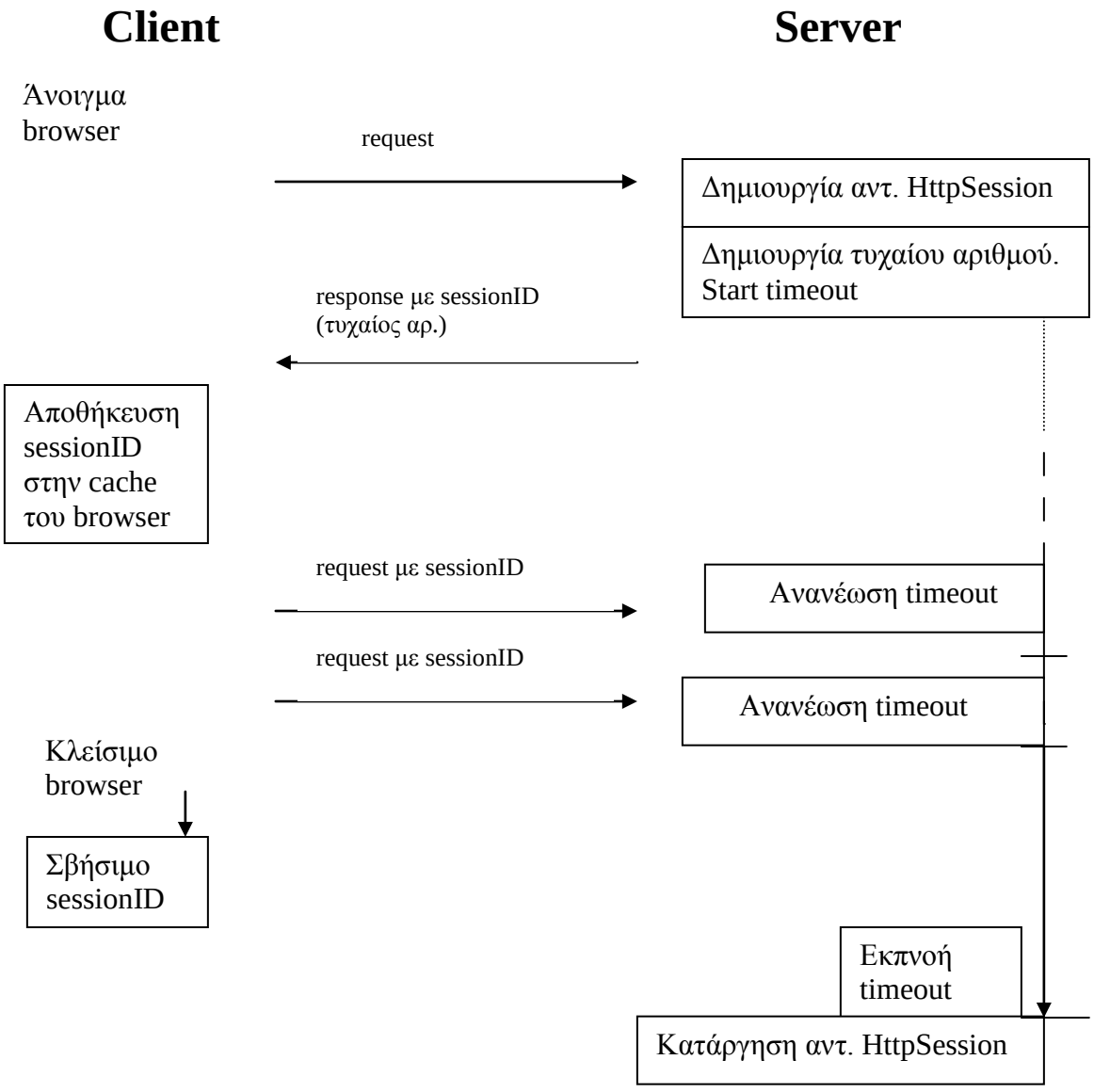
## 3.3.2. Session

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 δημιουργήσομε 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 πού 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 3.6.** 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 πού αποθηκεύονται στον δίσκο του client και διατηρούνται εκεί στην βάση ενός timeout
- ειδικά το sessionID cookie είναι ένα ζεύγος sessionID/τυχαίος αριθμός πού 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 3.1., 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 3.5.** 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 αποθηκεύομε τα ζεύγη όνομα παραμέτρου /τιμή πού θέλομε

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

Όταν μας ξαναμιλήσει ο client, αυτομάτως έχομε 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 ελέγχομε 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.

### 3.3.3. 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 σβήσομε 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.

## 3.4. 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 πού 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 πού 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..

Θα φτιάξουμε ένα παράδειγμα server-side scripting περίπου ανάλογο των προηγουμένων (Κώδικες 3.3, 3.4). Γράφομε σαν καθαρές .html εντολές, όσα ήταν πριν μέσα σε println(). Για το δυναμικό μέρος καταφεύγομε στην Java. Έτσι έχομε 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 παίρνομε 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 3.6.** Η Continent.jsp σελίδα δημιουργούσα session 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 3.7.** 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 θέλομε ανάμεσα σε '<%' και '%>'. Θυμίζομε 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*") πού 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 SessionServlet.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 των DestinationSelection και CitiesRecommendation 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 πού 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 πού 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 έχομε καμία συμμετοχή και δυνατότητα επέμβασης.

Ανεξαρτήτως τεχνολογίας (cookies, session, JSP, κλπ) το πλήρες σενάριο θα ήταν, την πρώτη φορά πού θα επισκεφθούμε το site να μας κατέβει η DestinationSelection σαν μία (στατική) σελίδα πού παίζει τον ρόλο φόρμας για αυτόν πού κάθεται 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 έχομε 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).Ας δούμε τώρα συνοπτικά τα πλεονεκτήματα του server-side scripting.

- Πλεονεκτήματα:
  - Επικοινωνία με βάσεις δεδομένων
  - Συμβατότητα με κάθε τύπο πελάτη, ανεξαρτησία από τύπο λειτουργικού συστήματος/φυλλομετρητή
  - Συγχρονισμός δεδομένων, συνεργατικές εφαρμογές
  - Έλεγχος Πρόσβασης
  - Δε χρειάζεται κάποια εγκατάσταση στον client

- Ασφάλεια κώδικα, προστασία από παραβίαση, προστασία πελάτου (διότι δεν έχουμε κώδικα εκτελούμενο στην πλευρά του πελάτη)

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. Με αυτή θα ασχοληθούμε στο αμέσως επόμενο κεφάλαιο.

# 4

# Javascript και client-side scripting

Η Javascript είναι μια γλώσσα συγγραφής σεναρίων (scripting language) που χρησιμοποιείται για να προσθέσει εφέ και διαλογικότητα (αλληλεπίδραση, διαδραστικότητα, interactivity) στις ιστοσελίδες μας.Είναι η κύρια client-side scripting γλώσσα και είναι ανταγωνιστική της γλώσσας προγραμματισμού VBScript.Η διαδικασία του client-side scripting έχει ως εξής:

1.Ένας κοινός browser αναζητά (request) μία .html σελίδα από τον server
2.Ο server βρίσκει τη σελίδα και τη στέλνει στο χρήστη
3.Η σελίδα παρουσιάζεται στον browser με τα όποια scripts να τρέχουν κατά τη διάρκεια της παρουσίασης ή μετά από αυτή(όταν π.χ. συμβεί κάποιο 'γεγονός')

Έτσι οι όποιες αλλαγές στις Web σελίδες,με το client-side scripting ,γίνονται αφού αυτές φτάσουν στον browser . Άρα καταλαβαίνουμε εύκολα πως ο κώδικας ο οποίος δημιουργεί αυτά τα scripts είναι εκεί,μέσα στην html σελίδα.Ο κώδικας της JavaScript περιέχεται ανάμεσα στα tags <script> και </script> και σαν χαρακτηριστικό (attribute) μπορούμε να χρησιμοποιήσουμε το type="text/javascript" ή το language="JavaScript". Μέσα σ' ένα αρχείο HTML μπορούμε να έχουμε όσα σύνολα tags <script> και </script> χρειαστούμε, είτε στο τμήμα head ή στο τμήμα body του εγγράφου.Δεν γίνεται μεταγλώττιση (compilation) του κώδικα της JavaScript, αρκεί μόνο ο φυλλομετρητής (browser) να υποστηρίζει την JavaScript. Τα γεγονότα (Events) και οι χειριστές γεγονότων (event handlers) είναι ένα πολύ σημαντικό μέρος στον JavaScript προγραμματισμό. Τα Events προκαλούνται από τις πράξεις του χρήστη. Αν ο χρήστης πατήσει ένα κουμπί, τότε συμβαίνει ένα Click-event. Αν ο δείκτης του mouse κινηθεί πάνω από μια διεύθηνση (link),τότε συμβαίνει ένα MouseOver-event. Υπάρχουν πολλά διαφορετικά events.Θέλουμε το javaScript πρόγραμμά μας να αντιδρά σε συγκεκριμένα events. Αυτό μπορεί να γίνει με την βοήθεια των event-handlers (χειριστές γεγονότων). Ένα κουμπί μπορεί να εμφανίζει ένα pop-up παράθυρο όταν πατιέται. Αυτό σημαίνει ότι το pop-up παράθυρο πρέπει να εμφανιστεί σαν απάντηση στο Click-event.

