ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Συγκριτική Μελέτη Αρχιτεκτονικών Σχεδίασης Διαδικτυακών Υπηρεσιών (Web Services)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
του
ΙΩΑΝΝΗ Μ. ΒΟΡΔΟΥ

Επιβλέπων : Ιάκωβος Σ. Βενιέρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Δεκέμβριος 2014

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Συγκριτική Μελέτη Αρχιτεκτονικών Σχεδίασης Διαδικτυακών Υπηρεσιών (Web Services)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιωάννης Μ. Βόρδος

Επιβλέπων: Ιάκωβος Στ. Βενιέρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 15 Δεκεμβρίου 2014.

.....	.....	.....
Ι. Βενιέρης	Δ. Κακλαμάνη	Ν. Ουζουνογλου
Καθηγητής Ε.Μ.Π	Καθηγήτρια Ε.Μ.Π	Καθηγητής Ε.Μ.Π

Αθήνα, Δεκέμβριος 2014

.....

Ιωάννης Μ. Βόρδος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

ΠΕΡΙΛΗΨΗ

Σκοπός αυτής της διπλωματικής είναι η συγκριτική μελέτη των δύο κύριων αρχιτεκτονικών μοντέλων ανάπτυξης διαδικτυακών υπηρεσιών και εφαρμογών, RESTful και SOAP Web Services.
Από τις αρχές της δεκαετίας του 2000 έχουν προταθεί και υλοποιηθεί δύο νέες τεχνολογίες/αρχιτεκτονικές για να δώσουν λύση στα προβλήματα ευχρηστίας και λειτουργικότητας των ηλεκτρονικών υπηρεσιών. Η πρώτη με το όνομα **Representational State Transfer – REST ή RESTful**, προτάθηκε από τον Roy Fielding στο πλαίσιο της διδακτορικής του διατριβής το 2000. Οι υπηρεσίες που παρέχει η εν λόγω αρχιτεκτονική είναι γνωστές ως REST Services. Η δεύτερη με το όνομα **Simple Object Access Protocol – SOAP** προτάθηκε το 1998 από τους Dave Winer, Don Box, Bob Atkinson, and Mohsen Al-Ghosein για την Microsoft. Μέσα από την αρχιτεκτονική των τεχνολογιών αυτών ορίζεται ένα σύνολο από προδιαγραφές και κανόνες με τους οποίους είναι δυνατή η δημιουργία και η παροχή ηλεκτρονικών υπηρεσιών μέσα στο Διαδίκτυο με αρκετά απλό τρόπο.

Η τεχνολογία των υπηρεσιών διαδικτύου (Web Services, REST Services) αφορά συνήθως στους προγραμματιστές εφαρμογών στο Διαδίκτυο και όχι στους απλούς χρήστες και χειριστές ηλεκτρονικών υπολογιστών.

Η εργασία αυτή αρχικά αναφέρεται στο τι είναι οι Υπηρεσίες Διαδικτύου, πως υλοποιούνται και τι αυτές προσφέρουν.

Στα επόμενα δύο κεφάλαια γίνεται μία λεπτομερής αναφορά στις επικρατούσες αρχιτεκτονικές Υπηρεσιών διαδικτύου, καθώς και τις αντιπροσωπευτικές υλοποιήσεις αυτών SOAP και REST με αναφορές στα τεχνικά χαρακτηριστικά τους. Με τον όρο Υπηρεσίες Διαδικτύου ή Υπηρεσίες Ιστού ή Web Services εννοούμε τις υπηρεσίες που καλύπτουν και οι δύο τεχνολογίες REST και SOAP. Παράλληλα, επιχειρείται η ανάδειξη των πλεονεκτημάτων και των μειονεκτημάτων και των δύο, στηριζόμενοι στη διεθνή βιβλιογραφία και αρθρογραφία, ενώ ακολούθως γίνεται μία σύγκριση αυτών των δύο τεχνολογιών/αρχιτεκτονικών, βασιζόμενοι σε παλαιότερες μελέτες της ακαδημαϊκής και όχι μόνο κοινότητας.
Στα πλαίσια της εργασίας αναπτύχθηκε εφαρμογή σε γλώσσα προγραμματισμού JAVA για την τεκμηρίωση των συμπερασμάτων της εργασίας.

Λέξεις κλειδιά

Υπηρεσίες Διαδικτύου, Υπηρεσίες Ιστού, Αρχιτεκτονικές Υπηρεσιών Ιστού, SOAP, REST, RESTful, Web Services, REST Services, HTTP, WSDL

ABSTRACT

The purpose of this diploma thesis is a comparative study of two main architectural models development of web services and applications, RESTful and SOAP Web Services.
Since the early 2000s have been proposed and implemented two new technologies / architectures to solve the problems of usability and functionality of electronic services. The first, named Representational State Transfer - REST or RESTful, proposed by Roy Fielding in the context of his doctoral thesis in 2000. This architecture provides services known as REST Services. The second architecture named Simple Object Access Protocol - SOAP proposed in 1998 by Dave Winer, Don Box, Bob Atkinson, and Mohsen Al-Ghosein for Microsoft. Through the architecture of these technologies, a set of standards and rules is defined in order to enable the creation and delivery of electronic services in the Internet with several simple ways.
The technology of web services (Web Services, REST Services) usually refers to the Internet application developers and not to ordinary users and computer operators.
This work at the beginning refers to what web services are, how they are implemented and what they offer.
In the next two chapters, one detailed reference to the prevailing architectural Web Services, and representative embodiments thereof SOAP and REST with reference to the technical characteristics. The term Web Services covering both technologies REST and SOAP. Furthermore, attempts to highlight the advantages and disadvantages of both, based on the international literature and then a comparison of these two technologies / architectures, based on previous academic studies.
In the framework of this diploma thesis, an application developed in JAVA programming language to support the conclusions of the work.

Keywords

SOAP, REST, RESTful, Web Services, REST Services, HTTP, WSDL, Web Services Architecture

ΕΥΧΑΡΙΣΤΙΕΣ

Σύμβουλος καθηγητής στην εκπόνηση της διπλωματικής ήταν ο κ. Ιάκωβος Βενιέρης. Θα ήθελα να τον ευχαριστήσω θερμά για την ανάθεση της διπλωματικής και για την ευκαιρία που μου δόθηκε μέσω αυτής να αποκτήσω πολλές γνώσεις πάνω σε σύγχρονα θέματα τεχνολογιών διαδικτύου.

Θα ήθελα επίσης να ευχαριστήσω τον υποψήφιο διδάκτορα Ανδρέα Καψάλη για την υποστήριξη και την καθοδήγησή του κατά τη διάρκεια εκπόνησης της διπλωματικής, τη διαθεσιμότητά του να απαντήσει σε κάθε είδους απορία και τη διαρκή ενθάρρυνσή του.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια μου ξεκινώντας από τη σύζυγό μου Νατάσα και συνεχίζοντας με τις δύο κόρες μου Ανδριάνα και Στέλλα για την υποστήριξή της προσπάθειας μου, την υπομονή και την παρότρυνση. Ελπίζω να αναπληρώσω το χρόνο που τους στέρησα την παρουσία μου για την ολοκλήρωση των σπουδών μου.

Ιωάννης Μ. Βόρδος

Πίνακας Περιεχομένων

Πίνακας Εικόνων

ΕΙΣΑΓΩΓΗ

Σήμερα το Διαδίκτυο αποτελεί για τις επιχειρήσεις ένα μέσο προβολής και δημοσιοποίησης των δραστηριοτήτων τους. Τα τελευταία χρόνια όμως πολλές επιχειρήσεις παγκοσμίως άρχισαν να χρησιμοποιούν το Διαδίκτυο ως μέσο παροχής υπηρεσιών προς τους πελάτες τους αλλά και προς άλλες επιχειρήσεις.

Με την εξέλιξη αυτή δημιουργήθηκε ένα σύνολο από θέματα που είχαν να κάνουν με την ευχρηστία, τη λειτουργικότητα, την προσβασιμότητα και την απλότητα των παρεχόμενων υπηρεσιών. Και ενώ οι προσπάθειες που έγιναν για να δοθούν λύσεις ήταν πολλές, το αποτέλεσμα ήταν το ίδιο: υπήρχε ένα μεγάλο σύνολο από έτοιμες υπηρεσίες στο Διαδίκτυο αλλά η χρησιμοποίησή τους ήταν πάρα πολύ μικρή.

Από τις αρχές της δεκαετίας του 2000 έχουν προταθεί και υλοποιηθεί δύο νέες τεχνολογίες/αρχιτεκτονικές για να δώσουν λύση στα προβλήματα ευχρηστίας και λειτουργικότητας των ηλεκτρονικών υπηρεσιών. Η πρώτη με το όνομα **Representational State Transfer – REST ή RESTful**, προτάθηκε από τον Roy Fielding στο πλαίσιο της διδακτορικής του διατριβής το 2000. Οι υπηρεσίες που παρέχει η ενλόγω αρχιτεκτονική είναι γνωστές ως REST Services. Η δεύτερη με το όνομα **Simple Object Access Protocol – SOAP** προτάθηκε το 1998 από τους Dave Winer, Don Box, Bob Atkinson, and Mohsen Al-Ghosein για την Microsoft, Η έκδοση SOAP 1.2 υποστηρίχθηκε ως προτεινόμενη από το world wide web consortium [www.w3c.org], με το όνομα web service στις 24 Ιουνίου 2003.

Μέσα από την αρχιτεκτονική των τεχνολογιών αυτών ορίζεται ένα σύνολο από προδιαγραφές και κανόνες με τους οποίους είναι δυνατή η δημιουργία και η παροχή ηλεκτρονικών υπηρεσιών μέσα στο Διαδίκτυο με αρκετά απλό τρόπο.

Έτσι τα τελευταία χρόνια οι ηλεκτρονικές υπηρεσίες πήραν νέα διάσταση και πλέον κάθε επιχείρηση μπορεί σχετικά εύκολα να δημιουργεί και να παρέχει υπηρεσίες στο Διαδίκτυο αλλά μπορεί με ακόμα μεγαλύτερη ευκολία να χρησιμοποιεί έτοιμες υπηρεσίες.

Η τεχνολογία των υπηρεσιών διαδικτύου (Web Services, REST Services) αφορά συνήθως στους προγραμματιστές εφαρμογών στο Διαδίκτυο και όχι στους απλούς χρήστες και χειριστές ηλεκτρονικών υπολογιστών.

Όμως η χρήση των τεχνολογιών αυτών, λαμβάνοντας υπόψη ότι η ενσωμάτωσή τους αποτελεί μία σχετικά εύκολη διαδικασία, θα πρέπει να περάσει σε κάθε δραστήριο επιχειρηματία. Με αυτό τον τρόπο οι επενδύσεις στην πληροφορική και στο Διαδίκτυο θα είναι πραγματικά αξιόλογες και ανταγωνιστικές τόσο για τα Ελληνικά όσο και για τα παγκόσμια δεδομένα.

Στην εργασία αυτή αρχικά θα αναφερθούμε στο τι είναι, πως υλοποιούνται και τι προσφέρουν οι Υπηρεσίες Διαδικτύου.

Στα επόμενα δύο κεφάλαια θα κάνουμε μία λεπτομερή αναφορά στις επικρατούσες αρχιτεκτονικές Υπηρεσιών διαδικτύου (Service-Oriented Architecture- SOA και Resource-Oriented Architecture- ROA), καθώς και τις αντιπροσωπευτικές υλοποιήσεις αυτών SOAP και REST με αναφορές στα τεχνικά χαρακτηριστικά τους. Για λόγους ευκολίας και χωρίς να δημιουργείται κάποια παρανόηση στο υπόλοιπο της εργασίας με τον όρο Υπηρεσίες Διαδικτύου ή Υπηρεσιών Ιστού ή Web Services θα εννοούμε τις υπηρεσίες που καλύπτουν και οι δύο τεχνολογίες REST και SOAP. Παράλληλα, θα επιχειρηθεί η ανάδειξη των πλεονεκτημάτων και των μειονεκτημάτων και των δύο, στηριζόμενοι στη διεθνή βιβλιογραφία και αρθρογραφία.

Ακολούθως θα γίνει μία σύγκριση αυτών των δύο τεχνολογιών/αρχιτεκτονικών, βασιζόμενοι σε παλαιότερες μελέτες της ακαδημαϊκής και όχι μόνο κοινότητας.

Τέλος θα προσπαθήσουμε να αποδείξουμε/τεκμηριώσουμε τα παραπάνω με τη χρήση καταλλήλου λογισμικού.

Κεφάλαιο Α: Υπηρεσίες Ιστού (Web Services)

Α.1 Γενικά

Δεδομένου ότι οι υπηρεσίες Ιστού έχουν γίνει το μεγάλο όραμα, όλο και περισσότεροι άνθρωποι αναζητούν την πρακτικότητα της εφαρμογής και της χρησιμοποίησης τους για επιχειρησιακό όφελος.

Όπως η ανάπτυξη της κινητής τηλεφωνίας είχε τη δυνατότητα να αλλάξει πλήρως τον τρόπο που οι άνθρωποι επικοινωνούν, έτσι και οι υπηρεσίες Ιστού έχουν τη δυνατότητα να αλλάξουν πλήρως τον τρόπο που οι εφαρμογές επικοινωνούν.

Οι υπηρεσίες Ιστού έχουν προκύψει από τη σύγκλιση τριών σημαντικών γεγονότων (Ebizq, 2002). Ο πρώτος ήταν η εμφάνιση του Διαδικτύου ως οικονομικώς αποδοτική και διαδεδομένη υποδομή. Ο δεύτερος ήταν η υιοθέτηση του XML ως πρότυπο. Το τρίτο γεγονός ήταν η εμφάνιση ενός συνόλου κοινών προτύπων υπηρεσιών Ιστού -- συγκεκριμένα, το απλό πρωτόκολλο πρόσβασης αντικειμένου (SOAP) και το πρωτόκολλο παραστατικής μεταφοράς κατάστασης (REST) -- που έχουν γίνει αποδεκτά από σημαντικούς προμηθευτές πλατφορμών αντί των ιδιόκτητων προτύπων όπως CORBA και DCOM.

Απλά εγκαθιστώντας τις υπηρεσίες Ιστού, καθίσταται η λειτουργία μίας εφαρμογής διαθέσιμη μέσω του Διαδικτύου με έναν τυποποιημένο, προγραμματιστικό τρόπο. Οι εφαρμογές που δεν θα μπορούσαν να προσεγγιστούν από την ακολουθία των άκαμπτων ιδιόκτητων πρωτοκόλλων, μπορούν τώρα να μιλήσουν η μία στην άλλη μέσω του Διαδικτύου, ανεξάρτητα από τη γλώσσα προγραμματισμού, την πλατφόρμα ή τα εσωτερικά πρωτόκολλά τους.

Οι υπηρεσίες Ιστού χρησιμοποιούν τις υπάρχουσες υποδομές IT και επιτρέπουν στις επιχειρήσεις να περιβάλουν τις υπάρχουσες εφαρμογές υπό ένα τυποποιημένο, συνεπές και επαναχρησιμοποιήσιμο σχήμα, έτσι καμία επένδυση δεν χάνεται. Παρέχουν έναν χαμηλού κόστους τρόπο να συνδεθούν οι εσωτερικές εφαρμογές και να συνεργαστούν μεταξύ των επιχειρησιακών συνεργατών.

Λίγοι άνθρωποι θα διαφωνούσαν στο ότι η σε πραγματικό χρόνο ολοκλήρωση εφαρμογής-προς-εφαρμογή είναι ένας ιδανικός στόχος που έχει αποδειχθεί δύσκολο να επιτευχθεί πλήρως. Η δυνατότητα να υπάρξουν εφαρμογές -- εσωτερικές ή εξωτερικές – οι οποίες να μοιράζονται δεδομένα αυτόματα χωρίς την ανθρώπινη επέμβαση, προκαλεί μείωση του κόστους, αύξηση της παραγωγικότητας, μειώσεις λαθών και άλλα ανταγωνιστικά πλεονεκτήματα που είναι εύκολο να προσδιορισθούν και να ποσοτικοποιηθούν.

Στον πραγματικό κόσμο, οι εφαρμογές γράφονται σε διαφορετικές γλώσσες χρησιμοποιώντας διαφορετικές πλατφόρμες, που προσαρμόζονται αυστηρά για συγκεκριμένες χρήσεις μέσα σε διαφορετικές εταιρίες και βρίσκονται πίσω από επίπεδα και firewalls και οι οποίες σχεδιάζονται με σαφή σκοπό την απαγόρευση επικοινωνίας με εξωτερικές εφαρμογές. Κανονικά, η σε πραγματικό χρόνο ολοκλήρωση απαιτούσε σύνθετες και ακριβές λύσεις, καθώς επίσης και ιδιωτικά δίκτυα για να εξασφαλιστούν ασφαλείς επικοινωνίες και διαχείριση.

Το κλειδί είναι η σε βιομηχανικό επίπεδο θέσπιση ενός συνόλου προτύπων που θα επέτρεπε στις εφαρμογές να επικοινωνούν, ανεξάρτητα από τη μητρική τους γλώσσα, και ένα οικονομικώς αποδοτικό και αξιόπιστο δίκτυο για τη μεταφορά. Σήμερα, με το Διαδίκτυο και τα νέα πρότυπα, είμαστε πιο κοντά από ποτέ στην επίτευξη της ολοκλήρωσης εφαρμογών σε πραγματικό χρόνο -- και αυτό χάρη στις υπηρεσίες Ιστού.

Α.2 Τι είναι ένα Web Service – WS.

Σύμφωνα με τον ορισμό που δίνεται από το W3C (Haas, 2004):

«Ένα WS είναι ένα σύστημα λογισμικού σχεδιασμένο να υποστηρίζει τη διαλειτουργική, μηχανή-προς-μηχανή, διάδραση μέσω ενός δικτύου. Έχει μία διεπαφή με περιγραφή που είναι επεξεργάσιμη από μηχανές (συγκεκριμένα, βάση της περιγραφής WSDL – Web Service Definition Language.) Τα άλλα συστήματα διαδρούν με τα WS με τρόπο που καθορίζεται από την περιγραφή του WS, χρησιμοποιώντας μηνύματα SOAP, τα οποία κανονικά μεταδίδονται με τη χρήση HTTP – HyperText

Transfer Protocol (με μία κωδικοποίηση σε XML σε συνδυασμό με άλλα Web πρότυπα.»

Όπως φαίνεται λοιπόν από τον παραπάνω ορισμό το W3C θεωρεί τα Web Services άμεσα συνδεδεμένα με το πρωτόκολλο SOAP. Η παραπάνω θεώρηση προκύπτει και από την ομιλία του προϊσταμένου προτυποποίησης των τεχνολογιών WS του W3C,Hugo Haas, στο Πανευρωπαϊκό Συνέδριο για τα WS το Νοέμβριο του 2005 στη Σουηδία (Haas, 2005). Στη συγκεκριμένη ομιλία με τίτλο «Ο Συμβιβασμός των Web Services με τα REST Services», αναφέρει μεταξύ άλλων ότι τα REST Services έχουν γίνει συνώνυμο των non-SOAP Services. Παρόλα αυτά, στην ίδια παρουσίαση αναφέρει ότι τα REST Services αποτελούν μέρος του ορισμού των Web Services, όπως αυτός αναφέρθηκε παραπάνω. Ενώ σε όλη την ομιλία του όταν αναφέρεται στα Web Services, αναφέρεται στη σχεδίαση SOAP, την οποία συγκρίνει με τη σχεδίαση REST. Εντούτοις καταλήγει στα ακόλουθα συμπεράσματα:

- Πρόκειται για ένα σύνολο τεχνολογιών (Web Services), οι οποίες μπορούν να υλοποιηθούν με 2 διαφορετικές σχεδιαστικές φιλοσοφίες (SOAP, REST).
- Και οι δύο προσεγγίσεις παρουσιάζουν πλεονεκτήματα.
- Υπάρχει μεγάλη επικάλυψη στις δυνατότητές τους.
- Δεν αναμένεται ο ανταγωνισμός των δύο σχεδιαστικών φιλοσοφιών να τερματιστεί σύντομα.
- Θα ήταν ιδιαίτερα χρήσιμο και για τις δύο κοινότητες να δουν τα πλεονεκτήματα της άλλης πλευράς.

Ένας πιο πλήρης ορισμός των Υπηρεσιών Ιστού παρέχεται από την IBM (IBM, 2007):

«Τα web services είναι μια τεχνολογία που επιτρέπει στις εφαρμογές να επικοινωνούν μεταξύ τους ανεξαρτήτως πλατφόρμας και γλώσσας προγραμματισμού. Ένα web service είναι μια διεπαφή λογισμικού (software interface) που περιγράφει μια συλλογή από λειτουργίες οι οποίες μπορούν

να προσεγγιστούν από το δίκτυο μέσω πρότυπων μηνυμάτων XML. Χρησιμοποιεί πρότυπα βασισμένα στη γλώσσα XML για να περιγράψει μία λειτουργία (operation) προς εκτέλεση και τα δεδομένα προς ανταλλαγή με κάποια άλλη εφαρμογή. Μια ομάδα από web services οι οποίες αλληλεπιδρούν μεταξύ τους καθορίζει μια εφαρμογή web services σύμφωνα με την Προσανατολισμένη στις Υπηρεσίες Αρχιτεκτονική (Service-Oriented Architecture – SOA).»

Τα web services λοιπόν αποτελούν μία αρχιτεκτονική κατανεμημένων συστημάτων κατασκευασμένη από πολλά διαφορετικά υπολογιστικά συστήματα τα οποία επικοινωνούν μέσω του δικτύου ώστε να δημιουργήσουν ένα σύστημα. Τα WS αποτελούνται από ένα σύνολο από πρότυπα τα οποία επιτρέπουν στους προγραμματιστές (developers) να υλοποιήσουν κατανεμημένες εφαρμογές - χρησιμοποιώντας διαφορετικά εργαλεία από διαφορετικούς προμηθευτές - ώστε να κατασκευάσουν εφαρμογές που χρησιμοποιούν ένα συνδυασμό από ενότητες λογισμικού (software modules). Οι ενότητες αυτές στη συνέχεια καλούνται από συστήματα που ανήκουν σε διαφορετικά τμήματα ενός οργανισμού ή σε διαφορετικούς οργανισμούς. (Rose India, 2008)

Α.3 Αρχιτεκτονικές Υπηρεσιών Ιστού

Τα Web Services παρέχουν λοιπόν τα τυποποιημένα μέσα για τη διαλειτουργικότητα μεταξύ διαφορετικών εφαρμογών λογισμικού, τα οποία «τρέχουν» σε μία μεγάλη ποικιλία πλατφορμών και/ή frameworks. Από την άλλη η αρχιτεκτονική των Web Services παρέχει ένα θεωρητικό μοντέλο και το πλαίσιο για την κατανόηση των Web Services, καθώς και των σχέσεων μεταξύ των περιεχομένων του μοντέλου.

Η αρχιτεκτονική δεν καθορίζει πως υλοποιούνται τα Web Services, ούτε θέτει περιορισμούς για το πώς αυτά μπορούν να συνδυαστούν. Αντίθετα, περιγράφει, τόσο τα ελάχιστα χαρακτηριστικά που είναι κοινά σε όλα τα WS, όσο και έναν αριθμό χαρακτηριστικών που είναι απαραίτητα από κάποια, αλλά όχι όλα τα WS.

Α.3.1 Χρησιμοι Ορισμοί

Έννοιες (Concepts)

Μια έννοια (Haas, 2004) αναμένεται να έχει κάποια αντιστοιχία με οποιαδήποτε υλοποίηση αρχιτεκτονικής. Για παράδειγμα, η έννοια του μηνύματος προσδιορίζει μια κλάση αντικειμένου (όχι όπως τα αντικείμενα και τις κλάσεις των αντικειμενοστρεφών γλωσσών προγραμματισμού) που αναμένουμε να είμαστε σε θέση να προσδιορίσουμε σε οποιοδήποτε πλαίσιο υπηρεσιών Ιστού. Η ακριβής μορφή ενός μηνύματος μπορεί να είναι διαφορετική στις διαφορετικές πραγματοποιήσεις. Η έννοια του μηνύματος μας λέει περισσότερο τι να ψάξουμε σε ένα συγκεκριμένο σύστημα παρά τον ορισμό της ακριβούς μορφής της.

Δεν έχουν όλες οι έννοιες μια υλοποίηση ως αντικείμενα ή δομές δεδομένων σε υπολογιστές ή συσκευές επικοινωνιών. Για παράδειγμα ένα πρόσωπο ή μία οργάνωση αναφέρεται στους ανθρώπους και τις ανθρώπινες οργανώσεις. Κάποιες άλλες έννοιες είναι περισσότερο αφαιρετικές όπως, η αξιοπιστία μηνυμάτων που δείχνει μια ιδιότητα της υπηρεσίας μεταφορών μηνυμάτων — μια ιδιότητα που δεν είναι απτή αλλά εν τούτοις είναι σημαντική στις υπηρεσίες Ιστού.

Κάθε έννοια παρουσιάζεται με έναν κανονικό, τυποποιημένο τρόπο που αποτελείται από έναν σύντομο ορισμό, μια απαρίθμηση των σχέσεων με άλλες έννοιες, και μια ελαφρώς πιο εκτεταμένη επεξηγηματική περιγραφή. Για παράδειγμα, η έννοια του εκπρόσωπου (agent) περιλαμβάνει ως συσχετισμό των εννοιών το γεγονός ότι ένας εκπρόσωπος (agent) είναι ένας υπολογιστικός πόρος, έχει μία ταυτότητα και έναν ιδιοκτήτη.

Σχέσεις (Relationships)

Οι σχέσεις (Haas, 2004) δείχνουν τις επιδράσεις μεταξύ των εννοιών. Γραμματικά, οι σχέσεις είναι ρήματα ή ακριβέστερα, κατηγορήματα. Η δήλωση μιας σχέσης λαμβάνει χαρακτηριστικά τη μορφή: έννοια-κατηγόρημα-έννοια.

Παραδείγματος χάριν, στον εκπρόσωπο (agent), δηλώνουμε ότι:

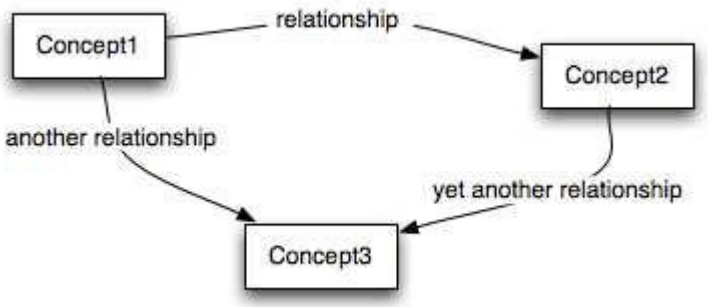
Ένας εκπρόσωπος (agent) είναι ένας υπολογιστικός πόρος.

Αυτή η δήλωση κάνει έναν ισχυρισμό, σε αυτήν την περίπτωση για τη φύση των εκπροσώπων.

Πολλές τέτοιες δηλώσεις είναι περιγραφικές, άλλες είναι οριστικές: Ένα μήνυμα έχει έναν αποστολέα μηνυμάτων. Μια τέτοια δήλωση κάνει έναν ισχυρισμό για τις έγκυρες περιπτώσεις της αρχιτεκτονικής: αναμένουμε να είμαστε σε θέση να προσδιορίσουμε τον αποστολέα μηνυμάτων σε οποιαδήποτε πραγματοποίηση της αρχιτεκτονικής. Αντιθέτως, οποιοδήποτε σύστημα για το οποίο δεν μπορούμε να προσδιορίσουμε τον αποστολέα ενός μηνύματος δεν εναρμονίζεται με την αρχιτεκτονική. Ακόμα κι αν μια υπηρεσία χρησιμοποιείται ανώνυμα, ο αποστολέας έχει ένα προσδιοριστικό αλλά δεν είναι δυνατό να συνδεθεί αυτό το προσδιοριστικό με ένα πραγματικό πρόσωπο ή μια οργάνωση.

Χάρτες έννοιας (Concept Maps)

Πολλές από τις έννοιες στην αρχιτεκτονική αναπαρίστανται με *τους χάρτες έννοιας* (Haas, 2004). Ένας χάρτης έννοιας είναι ένας άτυπος, γραφικός τρόπος να διευκρινιστούν οι βασικές έννοιες και οι σχέσεις μεταξύ τους. Για παράδειγμα το ακόλουθο διάγραμμα παρουσιάζει τρεις έννοιες που συσχετίζονται με διάφορους τρόπους. Κάθε κιβώτιο αντιπροσωπεύει μια έννοια, και κάθε βέλος αντιπροσωπεύει μια σχέση.



Εικόνα 1:Χάρτες έννοιας (Concept Maps) (Haas, 2004)

Η αξία ενός χάρτη έννοιας είναι ότι επιτρέπει τη γρήγορη πλοήγηση στις βασικές έννοιες και επεξηγεί πώς επιδρά η μία στην άλλη. Πρέπει να τονιστεί εντούτοις, ότι αυτά τα διαγράμματα είναι βοηθήματα πλοήγησης, το γραπτό κείμενο είναι η πηγή ορισμού αυτών.

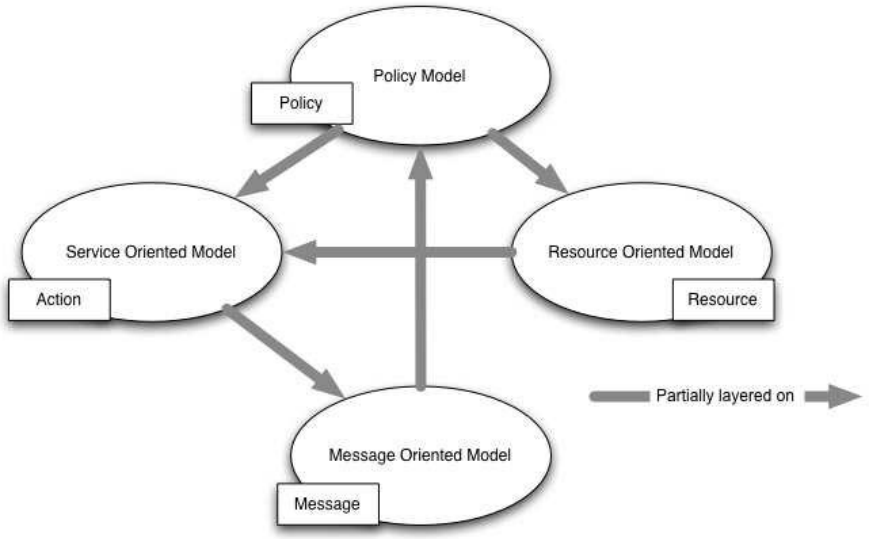
Μοντέλο (Model)

Ένα μοντέλο (Haas, 2004) είναι ένα συνεπές υποσύνολο της αρχιτεκτονικής που περιστρέφεται χαρακτηριστικά γύρω από μια ιδιαίτερη πτυχή της γενικής αρχιτεκτονικής. Αν και τα διαφορετικά μοντέλα μοιράζονται τις ίδιες έννοιες (συνήθως από διαφορετικές απόψεις) ο σημαντικότερος ρόλος ενός μοντέλου είναι να εξηγηθεί και να ενθυλακώσει ένα σημαντικό θέμα/ιδιότητα μέσα στη γενική αρχιτεκτονική υπηρεσιών Ιστού.

Για παράδειγμα, το προσανατολισμένο στο μήνυμα μοντέλο εστιάζει και εξηγεί τις υπηρεσίες Ιστού αυστηρά από την προοπτική ανταλλαγής μηνυμάτων. Συγκεκριμένα, δεν προσπαθεί να συσχετίσει τα μηνύματα με τις παρεχόμενες υπηρεσίες. Το προσανατολισμένο στις υπηρεσίες μοντέλο, εντούτοις, βρίσκεται πάνω από το προσανατολισμένο προς το μήνυμα μοντέλο και το επεκτείνει, προκειμένου να εξηγηθούν οι θεμελιώδεις έννοιες που περιλαμβάνονται στην υπηρεσία. Ουσιαστικά εξηγεί το σκοπό των μηνυμάτων στο προσανατολισμένο προς το μήνυμα μοντέλο.

Α.3.2 Τα αρχιτεκτονικά μοντέλα

Η αρχιτεκτονική των Υπηρεσιών Διαδικτύου ακολουθεί τέσσερα μοντέλα (Haas, 2004). Κάθε μοντέλο ονομάζεται/χαρακτηρίζεται από τη βασική έννοια που το μοντέλο διαχειρίζεται.



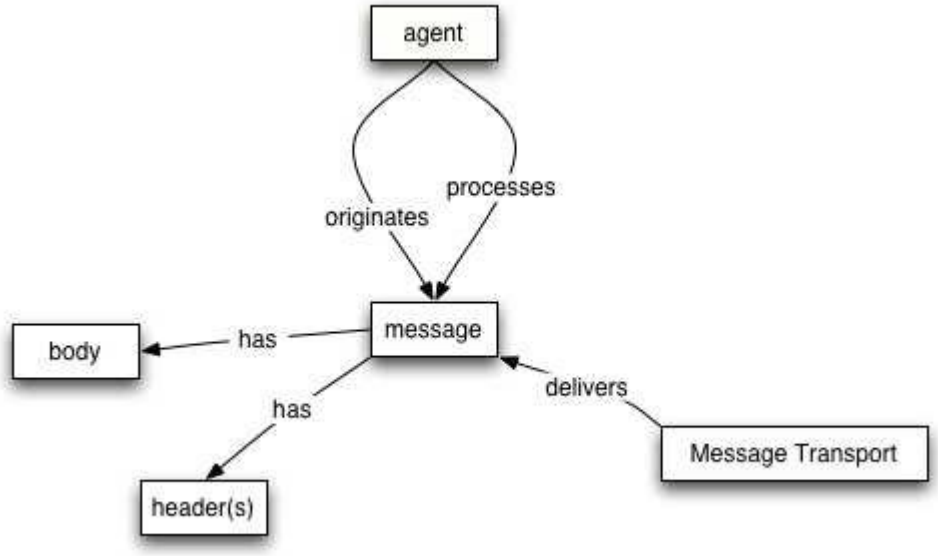
Εικόνα 2: Αρχιτεκτονικά Μοντέλα (Haas, 2004)

Τα τέσσερα μοντέλα είναι:

Το προσανατολισμένο στο μήνυμα μοντέλο (Haas, 2004). Το μοντέλο αυτό εστιάζει στα μηνύματα, τη δομή των μηνυμάτων, στη μεταφορά των μηνυμάτων κλπ — χωρίς ιδιαίτερη αναφορά ως προς το λόγο ύπαρξης των μηνυμάτων, ούτε στη σημασία τους.

Η ουσία του μοντέλου μηνυμάτων περιστρέφεται γύρω από μερικές βασικές έννοιες που φαίνονται στο σχήμα: ο εκπρόσωπος (agent) που στέλνει και λαμβάνει τα μηνύματα, τη δομή του μηνύματος από την άποψη των επικεφαλίδων των μηνυμάτων και των σωμάτων αυτών και των μηχανισμών που χρησιμοποιούνται για να παραδώσουν τα μηνύματα. Φυσικά, υπάρχουν πρόσθετες λεπτομέρειες που εξετάζονται: ο ρόλος των πολιτικών και πώς ελέγχουν το μοντέλο επιπέδου μηνυμάτων.

Το απλοποιημένο διάγραμμα παρουσιάζει τις βασικές έννοιες και τις βασικές σχέσεις.



Εικόνα 3: Απλοποιημένο Προσανατολισμένο στο Μήνυμα Μοντέλο (Haas, 2004)

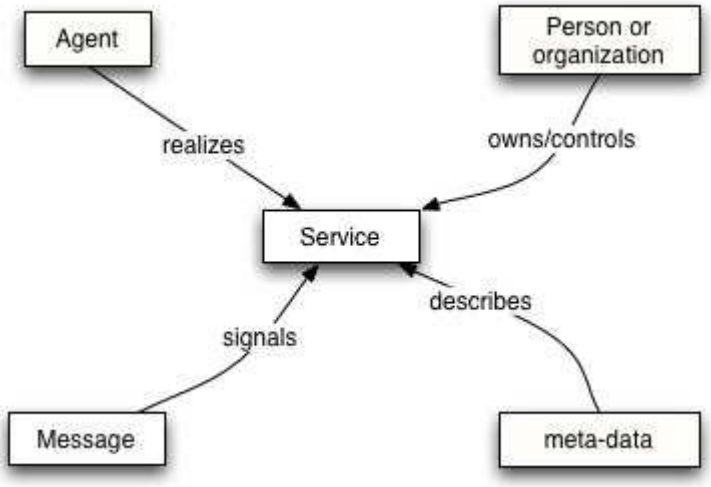
Το *Προσανατολισμένο στις Υπηρεσίες μοντέλο* (Haas, 2004). Το μοντέλο αυτό εστιάζει στα χαρακτηριστικά της υπηρεσίας, τις δράσεις (actions) κλπ. Ενώ σε οποιοδήποτε κατανεμημένο σύστημα, οι υπηρεσίες δεν μπορούν να υλοποιηθούν

επαρκώς χωρίς μερικά μέσα του μηνύματος, το αντίστροφο δεν ισχύει: τα μηνύματα δεν χρειάζεται να συσχετισθούν με υπηρεσίες.

Το Υπηρεσιοστρεφές μοντέλο είναι το πιο σύνθετο από όλα τα μοντέλα αρχιτεκτονικής. Εντούτοις, και αυτό περιστρέφεται γύρω από μερικές βασικές ιδέες. Μια υπηρεσία πραγματοποιείται από έναν εκπρόσωπο (agent) και χρησιμοποιείται από έναν άλλο εκπρόσωπο (agent). Οι υπηρεσίες κοινοποιούνται με τη βοήθεια των μηνυμάτων που ανταλλάσσονται μεταξύ των εκπροσώπων παρόχων και των εκπροσώπων αιτούντων.

Μια πολύ σημαντική πτυχή των υπηρεσιών είναι η σχέση τους με τον πραγματικό κόσμο: οι υπηρεσίες χρησιμοποιούνται συνήθως για να προσφέρουν λειτουργικότητα στον πραγματικό κόσμο. Αυτό μοντελοποιείται με την εισαγωγή της έννοιας του ιδιοκτήτη υπηρεσίας — όπου, εάν είναι πρόσωπο ή οργανισμός, έχει την ευθύνη για την υπηρεσία στον πραγματικό κόσμο.

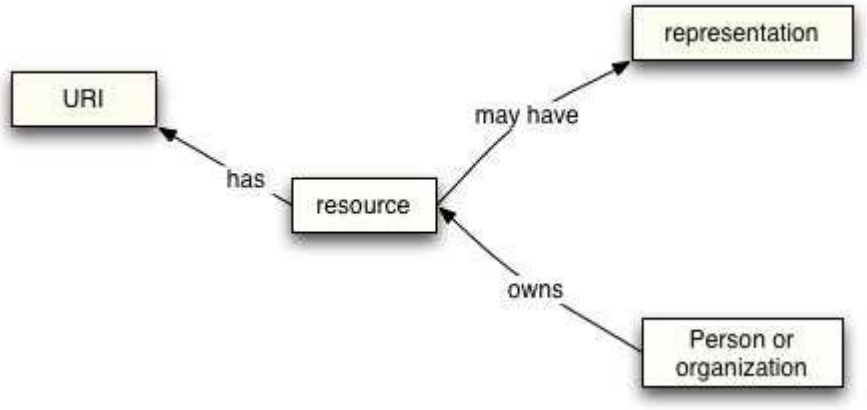
Τέλος, το υπηρεσιοστρεφές μοντέλο χρησιμοποιεί τα μεταδεδομένα, τα οποία, αποτελούν μια βασική ιδιότητα των υπηρεσιοστρεφών αρχιτεκτονικών. Αυτά τα μεταδεδομένα χρησιμοποιούνται για να τεκμηριώσουν πολλές ιδιότητες των υπηρεσιών: από τις λεπτομέρειες της διεπαφής και των δεσμεύσεων μεταφοράς μέχρι τη σημασιολογία της υπηρεσίας και τι περιορισμοί πολιτικής μπορεί να υπάρξουν στην υπηρεσία. Η Παροχή ακριβών περιγραφών είναι το κλειδί της επιτυχούς ανάπτυξης και χρήσης των υπηρεσιών μέσω του Διαδικτύου.



Εικόνα 4: Απλοποιημένο προσανατολισμένο στην υπηρεσία μοντέλο. (Haas, 2004)

Το *προσανατολισμένο στους πόρους* (Haas, 2004) *μοντέλο*. Το μοντέλο αυτό εστιάζει στους πόρους που υπάρχουν και έχουν ιδιοκτήτες.

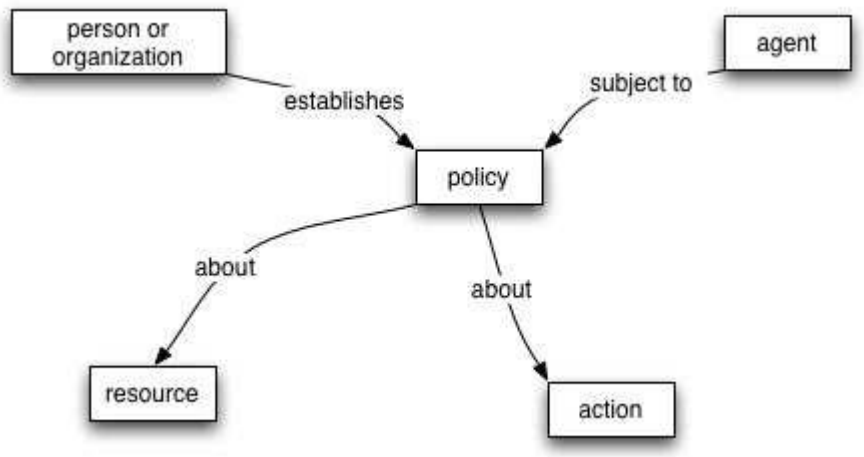
Το μοντέλο των πόρων υιοθετείται από την Αρχιτεκτονική Ιστού υπό την έννοια του πόρου και ενσωματώνει τις σχέσεις μεταξύ των πόρων και των ιδιοκτητών. Οι πόροι είναι θεμελιώδεις έννοιες οι οποίες υποστηρίζουν το μεγαλύτερο μέρος του Ιστού και των υπηρεσιών αυτού, για παράδειγμα, μία Υπηρεσία Ιστού είναι ένα αυτόνομο είδος πόρου το οποίο είναι σημαντικό για την αρχιτεκτονική Υπηρεσιών Ιστού. Το μοντέλο των πόρων, εστιάζει στα σημαντικά χαρακτηριστικά των πόρων που είναι συναφή με την έννοια του πόρου, ανεξάρτητα από το ρόλο που έχει ο πόρος στη δομή της Υπηρεσίας Ιστού. Για το λόγο αυτό εστιάζουμε σε θέματα όπως η ιδιοκτησία του πόρου, πολιτικές συναφής με τους πόρους και άλλα. Τότε, λόγω του γεγονότος ότι οι Υπηρεσίες είναι πόροι, αυτές οι ιδιότητες κληροδοτούνται στις Υπηρεσίες Ιστού από τους πόρους.



Εικόνα 5: Απλοποιημένο προσανατολισμένο στους πόρους μοντέλο. (Haas, 2004)

Το *προσανατολισμένο στην πολιτική μοντέλο* (Haas, 2004). Το μοντέλο αυτό εστιάζει στους περιορισμούς συμπεριφοράς των εκπροσώπων και των υπηρεσιών. Το γενικεύουμε στους πόρους δεδομένου ότι οι πολιτικές μπορούν να ισχύσουν εξίσου για τα έγγραφα (όπως οι περιγραφές των υπηρεσιών) καθώς επίσης και τους ενεργούς υπολογιστικούς πόρους.

Οι πολιτικές είναι για τους πόρους. Εφαρμόζονται στους εκπροσώπους που μπορεί να προσπαθήσουν να έχουν πρόσβαση σε κάποιους πόρους, και τίθενται σε ισχύ, ή εγκαθίστανται, από τους ανθρώπους που είναι υπεύθυνοι για τον πόρο.



Εικόνα 6: Απλοποιημένο προσανατολισμένο στην πολιτική μοντέλο. (Haas, 2004)

Α.3.3 Μοντέλα, Αρχιτεκτονικές και Πρωτόκολλα

Όπως αναφέρθηκε ανωτέρω μία αρχιτεκτονική μπορεί να αναπαρασταθεί από περισσότερα του ενός από τα ανωτέρω αρχιτεκτονικά μοντέλα. Παρόλα αυτά, οι αρχιτεκτονικές των επικρατουσών τεχνολογιών υπηρεσιών Ιστού SOAP και REST, περιγράφονται καλύτερα από το προσανατολισμένο στις υπηρεσίες και τους πόρους αντίστοιχα μοντέλο. Έτσι η τεχνολογία SOAP αποτελεί υλοποίηση της προσανατολισμένης στις υπηρεσίες αρχιτεκτονικής (SOA) (Weerawarana, 2008), ενώ η τεχνολογία REST της προσανατολισμένης στους πόρους αρχιτεκτονικής (ROA) (Richardson , 2007).

Είναι πολλοί οι συγγραφείς που αναφέρουν το πρωτόκολλο REST ως μία υλοποίηση αρχιτεκτονικής SOA (He, 2003). Αντίστοιχα, υπάρχουν και επικριτές της Προσανατολισμένης στις Υπηρεσίες Αρχιτεκτονικής SOA. Οι τελευταίοι υποστηρίζουν ότι ο ορισμός των SOA δεν είναι πλήρως καθορισμένος (Richardson , 2007) . Το W3C στην Αρχιτεκτονική των Web Services (όπου στην ουσία παρουσιάζει την αρχιτεκτονική SOA και την υλοποίηση SOAP) αναφέρεται στο REST (Haas, 2004) ως άλλο ένα πιο περιορισμένο αρχιτεκτονικό στυλ για αξιόπιστες εφαρμογές Ιστού. Παρόλα αυτά αναφέρει ότι η αρχιτεκτονική αυτή ενέπνευσε τόσο την ομάδα εργασίας του W3C, όσο και άλλους δημιουργούς Υπηρεσιών Ιστού.

Τα επόμενα δύο κεφάλαια αφιερώνονται στην εκτενή ανάλυση των δύο αυτών αρχιτεκτονικών μοντέλων και στις αντίστοιχες υλοποιήσεις αυτών.

Α.4 Πλεονεκτήματα σε σχέση με παλαιότερες κατανεμημένες τεχνολογίες

Τα πλεονεκτήματα που παρουσιάζουν τα WS σε σχέση με προηγούμενες κατανεμημένες τεχνολογίες περιλαμβάνουν τα ακόλουθα.

Α.4.1 Ευκολότερος χειρισμός δεδομένων

Το κυριότερο πρόβλημα στις κατανεμημένες τεχνολογίες ήταν το λεγόμενο tight-coupling ή στα ελληνικά η *στενή σύζευξη*. Μια εφαρμογή που καλούσε μια άλλη απομακρυσμένη ήταν αυστηρά δεμένη με αυτή από την κλήση λειτουργίας (function call) που εκτελούσε και τις παραμέτρους που περνούσε. Στα περισσότερα συστήματα πριν από την έλευση των Web Services ο τρόπος επικοινωνίας ήταν μια σταθερή διεπαφή με λίγη έως καθόλου ευελιξία ή προσαρμοστικότητα στα περιβάλλοντα ή τις ανάγκες που μεταβάλλονται συνεχώς.

Τα Web Services χρησιμοποιούν τη γλώσσα XML η οποία μπορεί να περιγράψει οποιαδήποτε δεδομένα σε ένα πραγματικά ανεξάρτητο από πλατφόρμα τρόπο για ανταλλαγή αυτών των δεδομένων μεταξύ συστημάτων. Με αυτόν τον τρόπο οδηγούμαστε σε εφαρμογές με *χαλαρή σύζευξη* (loosely-coupled). Επιπλέον τα web services μπορούν να λειτουργήσουν σε πιο αφηρημένο επίπεδο στο οποίο μπορούν να επαναξιολογήσουν, να τροποποιήσουν ή να χειριστούν τύπους δεδομένων δυναμικά κατά περίπτωση. Έτσι σε τεχνικό επίπεδο τα web services μπορούν να χειριστούν δεδομένα πολύ ευκολότερα και να επιτρέψουν στο λογισμικό να επικοινωνεί πιο ελεύθερα (IBM, 2007).

Α.4.2 Απλότητα πρωτοκόλλου επικοινωνίας

Τα Web Services υλοποιούνται κατά βάση από τα πρότυπα SOAP ή REST. Τα πρότυπα αυτά είναι πολύ πιο απλά από εκείνα παλαιότερων τεχνολογιών όπως αυτά που χρησιμοποιούνταν από τα κατανεμημένα περιβάλλοντα CORBA, DCOM, RPC. Έτσι το να δημιουργήσει κανείς μια υλοποίηση SOAP ή REST που υπόκειται

στα πρότυπα (standards-compliant) είναι πολύ πιο εύκολο. Σήμερα μπορεί να βρει κανείς υλοποιήσεις των SOAP και REST από τις μεγαλύτερες εταιρίες πληροφορικής αλλά ακόμη και από μεμονωμένους προγραμματιστές, πράγμα αδιανόητο για παλαιότερες κατανεμημένες τεχνολογίες.

