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

ΜΕΛΕΤΗ ΡΟΩΝ ΕΡΓΑΣΙΑΣ ΣΕ GRIDS ΑΠΟ ΔΙΚΤΥΑ ΑΙΣΘΗΤΗΡΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

ΕΥΑΝΘΙΑΣ Α. ΒΑΛΕΡΑ

Επιβλέπων : Βασίλειος Μάγκλαρης

Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2007

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

ΜΕΛΕΤΗ ΡΟΩΝ ΕΡΓΑΣΙΑΣ ΣΕ GRIDS ΑΠΟ ΔΙΚΤΥΑ ΑΙΣΘΗΤΗΡΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΕΥΑΝΘΙΑ Α. ΒΑΛΕΡΑ

Επιβλέπων : Βασίλειος Μάγκλαρης

Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 28η Ιουνίου 2007.

.....	.....	.....
Β. Μάγκλαρης	Σ. Παπαβασιλείου	Γ. Στασινόπουλος
Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής	Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2007

.....

ΕΥΑΝΘΙΑ Α. ΒΑΛΕΡΑ

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

ΠΕΡΙΛΗΨΗ

Ο όρος Web service αναφέρεται σε υπηρεσίες που βασίζονται στο Web (HTTP, XML) για να καταστεί δυνατή η ανταλλαγή μηνυμάτων με τους πελάτες ανεξαρτήτως προέλευσης. Η εφαρμογή πελάτης ενημερώνεται από το WSDL αρχείο της υπηρεσίας σχετικά με την δομή των μηνυμάτων που πρέπει να σταλούν και έτσι δύναται να χρησιμοποιήσει την συγκεκριμένη υπηρεσία.

Οι διάφορες υπηρεσίας μπορούν να συνδυαστούν σε ροές εργασίας οι οποίες καθορίζουν τον τρόπο, την σειρά ακόμα και τις συνθήκες που απαιτούνται για να κλιθούν οι διάφορες Web services.

Στην εργασία αυτήν εφαρμόζουμε μία ροή εργασίας η οποία καλεί υπηρεσίες οι οποίες με την σειρά τους ελέγχουν ένα ασύρματο ad-hoc δίκτυο αισθητήρων. Οι αισθητήρες αυτοί μετράνε μεταξύ άλλων την θερμοκρασία του περιβάλλοντος και μέσω μίας εφαρμογής πελάτη της ροής, τις παρουσιάζουν γραφικά στον χρήστη.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

Service Oriented Architecture (SOA), Web Services, Web Services Description Language (WSDL), Simple Object Access Protocol (SOAP), Web Service Resource Framework (WSRF), Muse, Business Process Execution Language (BPEL), ActiveBPEL Engine, Tomcat, Axis, Grids, Tmote sensors.

ABSTRACT

Web services, in the general meaning of the term, are web services offered via the Web. They are accessible through standard-based Internet protocols such as HTTP and use a standardized XML messaging system in order to communicate with clients who use different information systems and programming languages. The Web service can be invoked by a client application which, taking into account the information provided in the WSDL description, is able to send the appropriate messages via SOAP.

Many different Web services can be combined in business processes, which define the way, the sequence and even the transition conditions for the invocation of the available Web services.

The subject of this thesis is to deploy a business process which will invoke some services. These services control a wireless ad-hoc sensor network, consisting of sensors that collect, among others, measurements of temperature from their surroundings. Finally using a client application the user is able to view graphically the results of the measurements of every sensor node in the network.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ

Service Oriented Architecture (SOA), Web Services, Web Services Description Language (WSDL), Simple Object Access Protocol (SOAP), Web Service Resource Framework (WSRF), Muse, Business Process Execution Language (BPEL), ActiveBPEL Engine, Tomcat, Axis, Grids, Tmote sensors.

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω αρχικά τον επιβλέποντα καθηγητή κ. Βασίλειο Μάγκλαρη για την ευκαιρία που μου έδωσε να ασχοληθώ με την διπλωματική μου εργασία.

Επιπλέον, θα ήθελα να ευχαριστήσω τον υπεύθυνο της εργασίας μου, υποψήφιο διδάκτορα Λένη Άγγελο, για την καθοδήγηση και την πολύτιμη βοήθεια καθ' όλη τη διάρκεια της προετοιμασίας της εργασίας και κυρίως για την συμβολή του σε στιγμές που τα εμπόδια φαινόταν άλυτα.

Επίσης, θα ήθελα να ευχαριστήσω την υποψήφια διδάκτορα Πουλή Βασιλική για την βοήθεια και την υποστήριξή της για την περάτωση της εργασίας αυτής.

Τέλος, ευχαριστώ πολύ την οικογένεια μου για την ένθερμη υποστήριξη που μου προσέφερε κατά την διάρκεια της διπλωματικής μου εργασίας.

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΕΡΙΛΗΨΗ 5

ABSTRACT 6

ΕΥΧΑΡΙΣΤΙΕΣ 7

ΠΕΡΙΕΧΟΜΕΝΑ 9

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ 13

ΕΙΣΑΓΩΓΗ 15

ΚΕΦΑΛΑΙΟ 1: WEB SERVICES 17

1,1 XML 17

1.1.1 Τι είναι η XML……...17
1.1.2 Δομικές μονάδες της XML……..…18
1.1.3 DTD…….....18
1.1.4 XML Schema……...…...19
1.1.5 Κυριότερες διαφορές μεταξύ DTD και XML Schema..…….19

1.2 Simple Object Access Protocol 20

1.2.1 Ορισμός του SOAP…….……....20
1.2.2 Στόχοι του SOAP…….……...20
1.2.3 Προκάτοχοι του SOAP……...20
1.2.4 Το SOAP Envelope Framework…….22
1.2.4.1 SOAP Envelope……...……..23
1.2.4.2 SOAP Headers……...……23
1.2.4.3 SOAP Body……...……24
1.2.4.4 SOAP Versioning…...…….…...24
1.2.5 SOAP Messaging.……...25
1.2.5.1 Αριθμός συμμετεχόντων……25
1.2.5.2 Τρόποι αλληλεπίδρασης…..…….…….....26
1.2.5.3 Συγχρονισμός της ανταλλαγής μηνυμάτων……27
1.2.5.4 Απευθείας η με χρήση ουράς ανταλλαγή μηνυμάτων……27

1.2.5.5 Ποιότητα υπηρεσίας (Quality of Service)……...28
1.2.5.6 Μορφή μηνυμάτων…….29

1.2.6 SOAP-XML attributes……....29
1.2.7 Λάθη SOAP……....30

1.3 Service Oriented Architecture – SOA 31

1.3.1 SOA – Αρχιτεκτονική των Web Services……..….31

1.4 Web Service Description Language – WSDL 34

1.4.1 Ορισμός και σκοπός της WSDL……..34
1.4.2 Τεχνική περιγραφή της WSDL……... 35
1.4.3 Παράδειγμα WSDL…….36

1.5 Web services 39

1.5.1 Τι είναι τα Web services…….…39
1.5.2 Πλεονεκτήματα, μειονεκτήματα και οφέλη……. 40
1.5.3 Παραδείγματα κλήσης μίας Web service……42
1.5.4 Τεχνικά Χαρακτηριστικά των Web services……...43
1.5.4.1 Μοντέλο Επιπέδων…….43
1.5.4.1.1 Επίπεδο Μεταφοράς……....43
1.5.4.1.2 Επίπεδο Μηνυμάτων……...43
1.5.4.1.3 Επίπεδο Περιγραφής……...43
1.5.4.2 Κατάλογος Υπηρεσιών……...44
1.5.4.3 Τύποι καταχωρήσεων……44
1.5.4.4 Κατηγορίες Υλοποίησης…….46
1.5.4.4.1 Απλή Υπηρεσία……..46
1.5.4.4.2 Σύνθετη Υπηρεσία…….….47
1.5.4.4.3 Ενδιάμεση Υπηρεσία…….….47
1.5.4.4.4 Διάδρομος Υπηρεσιών……..…..48

ΚΕΦΑΛΑΙΟ 2: WSRF – MUSE 50

2.1 Web Service Resource Framework – WSRF 51

2.1.1 Τι είναι το WSRF…… 51
2.1.2 Η WSRF προδιαγραφή……52

2.1.3 WS-Resource Access Pattern Embodiments……...55
2.1.4 WS-Addressing……...55
2.1.5 WSDL Service Element Embodiment……57
2.1.6 WS – Resource Document……58
2.1.7 Notifications……...58

2.2 Muse 58

2.2.1 Τι είναι το Muse……..58
2.2.2 Σχηματισμός resources με την χρήση capabilities……..60
2.2.3 Ο deployment descriptor του Muse……65

ΚΕΦΑΛΑΙΟ 3: Business Process Execution Language-BPEL66

3.1 Τι είναι η BPEL……66
3.2 Είδη εφαρμογών SOA που είναι κατάλληλα για την BPEL..68
3.3 Σκοπός της BPEL……..69
3.4 Βασικές έννοιες της BPEL……69
3.4.1 Partner Links……..…..70
3.4.2 Partner Link Types……..….70
3.4.3 Variables……..…….71
3.5 Activities……72
3.6 Fault Handlers……73
3.7 Event Handlers……...74
3.8 Messages και Correlation……...75
3.9 H Pick Activity……...75
3.10 Εκκίνηση της ροής εργασίας……..76
3.11 Παράδειγμα BPEL διαδικασίας……..76
3.11.1 Δημιουργία του αρχείου BPEL……...77

ΚΕΦΑΛΑΙΟ 4: GRIDS 81

4.1 Τι είναι τα Grids…..……..…..81
4.2 Εξέλιξη των τεχνολογιών Grid……82

ΚΕΦΑΛΑΙΟ 5: Ad-hoc δίκτυα αισθητήρων 84

5.1 Δίκτυα αισθητήρων 84

5.1.1 Τι είναι ένα ασύρματο δίκτυο αισθητήρων……..……84

5.1.2 Ad-hoc δίκτυα αισθητήρων……84
5.1.3 Στόχοι και απαιτήσεις ενός δικτύου αισθητήρων……85
5.1.4 Αρχιτεκτονικές συστήματος……87

5.2 Αισθητήρες 88

5.2.1 Η χρησιμότητα των αισθητήρων……88
5.2.2 Λειτουργικά συστήματα δικτύων αισθητήρων……...89

ΚΕΦΑΛΑΙΟ 6: Αρχιτεκτονική της εφαρμογής 90

6.1 Εγκατάσταση των απαραίτητων εργαλείων 90

6.1.1 Tomcat……90
6.1.1.1 Έκδοση του Tomcat……..…..90
6.1.1.2 Εγκατάσταση του Tomcat……..….90
6.1.2 AXIS……...91
6.1.2.1 Έκδοση του AXIS……..91
6.1.2.2 Τι είναι ο AXIS……..….91
6.1.3 Εγκατάσταση του AXIS…….……92
6.1.3.1 JUDDI…….…....92
6.1.3.2 Εγκατάσταση του Juddi…….....92
6.1.3.3 UDDIBrowser…….…....93
6.1.4 ActiveBPEL……95
6.1.4.1 Έκδοση της ActiveBPEL Engine……95
6.1.4.2 Τι είναι η ActiveBPEL Engine……95
6.1.4.3 Αρχιτεκτονική της ActiveBPEL Engine……96
6.1.4.4 Εκκίνηση της ActiveBpel Engine……97
6.1.4.5 Εγκατάσταση της ActiveBPEL Engine……....97
6.1.5 ActiveBPEL Designer……...……..99
6.1.5.1 Τι είναι ο ActiveBPEL Designer …….……99
6.1.6 WSRF……..…100
6 .1.6.1 Έκδοση του WSRF…….……..…..100
6.1.6.2 Εγκατάσταση…….…...100
6.1.7 Ant……..…106
6.1.7.1 Τι είναι το Ant……..……..…106
6.1.8 TinyOS…….…...106
6.1.8.1 Τι είναι το TinyOS……....106
6.1.8.2 Έκδοση του TinyOS……..…107
6.1.8.3 Σχεδίαση του TinyOS……...107

6.1.8.4 Εγκατάσταση του TinyOS…….……...109
6.1.8.5 Η γλώσσα προγραμματισμού nesC……..…….….112
6.1.9 Tmote Sky…….....113
6.1.9.1 Τι είναι τα Tmote Sky……...113
6.1.9.2 Εγκατάσταση των Tmote Sky……...….....115
6.1.9.3 Προγραμματισμός των Tmote Sky……115
6.1.9.4 Trawler.....……..116
6.2 Υλοποίηση της εφαρμογής 118

6.2.1 Αρχιτεκτονική…….…….118

Ευρετήριο σχημάτων

Σχήμα 1: Παράδειγμα XML……..17
Σχήμα 2: Μορφή SOAP μηνύματος ……..……..22
Σχήμα 3: Σκελετός ενός SOAP μηνύματος……..23
Σχήμα 4: Αριθμός συμμετεχόντων……...26
Σχήμα 5: Τρόποι αλληλεπίδρασης ……...27
Σχήμα 6: Ανταλλαγή μηνυμάτων ……28
Σχήμα 7: Λάθη SOAP μηνυμάτων ……...30
Σχήμα 8: SOA αρχιτεκτονική ……..31
Σχήμα 9: Κλήση μίας Web service……42
Σχήμα 10: Απλή υπηρεσία ……....46
Σχήμα 11: Σύνθετη υπηρεσία ……...47
Σχήμα 12: Ενδιάμεση υπηρεσία ……....47
Σχήμα 13: Διάδρομος υπηρεσιών ……48
Σχήμα 14: Μοντέλο των Web services ……....49
Σχήμα 15: WSRF και SOA ……..51
Σχήμα 16: Stateless Web service ……..51
Σχήμα 17: Statefull Web service ……..52
Σχήμα 18: WS-Resource ……..53
Σχήμα 19: Συλλογή από WS-Resources ……..54
Σχήμα 20: Αρχές σχεδίασης του Apache Muse ……...59
Σχήμα 21: Συνένωση υλοποίησης του Muse ……...61
Σχήμα 22: Σχέση της BPEL με τα standards των Web services ……..67
Σχήμα 23: Σχέση BPEL και WSDL ……....67
Σχήμα 24: Σχηματισμός μιας νέας υπηρεσίας από ήδη υπάρχουσες ……...69
Σχήμα 25: Παράδειγμα ροής εργασίας ……76
Σχήμα 26: Απεικόνιση του Grid ……...82
Σχήμα 27: Δίκτυο αισθητήρων ……....86
Σχήμα 28: Αρχιτεκτονική της ActiveBpel Engine ……....96
Σχήμα 29: Η διασύνδεση της σελίδα http://localhost:8080/BpelAdmin ….....98
Σχήμα 30: Η διασύνδεση της σελίδας http://localhost:8080/ActiveBPEL_Tutorial..98
Σχήμα 31: Η διασύνδεση του ActiveBPEL Designer ……99
Σχήμα 32: Τα τμήματα του TinyOS……..108
Σχήμα 33: Η αρχιτεκτονική του TinyOS ……..109
Σχήμα 34: Μπροστινό και πίσω μέρος των Tmote Sky ……...114
Σχήμα 35: Αρχιτεκτονική της εφαρμογής ……118

ΕΙΣΑΓΩΓΗ

Σκοπός της διπλωματικής εργασίας είναι να διαβάσουμε τις μετρήσεις του περιβάλλοντος από τους κόμβους αισθητήρων που διαθέτουμε και οι οποίοι απαρτίζουν ένα ασύρματο ad-hoc δίκτυο αισθητήρων, καλώντας μία Web service. Οι κόμβοι αισθητήρων όμως αποτελούν stateful resources και έτσι για την μοντελοποίηση και την πρόσβαση σε αυτούς θα χρησιμοποιηθεί η προδιαγραφή Muse που αποτελεί το τελευταίο implementation της προδιαγραφής WSRF- Web Service Resource Framework. Στην συνέχεια θα συνδυάσουμε σε μία ροή εργασίας τις υπηρεσίες Ιστού που ελέγχουν τους αισθητήρες σε διαφορετικούς υπολογιστές.

Οι έννοιες που περιλαμβάνουν το μελετητικό τμήμα της εργασίας και οι οποίες θα αναλυθούν στα διάφορα κεφάλαια αυτής, περιλαμβάνουν τους όρους Service Oriented Architecture – SOA, Web services, Muse και Web Service Resource Framework, Business Process Execution Language – BPEL, tmote Sky sensors, TinyOS και Cygwin.

Στα κεφάλαια που θα ακολουθήσουν θα αναλυθούν οι ανωτέρω αναφερθείσες έννοιες, οι οποίες είναι αναγκαίες για την κατανόηση της ανάλυσης της υλοποίησης της εφαρμογής που πραγματοποιείται στο τελευταίο κεφάλαιο.

Συγκεκριμένα, στο πρώτο κεφάλαιο αναλύονται όλες οι τεχνολογίες, οι οποίες σχετίζονται με τις Web services – υπηρεσίες Ιστού, όπως XML, SOAP, SOA, WSDL καθώς και με αυτές καθαυτές τις υπηρεσίες Ιστού. Στο δεύτερο κεφάλαιο θα αναπτύξουμε τις προδιαγραφές WSRF και Muse.

Στην συνέχεια, στο κεφάλαιο τρία θα ασχοληθούμε με την ανάπτυξη της γλώσσας περιγραφής των ροών εργασίας BPEL.

Στο κεφάλαιο τέσσερα θα επιχειρήσουμε να ορίσουμε και να περιγράψουμε το Grid. Στο επόμενο, στο κεφάλαιο πέντε, θα περιγράψουμε τα ασύρματα ad-hoc δίκτυα αισθητήρων και θα αναφερθούμε στην χρησιμότητα και στα χαρακτηριστικά των κόμβων που απαρτίζουν τα δίκτυα αυτά.

Τέλος, στο κεφάλαιο έξι περιγράφεται αναλυτικά η σχεδίαση και η υλοποίηση της εφαρμογής μας.

1 Web services και τεχνολογίες τους

1.1 XML

1.1.1 Τι είναι η XML

Η XML (eXtensible Markup Language) είναι μια γλώσσα ανεξάρτητη από σύστημα και hardware για την αναπαράσταση δεδομένων σε ένα XML document. Η XML χρησιμοποιείται παγκοσμίως ως κοινώς αποδεκτή μεταγλώσσα σήμανσης για την ανταλλαγή πληροφοριών. Είναι επεκτάσιμη καθώς αποτελεί ένα σύνολο προκαθορισμένων κανόνων που πρέπει να ακολουθήσουμε κατά τη δόμηση των δεδομένων μας. Παρέχει μία πρότυπη και κοινή δομή για τη διανομή δεδομένων μεταξύ ανόμοιων συστημάτων και επιπλέον έχει ενσωματωμένο ένα μηχανισμό επικύρωσης δεδομένων, που εγγυάται την εγκυρότητα της δομής των δεδομένων.

Παράδειγμα XML

Ας δούμε πώς αναπαριστούμε δεδομένα με τη βοήθεια της XML σε ένα παράδειγμα όπου αναπαριστούμε τις προσωπικές πληροφορίες και τις πληροφορίες σχετικά με τις βάρδιες ενός υπαλλήλου σε έναν οργανισμό.

```
<employee>
      <shift id= "counter" time="4-12">
            <phone id = "1"> All phone information
                  <number>12345678</number >
            </phone>
       </shift >
       <shift id="help_desk" time="1-5">
             <phone id = "2"> All phone information
                   <number>12345678</number >
             </phone>
        </shift >
            ...
        <home-address>
              <street>3 Mukonou</street>
              <city>Athens</city>
        </home-address>
</employee>
```

Σχήμα 1: Παράδειγμα XML

Η XML χρησιμοποιεί τις διακριτικές ετικέτες “<>” και “</>” όπως και η HTML, αφού αποτελεί και αυτή μια γλώσσα σήμανσης. Οι διαφορές τους είναι κυρίως ως προς τον σκοπό της καθεμίας. Ενώ η XML σχεδιάστηκε για να περιγράφει δεδομένα και να εστιάσει στο τι είναι τα δεδομένα αυτά, η HTML σχεδιάστηκε για να προβάλει δεδομένα και να εστιάσει στο πώς φαίνονται αυτά τα δεδομένα.

1.1.2 Δομικές μονάδες της XML

Οι δύο αρχικές δομικές μονάδες που χρησιμοποιούνται στη δόμηση της XML είναι τα elements και τα attributes.

Elements (Στοιχεία)

Τα elements είναι ετικέτες, δομημένες σαν δένδρο όπως και στην HTML, που περιέχουν τιμές. Τα elements είναι οργανωμένα με ιεραρχικό τρόπο με ένα στοιχείο-πατέρα και στοιχεία-παιδιά.

Attributes (Ιδιότητες)

Τα attributes περιγράφουν τα στοιχεία αποτελεσματικά και με σαφήνεια, με σκοπό να παρέχουν περισσότερες πληροφορίες σχετικά με ένα στοιχείο και όχι για να παρέχουν τα ίδια τα δεδομένα. Στο παραπάνω παράδειγμα, το στοιχείο <shift> έχει μία ιδιότητα “id” με τιμές “counter” και “help_desk”. Αυτό βοηθάει στο να κάνουμε τα δεδομένα σε ένα έγγραφο XML αυτοπεριγραφικά.

Τα στοιχεία και οι ιδιότητες στην XML πρέπει να ακολουθούν κάποιους κανόνες ονομασίας. Μπορούν να περιέχουν γράμματα, αριθμούς και άλλους χαρακτήρες, αλλά δεν μπορούν να ξεκινούν με αριθμό ή άλλους χαρακτήρες. Επίσης δεν επιτρέπεται να ξεκινούν με την γραμματοσειρά xml και να περιέχουν κενά.

1.1.3 DTD

Το Document Type Definition (DTD) είναι μία προδιαγραφή, η οποία πρέπει να ακολουθηθεί όταν δημιουργείται ένα έγγραφο XML, όπως και σε μία γλώσσα προγραμματισμού πρέπει να ακολουθούνται οι προδιαγραφές της. Άλλη μία παρομοίωση μπορεί να γίνει μεταξύ των μεταγλωττιστών κάθε γλώσσας προγραμματισμού και των XML parsers οι οποίοι χρησιμοποιούν το DTD για να ελέγξουν την εγκυρότητα ενός XML εγγράφου. Το DTD βοηθάει στον καθορισμό της δομής ενός εγγράφου XML παρέχοντας ένα αυστηρό πλαίσιο και κανόνες που πρέπει να ακολουθηθούν για την δημιουργία εγγράφων XML. Επιπλέον, το DTD χρησιμοποιείται για τον έλεγχο της εγκυρότητας και της ακεραιότητας των δεδομένων που περιέχονται σε ένα έγγραφο XML. Ένα DTD μπορεί να χρησιμοποιηθεί απευθείας μέσα σε ένα έγγραφο XML ή μπορεί να υπάρχει εκτός του εγγράφου XML οπότε και αναφέρεται με ένα δεσμό μέσα στο έγγραφο XML.

Το DTD αποτελείται από τα στοιχεία elements, attributes και entities. Τα elements αποτελούν metadata για το στοιχείο. Καθορίζουν το είδος των δεδομένων του στοιχείου, τον αριθμό των περιστατικών του, καθώς και τις σχέσεις του με άλλα στοιχεία. Τα attributes καθορίζουν διάφορους κανόνες και ορισμούς των δεδομένων. Τέλος, τα entities χρησιμοποιούνται για να αναφερθούμε σε εξωτερικά αρχεία ή για να χρησιμοποιήσουμε συντομεύσεις σε κάποιο κείμενο.

1.1.4 XML Schema

Το DTD παρουσιάζει μειονεκτήματα, όπως το ότι δεν υποστηρίζει ισχυρούς τύπους δεδομένων, έχει σύνταξη διαφορετική από την XML και δεν είναι επεκτάσιμο. Έτσι προτάθηκε το XML Schema που είναι μια προηγμένη έκδοση του DTD.

Το XML Schema έχει σύνταξη όμοια με την XML, άρα μπορούμε να το επεξεργαστούμε με κάποιον XML parser. Επίσης, δεν καθορίζει μόνο τους βασικούς τύπους δεδομένων, όπως το DTD, αλλά μπορούμε να ορίσουμε και δικούς μας. Επιπλέον, το XML Schema παρέχει content-based επικύρωση, δηλαδή μπορεί να ορίσει τη σειρά με την οποία τα στοιχεία-παιδιά εμφανίζονται. Επίσης παρέχει επικύρωση στους ίδιους τους τύπους δεδομένων. Επιπρόσθετα το XML Schema υποστηρίζει την κληρονομικότητα, δηλαδή μπορούμε να παράγουμε νέους τύπους δεδομένων βάσει παλαιών. Ακόμα το XML Schema υποστηρίζει τα Namespaces παρέχοντας σε κάθε στοιχείο ένα μοναδικό αναγνωριστικό. Τέλος το XML Schema είναι επεκτάσιμο και μπορεί στο μέλλον να ενσωματώσει καινούριες λειτουργίες.

1.1.5 Κυριότερες διαφορές μεταξύ DTD και XML Schema

Οι κυριότερες διαφορές μεταξύ DTD και XML Schema μπορούν λοιπόν να συνοψιστούν στις εξής:

- Το XML Schema υποστηρίζει namespaces ενώ το DTD όχι.
- Το XML Schema χρησιμοποιεί σύνταξη XML, η οποία είναι ευκολονόητη, ενώ το DTD χρησιμοποιεί ειδική σύνταξη.
- Το XML Schema υποστηρίζει πρότυπους τύπους δεδομένων καθώς επίσης και τύπους ορισμένους από το χρήστη (user-defined) ενώ το DTD παρέχει μόνο τύπους κειμένου.
- Το XML Schema υποστηρίζει κληρονομικότητα ενώ το DTD όχι.

1.2 Simple Object Access Protocol

1.2.1 Ορισμός του SOAP

Το SOAP είναι ένα πρωτόκολλο που είναι βασισμένο στην XML και το οποίο επιτρέπει στις διάφορες εφαρμογές τη δυνατότητα να ανταλλάσουν πληροφορίες πάνω από κοινώς χρησιμοποιούμενα πρωτόκολλα.

Σύμφωνα με την W3C περιγράφεται ως: «Το SAOP 1.2 είναι ένα ελαφρύ πρωτόκολλο για την ανταλλαγή δομημένων πληροφοριών σε ένα αποκεντρωμένο, διανεμημένο περιβάλλον. Χρησιμοποιεί τεχνολογίες XML για να καθορίσει ένα επεκτάσιμο πλαίσιο παρέχοντας μια δομή μηνυμάτων η οποία μπορεί να ανταλλαχθεί πάνω από ποικίλα δικτυακά πρωτόκολλα. Το πλαίσιο έχει σχεδιαστεί να είναι ανεξάρτητο από οποιοδήποτε προγραμματιστικό μοντέλο και σημασιολογία υλοποίησης».

1.2.2 Στόχοι του SOAP

Το SOAP σχεδιάστηκε προσπαθώντας να τηρήσει δύο βασικά χαρακτηριστικά. Αρχικά την απλότητα και έπειτα την επεκτασιμότητα. Αυτό το επιτυγχάνει παραλείποντας από το πλαίσιο μηνυμάτων διάφορα χαρακτηριστικά γνωρίσματα, όπως αξιοπιστία, ασφάλεια, δρομολόγηση και patterns ανταλλαγής μηνυμάτων (Message Exchange Patterns – MEP), τα οποία συναντιούνται κυρίως σε κατανεμημένα δίκτυα.

1.2.3 Προκάτοχοι του SOAP

Διάφορες κατανεμημένες τεχνολογίες ανταλλαγής μηνυμάτων και απομακρυσμένης κλήσης διαδικασιών που αντικαταστήθηκαν από το SOAP αναφέρονται παρακάτω:

- **EDI:** Το Electronic Data Interchange αποτελεί το κυριότερο πρωτόκολλο ανταλλαγής μηνυμάτων στον χώρο των επιχειρήσεων. Κύριο χαρακτηριστικό του είναι το γεγονός ότι δε χρησιμοποιεί το διαδίκτυο αλλά ιδιωτικά δίκτυα. Έτσι καθίσταται μια ακριβή τεχνολογία από άποψη υποδομής. Στα μειονεκτήματα του EDI μπορούμε να προσθέσουμε το γεγονός ότι η μορφή των μηνυμάτων του είναι πολύ ειδική για κάθε βιομηχανία, άρα δεν είναι τόσο ευέλικτο όσο το SOAP. Έτσι το EDI χαρακτηρίζεται ως δύσκολη και ακριβή τεχνολογία.
- **CORBA:** Η Common Object Request Broker Architecture (CORBA) αποτελεί μία πρωτοποριακή προσπάθεια για την αξιόπιστη επικοινωνία αντικειμένων μεταξύ διαφορετικών συστημάτων. Το Object Management Group (OMG) ήθελε να δημιουργήσει ένα πρωτόκολλο ικανό να παρέχει

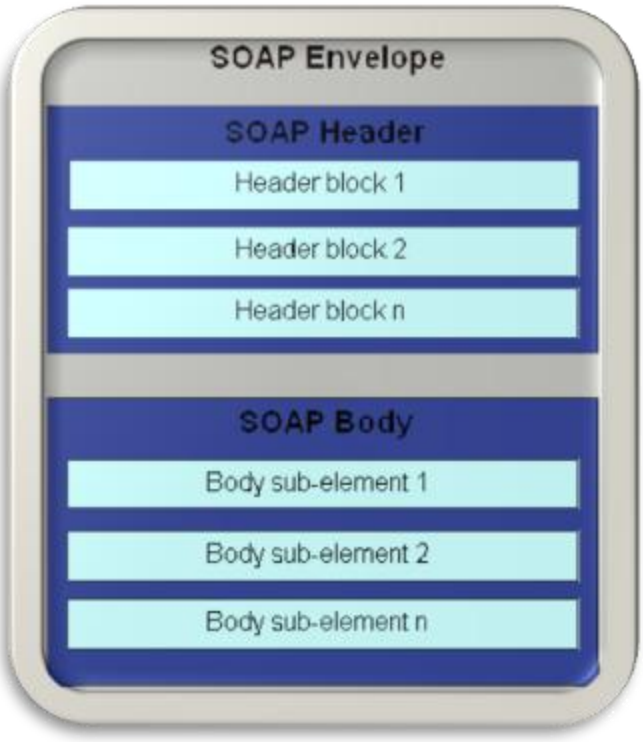
επικοινωνία μεταξύ αντικειμένων σε εντελώς διαφορετικά λειτουργικά συστήματα και γραμμένα σε διαφορετικές γλώσσες προγραμματισμού. ‘Ενα Object Request Broker (ORB) παρέχει τη αναγκαία διεπαφή ώστε ένα πρόγραμμα-πελάτης να μιλήσει με απομακρυσμένα αντικείμενα. Το CORBA θεωρείται μια δύσκολα κατανεμημένη τεχνολογία. Το OMG δημιούργησε μια πρότυπη γλώσσα την Interface Description Language (IDL), η οποία επιτρέπει τον καθορισμό της διεπαφής ενός αντικειμένου σε μία μορφή ανεξάρτητη από γλώσσα προγραμματισμού. Το OMG δημιούργησε επίσης και ένα πρωτόκολλο το Internet Inter-ORB Protocol (IIOP) για επικοινωνία μέσω μηνυμάτων πάνω από το Internet. Το OMG τώρα εργάζεται για τη δημιουργία ενός προτύπου διεπαφής μεταξύ SOAP και CORBA.

- **COM και DCOM:** Το Common Object Model (COM) είναι μια προδιαγραφή της Microsoft για ενοποίηση συστατικών (components integration) μέσα σε μια εφαρμογή. Η επικοινωνία είναι χαμηλού επιπέδου και επιτρέπει να αλληλεπιδρούν συστατικά γραμμένα σε διαφορετικές γλώσσες προγραμματισμού. Το Distributed COM (DCOM) είναι μια πιο πρόσφατη προσπάθεια που επιτρέπει συστατικά να αλληλεπιδρούν πάνω από ένα δίκτυο.
- **RMI:** Η πιο απλή μορφή επικοινωνίας στη Java μεταξύ αντικειμένων σε κατανεμημένα συστήματα παρέχεται από την επίκληση απομακρυσμένων μεθόδων - Remote Method Invocation (RMI). Το RMI επιτρέπει σε ένα αντικείμενο της Java που εκτελείται σε έναν υπολογιστή να καλέσει μια μέθοδο ενός άλλου αντικειμένου της Java που εκτελείται σε άλλο μηχάνημα. Αυτό που πρέπει να κάνει ο προγραμματιστής είναι να καθορίσει τη δημόσια διεπαφή που πρόκειται να εκτεθεί για ένα αντικείμενο. Με το βοηθητικό πρόγραμμα rmic εξετάζει τη διεπαφή και δημιουργεί κλάσεις γνωστές ως stub και skeleton. Στην πλευρά του διακομιστή η εφαρμογή δηλώνει τη διεπαφή στο rmiregistry και περιμένει νέες συνδέσεις. Μια εφαρμογή-πελάτης χρησιμοποιεί το rmiregistry, το οποίο «ζει» στο δίκτυο σε μία πρότυπη πόρτα, για να λάβει ένα στιγμιότυπο της stub κλάσης, η οποία υλοποιεί την επιθυμητή διεπαφή. Το RMI μπορεί όμως να χρησιμοποιηθεί μόνο μεταξύ εφαρμογών Java. Επίσης, και οι δύο πλευρές πρέπει να έχουν πρόσβαση στις κλάσεις της διεπαφής. Ακόμη ένα μειονέκτημα είναι ότι οι πόρτες (ports) που χρησιμοποιούνται συνήθως μπλοκάρονται από firewalls και proxy servers. Για αυτούς τους λόγους το RMI είναι περισσότερο επιτυχημένο στα τοπικά δίκτυα (intranets). Για τη πλατφόρμα της Java 2, η Sun δημιούργησε εργαλεία ικανά να γεφυρώσουν το κενό ανάμεσα σε αντικείμενα RMI και αντικείμενα CORBA γραμμένα σε άλλες γλώσσες προγραμματισμού. Αυτό το πακέτο ονομάζεται RMI-IIOP.

- **XML-RPC**: Το πρωτόκολλο XML-RPC είναι ο πρόδρομος του πρωτοκόλλου SOAP. Χρησιμοποιώντας το ευρέως υποστηριζόμενο πρωτόκολλο HTTP κάνει δυνατή την προσθήκη επεξεργασίας του XML-RPC σε κάθε εξυπηρετητή ιστού που υποστηρίζει CGI (Common Gateway Interface). Επίσης, κάνει το πέρασμα των firewalls δυνατό, καθιστώντας έτσι εύκολη την εγκατάσταση ενός εξυπηρετητή XML-RPC χωρίς συμβιβασμούς στην ασφάλεια. Το XML-RPC χαίρει απλότητας αλλά στερείται δυνατοτήτων σε σχέση με το SOAP. Η πρώτη έκδοση του SOAP (έκδοση 1.1) είχε σαν βάση της αυτό το πρωτόκολλο.

1.2.4 Το SOAP Envelope Framework

Το πιο σημαντικό μέρος του SOAP είναι το envelope framework. Παρόλο που αποτελείται από μερικά μόνο XML στοιχεία, παρέχει τη δομή και την επεκτασιμότητα που καθιστά το SOAP τόσο κατάλληλο για τη θεμελίωση όλων των XML-based κατανεμημένων λειτουργιών. Το SOAP envelope framework καθορίζει ένα μηχανισμό για αναγνώριση του είδους της πληροφορίας που βρίσκεται σε ένα μήνυμα, ποιός πρέπει να ασχοληθεί με τη συγκεκριμένη πληροφορία και τέλος, αν αυτό είναι υποχρεωτικό ή προαιρετικό. Έτσι, λοιπόν, ένα SOAP μήνυμα αποτελείται από τον υποχρεωτικό φάκελο (envelope) στον οποίο περιέχεται ένας αριθμός από προαιρετικές επικεφαλίδες (headers) και από ένα υποχρεωτικό σώμα (body). Αυτό φαίνεται στον παρακάτω σχήμα:



Σχήμα 2: Μορφή SOAP μηνύματος

Ο σκελετός ενός μηνύματος SOAP φαίνεται παρακάτω:

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    soap:encodingStyle=" http://www.w3.org/2001/12/soap-encoding">
 <soap:Header>
 ...
 ...
 </soap:Header>

