ΚΕΦΑΛΑΙΟ 3
ΥΠΗΡΕΣΙΕΣ ΕΠΙΚΟΙΝΩΝΙΑΣ ΚΑΙ ΕΚΤΈΛΕΣΗΣ ΜΕ JAVA

Σύνοψη

Αυτό το κεφάλαιο παρουσιάζει κρίσιμα θέματα και τεχνολογίες, απαραίτητα στην κατασκευή των κατανεμημένων συστημάτων (Κ.Σ.). Τα θέματα αυτά αφορούν στην επικοινωνία μεταξύ των απομακρυσμένων διεργασιών και στην απομακρυσμένη εκτέλεση των εντολών προγράμματος. Παρουσιάζονται οι υπηρεσίες ασύγχρονης επικοινωνίας μέσω μηνυμάτων, ως βασική μέθοδος επικοινωνίας στα Κ.Σ., καθώς και η απομακρυσμένη εκτέλεση διεργασιών. Προς τούτο εξετάζονται θέματα όπως: η (δια-)δικτύωση, τα πρωτόκολλα επικοινωνίας, οι υποδοχές (sockets), η απομακρυσμένη κλήση μεθόδων και η μεταβίβαση παραμέτρων σε αυτές, η σειριακοποίηση αντικειμένων, η κατασκευή απομακρυσμένων αντικειμένων, οι αναφορές προς τα αντικείμενα αυτά, και οι εξυπηρετητές αντικειμένων.. Τα παραπάνω εξετάζονται διεξοδικά μέσα από την παρουσίαση των κυρίαρχων σήμερα πλατφορμών και τεχνολογιών, όπως το Java RPC, το Java RMI, το CORBA και το DCOM, οι βασισμένες στο SOAP υπηρεσίες ιστού (Web Services) τα Restful Web Services και η γλώσσα σύνθεσης υπηρεσιών ιστού BPEL. Στα παραδείγματα που χρησιμοποιούνται γίνεται η χρήση της γλώσσας προγραμματισμού Java.

Προαπαιτούμενη γνώση

1) Δουληγέρης, Χ., Μητρόπουλος, Σ., 2015. Πληροφοριακά συστήματα στο διαδίκτυο. [ηλεκτρ. βιβλ.] Αθήνα: Σύνδεσμος Ελληνικών Ακαδημαϊκών Βιβλιοθηκών. Διαθέσιμο στο: http://hdl.handle.net/11419/3969
2) Savitch Walter (2015), Java, 7η Έκδοση, Εκδόσεις Α. Τζιόλα & Υιοί Α.Ε.
3) Shari Lawrence Pfleeger (2011), Τεχνολογία Λογισμικού Θεωρία και Πράξη, Έκδοση: 2η Αμερικανική, Εκδόσεις Κλειδάριθμος ΕΠΕ.
4) Βακάλη Α., Παπαμήτσιου Ζ. (2012), Πληροφοριακά Συστήματα Παγκοσμίου Ιστού, Έκδοση: 1η, Εκδόσεις Νέων Τεχνολογιών.
5) Else Lervik, Vegard B. Havdal (2004), Java Με UML: Αντικειμενοστραφής Σχεδίαση και Προγραμματισμός, Έκδοση: 1η, Εκδόσεις Κλειδάριθμος ΕΠΕ.

3.1. Ενδιάμεσο Λογισμικό στην Πράξη

Σε αυτό το κεφάλαιο θα μελετήσουμε θέματα για το ενδιάμεσο λογισμικό από την πλευρά των διαφόρων τεχνολογιών και τεχνοτροπιών υλοποίησης. Το ενδιάμεσο λογισμικό, όπως έχουμε ήδη αναφέρει, μας δίνει την δυνατότητα να κατασκευάζουμε κατανεμημένες εφαρμογές με τη χρήση κατανεμημένων αντικειμένων τα οποία μπορεί να τρέχουν σε διαφορετικής τεχνολογίας υπολογιστές και λειτουργικά συστήματα, ενώ είναι κατασκευασμένα με διαφορετικές γλώσσες προγραμματισμού. Η ανάγκη αλληλεπίδρασης των κατανεμημένων αντικειμένων – υπηρεσιών οδήγησε στην κατασκευή των πλατφορμών ενδιάμεσου λογισμικού. Παραδείγματα τέτοιων πλατφορμών αποτελούν τα:

- OMG CORBA (Object Management Group Common Object Request Broker Architecture),
- COM/DCOM (Microsoft Component Object Model - Distributed Component Object Model),
- Java RMI (Sun Java Remote Method Invocation)
- Web Services (SOAP και Restful).

Το ενδιάμεσο λογισμικό συνιστά ένα σύνολο από διεπαφές, υπηρεσίες και πρωτόκολλα, τα οποία «στέκονται» κάτω από τις εφαρμογές και παρέχουν υποστήριξη προς τις υπερκείμενες εφαρμογές. Μια βασική λειτουργικότητα του ενδιάμεσου λογισμικού είναι η επικοινωνία μεταξύ προγραμμάτων που βρίσκονται σε διαφορετικούς υπολογιστές, όπως π.χ. είναι η επικοινωνία μεταξύ πελάτη - εξυπηρετητή. Στην πραγματικότητα, η επικοινωνία αυτή είναι μεταξύ προγραμμάτων και απομακρυσμένων αντικειμένων λογισμικού. Οπότε ένα μοντέλο ενδιάμεσου λογισμικού προσφέρει τις διεπαφές εκείνες με όλους τους υποστηρικτικούς μηχανισμούς και τα σχετικά πρωτόκολλα, ώστε να καθίσταται εφικτή η επικοινωνία αυτή. Οι τέσσερις προαναφερθείσες τεχνολογίες εστιάζουν στην επικοινωνία, στην εύρεση και στην κλήση αντικειμένων, διάσπαρτων σε ένα σύνολο δικτυωμένων υπολογιστών.

Ας δώσουμε ένα παράδειγμα. Έστω ότι έχουμε ένα απομακρυσμένο (κατανεμημένο) αντικείμενο σε κάποιον εξυπηρετητή το οποίο υλοποιεί κάποια λειτουργικότητα η οποία είναι χρήσιμη ως υπηρεσία σε κάποια προγράμματα πελάτες. Μια τέτοια λειτουργικότητα θα μπορούσε να είναι αυτή μιας αριθμομηχανής (calculator). Σε αυτή την περίπτωση, το αντικείμενο εξυπηρετητής, θα πρέπει να είναι «ορατό» και ανιχνεύσιμο από τους πελάτες, ώστε να μπορούν να το προσπελάσουν. Επίσης, ο εξυπηρετητής θα πρέπει να είναι σε κατάσταση αναμονής, ώστε να δέχεται κλήσεις προσπέλασης από τους πελάτες μέσω δικτυακών απομακρυσμένων κλήσεων προκειμένου αυτός να προσφέρει τις υπηρεσίες του. Από την άλλη πλευρά, θα πρέπει να είναι γνωστό στον πελάτη, ο τρόπος με τον οποίο θα πρέπει να απευθύνει κλήσεις στο απομακρυσμένο αντικείμενο, δηλαδή στη ζητούμενη απομακρυσμένη υπηρεσία και θα πρέπει να είναι σε θέση να καταλάβει τα αποτελέσματα που του στέλνει ο εξυπηρετητής.

Για να επιτευχθούν τα παραπάνω χωρίς την χρήση τεχνολογιών και υπαρχόντων πλατφορμών λογισμικού απαιτείται πολύ προγραμματισμός, καθώς και προγραμματιστικές δεξιότητες. Στην κατεύθυνση να ελαχιστοποιηθεί η προγραμματιστική προσπάθεια, όλες οι απαιτούμενες δικτυακές κλήσεις θα μπορούσαν να απευθύνονται σε έναν αντιπρόσωπο (proxy) του απομακρυσμένου αντικειμένου, ο οποίος εγκαθίσταται στον πελάτη και αντιπροσωπεύει σε αυτόν το αντικείμενο εξυπηρετητής. Με αυτόν τον τρόπο, οι κλήσεις γίνονται τοπικά και στη συνέχεια μέσω της υποκείμενης πλατφόρμας πραγματοποιούνται οι κλήσεις προς το απομακρυσμένο αντικείμενο.

Τα προαναφερόμενα μοντέλα, στην κατεύθυνση παροχής κατάλληλων μηχανισμών υποστήριξης των παραπάνω διαδικασιών, βασίζονται με την ευρεία έννοια του όρου στη τεχνική της απομακρυσμένης κλήσης διαδικασίας (Remote Procedure Call - RPC), την οποία θα εξετάσουμε διεξοδικά στην επόμενη παράγραφο. Συνοπτικά, η όλη προσέγγιση είναι ότι ένας πελάτης κάνει μία τοπική κλήση στον υπολογιστή που βρίσκεται, προκειμένου αυτή να διαβιβασθεί στη συνέχεια σε ένα αντικείμενο, που είναι σε κάποιο απομακρυσμένο υπολογιστή.

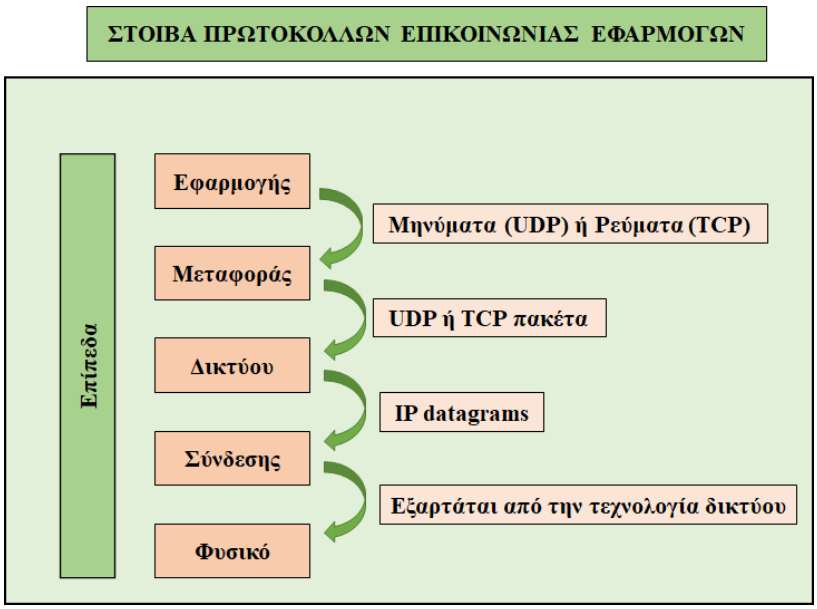
3.2. Απομακρυσμένη Κλήση Διαδικασιών (RPC)

Τα κατανεμημένα συστήματα για να λειτουργήσουν αυτονόητα απαιτούν την ύπαρξη δικτύου. Προκειμένου να επικοινωνεί ένα σημείο με ένα άλλο τελικό σημείο (π.χ. ένας υπολογιστής με έναν υπολογιστή) απαιτούνται πρωτόκολλα επικοινωνίας σε διάφορα επίπεδα. Τα επίπεδα αυτά κατά την στοίβα του ISO/OSI συνίστανται στο φυσικό επίπεδο που περιέχει τις προδιαγραφές για την αποστολή των διφύων (bits), και τη μετάδοσή τους μεταξύ αποστολέα και παραλήπτη, το επίπεδο σύνδεσης δεδομένων, το οποίο καθορίζει τη μετάδοση μιας σειράς διφύων σε ένα πλαίσιο για να επιτρέπεται έλεγχος σφαλμάτων και ροής, το επίπεδο δικτύου, το οποίο περιγράφει το πώς τα πακέτα σε ένα δίκτυο υπολογιστών πρέπει να δρομολογούνται μεταξύ των κόμβων δικτύου, το επίπεδο μεταφοράς, το οποίο παρέχει το κύριο υπόβαθρο επικοινωνίας για τα περισσότερα κατανεμημένα συστήματα, καθώς και τα επίπεδα, συνόδου, αναπαράστασης και εφαρμογής.

Τα κατανεμημένα συστήματα συνήθως απασχολούνται μέχρι το επίπεδο δικτύου και της διεπαφής μαζί του μέσω της χρήσης της διεύθυνσης IP, η οποία συνδυάζεται με τα πρωτόκολλα επιπέδου μεταφοράς προκειμένου να γίνεται αναφορά στις αναγκαίες υπηρεσίες. Τέτοια πρωτόκολλα είναι το TCP (Transmission Control Protocol), το οποίο είναι αξιόπιστο και προσανατολισμένο στη σύνδεση και τη ροή της πληροφορίας (ρεύμα δεδομένων), και το UDP (User Datagram Protocol), το οποίο είναι προσανατολισμένο στην αποστολή πακέτων δεδομένων και παρουσιάζει αναξιόπιστη επικοινωνία. Να σημειωθεί ότι ύπάρχει επέκταση του TCP και για συναλλαγές, το λεγόμενο T/TCP (TCP for Transactions). Επίσης, στο επίπεδο μεταφοράς είναι δυνατή και η πολλαπλή μετάδοση δεδομένων σε πολλαπλούς διαφορετικούς δέκτες (multicasting).

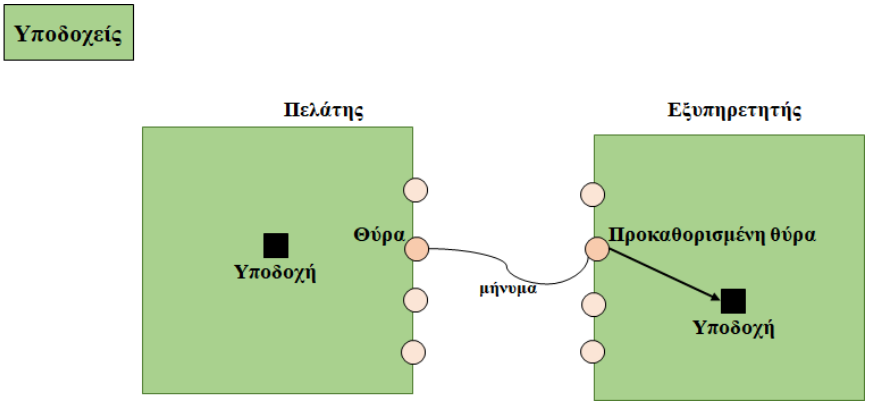
Τα πρωτόκολλα αυτά σκοπό έχουν την επίτευξη επικοινωνίας και αλληλεπίδρασης μεταξύ πελάτη και εξυπηρετητή (client – server) στο επίπεδο της εφαρμογής. Πολλά πρωτόκολλα επιπέδου εφαρμογής

υλοποιούνται κατευθείαν πάνω από τα πρωτόκολλα μεταφοράς (Σχήμα 3.1). Αυτό δημιουργεί μια ανεξαρτησία στο τι τρέχει πάνω από τα πρωτόκολλα μεταφοράς. Αυτό επιτυγχάνεται με τον μηχανισμό των θυρών (ports), ο οποίος, σε συνδυασμό με το πρωτόκολλο μεταφοράς, δημιουργεί μια αναφορά σε μια υπηρεσία εφαρμογής που τρέχει στον εκάστοτε υπολογιστή, όπως π.χ. είναι οι υπηρεσίες http, ftp, smtp, και nntp. Ο συνδυασμός θύρας και πρωτοκόλλου TCP/IP ή UDP/IP, δημιουργεί τους λεγόμενους υποδοχείς (sockets).



Σχήμα 3.1: Η στοίβα πρωτοκόλλων επικοινωνίας εφαμρογών

Το RPC σχετίζεται με την κλήση διαδικασιών – υπηρεσιών, που τρέχουν σε έναν απομακρυσμένο υπολογιστή και «ακούν» σε μία συγκεκριμένη θύρα πάνω από ένα συγκεκριμένο πρωτόκολλο (το TCP ή το UDP) (σχήμα 3.2). Ο απομακρυσμένος υπολογιστής προσδιορίζεται από τη διεύθυνσή του στο δίκτυο (IP). Η κλήση της απομακρυσμένης εφαρμογής προφανώς ενέχει και αυτή ένα πέρασμα παραμέτρων κατά την αίτηση και κατά την απόκριση της απομακρυσμένης διαδικασίας (remote procedure). Αυτό το μοντέλο επικοινωνίας στα κατανεμημένα συστήματα είναι πολύ γνωστό στο προγραμματιστές δεδομένου ότι οι απομακρυσμένες διαδικασίες καλούνται όπως οι τοπικές. Η μόνη διαφορά εδώ είναι ότι πρέπει ο προγραμματιστής να γνωρίζει την υποδοχή (socket) της υπηρεσίας στον απομακρυσμένο υπολογιστή, κάτι το οποίο όμως καταστρατηγεί την διαφάνεια τοποθεσίας που είναι ένα από τα χαρακτηριστικά που πρέπει να προσφέρουν τα κατανεμημένα συστήματα.
Ένα παράδειγμα όπου καλούνται απομακρυσμένα κάποιες υπηρεσίες, έστω και με μη προγραμματιστική διαφάνεια, είναι όταν υπάρχει ανάγκη να δημιουργηθεί ένα δίκτυο από εξυπηρετητές, οι οποίοι ο καθένας είναι εξειδικευμένος σε κάποια λειτουργία. Σε επίπεδο χρήστη, δεν γίνεται αντιληπτή η θέση της υπηρεσίας και άρα επιτυγχάνεται διαφάνεια τοποθεσίας.



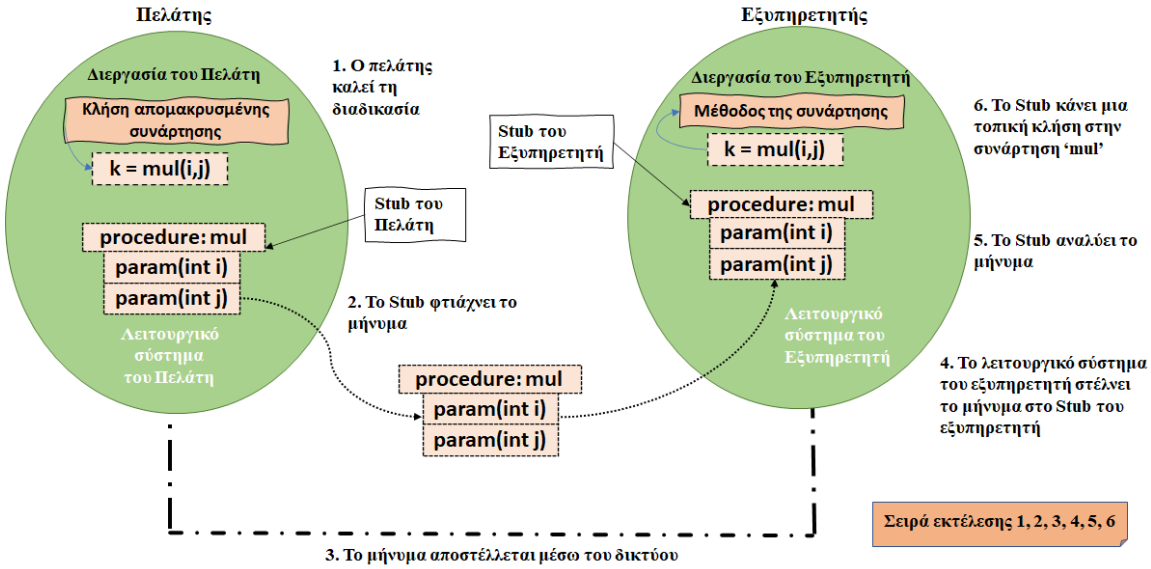
Σχήμα 3.2: Η λειτουργία των υποδοχέων (sockets)

Κατά την εκτέλεση RPC, προκειμένου να μπορέσει να συνεχίσει το πρόγραμμα από εκεί που είχε σταματήσει προκειμένου να τρέξει η απομακρυσμένη διαδικασία, απαιτείται οι τιμές των παραμέτρων και των μεταβλητών του εν εκτελέσει προγράμματος να τοποθετηθούν σε μια στοίβα (stack) μαζί και με άλλες πληροφορίες. Μετά την επιστροφή από τη διαδικασία επαναφέρονται οι τιμές των μεταβλητών, ανακτώνται και τα αποτελέσματα της κλήσης μέσω παραμέτρων.