Για ευκολία στη κατανόηση της λειτουργίας της Javascript και του client-side scripting θα χρησιμοποιήσουμε τις φόρμες DestinationSelection.html (κώδικας 3.3) και CitiesRecommendation.html (κώδικας 3.4) που γνωρίσαμε στο προηγούμενο κεφάλαιο κάνοντας βέβαια τις απαραίτητες αλλαγές και προσθήκες.Οι νεές πια φόρμες μετονομάζονται και 'σώζονται' μέσα από το notepad ως JsDestinationSelection.html και JsCitiesRecommendation.html αντίστοιχα.

```
<!-- 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>
<script type="text/javascript">

function RadioCheck() {
var selection = document.conti.continent;

for (i=0; i<selection.length; i++)
if (selection[i].checked == true){
alert( ‘ Welcome to Cookies. ‘ +selection[i].value +  '  is an interesting continent !');
}
}


function SetCookie() {
 var cookieName ='DestinationCookie';
 var selection = document.conti.continent;
 var nMinutes =1 ;
 var today = new Date();
 var expire = new Date();
for (i=0; i<selection.length; i++)
if (selection[i].checked == true){
 expire.setTime(today.getTime() + 60000*nMinutes);
 document.cookie = cookieName+"="+escape(selection[i].value)
            + ";expires="+expire.toGMTString();
}
}

</script>
</HEAD>
<BODY>
  <FORM name="conti" >
    <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="button" VALUE="Submit" onClick="RadioCheck();SetCookie() ;">
     <INPUT TYPE="reset"> </P>
  </FORM>
</BODY>
</HTML>
```

**Κώδικας 4.1.** Η φόρμα JsDestinationSelection.html

Ο event-handler που χρειαζόμαστε λέγεται onClick. Αυτός λέει στον υπολογιστή τί να γίνει όταν συμβεί το ανάλογο event.Έτσι εδώ όταν πατάμε το κουμπί submit βλέπουμε στο κώδικα πως καλούνται δύο συναρτήσεις.Η συνάρτηση RadioCheck() και η συνάρτηση SetCookie().Ένα Click-event πραγματοποιήθηκε και ο χειριστής γεγονότων μας οδηγεί μέσα στο script.Η εντολή alert() της συνάρτησης RadioCheck() σου επιτρέπει να δημιουργείς popup παράθυρα. Μέσα στη παρένθεση πρέπει να βάλετε το κείμενο του pop-up παραθύρου. Σε αυτή τη περίπτωση το κείμενο έχει ένα σταθερό και ένα μεταβλήτο μέρος.Το 'σταθερό' μύνημα είναι ότι περιεκλείεται μέσα στα " " και εμφανίζεται στο pop up μύνημα όπως έχει.Στη περίπτωσή μας "is an interesting continent ! " .Το 'μεταβλητό' μέρος του μυνήματος είναι η τιμή του radio button που εμείς επιλέξαμε.Πως έγινε αυτό θα το δούμε τώρα αμέσως.

Η JavaScript είναι μια αντικειμενοστραφής γλώσσα (object-oriented language), που σημαίνει ότι στην καρδιά της περιέχει ένα προκαθορισμένο σύνολο αντικειμένων τα οποία συσχετίζονται με τα διάφορα συστατικά (components) μιας HTML σελίδας. Για να μπορέσουμε να δούμε ή να διαχειριστούμε την κατάσταση ενός αντικειμένου απαιτείται η χρήση των ιδιοτήτων (properties) και των μεθόδων (methods). Σε κάθε αντικείμενο θα γίνεται αναφορά των ιδιοτήτων (properties), των μεθόδων (methods) και των χειριστηρίων συμβάντος (event handlers) που συσχετίζονται μ' αυτό.Το document είναι ένα αντικείμενο που δημιουργείται από τον φυλλομετρητή όταν φορτώνεται μια ιστοσελίδα και το οποίο περιέχει πληροφορίες για το τρέχον έγγραφο, όπως είναι ο τίτλος του , το χρώμα φόντου και οι φόρμες (forms) που περιέχει.Η form αποτελεί ιδιότητα του αντικειμένου document. Η κάθε φόρμα σ' ένα έγγραφο (document) αποτελεί ένα ξεχωριστό αντικείμενο (object) στο οποίο μπορούμε να αναφερθούμε μέσω του αντικειμένου form. Το αντικείμενο form είναι ένας πίνακας (array) που δημιουργείται καθώς ορίζονται οι φόρμες (forms) μέσω των HTML tags.Παρατηρούμε πως στη φόρμα που είχαμε γνωρίσει στα προηγούμενα παραδείγματα (κώδικας 1.1) ,το κουμπί submit ως input type=”submit” και η ιδιότητα action της φόρμας μας,μας όριζαν τον 'τόπο' που θέλαμε να 'δράσουμε' πατώντας το κουμπί submit στέλνοντας τις επιλογές μας στο servlet του server.Εδώ θέλουμε να δράσουμε στην ίδια html σελίδα,θέτοντας το κουμπί submit ως input type=”button” και δίνοντας ένα όνομα στη φόρμα μας.Και το όνομα αυτής "conti"!.Ετσι μπορούμε να απευθυνθούμε σε αυτή με την εντολή document.conti .Μας είναι τώρα εύκολο να χειριστούμε και να καταλάβουμε την μεταβλητή selection = document.conti.continent και την τιμή της selection[i].value .Με την συνάρτηση SetCookies() δημιουργούμε το επιθυμητό cookie που θα φιλοξενίσει ο browser μας.Ορίζουμε το όνομα του ως 'DestinationCookie' και το χρόνο που θα "φιλοξενηθεί"' από τον browser σε 1 λεπτό.Το Date() είναι αντικείμενο που αντικαθιστά ένα κανονικό τύπο ημερομηνίας (date type). Αν και δεν έχει καθόλου ιδιότητες (properties), διαθέτει πολλές μεθόδους (methods), οπως είναι η setTime() , η getTime() και η toGMTString().Οι μέθοδοι αυτές είναι προκαθορισμένες ,έχουν να κάνουν με την παρουσίαση της ημερομηνίας και ώρας στον browser και δε θα επεκταθούμε παραπέρα.