 <soap:Body>
 ...
 ...
 </soap:Body>
</soap:Envelope>
```

Σχήμα 3: Σκελετός ενός SOAP μηνύματος

1.2.4.1 SOAP Envelope

Τα SOAP μηνύματα είναι κείμενα XML που καθορίζουν έναν κώδικα επικοινωνίας σε ένα κατανεμημένο περιβάλλον. Το στοιχείο ρίζα (root) είναι το στοιχείο Envelope, το οποίο καθορίζεται στο http://schemas.xmlsoap.org/soap/ envelope/namespace. Επειδή το στοιχείο envelope είναι μοναδικά προσδιορισμένο από τον namespace του, επιτρέπει σε διάφορα εργαλεία την άμεση αναγνώριση ενός XML κειμένου ως SOAP μήνυμα. Όπως έχει αναφερθεί και παραπάνω, το στοιχείο Envelope μπορεί να περιέχει προαιρετικά ένα header στοιχείο καθώς και υποχρεωτικά ένα στοιχείο Body, το οποίο με τη σειρά του μπορεί να έχει όσα παιδιά θέλει. Αυτό το χαρακτηριστικό επεκτασιμότητας βοηθάει στην κωδικοποίηση των δεδομένων στα SOAP μηνύματα.

1.2.4.2 SOAP Headers

Οι επικεφαλίδες είναι οι πρωταρχικοί επεκτάσιμοι μηχανισμοί στο SOAP. Παρέχουν το μέσο με το οποίο πρόσθετα χαρακτηριστικά μπορούν να προστεθούν στο SOAP. Οι επικεφαλίδες καθορίζουν έναν πολύ κομψό, αλλά συγχρόνως πολύ απλό μηχανισμό για να επεκταθούν τα SOAP μηνύματα με έναν αποκεντρωμένο τρόπο. Τυπικές λειτουργίες των headers είναι η πιστοποίηση, η έγκριση, η διαχείριση των συναλλαγών καθώς και άλλες. Δηλαδή, κάθε πληροφορία που παρέχεται στην επικεφαλίδα χρησιμοποιείται για την εκτέλεση ενός request. Για παράδειγμα, μία υπηρεσία για πληρωμή χρειάζεται μόνο τον αριθμό από το λογαριασμό από τον οποίο θα πάρουμε τα χρήματα και τον αριθμό στον οποίο θα κατατεθούν. Από την πλευρά των web services όμως, η αίτηση μιας υπηρεσίας παρέχει πολύ περισσότερες πληροφορίες, όπως τα ατομικά στοιχεία του αιτούντος. Αυτές οι πρόσθετες πληροφορίες χειρίζονται κυρίως από δομές των υπηρεσιών έξω από την κύρια υπηρεσία συναλλαγής. Η

ενσωμάτωση των πληροφοριών αυτών ως μέρος του body του SOAP μηνύματος θα περιπλέξει την υπηρεσία, γι' αυτό τις κωδικοποιούμε στην επικεφαλίδα.

Ένα SOAP μήνυμα μπορεί να περιέχει οσεσδήποτε επικεφαλίδες. Στην περίπτωση που όντως υπάρχουν headers, αυτές πρέπει να είναι παιδιά του στοιχείου SOAP Header, το οποίο όταν υπάρχει πρέπει να εμφανίζεται σαν πρώτο παιδί του στοιχείου SOAP Envelope. Το παρακάτω παράδειγμα απεικονίζει ένα μήνυμα SOAP με δύο headers, Transaction και Priority. Και οι δύο επικεφαλίδες είναι μοναδικά ορισμένες από το συνδυασμό του ονόματος του στοιχείου και του namespace URI.

1.2.4.3 SOAP Body

Το στοιχείο SOAP Body περιέχει τις πληροφορίες που αποτελούν τον πυρήνα του SOAP μηνύματος. Όλα τα απ' ευθείας παιδιά του στοιχείου Body χαρακτηρίζονται ως "body entries" ή απλά "bodies". Τα bodies μπορούν να περιέχουν αυθαίρετα XML ή συγκεκριμένη μορφή που καθορίζεται από συγκεκριμένες συμβάσεις.

1.2.4.4 SOAP Versioning

Μία σημαντική παρατήρηση σχετικά με το SOAP είναι ότι το στοιχείο Envelope δε φέρει έκδοση του πρωτοκόλλου (version) όπως άλλα πρωτόκολλα σαν το HTTP (HTTP/1.0 vs. HTTP/1.1). Οι σχεδιαστές του SOAP προφανώς έκαναν αυτήν την επιλογή καθώς η εμπειρία έχει αποδείξει ότι ο καθορισμός εκδόσεων που βασίζονται απλά σε αριθμούς μπορεί να είναι εύθραυστος. Επιπλέον, μεταξύ των πρωτοκόλλων δεν υπάρχουν κοινοί κανόνες για τον καθορισμό των αλλαγών των εκδόσεων. Έτσι λοιπόν το SOAP αυξάνει τις δυνατότητες των XML namespaces και καθορίζει την έκδοση του πρωτοκόλλου να είναι το URI του SOAP envelope namespace. Ως αποτέλεσμα, το μόνο συμπέρασμα που μπορούμε να βγάλουμε για τις εκδόσεις των SOAP είναι αν είναι ίδιες ή διαφορετικές.

Το πλεονέκτημα έγκειται στο γεγονός ότι παρέχεται στις μηχανές των Web services η επιλογή του πώς να χειριστούν τα SOAP μηνύματα που έχουν έκδοση διαφορετική από αυτή που είναι προορισμένη για τη συγκεκριμένη μηχανή. Επειδή μία μηχανή που υποστηρίζει την τελευταία έκδοση του SOAP θα ξέρει και τις προηγούμενες εκδόσεις του specification, έχει μια ποικιλία επιλογών βασισμένη στο namespace του εισερχόμενου μηνύματος. Αν η έκδοση του μηνύματος είναι η ίδια με την έκδοση που η μηχανή ξέρει να επεξεργάζεται, τότε προφανώς δεν υπάρχει πρόβλημα. Αν η έκδοση του μηνύματος είναι παλαιότερη, η μηχανή έχει δύο επιλογές, είτε να δημιουργήσει ένα version mismatch error ή να προσπαθήσει να συνεχίσει στέλνοντας πληροφορίες στον πελάτη που αφορούν τις εκδόσεις τις οποίες μπορεί να δεχτεί. Και τέλος, αν η έκδοση του μηνύματος είναι μεταγενέστερη, η μηχανή μπορεί να δοκιμάσει να επεξεργαστεί το μήνυμα (όχι προτιμώμενο) ή να δηλώσει version mismatch error σε συνδυασμό με πληροφορίες σχετικά με τις εκδόσεις τις οποίες καταλαβαίνει.

1.2.5 SOAP Messaging

Ως μοντέλο στα κατανεμημένα συστήματα ο όρος messaging αναφέρεται σε ένα μηχανισμό που καθιστά δυνατή την επικοινωνία μεταξύ συστημάτων με το πέρασμα μηνυμάτων. Ως μήνυμα ορίζεται μια μονάδα επικοινωνίας που ενσωματώνει πληροφορία. Τα διάφορα μοντέλα μηνυμάτων μπορούν να ποικίλουν ανάλογα με τα παρακάτω κριτήρια:

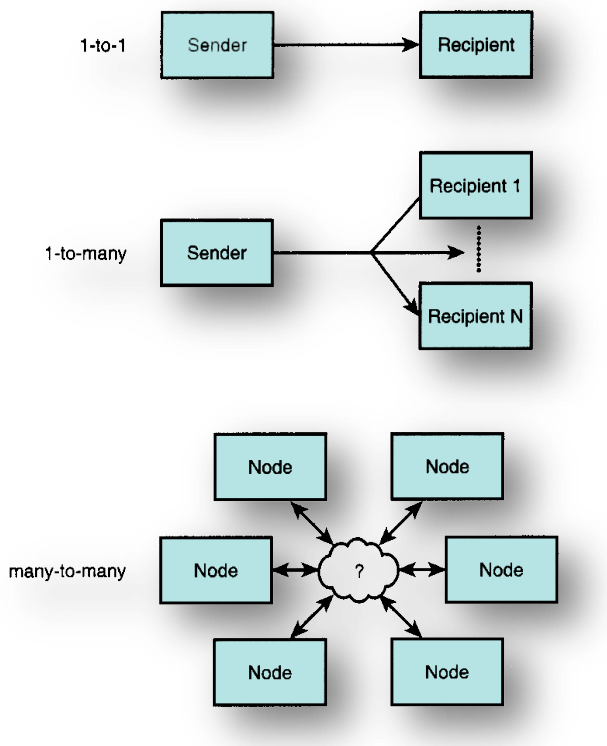
- Αριθμός συμμετεχόντων
- Τρόπος αλληλεπίδρασης
- Συγχρονισμός της ανταλλαγής μηνυμάτων
- Απευθείας ή με χρήση ουράς ανταλλαγή μηνυμάτων
- Ποιότητα υπηρεσίας (Quality of Service)
- Μορφή μηνυμάτων

1.2.5.1 Αριθμός συμμετεχόντων

Οι διάφοροι τρόποι στους οποίους μπορούμε να οργανώσουμε τις ανταλλαγές μηνυμάτων σύμφωνα με τον αριθμό των συμμετεχόντων είναι οι εξής:

- Η πρώτη περίπτωση είναι η 1-to-1 (point-to-point). Περιλαμβάνει μόνο δύο συστήματα και ένα απλό παράδειγμα είναι ένα σενάριο μιας ηλεκτρονικής παραγγελίας. Το μόνο που χρειάζεται να γνωρίζουμε είναι πού να στείλουμε το μήνυμα.
- Η δεύτερη περίπτωση είναι η 1-to-many, όπου ένα μήνυμα κατευθύνεται σε περισσότερους αποδέκτες. Συχνά συναντάτε με τον όρο publish/subscribe ή topic-based messaging.
- Τέλος, η τρίτη περίπτωση είναι η many-to-many που περιλαμβάνει τρόπους ανταλλαγής μηνυμάτων ανάμεσα σε πολλούς συμμετέχοντες.

Και οι οποίοι φαίνονται στο παρακάτω σχήμα:



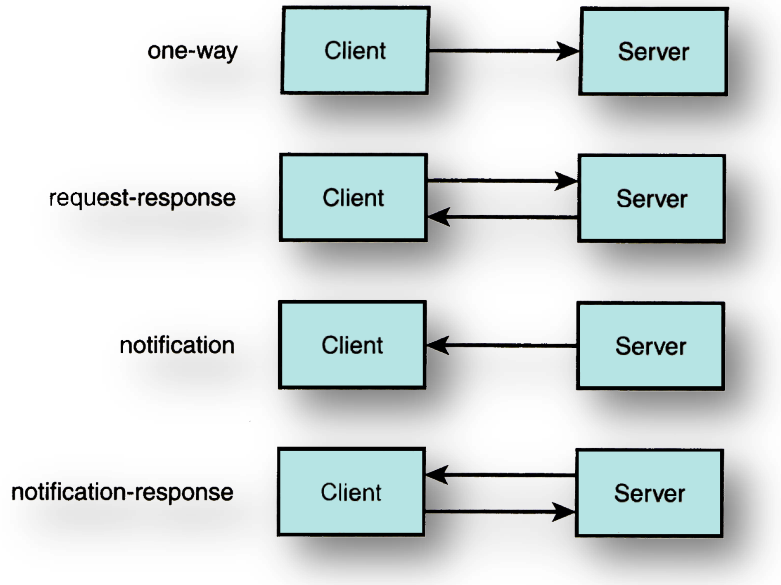
Σχήμα 4: Αριθμός συμμετεχόντων

1.2.5.2 Τρόποι αλληλεπίδρασης

Υπάρχουν τέσσερις τρόποι με τους οποίους μπορεί να γίνει η ανταλλαγή μηνυμάτων:

- **One-way (fire-and-forget).** Αποτελεί τον πιο απλό τρόπο. Έχουμε αποστολή ενός μηνύματος από ένα σύστημα σε ένα άλλο.
- **Request – Response.** Για κάθε αίτηση έχουμε και μία απάντηση.
- **Notification.** Αποτελεί τον καθρέπτη του one-way. Ο server στέλνει μήνυμα και όχι ο client.
- **Notification – Response.** Αποτελεί τον καθρέφτη του request – response.

Σχηματικά οι τρόποι αυτοί παρουσιάζονται στο παρακάτω σχήμα:



Σχήμα 5: Τρόποι αλληλεπίδρασης

1.2.5.3 Συγχρονισμός της ανταλλαγής μηνυμάτων

Τα μηνύματα μπορούν να είναι είτε σύγχρονα είτε ασύγχρονα. Στην σύγχρονη ανταλλαγή μηνυμάτων η διαδικασίας της αίτησης δεν τελειώνει αν δεν τελειώσει πρώτα η διαδικασία της επεξεργασίας. Ενώ στην ασύγχρονη μία απάντηση από την αίτηση θα σταλεί αμέσως (ή πολύ γρήγορα), πριν ολοκληρωθεί η επεξεργασία. Η απάντηση θα φτάσει μέσω callback.

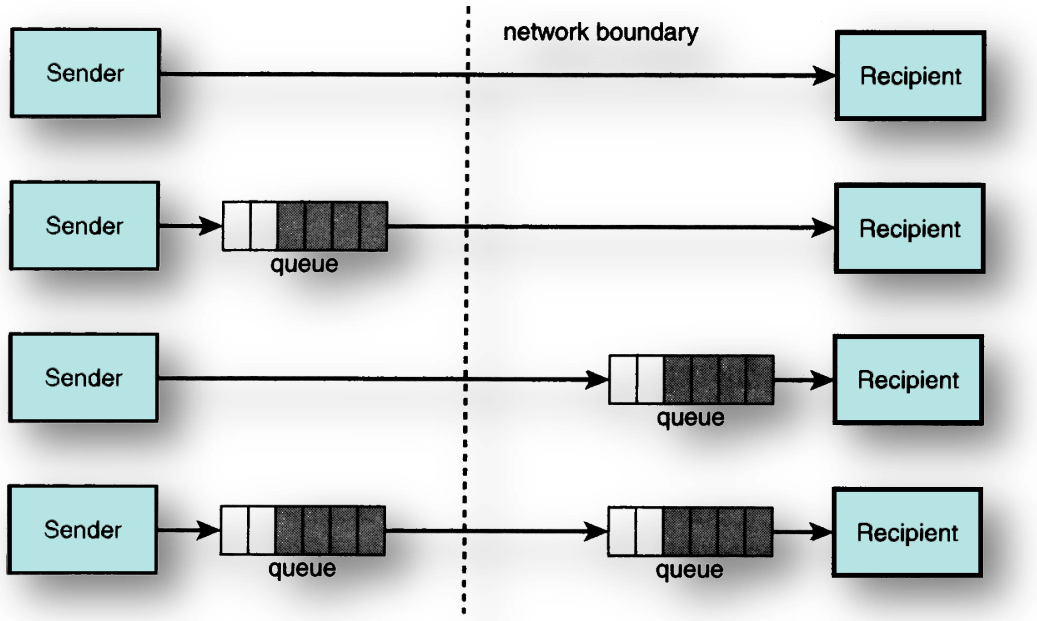
1.2.5.4 Απευθείας ή με χρήση ουράς ανταλλαγή μηνυμάτων

Απευθείας ανταλλαγή μηνυμάτων έχουμε όταν δεν μεσολαβεί middleware. Έτσι, πρέπει να υπάρχει απευθείας σύνδεση της πηγής και του στόχου. Για αυτόν το λόγο ονομάζεται συχνά και connection-oriented messaging. Μπορούμε να πετύχουμε ένα βαθμό ασύγχρονης ανταλλαγής χρησιμοποιώντας threads.

Η ανταλλαγή μηνυμάτων που δεν γίνεται απευθείας, συμπεριλαμβάνει σε ένα βαθμό την τοποθέτηση των μηνυμάτων σε ουρά. Οι ουρές προσφέρουν ένα είδος buffering. Ένα τυπικό παράδειγμα είναι ένας email server. Όταν στέλνουμε ένα email ο mail client μας δεν επικοινωνεί απευθείας με τον mail client του παραλήπτη, αλλά κατευθύνεται στον mail server. Ο τελευταίος αποθηκεύει το μήνυμα και περιμένει την κατάλληλη στιγμή για να το στείλει. Αυτήν είναι η λειτουργία του buffering, η οποία

είναι πολύ ισχυρή γιατί λειτουργεί και στις περιπτώσεις που οι servers είναι εκτός λειτουργίας για μεγάλο χρονικό διάστημα.

Το παρακάτω σχήμα παρουσιάζει αρχικά την απευθείας επικοινωνία. Το δεύτερο και τρίτο τμήμα είναι σύστημα με ουρά που λειτουργεί ως buffer. Ενώ το τελευταίο τμήμα παρουσιάζει το μεγαλύτερο ενδιαφέρον. Εδώ το μήνυμα μπορεί να μεταφερθεί από το τοπικό στο απομακρυσμένο σύστημα χωρίς κάποιος από τον αποστολέα ή τον παραλήπτη να είναι online.



Σχήμα 6: Ανταλλαγή μηνυμάτων

1.2.5.5 Ποιότητα υπηρεσίας (Quality of Service)

Η απευθείας ανταλλαγή μηνυμάτων παραθέτει παραμέτρους σχετικά με την ποιότητα της υπηρεσίας όπως security και transaction management. Όταν όμως έχουμε ανταλλαγή με ουρά, έχουμε επίσης και διαφορετικούς παραμέτρους του QoS. Για παράδειγμα, τα μηνύματα μπορούν να αποθηκευτούν σε μια ουρά με διάφορους τρόπους, όπως στην μνήμη ή σε κάποιο συνεχή χώρο αποθήκευσης, όπως μία βάση δεδομένων. Ο πρώτος για παράδειγμα, εγγυάται ταχύτητα αλλά όχι ασφάλεια. Ένα ακόμα χαρακτηριστικό που καθορίζει το QoS είναι ένα μήνυμα επιβεβαίωσης της παραλαβής ενός μηνύματος.

1.2.5.6 Μορφή μηνυμάτων

Η μορφή των δεδομένων σε ένα μήνυμα είναι από τα πιο σπουδαία χαρακτηριστικά. Τα περισσότερα συστήματα μηνυμάτων επιτρέπουν τη μεταφορά κειμένου και δυαδικών δεδομένων, γεγονός που επιτρέπει την μεταφορά XML. Άλλα συστήματα επιτρέπουν μόνο XML που αντιστοιχούν σε συγκεκριμένα σχήματα. Μερικά platform-focused συστήματα, όπως το Java Messaging Service (JMS) middleware και το Microsoft's .NET messaging server επιτρέπουν επίσης το αυτόματο serialization των δεδομένων της εφαρμογής (Java Objects στην περίπτωση του JMS και Common Language Runtime δομές δεδομένων στην περίπτωση της Microsoft).

Επίσης οι κανόνες σύνταξης για τα μηνύματα SOAP είναι οι ακόλουθοι:

- Πρέπει να είναι κωδικοποιημένα χρησιμοποιώντας XML.
- Πρέπει να χρησιμοποιούν το SOAP Envelope namespace.
- Πρέπει να χρησιμοποιούν το SOAP Encoding namespace.
- Πρέπει να περιέχουν αναφορά σε DTD.
- Πρέπει να περιέχουν XML Processing Instructions.

1.2.6 SOAP – XML attributes

Οι παρακάτω ιδιότητες XML έχουν ειδικό νόημα για την επεξεργασία ενός μηνύματος SOAP:

- ***encodingStyle***: Αποτελείται από ένα σύνολο κανόνων που καθορίζουν τον τρόπο που θα κωδικοποιηθούν οι διάφοροι τύποι δεδομένων καθώς και οι εφαρμογές, χρησιμοποιώντας XML. Επίσης το στυλ κωδικοποίησης αναφέρεται στον τρόπο με τον οποίο οι πληροφορίες μοιράζονται στις διάφορες πλατφόρμες ακόμα και όταν αυτές δεν έχουν κοινούς τύπους δεδομένων.

- ***role***: Χρησιμοποιείται για να υποδείξει τον SOAP node για τον οποίο προορίζεται το SOAP header block.

- ***mustUnderstand***: Χρησιμοποιείται για να επιβεβαιώσουμε ότι ο παραλήπτης ενός μηνύματος έχει καταλάβει πώς να το επεξεργαστεί και ότι η επεξεργασία ενός SOAP header block είναι υποχρεωτική ή προαιρετική.

- ***relay:*** Χρησιμοποιείται για να υποδείξει αν το SOAP header block που προορίζεται για ένα κόμβο επεξεργασίας πρέπει να αναμεταδοθεί αν δεν υποστεί επεξεργασία.

1.2.7 Λάθη SOAP

Κατά την διάρκεια επεξεργασίας των μηνυμάτων μπορεί να προκύψουν διάφορα λάθη. Πληροφορίες για τα λάθη αυτά μεταφέρονται από ειδικά μηνύματα όπως φαίνεται παρακάτω:

```
<?xml version="1.0"?>
    <soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
soap:encodingStyle=" http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
...
...
</soap:Header>
<soap:Body>
    <soap:Fault>
        <faultcode> . . . .</faultcode>
        <faultstring> . . . . </faultstring>
        <faultactor> . . . . </faultactor>
        <details> . . . . <details>
    </soap:Fault>
</soap:Body>
    </soap:Envelope>
```

Σχήμα 7: Λάθη SOAP μηνυμάτων

faultcode

Ο κωδικός αυτός αποτελεί ένα αλγοριθμικά παραγόμενο αναγνωριστικό για το είδος του λάθους, το οποίο πρέπει να είναι XML Qualified Name.

faultstring

Το αλφαριθμητικό λάθους αποτελεί μια περιγραφή του λάθους που μπορεί να γίνει εύκολα αντιληπτή από τον άνθρωπο.

faultactor

Ο δείκτης εντοπισμού λάθους υποδεικνύει σε ποιο στοιχείο εντοπίστηκε το λάθος .

details

Εδώ αναφέρονται ειδικές λεπτομέρειες για το λάθος που προέκυψε. Παρουσιάζονται όταν το λάθος βρίσκεται στο σώμα του μηνύματος και δεν πρέπει να χρησιμοποιούνται για λάθη που δεν προέρχονται από το σώμα.

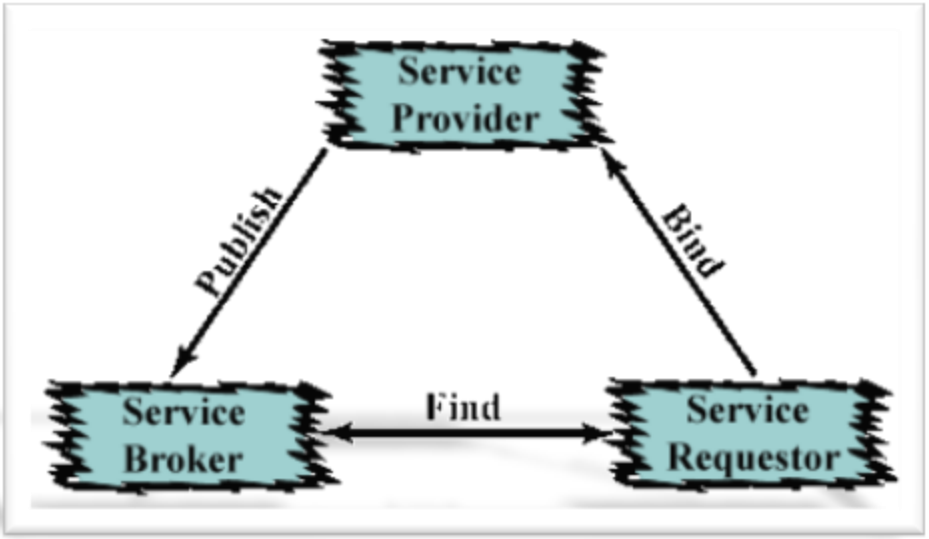
1.3 Service Oriented Architecture – SOA

1.3.1 SOA – Αρχιτεκτονική των Web Services

Η Service Oriented Αρχιτεκτονική είναι ένα πλαίσιο πληροφορικής που αναλύει τις επιχειρησιακές εφαρμογές στις διάφορες Web services που τις αποτελούν, δηλαδή σε μεμονωμένες επιχειρησιακές λειτουργίες με σκοπό την επαναχρησιμοποίηση των διάφορων υπηρεσιών για την υλοποίηση περεταίρω επιχειρησιακών εφαρμογών. Έτσι η service oriented αρχιτεκτονική έχει ως δομικό στοιχείο τις Web services.

Η Service Oriented Αρχιτεκτονική καθιερώνεται όλο και περισσότερο καθώς οι επιχειρησιακές εφαρμογές των διαφόρων οργανισμών γίνονται όλο και πιο πολύπλοκες και εκτείνονται και σε εφαρμογές εκτός αυτών. Καθώς η ανταγωνιστικότητα οδηγεί τις εταιρίες σε συνεχή βελτίωση των εφαρμογών τους με σκοπό την καλύτερη και άμεση εξυπηρέτηση των πελατών τους (Quality of Service) και την μείωση του Time to Market.

Το μοντέλο της service oriented αρχιτεκτονικής φαίνεται παρακάτω:



Σχήμα 8: SOA αρχιτεκτονική

Όπως φαίνεται από το παραπάνω σχήμα στην service oriented αρχιτεκτονική μπορούμε να διακρίνουμε τρεις ρόλους: τον παροχέα μιας υπηρεσίας, το χρήστη, καθώς και τον κατάλογο στον οποίο δημοσιεύονται οι υπηρεσίες.

Ο παροχέας είναι υπεύθυνος για την δημιουργία μίας περιγραφής της υπηρεσίας, για την δημοσίευση της περιγραφής αυτής σε κάποιον κατάλογο, καθώς και για την παραλαβή μηνυμάτων που προέρχονται από κάποια αίτηση για χρήση κάποιας Web service. Ένας παροχέας μπορεί να είναι κάποια εταιρεία που φιλοξενεί μια Web service. Είναι δηλαδή κάτι ανάλογο του "server side" σε μία σχέση client-server.

Ο χρήστης μιας υπηρεσίας είναι υπεύθυνος για τον εντοπισμό της περιγραφής μιας δημοσιευμένης υπηρεσίας σε έναν ή περισσότερους καταλόγους και είναι υπεύθυνος για τη χρησιμοποίηση της περιγραφής της υπηρεσίας για να κάνει bind σε μία υπηρεσία ή να την καλέσει. Κάθε καταναλωτής της υπηρεσίας μπορεί να θεωρηθεί ως αιτών της υπηρεσίας αυτής. Στο μοντέλο της σχέσης client-server ο αιτών θα μπορούσε να χαρακτηριστεί ως "client side".

Ο κατάλογος δημοσίευσης των υπηρεσιών είναι υπεύθυνος για τη διαφήμιση των περιγραφών των Web services που είναι δημοσιευμένες σε αυτόν από τον παροχέα και να επιτρέπει στους χρήστες να ψάχνουν τη συλλογή των περιγραφών των υπηρεσιών που περιέχονται σε αυτόν τον κατάλογο. Ο ρόλος του καταλόγου των υπηρεσιών είναι απλός, εκτελεί το ταίριασμα μεταξύ του αιτούντος της υπηρεσίας και του παροχέα αυτής. Μόλις ολοκληρωθεί το ταίριασμα ο κατάλογος δε χρειάζεται πλέον και όλες οι υπόλοιπες λειτουργίες θα γίνουν μεταξύ του χρήστη και του παροχέα.

Καθένας από τους παραπάνω ρόλους μπορεί να λειτουργεί σε οποιοδήποτε πρόγραμμα ή σε οποιοδήποτε κόμβο δικτύου. Σε μερικές περιπτώσεις κάποιο πρόγραμμα μπορεί να εκπληρώσει πολλαπλούς ρόλους. Για παράδειγμα, ένα πρόγραμμα μπορεί να είναι συγχρόνως ο παροχέας μιας υπηρεσίας που εξυπηρετεί έναν πελάτη και ο χρήστης μιας άλλης υπηρεσίας που παρέχεται από κάποιο άλλο πρόγραμμα.

Πέρα από τους ρόλους στην service oriented αρχιτεκτονική έχουμε και τρεις λειτουργίες: τη δημοσίευση, την εύρεση και τη σύνδεση.

Η λειτουργία της δημοσίευσης είναι μία πράξη καταχώρησης ή διαφήμισης μιας υπηρεσίας. Λειτουργεί ως σύμβαση μεταξύ του καταλόγου υπηρεσιών και του παροχέα. Όταν ο παροχέας δημοσιεύει μια περιγραφή μιας Web service σε κάποιον κατάλογο, στην ουσία διαφημίζει τις λεπτομέρειες της υπηρεσίας αυτής σε μια κοινότητα χρηστών. Οι ακριβείς πληροφορίες για το δημοσιευμένο API εξαρτώνται από την υλοποίηση του καταλόγου. Σε μερικά σενάρια απλής ή άμεσης δημοσίευσης το δίκτυο παίζει το ρόλο του καταλόγου. Με τη δημοσίευση να είναι απλά μία πράξη μετακίνησης της περιγραφής της υπηρεσίας σε μία δομή ενός καταλόγου ενός server σε μία Web application. Από την

άλλη, άλλες υλοποιήσεις καταχωρήσεων υπηρεσιών, όπως ο UDDI καθορίζουν μια πολύπλοκη διαδικασία δημοσίευσης.

Η λειτουργία της εύρεσης είναι η αντίστροφη διαδικασία της δημοσίευσης. Αποτελεί τη σύμβαση μεταξύ του χρήστη και του καταλόγου. Με τη λειτουργία αυτή ο χρήστης καθορίζει τα κριτήρια της αναζήτησής του, όπως τον τύπο και άλλες πλευρές της υπηρεσίας, την ποιότητα των εγγυήσεων κτλ. Ο κατάλογος κάνει το ταίριασμα μεταξύ των κριτηρίων αναζήτησης του χρήστη και των δημοσιευμένων σ' αυτόν περιγραφών υπηρεσιών. Το αποτέλεσμα της λειτουργίας της εύρεσης είναι μια λίστα από περιγραφές υπηρεσιών που ανταποκρίνονται στα κριτήρια. Φυσικά, η διαδικασία εύρεσης ποικίλλει ανάλογα με τον τρόπο υλοποίησης του κάθε καταλόγου υπηρεσιών. Απλοί κατάλογοι μπορεί να παρέχουν μια λειτουργία αναζήτησης με τίποτα πιο σύνθετο από μία αίτηση HTTP GET χωρίς παραμέτρους. Γεγονός που σημαίνει ότι η λειτουργία εύρεσης επιστρέφει όλες τις υπηρεσίες που είναι δημοσιευμένες και έγκειται στο χρήστη να διαλέξει την υπηρεσία που τον εξυπηρετεί.

Τέλος, η λειτουργία της σύνδεσης ενσαρκώνει τη σχέση του μοντέλου client server μεταξύ του χρήστη και του παροχέα. Μπορεί να είναι πολύπλοκη και δυναμική, όπως για παράδειγμα η δημιουργία ενός proxy client ή απλή και στατική.

1.4 Web Service Description Language – WSDL

1.4.1 Ορισμός και σκοπός της WSDL

Σύμφωνα με τον ορισμό του W3C:

"Η Web Services Description Language είναι ένα σχήμα XML για την περιγραφή δικτυακών υπηρεσιών σαν ένα σύνολο από τελικά σημεία που λειτουργούν σε μηνύματα τα οποία περιέχουν πληροφορία είτε προσανατολισμένη στα έγγραφα είτε προσανατολισμένη στις διαδικασίες".

Η WSDL, δηλαδή, μας βοηθάει να περιγράψουμε ένα σύνολο από μηνύματα και το πώς αυτά τα μηνύματα ανταλλάσσονται. Είναι μια προδιαγραφή που παρέχει πληροφορίες για όλες τις δημόσια διαθέσιμες πληροφορίες, για τους τύπους δεδομένων, για όλα τα μηνύματα αίτησης και απάντησης, για την σύνδεση και τα πρωτόκολλα μεταφοράς και τέλος για τις διευθύνσεις που χρειάζονται για τον εντοπισμό μιας υπηρεσίας. Οι λειτουργίες και τα μηνύματα περιγράφονται περιληπτικά, και τότε δένονται σε ένα συγκεκριμένο πρωτόκολλο δικτύων και μορφή μηνυμάτων για να καθορίσουν ένα τελικό σημείο, το οποίο με την σειρά του συνδυάζεται με άλλα τελικά σημεία και δημιουργούν τις υπηρεσίες. Η WSDL είναι επεκτάσιμη, επιτρέποντας την περιγραφή τελικών σημείων και των μηνυμάτων τους άσχετα από τη μορφή των μηνυμάτων και των πρωτοκόλλων δικτύων που χρησιμοποιούνται για την επικοινωνία.

Η σήμανση που χρησιμοποιείται σε ένα αρχείο WSDL για να περιγράψει μορφές μηνυμάτων βασίζεται στο πρότυπο του XML Schema το οποίο σημαίνει ότι είναι ταυτόχρονα ανεξάρτητη από γλώσσα προγραμματισμού και βασισμένη σε πρότυπα. Αυτό το γεγονός την κάνει κατάλληλη για να περιγράψει διεπαφές web services οι οποίες είναι προσβάσιμες από μία μεγάλη ποικιλία πλατφορμών και γλωσσών προγραμματισμού. Έτσι η WSDL παρέχει έναν τρόπο στους παροχείς υπηρεσιών να περιγράψουν τη βασική μορφή των αιτήσεων και απαντήσεων των υπηρεσιών πάνω από διαφορετικά πρωτόκολλα και κωδικοποιήσεις, καθώς επίσης χρησιμοποιείται για να περιγράψει τι μπορεί να κάνει ένα web service, πού βρίσκεται και πώς να το καλέσει κανείς.

1.4.2 Τεχνική περιγραφή της WSDL

Η WSDL χρησιμοποιεί τα παρακάτω στοιχεία για τον ορισμό δικτυακών υπηρεσιών :

Ορίζει υπηρεσίες σαν συλλογές από τελικά σημεία δικτύου ή αλλιώς **ports**. Στην WSDL ο περιγραφικός ορισμός των τελικών σημείων και των μηνυμάτων διαχωρίζεται από συγκεκριμένα δικτυακά πρωτόκολλα ή μορφές δεδομένων. Αυτό επιτρέπει την επαναχρησιμοποίηση των περιγραφικών ορισμών : των μηνυμάτων (**messages**).

Τα μηνύματα είναι αόριστες περιγραφές των δεδομένων που ανταλλάσσονται και των τύπων τελικών σημείων (**port types**).

Οι τύποι τελικών σημείων είναι συλλογές λειτουργιών.

Το συγκεκριμένο πρωτόκολλο και ο ορισμός της μορφής των δεδομένων για ένα συγκεκριμένο τύπο τελικών σημείων δημιουργεί μία επαναχρησιμοποιούμενη σύνδεση (**binding**).

Ένα τελικό σημείο (**port**) ορίζεται συνδέοντας μια διεύθυνση δικτύου με μία επαναχρησιμοποιούμενη σύνδεση (binding) και μία συλλογή τελικών σημείων ορίζουν μία υπηρεσία (**service**).

Ως εκ τούτου, ένα έγγραφο WSDL χρησιμοποιεί τα παρακάτω στοιχεία για τον ορισμό δικτυακών υπηρεσιών:

• **Types.** Ένα περίβλημα για ορισμούς τύπων δεδομένων χρησιμοποιώντας ένα σύστημα τύπων (όπως το XML Schema). Όταν η υπηρεσία χρησιμοποιεί μόνο απλούς τύπους (αλφαριθμητικά και ακεραίους) η χρήση του στοιχείου types δεν είναι απαραίτητη.

• **Message.** Ένας περιγραφικός ορισμός των δεδομένων που ανταλλάσσονται. Μήνυμα μιας κατεύθυνσης, είτε απάντησης, είτε αίτησης. Καθορίζει το όνομα του μηνύματος και περιέχει στοιχεία “part”, τα οποία αναφέρονται στις παραμέτρους των μηνυμάτων.

• **Operation.** Περιγραφή μίας λειτουργίας που υποστηρίζεται από μία υπηρεσία

• **Port Type.** Ένα περιγραφικό σύνολο από λειτουργίες που υποστηρίζονται από ένα ή περισσότερα τελικά σημεία. Δηλαδή ένας συνδυασμός από messages για την

υλοποίηση μιας λειτουργίας η οποία μπορεί να ποικίλει από μια απλή λειτουργία αίτησης –απάντησης έως μια αρκετά σύνθετη.

• **Binding.** Ένα συγκεκριμένο πρωτόκολλο και μια μορφή δεδομένων για ένα συγκεκριμένο τύπο τελικών σημείων που καθορίζει τον τρόπο υλοποίησης της υπηρεσίας πάνω στο καλώδιο.

• **Port.** Ένα μοναδικό τελικό σημείο που ορίζεται σαν συνδυασμός μίας σύνδεσης (binding) και μιας διεύθυνσης δικτύου.

• **Service.** Μία συλλογή από σχετικά τελικά σημεία.

1.4.3 Παράδειγμα WSDL

Το παρακάτω παράδειγμα είναι ένα έγγραφο WSDL που περιγράφει μια υπηρεσία για τις τιμές μετοχών:

```
<?xml version="1.0"?>
<definitions name="StockQuote"
        targetNamespace="http://example.com/stockquote.wsdl"
        xmlns:tns="http://example.com/stockquote.wsdl"
        xmlns:xsd1="http://example.com/stockquote.xsd"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns="http://schemas.xmlsoap.org/wsdl/">
        <types>
        <schema targetNamespace="http://example.com/stockquote.xsd"
        xmlns="http://www.w3.org/2000/10/XMLSchema">
                <element name="TradePriceRequest">
                        <complexType>
                                <all>
                                   <element name="tickerSymbol" type="string"/>
                                </all>
                        </complexType>
                </element>
                <element name="TradePrice">
                        <complexType>
                                <all>
                                <element name="price" type="float"/>
                                </all>
                        </complexType>
                </element>
        </schema>
        </types>
  <message name="GetLastTradePriceInput">
                <part name="body" element="xsd1:TradePriceRequest"/>
        </message>
```

```
        <message name="GetLastTradePriceOutput">
                <part name="body" element="xsd1:TradePrice"/>
        </message>
        <portType name="StockQuotePortType">
                <operation name="GetLastTradePrice">
                        <input message="tns:GetLastTradePriceInput"/>
                        <output message="tns:GetLastTradePriceOutput"/>
                </operation>
        </portType>
        <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">

        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
                <operation name="GetLastTradePrice">
                <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
                <input>
                        <soap:body use="literal"/>
                </input>
                <output>
                        <soap:body use="literal"/>
                </output>
                </operation>
        </binding>
        <service name="StockQuoteService">
                <documentation>My first service</documentation>
                <port name="StockQuotePort" binding="tns:StockQuoteBinding">
                        <soap:address location="http://example.com/stockquote"/>
                </port>
        </service>
</definitions>
```

Ακολουθεί μία αίτηση σε SOAP και η απάντηση της υπηρεσίας:

Αίτηση SOAP μέσω HTTP

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
     <SOAP-ENV:Body>
          <m:GetLastTradePrice
                    xmlns:m="Some-URI">
                    <symbol>MOT</symbol>
          </m:GetLastTradePrice>
      </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

Απάντηση της υπηρεσίας με SOAP μέσω HTTP

```
HTTP/1.1 200 OK Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    <SOAP-ENV:Body>
      <m:GetLastTradePriceResponse
         xmlns:m="Some-URI">
         <Price>14.5</Price>
     </m:GetLastTradePriceResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

1.5 Web services

1.5.1 Τι είναι τα Web services

Με τον όρο web services αναφερόμαστε σε μία αρχιτεκτονική που προσφέρει τη δυνατότητα δημιουργίας και χρήσης ηλεκτρονικών υπηρεσιών στο διαδίκτυο. Το Internet έχει μετατραπεί από ένα μέσο προβολής και δημοσιοποίησης σε μέσο παροχής υπηρεσιών επιχειρήσεων προς τους πελάτες τους, αλλά και προς άλλες επιχειρήσεις. Έτσι τέθηκε ένα θέμα για την ευχρηστία, τη λειτουργικότητα καθώς και την απλότητα των παρεχόμενων αυτών υπηρεσιών. Οι προσπάθειες για λύση αυτών των προβλημάτων οδήγησαν στη θέσπιση από το World Wide Web Consortium (*www.w3c.org*) της αρχιτεκτονικής των Web services.

Ο επίσημος ορισμός που δίνεται από το World Wide Web Consortium, είναι ότι οι Web services είναι μία εφαρμογή λογισμικού αναγνωρίσιμη από ένα URI, η οποία χρησιμοποιεί την γλώσσα XML για να αναγνωρίζει, περιγράφει και ανακαλύπτει το interface και τα bindings και συγχρόνως υποστηρίζει απ' ευθείας αλληλεπιδράσεις με άλλες λογισμικές εφαρμογές μέσω πρωτοκόλλων βασισμένων στο Internet με χρήση XML μηνυμάτων.

Σύμφωνα με τον ορισμό που δίνει η IBM, Web services είναι μια νέα γενιά εφαρμογών web. Είναι αυτό-περιγραφικές, ανεξάρτητες, αρθρωτές εφαρμογές που μπορούν να δημοσιευτούν, να εντοπιστούν και να κληθούν από το web. Web services εκτελούν συναρτήσεις, που μπορεί να είναι ο,τιδήποτε, από ένα απλό αίτημα έως μια περίπλοκη επιχειρησιακή διαδικασία. Μόλις μια υπηρεσία διαδικτύου αναπτυχθεί, άλλες εφαρμογές μπορούν να την αναζητήσουν και να την καλέσουν.

Τα επιχειρησιακά συστήματα απαιτούν όλο και περισσότερο τη διαλειτουργικότητα (interoperability) όσον αφορά τις πλατφόρμες και μια σειρά από εύκαμπτες υπηρεσίες που να μπορούν εύκολα να εξελιχθούν με την πάροδο του χρόνου. Η XML καθιερώνεται για την αντιπροσώπευση και τη διαβίβαση δομημένων δεδομένων που είναι ανεξάρτητα από γλώσσα προγραμματισμού, πλατφόρμα λογισμικού, και από το hardware. Οι υπηρεσίες Ιστού αναφέρονται στις μεθόδους για τα προγράμματα μέσω των τυποποιημένων τεχνολογιών XML με τις ανοικτές, τυποποιημένες διεπαφές που καλύπτουν τις διεπαφές προγράμματος εφαρμογών (APIs). Οι υπηρεσίες Ιστού είναι βασισμένες σε τέσσερα πρότυπα Διαδικτύου:

- Σχήμα XML
- Simple Object Access Protocol (Soap)
- Web Services Definition Language (WSDL)
- Universal Description, Discovery and Integration (UDDI).

Με αυτόν τον τρόπο σήμερα οι υπηρεσίες που χρησιμοποιούνται βασίζονται σε μια ενιαία αρχιτεκτονική ανάπτυξης και δημοσίευσης και είναι ανεξάρτητες από λειτουργικό σύστημα ή γλώσσα προγραμματισμού. Δύο βασικά χαρακτηριστικά που μπορούν να χαρακτηρίζουν μία Web service είναι ότι είναι ανακαλύψιμη και αυτό-περιγραφόμενη. Η δημοσίευση κάθε υπηρεσίας ιστού συνοδεύεται και από τη δημοσίευση ενός interface προς αυτήν, που τη γνωστοποιεί και ταυτόχρονα τη χαρακτηρίζει και την περιγράφει.

1.5.2 Πλεονεκτήματα, μειονεκτήματα και οφέλη

Πλεονεκτήματα μπορούμε να εντοπίσουμε και για τεχνικούς λόγους. Οι Web services είναι service-oriented παρέχοντας τη δυνατότητα επικοινωνίας από εφαρμογή σε εφαρμογή μέσα στο διαδίκτυο και εύκολή πρόσβαση σε ετερογενείς εφαρμογές και συσκευές. Οι λόγοι που μας οδηγούν στη χρησιμοποίηση Web services μπορούν να συνοψιστούν στους εξής:

- *Διαλειτουργικότητα*: Ένα web service είναι ανεξάρτητο τόσο από το λειτουργικό σύστημα όσο και από το hardware.
- *Εύκολη ενσωμάτωση:* Η δημιουργία ενός web service δεν απαιτεί αλλαγές στο μηχανισμό ενός συστήματος που ήδη λειτουργεί μέσα στο Internet.
- *Διαθεσιμότητα και δημοσίευση:* Πληροφορίες για τα διαθέσιμα Web services δημοσιεύονται σε ειδικούς καταλόγους οπότε η εύρεση και η χρήση τους καθίσταται εύκολη και γρήγορη.
- *Δυνατότητα επέκτασης:* Η δυνατότητα επέκτασης και ανανέωσης ενός Web service είναι δυνατή και εύκολη, προσφέροντας έτσι αναβαθμισμένες υπηρεσίες στους χρήστες του.
- *Δυνατότητα χρήσης λογισμικών συστημάτων:* Η δυνατότητα χρήσης έτοιμων υπηρεσιών καθιστά τα λογισμικά συστήματα και ειδικότερα τα websites που τις χρησιμοποιούν πιο λειτουργικά και φιλικά καθώς παρέχουν περισσότερες υπηρεσίες στους χρήστες.
- *Μικρό κόστος δημιουργίας και χρήσης:* Η δημιουργία του web service κοστίζει ελάχιστα ειδικά εάν σε ένα λογισμικό σύστημα υπάρχει ήδη κάποια διαδικασία που χρειάζεται να επεκταθεί σε on-line υπηρεσία. Επίσης το κόστος ενσωμάτωσης ενός web service σε κάποιο website ή σε μία δικτυακή εφαρμογή είναι πάρα πολύ μικρό.
- *Χρήση RPC μηχανισμών:* Η δυνατότητα κλήσης τους με χρήση RPC μηχανισμών βασισμένων στην XML που δεν εμποδίζονται από firewalls.

Όπως είναι φυσικό οι υπηρεσίες ιστού παρουσιάζουν και μειονεκτήματα:

- *Επιβάρυνση:* Η μετάδοση των δεδομένων, τα οποία είναι κωδικοποιημένα σε XML, δεν είναι τόσο αποδοτική όσο θα ήταν αν χρησιμοποιούνταν αποκλειστικά χρήση δυαδικού κώδικα. Έτσι, ενώ κερδίζουμε σε φορητότητα –portability- έχουμε επιβάρυνση όσον αφορά την απόδοση.