Η κλήση των απομακρυσμένων διαδικασιών απαιτεί το πέρασμα κάποιων παραμέτρων. Δεδομένου όμως ότι ενδέχεται οι επικοινωνούντες υπολογιστές να έχουν διαφορετική αναπαράσταση των δεδομένων (data representation), θα πρέπει να υπάρχει η κατάλληλη μετατροπή κατά το αίτημα και κατά την απάντηση στην αναπαράσταση των δεδομένων. Αυτό το πρόβλημα επιλύεται με τον μηχανισμό της πρόταξης (marshalling) και της αποπρόταξης (unmarshalling) των μηνυμάτων (messages), μέσω των οποίων πραγματοποιείται η επικοινωνία των απομακρυσμένων συστημάτων. Η πρόταξη και η αποπρόταξη είναι μηχανισμοί που χρησιμοποιούνται και στους άλλους μηχανισμούς επικοινωνίας των κατανεμημένων συστημάτων. Οι επικοινωνούντες υπολογιστές, οι οποίοι, όπως είπαμε, μπορεί να έχουν διαφορετική αναπαράσταση των δεδομένων τους, θα πρέπει υποστηρίζουν έναν μηχανισμό που ανασυντάσσει τις μεταβλητές του ενός υπολογιστή στην υποστηριζόμενη αναπαράσταση του άλλου υπολογιστή. Επιπρόσθετα, επειδή το πέρασμα των παραμέτρων του αιτήματος από τον πελάτη καθώς και της απάντησης από τον εξυπηρετητή διοχετεύονται μέσω δικτύου, θα πρέπει οι παράμετροι αυτοί να σειριακοποιηθούν, δηλαδή να γίνουν μια σειρά από byte.

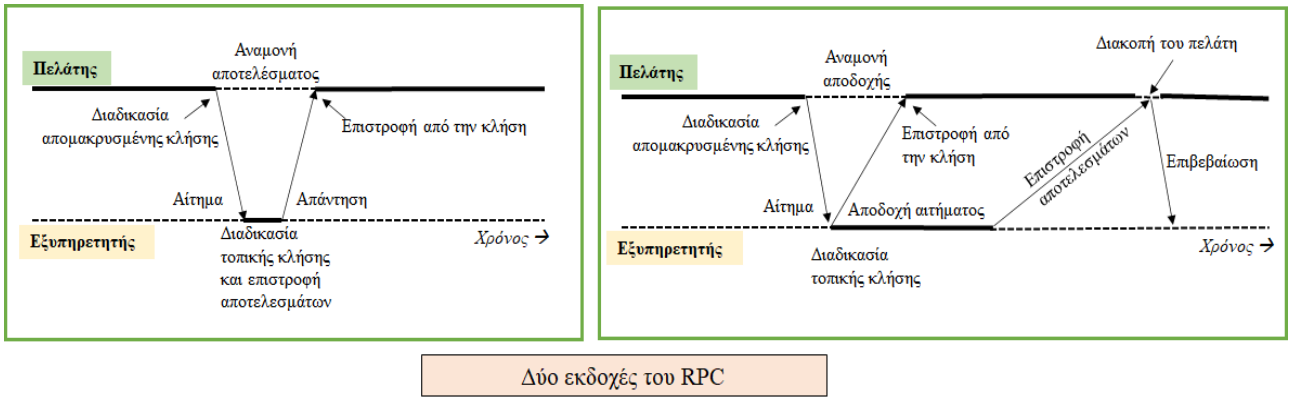
Σε κάθε περίπτωση, οι επικοινωνούντες υπολογιστές θα πρέπει να έχουν συμφωνήσει στη χρήση της ίδιας κωδικοποίησης σχετικά με τις τιμές των παραμέτρων που ανταλλάσσουν, όπως είναι οι συμβολοσειρές, οι ακέραιοι, οι πραγματικοί αριθμοί και οι διάφορες άλλες δομές δεδομένων, π.χ. ολόκληρα αντικείμενα. Θα πρέπει οι μεταφερόμενες παράμετροι μέσω των ανταλλασσόμενων μηνυμάτων να διαβάζονται από τον εκάστοτε υπολογιστή. Στο συγκεκριμένο μοντέλο επικοινωνίας υποθέτουμε ότι όλα τα ανταλλασσόμενα δεδομένα περνιούνται ως παράμετροι. Ενδεχόμενα να υπάρχουν εξαιρέσεις όταν δεν περνιέται μια τιμή παραμέτρου ή ένα ολόκληρο αντικείμενο, αλλά μια αναφορά προς κάποια καθολικά δεδομένα.

Κατά τη διαδικασία μιας απομακρυσμένης κλήσης, η τοπική κλήση γίνεται μέσω μιας διεπαφής, που ονομάζεται stub και η οποία λαμβάνει την κλήση του προγράμματος πελάτη, την οποία δομεί σε ένα μήνυμα σε κατάλληλη μορφοποίηση, το οποίο μεταφέρεται μέσω του δικτύου. Στην πλευρά του εξυπηρετητή υπάρχει μια αντίστοιχη διεπαφή, η οποία ονομάζεται skeleton. Η skeleton παραλαμβάνει το σειρικοποιημένο μήνυμα του πελάτη και αφού του κάνει συντακτική ανάλυση το αναδομεί στην κατάλληλη μορφή, ώστε να το προωθήσει στο αντικείμενο εξυπηρετητή. Η απάντηση του εξυπηρετητή προς το αντικείμενο πελάτη γίνεται με την αντίστροφη ακριβώς διαδικασία (σχήμα 3.3).



Σχήμα 3.3: Ο μηχανισμός επικοινωνίας πελάτη - εξυπηρετητή[1]

Να σημειωθεί ότι αναφορές σε υπηρεσίες εξυπηρετητή μέσω του μηχανισμού RPC μπορεί να λαμβάνει χώρα και στον καλούντα υπολογιστή, ο οποίος αναφέρεται ως localhost με IP: 127.0.0.1. Προφανώς σε αυτή την περίπτωση, η κλήση υπηρεσιών γίνεται σε τοπικές θύρες – υποδοχείς. Να αναφερθεί, επίσης, ότι ο μηχανισμός RPC δουλεύει ασύγχρονα. Αυτό σημαίνει ότι ο εξυπηρετητής μπορεί να λάβει την αίτηση του πελάτη σε κάποια μη καθορισμένη ώρα (δηλαδή μη σύγχρονα), όπως και ο πελάτης την απάντηση του εξυπηρετητή. Στο ενδιάμεσο χρονικό διάστημα ο πελάτης και ο εξυπηρετητής μπορούν να περιμένουν ή να εκτελούν άλλες εργασίες (Βλ. Σχήμα 3.4). Η χρησιμοποίηση του μηχανισμού RPC από έναν προγραμματιστή τον απαλλάσσει από την γνώση των υποκείμενων μηχανισμών επικοινωνίας κάτι που του διευκολύνει κατά πολύ τη δουλειά του. Βέβαια τα προγράμματα που αναπτύσσει θα πρέπει να περικλείουν τις απαραίτητες γεννήτριες και βιβλιοθήκες (libraries).



Σχήμα 3.4: Η επικοινωνία πελάτη – εξυπηρετητή με την τεχνική του RPC[1].

Στη συνέχεια δίνουμε ένα προγραμματιστικό παράδειγμα σε Java με χρήση του μηχανισμού RPC για την επικοινωνία πελάτη – εξυπηρετητή[2]. Ο κώδικας του πελάτη έχει ως εξής:

[1] Andrew S. Tanenbaum, Maarten Van Steen (2006), Κατανεμημένα Συστήματα: Αρχές και Υποδείγματα, Έκδοση: 1η/2006, Εκ. Κλειδάριθμος.
[2] Coulouris, J. Dollimore, T. Kindberg, G. Blair (2020), Κατανεμημένα Συστήματα, Έκδοση: 2η, DA VINCI M.E.Π.E.

```java
import java.net.*;
import java.io.*;

public class RPClient {
    public static void main(String args[]) {
        DatagramSocket ds = null;
        try {
            ds = new DatagramSocket();
            byte[] m = args[0].getBytes();
            InetAddress HostName = InetAddress.getByName(args[1]);
            int Port = 1234;
            DatagramPacket request = new DatagramPacket(m, m.length, HostName, Port);
            ds.send(request);
            byte[] buffer = new byte[1024];
            DatagramPacket response = new DatagramPacket(buffer, buffer.length);
            ds.receive(response);
            System.out.println("Response: " + new String(response.getData()));
        } catch (Exception e) {
            System.out.println("Exception: " + e.getMessage());
        } finally {
            if (ds != null) {
                ds.close();
            }
        }
    }
}
```

Ενώ ο κώδικας του εξυπηρετητή ως εξής:

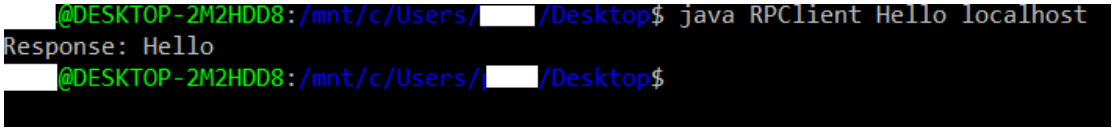
```java
import java.net.*;
import java.io.*;

public class RPCServer {
    public static void main(String args[]) {
        DatagramSocket ds = null;
        try {
            ds = new DatagramSocket(1234);
            byte[] buffer = new byte[1024];
            while(true) {
                DatagramPacket request = new DatagramPacket(buffer, buffer.length);
                ds.receive(request);
                DatagramPacket response = new DatagramPacket(request.getData(),
                request.getLength(), request.getAddress(), request.getPort());
                ds.send(response);
            }
        } catch (Exception e) {
            System.out.println("Exception: " + e.getMessage());
        } finally {
            if (ds != null) {
                ds.close();
            }
        }
    }
}
```

Για την εκτέλεση του εξυπηρετητή, με την εντολή javac RPCServer.java δημιουργούμε το αρχείο RPCServer.class, το οποίο στη συνέχεια τρέχουμε με χρήση της java RPCServer.class. Τα αρχεία με επέκταση .class είναι bytecode. Για την εκτέλεση στην πλευρά του πελάτη, αφού δημιουργήσουμε το .class, το τρέχουμε με παραμέτρους: (α) το μήνυμα που θέλουμε να αποστείλουμε (hello) και (β) τον εξυπηρετητή – host, στον οποίο θα αποσταλεί το μήνυμα. Ο εξυπηρετητής λαμβάνει το μήνυμα και το αποστέλλει πίσω στον πελάτη χωρίς να το επεξεργαστεί. Στο παραπάνω παράδειγμα θα μπορούσαν να συνδεθούν στον εξυπηρετητή πάνω από ένας πελάτες για την αποστολή μηνύματος. Η υπηρεσία αυτή

είναι μια υπηρεσία ηχούς (echo service), η οποία υλοποιείται με τον μηχανισμό RPC και με αποστολή πακέτων datagrams πάνω από το πρωτόκολλο UDP, τα οποία αποστέλλονται από τυχαία θύρα του πελάτη και από συγκεκριμένη σταθερή θύρα του εξυπηρετητή. Στις παρακάτω οθόνες εκτέλεσης (Εικόνα 3.1 και Εικόνα 3.2), ο εξυπηρετητής βρίσκεται στον ίδιο υπολογιστή (localhost) με τον πελάτη. Για λόγους συντομίας δεν θα επεκταθούμε στην επεξήγηση του κώδικα περαιτέρω.

Οι οθόνες που παράγονται μετά την εκτέλεση των προγραμμάτων έχουν ως εξής:

Εικόνα 3.1: Εκτέλεση εξυπηρετητή



Εικόνα 3.2: Εκτέλεση πελάτη

3.3. Απομακρυσμένη Κλήση Μεθόδων (RMI)

Τις τελευταίες δύο δεκαετίες, ο αντικειμενοστραφής κατανεμημένος υπολογισμός είναι η de-facto μέθοδος για την επικοινωνία συστημάτων που βασίζονται σε διαφορετικά, ετερογενή ή μη, μηχανήματα, ειδικά στα ενδοδίκτυα (Intranets) των επιχειρήσεων. Σήμερα, οι περισσότερες από τις επικοινωνίες μεταξύ διεργασιών υλοποιούνται με υπηρεσίες ιστού χρησιμοποιώντας ιδιόκτητες διεπαφές προγραμματισμού (Application Programming Interfaces – APIs), πολλές από τις οποίες βρίσκονται σε πλατφόρμες που βασίζονται στο υπολογιστικό νέφος (cloud). Στη συνέχεια θα εστιάσουμε στην αντικειμενοστραφή προσέγγιση.

Υπό τον όρο απομακρυσμένο αντικείμενο (remote object) εννοείται ένα στιγμιότυπο μιας κλάσης αντικειμένου, το οποίο φιλοξενείται από ένα πρόγραμμα σε έναν απομακρυσμένο υπολογιστή. Έτσι, όταν απαιτηθεί να κληθεί ένα απομακρυσμένο αντικείμενο, το πρόγραμμα στον καλών υπολογιστής οφείλει πρώτα να αναζητήσει που βρίσκεται το ζητούμενο αντικείμενο (υπηρεσία), να κατανοήσει τον τρόπο κλήσης του και τέλος να πραγματοποιήσει την κλήση του με τρόπο διαφανή προς τον χρήστη. Η απομακρυσμένη κλήση μεθόδων (Remote Method Invocation - RMI) είναι ένας μηχανισμός προκειμένου να πραγματοποιούμε κλήσεις προς τις μεθόδους ενός απομακρυσμένου αντικειμένου στην γλώσσα προγραμματισμού Java. Το RMI, το οποίο είναι μια τεχνολογία Java, αναπτύχθηκε για την επικοινωνία κατανεμημένων αντικειμένων στην Java. Υπάρχουν και άλλες πλατφόρμες κατανεμημένων αντικειμένων, όπως π.χ. το CORBA, στο οποίο τα αντικείμενα μπορεί να υλοποιηθούν και σε άλλες γλώσσες προγραμματισμού.

Όπως είναι κατανοητό, τα απομακρυσμένα αντικείμενα μπορεί να είναι διάσπαρτα σε διάφορους υπολογιστές, οι οποίοι συμμετέχουν σε ένα κατανεμημένο σύστημα. Και όπως έχουμε ήδη αναφέρει στο 2ο κεφάλαιο, ένα τμήμα κώδικα μπορεί να δέχεται μηνύματα αιτημάτων από τμήματα κώδικα τα οποία είναι εγκαταστημένα σε άλλους υπολογιστές με έναν τρόπο που είναι διαφανής και κάνει να φαίνεται η κλήση σαν να είναι τοπική, όπως μια κλήση του τύπου `accessDAO.updateValue(oldValue,newValue)` στην οποία δεν είναι εάν το αντικείμενο

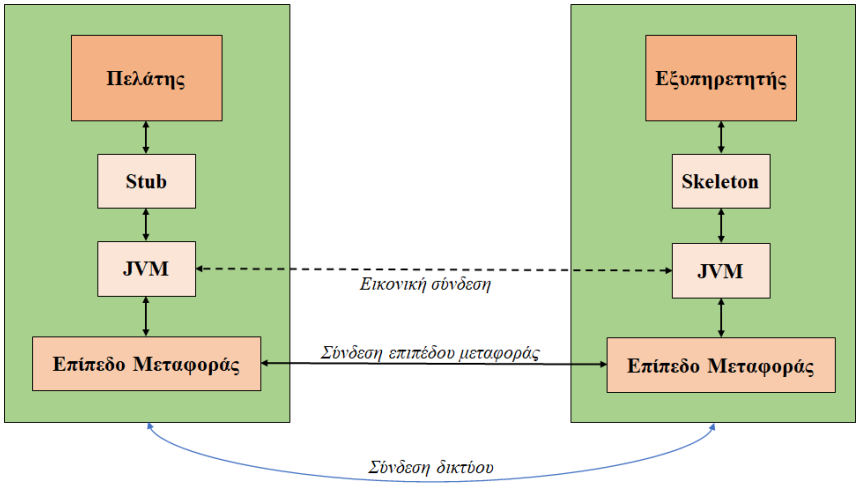
`accessDAO` βρίσκεται στον ίδιο υπολογιστή στον οποίο εκτελείται ο κώδικας ή σε έναν άλλον υπολογιστή.

Η Java μας παρέχει μια κατανεμημένη πλατφόρμα, την JVM (Java Virtual Machine), η οποία παρέχει την υποδομή ώστε ένα απομακρυσμένο αντικείμενο να καλείται με τη χρήση απομακρυσμένων κλήσεων μεθόδων. Οι απομακρυσμένες κλήσεις (remote invocation) περιγράφονται σε μία ή περισσότερες απομακρυσμένες διεπαφές (remote interfaces), οι οποίες προγραμματίζονται στη Java και οι οποίες δηλώνουν τις κλάσεις των αντικειμένων, καθώς και τις εμπεριεχόμενες σε αυτές μεθόδους. Αυτές οι μέθοδοι επί της ουσίας είναι υπηρεσίες που προσφέρει το απομακρυσμένο αντικείμενο.

Οι εφαρμογές Java RMI αποτελούνται από δύο προγράμματα, αυτά του πελάτη και του εξυπηρετητή. Το πρόγραμμα εξυπηρετητής είναι αυτό που δημιουργεί τα απομακρυσμένα αντικείμενα (remote objects) και καθορίζει τα δικαιώματα πρόσβασης σε αυτά. Ο εξυπηρετητής με την εκτέλεσή του δημιουργεί τα στιγμιότυπα των αντικειμένων και μπαίνει σε αναμονή προκειμένου να δεχθεί κλήσεις πάνω στις μεθόδους των αντικειμένων που φιλοξενεί και διατηρεί. Από την άλλη μεριά το πρόγραμμα πελάτης είναι αυτό που ενδιαφέρεται για την κλήση των μεθόδων του απομακρυσμένου αντικειμένου προκειμένου να λάβει συγκεκριμένα αποτελέσματα. Πριν γίνει η κλήση των απομακρυσμένων μεθόδων θα πρέπει να διασφαλιστεί η δυνατότητα πρόσβασης του πελάτη στο απομακρυσμένο αντικείμενο του εξυπηρετητή. Με άλλα λόγια το RMI είναι ένας πολύ χρήσιμος και ευέλικτος μηχανισμός επικοινωνίας μεταξύ απομακρυσμένων προγραμμάτων.

Οι πελάτες είναι διεργασίες (προγράμματα εν εκτελέσει), οι οποίες κάνουν κλήση των μεθόδων. Στη Java, όταν εκτελούμε κλήσεις RMI η επικοινωνία μεταξύ πελάτη και εξυπηρετητή γίνεται με ανταλλαγή μηνυμάτων μέσω του Java Virtual Machine (JVM),. Το μήνυμα εμπεριέχει μεταξύ άλλων τις παραμέτρους της απομακρυσμένης μεθόδου. Τη δημιουργία του μηνύματος σε κατάλληλη μορφή προς αποστολή, καθώς και την αποστολή αναλαμβάνει να κάνει μία ειδική κλάση που βρίσκεται στον πελάτη για τον σκοπό αυτό και καλείται stub (κορμός). Το stub είναι ο πληρεξούσιος του αντικειμένου στον πελάτη και είναι υπεύθυνο για την σειριακοποίηση του αιτήματος. Ο πελάτης κατεβάζει την κλάση stub από τον εξυπηρετητή στην περίπτωση που δεν την διαθέτει ήδη. Το μήνυμα αίτημα που στέλνει ο πελάτης το παραλαμβάνει ο εξυπηρετητής, όπου μία κλάση, η οποία ονομάζεται skeleton (σκελετός), αναλαμβάνει να αποσειριακοποιήσει τις παραμέτρους, καθώς επίσης να κάνει και την άμεση (τοπική) κλήση στο ζητούμενο αντικείμενο του εξυπηρετητή.

Όταν γίνεται μια κλήση RMI, ξεκινά μια σύνδεση μεταξύ της JVM του πελάτη με αυτή του εξυπηρετητή στον οποίο φιλοξενείται το απομακρυσμένο αντικείμενο. Κατόπιν και σύμφωνα με τη διεπαφή του απομακρυσμένου αντικειμένου συντάσσονται και μεταδίδονται οι παράμετροι στη JVM του εξυπηρετητή και αναμένεται το αποτέλεσμα της απομακρυσμένης κλήσης της μεθόδου. Όταν αυτό ληφθεί, διαβάζεται από το stub, η απάντηση που επιστρέφεται από την απομακρυσμένη κλήση και, τέλος, επιστρέφεται η τιμή στον αρχικό καλούντα. Παρατηρούμε ότι το stub είναι ένας απλός μηχανισμός κλήσης μεθόδων, ο οποίος επιτυγχάνει διαφάνεια επιπέδου δικτύου. Στον εξυπηρετητή, υπάρχει το αντίστοιχο skeleton του απομακρυσμένου αντικειμένου, το οποίο είναι υπεύθυνο για τη διαβίβαση της κλήσης σε αυτό. Το skeleton παραλαμβάνει την κλήση, διαβάζει τις παραμέτρους της μεθόδου, δημιουργεί την εντολή κλήσης της μεθόδου με την αποσειριακοποίηση του μηνύματος και πραγματοποιεί την τοπική κλήση στο αντικείμενο. Η διαδικασία αυτή ακολουθεί την αντίστροφη πορεία κατά την μετάδοση της απάντησης στη JVM του πελάτη (σχήμα 3.5).