Α.4.3 Απλότητα υποδομής

Τα web services λειτουργούν (MSDN, 2001) με πρότυπες γλώσσες και πρωτόκολλα όπως η XML , το HTTP και το TCP/IP. Η πλειονότητα των εταιριών έχουν ήδη την δικτυακή υποδομή και τους ανθρώπους με γνώσεις και εμπειρία που τη συντηρούν. Έτσι το κόστος για την εφαρμογή των web services είναι σημαντικά μικρότερο από αυτό των προηγούμενων τεχνολογιών.

Α.4.4 Ευκολία στην επικοινωνία

Με τις προηγούμενες τεχνολογίες η συνεργασία μεταξύ εταιριών ήταν ένα θέμα διότι κατανεμημένες τεχνολογίες όπως CORBA και DCOM χρησιμοποιούσαν μη πρότυπες πόρτες, πολλές φορές άγνωστες σε όλα τα εμπλεκόμενα μέρη. Το γεγονός αυτό δεν επέτρεπε δυναμική συνεργασία λόγω του ότι απαιτούσε μια χειροκίνητη διαδικασία για τη συνεργασία μιας εταιρίας με τους συνεργάτες της. Τα web services μπορούν να χρησιμοποιήσουν (μεταξύ άλλων) το HTTP ως πρωτόκολλο μεταφοράς μέσω της θύρας 80 (πρότυπη θύρα για το HTTP). Με αυτόν τον τρόπο οδηγούμαστε σε ευκολότερες και δυναμικές συνεργασίες μεταξύ των συστημάτων των εταιριών (Developer.com, 2003).

Α.4.5 Διαλειτουργικότητα και ευκολία ανάπτυξης νέων εφαρμογών

Οι προηγούμενες κατανεμημένες τεχνολογίες υπέφεραν από ζητήματα διαλειτουργικότητας διότι κάθε προμηθευτής (vendor) υλοποιούσε το δικό του πρότυπο για distributed object messaging. Με την XML σαν το μόνο πρότυπο στα web services, συστήματα φτιαγμένα από διαφορετικές τεχνολογίες όπως η Java και το .Net μπορούν να επικοινωνήσουν μεταξύ τους. Επιπλέον λόγω της απλότητας της XML είναι πολύ πιο εύκολο να γραφτούν νέες εφαρμογές σε μικρό χρονικό διάστημα.

Α.4.6 Ασφάλεια

Όπως αναφέρθηκε και προηγουμένως στις προηγούμενες αρχιτεκτονικές με κατανεμημένες τεχνολογίες όπως CORBA και DCOM χρησιμοποιούνταν μη πρότυπες πόρτες. Σαν αποτέλεσμα η συνεργασία σήμαινε άνοιγμα «οπών» στα τείχη προστασίας (firewalls) κάτι που πολλές φορές δεν ήταν αποδεκτό από τους ανθρώπους της πληροφορικής σε μια εταιρία, αφού έθετε σε κίνδυνο στην ασφάλεια των συστημάτων. Τα web services χρησιμοποιούν (μεταξύ άλλων) τα HTTP και HTTPS ως πρωτόκολλα μεταφοράς και τα περισσότερα τείχη προστασίας επιτρέπουν την πρόσβαση μέσω των θυρών 80 και 443 (πρότυπες θύρες για τα HTTP και HTTPS) εξασφαλίζοντας έτσι μεγαλύτερο επίπεδο ασφάλειας.

Επιπρόσθετα, οι τεχνολογίες SOAP και REST βασίζονται στο HTTP πρωτόκολλο, επομένως κληρονομούν όλες τις αδυναμίες αλλά και τις δυνατότητες αυτού. Μια δυνατότητα που προσφέρει ένα επίπεδο ασφάλειας είναι η χρήση του SSL (HTTPS). Η ασφάλεια του επιπέδου μεταφοράς (SSL) ασχολείται με τη διασφάλιση της διαδρομής των μηνυμάτων, χωρίς να επηρεάζει τα ίδια τα μηνύματα. Εξασφαλίζει, ότι κανένα μη εξουσιοδοτημένο άτομο δε μπορεί να έχει πρόσβαση στα μηνύματα, μεταξύ δύο σημείων.

Τέλος στην τεχνολογία SOAP παρέχεται επιπλέον ασφάλεια μέσω πρόσθετων πρωτόκολλων που ορίζονται στα WS-* πρότυπα (SOAP, WSDL, UDDI). Έτσι εξασφαλίζονται τα μηνύματα όταν αυτά ληφθούν από έναν ενδιάμεσο κόμβο και προωθηθούν σε κάποιο επίπεδο πάνω από το επίπεδο μεταφοράς. Επομένως παρέχεται επιπρόσθετη προστασία σε σχέση με την αποκλειστική χρήση της ασφάλειας του επιπέδου μεταφοράς (SSL), με την οποία διασφαλίζεται το μήνυμά μας μόνο όσο αυτό μεταδίδεται (είναι πάνω στο καλώδιο).

Βιβλιογραφία

Ebizq (2002) «Why We Need Web Services Networks», Ebizq, http://www.ebizq.net/topics/web_services/features/1542.html?page=1

Hugo Haas and others (2004) «Web Services Architecture», W3C.

IBM (2007) «New to SOA and Web services», IBM, http://www.ibm.com/developerworks/webservices/newto/service.html

Rose India (2008) «Web Services - Web Services Tutorials», Rose India, http://www.roseindia.net/webservices/webservices.shtml

Weerawarana S., Curbera F., Leymann F., Storey T., Ferguson D. (2008). «Αρχιτεκτονική Πλατφόρμας Υπηρεσιών Ιστού» (μετάφραση Δημήτρης Καρτσακλής – Επιστημονική Επιμέλεια Δρ. Χρήστος Γεωργιάδης). Κλειδάριθμος, Αθήνα.

Hao He (2003). «What Is Service-Oriented Architecture». O'Reilly Xml.com.September,. http://www.xml.com/lpt/a/1292

Richardson L. & Ruby S. (2007). «RESTful Web Services». O'Reilly. Sebastopol, CA. May.

MSDN (2001), «XML Web Services Basics», MSDN, http://msdn2.microsoft.com/en-us/library/ms996507.aspx

Developer.com (2003), «Web Services Tutorial: Understanding XML and XML Schema», "Part 1" http://www.developer.com/services/print.php/2195981
"Part 2" http://www.developer.com/services/print.php/10928_2195981_2

Hugo Haas (2005) «Reconciling Web Services and REST Services», ECOWS 2005, **http://www.w3.org/2005/Talks/1115-hh-k-ecows/#(1)**

Κεφάλαιο Β: Αρχιτεκτονική SOAP

Β.1 Εισαγωγή

Το SOAP είναι ένα πρωτόκολλο με τη χρήση του οποίου μπορούν οι υπηρεσίες διαδικτύου να ανταλλάξουν πληροφορίες μεταξύ τους. Για την ανταλλαγή μηνυμάτων μεταξύ των υπηρεσιών, χρησιμοποιεί το δικό του πρότυπο, ενώ για την επικοινωνία, χρησιμοποιεί τα πρωτόκολλα HTTP και SMTP. Αυτές του οι ιδιότητες, η χρήση δηλαδή ευρέως κοινών – χρησιμοποιούμενων πρωτοκόλλων, το κάνει να είναι ανεξάρτητο από την εφαρμογή και το υλικό της υπολογιστικής πλατφόρμας. Αυτή η ανεξαρτησία είναι και το ζητούμενο για τις υπηρεσίες διαδικτύου.

Επίσης το SOAP είναι ένα ιδιαίτερα λιτό και απλό πρωτόκολλο, προσφέροντας μόνο τα βασικά συστατικά για την ανταλλαγή των πληροφοριών, ενώ παραλείπει πολλά χαρακτηριστικά σχετικά με την ασφάλεια και την αξιοπιστία. Αυτό όμως δεν είναι πρόβλημα, γιατί μας παρέχει τη δυνατότητα να το επεκτείνουμε προσθέτοντας εμείς τις δυνατότητες που χρειαζόμαστε.

Συνοψίζοντας, αναφέρουμε ότι το SOAP είναι :

- Ένα ελαφρύ πρωτόκολλο για την ανταλλαγή πληροφοριών μεταξύ των υπηρεσιών διαδικτύου. Με την έννοια ελαφρύ, εννοούμε ότι το SOAP διαθέτει δύο κύριες ιδιότητες, την αποστολή και λήψη HTTP πακέτων και την επεξεργασία XML μηνυμάτων, οι οποίες σε σύγκριση με το πρωτόκολλο COBRA είναι ελαφριές (Lucas, 2014)

- Ένας τρόπος επικοινωνίας, ανεξάρτητος από την υλοποίηση αφού χρησιμοποιεί την XML για την ανταλλαγή των πληροφοριών.

- Ανεξάρτητο από την γλώσσα προγραμματισμού.

- Επεκτάσιμο.

- Χρησιμοποιεί κοινά πρωτόκολλα για την επικοινωνία του (HTTP, SMTP).

Β.2 Ιστορική Αναδρομή

Στις αρχές του 1998, το SOAP άρχισε να αναπτύσσεται στα πλαίσια ομάδων εργασίας της Microsoft, οι οποίες διερευνούσαν κατά πόσο ο κατανεμημένος υπολογισμός μπορεί να βασιστεί στην XML. Δηλαδή, προσπαθούσαν να δουν αν ήταν εφικτή η επικοινωνία μεταξύ εφαρμογών μέσω κλήσεων απομακρυσμένων διαδικασιών (Remote Procedure Calls - RPCs), χρησιμοποιώντας απλά πρωτόκολλα δικτύου, όπως το HTTP.

Το 1999 έκανε την εμφάνιση της, η πρώτη έκδοση του πρωτοκόλλου (1.0). Το Μάιο του 2000, υποβλήθηκε στον W3C, από τη Microsoft, την IBM, την ARIBA και άλλες εταιρείες η νέα έκδοσή του η 1.1. Τον Σεπτέμβριου του ίδιου έτους ο W3C δημιούργησε μια νέα ομάδα εργασίας για το πρωτόκολλο XML με στόχο να καταλήξουν σε ένα νέο πρωτόκολλο ανταλλαγής πληροφοριών βασισμένο σε XML και να το συστήσει ως επίσημο πρότυπο. Η ομάδα εργασίας τον Ιούλιο του 2001 υπέβαλε ένα σχέδιο εργασίας, το SOAP 1.2, το οποίο έγινε σύσταση του W3C (Box, 2001)

Β.3 Μήνυμα SOAP

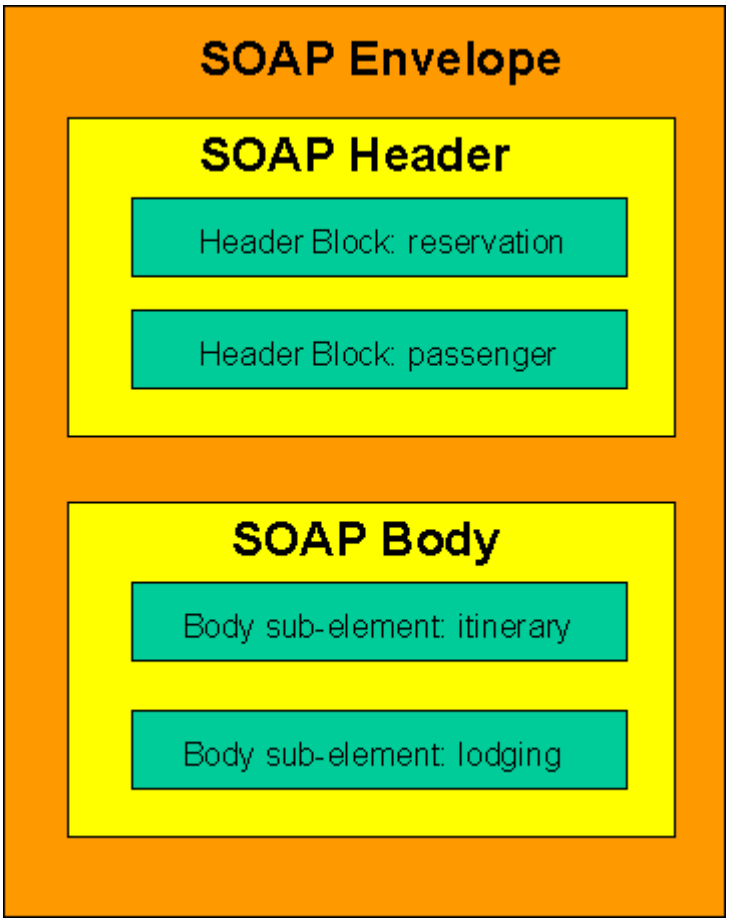
Κάθε μήνυμα SOAP είτε έρχεται από τον εξυπηρετούμενο, είτε από τον εξυπηρετητή, είναι ένα έγγραφο XML το οποίο περιέχει τα παρακάτω στοιχεία

Envelope: Αποτελεί το XML στοιχείο-ρίζα ενός μηνύματος SOAP. Χρησιμοποιείται για την αναγνώριση του XML εγγράφου ως μήνυμα SOAP. Σε αυτό περιέχονται όλα τα άλλα στοιχεία XML του μηνύματος SOAP.

Header: Είναι προαιρετικό και μας παρέχει τη δυνατότητα να επεκτείνουμε ένα μήνυμα SOAP περνώντας πληροφορίες που δεν ανήκουν στο ωφέλιμο φορτίο. Για παράδειγμα μπορούν να περαστούν οδηγίες ή γενικές πληροφορίες που σχετίζονται με την επεξεργασία του μηνύματος. Τα άμεσα εξαρτημένα στοιχεία του είναι τα **Header Block** και αντιπροσωπεύουν μια λογική ομαδοποίηση των

δεδομένων, που απευθύνονται στους ενδιάμεσους κόμβους SOAP που μπορεί να συναντήσει ένα μήνυμα στην πορεία του από τον αποστολέα στον παραλήπτη.

Body: Είναι υποχρεωτικό στοιχείο του μηνύματος και περιέχει τα δεδομένα που ανταλλάσσονται.



Εικόνα 7: Δομή μηνύματος SOAP (Mitra, 2001)

Β.3.1 Envelope

Το στοιχείο Envelope είναι υποχρεωτικό στοιχείο του μηνύματος, το οποίο προσδιορίζει ότι το συγκεκριμένο XML έγγραφο είναι μήνυμα SOAP. Όπως έχουμε αναφέρει είναι ρίζα του μηνύματος και περιέχει τα άλλα στοιχεία του μηνύματος.

Τα χαρακτηριστικά του στοιχείου είναι τα εξής:

- Το όνομα του στοιχείου πρέπει να είναι Envelope

- Το όνομα του namespace του να είναι: http://www.w3.org/2003/05/soap-envelope
- Περιέχει μηδέν ή περισσότερες ιδιότητες ορισμένες με namespace.
- Περιέχει ένα ή δυο παιδιά-στοιχεία με την ακόλουθη σειρά:
    - Προαιρετικό στοιχείο Header
    - Υποχρεωτικό στοιχείο Body (Gudgin et.al., 2007)

Β.3.2 Header

Όπως έχουμε αναφέρει το συγκεκριμένο στοιχείο είναι προαιρετικό. Προσφέρει όμως ένα πρόσθετο πλαίσιο για να καθορίσουμε περισσότερες πληροφορίες στο πεδίο της εφαρμογής. Με τον τρόπο αυτό μεταφέρουμε επιπρόσθετες πληροφορίες χωρίς να τροποποιούμε τον πυρήνα του μηνύματος. Για παράδειγμα μπορεί να χρησιμοποιηθεί για τον καθορισμό μιας ψηφιακής υπογραφής σε υπηρεσίες που προστατεύονται με κωδικό ή για τον καθορισμό ενός λογαριασμού για υπηρεσίες pay per use. Παρέχει δηλαδή έναν ανοικτό μηχανισμό για έλεγχο ταυτότητας, διαχείριση συναλλαγών και έγκρισης πληρωμής.

Οι λεπτομέρειες του στοιχείου είναι σκόπιμα ανοιχτές για να παρέχουν μεγάλη ευελιξία στους παρόχους εφαρμογών. Το πρωτόκολλο καθορίζει ότι πρέπει να περιλαμβάνει τα ακόλουθα:

- Το όνομα πρέπει να είναι Header.
- Το namespace πρέπει να έχει τιμή http://www.w3.org/2002/12/soap-envelope
- Περιέχει μηδέν ή περισσότερα attribute. Μεταξύ αυτών μπορεί να είναι οποιοδήποτε από τα ακόλουθα, τα οποία έχουν προκαθορισμένη σημασία για το πρωτόκολλο και θα περιγραφούν αμέσως μετά:
    - encodingStyle

- role
- mustUnderstand
- relay

- Περιέχει μηδέν ή περισσότερα element.

B.3.2.1 Role attribute

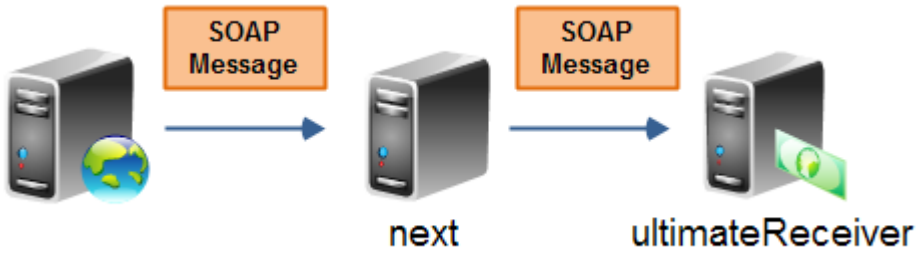
Το attribute αυτό, χρησιμοποιείται για να καθοριστεί ο SOAP κόμβος στον οποίο απευθύνεται το Header. Έχει τις ακόλουθες ιδιότητες:

- Όνομα role.
- Namespace http://www.w3.org/2002/12/soap-envelope

Η τιμή του συγκεκριμένου attribute είναι ένα URI που καθορίζει το ρόλο που ένας κόμβος SOAP μπορεί να αναλάβει. Οι τιμές που μπορεί να πάρει είναι:

- ultimateReceiver: η οποία δηλώνει ότι μόνο οι κόμβοι που έχουν το ρόλο ultimate receivers μπορούν να επεξεργαστούν το Header. Όλοι οι άλλοι κόμβοι θα πρέπει να τον αγνοούν. Αν παραλειφθεί είναι σαν να έχει την τιμή αυτή
- next: το οποίο σημαίνει ότι όλοι κόμβοι μπορούν να επεξεργαστούν το Header.
- none: κανένας κόμβος δε μπορεί να επεξεργαστεί το Header.

Στην εικόνα που ακολουθεί μπορούμε να δούμε την πορεία ενός SOAP μηνύματος διαμέσου τριών κόμβων, ένας εκ των οποίων είναι ο αποστολέας.



Εικόνα 8: Πορεία ενός SOAP μηνύματος διαμέσου τριών κόμβων (Jenkov, n.a)

B.3.2.2 mustUnderstund attribute

Χρησιμοποιείται για να καθορίσει αν η επεξεργασία ενός Header είναι υποχρεωτική ή προαιρετική. Μπορεί να πάρει δύο τιμές True και False (ή αντίστοιχα 1 και 0). Αν παραλειφθεί τότε θεωρείται ότι έχει τιμή False.

Όταν η τιμή της είναι αληθής τότε ο παραλήπτης αυτού του κομματιού του μηνύματος πρέπει να την κατανοήσει πλήρως και να αποδεχτεί όλους τους όρους της για να συνεχιστεί η επεξεργασία. Αντίστοιχα όταν η επικεφαλίδα έχει τιμή ψευδή (0), σημαίνει ότι είναι προαιρετική και το μήνυμα είναι επεξεργάσιμο και από παραλήπτες που αγνοούν πλήρως τις προδιαγραφές που ορίζει η επικεφαλίδα.

B.3.2.3 relay attribute

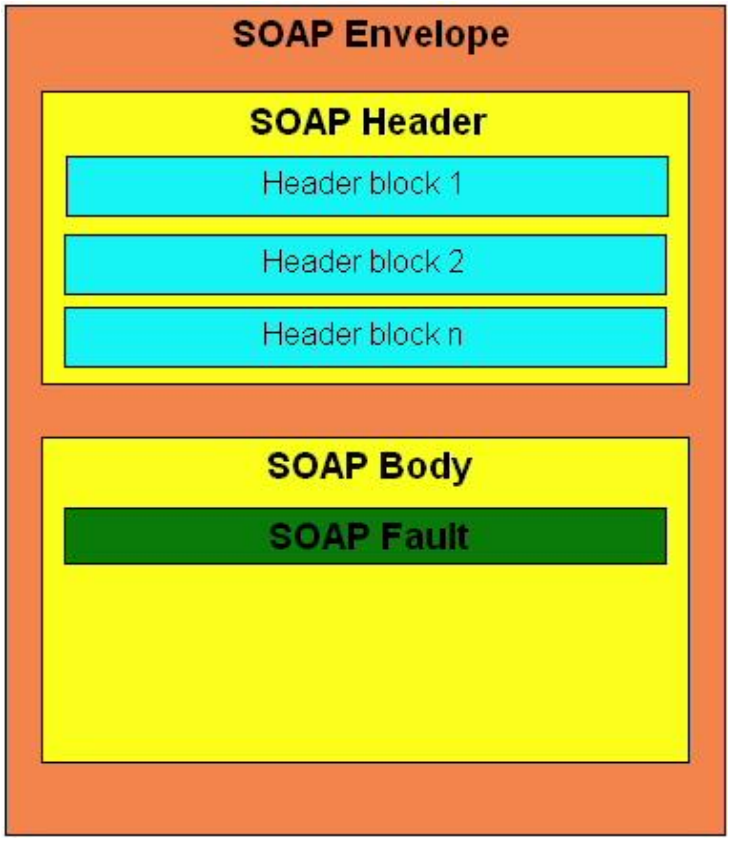
Χρησιμοποιείται για να υποδείξει αν το SOAP header block που προορίζεται για ένα κόμβο επεξεργασίας πρέπει να αναμεταδοθεί αν δεν υποστεί επεξεργασία. Μπορεί να πάρει μόνο δύο τιμές True ή False. Αν παραλειφθεί τότε θεωρείται ότι έχει τιμή False. (Cerami, 2002)

Β.3.3 Body

Το στοιχείο αυτό είναι υποχρεωτικό και περιλαμβάνει όλες τις πληροφορίες που θέλει να στείλει ο αποστολέας στον παραλήπτη. Πρέπει να έχει τα ακόλουθα χαρακτηριστικά:

- Το όνομά του να είναι Body.
- Το namespace να είναι http://www.w3.org/2002/12/soap-envelope
- Να περιέχει μηδέν ή περισσότερα attribute ορισμένα με namespace.
- Να περιέχει μηδέν ή περισσότερα στοιχεία παιδιά.

Αν κατά την επεξεργασία του μηνύματος προκληθεί κάποιο σφάλμα, τότε πρέπει να δημιουργηθεί ένα μήνυμα Fault. Ένα μήνυμα Fault είναι ένα απλό SOAP μήνυμα το οποίο περιέχει ένα στοιχείο Fault σαν μοναδικό στοιχείο-παιδί του Body, όπως απεικονίζεται στην επόμενη εικόνα.



Εικόνα 9: SOAP Fault μήνυμα (Δημητρίου, 2007)

B.3.3.1 SOAP Fault

Ένα στοιχείο SOAP Fault πρέπει να έχει τα ακόλουθα:

- Ένα όνομα με τιμή Fault
- Ένα namespace με τιμή http://www.w3.org/2002/12/soap-envelope
- Δύο ή περισσότερα στοιχεία παιδιά με την ακόλουθη σειρά
    - Code
    - Reason
    - Node
    - Role

- Detail

Το στοιχείο Code είναι υποχρεωτικό. Μέσα σε ένα στοιχείο Code μπορεί να φωλιάζει ένα στοιχείο Value και προαιρετικά ένα στοιχείο Subcode αν πρέπει να χωρίζουμε τους κωδικούς λάθους σε υποκατηγορίες. Το στοιχείο Value μπορεί να περιέχει μόνο ένα από τα παρακάτω:

- VersionMismatch: ο κόμβος που ανέφερε το λάθος, βρήκε ένα στοιχείο-ρίζα (root element) που δεν ήταν έγκυρο στοιχείο Envelope.
- MustUnderstand: Ο κόμβος επιστρέφει αυτό το λάθος, αν δεν καταλαβαίνει ένα συγκεκριμένο στοιχείο-παιδί του Header.
- DataEncodingUnknown: Αν ένα SOAP header block ή ένα SOAP body child που προοριζόταν για αυτόν τον κόμβο είχε κωδικοποίηση δεδομένων που δεν υποστηρίζει ο κόμβος.
- Sender: Το μήνυμα δεν ήταν σωστά δομημένο ή δεν περιείχε τη σωστή πληροφορία για να επιτύχει η επεξεργασία του.
- Receiver: Δεν μπόρεσε να γίνει η επεξεργασία του μηνύματος για λόγους που είχαν να κάνουν με την ίδια τη διαδικασία παρά με τα περιεχόμενα του μηνύματος. Το λάθος οφείλεται στον αποδέκτη και αν το μήνυμα αποστελλόταν κάποια στιγμή αργότερα μπορεί να ήταν επιτυχημένη η επεξεργασία του.

```
<env:Code>
  <env:Value>env:Sender</env:Value>
  <env:Subcode>
    <env:Value>env:Sender</env:Value>
    <env:Subcode>
    </env:Subcode>
  </env:Subcode>
</env:Code>
```

Το στοιχείο Reason είναι υποχρεωτικό. Περιέχει στοιχεία κειμένου, μέσω των οποίων δίνεται μια εξήγηση για το σφάλμα κατανοήσιμη από τον άνθρωπο. Μπορεί να περιέχει ένα lang attribute για να δείχνει τη γλώσσα που είναι γραμμένη το μήνυμα. Η τιμή του attribute πρέπει να έχει τον ISO κωδικό της γλώσσας.

Για παράδειγμα:

```
<env:Reason xmlns:xml="http://www.w3.org/XML/1998/namespace" >
 <env:Text >Error in Input Data</env:Text>
 <env:Text xml:lang="da">Fejl i input data</env:Text>
</env:Reason>
```

Το στοιχείο Node είναι προαιρετικό και χρησιμοποιείται για να παρέχει πληροφορίες για το ποιος κόμβος στη διαδρομή του SOAP μηνύματος, απέστειλε το SOAP Fault μήνυμα. Το στοιχείο Node έχει τα ακόλουθα χαρακτηριστικά:

- Το όνομά του πρέπει να είναι Node.
- Το όνομα του namespace να είναι `http://www.w3.org/2002/12/soap-envelope`

Ο τύπος αυτού του στοιχείου είναι xs:URI, δηλαδή κάθε κόμβος αναγνωρίζεται από ένα URI και η τιμή αυτού του στοιχείου είναι το URI του κόμβου επεξεργασίας στον οποίο προκλήθηκε το σφάλμα.

Όταν ο κόμβος που προκλήθηκε το σφάλμα είναι ο τελικός παραλήπτης, μπορεί να μην συμπεριλάβει αυτό το στοιχείο στο μήνυμα Fault. Αν όμως είναι κάποιος ενδιάμεσος κόμβος, τότε πρέπει να το συμπεριλάβει οπωσδήποτε.

Για παράδειγμα, ένα στοιχείο Node θα μπορούσε να ήταν το ακόλουθο:

```
<env:Node>http://jenkov.com/theNodeThatFailed</env:Node>
```

Το στοιχείο Role είναι και αυτό προαιρετικό και καθορίζει το ρόλο που είχε ο κόμβος στον οποίο συνέβει το σφάλμα. Τα χαρακτηριστικά του είναι:

- Το όνομά του πρέπει να είναι Role
- Το όνομα του namespace να είναι `http://www.w3.org/2002/12/soap-envelope`

Ο ρόλος που καθορίζεται μπορεί να πάρει μια από τις ακόλουθες τιμές:

- next
- none
- ultimateReceiver

Για παράδειγμα, ένα στοιχείο Role μπορεί να είναι το ακόλουθο:

```
<env:Role>
   http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver
</env:Role>
```

Το στοιχείο Detail είναι προαιρετικό. Χρησιμοποιείται για να μεταφέρει πληροφορίες σχετικά με λάθη με το SOAP Body. Οι πληροφορίες που περιέχει χρησιμοποιούνται για να περιγράψουν καλύτερα το λάθος, για παράδειγμα μπορεί να περιέχει πληροφορίες ότι το μήνυμα δεν είχε τα κατάλληλα πιστοποιητικά. Τα χαρακτηριστικά του είναι:

- Το όνομα του στοιχείου πρέπει να είναι Detail
- Το namespace πρέπει να είναι http://www.w3.org/2003/05/soap-envelope
- Μπορεί να περιέχει μηδέν ή περισσότερες ιδιότητες
- Μπορεί να περιέχει μηδέν ή περισσότερα στοιχεία-παιδιά, τα οποία ονομάζονται Detail Entry. Για τα στοιχεία – παιδιά ισχύουν τα ακόλουθα:
    - Το όνομά μπορεί να είναι ορισμένο με namespace
    - Μπορεί να περιέχει μηδέν ή περισσότερα στοιχεία-παιδιά
    - Μπορεί να περιέχει μηδέν ή περισσότερους κόμβους-παιδιά χαρακτήρων
    - Μπορεί να περιέχει μηδέν ή περισσότερες ιδιότητες

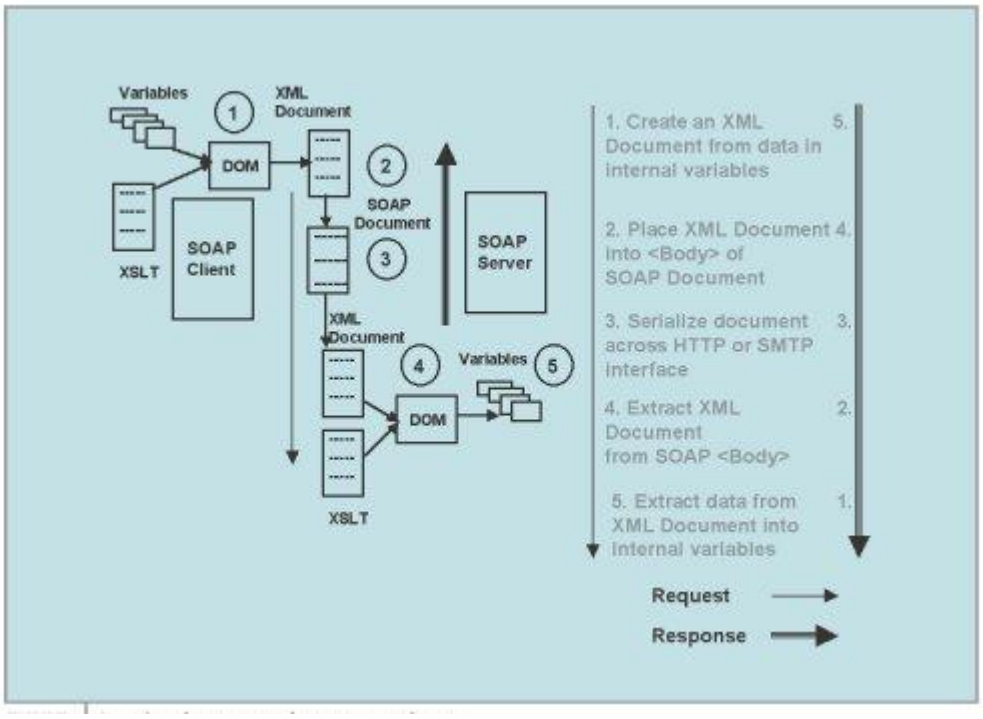
Ένα παράδειγμα χρήσης του είναι:

```
<env:Detail
    <jj:maxRelayTime
        xmlns:jj="http://jenkov.com" >10000</jj:MaxRelayTime>
  </env:Detail>
```

B.4 Μοντέλο προγραμματισμού

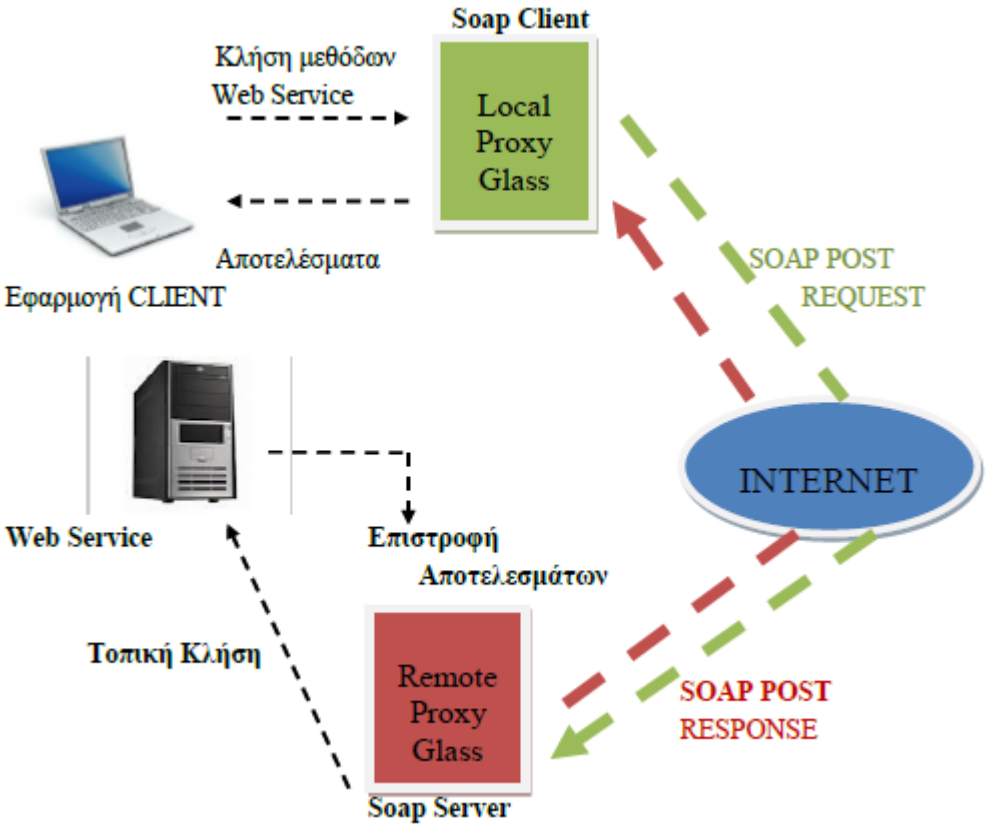
Το SOAP υποστηρίζει δύο μοντέλα προγραμματισμού: κυριολεκτικά (literals) εγγράφων, και κλήσεις απομακρυσμένων διαδικασιών (Remote Procedure Calls, RPC) (Weeravarana, 2005).

Για το μοντέλο κυριολεκτικά εγγράφων, το στοιχείο Body περιέχει ένα επιχειρηματικό έγγραφο, όπως μια παραγγελία ή μία κράτηση θέσης σε κάποια πτήση, το οποίο επεξεργάζεται η υπηρεσία που κάνει την αίτηση. Αφού επεξεργαστεί το έγγραφο, επιστρέφει ένα διαφορετικό έγγραφο το οποίο μπορεί, για παράδειγμα, να περιέχει πληροφορίες επιβεβαίωσης ότι η παραγγελία ή η κράτηση έχει ολοκληρωθεί.



Εικόνα 10: Μοντέλο κυριολεκτικά εγγράφων (SYS-CON, 2004)

Για το μοντέλο RPC το στοιχείο Body σε ένα μήνυμα αίτησης περιέχει ένα όνομα διαδικασίας προς κλήση, μαζί με ένα σύνολο κωδικοποιημένων ορισμάτων για τη διαδικασία αυτή. Αφού κληθεί η διαδικασία αυτή, επιστρέφονται ένα αποτέλεσμα και ένα σύνολο τιμών.



Εικόνα 11: Remote Procedure Call (Παπαργύρη, 2012)

B.5 Μοτίβα ανταλλαγής μηνυμάτων

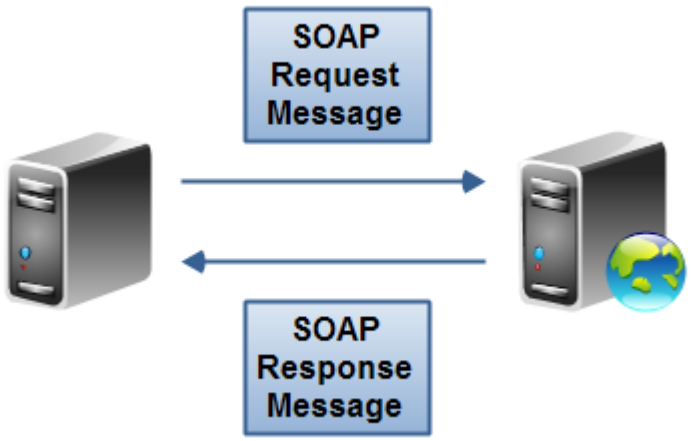
Το SOAP είναι ένα απλό πλαίσιο μηνυμάτων για ανταλλαγή μηνυμάτων σε μορφή XML μεταξύ ενός αρχικού αποστολέας και ενός τελικού αποδέκτη. Τα πιο ενδιαφέροντα σενάρια περιλαμβάνουν πολλαπλές ανταλλαγές μηνυμάτων μεταξύ των δύο αυτών κόμβων και καλούνται Μακροπρόθεσμες Συνδιαλέξεις (Weeravarana, 2005). Η πιο απλή από αυτές τις περιπτώσεις είναι η μορφή αίτηση-απάντηση (request-response) (Δημητρίου, 2007).

Β.5.1 Αίτηση – Απάντηση

Στην περίπτωση αυτή ο SOAP πελάτης στέλνει ένα SOAP μήνυμα αιτήματος στην υπηρεσία του SOAP κόμβου του παραλήπτη και η υπηρεσία απαντά με ένα SOAP μήνυμα απάντησης.

Η διαδικασία αυτή χρησιμοποιείται όταν ο SOAP πελάτης πρέπει να στείλει κάποια δεδομένα στην υπηρεσία για να εκτελέσει κάποια εργασία. Για παράδειγμα όταν ο πελάτης θέλει να αποθηκεύσει κάποια δεδομένα, πρέπει να τα στείλει με ένα μήνυμα SOAP αίτημα. Όταν η υπηρεσία ολοκληρώσει τη διαδικασία, απαντά με ένα μήνυμα SOAP απάντηση. Η σημασία των μηνυμάτων καθορίζεται από τις ίδιες τις εφαρμογές.

Η διαδικασία που προαναφέραμε απεικονίζεται στην ακόλουθη εικόνα.



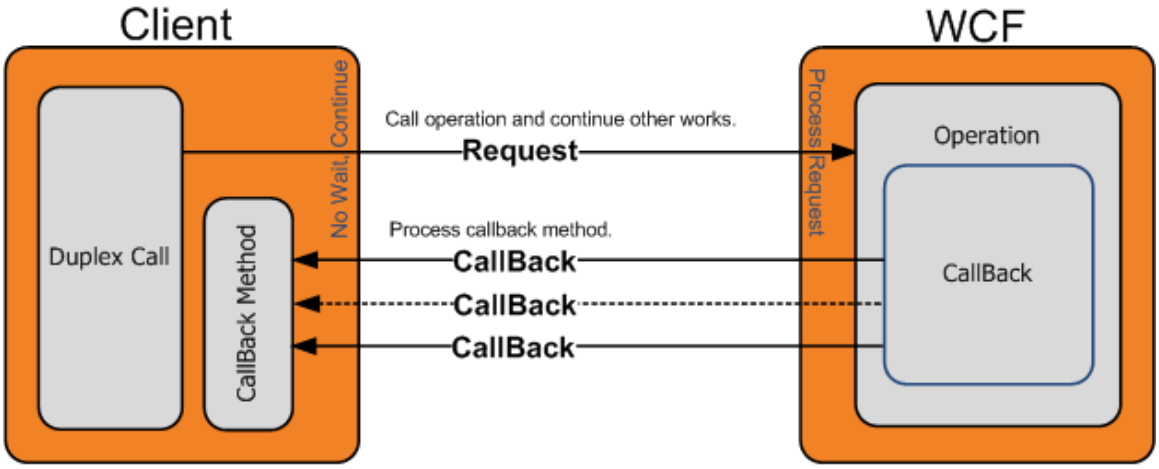
Εικόνα 12: Αίτηση – Απάντηση (Jenkov, n.a)

Β.5.2 Μακροπρόθεσμη Συνδιάλεξη

Ορισμένες επιχειρηματικές λειτουργίες απαιτούν ανταλλαγές μηνυμάτων μεταξύ υπηρεσιών, οι οποίες είναι πιο σύνθετες από μια απλή αίτηση/απάντηση. Αυτό μπορεί να οδηγήσει σε ανταλλαγές μηνυμάτων, οι οποίες συνεχίζονται μεταξύ δύο υπηρεσιών Ιστού για εκτεταμένες χρονικές περιόδους. Τέτοιου είδους μακροπρόθεσμες συνδιαλέξεις απιτούν πρόσθετες πληροφορίες συσχέτισης,ώστε να εξασφαλίζεται ότι η κατάσταση του μηνύματος διατηρείται μεταξύ των επικοινωνουσών υπηρεσιών.

Όπως ακριβώς συμβαίνει με τις συνδιαλέξεις αίτησης/απάντησης, η εξασφάλιση της σωστής διαχείρισης όλων των μηνυμάτων σε ένα μοτίβο μακροπρόθεσμης συνδιάλεξης, είναι μια εργασία που μπορεί να ανατεθεί στα επίπεδα εφαρμογής ή ενδιάμεσου λογισμικού των υπηρεσιών Ιστού. Αυτά τα επίπεδα μπορεί να χρειάζεται να ξεχωρίζουν πολλές περιπτώσεισμιας μακροπρόθεσμης συνδιάλεξης μεταξύ των ίδιων δύο υπηρεσιών.

Επομένως, ίσως είναι χρήσιμο να χρησιμοποιηθεί στο επιχειρηματικό έγγραφο ένα αναγνωριστικό συνδιάλεξης μέσα στο στοιχείο Body, ή μια ξεχωριστή κεφαλίδα η οποία θα περιέχει ένα αναγνωριστικό συνδιάλεξης που θα προστίθεται σε κάθεμήνυμα SOAP το οποίο αποτελεί μέρος της συνδιάλεξης.



Εικόνα 13: Παράδειγμα Μακροπρόθεσμης Συνδιάλεξης (Shemeer, 2013)

B.5 Παράδειγμα ενός SOAP μηνύματος

Στην ενότητα αυτή θα περιγράψουμε τα πεδία του SOAP μηνύματος μέσω ενός παραδείγματος από μια εφαρμογή ταξιδιωτικών κρατήσεων.