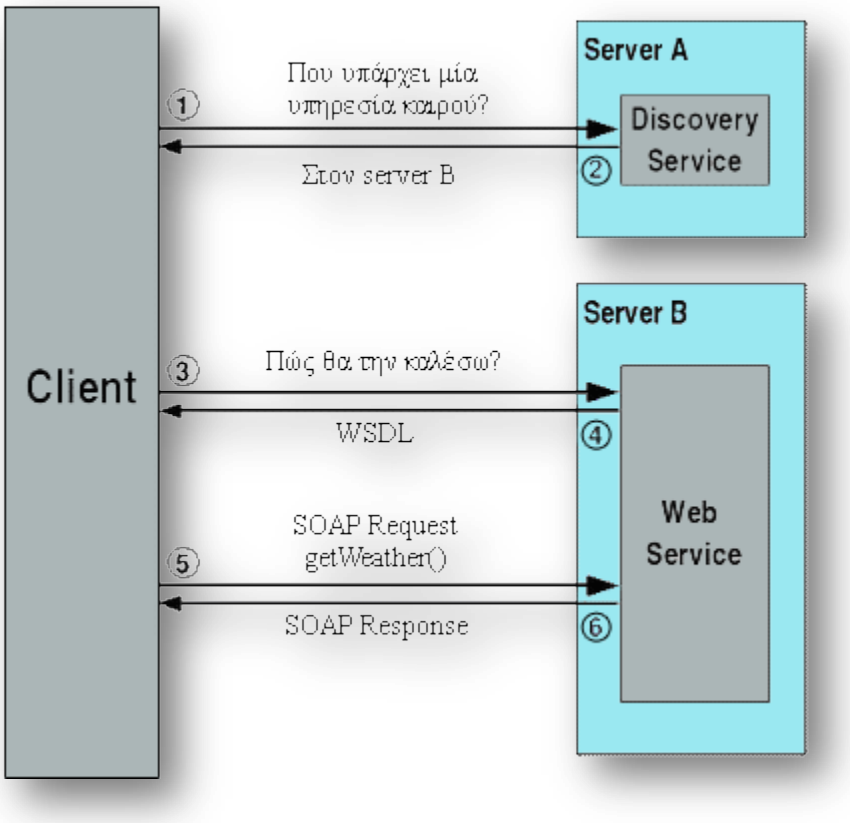
- *Έλλειψη ευελιξίας:* Οι Web services επιτρέπουν μόνο συγκεκριμένες μεθόδους κλήσης των υπηρεσιών τους, γεγονός που τις καθιστά όχι και τόσο ευπροσάρμοστες. Στη βελτίωση αυτής της κατάστασης συμβάλλουν νέες προδιαγραφές υπηρεσιών δικτύου, όπως το WSRF, οι οποίες προσάπτουν στις web services ευελιξία.

Αν συγκρίνουμε τις Web services με άλλες τεχνολογίες όπως η CORBA και η EJB θα διακρίναμε μία σημαντική διαφορά. Οι τελευταίες απευθύνονται σε συστήματα με υψηλό βαθμό διασύνδεσης, όπου παρατηρείται υψηλός βαθμός εξάρτησης μεταξύ πελάτη και εξυπηρετητή. Ενώ οι Web services προσανατολίζονται σε περιβάλλοντα χαλαρά συνδεδεμένων υπολογιστικών συστημάτων. Τα συστήματα με υψηλό βαθμό διασύνδεσης είναι ιδανικά για εφαρμογές intranet, αλλά δεν αποδίδουν αρκετά καλά σε δίκτυα μεγαλύτερης κλίμακας, όπως το internet. Αυτό καθιστά τις Web services κατάλληλες για εφαρμογές ευρέων δικτύων, όπως είναι τα υπολογιστικά πλέγματα.

Τα οφέλη που μπορούμε να αποκομίσουμε από την χρήση των Web services μπορούν να μεταφραστούν σε χρόνο και τελικό αποτέλεσμα. Φανταστείτε για παράδειγμα να θέλετε να δημιουργήσετε ένα website το οποίο θέλετε να παρέχει πολλές υπηρεσίες, όπως τον καιρό, την κίνηση στους δρόμους της πόλης, ένα εορτολόγιο, τις τιμές των μετοχών κτλ. Η ιδέα της δημιουργίας από το μηδέν ενός portal δεν είναι η σωστή λύση, όσον αφορά το κόστος και τον απαιτούμενο χρόνο. Η λύση είναι τόσο απλή όσο το να ανακαλύψει κανείς δωρεάν ή ακόμα και επί πληρωμή τα Web services που επιθυμεί και να επικοινωνήσει με τον αντίστοιχο SOAP server για να ενσωματώσει στην ιστοσελίδα του τις υπηρεσίες αυτές.

1.5.3 Παράδειγμα κλήσης μίας Web service

Παρακάτω απεικονίζεται μία συνήθης διαδικασία κλήσης μίας Web service.



Σχήμα 9: Κλήση μίας Web service

Ας δούμε πιο αναλυτικά τα παραπάνω βήματα.

1. Ένα χαρακτηριστικό των Web services είναι ότι συνήθως δεν έχουν γνώση για τις υπηρεσίες ιστού που καλούν. Για αυτόν το λόγο το πρώτο βήμα είναι να ανακαλυφθεί η υπηρεσία. Για να γίνει αυτό πρέπει να επικοινωνήσουμε με μία υπηρεσία ανακάλυψης υπηρεσιών, η οποία είναι με την σειρά της και η ίδια μια Web service.

2. Η υπηρεσία ανακάλυψης με την σειρά της, θα στείλει την απάντηση με ποιον server πρέπει να συνδεθούμε για να βρούμε την υπηρεσία που θέλουμε.

3. Το επόμενο βήμα είναι να συνδεθούμε με τον εξυπηρετητή που έχει την υπηρεσία που ζητάμε, για να μάθουμε πώς μπορούμε να την καλέσουμε. Έτσι θα ζητήσουμε από την υπηρεσία να μας δώσει μια περιγραφή της, δηλαδή το WSDL αρχείο για να ενημερωθούμε για τις μεθόδους της και τον τρόπο με τον οποίο μπορούμε να την καλέσουμε

4. Η υπηρεσία θα αποκριθεί και θα μας στείλει το WSDL αρχείο που ζητήσαμε.

5. Γνωρίζοντας τον τρόπο με τον οποίο μπορούμε να την καλέσουμε, θα χρησιμοποιήσουμε το πρωτόκολλο SOAP για να στείλουμε μια αίτηση για να μάθουμε τον καιρό σε μία πόλη.

6. Το τελευταίο βήμα αποτελείται από την απάντηση SOAP που θα λάβουμε, η οποία θα περιέχει ένα μήνυμα με τις πληροφορίες για τον καιρό που ζητήσαμε ή ένα μήνυμα σφάλματος, όταν η αίτηση SOAP είναι λανθασμένη.

1.5.4 Τεχνικά στοιχεία των Web Services

1.5.4.1 Μοντέλο Επιπέδων

Ξεκινώντας από το χαμηλότερο επίπεδο, αυτό της μεταφοράς δεδομένων, δημιουργείται ένα μοντέλο επιπέδων που χαρακτηρίζει την τεχνολογία των web services. Έτσι έχουμε από κάτω προς τα πάνω:

1.5.4.1.1 Επίπεδο Μεταφοράς

Ο σκοπός του επιπέδου αυτού είναι η μεταφορά δεδομένων των Web services. Τα πρωτόκολλα που χρησιμοποιούνται από τις Web services είναι κυρίως τα HTTP, SMTP, FTP. Σαφώς χρησιμοποιούνται και άλλα. Το πιο ευρέως χρησιμοποιούμενο όμως είναι το http, καθώς αποτελεί ένα πρότυπο για διαλειτουργικά συστήματα επειδή δεν μπλοκάρεται από firewalls.

1.5.4.1.2 Επίπεδο Μηνυμάτων

Ο σκοπός του επιπέδου αυτού είναι η περιγραφή της μορφής των δεδομένων με απώτερο σκοπό τη μεταφορά αυτών διαμέσου των διάφορων υπηρεσιών. Κύρια μορφή κωδικοποίησης των δεδομένων είναι η XML. Το πρωτόκολλο SOAP (Simple Object Access Protocol) καθορίζει τον τύπο των αιτήσεων που γίνονται στον εξυπηρετητή, και τον τύπο των αποκρίσεών του. Μπορούν να χρησιμοποιηθούν και άλλες γλώσσες για κλήση μίας υπηρεσίας, όπως η Xml-RPC ή κάποια άλλη XML, όμως το SOAP είναι μακράν η πιο δημοφιλής επιλογή για κλήση υπηρεσιών ιστού. Το SOAP είναι μια προδιαγραφή που καθορίζει πώς θα διαμορφωθεί και θα διαβαστεί ένα μήνυμα γραμμένο σε XML που χρησιμοποιείται από μια υπηρεσία. Ένα SOAP μήνυμα αποτελείται από τρεις περιοχές: το φάκελο, την επικεφαλίδα και το σώμα.

1.5.4.1.3 Επίπεδο Περιγραφής

Για κάθε μέθοδο της υπηρεσίας που ορίζουμε μία αντιστοιχία με μία λειτουργία του SOAP Server. Το σύνολο των λειτουργιών του SOAP Server, καθώς και τα υπόλοιπα χαρακτηριστικά του, περιγράφονται σε ένα αρχείο που ονομάζεται WSDL

(Web Service Description Language) και το οποίο δημοσιεύεται στο Internet, έτσι ώστε να δίνεται η δυνατότητα σε κάθε ενδιαφερόμενο χρήστη να μπορεί άμεσα να χρησιμοποιήσει την υπηρεσία. Το αρχείο αυτό περιέχει τις μεθόδους τις οποίες παρέχει η υπηρεσία, τα μηνύματα που δέχεται, καθώς και το πρωτόκολλο που πρέπει να χρησιμοποιηθεί για να κληθεί η υπηρεσία. Μία περιγραφή υπηρεσίας σε WSDL περιλαμβάνει ένα σύνολο από ειδικές διευθύνσεις (endpoints) που ανταλλάσσουν μηνύματα μεταξύ τους, σύμφωνα με το XML schema της W3C κοινότητας. Οι διεπιφάνειες των υπηρεσιών (portTypes) αποτελούν ακολουθίες απλών ανταλλαγών μηνυμάτων (operations) που δεσμεύονται από κάποιο πρωτόκολλο δικτύου, μία φόρμα κωδικοποίησης δεδομένων και μία endpoint διεύθυνση.

Οι υπηρεσίες ιστού, λοιπόν, χωρίζουν το πρωτόκολλο που χρησιμοποιείται για την αλληλεπίδραση μεταξύ τους σε τρία μέρη. Αυτά είναι η διεπαφή, η οποία περιέχει τα μηνύματα που ανταλλάσσονται, το πρωτόκολλο επικοινωνίας (message protocol, π.χ. SOAP) και το πρωτόκολλο μεταφοράς (π.χ. HTTP).

Οι υπηρεσίες ιστού δεν πραγματοποιούν όμως αποθήκευση της κατάστασής τους. Αυτό σημαίνει ότι η υπηρεσία δεν παρέχει πρόσβαση σε πληροφορία που δεν περιλαμβάνεται στα μηνύματα που λαμβάνει. Πολλές εφαρμογές όμως, και ιδιαίτερα οι εφαρμογές πλέγματος, απαιτούν αποθήκευση της κατάστασής τους. Τόσο το OGSA/OGSI όσο και το WSRF προτείνουν τροποποιήσεις των υπηρεσιών δικτύου που επιλαμβάνονται αυτού του προβλήματος.

1.5.4.2 Κατάλογος υπηρεσιών

Το UDDI (Universal Description, Discovery, and Integration) αποτελεί ένα πρωτόκολλο καταχώρησης για web services. Είναι από μόνο του ένα Web service και χρησιμοποιείται για να μπορούμε να παρέχουμε πληροφορίες για τα web services. Όπως αναφέραμε και παραπάνω το αρχικό μας βήμα είναι να εντοπίσουμε την υπηρεσία που μας ενδιαφέρει. Ο χώρος που θα την αναζητήσουμε είναι οι UDDI κατάλογοι. Τα πρωτόκολλα που χρησιμοποιούνται τόσο από τον πάροχο όσο και από τον καταναλωτή είναι τα SOAP και HTTP.

Οι κατάλογοι αυτοί παρέχουν πληροφορίες για το σημείο που μπορούμε να τις εντοπίσουμε , τεχνικές πληροφορίες που τις χαρακτηρίζουν και πληροφορίες για τον ιδιοκτήτη της εταιρίας που παρέχει την υπηρεσία και την πολιτική του. Κάθε καταχώρηση περιέχει το wsdl αρχείο και τη διεύθυνση που λειτουργεί η υπηρεσία στο Internet.

1.5.4.3 Τύποι καταχωρήσεων

Το ηλεκτρονικό εμπόριο, η αγορά και η πώληση των αγαθών και των υπηρεσιών πέρα από τα ηλεκτρονικά κανάλια, έχουν γίνει ένα φαινόμενο. Υπάρχουν διαφορετικοί τύποι καταχωρήσεων μίας υπηρεσίας. Πιο συγκεκριμένα, υπάρχουν καταχωρήσεις που μπορούν να γίνουν για υπηρεσίες από όλο τον κόσμο και που απευθύνονται σε όλο τον κόσμο, αλλά και καταχωρίσεις που απευθύνονται μόνο σε

εξειδικευμένες επιχειρήσεις. Υπάρχουν και καταχωρήσεις υπηρεσιών για πιο εξειδικευμένες περιπτώσεις. Το ηλεκτρονικό εμπόριο μπορεί να διαιρεθεί σε τρεις σημαντικές κατηγορίες:

- B2C (Business to Consumer): Εκείνες οι πτυχές του ηλεκτρονικού εμπορίου που περιλαμβάνουν τις πωλήσεις Διαδικτύου από τις επιχειρήσεις στους καταναλωτές.

- B2B (Business to Business): Εκείνες οι πτυχές του ηλεκτρονικού εμπορίου που περιλαμβάνουν την ανταλλαγή των αγαθών ή των υπηρεσιών μεταξύ των επιχειρήσεων μέσω του Διαδικτύου.

- C2C (Consumer to Consumer): Εκείνες οι πτυχές του ηλεκτρονικού εμπορίου που περιλαμβάνουν τις πωλήσεις Διαδικτύου από τους καταναλωτές στους καταναλωτές.

Κυρίως όμως προωθούνται οι τύποι καταχωρήσεων που εξυπηρετούν επιχειρήσεις, οι οποίες χρησιμοποιούν τις υπηρεσίες αυτές για εμπορικές και ενδοεπιχειρηματικές ολοκληρωμένες εφαρμογές. Η B2B κατηγορία περιλαμβάνει την ανταλλαγή των αγαθών ή των υπηρεσιών μεταξύ των επιχειρήσεων με τη λαμβάνουσα επιχείρηση που σκοπεύει να χρησιμοποιήσει το λαμβανόμενο αγαθό ή την υπηρεσία στην προώθηση των διαδικασιών παραγωγής της. Όλες οι επιχειρήσεις μπορούν να είναι συγχρόνως προμηθευτές αλλά και πελάτες άλλων επιχειρήσεων. Η B2B περιλαμβάνεται έτσι σε όλες τις πτυχές της αλυσίδας αυτής. Επομένως, τα B2B συστήματα είναι ένα μέρος των προσπαθειών μιας επιχείρησης να ελέγξει και να βελτιώσει τις σχέσεις της με τις άλλες επιχειρήσεις στην αλυσίδα.

Αλλά το ηλεκτρονικό εμπόριο δεν αφορά μόνο την αγορά και την πώληση στη παγκόσμια αγορά, αλλά την αναγνώριση των παγκόσμιων ευκαιριών και τη μείωση της παραγωγής και του κόστους πωλήσεων της επιχείρησης. Ιστορικά μπορούμε να αναγνωρίσουμε εκείνα τα B2B συστήματα που έχουν υπάρξει πολύ πριν εφευρεθούν οι υπολογιστές ή το Διαδίκτυο. Τέτοια συστήματα περιλαμβάνουν εκείνα που δεν ήταν αρχικά βασισμένα στην τεχνολογία, όπως το ταχυδρομικό σύστημα, το τραπεζικό σύστημα, το σύστημα λογιστικής, το νομικό σύστημα, το φορολογικό σύστημα, κ.λπ. Στους χρόνους τους, αυτά τα συστήματα θεωρήθηκαν υποδείγματα των καινοτόμων μέσων να διευκολυνθεί η ανταλλαγή και πληροφοριών και αγαθών μεταξύ των επιχειρησιακών οντοτήτων. Σήμερα, όλα αυτά τα συστήματα χρησιμοποιούν την τεχνολογία σε μεγάλο μέρος. Άλλα συστήματα τεχνολογίας που εκπλήρωσαν τους παρόμοιους ρόλους στο πρόσφατο παρελθόν περιλαμβάνουν τον τηλέγραφο, το τηλέφωνο, και τα τηλεοπτικά συστήματα. Λαμβάνοντας υπόψιν αυτήν την προοπτική, μπορούμε να πούμε ότι τα B2B συστήματα δεν είναι νέα αλλά έχουν εξελιχθεί από τα παλαιότερα. Η κύρια διαφορά εντούτοις είναι ότι οι τεχνολογίες ενημέρωσης και επικοινωνιών (Information and Communication Technologies) είναι τώρα βασικά συστατικά των B2B συστημάτων.

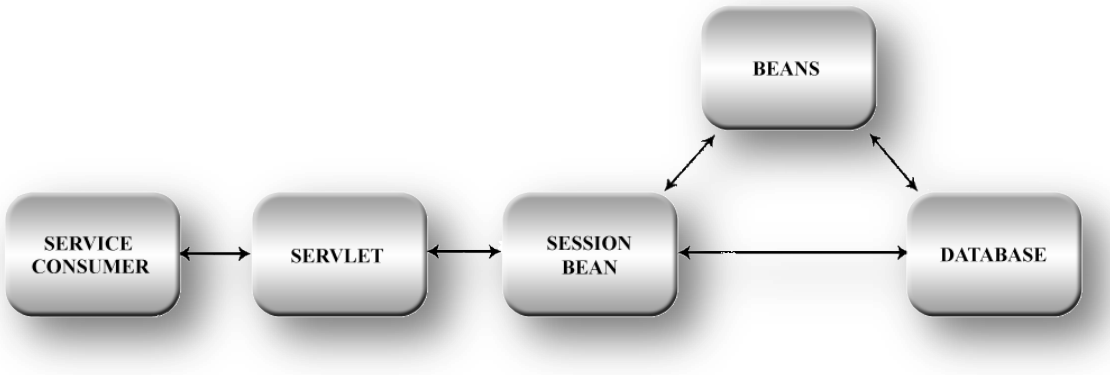
Τα ευρέως αποδεκτά πρότυπα όπως η Web Services Description Language (WSDL) [WSDL 2001], το Simple Object Access Protocol (SOAP) [SOAP 2000], οι κατάλογοι καταχωρήσεων υπηρεσιών, όπως το UDDI [UDDI 2003] και το ebXML [ebXML 2003], καθιστούν δυνατή τη δυναμική επίκληση Υπηρεσιών Ιστού. Δηλαδή όταν η υπηρεσία που χρησιμοποιείται είναι γνωστή, η περιγραφή WSDL της μπορεί να προσεγγιστεί από το registry μέσω ενός προγράμματος που χρησιμοποιεί τις πληροφορίες στην WSDL περιγραφή, τη σύνδεση και τις διαδικασίες για να έχει πρόσβαση δυναμικά στην υπηρεσία. Εντούτοις για να είναι σε θέση να εκμεταλλευτεί κανείς πλήρως τις υπηρεσίες ιστού, οι σημασιολογικές τους πληροφορίες πρέπει να είναι διαθέσιμες. Αυτό μπορεί να επιτευχθεί μέσω του ebXML. Το ebXM είναι ένα σύνολο προδιαγραφών για XML μηνύματα που προσφέρεται από τον OASIS μαζί με τον UN/CEFACT. Ένα ebXML registry επιτρέπει τον καθορισμό της σημασιολογίας βασικά μέσω δύο μηχανισμών. Κατ' αρχάς, επιτρέπει στις ιδιότητες των αντικειμένων των registries να καθορίζονται μέσω «slots» και, αφετέρου, τα metadata μπορούν να αποθηκευτούν σε ένα registry μέσω ενός μηχανισμού «ταξινόμησης». Αυτές οι πληροφορίες μπορούν έπειτα να χρησιμοποιηθούν για την ανακάλυψη των υπηρεσιών, χρησιμοποιώντας τους μηχανισμούς αναζήτησης του ebXML.

1.5.4.4 Κατηγορίες υλοποίησης

Εξαιτίας της ποικιλίας των διάφορων πλατφορμών, στις οποίες μπορεί να αναφέρεται μία υπηρεσία ιστού, παρατηρούμε μία κατηγοριοποίηση, η οποία αναφέρεται στα μοντέλα υλοποίησης τους και η οποία ορίζει 4 τύπους, αυτόν της απλής υπηρεσίας, της σύνθετης υπηρεσίας, της ενδιάμεσης υπηρεσίας και αυτής του ενός διαδρόμου υπηρεσιών.

1.5.4.4.1 Απλή υπηρεσία

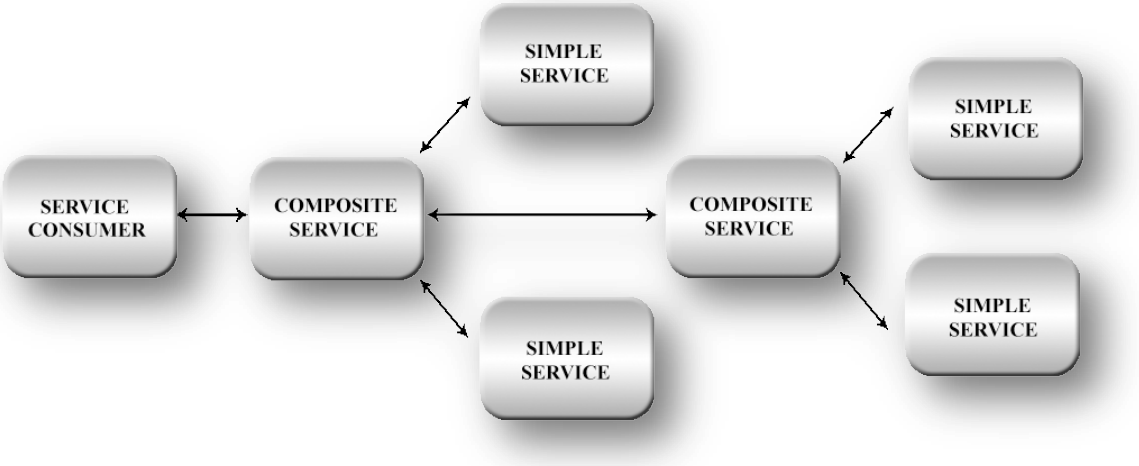
Στο παρακάτω σχήμα φαίνεται μια απλή υπηρεσία, όπου ένα servlet έχει πρόσβαση σε μια πηγή δεδομένων άμεσα.



Σχήμα 10: Απλή υπηρεσία

1.5.4.4.2 Σύνθετη Υπηρεσία

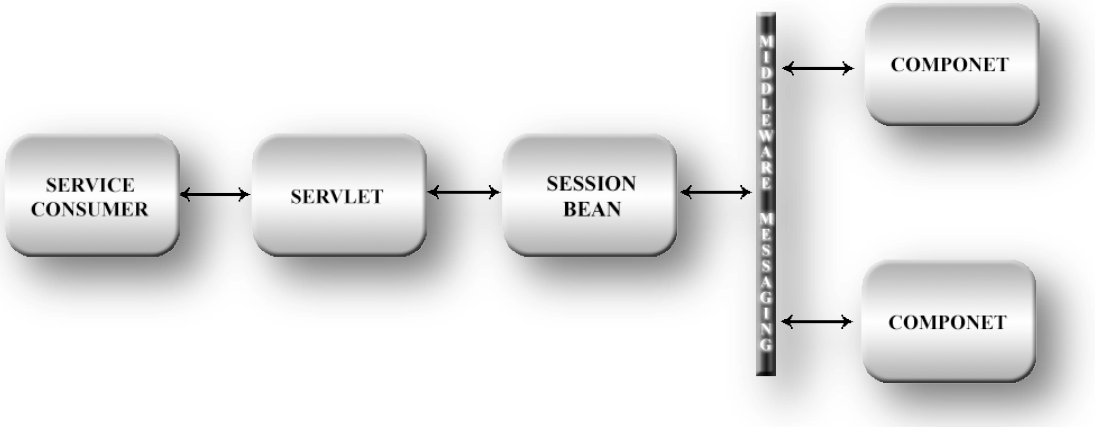
Μια σύνθετη υπηρεσία αποτελείται από τη σύνθεση απλών ή και σύνθετων υπηρεσιών. Και εκτός του πλεονεκτήματος να λύνει πιο πολύπλοκα προβλήματα, δίνει στον καταναλωτή της τη δυνατότητα να εκμεταλλευτεί ξεχωριστά την κάθε υπηρεσία. Στο παρακάτω σχήμα φαίνεται μια σύνθετη υπηρεσία.



Σχήμα 11: Σύνθετη υπηρεσία

1.5.4.4.3 Ενδιάμεση Υπηρεσία

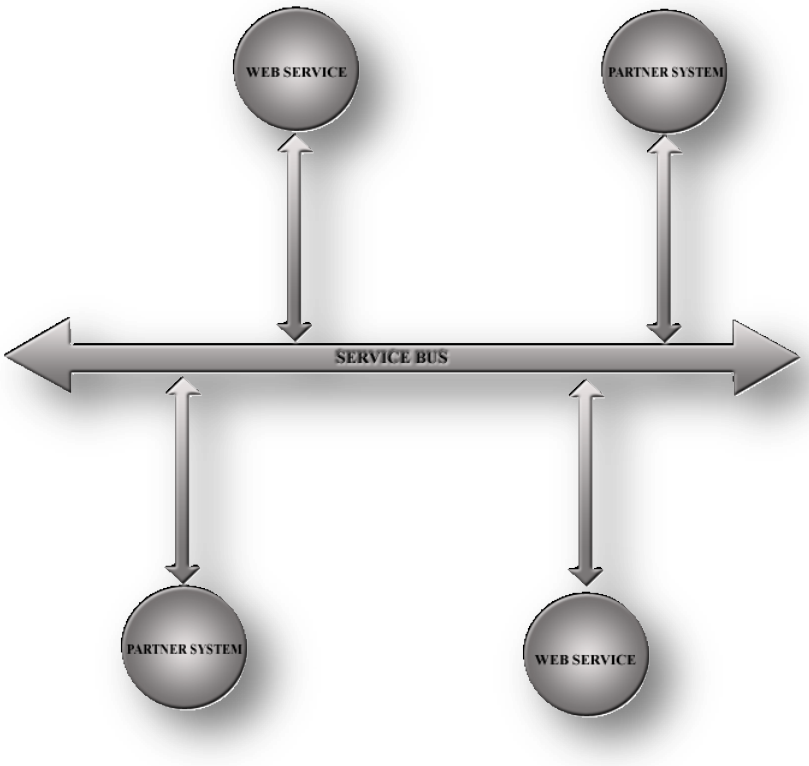
Μια web service μπορεί να χαρακτηριστεί ως ενδιάμεση υπηρεσία όταν για παράδειγμα λειτουργεί σαν ενδιάμεσος διάδρομος. Όταν ο αποστολέας δεν γνωρίζει τη διεύθυνση του παραλήπτη, μία υπηρεσία μεσολαβεί στην έμμεση αυτήν επικοινωνία προωθώντας το μήνυμα στη σωστή διεύθυνση. Στο παρακάτω σχήμα φαίνεται μια ενδιάμεση υπηρεσία.



Σχήμα 12: Ενδιάμεση υπηρεσία

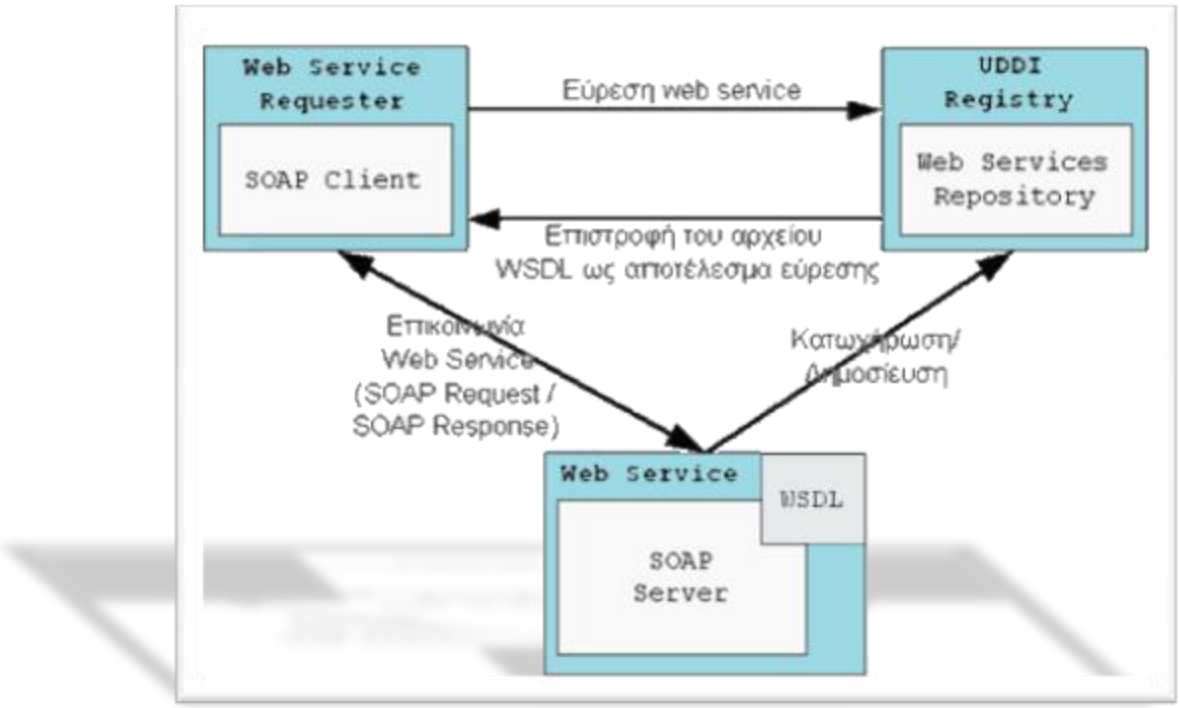
1.5.4.4.4 Διάδρομος Υπηρεσιών

Αναφερόμαστε σε διάδρομο υπηρεσιών όταν οι διάφορες υπηρεσίες μας επικοινωνούν μεταξύ τους μέσω τρίτων. Σε αυτήν την περίπτωση ο καταναλωτής δεν γνωρίζει τη διεύθυνση της υπηρεσίας. Τη δρομολόγηση των μηνυμάτων αναλαμβάνει το λογισμικό διαχείρισης του διαδρόμου. Επιπλέον ο καταναλωτής έχει τη δυνατότητα να αποστέλλει μηνύματα σε πολλές υπηρεσίες ταυτόχρονα. Στο παρακάτω σχήμα φαίνεται ένας διάδρομος υπηρεσιών.



Σχήμα 13: Διάδρομος υπηρεσιών

Το μοντέλο και η χρήση των Web services φαίνεται παρακάτω:



Σχήμα 14: Μοντέλο των Web services

2 WSRF - MUSE

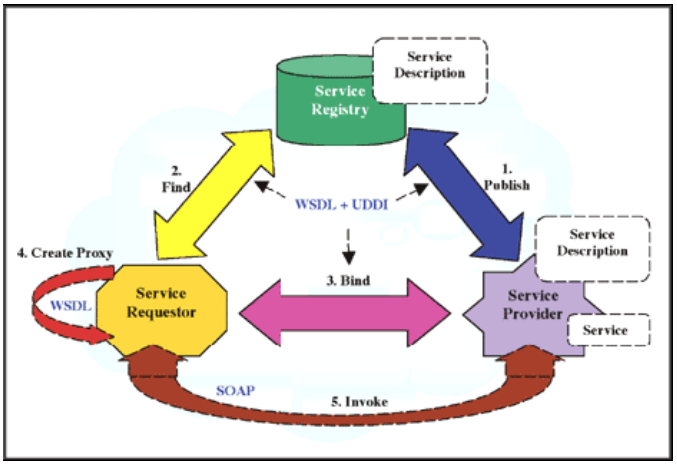
2.1 Web Service Resource Framework -WSRF

2.1.1 Τι είναι το WSRF

Οι Web services πρέπει συχνά να παρέχουν στους χρήστες τους τη δυνατότητα να προσεγγίσουν και να χειριστούν καταστάσεις, δηλ., τιμές δεδομένων που εξελίσσονται και με τις οποίες αλληλεπιδρούν. Και ενώ οι υπηρεσίες Ιστού διαχειρίζονται επιτυχώς τις καταστάσεις δεδομένων σήμερα, πρέπει να καθορίσουμε τις συμβάσεις για τη διαχείριση των καταστάσεων, έτσι ώστε οι εφαρμογές να ανακαλύπτουν και να αλληλεπιδρούν με stateful resources με τους τυποποιημένους και διαλειτουργικούς τρόπους. Το πλαίσιο των WS-Resources καθορίζει αυτές τις συμβάσεις μέσα στο πλαίσιο των καθιερωμένων προτύπων των υπηρεσιών Ιστού.

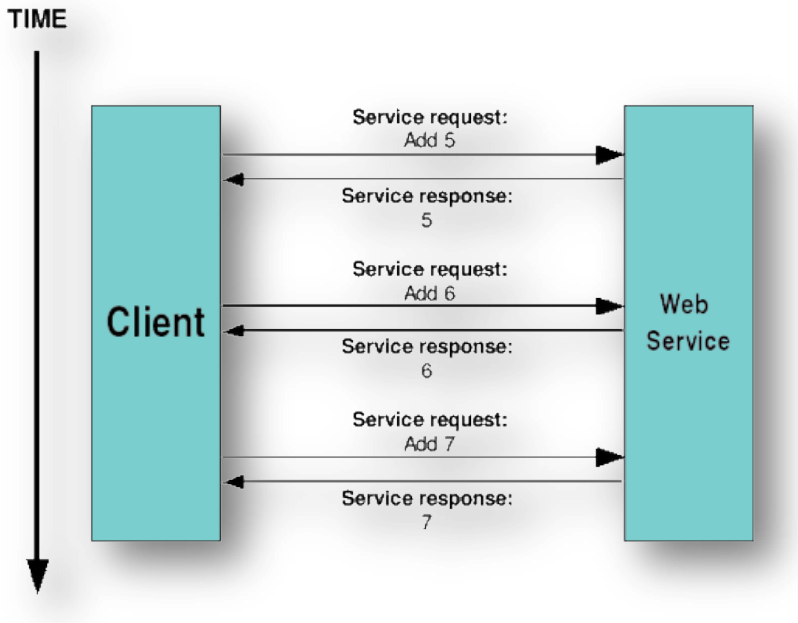
Η WSRF είναι μια προδιαγραφή για τη μοντελοποίηση και την πρόσβαση stateful resources χρησιμοποιώντας Web services. Περιλαμβάνει μηχανισμούς για την περιγραφή της κατάστασης τους, για την υποστήριξη και διαχείριση αυτής μέσω ιδιοτήτων που σχετίζονται με Web services καθώς και για την περιγραφή της επεκτασιμότητας των μηχανισμών αυτών σε ομάδες Web services. Δηλαδή είναι μια προδιαγραφή που καθορίζει τη σχέση μιας Web Service με ένα Resource (WS-Resource) που την αφορά, καθώς επίσης και τον τρόπο με τον οποίο οι διάφορες Web Services έχουν πρόσβαση στα αντίστοιχα WS Resources, γεγονός στο οποίο αναφερόμαστε με τον όρο WS-Resource Access Pattern και τέλος διάφορες πραγματικές υλοποιήσεις του WS-Resource Access Pattern. Οι εξελίξεις στις αρχιτεκτονικές πλέγματος (Grid Computing) οδηγούν στην ενοποίηση του Grid και των Web Services και αυτό γίνεται μέσω του WSRF. Έτσι το WSRF δείχνει πώς το Grid computing μπορεί να επωφεληθεί και να χρησιμοποιήσει service-oriented architectures (SOA).

Στο ακόλουθο σχήμα βλέπουμε την αλληλεπίδραση του WSRF με την SOA αρχιτεκτονική:



Σχήμα 15: WSRF και SOA

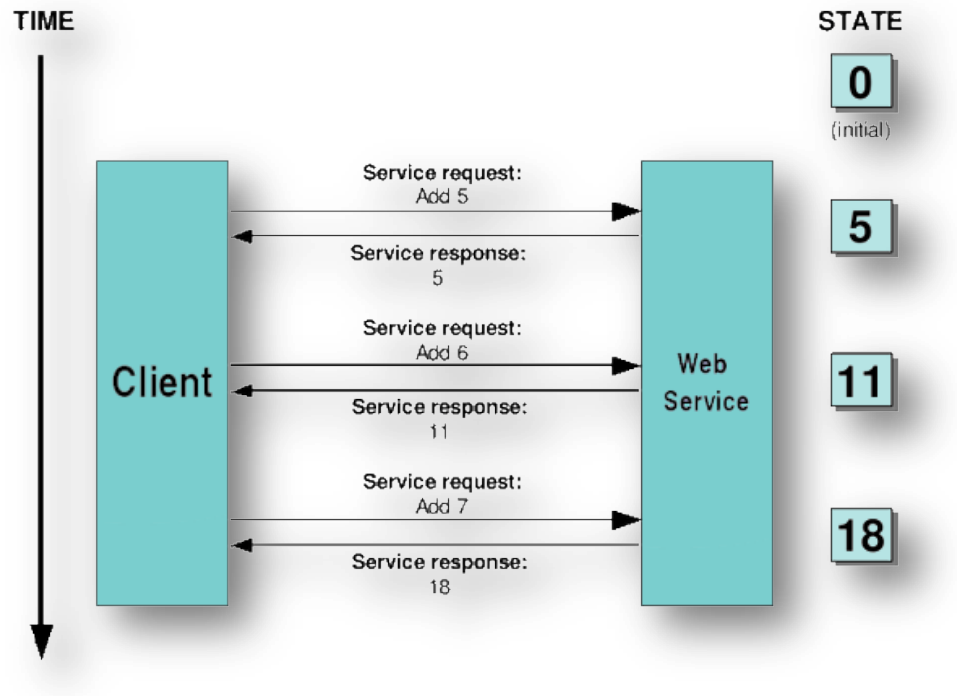
Παρακάτω φαίνεται μία stateless Web service:



Σχήμα 16: Stateless Web service

Το γεγονός ότι μία Web service είναι stateless δεν σημαίνει ότι είναι απαραιτήτως κακό. Υπάρχει μία πληθώρα υπηρεσιών που δε χρειάζεται να κρατάει καταστάσεις. Όπως για παράδειγμα, μία υπηρεσία που μας ενημερώνει για τον καιρό δεν χρειάζεται για να λειτουργήσει να έχει την πληροφορία του καιρού για τις προηγούμενες μέρες.

Παρόλα αυτά, οι εφαρμογές του Grid απαιτούν συνήθως τη δυνατότητα συγκράτησης καταστάσεων, όπως φαίνεται στο παρακάτω σχήμα:



Σχήμα 17: Statefull Web service

2.1.2 Η WSRF προδιαγραφή

Βασικός άξονας της προσέγγισης των υπολογιστικών πλεγμάτων που εισάγει το WSRF είναι η δημιουργία πόρων με δυνατότητα αποθήκευσης της κατάστασής τους, έτσι ένας πόρος (WS – Resource) πρέπει να περιλαμβάνει ένα συγκεκριμένο σύνολο δεδομένων της κατάστασής του, να έχει έναν καλά ορισμένο κύκλο ζωής και τέλος να υπόκειται σε ανακάλυψη και χρήση από πολλές υπηρεσίες πλέγματος.

Ένας πόρος λοιπόν, με δυνατότητα αποθήκευσης κατάστασης, δεν είναι ο ίδιος μια υπηρεσία ιστού, αλλά ελέγχεται από υπηρεσίες. Η κατάσταση ενός πόρου ορίζεται από τις τιμές που περιέχονται σε ένα ξεχωριστό κείμενο ιδιοτήτων του πόρου (WS – Resource Properties Document).

Για να πετύχει τους στόχους της, η προδιαγραφή πρέπει να τηρεί τις εξής απαιτήσεις:

- Να ορίσει τον όρο “resource”.
- Να ορίσει τον όρο “WS-Resource”, περιγράφοντας τη σχέση μεταξύ Web services και resources.
- Να ορίσει τον όρο “WS-Resource Access Pattern”, τον τρόπο με τον οποίο ένα resource μπορεί να διακριθεί σε μία ανταλλαγή μηνύματος μίας Web service και του αιτούντα.
- Να ορίσει μία ή περισσότερες πραγματικές υλοποιήσεις του WS-Resource Access Pattern.