Σχήμα 3.5: Η πραγμάτωση εικονικής σύνδεσης πελάτη - εξυπηρετητή

Ο εξυπηρετητής, μιας κατανεμημένης εφαρμογής η οποία είναι βασισμένη στο Java RMI δημιουργεί τα (απομακρυσμένα) αντικείμενα, καθώς και τις απομακρυσμένες αναφορές (remote object references) τους, ώστε να είναι δυνατή η πρόσβαση σε αυτά, και περιμένει τους πελάτες να καλέσουν τις μεθόδους τους. Η εφαρμογή πελάτης καλεί τις μεθόδους πάνω στα απομακρυσμένα αντικείμενα μέσω της απομακρυσμένης αναφοράς, η οποία καταχωρείται σε ένα κατάλληλο μητρώο προς αναζήτηση από τους πελάτες.

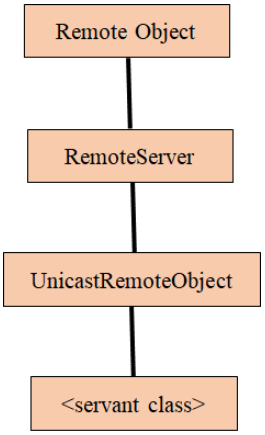
Οι τιμές μεταβιβάζονται με τιμή, ενώ τοπικά αντικείμενα μπορεί να μεταβιβασθούν με τιμή αφού πρώτα σειριακοποιηθούν. Μετά την αποστολή τους γίνεται αποσειριακοποίηση στον παραλήπτη. Τα αντικείμενα αυτά, προκειμένου να σειριακοποιηθούν, θα πρέπει να υλοποιούν τη διεπαφή `java.io.Serializable`. Η μεταβίβαση απομακρυσμένων αντικειμένων γίνεται με αναφορά - στην πραγματικότητα γίνεται αντιγραφή του πληρεξούσιου οποίος περιέχει και την αναφορά. Οι κλήσεις αυτές είναι συνήθως ασύγχρονες. Με άλλα λόγια δεν υπάρχει εμποδισμός του πελάτη κατά τη διάρκεια της κλήσης. Στην περίπτωση προβλήματος το σύστημα δίνει εξαιρέσεις `RemoteException`. Το σημαντικό με τις ασύγχρονες κλήσεις είναι ότι μπορεί να υιοθετηθεί η χρήση πολλών νημάτων με τη λογική του εξυπηρετητή εφαρμογής (application server). Το Java RMI εκτελείται πάνω από το TCP χωρίς αυτό να είναι δεσμευτικό.
Οι αναφορές σε απομακρυσμένα αντικείμενα, λοιπόν, είναι κάτι πολύ βασικό στην όλη επικοινωνία Java RMI. Μια απομακρυσμένη αναφορά αποτελείται από τα εξής μέρη σύμφωνα με το σχήμα 3.6:

32 bits	*32 bits*	*32 bits*	*32 bits*	
Διεύθυνση Διαδικτύου	**Αριθμός θύρας**	**Χρονο-σφραγίδα**	**Αριθμός αντικειμένου**	**Διεπαφή απομακρυσμένου αντικειμένου**

Σχήμα 3.6: Απομακρυσμένη αναφορά αντικειμένου

Αντί αναφοράς μπορεί να μεταβιβασθεί ο πληρεξούσιος του αντικειμένου. Ο πληρεξούσιος είναι σειριακοποιήσιμος. Επίσης, μπορεί να γίνεται κατέβασμα του κώδικα του για ευκολία στην εγκατάσταση των προγραμμάτων, αλλά και για λόγους ταχύτητας και οικονομίας.
Το μητρώο συστήματος RMI, το οποίο καλείται rmiregistry, ουσιαστικά είναι ένας διαχειριστής αντικειμένων, ο οποίος προσφέρει και υπηρεσία ονομασίας. Δεσμεύει τα αντικείμενα ενός συστήματος, ενώ χρησιμοποιεί για απεικόνιση των ονομάτων σε αναφορές αντικειμένων, το μορφότυπο URL, το οποίο ξεκινάει με rmi://. Η εμβέλεια του μητρώου περιορίζεται στο επίπεδο υπολογιστή, δηλαδή κάθε υπολογιστής έχει τουλάχιστον ένα ανεξάρτητο μητρώο. Υπό αυτή την έννοια, το μητρώο RMI δεν είναι μια κατανεμημένη υπηρεσία ονομασίας και για τον λόγο αυτόν ο πελάτης θα πρέπει να γνωρίζει εκ των προτέρων σε ποια μηχανή και θύρα να ρωτήσει. Δεδομένης της δυνατότητας τα μητρώα να

ακούνε σε διάφορες θύρες, αυτό συνεπάγεται ότι μπορούμε να έχουμε πολλά μητρώα σε έναν υπολογιστή, ενώ ένα μητρώο μπορεί να εξυπηρετεί ταυτόχρονα πολλούς εξυπηρετητές.
Είπαμε ότι το μητρώο δεσμεύει τα αντικείμενα ενός συστήματος RMI. Η χρήση της διεπαφής `java.rmi.Naming` περιέχει την μέθοδο `void bind(String url, Remote obj)`, η οποία δεσμεύει ένα όνομα με ένα αντικείμενο, ενώ όλα τα αντικείμενά της θα πρέπει να υλοποιούν τη διεπαφή Remote. Το όνομα των αντικειμένων είναι της μορφής `rmi://<διεύθυνση>:<θύρα>/<όνομα>`. Υπάρχει και η μέθοδος `void rebind(String url, Remote obj)`, η οποία δεσμεύει ένα νέο αντικείμενο με ένα όνομα. Εάν υπάρχει ήδη το όνομα αυτό στο μητρώο τότε εγγράφεται το καινούργιο αντικείμενο στη θέση του παλιού. Οι πελάτες καλούν τη μέθοδο `remote lookup (String url)` προκειμένου να τους επιστραφεί η αναφορά που αντιστοιχεί στο όνομα του αντικειμένου το οποίο ψάχνουν. Με την `void unbind (String url)` ακυρώνεται η δέσμευση ονόματος για κάποιο αντικείμενο. Εάν θέλουμε να δούμε τα αντικείμενα που είναι δεσμευμένα σε ένα μητρώο σε μορφή πίνακα, τότε εκτελούμε την `String[] list (String url)`. Τα απομακρυσμένα αντικείμενα στο Java RMI χαρακτηρίζονται εν γένει ως παροδικά και επεκτείνουν την τάξη `java.rmi.UnicastRemoteObject` και έχουν πάντα την ίδια αρχική κατάσταση, ενώ υπάρχει μια λίστα αναφορών ανά αντικείμενο. Η κλάση του αντικειμένου χαρακτηρίζεται ως servant (υπηρέτης). Η ιεραρχία κλάσεων φαίνεται στο σχήμα 3.7:



Σχήμα 3.7: Η ιεραρχία κλάσεων του απομακρυσμένου αντικειμένου στο Java RMI

3.3.1 Ανάπτυξη Εφαρμογών με Java RMI

Για την ανάπτυξη μιας εφαρμογής με το Java RMI, καταρχάς απαιτείται η συγγραφή της απομακρυσμένης διεπαφής (remote interface), η οποία σε κάθε περίπτωση θα πρέπει να επεκτείνει τη διεπαφή java.rmi.Remote, ενώ οι μέθοδοι της να ρίχνουν (throw) ή αλλιώς να προκαλούν εξαιρέσεις RemoteException. Είναι προφανές ότι οι μέθοδοι εδώ αναφέρονται μόνο σε απομακρυσμένες μεθόδους.

Αφού προσδιοριστεί η απομακρυσμένη διεπαφή, τότε γίνεται η υλοποίησή της και για την ακρίβεια υλοποιούνται όλες εκείνες οι μέθοδοι που αναφέρονται σε αυτήν, διαφορετικά δεν θα είναι δυνατή η εκτέλεσή τους. Ο κώδικας του απομακρυσμένου αντικειμένου που υλοποιεί την απομακρυσμένη διεπαφή πρέπει να ανήκει σε μία τάξη υπηρέτη, ενώ απαιτείται και κατασκευαστής για την αυτόματη εξαγωγή αντικειμένου. Εάν το εν λόγω αντικείμενο υλοποιεί κι άλλες μεθόδους μη περιγραφόμενες στην απομακρυσμένη διεπαφή, τότε αυτές είναι τοπικές και δεν μπορεί να κληθούν από τον πελάτη.

Το αντικείμενο που υλοποιούμε στον εξυπηρετητή είναι συνήθως ένα παροδικό αντικείμενο το οποίο επεκτείνει την κλάση `java.rmi.UnicastRemoteObject`. Μια τέτοια δομή είναι κατάλληλη για την επικοινωνία αντικείμενο-προς-αντικείμενο που είναι σημαντική σε μια ομότιμη επικοινωνία συστημάτων.

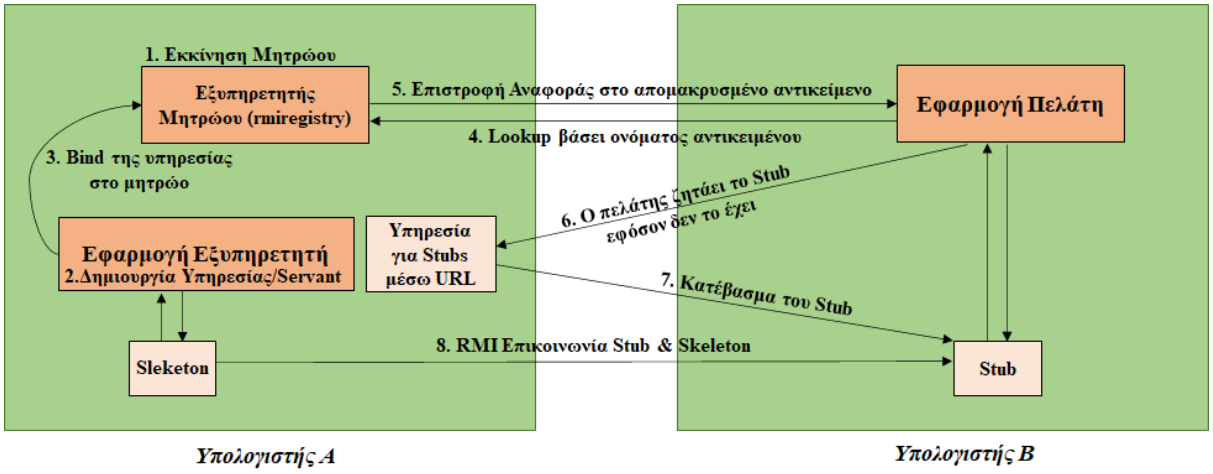
Αφού έχουν υλοποιηθεί η απομακρυσμένη διεπαφή και το αντίστοιχο αντικείμενο, πρέπει να παραχθούν ο κορμός (stub) πελάτη και ο σκελετός (skeleton) εξυπηρετητή. Για να γίνει αυτό απαιτείται πρώτα μεταγλώττιση της διεπαφής και της υλοποίησής της. Αυτό γίνεται με την κλήση του rmic στην κλάση υλοποίησης, οπότε και δημιουργούνται οι κλάσεις κορμού και σκελετού, σε μορφή κώδικα δυφιοσυλλαβών (bytecode).

Έχοντας πλέον έτοιμο το αντικείμενο και την υποδομή για τις απομακρυσμένες κλήσεις σε αυτό, θα πρέπει να υλοποιηθούν τα προγράμματα εξυπηρετητή και πελάτη. Ο εξυπηρετητής είναι αυτός που κατασκευάζει τα απομακρυσμένα αντικείμενα και τα συντηρεί. Τα δημιουργεί με εντολές τύπου `MyObjImpl obj = new MyObjImpl();`, ενώ τα εγγράφει στο μητρώο με εντολές του τύπου `Naming.rebind ("Object1", obj);`.

Από την πλευρά του ο πελάτης οφείλει να αναζητάει τα απομακρυσμένα αντικείμενα στο μητρώο και να ανακτά τις αναφορές τους από αυτό, με εντολές τύπου: `MyObj objStub = (MyObj)Naming.lookup("rmi://όνομα/Obj1");`.

Για να εκτελεστεί, όμως, η απομακρυσμένη κλήση δεν αρκεί μόνο η απομακρυσμένη αναφορά αλλά και ο πληρεξούσιος του αντικειμένου, το οποίο γίνεται με αυτόματο κατέβασμα ενός πληρεξούσιου από τον εξυπηρετητή. Υπάρχει δυνατότητα για χρήση υπο-τάξεων και δυναμική προσθήκη τους σε απομακρυσμένες κλήσεις.

Το Java RMI στην απλούστερη μορφή του είναι ένας ανασφαλής τρόπος επικοινωνίας στο δίκτυο εκτός εάν καθορισθεί κάποια προφύλαξη μέσω ενός διαχειριστή ασφαλείας. Υπάρχει μία κλάση γνωστή ως `RMISecurityManager` που το κάνει αυτό εφικτό. Το RMISecurityManager είναι ο διαχειριστής Ασφαλείας RMI και καθορίζεται με τη μέθοδο `setSecurityManager`. Επιβάλλει μια πολιτική ασφαλείας για τους κορμούς, η οποία εξ ορισμού είναι πολύ συντηρητική, δηλαδή δεν επιτρέπεται καμία προσπέλαση. Οπότε ο προγραμματιστής καλείται να καθορίσει τους κανόνες του τι επιτρέπεται και τι όχι, στην λειτουργία της Java RMI εφαρμογής.



Σχήμα 3.8: Η λειτουργία του Java RMI

Το σχήμα 3.8 δείχνει το κύκλωμα λειτουργίας του Java RMI. Στη συνέχεια περιγράφονται τα έξι βήματα με τα οποία δημιουργούμε μια εφαρμογή Java RMI:

1. Δημιουργία της απομακρυσμένης διεπαφής (remote interface)
2. Υλοποίηση της απομακρυσμένης διεπαφής, δηλαδή του απομακρυσμένου αντικειμένου και των μεθόδων της διεπαφής.
3. Δημιουργία των αντικειμένων στελέχους (stub) και σκελετού (skeleton) χρησιμοποιώντας το εργαλείο rmic (RMI compiler)
4. Εκκίνηση της υπηρεσίας μητρώου με το εργαλείο rmiregistry.
5. Υλοποίηση και εκκίνηση του εξυπηρετητή που φιλοξενεί το απομακρυσμένο αντικείμενο.

6. Υλοποίηση και εκκίνηση της εφαρμογής πελάτης.

Ας δώσουμε ένα παράδειγμα. Έστω ότι θέλουμε να καλέσουμε την μέθοδο `remoteMethod` ενός απομακρυσμένου αντικειμένου `remoteObject`, η οποία έχει δύο ορίσματα: τα `arg1` και `arg2`, δηλαδή έχουμε: `remoteObject.remoteMethod(arg1, arg2)`. Αρχικά, έχουμε κλήση της μεθόδου του απομακρυσμένου αντικειμένου (`remoteObject`) στο επίπεδο πληρεξούσιου του αντικειμένου, δηλαδή του stub, το οποίο συγκεντρώνει τις τιμές των παραμέτρων, που είναι απαραίτητα για την κλήση της μεθόδου. Ταυτόχρονα ενημερώνει την JVM, ότι υπάρχει απομακρυσμένη αναφορά και ότι ο ζητούμενος κώδικας πρέπει να εκτελεστεί απομακρυσμένα. Βέβαια το απομακρυσμένο αντικείμενο μπορεί να βρίσκεται και στον αυτό υπολογιστή (localhost). Αν η αναφορά αφορά άλλον υπολογιστή, τότε αναλαμβάνει το επίπεδο μεταφοράς την αποστολή των δεδομένων που απαιτούνται για την κλήση της απομακρυσμένης μεθόδου, αφού πρώτα έχουν σειριακοποιηθεί.

Παρακάτω, έχουμε το 1ο βήμα δημιουργίας της απομακρυσμένης διεπαφής, η οποία αφορά σε μια μέθοδο υπολογισμού του μέσου όρου τριών ακέραιων αριθμών:

Βήμα 1ο

```
import java.rmi.*;

public interface computeAvgCost extends Remote{

    public int avgCost(int electricityBill,int waterBill, int phoneBill)throws RemoteException;
}
```

Στην συνέχεια έχουμε την υλοποίηση του απομακρυσμένου αντικειμένου.

Βήμα 2ο

```
import java.rmi.*;
import java.rmi.server.*;

public class computeAvgCostRemote extends UnicastRemoteObject implements computeAvgCost{

    computeAvgCostRemote()throws RemoteException{
        super();
    }

    public int avgCost(int electricityBill,int waterBill, int phoneBill){
        return (electricityBill + waterBill + phoneBill) / 3;
    }

}
```

Το επόμενο βήμα είναι να δημιουργήσουμε τα αντικείμενα κορμού και σκελετού χρησιμοποιώντας τον μεταγλωττιστή RMI, δηλαδή το εργαλείο rmic.

Βήμα 3ο

```
rmic computeAvgCostRemote
```

Μετά ξεκινάμε την υπηρεσία μητρώου χρησιμοποιώντας το εργαλείο rmiregistry. Εάν δεν καθορίσουμε τον αριθμό θύρας, τότε η υπηρεσία αυτή «ακούει» στην θύρα 1099. Στο παράδειγμά μας, χρησιμοποιείται η θύρα 1234. Βάζοντας στο τέλος το σύμβολο &, η υπηρεσία αυτή «τρέχει» στο παρασκήνιο (background).

Βήμα 4ο

```
rmiregistry 1234
```

Το απομακρυσμένο αντικείμενο, δηλαδή αυτό που προσφέρει τις υπηρεσίες μέσω Java RMI, πρέπει να φιλοξενείται σε ένα πρόγραμμα εξυπηρετητή. Το αντικείμενο δημιουργείται, ενώ ταυτόχρονα δηλώνεται στο μητρώο του RMI, προκειμένου να είναι δυνατή η ανάκτησή του. Στο παράδειγμα, δεσμεύουμε το απομακρυσμένο αντικείμενο με το όνομα avgBills.

Βήμα 5ο

```
import java.rmi.*;
import java.rmi.registry.*;

public class administrator{

    public static void main(String args[]){
        try{

            computeAvgCost stub=new computeAvgCostRemote ();
            Naming.rebind("rmi://localhost:1234/avgBills",stub);

        }catch(Exception e){System.out.println(e);}
    }
}
```

Τέλος, δημιουργείται το πρόγραμμα πελάτης, το οποίο κάνει την κλήση της απομακρυσμένης μεθόδου, αφού πρώτα αναζητήσει το αντικείμενο που την υλοποιεί μέσω του μητρώου |RMI και της μεθόδου lookup().

Βήμα 6ο

```
import java.rmi.*;

public class customer{

    public static void main(String args[]){
        try{

            computeAvgCost stub=(computeAvgCost)Naming.lookup("rmi://localhost:1234/avgBills");
            System.out.println("Average Cost: " + stub.avgCost(35,22,67));

        }catch(Exception e){System.out.println(e);}
    }

}
```

Στο παράδειγμά μας, το πρόγραμμα του εξυπηρετητή εκτελείται στο localhost. Εάν θέλαμε να αποκτήσουμε πρόσβαση στο απομακρυσμένο αντικείμενο σε άλλο μηχάνημα, θα πρέπει να αλλάξουμε το localhost με το όνομα του κεντρικού υπολογιστή (ή τη διεύθυνση IP) όπου βρίσκεται το απομακρυσμένο αντικείμενο.