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
        env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
         env:mustUnderstand="true">
```

```
  <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
  <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
 </m:reservation>
 <n:passenger xmlns:n="http://mycompany.example.com/employees"
       env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
        env:mustUnderstand="true">
  <n:name>Åke Jógvan Øyvind</n:name>
 </n:passenger>
</env:Header>
<env:Body>
 <p:itinerary
  xmlns:p="http://travelcompany.example.org/reservation/travel">
  <p:departure>
   <p:departing>New York</p:departing>
   <p:arriving>Los Angeles</p:arriving>
   <p:departureDate>2001-12-14</p:departureDate>
   <p:departureTime>late afternoon</p:departureTime>
   <p:seatPreference>aisle</p:seatPreference>
  </p:departure>
  <p:return>
   <p:departing>Los Angeles</p:departing>
   <p:arriving>New York</p:arriving>
   <p:departureDate>2001-12-20</p:departureDate>
   <p:departureTime>mid-morning</p:departureTime>
   <p:seatPreference/>
  </p:return>
 </p:itinerary>
 <q:lodging
  xmlns:q="http://travelcompany.example.org/reservation/hotels">
  <q:preference>none</q:preference>
 </q:lodging>
</env:Body>
</env:Envelope>
```

Το παράδειγμα αυτό περιέχει ένα στοιχείο Envelope και δύο sub elements το Header και το Body.

Όπως βλέπουμε το στοιχείο Header περιέχει δύο Header Blocks (ένα reservation και ένα passenger) τα οποία έχουν οριστεί στο δικό τους namespace.

Το reservation στο http://travelcompany.example.org/reservation και το passenger στο http://mycompany.example.com/employees. Τα δύο Header Block περιέχουν κάποιες “meta” πληροφορίες όπως ένα timestamp και ένα ονοματεπώνυμο του ταξιδιώτη για τη συγκεκριμένη κράτηση.

Τα δύο αυτά πεδία, πρέπει να επεξεργαστούν από τον επόμενο SOAP κόμβο στη διαδρομή του μηνύματος ή από τον τελικό κόμβο, αν δεν υπάρχει ενδιάμεσος. Το ότι πρέπει να επεξεργαστούν από τον επόμενο κόμβο καθορίζεται από την παρουσία του attribute env:role με τιμή http://www.w3.org/2003/05/soap-envelope/role/next. Το attribute env:mustUnderstand με τιμή true, δηλώνει ότι οι κόμβοι που θα επεξεργαστούν το Header πρέπει να το κάνουν με τρόπο σύμφωνο με τις προδιαγραφές για την επεξεργασία του Header, διαφορετικά να μην επεξεργαστούν το μήνυμα και να δηλώσουν σφάλμα.

Η απόφαση για το ποια δεδομένα θα είναι στο Header και ποια στο Body, καθορίζεται στη φάση του σχεδιασμού της εφαρμογής. Αυτό που θα πρέπει να λάβουμε υπόψη είναι το ότι οι Header element μπορεί να απευθύνονται σε διάφορους κόμβους που θα προκύψουν κατά τη διαδρομή του μηνύματος. Στο παράδειγμά μας τα δεδομένα στο Header μπορούν να αλλοιωθούν από τους ενδιάμεσους SOAP κόμβους, προσθέτοντας πληροφορίες που σχετίζονται με ταξιδιωτικές πολιτικές.

Το στοιχείο Body καθώς και τα παιδιά του itinerary και lodging, προορίζεται για την ανταλλαγή πληροφοριών μεταξύ του αρχικού αποστολέα SOAP και του παραλήπτη του μηνύματος, ο οποίος είναι και η εφαρμογή της ταξιδιωτικής υπηρεσίας. Τα δεδομένα αυτά γίνονται κατανοητά μόνο από τον τελικό αποδέκτη με κανόνες που δεν περιγράφονται στο πρωτόκολλο SOAP, αλλά έχουν καθοριστεί κατά τη σχεδίαση της εφαρμογής (Gudgin et.al., 2007).

Βιβλιογραφία

Δημητρίου Θεοχ. (2007), «SOAP», http://www.it.uom.gr/project/soap/Theory/SOAP.html

Box D. (2001) «A Brief History of SOAP», http://www.xml.com/pub/a/ws/2001/04/04/soap.html

Cerami E. (2002) «Web Services Essentials Distributed Applications with XML-RPC, SOAP, UDDI & WSDL», O'Reilly

Gudgin M., Hadley M., Mendelsohn N., Moreau J.J., Nielsen H.F., Karmarkar A., Lafon Y. (2007), « SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)», http://www.w3.org/TR/2007/REC-soap12-part1-20070427/#soaprole

Jenkov J. (n.a), «SOAP Roles», http://tutorials.jenkov.com/soap/roles.html

Lucas S. (2014) «Networking», http://orb.essex.ac.uk/CE/CE881/lectures/

Mitra N. (2001), «SOAP Version 1.2 Part 0: Primer», http://www.w3.org/TR/2001/WD-soap12-part0-20011217/

Παπαργύρη Τ. (2012), «Σχεδίαση και Ανάπτυξη WEB SERVICE», Διπλωματική Εργασία Πανεπιστήμιο Πατρών, http://nemertes.lis.upatras.gr/jspui/bitstream/10889/6122/3/Nimertis_Papargyri(ele).pdf

Weerawarana S.και άλλοι (2005), «Αρχιτεκτονική Πλατφόρμας Υπηρεσιών Ιστού», Κλειδάριθμος (Επιστημονική Επιμέλεια Ελληνικής Έκδοσης Δρ. Γεωργιάδης Χ.)

SYS-CON (2004), «SOAP's Two Messaging Styles», SYS-CON Media Inc, http://www2.sys-con.com/itsg/virtualcd/webservices/archives/0311/hollar/index.html

Shemeer NS (2013), « WCF - Message Exchange Patterns (MEPs)», http://www.codeproject.com/Articles/566543/WCF-Message-Exchange-Patterns-MEPs

Κεφάλαιο Γ: Αρχιτεκτονική REST

Γ.1 Εισαγωγή

Η REST (Representational State Transfer), είναι μια αρχιτεκτονική λογισμικού για κατανεμημένα συστήματα όπως ο παγκόσμιος ιστός. Ο όρος REST εισήχθη για πρώτη φορά το 2000 από τον Roy Fielding στη διδακτορική διατριβή του. Η ανάπτυξή του έγινε παράλληλα με αυτή του HTTP 1.1.

Το REST εμφανίζεται τα τελευταία χρόνια να είναι το ανερχόμενο μοντέλο για web services. Λόγω της απλότητας του, έχει εκτοπίσει σε ένα μεγάλο βαθμό το SOAP και το WSDL.

Την αρχιτεκτονική REST, την αποτελούν οι πελάτες και οι εξυπηρετητές. Οι πελάτες ξεκινούν αιτήματα προς τους εξυπηρετητές, οι οποίοι με τη σειρά τους, αφού τα επεξεργαστούν, επιστρέφουν στους πελάτες που τους έστειλαν το αίτημα τις ανάλογες απαντήσεις. Τα αιτήματα και οι απαντήσεις έχουν υλοποιηθεί γύρω από αναπαραστάσεις των πόρων. Πόρος μπορεί να είναι οποιοδήποτε πληροφοριακό αντικείμενο με κάποια σημασία και η αναπαράστασή του αποτυπώνει ένα στιγμιότυπό του σε μια δεδομένη χρονική στιγμή ή μετά από κάποια ενέργεια που εκτέλεσε. Στο κεφάλαιο αυτό θα εξηγήσουμε αναλυτικά τις έννοιες που αναφέραμε καθώς θα περιγράψουμε τη δομή της αρχιτεκτονικής.

Γ2 Περιγραφή περιορισμών – στοιχείων

Στην ενότητα αυτή θα περιγράψουμε τα βήματα που ακολουθήθηκαν για τη δημιουργία του REST.

Υπάρχουν δύο τρόποι που ένας σχεδιαστής μπορεί να ξεκινήσει την υλοποίηση μιας αρχιτεκτονικής:

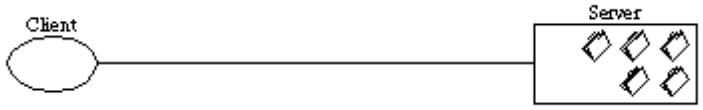
- Η πρώτη προσέγγιση όπου ο σχεδιαστής θα ξεκινήσει χωρίς να έχει τίποτα, με μια λευκή πλάκα και υλοποιεί την αρχιτεκτονική του από οικεία συστατικά, μέχρι να ικανοποιηθούν οι ανάγκες του συστήματος.
- Ο δεύτερος τρόπος, όπου ο σχεδιαστής ξεκινά, έχοντας το σύστημα που χρειάζεται στο σύνολό του και αρχίζει να εφαρμόζει περιορισμούς στα στοιχεία του συστήματος με σκοπό να διαφοροποιήσει το χώρο και να του προσδώσει τα επιθυμητά χαρακτηριστικά.

Το Rest έχει αναπτυχθεί χρησιμοποιώντας το δεύτερο τρόπο. Το αρχικό σύστημα, στο οποίο θα στηριχθούμε για την υλοποίηση της Rest είναι το Null style (το Null δηλώνει την έλλειψη περιορισμών στο σύστημα) το οποίο βλέπουμε στην επόμενη εικόνα.

Εικόνα 14: NULL style (Fielding, 2000)

Γ.2.1 Client Server

Ο πρώτος περιορισμός που εφαρμόζουμε είναι ο Client – Server. Ο περιορισμός αυτός, αφορά το διαχωρισμό του user interface (διεπαφή του χρήστη) από το σύστημα αποθήκευσης των δεδομένων. Με τον τρόπο αυτό βελτιώνουμε τη δυνατότητα επέκτασης ενώ παράλληλα η διεπαφή του χρήστη μπορεί να μεταφερθεί εύκολα σε άλλες πλατφόρμες. Το πιο σημαντικό όμως είναι ότι ο διαχωρισμός αυτός επιτρέπει την αυτόνομη εξέλιξη των συστατικών του.



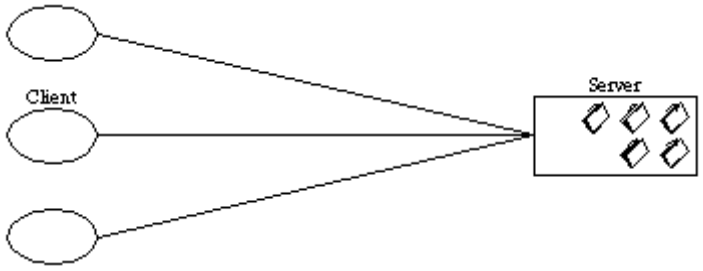
Εικόνα 15: Client – Server (Fielding, 2000)

Γ.2.2 Stateless

Στη συνέχεια προσθέτουμε τον περιορισμό ότι η αλληλεπίδραση Client – Server πρέπει να είναι stateless. Με τον όρο stateless εννοούμε ότι κάθε αίτημα από τον Client στον Server πρέπει να περιέχει όλη την απαραίτητη πληροφορία για να γίνει κατανοητό από τον Server, χωρίς να χρησιμοποιεί κάποιο αποθηκευμένο κείμενο στον Server, δηλαδή να μην εξαρτάται από προηγούμενη κατάσταση. Επομένως όλη η κατάσταση – πληροφορία της συνεδρίας (session) αποθηκεύεται στον Client.

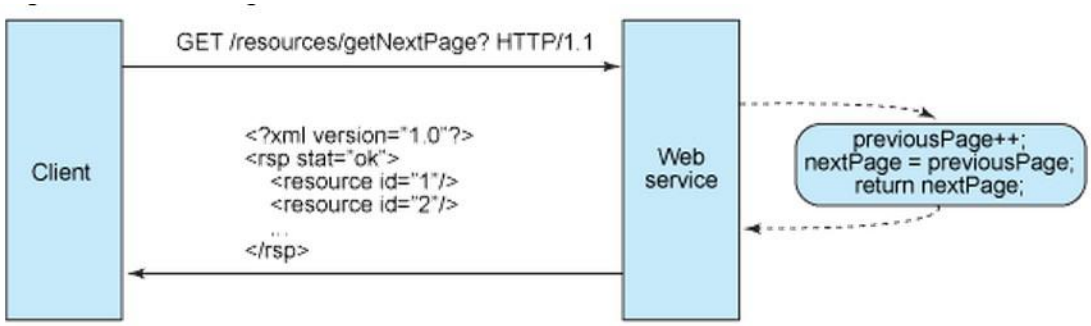
Ο συγκεκριμένος περιορισμός επάγει τις ακόλουθες ιδιότητες:

- Visibility: βελτιώνεται, γιατί ένα σύστημα παρακολούθησης για να καθορίσει τη φύση του αιτήματος αρκεί να εξετάσει το συγκεκριμένο αίτημα και τίποτα άλλο.
- Reliability: βελτιώνεται γιατί είναι ευκολότερη η ανάκαμψη του συστήματος από κάποια αποτυχία
- Scalability: επειδή δεν είναι απαραίτητη η αποθήκευση της κατάστασης των αιτημάτων, ο Server απελευθερώνει ότι πόρους είχε δεσμεύσει για την εξυπηρέτηση του αιτήματος μετά την ολοκλήρωσή του. Επίσης η υλοποίηση του λογισμικού από την πλευρά του server είναι απλούστερη γιατί δεν απαιτείται διατήρηση πληροφοριών μεταξύ αιτημάτων (Fielding, 2000).



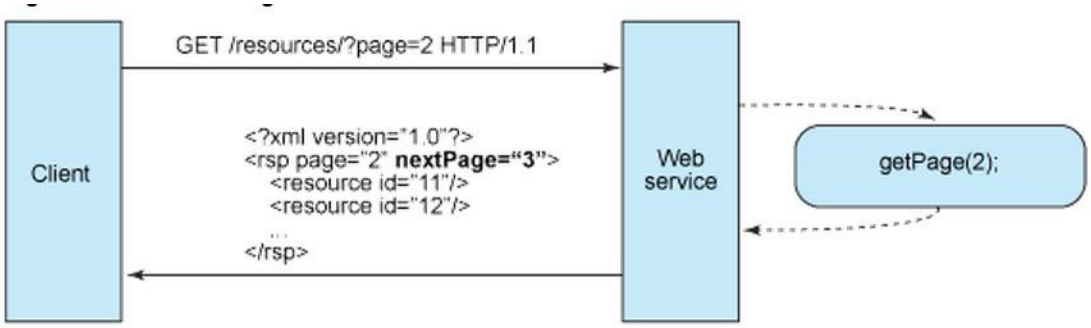
Εικόνα 16: Client Stateless Server (Fielding, 2000)

Για να ξεκαθαρίσουμε τη διαφορά μεταξύ statefull και stateless θα παρουσιάσουμε ένα παράδειγμα από την κάθε μια στις εικόνες που ακολουθούν:



Εικόνα 17: Stateful (Rodriguez, 2008)

Στην παράδειγμα με την stateful, ο εξυπηρετητής κρατάει τον τρέχοντα αριθμό της σελίδας (previousPage) που βλέπει ο πελάτης και υπολογίζει την επόμενη σελίδα που θα δει με βάση τον αριθμό αυτό (nextPage=previousPage++).



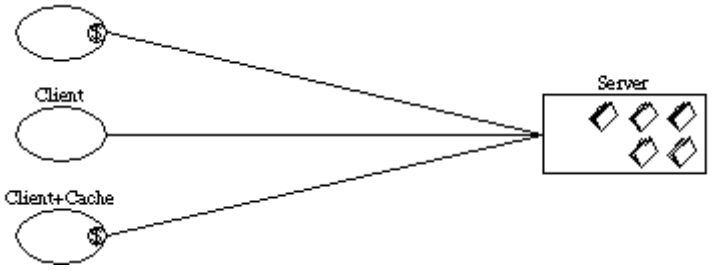
Εικόνα 18: Stateless (Rodriguez, 2008)

Στο stateless παράδειγμα, ο πελάτης γνωρίζει την τρέχουσα σελίδα, υπολογίζει την επόμενη και τη ζητά από τον εξυπηρετητή. Ο εξυπηρετητής δεν προβαίνει σε κανέναν υπολογισμό (Rodriguez, 2008).

Γ.2.3 Cache

Με σκοπό τη βελτίωση της αποτελεσματικότητας του δικτύου προσθέτουμε τον περιορισμό της cache, ο οποίος καθορίζει ότι τα δεδομένα που περιέχονται σε μια απάντηση σε ένα αίτημα πρέπει να έχουν δηλωθεί αν θα είναι cacheable ή όχι. Αν είναι cacheable, ο πελάτης μπορεί να τα αποθηκεύσει σε μια μνήμη cache για να τα χρησιμοποιήσει σε επόμενο αίτημα.

Η τεχνική αυτή χρησιμοποιείται ευρύτερα στο διαδίκτυο για να παρουσιάζει ίδια αιτήματα στους πελάτες χωρίς να αποστέλλονται στον Server. Έτσι μειώνεται ο αριθμός των αλληλεπιδράσεων μεταξύ Client – Server.



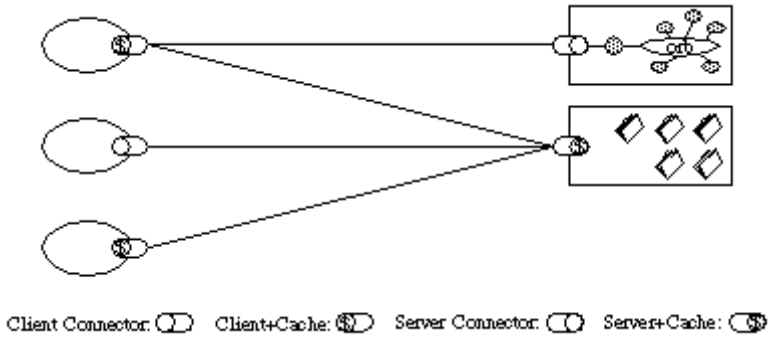
Εικόνα 19: Client - Cache - Stateless – Server (Fielding, 2000)

Γ.2.4 Uniform Interface

Το βασικό χαρακτηριστικό, το οποίο διακρίνει την αρχιτεκτονική REST, από τις άλλες αρχιτεκτονικές είναι η ενιαία διεπαφή μεταξύ των συστατικών του. Με την εφαρμογή της συγκεκριμένης αρχής της μηχανικής λογισμικού, επιτυγχάνεται :

- Η απλοποίηση της συνολικής αρχιτεκτονικής του συστήματος.
- Εφόσον οι υλοποιήσεις έχουν αποσυνδεθεί από τις υπηρεσίες που παρέχουν, ενθαρρύνεται η ανεξάρτητη εξέλιξή τους.

Το μειονέκτημα της συγκεκριμένης υλοποίησης είναι ότι η ενιαία διεπαφή υποβαθμίζει την αποτελεσματικότητα, γιατί οι πληροφορίες μεταφέρονται σε τυποποιημένη μορφή και όχι σε κάποια μορφή υλοποιημένη για τις συγκεκριμένες ανάγκες της εφαρμογής. Η διεπαφή REST έχει σχεδιαστεί για να είναι αποτελεσματική, για μεταφορά δεδομένων υπερμέσων. Για την υλοποίηση της ενιαίας διεπαφής είναι απαραίτητοι τέσσερις περιορισμοί στις διεπαφές. Οι περιορισμοί αυτοί, θα αναφερθούν στο Γ3.



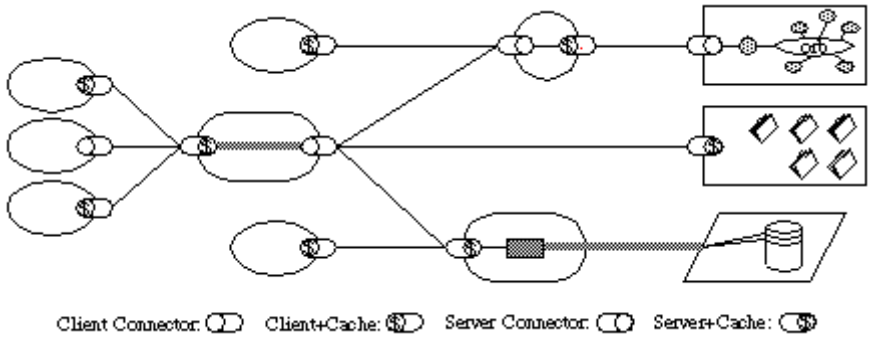
Εικόνα 20: Uniform - Client - Cache - Stateless – Server (Fielding, 2000)

Γ.2.5 Layered System

Για την περαιτέρω βελτίωση της συμπεριφοράς προσθέτουμε τον περιορισμό του σχεδιασμού σε επίπεδα. Ένα πολυεπίπεδο σύστημα επιτρέπει την εφαρμογή σε αυτά πολυεπίπεδων αρχιτεκτονικών που έχουν σαν πλεονέκτημα ότι κάθε στοιχείο της αρχιτεκτονικής δε γνωρίζει τίποτα πέρα από τις απαιτήσεις του/των επιπέδων που αλληλεπιδρά. Με τον τρόπο αυτό περιορίζουμε την πολυπλοκότητα του συστήματος και αυξάνουμε την ανεξαρτησία των στοιχείων στα διάφορα επίπεδα.

Με τη συγκεκριμένη υλοποίηση επιτυγχάνουμε την επεκτασιμότητα του συστήματος, επιτρέποντας την εξισορρόπηση φορτίου των υπηρεσιών σε πολλαπλά δίκτυα και Server.

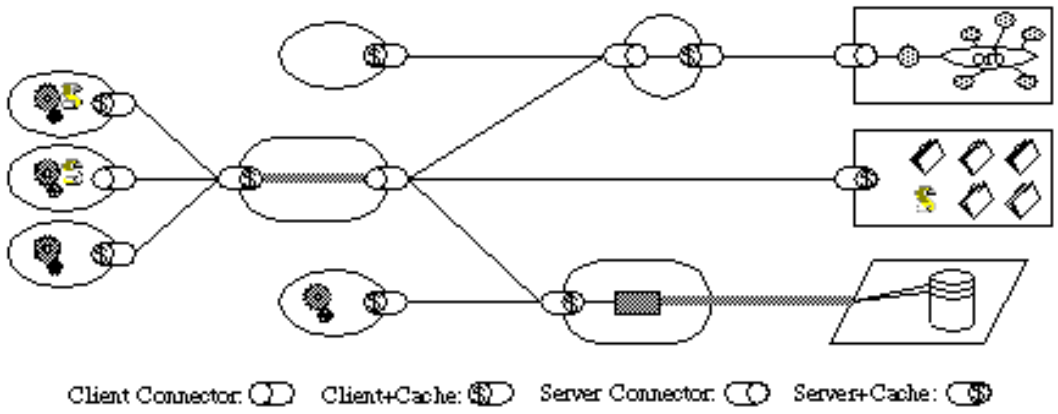
Το μειονέκτημά τους είναι ότι αυξάνουν την καθυστέρηση στην επεξεργασία των δεδομένων μειώνοντας την απόδοση.



Εικόνα 21: Uniform - Layered - Client - Cache - Stateless – Server (Fielding, 2000)

Γ.2.6 Code on Demand

Είναι η τελευταία προαιρετική απαίτηση της αρχιτεκτονικής REST. Επιτρέπει ο πελάτης να λάβει από τον Server κώδικα και να τον εκτελέσει, επεκτείνοντας με αυτό τον τρόπο τις δυνατότητές του. Ο κώδικας είναι σε μορφή applet ή script. Η συγκεκριμένη απαίτηση απλοποιεί την αρχική υλοποίηση των Client, μειώνοντας τον αριθμό των χαρακτηριστικών που προαπαιτούνται, εφόσον αυτά μπορούν να ληφθούν από τον Server (Fielding, 2000).



Εικόνα 22: REST (Fielding, 2000)

Γ3 Στοιχεία Αρχιτεκτονικής

Στην ενότητα αυτή, θα παρουσιάσουμε τα βασικά στοιχεία που δομούν την αρχιτεκτονική REST. Θα πρέπει όμως να επισημάνουμε ότι, η αρχιτεκτονική REST επικεντρώνεται σε ότι αφορά τους ρόλους των στοιχείων, τους περιορισμούς και την αλληλεπίδρασή τους με άλλα, ενώ αγνοεί τις λεπτομέρειες της εφαρμογής τους και της σύνταξης τους.

Γ.3.1 Recourses

Πόρος είναι οτιδήποτε που είναι αρκετά σημαντικό να αναφέρεται σαν ένα πράγμα από μόνο του, να έχει δηλαδή μια ταυτότητα. Είναι συνήθως κάτι που μπορεί να αποθηκευθεί σε έναν υπολογιστή: ένα ηλεκτρονικό αρχείο, μια εγγραφή σε μια βάση δεδομένων, το αποτέλεσμα ενός αλγόριθμου. Όμως πόρος μπορεί να είναι ένας άνθρωπος, ένα χρώμα, μια έννοια κ.λπ. Ο μόνος περιορισμός που υφίσταται είναι ότι πρέπει να έχει ένα URL.

Τους πόρους μπορούμε να τους διακρίνουμε σε:

- Στατικούς: αν εξεταστεί σε οποιαδήποτε χρονική στιγμή μετά τη δημιουργία του, αντιστοιχεί στην ίδια τιμή που έχει οριστεί.
- Δυναμικούς: αν έχουν υψηλό βαθμό διακύμανσης της τιμής που τους ορίζεται σε σχέση με το χρόνο.

Η διάκριση αυτή μπορεί να γίνει καλύτερα κατανοητή με το ακόλουθο παράδειγμα: Η προτεινόμενη έκδοση ενός ακαδημαϊκού εγγράφου του συγγραφέα Χ σε ένα γνωστικό αντικείμενο, έχει μια τιμή που δύναται να αλλάξει με την πάροδο του χρόνου καθώς το κείμενο θα μπορούσε να υποστεί βελτιώσεις ή και αλλαγές. Αυτό συνιστά έναν δυναμικό πόρο. Αντίθετα η έκδοση ενός ακαδημαϊκού συγγράμματος στο συνέδριο Χ είναι στατικός πόρος διότι στο συγκεκριμένο συνέδριο υπάρχει μόνο μια έκδοση του συγγράμματος.

Γ.3.2 Representation

Όπως αναφέραμε στην προηγούμενη παράγραφο, ο άνθρωπος, μια έννοια, μια εγγραφή σε βάση δεδομένων θεωρούνται πόροι αν έχουν URL. Πώς όμως θα μεταφέρεις έναν άνθρωπο διαμέσου του διαδικτύου; Τι έννοια έχει για κάποιον ένα σύνολο δυαδικών δεδομένων (δηλαδή η εγγραφή σε μια βάση δεδομένων) αν δεν γνωρίζει τη βάση δεδομένων στην οποία υπήρχαν αυτά;

Επομένως όταν ζητάμε, μέσω μιας GET, έναν πόρο, ο Server, πρέπει να μας αποστέλλει μια απάντηση η οποία θα περιέχει, όχι μόνο τα δεδομένα του πόρου αλλά και ένα σύνολο από τα μεταδεδομένα που τα περιγράφουν. Επομένως η αναπαράσταση είναι μια αλληλουχία από bytes με επιπλέον μεταδεδομένα, που περιγράφουν την τρέχουσα ή την αποσκοπούσα κατάσταση ενός πόρου.

Η μορφή των μεταδεδομένων είναι ζευγάρια όνομα – τιμή όπου το όνομα περιγράφει τη δομή και τη σημασιολογία της τιμής (Richardson & Amundsen, 2013)

Γ.3.3 Connectors

Το REST χρησιμοποιεί διάφορους τύπους από connector οι οποίοι περιλαμβάνουν τις απαραίτητες ενέργειες – διαδικασίες για την πρόσβαση σε πόρους και την μεταφορά απεικονίσεων. Παρουσιάζοντας μια αφηρημένη διεπαφή, ενισχύουν την απλότητα στην αρχιτεκτονική με το να κρύβουν την πολυπλοκότητα υλοποίησης των μηχανισμών επικοινωνίας παρέχοντας διαχωρισμό στις ανάγκες των εμπλεκόμενων μελών. Η αρχιτεκτονική REST χρησιμοποιεί διάφορους τύπους connector, οι οποίοι απεικονίζονται στον επόμενο πίνακα:

Connector	**Σύγχρονα παραδείγματα του WEB**
client	libwww, libwww-perl
server	libwww, Apache API, NSAPI

cache	browser cache, Akamai cache network
resolver	bind (DNS lookup library)
tunnel	SOCKS, SSL after HTTP CONNECT

Εικόνα 23: Λίστα με connector (Fielding, 2000)

Όπως έχουμε περιγράψει, όλες οι αλληλεπιδράσεις REST είναι stateless. Δηλαδή κάθε αίτημα περιέχει όλες τις απαραίτητες πληροφορίες, έτσι ώστε ο connector να το κατανοήσει, ανεξάρτητα από προηγούμενα αιτήματα που έχουν υποβληθεί. Ο περιορισμός αυτός επιτρέπει στους connector να μη διατηρούν την κατάσταση των εφαρμογών μεταξύ των αιτημάτων, περιορίζοντας την κατανάλωση φυσικών πόρων και βελτιώνοντας την επεκτασιμότητα.

Οι κύριοι τύποι των connector είναι ο Client και ο Server. Η βασική διαφορά τους είναι ότι ο client ξεκινά την επικοινωνία, αποστέλλοντας ένα αίτημα ενώ ο server περιμένει για συνδέσεις και απαντά σε αιτήματα.

Ο επόμενος τύπος είναι η cache, που μπορεί να βρίσκεται στη διεπαφή είτε του client είτε του server. Ο κύριος σκοπός της είναι η αποφυγή επανάληψης της διαδικασίας αποστολής αιτήματος ή απάντησης, με στόχο τη μείωση του χρόνου αλληλεπίδρασης. Η cache μπορεί να είναι κοινόχρηστη ή μη και μπορεί να αποφασίσει αν μια απάντηση πρέπει να αποθηκευτεί ή όχι. Γενικά, όλες οι απαντήσεις σε αιτήματα ανάκτησης αποθηκεύονται στην cache, με σκοπό την επαναχρησιμοποίησή τους, ενώ απαντήσεις σε άλλα αιτήματα δεν αποθηκεύονται. Αν ένα αίτημα περιλαμβάνει αυθεντικοποίηση του χρήστη, τότε η απάντηση θα αποθηκευτεί σε μη κοινόχρηστη cache. Όλα τα παραπάνω μπορούν να τροποποιηθούν (αποθήκευση ή όχι στην cache, χρόνος παραμονής στην cache) αν ο client στείλει την αντίστοιχη παράμετρο.

Ο resolver είναι υπεύθυνος για την μετάφραση (μερική ή ολική) των πόρων σε διευθύνσεις δικτύου. Για παράδειγμα τα περισσότερα URI περιλαμβάνουν ένα DNS όνομα. Ο resolver θα πάρει αυτό το όνομα και θα βρει την IP διεύθυνση.

Ο τελευταίος τύπος connector είναι το tunnel, που είναι υπεύθυνο για την αναμετάδοση της πληροφορίας πέρα από το όριο σύνδεσης π.χ. ένα firewall.

Γ.3.4 Components

Τα Components της αρχιτεκτονικής REST είναι:

- Origin Server (Apache, IIS): είναι η τελική πηγή για τις αναπαραστάσεις των πόρων και πρέπει να είναι ο τελικός αποδέκτης κάθε αιτήματος.
- Gateway (SQUID, CGI, Reverse proxy): λειτουργεί ως ενδιάμεσος, αλλά επιβάλλεται από το διακομιστή. Η λειτουργία που προσφέρει είναι παρόμοια με του proxy.
- Proxy (CERN, Netscape): λειτουργεί ως ενδιάμεσος και επιλέγεται από τον πελάτη για να παρέχει υπηρεσίες διασύνδεσης, ενθυλάκωσης, μετάφρασης δεδομένων κ.λ.π
- User Agent: Χρησιμοποιεί μια σύνδεση πελάτη για να ξεκινήσει μια αίτηση και είναι ο τελικός παραλήπτης της απάντησης (Fielding, 2000)

Γ.4 Παράδειγμα χρήσης

Στην ενότητα αυτή θα περιγράψουμε τον τρόπο χρήσης της αρχιτεκτονικής REST με χρήση ενός απλού παραδείγματος.

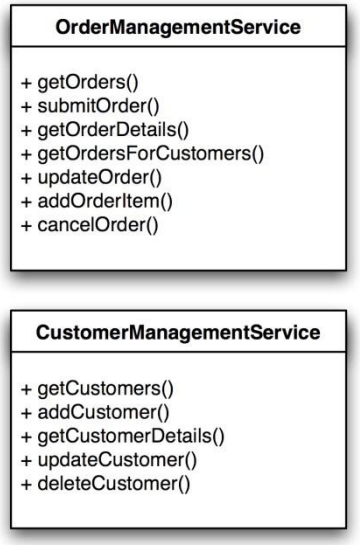
Όπως αναφέραμε κάθε πόρος έχει ένα μοναδικό αναγνωριστικό, το URI. Για να αποκτήσουμε πρόσβαση στον πόρο πρέπει να χρησιμοποιήσουμε το URI του, με τη βοήθεια των HTTP μεθόδων. Για να επιτύχουμε την αλλαγή των οντοτήτων ενός πόρου χρησιμοποιούμε την αρχή CRUD δηλαδή Create, Retrieve, Update και Delete, ώστε να έχουμε πρόσβαση σε όλες τις βασικές λειτουργίες του πόρου.

Πίνακας 1: Αντιστοιχία μεθόδων HTTP με λειτουργίες CRUD

HTTP μέθοδοι	CRUD λειτουργίες
GET	Retrieve
POST	Create
PUT	Update

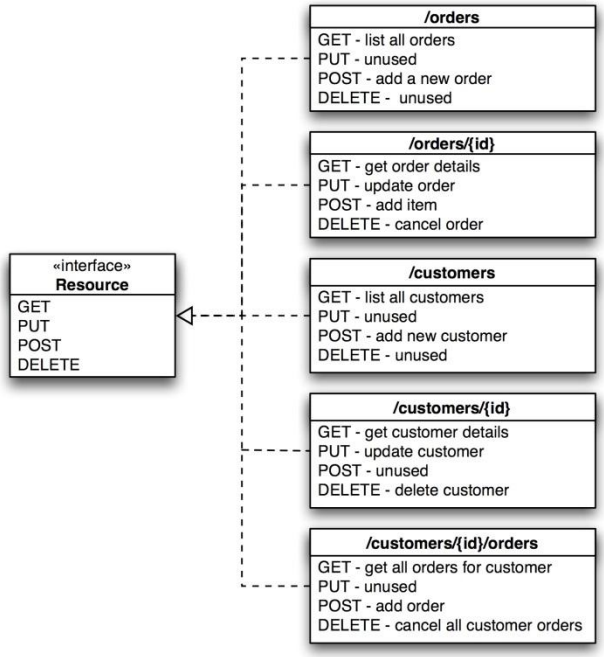
DELETE	Delete

Ας υποθέσουμε ότι θέλουμε να εκθέσουμε μέρος της λειτουργικότητας ενός ηλεκτρονικού καταστήματος στο διαδίκτυο, μέσω δύο web service, ένα για τη διαχείριση παραγγελιών και ένα για τη διαχείριση πελατών, όπως την επόμενη εικόνα.



Εικόνα 24: Web service e-shop (Tikov, 2007)

Αν πρέπει κάποιος πελάτης να καταναλώσει αυτά τα web service, πρέπει να υλοποιηθεί κώδικας ειδικά για αυτά. Σε αντίθεση, σε μια προσέγγιση της REST αρχιτεκτονικής με χρήση του HTTP πρωτοκόλλου θα κάνουμε χρήση της διεπαφής που συνθέτει το HTTP πρωτόκολλο. Οπότε θα καταλήξουμε σε κάτι όπως η ακόλουθη εικόνα:



Εικόνα 25: Λειτουργίες e-shop και πρόσβαση αυτών μέσω REST και HTTP (Tikov, 2007)

Στην εικόνα αυτή μπορούμε να δούμε ότι συγκεκριμένες λειτουργίες των web service έχουν αντιστοιχηθεί στις μεθόδους του HTTP.

Για παράδειγμα, για να αποκτήσει κάποιος πρόσβαση στη λίστα των πελατών του καταστήματος πρέπει να απευθυνθεί στη διεύθυνση: http://www.e-shop.com/customers/ και αν την καλέσει ο χρήστης θα λάβει ένα μήνυμα απάντησης το οποίο θα είναι όπως φαίνεται παρακάτω :