Το παρακάτω συγκεκριμένο παράδειγμα είναι το ίδιο με αυτό του κώδικα 3.4 τροποποιημένο βέβαια στις ανάγκες του client-side scripting.

```
<!-- Cities available from the chosen continent -->
<HTML>
<HEAD>
  <TITLE>Cookie taken into Account</TITLE>

<script language="javascript">

function getCookie(cookieName){
 var theCookie=" "+document.cookie;
 var ind=theCookie.indexOf(" "+cookieName+"=");
 if (ind==-1) ind=theCookie.indexOf(";"+cookieName+"=");
 if (ind==-1 || cookieName=="") return "";
 var ind1=theCookie.indexOf(";",ind+1);
 if (ind1==-1) ind1=theCookie.length;
 return unescape(theCookie.substring(ind+cookieName.length+2,ind1));
}


function getCities() {
var conti = getCookie('DestinationCookie');
var continent = new Array ('Europe','Asia' ,'America' ,'Africa');
var cities = new Array('Athens and Rome', 'Peking and Amman', 'New York', 'Cairo');
if ( conti.length != 0 ) {

for (i=0; i<continent.length; i++)
if (continent[i] == conti)
alert( 'In '  + conti + ' we recommend to visit ' + cities[i] +   '.');

}

else alert('Sorry, no recommendation possible ! You did not select a continent or the cookies have
expired.');
}

</script>

</HEAD>
<BODY>
  <FORM>
    Press
    <INPUT TYPE=submit VALUE="Recommend Cities" onclick="getCities()" >
    for cities in your chosen continent
  </FORM>
</BODY>
</HTML>
```

**Κώδικας 4.2.** Η φόρμα JsCitiesRecommendation.html

Όπως και στη προηγούμενη φόρμα μας έτσι κι εδώ έχουμε ένα click-event.Πατώντας το κουμπί Recommend Cities καλείται η συνάρτηση getCities() που με τη σειρά της καλεί την getCookie() η οποία έχει ως όρισμα το όνομα του cookie (getCookie(CookieName)).Άλλο ένα μύνημα alert εμφανίζεται στον browser μας ! Τέλος για το deployment απλώς θέτουμε τις JsDestinationSelection.html και JsCitiesRecommendation.html μέσα σε ένα folder π.χ. JavascriptExamples κάτω από το webapps και χρησιμοποιούμε τις διευθύνσεις:

http://localhost:8080/ JavascriptExamples/ JsDestinationSelection.html και
http://localhost:8080/ JavascriptExamples/ JsCitiesRecommendation.html

Τα όσα εμφανίζονται στο παραπάνω παράδειγμα μας δεν διαφέρουν ουσιαστικά από εκείνα του παραδείγματος του προηγούμενο κεφαλαίου με τα servlets.Την ίδια δουλεία φαίνεται να κάνουν και τα δύο ενώ με την Javascript καταφέραμε και προσθέσαμε διάφορα εφέ και επίσης αλληλεπίδραση με το χρήστη.Προσέξατε κάτι άλλο;!

Όλη η 'δουλειά' έχει πια μεταφερθεί στον browser και κατ' επέκταση στο χρήστη.Ζητήσαμε την σελίδα απο τον server ,μας την έδωσε,αλλά από εκεί και πέρα όλα όσα συνέβησαν έγιναν στη πλευρά του χρήστη με τη βοήθεια του browser.Δε χρειάστηκε σε κανένα σημείο της εργασίας μας να 'πάμε' στη πλευρά του server πέρα από την αναζήτηση της σελίδας που επιθυμούμε.Έτσι από το server-side scripting που είδαμε στο προηγούμενο κεφάλαιο περάσαμε στο client-side scripting χρησιμοποιώντας την γλώσσα Javascript.Ας δούμε συντοπτικά τα πλεονεκτήματα του client-side scripting.

- Πλεονεκτήματα:
  - Δυναμική αλληλεπίδραση
  - Μείωση του φορτίου του server
  - Βολικό για περιπτώσεις εφαρμογών με υψηλές υπολογιστικές απαιτήσεις (π.χ. κινούμενα σχέδια)
  - Ασφάλεια Εξυπηρετητή, προστασία από σφάλματα

Τέλος, συναντάμε αρκετές φορές συνδυασμό των παραπάνω. Client-side scripting για πρόσβαση στα δεδομένα του browser και γρήγορη εκτέλεση τοπικά και server-side scripting για δυναμικά περιεχόμενα στις σελίδες και χρήση βάσεων δεδομένων. Κλασικό παράδειγμα είναι η AJAX. Με αυτόν τον συνδυασμό, μπορεί να επιτευχθεί δυναμική αλλαγή περιεχομένων στις ιστοσελίδες, χωρίς ξαναφόρτωμα αυτών!Τη τεχνολογία θα την δούμε σε επόμενο κεφάλαιο.

# 5

# AJAX (Asynchronous JavaScript και XML)

Τα αρχικά AJAX (Asynchronous JavaScript and XML) περιλαμβάνουν έναν ιδιαίτερο τρόπο συνδυασμού και χρήσης της JavaScript και του XML χωρίς να πρόκειται για μία νέα 'γλώσσα' ή τεχνολογία. Σκοπός είναι η δραματική αύξηση της δυναμικότητας της αλληλεπίδρασης (interactivity) στην πλευρά του πελάτη μεταξύ ανθρώπινου χρήστη και browser. Βασικό χαρακτηριστικό και περιορισμός στα όσα είδαμε μέχρι τώρα είναι ότι και για την ελάχιστη αλλαγή στην πλευρά του χρήστη πρέπει ο server να δημιουργήσει και στείλει μία *ολόκληρη νέα* σελίδα. Με την μεθοδολογία του AJAX αυτό θα γίνεται τώρα επιλεκτικά σε διάφορες περιοχές / οικόπεδα της σελίδας με το υπόλοιπο μέρος της να μένει σταθερό και να μην χρειάζεται ανανέωση από τον server. Προς τούτο πρέπει να έχομε πρόσβαση σε κάποιο δομημένο τρόπο που χρησιμοποιεί ο browser για να 'ζωγραφίσει' την σελίδα. Μέχρι τώρα υποθέταμε απλά ότι ο browser, σαν μαύρο κουτί, καθοδηγείται από την html που λαμβάνει μέσω του HTTP response και δημιουργεί την σελίδα. Οι μοντέρνοι όμως browsers περνούν από ένα ενδιάμεσο στάδιο, κατασκευάζοντας πρώτα το 'DOM της σελίδας'. Ο browser λαμβάνει κείμενο html, το εκλαμβάνει σαν xml και δημιουργεί στην μνήμη του το σχετικό XHTML DOM πού είδαμε στο προηγούμενο κεφάλαιο. Γι' αυτό και επιμένομε στη αυστηρή χρήση της XHTML και όχι απλής HTML. Σε αυτήν όλα τα στοιχεία κλείνουν και όλα συμφωνούν με τις επιταγές της XML. Το DOM αυτό παραμένει στην μνήμη του πελάτη και ταυτόχρονα καθοδηγεί την εμφάνιση επί της οθόνης. Μπορούμε όμως τώρα να επεμβαίνουμε επιλεκτικά σε αυτό το DOM και να επηρεάζομε την τελική εμφάνιση και λειτουργικότητά της σελίδας, όπου και όπως θέλομε, χωρίς να πειράζομε το υπόλοιπό της. Επιτυγχάνεται τελικά ένας επιπλέον βαθμός διαδραστικότητας. Είδαμε πώς ξεφύγαμε από τις στατικές εφαρμογές web, σε δυναμικές εφαρμογές όπου λαμβάνονται υπ' όψιν οι επιλογές του χρήστη για την παρουσίαση της επόμενης σελίδας. Παρ' όλο πού μιλούσαμε για δυναμικές σελίδες, ή κάθε μία από μόνη της είναι στατική, καθόσον ζωγραφίζεται μία φορά και δεν αλλάζει ποτέ η ίδια. Τώρα θα έχομε πραγματικά δυναμικές σελίδες, όπου επιλεκτικά στην διάρκεια της εμφάνισής τους θα αλλάζουν περιοχές τους, χαρακτηριστικά τους (εμφανισιακά ή άλλα) και επιμέρους λειτουργικότητα (π.χ. ο ρόλος κάποιο κουμπιού), πάντα με βάση την ίδια εμφανιζόμενη σελίδα. Ο γενικός τρόπος είναι ότι εμείς θα αλλάζομε το DOM μέσω JavaScript στον browser (client side) και αυτόματα οι μετατροπές θα αντικατοπτρίζονται στην εμφάνιση ή λειτουργικότητα της εμφανιζόμενης σελίδας. Οι αλλαγές όμως αυτές εν μέρει θα προέρχονται από τον sever, π.χ. για να εμφανισθούν νέες τιμές σε έναν σταθερό πίνακα. Επομένως το HTTP response θα μεταφέρει τώρα όχι ολόκληρη σελίδα, αλλά επιλεκτικά στοιχεία που θα πρέπει να εμφανισθούν στη θέση άλλων που εστάλησαν πιο πριν.