Τα προγράμματα πελάτης και εξυπηρετητής θα πρέπει προφανώς πρώτα να μεταγλωττιστούν, όπως π.χ. το πρόγραμμα εξυπηρετητής με την εντολή javac administrator.java. Η εκτέλεση του εξυπηρετητή γίνεται με java administrator hostname method parameters. Αντίστοιχα και για τον πελάτη έχουμε μεταγλώττιση του προγράμματος πελάτη με javac customer.java. Η εκτέλεση του πελάτη γίνεται με

καθορισμό της θέσης του stub μέσω URL, όπως δείξαμε στο σχήμα 3.8. Παραδείγματος χάρη με την εντολή:

```
java -Djava.rmi.server.codebase =
http://hostname/~username/directory/ customer.class hostname method
parameters.
```

Στην συνέχεια παρουσιάζουμε ένα πιο εκτεταμένο παράδειγμα για το Java RMI. Συγκεκριμένα, στο πρώτο βήμα, παρουσιάζεται η δημιουργία της απομακρυσμένης διεπαφής μέσω της οποίας ένας πελάτης μπορεί να πραγματοποιήσει τις παρακάτω λειτουργίες ενός τραπεζικού λογαριασμού:

- Εμφάνιση υπολοίπου λογαριασμού
- Κατάθεση ποσού σε λογαριασμό
- Ανάληψη ποσού από λογαριασμό
- Μεταφορά ποσού σε διαθέσιμο λογαριασμό
- Μεταφορά ποσού σε άλλους λογαριασμούς

Βήμα 1ο

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;

public interface Account extends java.rmi.Remote {
  public String getName() throws RemoteException;
  public float getBalance() throws RemoteException;
  public void withdraw(float amount) throws RemoteException;
  public void deposit(float amount) throws RemoteException;
  public void transfer(float amount, Account acc) throws RemoteException;
  public void transfer(List amounts, List accs) throws RemoteException;
}
```

Στην συνέχεια απεικονίζεται η υλοποίηση του απομακρυσμένου αντικειμένου.

Βήμα 2ο

```java
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
import java.util.List;
import java.util.ListIterator;

public class AccountImpl extends UnicastRemoteObject implements Account {
  private float acounBalance = 0;
  private String accountName = "";
  public AccountImpl(String name) throws RemoteException {
    accountName = name;
  }
  public String getName() throws RemoteException {
    return accountName;
  }
  public float getBalance() throws RemoteException {
    return acounBalance;
  }
  public void withdraw(float amount) throws RemoteException {
    acounBalance -= amount;
    acounBalance = Math.max(acounBalance, 0);
  }
  public void deposit(float amount) throws RemoteException {
    acounBalance += amount;
  }
  public void transfer(float amount, Account acc) throws RemoteException {
    acc.withdraw(amount);
    this.deposit(amount);
  }
  public void transfer(List amounts, List accs) throws RemoteException {
    ListIterator amountCurs = amounts.listIterator();
    ListIterator accCurs = accs.listIterator();
    while (amountCurs.hasNext() && accCurs.hasNext()) {
      Float amount = (Float)amountCurs.next();
      Account account = (Account)accCurs.next();
      this.transfer(amount.floatValue(), account);
    }
  }
}
```

Δημιουργούνται τα αντικείμενα κορμού και σκελετού χρησιμοποιώντας τον μεταγλωττιστή RMI όπως φαίνεται παρακάτω:

Βήμα 3ο

```
rmic AccountImpl
```

Ξεκινάμε την υπηρεσία μητρώου χρησιμοποιώντας το εργαλείο rmiregistry.

Βήμα 4ο

```
rmiregistry &
```

Ο εξυπηρετητής είναι υπεύθυνος να δημιουργήσει το απομακρυσμένο αντικείμενο της διεπαφής Account ώστε να γίνει διαθέσιμο σε όλους τους πελάτες που θέλουν να συνδεθούν σε αυτό.

Βήμα 5ο

```
import java.rmi.*;

public class RegAccount {
  public static void main(String argv[]) {
    try {
      System.setSecurityManager(new MyRMISecurityManager());
      AccountImpl acct = new AccountImpl("customer");
      Naming.rebind("rmi://localhost/customer", acct);
      System.out.println("Registered account.");
    }catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

*

Κάθε πελάτης που επιθυμεί να χρησιμοποιήσει το απομακρυσμένο αντικείμενο, αναζητεί την διεπαφή με την χρήση της lookup υπηρεσίας και καλεί την μέθοδο (λειτουργία) του απομακρυσμένου αντικειμένου που επιθυμεί.

Βήμα 6ο

```
import java.rmi.Naming;
import java.rmi.RMISecurityManager;

public class customer {
  public static void main(String argv[]) {
    try {
      System.setSecurityManager(new MyRMISecurityManager());
      Account customer = (Account)Naming.lookup("rmi://localhost/customer");
      customer.deposit(1000);
      System.out.println("Deposited 1000 in the account from " + customer.getName());
      System.out.println("Balance now totals: " + customer.getBalance());
      customer.withdraw(500);
      System.out.println("New Balance totals: " + customer.getBalance());
    }
    catch (Exception e) {
      System.out.println("Error while looking up account:");
      e.printStackTrace();
    }
  }
}
```

Κατόπιν μεταγλωττίζουμε τα προγράμματα του πελάτη και του εξυπηρετητή, και δεδομένου ότι έχουμε ξεκινήσει ήδη το rmiregistry (βήμα 4ο), «τρέχουμε» πρώτα τον εξυπηρετητή RegAccount και μετά τον πελάτη customer.

3.3.2. Πράκτορες (Agents) με Java RMI

Ένας πράκτορας είναι ένα κομμάτι λογισμικού, το οποίο δύναται να μετακομίζει εντός ενός δικτύου υπολογιστών. Δηλαδή «κατεβαίνει» από έναν υπολογιστή σε έναν άλλο υπολογιστή προκειμένου να εκτελέσει κάποιες εργασίες και να επιστρέψει το αποτέλεσμα αυτών των εργασιών στον υπολογιστή που τον απέστειλε. Έτσι, ένας πράκτορας, π.χ. μπορεί να μετακινείται σε διάφορους ιστοχώρους για την συλλογή χρήσιμων πληροφοριών, όπως είναι ένα πακέτο προσφορών για κάποια προϊόντα, και να τα προωθεί πίσω στον αποστολέα του. Σε αυτή την κατεύθυνση το Java RMI υποστηρίζει τη δημιουργία και τη λειτουργία πρακτόρων, κι αυτό γιατί διαθέτει την υποδομή για την αποστολή ολόκληρων αντικειμένων.

Δεδομένου ότι μιλάμε για Java RMI τεχνολογία, απαιτείται καταρχάς η δημιουργία της διεπαφής του πράκτορα, η οποία μπορεί να είναι του τύπου:

```
public interface My_Agent { public void doThis(); }
```

Στην μέθοδο `dothis()` εμπεριέχεται ο κώδικας σε Java, που εκτελεί τις εργασίες που θέλουμε να κάνει ο πράκτορας στον υπολογιστή στον οποίο μετακινείται. Οπότε θα πρέπει να υπάρχει από την μεριά του υπολογιστή - εξυπηρετητή, ένα απομακρυσμένο αντικείμενο, το οποίο θα λαμβάνει τον πράκτορα τοπικά. Προς τούτο, χρειαζόμαστε μια διεπαφή για ένα τέτοιο απομακρυσμένο αντικείμενο, η οποία θα μπορούσε να ορισθεί ως εξής:

```
public interface receptionObject extends Remote
 { public My_Agent receive(My_Agent agent) throws RemoteException; }
```

Στη συνέχεια δίνεται ο κώδικας για την κλήση του πράκτορα, ο οποίος τοποθετείται μέσα στην κλάση που υλοποιεί τη `receptionObject`:

```
public My_Agent receive(My_Agent agent) throws RemoteException;
     { agent.doThis(); return agent;}
```

Με την λήψη του πράκτορα εκτελείται η μέθοδος `doThis()` και κατόπιν επιστρέφει ο πράκτορας στον υπολογιστή - πελάτη από όπου εστάλη με όλες τις πληροφορίες που συνέλεξε. Στην μέθοδο `doThis()` περιλαμβάνονται όλες οι εργασίες που θα πρέπει να κάνει ο πράκτορας με τη μετακίνησή του στον εξυπηρετητή. Να σημειώσουμε ότι η μετακίνηση ενός πράκτορα μπορεί να αφορά όχι μόνο τη συλλογή πληροφοριών, αλλά και ενέργειες διαμόρφωσης (configuration) στοιχείων στον εξυπηρετητή.

3.3.3. Διαχειριστικά Ζητήματα

Στο Java RMI οι μεταβλητές αντικειμένων είναι μεταβλητές αναφοράς, δηλαδή δείκτες προς κάποιο αντικείμενο. Είναι προφανές ότι όταν δεν υπάρχουν μεταβλητές αναφοράς σε κάποιο αντικείμενο, τότε το αντικείμενο δεν δύναται να χρησιμοποιηθεί και είναι σαν σκουπίδι. Μπορούμε να σκεφτούμε κάτι ανάλογο με τα αρχεία σε ένα σύστημα αρχείων, όπου δεν υπάρχει αναφορά προς αυτά, οπότε χαρακτηρίζονται ως σκουπίδια και θα πρέπει να υπάρχει πρόβλεψη για ανασυγκρότηση της μνήμης.

Αντιστοίχως στην Java υπάρχει μια διαδικασία, η οποία λέγεται συλλογή απορριμμάτων και με την οποία απελευθερώνεται η μνήμη από το αντικείμενο μέσω της καταστροφής του. Αυτό γίνεται όταν αναφερόμαστε στην τοπική χρήση των αντικειμένων. Αλλά και τα απομακρυσμένα αντικείμενα υποστηρίζουν μεταβλητές αναφοράς. Όταν δεν υπάρχουν αυτές οι μεταβλητές αναφοράς προφανώς δεν θα μπορεί να υπάρχει πρόσβαση σε αυτά τα αντικείμενα. Άρα και πάλι απαιτείται η χρήση μια υπηρεσία συλλογής απομακρυσμένων αντικειμένων, τα οποία είναι χρηστικά σκουπίδια. Αυτή τη συλλογή την αναλαμβάνει ένας κατάλληλος προς τούτο συλλέκτης του Java RMI. Συγκεκριμένα, όταν ένα απομακρυσμένο αντικείμενο δεν έχει μεταβλητή ή μεταβλητές αναφοράς, τότε αυτό καταστρέφεται και απελευθερώνεται η μνήμη που καταλάμβανε.

3.4. Common Object Request Broker Architecture (CORBA)

Το Common Object Request Broker Architecture (CORBA) είναι μια προδιαγραφή του Object Management Group για την επίτευξη διαλειτουργικότητας μεταξύ κατανεμημένων συστημάτων. Ο σκοπός του ήταν να αποτελέσει μια αρχιτεκτονική που θα επέτρεπε σε ετερογενή συστήματα να επικοινωνήσουν σε επίπεδο αντικειμένων, ανεξαρτήτως του ποιος έχει σχεδιάσει τις κατανεμημένες εφαρμογές.

Η προδιαγραφή CORBA 1.1 καθόριζε την Interface Definition Language (IDL) και τα Application Programming Interfaces (APIs) τα οποία θα επέτρεπαν την αλληλεπίδραση αντικειμένων πελάτη/εξυπηρετητή μέσω μιας συγκεκριμένης υλοποίησης, αυτής του Object Request Broker (ORB). Ο ORB είναι εκείνο το ενδιάμεσο λογισμικό, το οποίο διαχειρίζεται την επικοινωνία μεταξύ κατανεμημένων αντικειμένων μεταξύ πελάτη και εξυπηρετητή. Η προδιαγραφή CORBA 2.0 αντιμετώπισε προβλήματα στην διαλειτουργικότητα όταν συνεργάζονταν ORBs από διαφορετικούς κατασκευαστές.

Θεωρητικά, ένας πελάτης που κάνει μια αίτηση προς ένα απομακρυσμένο αντικείμενο δεν χρειάζεται να γνωρίζει πού βρίσκεται αυτό, τη γλώσσα προγραμματισμού του, το λειτουργικό του σύστημα ή και άλλες παραμέτρους του συστήματος στο οποίο βρίσκεται. Ένα αντικείμενο CORBA αντιπροσωπεύεται στον «έξω κόσμο» από μία διεπαφή, η οποία είναι ένα σύνολο μεθόδων. Μία συγκεκριμένη οντότητα (στιγμιότυπο) αυτού του αντικειμένου καθορίζεται από μία αναφορά αντικειμένου. Ο πελάτης ενός αντικειμένου CORBA ανακτά την αναφορά αυτού του αντικειμένου και τη χρησιμοποιεί προκειμένου να κάνει κλήσεις στις μεθόδους του αντικειμένου, όπως θα έκανε αν το αντικείμενο βρισκόταν στον πελάτη. Το ORB είναι υπεύθυνο για όλους τους μηχανισμούς που απαιτούνται για να βρει την υλοποίηση του αντικειμένου, να το προετοιμάσει να λάβει την αίτηση, να του στείλει την αίτηση και να μεταφέρει την απάντησή του (αν υπάρχει) πίσω στον πελάτη.

Το Object Request Broker (ORB) είναι ο πυρήνας του συστήματος CORBA, ενώ προσφέρει και υπηρεσίες εντοπισμού υπηρεσιών του CORBA. Το GIOP είναι το γενικό πρωτόκολλο επικοινωνίας ORB και προσφέρει διαλειτουργικότητα μεταξύ διαφορετικών ORB. Η επικοινωνία γίνεται πάνω από TCP/IP για αξιόπιστη επικοινωνία. Οι υπηρεσίες CORBA παρέχονται από το σύστημα και είναι παρόμοιες με τις υπηρεσίες του λειτουργικού συστήματος. Συμπεριλαμβάνουν υπηρεσίες ονομασίας (naming), συναλλαγών (transaction) και γεγονότων (event).

Το βασικό πλεονέκτημα της προδιαγραφής CORBA είναι ότι οι πελάτες και οι εξυπηρετητές μπορεί να είναι υλοποιημένοι σε οποιαδήποτε γλώσσα προγραμματισμού. Αυτό είναι εφικτό εξαιτίας του ότι τα αντικείμενα ορίζονται με ένα υψηλό επίπεδο αφαίρεσης που παρέχεται από την IDL. Αφού καθοριστεί ένα αρχείο IDL, ο μεταγλωττιστής αναλαμβάνει να αντιστοιχίσει το αρχείο IDL σε μια συγκεκριμένη γλώσσα προγραμματισμού, όπως π.χ. σε C, C++, Smalltalk, COBOL, Ada, Java. Οι απομακρυσμένες διεπαφές καθορίζονται με τη γλώσσα IDL, ενώ κάθε δήλωση IDL μεταφράζεται σε κάποια γλώσσα. Η σύνταξη της IDL είναι παρόμοια με αυτήν της C++. Η κάθε προσδιοριζόμενη διεπαφή περιέχει μεθόδους και τύπους. Ένα τύπος Object προσδιορίζει αναφορές αντικειμένων και είναι κατάλληλος για οποιοδήποτε αντικείμενο. Τα αντικείμενα είναι τύποι που υλοποιούν μία διεπαφή.

Η μεταβίβαση των παραμέτρων γίνεται με δομημένους τύπους με τιμή, καθώς και με αντικείμενα με αναφορά. Το CORBA διαθέτει και υπηρεσία γεγονότων για την παραγωγή ειδοποιήσεων συστημικών συμβάντων. Βέβαια δεν πρέπει να μπερδεύονται με τα μηνύματα, η ανταλλαγή των οποίων αποτελεί τη βάση της επικοινωνίας μεταξύ των αντικειμένων και τα οποία αποθηκεύονται ή συσσωρεύονται σε αντίστοιχες ουρές. Υπάρχουν οκτώ είδη μηνυμάτων: η αίτηση κλήσης, η απόκριση κλήσης, η αίτηση και απόκριση εντοπισμού υλοποίησης, η ακύρωση αίτησης, το κλείσιμο σύνδεσης, το σφάλμα μηνύματος και το θραύσμα ή κατακερματισμός.

Οι κλήσεις στο CORBA μπορεί να είναι στατικές, δηλαδή ο πελάτης να γνωρίζει τη διεπαφή εκ των προτέρων και να κάνει χρήση εξειδικευμένων πληρεξούσιων των αντικειμένων. Οι πληρεξούσιοι προφανώς παράγονται από την IDL. Επίσης, οι κλήσεις μπορεί να είναι δυναμικές με την χρήση των διεπαφών DII (Dynamic Invocation Interface). Αυτό απαιτεί αποθήκη διεπαφών με τα στοιχεία των διεπαφών, αντιστοίχιση ονόματος τύπου με τα στοιχεία της διεπαφής και αναζήτηση διεπαφής στην αποθήκη. Αντίστοιχη προσέγγιση υπάρχει και με τις υλοποιήσεις όπου αποθηκεύονται στοιχεία υλοποίησης και ενεργοποίησης των αντικειμένων. Η δέσμευση των αντικειμένων γίνεται αρχικά με παραπομπή σε μια αποθήκη υλοποιήσεων και στη συνέχεια με ανακατεύθυνση στο αντικείμενο. Τα αντικείμενα ακολουθούν μια δομή που περιέχει το όνομα και το είδος. Η δομή εξαρτάται από την εφαρμογή.

Σχετικά με την υλοποίηση συστημάτων CORBA, να αναφέρουμε εν συντομία ότι ξεκινάμε με την συγγραφή της διεπαφής σε IDL (Interface Definition Language), ενώ γίνεται η μεταγλώττιση στην επιθυμητή γλώσσα. Το πρόγραμμα πελάτη πρέπει να υλοποιεί την απομακρυσμένη διεπαφή, ενώ το πρόγραμμα εξυπηρετητή δημιουργεί και αρχικοποιεί το ORB. Κατόπιν, ενεργοποιείται ένας διαχειριστής Φορητός Προσαρμογέας Αντικειμένων (POA), ο οποίος μετατρέπει τον κώδικα σε αντικείμενο CORBA. Το στιγμιότυπο ενός αντικειμένου είναι ένας υπηρέτης (servant). Το POA πραγματοποιεί προσαρμοσμένα την κλήση κατάλληλου υπηρέτη. Ο τρόπος κλήσης από το POA είναι τυποποιημένος και ανεξάρτητος από ORB, ενώ διαθέτει δική του πολιτική ενεργοποίησης.

Ο εξυπηρετητής δημιουργεί τα αντικείμενα υπηρέτες και τα εγγράφει στο ORB. Γίνεται δέσμευση του ονόματος αντικειμένου με αναφορά. Οι μέθοδοι δέσμευσης γίνονται μέσω της διεπαφής NamingContext, με τις μεθόδους: `void bind (in Name n, in Object obj)`, `void unbind (in Name n)`: επαναδέσμευση και `void bind_new_context (in Name n)`. Υπάρχει και η δυνατότητα επίλυσης ονόματος με την μέθοδο `Object resolve (in Name n)`. Θα εξηγήσουμε σε επόμενο κεφάλαιο τι είναι η επίλυση ενός ονόματος. Στην πλευρά του πελάτη, αντίστοιχα, πραγματοποιείται η δημιουργία και η αρχικοποίηση του ORB και η αναζήτηση της αναφοράς από το όνομα του αντικειμένου.

Παρόλα αυτά υπάρχουν και πολλά προβλήματα:

- Για να μπορέσει η IDL να μεταφραστεί σε διάφορες γλώσσες, πρέπει να περιοριστεί σε χαρακτηριστικά που υπάρχουν κοινά σε όλες τις υποστηριζόμενες γλώσσες.
- Για να καλυφθούν πολλοί τομείς εφαρμογών, η προδιαγραφή CORBA παρέχει ποικίλες «υπηρεσίες». Αυτό κάνει την προδιαγραφή πολύ «βαριά» και την ανάπτυξη κατανεμημένων εφαρμογών σαφώς πιο πολύπλοκη. Επιπλέον, οι διάφοροι κατασκευαστές έχουν υλοποιήσει μικρό υποσύνολο αυτών των υπηρεσιών.
- Για να επικοινωνήσει ο πελάτης με τον εξυπηρετητή χρειάζονται και στις δύο πλευρές συμβατά ORBs. Βέβαια με την δεύτερη έκδοση της προδιαγραφής διαφορετικά ORBs μπορούν να συνεργαστούν, όμως αυτή η συνεργασία δεν επεκτείνεται και σε υψηλότερου επιπέδου υπηρεσίες, όπως η ασφάλεια. Επιπλέον, καταργούνται οι όποιες βελτιστοποιήσεις έχει κάνει ο κάθε κατασκευαστής.
- Χρησιμοποιείται κυρίως σε ενδοδίκτυα και πολύ δύσκολα μπορεί να χρησιμοποιηθεί μέσω του Internet, κυρίως λόγω προβλημάτων με τα firewalls.