```
{
    "name": "John Doe",
    "code": "1123344",
    "phone_number": "222123444",
    "address": "John Doe address",
    "id": "1",
    "resource_uri": "/api/customers/1/",
}
```


Εικόνα 26: Απάντηση σε κλήση REST web service

Στην απάντηση του εξυπηρετητή περιλαμβάνεται και η διεύθυνση του πόρου : “resource_uri” : “/api/customer/1/” το οποίο σημαίνει ότι εάν θέλουμε να δούμε τις

πληροφορίες που αφορούν τον πελάτη John Doe μπορούμε να πληκτρολογήσουμε κατευθείαν τη διεύθυνση :

http://www.e-shop.com/customers/1/

και η απάντηση που θα λάβουμε θα περιλαμβάνει μόνο τα στοιχεία του πελάτη που ζητήσαμε.

Ο παραπάνω τρόπος που αναφέραμε, μας έδειξε πώς μπορούμε να καταναλώσουμε ένα web service χρησιμοποιώντας την αρχιτεκτονική REST, μέσω του HTTP. Παρόμοια διαδικασία ισχύει και για τους άλλους πόρους που διαθέτει το κατάστημά μας (Tilkov, 2007).

Βιβλιογραφία

Fielding, R. T. «*Architectural Styles and the Design of Network-based Software Architectures*», Doctoral dissertation, University of California, Irvine, 2000

Richardson L. & Amundsen M. (2013), «*RESTful WEB APIs*», O' REILY

Rodriguez A. (2008), RESTful Web services: The basics, http://www.ibm.com/developerworks/webservices/library/ws-restful/

Tilkov S. (2007), «*A brief introduction to REST*», http://www.infoq.com/articles/rest-introduction

Κεφάλαιο Δ: Σύγκριση Αρχιτεκτονικών SOAP - REST

Δ.1 Εισαγωγή

Έχοντας μελετήσει και παρουσιάσει αναλυτικά τόσο το SOAP όσο και το REST (Representational State Transfer), στο σημείο αυτό θα παρουσιαστεί μια σύγκριση μεταξύ τους. Στη σύγκριση αυτή, θα εξεταστούν τα υπέρ και τα κατά της κάθε λύσης και στο τέλος θα προσπαθήσουμε να προτείνουμε τις εφαρμογές για τις οποίες ενδείκνυται η κάθε λύση.

Βέβαια πρέπει να αναφέρουμε ότι η σύγκριση δε μπορεί να γίνει θέτοντας τη μια λύση απέναντι στην άλλη, γιατί είναι ανόμοιες. Για να ξεπεράσουμε αυτή τη δυσκολία, θα θέσουμε διάφορα ζητήματα, όπως της ασφάλειας και θα μελετήσουμε την κάθε λύση παρουσιάζοντας τα πλεονεκτήματα ή τα μειονεκτήματά της.

Δ.2 Απόδοση

Μία μετρήσιμη παράμετρος αναφορικά με την απόδοση των εξεταζόμενων υπηρεσιών είναι ο χρόνος απόκρισης στο περιβάλλον του χρήστη και η οποία έχει χρησιμοποιηθεί σε αρκετές μελέτες στο παρελθόν για τη σύγκριση των δύο τεχνολογιών (Potti, 2012) (Mulligan, 2009).

Όπως έχουμε ήδη αναφέρει έως τώρα και οι δύο εξεταζόμενες τεχνολογίες χρησιμοποιούν το πρωτόκολλο XML για την περιγραφή των αντικειμένων (το REST μπορεί επίσης να χρησιμοποιήσει και το πρωτόκολλο JSON), καθώς επίσης και το πρωτόκολλο HTTP για το επίπεδο μεταφοράς και την κλήση των υπηρεσιών. Όπως θα περίμενε κανείς οι απόδοσή τους, τουλάχιστον σε ότι αφορά τους χρόνους απόκρισης θα έπρεπε να είναι εφάμιλλη. Παρόλα αυτά, σε σειρά μελετών έχει αποδειχθεί ότι οι υλοποιήσεις REST εμφανίζουν σημαντικά

μειωμένους χρόνους απόδοσης συγκριτικά με εκείνους του SOAP σε ευθεία σύγκριση μεταξύ τους (Potti, 2012) (Mulligan, 2009) .

Ο κυριότερος λόγος που προκαλεί την εν λόγω διαφορά είναι η επιπρόσθετη επιβάρυνση (overhead) που προσδίδει σε κάθε αίτηση/απάντηση ο φάκελος (envelop) του πρωτοκόλλου SOAP (Mulligan, 2009). Η διαφορά στους χρόνους απόκρισης ποικίλει ανάλογα με την ίδια την υπηρεσία, το δικτυακό εξοπλισμό και τις επιδόσεις του διακομιστή.

Ο ίδιος λόγος (overhead) προκαλεί κατ' αντιστοιχία αυξημένο throughput για τις υλοποιήσεις REST και συνεπώς βέλτιστη αξιοποίηση του διατιθέμενου εύρους ζώνης (bandwidth) (Mulligan, 2009).

Δ.3 Ασφάλεια

Ένα από τα πιο σημαντικά θέματα, το οποίο πρέπει να μελετήσουμε είναι η ασφάλεια που παρέχουν οι δύο λύσεις. Όπως γνωρίζουμε και οι δύο χρησιμοποιούν γνωστά πρωτόκολλα (όπως το HTTP), πόρτες και μεθόδους πρόσβασης σε back-end υπηρεσίες, και είναι εκτεθειμένες σε διάφορες ευπάθειες (vulnerabilities) όπως:

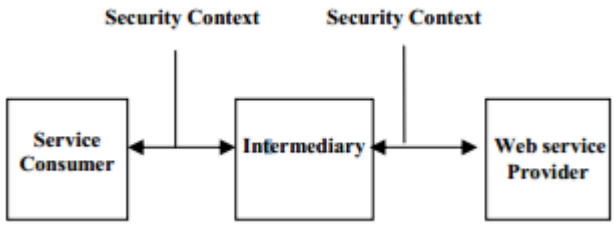
- Αδυναμίες του Λειτουργικού Συστήματος
- Αδυναμίες του Web server (για παράδειγμα Apache or IIS)
- Προβλήματα της SQL
- Επιθέσεις σε Application server
- Επιθέσεις Denial of Service
- Παρακολούθηση μηνυμάτων – αντιγραφή – πλαστογράφηση πηγής
- Μεταφορά ιών (στα μηνύματα ή σαν επισυναπτόμενα) κ.ά. (McKinley, 2012)

Ας δούμε τώρα πώς η κάθε λύση αντιμετωπίζει τα προβλήματα ασφάλειας.

Δ.3.1 SOAP

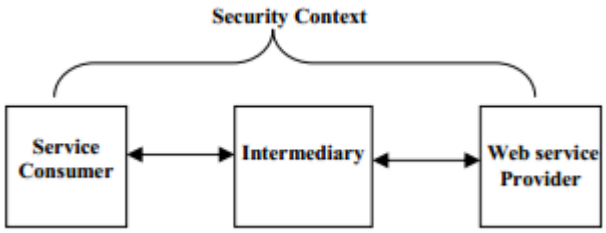
Η ασφάλεια του SOAP χωρίζεται σε δύο τμήματα. Στην ασφάλεια του μηνύματος και την ασφάλεια του επιπέδου μεταφοράς (transport layer).

Η ασφάλεια του επιπέδου μεταφοράς (SSL) ασχολείται με τη διασφάλιση της διαδρομής των μηνυμάτων, χωρίς να επηρεάζει τα ίδια τα μηνύματα. Εξασφαλίζει, ότι κανένα μη εξουσιοδοτημένο άτομο δε μπορεί να έχει πρόσβαση σε αυτά, μεταξύ δύο σημείων (όπως δείχνει η επόμενη εικόνα). Τι γίνεται όμως όταν ένα μήνυμα ληφθεί από έναν ενδιάμεσο κόμβο και προωθηθεί σε κάποιο επίπεδο πάνω από το επίπεδο μεταφοράς; Τότε δυστυχώς έχει χαθεί η προστασία του μηνύματος. Επομένως ακόμα και αν χρησιμοποιούμε την ασφάλεια του επιπέδου μεταφοράς (SSL), έχουμε προστατέψει το μήνυμά μας όσο αυτό μεταδίδεται (είναι πάνω στο καλώδιο). Δυστυχώς δε μας διασφαλίζει το επίπεδο αυτό, όταν το μήνυμα φθάσει σε κάποιον ενδιάμεσο κόμβο.



Εικόνα 27: Point to point ασφάλεια (Rahaman, Schaad, & Rits, 2006)

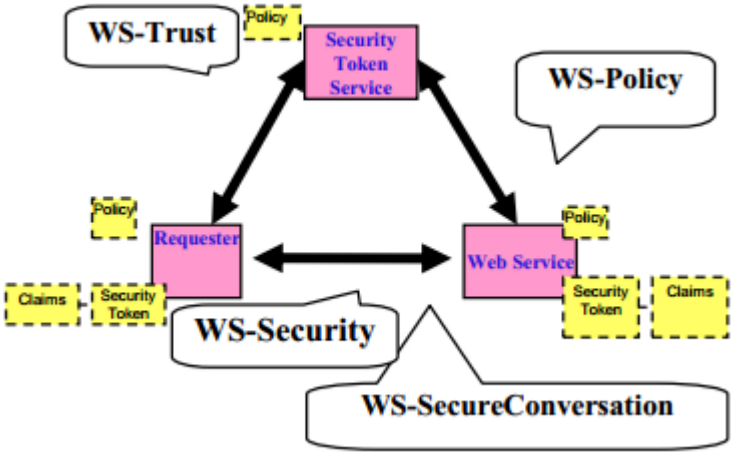
Άρα χρειαζόμαστε ένα ακόμα επίπεδο ασφάλειας, που θα μας εξασφαλίζει από το ένα άκρο στο άλλο. Αυτό μας το παρέχει η ασφάλεια του μηνύματος, η οποία προστατεύει το ίδιο το μήνυμα. Με αυτό τον τρόπο διασφαλίζεται η ασφάλεια στο SOAP από το ένα άκρο στο άλλο.



Εικόνα 28: End to End ασφάλεια (Rahaman, Schaad, & Rits, 2006)

Εκτός από την ασφάλεια που αναφέραμε, υπάρχει και η ασφάλεια που παρέχεται από τα πρόσθετα πρωτόκολλα που ορίζονται στα WS-* πρότυπα (SOAP, WSDL, UDDI) μέσω των διαφορετικών πλατφορμών με τη βοήθεια των οποίων υλοποιούνται (J2EE, .NET κ.λ.π). Όταν αναφερόμαστε στο WS-* εννοούμε ένα σύνολο από προδιαγραφές οι οποίες σχετίζονται με τα Web Services.

Σε αυτές τις προδιαγραφές ανήκουν και τα WS-Trust, WS-Security, WS-Policy κ.λ.π. Όλες αυτές οι προδιαγραφές – στάνταρντ ακολουθούν μια εξελικτική διαδικασία με σκοπό να διασφαλίσουν την ασφάλεια από το ένα άκρο στο άλλο. Στην εικόνα που ακολουθεί μπορούμε να δούμε μια αρχιτεκτονική με την χρήση συγκεκριμένων προδιαγραφών.

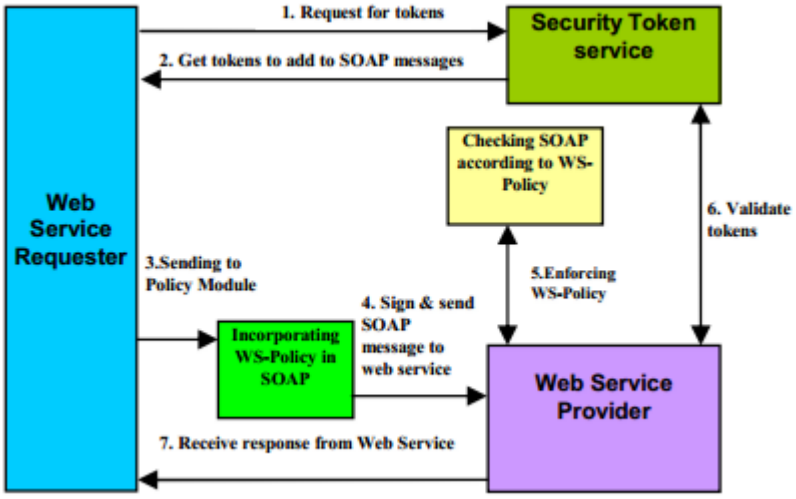


Εικόνα 29: Αρχιτεκτονική Ασφαλείας WS-* (Rahaman, Schaad, & Rits, 2006)

- Το WS-Security περιγράφει πώς θα γίνει η επισύναψη της υπογραφής και κλειδιών κρυπτογράφησης στο προς μετάδοση μήνυμα.
- Το WS-Policy καθορίζει μια πολιτική από κανόνες για το πώς οι υπηρεσίες θα αλληλεπιδρούν.
- Το WS-Trust καθορίζει το μοντέλο εμπιστοσύνης

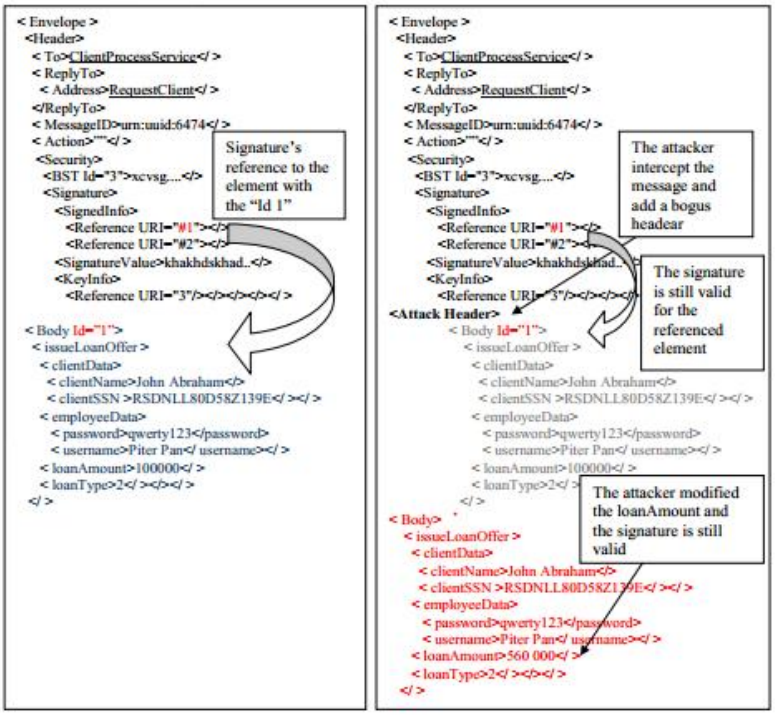
Όμως η εφαρμογή των κανόνων του WS-Security σε κάθε μήνυμα είναι αναποτελεσματική (γιατί μειώνεται η απόδοση, εφόσον η κρυπτογραφία απαιτεί πόρους, υπολογιστική ισχύ και χρόνο). Αντί αυτής μπορεί να εφαρμοστεί το WS-SecureConversation το οποίο καθορίζει πώς θα επιτευχθεί μια ασφαλή συνεδρία μεταξύ υπηρεσιών που ανταλλάσσουν δεδομένα, βασιζόμενα σε κανόνες WS-Policy, WS-Trust, WS-Privacy.

Εφαρμόζοντας τους κανόνες που προαναφέρθηκαν, μπορούμε να δούμε τη ροή του μηνύματος στην επόμενη εικόνα (Rahaman, Schaad, & Rits, 2006) :



Εικόνα 30: Ροή μηνύματος (Rahaman, Schaad, & Rits, 2006)

Ένα άλλο πρόβλημα ασφαλείας που μπορεί να παρουσιαστεί είναι το XML rewriting attack. Το πρόβλημα παρουσιάζεται με την αδυναμία που έχει η XML Digital Signature, η οποία περιέχει αναφορά μέσω ενός URI στα δεδομένα που υπογράφονται.



Εικόνα 31: SOAP μήνυμα πριν και μετά την επίθεση (Benameur, Kadir & Fenet, 2008)

Για να ξεπεραστεί το συγκεκριμένο πρόβλημα έχουν προταθεί διάφορες λύσεις όπως SOAP Account, WS-Policy, WSE Policy Advisor κ.λ.π. (Benameur, Kadir & Fenet, 2008)

Δ.3.2 REST

Όπως αναφέραμε στο προηγούμενο κεφάλαιο το REST βασίζεται στο HTTP πρωτόκολλο, επομένως κληρονομεί όλες τις αδυναμίες αλλά και τις δυνατότητες. Μια δυνατότητα που προσφέρει ένα επίπεδο ασφάλειας είναι η χρήση του SSL.

Δεν πρέπει όμως να περιοριστούμε μόνο σε αυτό. Πρέπει να χρησιμοποιηθούν και άλλες τεχνικές όπως:

- Authentication and Session Management: μέσω της διαδικασίας του Authentication γίνεται επιτρεπτή η χρήση των υπηρεσιών. Όμως η

διαδικασία αυτή πρέπει να γίνεται μέσω session, τα οποία δεν πρέπει να εμφανίζονται στο URL γιατί μπορούν να κλαπούν.

- Προστασία HTTP Μεθόδων: Τα REST Web Service χρησιμοποιούν μεθόδους Get, Post, Put και Delete. Πριν την εκτέλεση των μεθόδων πρέπει να ελέγξουμε αν πρέπει να εφαρμοστούν για το συγκεκριμένο resource. Για παράδειγμα αν έχουμε φτιάξει ένα REST Web Service για βιβλιοθήκη, δε θέλουμε όλοι οι χρήστες να έχουν δυνατότητα να εκτελέσουν μια μέθοδο delete, ενώ μπορούν να εκτελέσουν μια μέθοδο get.
- Έλεγχος τιμών εισόδου: πρέπει να ελέγχονται οι τιμές εισόδου, για να δούμε αν ανήκουν στα αποδεκτά όρια, αν προκαλούν SQL injection κ.λ.π. (Oftedal & Stock, 2014)

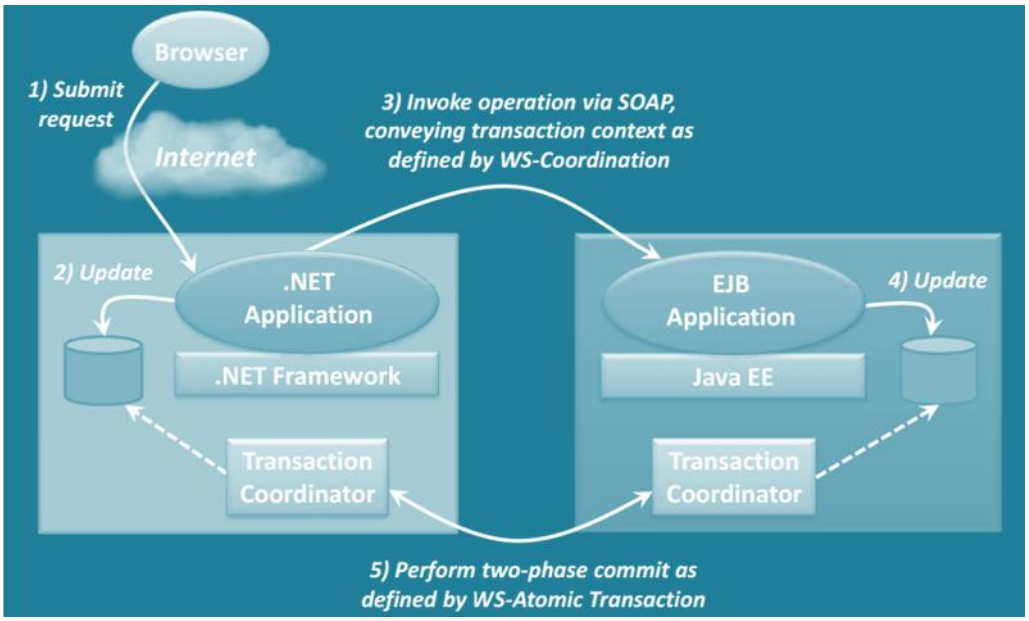
Δ.4 Transaction

Τι γίνεται αν θέλουμε η εφαρμογή μας να υποστηρίζει ACID transaction; Το ACID προέρχεται από τα αρχικά των:

- Atomicity: κάθε transaction πρέπει να είναι μια αυτόνομη εργασία που είτε θα γίνουν όλα όσα αυτή ζητάει να γίνουν, είτε δεν θα γίνει τίποτα.
- Consistency: Σημαίνει ότι μετά την ολοκλήρωση του transaction όλα θα είναι σε μια σταθερή κατάσταση.
- Isolation: οι αλλαγές που ζητάει ένα transaction να γίνουν, πρέπει να είναι απομονωμένες από τις αλλαγές που κάποια άλλα transaction που εκτελούνται ταυτόχρονα κάνουν.
- Durability: αφού το transaction μας έχει ολοκληρωθεί, αυτό παραμένει ακόμα και αν το σύστημα μας έχει failure.

Στην υλοποίηση των παραπάνω υπερισχύει το SOAP γιατί έχει και τη βοήθεια των WS-* και ιδιαίτερα του WS-Atomic Transactions το οποίο είναι ένα πρωτόκολλο two-phase commit για διαλειτουργικότητα των web services. Στην

εικόνα που ακολουθεί βλέπουμε ένα παράδειγμα χρήσης του WS-Atomic Transction. (Chappell, 2009)



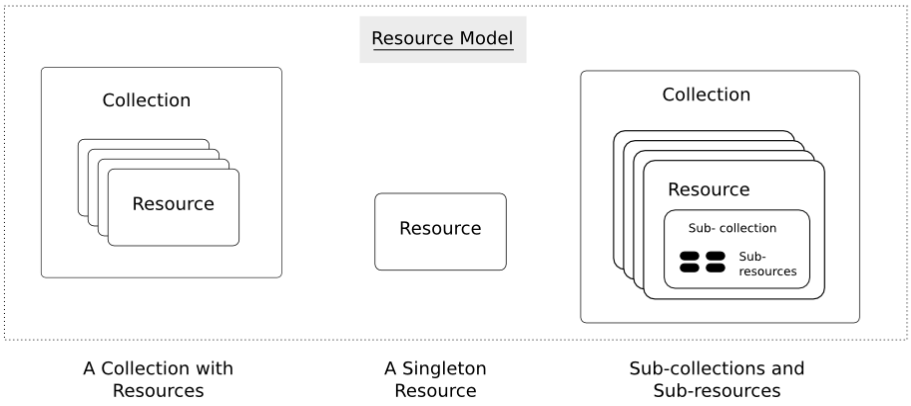
Εικόνα 32: Παράδειγμα χρήσης του WS-Atomic Transction (Chappell, 2009)

Η παραπάνω απαίτηση (για ACID transaction) ισχύει κυρίως για εφαρμογές για επιχειρήσεις, ενώ δεν υπάρχει η συγκεκριμένη απαίτηση για απλές εφαρμογές διαδικτύου.

Δ.5 Υπηρεσίες ή Πόροι;

Όπως γνωρίζουμε (Γ.3.1) η αρχιτεκτονική REST βασίζεται στην έννοια των πόρων που συνήθως απεικονίζουν δεδομένα. Όπως έχουμε αναφέρει, ένας πόρος είναι ένα αντικείμενο με ένα τύπο, δεδομένα, σχέσεις με άλλους πόρους καθώς και μεθόδους που μπορούν να εφαρμοστούν σε αυτό. Μοιάζει με ένα αντικείμενο μιας αντικειμενοστρεφούς γλώσσας προγραμματισμού με τη μόνη διαφορά ότι οι εφαρμοζόμενες μέθοδοι είναι οι get, post, put και delete.

Οι πόροι μπορούν να υπάρξουν μόνοι τους (singleton) ή να ομαδοποιηθούν, ανάλογα με τον τύπο τους, σε collection.

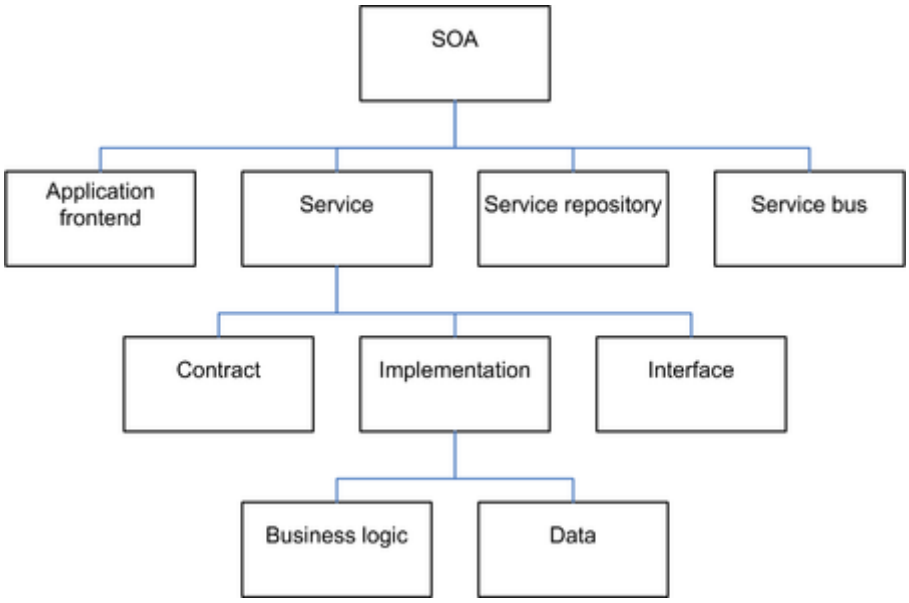


Εικόνα 33: Ομαδοποίηση πόρων (Jansen, 2011)

Όλες οι εφαρμογές που βασίζονται στην αρχιτεκτονική REST παρέχουν πρόσβαση στους πόρους τους μέσω του ίδιου interface.

Αντίθετα το πρωτόκολλο SOAP είναι προσανατολισμένο σε υπηρεσίες, κάθε μια από τις οποίες υλοποιεί κάποια επιχειρηματική λογική (Chappell, 2009). Η λειτουργικότητα μιας υπηρεσίας ορίζεται από τη διεπαφή της, η οποία μπορεί να υποστηριχτεί από πολλές υλοποιήσεις. Η διεπαφή μπορεί να υλοποιηθεί είτε μέσω RPC είτε μέσω ανταλλαγής μηνυμάτων (XML). Με βάση τη συγκεκριμένη λογική αναπτύχθηκε η SOA η αρχιτεκτονική που είναι βασισμένη στις υπηρεσίες.

Στην εικόνα που ακολουθεί μπορούμε να δούμε καλύτερα τα στοιχεία που αποτελούν μια αρχιτεκτονική βασισμένη στις υπηρεσίες (SOA). Όπως μπορούμε να δούμε στη βάση είναι η Επιχειρησιακή Λογική και τα Δεδομένα. Αμέσως μετά έρχεται η υλοποίηση και η διεπαφή. Στο τελευταίο επίπεδο υπάρχει το frontend της εφαρμογής και η ίδια η υπηρεσία.



Εικόνα 34: Στοιχεία του SOA (Krafzig, Banke & Slama, 2004)

Για να γίνει καλύτερα κατανοητή η διαφορά θα παρουσιάσουμε ένα παράδειγμα μιας τράπεζας. Έστω ότι θέλουμε να έχουμε δύο web service ένα για να βλέπουμε το υπόλοιπο του λογαριασμού και ένα για ενημέρωση υπολοίπου. Και τα δύο web service μπορούν να υλοποιηθούν είτε με REST είτε με SOAP με την ακόλουθη μορφή:

GetBalance(Account)

UpdateBalance(Account, Amount)

Έστω ότι θέλουμε να υλοποιήσουμε άλλο ένα web service για μεταφορά χρημάτων από ένα λογαριασμό σε ένα άλλο.

Transfer(FromAccount, ToAccount, Amount)

Το συγκεκριμένο web service, είναι δύσκολο να πραγματοποιηθεί μέσω του REST που δίνει πρόσβαση στους πόρους, ενώ αντίθετα μπορεί να κατασκευαστεί εύκολα μέσω SOAP, αρκεί να καθοριστεί η λογική που πρέπει να υλοποιηθεί (Chappell, 2008)

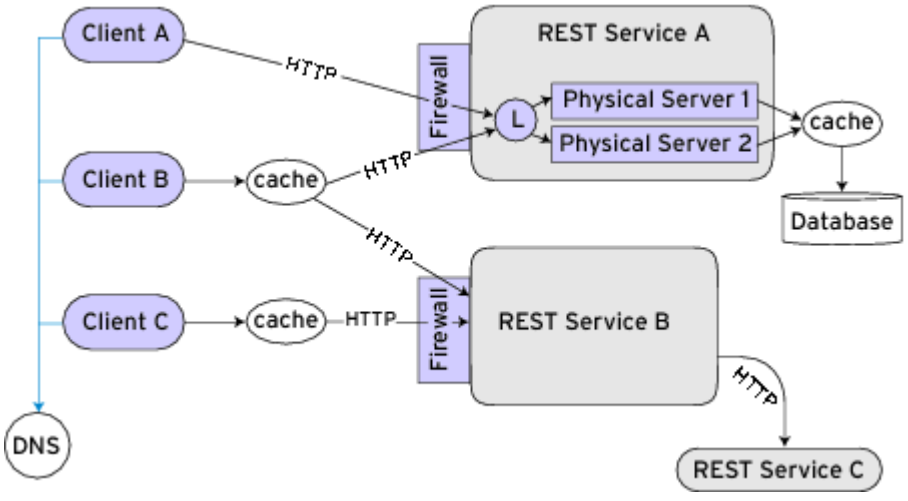
Δ.6 Προσωρινή αποθήκευση δεδομένων

Το μεγαλύτερο μέρος των αιτημάτων σε ένα web service είναι ερωτήματα get. Τα αποτελέσματα των ερωτημάτων αυτών μπορούν να αποθηκευθούν προσωρινά για μελλοντική χρήση. Με την προσωρινή αποθήκευση δεδομένων βελτιώνουμε την ταχύτητα και μειώνουμε το φορτίο του κεντρικού υπολογιστή

Στο σημείο αυτό το REST υπερτερεί σε σχέση με το SOAP. Χρησιμοποιεί το caching του HTTP πρωτοκόλλου και του εξυπηρετητή διαδικτύου, οπότε προγραμματίζεται εύκολα. Στο SOAP απαιτείται ειδικός προγραμματισμός στον κώδικα της εφαρμογής. (Chappell, 2009)

Δ.7 Load Balancing

Με το Load Balancing εννοούμε την κατανομή φορτίου των αιτημάτων των client σε πολλούς κεντρικούς υπολογιστές. Στο σημείο αυτό υπερισχύει το REST λόγω της άμεσης σύνδεσής του με το HTTP πρωτόκολλο. (Bruno, 2008)

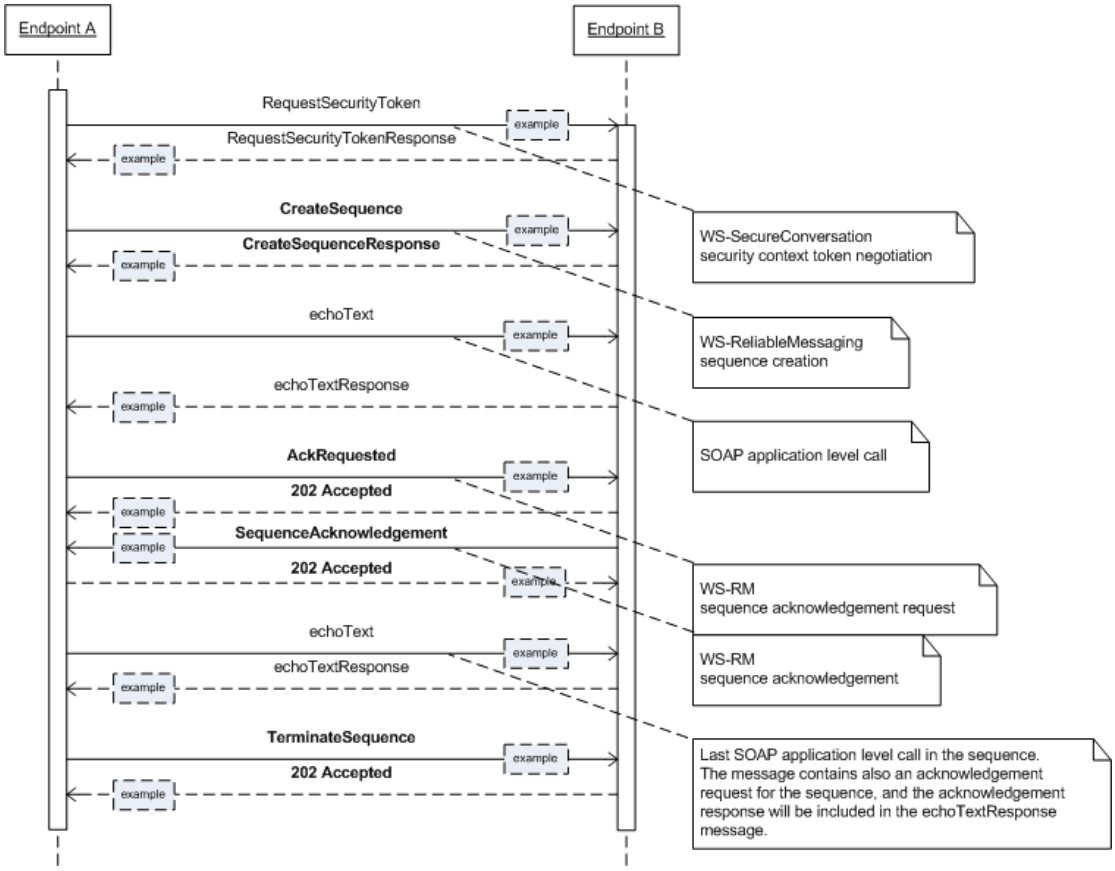


Εικόνα 35: Αρχιτεκτονική REST με Load Balancing και Cache (Bruno, 2008)

Δ.8 Αξιοπιστία

Η αρχιτεκτονική REST θεωρεί ότι η εφαρμογή είναι αυτή που θα διαχειριστεί τα λάθη της επικοινωνίας.

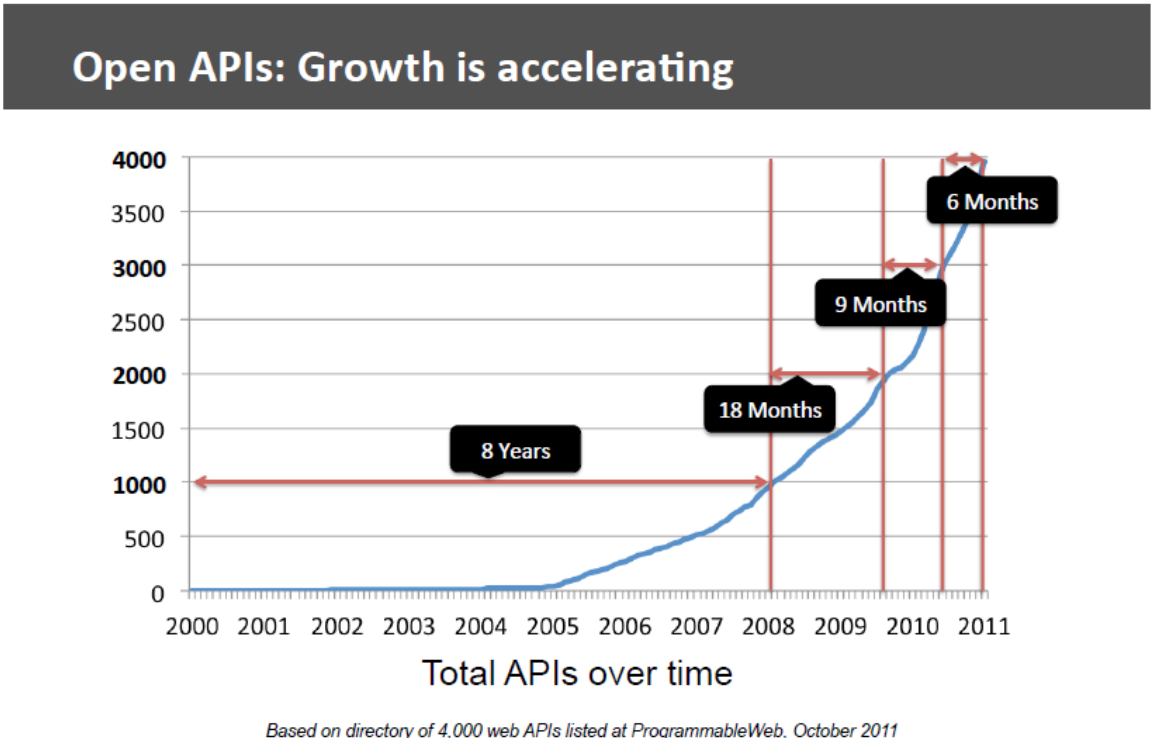
Αντίθετα το SOAP έχει το WS-ReliableMessaging το οποίο καθορίζει μηχανισμούς για την εξασφάλιση της παράδοσης των μηνυμάτων, με τη σωστή σειρά, και την εξάλειψη των διπλότυπων, ακόμα και αν χαθεί η σύνδεση μεταξύ κεντρικού υπολογιστή και πελάτη ή ο κεντρικός υπολογιστής επανεκκινηθεί. (Chappell, 2009)



Εικόνα 36: Σενάριο χρήσης SOAP με WS-ReliableMessaging

Δ.9 Διείσδυση στην αγορά

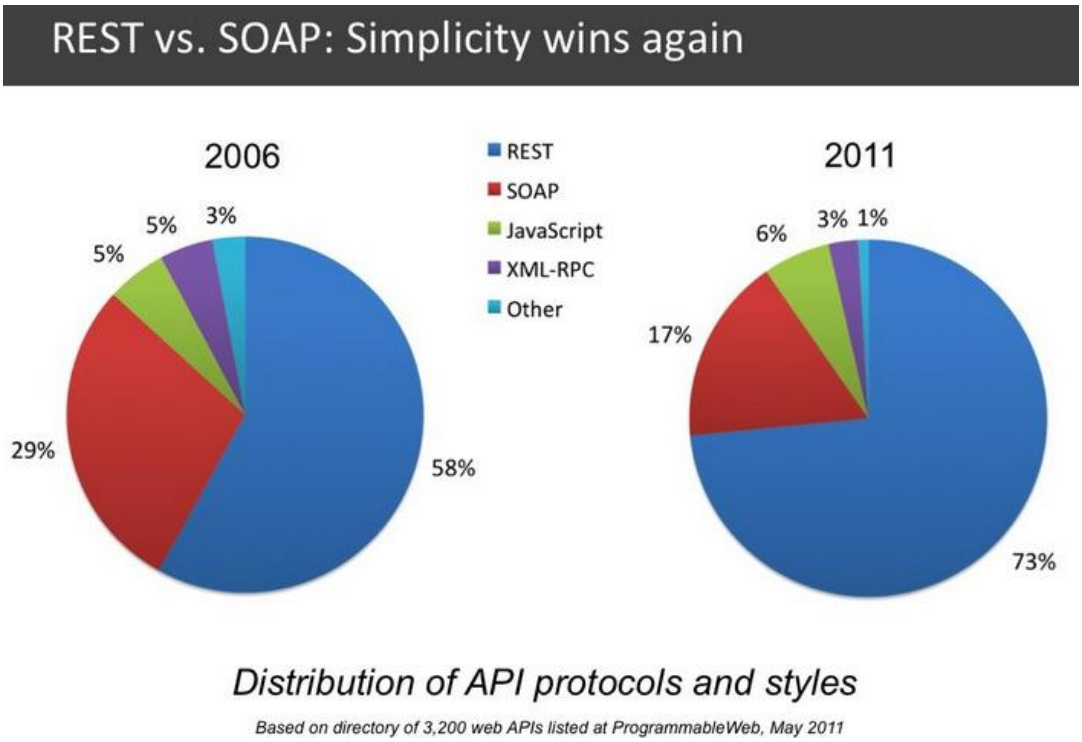
Η αγορά των υπηρεσιών ιστού μεγαλώνει με εκθετικούς ρυθμούς τα τελευταία χρόνια,όπως φαίνεται και στην ακόλουθη εικόνα.



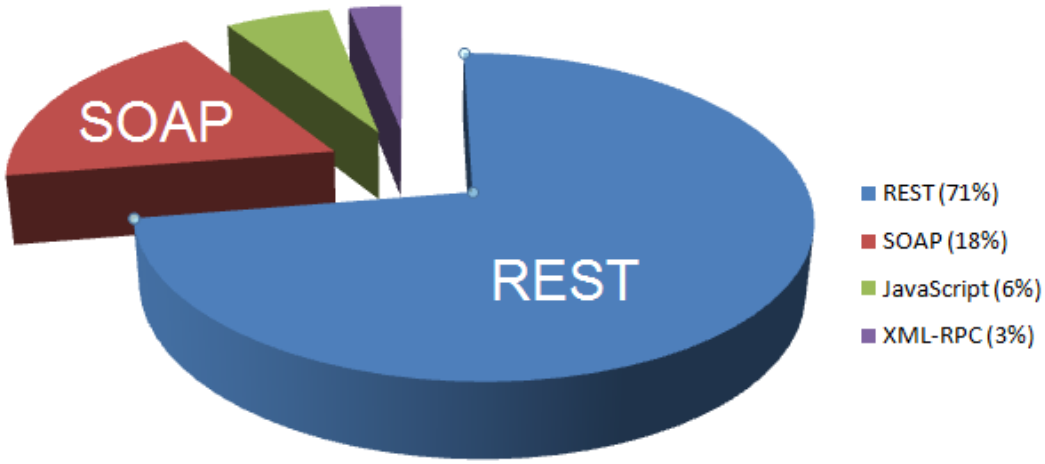
Εικόνα 37: Η αύξηση των απαιτήσεων υπηρεσιών ιστού (Open APIs) (Musser, 2011)

Η διεισδυτικότητα των διαφόρων τεχνολογιών ποικίλει χρόνο με το χρόνο και οι απαιτήσεις αλλάζουν καθώς η αγορά ωριμάζει ενώ οι απαιτήσεις της φορητότητας των συσκευών και της απλότητας στην υλοποίηση γίνονται εντονότερες.

Η απεικόνιση της διεισδυτικότητας των διαφόρων τεχνολογιών, σύμφωνα με την ιστοσελίδα programmableWeb (ένα από τα μεγαλύτερα παγκοσμίως API Repositories) συγκριτικά για τα έτη 2006 και 2011 φαίνεται στην ακόλουθη εικόνα.



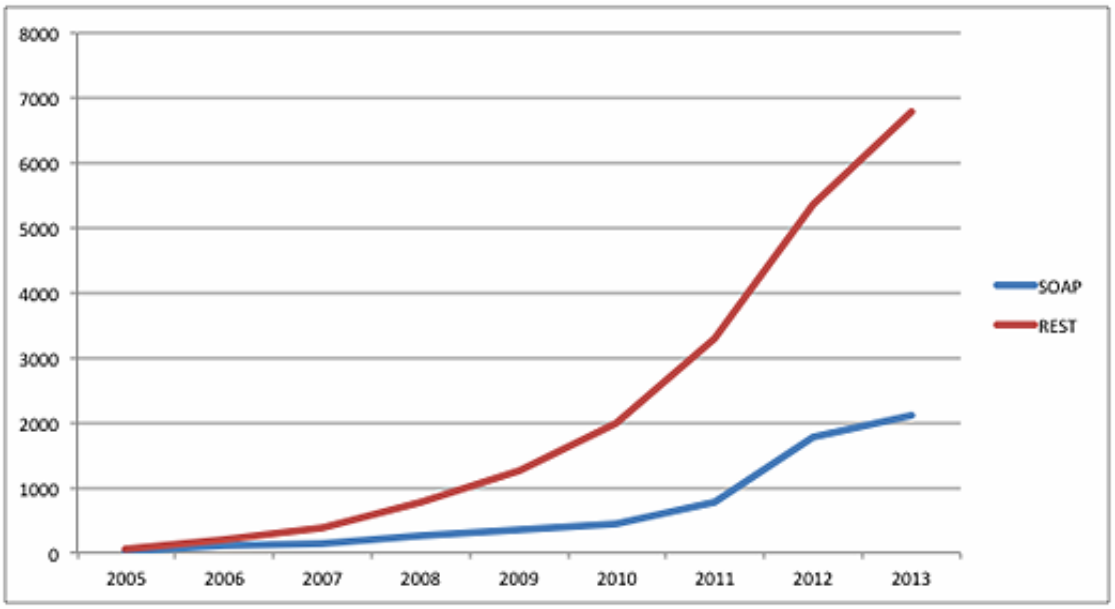
Εικόνα 38: Η διείσδυση των τεχνολογιών στην αγορά (Musser, 2011)



Εικόνα 39: Η διείσδυση των τεχνολογιών στην αγορά για το 2003 (ProgrammableWeb.com)

Όπως προκύπτει άμεσα από την παραπάνω εικόνα η διείσδυση των REST και SOAP τα τελευταία χρόνια καλύπτει το 90% της αγοράς, με το REST να έχει σημαντικό προβάδισμα με 71% (2003). Στο σημείο αυτό θα πρέπει να σημειωθεί

ότι εκφράζονται επιφυλάξεις από τους υποστηρικτές του SOAP ότι η σύγκριση μεταξύ SOAP και REST δεν περιλαμβάνει όλα τα στοιχεία καθώς, όπως ισχυρίζονται δεν περιλαμβάνει στοιχεία από τις εμπορικές εφαρμογές. Παρόλα αυτά το programmableWeb παρέχει μία ευθεία σύγκριση αυτών για τα τελευταία 8 έτη. Σε αυτή τη σύγκριση εμφανίζεται το πολύ σημαντικό προβάδισμα του REST έναντι του SOAP το οποίο βαίνει αυξανόμενο με το πέρασμα του χρόνου, όπως φαίνεται στην ακόλουθη εικόνα.



Εικόνα 40: Σύγκριση χρήσης SOAP και REST τα τελευταία έτη (ProgrammableWeb.com)

Δ.8 Συμπεράσματα

Μετά από τη σύγκριση των δύο λύσεων και την παρουσίαση των πλεονεκτημάτων και των μειονεκτημάτων τους, δε μπορούμε να καταλήξουμε ποια από τις δύο είναι καλύτερη.

Για να βρούμε ποια από τις δύο θα χρησιμοποιήσουμε μπορούμε να απαντήσουμε σε ορισμένες ερωτήσεις:

- Το service θα χρησιμοποιεί δεδομένα ή λογική; Στην περίπτωση των δεδομένων καλύτερη επιλογή είναι το REST, ενώ στην περίπτωση της λογικής το SOAP.
- Χρειάζεται το service υποστήριξη των δυνατοτήτων των WS-*;

- Ποια είναι η καλύτερη επιλογή για τους προγραμματιστές που θα υλοποιήσουν τους client;

Καταλήγοντας, αναφέρουμε ότι το SOAP χρησιμοποιείται κυρίως σε εταιρικά περιβάλλοντα όπου υπάρχουν αυξημένες απαιτήσεις, ενώ το REST κυρίως σε διάφορους δικτυακούς τόπους κοινωνικής δικτύωσης όπως το Twitter κ.λ.π. Ο πίνακας που ακολουθεί περιέχει συνοπτικά τα βασικότερα σημεία σύγκρισης του SOAP με το REST.

	SOAP	**REST**
Χαρακτηριστικό	Πρωτόκολλο βασισμένο σε XML	Βασισμένο στα πρότυπα HTTP και XML
Απόδοση	Υψηλότεροι χρόνοι απόδοσης Μικρότερο throughput Περιορισμοί στο Bandwidth Λόγω του overhead ανά αίτηση	Χαμηλότεροι χρόνοι απόδοσης Μεγαλύτερο throughput Καλύτερη αξιοποίηση Bandwidth
Υπηρεσίες / Πόροι	Σχετίζεται με υπηρεσίες	Σχετίζεται με πόρους
Atomic Transaction	Υποστηρίζει μέσω του WS-Atomic Transactions	Δεν υποστηρίζει
Ασφάλεια	Χρήση του SSL του HTTP. Έχει ένα σύνολο από πρότυπα WS για την υποστήριξη της ασφάλειας (WS-Trust, WS-Security, WS-Policy κ.λ.π.)	Στην ασφάλεια βασίζεται στο SSL του HTTP
Προσωρινή αποθήκευση δεδομένων	Απαιτείται ειδικός προγραμματισμός	Εύκολη Υλοποίηση
Load Balancing	Δύσκολη Υλοποίηση	Εύκολη Υλοποίηση
Αξιοπιστία	Με τη βοήθεια των WS (WS-ReliableMessaging) έχει υψηλή αξιοπιστία	Όχι Αξιόπιστο
Διείσδυση στην αγορά	18%	71%

Βιβλιογραφία

Benameur A. , Kadir F.A. & Fenet S. (2008), «XML Rewriting Attacks: Existing Solutions and their Limitations», http://arxiv.org/abs/0812.4181

Bruno, E.J (2008), «SOA, Web Services, and RESTful Systems», http://www.drdobbs.com/web-development/soa-web-services-and-restful-systems/199902676

Chappell D. (2008), «SOAP/WS-* and REST: Complementary Communication Styles», http://www.slideboom.com/presentations/35165/SOAP%2FWS-*-and-REST%3A%0BComplementary-Communication-Styles

Chappell D. (2009), «SOAP vs. REST: Complements or Competitors?», http://proceedings.esri.com/library/userconf/devsummit09/papers/keynote_chappell.pdf

Gahlout A. (2013), «Difference between REST and SOAP», http://www.counsellingbyabhi.com/2013/03/difference-between-rest-and-soap.html

Jansen G. (2011), «Resources», http://restful-api-design.readthedocs.org/en/latest/resources.html

Krafzig D., Banke K. & Slama D. (2004), «Enterprise SOA: Service-Oriented Architecture Best Practices», Prentice Hall

McKinley B. (2012), «Web Services: SOAP vs. REST», http://www.security-eh.com/members/securityeh/blog/VIEW/00000002/00000034/Web-Services-SOAP-vs-REST.html

Oftedal E. & Stock A. (2014), «REST Security Cheat Sheet», https://www.owasp.org/index.php/REST_Security_Cheat_Sheet

Rahaman M.A, Schaad A. & Rits M. (2006), «Towards secure SOAP message exchange in SOA», http://www.eurecom.fr/en/publication/2692/download/rs-publi-2692.pdf

Musser J. (2011), Open APIs: State of the Market, ProgrammableWeb, http://qconsf.com/sf2011/dl/qcon-sanfran-2011/slides/JohnMusser_OpenAPIs2011StateOfTheMarket.pdf

Potti P. K. και άλλοι (2012), «Comparing Performance of Web Service Interaction Styles: SOAP vs. REST», http://proc.conisar.org/2012/pdf/2208.pdf

Mulligan G. και άλλοι (2009) «A COMPARISON OF SOAP AND REST IMPLEMENTATIONS OF A SERVICE BASED INTERACTION INDEPENDENCE MIDDLEWARE FRAMEWORK», www.informs-sim.org/wsc09papers/133.pdf

Κεφάλαιο Ε: Τεκμηρίωση Συμπερασμάτων με Χρήση Κώδικα JAVA

Ε.1 Εισαγωγή

Στο κεφάλαιο αυτό, θα δημιουργήσουμε δύο απλά web service χρησιμοποιώντας τις δύο λύσεις SOAP και REST. Η προσπάθεια της τεκμηρίωσης θα εστιαστεί σε μετρήσιμα κυρίως μεγέθη σε πειραματικό περιβάλλον και για το λόγο αυτό η κυριότερη παράμετρος που μπορεί άμεσα να αξιολογηθεί είναι η απόδοση. Παράλληλα και στο πλαίσιο της υλοποίησης των web services μπορεί να αξιολογηθεί και το επίπεδο ευκολίας δημιουργίας αυτών.

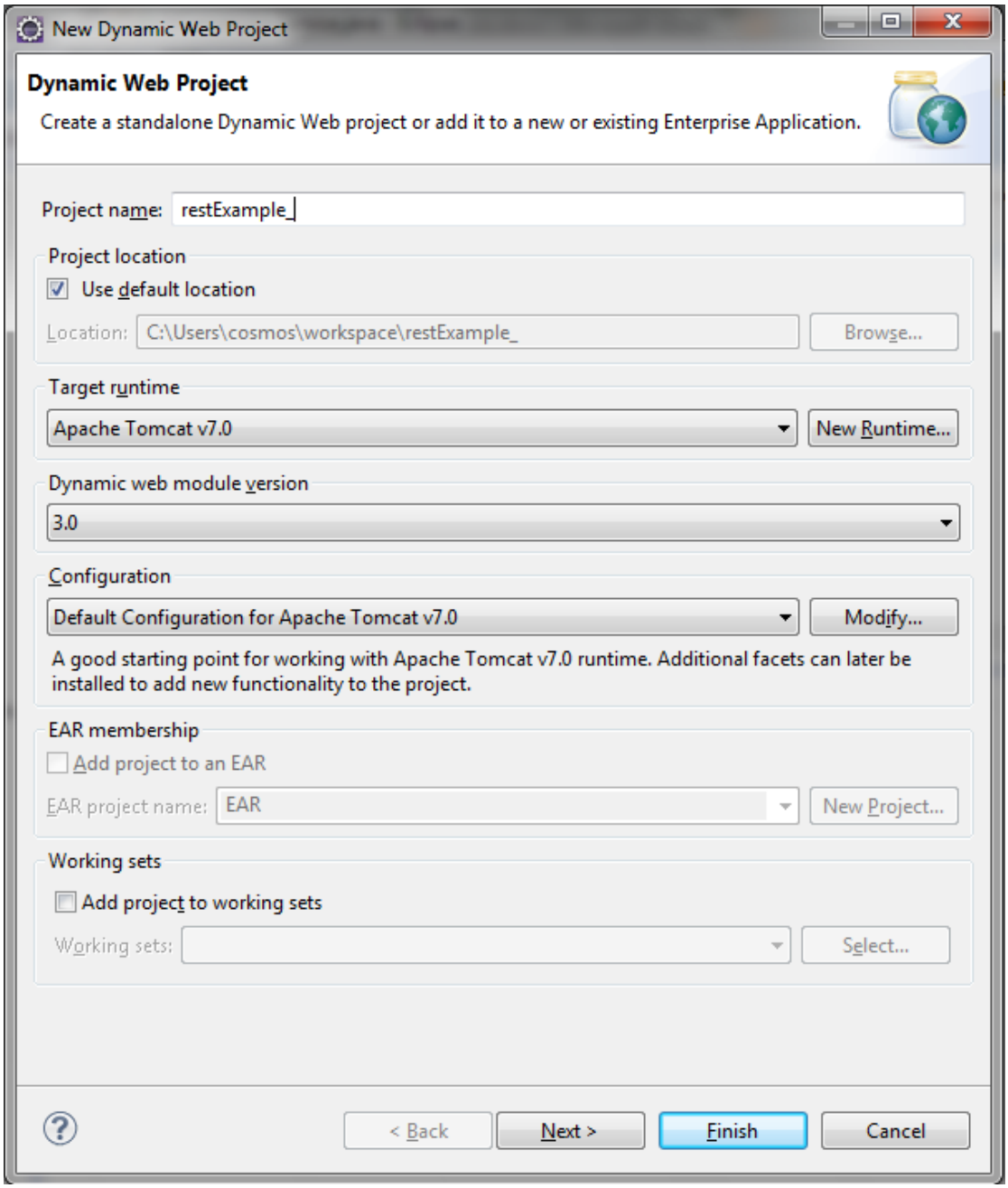
Για τη δημιουργία των web service θα χρησιμοποιήσουμε την Eclipse και άλλα εργαλεία που είναι απαραίτητα.

Ε.2 Δημιουργία REST web service

Για τη δημιουργία web service με REST θα χρησιμοποιήσουμε το Jersey το οποίο είναι ένα framework για τη δημιουργία RESTful web service στη Java και βασίζεται στο JAX-RS (Java API for RESTful Web Services). Το Jersey μπορούμε να το κατεβάσουμε από το https://jersey.java.net/download.html

Επιλέγουμε την έκδοση 1.18 την οποία αφού κατεβάσουμε αποσυμπιέζουμε στο σκληρό δίσκο.

Ανοίγουμε την Eclipse και δημιουργούμε ένα νέο Dynamic Web project.



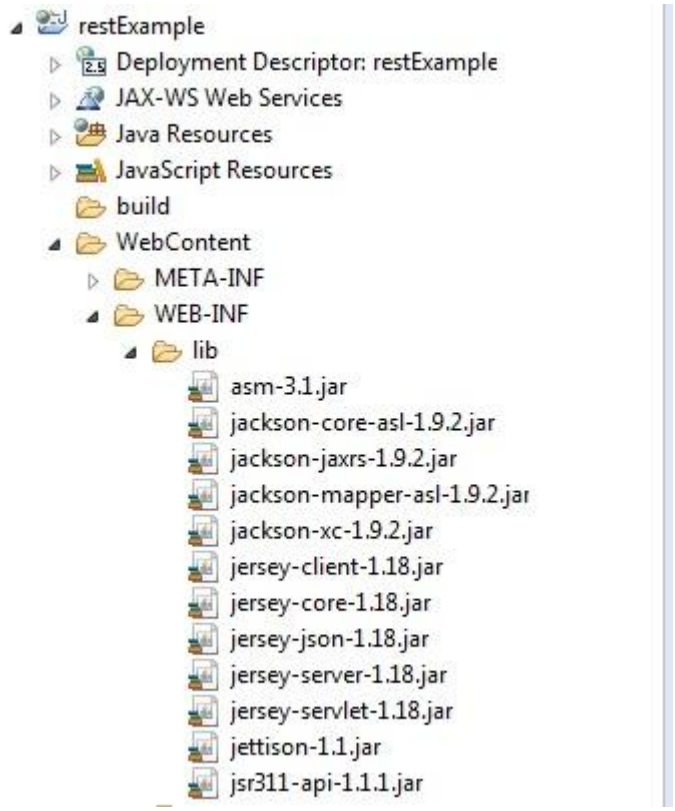
Εικόνα 41: Δημιουργία νέου Dynamic Web project στο Eclipse

Στη συνέχεια τα jar αρχεία που υπήρχαν στον κατάλογο lib που αποσυμπιέσαμε:

- asm-3.1
- jackson-core-asl-1.9.2
- jackson-jaxrs-1.9.2

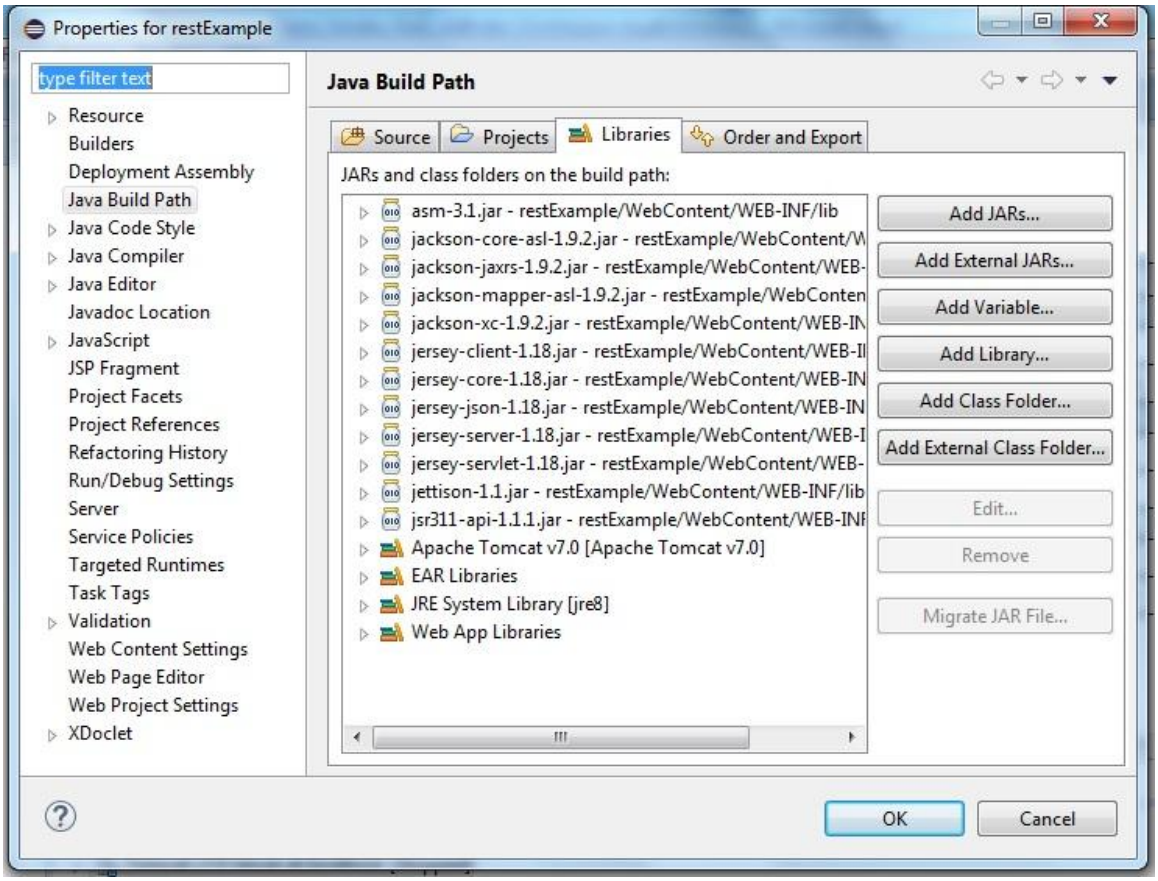
- jackson-mapper-asl-1.9.2
- jackson-xc-1.9.2
- jersey-client-1.18
- jersey-core-1.18
- jersey-json-1.18
- jersey-server-1.18
- jersey-servlet-1.18
- jettison-1.1
- jsr311-api-1.1.1

τα αντιγράφουμε στον κατάλογο WebContent/WEB-INF/lib



Εικόνα 42: Εισαγωγή βιβλιοθηκών στο Eclipse

Επίσης πρέπει να τα προσθέσουμε στο Build Path του project, επιλέγοντας Build Path και στη συνέχεια Configure Build Path.



Εικόνα 43: Java Build Path στο Eclipse

Δημιουργούμε ένα νέο package που το ονομάζουμε org.restexample.webservice και μέσα σε αυτό δημιουργούμε μια νέα κλάση Java με όνομα ParametrizedMethodServicesUsingREST η οποία περιέχει τις ακόλουθες εντολές[1]:

```
package org.restexample.webservice;

import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("basic")
public class ParametrizedMethodServicesUsingREST {

    @Path("name")
    @POST
```

[1] Ο Κώδικας αποτελεί τροποποίηση εκείνου που διατίθεται στο http://www.javatechtipssharedbygaurav.com/2013/05/rest-web-service-example-with.html

```
    @Produces(MediaType.TEXT_PLAIN)
    public String displayName(String name) {
        return "My Name is " + name;
    }

    public boolean isParsableToInteger(String strValue) {
        try {
            Integer.parseInt(strValue);
            return true;
        } catch (NumberFormatException nfe) {
            return false;
        }
    }

    @Path("countVowels")
    @POST
    @Produces(MediaType.TEXT_PLAIN)
    public String countVowels(String word) {

        int count = 0;
        String result = null;
        for (int i = 0; i < word.length(); i++) {
            if (word.charAt(i) == 'a' || word.charAt(i) == 'A') {
                count++;
            } else if (word.charAt(i) == 'e' || word.charAt(i) == 'E') {
                count++;
            } else if (word.charAt(i) == 'i' || word.charAt(i) == 'I') {
                count++;
            } else if (word.charAt(i) == 'o' || word.charAt(i) == 'O') {
                count++;
            } else if (word.charAt(i) == 'u' || word.charAt(i) == 'U') {
                count++;
            }
        }
        System.out
                .println("Number of vowels in the given string is : " + count);
        result = String.valueOf(count) +" vowels found in the given string
"+word ;
        System.out.println(result);
        return result;
    }

    @Path("factorial")
    @POST
    @Produces(MediaType.TEXT_PLAIN)
    public String getFactorial(String inputString) {

        String factorialResult = "Not a Valid Number";

        if (inputString == null || inputString.trim().length() == 0) {
            return "";
        }

        if (isParsableToInteger(inputString)) {

            Long number = Long.parseLong(inputString);
```

```
            if (number < 0) {

                return factorialResult = "Number must be positive";
            } else {

                Long factorial = number;

                for (Long i = (number - 1); i > 1; i--) {
                    factorial = factorial * i;
                }
                factorialResult = "Factorial of given number " + number
                        + " is " + String.valueOf(factorial);

            }

        }
        return factorialResult;

    }

}
```

Ας δούμε εν συντομία το πρόγραμμα. Όπως μπορούμε να δούμε χρησιμοποιούμε τις βασικές εντολές της Java και επιπλέον τα ακόλουθα που είναι τμήμα του Jersey framework:

- @Path(/your_path_at_class_level) : Καθορίζει το path για το base URL. Το base URL καθορίζεται από το όνομα του application, το servlet και το URL pattern από το αρχείο web.xml.
- @Path(/your_path_at_method_level): Καθορίζει το path για το base URL /your_path_at_class_level+ /your_path_at_method_level
- @Produces(MediaType.TEXT_XML [, more-types]): Το @Produces καθορίζει ποιος MIME τύπος παραδίδεται από μια μέθοδο που έχει δηλωθεί με μια @POST. Στο παράδειγμά μας χρησιμοποιούμε το TEXT_PLAIN.

Το service προσφέρει 3 επιλογές:

- Με τη μέθοδο displayName(String name) επιστρέφει το κείμενο που δόθηκε (String name) με το πρόθεμα: “My Name is:”. Το κείμενο μπορεί να είναι σημαντικά μεγάλο δίνοντας τη δυνατότητα να δοκιμάσουμε την απόδοση του service με μεγάλο αριθμό χαρακτήρων.

- Με τη μέθοδο countVowels(String word) υπολογίζει τον αριθμό των φωνηέντων που περιλαμβάνονται σε ένα κείμενο. Και εδώ επίσης το κείμενο μπορεί να είναι σημαντικά μεγάλο δίνοντας τη δυνατότητα να δοκιμάσουμε την απόδοση του service με μεγάλο αριθμό χαρακτήρων, ενώ παράλληλα εκτελείται μία εκτεταμένη επαναληπτική διαδικασία με συγκρίσεις χαρακτήρων που σε συνάρτηση με μεγάλο κείμενο δημιουργούν πίεση στο service.

- Με τη μέθοδο getFactorial(String inputString) υπολογίζεται το παραγοντικό ενός φυσικού αριθμού. Η επαναλπτική διαδικασία σε συνδυασμό με τις πράξεις πολλαπλασιασμού μεγάλων αριθμών (Long) αναμένεται και πάλι να πιέσουν το service.

- Η μέθοδος isParsableToInteger(String strValue) είναι βοηθητική της μεθόδου getFactorial(String inputString) για τον έλεγχο των τιμών εισόδου από το χρήστη.

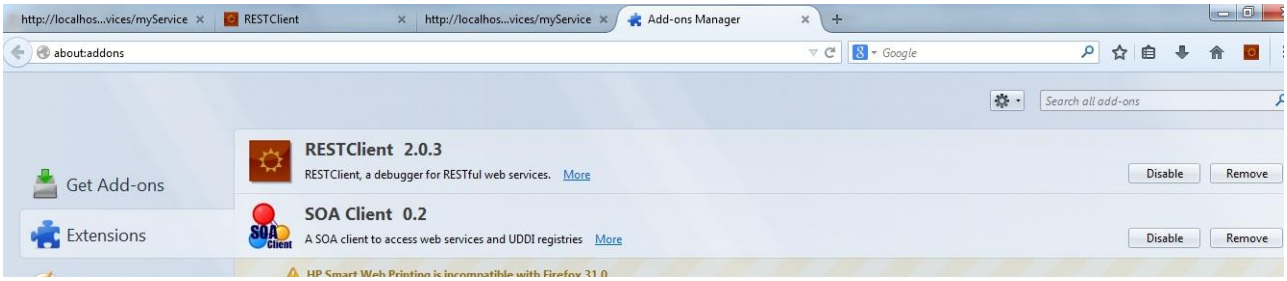
Τώρα πρέπει να δημιουργήσουμε το αρχείο web.xml το οποίο θα το τοποθετήσουμε στο WebContent/WEB-INF/

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
  <display-name>restExample</display-name>
  <servlet>
    <servlet-name>Jersey REST Service</servlet-name>
    <servlet-
class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
    <init-param>
      <param-name>jersey.config.server.provider.packages</param-name>
      <param-value>org.restexample.webservice</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Jersey REST Service</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
    </web-app>
```

Το αρχείο καθορίζει τις βασικές παραμέτρους της εφαρμογής μας, όπως το display name που έχει δηλωθεί restExample, το url-pattern που είναι /rest/ κ.λ.π

Με την ολοκλήρωση του συγκεκριμένου βήματος, μπορούμε να δοκιμάσουμε το web service. Πατάμε δεξί κλικ στο project επιλέγουμε Run As και στη συνέχεια Run On Server. Αν έχουμε ορίσει τον tomcat σαν web server στην Eclipse, θα ξεκινήσει και θα φορτώσει το web service.