Προχωρώντας πιο πέρα στο ίδιο σκεπτικό μπορούμε να εμπλέξομε και μετασχηματισμούς XSLT, που ούτως ή άλλως λογίζονται μέσα στην οικογένεια της XML. Το HTTP response μεταφέρει τώρα μόνο την γυμνή πληροφορία σαν δομή XML. Η εμφάνισή της μπορεί να γίνει με μία εντολή JavaScript πού καλεί ένα XSLT (που έχει σταλεί στην αρχήν της συνόδου) για να δημιουργήσει το DOM της σελίδας που εμφανίζεται στην οθόνη. Πάλι έχομε μετατόπιση της λειτουργικότητας στην πλευρά του πελάτη, δηλαδή εμπλέκομε και client-side processing στην μορφή του client-side scripting, πρακτικά client-side JavaScript.

Το βασικότερο στοιχείο στον AJAX είναι το αντικείμενο XMLHttpRequest, το οποίο δημιουργούμε και διαχειριζόμαστε μέσα στο JavaScript πρόγραμμα στον browser. Μέσα και πίσω από αυτό κρύβεται όλη η αλληλεπίδραση με τον server. Φανταζόμαστε ότι η σελίδα ήδη υπάρχει στον browser. Η ονομασία υποδηλώνει ότι το HTTP request μας θα έχει σαν περιεχόμενο μια δομή XML, πράγμα πολύ πιο πλούσιο από τα απλά ζεύγη όνομα /τιμή παραμέτρου χρήστη. Η απάντηση από τον server μπορεί να έρθει όπως πριν σαν html/text, αλλά τώρα και σαν XML. Όταν λοιπόν έχομε ήδη μία σελίδα στον browser, όχι μόνον σαν κείμενο html άλλα με όλη την μορφή της σε DOM, ερωτάμε τον server και δεχόμαστε την απάντηση του σε XML. Με τις τεχνολογίες που μάθαμε μπορούμε τώρα να εξάγομε τα επιθυμητά δεδομένα από το XML της απόκρισης και διαμορφώνοντας με client-side scripting το DOM της τρέχουσας σελίδας να την αλλάξομε επιλεκτικά.

Συνοπτικά το AJAX δεν είναι μια νέα τεχνολογία, αλλά με χρήση αυτών που μάθαμε επιτυγχάνει την δυναμική ενημέρωση μέρους της τρέχουσας σελίδας στον browser αποφεύγοντας την αποστολή από τον server και λήψη / εμφάνιση από τον browser μίας ολόκληρης νέας σελίδας.

Για να δούμε και να καταλάβουμε τις δυνατότητες του AJAX θα χρησιμοποιήσουμε και πάλι το παράδειγμα του κεφαλαίου 3.Αυτή τη φορά όμως όλα όσα εμφανίζονται σε διαφορετικές σελίδες στον browser μας θα εμφανίζονται μόνο σε μία.

```
<html>
<head>
<script type="text/javascript">


function getXmlHttpRequestObject() {
if (window.XMLHttpRequest) {
return new XMLHttpRequest(); //To support the browsers IE7+, Firefox, Chrome, Opera, Safari
} else if(window.ActiveXObject) {
return new ActiveXObject("Microsoft.XMLHTTP"); // For the browsers IE6, IE5
} else {
alert("Error due to old verion of browser upgrade your browser");
}
}
var xmlhttp = getXmlHttpRequestObject();
```

```
function getContinent() {
if (xmlhttp.readyState == 4 || xmlhttp.readyState == 0) {
xmlhttp.open("GET", 'AjaxServlet', true);
xmlhttp.onreadystatechange = handlegetContinent;
xmlhttp.send(null);
}
}



function SetCookie() {
 var cookieName ='name';
 var selection = document.conti.continent;
 var nMinutes =1 ;
 var today = new Date();
 var expire = new Date();
for (i=0; i<selection.length; i++)
if (selection[i].checked == true){
 expire.setTime(today.getTime() + 60000*nMinutes);
 document.cookie = cookieName+"="+escape(selection[i].value)
              + ";expires="+expire.toGMTString();
}
}

function getCookie(cookieName){
 var theCookie=" "+document.cookie;
 var ind=theCookie.indexOf(" "+cookieName+"=");
 if (ind==-1) ind=theCookie.indexOf(";"+cookieName+"=");
 if (ind==-1 || cookieName=="") return "";
 var ind1=theCookie.indexOf(";",ind+1);
 if (ind1==-1) ind1=theCookie.length;
 return unescape(theCookie.substring(ind+cookieName.length+2,ind1));
}

function getCities() {
var conti = getCookie('name');
var continent = new Array ('Europe','Asia' ,'America' ,'Africa');
var cities = new Array('Athens and Rome', 'Peking and Amman', 'New York', 'Cairo');
if ( conti.length != 0 ) {

for (i=0; i<continent.length; i++)
if (continent[i] == conti)
alert( 'In '  + conti + ' we recommend to visit ' + cities[i] +   '.');

}

else alert('Sorry, no recommendation possible ! You did not select a continent or the cookies have expired.');
}

function handlegetContinent() {
if (xmlhttp.readyState == 4  && xmlhttp.status==200   ) {



////////////////////////  Folowing lines for xml case ONLY - ////////////////////////////////
/////////////// extracts text from xml doc at server, with content ////////////////////////
```

```
//<vehicles><car>Toyota</car><car>Opel</car><car>VW</car></vehicles>
////////////////////////////////// uncomment 6 lines below ////////////////////////////////////
//      xmlDoc=xmlhttp.responseXML;
//      txt="";
//      x=xmlDoc.getElementsByTagName("car");
//     for (i=0;i<x.length;i++)
//        {txt= txt+ x[i].childNodes[0].nodeValue + "<br />"; }
//     document.getElementById("myDiv").innerHTML=txt;
//////////////////////////////////////////////////////////////////////////////////////////////////


document.getElementById("myDiv").innerHTML = xmlhttp.responseText;
}
}


</script>
<head>
 <title>My First Ajax Example</title>
</head>
<body>

<form name="conti">

<STRONG>Select the Continent of your Destination:<br> </STRONG>
<PRE>

<input type="radio" onclick="SetCookie();getContinent();" name="continent" value="Europe">check here
for Europe<br>
<input type="radio" onclick="SetCookie();getContinent();" name="continent" value="Asia">check here for
Asia<br>
<input type="radio" onclick="SetCookie();getContinent();" name="continent" value="America">check here
for America<br>
<input type="radio" onclick="SetCookie();getContinent();" name="continent" value="Africa">check here for
Africa<br>
<br>
<div id="myDiv"> And then press the "Recommend Cities" button below for cities in your chosen continent
</div>
<br>
</PRE>
<input type="button" value="Recommend cities" onclick="getCities()">

</form>
</body>
</html>
```

**Κώδικας 5.1.** Η φόρμα MyAjax.html

Υποθέτομε ότι η παραπάνω είναι μία σελίδα που έχει έλθει και την εμφανίζει ο browser. Για δοκιμή απλώς την φέρνομε πάνω στο desktop σαν .html και την ανοίγομε. Το body

αυτής της σελίδας είναι κάτω από το head (επικεφαλίδα) και περιέχει ένα div ,ένα button και τέσσερα radio buttons. Το συμβάν onclick των radio buttons έχει ως αποτέλεσμα την κλήση δύο συναρτήσεων, συγκεκριμένα της SetCookie() και της getContinent(). Αυτές ορίζονται μέσα στο στοιχείο script που στο παράδειγμα μας είναι ενσωματωμένο στον header. Αυτό σημαίνει ότι μόλις πατήσομε το radio button επιλέγοντας μια ήπειρο θα εκτελεστεί η συνάρτηση getContinent(), και ότι παράξει αυτή σαν κείμενο θα μπει σε επιλεγμένη θέση της σελίδας. Επομένως πετύχαμε προγραμματιστικά με JavaScript να αλλάξουμε ένα μέρος μόνον της παρούσης σελίδας – όχι να φέρομε μία εντελώς νέαν.Οι συναρτησεις getCities() ,SetCookies() και GetCookies() μας είναι ήδη γνωστές από το προηγούμενο κεφάλαιο,αναφερόμενο στο client-side scripting και για αυτό δε θα ασχοληθούμε ξανά με αυτές.