3.5. Distributed Component Object Model (DCOM)

Το Microsoft Distributed Component Object Model (DCOM) επιτρέπει κλήσεις σε απομακρυσμένα αντικείμενα χρησιμοποιώντας ένα επίπεδο που «κάθεται» πάνω από τον μηχανισμό DCE RPC και το οποίο αλληλοεπιδρά με τις υπηρεσίες του Component Object Model (COM). Ένα αντικείμενο COM μπορεί να υποστηρίξει πολλαπλές διεπαφές, κάθε μία από τις οποίες αντιπροσωπεύει μια διαφορετική συμπεριφορά ενός αντικειμένου. Μια διεπαφή αποτελείται από ένα σύνολο μεθόδων που η λειτουργία τους σχετίζεται. Ένας πελάτης COM αλληλοεπιδρά με ένα αντικείμενο COM παίρνοντας έναν δείκτη για μία από τις διεπαφές του αντικειμένου, και καλώντας μεθόδους μέσω αυτού του δείκτη, σαν να βρισκόταν το αντικείμενο τοπικά στον πελάτη.

Το COM καθορίζει ότι κάθε διεπαφή πρέπει να ακολουθεί μια τυποποιημένη διάταξη μνήμης, η οποία είναι παρόμοια με αυτή της C++. Επειδή η προδιαγραφή είναι σε δυαδικό επίπεδο, επιτρέπει την ολοκλήρωση εφαρμογών που είναι γραμμένες σε διαφορετικές γλώσσες προγραμματισμού. Επίσης, παρότι το DCOM συσχετίζεται συνήθως με τα λειτουργικά συστήματα της Microsoft, μπορεί να υλοποιηθεί και σε συστήματα Unix και Apple.

Το DCOM αντιμετωπίζει κι αυτό αρκετά προβλήματα. Καταρχάς εισάγει μεγάλη πολυπλοκότητα. Κάθε φορά που ένα συστατικό του εξυπηρετητή αλλάζει, αυτές οι αλλαγές πρέπει να διαδοθούν σε όλους τους υπάρχοντες πελάτες. Επίσης, το DCOM είναι ένα «φλύαρο» πρωτόκολλο αφού ελέγχει

πολύ συχνά την παρουσία των πελατών μέσω pinging για να διαπιστώσει αν είναι ενεργοί, ενώ χρειάζονται πάρα πολλές ανταλλαγές πακέτων για να πραγματοποιηθεί μια απλή κλήση μεθόδου.

Τέλος, η χρήση του DCOM διαμέσου firewalls είναι πολύ προβληματική, επειδή εκχωρεί δυναμικά μία θύρα ανά διαδικασία και απαιτεί να είναι ανοιχτές οι θύρες UDP και TCP 135-139.

3.6 Υπηρεσίες Ιστού

3.6.1. Εισαγωγικά

Οι *υπηρεσίες ιστού* (Web Services - WS) είναι απομακρυσμένα αντικείμενα (υπηρεσίες) επιπέδου εφαρμογής, που επιτρέπουν την επικοινωνία μεταξύ εφαρμογών μέσω του διαδικτύου. Είναι συνήθως ανεξάρτητες από την τεχνολογία ή τη γλώσσα στην οποία βασίζονται οι υπολογιστές καθώς χρησιμοποιούν τυποποιημένα πρωτόκολλα, όπως π.χ. το eXtensible Markup Language (XML) για την ανταλλαγή πληροφοριών. Συγκεκριμένα, την δεκαετία του 2000, το κονσόρτσιουμ W3C (WWW Consortium) δέχτηκε μια πρόταση για το πρωτόκολλο Simple Object Access (SOAP). Αυτό το πρωτόκολλο καθόριζε μια μορφή μηνυμάτων που βασίζεται σε XML και δημιούργησε ένα πλαίσιο μετάδοσης για την επικοινωνία μεταξύ εφαρμογών μέσω του πρωτοκόλλου HTTP. Το SOAP ήταν μια ελκυστικότερη λύση σε σχέση με τα παραδοσιακά πρωτόκολλα, όπως αυτά των CORBA και DCOM. Συγκεκριμένα, ένας πελάτης ή χρήστης μπορεί να καλέσει μια υπηρεσία ιστού στέλνοντας ένα μήνυμα SOAP πάνω από το πρωτόκολλο HTTP και στη συνέχεια επιστρέφει ένα μήνυμα απάντησης επίσης SOAP πάνω από το HTTP.

Οι υπηρεσίες ιστού εξελίχθηκαν ώστε να επιτρέπεται η επικοινωνία μεταξύ του πελάτη και των εφαρμογών ενός εξυπηρετητή όχι μόνο μέσω του προτύπου XML, αλλά και του JSON και άλλων ανοιχτών προτύπων για χρήση στον παγκόσμιο ιστό. Οπότε μπορούμε να πούμε ότι υπάρχουν ΄δυο βασικοί τύποι υπηρεσιών ιστού, αυτές που βασίζονται στο Simple Object Access Protocol (SOAP) και αυτές που βασίζονται στο Representative State Transfer (REST), όπου:

- Το SOAP (Simple Object Access Protocol) που σημαίνει Πρωτόκολλο Πρόσβασης Απλού Αντικειμένου, είναι μια προδιαγραφή που επιτρέπει στις εφαρμογές να επικοινωνούν με άλλες εφαρμογές. Χρησιμοποιεί ένα πρωτόκολλο που βασίζεται στην XML για πρόσβαση σε υπηρεσίες ιστού μέσω του HTTP
- Το REST (Representational State Transfer) που σημαίνει αντιπροσωπευτική κατάσταση μεταφοράς, είναι μια υπηρεσία που αναπτύχθηκε στην αρχιτεκτονική REST και συχνά ονομάζεται υπηρεσία RESTful. Είναι ελαφρύ, διατηρήσιμο και επεκτάσιμο. Το υποκείμενο πρωτόκολλο για το REST είναι το HTTP.

Το SOAP είναι ένα πρωτόκολλο ανταλλαγής μηνυμάτων για την ανταλλαγή πληροφοριών μεταξύ δύο υπολογιστών που βασίζονται στην XML μέσω του Διαδικτύου. Τα μηνύματα SOAP είναι καθαρά γραμμένα σε XML και γι' αυτό είναι ανεξάρτητα από την πλατφόρμα και τη γλώσσα. Ένα μήνυμα SOAP περιέχει:

- Ένα *Envelope* που δείχνει την αρχή και το τέλος του μηνύματος
- Ένα *Header* που περιλαμβάνει χαρακτηριστικά που χρησιμοποιούνται για την επεξεργασία του μηνύματος
- Ένα *Body* που κρατά τα δεδομένα XML που πρόκειται να σταλούν και δεν μπορεί να παραλειφθεί
- Ένα *Fault* που παρέχει μηνύματα σφάλματος κατά την επεξεργασία

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope" SOAP-ENV:encodi

   <SOAP-ENV:Header>
      ...
      ...
   </SOAP-ENV:Header>

   <SOAP-ENV:Body>
      ...
      ...
      <SOAP-ENV:Fault>
         ...
         ...
      </SOAP-ENV:Fault>
      ...
   </SOAP-ENV:Body>