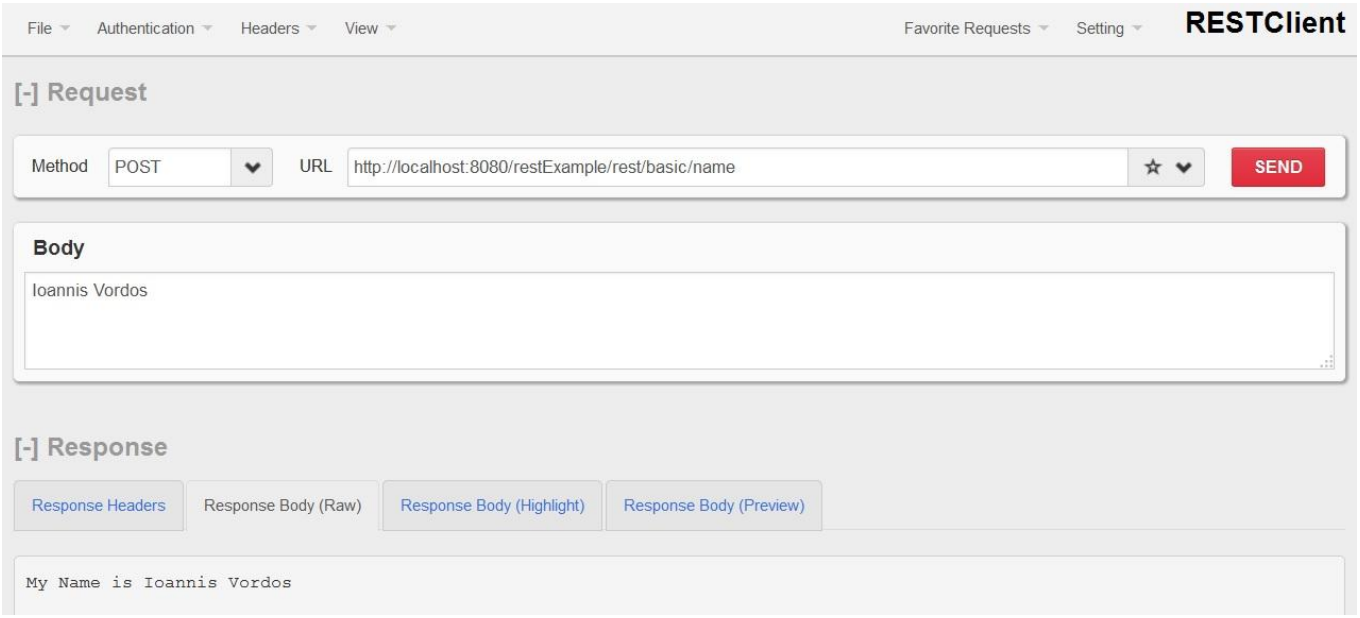
Για τη δοκιμή του service θα χρησιμοποιήσουμε το browser firefox και το add-on RESTClient 2.0.3.



Εικόνα 44: Το RESTClient 2.0.3 add-on του Firefox

Με το add-on αυτό στην ουσία δημιουργούμε ένα client και το ακόλουθο παράδειγμα αποτελεί μία χρήση της μεθόδου displayName του service. Η κλήση γίνετε με τη μέθοδο POST, το κείμενο που δίνεται είναι το ονοματεπώνυμο του συντάκτη και το χρησιμοποιούμενο link είναι:

http://localhost:8080/restExample/rest/basic/name



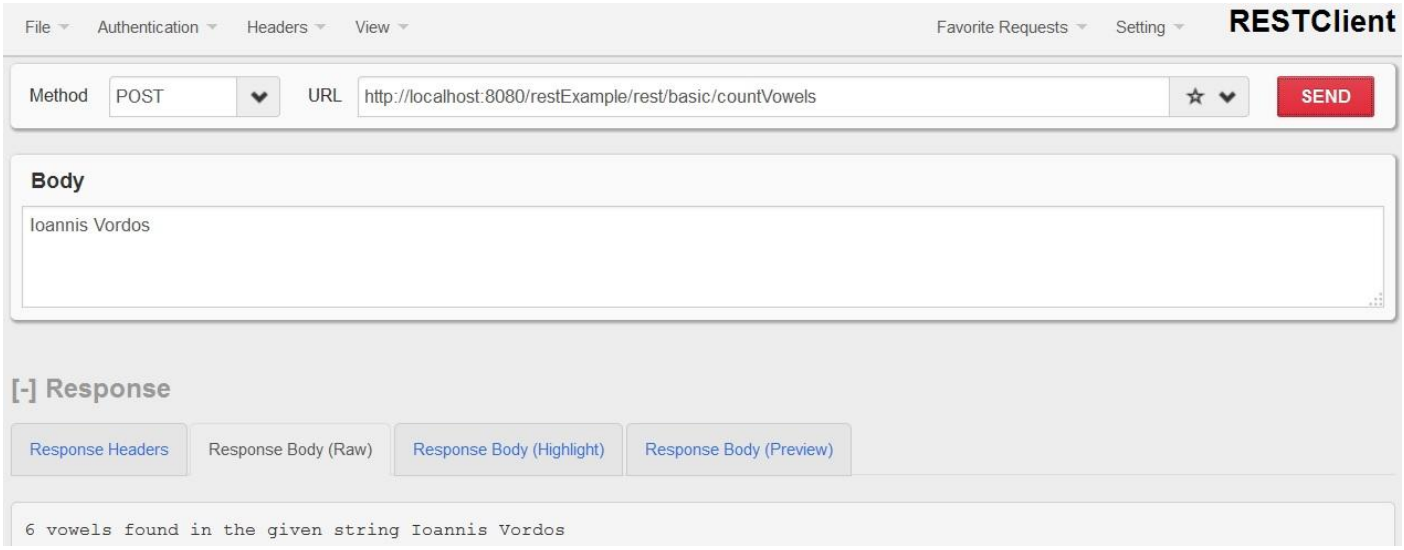
Εικόνα 45: Δοκιμή της μεθόδου displayName του REST Service

Ας αναλύσουμε λίγο το link που δώσαμε:

- http://localhost:8080 είναι η διεύθυνση:πόρτα που ακροάται ο web server
- restExample είναι το display name που έχει δηλωθεί στο web.xml
- rest είναι το url-pattern που έχει δηλωθεί στο web.xml
- basic είναι το @Path(/your_path_at_class_level)
- name είναι το @Path(/your_path_at_method_level), ενώ η τιμή “Ioannis Vordos” στο πεδίο Body είναι η παράμετρος που θα περάσει στον κώδικα Java για να επιστρέψει το string “My Name is Ioannis Vordos” που φαίνεται στο πεδίο Response Body (Raw).

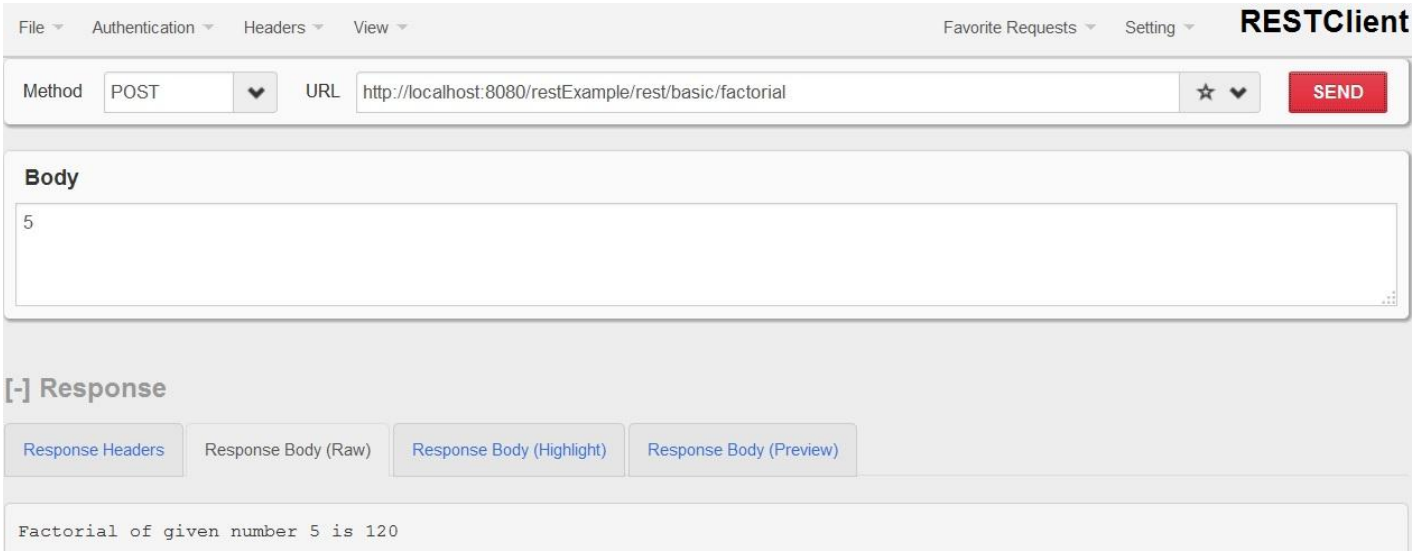
Τα αντίστοιχα αποτελέσματα προκύπτουν αν χρησιμοποιήσουμε τα κατάλληλα Path για τις άλλες δύο μεθόδους ως εξής:

http://localhost:8080/restExample/rest/basic/countVowels



Εικόνα 46: Δοκιμή της μεθόδου countVowels του REST Service

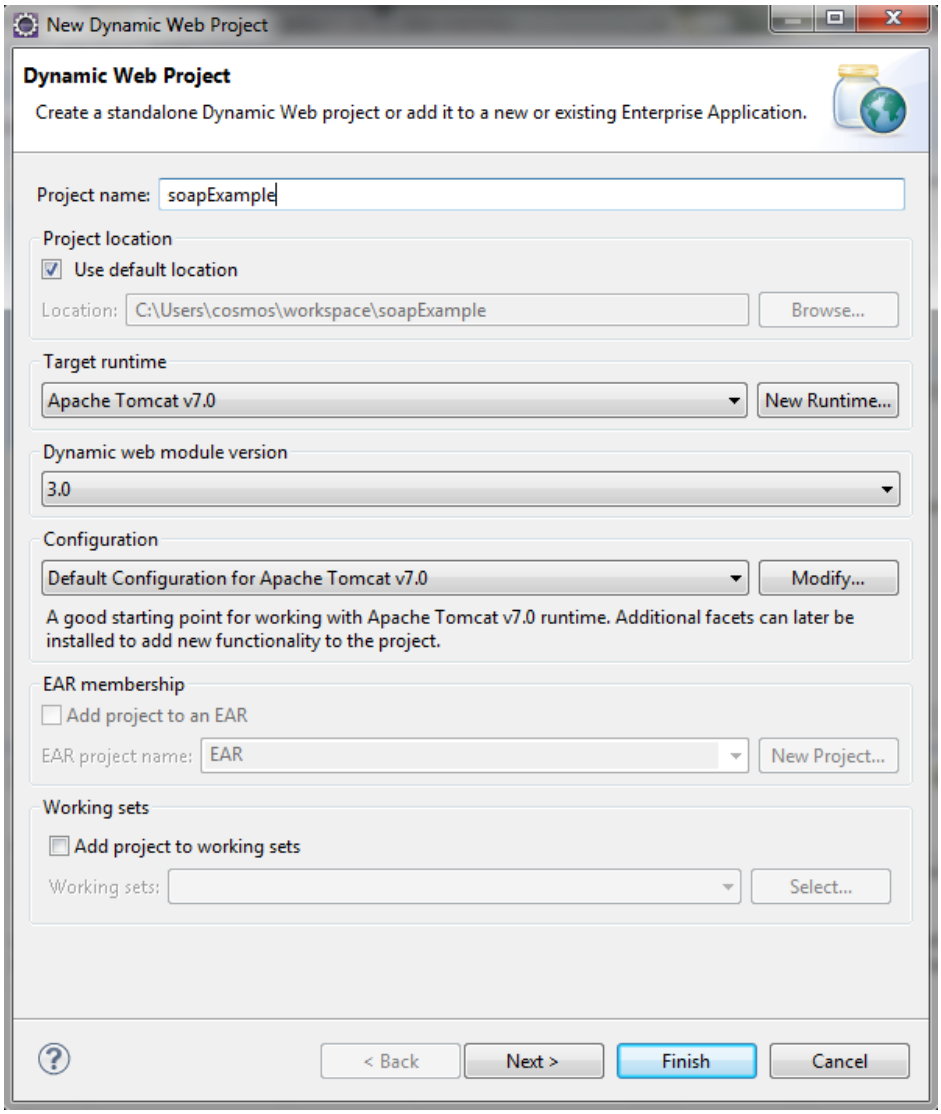
http://localhost:8080/restExample/rest/basic/factorial



Εικόνα 47: Δοκιμή της μεθόδου getFactorial του REST Service

Ε.3 Δημιουργία SOAP web service

Για τη δημιουργία ενός SOAP web service με την Eclipse θα ξεκινήσουμε ακολουθώντας τα ίδια βήματα, δηλαδή θα δημιουργήσουμε ένα Dynamic Web Project.



Εικόνα 48: Δημιουργία νέου Dynamic Web project στο Eclipse

Στη συνέχεια μέσα στο project θα δημιουργήσουμε ένα νέο package που θα το ονομάσουμε org.soapexample.webservice. Μέσα στο package θα προσθέσουμε μια νέα Java Class με όνομα myService.java η οποία θα περιέχει τις μεθόδους που θέλουμε να προσφέρουμε σαν web service.

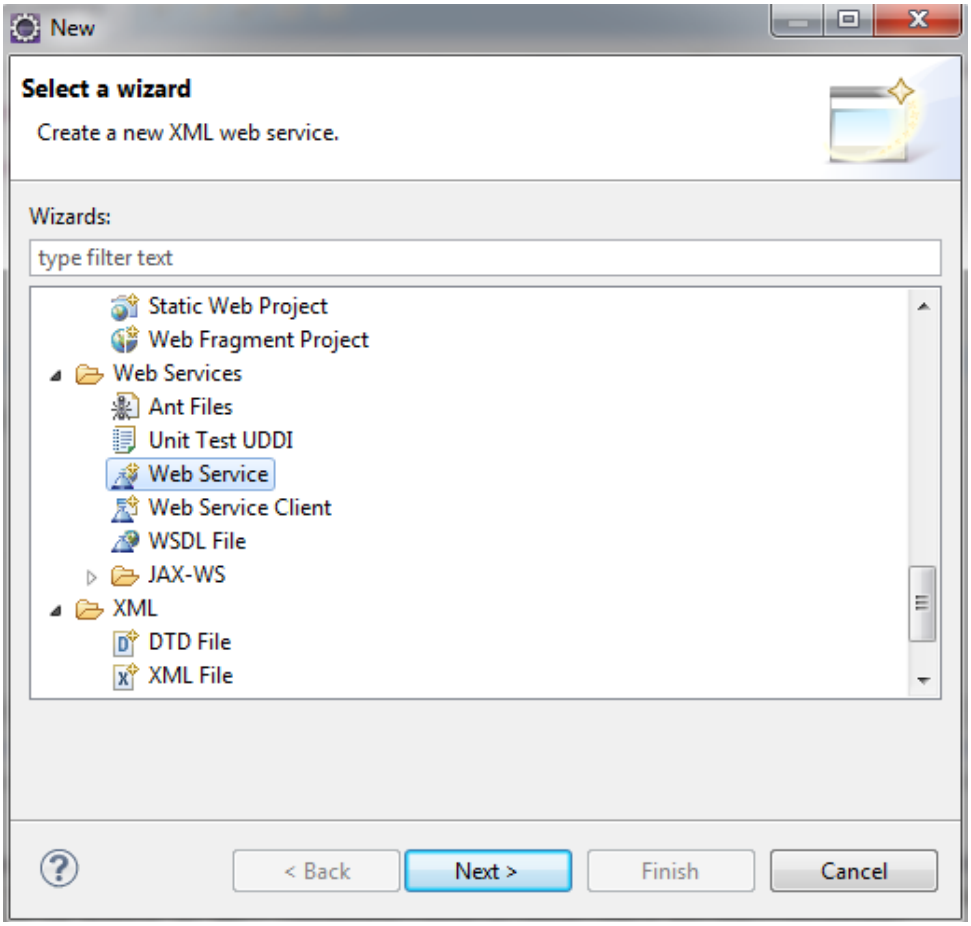
```
package org.soapexample.webservice;

public class myService {
        public String displayName(String name) {
        return "My Name is " + name;
    }


    public String countVowels(String word) {
```

```
        int count = 0;
        String result = null;
        for (int i = 0; i < word.length(); i++) {
            if (word.charAt(i) == 'a' || word.charAt(i) == 'A') {
                count++;
            } else if (word.charAt(i) == 'e' || word.charAt(i) == 'E') {
                count++;
            } else if (word.charAt(i) == 'i' || word.charAt(i) == 'I') {
                count++;
            } else if (word.charAt(i) == 'o' || word.charAt(i) == 'O') {
                count++;
            } else if (word.charAt(i) == 'u' || word.charAt(i) == 'U') {
                count++;
            }
        }
        System.out
                .println("Number of vowels in the given string is : " + count);
        result = String.valueOf(count) +" vowels found in the given string
"+word ;
        System.out.println(result);
        return result;
    }


    public String getFactorial(String inputString) {

        String factorialResult = "Not a Valid Number";
        boolean check;

        if (inputString == null || inputString.trim().length() == 0) {
            return "";
        }
        try {
            Integer.parseInt(inputString);
            check = true;
        } catch (NumberFormatException nfe) {
            check = false;
        }
        if (check) {

            Long number = Long.parseLong(inputString);

            if (number < 0) {

                return factorialResult = "Number must be positive";
            } else {

                Long factorial = number;

                for (Long i = (number - 1); i > 1; i--) {
                    factorial = factorial * i;
                }
                factorialResult = "Factorial of given number " + number
                        + " is " + String.valueOf(factorial);
```

```
            }
        }
        return factorialResult;
    }
}
```

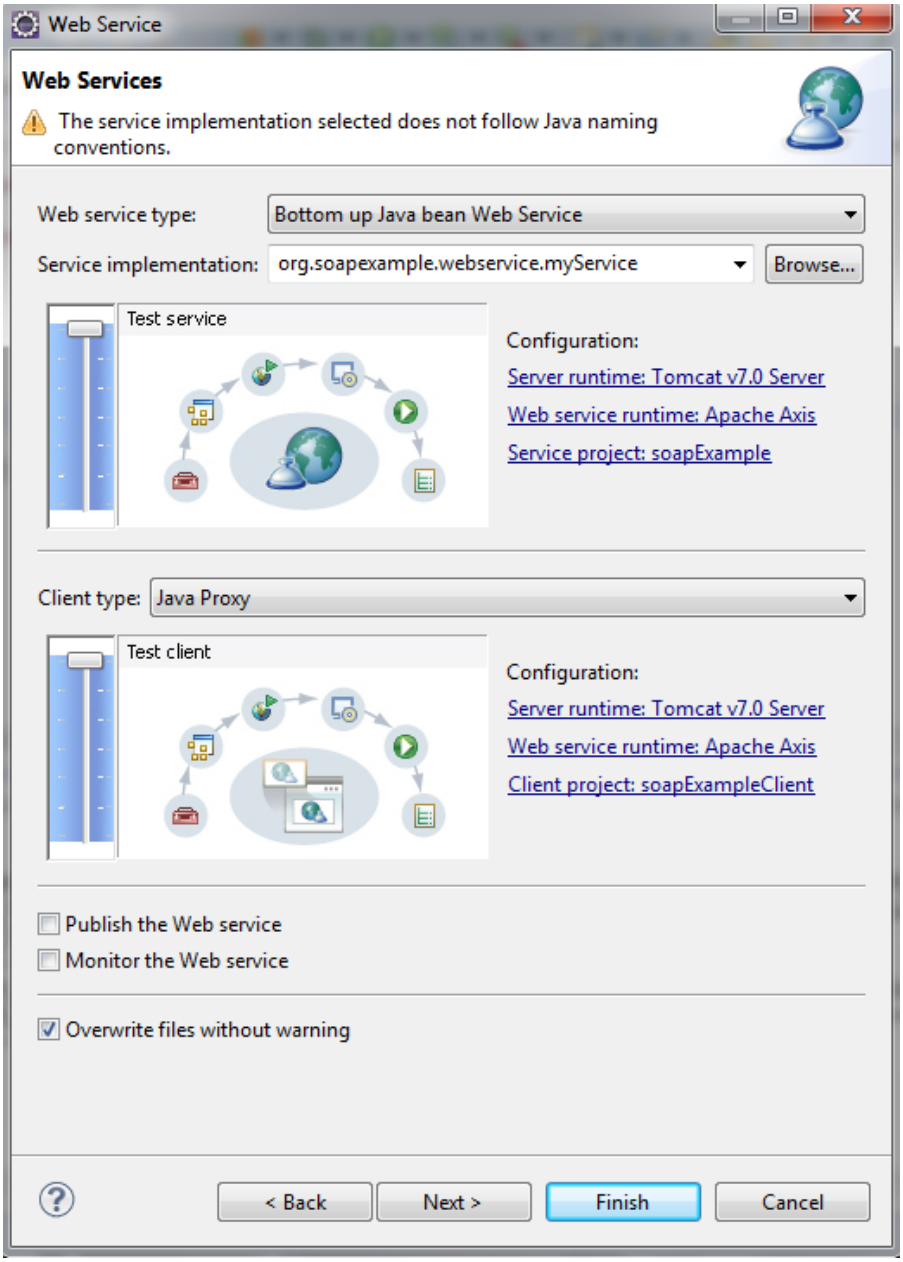
Όπως παρατηρούμε ο κώδικας είναι πανομοιότυπος με εκείνον του REST service και αποτελείται από τις ίδιες μεθόδους με κατάλληλη τροποποίηση από το συντάκτη.

Στη συνέχεια για να δημιουργήσουμε το web service κάνουμε δεξί κλικ πάνω στο αρχείο myService.java και επιλέγουμε New και στη συνέχεια Other. Στο παράθυρο που θα εμφανιστεί επιλέγουμε Web Service.



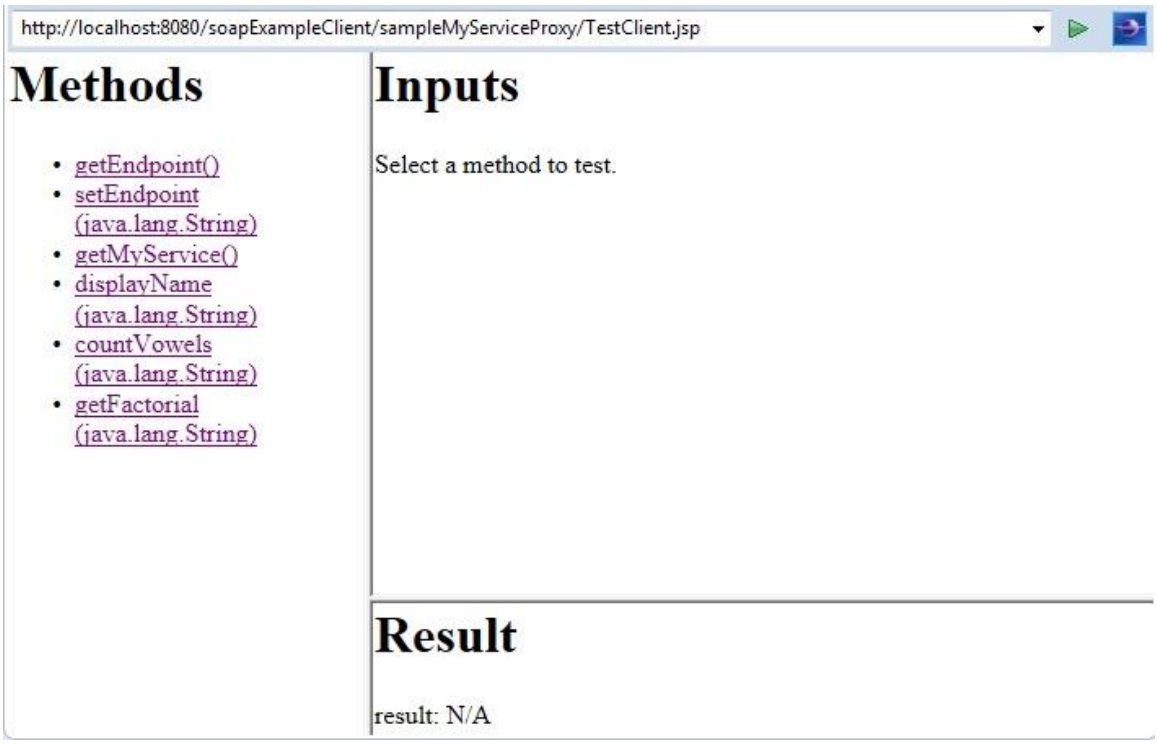
Εικόνα 49: Δημιουργία Web Service στο Eclipse

Στη συνέχεια θα εμφανιστεί η ακόλουθη εικόνα, όπου έχουμε μια σειρά από επιλογές όπως να δημιουργήσουμε το web service, να το κάνουμε publish, να δημιουργήσουμε έναν client και όλα αυτά με τη χρήση δύο κάθετων slider. Επιλέγουμε να κάνουμε Test Service και Test Client και μπορούμε να πατήσουμε το Finish ή το Next για να δούμε τα βήματα δημιουργίας του web service και του client.



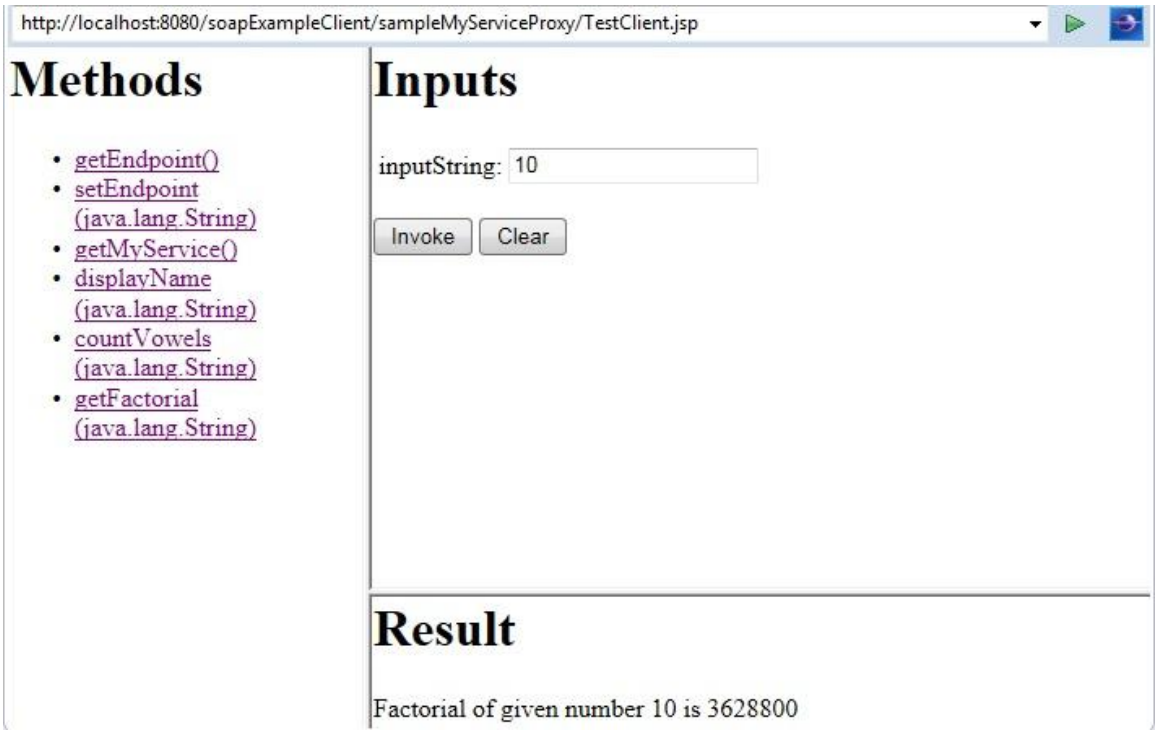
Εικόνα 50: Δημιουργία νέου Web Service στο Eclipse

Αφού ολοκληρωθεί η διαδικασία θα δούμε την εικόνα που ακολουθεί όπου μπορούμε μέσα από ένα γραφικό περιβάλλον να έχουμε πρόσβαση στα web service επιλέγοντας ένα από τα displayName, countVowels ή getFactorial.



Εικόνα 51: Πρόσβαση στο Web Service

Για παράδειγμα επιλέγοντας getFactorial και δίνοντας στο text box (inputString) την τιμή 10 και πατώντας το invoke θα δω το αποτέλεσμα της κλήσης του web service.



Εικόνα 52: Δοκιμή του Web Service

Ε.4 Εργαλεία Παρακολούθησης

Για να συγκρίνουμε τις δύο τεχνολογίες και τα δύο πανομοιότυπα services που δημιουργήσαμε με αυτές θα χρησιμοποιήσουμε το εργαλείο jmeter της Apache, το οποίο είναι διαθέσιμο στην ιστοσελίδα http://jmeter.apache.org/

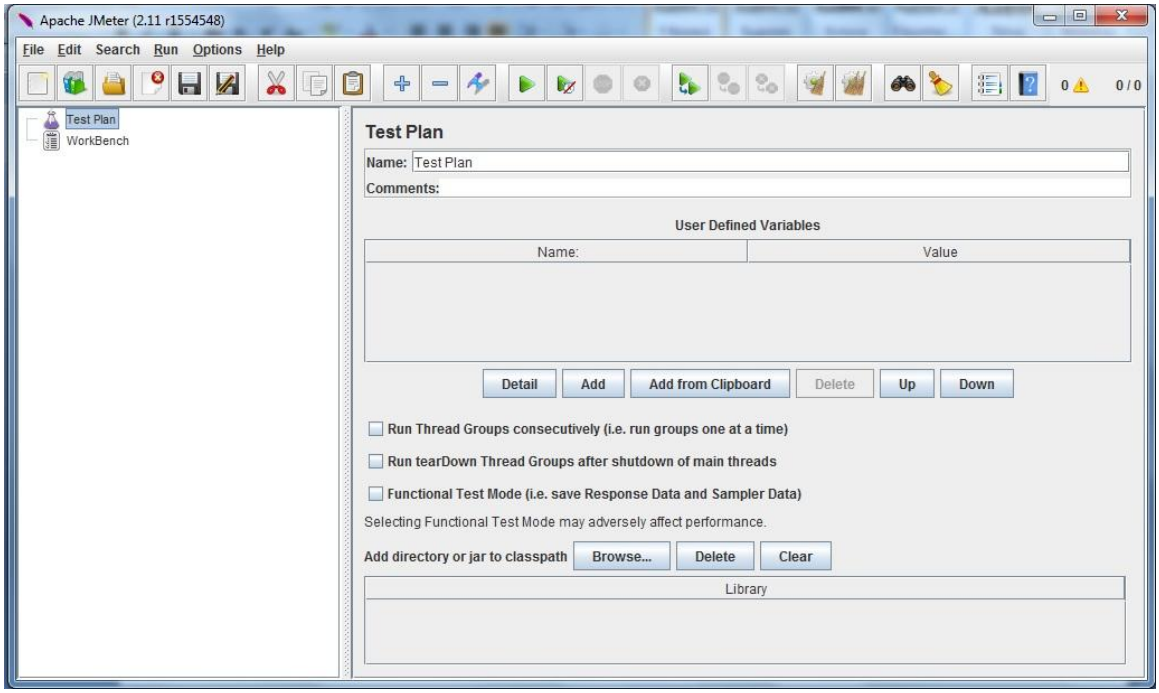
Εικόνα 53: JMeter της Apache

Το Apache JMeter μπορεί να χρησιμοποιηθεί για τον έλεγχο επιδόσεων τόσο σε στατικούς όσο και δυναμικούς πόρους (αρχεία, δυναμικές γλώσσες Web - PHP,

Java, ASP.NET, κ.λπ. -, αντικείμενα Java, βάσεις δεδομένων και queries, FTP Servers και άλλα). Μπορεί να χρησιμοποιηθεί για να προσομοιώσει ένα βαρύ φορτίο σε ένα εξυπηρετητή, μία ομάδα εξυπηρετητών, ένα δίκτυο ή αντικείμενο για να δοκιμάσουν την αντοχή τους ή να αναλύσουν τη συνολική απόδοση κάτω από διάφορους τύπους φορτίου. Μπορούμε να το χρησιμοποιήσετε για να κάνουμε μια γραφική ανάλυση των επιδόσεων ή να δοκιμάσουμε τη συμπεριφορά server/script/objects αντικειμένων κάτω από βαρύ ταυτόχρονο φορτίο.

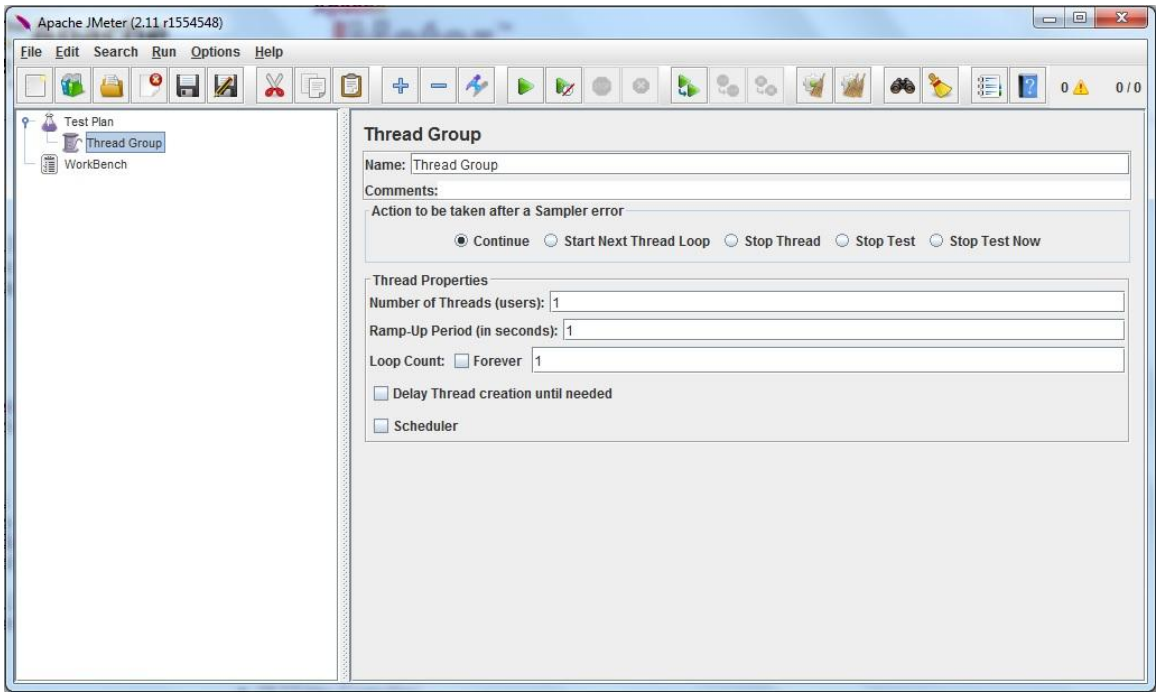
Ε.4.1 Προετοιμασία JMeter

Κατεβάζουμε και αποσυμπιέζουμε το apache-jmeter-2.11.zip. Ξεκινώντας το εκτελέσιμο jmeter.bat εκκινεί το πρόγραμμα.



Εικόνα 54: Δημιουργία Test Plan στο JMeter

Στη συνέχεια προσθέτουμε ένα Thread Group, από το οποίο μπορούμε να διαχειριζόμαστε τον αριθμό των ταυτόχρονων threads (χρηστών), τον αριθμό των επαναλήψεων της δοκιμής και άλλα.

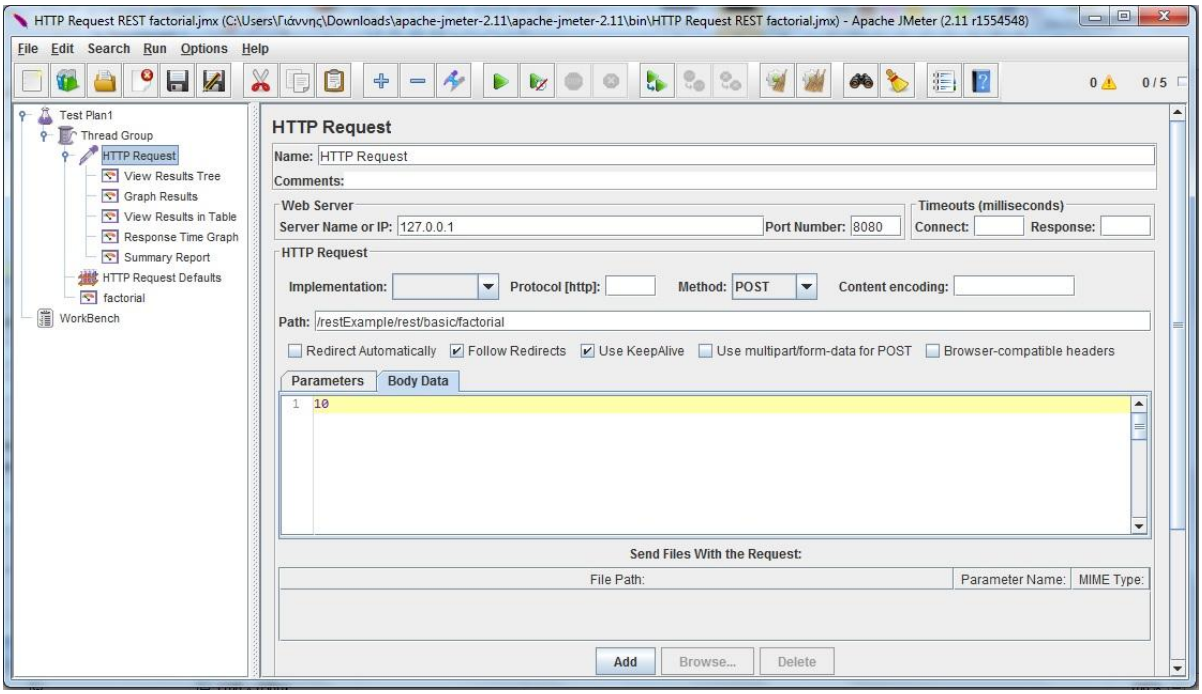


Εικόνα 55: Δημιουργία Thread Group στο JMeter

Στη συνέχεια προσθέτουμε ένα sampler με τον οποίο θα γίνει προσομοίωση του client για κάθε ένα από τα δύο services.

Ε.4.2 Δημιουργία περιβάλλοντος ελέγχου Rest Web Service

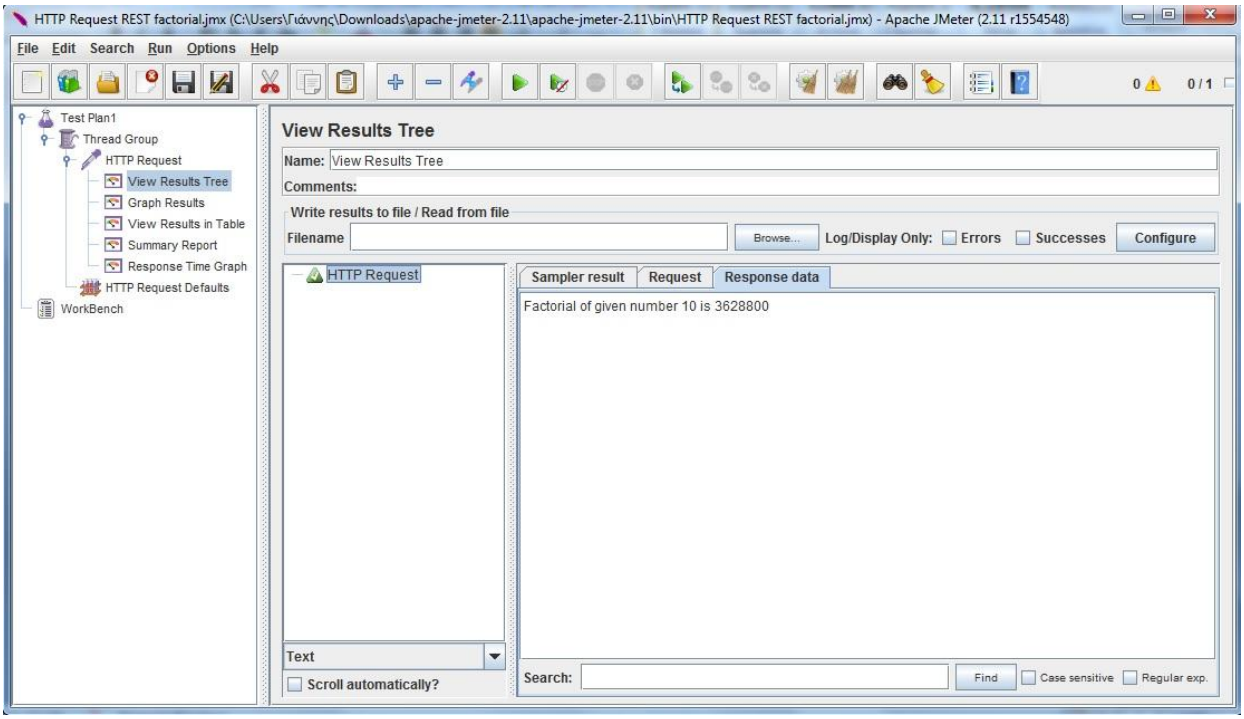
Για τη δοκιμή του REST service στο Thread Group προσθέτουμε το sampler Http Request.



Εικόνα 56: Δημιουργία HTTP Request στο JMeter

Στο sampler προσθέτουμε καταγραφείς (listeners) με τους οποίους στη συνέχεια θα συλλέξουμε τα στοιχεία από τις δοκιμές για να κάνουμε τις απαραίτητες συγκρίσεις. Εδώ θα χρησιμοποιήσουμε τους View Results Tree (για έλεγχο ορθότητας επιστροφών), Graph Results (για δημιουργία γραφήματος αποτελεσμάτων), View results in Table (για εξαγωγή δεδομένων σε πίνακα), Response Time Graph (για καταγραφή χρόνου απόκρισης) και Summary Report (για συγκεντρωτικά αποτελέσματα). Στο HTTP Request (sampler) εισάγουμε την IP του server μας που εδώ είναι 127.0.0.1 (localhost), το port που ακροάται ο server (8080), τη μέθοδο που θα χρησιμοποιήσουμε (POST), το Path του service (για τον υπολογισμό του factorial είναι: /restExample/rest/basic/factorial), τέλος στο πεδίο Body Data εισάγουμε την τιμή της παραμέτρου που θα δοθεί στο server για την παραγωγή του αποτελέσματος (10).

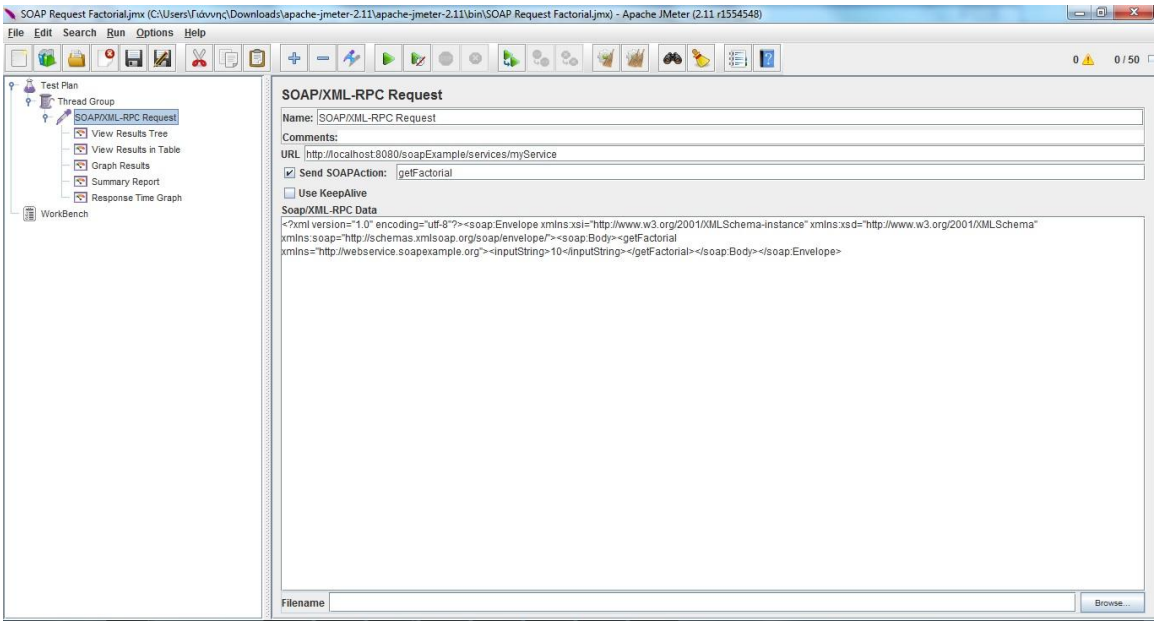
Με εκκίνηση της δοκιμής για ένα thread και μία επανάληψη δίνει τα ακόλουθα αποτελέσματα:



Εικόνα 57: Εισαγωγή Listeners στο JMeter

Ε.4.3 Δημιουργία περιβάλλοντος ελέγχου Soap Web Service

Για τη δοκιμή του SOAP service στο Thread Group προσθέτουμε το sampler SOAP/XML-RPC Request.