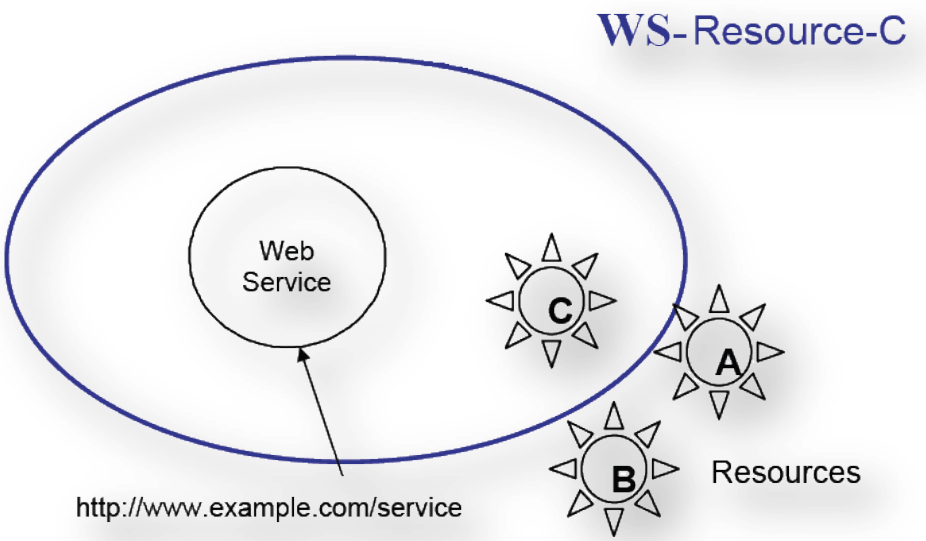
Έτσι λοιπόν ένα *resource* είναι μια λογική οντότητα με τα παρακάτω χαρακτηριστικά:

- Πρέπει να είναι αναγνωρίσιμο, πρέπει να έχει τουλάχιστον έναν resource identifier. Ο αναγνωριστής αυτός ενσωματώνει επαρκείς πληροφορίες για να διαχωρίζεται το ένα resource από το άλλο.
- Πρέπει να έχει ένα σύνολο από μηδέν ή περισσότερες ιδιότητες, οι οποίες θα εκφράζονται σε XML μορφή.
- Πρέπει να έχει ορισμένο κύκλο ζωής.

Ένα *WS-Resource* είναι μία υπηρεσία Ιστού μέσω της οποίας ένα resource μπορεί να προσπελαστεί. Ένα WS-Resource καθορίζεται περαιτέρω ως εξής:

- Ένα προσδιοριστικό του resource πρέπει να εμφανιστεί ως τμήμα οποιουδήποτε μηνύματος σε έναν WS-πόρο για να επιτρέψει στο WS-resource να αποσαφηνίσει τον πόρο που στοχεύει το μήνυμα. Αναφερόμαστε σε αυτό το σχέδιο πρόσβασης ως "WS-Resource Access Pattern".
- Το σύνολο ιδιοτήτων του resource πρέπει να εκφραστεί χρησιμοποιώντας XML περιγραφή. Το WS-Resource πρέπει να υποστηρίζει πρόσβαση στις ιδιότητες του πόρου μέσω των ανταλλαγών μηνυμάτων που καθορίζονται από τις ιδιότητες του WS-Resource specification.
- Εάν η πρόσβαση στον κύκλο ζωής ενός resource εκτίθεται μέσω ενός WS-Resource, τότε το τελευταίο μπορεί να υποστηρίζει τις ανταλλαγές μηνυμάτων που καθορίζονται το WS-Resource Lifetime specification.

Τέλος, μια *αναφορά σε ένα WS-Resource* (ή απλώς αναφορά) είναι μια απεικόνιση μέσω της οποίας ένα WS-Resource μπορεί να προσπελαστεί. Μια αναφορά περιέχει έναν αναγνωριστή για το resource και μπορεί να περιέχει ακόμα επιπλέον πληροφορίες απαραίτητες για να προσπελαστεί ένα WS-Resource. Για ένα δεδομένο αναγνωριστή resources μπορούν να υπάρξουν πολλές αναφορές.



Σχήμα 18: WS-Resource

Στο παραπάνω παράδειγμα βλέπουμε μία Web service που έχει URL με διεύθυνση http://www.example.com/service. Αυτή η Web service έχει πρόσβαση σε τρία resources τα "A", "B" και "C". Ως WS-Resource-C ονομάζεται η σύνθεση της Web service και του resource C. Μία αναφορά στο WS-Resource-C μπορεί να εμφανιστεί ως εξής:

```
<wsa:EndpointReference>
<wsa:Address>
http://www.example.com/service?res=C
</wsa:Address>
...
</wsa:EndpointReference>
```

Ένα μήνυμα στο WS-Resource, το οποίο θα χρησιμοποιεί SOAP binding θα είναι:

```
<s11:Envelope…>
<s11:Header>
<wsa:To>http://www.example.com/service?res=C</wsa:To>
...
</s11:Header>
<s11:Body>
...
</s11:Body>
</s11:Envelope>
```

2.1.3 WS-Resource Access Pattern Embodiments

Όπως έχει αναφερθεί και παραπάνω, ο όρος WS-Resource Access Pattern ορίζει μια έννοια περιγράφοντας πώς μία Web service αναγνωρίζει σε ποιο resource απευθύνεται ένα μήνυμα σε ένα WS-Resource. Υπάρχουν πολλοί τρόποι με τους οποίους μπορεί να γίνει αυτό. Οι υλοποιήσεις αυτές του WS-Resource Access Pattern ονομάζονται "Embodiments". Ένα WS-Resourse πρέπει να υποστηρίζει τουλάχιστον ένα embodiment.

Κάθε embodiment του WS-Resource Access Pattern πρέπει να:

- Καθορίζει την μορφή της WS-Resource αναφοράς
- Καθορίζει πώς ο αναγνωριστής ενός resource εμφανίζεται σε μία αναφορά ενός WS-Resource
- Καθορίζει πώς ένας αναγνωριστής ενός resource εμφανίζεται σε ένα μήνυμα.

2.1.4 WS-Addressing

Υπάρχουν περιπτώσεις όπου ένας πελάτης χρησιμοποιεί ταυτόχρονα πολλούς πόρους, ενώ ένας πόρος μπορεί επίσης να προσπελαύνεται ταυτόχρονα από πολλούς χρήστες. Έτσι, υπάρχουν πολλά στιγμιότυπα του ίδιου πόρου (WS – Resources), με την ίδια διεπαφή (interface) στον ίδιο χρόνο. Όταν ο πελάτης στέλνει ένα μήνυμα σε έναν εξυπηρετητή που μοιράζεται έναν πόρο με άλλους, ο εξυπηρετητής πρέπει να γνωρίζει σε ποιο στιγμιότυπο του πόρου θα πρέπει να στείλει το μήνυμα. Το WSRF χρησιμοποιεί ένα διαφορετικό μοντέλο για τη διευθυνσιοδότηση των πόρων (WS – Addressing model). Με βάση αυτό ορίζεται μια, ανεξάρτητη δικτύου, αναφορά δείκτη (endpoint) για κάθε υπηρεσία ιστού. Στην περίπτωση του WSRF, η αναφορά είναι δείκτης σε κάποιο πόρο (WS – Resource).

Έτσι, έχουμε υλοποιήσεις όπου χρησιμοποιείται η WS διευθυνσιοδότηση (WS-Addressing). Σε αυτήν την υλοποίηση, μία μορφή αναφοράς σε ένα WS-Resource είναι μία αναφορά δείκτη (endpoint) ή ακριβέστερα ένα XML element, του οποίου ο τύπος προέρχεται από το σύνθετο τύπο που ονομάζεται EndpointReferenceType και ορίζεται από την WS-Addressing specification.

Η διεύθυνση του δείκτη της Web service περιέχεται στις πληροφορίες του "wsa:Address element". Υπάρχουν δύο τρόποι με τους οποίους μπορεί να εμφανιστεί ο αναγνωριστής ενός resource. Ο πρώτος είναι να είναι στα περιεχόμενα του "wsa:ReferenceProperty" στοιχείου του δείκτη αναφοράς και ονομάζεται "WS-Addressing embodiment using Reference Properties" ενώ ο δεύτερος να είναι ενσωματωμένος ως μέρος των πληροφοριών του "wsa:Address" στοιχείου του δείκτη αναφοράς και ονομάζεται "WS-Addressing embodiment using Address"

Το παρακάτω παράδειγμα απεικονίζει μία συλλογή από WS-Resources:

Σχήμα 19: Συλλογή από WS-Resources

Υπάρχει μία Web service με URL με διεύθυνση http://www.example.com/service και η οποία παρέχει πρόσβαση σε δύο resources, τα οποία ορίζονται απλά ως "R1" και "R2". Μία αναφορά στο WS-Resource που θα σχετίζονταν με την Web service και τον αναγνωριστή του resource με την τιμή "R1" θα ήταν ως εξής:

```
<wsa:EndpointReference>
                <wsa:Address>http://www.example.com/service</wsa:Address>
                <wsa:ReferenceProperties>
 <tns:SomeDisambiguatorElement>R1</tns:SomeDisambiguatorElement>
                </wsa:ReferenceProperties> ?
…
        </wsa:EndpointReference>
```

Αυτή η αναφορά χρησιμοποιεί την υλοποίηση που ονομάζεται "WS-Addressing embodiment using Reference Properties". Ένα παράδειγμα για ένα μήνυμα GetResourceProperties, με SOAP/HTTP binding, σύμφωνα με την υλοποίηση WS-Resource Access Pattern θα ήταν ως εξής:

<

```
S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/2004/11/wsrf-WS203
ResourceProperties-1.2-draft-05.xsd">
   <S:Header>
        <wsa:To> http://www.example.com/service </wsa:To>
        <wsa:Action>
http://docs.oasis-open.org/wsrf/2004/11/WSResourceProperties /GetResourceProperty
        </wsa:Action>
```

```
<tns:SomeDisambiguatorElement>R1</tns:SomeDisambiguatorElement>
        ...
    </S:Header>
    <S:Body>
        <wsrf-rp:GetResourceProperty…
         ...
    </S:Body>
</S:Envelope>
```

Ο τρόπος αυτός διευθυνσιοδότησης επιτρέπει επίσης την παροχή επιπρόσθετων πληροφοριών μαζί με τη διεύθυνση του πόρου. Οι πληροφορίες και η διεύθυνση του πόρου παρέχονται σε μορφή XML από ένα αναγνωριστικό πόρου (resource identifier). Αυτό το αναγνωριστικό χρησιμοποιείται για να ξεχωρίζει το ένα στιγμιότυπο από το άλλο. Μια αναφορά δείκτη (WS – Addressing endpoint reference) πρέπει να περιλαμβάνει ένα «στοιχείο-παιδί» που θα αναφέρει τις ιδιότητες του πόρου και θα αναγνωρίζει αυτόν που θα σχετίζεται με τις ανταλλαγές μηνυμάτων που θα πραγματοποιούνται με αυτήν την αναφορά δείκτη. Οι πληροφορίες που περιέχει μια αναφορά δείκτη είναι ασήμαντες για τις υπηρεσίες δικτύου και χρησιμοποιούνται από το μοντέλο διευθυνσιοδότησης μόνο, για την αντιστοίχηση της αναφοράς με τον εκάστοτε πόρο. Το μοντέλο χρησιμοποιεί μια προσέγγιση μεταφοράς ανεξάρτητης πρωτοκόλλου, για να συνδέσει τον αποστολέα και τον παραλήπτη. Αυτό σημαίνει ότι μπορεί να υλοποιηθεί πάνω από το SOAP ή οποιοδήποτε άλλο πρωτόκολλο.

2.1.5 WSDL Service Element Embodiment

Σε αυτήν την υλοποίηση χρησιμοποιείται η WSDL. Η μορφή της αναφοράς αυτής είναι ένα WSDL στοιχείο ορισμών το οποίο περιέχει ακριβώς ένα παιδί στοιχείο το οποίο με τη σειρά του περιέχει ένα ή περισσότερα WSDL port παιδιά, καθένα από αυτά δεμένα με το ίδιο portType. Κάθε port προσφέρει ένα πιθανό binding στο ίδιο WS-resource. Ακολουθεί ένα παράδειγμα μίας αναφοράς σε ένα WS-Resource:

```
<wsdl:definitions … >
            <wsdl:service name=”svc”>
                <wsdl:port … >
                    <soap:address=”http://www.example.com/R1”/>
                </wsdl:port>
            </wsdl:service>
    </wsdl:definitions>
```

Στη συγκεκριμένη περίπτωση, τα μηνύματα που στέλνονται στο http://www.example.com/R1 στέλνονται στην πραγματικότητα στην Web service, η οποία προσφέρει πρόσβαση στο resource, που στο συγκεκριμένο παράδειγμα αναφέρεται ως “R1”. Πρέπει να σημειώσουμε ότι ακόμα και αν ο αναγνωριστής ενός resource δεν εμφανίζεται μέσα στο SOAP envelope που περιέχεται στον φάκελο που

σχετίζεται με τις αναφορές, πρέπει να εμφανίζεται σαν μέρος του HTTP μηνύματος με την μορφή ενός URL.

2.1.6 WS – Resource Properties Document

Τα δεδομένα κάθε πόρου μεταβάλλονται κατά τη διάρκεια ζωής του. Στο WSRF η κατάσταση ενός πόρου περιγράφεται σε ένα κείμενο ιδιοτήτων υπηρεσίας, το οποίο προσπελαύνεται από απλές ανταλλαγές μηνυμάτων. Τις ανταλλαγές αυτές δεν είναι ανάγκη να πραγματοποιούν συγκεκριμένες λειτουργίες, αλλά το WSRF έχει υλοποιήσει μερικές λειτουργίες εντός του (WS – Resource Properties Document) για την ανάκτηση και τροποποίηση στοιχείων XML που περιέχουν τις ιδιότητες του πόρου. Αυτές είναι οι GetResourceProperty, η οποία επιστρέφει την τιμή ενός XML στοιχείου δεδομένου του ονόματός του, η GetMultipleResourceProperties, η οποία επιστρέφει τιμές για πολλά στοιχεία, η SetResourceProperties και η QueryResourceProperties.

2.1.7 Notifications

Το βασικό σχήμα γνωστοποιήσεων ορίζεται στο WS – BaseNotification. Το μοντέλο WS – Resource ορίζει μια περιγραφή XML των θεμάτων. Ένας πελάτης μπορεί να δηλώσει ότι επιθυμεί να λαμβάνει γνωστοποιήσεις για τα συγκεκριμένα θέματα σύμφωνα με τους όρους που ορίζει το πλαίσιο WS – Topics.

Η διεπαφή WS – Topics ορίζει ένα μηχανισμό για την οργάνωση και κατηγοριοποίηση στοιχείων περιεχομένου που μπορεί να ενδιαφέρουν για την αποστολή γνωστοποιήσεων.

2.2 Muse

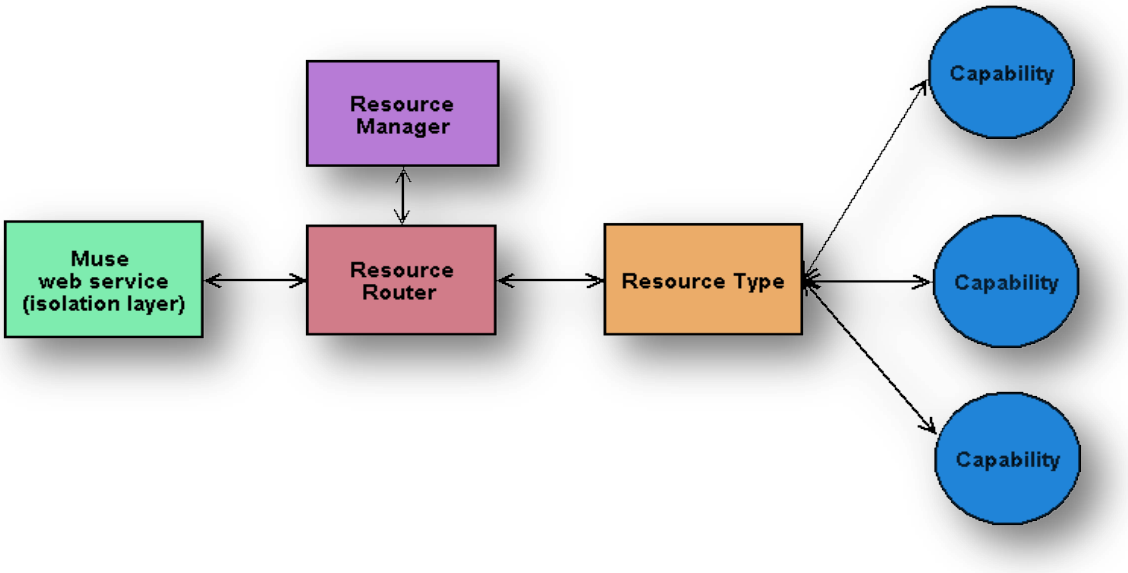
2.2.1 Τι είναι το Muse

Το Apache Muse Project είναι μία βασισμένη στην Java υλοποίηση του WS-ResourceFramework, WS-BaseNotification και WS-DistributedManagement. Είναι ένα πλαίσιο πάνω στο οποίο μπορούν να χτιστούν διασυνδέσεις των Web Services για τη διαχείριση των resources αποφεύγοντας τα ανωτέρω χρονοβόρα και κουραστικά specifications. Εφαρμογές του Muse μπορούν να υλοποιηθούν είτε με τον Apache Access 2, είτε σε OSGi περιβάλλοντα.

Η σχεδίαση του Apache Muse έγινε με τέσσερις αρχές:

- The Capability – Η capability μιας πηγής είναι μια συλλογή δεδομένων και λειτουργιών, η οποία εκτίθεται δια μέσου των web services. Είναι η βασική μονάδα σχεδίασης του Muse. Οι Capabilities συναθροίζονται για να καθορίσουν τον τύπο ενός resource και καθορίζονται ως απλές κλάσεις της Java. Οι κλάσεις των capabilities επιθεωρούνται κατά την αρχικοποίηση για να καθορισθεί πώς ταιριάζουν στο interface του αρχείου WSDL των resources.
- The Resource – Ένα Resource είναι μια συλλογή από capabilities που εκτίθενται διαμέσου ενός interface ενός Web service. Επίσης, λειτουργεί ως διασύνδεση μεταξύ των capabilities που επιθυμούν να επικοινωνήσουν με κάποια άλλα capabilities.
- The Implied Resource Pattern – Κάθε resource έχει ένα μοναδικό endpoint reference (EPR). Αυτός ο τύπος δεδομένων καθορίζεται από την WS-Addressing προδιαγραφή για να περιέχει τα βασικά δεδομένα που χρειάζονται για να εντοπιστεί ένα συγκεκριμένο resource και για να καλέσουμε τις λειτουργίες του. Το Implied Resource Pattern επιτρέπει στους χρήστες να δώσουνε ονόματα σε κάθε EPR με σκοπό να διαχωριστούν τα διάφορα instances ενός τύπου resource με την ίδια διεύθυνση (URL).
- The Isolation Layer – Τα resources του Muse μπορούν να αναπτυχθούν σε διαφορετικά περιβάλλοντα, όπως J2EE application servers, OSGI κτλ. αλλά παρόλα αυτά υπάρχει μόνο ένα προγραμματιστικό μοντέλο. Έτσι, οι προγραμματιστές μπορούν να αναπτύξουν τις εφαρμογές τους χωρίς αλλαγές στον κώδικα σε διάφορα περιβάλλοντα. Αυτή η μεταφερσιμότητα του κώδικα μειώνει το χρόνο εκμάθησης του Muse.

Το ακόλουθο διάγραμμα δείχνει πώς οι τέσσερις αυτές αρχές συνεργάζονται:



Σχήμα 20: Αρχές σχεδίασης του Apache Muse

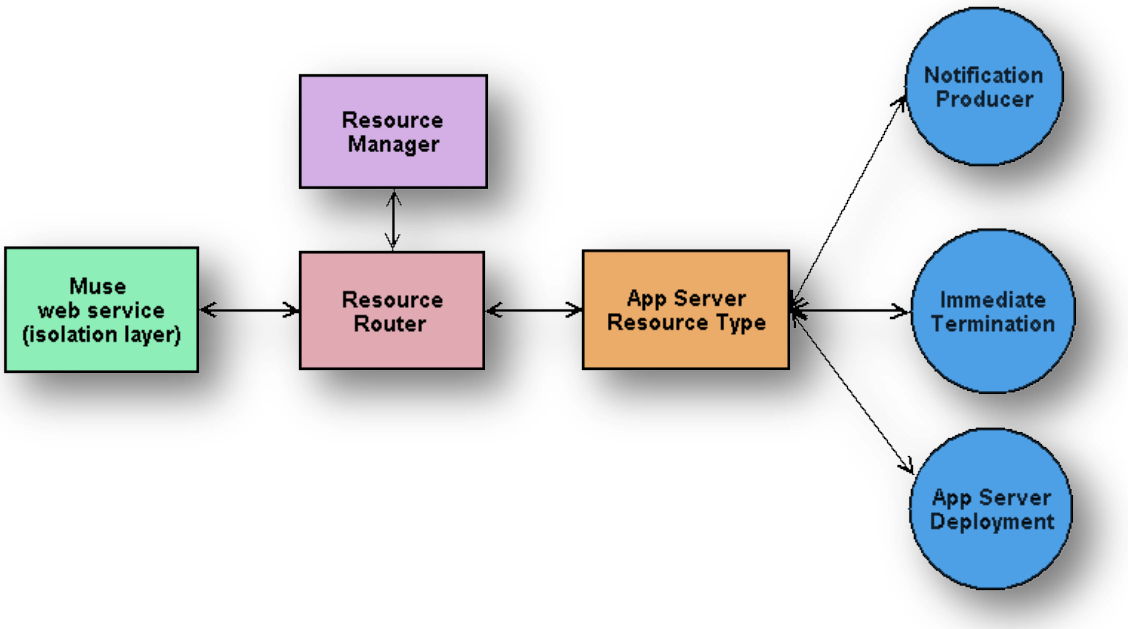
2.2.2 Σχηματισμός resources με την χρήση capabilities

Το WSDL αρχείο που καθορίζει την διασύνδεση ενός resource μίας web service πρέπει να ορίζει ένα port type για το resource που θα περιγράφει όλες τις δημόσιες υπηρεσίες του και θα παρέχει συνδέσεις στον πελάτη που επιθυμεί να τις προσπελάσει και να τις διαχειριστεί. Το WSDL αρχείο δεν παρέχει καμία πληροφορία ή αναφορά στον τρόπο που το resource έχει υλοποιηθεί.

Όλα τα Muse resources είναι συλλογές από μικρότερες μονάδες ή λειτουργίες που ονομάζονται capabilities. Η ιδέα του capability περιγράφηκε αρχικά στο WSDM MUWS 1.0 ως μέσο επικοινωνίας με τους πελάτες όσον αφορά την ανταλλαγή δεδομένων και λειτουργιών. Παρόλο που χρησιμοποιούνταν τα capabilities για να διαφημίσουν τις λειτουργίες στους πελάτες, ο τρόπος που αυτό γίνεται στο WSDM MUWS specification θυμίζει τον τρόπο που οι Java developers σπάνε μεγάλα κομμάτια λογισμικού σε μικρά σχετικών λειτουργιών. Το Muse έχει γενικεύσει την ιδέα του capability του WSDN MUWS, για να το καταστήσει ευκολότερο για τους προγραμματιστές να υλοποιήσουν resources ως επαναχρησιμοποιούμενες μονάδες λειτουργιών αντί για κλάσεις Java.

Αν το επίπεδο των capabilities είναι το επίπεδο όπου οι προγραμματιστές υλοποιούν την λογική της υπηρεσίας, εξακολουθούμε να χρειαζόμαστε έναν τρόπο να συνδυάσουμε αυτές τις μονάδες για να ταιριάξουμε το WSDL interface που χρησιμοποιούν οι πελάτες για να εκμεταλλευτούν το resource.

Το resource είναι η ιδέα του Muse για την υλοποίηση της συνένωσης αυτής και παρέχει στα κατώτερα επίπεδα των web services ένα απλό interface για την αποστολή των εισερχόμενων μηνυμάτων SOAP. Στο εσωτερικό του Muse framework όλη αυτή η υλοποίηση δεν είναι ορατή από το ανώτατο επίπεδο του κώδικα, η SOAP μηχανή, το isolation layer, και ο router δε γνωρίζουν το προγραμματιστικό αυτό μοντέλο. Ως παράδειγμα, μία server resource εφαρμογή υλοποιημένη με το WSN NotificationProducer, WSRL ImmediateResourceTermination και κοινές AppServerDeployment capabilities θα έκρυβε την συνένωση αυτή, όπως φαίνεται στο παρακάτω σχήμα:



Σχήμα 21: Συνένωση υλοποίησης του Muse

Το Java αντικείμενο που εκπροσωπεί το instance του resource θα εκτελέσει την αποστολή των αιτήσεων στο κατάλληλο capability βασισμένο σε πληροφορίες που ανακτηθήκαν κατά την αρχικοποίηση. Όταν δημιουργείται ένα resource το Muse framework χρησιμοποιεί το WSDL του resource για να καθορίσει τι πρέπει να ικανοποιεί ένα resource και αν όντως δύναται να το κάνει. Αυτός ο έλεγχος γίνεται ερευνώντας όλους τους ορισμούς των capabilities για να βεβαιωθεί ότι οι λειτουργίες στο WSDL port type έχουν αντίστοιχες Java μεθόδους στη συλλογή των capabilities.

Τέλος, το resource είναι υπεύθυνο για να ελέγχει ότι οι capabilities θα λάβουν σωστά notifications από όλα τα resource lifecycle γεγονότα, έτσι ώστε να μπορούν να λάβουν τα κατάλληλα μέτρα που χρειάζονται κατά την αρχικοποίηση και τον τερματισμό. Η διασύνδεση του Capability που ορίζεται από το Muse διαθέτει τέσσερα βασικά lifecycle γεγονότα:

- **Initialization** – *Capability.initialize()* – Χρησιμοποιείται όταν η capability μπορεί να εκτελέσει μόνης της τα startup tasks, και δεν χρειάζεται να ανακατευτεί με τη χρήση άλλων capabilities που βρίσκονται στο resource.
- **Post-initialization** – *Capability.initializeCompleted()* – Χρησιμοποιείται όταν η capability μπορεί να εκτελέσει όλα τα εναπομείναντα startup tasks; Specifically, να αναζητήσει και να διαχειριστεί τα υπόλοιπα capabilities με τη γνώση ότι είναι σε μία σταθερή κατάσταση.
- **Pre-shutdown** – *Capability.prepareShutdown()* – Χρησιμοποιείται όταν η capability μπορεί να εκτελέσει shutdown tasks που περιλαμβάνουν αναζήτηση και διαχείριση άλλων capabilities. Αυτό ονομάζεται *last call* – μία ευκαιρία για το capability να πάρει ό,τι πληροφορίες χρειάζεται από

άλλα στοιχεία πριν αυτά τερματιστούν και η σταθερότητά τους είναι αμφίβολη.

- **Shutdown** – *Capability.shutdown()* – Εφοδιασμένη με όλα τα δεδομένα που χρειάζεται πριν από το shutdown, η capability μπορεί να εκτελέσει όλα τα shutdown tasks, συμπεριλαμβανομένου του persistence, του configuration ή των notifications.

Το Muse καθορίζει τις διεπαφές και τις εφαρμογές για όλα τα capabilities των WSRF, WSN, και WSDM. Αυτά πρέπει να χρησιμεύσουν ως ένα πρότυπο για όλους τους προγραμματιστές που καθορίζουν τα συνήθη capabilities για συγκεκριμένα χαρακτηριστικά γνωρίσματα των προϊόντων. Περαιτέρω, οι διεπαφές πρέπει να χρησιμεύσουν ως τα de facto πρότυπα στις χρήστες που επιθυμούν να παρέχουν εναλλακτικές εφαρμογές των τυποποιημένων capabilities για το resource. Στην πράξη τέτοια αντικατάσταση αναμένεται και σχεδιάζεται, αλλά η εμμονή στις διεπαφές του Muse είναι σημαντική προκειμένου να εξασφαλιστεί φορητότητα κώδικα.

Η προδιαγραφή WS-ResourceLifetime (WSRL) (που καθορίζεται στην οικογένεια WSRF των specs) καθορίζει τα δύο port types: ImmediateResourceTermination και ScheduledResourceTermination. Και τα δύο χαρτογραφούν τις μεμονωμένες ικανότητες του Muse. Το ScheduledResourceTermination capability ορίζει δύο ιδιότητες και μία λειτουργία:

- ***wsrf-rl:CurrentTime*** – ο χρόνος σύμφωνα με το σύστημα στο οποίο τρέχει η εφαρμογή.
- ***wsrf-rl:TerminationTime*** – η χρονική στιγμή κατά την οποία το resource είναι προγραμματισμένο να αυτόκαταστραφεί. Αν είναι μηδέν, το resource είναι αθάνατο.
- ***wsrf-rl:SetTerminationTime*** – μία λειτουργία που καθορίζεται από το WSRL για να καθορίσει την *wsrf-rl:TerminationTime* ιδιότητα.

Η Java διασύνδεση για αυτό το capability βρίσκεται στο jar αρχείο *muse-wsrf-api.jar* και ο κώδικας είναι ο εξής:

```
package org.apache.muse.ws.resource.lifetime;

import java.util.Date;

import javax.xml.namespace.QName;

import org.apache.muse.ws.resource.WsResourceCapability;
import org.apache.muse.ws.resource.basefaults.BaseFault;
import org.apache.muse.ws.resource.lifetime.faults.TerminationTimeChangeRejectedFault;
import org.apache.muse.ws.resource.lifetime.faults.UnableToSetTerminationTimeFault;
```

```
public interface ScheduledTermination extends WsResourceCapability

{
  QName[] PROPERTIES = new QName[]{
    WsrlConstants.CURRENT_TIME_QNAME,
    WsrlConstants.TERMINATION_TIME_QNAME
     };

  Date getCurrentTime()
    throws BaseFault;

  Date getTerminationTime()
    throws BaseFault;

  Date setTerminationTime(Date time)
    throws UnableToSetTerminationTimeFault, TerminationTimeChangeRejectedFault;
}
```

Ένα παράδειγμα για τη χρήση του ScheduledResourceTermination φαίνεται παρακάτω:

```
package org.apache.muse.ws.resource.lifetime.impl;

public class SimpleScheduledTermination

extends AbstractWsResourceCapability implements ScheduledTermination

{

  // when there is a valid wsrl:TerminationTime, this timer will
  // be set to execute a task that calls Resource.shutdown()
  private Timer _terminationTimer = null;

  public QName[] getPropertyNames()

  {    return PROPERTIES;  }

    public void initialize()
    throws SoapFault
  {
    super.initialize();
    // create the timer that can be used for acting on the
    // termination time, but don't start it
    TimerTask scheduledDestruction = new DestroyTimerTask(getWsResource());
    _terminationTimer = new Timer(scheduledDestruction);
  }
```

```
public Date getCurrentTime()

  {

    return new Date();
  }
  public Date getTerminationTime()
  {
    return _terminationTimer.getScheduledTime();
  }
  public Date setTerminationTime(Date time)
  {
    // no value - cancel timer (we are now immortal)
    if (time == null)
      _terminationTimer.cancel();
    // real value - if no previous value, start timer
    else if (_terminationTimer.getScheduledTime() == null)
      _terminationTimer.schedule(time);
    // real value - timer is started, so stop and start
    else
      _terminationTimer.reschedule(time);
    return time;
  }
  public void shutdown()
    throws SoapFault
  {
    _terminationTimer.cancel();
    super.shutdown();
  }
  class DestroyTimerTask extends TimerTask
  {
    private WsResource _resource = null;
    public DestroyTimerTask(WsResource resource)
    {
      _resource = resource;
    }
    // Invokes the resource's shutdown() operation.
    public void run()
    {
          try {
        _resource.shutdown();
      }
      catch (SoapFault fault)
      {
        LoggingUtils.logError(_resource.getLog(), fault);
      }
    }
  }
}
```

2.2.3 Ο deployment descriptor του Muse

Ο deployment descriptor του Muse είναι το αρχείο *muse.xml.* Το περιεχόμενό του είναι το ίδιο για όλες τις πλατφόρμες. Περιέχει πληροφορίες για το WS-resource που υλοποιούμε και χρησιμοποιείται για να αρχικοποιήσουμε τις δομές δεδομένων που χρειάζονται για να επικυρώσουμε και να κατευθύνουμε αιτήσεις στα σωστά capabilities των αντικειμένων. Με άλλα λόγια είναι το αρχείο που επιτρέπει στο Muse να ενώνει όλες τις ξεχωριστές κλάσεις των capabilities σε έναν τύπο resource.

Το XML schema για το muse.xml περιέχεται σε όλα τα projects που δημιουργούνται από το WSDL2Java έτσι ώστε να μπορούμε να το ενημερώνουμε κάθε φορά που κάνουμε αλλαγές στον descriptor.

Το root στοιχείο του Muse deployment descriptor είναι <muse/>. Περιέχει ένα sequence από τρία παιδιά στοιχεία:

- **<router/>** - τα δεδομένα για τη διαμόρφωση του router της κεντρικής μηχανής, η οποία αντιστοιχεί SOAP αιτήσεις σε πραγματικές μεθόδους Java κλήσεων.
- **<resource-type/>** - τα δεδομένα για τη διαμόρφωση των resources που εκτίθενται ως web services. Εδώ ενώνονται όλες οι πληροφορίες από τα διάφορα deployment artifacts (όπως WSDL και XSD) και Java κώδικας για να παρέχει πληροφορίες που χρειάζονται για να παραχθούν instances για το συγκεκριμένο τύπο resource.
- **<custom-serializer/>** - προαιρετικό χαρακτηριστικό που επιτρέπει την επέκταση του συστήματος XML serialization για το χειρισμό των complex types.

3 Business Process Execution Language - BPEL

3.1 Τι είναι η BPEL

Η *Business Process Execution Language for Web Services* (BPEL4WS) είναι μία γλώσσα βασισμένη σε XML για την τυποποίηση των επιχειρησιακών διαδικασιών σε ένα διανεμημένο ή υπολογιστικό περιβάλλον πλέγματος που επιτρέπει στις διάφορες επιχειρήσεις να διασυνδέσουν τις εφαρμογές τους καθώς και τα δεδομένα τους. Σχεδιασμένη ως συνδυασμός της Web Services Flow Language της IBM και της προδιαγραφής XLANG της Microsoft, η ανεξάρτητη πλατφόρμας BPEL επιτρέπει στις επιχειρήσεις να κρατήσουν τα εσωτερικά επιχειρησιακά πρωτόκολλα χωριστά από τα πρωτόκολλα που χρησιμοποιούνται για επικοινωνία με άλλες επιχειρήσεις, έτσι ώστε να μπορούν να αλλάξουν τις εσωτερικές διαδικασίες χωρίς τις επιπτώσεις της ανταλλαγής των στοιχείων από επιχείρηση σε επιχείρηση. Ένα έγγραφο BPEL, παραδείγματος χάριν, παρακολουθεί όλες τις επιχειρησιακές διαδικασίες που συνδέονται με μια συναλλαγή και εξασφαλίζει ότι οι διαδικασίες εκτελούνται με σωστή σειρά μέσω της αυτοματοποίησης των μηνυμάτων.

Δηλαδή η BPEL καθορίζει ένα πρότυπο και μια γραμματική για την περιγραφή της συμπεριφοράς μιας επιχειρησιακής διαδικασίας βασισμένης στις αλληλεπιδράσεις μεταξύ της διαδικασίας και των συνεργατών της (partners). Η αλληλεπίδραση με κάθε συνεργάτη εμφανίζεται μέσω των διεπαφών των Web services και η δομή της σχέσης στο επίπεδο διεπαφών περιγράφεται σε αυτό που καλείται partnerLink. Η διαδικασία BPEL καθορίζει πώς οι πολλαπλές αλληλεπιδράσεις υπηρεσιών με τους partners συντονίζονται για να επιτύχουν έναν επιχειρησιακό στόχο, καθώς επίσης και την απαραίτητη λογική για αυτόν το συντονισμό. Η BPEL εισάγει επίσης τους σημασιολογικούς μηχανισμούς για τα business exceptions και τα processing faults. Επιπλέον, η BPEL εισάγει ένα μηχανισμό για να καθορίσει πώς οι μεμονωμένες ή σύνθετες δραστηριότητες (activities) μέσα σε μια μονάδα εργασίας πρόκειται να αντισταθμιστούν σε περιπτώσεις όπου εμφανίζονται εξαιρέσεις ή αλλαγές σχεδίων των συνεργατών.

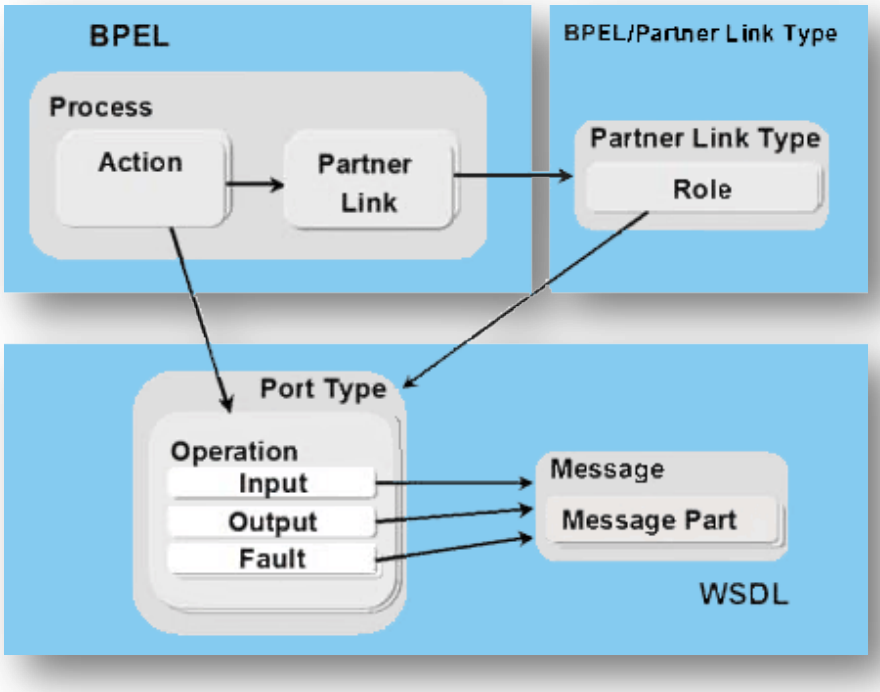
Η BPEL στηρίζεται στην προσέγγιση της SOA αρχιτεκτονικής. Οι προηγούμενες αρχιτεκτονικές προσεγγίσεις όπως CORBA, DCOM, και J2EE JCA υπέφεραν από τα προβλήματα διαλειτουργικότητας, τις ανόμοιες φιλοσοφίες πλαισίου (παραδείγματος χάριν, που ενσωματώνουν DCOM με CORBA), και μια βασική έλλειψη σε προγραμματιστικό επίπεδο υποστήριξης της περιγραφής των επιχειρησιακών διαδικασιών. Η BPEL είναι μια γλώσσα που περιγράφει τις επιχειρησιακές διαδικασίες και χτίζεται επάνω στα ανοικτά πρότυπα WSDL (για την

περιγραφή των APIs των Web services), του SOAP (για την μορφή μηνυμάτων), και του σχήματος XML (για την περιγραφή του περιεχομένου μηνυμάτων).

Η BPEL βασίζεται στην τεχνολογία των Web services και έτσι συνδέεται με standards όπως WSDL, XML, SOAP και UDDI. Το παρακάτω σχήμα παρουσιάζει τη σχέση μεταξύ της BPEL και των προαναφερθέντων standards:

Σχήμα 22: Σχέση της BPEL με τα standards των Web services

Ενώ παρακάτω απεικονίζεται η σχέση μεταξύ της BPEL ροής και της WSDL:



Σχήμα 23: Σχέση BPEL και WSDL

Η BPEL είναι ελεύθερο λογισμικό, διαλειτουργικό, και φορητό. Επιτρέπει τη σχεδίαση των αλληλεπιδράσεων επιχειρησιακών διαδικασιών που προσανατολίζονται σε sequences client-server και peer-to-peer stateful ανταλλαγές μηνυμάτων.

Η BPEL είναι μια "orchestration γλώσσα" και όχι μια "choreography γλώσσα". Η αρχική διαφορά μεταξύ του orchestration και του choreography είναι το πεδίο - scope. Ένα πρότυπο orchestration παρέχει ένα πεδίο που εστιάζει συγκεκριμένα στην άποψη ενός συμμετέχοντος (π.χ. ένα peer to peer πρότυπο). Αντ' αυτού, ένα πρότυπο choreography καλύπτει όλα τα συμβαλλόμενα μέρη και τις σχετικές αλληλεπιδράσεις τους, που δίνουν μια σφαιρική άποψη του συστήματος. Οι διακρίσεις του orchestration και του choreography είναι βασισμένες στις αναλογίες: το orchestration περιγράφει τον κεντρικό έλεγχο της συμπεριφοράς σε μια ορχήστρα, ενώ η choreography είναι για το διανεμημένο έλεγχο της συμπεριφοράς, όπου οι μεμονωμένοι συμμετέχοντες εκτελούν την επεξεργασία βασισμένη σε εξωτερικά γεγονότα, όπως σε μία χορογραφία όπου οι χορευτές αντιδρούν στις συμπεριφορές των υπόλοιπων χορευτών.