</SOAP_ENV:Envelope>
```

Εικόνα 3.3: Δομή ενός SOAP μηνύματος

Η REST είναι μια αρχιτεκτονική βασισμένη στον Παγκόσμιο Ιστό (World Wide Web), που επιτυγχάνει επικοινωνία δεδομένων χρησιμοποιώντας μια διεπαφή όπως το HTTP ή άλλα πρωτόκολλα μεταφοράς που χρησιμοποιούν το τυπικό Uniform Resource Identifier (URI). Ο σχεδιασμός είναι τέτοιος ώστε κάθε στοιχείο σε μια υπηρεσία ιστού RESTful είναι ένας πόρος στον οποίο μπορούμε να έχουμε πρόσβαση χρησιμοποιώντας μεθόδους HTTP, εφόσον το HTTP είναι το επιλεγμένο πρωτόκολλο.

Οι ιδιότητες RESTful χρησιμοποιώντας HTTP είναι:

- *Εκπροσώπηση:* οι πόροι εκπροσωπούνται σε διαφορετικές μορφές και θα πρέπει να αποτελούν μια πλήρη αναπαράσταση του πόρου.
- *Μηνύματα:* Έτσι αλληλοεπιδρά ο πελάτης και ο εξυπηρετητής. Μαζί με τα δεδομένα, τα μηνύματα περιέχουν μεταδεδομένα (metadata) σχετικά με το μήνυμα. Κατά την πρόσβαση σε έναν πόρο RESTful χρησιμοποιώντας το HTTP, οι μέθοδοι που χρησιμοποιούνται συνήθως είναι το GET (διαβάζει έναν πόρο), το PUT (δημιουργεί έναν πόρο), το DELETE (αφαιρεί έναν πόρο) και το POST (ενημερώνει έναν υπάρχοντα πόρο)

Η μορφή μηνύματος 'αιτήματος' REST έχει όπως φαίνεται στον πίνακα 3.1.

<VERB>: **Μέθοδος του HTTP**	**<URI>** **Το URI του πόρου για την εκτέλεση της λειτουργίας**	**<HTTP version>**
<Request Header> **Περιέχει πληροφορία σχετικά με το μήνυμα, όπως το μορφότυπο του σώματος του μηνύματος, το μορφότυπο που υποστηρίζεται από τον πελάτη, κ.ά.**		
<Request Body> **Περιέχει το πραγματικό μήνυμα του πελάτη.**		

Πίνακας 3.1: Η δομή ενός μηνύματος στα restful WS

Το header απόκρισης και το body διατηρούν παρόμοιες πληροφορίες με το αίτημα, μόνο που οι πληροφορίες είναι διαφορετικές, καθώς είναι διαφορετική και η απόκριση του εξυπηρετητή.

3.6.2. Λειτουργίες μιας εφαρμογής βασισμένη σε υπηρεσίες ιστού

Μετά την προδιαγραφή του SOAP, το W3C έβγαλε τις προδιαγραφές WSDL (Web Services Description Language), το οποίο αφορά μια υλοποίηση βασισμένη επίσης στην XML, οι οποίες προσδιορίζουν μια γλώσσα περιγραφής των διεπαφών των υπηρεσιών ιστού. Όπως είδαμε και στις άλλες τεχνολογίες, όπως Java RMI και CORBA, ο ορισμός της διεπαφής των αντικειμένων είναι ζωτικής σημασίας για τη δυνατότητα της κλήσης τους. Τέλος, προσδιορίστηκε ο μηχανισμός UDDI (Universal Description, Discovery, and Integration) για την ανακάλυψη των διαθέσιμων διεπαφών, που περιγράφουν τις διάφορες υπηρεσίες ιστού. Αυτές οι τεχνολογίες προσδιορίζουν έναν νέο τρόπο επικοινωνίας σχετικά με την χρήση των υπηρεσιών.

Πιο συγκεκριμένα, οι υπηρεσίες ιστού, ως εξέλιξη των προηγούμενων τεχνολογιών των κατανεμημένων συστημάτων, εξελίσσουν περαιτέρω την υπηρεσιοστραφή προσέγγιση στα κατανεμημένα πληροφοριακά συστήματα, η οποία ουσιαστικά είναι ένα μοντέλο σχεδίασης και ανάπτυξης εφαρμογών όπου ως κύριο χαρακτηριστικό είναι το γεγονός ότι η επικοινωνία μεταξύ των διαφόρων υπηρεσιών πραγματοποιείται μέσω ενός κοινού πρωτοκόλλου. Οι προσφερόμενες υπηρεσίες θα πρέπει να είναι δυνατόν να κατανοηθούν ως προς την λειτουργικότητά τους από άλλες εφαρμογές, ενώ θα πρέπει επίσης να μπορούν να ανακαλυφθούν προς χρήση. Το τελευταίο απαιτεί την καταχώρησή τους σε υπηρεσία καταλόγου. Μετά την εύρεση της επιθυμητής υπηρεσίας, τότε αυτή καλείται από την εφαρμογή πελάτη μέσω των προαναφερθέντων πρωτοκόλλων SOAP ή REST. Όπως είδαμε και στο πρώτο κεφάλαιο, η αρχιτεκτονική των υπηρεσιών ιστού περιλαμβάνει τον Πάροχος Υπηρεσίας, την Υπηρεσία Καταλόγου και τον Αιτούντα Υπηρεσίας.

Ο *πάροχος υπηρεσίας* είναι υπεύθυνος να κάνει διαθέσιμη την διεπαφή μέσω της οποίας άλλες υπηρεσίες ιστού θα μπορούν να έχουν πρόσβαση και να αιτηθούν την επικοινωνία. Σε ένα μοντέλο πελάτη-εξυπηρετητή, ο πάροχος υπηρεσιών προσομοιάζει με τον ρόλο του εξυπηρετητή. Βέβαια ο πάροχος υπηρεσίας ως δημιουργός αυτής είναι αυτονόητο ότι θα πρέπει να φροντίσει για την διαθεσιμότητά της, την πρόσβαση σε αυτήν, καθώς και για την ασφάλειά της. Άρα, θα πρέπει να κοινοποιήσει στην κατάλληλη υπηρεσία καταλόγου τα στοιχεία πρόσβασης στην υπηρεσίας ιστού, με την αντίστοιχη περιγραφή της και τα δικαιώματα πρόσβασης σε αυτήν.

Η *υπηρεσία καταλόγου*, λοιπόν, έχει ως βασικό της ρόλο την εγγραφή και διατήρηση σε αυτήν των περιγραφών ενός συνόλου υπηρεσιών ιστού. Προφανώς ο κάθε κατάλογος έχει το δικό του σύνολο υπηρεσιών, οπότε καθορίζονται το ποιες υπηρεσίες δύνανται να εγγράφονται, ποιοι θα έχουν πρόσβαση σε αυτόν και με ποια δικαιώματα. Παραδείγματος χάρη, ένας δημόσιος κατάλογος έχει σαφώς πολύ περισσότερα δικαιώματα από έναν ιδιωτικό κατάλογο. Ο ρόλος της υπηρεσίας αυτής είναι ζωτικής σημασίας για την διάχυση της χρήσης των υπηρεσιών ιστού και είναι αυτή που δημιουργεί την ευέλικτη επικοινωνία μεταξύ εφαρμογών.

Ο *αιτών υπηρεσίας* είναι αυτός που αναζητά μια υπηρεσία ιστού στον αντίστοιχο κατάλογο, κατεβάζει την περιγραφή της υπηρεσίας από τον κατάλογο αυτόν και τέλος συνδέεται με την απομακρυσμένη υπηρεσία προκειμένου να αποκτήσει πρόσβαση στην λειτουργικότητά της. Είναι προφανές ότι ο αιτών την υπηρεσία οφείλει να έχει κατάλληλα δικαιώματα πρόσβασης σε αυτήν.

Αυτό το κύκλωμα οντοτήτων και υπηρεσιών μας εισαγάγει στην έννοια της υπηρεσιοστραφής αρχιτεκτονικής βασισμένης στις υπηρεσίες ιστού. Προφανώς υπηρεσίες εφαρμογών δεν υποστηρίζουν μόνο οι υπηρεσίες ιστού, εντούτοις οι τελευταίες παρέχουν μια γρήγορη και αποδοτική λύση για την σχεδίαση και την ανάπτυξη λογισμικού εφαρμογών. Η υπηρεσιοστραφής προσέγγιση επιπλέον βοηθάει τις επιχειρήσεις να γεφυρώσουν και να αυτοματοποιήσουν τις διαδικασίες τους ανεξάρτητα από τα λειτουργικά συστήματα ή τις γλώσσες προγραμματισμού. Ένας σημαντικός στόχος της υπηρεσιοστραφής προσέγγισης είναι η ευθυγράμμιση των στόχων των πληροφοριακών συστημάτων με τους επιχειρηματικούς στόχους. Αυτή η ευθυγράμμιση είναι εκ των ουκ άνευ και η

υπηρεσιοστραφής προσέγγιση βοηθάει σε αυτή την κατεύθυνση λόγω της ευελιξίας και της αρθρωτότητας που διαθέτει. Δεν θα επεκταθούμε περισσότερο στο εν λόγω θέμα διότι έχουμε μιλήσει εκτενώς για τις υπηρεσιοστραφείς αρχιτεκτονικές, όπως και για τις υπηρεσίες ιστού στο άλλο βιβλίο μας «Πληροφοριακά Συστήματα στο Διαδίκτυο».

Συνοπτικά θα λέγαμε ότι από αρχιτεκτονική άποψη στο κατώτερο επίπεδο βρίσκεται το επίπεδο μεταφοράς για την μεταφορά των μηνυμάτων – δεδομένων μεταξύ αιτούντος και παρόχου της υπηρεσίας. Κατόπιν απαιτείται το πρωτόκολλο αναπαράστασης των δεδομένων – μηνυμάτων προκειμένου αυτά να γίνουν κατανοητά και από την πλευρά του παρόχου, αλλά και του αιτούντος. Έπεται η περιγραφή των υπηρεσιών προκειμένου να καθορισθεί η δόμηση των αιτημάτων προς αυτές. Η γλώσσα περιγραφής WSDL χρησιμοποιείται γι' αυτό τον σκοπό. Κατόπιν απαιτείται η ανακάλυψη των υπηρεσιών ιστού. Το πρωτόκολλο καταλόγου και ανακάλυψης UDDI χρησιμοποιείται γι' αυτό τον σκοπό, που είναι κάτι ανάλογο του «χρυσού οδηγού». Επίσης, έχουν αναπτυχθεί πρωτόκολλα για την αξιοπιστία των υπηρεσιών ιστού και την ασφάλειά τους, όπως είναι τα WS- Security και το WS-Reliability. Τέλος, έχοντας αναπτύξει όλη την υποδομή για την ανάπτυξη και την διάχυση των υπηρεσιών ιστών, εξετάζεται η μεταξύ τους συνεργασία και σύνθεση, ώστε να αναπτυχθούν αυτοματισμοί διαδικασιών και βέλτιστη λειτουργικότητα. Σε αυτό το επίπεδο έχουν αναπτυχθεί γλώσσες σύνθεσης υπηρεσιών ιστού, όπως είναι η BPEL (Business Process Execution Language) και η CDL (Choreography Description Language). Στο τέλος του κεφαλαίου παρατίθεται ένα παράδειγμα σύνθεσης υπηρεσιών ιστού.

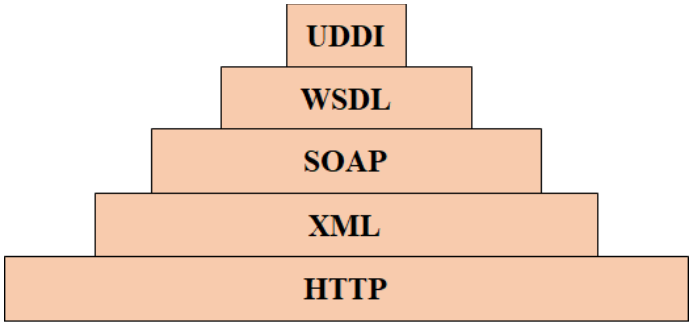
3.6.3. Η σχέση μεταξύ SOAP και RPC

Οι κατασκευαστές συγχέουν συχνά το πρωτόκολλο response - request με τις κλήσεις RPC, ενώ στην πραγματοποίηση δεν υπάρχει ταύτιση. Οι κλήσεις RPC χρησιμοποιούν το πρωτόκολλο request - response, αλλά το πρωτόκολλο αυτό δεν είναι απαραιτήτως RPC. Όπως είδαμε, το RPC είναι μια τεχνική για την κατασκευή κατανεμημένων εφαρμογών. Η τεχνική αυτή είναι βασισμένη στην επέκταση της έννοιας της τοπικής κλήσης διαδικασιών, έτσι ώστε η διαδικασία που καλείται να μην χρειάζεται να υπάρχει στον ίδιο χώρο διευθύνσεων με τη διαδικασία που την καλεί. Οι δύο διεργασίες μπορεί να βρίσκονται στο ίδιο σύστημα ή σε διαφορετικά συστήματα που συνδέονται με μια σύνδεση δικτύου. Με τη χρησιμοποίηση του RPC, οι προγραμματιστές κατανεμημένων εφαρμογών αποφεύγουν τις λεπτομέρειες της διεπαφής με το δίκτυο. Η ανεξαρτησία μεταφοράς απομονώνει την εφαρμογή από τα φυσικά και λογικά στοιχεία του μηχανισμού επικοινωνίας των δεδομένων και επιτρέπει στην εφαρμογή να χρησιμοποιήσει μια ποικιλία μεταφορών. Το RPC απαιτεί μια μετάφραση της μεθόδου υπογραφής στα μηνύματα SOAP. Λόγω της δημοτικότητας του RPC, το SOAP στην πραγματικότητα περιγράφει *μια σύμβαση για τη χρησιμοποίηση κλήσεων RPC*.

Ποια όμως είναι η χρησιμότητα του SOAP, εφόσον υπάρχει το RPC; Οι σημερινές εφαρμογές επικοινωνούν με χρήση των απομακρυσμένων κλήσεων διαδικασίας μεταξύ των αντικειμένων, όπως π.χ. μεταξύ DCOM και CORBA, αλλά το HTTP δεν σχεδιάστηκε για αυτό. Το RPC εμπεριέχει ένα πρόβλημα συμβατότητας και ασφάλειας. Τα firewalls και οι proxy servers υπό κανονικές συνθήκες εμποδίζουν αυτό το είδος κυκλοφορίας. Τα firewalls σκοπό έχουν να προστατεύουν τα δεδομένα κάποιου οργανισμού που παρέχει υπηρεσίες ιστού, αλλά από την άλλη δυσκολεύουν την παροχή υπηρεσιών βασισμένες στον παγκόσμιο ιστό. Για παράδειγμα, ας υποθέσουμε ότι μια εταιρία πουλάει τα αγαθά της μέσω του διαδικτύου. Μια υπηρεσία ιστού πώλησης αυτών των αγαθών προφανώς χρειάζεται πληροφορίες, όπως π.χ. το ποια προϊόντα είναι άμεσα διαθέσιμα. Μια τέτοια πληροφορία απαιτεί πρόσβαση στα εσωτερικά συστήματα της εταιρίας. Προκειμένου να υπάρχει ασφαλής πρόσβαση σε αυτά τα συστήματα υπάρχουν τεχνικές ασφαλείας τις οποίες όμως συζητιούνται στο κεφάλαιο 8.

3.6.4. Πλεονεκτήματα των υπηρεσιών ιστού σε σχέση με τις παλαιότερες κατανεμημένες τεχνολογίες

Τα πλεονεκτήματα των υπηρεσιών ιστού βασίζονται στη στοίβα πρωτοκόλλων τους, ένα σύνολο ανοιχτών προδιαγραφών που είναι ήδη υπάρχοντα πρότυπα του διαδικτύου ή είναι προδιαγραφές που είναι ευρέως αποδεκτές. Η βασική στοίβα αυτών των πρωτοκόλλων φαίνεται στο σχήμα 3.9:



Σχήμα 3.9: Στοίβα πρωτοκόλλων στις υπηρεσίες ιστού

Στη βάση της στοίβας έχουμε το HTTP, ένα πρωτόκολλο που μοιάζει με το RPC που χρησιμοποιούν τα CORBA και DCOM, αλλά είναι πιο απλό, ευρέως διαδομένο και «φιλικό» προς τα firewalls. Αυτό γιατί είναι το πρωτόκολλο επικοινωνίας του διαδικτύου και όλα τα firewalls έχουν εξ ορισμού ανοιχτή τη θύρα 80 για επικοινωνία HTTP. Στο HTTP, και η αίτηση και η απάντηση μπορούν να περιέχουν αυθαίρετα μεγάλο ποσό πληροφορίας. Επίσης οι HTTP επικεφαλίδες χρησιμοποιούν απλό κείμενο, που κάνει ευκολότερη την χρήση του από όλους του προγραμματιστές. Επίσης, χρησιμοποιεί URLs ως αναφορές στα αντικείμενα που είναι ένας πολύ πιο απλός και κατανοητός τρόπος από τα IORs και OBJREFs που συναντάμε στα CORBA και DCOM αντιστοίχως.

Επίσης, έχουμε μια γλώσσα κοινής περιγραφής δεδομένων, την XML, η οποία επίσης είναι ιδιαίτερα διαδεδομένη. Επειδή το HTTP δεν έχει περιορισμούς στο μέγεθος των δεδομένων (payload) που μεταφέρει, δεν έχει μηχανισμούς αναπαράστασης των τιμών των παραμέτρων, όπως συμβαίνει στα μηνύματα RPC. Εδώ είναι ο ρόλος της XML. Η XML είναι μια γλώσσα σήμανσης δεδομένων που είναι ανεξάρτητη από πλατφόρμες. Επιτρέπει την σειριακοποίηση των δεδομένων σε μια μορφή που αποκωδικοποιείται εύκολα από κάθε πλατφόρμα. Παρόλα αυτά, είναι πολύ εύκολη στη χρήση. Επιπλέον, έχει υιοθετηθεί ευρέως και έτσι, σε συνδυασμό με το HTTP, λύνει τα προβλήματα διαλειτουργικότητας και πολυπλοκότητας των CORBA και DCOM.

Το SOAP είναι ένα πρωτόκολλο ανταλλαγής μηνυμάτων βασισμένο στην XML, έτσι είναι και αυτό ανεξάρτητο από γλώσσα και πλατφόρμα. Αν και το SOAP δεν είναι «δεμένο» με κάποιο δικτυακό πρωτόκολλο στο σύνολο των περιπτώσεων χρησιμοποιείται με το HTTP. Χρησιμοποιεί την WSDL ως μια μορφή IDL. Η WSDL η οποία καθορίζει τη διεπαφή μιας υπηρεσίας καθώς και χαρακτηριστικά υλοποίησης της υπηρεσίας επίσης βασίζεται στην XML. Ουσιαστικά το SOAP είναι μία αίτηση ή μια απάντηση HTTP που ακολουθεί συγκεκριμένους κανόνες κωδικοποίησης. Ένας τελικός κόμβος που λαμβάνει μηνύματα SOAP είναι απλά το αντιστοιχιζόμενο από ένα HTTP URL, που καθορίζει τον στόχο για την κλήση μιας μεθόδου. Το SOAP είναι ένα πρωτόκολλο σαφώς πιο απλό από αυτά που χρησιμοποιούνταν από τις τεχνολογίες CORBA και DCOM. Έτσι, το να δημιουργήσει κανείς μια υλοποίηση SOAP είναι πολύ πιο εύκολο.

Ανακεφαλαιώνοντας, τα βασικά πλεονεκτήματα των υπηρεσιών ιστού συνοψίζονται στα ακόλουθα σημεία:

- Χρησιμοποιούν πρωτόκολλα και προδιαγραφές (HTTP και XML) τα οποία είναι ευρέως διαδεδομένα και τα οποία χρησιμοποιούνται από όλες τις πλατφόρμες, έτσι δεν προσθέτουμε πολυπλοκότητα ενώ προσφέρουν απόλυτη διαλειτουργικότητα.
- Χάρη στο HTTP είναι ιδανικές για κατανεμημένες εφαρμογές που επικοινωνούν μέσω του διαδικτύου και δεν αντιμετωπίζουν προβλήματα με τα firewalls.

- Χρησιμοποιούν ως αναγνωριστικό των αντικειμένων την ευρέως διαδεδομένη έννοια των URLs

3.6.5. Παραδείγματα Υπηρεσίων Ιστού τύπου Restful

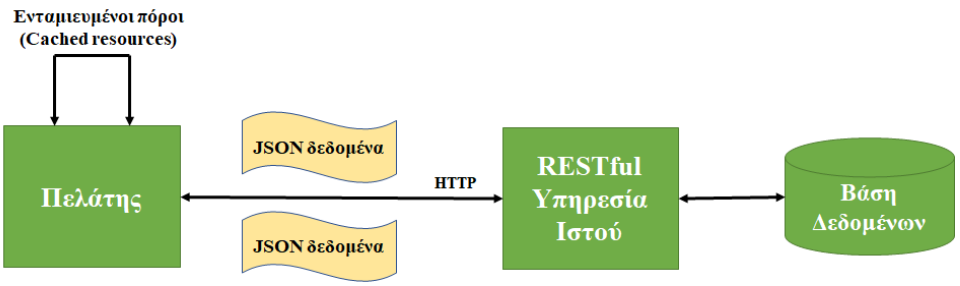
Σε αυτή την υποενότητα θα εστιάσουμε λίγο περισσότερο στη νεότερη τεχνολογία υλοποίησης υπηρεσιών ιστού, αυτή την τύπου restful. Αυτή η τεχνολογία προέκυψε από την αντιπροσωπευτική μεταφορά κατάστασης (REST), που είναι ένα πρότυπο αρχιτεκτονικού σχεδιασμού για Application Programming Interfaces (API). Τα API που ακολουθούν αυτό το μοτίβο ονομάζονται REST API ή RESTful API. Το REST καθορίζει ορισμένα πρότυπα μεταξύ συστημάτων υπολογιστών στον παγκόσμιο ιστό (WWW) , τα οποία διευκολύνουν την επικοινωνία μεταξύ των συστημάτων.

Το REST επιτρέπει στον πελάτη και στον εξυπηρετητή να υλοποιηθούν ανεξάρτητα χωρίς τη γνώση του ενός από το άλλο. Αυτό σημαίνει ότι ο κώδικας και στις δύο πλευρές μπορεί να τροποποιηθεί χωρίς να χρειάζεται να «ανησυχούμε» για το αποτέλεσμα της τροποποίησης από την άλλη πλευρά. Τα συστήματα που ακολουθούν το αρχιτεκτονικό τύπου REST δεν διατηρούν την κατάσταση του συστήματος, είναι δηλαδή stateless. Μα οντότητα, δηλαδή, δεν χρειάζεται να γνωρίζει την κατάσταση της άλλης οντότητας. Αυτό επιτυγχάνεται μέσω της χρήσης πόρων (το αντικείμενο ή οι πληροφορίες που χρειάζονται για την αποθήκευση ή την αποστολή σε άλλες οντότητες). Δεδομένου ότι οι ενέργειες σε πόρους είναι πλήρως τυποποιημένες, δεν απαιτείται η γνώση της κατάστασης της άλλης οντότητας. Υιοθετώντας την προσέγγιση του REST, μπορούμε να επωφεληθούμε από την χαλαρή σύνδεση μεταξύ του εξυπηρετητή και του πελάτη, καθώς και από τα χαρακτηριστικά του API REST, δηλαδή το modularity και το statelessness, τα οποία επιτρέπουν στις εφαρμογές να είναι πιο αξιόπιστες, να έχουν καλύτερη απόδοση και να είναι πιο επεκτάσιμες.

Συνοπτικά, μερικά από τα οφέλη του REST είναι:

- η απλότητά του στην υλοποίηση, σε σχέση και με τις υπηρεσίες ιστού τις βασισμένες στο SOAP.
- η απλή επικοινωνία με τον εξυπηρετητή μέσω HTTP που υποστηρίζει απλές μορφές XML και JSON σε σύγκριση με το SOAP που υποστηρίζει μόνο XML.
- η ελαφριά (light-weight) υλοποίηση.
- η κλιμάκωση ή αλλιώς ελαστικότητα, καθώς η αρχιτεκτονική REST μας επιτρέπει να προσθέσουμε εύκολα πολλές εμφανίσεις των στοιχείων ή της εφαρμογής πίσω από τους εξισορροπητές φορτίου (workload balancers).

Το σχήμα 3.10 απεικονίζει την αλληλεπίδραση μεταξύ του πελάτη και του εξυπηρετητή που φιλοξενεί μια υπηρεσία ιστού restful, όπου μέσω του πρωτοκόλλου http στέλνουν μηνύματα json ο ένας στον άλλον. Όταν ο πελάτης ζητά έναν προσωρινά αποθηκευμένο πόρο, αντί να τον ζητά από τον εξυπηρετητή, ο πόρος επιστρέφεται από την προσωρινή μνήμη του πελάτη. Εδώ βέβαια τίθεται θέμα ασυμβατότητας όταν ο απομακρυσμένος πόρος αλλάξει και είναι διαφορετικός από τον προσωρινά αποθηκευμένο.



Σχήμα 3.10: Αλληλεπίδραση μεταξύ πελάτη - εξυπηρετητή με χρήση restful WS

Στην συνέχεια ακολουθούν παραδείγματα όπου χρησιμοποιήθηκε το REST Assured. Το REST Assured είναι ένα API Java Domain Specific Language (DSL), το οποίο χρησιμοποιείται για τη σύνταξη αυτοματοποιημένων δοκιμών για RESTful API. Χρησιμοποιείται για τη διεκπεραίωση υπηρεσιών ιστού REST σε αιτήματα POST, GET, PUT, DELETE και επικυρώνει την απόκριση του αιτήματος.

Έστω ότι έχουμε μια τάξη μαθητών, με τα ακόλουθα πεδία και τύπους:

Access Specifier	Type	Field
private	Long	**id**
private	String	**firstName**
private	String	**lastName**
private	String	**gender**

Παράδειγμα 1 : *Λήψη όλων των εγγραφών των μαθητών με την μέθοδος GET της HTTP και τις εξής παραμέτρους:*

- URL προορισμού: http://unipilabs.com:6565/
- Resource Path: /rest/students

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import static org.hamcrest.Matchers.equalTo;
import org.testng.annotations.Test;
import static org.testng.Assert.assertEquals;
import static org.testng.Assert.assertTrue;

import io.restassured.response.Response;
import io.restassured.RestAssured;

public class GETRequestTest {

    private static Logger LOG = LoggerFactory.getLogger(GETRequestTest.class);

    @Test
    public void testGetAllStudentRecords() {

        String url = "http://unipilabs.com:6565/rest/students";

        /**
        * Example 1 - GET all the existing student's record
        */
        LOG.info("Step - 1 : Send GET Request");
        Response response = RestAssured.given().get(url).andReturn();

        LOG.info("Step - 2 : Print the JSON response body");
        response.getBody().prettyPrint();

        LOG.info("Step - 3 : Assert StatusCode = 200");
        assertEquals(response.getStatusCode(), 200, "http status code");

        LOG.info("Step - 4 : Verify that the response contains id = 101");
        LOG.info("list of Student's Id " +response.getBody().jsonPath().getList("id"));
        assertTrue(response.getBody().jsonPath().getList("id").contains(101));
    }
}
```

Ο παραπάνω κώδικας πραγματοποιεί ένα GET αίτημα (request). Πιο συγκεκριμένα, το εξής:

```
Response response = RestAssured.given().get(url).andReturn();
```

Στο step 1 λαμβάνεται το αίτημα από τον πελάτη και με το step 2 επιστρέφεται το αποτέλεσμα σε μορφή json. Στο επόμενο βήμα (step 3) γίνεται έλεγχος, ώστε να επιβεβαιώσουμε ότι το response status

είναι 200, δηλαδή ότι το αίτημα πραγματοποιήθηκε επιτυχώς. Η τελευταία γραμμή (step 4) επαληθεύει (verify) ότι υπάρχει ένας φοιτητής με id 101.

Παράδειγμα 2 : *Η μέθοδος HTTP POST στέλνει δεδομένα στον εξυπηρετητή για να δημιουργήσει έναν νέο πόρο.*

Ο τύπος σώματος αιτήματος υποδεικνύεται από το πεδίο κεφαλίδας Content-Type. Ας δούμε ένα παράδειγμα για να το κατανοήσουμε καλύτερα. Συγκεκριμένα, έστω ότι πραγματοποιείται ένα αίτημα POST για τη δεδομένη διεύθυνση URL, προκειμένου να εγγράψουμε έναν χρήστη μέσω email και password, όπως παρακάτω:

```
curl -iX POST -H "Content-Type: application/json" -d '{ "email": "user1@unipi.gr", "password": "123456" }'  https://unipi.gr/api/register
```

Στην αίτηση POST παραπάνω, (-iX είναι για μέθοδο HTTP, -H είναι συντομογραφία -header, -d για -data) τα δεδομένα διαβιβάζονται με τη μορφή μιας συμβολοσειράς JSON. Η έξοδος εκτέλεσης στο τερματικό εμφανίζει κεφαλίδες απόκρισης, και ένα σώμα μηνυμάτων σε μορφή JSON που περιέχει αναγνωριστικό και ζεύγη κλειδιών / τιμών για το http response, όπως φαίνεται παρακάτω:

```
curl -iX POST -H "Content-Type: application/json" -d '{ "email":
"user1@unipi.gr", "password": "123456" }'  https://unipi.gr/api/register

HTTP/2 200
date: Sat, 12 Jun 2021 18:23:59 GMT
content-type: application/json; charset=utf-8
content-length: 36
x-powered-by: Express
access-control-allow-origin: *
etag: W/"24-4iP0za1geN2he+ohu8F0FhCjLks"
via: 1.1 vegur
cf-cache-status: DYNAMIC
cf-request-id: 0aa31079ae0000c5306eb4e000000001
expect-ct: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-
cgi/beacon/expect-ct"
report-to:
{"endpoints":[{"url":"https:\/\/a.nel.cloudflare.com\/report\/v2?s=Hfk9iI0%2F0Zb
gXPwVXi95cyAM0iZbn3XlagyIdfufpD2oIWg18XxsqAYDwL6HKGMk6y8bHN3kn2eeLjacNXAwtklfG2p
E2mgbGcylnR0oMjt8dKvf9YkD"}],"group":"cf-nel","max_age":604800}
nel: {"report_to":"cf-nel","max_age":604800}
server: cloudflare
cf-ray: 65e51d091957c530-ORD
alt-svc: h3-27=":443"; ma=86400, h3-28=":443"; ma=86400, h3-29=":443"; ma=86400,
h3=":443"; ma=86400

{"id":4,"token":"QpwL5tke4Pnpja7X4"}
```

Παράδειγμα 3: *Η μέθοδος HTTP PUT θα ενημερώσει έναν υπάρχοντα πόρο στον εξυπηρετητή.*

Το PUT είναι αδιάφορο, δηλαδή ότι ένα αποτέλεσμα θα παραμείνει το ίδιο ακόμα κι αν καλείται πολλές φορές διαδοχικά. Για παράδειγμα, η κλήση του POST αρκετές φορές με τα ίδια δεδομένα αιτήματος μπορεί να δημιουργεί έναν νέο πόρο κάθε φορά, επομένως, δεν είναι αδιάφορο. Ας καταλάβουμε καλύτερα τι σημαίνει αυτό με το παρακάτω παράδειγμα.