Εικόνα 58: Δημιουργία SOAP Request στο JMeter

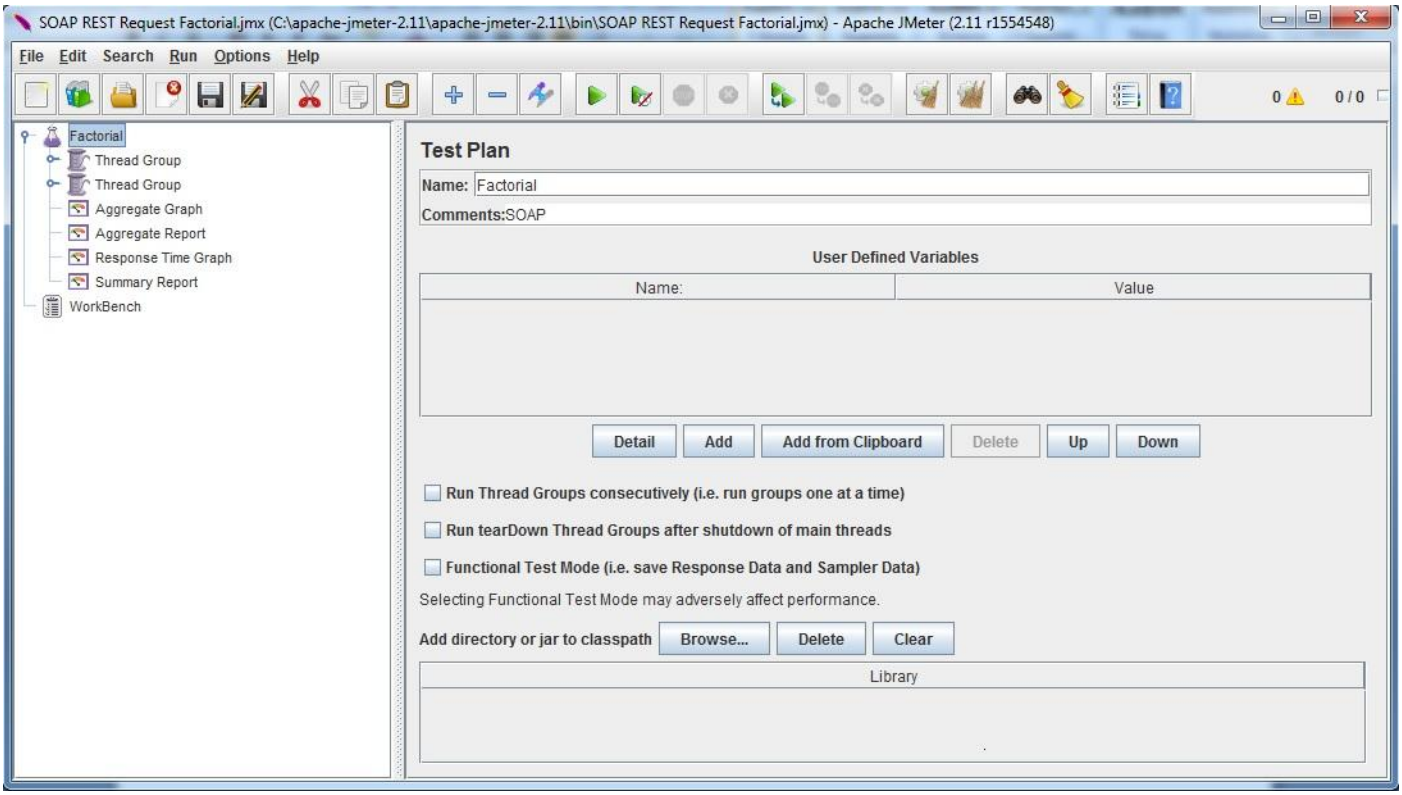
Στο sampler προσθέτουμε τους ίδιους καταγραφείς (listeners) με τους οποίους στη συνέχεια θα συλλέξουμε τα στοιχεία από τις δοκιμές για να κάνουμε τις απαραίτητες συγκρίσεις. Και εδώ θα χρησιμοποιήσουμε τους View Results Tree (για έλεγχο ορθότητας επιστροφών), Graph Results (για δημιουργία γραφήματος αποτελεσμάτων), View results in Table (για εξαγωγή δεδομένων σε πίνακα), Response Time Graph (για καταγραφή χρόνου απόκρισης) και Summary Report (για συγκεντρωτικά αποτελέσματα). Στο SOAP/XML-RPC Request (sampler) εισάγουμε το URL του server μας που εδώ είναι http://localhost:8080/soapExample/services/myService, εισάγουμε επίσης το επιλεγόμενο SOAPAction που εδώ είναι πάλι το getFactorial, τέλος στο πεδίο SOAP/XML-RPC Data εισάγουμε το SOAP envelope που θα στείλει ο client που εδώ για να στείλουμε στη μέθοδο getFactorial την τιμή 10, είναι:

```
<?xml version="1.0" encoding="utf-8"?>
      <soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
            <soap:Body>
                  <getFactorial xmlns="http://webservice.soapexample.org">
                        <inputString>
                              20
                        </inputString>
                  </getFactorial>
            </soap:Body>
      </soap:Envelope>
```

Ε4.4 Δημιουργία συγκριτικού περιβάλλοντος ελέγχου

Τέλος δημιουργείται ένα περιβάλλον ελέγχου για ταυτόχρονη σύγκριση των δύο παραπάνω υπηρεσιών. Αυτό επιτυγχάνεται με συγχώνευση (merge) του περιβάλλοντος ελέγχου (sample) REST με εκείνο του SOAP. Παράλληλα διαγράφουμε τους επιμέρους καταγραφείς (listener) για εξοικονόμηση πόρων μνήμης και προσθέτουμε νέους καταγραφείς (listener) για τη συγκριτική διαδικασία οι οποίοι είναι Aggregate Graph (για τη δημιουργία διαγράμματος σύγκρισης), Aggregate Report (για παροχή συγκεντρωτικού συγκριτικού πίνακα), Response Time Graph (για την απεικόνιση του χρόνου απόκρισης των υπηρεσιών) και

Summary Report (για συγκεντρωτικά συγκριτικά αποτελέσματα). Η επόμενη εικόνα παρουσιάζει τη δομή του συγκριτικού περιβάλλοντος ελέγχου για την υπηρεσία (service) getFactorial.



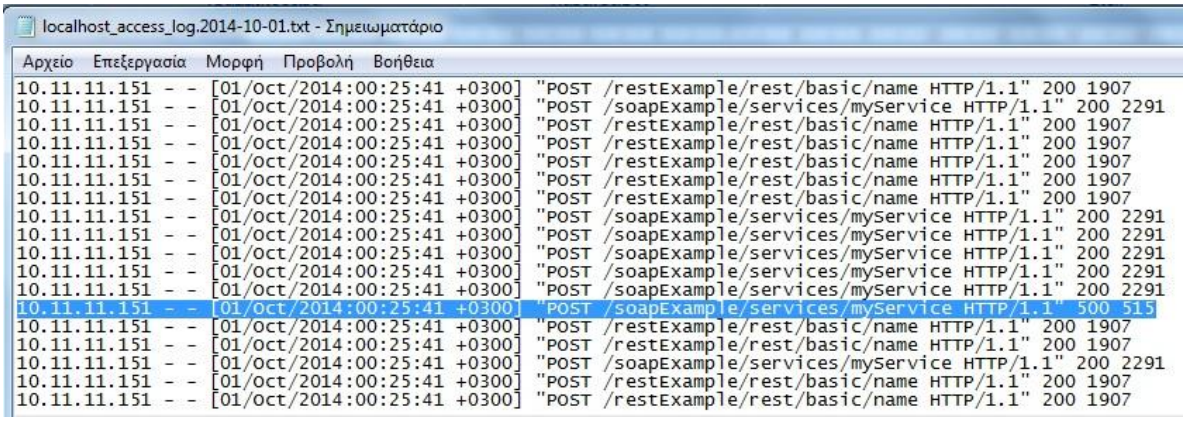
Εικόνα 59: Δημιουργία συγκριτικού Test Plan στο JMeter

Για την βέλτιστη αξιοποίηση των πόρων του συστήματος από της υπό εξέταση υπηρεσίες, κατά τη διάρκεια των δοκιμών χρησιμοποιήθηκε ένας υπολογιστής για την υποστήριξη των υπηρεσιών (server) και ένας για την εκτέλεση των δοκιμών με το jmeter. Οι δύο υπολογιστές βρίσκονταν συνδεδεμένοι σε τοπικό δίκτυο LAN με υποδομή δικτύου 100 Mbps (καλωδίωση και switch). Οι δύο υπολογιστές διέθεταν τα ίδια τεχνικά χαρακτηριστικά (PROCESSOR: Intel i3, RAM: 4GB, OS: Windows 7).

Τα σφάλματα (errors) που παρουσιάζονται στο SOAP Web Service σε διάφορες συνθήκες κίνησης, οφείλονται σε προβλήματα υλοποίησης (bugs) των Web Services με χρήση της συγκεκριμένης πλατφόρμας υλοποίησης (eclipse, apache tomcat). Το σφάλμα οφείλεται στο exception: java.util.ConcurrentModificationException, το οποίο εμφανίζεται σε πολυνηματικά (multithreaded) περιβάλλοντα. Το επιστρεφόμενο μήνυμα ήταν της μορφής:

```
<?xml version="1.0" encoding="UTF-8"?>
        <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                <soapenv:Body>
                        <soapenv:Fault>
                                <faultcode>
                                        soapenv:Server.userException
                                </faultcode><faultstring>
                                        java.util.ConcurrentModificationException
                                </faultstring>
                                        <detail>
                                                <ns1:hostname
xmlns:ns1="http://xml.apache.org/axis/">
                                                        Wermis
                                                </ns1:hostname>
                                        </detail>
                        </soapenv:Fault>
                </soapenv:Body>
        </soapenv:Envelope>
```

Από τα Logs του server είχαμε τα ακόλουθα:



Εικόνα 60: Σφάλμα (error) στις καταγραφές (logs) του εξυπηρετητή (server)

Τα σφάλματα εμφανίζονται με τυχαία συχνότητα και σε όλα τα σενάρια κίνησης (χαμηλή έως υψηλή). Λόγω του σημαντικά μικρού ποσοστού (κάτω από 0,5%) εκτιμάται ότι δεν επηρεάζουν την αξιοπιστία των μετρήσεων.

E.5 Αποτελέσματα

Στην ενότητα αυτή θα εκτελέσουμε δοκιμές των τριών μεθόδων των δύο web services (REST και SOAP) που δημιουργήσαμε πριν με τρία διαφορετικά σενάρια και θα συγκρίνουμε τα αποτελέσματα. Τα σενάρια ελέγχου θα είναι:

- Χαμηλής κίνησης. Με πέντε ταυτόχρονους χρήστες (threads) και 100 επαναλήψεις (loops) των αιτήσεων.
- Μεσαίας κίνησης. Με 20 ταυτόχρονους χρήστες και 1000 επαναλήψεις.
- Υψηλής κίνησης. Με 50 ταυτόχρονους χρήστες και 1000 επαναλήψεις

Ε.5.1 Μέθοδος getFactorial

Εδώ θα εξεταστεί η μέθοδος getFactorial με υπολογισμό του παραγοντικού του 20, το οποίο είναι ο μέγιστος αριθμός για τον οποίο η μέθοδος επιστρέφει σωστά αποτελέσματα.

Ε.5.1.1 Σενάριο Χαμηλής Κίνησης (5 χρήστες – 100 επαναλήψεις)

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
SOAP/XML-RPC ...	500	2	1	20	0.96	0.00%	435.5/sec	187.15	440.0
HTTP Request	500	0	0	18	0.89	0.00%	492.1/sec	95.73	199.2
TOTAL	1000	1	0	20	1.11	0.00%	871.1/sec	271.87	319.6

Εικόνα 61: Συγκεντρωτική αναφορά χαμηλής κίνησης getFactorial

Aggregate Report

Name: Aggregate Report

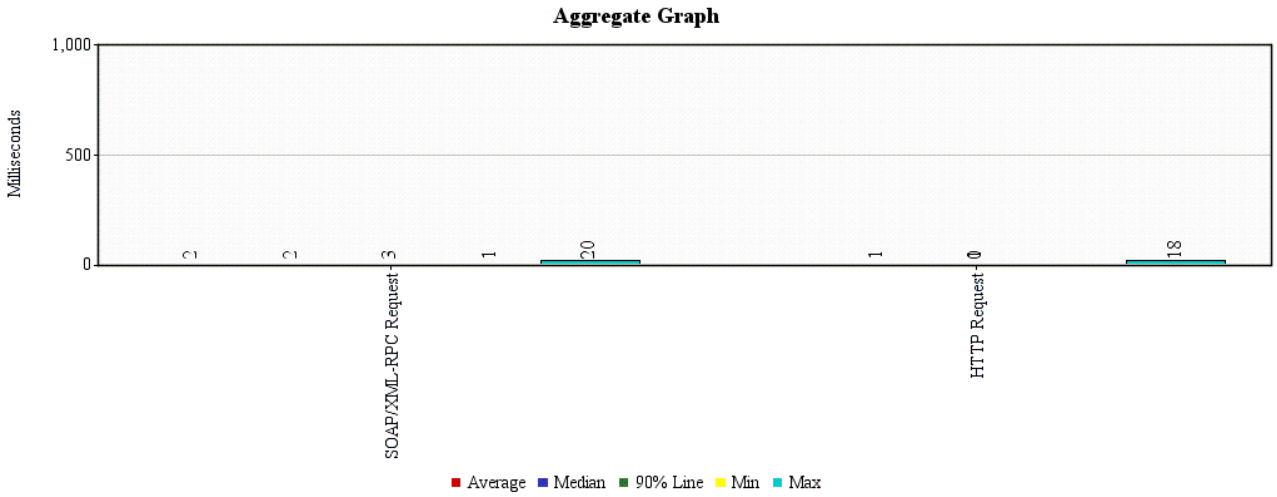
Comments:

Write results to file / Read from file

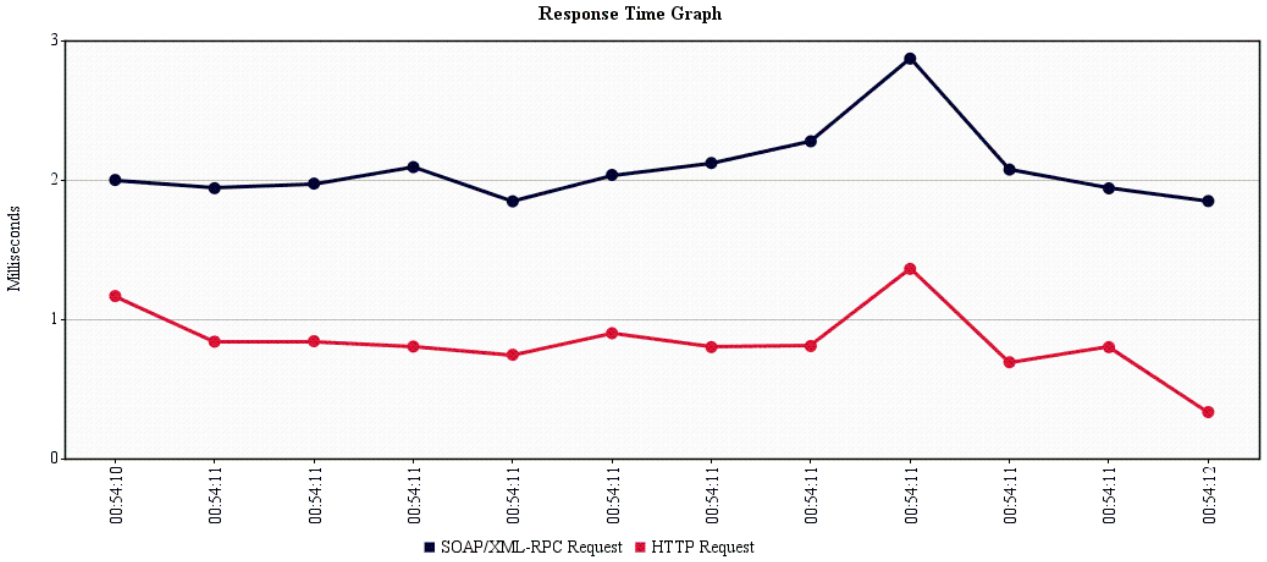
Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
SOAP/XML-RPC ...	500	2	2	3	1	20	0.00%	435.5/sec	187.1
HTTP Request	500	0	1	1	0	18	0.00%	492.1/sec	95.7
TOTAL	1000	1	1	2	0	20	0.00%	871.1/sec	271.9

Εικόνα 62: Συγκεντρωτική συγκριτική αναφορά χαμηλής κίνησης getFactorial



Εικόνα 63: Συγκεντρωτικό συγκριτικό διάγραμμα χαμηλής κίνησης getFactorial



Εικόνα 64: Συγκεντρωτικό διάγραμμα χρόνου απόκρισης χαμηλής κίνησης getFactorial

Κατά τη διάρκεια της μέτρησης η απασχόληση της CPU στο server έφτασε στιγμιαία το 20% ενώ της μνήμης ήταν 49%.

Ε.5.1.2 Σενάριο Μεσαίας Κίνησης (20 χρήστες – 1000 επαναλήψεις)

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
SOAP/XML-RP...	20000	4	1	98	5.33	0.21%	2724.4/sec	1171.01	440.1
HTTP Request	20000	2	0	77	4.09	0.00%	3061.8/sec	595.60	199.2
TOTAL	40000	3	0	98	4.86	0.11%	5448.8/sec	1700.97	319.7

Εικόνα 65: Συγκεντρωτική αναφορά μεσαίας κίνησης getFactorial

Aggregate Report

Name: Aggregate Report

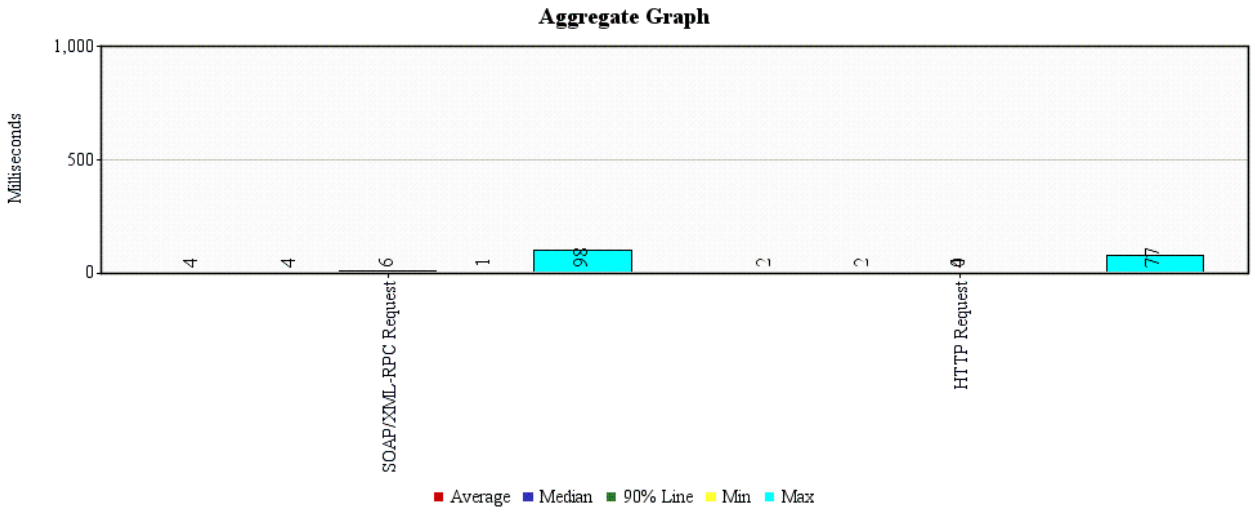
Comments:

Write results to file / Read from file

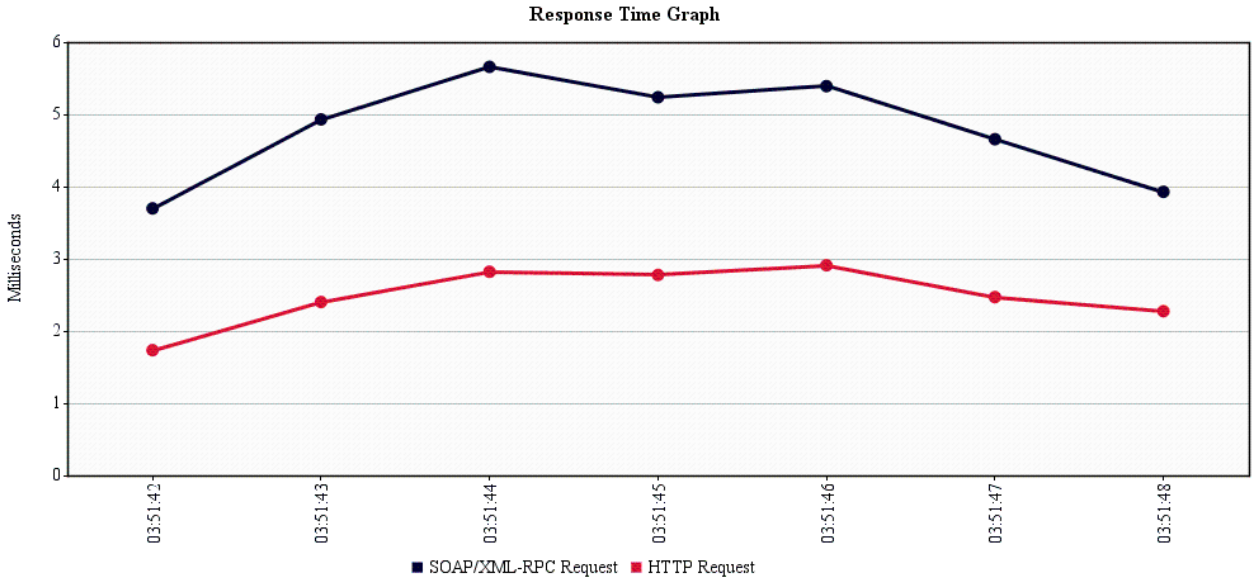
Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
SOAP/XML-RP...	20000	4	4	6	1	98	0.21%	2724.4/sec	1171.0
HTTP Request	20000	2	2	4	0	77	0.00%	3061.8/sec	595.6
TOTAL	40000	3	3	5	0	98	0.11%	5448.8/sec	1701.0

Εικόνα 66: Συγκεντρωτική συγκριτική αναφορά μεσαίας κίνησης getFactorial



Εικόνα 67: Συγκεντρωτικό συγκριτικό διάγραμμα μεσαίας κίνησης getFactorial



Εικόνα 68: Συγκεντρωτικό διάγραμμα χρόνου απόκρισης μεσαίας κίνησης getFactorial

Κατά τη διάρκεια της μέτρησης η απασχόληση της CPU έφτασε στιγμιαία το 33% ενώ της μνήμης ήταν 54%.

Ε.5.1.3 Σενάριο Υψηλής Κίνησης (50 χρήστες – 1000 επαναλήψεις)

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
SOAP/XML-RP...	50000	6	1	107	9.62	0.18%	2319.5/sec	996.93	440.1
HTTP Request	50000	3	0	96	7.43	0.00%	2412.4/sec	469.27	199.2
TOTAL	100000	4	0	107	8.69	0.09%	4639.1/sec	1448.13	319.7

Εικόνα 69: Συγκεντρωτική αναφορά υψηλής κίνησης getFactorial

Aggregate Report

Name: Aggregate Report

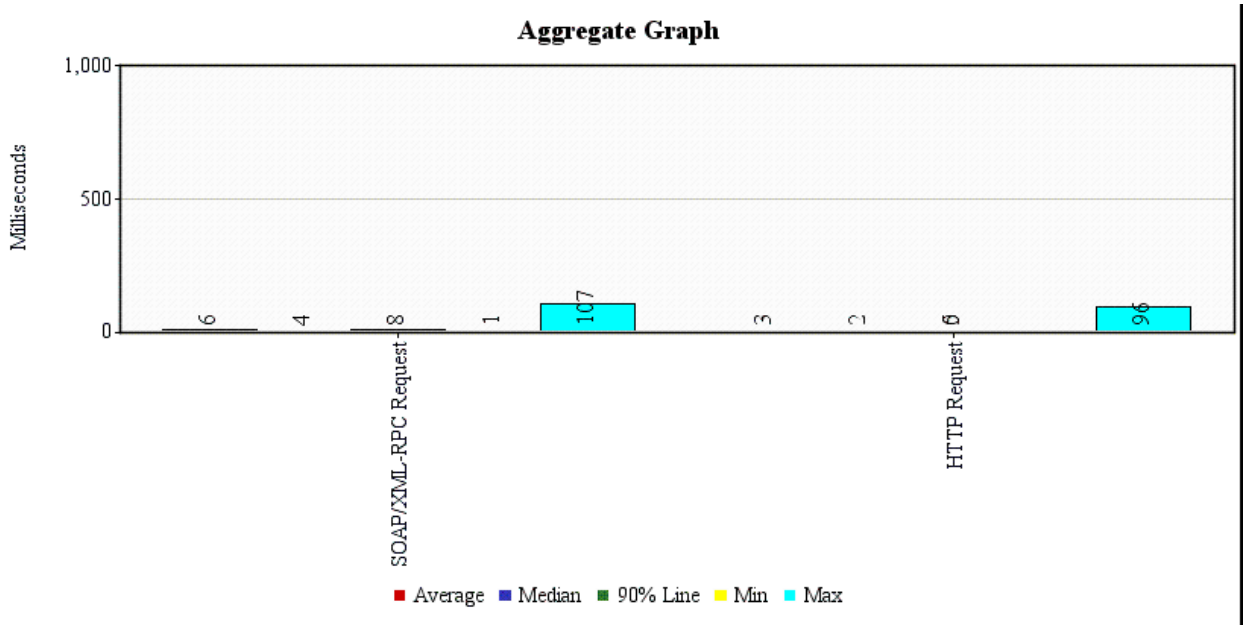
Comments:

Write results to file / Read from file

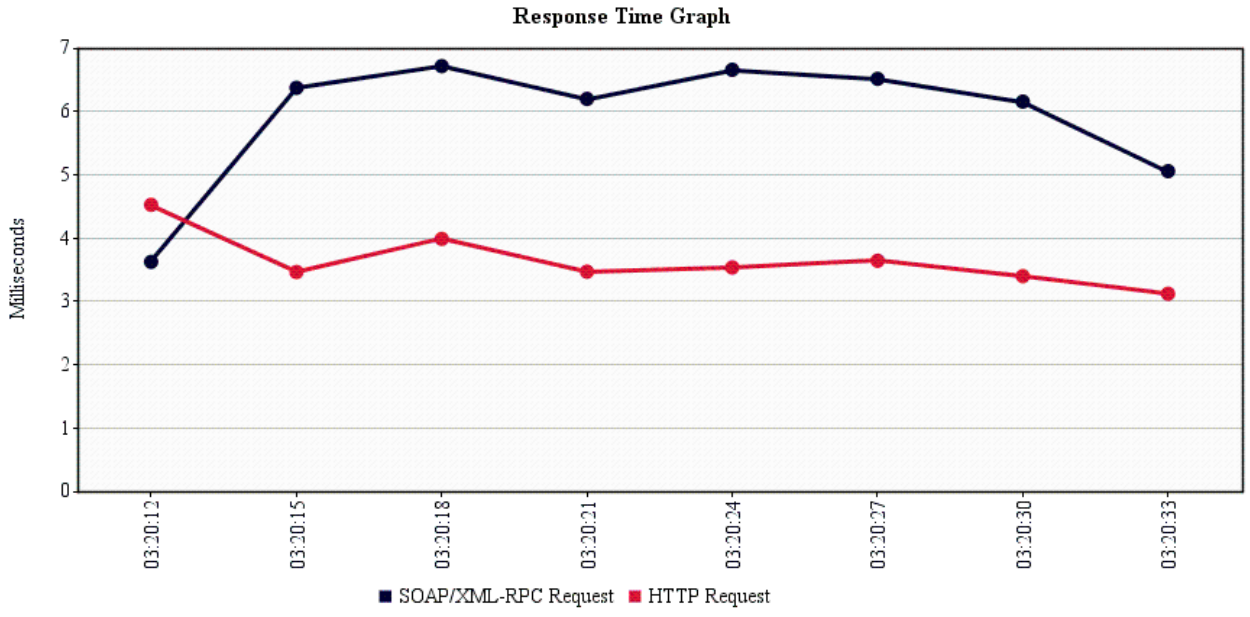
Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
SOAP/XML-RP...	50000	6	4	8	1	107	0.18%	2319.5/sec	996.9
HTTP Request	50000	3	2	5	0	96	0.00%	2412.4/sec	469.3
TOTAL	100000	4	3	7	0	107	0.09%	4639.1/sec	1448.1

Εικόνα 70: Συγκεντρωτική συγκριτική αναφορά υψηλής κίνησης getFactorial



Εικόνα 71: Συγκεντρωτικό συγκριτικό διάγραμμα υψηλής κίνησης getFactorial



Εικόνα 72: Συγκεντρωτικό διάγραμμα χρόνου απόκρισης υψηλής κίνησης getFactorial

Κατά τη διάρκεια της μέτρησης η απασχόληση της CPU έφτασε στιγμιαία το 44% ενώ της μνήμης ήταν έως 55%.

Ε.5.2 Μέθοδος displayName

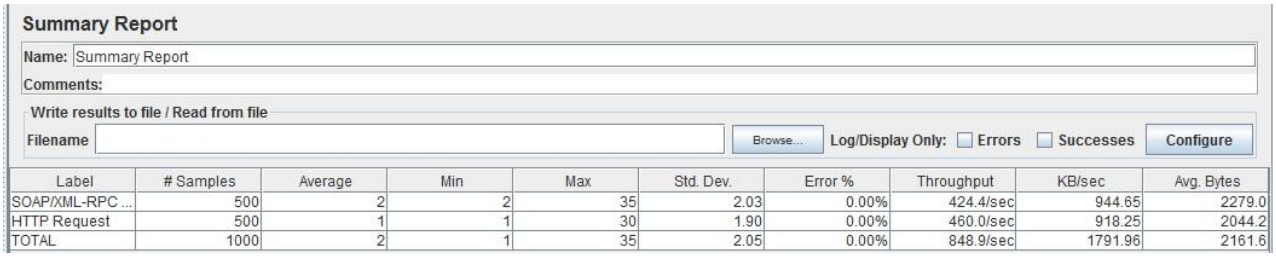
Εδώ θα εξεταστεί η μέθοδος displayName, όπου αντί ενός απλού ονόματος θα δίδεται σημαντικά μεγάλο κείμενο στην προσπάθεια να πιεστούν τα εξεταζόμενα services, τόσο κατά την αποστολή, όσο και κατά τη λήψη μεγάλων κειμένων χαρακτήρων. Το κείμενο που χρησιμοποιείται εδώ είναι[2]:

«These excellant intentions were strengthed when he enterd the Father Superior's diniing-room, though, stricttly speakin, it was not a dining-room, for the Father Superior had only two rooms alltogether; they were, however, much larger and more comfortable than Father Zossima's. But tehre was was no great luxury about the furnishng of these rooms eithar. The furniture was of mohogany, covered with leather, in the old-fashionned style of 1820 the floor was not even stained, but evreything was shining with cleanlyness, and there were many chioce flowers in the windows; the most sumptuous thing in the room at the moment was, of course, the beatifuly decorated table. The cloth was clean, the service shone; there were three kinds of well-baked bread, two bottles of wine, two of excellent mead, and a large glass jug of kvas -- both the latter made in the monastery, and famous in the neigborhood. There was no vodka. Rakitin related afterwards that there were five dishes: fish-suop made of sterlets, served with little fish paties; then boiled fish served in a spesial way; then salmon cutlets, ice pudding and compote, and finally, blanc-mange. Rakitin found out about all these good things, for he could not resist peeping into the kitchen, where he already had a footing. He had a footting everywhere, and got informaiton about everything. He was of an uneasy and envious temper. He was well aware of his own considerable abilities, and nervously exaggerated them in his self-conceit. He knew he would play a prominant part of some sort, but Alyosha, who was attached to him, was distressed to see that his friend Rakitin was dishonorble, and quite unconscios of being so himself, considering, on the contrary, that because he would not steal moneey left

[2] Το κείμενο βρέθηκε εδώ:
http://www.karamasoft.com/MVCSuiteSample/UltimateSpellMVCSamples/VeryLongText

on the table he was a man of the highest integrity. Neither Alyosha nor anyone else could have infleunced him in that.»

Ε.5.2.1 Σενάριο Χαμηλής Κίνησης (5 χρήστες – 100 επαναλήψεις)



Summary Report

Name: Summary Report

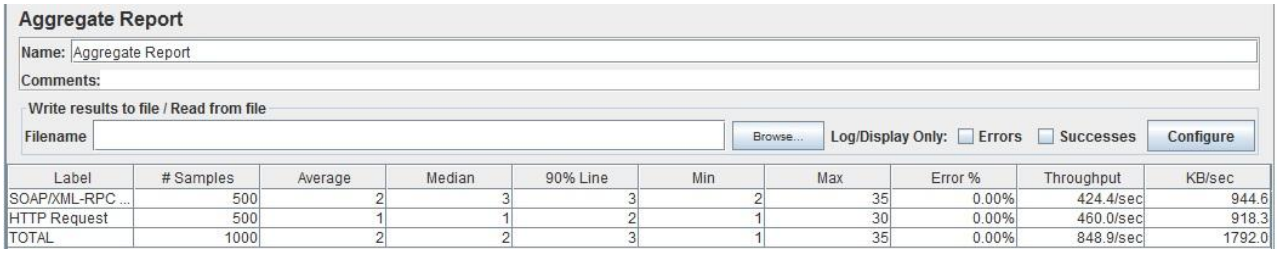
Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
SOAP/XML-RPC ...	500	2	2	35	2.03	0.00%	424.4/sec	944.65	2279.0
HTTP Request	500	1	1	30	1.90	0.00%	460.0/sec	918.25	2044.2
TOTAL	1000	2	1	35	2.05	0.00%	848.9/sec	1791.96	2161.6

Εικόνα 73: Συγκεντρωτική αναφορά χαμηλής κίνησης displayName



Aggregate Report

Name: Aggregate Report

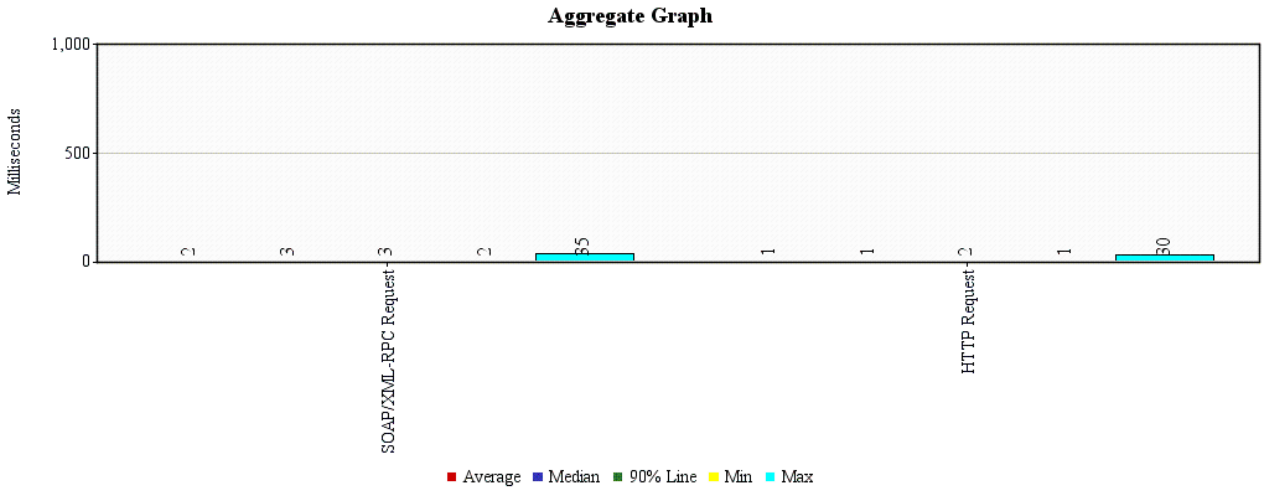
Comments:

Write results to file / Read from file

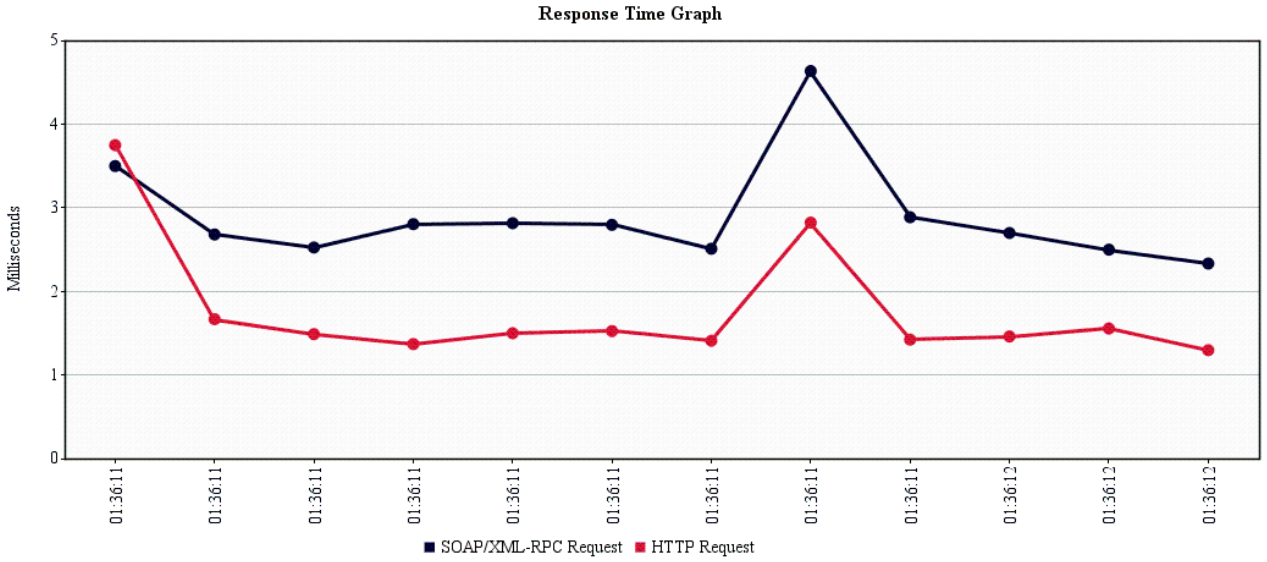
Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
SOAP/XML-RPC ...	500	2	3	3	2	35	0.00%	424.4/sec	944.6
HTTP Request	500	1	1	2	1	30	0.00%	460.0/sec	918.3
TOTAL	1000	2	2	3	1	35	0.00%	848.9/sec	1792.0

Εικόνα 74: Συγκεντρωτική συγκριτική αναφορά χαμηλής κίνησης displayName



Εικόνα 75: Συγκεντρωτικό συγκριτικό διάγραμμα χαμηλής κίνησης displayName



Εικόνα 76: Συγκεντρωτικό διάγραμμα χρόνου απόκρισης χαμηλής κίνησης displayName

Κατά τη διάρκεια της μέτρησης η απασχόληση της CPU έφτασε στιγμιαία το 10%. Επίσης η απασχόληση της μνήμης ήταν έως 53%.

E.5.2.2 Σενάριο Μεσαίας Κίνησης (20 χρήστες – 1000 επαναλήψεις)

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
SOAP/XML-RP...	20000	5	1	272	7.48	0.03%	1327.8/sec	2954.46	2278.6
HTTP Request	20000	3	1	279	6.80	0.00%	1407.0/sec	2808.69	2044.2
TOTAL	40000	4	1	279	7.24	0.01%	2655.5/sec	5605.03	2161.4

Εικόνα 77: Συγκεντρωτική αναφορά μεσαίας κίνησης displayName

Aggregate Report

Name: Aggregate Report

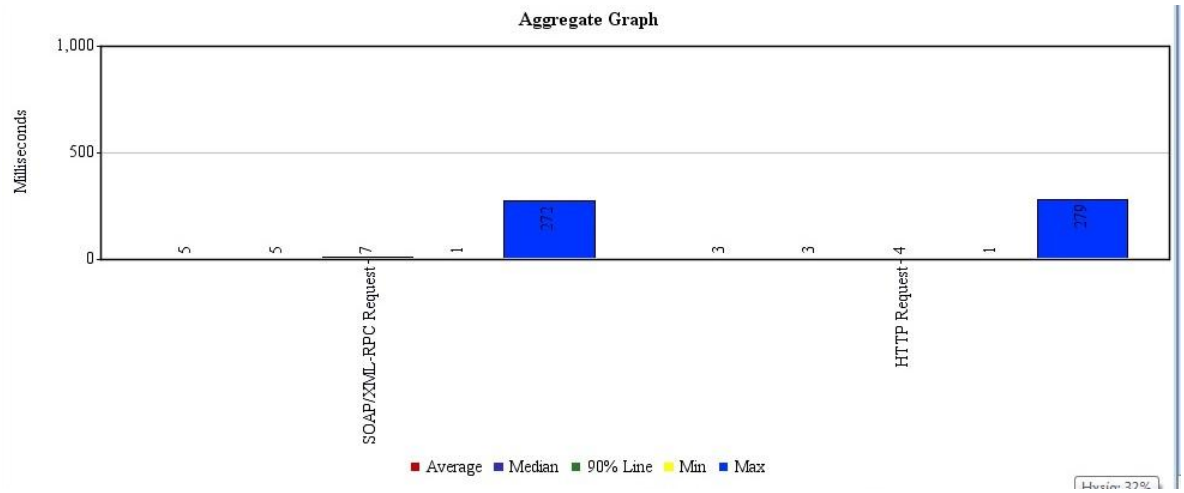
Comments:

Write results to file / Read from file

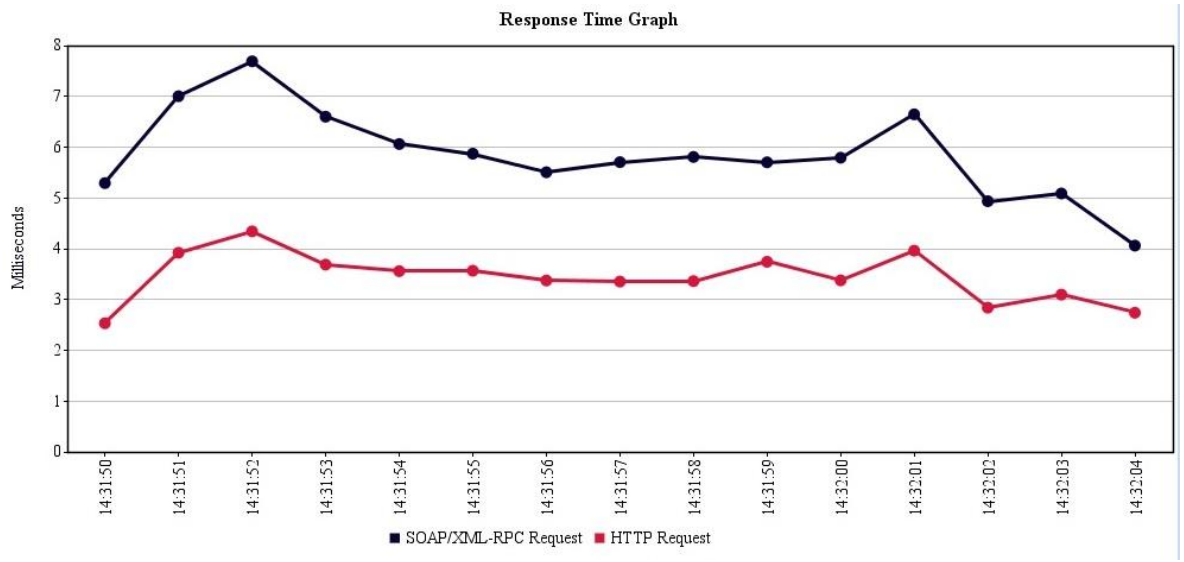
Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
SOAP/XML-RP...	20000	5	5	7	1	272	0.03%	1327.8/sec	2954.5
HTTP Request	20000	3	3	4	1	279	0.00%	1407.0/sec	2808.7
TOTAL	40000	4	4	6	1	279	0.01%	2655.5/sec	5605.0

Εικόνα 78: Συγκεντρωτική συγκριτική αναφορά μεσαίας κίνησης displayName



Εικόνα 79: Συγκεντρωτικό συγκριτικό διάγραμμα μεσαίας κίνησης displayName



Εικόνα 80: Συγκεντρωτικό διάγραμμα χρόνου απόκρισης μεσαίας κίνησης displayName

Κατά τη διάρκεια της μέτρησης η απασχόληση της CPU έφτασε στιγμιαία το 43%. Επίσης η απασχόληση της μνήμης ήταν έως 54%.

Ε.5.2.3 Σενάριο Υψηλής Κίνησης (50 χρήστες – 1000 επαναλήψεις)

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
SOAP/XML-RP...	50000	15	2	317	15.45	0.08%	1851.8/sec	4118.73	2277.6
HTTP Request	50000	10	1	288	9.08	0.00%	2016.1/sec	4024.76	2044.2
TOTAL	100000	13	1	317	12.92	0.04%	3703.6/sec	7815.41	2160.9

Εικόνα 81: Συγκεντρωτική αναφορά υψηλής κίνησης displayName

Aggregate Report

Name: Aggregate Report

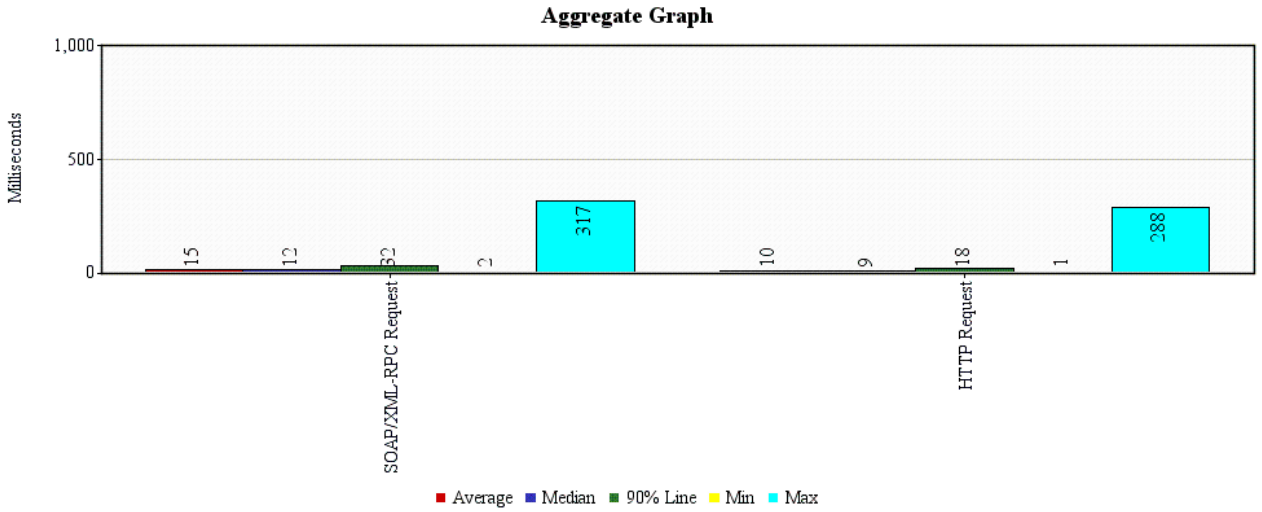
Comments:

Write results to file / Read from file

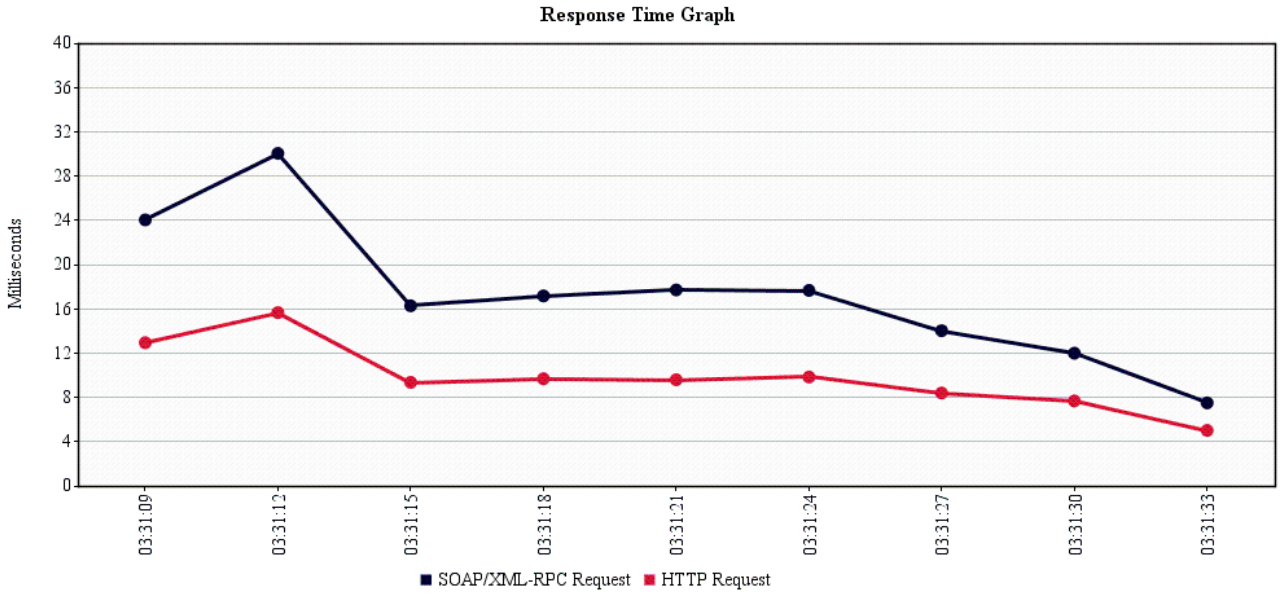
Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
SOAP/XML-RP...	50000	15	12	32	2	317	0.08%	1851.8/sec	4118.7
HTTP Request	50000	10	9	18	1	288	0.00%	2016.1/sec	4024.8
TOTAL	100000	13	10	22	1	317	0.04%	3703.6/sec	7815.4

Εικόνα 82: Συγκεντρωτική συγκριτική αναφορά υψηλής κίνησης displayName



Εικόνα 83: Συγκεντρωτικό συγκριτικό διάγραμμα υψηλής κίνησης displayName



Εικόνα 84: Συγκεντρωτικό διάγραμμα χρόνου απόκρισης υψηλής κίνησης displayName

Κατά τη διάρκεια της μέτρησης η απασχόληση της CPU έφτασε στιγμιαία το 52%. Επίσης η απασχόληση της μνήμης ήταν έως 55%.