Επιπλέον η νέα πληροφορία θα έλθει από τον server, στο παράδειγμα θα παραχθεί από το παρακάτω AjaxServlet. Αυτό γίνεται μέσω του αντικειμένου xmlhttp που είναι το επίσημο XMLHttpRequest είτε το ActiveXObject("Microsoft.XMLHTTP") στην περίπτωση ενός παλαιοτέρου MS browser. Αφού αυτό δημιουργηθεί και κληθεί η getContinent() γίνονται τα ακόλουθα:

- Με την μέθοδό open του xmlhttp προετοιμάζομε ένα request, εδώ με GET και στο συγκεκριμένο url για να βρούμε μέσα στον server το ζητούμενο κείμενο(παραγόμενο από το AjaxServlet)
  xmlhttp.open("GET", 'AjaxServlet', true);

Εδώ δε δώσαμε κάποιο url αλλά το όνομα του servlet καθώς όπως θα δούμε την MyAjax.html θα την τοποθετήσουμε αυτούσια κάτω από το ..webapps/ajax .

- Το request φεύγει με την μέθοδο send
  xmlhttp.send();
  ενώ έχομε επιλέξει το 'asynchronous' ('true' στην παραπάνω open). Αυτό σημαίνει ότι ο κώδικας JavaScript δεν θα κολλήσει στο σημείο αυτό περιμένοντας την απάντηση του server. Τούτο θα συνέβαινε με την επιλογή 'synchronous' ('false' στην παραπάνω open). Και πότε θα ασχοληθούμε με την απάντηση του server στην ασύγχρονη περίπτωση; Όταν με το έναυσμα της λήψης της δημιουργηθεί εσωτερικό συμβάν και σηκωθεί η σημαία xmlhttp.onreadystatechange. Έχομε ορίσει και προγραμματίσει την σχετική handlegetContinent() η οποία θα κληθεί αυτομάτως και θα εκτελώντας τα κατωτέρω.

- Με την λήψη της απάντησης από τον server εξετάζομε αν οι μεταβλητές readyState και xmlhttp.status του αντικειμένου xmlhttp πήραν της τιμές 4 και 200 αντίστοιχα. Το 200 είναι το γνωστό HTTP/1.1 200 OK .Τότε θα τρέξει η γραμμή
  document.getElementById("myDiv").innerHTML=xmlhttp.responseText;
  Στην αριστερή πλευρά απευθυνόμαστε στο element με id="myDiv", που είναι στοιχείο του 'document' και το document αυτό δεν είναι άλλο από την ίδια την html σελίδα σε μορφή DOM μέσα στην μνήμη του browser. Πράγματι η μέθοδος getElementById του document ειναι μέρος του API του XHTML και επιστρέφει στοιχείο σύμφωνα με την προκαθορισμένη από εμάς τιμή ενός 'id' attribute, το οποίο έχομε προβλέψει. Στο στοιχείο document.getElementById("myDiv") υπάρχει σαν

περιεχόμενο το innerHTML, δηλ. το μέρος του html που είναι το εσωτερικό του div με id="myDiv". Αυτό το γεμίζομε με την ποσότητα δεξιά, δηλ με το responseText του xmlhttp που δεν είναι άλλο από αυτό που έστειλε ο server με το μήνυμα response της απάντησης του.

**Προτιμούμε το GET** έναντι του POST, όταν έχομε να στείλομε ένα μικρό αριθμό ζευγών όνομα/τιμή παραμέτρου χρήστη από τον browser στον server μέσω του url αντικειμένου XMLHttpRequest, όπως το ξέρομε, π.χ.
xmlhttp.open("GET"," …/get_example?firstname=John&lastname=Smith",true);
Υπάρχει και η λεπτομέρεια του να αποφύγομε την άντληση της πληροφορία από σελίδα που δεν έρχεται πράγματι εκ νέου από τον server, αλλά από την ίδια που έχει ενδεχομένως αναζητηθεί προηγουμένως και είναι cached στον browser. Αυτό γίνεται με το να αναζητήσομε κάτι που είναι εγγυημένα νέο, δηλ. να θέσομε μία τυχαία (άρα και νέα) τιμή σε μία (ψευδο)παράμετρο του GET.
xmlhttp.open("GET"," …/get_example?t=" + Math.random()+"&firstname=John&lastname=Smith",true);

**Με το POST** μπορούμε να στείλομε στο σώμα (body), πάντα σε μορφή κειμένου, ενδεχομένως απροσδιορίστου μήκους, κείμενο, αλλά και xml, δηλ. δεδομένα / επιλογές του χρήστη μέσα σε δομές που έχομε εμείς προκαθορίσει. Μπορούμε να πληροφορήσομε για αυτό τον server μέσω της επικεφαλίδας (header) του μηνύματος, θέτοντας μετά το open, γραμμές setRequestHeader(*header,value*) δηλαδή
xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");
και μετά
xmlhttp.send(text_content_of_POST_body);

## 5.1.Τροφοδοσία του innerHTML με Κείμενο Text η XML

Σημαντική είναι η ιδιότητα (attribute) x.innerHTML ενός στοιχείου x, όπως document.getElementById("myDiv").innerHTML. Η γραμμή αυτή είναι το κλειδί του AJAX, δεδομένου ότι εντοπίζει το που θα καταλήξει το 'κείμενο'- text που αντλούμε από τον server. Το κείμενο αυτό είτε θα έρχεται έτοιμο, είτε θα έρχονται από τον server τα δεδομένα για αυτό, π.χ. σαν xml, και θα διαμορφώνεται τοπικά, Αντίστοιχα εργαζόμεθα με την ιδιότητα responseText / responseXML του αντικειμένου XMLHttpRequest.

Στην **περίπτωση text** έχομε:
document.getElementById("myDiv").innerHTML=xmlhttp.responseText;

Στην **περίπτωση xml** αντλούμε από τον server κάτι σαν το αρχείο someXML.xml, και η απάντηση έρχεται με
xmlDoc=xmlhttp.responseXML;

Τώρα εξάγομε (με XML - όχι XHTML! - DOM) από το xmlDoc τα δεδομένα που θα μπουν στο text, που με την σειρά του θα μπει στο innerHTML. Ο σχετικός κώδικας είναι ενσωματωμένος στο παραπάνω παράδειγμα του Σχήματος, το οποίο δείχνει εναλλακτικά όλες τις περιπτώσεις άντλησης κειμένου πληροφορίας από τον server: κείμενο παραγόμενο από servlet, xml που μετατρέπεται σε κείμενο στον client

Ας δούμε τώρα το servlet εκείνο μέσω του οποίου θα παραχθεί δυναμικά το κείμενο που θα αντικαταστήσει το 'στατικό' κείμενο στη σελίδα μας.

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.http.*;
import javax.servlet.ServletException;

public class AccessServlet extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException

{
PrintWriter output;
    Cookie cookies[];
    cookies = request.getCookies();
     response.setContentType( "text/html" );
     output = response.getWriter();
     if ( cookies.length != 0 ) {
        for ( int i = 0; i < cookies.length; i++ )
     output.println( "Welcome to Cookies. " +cookies[i].getValue()+ " is an interesting continent !" );
       }

}

}
```

**Κώδικας 5.2.** Το AccessServlet

Για να αναγνωρίζεται λοιπόν το servlet μας χρειάζεται και ο γνωστός μας Deployment Descriptor (DD) ο οποίος πάντα καλείται web.xml .Το servlet μας είναι γνωστό στο διαδίκτυο ως AjaxServlet.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
<display-name>ajaxWithServlet</display-name>
```

```
<servlet>
<servlet-name>AccessServlet</servlet-name>
<servlet-class>AccessServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>AccessServlet</servlet-name>
<url-pattern>/AjaxServlet/*</url-pattern>
</servlet-mapping>

</web-app>
```

**Σχήμα 5. 1** Το web.xml σαν Deployment Descriptor της AccessServlet.class

Απομένει να φροντίσομε ώστε και ο server να στείλει πίσω το απλό κείμενο (το παραγόμενο απο το AjaxServlet )που θα μπει στο div. Ετσι για το deployment κατα τα γνωστά,σύμφωνα με το σχήμα .....δημιουργούμε ένα φάκελο κάτω από το webapps π.χ. ajax και κάτω από αυτό το WEB-INF μαζί όμως με τη σελίδας μας,την MyAjax.html.Σηκώνουμε τον Tomcat ,πληκτρολογούμε την διευθυνση :