3.2 Είδη εφαρμογών SOA που είναι κατάλληλα για την BPEL

Η BPEL έχει ως σκοπό να διαχειριστεί τις ροές εργασίας στα endpoints των Web services. Τα endpoints των Web services μπορούν να ποικίλουν ιδιαίτερα όσον αφορά την granularity, την συναλλακτικότητα και τη διαθεσιμότητά τους. Παραδείγματος χάριν, ένα endpoint μπορεί να παρέχει πρόσβαση σε μια βάση δεδομένων, ένα άλλο μπορεί να παρέχει πρόσβαση σε μια λειτουργία SAP, ένα άλλο πρόσβαση σε μια σειρά CICS συναλλαγών ή ακόμα να αντιπροσωπεύει μια διεπαφή σε μια ολόκληρη επιχειρησιακή διαδικασία μέσα σε μια οργάνωση πελατών ή συνεργατών. Ανεξάρτητα από την πιθανή ποικιλομορφία τους, τα διάφορα endpoints των Web services έχουν ένα κοινό και ουσιαστικό χαρακτηριστικό, παρουσιάζουν μια εξωτερική διεπαφή που περιγράφεται από τη γλώσσα WSDL. Οποιοδήποτε endpoint των Web services έχει ένα έγκυρο WSDL μπορεί να ενσωματωθεί σε μια διαδικασία BPEL.

Σε ένα επιχειρησιακό πλαίσιο, επομένως, μια υποθετική BPEL διαδικασία αυτοματοποιεί τη διαχείριση εντολής αγοράς:

- Λαμβάνει και επικυρώνει μία ηλεκτρονική (βασισμένη σε XML) εντολή αγοράς.
- Ελέγχει μια SAP Accounts Receivable λειτουργία για να εκτελέσει τον έλεγχο πιστωτικού υπολοίπου των πελατών.
- Ελέγχει μια βάση δεδομένων Oracle για να ενημερωθεί για τα στοιχεία της απογραφής.
- Ελέγχει μια CICS εισόδων διαταγής συναλλαγή.
- Αποστέλλει μια εντολή επιβεβαίωσης/απόρριψης αγοράς.

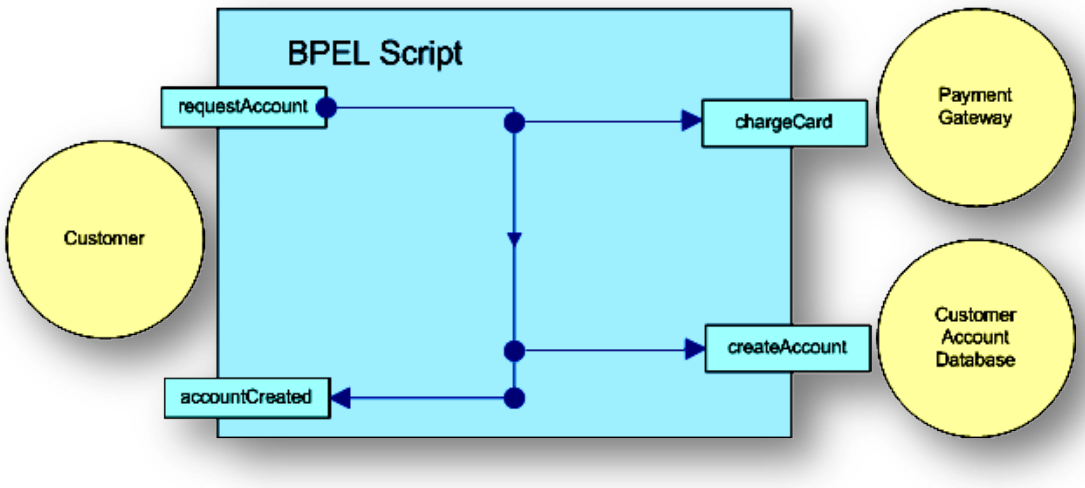
Στο υποθετικό ανωτέρω σενάριο, η λειτουργία SAP, η ερώτηση στην Oracle, και η CICS συναλλαγή, αντιπροσωπεύουν τα endpoints των Web services με τις έγκυρες διασυνδέσεις του WSDL. Η BPEL, γενικά, έχει ως σκοπό να υποστηρίξει συνθέσεις διαδικασιών, όπως αυτήν που περιγράφεται παραπάνω.

3.3 Σκοπός της BPEL

Η BPEL παρέχει μια γλώσσα και ένα περιβάλλον εκτέλεσης για τη δημιουργία μακροπρόθεσμων επιχειρησιακών διαδικασιών από υπάρχουσες Web services. Μπορεί να χειριστεί σύγχρονες καθώς και ασύγχρονες Web services. Υποστηρίζει χειρισμό faults, χειρισμό γεγονότων, και transaction-style compensation, επιτρέποντας την ανάπτυξη ισχυρών επιχειρησιακών διαδικασιών.

Η BPEL ταιριάζει ιδανικά στις καταστάσεις όπου τα συστήματα εκθέτουν ήδη τις επιχειρησιακές λειτουργίες τους μέσω των WSDL αρχείων των Web services. Παραδείγματος χάριν, μια βάση δεδομένων πελάτης-λογαριασμού και μια πύλη πληρωμής όπου και οι δύο εκθέτουν τα συστήματά τους μέσω των WSDL αρχείων, μπορούν γρήγορα και εύκολα να συντεθούν σε μια υπηρεσία λογαριασμού-πληρωμής. Αυτός είναι ο σκοπός για την BPEL: δημιουργία νέων επιχειρησιακών υπηρεσιών από τις υπάρχουσες επιχειρησιακές λειτουργίες.

Παρακάτω φαίνεται ο σχηματισμός μιας νέας υπηρεσίας από ήδη υπάρχουσες:



Σχήμα 24: Σχηματισμός μιας νέας υπηρεσίας από ήδη υπάρχουσες

3.4 Βασικές έννοιες της BPEL

Οι βασικές έννοιες της BPEL συνοψίζονται στους όρους Partner Link, Variables, Activities, Fault Handlers και Event handlers.

3.4.1 Partner Links

Τα Partner Links είναι συνήθως το αρχικό σημείο από το οποίο ξεκινάει το γράψιμο των BPEL scripts. Ένα `partnerLink` στοιχείο αντιπροσωπεύει την επικοινωνία μεταξύ δύο συνεργατών (partners) μίας ροής εργασίας (business process). Τα Partner Links ορίζονται στις WSDL διεπαφές και μπορούμε να τους προσδώσουμε δύο στυλ: client-server και peer-to-peer.

3.4.2 Partner Link Types

Τα Partner Links παραπέμπουν στους ορισμούς των partnerLinkType, οι οποίοι καθορίζονται συνήθως σε ένα σχετικό αρχείο WSDL. Είναι το partnerLinkType που παραπέμπει στους portType ορισμούς και τους σχετικούς ρόλους τους. Προσθέτοντας ένα partnerLink σε ένα BPEL script, καθορίζουμε εάν το BPEL script ή ο partner υλοποιεί τους ρόλους που καθορίζονται στον ορισμό του partnerLinkType. Παραδείγματος χάριν, το παρακάτω είναι ένας client-server-style καθορισμός του partnerLinkType για μια πληρωμή που λαμβάνεται από έναν ορισμό WSDL:

```
<plnk:partnerLinkType name="PaymentGatewayLinkType">
    <plnk:role name="serviceProvider">
        <plnk:portType name="tns:PaymentGateway"/>
    </plnk:role>
</plnk:partnerLinkType>
```

Από το παραπάνω ορίζουμε ότι το partnerLinkType ονομάζεται PaymentGatewayLinkType και ότι υποστηρίζει ένα portType που ονομάζεται tns:PaymentGateway και το οποίο είναι συσχετισμένο με τον ρόλο serviceProvider. Αν αυτό το partnerLinkType είχε δύο ορισμούς θα ήταν ένα peer-to-peer-style partnerLinkType.

Το ακόλουθο XML τμήμα κώδικα μπορεί να χρησιμοποιηθεί για την πρόσθεση ενός partnerLink σε ένα BPEL script δηλώνοντας ότι το BPEL script δρα ως server σε μία client-server σχέση.

```
<bpws:partnerLinks>
    <bpws:partnerLink myRole="serviceProvider" name="paymentGateway" partnerLinkT
    </bpws:partnerLink>
</bpws:partnerLinks>
```

Αυτό προσθέτει ένα partnerLink που ονομάζεται paymentGateway στο BPEL script. Το myRole attribute υποδεικνύει ότι το BPEL script είναι αυτό που παρέχει τη διασύνδεση. Σε αντίθεση, για να υποδείξουμε ότι ο partner είναι αυτός που θα υλοποιήσει την διασύνδεση, το attribute partnerRole πρέπει να χρησιμοποιηθεί ως εξής:

```
<bpws:partnerLinks>
    <bpws:partnerLink partnerRole="serviceProvider" name="paymentGateway" partnerLinkT
    </bpws:partnerLink>
</bpws:partnerLinks>
```

Ενώ αν χρειαστεί να αναφερθούμε σε ορισμούς peer-to-peer partnerLinkType, τότε θα πρέπει να ορίσουμε και το partnerRole και το myRole attribute.

3.4.3 Variables

Οι μεταβλητές – variables ορίζουν containers για τα δεδομένα που θα χρησιμοποιηθούν στη ροή εργασίας. Κάθε μεταβλητή παρέχει έναν ορισμό ως WSDL μήνυμα, ένα σχήμα ή έναν απλό τύπο. Στην BPEL ορίζουμε μεταβλητές τύπου μηνυμάτων χρησιμοποιώντας το attribute messageType. Ένα παράδειγμα είναι το ακόλουθο:

```
<bpws:variables>
    <bpws:variable messageType="ns:creditInformationMessage" name="creditInformationMessage">
    <bpws:variable messageType="ns:approvalMessage" name="approvalMessage">
</bpws:variables>
```

Ορίζουμε μεταβλητές χρησιμοποιώντας το attribute element, όπως φαίνεται παρακάτω:

```
<bpws:variable element="ns2:Records" name="records">
```

Επίσης ορίζουμε μεταβλητές απλών τύπων χρησιμοποιώντας το attribute type, όπως φαίνεται παρακάτω:

```
<bpws:variable type="xsd:string" name="level">
```

Οι μεταβλητές επιτρέπουν στις διαδικασίες να διατηρήσουν τα δεδομένα και την ιστορία της διαδικασίας, βασιζόμενες στα μηνύματα που ανταλλάσσονται. Πριν εφαρμοστεί μία διαδικασία χρησιμοποιώντας μια ακολουθία δραστηριοτήτων, πρέπει να δημιουργηθούν οι μεταβλητές για να κρατήσουν τα μηνύματα αιτήματος και απάντησης.

Επειδή τα BPEL script μπορούν μόνο να στείλουν και να λάβουν τα μηνύματα, οι μεταβλητές που δηλώνονται σε σχέση με τους τύπους σχημάτων XML παραμένουν τοπικές στο script και χρησιμοποιούνται ως μετρητές στους βρόχους ή για να κρατάνε μεμονωμένες παραμέτρους σε κάποια λειτουργία. Για να χρησιμοποιήσετε μια μεταβλητή που δηλώνεται κατ' αυτόν τον τρόπο σε μια λειτουργία, πρέπει να την αντιγράψετε σε ένα μήνυμα.

Τέλος, μπορούμε να τροποποιήσουμε μεταβλητές χρησιμοποιώντας την activity assign της BPEL, η οποία είναι ένα container για μια activity copy με στοιχεία παιδιά from και to,όπως φαίνεται παρακάτω:

```
<assign>
     <copy>
         <from variable="risk" part="level">
        </from>
         <to variable="approval" part="accept">
     </copy>
 </assign>
```

3.5 Activities

Η BPEL γλώσσα περιλαμβάνει XML προγραμματιστικές δηλώσεις που επιτρέπουν την εκτέλεση εύρωστων διαδικασιών. Αυτές οι δηλώσεις ονομάζονται activities και μπορούν να κατηγοριοποιηθούν ως εξής:

- **Communication activities**, όπως receive, reply and invoke, οι οποίες επιτρέπουν στους πελάτες και στις Web services να επικοινωνούν με ανταλλαγή μηνυμάτων. Activities που σχετίζονται με γεγονότα όπως pick, onMessage, και onAlarm περιλαμβάνονται επίσης σε αυτήν την κατηγορία.
- **Control activities**, όπως while, switch, sequence, flow και wait, οι οποίες ελέγχουν πώς εκτελείται η ροή της διαδικασίας.
- **Fault activities**, όπως throw, terminate, catch, catchall και compensate, οι οποίες ορίζουν πώς να αντιμετωπιστούν τα λάθη που συμβαίνουν κατά τη διάρκεια της διαδικασίας.

Οι βασικές activities είναι οι εξής:

- <invoke> : κλήση μιας λειτουργίας
- <receive> : αναμονή για λήψη μηνύματος
- <reply> : αποστολή απάντησης σε μία λειτουργία request-reply
- <wait> : αναμονή
- <assign> : αντιγραφή δεδομένων από ένα σημείο σε ένα άλλο
- <throw> : δήλωση λάθους
- <terminate> : τερματισμός της διαδικασίας
- <empty> : κενή εντολή

Για να καθορίσουμε πώς εκτελείται μία ροή εργασίας, τοποθετούμε όλες τις activities μέσα σε μία activity sequence, η οποία εκτελεί όλες τις activities σειριακά. Οι πληροφορίες ανταλλάσσονται μεταξύ των διαφόρων activities μέσω των global μεταβλητών.

Κάθε activity έχει προαιρετικά τα ακόλουθα attributes:

- **name:** Ένα όνομα τύπου ncname
- **joinCondition:** Μία έκφραση boolean που χρησιμοποιείται για να ορίσει τις απαιτήσεις σχετικά με παράλληλες ροές εργασιών σε ένα flow.
- **suppressJoinFailure:** Υποδεικνύει αν ένα joint failure πρέπει να κατασταλεί.

Κάθε activity έχει επίσης προαιρετικά στοιχεία source και target, τα οποία είναι φωλιασμένα σε μία activity. Αυτά τα στοιχεία χρησιμοποιούνται για να συγχρονίσουν σχέσεις μεταξύ links. Κάθε link ορίζεται ανεξάρτητα και παίρνει ένα όνομα. Αναφορά του ονόματος του link γίνεται μέσω του linkName attribute του source element.

Μια activity μπορεί να οριστεί ως source ενός ή περισσότερων συνδέσεων με τον ορισμό ενός ή περισσότερων στοιχείων source. Κάθε στοιχείο source πρέπει να παραπέμπει σε ένα διαφορετικό όνομα link. Επιπλέον, κάθε στοιχείο source μπορεί προαιρετικά να καθορίζει μια ιδιότητα transitionCondition, η οποία διευκρινίζει έναν όρο για την επιλογή ενός link.

Ομοίως, μια δραστηριότητα μπορεί να οριστεί ως target ενός ή περισσότερων links ορίζοντας ενός όνομα link ευδιάκριτο από όλα τα άλλα στοιχεία source μέσα στην activity. Κάθε στοιχείο target που φωλιάζει μέσα σε μια δραστηριότητα πρέπει να παραπέμπει σε διαφορετικό όνομα συνδέσεων.

3.6 Fault Handlers

Όταν συμβεί ένα λάθος στην BPEL διαδικασία, τότε «ρίχνεται» ένα fault. Στην BPEL έχουμε δύο είδη faults:

- **Process-specific** faults, τα οποία εμφανίζονται όταν μία throw activity εκτελείται ή όταν μία invoke activity επιστρέφει ένα fault ως απάντηση. Τα μηνύματα των faults μπορούν να καθοριστούν στο WSDL για τη διαδικασία BPEL ή για την υπηρεσία των partners. Στο WSDL, καθορίζεται ένα μήνυμα faults ως τμήμα του API και έπειτα παραπέμπει το μήνυμα αυτό κατά τη συσχέτιση ενός fault με μια λειτουργία.
- **Runtime-specific** faults, τα οποία δεν είναι καθορισμένα από το χρήστη και δεν εμφανίζονται στο WSDL για μια διαδικασία BPEL ή μια υπηρεσία. Η προδιαγραφή BPEL καθορίζει διάφορα τυποποιημένα faults, παραδείγματος χάριν, selectionFailure, το οποίο μπορεί να εμφανιστεί όταν εφαρμόζεται μια έκφραση XPath σε ένα εισερχόμενο έγγραφο.

Σε μια διαδικασία BPEL, τα faults που πιάνονται χρησιμοποιώντας τα faultHandlers, είναι αυτά που περιλαμβάνουν τις δραστηριότητες catch και catchAll. Η catch activity χειρίζεται ένα συγκεκριμένο fault, ενώ η catchAll activity χειρίζεται

οποιοδήποτε fault που δεν αντιμετωπίζεται από τη catch activity. Παραπομπή στο fault γίνεται μέσα από την catch activity χρησιμοποιώντας τις ιδιότητες faultName και faultMessageType. Ο ορισμός ενός faultVariable που παρέχει συμπληρωματικές πληροφορίες για το fault είναι προαιρετικός.

Εάν κανένα δεδομένο δε συνδέεται με το fault, δεν υπάρχει καμία ανάγκη να δηλωθεί το fault μέσα στο WSDL των υπηρεσιών. Η BPEL παρέχει έναν ελαφρύ μηχανισμό, που επιτρέπει την ονομασία των faults στην throw activity και χρησιμοποιεί το ίδιο όνομα στην catch activity, παραδείγματος χάριν:

```
<bpws:throw faultName="tns:TimedOut">
```

Τα faults μπορούν να «ξανά-ριχτούν» χρησιμοποιώντας τη throw activity, αλλά για να σταλεί ένα fault πίσω στον πελάτη πρέπει να οριστεί μία reply activity που αναφέρεται στο fault, όπως παρακάτω:

```
<faultHandlers>
    <catch faultVariable="error"
            <faultMessageType="ns:errorMessage">
            <reply partnerLink="customer" portType="ns:loanServicePT"
                operation="request" variable="error"
                faultName="ns:unableToHandleRequest">
            </reply>
    </catch>
</faultHandlers>
```

3.7 Event Handlers

Μόλις αρχίσει ένα instance μίας BPEL διαδικασίας την εκτέλεση, επεξεργάζεται τα μηνύματα χρησιμοποιώντας μία receive activity μόνο εάν η διαδικασία έχει φθάσει σε εκείνο το σημείο στην εκτέλεσή της. Οι event handlers παρέχουν έναν τρόπο να αποκρίνονται στα εξωτερικά και εσωτερικά γεγονότα, ενώ η BPEL διαδικασία εκτελείται, επιτρέποντας την ανάπτυξη πιο εύκαμπτων διαδικασιών. Η BPEL υποστηρίζει δύο τύπους γεγονότων:

Γεγονότα μηνυμάτων, τα οποία προκαλούνται εξωτερικά από έναν πελάτη που στέλνει ένα μήνυμα, παραδείγματος χάριν, ένα μήνυμα ακύρωσης ή μια λειτουργία έρευνας. Καθορίζουμε τα γεγονότα μηνυμάτων χρησιμοποιώντας την activity onMessage, ορίζοντας το partnerLink, το portType και τα attributes των λειτουργιών, παραδείγματος χάριν:

```
<eventHandlers>
   <onMessage partnerLink="buyer" portType="ticketsales operation="cancel" variable="cancelDetails">
      <terminate/>
   </onMessage>
</eventHandlers
```

Συναγερμοί (alarms), οι οποίοι είναι γεγονότα χρονομέτρων που παράγονται από το ίδιο το BPEL script. Καθορίζουμε τους συναγερμούς χρησιμοποιώντας την activity onAlarm, και τα attributes for και until. Το for attribute διευκρινίζει τη διάρκεια από τη στιγμή που συνέβη ο συναγερμός και μετά. Το ρολόι για τη διάρκεια αρχικοποιείται στο χρονικό σημείο που αρχίζει και το σχετικό πεδίο. Το until attribute καθορίζει το συγκεκριμένο χρονικό σημείο που πυροδοτείτε ο συναγερμός. Παρακάτω είναι ένα παράδειγμα συναγερμού χρησιμοποιώντας τη for ιδιότητα:

```
<eventHandlers>
    <onAlarm for= "bpws:getVariableData(transactionDetails,processDuration)">
    </onAlarm>
</eventHandlers>
```

3.8 Messages και Correlation

Μόλις δημιουργηθεί ένα instance της διαδικασίας, ένα εισερχόμενο μήνυμα περνά μόνο υπό τον όρο ότι μπορεί να συσχετιστεί με εκείνο το instance της διαδικασίας. Στην πράξη, αυτό σημαίνει ότι είτε:

- Το script περιέχει μία receive activity, που παραπέμπει σε ένα correlation set, και το σύνολο αυτό ταιριάζει με το εισερχόμενο μήνυμα.
- Το script διευκρινίζει έναν onMessage event handler, που παραπέμπει σε ένα correlation set, και το σύνολο αυτό ταιριάζει με το εισερχόμενο μήνυμα.

Όταν ένα μήνυμα φθάνει, η BPEL μηχανή ψάχνει για μία receive ή onMessage activity που αντιστοιχεί στον τύπο μηνυμάτων. Εάν η activity ορίζει ένα correlation set, η μηχανή BPEL ελέγχει για να δει εάν ταιριάζει με το εισερχόμενο μήνυμα. Σε αυτή την περίπτωση, αναθέτει το μήνυμα στο instance της διαδικασίας ως εκκρεμές μήνυμα. Στην περίπτωση ενός event handler, το instance της διαδικασίας BPEL αρχίζει να επεξεργάζεται το μήνυμα αμέσως. Στην περίπτωση της receive activity, το instance της διαδικασίας επεξεργάζεται το μήνυμα όταν η activity είναι έτοιμη να εκτελεστεί. Αυτό επιτρέπει στην BPEL διαδικασία να χειριστεί τα μηνύματα που φθάνουν νωρίς ή έξω από την ακολουθία.

3.9 Η Pick Activity

Η BPEL παρέχει ένα πρόσθετο κατασκεύασμα που λειτουργεί με τους event handlers, την pick δραστηριότητα. Όταν εκτελείται, η pick activity εμποδίζει το νήμα εκτέλεσης ενός instance μιας διαδικασίας έως ότου συμβεί ένα γεγονός μηνυμάτων ή ένα γεγονός συναγερμών. Αυτά τα γεγονότα ορίζονται ως παιδιά της pick activity. Η pick activity εκτελεί μόνο ένα από τα παιδιά της σε first-come-first-served βάση.

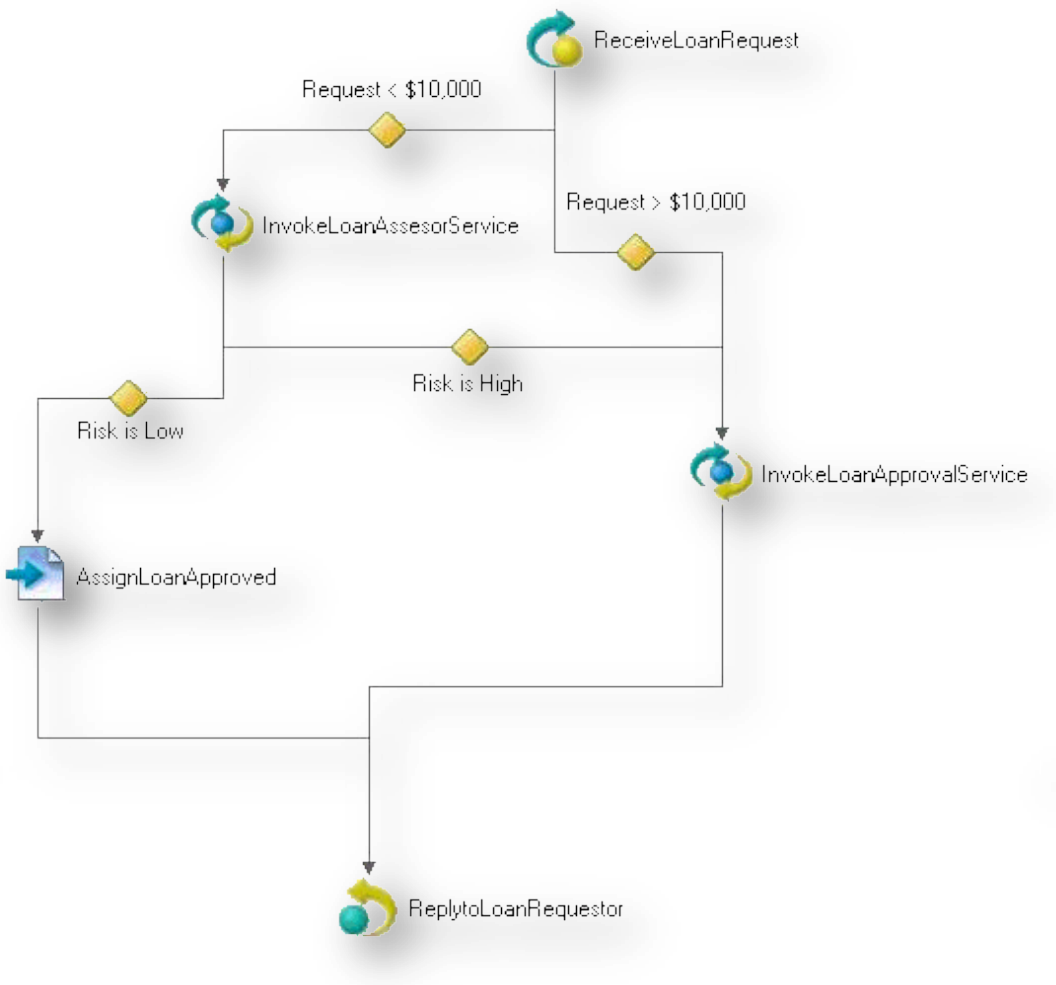
Μια χαρακτηριστική χρήση της pick activity είναι όταν η pick έχει δύο παιδιά event handler: ένα στοιχείο onMessage και ένα στοιχείο onAlarm. Σε αυτήν την περίπτωση, το instance της BPEL διαδικασίας εμποδίζεται έως ότου είτε φτάσει ένα μήνυμα ή ένα χρονικό όριο λήξει.

3.10 Εκκίνηση της ροής εργασίας

Για να ξεκινήσει μία ροή εργασίας πρέπει να σταλεί σε αυτήν ένα μήνυμα. Για να μπορέσει όμως να δεχτεί το μήνυμα πρέπει να περιέχει μία activity receive, της οποίας η ιδιότητα *createInstance* θα έχει την τιμή true.

3.11 Παράδειγμα BPEL διαδικασίας

Παρακάτω θα παρουσιάσουμε ένα παράδειγμα μίας BPEL διαδικασίας. Το παράδειγμα αυτό είναι μία αίτηση για λήψη δανείου. Στο παρακάτω σχήμα φαίνεται η ροή εργασίας που ακολουθείται για την λήψη της έγκρισης του δανείου:



Σχήμα 25: Παράδειγμα ροής εργασίας

Η λογική της έγκρισης περιγράφεται με τον παρακάτω ψευδοκώδικα:

```
String processLoanRequest(message) {
    if (message.amount < 10000 && "low".equals(assessor.risk(message)))
        return "approved";
    else
        return approver.approve(message);
}
```

Όπου η assessor.risk και η approver.approve μέθοδοι αποτελούν κλήσεις εξωτερικών Web services.

3.11.1 Δημιουργία του αρχείου BPEL

Αντί να δημιουργήσουμε ένα αρχείο BPEL από την κορυφή προς τα κάτω, είναι προτιμότερο να το χτίσουμε από το εσωτερικό προς τα έξω, αρχίζοντας από τη διαδικασία και τις activities που είναι οι δομικές μονάδες της διαδικασίας. Έπειτα, θα συνδέσουμε τις δραστηριότητες με τις συνδέσεις. Τέλος, θα συμπληρώσουμε όλες τις λεπτομέρειες που απαιτούνται για τη διαδικασία BPEL για να επικοινωνήσουν με τον έξω κόσμο.

Θα αρχίσουμε με τη διαδικασία. Αυτό συμβαίνει να είναι η ρίζα του εγγράφου XML, έτσι κάνει ένα καλό εξωτερικό shell για το αρχείο BPEL. Μια διαδικασία μπορεί μόνο να περιέχει μια top-level activity. Θα προσθέσουμε μια δραστηριότητα ροής. Δεδομένου ότι οι περισσότερες ροές εργασίας αποτελούνται από περισσότερες από μια δραστηριότητες, η κορυφαία δραστηριότητα είναι συνήθως ένα container όπως μια ροή που μπορεί να περιέχει τις πολλαπλές δραστηριότητες.

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="loanApprovalProcess"
 suppressJoinFailure="yes"
 targetNamespace="http://acme.com/loanprocessing"
 xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
 xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
 xmlns:lns="http://loans.org/wsdl/loan-approval"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <flow>
  </flow>
</process>
```

Οι ιδιότητες ονόματος μπορούν να είναι ο,τιδήποτε. Ορίζοντας suppressJoinFailure= " yes " επιλέγουμε οι κοινές αποτυχίες να μην ρίχνουν faults.

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="loanApprovalProcess"
 suppressJoinFailure="yes"
 targetNamespace="http://acme.com/loanprocessing"
 xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
 xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
 xmlns:lns="http://loans.org/wsdl/loan-approval"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <flow>
    <receive createInstance="yes" operation="request"
     partnerLink="customer" portType="lns:loanServicePT"
     variable="request">
    </receive>
    <reply operation="request" partnerLink="customer"
     portType="lns:loanServicePT" variable="approval">
    </reply>
  </flow>
</process>
```

Θέλουμε να προκαλέσουμε την διαδικασία έγκρισης δανείου όταν λαμβάνουμε ένα μήνυμα υπηρεσιών δανείου. Γι' αυτό το createInstance τίθεται σε "yes". Αυτό σημαίνει ότι όταν βλέπει η μηχανή ActiveBPEL ένα εισερχόμενο μήνυμα που ταιριάζει με τα port type και την λειτουργία του WSDL, αυτό θα δημιουργήσει ένα νέο instance μίας διαδικασίας και θα δώσει το εισερχόμενο μήνυμα στην receive δραστηριότητα.

Όσον αφορά τις μεταβλητές που χρειαζόμαστε, θα καθορίσουμε μία για το μήνυμα που παραλαμβάνεται από τον πελάτη, μία για το μήνυμα αξιολόγησης του κινδύνου, μία για το μήνυμα απάντησης έγκρισης και μία για να κρατήσει οποιαδήποτε μηνύματα λάθους.

```
<variables>
  <variable messageType="lns:creditInformationMessage"
      name="request"/>
  <variable messageType="lns:riskAssessmentMessage"
      name="risk"/>
  <variable messageType="lns:approvalMessage"
      name="approval"/>
  <variable messageType="lns:errorMessage"
      name="error"/>
  </variables>
```

Υπάρχουν ακόμα δύο partners στην διαδικασία: ο assessor και ο approver. Η δουλειά του assessor είναι να κάνει μία γρήγορη αποτίμηση του ρίσκου για την χορήγηση του δανείου. Αν η χορήγηση του δανείου κριθεί υψηλού κινδύνου τότε προωθείται στον approver. Επικοινωνούμε με κάθε partner χρησιμοποιώντας την invoke activity.

```
<invoke inputVariable="request" name="InvokeAssessor"
    operation="check" outputVariable="risk"
    partnerLink="assessor" portType="lns:riskAssessmentPT">
</invoke>
<invoke inputVariable="request" name="InvokeApprover"
    operation="approve" outputVariable="approval"
    partnerLink="approver" portType="lns:loanApprovalPT">
</invoke>
```

Στην συνέχεια ορίζουμε τα links:

```
<links>
  <link name="receive-to-assess"/>
  <link name="receive-to-approval"/>
  <link name="assess-to-setMessage"/>
  <link name="assess-to-approval"/>
  <link name="setMessage-to-reply"/>
  <link name="approval-to-reply"/>
</links>
```

Το επόμενο βήμα είναι να δώσουμε τιμές στις μεταβλητές. Παραδείγματος χάριν, αν ο assessor χαρακτηρίσει την χορήγηση δανείου ως χαμηλού ρίσκου, τότε θέτουμε την τιμή "yes" στην μεταβλητή "approval".

```
<assign>
  <target linkName="assess-to-setMessage"/>
  <source linkName="setMessage-to-reply"/>
  <copy>
    <from expression="'yes'"/>
    <to part="accept" variable="approval"/>
  </copy>
</assign>
```

Τα links που προέρχονται από τη receive activity είναι δύο. Η συνθήκη μετάβασης καθορίζει σε ποιο από τα δύο links θα συνεχιστεί η διαδικασία.

```
<receive createInstance="yes" operation="request"
 partnerLink="customer" portType="lns:loanServicePT"
 variable="request">
  <source linkName="receive-to-assess"
   transitionCondition=
       "bpws:getVariableData('request','amount') &lt; 10000"/>
  <source linkName="receive-to-approval"
   transitionCondition=
       "bpws:getVariableData('request','amount') &gt;= 10000"/>
 </receive>
```

Τα <source> elements λένε ότι η receive είναι στο άλλο άκρο του link, πάνε από την μία activity στην άλλη. Κάθε μία από τις άλλες δύο activities χρησιμοποιούν targets για να καθορίσουν ποιο link θα ακολουθήσουν.

Παρακάτω φαίνεται η invoke activity που είναι το target του receive-to-assess link.

```
<invoke inputVariable="request" name="invokeAssessor"
    operation="check" outputVariable="riskAssessment"
    partnerLink="assessor" portType="asns:riskAssessmentPT">
  <target linkName="receive-to-assess"/>
</invoke>
```

4 Grids

4.1 Τι είναι τα Grids

Με τον όρο Grid computing μπορούμε να αναφερθούμε σε κάποια από τις παρακάτω έννοιες:

- Ένα τοπικό cluster υπολογιστών που είναι όπως ένα "πλέγμα- grid" επειδή αποτελείται από τους πολλαπλούς κόμβους.
- Προσφορά online εξυπηρέτησης υπολογιστικού φόρτου ή αποθήκευσης ως μετρημένη εμπορική υπηρεσία, γνωστή ως utility computing, computing on demand, ή cloud computing.
- Δημιουργία ενός «εικονικού υπερυπολογιστή» με τη χρησιμοποίηση των εφεδρικών πόρων υπολογισμού μέσα σε μια οργάνωση.
- Δημιουργία ενός «εικονικού υπερυπολογιστή» με τη χρησιμοποίηση ενός δικτύου υπολογιστών που είναι γεωγραφικά διασκορπισμένοι. Ο «εθελοντικός υπολογισμός», που εστιάζει γενικά σε επιστημονικά, μαθηματικά, και ακαδημαϊκά προβλήματα, είναι η πιο ευρέως διαδεδομένη εφαρμογή αυτής της τεχνολογίας.

Αυτοί οι ποικίλοι ορισμοί καλύπτουν το φάσμα του «διανεμημένου υπολογισμού» και μερικές φορές οι δύο όροι χρησιμοποιούνται ως συνώνυμα.

Διάφοροι ορισμοί που έχουν δοθεί ανά διαστήματα από διάφορους φορείς:

- Μία μορφή παράλληλου και κατανεμημένου συστήματος το οποίο κάνει δυνατή την κοινή χρήση, επιλογή και συνάθροιση γεωγραφικά κατανεμημένων αυτόνομων υπολογιστικών πόρων, δυναμικά κατά τη διάρκεια της εκτέλεσης, ανάλογα με τη διαθεσιμότητα, ικανότητα, απόδοση, το κόστος και την ποιότητα υπηρεσίας που επιθυμούν οι χρήστες.
- Μία υπηρεσία για κοινή χρήση της υπολογιστικής ισχύος και της ικανότητας αποθήκευσης δεδομένων μέσω του διαδικτύου.
- Τα "υπολογιστικά πλέγματα" ανήκουν στην ομάδα των κατανεμημένων συστημάτων, περιλαμβάνοντας πολλούς δικτυακούς κόμβους, ώστε να ενσωματώσουν μία νέα γενιά αποδοτικότερου λογισμικού και παράλληλα να ενισχυθεί η συνεργασία μεταξύ των παρόχων υπολογιστικής ισχύος σε όλα τα πλάτη της γης.

Παρακάτω φαίνεται μία γραφική απεικόνιση του Grid:

Σχήμα 26: Απεικόνιση του Grid

Το Grid λοιπόν παρέχει τις υπηρεσίες δικτύωσης και διασύνδεσης ενός ενδεχομένως απεριόριστου αριθμού διαθέσιμων υπολογιστών μέσα στο "πλέγμα". Το γεγονός ότι τα "υπολογιστικά πλέγματα" εστιάζουν στην παροχή υπολογισμών πέρα από το administrative domain, τα διαχωρίζει από τα παραδοσιακά κατανεμημένα συστήματα και τα clusters. Τα Grids μπορούν να χρησιμοποιηθούν για επίλυση πολύπλοκων προβλημάτων, όπως είναι η προσομοίωση σεισμού, η σχεδίαση κλιματολογικών μοντέλων, η αναδίπλωση πρωτεϊνών, η σχεδίαση τεχνο-οικονομικών μοντέλων κτλ. Επίσης αποτελούν το μέσο για την κατασκευή utility grids δικτύων, τα οποία θα παρέχουν πληροφορία για εμπορική ή μη χρήση οποτεδήποτε το επιθυμεί ο χρήστης δηλαδή on-demand networks, ο οποίος θα χρεώνεται ανάλογα με το μέγεθος και την αξία της πληροφορίας, όπως ακριβώς συμβαίνει με το ηλεκτρικό ρεύμα, ή το νερό.

Οι τεχνολογίες Grid εστιάζουν σε δυναμική κατανομή πόρων μεταξύ των οργανισμών και έτσι συμπληρώνουν παρά ανταγωνίζονται τις υπάρχουσες κατανεμημένες τεχνολογίες υπολογισμού.

Στο παρελθόν υπήρξαν σημαντικές υλοποιήσεις που βασίζονταν στην ιδέα του κατανεμημένου προγραμματισμού και που ενσωματώνονται στην αρχιτεκτονική των Grid. Ορισμένες από αυτές είναι Condor, Codine, Legion, Nimrod και Unicore.

4.2 Εξέλιξη των τεχνολογιών Grid

Οι τεχνολογίες του Grid που χρησιμοποιούνται σήμερα είναι προϊόν 10 χρόνων έρευνας και ανάπτυξης τόσο σε ακαδημαϊκό, όσο και σε επαγγελματικό τομέα. Στην ανάπτυξη αυτήν μπορούμε να διακρίνουμε τέσσερις κύριες φάσεις:

- *Αρχικές λύσεις:* Είναι οι λύσεις που δόθηκαν στις αρχές της δεκαετίας του ’90, όταν ξεκίνησε η έρευνα των συστημάτων αυτών με κύριο σκοπό την εξερεύνηση των δυνατοτήτων τους. Δεδομένου ότι τότε η διαλειτουργικότητα δεν ήταν στο επίκεντρο του ενδιαφέροντος και ότι οι εφαρμογές κατασκευάζονταν πάνω από τα πρωτόκολλα διαδικτύου, είχαν περιορισμένη λειτουργικότητα σε τομείς όπως ασφάλεια, δυνατότητα κλιμάκωσης και αξιοπιστία.

- *Globus Toolkit:* Το Globus Toolkit Version 2 - GT2 καθιερώθηκε ως το πρότυπο για την σχεδίαση συστημάτων και εφαρμογών για το Grid, από το 1997. Επικεντρώνοντας κυρίως στην ευχρηστία και στη διαλειτουργικότητα, το GT2 όρισε και υλοποίησε νέα πρωτόκολλα , APIs και υπηρεσίες. Πλέον η σουίτα πρωτοκόλλων GT2 κάνει χρήση υπαρχόντων διαδικτυακών προτύπων για μεταφορά δεδομένων, ανακάλυψη πόρων και ασφάλεια. Μερικά στοιχεία της σουίτας πρωτοκόλλων GT2 κωδικοποιήθηκαν σε αυστηρές τεχνικές προδιαγραφές και χρησιμοποιήθηκαν σε πολλαπλές υλοποιήσεις. Παρόλα αυτά όμως, οι προτυποποιήσεις που εισήγαγε το GT2 δεν ήταν αρκετά αυστηρές, ούτε υπόκειντο αποκλειστικά σε δημόσια χρήση. Ένα άλλο σύστημα που εμφανίστηκε εκείνη την εποχή είναι το σύστημα υψηλής απόκρισης Condor.

- *Open Grid Services Architecture - OGSA:* Το έτος 2002 εμφανίστηκε η αρχιτεκτονική ανοικτών υπηρεσιών Grid - OGSA. Βασισμένο στο GT2 και σημαντικά επεκταμένο ως προς τις δυνατότητές του, το OGSA υποστηρίζει τη χρήση των υπολογιστικών πλεγμάτων με τις ευρείες βιομηχανικές πρωτοβουλίες, μέσω μίας προσανατολισμένης σε υπηρεσίες αρχιτεκτονική (service oriented architecture), στην οποία οι υπηρεσίες πλέγματος οικοδομούνται πάνω στις Web services. Το OGSA όχι μόνο ορίζει ένα σύνολο standard interfaces και συμπεριφορών αλλά επιπλέον παρέχει ένα πλαίσιο μέσω του οποίου μπορεί κανείς να ορίσει ένα ευρύ φάσμα διαλειτουργικών και μεταφέρσιμων υπηρεσιών.