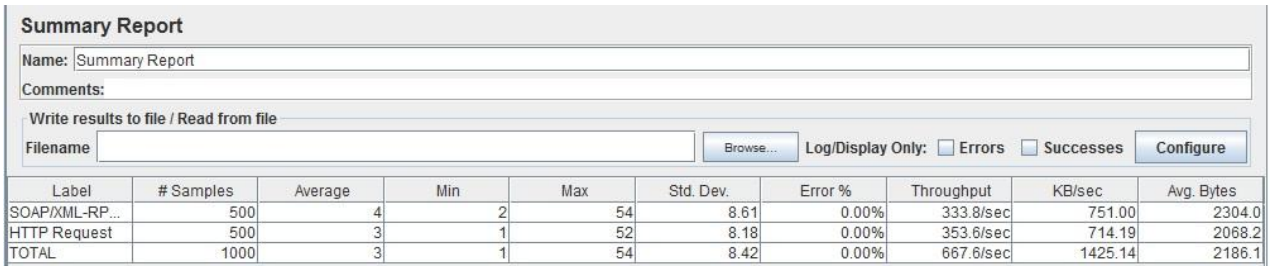
Ε.5.3 Μέθοδος countVowels

Εδώ θα εξεταστεί η μέθοδος countVowels, όπου αντί μίας απλής λέξης θα δίδεται το ίδιο μεγάλο κείμενο που δόθηκε στη μέθοδο displayName παραπάνω στην προσπάθεια να πιεστούν τα εξεταζόμενα services, τόσο κατά την αποστολή, όσο και κατά τη λήψη μεγάλων κειμένων χαρακτήρων, αλλά και κατά τη διάρκεια των υπολογισμών στο server. Το κείμενο που χρησιμοποιείται και εδώ περιέχει 573 φωνήεντα είναι:

«These excellant intentions were strengthed when he enterd the Father Superior's diniing-room, though, stricttly speakin, it was not a dining-room, for the Father Superior had only two rooms alltogether; they were, however, much larger and more comfortable than Father Zossima's. But tehre was was no great luxury about the furnishng of these rooms eithar. The furniture was of mohogany, covered with leather, in the old-fashionned style of 1820 the floor was not even stained, but evreything was shining with cleanlyness, and there were many chioce flowers in the windows; the most sumptuous thing in the room at the moment was, of course,

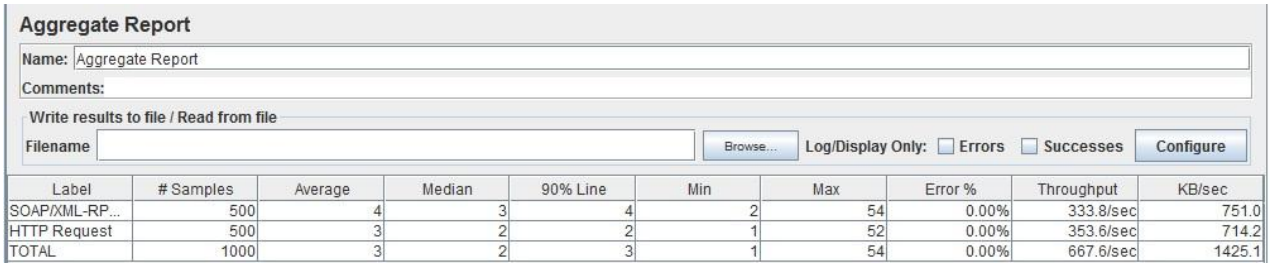
the beatifuly decorated table. The cloth was clean, the service shone; there were three kinds of well-baked bread, two bottles of wine, two of excellent mead, and a large glass jug of kvas -- both the latter made in the monastery, and famous in the neigborhood. There was no vodka. Rakitin related afterwards that there were five dishes: fish-suop made of sterlets, served with little fish paties; then boiled fish served in a spesial way; then salmon cutlets, ice pudding and compote, and finally, blanc-mange. Rakitin found out about all these good things, for he could not resist peeping into the kitchen, where he already had a footing. He had a footting everywhere, and got informaiton about everything. He was of an uneasy and envious temper. He was well aware of his own considerable abilities, and nervously exaggerated them in his self-conceit. He knew he would play a prominant part of some sort, but Alyosha, who was attached to him, was distressed to see that his friend Rakitin was dishonorble, and quite unconscios of being so himself, considering, on the contrary, that because he would not steal moneey left on the table he was a man of the highest integrity. Neither Alyosha nor anyone else could have infleunced him in that.»

Ε.5.3.1 Σενάριο Χαμηλής Κίνησης (5 χρήστες – 100 επαναλήψεις)



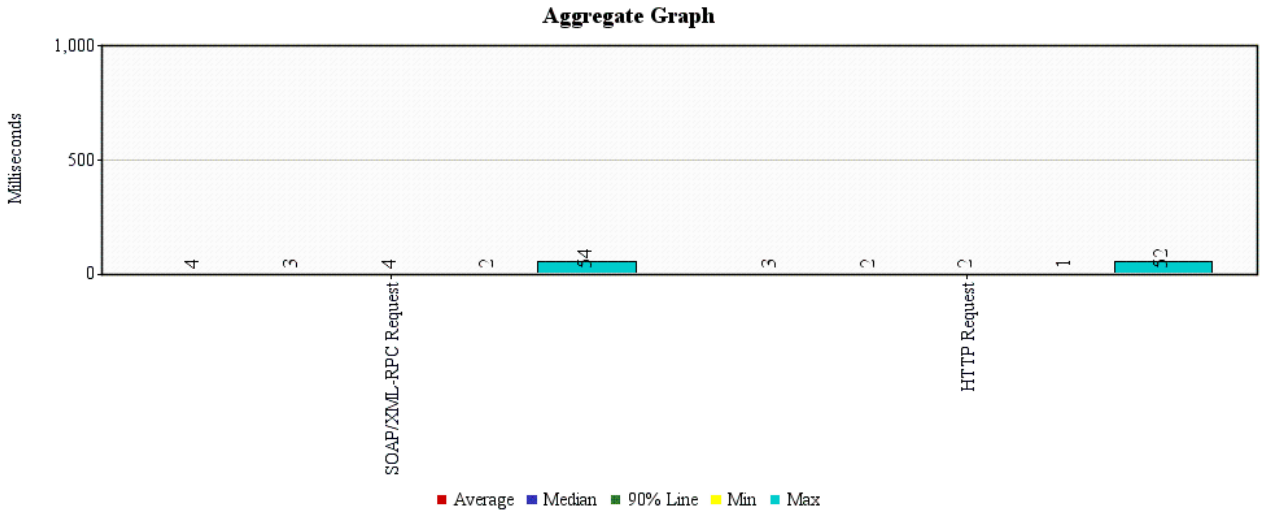
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
SOAP/XML-RP...	500	4	2	54	8.61	0.00%	333.8/sec	751.00	2304.0
HTTP Request	500	3	1	52	8.18	0.00%	353.6/sec	714.19	2068.2
TOTAL	1000	3	1	54	8.42	0.00%	667.6/sec	1425.14	2186.1

Εικόνα 85: Συγκεντρωτική αναφορά χαμηλής κίνησης countVowels

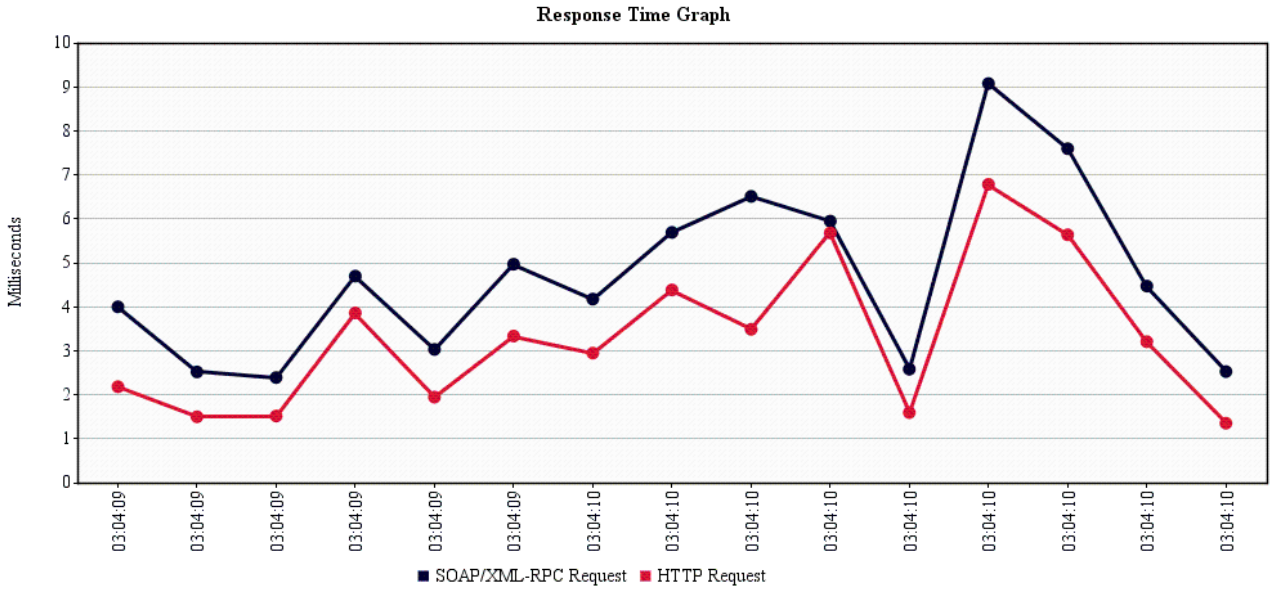


Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
SOAP/XML-RP...	500	4	3	4	2	54	0.00%	333.8/sec	751.0
HTTP Request	500	3	2	2	1	52	0.00%	353.6/sec	714.2
TOTAL	1000	3	2	3	1	54	0.00%	667.6/sec	1425.1

Εικόνα 86: Συγκεντρωτική συγκριτική αναφορά χαμηλής κίνησης countVowels



Εικόνα 87: Συγκεντρωτικό συγκριτικό διάγραμμα χαμηλής κίνησης countVowels



Εικόνα 88: Συγκεντρωτικό διάγραμμα χρόνου απόκρισης χαμηλής κίνησης countVowels

Κατά τη διάρκεια της μέτρησης η απασχόληση της CPU έφτασε στιγμιαία το 14%. Επίσης η απασχόληση της μνήμης ήταν έως 55%.

Ε.5.3.2 Σενάριο Μεσαίας Κίνησης (20 χρήστες – 500 επαναλήψεις)

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
SOAP/XML-RP...	20000	53	2	279	50.79	0.05%	326.5/sec	734.30	2303.1
HTTP Request	20000	50	1	271	51.10	0.00%	328.6/sec	663.71	2068.2
TOTAL	40000	52	1	279	50.96	0.03%	653.0/sec	1393.70	2185.6

Εικόνα 89: Συγκεντρωτική αναφορά μεσαίας κίνησης countVowels

Aggregate Report

Name: Aggregate Report

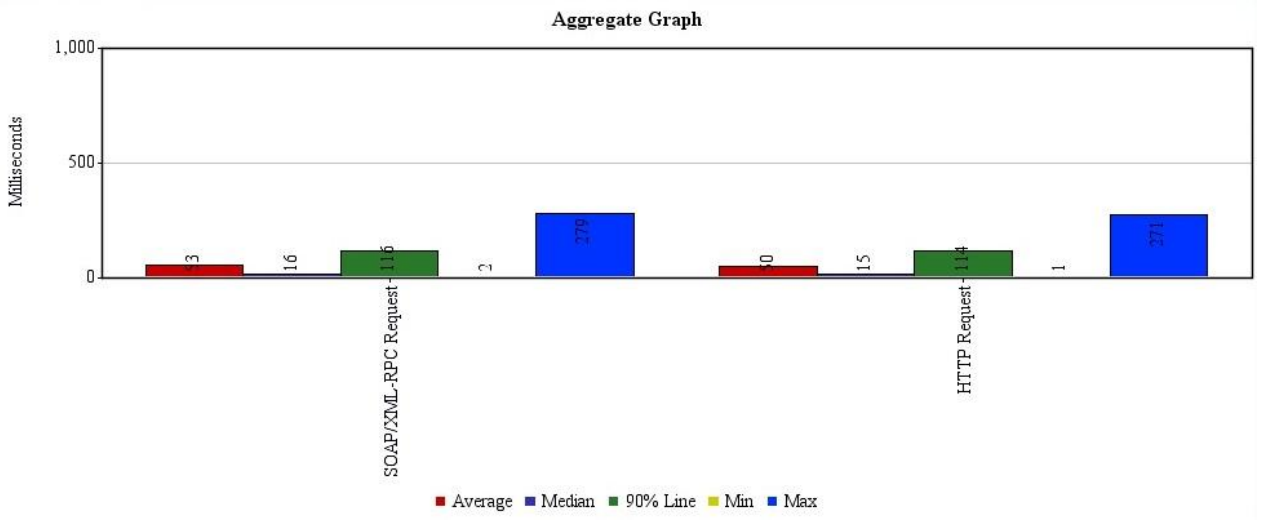
Comments:

Write results to file / Read from file

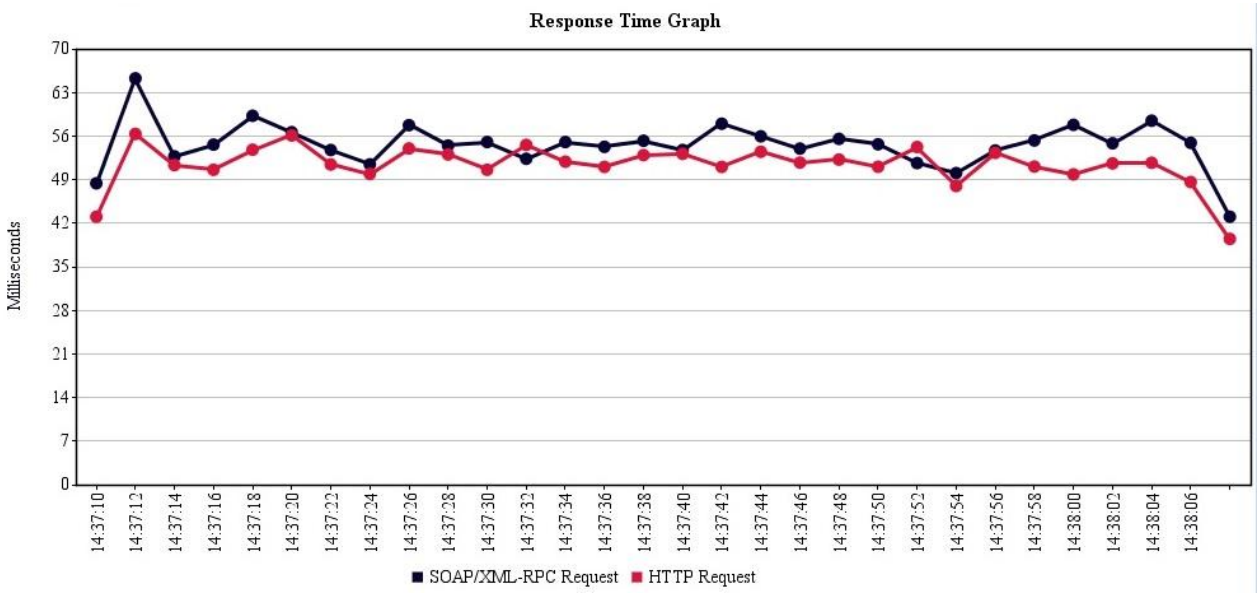
Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
SOAP/XML-RP...	20000	53	16	116	2	279	0.05%	326.5/sec	734.3
HTTP Request	20000	50	15	114	1	271	0.00%	328.6/sec	663.7
TOTAL	40000	52	15	115	1	279	0.03%	653.0/sec	1393.7

Εικόνα 90: Συγκεντρωτική συγκριτική αναφορά μεσαίας κίνησης countVowels



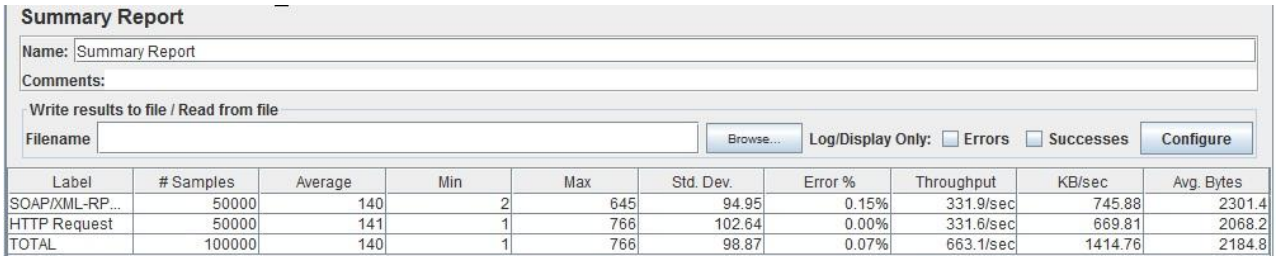
Εικόνα 91: Συγκεντρωτικό συγκριτικό διάγραμμα μεσαίας κίνησης countVowels



Εικόνα 92: Συγκεντρωτικό διάγραμμα χρόνου απόκρισης μεσαίας κίνησης countVowels

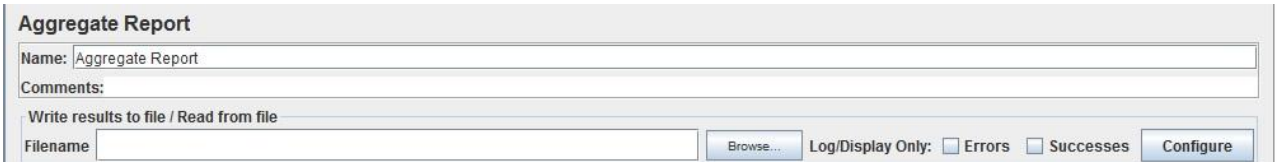
Κατά τη διάρκεια της μέτρησης η απασχόληση της CPU έφτασε στιγμιαία το 30%. Επίσης η απασχόληση της μνήμης ήταν έως 54%.

Ε.5.2.3 Σενάριο Υψηλής Κίνησης (50 χρήστες – 1000 επαναλήψεις)



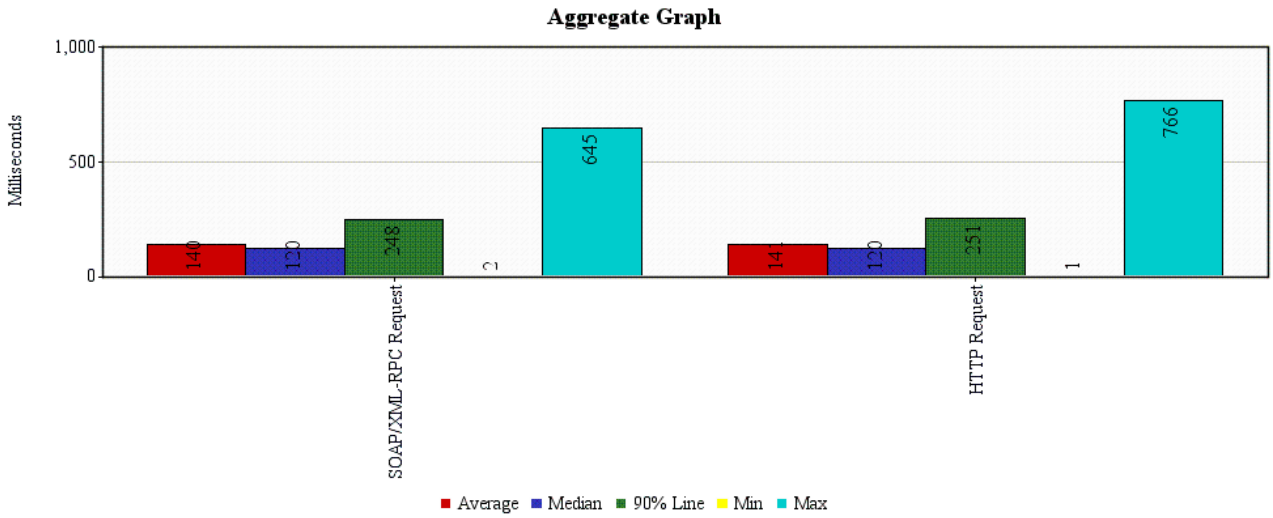
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
SOAP/XML-RP...	50000	140	2	645	94.95	0.15%	331.9/sec	745.88	2301.4
HTTP Request	50000	141	1	766	102.64	0.00%	331.6/sec	669.81	2068.2
TOTAL	100000	140	1	766	98.87	0.07%	663.1/sec	1414.76	2184.8

Εικόνα 93: Συγκεντρωτική αναφορά υψηλής κίνησης countVowels

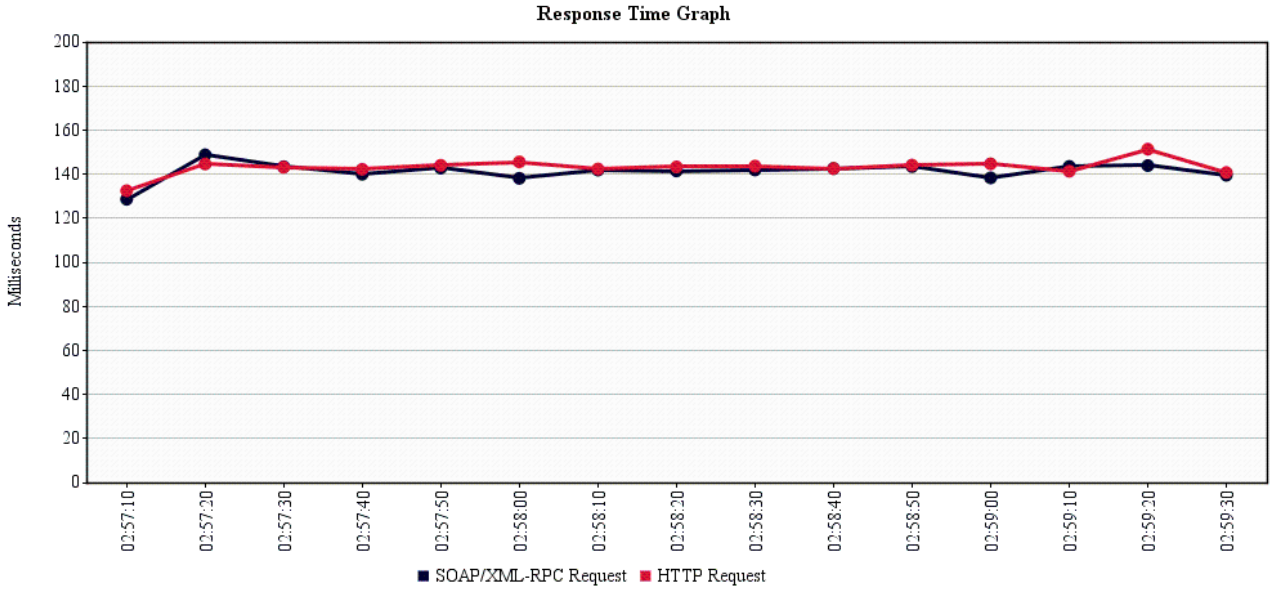


Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB/sec
SOAP/XML-RP...	50000	140	120	248	2	645	0.15%	331.9/sec	745.9
HTTP Request	50000	141	120	251	1	766	0.00%	331.6/sec	669.8
TOTAL	100000	140	120	250	1	766	0.07%	663.1/sec	1414.8

Εικόνα 94: Συγκεντρωτική συγκριτική αναφορά υψηλής κίνησης countVowels



Εικόνα 95: Συγκεντρωτικό συγκριτικό διάγραμμα υψηλής κίνησης countVowels



Εικόνα 96: Συγκεντρωτικό διάγραμμα χρόνου απόκρισης υψηλής κίνησης countVowels

Κατά τη διάρκεια της μέτρησης η απασχόληση της CPU έφτασε στιγμιαία το 73%. Επίσης η απασχόληση της μνήμης ήταν έως 55%.

Στην προσπάθεια να εκτελεστούν επιπλέον δοκιμές με ακόμα εξαντλητικότερα στοιχεία (περισσότερα threads ή/και περισσότερες επαναλήψεις) αντιμετωπίστηκαν προβλήματα υπερχείλισης μνήμης που οφείλονται στο μεγάλο όγκο δεδομένων που συγκεντρώνουν οι καταγραφείς (listeners).

Ε.6 Συμπεράσματα

Από τις ανωτέρω δοκιμές προκύπτουν τα ακόλουθα συμπεράσματα:

Ο μέσος αριθμός διακινούμενων Bytes στο SOAP είναι μεγαλύτερος κατά 200-300 Bytes ανά αίτηση (request) κατά μέσο όρο σε σχέση με εκείνον του REST, γεγονός που εξηγείται από το overhead που προσθέτει το envelope του SOAP ανά request.

Το REST εμφανίζεται ταχύτερο σε χρόνους απόκρισης σχεδόν σε όλες τις δοκιμές (πλην της τελευταίας), καθώς διακινεί λιγότερα δεδομένα από και προς τον Web Server.

Οι ανωτέρω διαφορές εξομαλύνονται στην εξεταζόμενη μέθοδο countVowels και οριακά αντιστρέφονται υπό συνθήκες υψηλής κίνησης. Το γεγονός αυτό μπορεί να εξηγηθεί από το πολύ μεγάλο κείμενο που δίδεται στο συγκεκριμένο σενάριο και κατά συνέπεια το μεγάλο χρόνο επεξεργασίας αυτού στον server για τον υπολογισμό του αριθμού των περιεχομένων φωνηέντων. Συνεπώς σε αυτή την περίπτωση το μειονέκτημα του SOAP αναφορικά με το επιπρόσθετο overhead κατά τη μετάδοση του πακέτου εκμηδενίζεται, καθώς ο μεγαλύτερος χρόνος σε αυτό το σενάριο δαπανάται στην επεξεργασία του περιεχομένου του request στο server.

Εν κατακλείδι το REST παρέχει σημαντικά καλύτερους χρόνους απόκρισης και είναι σκόπιμο να χρησιμοποιείται σε απλές υπηρεσίες όπου η ταχύτητα αποτελεί κρίσιμο συντελεστή αποδοτικότητας. Από την άλλη πλευρά όταν η υπηρεσία απαιτεί σημαντική επεξεργασία από τον server, αλλά και επιπλέον χαρακτηριστικά όπως επιπρόσθετη ασφάλεια, αυθεντικότητα και αξιοπιστία, εκτιμάται ως σκόπιμη η χρήση του SOAP.

Παράρτημα Α: Συναφείς Τεχνολογίες

E.1 HTML

Η HTML είναι ακρωνύμιο των λέξεων HyperText Markup Language και βασίζεται στη γλώσσα SGML. Χρησιμοποιείται για τη δόμηση των σελίδων του World Wide Web.

Το 1990 ο Tim Berners-Lee από το Cern, δημιούργησε ένα νέο πρωτόκολλο, το HTTP, με τη βοήθεια του οποίου μπορούσαν να μεταφέρονται αρχεία και αντικείμενα μέσα από το Διαδίκτυο. Οι σελίδες που ήταν η βάση του WWW ήταν γραμμένες στην πρώτη έκδοση της γλώσσα HTML.

Αρχικά είχε κατασκευαστεί για τη μορφοποίηση του κειμένου, αλλά στη συνέχεια επέτρεψε την ενσωμάτωση εικόνων, ήχου, video κ.λ.π. Για τη μορφοποίηση των παραπάνω χρησιμοποιεί έναν αριθμό από tag, τα οποία ο web browser, όταν διαβάζει μια σελίδα, τα αποκωδικοποιεί σε κατάλληλα χαρακτηριστικά, με αποτέλεσμα την ορθή εμφάνιση και λειτουργικότητά της (Εργαστήριο Εφαρμογών Πληροφορικής Στα ΜΜΕ, 2004)

E.2 XML

Η XML (eXtensible Markup Language) είναι μια γλώσσα markup η οποία μας βοηθά στην απεικόνιση των δεδομένων και δεν εξαρτάται από καμία πλατφόρμα. Δηλαδή, μας επιτρέπει να δημιουργήσουμε δεδομένα τα οποία μπορούν να διαβαστούν από οποιαδήποτε εφαρμογή, σε οποιαδήποτε πλατφόρμα είτε από τον άνθρωπο είτε υπολογιστική μηχανή. Καθορίζει ένα σύνολο κανόνων για την κωδικοποίηση των εγγράφων, σε τέτοια μορφή η οποία να είναι αναγνώσιμη, ή ακόμα και να μπορεί να δημιουργηθεί από τον άνθρωπο.

Η XML είναι η εξέλιξη των γλωσσών markup, οι οποίες στην πορεία τους δημιούργησαν την SGML (η οποία ήταν αρκετά περίπλοκη) και την HTML (η οποία ήταν μια γλώσσα περιγραφής σελίδας) (IBM, n.a). Τόσο η SGML όσο και η HTML

δεν μπορούσαν να περιγράψουν δεδομένα. Με την XML μπορούμε να δημιουργήσουμε τα δικά μας στοιχεία (element) με την βοήθεια των οποίων περιγράφουμε τα δεδομένα μας εσωκλείοντάς τα ανάμεσα σε tag.

Για παράδειγμα, έστω ότι θέλαμε να δημιουργήσουμε ένα βιβλίο συνταγών σε XML. Έστω ότι η πρώτη συνταγή μας αφορά τα βήματα δημιουργίας ενός παγωτού. Για να επισημάνουμε το όνομα της συνταγής το ενσωματώνουμε ανάμεσα σε element το οποίο μπορούμε να το ονομάσουμε recipename. Στη συνέχεια, για κάθε μια πληροφορία που θέλουμε να περιγράψουμε δημιουργούμε ένα element. Μπορούμε επίσης, να δημιουργήσουμε κανόνες για το ποια δεδομένα (π.χ. τύπος) μπορεί να περιέχει ένα element. Μπορούμε να συνεχίσουμε τη δημιουργία της XML συνταγής μας και να καταλήγουμε στην επόμενη:

```
<?xml version="1.0" encoding="UTF-8"?>

<recipe type="dessert">

<recipename cuisine="american" servings="1">Ice Cream Sundae</recipename>

<ingredlist>

<listitem><quantity units="cups">0.5</quantity>

<itemdescription>vanilla ice cream</itemdescription></listitem>

<listitem><quantity units="tablespoons">3</quantity>

<itemdescription>chocolate syrup or chocolate fudge</itemdescription></listitem>

<listitem><quantity units="tablespoons">1</quantity>

<itemdescription>nuts</itemdescription></listitem>

<listitem><quantity units="each">1</quantity>

<itemdescription>cherry</itemdescription></listitem>

</ingredlist>

<utensils>

<listitem><quantity units="each">1</quantity>
```

<utensilname>bowl</utensilname></listitem>

<listitem><quantity units="each">1</quantity>

<utensilname>spoons</utensilname></listitem>

<listitem><quantity units="each">1</quantity>

<utensilname>ice cream scoop</utensilname></listitem>

</utensils>

<directions>

<step>Using ice cream scoop, place vanilla ice cream into bowl.</step>

<step>Drizzle chocolate syrup or chocolate fudge over the ice cream.</step>

<step>Sprinkle nuts over the mound of chocolate and ice cream.</step>

<step>Place cherry on top of mound with stem pointing upward.</step>

<step>Serve.</step>

</directions>

<variations>

<option>Replace nuts with raisins.</option>

<option>Use chocolate ice cream instead of vanilla ice cream.</option>

</variations>

<preptime>5 minutes</preptime>

</recipe>

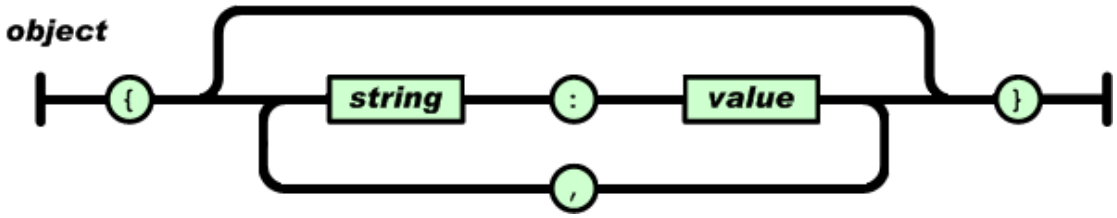
Η πρώτη γραμμή της παραπάνω XML περιγράφει την έκδοση της XML που χρησιμοποιούμε καθώς και την κωδικοποίηση των χαρακτήρων. Στη συνέχεια έχουν δημιουργηθεί μια σειρά από element και attribute (ιδιότητες των element) για να περιγράψουν τη συνταγή μας (Sconnard, 2003).

E.3 JSON

Το JSON (Javascript Object Notation) είναι ένα ελαφρύ πρότυπο ανταλλαγής δεδομένων, το οποίο βασίζεται σε ένα υποσύνολο της γλώσσας προγραμματισμού JavaScript. Το βασικό χαρακτηριστικό του είναι ότι ενώ χρησιμοποιεί πρακτικές που είναι γνωστές σε γλώσσες προγραμματισμού, είναι τελείως ανεξάρτητο από αυτές. Αυτό το χαρακτηριστικό, κάνει το JSON ιδανικό πρότυπο για ανταλλαγή δεδομένων.

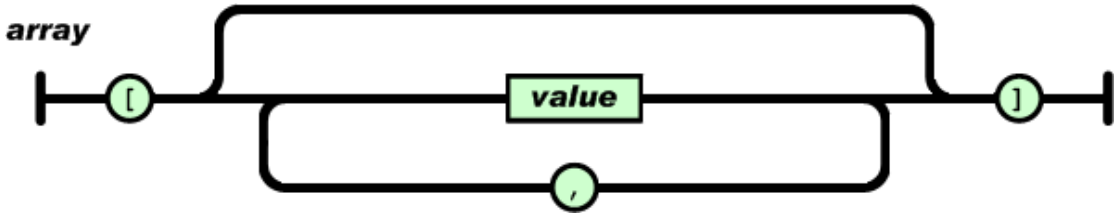
Το JSON περιέχει δύο δομές:

- Μια συλλογή από ζευγάρια όνομα και τιμή, που το αντίστοιχό της σε μια γλώσσα προγραμματισμού μπορεί να είναι ένα αντικείμενο. Η αρχή του δηλώνεται με αριστερό άγκιστρο ({) και το τέλος του με δεξί άγκιστρο (}). Ανάμεσά τους υπάρχουν ζευγάρια ονόματος τιμής που διαχωρίζονται με άνω κάτω τελεία (:). Τα ζευγάρια μεταξύ τους διαχωρίζονται με κόμμα(,).



Εικόνα 97: JSON Αντικείμενο (json.org, n.a)

- Μια ταξινομημένη λίστα τιμών, της οποίας το αντίστοιχο μπορεί να είναι ένας πίνακας. Η αρχή του δηλώνεται με αριστερή αγκύλη ([) και το τέλος του με δεξιά αγκύλη (]). Οι τιμές διαχωρίζονται με κόμμα.

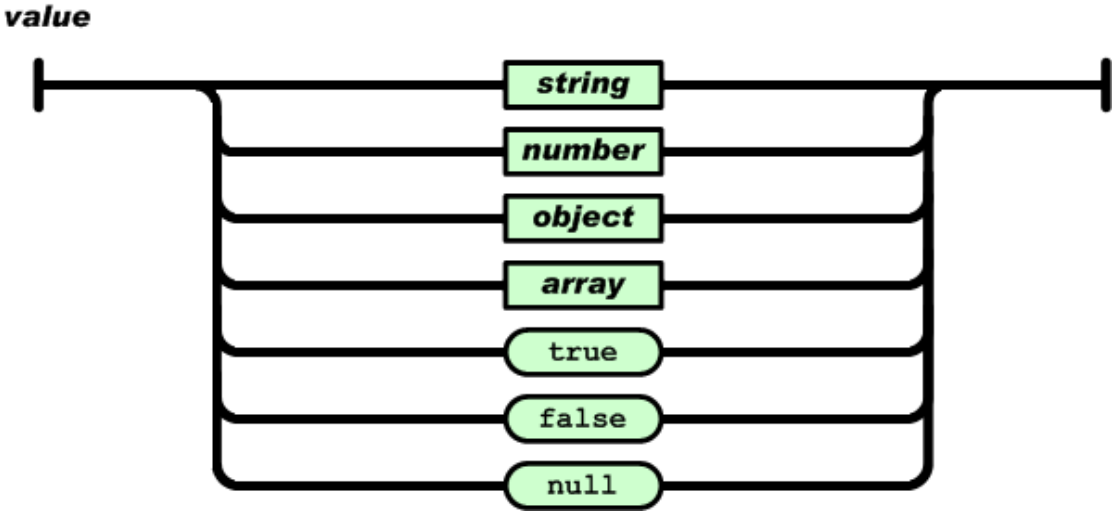


Εικόνα 98: JSON πίνακας (json.org, n.a)

Όπου αναφέρουμε value μπορεί να είναι:

- String ανάμεσα σε διπλά εισαγωγικά

- Αριθμός
- Boolean (True/False)
- Null
- Αντικείμενο
- Πίνακας (json.org, n.a)



Εικόνα 99: JSON τιμή (json.org, n.a)

Ας δούμε μερικά παραδείγματα:

- Πίνακας: *var myArray = ["John Doe", 29, true, null];*
- Πίνακας με αντικείμενα:

```
var myArray = [
 { "name": "John Doe", "age": 29 },
 { "name": "Anna Smith", "age": 24 },
 { "name": "Peter Jones", "age": 39 }
];
```

- Αντικείμενο:

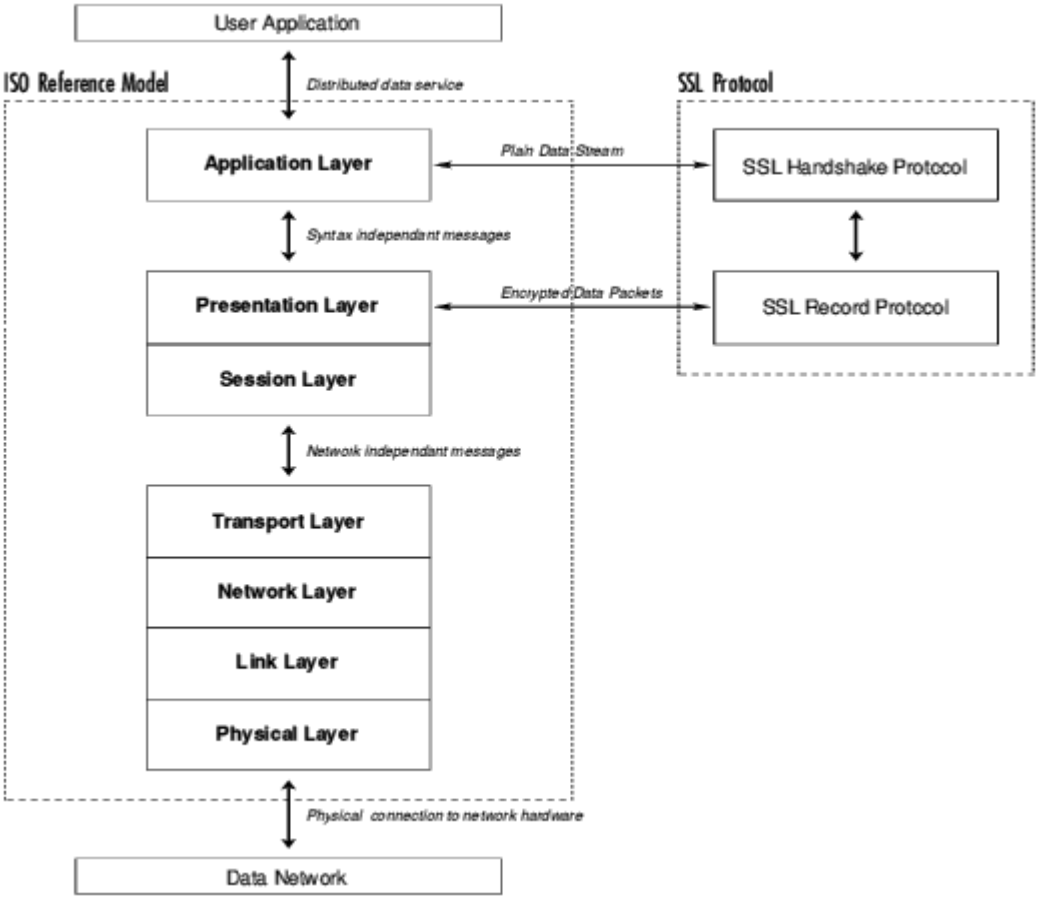
```
var myObject = {
 "first": "John",
 "last": "Doe",
 "age": 39,
 "sex": "M",
 "salary": 70000,
```

"registered": true

};(json.com, n.a)

E.4 SSL

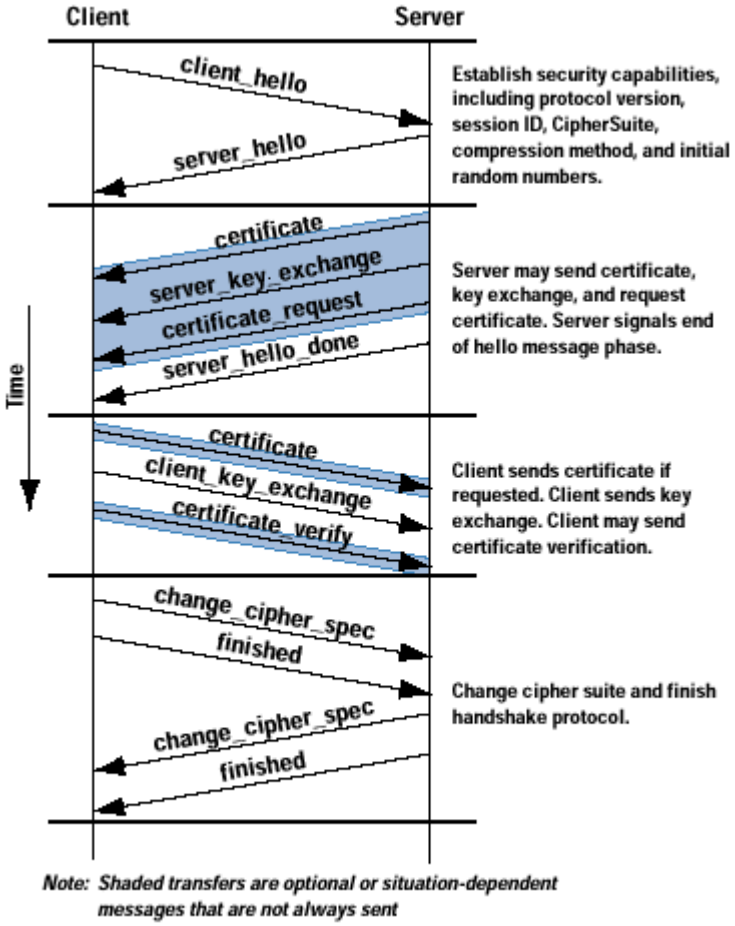
Το πρωτόκολλο SSL αναπτύχθηκε από τη Netscape Communications Corporation για να παρέχει απόρρητη επικοινωνία μεταξύ δύο συστημάτων. Σαν ένα πρωτόκολλο επικοινωνίας πρέπει να συμμορφώνεται με το μοντέλο OSI. Στην εικόνα που ακολουθεί βλέπουμε την αντιστοίχιση με το OSI μοντέλο.



Εικόνα 100: SSL και OSI (Μάγκος & Νιξαρλίδης, 1999)

Όπως μπορούμε να δούμε από την προηγούμενη εικόνα, το SSL χωρίζεται σε δύο μέρη:

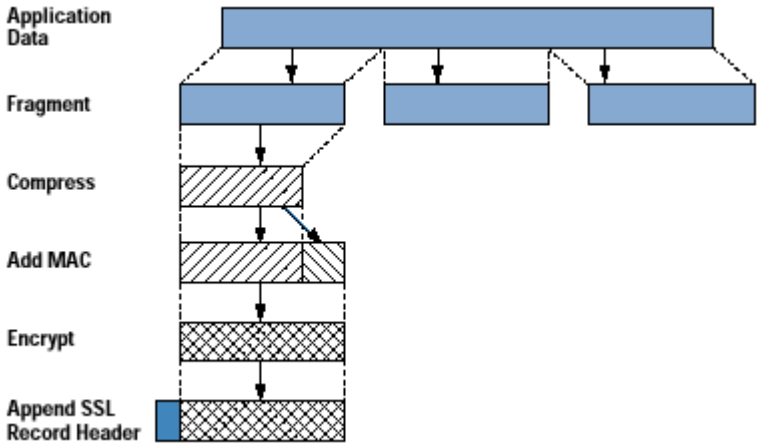
- το SSL Handshake Protocol (SSLHP) το οποίο ασχολείται με το Handshake για τους αλγόριθμους κρυπτογράφησης και του κώδικα επικύρωσης μηνυμάτων (Message Authentication Code, MAC) που θα χρησιμοποιηθούν, καθώς και με την πιστοποίηση της ταυτότητας του server (αν ζητηθεί από τον client). Στην εικόνα που ακολουθεί βλέπουμε τις φάσεις που ακολουθούνται για το handshake.



Εικόνα 101: Διαδικασία Handshake (Stallings, 1998)

- το SSL Record Protocol (SSLRP) το οποίο δημιουργεί τα πακέτα από τα προς μετάδοση δεδομένα, τα κρυπτογραφεί και τα μεταδίδει, ενώ ταυτόχρονα αποκρυπτογραφεί τα παραλαμβανόμενα πακέτα. Παρέχει δύο υπηρεσίες: εμπιστευτικότητα με την κρυπτογράφηση του μηνύματος και ακεραιότητα χρησιμοποιώντας τον κώδικα επικύρωσης μηνυμάτων (Message Authentication Code, MAC). Στην εικόνα που ακολουθεί μπορούμε να δούμε τη λειτουργία του SSLRP. Δέχεται το προς μετάδοση μήνυμα, το οποίο σπάει σε τμήματα, τα οποία

μπορεί να τα συμπιέσει, εφαρμόζει το MAC, κωδικοποιεί, προσθέτει την κεφαλίδα και το αποστέλλει. Στη λήψη κρυπτογραφημένων δεδομένων ακολουθείται η αντίστροφη διαδικασία (Stallings, 1998).



Εικόνα 102: Λειτουργία του SSL Record (Stallings, 1998)

E.5 WSDL

Καθώς τα πρωτόκολλα επικοινωνίας και οι μορφές μηνυμάτων τυποποιούνται, γίνεται απαραίτητη η ανάγκη να περιγραφεί και η επικοινωνία με κάποιο δομημένο τρόπο. Η WSDL (Web Services Description Language) έρχεται να καλύψει την ανάγκη αυτή, καθώς ορίζει μια XML γραμματική, για την περιγραφή των υπηρεσιών διαδικτύου, σαν μια συλλογή παραμέτρων επικοινωνίας. Ένα WSDL έγγραφο χρησιμοποιείται για να ορίσει web service και αποτελείται από ένα σύνολο από ορισμούς. Το element στη ρίζα είναι το definitions ενώ μέσα σε αυτό δηλώνονται τα ακόλουθα element:

- types: παρέχει τους ορισμούς των τύπων των δεδομένων που χρησιμοποιούνται στην ανταλλαγή των μηνυμάτων. Η δομή του είναι:

```
<types>
    <xsd:schema .... />*
</types>
```

Όπως βλέπουμε το WSDL προτιμά τη χρήση του xsd για τον καθορισμό του τύπου των δεδομένων.

- messages: απεικονίζει ένα αφηρημένο ορισμό των δεδομένων που μεταδίδονται. Μπορεί να αποτελείται από περισσότερα από ένα λογικά τμήματα, όπου το κάθε λογικό τμήμα σχετίζεται με ένα τύπο από κάποιο σύστημα τύπων, ακολουθώντας τη δομή element type που ακολουθεί:

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

όπου το element αναφέρεται σε ένα XSD element και το type αναφέρεται σε ένα XSD simple Type ή complex Type.

- portType: είναι ένα σύνολο από αφηρημένες λειτουργίες που μπορεί να εκτελέσει το web service. Το τμήμα δηλώσεων έχει τη μορφή:

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

Το attribute name καθορίζει ένα μοναδικό όνομα, ανάμεσα στα άλλα ορισμένα portType μέσα στο WSDL έγγραφο. Στο WSDL καθορίζονται τέσσερις τύποι λειτουργιών που μπορεί να υποστηρίξει ένα endpoint:

  - One-way. Το endpoint δέχεται μηνύματα, αλλά δεν επιστρέφει απαντήσεις. Η δομή είναι:

```
<wsdl:operation name="nmtoken">
    <wsdl:input name="nmtoken"? message="qname"/>
</wsdl:operation>
```

Όπου το element input καθορίζει την αφηρημένη μορφή του μηνύματος για τη λειτουργία της μιας κατεύθυνσης.

- Request-response. Το endpoint δέχεται μηνύματα και θα επιστρέψει απάντηση.

```
<wsdl:operation name="nmtoken" parameterOrder="nmtokens">
    <wsdl:input name="nmtoken"? message="qname"/>
    <wsdl:output name="nmtoken"? message="qname"/>
    <wsdl:fault name="nmtoken" message="qname"/>*
</wsdl:operation>
```

Όπου τα element input και output καθορίζουν την αφηρημένη μορφή του μηνύματος για το αίτημα και την απάντηση ενώ το element fault είναι προαιρετικό και περιέχει την αφηρημένη μορφή του μηνύματος για την περιγραφή μηνυμάτων σφάλματος.