Ας κάνουμε ένα αίτημα GET για τη δεδομένη διεύθυνση URL και ας πάρουμε μια υπάρχουσα εγγραφή με το αναγνωριστικό 1:

```
curl -X GET  https://jsontest.testPut.com/posts/1
```

Ενημερώνουμε τον τίτλο αυτής της εγγραφής χρησιμοποιώντας το αίτημα PUT:

```
curl -X PUT -d '{"title":"This is an updated post"}'  https://jsontest.testPut.com/posts/1
```

Στην παραπάνω αίτηση PUT, (-X είναι για μέθοδο HTTP και -d για τα data) τα δεδομένα διαβιβάζονται με τη μορφή συμβολοσειράς JSON. Η έξοδος είναι το μήνυμα απόκρισης σε μορφή JSON με ενημερωμένο τον τίτλο και με κωδικό κατάστασης απόκρισης HTTP 200 OK. Οπότε έχουμε το εξής αποτέλεσμα εκτέλεσης στο τερματικό:

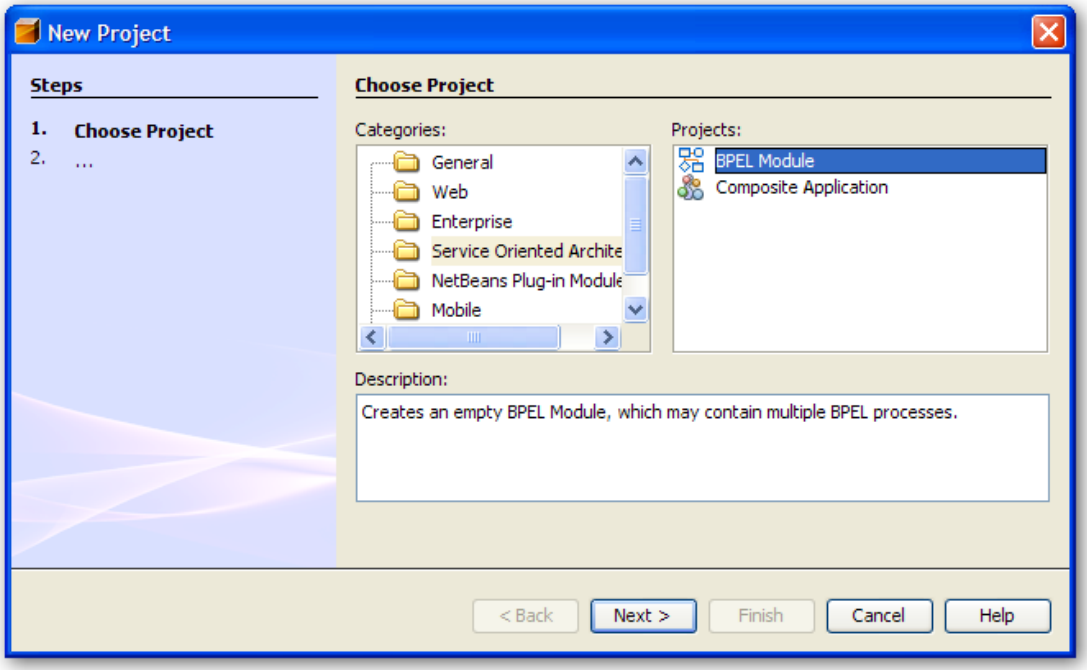
```
curl -X GET https://jsontest.testPut.com/posts/1
{
  "userId": 1,
  "id": 1,
  "title": "sunt aut facere repellat provident occaecati excepturi optio
reprehenderit",
  "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita
et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est
autem sunt rem eveniet architecto"


curl  -iX PUT -d   '{"title":"This is an updated post"}'
https://jsontest.testPut.com/posts/56
HTTP/2 200
date: Sat, 12 Jun 2021 18:33:59 GMT
content-type: application/json; charset=utf-8
content-length: 63
x-powered-by: Express
x-ratelimit-limit: 1000
x-ratelimit-remaining: 999
x-ratelimit-reset: 1623522861
vary: Origin, Accept-Encoding
access-control-allow-credentials: true
cache-control: no-cache
pragma: no-cache
expires: -1
x-content-type-options: nosniff
etag: W/"3f-QG4Nxrinbpdlk4EMxxKUPNrIr2s"
via: 1.1 vegur
cf-cache-status: DYNAMIC
cf-request-id: 0aa319a1e70000584b5a3af000000001
expect-ct: max-age=604800, report-uri="https://report-
uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
report-to:
{"endpoints":[{"url":"https:\/\/a.nel.cloudflare.com\/report\/v2?s=oLYFr
pdeWd%2F7bh6aV8JgoRM6aZGGWv2sxgP1ra7h6kNdEFEWzQKCdfvQ3tXdsMZ6WuqHUY9O6xQ
NJ%2F3aSLXiCn7cnCPKzA6nLkXsH4%2FkKUyILD%2BdA8kBhiwXSH%2FSoWbwMf7gk9J6q2x
SUg%3D%3D"}],"group":"cf-nel","max_age":604800}
nel: {"report_to":"cf-nel","max_age":604800}
server: cloudflare
cf-ray: 65e52bafd994584b-ORD
alt-svc: h3-27=":443"; ma=86400, h3-28=":443"; ma=86400, h3-29=":443";
ma=86400, h3=":443"; ma=86400
{
  "{\"title\":\"This is an updated post\"}": "",
  "id": 56
}
```

3.7 Σύνθεση Υπηρεσιών Ιστού: Κατασκευή μιας Κατανεμημένης Υπηρεσίας Ιστού

Το BPEL (Business Process Execution Language) είναι μια γλώσσα σύνθεσης προ-υπαρχουσών υπηρεσιών ιστού. Στην πραγματικότητα είναι μια γλώσσα σήμανσης δεδομένων, η οποία προσδιορίζει τον ρόλο της κάθε υπηρεσίας στα πλαίσια της σύνθεσης. Παρακάτω θα δειχθεί η κατασκευή μιας τέτοιας διεργασίας με την γλώσσα BPEL. Θα δημιουργήσουμε μια απλή περίπτωση σύνθεσης υπηρεσιών ιστών, δίνοντας ένα όνομα και ένα email, τα οποία ενώνονται σε ένα αλφαριθμητικό. Η υλοποίηση που παρουσιάζεται παρακάτω έχει γίνει στο περιβάλλον του Netbeans, το οποίο απαιτεί δύο projects για να δημιουργηθεί μία διεργασία BPEL. Συγκεκριμένα:

- Ένα BPEL module
- Μία Composite application που θα παραπέμπει σε ένα ή περισσότερα BPEL modules.

Όπως φαίνεται στο επόμενο wizard (εικόνα 3.4) δημιουργίας ενός New Project, από το πεδίο Categories επιλέγουμε το Service Oriented Architecture (SOA), ενώ από το πεδίο Projects επιλέγουμε το BPEL module.



Εικόνα 3.4: Wizard δημιουργίας ενός New Project

Επιλέγοντας ως όνομα του project το *ContactBPEL*, για να χτίσουμε μία διεργασία τύπου BPEL, δημιουργούμε τα ακόλουθα αρχεία:

- Ένα *XML Schema* που καθορίζει τα μηνύματα που θα χρησιμοποιηθούν ως είσοδος και έξοδος της διεργασίας BPEL, αφού εκτεθεί ως υπηρεσία Ιστού.
- Ένα *έγγραφο WSDL* που θα καθορίζει τη στατική διεπαφή της υπηρεσίας μας με τη χρήση του XML Schema για του τύπους δεδομένων και τις λειτουργίες που θα εκτίθενται από την υπηρεσία.
- Την ίδια τη *διεργασία BPEL*.

Καταρχάς επιλέγουμε ως τύπο αρχείου (file type) το XML Schema και ως όνομα το «schema». Οπότε δημιουργείται αυτόματα ο παρακάτω κώδικας στο αρχείο schema.xsd:

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

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://xml.netbeans.org/schema/schema"
        xmlns:tns="http://xml.netbeans.org/schema/schema"
```

```
        elementFormDefault="qualified">

</xsd:schema>
```

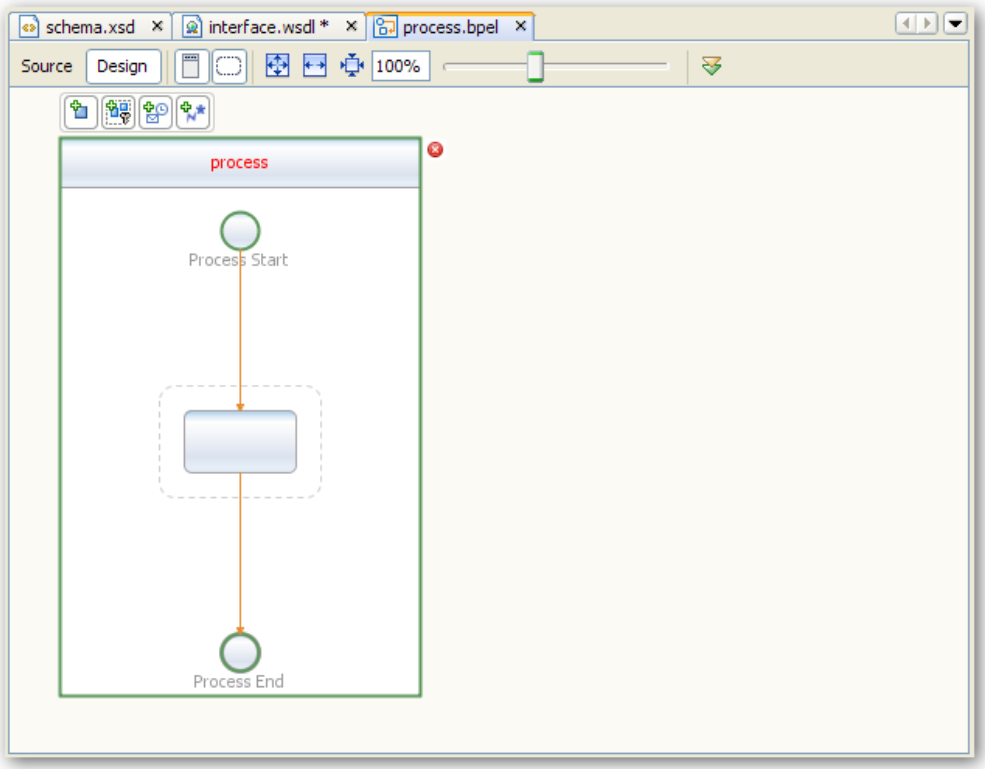
Τροποποιούμε τον παραπάνω κώδικα ως εξής:

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

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://xml.netbeans.org/schema/schema"
        xmlns:tns="http://xml.netbeans.org/schema/schema"
        elementFormDefault="qualified">

 <xsd:complexType name="ContactType">
  <xsd:sequence>
   <xsd:element name="name" type="xsd:string"></xsd:element>
   <xsd:element name="email" type="xsd:string"></xsd:element>
  </xsd:sequence>
 </xsd:complexType>
 <xsd:element name="contact" type="tns:ContactType"></xsd:element>
 <xsd:element name="compactContact" type="xsd:string"></xsd:element>

</xsd:schema>
```

Όταν καθορίζεται το XML schema θα πρέπει να προσεχθεί το εξής: Για να λειτουργήσει σωστά το πλαίσιο JAX-WS (Java API for XML Web Services) πρέπει για κάθε complexType να καθορίζεται ένα στοιχείο αυτού του τύπου. Διαφορετικά, θα υπάρχουν σφάλματα κατά τη δημιουργία του εγγράφου WSDL. Στον παραπάνω κώδικα έχουμε καθορίσει το στοιχείο *contact* να είναι σύνθετου τύπου *ContactType*.

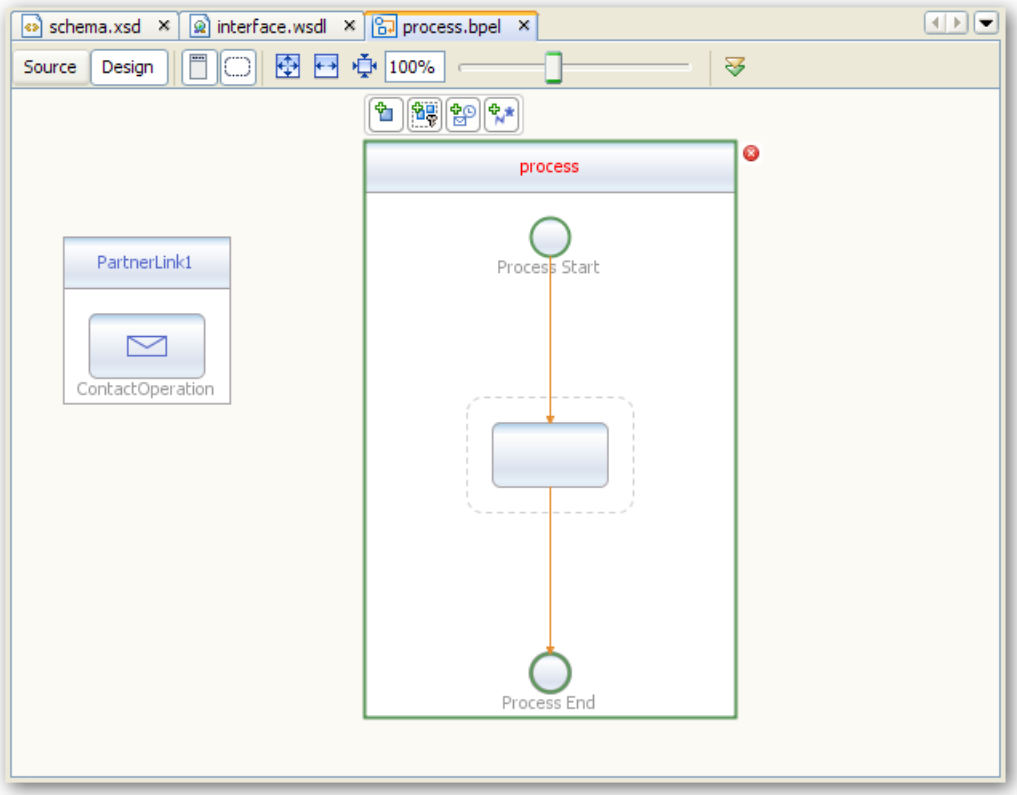
Κατόπιν θα δημιουργήσουμε το έγγραφο WSDL, όπου πάλι στο «New File» και στο πεδίο Categories επιλέγουμε ως τύπο αρχείου WSDL Document. Έστω ότι επιλέγουμε για το έγγραφο WSDL, το όνομα interface, ενώ ταυτόχρονα δηλώνουμε “Import XML Schema File(s)” για να εισάγουμε το αρχείο schema.xsd.

Καθορίζουμε το αφηρημένο μέρος του WSDL όπου παραπέμπουμε στα στοιχεία (και όχι στον σύνθετο τύπο), που έχουμε ορίσει να έχουν το ρόλο της εισόδου και της εξόδου στην διεργασία BPEL. Στο πεδίο Message Part Name γράφουμε body. Κατόπιν καθορίζουμε το είδος των μηνυμάτων SOAP σε “Document Literal”. Οπότε αυτόματα δημιουργείται το αρχείο WSDL με βάση το XML schema που δημιουργήσαμε πιο πριν καθώς και τις ρυθμίσεις. Το επόμενο βήμα είναι η δημιουργία του αρχείου της διεργασίας BPEL. Εδώ, ο τύπος αρχείου είναι BPEL process, στην κατηγορία Service Oriented Architecture. Έστω ότι το όνομα της διεργασίας είναι process. Με τη δημιουργία της εμφανίζεται στον editor το παρακάτω διάγραμμα της διεργασίας:



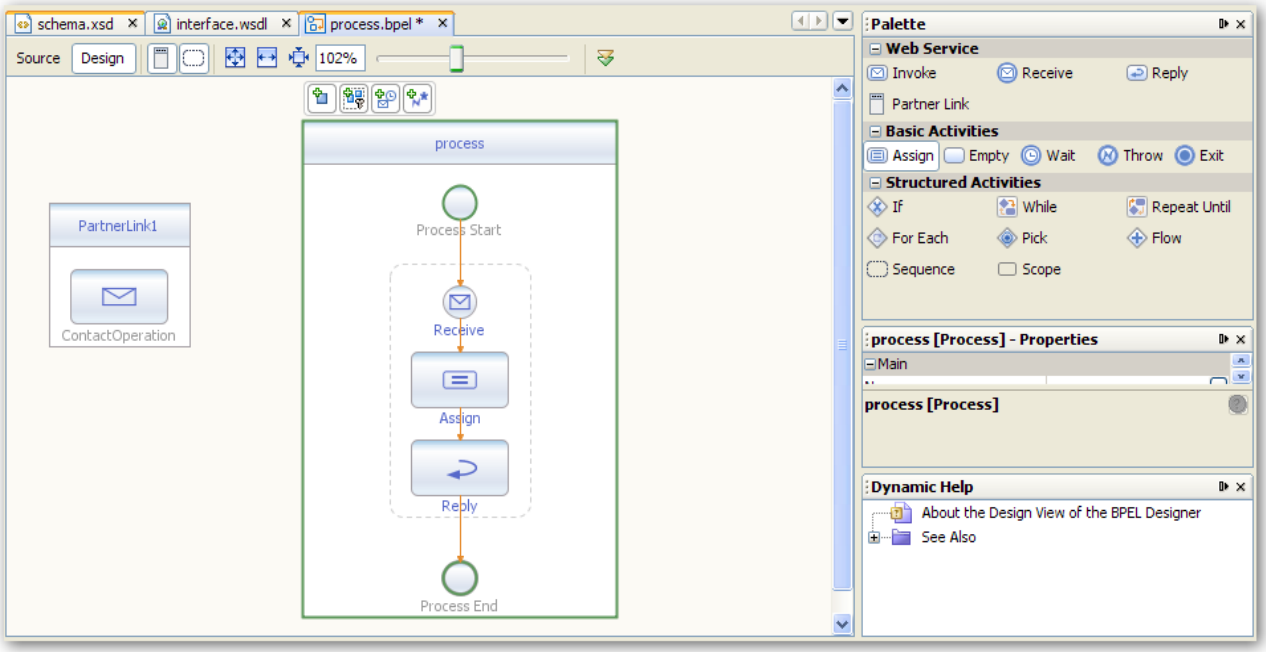
Εικόνα 3.5: Αρχικό διάγραμμα διεργασίας

Σέρνοντας το αρχείο WSDL από την καρτέλα Projects πάνω στο σχεδιάγραμμα, ανοίγει αυτόματα ένα παράθυρο δημιουργίας *partner link* και αποδεχόμενοι τις προεπιλεγμένες ρυθμίσεις, δημιουργείται το παρακάτω διάγραμμα (εικόνα 3.6):



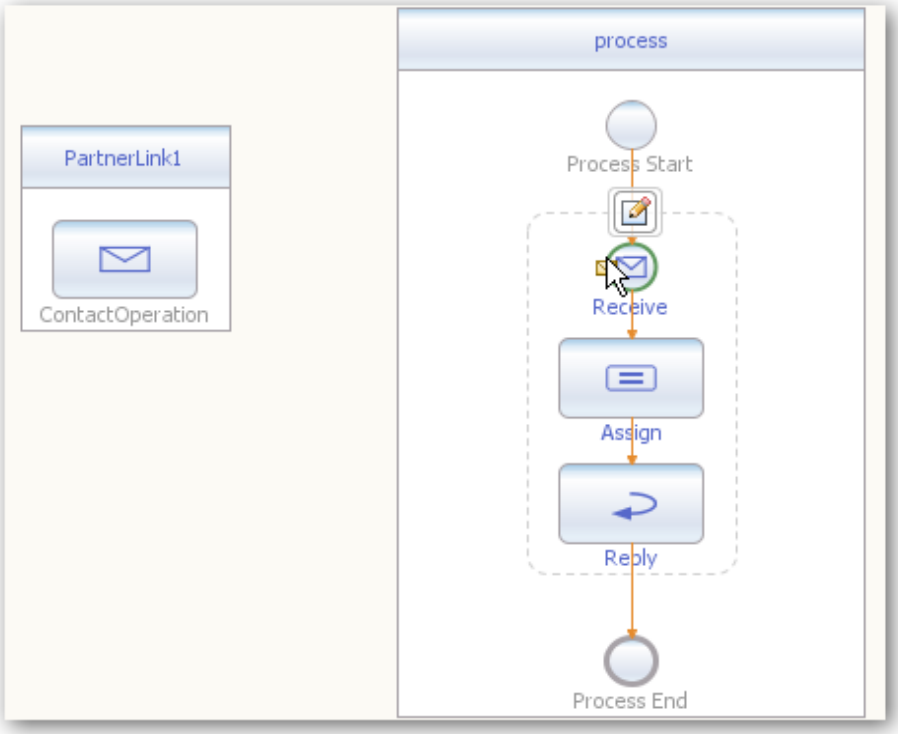
Εικόνα 3.6: Δημιουργία partner link

Διαγράφοντας από το διάγραμμα το άδειο παραλληλόγραμμο, που είναι η ενέργεια do nothing, και σέρνοντας από την παλέτα που υπάρχει στα δεξιά, τα στοιχεία Receive, Assign και Reply, δημιουργείται η εικόνα 3.7:



Εικόνα 3.7: Εμπλουτισμός της διαδικασίας.

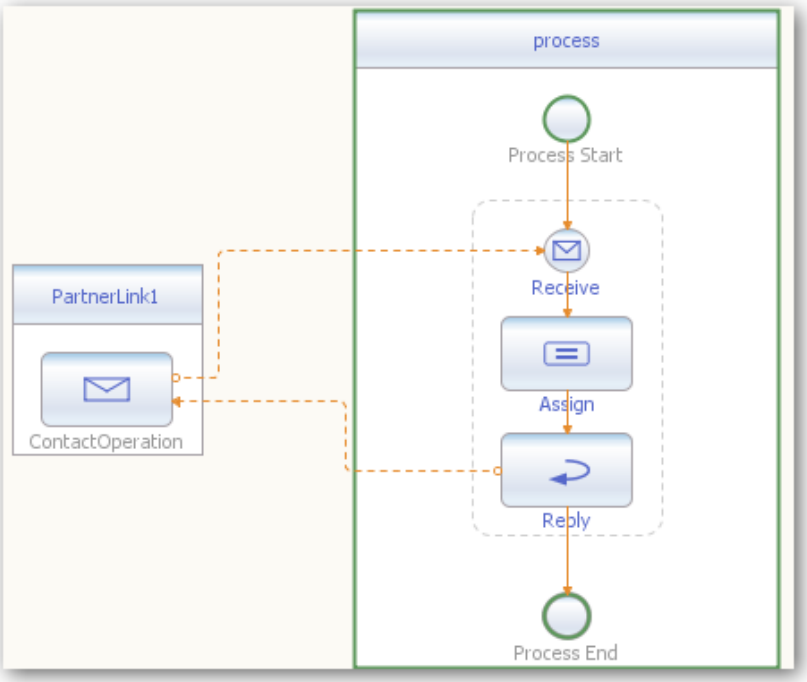
Επιλέγοντας τη δραστηριότητα Receive βλέπουμε ότι εμφανίζεται ένας κίτρινος φάκελος (μήνυμα) (εικόνα 3.8).