- *Διαχειριζόμενα, κοινά εικονικά συστήματα:* Παρόλο που ο ορισμός των τεχνικών προδιαγραφών OGSA έλυσε πολλά προβλήματα, υπάρχουν ακόμα πολλά βήματα που πρέπει να γίνουν. Βασισμένο στην OGSA, διακρίνεται ένα σύνολο διαλειτουργικών υπηρεσιών και συστημάτων που αντιμετωπίζουν την αυξανόμενη κλιμάκωση με μεγαλύτερο αριθμό οντοτήτων, αυξανόμενο βαθμό εικονικότητας πόρων, πλουσιότερους τρόπους κοινής χρήσης και αυξανόμενες ποιότητες υπηρεσιών μέσω μίας πληθώρας ενεργών διαχειριστικών τρόπων. Η σημερινή πρόκληση είναι η αντιμετώπιση προβλημάτων στις γνωστικές περιοχές της πληροφορικής, δηλαδή peer-to-peer, knowledge-based, αυτό-διαχειριζόμενα συστήματα.

5 Ad-hoc δίκτυα αισθητήρων

5.1 Δίκτυα αισθητήρων

5.1.1 Τι είναι ένα ασύρματο δίκτυο αισθητήρων

Ένα ασύρματο δίκτυο αισθητήρων (Wireless Sensor Networks) είναι ένα ασύρματο δίκτυο που αποτελείται από χωρικά κατανεμημένες αυτόνομες συσκευές που χρησιμοποιούν αισθητήρες για να ελέγξουν τις φυσικές ή περιβαλλοντικές συνθήκες, όπως η θερμοκρασία, ο ήχος, η δόνηση, η πίεση, η κίνηση ή οι ρύποι, στις διαφορετικές θέσεις. Η ανάπτυξη των ασύρματων δικτύων αισθητήρων παρακινήθηκε αρχικά από τις στρατιωτικές εφαρμογές, όπως η επιτήρηση πεδίων μαχών. Εντούτοις, τα ασύρματα δίκτυα αισθητήρων χρησιμοποιούνται τώρα σε πολλούς πολιτικούς τομείς, συμπεριλαμβανομένου του περιβάλλοντος και της παρακολούθησης βιότοπων, των εφαρμογών υγειονομικής περίθαλψης, της οικιακής αυτοματοποίησης, και του ελέγχου της κυκλοφορίας.

5.1.2 Ad-hoc δίκτυα αισθητήρων

Ένα ασύρματο ad-hoc δίκτυο αισθητήρων αποτελείται από διάφορους αισθητήρες που διαδίδονται πέρα από μια γεωγραφική περιοχή. Κάθε αισθητήρας έχει ικανότητα ασύρματης επικοινωνίας και κάποιο επίπεδο νοημοσύνης για την επεξεργασία σήματος και τη δικτύωση των στοιχείων. Μερικά παραδείγματα των ασύρματων ad-hoc δικτύων αισθητήρων είναι τα ακόλουθα:

- Στρατιωτικά δίκτυα αισθητήρων για την ανίχνευση και την λήψη όσο το δυνατόν περισσοτέρων πληροφοριών για τις εχθρικές κινήσεις, τις εκρήξεις, και άλλα φαινόμενα ενδιαφέροντος.
- Δίκτυα αισθητήρων για την ανίχνευση και των χαρακτηρισμό των χημικών, βιολογικών, ραδιολογικών, πυρηνικών, και εκρηκτικών επιθέσεων καθώς και υλικών.
- Δίκτυα αισθητήρων για την ανίχνευση και τον έλεγχο των περιβαλλοντικών αλλαγών στις πεδιάδες, τα δάση, τους ωκεανούς, κ.λπ.
- Ασύρματα δίκτυα αισθητήρων ρύθμισης κυκλοφορίας για έλεγχο της κυκλοφορίας οχημάτων στις εθνικές οδούς ή στα κορεσμένα μέρη μιας πόλης.
- Ασύρματα δίκτυα αισθητήρων επιτήρησης για παροχή ασφάλειας σε εμπορικά πολυκαταστήματα, χώρους στάθμευσης, και άλλων εγκαταστάσεων.
- Ασύρματα δίκτυα αισθητήρων χώρων στάθμευσης για καθορισμό των σημείων που είναι κατειλημμένα και αυτών που είναι ελεύθερα.

Ο ανωτέρω κατάλογος προτείνει ότι τα ασύρματα ad-hoc δίκτυα αισθητήρων προσφέρουν ορισμένες ικανότητες και αυξάνουν την λειτουργική αποδοτικότητα σε πολλές εφαρμογές καθώς επίσης βοηθούν στην εθνική προσπάθεια να αυξηθεί η επαγρύπνηση στις πιθανές τρομοκρατικές απειλές.

Δύο τρόποι να ταξινομηθούν τα ασύρματα ad-hoc δίκτυα αισθητήρων είναι εάν οι κόμβοι είναι ή όχι χωριστά διευθυνσιοδοτημένοι, και εάν τα στοιχεία στο δίκτυο αθροίζονται. Οι κόμβοι αισθητήρων σε ένα δίκτυο χώρων στάθμευσης πρέπει να είναι χωριστά διευθυνσιοδοτημένοι, έτσι ώστε κάποιος να μπορεί να καθορίσει τις ελεύθερες θέσεις. Αυτή η εφαρμογή δείχνει ότι μπορεί να είναι απαραίτητο η μετάδοση μηνυμάτων σε όλους τους κόμβους του δικτύου. Εάν κάποιος θέλει να καθορίσει τη θερμοκρασία σε μια γωνία ενός δωματίου, τότε η διευθυνσιοδότηση μπορεί να μην είναι τόσο σημαντική. Οποιοσδήποτε κόμβος στη δεδομένη περιοχή μπορεί να αποκριθεί. Η δυνατότητα των δικτύων αισθητήρων να αθροίζουν τα συλλεχθέντα στοιχεία μπορεί να μειώσει πολύ τον αριθμό μηνυμάτων που πρέπει να διαβιβαστούν μέσω του δικτύου.

5.1.3 Στόχοι και απαιτήσεις ενός δικτύου αισθητήρων

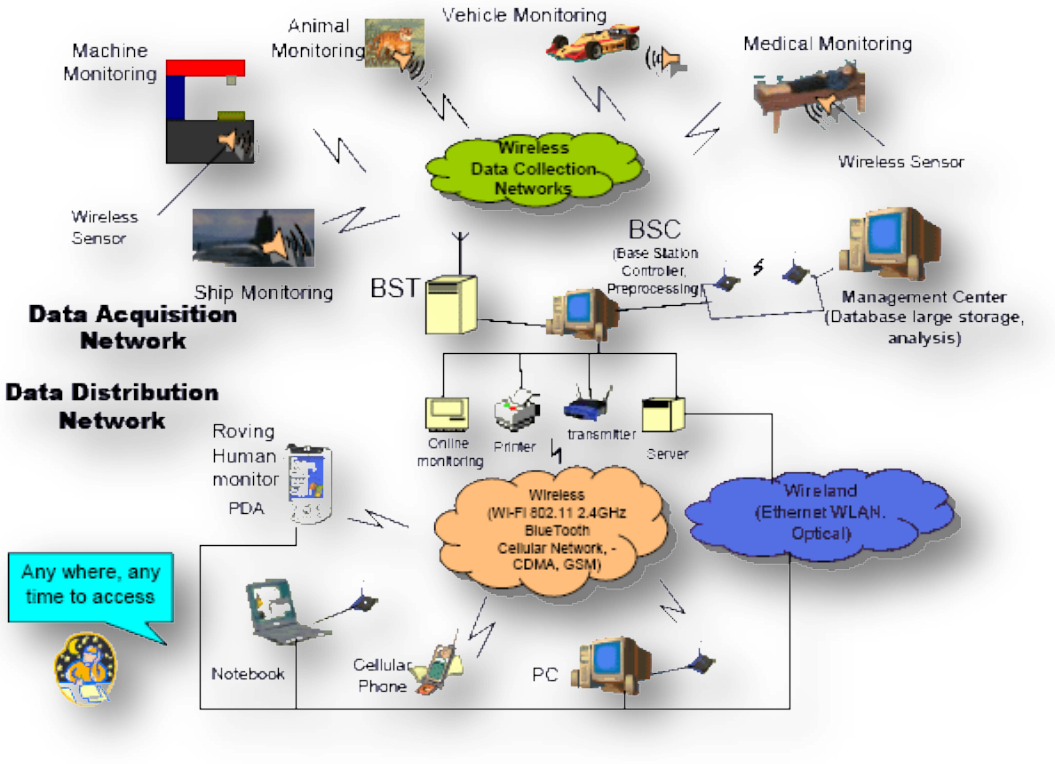
Οι βασικοί στόχοι ενός ασύρματου ad-hoc δικτύου αισθητήρων εξαρτώνται γενικά από την εφαρμογή, αλλά οι ακόλουθοι στόχοι είναι κοινοί για πολλά δίκτυα:

- Καθορισμός της αξίας κάποιας παραμέτρου σε μια δεδομένη θέση: Σε ένα περιβαλλοντικό δίκτυο, κανείς μπορεί να θέλει να ξέρει τη θερμοκρασία, την ατμοσφαιρική πίεση, το ποσό φωτός του ήλιου και τη σχετική υγρασία σε διάφορες θέσεις. Αυτό το παράδειγμα δείχνει ότι ένας δεδομένος κόμβος αισθητήρων μπορεί να συνδεθεί με τους διαφορετικούς τύπους αισθητήρων, κάθε ένας με ένα διαφορετικό ρυθμό δειγματοληψίας των τιμών.
- Ανίχνευση του περιστατικού των γεγονότων ενδιαφέροντος και υπολογισμός των παραμέτρων των ανιχνευμένων γεγονότων: Στο δίκτυο αισθητήρων κυκλοφορίας, κάποιος θα επιθυμούσε να ανιχνεύσει ένα όχημα που κινείται μέσω μιας διασταύρωσης και να υπολογίσει την ταχύτητα και την κατεύθυνση του οχήματος.
- Ταξινόμηση ενός ανιχνευμένου αντικειμένου: Είναι ένα όχημα σε ένα δίκτυο αισθητήρων κυκλοφορίας αυτοκίνητο, μίνι-φορτηγό, ελαφρύ φορτηγό, λεωφορείο, κ.λπ.
- Παρακολούθηση ενός αντικειμένου: Σε ένα στρατιωτικό δίκτυο αισθητήρων, ακολουθήστε ένα εχθρικό όχημα καθώς κινείται μέσω της γεωγραφικής περιοχής που καλύπτεται από το δίκτυο.

Σε αυτούς τους τέσσερις στόχους, μια σημαντική απαίτηση του δικτύου αισθητήρων είναι ότι τα απαραίτητα στοιχεία διαδίδονται στο κατάλληλο κόμβο και στους κατάλληλους χρήστες. Σε μερικές περιπτώσεις, υπάρχουν αρκετά αυστηρές

χρονικές απαιτήσεις σε αυτήν την επικοινωνία. Παραδείγματος χάριν, η ανίχνευση ενός εισβολέα σε ένα δίκτυο παρακολούθησης πρέπει να κοινοποιηθεί αμέσως στην αστυνομία έτσι ώστε να μπορέσουν να ληφθούν μέτρα.

Παρακάτω φαίνεται ένα ασύρματο δίκτυο αισθητήρων:



Σχήμα 27: Δίκτυο αισθητήρων

Οι απαιτήσεις των ασύρματων ad-hoc δικτύων αισθητήρων περιλαμβάνουν τα εξής:

- Μεγάλο αριθμό (συνήθως στάσιμων) αισθητήρων: Εκτός από την επέκταση των αισθητήρων στην ωκεάνια επιφάνεια ή τη χρήση των κινητών, τηλεκατευθυνόμενων, ρομποτικών αισθητήρων σε στρατιωτικές εφαρμογές, οι περισσότεροι κόμβοι σε ένα έξυπνο δίκτυο αισθητήρων είναι στάσιμοι. Δίκτυα 10.000 ή ακόμα και 100.000 κόμβων προβλέπονται, έτσι η εξελιξιμότητα είναι ένα σημαντικό ζήτημα.
- Χαμηλή ενεργειακή χρήση: Δεδομένου ότι σε πολλές εφαρμογές οι κόμβοι αισθητήρων θα τοποθετηθούν σε μια απομακρυσμένη περιοχή, η τροφοδότηση ενός κόμβου μπορεί να μην είναι δυνατή. Σε αυτήν την περίπτωση, η διάρκεια ζωής ενός κόμβου μπορεί να καθοριστεί από τη ζωή μπαταριών, για αυτό τον λόγο απαιτείται η ελαχιστοποίηση των ενεργειακών δαπανών.
- Δυνατότητα αυτό-οργάνωσης των δικτύων: Λαμβάνοντας υπόψη το μεγάλο αριθμό κόμβων και την πιθανή τοποθέτησή τους σε εχθρικές

θέσεις, είναι σημαντικό το δίκτυο να είναι σε θέση να αυτο-οργανώνεται, διότι η χειρωνακτική διαμόρφωση δεν είναι εφικτή. Επιπλέον, μερικοί κόμβοι μπορεί να αποτύχουν (είτε από την έλλειψη ενέργειας είτε από τη φυσική καταστροφή), και οι νέοι κόμβοι πρέπει να μπορούν να ενώσουν το δίκτυο. Επομένως, το δίκτυο πρέπει να είναι σε θέση να μετατρέπεται περιοδικά έτσι ώστε να μπορεί να συνεχίσει να λειτουργεί. Μεμονωμένοι κόμβοι μπορούν να αποσυνδεθούν από το υπόλοιπο δίκτυο, αλλά ένας υψηλός βαθμός συνδετικότητας πρέπει να διατηρηθεί.

- Συνεργατική επεξεργασία σήματος: ακόμα ένα γεγονός που ξεχωρίζει αυτά τα δίκτυα από τα MANETs είναι ότι ο τελικός στόχος είναι η ανίχνευση/εκτίμηση κάποιων γεγονότων ενδιαφέροντος, και όχι απλά μόνο η επικοινωνία. Για να βελτιώσουμε την απόδοση της ανίχνευσης /εκτίμησης, είναι συνήθως αρκετά χρήσιμο να συγχωνεύουμε δεδομένα από πολλούς αισθητήρες. Αυτήν η συγχώνευση δεδομένων απαιτεί την μετάδοση δεδομένων και μηνυμάτων ελέγχου, και έτσι μπορεί να τεθούν περιορισμοί στην αρχιτεκτονική των δικτύων.
- Δυνατότητα άντλησης πληροφοριών: κάποιος χρήστης μπορεί να θέλει τις πληροφορίες ενός μόνο κόμβου ή μίας ομάδας από κόμβους για συλλογή πληροφοριών για συγκεκριμένη περιοχή. Εξαιτίας της συγχώνεψης δεδομένων που πραγματοποιείται μπορεί να μην είναι δυνατόν να μεταδοθεί ένας μεγάλος όγκος πληροφοριών από το δίκτυο. Αντί αυτού, διάφοροι τοπικοί κόμβοι θα συλλέξουν δεδομένα από δεδομένες περιοχές και θα δημιουργήσουν περιληπτικά μηνύματα. Έτσι μία αίτηση για άντληση πληροφοριών από έναν κόμβο μπορεί να κατευθυνθεί στον πλησιέστερο τοπικό κόμβο που συλλέγει δεδομένα.

5.1.4 Αρχιτεκτονικές συστήματος

Με την απερχόμενη διάθεση χαμηλού κόστους, μικρής εμβέλειας ασυρμάτων με προοδευτική ασύρματη δικτύωση αναμένεται ότι τα ασύρματα ad-hoc δίκτυα αισθητήρων θα αναπτυχθούνε. Σε αυτά τα δίκτυα, κάθε κόμβος μπορεί να εξοπλιστεί από μια ποικιλία αισθητήρων όπως ακουστικοί, σεισμικοί, υπέρυθροι ακινησίας /κίνησης videocamera, κτλ. Αυτοί οι κόμβοι μπορεί να οργανωθούν σε clusters έτσι ώστε ένα γεγονός να μπορεί να ανιχνευτεί από τους περισσότερους , αν όχι όλους τους κόμβους ενός cluster. Κάθε κόμβος μπορεί να έχει επαρκή επεξεργαστική ισχύ να παίρνει κάποια απόφαση και να την μεταδίδει στους υπόλοιπους κόμβους του cluster. Ένας κόμβος μπορεί να δρα σαν ο master του cluster, και μπορεί να έχει μεγαλύτερη εμβέλεια ασύρματης εκπομπής χρησιμοποιώντας πρωτόκολλα όπως το IEEE 802.11 ή το Bluetooth.

5.2 Αισθητήρες

5.2.1 Η χρησιμότητα των αισθητήρων

Οι αισθητήρες που απαρτίζουν τα δίκτυα αυτά μπορούν να μετρήσουν:

- Απόσταση, κατεύθυνση, ταχύτητα
- Υγρασία, σύσταση εδάφους
- Θερμοκρασία, χημικά
- Ηλιακή ακτινοβολία, κίνηση, δονήσεις
- Σεισμικά και ακουστικά δεδομένα

Ένας αισθητήρας είναι ένας τύπος μετατροπέα-transducer. Υπάρχουν αισθητήρες που οι μετρήσεις τους είναι άμεσα αντιληπτές και αναγνωρίσιμες από τον άνθρωπο, παραδείγματος χάριν ένα θερμόμετρο υδραργύρου. Άλλοι αισθητήρες πρέπει να ζευγαρωθούν με έναν δείκτη ή μια οθόνη απεικόνισης των ενδείξεων, για παράδειγμα ένα θερμοηλεκτρικό ζεύγος. Οι περισσότεροι αισθητήρες είναι ηλεκτρικοί ή ηλεκτρονικοί, αν και υπάρχουν και άλλοι τύποι.

Το μέγεθος ενός κόμβου αισθητήρων μπορεί να ποικίλει από δεκάδες εκατοστά ως συσκευές με μέγεθος κόκκου σκόνης. Το κόστος των αισθητήρων είναι ομοίως μεταβλητό, κυμαινόμενο από εκατοντάδες δολάρια ως μερικά σεντς, ανάλογα με το μέγεθος του δικτύου αισθητήρων και την πολυπλοκότητα που απαιτείται από τους μεμονωμένους κόμβους αισθητήρων. Περιορισμοί μεγέθους και κόστους οδηγούν σε αντίστοιχους περιορισμούς σε πόρους όπως η ενέργεια, η μνήμη, η υπολογιστική ταχύτητα και το εύρος ζώνης.

Οι κόμβοι αισθητήρων μπορούν να χαρακτηριστούν ως μικροί υπολογιστές, κυρίως αν λάβουμε υπόψιν τα interfaces τους και τα συστατικά τους. Αποτελούνται συνήθως από μια μονάδα επεξεργασίας με περιορισμένη υπολογιστική δύναμη και περιορισμένη μνήμη, αισθητήρες, μια συσκευή επικοινωνίας (συνήθως ράδιο πομποδέκτες), και μια πηγή ενέργειας συνήθως υπό μορφή μπαταρίας. Επίσης πιθανό είναι να περιλαμβάνουν ενότητες ενεργειακής συγκομιδής, ASICs και ενδεχομένως επιπλέον συσκευές επικοινωνίας (π.χ. RS232 ή USB).

Οι αισθητήρες χρησιμοποιούνται στην καθημερινή ζωή. Οι εφαρμογές τους σχετίζονται με αυτοκίνητα, μηχανές, το διάστημα, την ιατρική, τη βιομηχανία και τη ρομποτική.

Η τεχνολογική πρόοδος επιτρέπει όλο και περισσότερο την κατασκευή αισθητήρων σε μικροσκοπική κλίμακα ως microsensors χρησιμοποιώντας την τεχνολογία MEMS. Στις περισσότερες περιπτώσεις ένα microsensor φθάνει σε σημαντικά υψηλότερες ταχύτητες και ευαισθησία έναντι των μακροσκοπικών προσεγγίσεων.

Η κύρια πρόκληση είναι να παραχθούν κόμβοι αισθητήρων με χαμηλότερο κόστος και μικροσκοπικοί. Όσον αφορά αυτούς τους στόχους, οι τρέχοντες κόμβοι αισθητήρων είναι κυρίως πρωτότυπα. Η μικροσκοπικότητα και το χαμηλό κόστος μπορούν να προκύψουν από την πρόσφατη και μελλοντική πρόοδο στους τομείς MEMS και NEMS. Μερικοί από τους κόμβους είναι ακόμα σε ερευνητικό στάδιο.

5.2.2 Λειτουργικά συστήματα δικτύων αισθητήρων

Τα λειτουργικά συστήματα για τους ασύρματους κόμβους δικτύων αισθητήρων είναι λιγότερο σύνθετα από τα γενικής χρήσης λειτουργικά συστήματα και λόγω των πρόσθετων απαιτήσεων των εφαρμογών δικτύων αισθητήρων και λόγω των περιορισμών των πόρων στο hardware των πλατφορμών των δικτύων αισθητήρων. Παραδείγματος χάριν, οι εφαρμογές δικτύων αισθητήρων δεν είναι αλληλοδραστικές με τον ίδιο τρόπο όπως οι εφαρμογές για τα PC. Για αυτόν τον λόγο, το λειτουργικό σύστημα δεν χρειάζεται να περιλάβει υποστήριξη για interfaces για επικοινωνία με τον χρήστη. Επιπλέον, οι περιορισμοί του hardware όσον αφορά την μνήμη και την χαρτογράφηση της μνήμης καθιστά τους μηχανισμούς όπως η εικονική μνήμη είτε άχρηστους είτε αδύνατους να εφαρμοστούν.

Το hardware των ασύρματων δικτύων αισθητήρων δεν είναι διαφορετικό από τα παραδοσιακά κατανεμημένα συστήματα και είναι επομένως δυνατό να χρησιμοποιηθούν ενσωματωμένα λειτουργικά συστήματα όπως τα eCos ή uC/OS για τα δίκτυα αυτά. Εντούτοις, αυτά τα λειτουργικά συστήματα σχεδιάζονται συχνά με ιδιότητες πραγματικού χρόνου. Αντίθετα από τα παραδοσιακά ενσωματωμένα λειτουργικά συστήματα, τα λειτουργικά συστήματα που στοχεύουν συγκεκριμένα στα δίκτυα αισθητήρων συχνά δεν έχουν ιδιότητες πραγματικού χρόνου. Το TinyOS είναι ίσως το πρώτο λειτουργικό σύστημα που σχεδιάστηκε συγκεκριμένα για τα ασύρματα δίκτυα αισθητήρων. Αντίθετα από τα περισσότερα άλλα λειτουργικά συστήματα, το TinyOS είναι βασισμένο σε ένα event-driven πρότυπο προγραμματισμού αντί για multithreading. Τα προγράμματα TinyOS αποτελούνται από event-handlers και tasks. Όταν συμβαίνει ένα εξωτερικό γεγονός, όπως ένα εισερχόμενο πακέτο στοιχείων ή μια ανάγνωση αισθητήρων, το TinyOS καλεί τον αρμόδιο event-handler για να χειριστεί το γεγονός. Το σύστημα TinyOS καθώς και τα προγράμματα που γράφονται για TinyOS γράφονται σε μια ειδική γλώσσα προγραμματισμού αποκαλούμενη nesC που είναι μια επέκταση στη γλώσσα προγραμματισμού C. Η NesC έχει ως σκοπό να ανιχνεύει τις συνθήκες μετάβασης μεταξύ tasks και event-handlers.

Υπάρχουν επίσης λειτουργικά συστήματα που επιτρέπουν προγραμματισμό σε C, όπως το Contiki, το MANTIS, το BTnut, το SOS και το nano-RK. Το Contiki έχει ως σκοπό να υποστηρίξει την φόρτωση modules από το δίκτυο και τη φόρτωση αρχείων ELF. Ο πυρήνας του Contiki είναι event-driven, όπως του TinyOS, αλλά σε αντίθεση με το TinyOS υποστηρίζει multithreading. Αντίθετα από τον event-driven πυρήνα του Contiki, οι πυρήνες του MANTIS και του nano-RK είναι βασισμένοι σε multithreading. Όπως το TinyOS και το Contiki, το SOS είναι ένα event-driven λειτουργικό σύστημα. Το πρωταρχικό χαρακτηριστικό γνώρισμα του SOS είναι η υποστήριξή του για τα φορτώσιμα modules. Ένα πλήρες σύστημα χτίζεται από μικρότερες ενότητες, ενδεχομένως στο χρόνο εκτέλεσης. Για να υποστηρίξει τον έμφυτο δυναμισμό στη διεπαφή ενότητάς του, το SOS εστιάζει επίσης στην υποστήριξη για τη δυναμική διαχείριση μνήμης. Το BTnut είναι βασισμένο στο συνεργατικό multithreading και σε απλό κώδικα C.

6 Αρχιτεκτονική της εφαρμογής

6.1 Εγκατάσταση των απαραίτητων εργαλείων

Τα εργαλεία που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής είναι τα Tomcat, Axis, Juddi, ActiveBPEL Engine, ActiveBPEL Designer και WSRF – MUSE, TinyOS και Cygwin.

6.1.1 Tomcat

6.1.1.1 Έκδοση του Tomcat

Η έκδοση του Tomcat που χρησιμοποιήσαμε είναι η 5.5.23.

6.1.1.2 Εγκατάσταση του Tomcat

Ο Tomcat είναι ένα open-source λογισμικό, διαθέσιμο στην σελίδα http://jakarta.apache.org. Είναι ένας Servlet και JSP container που παρέχει ένα απλό (και ελεύθερο) περιβάλλον για την ανάπτυξη και τη δοκιμή των εφαρμογών Ιστού, δηλαδή μια Servlet μηχανή. Ο Tomcat είναι η εφαρμογή αναφοράς για το Servlet API και JSP. Αφού εγκαταστήσουμε τον Tomcat στο δικό του directory, μπορούμε να δούμε όλα τα subdirectories από τα οποία αποτελείται και τα οποία είναι:

DIRECTORY	DESCRIPTION
bin	You'll find a number of batch files here. This is where you start/stop the server using startup.bat and shutdown.bat. Some key environmental variables are also defined here. Finally, if you want to precompile your JSP pages, the jspc.bat file can simplify the process.
conf	This directory contains all the configuration files for the server. The main configuration files are in XML. You may want to glance through the XML files to familiarize yourself with their content. The security permissions are also here in the file tomcat.policy.
doc	You will find various documents in this directory such as the User's Guide, Tomcat security, and application development. The API documentation for Java Servlets is available as a separate download from Apache web site.
lib	Several important Jar files are located here including servlet.jar, jaxp.jar, and webserver.jar. Tomcat relies on XML parsers because most of its configuration files are in XML. These jar files are added to your CLASSPATH when you start the Tomcat server.
logs	Tomcat puts its log files in this directory. Servlet invocations are stored in servlet.log and JSP pages are logged in jasper.log. These files can be very helpful when debugging your applications.
src	Source code for the server.
webapps	Your application files are stored under this directory. I'll discuss the

	structure and the mechanics later in this article. As an application developer, you'll spend most of your time working in the webapps directory.
work	Tomcat produces a number of temporary/working files, which are stored in this directory. For example, a JSP page first needs to be converted into a Servlet and then compiled. The Servlet source code and the class file produced by the compiler are stored in this directory. You should not delete items from this directory while the server is working. Sometimes it is helpful to look at the translated Servlet source code for debugging purposes.

Αφού εγκαταστήσουμε τον Tomcat, χρησιμοποιώντας το πρόγραμμα αυτόματης εγκατάστασης .exe, πρέπει να θέσουμε την μεταβλητή περιβάλλοντος JAVA_HOME να βλέπει τον φάκελο όπου είναι εγκατεστημένη, έτσι να μπορεί να δει ο Tomcat που αυτή βρίσκεται. Επίσης πρέπει να θέσουμε την μεταβλητή περιβάλλοντος CATALINA_HOME να δείχνει στον φάκελο του Tomcat.

Για να ελέγξουμε αν η εγκατάσταση έγινε σωστά πάμε στην σελίδα http://locallhost:8080/, και τότε πρέπει να δούμε την σελίδα του Tomcat.

6.1.2 AXIS2

6.1.2.1 Έκδοση του AXIS2

Η έκδοση του Axis2 που χρησιμοποιήσαμε είναι η τελευταία Axis-1.2

6.1.2.2 Τι είναι ο AXIS

Σύμφωνα με το README αρχείο, το ακρωνύμιο AXIS αντιπροσωπεύει τα αρχικά των λέξεων "Apache eXtensible Interaction System".

Ο Apache Axis είναι μια εφαρμογή του SOAP ("Simple Object Access Protocol"). Σύμφωνα με την προδιαγραφή σχεδίων της W3C: Το SOAP είναι ένα ελαφρύ πρωτόκολλο για την ανταλλαγή των δομημένων πληροφοριών σε ένα αποκεντρωμένο, κατανεμημένο περιβάλλον. Είναι ένα βασισμένο στο XML πρωτόκολλο που αποτελείται από τρία μέρη: έναν φάκελο που καθορίζει ένα πλαίσιο για το τι περιέχετε σε ένα μήνυμα και πώς αυτό να επεξεργαστεί, ένα σύνολο κανόνων κωδικοποίησης για δήλωση περιπτώσεων application- defined datatypes, και μια σύμβαση για την αντιπροσώπευση των απομακρυσμένων κλήσεων και των απαντήσεων αυτών.

Το να κωδικοποιήσει ένας πελάτη μιας υπηρεσίας Ιστού το αίτημα του καθώς και να αποκωδικοποιήσει τις απαντήσεις που του επιστρέφονται, σε και από XML θα ήταν επίπονο και δύσκολο. Το ίδιο πράγμα ισχύει εάν γράφουμε την υπηρεσία Ιστού οι ίδιοι. Για αυτόν τον λόγο χρησιμοποιούμε τον Apache Axis.

Ο Axis, όπως αναφέραμε παραπάνω, είναι μια εφαρμογή του πρωτοκόλλου SOAP. Μας γλιτώνει από τις λεπτομέρειες της εξέτασης του SOAP και του WSDL. Χρησιμοποιούμε τον Axis από την πλευρά του server για να γράψουμε την web

service μας (και να την επεκτείνουμε ως Tomcat webapp), και χρησιμοποιούμε τον Axis από την πλευρά του client για να κάνουμε το γράψιμο του client πολύ γρήγορα. Ο Axis είναι ουσιαστικά Apache SOAP 3.0. Έχει στην ουσία ξαναγραφτεί από την αρχή, σχεδιασμένος γύρω από ένα πρότυπο που χρησιμοποιεί το SAX εσωτερικά αντί του DOM. Ο σκοπός είναι να δημιουργηθεί μία πιο μορφοποιημένη, πιο εύκαμπτη, και υψηλών-αποδόσεων υλοποίηση του SOAP.

Χρησιμοποιώντας τον Axis, μπορούμε να επιλέξουμε από 4 διαφορετικά "styles" της web service: RPC, "Document", "Wrapped", and "Message". Η προεπιλογή είναι RPC.

Όταν χρησιμοποιούμε τον Axis για να γράψουμε τον client, δεν χρειάζεται να ασχοληθούμε με SOAP/XML/JAX-RPC. Όλα τα φροντίζει ο Axis για μας, το μόνο που έχουμε να κάνουμε είναι να γράψουμε τα method calls για την web service σαν να ήταν κάποιο τοπικό αντικείμενο. Το ίδιο ισχύει και για την Web service: απλά γράφουμε τις κλάσεις και τις instance methods και όλα τα υπόλοιπα γίνονται από τον Axis. Στην συνέχεια τοποθετούμε την web service στις Tomcat webapp. Ο πελάτης που έχει πρόσβαση στην υπηρεσία Ιστού μπορεί να είναι ένα κανονικό πρόγραμμα σε Java.

6.1.2.3 AXIS1 και AXIS2

Μια νέα αρχιτεκτονική για τον Axis προτάθηκε κατά τη διάρκεια συνεδρίου τον Αυγούστου του 2004 στο Colombo, Σρι Λάνκα. Η νέα αρχιτεκτονική στην οποία είναι βασισμένος ο Axis2 είναι πιο εύκαμπτη, αποδοτική και διαμορφώσιμη σε σύγκριση με την αρχιτεκτονική του Axis1.x. Μερικές καθιερωμένες έννοιες από τον Axis 1.x, όπως οι handlers κ.λπ., έχουν διατηρηθεί στη νέα αρχιτεκτονική.

Ο Apache Axis2 είναι αποδοτικότερο, πιο modular και περισσότερο XML προσανατολισμένος από την παλαιότερη έκδοση. Έχει ως σκοπό να υποστηρίξει την εύκολη προσθήκη plug-in "modules" που επεκτείνουν τη λειτουργία τους για χαρακτηριστικά γνωρίσματα όπως η ασφάλεια και η αξιοπιστία. Τα modules που είναι διαθέσιμα σήμερα ή υπό ανάπτυξη είναι:

- WS-ReliableMessaging
- WS-Coordination και WS-AtomicTransaction
- WS-Security
- WS-Addressing

6.1.3 JUDDI

6.1.3.1 Έκδοση του Juddi

Η έκδοση του Juddi που χρησιμοποιήσαμε είναι η juddi-0.9rc4.

6.1.3.2 Εγκατάσταση του Juddi

Για να λειτουργήσει το Juddi, πρέπει να το συνδέσουμε με μία βάση δεδομένων. Η βάση αυτή θα είναι η PostgreSQL.

Όσον αφορά την εγκατάσταση πρέπει να ακολουθηθούν τα εξής βήματα:

1. Ο φάκελος **{JUDDI-HOME}/webapps/juddi** πρέπει να τοποθετηθεί στον **{TOMCAT-HOME}/webapps**. Πρέπει ο **{TOMCAT-HOME}/webapps /juddi** φάκελος να περιέχει τα *index.html*, *happyjuddi.jsp* μεταξύ των υπόλοιπων υποφακέλων. Αυτό εγκαθιστά την Juddi web application στον Tomcat.
2. Στο **{TOMCAT-HOME}/webapps/juddi/META-INF** φάκελο, τοποθετούμε ένα αρχείο με το όνομα **context.xml**. Το περιεχόμενό του πρέπει να είναι :

```
<Context path="/juddi" docBase="juddi" debug="5"
reloadable="true" crossContext="true">

 <Logger className="org.apache.catalina.logger.FileLogger"
             prefix="localhost_juddiDB_log" suffix=".txt"
             timestamp="true"/>

 <Resource name="jdbc/juddiDB"
        auth="Container"
        type="javax.sql.DataSource"
        username="sa" password=""
        driverClassName="org.hsqldb.jdbcDriver"
        url="jdbc:hsqldb:file:{TOMCAT-HOME}/db/uddi"
        validationQuery="select publisher_ID from PUBLISHER"/>

 <Resource name="jdbc/juddi"
        auth="Container"
        type="javax.sql.DataSource"
        username="sa" password=""
        driverClassName="org.hsqldb.jdbcDriver"
        url="jdbc:hsqldb:file:{TOMCAT-HOME}/db/uddi"
        validationQuery="select publisher_ID from PUBLISHER"/>

</Context>
```

3. Διορθώνουμε το αρχείο {TOMCAT-HOME}/webapps/juddi/WEB-INF/ web.xml προσθέτοντας ακόμη ένα *resource-ref* element, έτσι ώστε να υπάρχουν δύο. Επεξεργαζόμαστε το ένα element από τα δύο έτσι ώστε να έχει ένα <*res-ref-name*> με τιμή jdbc/juddi.
4. Επίσης αλλάζουμε το αρχείο server.xml, το οποίο βρίσκεται στον φάκελο {TOMCAT-HOME}/conf, προσθέτοντας μέσα στο element host και ακριβώς πριν αυτό κλείσει το εξής:

```
<Context path="/juddi" docBase="juddi" debug="5"
reloadable="true" crossContext="true">

<Logger className="org.apache.catalina.logger.FileLogger"
            prefix="localhost_DBTest_log." suffix=".txt"
            timestamp="true"/>


<Resource name="jdbc/juddiDB" auth="Container"
```

```
                type="javax.sql.DataSource"/>

  <ResourceParams name="jdbc/juddiDB">
      <parameter>
        <name>factory</name>
     <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
        </parameter>

<!-- Maximum number of dB connections in pool. Make sure you
     configure your mysqld max_connections large enough to handle
     all of your db connections. Set to 0 for no limit.-->
      <parameter>
        <name>maxActive</name>
        <value>100</value>
      </parameter>

<!-- Maximum number of idle dB connections to retain in pool.
  Set to -1 for no limit.  See also the DBCP documentation on
  this and the minEvictableIdleTimeMillis configuration
  parameter.-->

      <parameter>
        <name>maxIdle</name>
        <value>30</value>
      </parameter>

<!-- Maximum time to wait for a dB connection to become available
     in ms, in this example 10 seconds. An Exception is thrown if
     this timeout is exceeded.  Set to -1 to wait indefinitely.
     -->
      <parameter>
        <name>maxWait</name>
        <value>10000</value>
      </parameter>

<!-- MySQL dB username and password for dB connections  -->

      <parameter>
       <name>username</name>
       <value>sa</value>
      </parameter>

      <parameter>
       <name>password</name>
       <value></value>
      </parameter>

  <!-- Class name for the old mm.mysql JDBC driver - uncomment
  this entry and comment next if you want to use this driver - we
  recommend using Connector/J though

            <parameter>
               <name>driverClassName</name>
               <value>org.gjt.mm.mysql.Driver</value>
            </parameter>
   -->

  <!-- Class name for the official MySQL Connector/J driver -->

        <parameter>
              <name>driverClassName</name>
```

```
                    <value>org.hsqldb.jdbcDriver</value>
                </parameter>

      <!-- The JDBC connection url for connecting to your MySQL dB.
       The autoReconnect=true argument to the url makes sure that
      the mm.mysql JDBC Driver will automatically reconnect if mysqld
      closed the connection.  mysqld by default closes idle
      connections after 8 hours.-->

    <parameter>
      <name>url</name>
<value>jdbc:hsqldb.hsql://localhost:9001/comp200?autoReconnect=true</
value>
    </parameter>

  </ResourceParams>
  </Context>
```

5. Τέλος ελέγχουμε αν η εγκατάσταση έγινε σωστά. Αυτό μπορούμε να το κάνουμε αν ξεκινήσουμε τους servers της βάσης και τον Tomcat και πάμε στην σελίδα http://localhost:8080/juddi/happyjuddi.jsp. Αν δεν υπάρχει κανένα πρόβλημα, δηλαδή αν δεν υπάρχει κόκκινο κείμενο, τότε η εγκατάσταση έχει γίνει σωστά.

6.1.3.3 UDDIBrowser

Ο UDDIBrowser είναι ένα ελεύθερο λογισμικό, γραμμένο σε Java με χρήσης των βιβλιοθηκών του Swing, που προσφέρει ένα φιλικό interface που επιτρέπει στους χρήστες του να βρίσκουν και να χειρίζονται τα περιεχόμενα των UDDI εγγραφών. Ο UDDIBrowser βρίσκεται στην σελίδα http://uddibrowser.org.

6.1.4 ActiveBPEL

6.1.4.1 Έκδοση της ActiveBPEL Engine

Η έκδοση της μηχανής ActiveBPEL που χρησιμοποιήθηκε είναι η 3.1.

6.1.4.2 Τί είναι η ActiveBPEL Engine

Η ActiveBPEL είναι μία εύρωστη μηχανή ικανή να δημιουργεί ροές εργασίας, σχεδιασμένη για τα standards της Business Process Execution Language (BPEL).

Τα βασικά στοιχεία που ενθαρρύνουν την χρήση της ActiveBPEL ως μηχανής για την δημιουργία των ροών εργασίας μας είναι:

- **Αρτιότητα.** Η ActiveBPEL μηχανή που ευρέως εφαρμόζει την BPEL4WS 1.1 specification καθώς και το WSBPEL 2.0 standard. Η μηχανή υποστηρίζει πλήρως τις BPEL activities, τους χειρισμούς γεγονότων, τους χειρισμούς λαθών καθώς και scope/compensation διαχείριση.
- **Βιομηχανική ισχύ.** Πέρα από την περιεκτική υποστήριξη της BPEL, η ActiveBPEL μηχανή χαρακτηρίζεται από τελευταίας τεχνολογίας γνωρίσματα

όπως ανάπτυξη πακέτων(deployment packaging), ειδοποιήσεις γεγονότων και κονσόλες για τα API.