- Solicit-response. Το endpoint στέλνει μήνυμα και περιμένει να λάβει απάντηση.

```
<wsdl:operation name="nmtoken" parameterOrder="nmtokens">
    <wsdl:output name="nmtoken"? message="qname"/>
    <wsdl:input name="nmtoken"? message="qname"/>
    <wsdl:fault name="nmtoken" message="qname"/>*
</wsdl:operation>
```

Όπου τα element output και input καθορίζουν την αφηρημένη μορφή του μηνύματος για το solicit αίτημα και την απάντηση ενώ το element fault είναι προαιρετικό και περιέχει την αφηρημένη μορφή του μηνύματος για την περιγραφή μηνυμάτων σφάλματος.

- Notification. Το endpoint στέλνει μήνυμα αλλά δεν περιμένει απάντηση.

```
<wsdl:operation name="nmtoken">
    <wsdl:output name="nmtoken"? message="qname"/>
</wsdl:operation>
```

Όπου το element output καθορίζει τη μορφή του αφηρημένου μηνύματος για το notification.

- binding: καθορίζει συγκεκριμένο πρωτόκολλο και προδιαγραφές που χρησιμοποιούνται από το portType. Μπορεί να υπάρχουν περισσότερα από ένα bindings για ένα portType. Η δομή του binding είναι:

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

```
</wsdl:operation>

</wsdl:binding>
```

Το attribute name χρησιμοποιείται για να αποδώσει στο binding ένα μοναδικό όνομα και αναφέρεται στο portType με τη βοήθεια του attribute type. Υπάρχουν δύο βασικοί κανόνες για το binding.

- Πρέπει να περιγράφει ακριβώς ένα πρωτόκολλο.
- Δεν πρέπει να καθορίζει διεύθυνση.

- port: καθορίζει μια διεύθυνση για τη διασύνδεση, ορίζοντας ένα τελικό σημείο επικοινωνίας. Η δομή του port είναι:

```
<wsdl:port name="nmtoken" binding="qname"> *

    <-- extensibility element (1) -->

</wsdl:port>
```

Το attribute port χρησιμοποιείται για να αποδώσει στο port ένα μοναδικό όνομα και το συνδέει με το binding μέσω του attribute binding. Και στο port υπάρχουν δύο κανόνες:

- Το port δεν πρέπει να καθορίζει περισσότερες από μια διευθύνσεις.
- Το port δεν πρέπει να έχει πληροφορίες για το binding εκτός από την πληροφορία της διεύθυνσης.

- service: χρησιμοποιείται για να περιέχει ένα σύνολο από σχετιζόμενα port. Η δομή του είναι:

```
<wsdl:service name="nmtoken"> *

    <wsdl:port .... />*

</wsdl:service>
```

Το attribute name χρησιμοποιείται για να δώσει ένα μοναδικό όνομα στο service.

Στη συνέχεια θα δούμε ένα παράδειγμα ενός WSDL εγγράφου:

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
       <schema targetNamespace="http://example.com/stockquote.xsd"
               xmlns="http://www.w3.org/2000/10/XMLSchema">
           <element name="TradePriceRequest">
              <complexType>
                  <all>
                      <element name="tickerSymbol" type="string"/>
                  </all>
              </complexType>
           </element>
           <element name="TradePrice">
              <complexType>
                  <all>
                      <element name="price" type="float"/>
                  </all>
              </complexType>
           </element>
       </schema>
    </types>

    <message name="GetLastTradePriceInput">
        <part name="body" element="xsd1:TradePriceRequest"/>
    </message>

    <message name="GetLastTradePriceOutput">
        <part name="body" element="xsd1:TradePrice"/>
    </message>

    <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
           <input message="tns:GetLastTradePriceInput"/>
           <output message="tns:GetLastTradePriceOutput"/>
        </operation>
    </portType>

    <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetLastTradePrice">
           <soap:operation
soapAction="http://example.com/GetLastTradePrice"/>
           <input>
               <soap:body use="literal"/>
           </input>
           <output>
               <soap:body use="literal"/>
           </output>
        </operation>
```

```
    </binding>

    <service name="StockQuoteService">
        <documentation>My first service</documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteBinding">
           <soap:address location="http://example.com/stockquote"/>
        </port>
    </service>

</definitions>
```

Όπως βλέπουμε στο παραπάνω έγγραφο, ορίζονται δύο μηνύματα το GetLastTradePriceInput και το GetLastTradePriceOutput. Και τα δύο μηνύματα αποτελούνται από ένα τμήμα με όνομα body, όπου όμως το πρώτο μήνυμα αναφέρεται στο XSD element TradePriceRequest, ενώ το δεύτερο στο XSD element TradePrice.

Στη συνέχεια, ορίζεται ένα portType με όνομα StockQuotePortType και με ένα operation το GetLastTradePrice, το οποίο υποστηρίζει τη λειτουργία request – response. ΤΟ μήνυμα εισόδου ονομάζεται GetLastTradePriceInput, ενώ το μήνυμα εξόδου GetLastTradePriceOutput.

Το binding έχει όνομα StockQuoteSoapBinding και συνδέεται με το παραπάνω ορισμένο portType. Κατόπιν, μέσω του element <soap:binding> ορίζεται ότι το πρωτόκολλο επικοινωνίας θα είναι το soap, ενώ το πρωτόκολλο μεταφοράς των μηνυμάτων soap θα είναι το http.

Τέλος με το element service δηλώνεται το όνομα του service ότι θα είναι StockQuoteService και δηλώνεται ότι η διεύθυνσή του θα είναι http://example.com/stockquote (Christensen et.al, 2001).

E.6 UDDI

Όπως αναφέρουν οι (Clement et.al., 2004) τα web service αποκτούν νόημα, όταν οι χρήστες μπορούν να τα βρουν και να τα χρησιμοποιήσουν. Το UDDI (Universal Description Discovery and Integration) είναι ένα σύνολο υπηρεσιών που εστιάζονται στην ανεύρεση:

- Επιχειρήσεων, οργανισμών και άλλων παρόχων web service

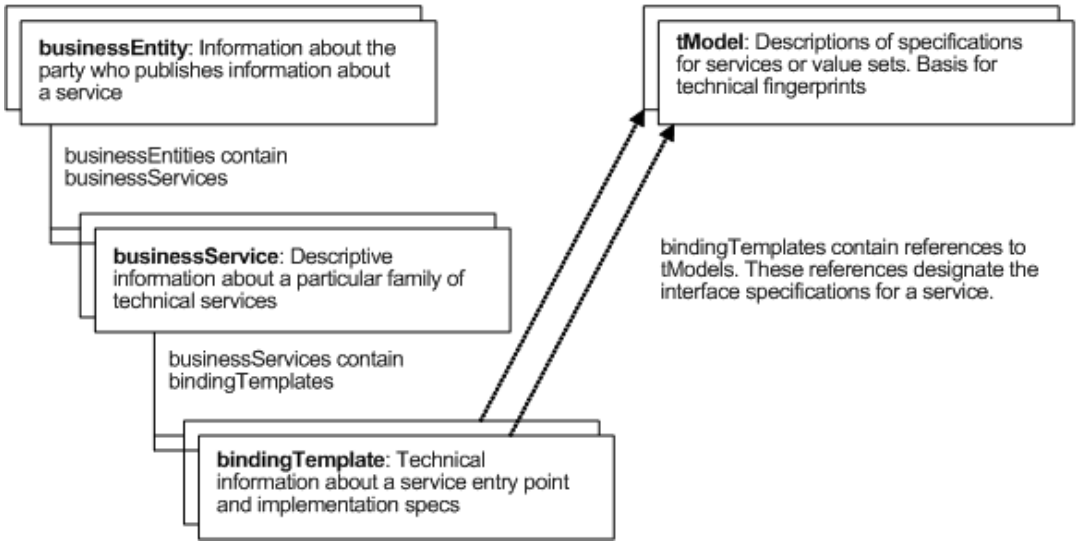
- Των web service που διαθέτουν και
- Των τεχνικών χαρακτηριστικών των διεπαφών που χρησιμοποιούνται για την απόκτηση πρόσβασης στις υπηρεσίες.

Επομένως, ο κεντρικός σκοπός του UDDI είναι η αναπαράσταση των δεδομένων καθώς και των μεταδεδομένων των web service, μέσω της δημιουργίας ενός μητρώου. Το μητρώο αυτό προσφέρει, ένα μηχανισμό για την ταξινόμηση, δημιουργία καταλόγου και τη διαχείριση των web service, έτσι ώστε να μπορούν εύκολα να ανακαλυφθούν και καταναλωθούν. Η ενεργή προδιαγραφή του UDDI είναι η 3.0.2 η οποία καθορίζει τον τρόπο που μπορεί κάποιος να δημοσιεύσει ή να αναζητήσει πληροφορίες χρησιμοποιώντας το UDDI (Oracle, 2010).

Η πληροφορία που υπάρχει εντός του UDDI αναπαρίσταται με τους ακόλουθους τέσσερις τύπους δεδομένων:

- businessEntity,
- businessService,
- bindingTemplate και
- tModel

Οι τέσσερις τύποι καθώς και οι μεταξύ τους σχέση απεικονίζεται στην επόμενη εικόνα:



Εικόνα 103: Δομή Δεδομένων UDDI (Oracle, 2010)

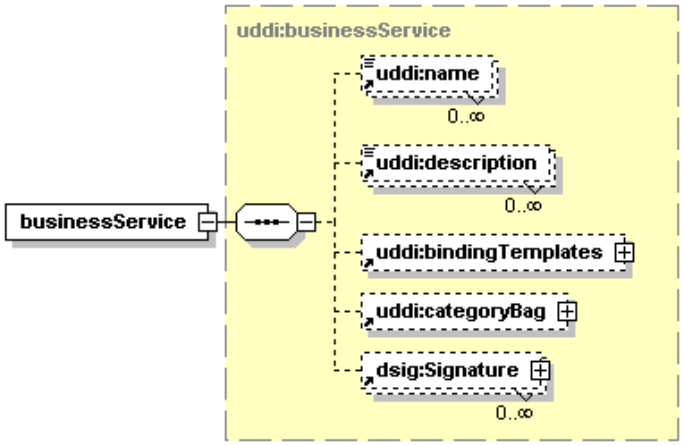
Η businessEntity περιέχει πληροφορίες για την επιχείρηση ή την ομάδα ανθρώπων που παρέχουν τις υπηρεσίες. Μια business Entity μπορεί να χαρακτηρισθεί:

- από ένα σύνολο από ονόματα,
- περιγραφές και στοιχεία επικοινωνίας του παρόχου της υπηρεσίας,
- κατηγορίες που καθορίζουν τα χαρακτηριστικά της business Entity,
- μοναδικά χαρακτηριστικά και
- τις URL τοποθεσίες των υπηρεσιών.

Όλα τα παραπάνω συνθέτουν τη δομή ενός businessEntity η οποία απεικονίζεται στην επόμενη εικόνα.

Εικόνα 104: Δομή businessEntity (Clement et.al., 2004)

Κάθε businessService αντιπροσωπεύει μια λογική ομαδοποίηση των web service και είναι λογικό παιδί ενός businessEntity. Περιέχει περιγραφικές πληροφορίες, όπως ονόματα – περιγραφές – πληροφορίες ταξινόμησης, τα οποία περιγράφουν το σκοπό των επιμέρους υπηρεσιών που βρίσκονται σ' αυτό. Για να γίνει περισσότερο κατανοητό, ένα businessService, θα μπορούσε να περιέχει ένα σύνολο από web service σχετικές με παραγγελίες (υποβολή, επιβεβαίωση, ενημέρωση) που παρέχονται από κάποια εταιρεία. Η δομή του businessService απεικονίζεται στην εικόνα που ακολουθεί:

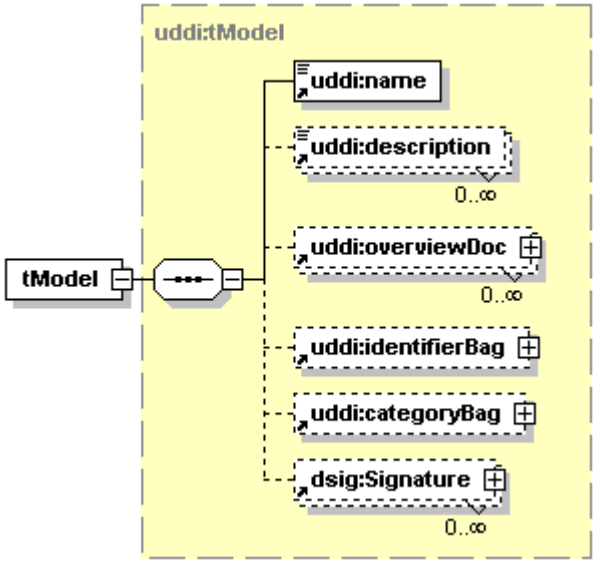


Εικόνα 105: Δομή businessService (Clement et.al., 2004)

Κάθε bindingTamplete, αντιπροσωπεύει ένα αυτόνομο web service. Σε αντίθεση με το businessService και businessEntity τα οποία είναι προσανατολισμένα σε βοηθητικές πληροφορίες σχετικά με τους παρόχους και τις υπηρεσίες, περιέχει τεχνικές πληροφορίες που χρειάζονται οι εφαρμογές για να συνδεθούν και να αλληλοεπιδράσουν με την υπηρεσία που περιγράφουν. Περιλαμβάνουν είτε το σημείο πρόσβασης για το web service είτε ένα μηχανισμό που οδηγεί στο σημείο πρόσβασης. Κάθε bindingTemplate είναι παιδί ενός businessService, το οποίο μπορεί να περιέχει μεταδεδομένα που επιτρέπουν την ανακάλυψη του bindingTemplate.

Τα tmodels (technical models) χρησιμοποιούνται για να προσδιορίσουν μοναδικές έννοιες ή δομές. Η δομή τους είναι τέτοια, ώστε να επιτρέπει την επαναχρησιμοποίησή τους, για το λόγο αυτό τα tModels στο παραπάνω σχήμα είναι έξω από τη σχέση γονέα – παιδιού που έχουν τα businessEntity, businessService και businessTemplate.

Η δομή ενός tModel απεικονίζεται στην επόμενη εικόνα:



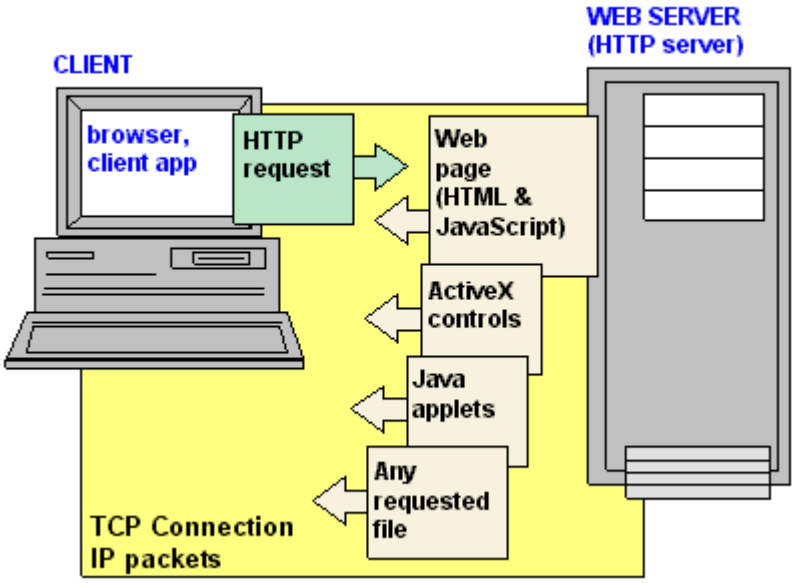
Εικόνα 106: Δομή tModel (Clement et.al., 2004)

Κάθε ξεχωριστό χαρακτηριστικό που αφορά transport, πρωτόκολλο, namespace, απεικονίζεται από ένα tModel. Παραδείγματα tModels που επιτρέπουν τη διαλειτουργικότητα των web service περιλαμβάνουν αυτά που βασίζονται στο WSDL, στο XSD (XML Schema Definition) και άλλα έγγραφα τα οποία περιγράφουν και καθορίζουν τη συμπεριφορά τους με την οποία έχει επιλέξει να συμμορφωθεί ένα web service. Για να μπορέσουν να περιγράψουν ότι μια υπηρεσία συμμορφώνεται με ένα σύνολο προδιαγραφών, τα tModels τοποθετούνται στο bindingTemplate. Τα bindingTemplate που αναφέρονται στα ίδια tModels λέμε ότι έχουν το ίδιο δαχτυλικό αποτύπωμα (Clement et.al., 2004).

E.7 HTTP

Το HyperText Transfer Protocol είναι ένα πρωτόκολλο επικοινωνίας που χρησιμοποιείται για τη διασύνδεση και επικοινωνία μεταξύ των web server και των φυλλομετρητών των χρηστών στο διαδίκτυο.

Το Http είναι stateless, δηλαδή δε διατηρεί καμία πληροφορία για μια σύνδεση μετά από τη διεκπεραίωση του αντίστοιχου αιτήματος.



Εικόνα 107: HTTP request - response (pcmag, n.a)

Όπως βλέπουμε από την προηγούμενη εικόνα το HTTP είναι ένα πρωτόκολλο request/response. Δηλαδή ο πελάτης εγκαθιστά μια σύνδεση με το server, και αποστέλλει το αίτημά του. Το αίτημά του περιλαμβάνει:

- Τη μέθοδο request που οι πιο συχνά χρησιμοποιούμενες είναι:
    - GET: για ανάκτηση πληροφορίας – αντικειμένου
    - HEAD: είναι παρόμοια με την GET μόνο που ο server επιστρέφει μόνο πληροφορίες για την οντότητα που ζητήσαμε και όχι την ίδια την οντότητα στο σώμα μηνύματος.
    - POST: για να ενημερώσει το server να δεχτεί μια οντότητα σαν ένα νέο στιγμιότυπο του αντικειμένου που περιγράφεται στο URI.
    - DELETE: για τη διαγραφή μιας οντότητας
- Ένα Universal Resource Identifier (URI) που δείχνει στον πόρο που θα εφαρμοστεί η μέθοδος
- Την έκδοση του πρωτοκόλλου

- Ένα μήνυμα στη μορφή MIME (Multipurpose Internet Mail Extensions), το οποίο περιέχει πληροφορία σχετικά με τον πελάτη κ.λπ.

Ο server απαντά με ένα μήνυμα που περιλαμβάνει:

- Μια status line που περιέχει την έκδοση του πρωτοκόλλου και τον κωδικό επιτυχίας ή αποτυχίας

- Ένα μήνυμα στη μορφή MIME που περιέχει πληροφορίες σχετικά με το server μεταπληροφορίες σχετικά με το αντικείμενο που αποστέλλεται κ.λπ. (Fielding et. al., 1999)

E.8 Eclipse

Η Eclipse έχει δημιουργηθεί από μια Open Source κοινότητα και είναι ένα ολοκληρωμένο περιβάλλον στο οποίο μπορούμε να γράψουμε και να εκτελέσουμε κώδικα για Java, C/C++ και PHP. Για την εκτέλεση ενός Java προγράμματος είναι απαραίτητη η εγκατάσταση του Java Runtime ή του Java Development Kit. Μπορούμε να αυξήσουμε τις δυνατότητες της Eclipse εγκαθιστώντας νέα plugin (Vogel, 2011).

E.9 Tomcat

Είναι ένας open source servlet container που έχει κατασκευαστεί από το Apache Software Foundation. Υλοποιεί τις προδιαγραφές Java Servlet και Java Server Pages και επιπλέον παρέχει και έναν Http web server για την εξυπηρέτηση απλών σελίδων (Apache Software Foundation, 2014)

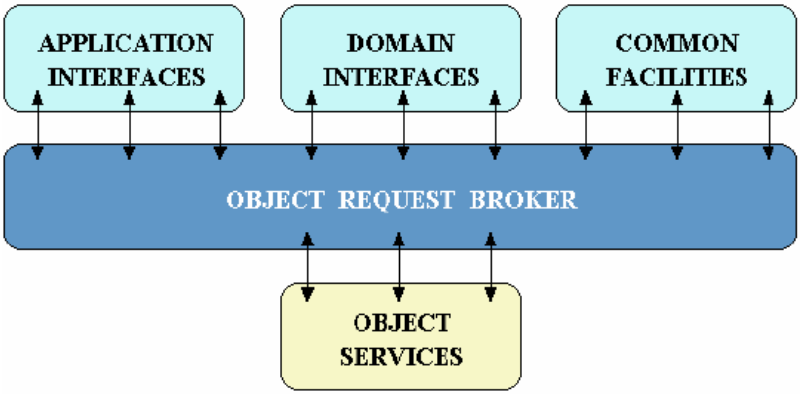
E.10 Παλαιότερες Τεχνολογίες Κατανεμημένων Συστημάτων

Οι σημαντικότερες παλαιότερες τεχνολογίες τις οποίες έχουν πλέον εκτοπίσει οι τεχνολογίες SOAP και REST, είναι η CORBA, η DCOM και η Java/RMI. Και οι τρεις αυτές τεχνολογίες, όπως αναφέρεται στη βιβλιογραφία, στηρίζονταν στην

προσανατολισμένη στο αντικείμενο αρχιτεκτονική (Thelin, 2003). Ακολουθεί μία περιγραφή των τεχνολογιών αυτών.

E.10.1 CORBA

CORBA – Common Object Request Broker Architecture, είναι μία ανοιχτή υποδομή κατανεμημένων αντικειμένων πληροφορικής που έχει τυποποιηθεί από την Object Management Group (OMG) και είναι μια προδιαγραφή που βασίζεται σε τεχνολογίες που προτείνονται (και εν μέρει παρέχονται) από τη βιομηχανία λογισμικού. Το CORBA υπήρξε το πλέον χρησιμοποιούμενο πρότυπο middleware στην αγορά εκτός περιβάλλοντος Windows. Η OMG ιδρύθηκε το 1989 για να προωθήσει την υιοθέτηση αντικειμενοστρεφούς τεχνολογίας και στοιχείων επαναχρησιμοποιούμενου λογισμικού. Θα ακολουθήσει μία παρουσίαση των στοιχείων που περιγράφονται στο Αρχιτεκτονικό Μοντέλο Αναφοράς (Reference Model Architecture- OMA) της OMG [Sørensen,2003] και φαίνεται στην ακόλουθη εικόνα.



Εικόνα 108: OMG Reference Model Architecture (Vinoski,1997)

Object Services. Είναι διεπαφές ανεξάρτητες από την τοποθεσία και χρησιμοποιούνται από πολλά προγράμματα κατανεμημένων αντικειμένων. Η OMG έχει καθορίσει υπηρεσίες αντικειμένων (object services) όπως η υπηρεσία ονοματοδοσίας η οποία επιτρέπει στους πελάτες να βρίσκουν αντικείμενα βασιζόμενοι σε ονόματα.

Υπάρχουν επίσης προδιαγραφές για διαχείριση κύκλου-ζωής (lifecycle management), ασφάλεια (security), συναλλαγές (transactions), επισήμανση

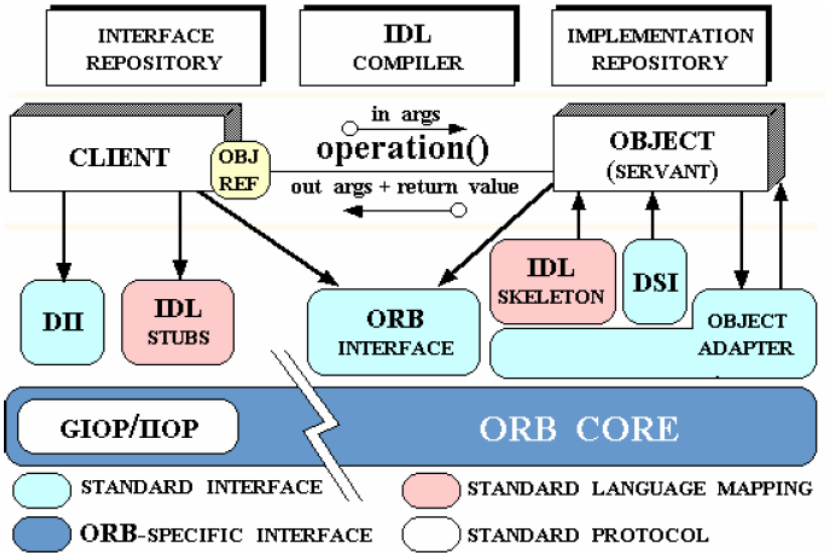
εκδηλώσεων (event notification), αναζήτηση αντικειμένων (object query) κ.α. Οι κατασκευαστές έχουν ωστόσο εγκαταλείψει μερικές από τις υπηρεσίες αυτές, επειδή συνήθως δεν είναι πρακτικές ή αποτελεσματικές.

Common Facilities. Είναι ευκολίες προσανατολισμένες προς τις εφαρμογές των τελικών χρηστών. Ένα παράδειγμα μίας κοινής ευκολίας είναι η Distributed Document Component Facility (DDCF). Η DDCF επιτρέπει την αναπαράσταση και ανταλλαγή αντικειμένων βασιζόμενη σε ένα μοντέλο εγγράφων, για παράδειγμα διευκολύνει την υπερσύνδεση (link) ενός αντικειμένου λογιστικού φύλλου (spreadsheet object) μέσα σε ένα έγγραφο κειμένου. Οι common facilities έχουν πλέον σχεδόν εγκαταλειφτεί ή αντικατασταθεί από νέες τεχνολογίες της βιομηχανίας.

Domain Interfaces. Οι διεπαφές περιοχής καλύπτουν τομείς παρεμφερείς με εκείνους των Object Services και Common Facilities αλλά είναι προσανατολισμένες σε συγκεκριμένες περιοχές εφαρμογών όπως οι κατασκευές, οι τηλεπικοινωνίες κ.α.

Application Interfaces. Οι διεπαφές εφαρμογών αναπτύσσονται εξειδικευμένα για κάθε συγκεκριμένη εφαρμογή.

Το **Object Request Broker** (ORB) είναι ο πυρήνας της αρχιτεκτονικής CORBA. Το ORB είναι ένας κεντρικός δίαυλος αντικειμένων όπου τα αντικείμενα CORBA αλληλεπιδρούν διαφανώς με άλλα τοπικά ή απομακρυσμένα αντικείμενα CORBA [Vinoski,1997]. Ο δίαυλος αυτός παρέχει βασικές ευκολίες ανταλλαγής μηνυμάτων, επικοινωνιών, καταλόγου, ασφάλειας και διαφάνειας τοποθεσίας και απαλλάσσει τις εφαρμογές από λεπτομέρειες σχετικά με το υλικό, το δίκτυο και το σύστημα. Το ORB απλουστεύει τον κατανεμημένο προγραμματισμό αποδεσμεύοντας τον πελάτη από λεπτομέρειες κλήσης μεθόδων. Αυτό κάνει τις αιτήσεις πελατών να εμφανίζονται σαν κλήσεις τοπικών διαδικασιών. Η επόμενη εικόνα παρουσιάζει την Αρχιτεκτονική CORBA ORB.



Εικόνα 109: CORBA ORB Architecture (Schmidt,2006)

Το **Internet Inter-ORB Protocol (IIOP)** αναπτύχθηκε με την προδιαγραφή CORBA 2.0 ως το πρωτόκολλο που θα χρησιμοποιείται για την επικοινωνία μεταξύ ORB διαφορετικών κατασκευαστών. Το IIOP είναι μία εξειδίκευση του General Inter-ORB Protocol (GIOP) και τρέχει πάνω από το TCP/IP. Το CORBA στηρίζεται στο IIOP ή άλλες εξειδικεύσεις του GIOP για πρόσβαση σε απομακρυσμένα αντικείμενα. Ένα αντικείμενο CORBA είναι μία οντότητα προγράμματος που αποτελείται από μία ταυτότητα, μία διεπαφή, και μία υλοποίηση, γνωστή ως Servant. Η servant είναι μία υλοποίηση οντότητας γλώσσας προγραμματισμού η οποία καθορίζει τις λειτουργίες που υποστηρίζει μία διεπαφή CORBA IDL [Schmidt,2006].

Η OMG έχει ορίσει την Interface Definition Language (IDL), η οποία δηλώνει τις διεπαφές και τις μεθόδους ενός εξυπηρετητή αντικειμένων CORBA. Κάθε αντικείμενο CORBA πρέπει να δηλώνεται με IDL. Ο μεταγλωττιστής IDL δημιουργεί στελέχη και σκελετούς, οι οποίοι λειτουργούν σαν κόλλα μεταξύ των εφαρμογών πελατών και εξυπηρετητών, αντίστοιχα, και του ORB. Η σύνταξη της CORBA IDL είναι παρεμφερής με εκείνη της C++, και περιλαμβάνει σημασιολογία για πολλαπλή κληρονομικότητα μέσω των διεπαφών και των εξαιρέσεων που ορίζονται από το χρήστη.

Όταν ένας πελάτης CORBA ζητάει μία υπηρεσία από έναν εξυπηρετητή αντικειμένων CORBA, ο ORB είναι υπεύθυνος για:

- Να ανακαλύψει την υλοποίηση του εξυπηρετητή αντικειμένων.
- Να προετοιμάσει το αντικείμενο για κλήσεις.
- Να παραδώσει την αναφορά του αντικειμένου πίσω στο αντικείμενο πελάτη.
- Να επικοινωνήσει τις αιτήσεις στο αντικείμενο του εξυπηρετητή.
- Να επιστρέψει τα αποτελέσματα του εξυπηρετητή πίσω στον πελάτη.

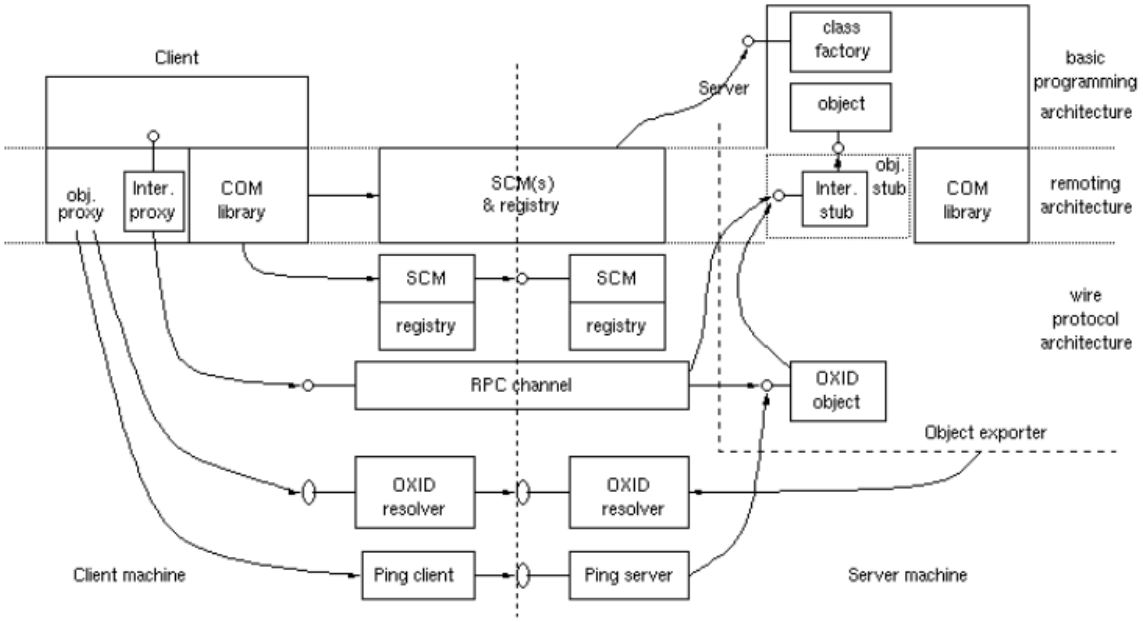
Οι αλληλεπιδράσεις γίνονται είτε μέσω της διεπαφής ORB ή μέσω ενός Object Adapter (Basic or Portable Object Adapter - POA). Η **διεπαφή (interface) ORB** είναι μία αφαιρετική διεπαφή του ORB για να αποσυνδέσει τις εφαρμογές από λεπτομέρειες υλοποίησης. Αυτή η διεπαφή παρέχει διάφορες βοηθητικές συναρτήσεις, όπως εκείνες για τη μετατροπή αναφορών αντικειμένων σε strings και το αντίθετο. Ο **προσαρμογέας Αντικειμένου (Object Adapter)** βοηθάει το ORB με την παράδοση αιτημάτων στα αντικείμενα και την ενεργοποίηση των αντικειμένων. Ο προσαρμογέας αντικειμένου συνδέει τις υλοποιήσεις των αντικειμένων με το ORB [Schmidt,2006].

E.10.2 DCOM

DCOM – Distributed Component Object Model, είναι ένα πρότυπο που έχει αναπτύξει η Microsoft και αποτελεί μία κατανεμημένη επέκταση του προτύπου COM [Microsoft]. Το πρότυπο COM βασίζεται στην ανάπτυξη της τεχνολογίας των σύνθετων εγγράφων για την ενσωμάτωση τμημάτων εγγράφου (για παράδειγμα λογιστικά φύλλα, εικόνες, παρουσιάσεις, επεξεργαστές κειμένου κλπ) που δημιουργούνται από διαφορετικές εφαρμογές των Windows. Το COM χρησιμοποιεί κατά βάση το μοντέλο τμηματικού λογισμικού (component software model) και κυριαρχεί στην αγορά των desktop.

Το Distributed Component Object Model (DCOM) υποστηρίζει απομακρυσμένα αντικείμενα εκτελούμενο πάνω από ένα πρωτόκολλο που λέγεται Object Remote Procedure Call (ORPC) [Raj,1998]. Το ORPC είναι κατασκευασμένο στην κορυφή του DCE/RPC και αλληλεπιδρά με τις run-time

υπηρεσίες του COM. Στην επόμενη εικόνα φαίνεται η αρχιτεκτονική του DCOM [Chung,1997].



Εικόνα 110: DCOM overall architecture (Chung,1997)

Το COM στηρίζεται σε πίνακες μεταβλητών διαδικασιών (procedure variables) (dispatch tables) που περιέχουν δείκτες συναρτήσεων οι οποίες αντιπροσωπεύουν τις διεπαφές. Αυτή η διάταξη μνήμης συμμορφώνεται με τη διάταξη του εικονικού πίνακα της C++ (vtable) και χρησιμοποιείται για στατικές κλήσεις αντικειμένων. Οι εξυπηρετητές αντικειμένων DCOM υποστηρίζουν πολλαπλές διεπαφές όπου η καθεμία αντιπροσωπεύει ένα διαφορετικό χαρακτηριστικό του αντικειμένου. Ένας πελάτης αποκτά ένα δείκτη σε μία από τις διεπαφές του διακομιστή και στη συνέχεια αρχίζει να καλεί τις δημοσιευμένες μεθόδους του διακομιστή μέσω του δείκτη διεπαφής όπως εάν το αντικείμενο βρισκόταν στο χώρο διευθύνσεων του πελάτη.

Η Microsoft Interface Definition Language, MIDL, χρησιμοποιείται για τον ορισμό διεπαφών για τα αντικείμενα DCOM (και RPC). Η MIDL είναι μία επέκταση της DCE (RPC) IDL και βασίζεται σε σύνταξη και τύπους μεταβλητών της γλώσσας C. Η MIDL υποστηρίζει 2 διαφορετικές μορφές περιγραφών διεπαφών, τη βασική διεπαφή του COM γνωστή ως IUnknown, και τη διεπαφή αυτοματισμού OLE, IDispatch. Κάθε αντικείμενο (D)COM πρέπει να υλοποιεί τη διεπαφή IUnknown. Η

διεπαφή IDispatch επεκτείνει τη διεπαφή IUnknown και αποτελεί της διεπαφή πύλη πρόσβασης σε πολλές άλλες διεπαφές [Grimes97]. Η MIDL αντιστοιχεί επίσης κλάσεις σε διεπαφές. Η IUnknown παρέχει μία στάνταρ μέθοδο QueryInterface() για πλοήγηση μεταξύ των διεπαφών. Επιπρόσθετα η διεπαφή IUnknown περιλαμβάνει 2 μεθόδους για μέτρηση αναφορών: AddRef() και Release().

Ο μεταγλωττιστής MIDL δημιουργεί **βιβλιοθήκες τύπων** (**type libraries)** (.tlb) οι οποίες αποθηκεύουν τις πληροφορίες τύπου των αντικειμένου. Η βιβλιοθήκη τύπων μπορεί να χρησιμοποιηθεί για δυναμική κλήση αντικειμένων που υλοποιούν τη διεπαφή IDispatch. Ένα αντικείμενο στην COM επιτρέπεται να υλοποιεί **διττές διεπαφές**, όπου μία διεπαφή υλοποιείται μέσω IDispatch και μέσω vtable, ταυτόχρονα.

Ένας Universally Unique Identifier (UUID) αναγνωρίζει μοναδικά κάθε κλάση (CLSID) και κάθε διεπαφή (IID) στην COM. Η MIDL δεν καθορίζει εξαιρέσεις ορισμένες από το χρήστη (user-defined exceptions). Αντίθετα η DCOM παρέχει στάνταρ εξαιρέσεις COM που επιστρέφονται από ένα στάνταρ κώδικα επιστροφής, τον HRESULT. Οι εξυπηρετητές αντικειμένων γίνονται διαθέσιμοι μέσω The server objects are made available μέσω του μητρώου του συστήματος, με την καταχώρηση του διακομιστή μεσολάβησης.

Η περιγραφή COM βρίσκεται στο δυαδικό επίπεδο. Αυτό επιτρέπει να γράφονται τα στοιχεία σε ποικιλία γλωσσών προγραμματισμού. Η πλατφόρμα υλικού πρέπει να υποστηρίζει υπηρεσίες COM για να μπορεί να παρέχει DCOM.

E.10.3 Java/RMI

Java/RMI – Java/Remote Method Invocation. Είναι ένα πρότυπο που αναπτύχθηκε από την JavaSoft. Η Java από μία απλή γλώσσα προγραμματισμού εξελίχθηκε σε τρείς βασικές και πλήρως συμβατές πλατφόρμες: την J2SE (Java 2 Standard Edition), την J2EE (Java 2 Enterprise Edition) και την J2ME (Java 2 Micro Edition). Η J2SE είναι η βασική γλώσσα προγραμματισμού και η εργαλειοθήκη για προγραμματισμό και ανάπτυξη στοιχείων. Η J2EE συμπληρώνει

την J2SE και είναι ένα σύνολο τεχνολογιών και στοιχείων για εμπορική και διαδικτυακή ανάπτυξη. Η J2ME χρησιμοποιείται για τη δημιουργία λογισμικού για ενσωματωμένα, κινητά συστήματα και άλλες μικρές συσκευές όπως τα Personal Digital Assistants (PDA) and τα κινητά τηλέφωνα.

Η RMI υποστηρίζει απομακρυσμένα αντικείμενα εκτελώντας ένα πρωτόκολλο που λέγεται Java Remote Method Protocol (JRMP). Η προτεραιοποίηση (serialization) των αντικειμένων χρησιμοποιείται ευρέως για να συγκεντρώσει και να διασπείρει τα αντικείμενα σαν ρεύματα. Οι πελάτες και οι εξυπηρετητές πρέπει αμφότεροι να είναι υλοποιημένοι σε Java για να είναι σε θέση να χρησιμοποιήσουν την προτεραιοποίηση των αντικειμένων. Ο Java εξυπηρετητής αντικειμένων ορίζει διεπαφές οι οποίες μπορεί να χρησιμοποιηθούν για να αποκτηθεί πρόσβαση στα αντικείμενα έξω από το τρέχον Java virtual machine (JVM) από κάποιο άλλο JVM το οποίο εκτελείται για παράδειγμα σε άλλη μηχανή. Μία εγγραφή μητρώου RMI σε έναν εξυπηρετητή διαθέτει πληροφορίες για τα διαθέσιμα αντικείμενα του εξυπηρετητή και παρέχει υπηρεσία ονοματοδοσίας για το RMI. Ένας πελάτης αποκτά πρόσβαση σε μία αναφορά εξυπηρετητή αντικειμένου μέσω της εγγραφής μητρώου RMI στον εξυπηρετητή και καλεί μεθόδους στον εξυπηρετητή αντικειμένων σαν να βρίσκονταν το αντικείμενο στην περιοχή διευθύνσεων του πελάτη. Τα αντικείμενα του εξυπηρετητή ονοματίζονται με χρήση URLs και ο πελάτης καλεί την αναφορά εξυπηρετητή αντικειμένου τοποθετώντας την κατάλληλη URL.

Όταν ένας πελάτης Java/RMI καλεί μία υπηρεσία από έναν εξυπηρετητή Java/RMI, εκτελεί τα ακόλουθα [Oracle,2010]:

- Ξεκινά μία σύνδεση με την απομακρυσμένη JVM που περιέχει το απομακρυσμένο αντικείμενο.
- Συγκεντρώνει τις παραμέτρους για το απομακρυσμένο JVM.
- Αναμένει τα αποτελέσματα από την κλήση της μεθόδου.
- Αποκωδικοποιεί την επιστρεφόμενη τιμή ή την επιστρεφόμενη εξαίρεση.
- Επιστρέφει την τιμή στον καλούντα.

Χρησιμοποιώντας προτεραιοποίηση των αντικειμένων, ο κώδικας και τα δεδομένα μπορούν να περάσουν ταυτόχρονα μεταξύ ενός εξυπηρετητή και ενός πελάτη. Αυτό επιτρέπει διαφορετικές υλοποιήσεις ενός αντικειμένου να τρέχουν ταυτόχρονα στη μηχανή του εξυπηρετητή και του πελάτη. Για να εξασφαλιστεί ότι ο κώδικας λαμβάνεται και μεταφορτώνεται ασφαλώς, το RMI παρέχει επιπλέον ασφάλεια.

Για να δηλωθεί απομακρυσμένη πρόσβαση στα αντικείμενα εξυπηρετητή στην Java, κάθε αντικείμενο εξυπηρετητή πρέπει να υλοποιεί τη διεπαφή java.rmi.Remote. Η κλάση java.rmi.server.RemoteObject και οι υποκλασεις αυτής, java.rmi.server.RemoteServer, java.rmi.server.UnicastRemoteObject και java.rmi.activation.Activatable παρέχουν τις RMI συναρτήσεις εξυπηρετητή. Η κλαση java.rmi.server.RemoteObject παρέχει υλοποιήσεις για τις java.lang.Object μεθόδους, hashCode, equals, και toString οι οποίες είναι κατάλληλες για απομακρυσμένα αντικείμενα. Οι κλασεις UnicastRemoteObject και Activatable παρέχουν μεθόδους που απαιτούνται για τη δημιουργία απομακρυσμένων αντικειμένων και τα καθιστούν διαθέσιμα σε απομακρυσμένους πελάτες. Οι υποκλάσεις αναγνωρίζουν τη σημασιολογία των απομακρυσμένων αναφορών, για παράδειγμα πότε ένας εξυπηρετητής είναι ένα απλό απομακρυσμένο αντικείμενο ή είναι ενεργοποιούμενο απομακρυσμένο αντικείμενο (εκτελείται όταν καλείται) [Oracle,2010]. Η κλαση java.rmi.server.UnicastRemoteObject καθορίζει ένα μονήρες (unicast) απομακρυσμένο αντικείμενο του οποίου οι αναφορές είναι έγκυρες μόνο όσο η διεργασία του εξυπηρετητή είναι ενεργή. Η κλάση java.rmi.activation.Activatable είναι μία αφαιρετική κλαση η οποία καθορίζει ένα ενεργοποιούμενο απομακρυσμένο αντικείμενο το οποίο ξεκινά να εκτελείται όταν οι απομακρυσμένες μέθοδοί του καλούνται και μπορεί να απενεργοποιήσει τον εαυτό του όταν απαιτηθεί [Oracle,2010].

Η αρχιτεκτονική Java/RMI μπορεί να χρησιμοποιηθεί σε μία ποικιλία πλατφορμών και λειτουργικών συστημάτων εφόσον διατίθεται υλοποίηση JVM για αυτές τις πλατφόρμες.

Βιβλιογραφία

Apache Software Foundation (2014), Apache Tomcat, Διαθέσιμο στην ηλεκτρονική διεύθυνση: http://tomcat.apache.org/

Christensen E., Curbera F., Meredith G. & Weerawarana S. (2001), Web Services Description Language (WSDL) 1.1, Διαθέσιμο στην ηλεκτρονική διεύθυνση: http://www.w3.org/TR/wsdl

Clement L., Hately. A, Riegen C. & Rogers T. (2004), UDDI Version 3.0.2 UDDI Spec Technical Committee Draft, Διαθέσιμο στην ηλεκτρονική διεύθυνση: http://uddi.org/pubs/uddi-v3.0.2-20041019.htm

Fielding R., Irvine UC., Gettys J, Mogul J.,FrystykH. Masinter L., Leach P. & T. Berners-Lee (1999), Hypertext Transfer Protocol -- HTTP/1.1, Διαθέσιμο στην ηλεκτρονική διεύθυνση: https://tools.ietf.org/html/rfc2616#section-9.2

json.com, (n.a), JSON (JavaScript Object Notation), Διαθέσιμο στην ηλεκτρονική διεύθυνση: http://www.json.com/

Json.org (n.a), Εισαγωγή στο JSON, Διαθέσιμο στην ηλεκτρονική διεύθυνση: http://json.org/json-el.html

IBM (n.a), New to XML, Διαθέσιμο στην ηλεκτρονική διεύθυνση: http://www.ibm.com/developerworks/xml/newto/

Oracle (2010), UDDI, Διαθέσιμο στην ηλεκτρονική διεύθυνση: http://docs.oracle.com/cd/E14571_01/doc.1111/e15867/uddi.htm

PcMag (n.a), Definition of:HTTP, Διαθέσιμο στην ηλεκτρονική διεύθυνση: http://www.pcmag.com/encyclopedia/term/44501/http

Skonnard A. (2003), Understanding XML Schema, Διαθέσιμο στην ηλεκτρονική διεύθυνση: http://msdn.microsoft.com/en-us/library/aa468557.aspx

Stallings W. (1998), SSL: Foundation for Web Security, Διαθέσιμο στην ηλεκτρονική διεύθυνση: http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_1-1/ssl.html

Vogel L. (2011), Eclipse IDE Tutorial, Διαθέσιμο στην ηλεκτρονική διεύθυνση: http://pdplab.it.uom.gr/teaching/sunjava/eclipse-java.html

Εργαστήριο Εφαρμογών Πληροφορικής Στα ΜΜΕ (2004), Εισαγωγή στην HTML, Διαθέσιμο στην ηλεκτρονική διεύθυνση: http://pacific.jour.auth.gr/html/

Μάγκος Κ. & ΝιξαρλίδηςΑ. (1999), Secure Socket Layer (SSL), Διαθέσιμο στην ηλεκτρονική διεύθυνση: http://www.islab.demokritos.gr/gr/html/ptixiakes/kostas-aris_ptyxiakh/Phtml/ssl.htm

Thelin J. (2003) A Comparison of Service-oriented, Resource-oriented, and Object-oriented Architecture Styles, Cape Clear Software Inc. Διαθέσιμο στην ηλεκτρονική διεύθυνση: http://www.omg.org/news/meetings/workshops/WebServEurope_Manual/03-1_Thelin.pdf

Sørensen C. F. (2003) Object-Oriented Systems: A Comparison of Distributed Object Technologies, Διαθέσιμο στην ηλεκτρονική διεύθυνση: http://www.idi.ntnu.no/grupper/su/courses/dif8901/Essay2001/essay2001-calle.pdf

Vinoski S. (1997), CORBA: Integrating diverse applications within distributed heterogeneous environments, in IEEE Communications, vol. 14, no. 2, Feb. 1997. http://www.cs.wustl.edu/~schmidt/PDF/vinoski.pdf

Schmidt D. (2006), Overview of CORBA, Διαθέσιμο στην ηλεκτρονική διεύθυνση: http://www.cs.wustl.edu/~schmidt/corba-overview.html

Microsoft (n.a.), Distributed Component Object Model, Διαθέσιμο στην ηλεκτρονική διεύθυνση: http://technet.microsoft.com/en-us/library/cc958799.aspx

Raj G.(1998): A Detailed Comparison of CORBA, DCOM and Java/RMI, Διαθέσιμο στην ηλεκτρονική διεύθυνση: http://www.i3s.unice.fr/~riveill/cours/car/98-comparison.html

Chung E., Huang Y., Yajni S.(1997). DCOM and CORBA Side by Side, Step by Step, and Layer by Layer, Διαθέσιμο στην ηλεκτρονική διεύθυνση: https://www.ece.cmu.edu/~ece749/docs/DCOMvsCORBA.pdf

Grimes R. (1997). Professional DCOM Programming. WROX Press Ltd. 1997.

Oracle (2010). Java Remote Method Invocation Specification, Διαθέσιμο στην ηλεκτρονική διεύθυνση: http://docs.oracle.com/javase/6/docs/platform/rmi/spec/rmiTOC.html