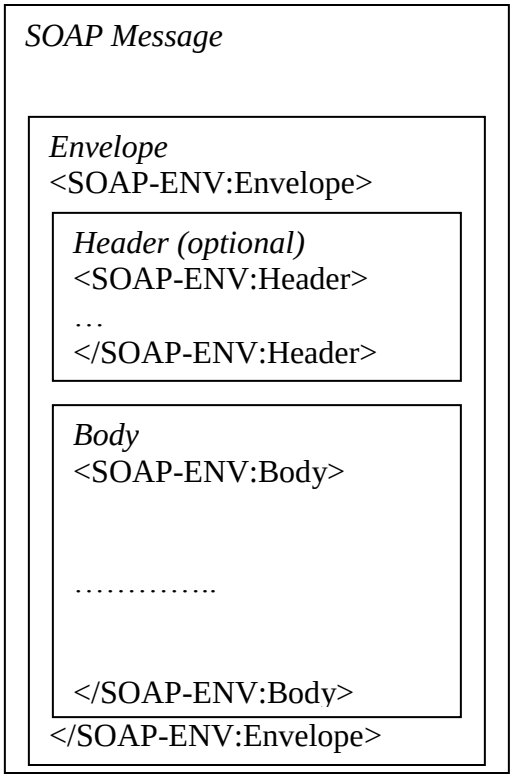
http://localhost:8080/ajax/MyAjax.html

και επιβεβαιώνουμε τα παραπάνω.

# 6

# 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,. 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) και από την Microsoft (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) πού 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.) και για γενική μεταφορά πληροφορίας άσχετης με τις εξ αποστάσεως κλήσεις (SOAP Messaging).



**Σχήμα 6.1.** Γενική δομή μηνύματος SOAP και σχέση μεταξύ envelope, (header) και body

## 6.1. 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 .

```
// 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 6.2.** 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 6.3.** 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 παρείχετο 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 6.4.** 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.

## 6.1.1. Παράδειγμα Υπηρεσίας SOAP-RPC

Εδώ θα χρειασθούμε ένα 'servlet engine'. Τα servlets είναι memory-resident προγράμματα πού τρέχουν μέσα σε ένα servlet engine (ή 'servlet container'). Επειδή ευρίσκονται ανά πάσα στιγμή στην μνήμη ανταποκρίνονται ταχύτατα σε αιτήσεις πού 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, πού 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) έχομε δυναμικές σελίδες web χρησιμοποιώντας περαιτέρω εξειδικευμένα servlets.

Ένας web server μπορεί να ανταποκρίνεται σε πακέτα HTTP. Τούτο σημαίνει ότι μπορεί να αντιμετωπίζει αιτήσεις πού 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) πού 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 πού θα δούμε στην ενότητα αυτή το περιεχόμενο κάθε μηνύματος πού είτε λαμβάνει, είτε εντέλλεται να αποστείλει πίσω στον client είναι κωδικοποιημένο με την μορφή του πρωτοκόλλου SOAP.

## 6.1.2. SOAP Server

Αφού εγκαταστήσομε τον Tomcat Server από το http://tomcat.apache.org μπαίνουμε στο folder C:\ apache-tomcat-7.0.30\bin. Εκεί μέσα βλέπομε όσα μπορούμε να κάνομε μέσω command line. Με την εντολή startup, βλέπουμε τον Tomcat να σηκώνεται σε διαφορετικό παράθυρο.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/rpcrouter βλέπομε ότι ανταποκρίνεται (στην τελευταία έστω και αρνητικά). Επίσης ότι xyz.html βάλομε στο C:\apache-tomcat-7.0.30\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("Beatle","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 6.1.** Η Vcatalog πού πραγματοποιεί την υπηρεσία VehicleCatalog

Μεταγλωττίζοντας τον παραπάνω κώδικα με javac, έχομε 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 γράφομε 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 πού 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 πού 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 πού 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 6.5.** Το VCatalogDD.xml, Deployment Descriptor της υπηρεσίαςVehicleCatalog

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 αναφέρεται στην εσωτερική δομή του κάθε συγκεκριμένου server, χωρίς επιρροή στην δομή και σύνταξη των μηνυμάτων SOAP.

Τώρα δε απομένει παρά να θέσομε την δημιουργηθείσα υπηρεσία στην διάθεση του server. Προς τούτο ρίχνομε το παραπάνω αρχείο V.jar (πού περιέχει τον κώδικα της υπηρεσίας), μέσα στο folder C:\ apache-tomcat-7.0.30\lib, όπου ο Tomcat αναμένει να ευρίσκει κάθε ζητούμενη υπηρεσία.Ρίχνουμε και το VCatalogDD.xml σε ένα folder έστω τον C:/ .Τέλος με την παρακάτω command line διατάζομε τον server να εκθέσει την υπηρεσία

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

Με αυτήν την command line εντολή τρέχομε 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) γράψομε μόνο την παράμετρο list βλέπομε 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).

Συνεχίζομε με την υπηρεσία εκτεθειμένη. Αυτή η ίδια, δηλ. η VShop.VCatalog, στο διαδίκτυο γνωστή ως VehicleCatalog, ευρίσκεται έτοιμη να ανταποκριθεί, στο folder C:\ apache-tomcat-7.0.30\lib, χωρίς να έχει ακόμη κληθεί από κανέναν.

## 6.1.3. SOAP Client

Μεταγλωττίζομε 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 urll = 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(urll, model, manufacturer);
      }  catch (Exception e) {e.printStackTrace();}
    }
}
```

**Κώδικας 6.2.** 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. Μέσα απο το folder όπου βρίσκεται η VadderLister.class δίνουμε την εντολή

java VAdderLister http://localhost:8080/soap/servlet/rpcrouter “Cortina” “Ford”

και έτσι προστίθεται στον κατάλογο πού διατηρείται στον server ένα νέο μοντέλο (Cortina) με τον κατασκευαστή του (Ford), πράγμα πού διαπιστώνεται ευθύς μετά με τον πλήρη Hashtable πού επιστρέφεται από την listV. Ο Hashtable αυτός είναι και η πιο πολύπλοκη δομή πού έχομε 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 πού 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 πού 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, αποτυπώνομε το αντικείμενο της γενικής class Parameter, σε αντικείμενο της class πού γνωρίζομε 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 αναμένομε αντικείμενο της class Hashtable, την αποτυπώνομε 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.

## 6.1.4.Ανακεφαλαίωση

Τα παραπάνω βήματα συνοψίζονται ως ακολούθως:

(α) Σηκώσαμε έναν (Tomcat) server.
(β) Θέσαμε σε συγκεκριμένο subfolder του server την class (σε μορφή jar) πού 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 πού 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 πού 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 πού 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 πού σηκώνεται και συμβουλεύεται τον Deployment Descriptor ποίους 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 ποίου τύπου θα είναι η απόκριση για να την κωδικοποίηση και την αποστείλει

πίσω στον client. Στο παράδειγμά μας μάλιστα υπήρχε και η περίπτωση ενός Hashtable. Όλα γίνονται από το SOAP engine πού είναι ενσωματωμένο στον server. Στην πλευρά του client έχομε 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 (απο- και κωδικοποιήσει). Με άλλα λόγια δεν έχομε ακόμη ξεφύγει ουσιαστικά από τις δυνατότητες του απλού RPC. Είναι επίσης προφανές ότι για την μεταφορά (πάντα μέσω SOAP) πιο πολύπλοκων η πιο εξειδικευμένων δομών, χρειάζονται να επιστρατευθεί επιπλέον κατάλληλος κώδικας. Αυτό θα το δούμε τώρα για την μεταφορά 'σειριακά μέσω του σύρματος' αντικειμένων της Java.