- **Μελλοντική ανάπτυξη.** Ως διανομέας εμπορικών προϊόντων που βασίζονται στην ActiveBPEL μηχανή, η Active Endpoints έχει δεσμευτεί για την συνεχή ανάπτυξη των τεχνολογιών της ActiveBPEL. Το project ελεύθερου λογισμικού της ActiveBPEL θα συνεχίσει να ωφελείται από την Active Endpoints καθώς και από την κοινότητα της ActiveBPEL

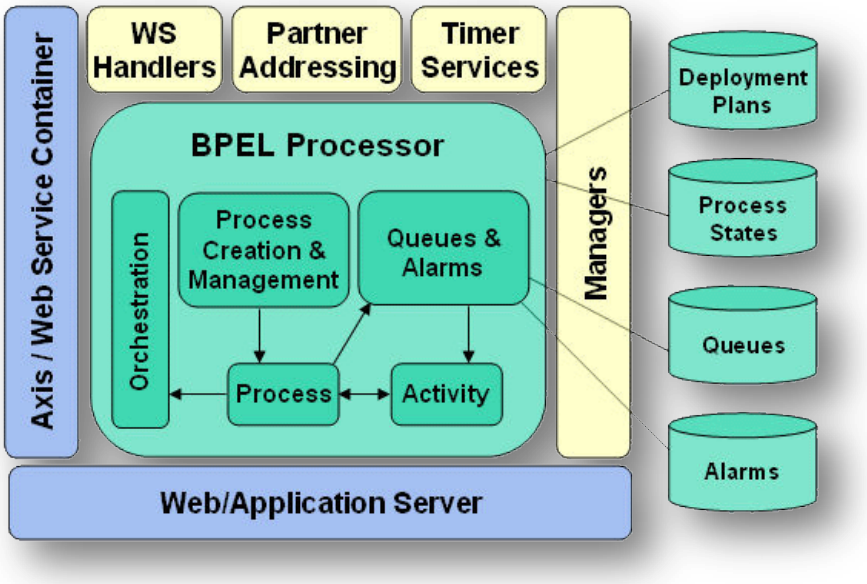
Η ActiveBPEL είναι μία ελεύθερου λογισμικού υλοποίηση της BPEL μηχανής, γραμμένη σε Java. Διαβάζει ορισμούς BPEL διαδικασιών (καθώς και άλλες εισόδους όπως WSDL αρχεία) και δημιουργεί αναπαραστάσεις BPEL ροών εργασίας. Όταν ένα εισερχόμενο μήνυμα προκαλεί την κλήση μιας <activity>, η μηχανή δημιουργεί ένα instance της ροής εργασίας και ξεκινάει να τρέχει.

Η ActiveBPEL μηχανή απαιτεί έναν σωστά εγκατεστημένο servlet container, όπως είναι ο Tomcat ο οποίος μπορεί να καλεί τις web services της ActiveBPEL.

6.1.4.3 Αρχιτεκτονική της ActiveBPEL Engine

Υπάρχουν τρεις κύριες περιοχές στην αρχιτεκτονική της μηχανής ActiveBPEL: η μηχανή (engine), οι διαδικασίες (processes), και οι δραστηριότητες (activities). Η μηχανή ActiveBPEL συντονίζει την εκτέλεση μιας ή περισσότερων διαδικασιών BPEL. Οι διαδικασίες αποτελούνται στη συνέχεια από τις δραστηριότητες, οι οποίες μπορούν να περιέχουν ή να συνδέονται με περαιτέρω δραστηριότητες. Η μηχανή ActiveBPEL δημιουργεί μια διαδικασία από έναν ορισμό μίας process BPEL (ένα αρχείο XML) και έπειτα εκτελεί την διαδικασία αυτήν.

Η αρχιτεκτονική της ActiveBPEL Engine φαίνεται στο παρακάτω σχήμα:



Σχήμα 28: Αρχιτεκτονική της ActiveBpel Engine

6.1.4.4 Εκκίνηση της ActiveBpel Engine

Η Εκκίνηση της ActiveBpel Engine καθώς και των απαραιτήτων υπηρεσιών μπορεί να περιγραφεί από τον ακόλουθο ψευδοκώδικα:

```
engine = new Engine(getEngineConfigurationInfo(),
            createQueueManager(),
            createProcessStateManager());
engine.setPlanManager(createProcessDeploymentProvider());
createProcessDeploymentManager();
createWorkManager();
createAlarmAndTimerService();
```

6.1.4.5 Εγκατάσταση της ActiveBPEL Engine

Για αναλυτικές οδηγίες εγκατάστασης της ActiveBPEL Engine μπορούμε να ανατρέξουμε στην σελίδα της: http://www.active-endpoints.com/installation-guide.htm

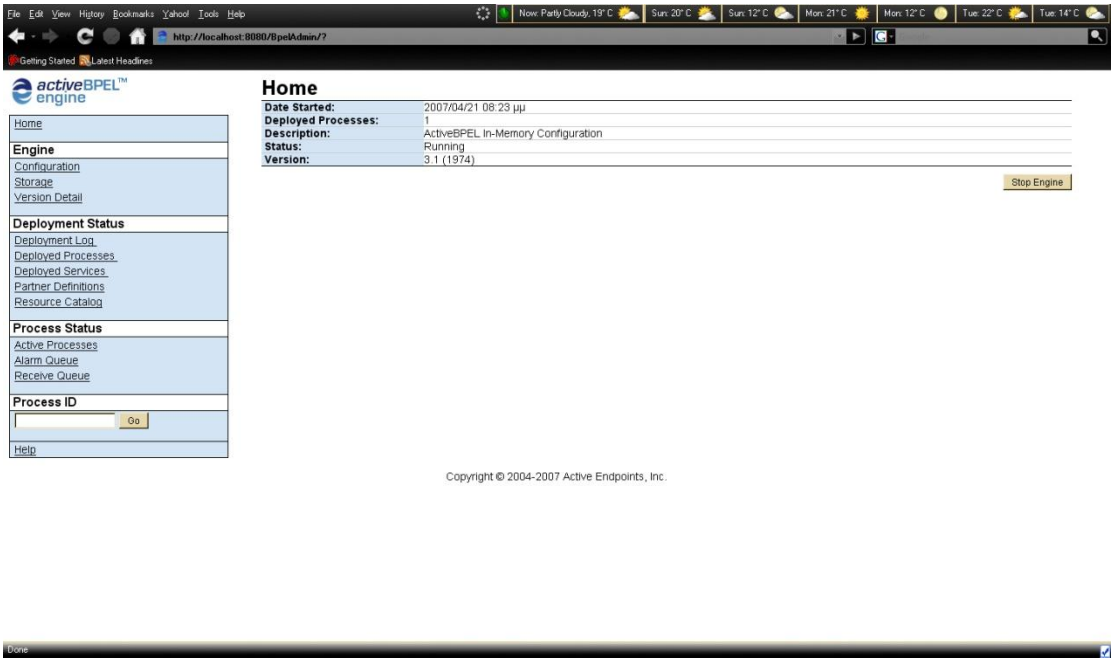
Αρχικά θα κατεβάσουμε τα binary zip αρχεία από την σελίδα http://www.active-endpoints.com/active-bpel-engine-download.htm.

Στην συνέχεια τρέχουμε το script install.bat. Έτσι αντιγράφουμε τα περιεχόμενα της lib στο $CATALINA_HOME/shared/lib φάκελο και δημιουργούμε το φάκελο $CATALINA_HOME/bpr όπου θα αναπτύσσονται τα .bpr αρχεία των BPEL διαδικασιών.

Υπάρχουν μερικοί παράμετροι του server που σχηματίζονται από μικρά XML αρχεία, που ονομάζονται aeEngineConfig.xml και τα οποία βρίσκονται στον φάκελο $CATALINA_HOME/bpr. Τέτοιες αλλαγές γίνονται από την σελίδα BpelAdmin, η οποία βρίσκεται στο http://localhost:8080/BpelAdmin/config.jsp

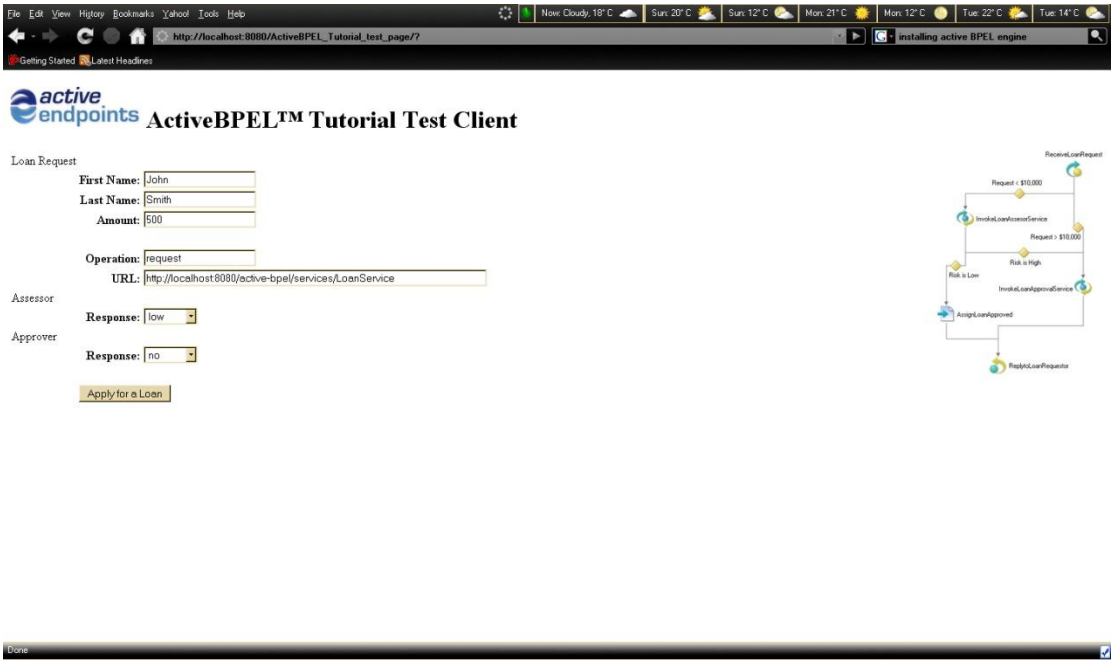
Online βοήθεια για το configuration της μηχανής είναι διαθέσιμο κάνοντας κλικ στο σύνδεσμο "Help" στο μενού της αριστερής πλευράς στο κάτω μέρος της BpelAdmin σελίδας.

Η γραφική διασύνδεση, η οποία μπορεί να βρεθεί στη σελίδα http://localhost:8080/BpelAdmin είναι η εξής:



Σχήμα 28: Η διασύνδεση της σελίδα http://localhost:8080/BpelAdmin

Ενώ αν τρέξουμε κάποια ροή, πληκτρολογώντας το URL της εφαρμογής, για παράδειγμα `http://localhost:8080/ActiveBPEL_Tutorial_test_page`, το αποτέλεσμα θα είναι κάπως έτσι, ανάλογα πάντα με την εφαρμογή:



Σχήμα 29: Η διασύνδεση της σελίδας http://localhost:8080/ActiveBPEL_Tutorial_test_page

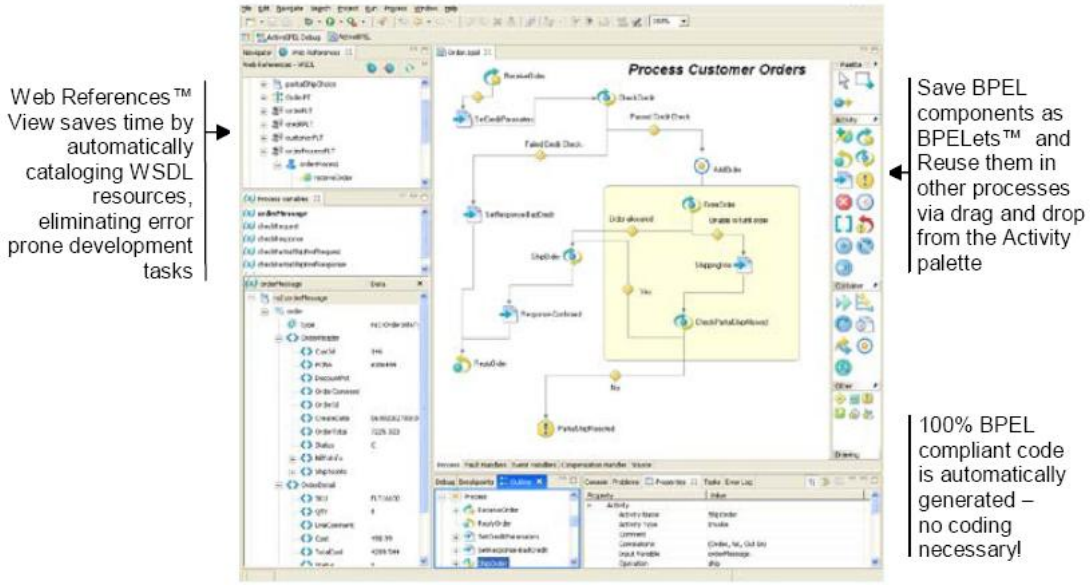
6.1.5 ActiveBPEL Designer

6.1.5.1 Τι είναι ο ActiveBPEL Designer

Η δημιουργία των BPEL αρχείων με το χέρι κουραστική και σίγουρα είναι επιρρεπές σε λάθη. Για αυτό η Active Endpoints προσφέρει σαν ελεύθερο λογισμικό τον ActiveBPEL Designer για την αυτόματη κατασκευή καλοδημιουργημένων BPEL αρχείων καθώς και ο,τιδήποτε άλλο χρειαστεί για την ανάπτυξη των ροών εργασίας στην ActiveBPEL μηχανή. Ο ActiveBPEL Designer είναι ένα ολοκληρωμένο περιβάλλον για ανάπτυξη για γρήγορο στήσιμο και έλεγχο BPEL ροών εργασίας. Σε αντίθεση με τα υπόλοιπα εργαλεία δημιουργίας BPEL αρχείων ο ActiveBPEL Designer προσφέρει ένα γραφικό GUI, σε μορφή "drag and drop", όπου όλες οι ροές εργασίας είναι άμεσα και εύκολα ανακλήσιμες.

Ο ActiveBPEL Designer μπορεί να κατέβει από την σελίδα της active-endpoints http://www.active-endpoints.com/active-bpel-designer.htm και η εγκατάσταση του είναι πολύ απλή, μέσω .exe αρχείου.

Το γραφικό περιβάλλον του είναι το εξής:



Σχήμα 30: Η διασύνδεση του ActiveBPEL Designer

Στο help menu του υπάρχει ένα πολύ χρήσιμο tutorial για την δημιουργία των BPEL αρχείων και όλες τις αρχές λειτουργίας του.

Αφού κάνουμε deploy την ροή εργασίας και ξεκινήσουμε τον server θα χρειαστεί να κάνουμε τα εξής τελευταία βήματα για να τρέξει η ροή:

1. Να αντιγράψουμε το *ActiveBPEL Designer Installation folder*\Samples\ActiveBPEL_Tutorial_Complete_Files\ActiveBPEL_Tutorial_

test_page.war στο *ActiveBPEL Designer Installation folder*\Server \ActiveBPEL_Tomcat\webapps.

2. Να αντιγράψουμε το *ActiveBPEL Designer Installation folder*\Samples \ActiveBPEL_Tutorial_Complete_Files\loan_approval_config.xml στο *ActiveBPEL Designer Installation folder*\Server\ActiveBPEL_Tomcat\temp.
3. Να αντιγράψουμε το *ActiveBPEL Designer Installation folder*\ Samples\ ActiveBPEL_Tutorial_Complete_Files\ActiveBPEL_tutorial_web_services. wsr στο *ActiveBPEL Designer Installation folder*\Server \ActiveBPEL_Tomcat\bpr

6.1.6 WSRF

6.1.6.1 Έκδοση του WSRF

Η έκδοση του WSRF που θα χρησιμοποιηθεί είναι η τελευταία, η 1.1.

6.1.6.2 Εγκατάσταση

Για την πλατφόρμα WSRF έχουμε μιλήσει παραπάνω στο αντίστοιχο κεφάλαιο. Εδώ θα περιγράψουμε τον τρόπο εγκατάστασης της. Το WSRF είναι πακεταρισμένο ως μία Web application. Πιο συγκεκριμένα ως μία Apache Axis application με επιπλέον λειτουργικότητα και άρα με επιπλέον απαίτηση της εφαρμογής της WSRF specification. Έτσι αφού κάνουμε unzip την binary distribution του WSRF, τοποθετούμε τον φάκελο, έστω wsrf-1.1, σε οποιαδήποτε τοποθεσία στον υπολογιστή μας. Από εκεί αντιγράφουμε τον φάκελο webapps/wsrf στον TOMCAT_HOME/webapps. Στην συνέχεια εκκινούμε τον TOMCAT και από μια σελίδα περιήγησης αναζητούμε την διεύθυνση http://localhost:8080/wsrf. Η αρχική σελίδα του WSRF εμφανίζεται. Αν επιλέξουμε View θα δούμε τις ήδη αναπτυγμένες Web services, οι οποίες είναι οι: AdminService, και Version. Επιλέγοντας τα links για τα WSDL αρχεία αυτών βεβαιώνουμε ότι οι υπηρεσίες των WSDL είναι δημοσιευμένες.

Στο binary distribution μπορούμε να βρούμε ένα demo. Αυτό βρίσκεται στο wsrf-1.1/examples/filesystem. Αυτό το παράδειγμα είναι ένα αρχείο σε σύστημα UNIX, του οποίου οι διαχειριστικές δυνατότητες παρουσιάζουν μία Web service με χρήση WSRF. Για να δούμε λοιπόν την παραπάνω εφαρμογή αρκεί να εκτελέσουμε τα παρακάτω βήματα:

1. Ανοίγουμε μία γραμμή εντολών και οδηγούμαστε στον φάκελο wsrf-1.1/examples/ filesystem.
2. Εκτελούμε την εξής εντολή:

   ant compile deploy

3. Εκκινούμε τον Tomcat.
4. Πηγαίνουμε στην σελίδα http://localhost:8080/wsrf/services και επιβεβαιώνουμε ότι το filesystem service είναι αναπτυγμένο.

5. Εδώ, ένας Ant-based SOAP πελάτης, ο οποίος περιέχεται στην συγκεκριμένη διανομή, χρησιμοποιείται για να στείλει μία SOAP αίτηση στο filesystem Web service. Η αίτηση χρησιμοποιεί μία GetMultipleResourceProperties λειτουργία για να επιστρέψει μία λίστα με τις ιδιότητες του resource. Από μία γραμμή εντολών οδηγούμαστε στο directory srf-1.1/examples/filesystem και τρέχουμε την εξής εντολή:

```
ant -f soapclient.xml -
Durl=http://localhost:8080/wsrf/services/filesystem -Dxml=req
```

Δηλαδή, για να δουλέψουμε με το wsrf θα πρέπει να ακολουθήσουμε τα εξής βήματα. Αρχικά πρέπει να δημιουργήσουμε έναν φάκελο για την αποθήκευση των αρχείων μας. Δημιουργούμε, έτσι, έναν φάκελο με το όνομα "work". Εκεί θα αντιγράψουμε τα template αρχεία από τον φάκελο template. Τα αρχεία αυτά περιέχουν ένα WSRF-based WSDL template, ένα ANT build script, και έναν SOAP πελάτη. Αντιγράφουμε ακόμα το FileSystem.wsdl αρχείο που βρίσκεται στο wsrf-1.1/examples/filesystem/src/wsdl/ φάκελο. Το FileSystem.wsdl αρχείο δημιουργήθηκε χρησιμοποιώντας το αρχείο WSRF WSDL template WORK TEMPLATE_.wsdl. Στην συνέχεια χρησιμοποιώντας το Wsdl2Java tool θα δημιουργήσουμε ένα σύνολο από artifacts:

- XMLBeans για όλα τους τύπους των XML Σχημάτων και στοιχεία που προσδιορίζονται στο types section του WSDL
- μία abstract base Resource κλάση
- μία abstract base Service κλάση
- μία abstract base Home κλάση
- μία Resource κλάση
- μία Service κλάση
- μία Home κλάση
- μία CustomOperationsPortType διασύνδεση
- μία PropertyQNames διασύνδεση
- ένα Axis deploy.wsdd αρχείο
- ένα wsrf-config.xml αρχείο

Για να τα δημιουργήσουμε όλα αυτά θα χρησιμοποιήσουμε ένα Ant script το WORK/build.xml που έχει ως στόχο το Wsdl2Java tool. Έτσι οδηγούμαστε στο φάκελο WORK σε μία γραμμή εντολών και εκτελούμε:

ant generate

Τα artifacts θα δημιουργηθούν στο WORK/generated.

(Για να λειτουργήσει το ant πρέπει να βγάλουμε τα σχόλια από το build.properties αρχείο για την μεταβλητή wsrf.webapp.bin)

Στην συνέχεια πρέπει να τροποποιήσουμε την Home κλάση για να περιέχει μία μέθοδο init. Η κλάση αυτή χρησιμοποιείται για να βρίσκει ένα instance ενός resource. Μπορεί να λειτουργήσει σαν εργοστάσιο για να παράγει instances όταν αυτό ζητηθεί ή να παράγει όλα τα instances. Θεωρείτε το αρχικό βασικό σημείο για τον εντοπισμό ενός instance. Άρα στο αρχείο WORK/generated/filesystem/src /java/org/apache/ws/resource/example/filesystem/FilesystemHome.java αντικαθιστούμε την μέθοδο init() με την παρακάτω:

```
public void init() throws Exception
{
    super.init();
    add( createInstance( LVOL1_ID ) );
    add( createInstance( LVOL2_ID ) );
}
```

καθώς και αντιγράφουμε τις μεταβλητές στιγμιοτύπου:

```
private static final String LVOL1_ID = "/dev/vg00/lvol1"
private static final String LVOL2_ID = "/dev/vg00/lvol2"
```

Το επόμενο βήμα είναι να τροποποιήσουμε την Resource κλάση, έτσι ώστε να κάνει implement την init μέθοδο όπως και διάφορες άλλες για τις λειτουργίες του filesystem. Η κλάση αυτή είναι μία αναπαράσταση ενός stateful στιγμιότυπου για την Web service μας. Το resource διατηρεί το resource id καθώς και το ResourcePropertySet. Το resource id είναι ένας μοναδικός αναγνωριστής για κάθε στιγμιότυπο της Web service. Μας επιτρέπει να έχουμε πολλαπλά στιγμιότυπα του resource, το καθένα με διαφορετική κατάσταση, και όλα από την ίδια Web service. Οι καταστάσεις των ιδιοτήτων αντιπροσωπεύονται από το ResourcePropertySet. Το ResourcePropertySet είναι μία Java representation του Resource Properties document που ορίζεται στο types section του WSDL αρχείου. Επομένως για να το τροποποιήσουμε ανοίγουμε το WORK/generated/filesystem/src/java/org/apache/ws/ resource/example/filesystem/FilesystemResource.java αρχείο και αντικαθιστούμε την init μέθοδο με την εξής:

```
    private example.filesystem.backend.FileSystem m_filesystem;

    public void init()
    {
      super.init();

m_filesystem = new example.filesystem.backend.UnixFileSystem(
(String)getID() );

        // create a mock object representing the backend filesystem

      org.apache.ws.resource.properties.ResourcePropertySet
resourcePropertySet = getResourcePropertySet();

      org.apache.ws.resource.properties.ResourceProperty
resourceProperty;

        try
        {

            resourceProperty = resourcePropertySet.get(
FilesystemPropertyQNames.DEVICESPECIALFILE );
            DeviceSpecialFileDocument deviceDocXBean =
DeviceSpecialFileDocument.Factory.newInstance();
```

```
            deviceDocXBean.setDeviceSpecialFile(
m_filesystem.getDeviceSpecialFile() );

            resourceProperty.add( deviceDocXBean );
            resourceProperty.getMetaData().setReadOnly( true );
            resourceProperty = resourcePropertySet.get(
FilesystemPropertyQNames.TYPE );

            TypeDocument typeDocXBean =
TypeDocument.Factory.newInstance();

            typeDocXBean.setType( m_filesystem.getType() );
            resourceProperty.add( typeDocXBean );
            resourceProperty.getMetaData().setReadOnly( true );
            BackupFrequencyDocument backupDocXBean =
BackupFrequencyDocument.Factory.newInstance();

            backupDocXBean.setBackupFrequency(
m_filesystem.getBackupFrequency() );

            resourceProperty = resourcePropertySet.get(
FilesystemPropertyQNames.BACKUPFREQUENCY );
            resourceProperty.add( backupDocXBean );
            resourceProperty.setCallback( new
example.filesystem.callback.BackupFrequencyCallback( m_filesystem )
);
            CommentDocument commentDocXBean =
CommentDocument.Factory.newInstance();
            commentDocXBean.setComment( m_filesystem.getComment() );
            resourceProperty = resourcePropertySet.get(
FilesystemPropertyQNames.COMMENT );

            resourceProperty.add( commentDocXBean );
            resourceProperty.setCallback( new
example.filesystem.callback.CommentCallback( m_filesystem ) );
            FsckPassNumberDocument fsckDocXBean =
FsckPassNumberDocument.Factory.newInstance();
            fsckDocXBean.setFsckPassNumber(
m_filesystem.getFsckPassNumber() );

            resourceProperty = resourcePropertySet.get(
FilesystemPropertyQNames.FSCKPASSNUMBER );
            resourceProperty.add( fsckDocXBean );
            resourceProperty.setCallback( new
example.filesystem.callback.FsckPassNumberCallback( m_filesystem ) );
            MountPointDirectoryDocument mountPointDocXBean =
MountPointDirectoryDocument.Factory.newInstance();
            mountPointDocXBean.setMountPointDirectory(
m_filesystem.getMountPoint() );
            resourceProperty = resourcePropertySet.get(
FilesystemPropertyQNames.MOUNTPOINTDIRECTORY );
            resourceProperty.add( mountPointDocXBean );
            resourceProperty.setCallback( new
example.filesystem.callback.MountPointCallback( m_filesystem ) );
            OptionsDocument optionsDocXBean =
OptionsDocument.Factory.newInstance();

org.apache.ws.resource.example.filesystem.OptionsDocument.Options
options = optionsDocXBean.addNewOptions();

            java.util.List backendOptions =
```

```
m_filesystem.getOptions();

            for ( int i = 0; i < backendOptions.size(); i++ )
            {
               options.addOption( (String) backendOptions.get( i ) );
            }
            resourceProperty = resourcePropertySet.get(
FilesystemPropertyQNames.OPTIONS );

            resourceProperty.add( optionsDocXBean );

            resourceProperty.setCallback( new
example.filesystem.callback.OptionsCallback( m_filesystem ) );

           }
        catch ( Exception e )
        {
            throw new javax.xml.rpc.JAXRPCException( "There was a
problem in initializing your resource properties.  Please check your
init() method. Cause: " + e.getLocalizedMessage() );

        }
    }

    public void mount() throws Exception
    {
        m_filesystem.mount();
    }
    public void unmount() throws Exception
    {
        m_filesystem.unmount();
    }
    public boolean isMounted()
    {
        return m_filesystem.isMounted();
    }
```

Τώρα πλέον μπορούμε να κάνουμε compile ότι έχουμε δημιουργήσει μέχρι εδώ και να το κάνουμε deploy στην WSRF Web application. Υπάρχει ένα Ant script που θα μας βοηθήσει για αυτό. Ακολουθούμε τα παρακάτω βήματα:

1. Αντιγράφουμε το wsrf-1.1/examples/filesystem/src/java/example στο WORK/ generated/filesystem/src/java.
2. Στην γραμμή εντολών, οδηγούμαστε στον φάκελο WORK/generated /filesystem.
3. Βγάζουμε από τα κόμματα και τροποποιούμε τις ρυθμίσεις για το proxy, αν επιθυμούμε ένας proxy να συνδεθεί με εξωτερικά Web sites.
4. Εκτελούμε ant compile deploy. Η υπηρεσία έχει γίνει compiled και deployed στην WSRF Web application.
5. Εκκινούμε τον Tomcat.

6. Και στην διεύθυνση http://localhost:8080/wsrf/services επιβεβαιώνουμε ότι η filesystem υπηρεσία είναι deployed.

Το τελευταίο βήμα είναι να καλέσουμε αυτήν την υπηρεσία. Έτσι χρησιμοποιούμε έναν SOAP πελάτη, που βρίσκεται στο wsrf-1.1 /examples/filesystem/requests για να στέλνει αιτήσεις στην deployed στον Tomcat υπηρεσία μας. Για να γίνει αυτό οδηγούμαστε στον φάκελο WORK σε μία γραμμή εντολών και καλούμε ένα Ant script εκτελώντας σε μία γραμμή:

```
ant -f soapclient.xml -
Durl=http://localhost:8080/wsrf/services/filesystem -
Dxml=wsrf-1.1/examples/filesystem/requests/
QueryResourceProperties_allProps.soap
```

6.1.7 Ant

6.1.7.1 Τι είναι το Ant

Το Apache Ant είναι ένα εργαλείο λογισμικού από την Apache για να αυτοματοποιεί το χτίσιμο του λογισμικού των διαδικασιών. Είναι όμοιο με το make αλλά είναι γραμμένο σε Java, απαιτεί την πλατφόρμα Java, και είναι καταλληλότερο για δημιουργία προγραμμάτων Java.

Η πιο αξιοπρόσεχτη διαφορά μεταξύ του Ant και του make είναι ότι το Ant χρησιμοποιεί XML για να περιγράψει τη διαδικασία κατασκευής και τις εξαρτήσεις του, ενώ το make έχει το format του Makefile. Εξ ορισμού το αρχείο XML ονομάζεται build.xml.

6.1.8 TinyOS

6.1.8.1 Τι είναι το TinyOS

Τα τελευταία χρόνια τα ασύρματα δίκτυα, μέρος των οποίων αποτελούν και τα ασύρματα δίκτυα αισθητήρων, γνωρίζουν μεγάλη επιτυχία και δημοτικότητα. Στην ανάπτυξη αυτή έχει συμβάλει καθοριστικά το λειτουργικό σύστημα TinyOS το οποίο σχεδιάστηκε για τέτοιου είδους δίκτυα.

Motes ονομάζονται οι συσκευές που απαρτίζουν τα ασύρματα αυτά δίκτυα και συνήθως χαρακτηρίζονται από περιορισμένες δυνατότητες. Παρόλα αυτά το TinyOS προσφέρει ευελιξία στην δημιουργία εφαρμογών ασύρματων δικτύων αισθητήρων, παρέχοντας ένα σύνολο από βασικές υπηρεσίες.

6.1.8.2 Έκδοση του TinyOS

Η έκδοση του TinyOS που θα χρησιμοποιηθεί είναι η 1.1.0 η οποία κυκλοφόρησε τον Σεπτέμβριο του 2003 και αποτελεί μια μεγάλη βελτίωση σε σχέση με την 0.6.1 (2001) με πολλές αλλαγές, η βασικότερη από τις οποίες είναι ότι ολόκληρο το σύστημα ξαναγράφτηκε στη γλώσσα nesC. Τον Νοέμβριο του 2006 κυκλοφόρησε και η έκδοση 2.0.

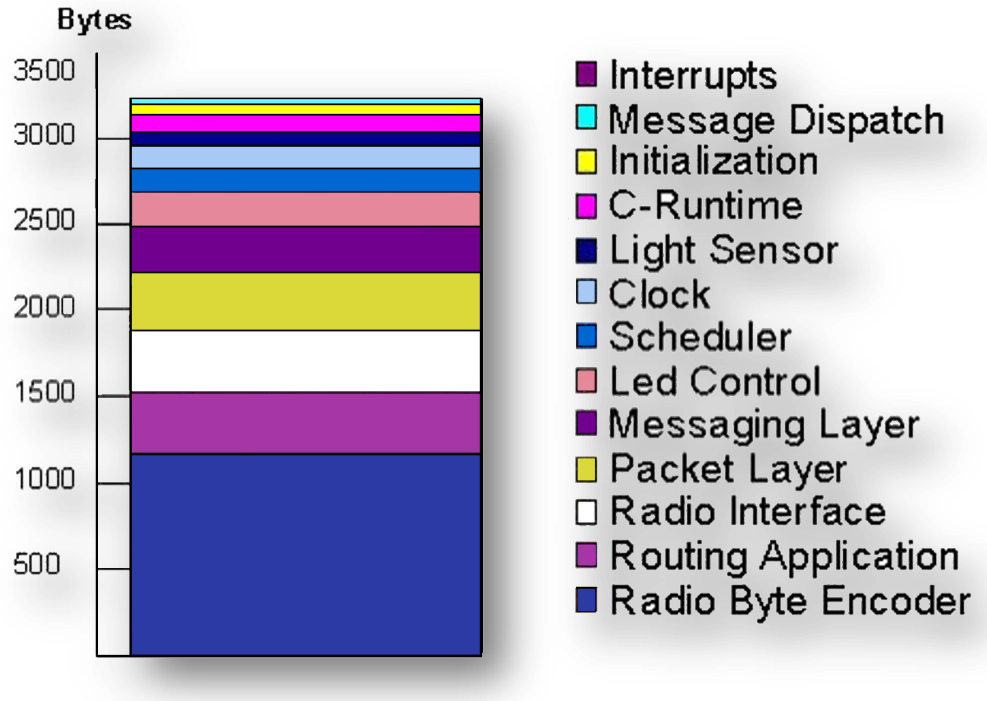
6.1.8.3 Σχεδίαση του TinyOS

Τα προβλήματα που είχε να λύσει το TinyOS ήταν:

- Η χαμηλή κατανάλωση ενέργειας.
- Ιδιαίτερα υψηλές απαιτήσεις για συγχρονισμό:
  - ❖ Ροή πληροφορίας από πολλές πηγές (αισθητήρες, πομποδέκτης)
  - ❖ Μικρή μνήμη, που σημαίνει ότι δεν μπορεί να γίνει buffering, άρα πρέπει να επεξεργαστούμε γρήγορα τα μηνύματα που δεχόμαστε, αλλιώς μπορεί να τα χάσουμε
- Μικρό μέγεθος συνολικά του συστήματος
- Η σχεδίαση να είναι modular για να μπορούμε να φτιάξουμε γρήγορα και εύκολα εφαρμογές.

Όταν ένα mote τρέχει TinyOS έχει εγκατεστημένο στη flash μνήμη του ένα binary εκτελέσιμο image (TinyOS application) με τις βιβλιοθήκες του TinyOS που χρειαζόμαστε, συνδεδεμένες με την εφαρμογή που θέλουμε να εκτελέσουμε. Το TinyOS από μόνο του δεν εκτελεί κάποια ιδιαίτερη λειτουργία και ούτε έχει κάποιο user interface (όπως π.χ το shell στο Unix), οπότε δεν έχει κανένα νόημα να το εγκαταστήσουμε μόνο του σε ένα mote.

Στο TinyOS δεν υπάρχει η έννοια της διεργασίας (process) όπως αυτή ορίζεται στα σύγχρονα λειτουργικά συστήματα. Άλλες έννοιες που δεν συναντάμε είναι αυτές του kernel, της διαχείρισης διαδικασιών (process management), της εικονικής μνήμης (virtual memory), της δυναμικής κατανομής μνήμης (dynamic memory allocation) και των software σημάτων (signals). Εξαιτίας της απουσίας του kernel η διαχείριση του hardware γίνεται απευθείας , μόνο μια διαδικασία υπάρχει στο σύστημα, ο χώρος διευθύνσεων είναι γραμμικός και έχουμε στατική ανάθεση μνήμης όταν γίνεται compile η εφαρμογή στο PC μας. Όλα αυτά συντελούν στο πολύ μικρό μέγεθος του TinyOS, το οποίο δεν ξεπερνά τα 3400 bytes με όλα τα components του συστήματος συμπεριλαμβανόμενα.



Σχήμα 32: Τα τμήματα του TinyOS

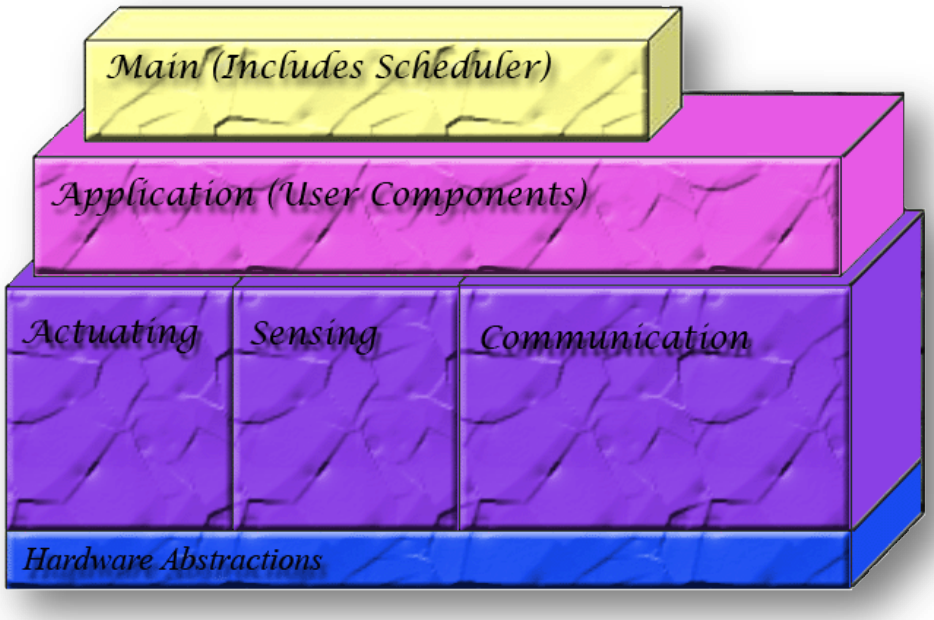
Από την άλλη, επικρατεί η έννοια του component, το οποίο είναι κατά κάποιο τρόπο η αφαίρεση ενός λειτουργικού module του συστήματος. Θα μπορούσαμε να το χαρακτηρίσουμε με ένα πεπερασμένο αυτόματο και θυμίζει τα module που συναντάμε σε γλώσσες περιγραφής υλικού όπως η Verilog και η VHDL, με αρκετές όμως διαφορές. Αν και τα περισσότερα components είναι software modules, μερικά απλά χρησιμοποιούνται ως ένα απλό interface για το hardware του συστήματος. Άλλη σημαντική έννοια είναι αυτή του event. Τα events χρησιμοποιούνται για την επικοινωνία μεταξύ των components και οι αντίστοιχοι handlers που υπάρχουν μέσα στα components μεταβάλλουν την εσωτερική κατάσταση των components τους. Τα events μπορούν να είναι δύο ειδών:

1. Εξωτερικά, προκαλούνται από hardware interrupts. Τέτοια interrupts έχουμε μόνο από τον timer και τον πομποδέκτη.
2. Εσωτερικά, προκαλούνται από event handlers μέσα στα components τα οποία αρχικά ξύπνησαν από κάποιο εξωτερικό event και στη συνέχεια έστειλαν ένα event σε κάποιο άλλο component.

Μια TOS application αποτελείται από ένα scheduler και ένα γράφημα από components, η διασύνδεση των οποίων δείχνει την επικοινωνία μεταξύ τους και τη ροή των events. Αυτή η διασύνδεση ονομάζεται wiring specification και είναι ανεξάρτητη από τα components. Η εφαρμογή συνδέει μόνο τα components τα οποία χρειάζεται για να λειτουργήσει και έτσι στο image που φορτώνουμε στη flash περιέχονται μόνο αυτά και όχι όλα τα components του συστήματος. Οι συνδέσεις

αυτές μεταξύ components, οι οποίες ονομάζονται interfaces, είναι διπλής κατεύθυνσης (bidirectional). Άλλη μια βασική έννοια μίας TOS application είναι αυτήν των tasks, τα οποία χρησιμοποιούνται για εργασίες οι οποίες δεν είναι απαραίτητο να εκτελεστούν αμέσως. Τα tasks μπορούν να καλέσουν άλλες συναρτήσεις ή να προκαλέσουν κάποιο event. Προφανώς είναι απόλυτα σημαντικό να μην χάνουμε μηνύματα και μετρήσεις από τους αισθητήρες και για αυτό γράφουμε όσο το δυνατόν πιο απλά components, που εκτελούνται πολύ γρήγορα. Αν υπάρχουν εργασίες που μπορούν να εκτελεστούν οποιαδήποτε χρονική στιγμή, τις γράφουμε ως tasks και τις στέλνουμε σε μία στοίβα. Η στοίβα αυτή είναι μια απλή FIFO στοίβα στην οποία τοποθετούνται δύο ειδών αντικείμενα, events και tasks. Η στοίβα αυτή ονομάζεται scheduler.

Η χρονοδρομολόγηση που ακολουθεί ο scheduler είναι απλή και πηγάζει από τα προαναφερθέντα. Αφού ένα event έχει μεγαλύτερη σημασία από ένα task, προφανώς θα έχει και μεγαλύτερη προτεραιότητα, οπότε αν υπάρχουν μόνο tasks στη στοίβα και ξαφνικά έρθει ένα event, μπορεί να διακοπεί η εκτέλεση του task και να αρχίσει η εκτέλεση του event. Δηλαδή ένα task μπορεί να γίνει pre-empted από ένα event. Το αντίθετο δεν ισχύει φυσικά. Η εκτέλεση των events και των tasks είναι ατομική.



Σχήμα 33: Η αρχιτεκτονική του TinyOS

6.1.8.4 Εγκατάσταση του TinyOS

Η εγκατάσταση του TinyOS γίνεται αυτόματα από πρόγραμμα εγκατάστασης των motes που θα χρησιμοποιήσουμε. Πριν χρησιμοποιήσουμε όμως το περιβάλλον πρέπει να κάνουμε τα εξής:

1. Να κατεβάσουμε το GNU GCC Compiler (έκδοση 3.2 ή 3.3)και να τον τοποθετήσουμε στον φάκελο /usr/bin/. Έστω ο κεντρικός φάκελος του tinyOS `<tinyos-root-path>` και του GCC compiler `<gcc-path>`.