Εικόνα 3.8: Εισερχόμενο μήνυμα.

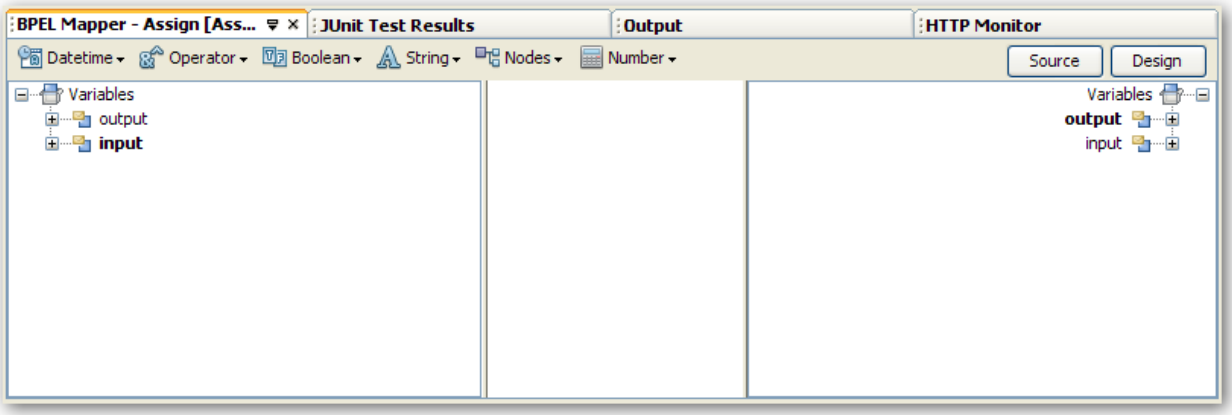
Σέρνουμε τον φάκελο αυτόν μέχρι την ContactOperation και δημιουργείται ένα βέλος που ξεκινάει από την ContactOperation και καταλήγει στο Receive. Έτσι καθορίζουμε ότι η διεργασία θα λαμβάνει το

εισερχόμενο μήνυμα από την ContactOperation. Επιλέγοντας τη δραστηριότητα Reply εμφανίζεται ένας παρόμοιος φάκελος μηνύματος. Τον σέρνουμε και αυτόν στο ContactOperation καθορίζοντας ότι εκεί θα επιστρέφεται η απάντηση. Έτσι πλέον το σχεδιάγραμμά μας γίνεται ως εξής (εικόνα 3.9):



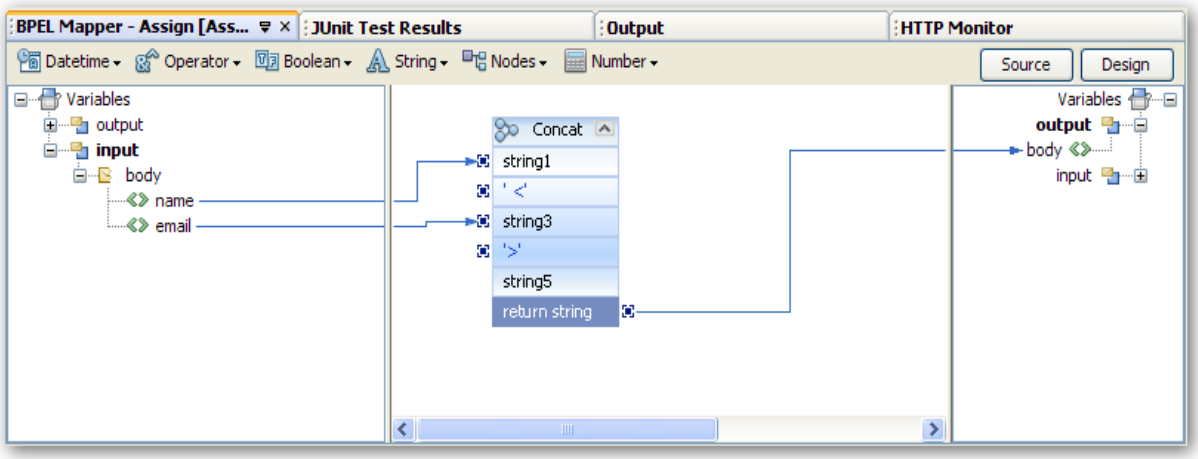
Εικόνα 3.9: Καθορισμός επιστροφής απάντησης

Τώρα χρειάζεται να δημιουργήσουμε δύο μεταβλητές. Μία για τη δραστηριότητα Receive και μία για τη δραστηριότητα Reply. Κάνουμε διπλό κλικ πάνω στη δραστηριότητα Receive, και εμφανίζεται ένα παράθυρο με τις ιδιότητές της - έτσι δημιουργούμε μια μεταβλητή με το όνομα input. Παρόμοια δημιουργούμε μια μεταβλητή Normal Output με το όνομα output για τη δραστηριότητα Reply. Στη συνέχεια ρυθμίζουμε τη δραστηριότητα *Assign*. Κάνουμε δεξί κλικ πάνω της και επιλέγουμε *Show BPEL Mapper*. Στο κάτω μέρος της οθόνης θα εμφανισθεί ο BPEL Mapper (εικόνα 3.10).



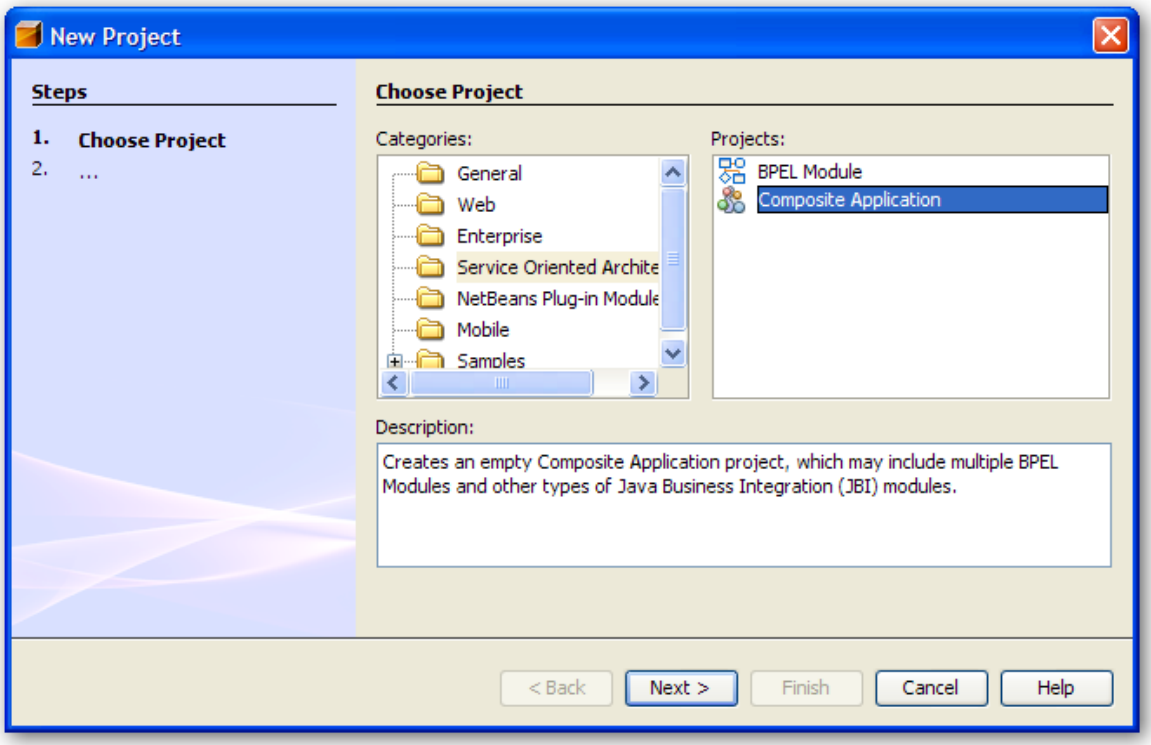
Εικόνα 3.10: BPEL Mapper

Από το drop-down μενού *Strings* επιλέγουμε τη συνάρτηση *Concat*, και τη διαμορφώνουμε όπως φαίνεται στην παρακάτω εικόνα (εικόνα 3.11):



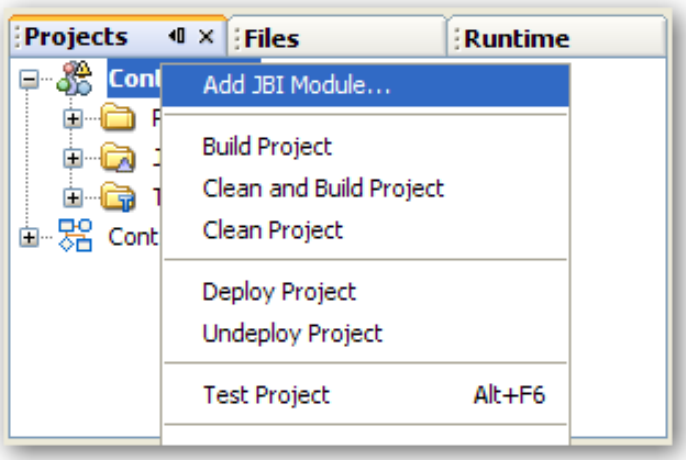
Εικόνα 3.11: Η Συνένωση (Concatenation)

Η συνάρτηση Concat χρησιμοποιείται για να συνενώσει δύο αλφαριθμητικά. Επομένως, η παραπάνω συνάρτηση Concat δημιουργεί το αλφαριθμητικό "name <email>" και το εκχωρεί στην μεταβλητή output. Για να χρησιμοποιήσουμε τη διεργασία BPEL που δημιουργήσαμε πρέπει να δημιουργήσουμε μια σύνθετη εφαρμογή (*Composite Application)* στην οποία θα προσθέσουμε μια ενότητα *JBI* (Java Business Interaction) που είναι το BPEL project μας. Έτσι, επιλέγουμε New Project στην κατηγορία Service Oriented Architecture και Composite Application από το πεδίο Projects (εικόνα 3.12).



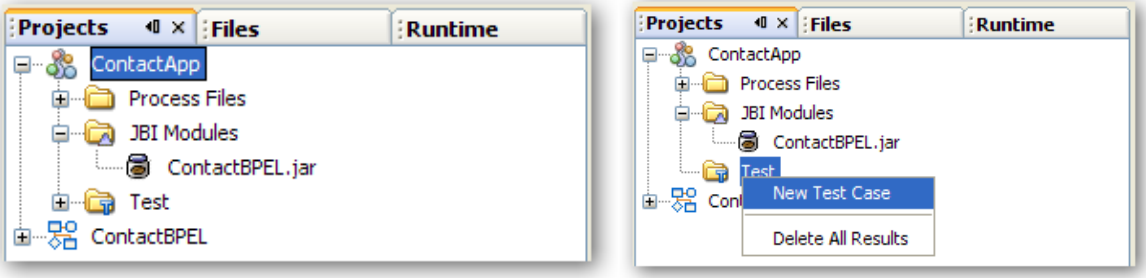
Εικόνα 3.12: New Project: Composite Application

Έστω ότι επιλέγουμε ως όνομα του project το ContactAp. Στη συνέχεια, πατώντας δεξί κλικ πάνω στο όνομα του project, επιλέγουμε Add JBI Module (εικόνα 3.13).



Εικόνα 13: Δημιουργία JBI Module

Στο παράθυρο που ανοίγει βρίσκουμε τον κατάλογο του project ContactBPEL και πατάμε OK. Τώρα βλέπουμε ότι έχει προστεθεί το αρχείο ContactBPEL.jar κάτω από τον φάκελο JBI Modules. Για να ελέγξουμε αν λειτουργεί σωστά το BPEL project μας πρέπει να δημιουργήσουμε ένα Test Case. Κάνουμε δεξί κλικ πάνω στον φάκελο Test και επιλέγουμε New Test Case (εικόνα 3.14).



Εικόνα 14: Δημιουργία New Test Case

Βιβλιογραφικές αναφορές

[1] Andrew S. Tanenbaum, Maarten Van Steen (2006), Κατανεμημένα Συστήματα: Αρχές και Υποδείγματα, Έκδοση: 1η/2006, Εκδόσεις Κλειδάριθμος.

[2] Coulouris, J. Dollimore, T. Kindberg, G. Blair (2020), Κατανεμημένα Συστήματα, Έκδοση: 2η, DA VINCI M.E.Π.Ε.

[3] Κάβουρας Ι.Κ., Μήλης Ι.Ζ., Ρουκουνάκη Α.Α., Ξηλωμένος Γ.Β. (2011), Κατανεμημένα συστήματα σε Java, 3η έκδοση, Εκδόσεις Κλειδάριθμος ΕΠΕ.

[4] Dr. Cesare Pautasso, Distributed Systems Web Services, Computer Science Department, Swiss Federal Institute of Technology (ETHZ), [Online], Available at: https://people.inf.ethz.ch/pautasso

[5] Darrel Ince (2007), Κατανεμημένες εφαρμογές και ηλεκτρονικό εμπόριο, Εκδόσεις Πανεπιστημίου Μακεδονίας, [eBook], Θεσσαλονίκη

[6] Κατανεμημένα Αντικείμενα με Java RMI, Parallel Distributed Processing Laboratory (PDP Lab), Department of Applied Informatics, University of Macedonia, Θεσσαλονίκη

[7] Tutorials Point simply easy learning, Java RMI – Introduction & Java RMI - RMI Application, Java RMI Tutorial, [Online], Available at: tutorialspoint.com.

[8] Εισαγωγή στα Web services, Parallel Distributed Processing Laboratory (PDP Lab), Department of Applied Informatics, University of Macedonia, Θεσσαλονίκη

[9] Παπαργύρη Τ. (2012), Σχεδιαση και Αναπτυξη WEB SERVICE, Διπλωματική Εργασία, Πανεπιστήμιο Πατρών, Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών, Πάτρα

[10] Clement Adedeji (2020), Webservices; Soap And Rest- A Simple Introduction, Reply Solidsoft, [Online], Available at: reply.com.

[11] Thomas Erl (2004), Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)

Κριτήρια αξιολόγησης

Ερώτηση 1

Τι είναι η πρόταξη;

α. Να βάλω τις μεταβλητές σε μια σειρά πριν την αποστολή τους.

β. Η σειριακοποίηση των δεδομένων πριν την αποστολή τους μέσω δικτύου

γ. Η αποσειριακοποίηση των δεδομένων πριν την αποστολή τους μέσω δικτύου

δ. Η σειριακοποίηση των δεδομένων πριν το πέρασμά τους σε μια διαδικασία.

Ερώτηση 2

Τι είναι η Υποδοχή (sockect);

α. Η διεύθυνση δικτύου

β. Η θύρα μιας υπηρεσίας.

γ. Η θύρα μιας υπηρεσίας σε συνδυασμό με το IP του υπολογιστή στον οποίο βρίσκεται.

δ. Η θύρα του δωματίου υπολογιστή.

Ερώτηση 3

Το RPC προσφέρει διαφάνεια τοποθεσίας:

α. Προσφέρει αδιαφάνεια τοποθεσίας στον χρήστη.

β. Προσφέρει διαφάνεια τοποθεσίας και στον χρήστη και στον προγραμματιστή.

γ. Προσφέρει διαφάνεια τοποθεσίας στον προγραμματιστή.

δ. Προσφέρει διαφάνεια τοποθεσίας στον χρήστη.

Ερώτηση 4

Ποιο από τα πρωτόκολλα μεταφοράς είναι αξιόπιστο;

α. UDP

β. TCP

γ. UDP/TCP

δ. Κανένα από τα δύο.

Ερώτηση 5

Τι είναι τα stub και skeleton;

α. Πληρεξούσιοι της υλοποίησης ενός απομακρυσμένου αντικειμένου για την επικοινωνία πελάτη – εξυπηρετητή.

β. Ο κορμός (stub) ενός δέντρου σε μια δεντρική δομή δεδομένων.

γ. Ο σκελετός (skeleton) μιας δομής ενός προγράμματος.

δ. Τίποτα από τα παραπάνω.

Ερώτηση 6

Τι ρόλο παίζει η διεπαφή (interface) στο Java RMI;
α. Κανέναν απολύτως.
β. Περιέχει το όνομα και τις παραμέτρους των μεθόδων που υλοποιεί το απομακρυσμένο αντικείμενο.
γ. Περιέχει τις υλοποιήσεις των μεθόδων που υλοποιεί το απομακρυσμένο αντικείμενο.
δ. Περιέχει το όνομα και τις παραμέτρους των μεθόδων μιας διεπαφής.

Ερώτηση 7
Τι είναι το RMI registry;
α. Μια υπηρεσία που αφορά τα προγράμματα ενός υπολογιστή.
β. Μια υπηρεσία μητρώου ασφαλισμένων.
γ. Μια υπηρεσία μητρώου εγγραφής απομακρυσμένων αντικειμένων με τις αναφορές τους.
δ. Μια υπηρεσία μητρώου εγγραφής αντικειμένων με τις αναφορές τους στο σύστημα CORBA.

Ερώτηση 8
Το CORBA ως γλώσσα περιγραφής των διεπαφών των απομακρυσμένων αντικειμένων έχει:
α. C++
β. Java
γ. Remote Interface
δ. IDL

Ερώτηση 9
Το DCOM ως πλατφόρμα λειτουργεί:
α. Μόνο σε Microsoft Λειτουργικά Συστήματα.
β. Όχι μόνο σε Windows, αλλά και σε Unix.
γ. Δεν υποστηρίζει κατανεμημένες λειτουργίες.
δ. Τίποτα από τα παραπάνω.

Ερώτηση 10
Τι είναι το WSDL;
α. Υπηρεσία.
β. Μητρώο.
γ. Διεπαφή.
δ. Κατανεμημένο Αντικείμενο.

Ερώτηση 11
Τι είναι το UDDI;
α. Εφαρμογή.
β. Υπηρεσία Ανακάλυψης και Ευρετηρίασης.
γ. Διεπαφή.
δ. Κατανεμημένο Αντικείμενο.

Ερώτηση 12
Τι ισχύει για τα Restful Web Services;
α. Ευκολία στην υλοποίηση και στην επικοινωνία.
β. Μη κλιμάκωση υπηρεσιών.
γ. Βαριές υπηρεσίες.
δ. Δύσχρηστες στην επικοινωνία.

Ερώτηση 13

Τι είναι η BPEL;

α. Μια γλώσσα προγραμματισμού χαμηλού επίπεδου.

β. Έχει να κάνει με την αποσύνθεση υπηρεσιών ιστού.

γ. Συνθέτει νέες υπηρεσίες ιστού χρησιμοποιώντας τις ήδη υπάρχουσες.

δ. Τίποτα από τα παρπάνω.

Ερώτηση 14

Ποια τεχνολογία θα προτιμούσατε για μια κατανεμημένη εφαρμογή;

α. Ανάλογα με τις απαιτήσεις της εφαρμογής.

β. Java RMI.

γ. CORBA.

δ. Web Services.

Ερώτηση 15

Ποια η σχέση των κατανεμημένων εφαρμογών και των κατανεμημένων αντικειμένων;

α. Τα κατανεμημένα αντικείμενα και οι κατανεμημένες εφαρμογές σχετίζονται ως προς την αποδοτικότητά τους.

β. Τα κατανεμημένα αντικείμενα και οι κατανεμημένες εφαρμογές είναι ασυσχέτιστα.

γ. Τα κατανεμημένα αντικείμενα βασίζονται στις κατανεμημένες εφαρμογές.

δ. Οι κατανεμημένες εφαρμογές βασίζονται στα κατανεμημένα αντικείμενα.