## 6.2. 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 δίδει αντικείμενα JavaBeans σχετικά με το προηγούμενο παράδειγμά μας (μοντέλα αυτοκινήτων, κατασκευαστές, τώρα πρόσθετα και έτος). Όλες οι αναφερθείσες απαιτήσεις εκπληρούνται 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 πού 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. Outputs an informative string
    public String toString(){
        return "" + VModel + "' by " + VManufacturer + " (" + VYear + ")";}
}
```

**Κώδικας 6.3.** Ορισμός VehicleBean

## 6.2.1 SOAP Server (με JavaBeans)

Ας υποθέσουμε ότι τώρα θέλουμε να διαθέσουμε μία υπηρεσία, παρόμοια με την προηγουμένη VehicleCatalog. Η παρακάτω παρουσιαζόμενη BVehicleCatalog θα είναι γραμμένη για το παραπάνω VehicleBean και οπωσδήποτε πιο κομψή καθόσον 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 6.4.** Η BVCatalog πού πραγματοποιεί την υπηρεσία BVehicleCatalog

Η BVCatalog.java πρέπει να βρει κατά την μεταγλώττισή της την BVShop/VehicleBean.class (λόγω του package BVShop). Ευρισκόμενοι στο anyForder, δημιουργούμε το παιδί anyForder/BVShop, ρίχνομε μέσα στο BVShop τις BVCatalog.java, VehicleBean.java (και οι δύο με ‘package BVShop;’ σαν πρώτη γραμμή) και μεταγλωττίζομε μέσα από το anyForder

javac BVShop/BVCatalog.java

Μέσα στο BVShop έχουν τώρα δημιουργηθεί τα VehicleBean.class και BVCatalog.class. Μέσα από το anyForder δημιουργούμε το BV.jar με την εντολή

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

Το BV.jar εμπεριέχει δύο classes: την VehicleBean.class και την BVCatalog.class. Το όνομα του αρχείου BV.jar είναι αυθαίρετο και η συγκεκριμένη επιλογή του δεν έχει καμμία σημασία. Πάλι ρίχνομε όπως πριν το BV.jar μέσα στο C:\ apache-tomcat-7.0.30\lib.

Ο νέος Deployment Descriptor είναι ο BVCatalogDD.xml (σε όλα τα νέα έχομε προσθέσει το 'B' – λόγω 'Beans'). Δεν ενοχλεί το ότι μερικές από τις μεθόδους έχουν τα ίδια ονόματα, καθόσον 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 6.6.** 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 ανταλλάσσομε 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 ιδιαιτέρους serializer. Τούτο είναι αποφευκτέο και αξίζει αντ΄ αυτού να φροντίσομε 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. 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.

## 6.2.2 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 προηγουμένης VAdderLister.java). Ιδιαιτέρως αναφέρομε τις classes SOAPMappingRegistry και BeanSerializer, πού θα ασχοληθούν με τα beans. Το αντικείμενο reg της class SOAPMappingRegistry (‘μητρώον 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 προσδιορίζομε. Έτσι με την μέθοδο mapTypes, καθορίζομε (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 αντιστοιχήσεις πού γίνονται στον server πρωτυποποιμένα 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 Hashtabel 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 6.5.** 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 έχομε 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. Μεταγλωττίζομε το VehicleBean.java στο ίδιο folder (αυτήν την φορά χωρίς την πρώτη εντολή package) και μετά μεταγλωττίζομε το BVAdderLister.java, όπου βεβαίως χρησιμοποιείται η VehicleBean.class. Μετά τρέχομε τον client από το ίδιο folder (που συνυπάρχουν οι VehicleBean.class BVAdderLister.class )και με

java 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 Monitior. Στο τελευταίο διαπιστώνουμε τον όγκο και σημασία της δουλειάς πού έγινε από το SOAP, σχεδόν μαγικά !

### 6.2.3. Serialization και Deserialization

Εξετάζοντας το folder ...\ org\apache\soap\encoding\soapenc κάνοντας extract το αρχείο soap.jar διαπιστώνομε ότι υπάρχουν classes για serialization / deserialization όλων των δομών πού συναντήσαμε στο απλό και στο SOAP-RPC, π.χ. array, boolen, date, hashtable, bean, …. Ιδιαίτερο ενδιαφέρον έχει η BeanSerializer class. Αυτή εμπεριέχει και τον deserializer. Στις παρακάτω περιγραφόμενες διαδικασίες φαίνεται καθαρά πως χρησιμεύει η δομή πού 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) πού 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, πού 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.

## 6.3. Συνεισφορά του SOAP στην Αναφορά Σφαλμάτων

Εδώ θα δούμε πώς το SOAP μπορεί ακόμη και να επιστρέψει στον client μηνύματα σφάλματος πού δημιουργήθηκαν στον server. Ήδη στον client είδαμε ότι από το αντικείμενο response εξάγομε με fault.getFaultString() μία περιγραφή του αντικειμένου fault. Αυτό το fault έρχεται πίσω, εφόσον δημιουργηθεί στον server, και προέρχεται από το διαγνωστικό πού 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 έχομε ήδη επικαλεσθεί στο στοιχείο <isd:faultListener>

την class org.apache.soap.server.DOMFaultListener και η επιθυμητή αναφορά σφάλματος ήδη επιστρέφεται στον client. Δεν έχομε λοιπόν παρά να την δούμε να μεταφέρεται μέσω του TCP Monitor. Προκαλούμε ένα σφάλμα μέσα στην μέθοδο addV του server (πού εμφανίζεται στο runtime): απλά απομακρύνομε την γραμμή catalog = New Hashtable. Η addV δεν έχει που να εγγράψει και επιστρέφεται μία αναφορά σφάλματος πού 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 εκτυπώναμε το περιεχόμενο text του <stacktrace> element, θα βλέπαμε την πλήρη αναφορά σφαλμάτων πού 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 πού γνωρίσαμε να ασχολείται με την κωδικοποίηση (serialization / deserialization) μεταβλητών, είτε εισόδου σε, είτε εξόδου από κλήσεις μεθόδων. Αν θεωρήσομε το ίδιο το σφάλμα σαν ένα γενικό κείμενο XML, μπορούμε να πούμε ότι στην περίπτωση αυτή έχομε 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.

## 6.4. 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 πού 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 πού 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 παρατηρήσομε 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 ιδίου 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 πού έρχεται με το αντικείμενο req, μπορεί να καταλήγει στο αντικείμενο res. Στην περίπτωση αυτή έχομε 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, πού αναφέρεται στο όνομα πού θα δώσουμε στην υπηρεσία SOAP Messaging, πού κατασκευάζομε (urn:Messaging_Service).

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

**Σχήμα 6.7.** 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 env της class Envelope. Γενικά ο όρος 'marshall' υπονοεί την μετατροπή ενός αντικειμένου 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 env 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 coverts the .xml file into a SOAP message
  // and the sends this to seviceURL
  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 6.6.** 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 πού 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().getBody Entries();
```

```
 Element vehicles_in_DOM = (Element)bodyEntries.iterator().next;
// .............}
```

**Κώδικας 6.7.** 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 έχομε 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. ΄Ενας πιο πολύπλοκος server, θα μπορούσε να ασχοληθεί με πολλά κείμενα XML λαμβανόμενα μέσα στο ίδιο μήνυμα SOAP. Η env είναι αντικείμενο της class Envelope (έχομε κάνει import org.apache.soap.Envelope), η μέθοδος getBody() του αντικειμένου env επιστρέφει ένα αντικείμενο της class Body (ΔΕΝ έχομε 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 έχομε 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 πού 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 ποίος 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), πού μας δόθηκε με την κλήση της vehiclesSOAP, καθορίζει το τι θα φύγει και τι μορφής είναι αυτό πού 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.getAttributeNode("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 6.8.** Ο server δέχεται, μετατρέπει και επαναστέλλει το μήνυμα SOAP

Και τέλος μένει το MessagingServiceDD.xml, δηλ. ο Deployment Descriptor. Ορίζομε 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 6.8.** 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 messagerouter και όχι του rpcrouter, όπως πριν!Μέσα από τον folder που βρίσκεται η MessagingClient.class δίνουμε την εντολή
java MessagingClient http://localhost:8081/soap/servlet/**messagerouter**
*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 αποστελλόμενο 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 ενδιάμεσος, θα επενεργεί μόνο σε μηνύματα SOAP XML αποτελώντας κατ΄ εξοχήν περίπτωση SOAP Messaging και όχι SOAP-RPC.

# 7

# Υλοποίηση

Στο κεφάλαιο αυτό θα δούμε την εγκατάσταση (Windows 2000 ή XP) της Java και των σχετικών αρχείων (jar files) που αναφέρθηκαν και χρησιμοποιούνται στα παραδείγματα αυτών των σημειώσεων καθώς και την εγκατάσταση και τις οδηγίες για τον Tomcat.

## 7.1.Εγκατάσταση Java

Στο site της Java (http://java.sun.com ) βρίσκουμε και κατεβάζουμε το Java SE 7u7. 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 development tools που περιέχει και το Java Runtime Environment (jre) 7 update7. Δεν είναι υποχρεωτικό να εγκαταστήσουμε το Source code. Καλό είναι η εγκατάσταση να γίνει σε κάποιο root directory (C:\ ή D:\). Εδώ υποθέτουμε το C:\ ( ή C:\Program Files\Java\jdk1.7.0_07)

## 7.2.Ορισμός των μεταβλητών συστήματος

Για να είναι ποιο εύκολη η χρήση του μεταγλωττιστή της Java (Java Compiler – javac) καλό είναι να θέσουμε κάποια environment variables ως εξής:

Με σκοπό να τρέξει ο Tomcat στο περιβάλλον της JAVA πρέπει να ορίσουμε ως μεταβλητή περιβάλλοντος την μεταβλητή με όνομα JAVA_HOME:

1. Δεξί-click στο My Computer εικονίδιο και επιλέξουμε
Properties.
2. Click στο Advanced και επιλέγουμε το Environment Variables.
3. Κάτω απο το System Variables, πατάμε click στο New.
4. Στο πεδίο Variable name , πληκτρολογούμε JAVA_HOME.
5. Στο πεδίο Variable value, πληκτρολογούμε το path εγκατάστασης της Java .(π.χ. C:\Program Files\Java\jdk1.7.0_07)
6. Click στο OK.
7. Click στο Apply Changes.

Αφού ορίσαμε την μεταβλητη JAVA_HOME τώρα πρέπει να προσθέσουμε το path της JAVA στο γενικό path του συστήματος(αν δεν υπάρχει το δημιουργούμε όπως πριν!):

1. Δεξί-click στο My Computer εικονίδιο και επιλέξουμε
Properties.
2. Click στο Advanced και επιλέγουμε το Environment Variables.
3. Κάτω απο το System Variables, πατάμε click στο Path.
4. Πατάμε edit και στο πεδίο Variable value, προσθέτουμε το path της εγκατάστασης της JAVA ακολουθούμενο από ';' (π.χ. C:\Program Files\Java\jdk1.7.0_07\bin;)..
5. Click στο OK.
6. Click στο Apply Changes

## 7.3.Εγκατάσταση Tomcat

Στο site http://tomcat.apache.org/ επιλέγουμε την τελευταία έκδοση του tomcat (7.0.30) και κατεβάζουμε το αρχείο zip . Εξάγοντας αυτό σε οποιοδήποτε folder ( εδώ υποθέτουμε στο C:\), δημιουργείται subfolder με όνομα apache-tomcat-7.0.30. Για να τρέξει ο Tomcat αρκεί να μπούμε στο ...\bin και να γράψουμε startup..

## 7.4.Εγκατάσταση επί πλέον αρχείων.

Θα πρέπει να εγκαταστήσουμε κάποια επιπλέον αρχεία-βιβλιοθήκες για τις ανάγκες υλοποίησης των εφαρμογών μας τόσο στη πλευρά του client όσο και στη πλευρά του server .

### 7.4.1.Εγκατάσταση στη πλευρά του client.

Για να εγκαταστήσουμε τα επιπλέον αρχεία - βιβλιοθήκες (Java Archive – επέκταση .jar ) πρέπει να αντιγράψουμε τα παρακάτω αρχεία στους παρακάτω δύο φακέλους:
(JAVA_HOME)\jre\lib\ext (... και C ή D:\Program Files\Java\jre7\lib\ext).Τα αρχεία αυτα είναι:
1.jdom.jar,
2.xalan.jar,
3.xmlrpc-1.2-b1.jar,
4.xerces.jar,
5.soap.jar,

6.mail.jar,
7.activation.jar,
8.axis.jar,
(που βρίσκονται στο site http://courses.cn.ntua.gr/course/view.php?id=47 ), αλλά και το αρχείο :
9.servlet-api.jar, (που βρίσκεται στο C:\ apache-tomcat-7.0.32\lib),

Με αυτό καλύπτουμε τις ανάγκες του client. Στην πλευρά του βλέπει μόνο τις classes της Java.

## 7.4.2.Εγκατάσταση στη πλευρά του server.

Για τον server τα ίδια αρχεία πρέπει να ριχθούν και στο C:\ apache-tomcat-7.0.32\lib.Επίσης ρίχουμε μέσα στο \webapps το αρχείο:
1.soap.war
(που βρίσκεται στο παραπάνω site.).

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.

## 7.5 Χρήσιμες οδηγίες για την υλοποίηση των εφαρμογών

Εδώ δίνοται κάποιες χρήσιμες οδηγίες ικανές ώστε ο χρήστης να μπόρει να τρέξει όλα τα παραδείγματα που υπάρχουν στα προηγούμενα κεφάλαια ενώ δίνονται παρακάτω ,για ευκολία στην αναζήτηση ,συγκεντωμένες οι πιο πολύπλοκες εντολές που χρειάζονται για την υλοποίηση.

## 7.5.1.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 βλέπομε 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.Η μεταγλώτιση μπορεί επίσης να γίνει με την εξής εντολή: javac program1.java program2.java .Για ένα πιο εξελιγμένο σενάριο, βλ. την επόμενη παράγραφο.

## 7.5.2 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 θέλομε να δημιουργήσομε program12.jar, πού να περιέχει το κώδικα των program1 και program2 όπως πάνω. Έστω subfolder το όνομα ενός subfolder (πού θα δημιουργήσομε παρακάτω), subfolder του folder όπου εργαζόμαστε

Σαν πρώτη γραμμή των program1.java και program2.java πού ευρίσκονται στο folder γράφομε

package subfolder;

Μεταγλωττίζομε πρώτα το program2.java, εφόσον αυτό χρησιμοποιείται από το program1.java. Μετά δημιουργούμε το subfolder, στο οποίο ρίχνομε 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 ανοίξομε 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 C:\ apache-tomcat-7.0.32\lib) και βλέπει τα ονόματα όλων των classes (εδώ σαν subfolder/program1.class και subfolder/program2.class).

## 7.5.3Χρησιμοποιούμενες 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*.

**Δημιουργία .jar**
jar cvf BV.jar BVShop/VehicleBean.class BVShop/BVCatalog.class
jar cvf MServer.jar MSS/MessagingServer.class
**(μετά τοποθέτηση των .jar στο** C:\ apache-tomcat-7.0.32\lib **του Tomcat)**

**Directory για startup, shutdown και εντολές ελέγχου του Tomcat**
cd *C:\ apache-tomcat-7.0.32\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

Τα παραπάνω γίνονται και μέσω του browser από **http://localhost:8080/soap/admin**

**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!)

# Βιβλιογραφία και Χρήσιμα Links


[1] World Wide Web Consortium (W3C), http://www.w3.org.
[2] Τεχνολογίες XML, www.w3.org/xml
[3] Beginning Java Web Services, H.Bequet, M.M. Kunnumpurath, S. Rhody, A. Tost, Wrox
[4] Office 2003 XML, Evan Lenz, Mary McRae, Simon St.laurent, O'Reilly
[5] Java, how to program, Deitel & Deitel, Prentice Hall.
[6] Java 2 Programmer, Bill Brogden & Marcus Green, QUW Certification.
[7] Java Cookbook, Ian Darwin, O' Reilly.
[8] Στο www.w3schools.com tutorials για JavaScript, DOM, XSLT
[9] Apache Software Foundation http://tomcat.apache.org/
[10] Java http://www.oracle.com
[11] Javascript http://dide.flo.sch.gr/Plinet/plinet.html#a7
[12] Wikipedia-SOAP http://en.wikipedia.org/wiki/SOAP
[13] Lalilуna-Java tutorials and Development http://www.lalilуna.de/first-hibernate-example-tutorial.html
[14] Ajax tutorial http://www.learn-ajax-tutorial.com/AjaxBasics.cfm#.UHV1va5P2j5
[15] Javascript tutorial http://www.w3schools.com/js/default.asp
[16] Java servlet tutorial http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/
[17] Java servlet Technlology http://www.oracle.com/technetwork/java/index-jsp-135475.html
[18]Using SOAP with tomcat
http://onjava.com/pub/a/onjava/2002/02/27/tomcat.html?page=1
[19] http://courses.cn.ntua.gr/course/view.php?id=47