2. Στον home φάκελο του λογαριασμού μας δημιουργούμε έναν φάκελο (tinyOS home directory) για τα δικά μας αρχεία του tinyOS (π.χ. αλγόριθμοι, τοπολογίες, κλπ.). Στον νέο φάκελο, έστω `<tinyos-home-path>`, πρέπει να αντιγραφούν τα αρχικά αρχεία αλγορίθμων του tinyOS που βρίσκονται στον φάκελο `<tinyos-root-path>/apps`. Επίσης στον φάκελο `<tinyos-home-path>` πρέπει να αντιγραφεί το αρχείο `<tinyos-root-path>/Makefile`, να δημιουργηθεί ένας επιπλέον φάκελος `<tinyos-home-path>/gcc` και μέσα να δημιουργηθεί ένας συμβολικός σύνδεσμος `gcc` στον GNU GCC compiler (έκδοση 3.2 ή 3.3) `<gcc-path>`

csh/tcsh/sh/bash:

```
mkdir <tinyos-home-path>
cp -R <tinyos-root-path>/apps <tinyos-home-path>
cp <tinyos-root-path>/Makefile <tinyos-home-path>
mkdir <tinyos-home-path>/gcc
ln -s <gcc-path> <tinyos-home-path>/gcc/gcc
```

3. Πρέπει να δημιουργήσουμε ένα script που να κάνει χρήση του εργαλείου `make` με την συγκεκριμένη έκδοση του GNU GCC Compiler. Το script πρέπει να τοποθετηθεί στον φάκελο `<tinyos-home-path>` και να ορισθεί ως εκτελέσιμο (`chmod +x tosmake.sh`).

Περιεχόμενα αρχείου tosmake.sh:

```
#!/bin/bash
#
PATH=<tinyos-home-path>/gcc:$PATH
make $*
```

4. Πρέπει να αναθέσουμε την μεταβλητή περιβάλλοντος TOSROOT ίση με <tinyos-root-path> και την μεταβλητή περιβάλλοντος TOSDIR ίση με "<tinyos-root-path>/tos":

csh/tcsh:

```
setenv TOSROOT <tinyos-root-path>
setenv TOSDIR <tinyos-root-path>/tos
```

sh/bash:

```
export TOSROOT=<tinyos-root-path>
export TOSDIR= <tinyos-root-path>/tos
```

5. Πρέπει να συμπληρώσουμε την μεταβλητή περιβάλλοντος PATH, για να γίνει πιο εύκολη η χρήση του script tosmake.sh.

csh/tcsh:

```
setenv PATH <tinyos-home-path>:${PATH}
```

sh/bash:

```
PATH=<tinyos-home-path>:$PATH
```

```
  export PATH
```

6. Επίσης πρέπει να τροποποιήσουμε και την μεταβλητή περιβάλλοντος `CLASSPATH`.

csh/tcsh:
```
  setenv CLASSPATH "`$TOSROOT/tools/java/javapath`":${CLASSPATH}
```

sh/bash:
```
  CLASSPATH="`$TOSROOT/tools/java/javapath`":$CLASSPATH
  export CLASSPATH
```

7. Στο αρχείο <tinyos-home-path>/apps/Makerules εντοπίζουμε την γραμμή:

```
  NESC_FLAGS := -Wnesc-all
```

Και την αλλάζουμε σε:

```
  NESC_FLAGS := -Wnesc-all -topdir=<tinyos-home-path>
```

8. Τέλος μπορούμε να κάνουμε μια δοκιμή με τις επόμενες εντολές:

```
 cd <tinyos-home-path>/apps/Blink
 tosmake.sh pc
```

το περιβάλλον θα επιστρέψει:

```
     compiling Blink to a pc binary
 ncc -o build/pc/main.exe -g -O0 -board=micasb -pthread -target=pc
 -DCC1K_DEF_FREQ=900000000 -Wall -Wshadow -DDEF_TOS_AM_GROUP=0x7d
 -Wnesc-all -fnesc-nido-tosnodes=1000 -fnesc-cfile=build/pc/app.c
Blink.nc -lm
 /opt/tinyos-1.x/tos/platform/pc/heap_array.c:193: warning:
`__nesc_nido_resolve' defined but not used
     compiled Blink to build/pc/main.exe
            25776 bytes in ROM
           618024 bytes in RAM
```

και στην συνέχεια πληκτρολογούμε:

```
 export DBG=led
 ./build/pc/main.exe 1
```

το περιβάλλον πρέπει να επιστρέψει:

```
 0: LEDS: Red off.
 0: LEDS: Red on.
 0: LEDS: Red off.
 0: LEDS: Red on.
 0: LEDS: Red off.
 0: LEDS: Red on.
 0: LEDS: Red on.
 ...
 Exiting on SIGINT at 4:3:10.63160175.
```

6.1.8.5 Η γλώσσα προγραμματισμού nesC

Η γλώσσα προγραμματισμού nesC αναπτύχθηκε στο πανεπιστήμιο της Καλιφόρνια Berkeley σε συνεργασία με την ομάδα που δημιούργησε το TinyOS. Όπως το TinyOS, έτσι και η nesC είναι ένα *open-source project* και όποιος επιθυμεί να δει τον κώδικα για το μεταγλωττιστή της nesC για το TinyOS ή να τον τροποποιήσει, μπορεί να επισκεφθεί τη σχετική ιστοσελίδα nesC για να προμηθευτεί τα αντίστοιχα αρχεία. Η nesC χρησιμοποιείται ως η επίσημη γλώσσα για το TinyOS από την έκδοση 1.0 και μετά. Πριν από τη nesC χρησιμοποιούταν ένα μίγμα από καθαρή C και πολλές μάκρο-εντολές, το οποίο, όμως, ήταν δυσανάγνωστο και όχι εύκολα κατανοητό με αποτέλεσμα να υπάρχουν προβλήματα ασάφειας και debugging.

Μερικές από τις αρχές που διέπουν τη σχεδίαση της nesC είναι:

- **Η nesC βασίζεται στη C:** Αφενός μεν γιατί οι μεταγλωττιστές της C παράγουν αποδοτικό κώδικα για όλους του microcontrollers που χρησιμοποιούνται ως επεξεργαστές σε συστήματα έξυπνης σκόνης και αφετέρου ένα μεγάλο μέρος των προγραμματιστών σε embedded συστήματα χρησιμοποιεί τη C ως γλώσσα προγραμματισμού.
- **Ολική ανάλυση προγράμματος κατά τη διάρκεια της μεταγλώττισης:** Το οποίο συνεπάγεται ακριβέστερο έλεγχο για λάθη και συγχρονισμό (race conditions) στο πρόγραμμα και πιο αποδοτικό κώδικα, δηλαδή μικρό μέγεθος, το οποίο επιβάλλεται από το μικρό μέγεθος της διαθέσιμης μνήμης.
- **Στατικός κώδικας:** Σε μία TOS application έχουμε στατική κατανομή της μνήμης κατά τη διάρκεια της μεταγλώττισης Το γράφημα διασυνδέσεων μεταξύ των διάφορων components είναι σταθερό και γνωστό, το οποίο εξάλλου εξαλείφει την ανάγκη για δυναμική δέσμευση μνήμης και ενθαρρύνει ένα ευέλικτο σχεδιασμό.
- **Η nesC υποστηρίζει και αντικατοπτρίζει τη φιλοσοφία του TinyOS.**

Μία TOS application αποτελείται από διάφορα components, τα οποία είτε είναι module είτε configuration. Τα module υλοποιούν τη λογική που αλλάζει την κατάσταση του αυτόματου, στο οποίο αντιστοιχεί ουσιαστικά ένα component. Αυτό γίνεται με κώδικα που μοιάζει πολύ με C. Πρακτικά αυτό σημαίνει ότι υλοποιούν τα interface, τα οποία έχουν δηλώσει ότι χρησιμοποιούν. Η αντιστοίχηση interface στα modules, δηλαδή ποια interface χρησιμοποιεί ένα module, γίνεται με ένα configuration. Ένα interface χρησιμοποιείται για να ορίσει τον τρόπο που επικοινωνούν δύο modules. Από τα δύο modules, το ένα ονομάζεται χρήστης (user) και το άλλο παροχέας (provider). Έστω ότι έχουμε ένα module πολύ γενικό και αφηρημένο. Για να μπορέσει να εκτελέσει μια λειτουργία χρειάζεται τις υπηρεσίες από άλλα modules, τα οποία είναι λιγότερο γενικά και πιο κοντά ας πούμε στο hardware. Έτσι, το πρώτο module θα χρησιμοποιήσει το interface με το οποίο συνδέεται με ένα module πιο κάτω στην ιεραρχία. Το πρώτο module θα γίνει user και το δεύτερο provider. Ένα interface είναι και το interface ADC (Analog to Digital

Converter) που στην ουσία παίρνει τις μετρήσεις και τις μετατρέπει σε ψηφιακές και τις στέλνει σε ανώτερα επίπεδα , από όπου μπορούμε και τις διαβάζουμε.

Τα interface είναι ο μόνος τρόπος αλληλεπίδρασης μεταξύ των modules και περιέχουν εντολές (commands) και γεγονότα (events). Οι εντολές πρέπει να υλοποιηθούν από τη μεριά του provider ενώ τα events από τη μεριά του user. Οι εντολές μας επιτρέπουν να ελέγξουμε ένα module, ενώ τα events είναι οι αντιδράσεις που προκύπτουν από την εκτέλεση των εντολών.

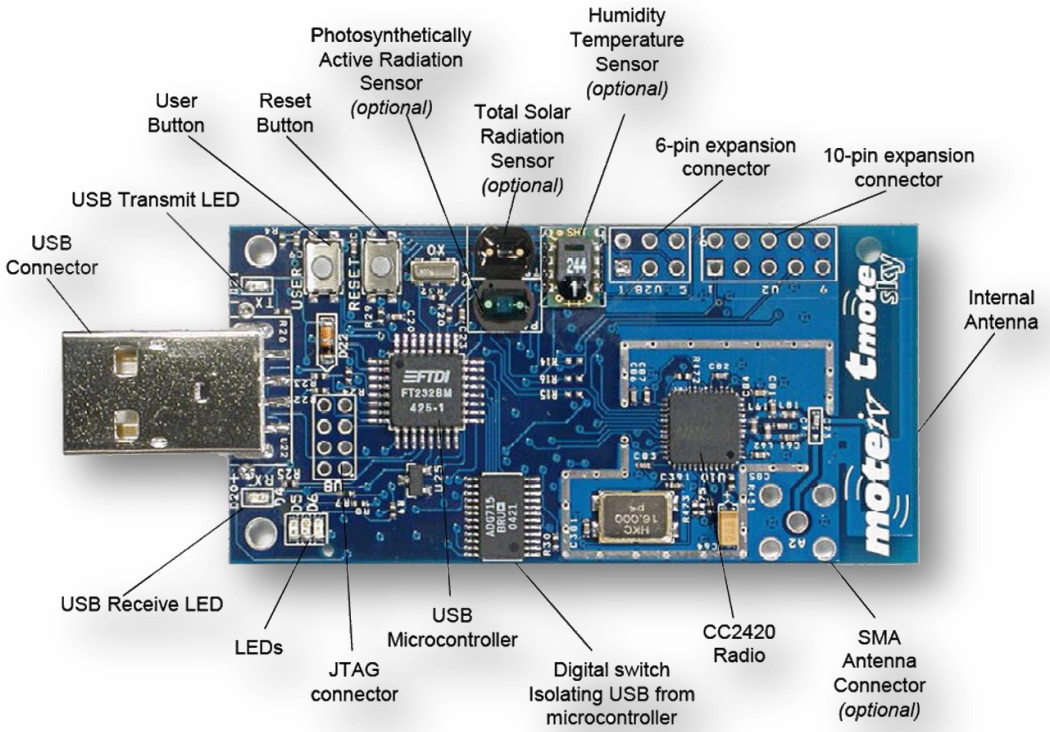
6.1.9 Tmote Sky

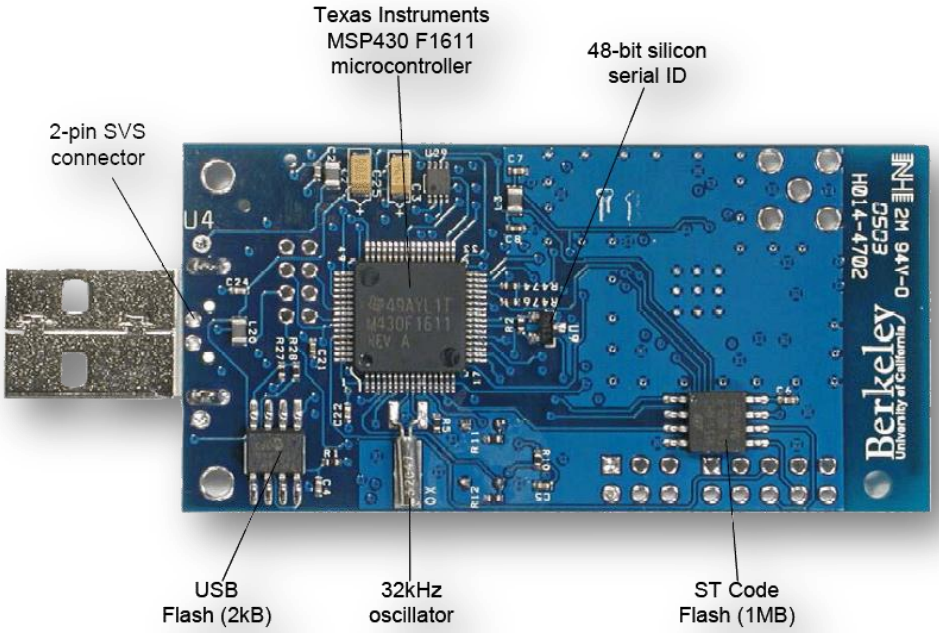
6.1.9.1 Τι είναι τα Tmote Sky

Τα motes που θα χρησιμοποιήσουμε θα είναι τα tmote sky. Τα tmote sky είναι από τα πιο διαδεδομένα ασύρματα motes σήμερα. Αποτελούν την επόμενη γενιά για motes με πολύ μικρή κατανάλωση ενέργειας, τα οποία φέρουν sensors για μέτρηση θερμοκρασίας, υγρασίας και τάσης, δυνατότητα ασύρματης εκπομπής και λήψης, κεραία, microcontroller και προγραμματιστικές δυνατότητες. Διαθέτουν 1MB μνήμη, 10KB RAM, 48KB Memory Flash και 250 kbps Radio bandwidth. Μπορούν έτσι να αποτελέσουν μια εύρωστη λύση για πολλές εφαρμογές ασύρματων δικτύων αισθητήρων. Τα tmote sky υποστηρίζουν το TinyOS ως λειτουργικό σύστημα σε κάθε ένα από αυτά και οι εφαρμογές μπορούν να προγραμματιστούν σε κάθε ένα από αυτά.

Τα βασικά χαρακτηριστικά τους είναι:

• 250kbps 2.4GHz IEEE 802.15.4 Chipcon Wireless Transceiver.
• Σύνδεση με άλλες IEEE 802.15.4 συσκευές.
• 8MHz Texas Instruments MSP430 microcontroller (10k RAM, 48k Flash)
• Ενσωματωμένο ADC, DAC, Supply Voltage Supervisor, and DMA Controller
• Ενσωματωμένη onboard κεραία με 50m εμβέλεια σε εσωτερικούς χώρους / 125m εμβέλεια σε εξωτερικούς.
• Ενσωματωμένα sensors για μέτρηση υγρασίας, θερμοκρασίας και φωτός.
• Εξαιρετικά χαμηλή κατανάλωση ρεύματος.
• Γρήγορο ξύπνημα από ύπνο (<6μs).
• Hardware link-layer κρυπτογράφηση και πιστοποίηση.
• Προγραμματισμός και συλλογή δεδομένων μέσω USB καλωδίου.
• 16-pin υποδοχή επέκτασης και δυνατότητα επιλογής SMA σύνδεσης κεραίας.
• TinyOS υποστήριξη: mesh networking and communication implementation.
• Συμμορφώνεται με το FCC Part 15 και τους κανονισμούς Βιομηχανίας του Καναδά.





Σχήμα 34: Μπροστινό και πίσω μέρος των Tmote Sky

6.1.9.2 Εγκατάσταση των Tmote Sky

Η εγκατάσταση των tmote sky, δηλαδή των FTDI drivers, που πρέπει να εγκατασταθούνε στον host υπολογιστή με τον οποίο θα επικοινωνούνε καθώς και η εγκατάσταση των TinyOS και Cygwin, γίνεται από το πρόγραμμα εγκατάστασης που υπάρχει στο CD των tmote sky. Τα tmote sky εμφανίζονται σαν μια COM port στον device manager των Windows. Πολλά tmote sky μπορούν να συνδεθούν ταυτόχρονα στην ίδια USB port και σε κάθε ένα από αυτά θα ανατεθεί μια διαφορετική COM port.

Για να δούμε ποια tmote sky είναι συνδεδεμένα και σε ποιο port μπορούμε να εκτελέσουμε την εξής εντολή:

```
> motelist
Reference CommPort Description
--------- -------- -----------------------------------
M49WD0S6 COM6 Moteiv tmote sky
```

6.1.9.3 Προγραμματισμός των Tmote Sky

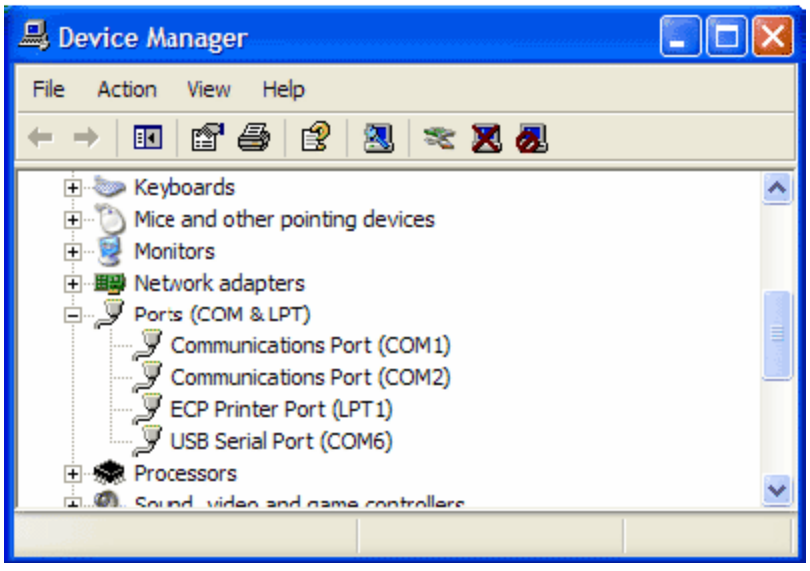
Για να προγραμματίσουμε τα tmote sky αρχικά πρέπει να ανοίξουμε το κεντρικό shell του Cygwin. Αυτό ανοίγει στον φάκελο /opt/moteiv/. Για να κάνουμε compile την εφαρμογή Delta, την βασική εφαρμογή του Moteiv's για wireless sensing και mesh networking απλά αλλάζουμε directory και πάμε στο */opt/moteiv/apps/Delta* και εκτελούμε "make tmote". Δηλαδή:

cd /opt/moteiv/apps/Delta

make tmote

Για να εγκαταστήσουμε μια εφαρμογή όταν μόνο ένα tmote είναι συνδεδεμένο εκτελούμε την επόμενη εντολή.

make tmote reinstall,1



“reinstall,1” σημαίνει προγραμμάτισε το mote με την ήδη compiled binary image και θέσε την διεύθυνση του δικτύου του mote σε 1.

Για να εγκαταστήσουμε μια εφαρμογή σε ένα tmote αν περισσότερα από ένα είναι συνδεδεμένα πρέπει να καθορίσουμε τη σειριακή port όπου πρέπει να εγκατασταθεί, χρησιμοποιώντας την εντολή:

make tmote reinstall,1 bsl,3

“bsl,3” σημαίνει στείλει το πρόγραμμα χρησιμοποιώντας το Boot Strap Loader στην COM4. Ιδιαίτερη προσοχή θέλει το γεγονός ότι το πρόγραμμα bsl την δεικτοδότηση των Linux για τα COM ports, οπότε το COM4 χαρακτηρίζεται ως “3” για το bsl. Προσοχή θέλει επίσης η σωστή διανομή διευθύνσεων δικτύου σε κάθε mote, δηλαδή να τους αποδοθεί από μία μοναδική, απλά αλλάζοντας τον αριθμό μετά από το reinstall.

Άλλες εφαρμογές είναι το Oscilloscope, το οποίο παίρνει όλες τις μετρήσεις από τον αισθητήρα στον οποίο είναι εγκαταστημένη καθώς και η εφαρμογή TOSBase, η οποία φορτώνεται στον αισθητήρα εκείνον ο οποίος θα παίξει τον ρόλο της βάσης, δηλαδή θα είναι συνδεδεμένος με τον υπολογιστή και θα συλλέγει τις μετρήσεις από τους υπόλοιπους αισθητήρες στους οποίους έχουμε εγκαταστήσει την εφαρμογή Oscilloscope.

6.1.9.4 Trawler

Ο Trawler είναι ένα πρόγραμμα γραμμένο σε Java το οποίο εμφανίζει μία γραφική διασύνδεση για να παρουσιάσει αφενός την τοπολογία του δικτύου και αφετέρου τις μετρήσεις από το πρόγραμμα Delta. Μπορούμε να τρέξουμε τον Trawler απλά πληκτρολογώντας την εξής εντολή:

MOTECOM=serial@COM4:tmote java com.moteiv.trawler.Trawler

Η μεταβλητή MOTECOM=“serial@COM4:tmote”, λέει στα Java tools να επικοινωνήσουν χρησιμοποιώντας το σειριακό πρωτόκολλο πάνω από την θύρα COM4 όπου είναι προσαρτημένο ένα mote.

Όταν ξεκινήσει ο Trawler, θα αρχίσει την διαδικασία εγκατάστασης ενός ad-hoc mesh δικτύου και θα εμφανίσει την τοπολογία του δικτύου στην οθόνη.

Ο Trawler συμπεριλαμβάνει ακόμη ένα αριθμό από γνωρίσματα που μας βοηθούν με τις διαδικασίες συλλογής δεδομένων. Επιλέγοντας το “Sensor readings” tab, ο Trawler εμφανίζει τις τιμές της θερμοκρασίας των motes του δικτύου. Στο περιβάλλον αυτό μπορούμε να εστιάσουμε πιέζοντας τα κουμπιά “Zoom In” και “Zoom Out”. Μπορούμε, επίσης, να εστιάσουμε απλώς επιλέγοντας με το ποντίκι μία περιοχή και αυτόματα ο Trawler θα ζουμάρει για να παρουσιάσει τα επιλεγμένα δεδομένα. Ακόμη, μπορούμε να μετακινηθούμε πάνω, κάτω, αριστερά, δεξιά χρησιμοποιώντας τα αντίστοιχα βέλη στην κάτω δεξιά πλευρά της οθόνης.

Άλλη μία δυνατότητα που μας δίνει ο Trawler είναι να επιλέξουμε το “links” tab οπότε και θα εμφανιστεί ένα γράφημα που βοηθάει στην επιφόρτιση του δικτύου, από το οποίο μπορούμε να καθορίσουμε αν η ποιότητα σύνδεσης μεταξύ των κόμβων είναι χαμηλή, ώστε να το διορθώσουμε είτε μετακινώντας κόμβους, είτε προσθέτοντας καινούριους.

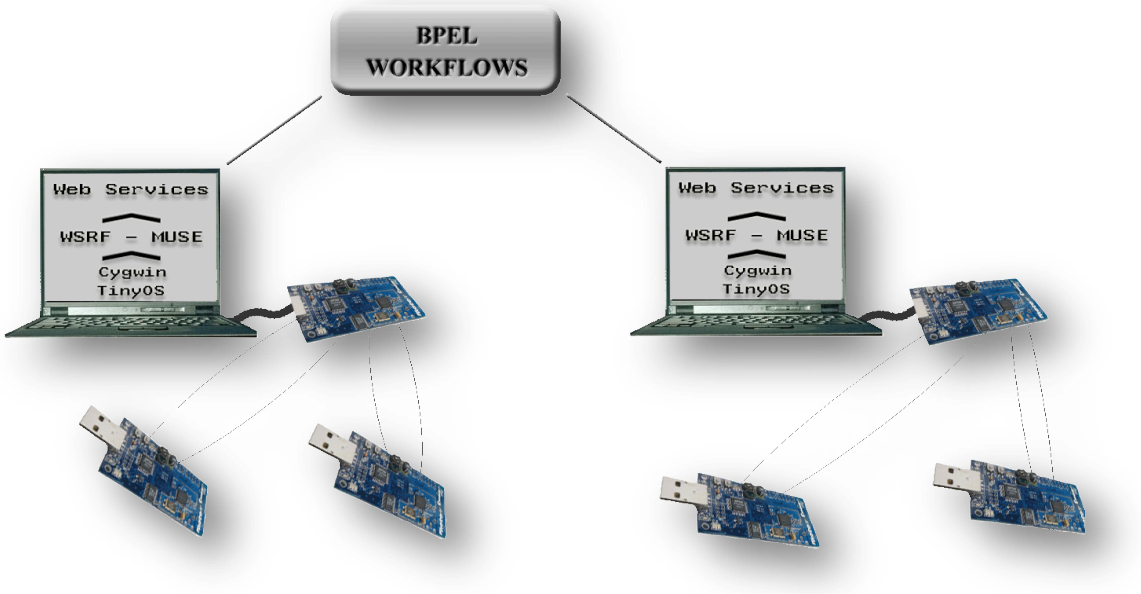
Τέλος μπορούμε να αποθηκεύσουμε τα δεδομένα σε ένα αρχείο επιλέγοντας το “Log Packets” από την “Visualization Controls” μπάρα. Σε περίπτωση που το “Visualization Controls” δεν είναι εμφανές, κάνουμε minimize τον Trawler.

API documentation για την Delta application υπάρχει στο directory /opt/moteiv/doc/nesdoc.

6.2 Υλοποίηση της εφαρμογής

6.2.1 Αρχιτεκτονική

Η αρχιτεκτονική της εφαρμογής που υλοποιήσαμε φαίνεται παρακάτω:



Σχήμα 35: Αρχιτεκτονική της εφαρμογής

Έχουμε δύο υπολογιστές, ο καθένας από τους οποίους ελέγχει δύο κόμβους αισθητήρων tmote sky. Για να το επιτύχουμε αυτό συνδέουμε έναν κόμβο αισθητήρα μέσω USB στον υπολογιστή και του φορτώνουμε την εφαρμογή TOSBase. Ο κόμβος αυτός θα παίξει τον ρόλο της βάσης. Δηλαδή θα είναι συνδεδεμένος με τον υπολογιστή και θα συλλέγει τις μετρήσεις από τους υπόλοιπους, στους οποίους θα έχουμε εγκαταστήσει την εφαρμογή Oscilloscope, που μετράνε το περιβάλλον.

Αρχικά θα φορτώσουμε σε έναν κόμβο την εφαρμογή TOSBase. Συνδέουμε μέσω USB τον κόμβο στον υπολογιστή και εκτελώντας την εντολή *motelist* βλέπουμε σε ποιο COM είναι συνδεδεμένο, έστω στο 5. Στην συνέχεια θα οδηγηθούμε μέσω του Cygwin στον φάκελο με την εφαρμογή TOSBase εκτελώντας την εντολή:

cd /opt/moteiv/apps/Delta

Στην συνέχεια πρέπει να την κάνουμε compile εκτελώντας την εντολή:

make tmote

Και τέλος να την εγκαταστήσουμε στον κόμβο εκτελώντας την εφαρμογή:

make tmote reinstall,3 bsl,4

Οι δύο κόμβοι που μετράνε θα έχουν διευθύνσεις 1 και 2, ενώ η βάση 3. Το 4 σημαίνει ότι θα χρησιμοποιήσουμε το Boot Strap Loader στην COM5.

Επαναλαμβάνουμε την ίδια διαδικασία για τους δύο άλλους κόμβους με την διαφορά ότι φορτώνουμε την εφαρμογή Oscilloscope και σε διαφορετικά COM. Τους δύο αυτούς κόμβους τους αποσυνδέουμε στην συνέχεια από τον υπολογιστή

Στην συνέχεια, έχουμε γράψει ένα πρόγραμμα με το οποίο μπορούμε να παίρνουμε τις μετρήσεις που στέλνουν οι δύο αυτοί κόμβοι αισθητήρων μέσω του κόμβου αισθητήρα βάσης και να μετατρέπουμε τα δεδομένα που παίρνουμε σε αντίστοιχες τιμές της μετρούμενης κλίμακας. Το πρόγραμμα αυτό είναι ο DataCollector. Τα δεδομένα που μετράνε οι αισθητήρες που χρησιμοποιήσαμε είναι θερμοκρασία, υγρασία, ενεργή φωτοσυνθετική ακτινοβολία (PAR), συνολική ηλιακή ακτινοβολία (TSR) καθώς και εσωτερική τάση και ακτινοβολία.

Αρκετές φορές το δευτερόλεπτο λαμβάνουμε πακέτα από κάθε έναν από τους αισθητήρες κάθε κόμβου. Κάθε αισθητήρας που μετράει κάποιο από τα παραπάνω μεγέθη αντιστοιχεί σε ένα κανάλι. Τα κανάλια αυτά αντιστοιχούν με την σειρά τους στους αριθμούς 0 έως 5. Η αντιστοιχία μεταξύ αριθμών και μετρούμενων μεγεθών είναι η ακόλουθη:

0: Humidity
1: Temperature
2: TSR
3: PAR
4: Internal Temperature
5: Internal Voltage

Τα πακέτα αυτά περιλαμβάνουν 10 μετρήσεις. Με το πρόγραμμα DataCollector διαβάζουμε τα πακέτα αυτά, παίρνουμε τον μέσο όρο τον μετρήσεων αυτών και τον αποθηκεύουμε σε ένα αρχείο. Το όνομα του αρχείου θα είναι LegendAB. Όπου AB δύο αριθμοί. Ο πρώτος είναι ο αριθμός που προσδιορίζει τον κόμβο αισθητήρα και είναι αυτός που του έχουμε δώσει εμείς την ώρα που εγκαθιστούμε την εφαρμογή. Ο δεύτερος αριθμός είναι ο αριθμός του καναλιού. Έτσι, λοιπόν, το αρχείο που περιέχει την μέτρηση της θερμοκρασίας από τον κόμβο αισθητήρα με Id ίσο με 2 θα είναι το αρχείο Legend21.txt.

Τα δεδομένα όμως που θέλουμε να αποθηκεύσουμε στα αρχεία πρέπει να μετατραπούν στις αντίστοιχες τιμές επειδή τα δεδομένα που δίνουν οι αισθητήρες είναι ανεπεξέργαστα. Έτσι λοιπόν πρέπει να κάνουμε τις παρακάτω μετατροπές.

Ας ξεκινήσουμε με τα κανάλια 4 και 5 επειδή χρησιμοποιούν τον εσωτερικό 12-bits ADC (Analog to Digital Converter). Για να μετατρέψουμε την μέτρηση του αισθητήρα της εσωτερικής τάσης στο αντίστοιχο αριθμό που δείχνει τάση εφαρμόζουμε την παρακάτω εξίσωση:

(1) value/4096 * Vref, όπου Vref = 1.5V

Ο εσωτερικός αισθητήρας που μετράει την τάση μετράει το Vcc/2, οπότε

πρέπει να πολλαπλασιάσουμε το παραπάνω αποτέλεσμα με 2 για να πάρουμε την τάση τροφοδοσίας (Vcc).

Ομοίως με την εσωτερική τάση, η εσωτερική θερμοκρασία είναι ένα μη καλιμπραρισμένο μέγεθος, το οποίο μετριέται και αυτό από τον 12-bit converter. Η παρακάτω εξίσωση μετατρέπει το μέγεθος αυτό σε βαθμούς Κελσίου:

T = (Vtemp - 0.986)/0.00355

Όσον αφορά την ενεργή φωτοσυνθετική (PAR) και την ολική ηλιακή ακτινοβολία (TSR), παίρνουμε τις μη καλιμπραρισμένες τιμές από τον 12-bit converter με Vref=1.5V. Οι φωτοδίοδοι δημιουργούν ρεύμα μέσω μίας αντίστασης 100kOhm. Αφού υπολογίσουμε την τάση από την εξίσωση (1), την μετατρέπουμε σε ρεύμα χρησιμοποιώντας την εξίσωση V=IR:

(2) I = Vsensor / 100,000, όπου Vsensor το αποτέλεσμα της εξίσωσης (1).

Βασιζόμενοι σε γραφικές παραστάσεις που είναι διαθέσιμες στο Hamamatsu S1087 datasheet, το ρεύμα του αισθητήρα μπορεί να μετατραπεί σε Lux χρησιμοποιώντας τις παρακάτω εξισώσεις:

```
S1087     lx = 0.625 * 1e6 * I * 1000
S1087-01 lx = 0.769 * 1e5 * I * 1000
```

Η υγρασία και η θερμοκρασία μετρούνται από τον εξωτερικό Sensirion αισθητήρα. Η μετατροπή σε SI μονάδες γίνεται ως εξής: Για την θερμοκρασία, η εφαρμογή Oscilloscope επιστρέφει μία 14-bit τιμή, η οποία μπορεί να μετατραπεί σε βαθμούς Κελσίου με την εξίσωση:

(3) temperature = -39.60 + 0.01*Sot, όπου Sot είναι η τιμή του αισθητήρα.

Για την υγρασία η 12-bit τιμή που δίνει ο αισθητήρας δεν είναι αντισταθμισμένη σε σχέση με την θερμοκρασία. Η τιμή της δίνεται από την παρακάτω εξίσωση:

(4) humidity = -4 + 0.0405*SOrh + (-2.8 * 10^-6)*(SOrh^2), όπου SOrh η τιμή που δίνει ο αισθητήρας.

Αν λάβουμε υπόψιν μας και την θερμοκρασία μπορούμε να διορθώσουμε την τιμή της υγρασίας ως εξής:

(5) humidity_true = (Tc - 25) * (0.01 + 0.00008*SOrh) + humidity, όπου Tc η θερμοκρασία σε βαθμούς Κελσίου όπως μετρήθηκε στην εξίσωση (3) και η τιμή που προκύπτει από την εξίσωση (4).

Παρακάτω φαίνεται ο κώδικας του προγράμματος DataCollector, που μαζεύει τις μετρήσεις από τους αισθητήρες και τις γράφει στα αρχεία που αναφέραμε παραπάνω. Η κλάση αυτή γίνεται deployed στον φάκελο cygwin/opt/ moteiv/tools/java. Για να το τρέξουμε πρέπει να ανοίξουμε ένα Shell του Cygwin.

```
package gr.ntua.netmode.sensors;

import net.tinyos.message.Message;
import net.tinyos.message.MessageListener;
import net.tinyos.message.MoteIF;
import net.tinyos.oscope.OscopeMsg;
import net.tinyos.util.PrintStreamMessenger;
import com.moteiv.oscope.Channel;
import com.moteiv.trawler.MultiHopMsg;
import java.io.File;
import java.io.DataOutputStream;
import java.io.FileOutputStream;
import java.io.*;

public class DataCollector implements MessageListener {
	static int NUM_READINGS = 10;
	String LegendName = null;

	public DataCollector() {
		MoteIF mote = null;

		try {
			mote = new MoteIF();
			mote.registerListener(new OscopeMsg(), this);

			} catch (Exception e) {
					e.printStackTrace();
				}
	}
	public void messageReceived(int dest_addr, Message msg) {

		System.out.println("Received Message: " + msg);
		if (msg instanceof OscopeMsg) {
			oscopeReceived(dest_addr, (OscopeMsg) msg);
			} else {
				throw new RuntimeException("messageReceived:
Got bad message type: " + msg);
					}
		}
	public void oscopeReceived(int dest_addr, OscopeMsg omsg) {

		System.out.println("OscopeReceived invoked");
		int moteID, packetNum, channelID, i;
		Channel channel;
		moteID = omsg.get_sourceMoteID();
		channelID = omsg.get_channel();
		packetNum = omsg.get_lastSampleNumber();
		channel = findChannel(moteID, channelID);
		if (channel.getLastPoint() == -1) {
			channel.setLastPoint(packetNum);
		}
		int packetLoss = packetNum - channel.getLastPoint() -
NUM_READINGS;
		for (int j = 0; j < packetLoss; j++) {
			// Add "NUM_READINGS" blank points for each lost
packet
			for (i = 0; i < NUM_READINGS; i++)
				channel.addPoint(null);
		}
		channel.setLastPoint(packetNum);
		int limit = omsg.numElements_data();
```

```
int [] values = new int[10];
          String average_str = null;
          int average_int = 0;
            try{
                  String Legend = LegendName;
                  File f = new File(Legend+".txt");
                  FileOutputStream fouts =new
FileOutputStream(Legend+".txt");
                  DataOutputStream douts =new
DataOutputStream(fouts);
                  if (f!=null)
                  {
                        for (i = 0; i < limit; i++){
                              int val = omsg.getElement_data(i);
                              System.err.println("val: " + val);
                              values[i]=val;
                        }

                        for(i = 0; i <10; i++){
                              average_int+=values[i];
                        }

                        average_int= (int)average_int/10;

                        //convert to the corresponding value
                        //depending on the channel's number

                        if (ch="0"){
                                    Humidity = average_int;
                                    average_int = -4 +
0.0405*Humidity +(-2.8*0,000001)*(Humidity*Humidity);
                              }
                        else if (ch="1"){
                                    Temperature = average_int;
                                    average_int = -39.60 +
0.01*Temperature;
                              }
                        average_str= Integer.toString(average_int);
                        String s1="";
                    String s2=" ";
                    average_str.replaceAll(s2,s1);
                        douts.writeChars(average_str);
                        douts.flush();
                      //f.close();
                        douts.close();
                        fouts.close();
                  }
                  }
      catch (IOException e){
            System.err.println("Error: "+e);
            System.exit(1);
      }

            System.out.println("OscopeReceived finished");
            System.out.println("---------------------------------");
            System.out.println();
            System.out.println();
            System.out.println();

      }
```

```
public Channel findChannel(int moteID, int channelID) {

            System.out.println("**********************");
            System.out.println("FindChannel invoked");
            String legend = "" + moteID + ", " + channelID;
            LegendName= "Legend" + moteID + channelID ;
            Channel c = null;
            if (c == null) {
                  System.out.println("Creating Channel for legend: "
+ legend);
                  c = new Channel();
                  }
            System.out.println("FindChannel finished");
            System.out.println("**********************");
            return c;
      }
      public static void main(String[] args) {
            DataCollector t = new DataCollector();

      }
}
```

Όπως έχουμε αναφέρει και παραπάνω επειδή οι Web services είναι γενικώς stateless, για να μπορούμε – κυρίως σε εφαρμογές Grid – να ελέγχουμε statefull resources χρησιμοποιούμε το WSRF–MUSE. Η πορεία που ακολουθήθηκε περιγράφεται παρακάτω.

Ένα έγγραφο WSDL καθορίζει το interface μίας Web Service που οι απομακρυσμένοι πελάτες θα χρησιμοποιήσουν για να επικοινωνήσουν με το resource. Ενώ η εφαρμογή ενός resource μπορεί να έχει πολλά συστατικά και APIs, το WSDL θα τεκμηριώσει μόνο εκείνα τα πράγματα που μπορούν να επικαλεστούν άμεσα από έναν client. Εκτός από την παροχή της διεπαφής για τους πελάτες, το WSDL των resources χρησιμοποιείται επίσης από το Muse στο ξεκίνημα της εφαρμογής για να καθορίσει την σύμβαση του resource - δηλαδή ποια αιτήματα ισχύουν, ποια APIs πρέπει να περιοριστούν, και πώς να μετατρέψει τα στοιχεία από XML σε αντικείμενα της Java (και αντίθετα). Το αρχείο WSDL, επιπλέον, είναι το σημαντικότερο έγγραφο σε μια βασισμένη στο Muse εφαρμογή επειδή επιβεβαιώνει τι περιέχεται στον deployment descriptor και παρέχει τις λεπτομέρειες που απαιτούνται για να επεξεργαστούν τα αιτήματα SOAP.

Το Muse παρέχει τα εργαλεία που επιτρέπουν την παραγωγή των απαραίτητων αρχείων, για το implementation της εφαρμογής, από ένα WSDL. Κατ' αρχάς, πρέπει να δημιουργήσουμε ένα WSDL αρχείο.

.

Η δημιουργία του WSDL μπορεί να είναι δύσκολη, ακόμα κι αν χρησιμοποιηθεί ένα IDE με έναν συντάκτη WSDL. Για να απλοποιηθεί η διαδικασία δημιουργίας WSDL υπάρχουν πρότυπα και συμβάσεις. Το πρότυπο WSDL παρέχει μια διεπαφή της Web service που περιλαμβάνει όλες τις ιδιότητες και διαδικασίες που καθορίζονται από τα port types WSRF, WSN, και WSDM, και έτσι το μόνο που πρέπει να γίνει είναι τροποποίηση του προτύπου και προσθήκες για τις επιπλέον ιδιότητες και τις διαδικασίες.

Το WSDL που περιγράφει την εφαρμογή μας παρουσιάζεται παρακάτω:

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

<wsdl:definitions

      targetNamespace="http://ws.apache.org/muse/test/sensor"
      xmlns:tns="http://ws.apache.org/muse/test/sensor"
      xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:wsa="http://www.w3.org/2005/08/addressing"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns:wsdl-soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex"
      xmlns:wsrf-r="http://docs.oasis-open.org/wsrf/r-2"
      xmlns:wsrf-rl="http://docs.oasis-open.org/wsrf/rl-2"
      xmlns:wsrf-bf="http://docs.oasis-open.org/wsrf/bf-2"
      xmlns:wsrf-rp="http://docs.oasis-open.org/wsrf/rp-2"
      name="SensorResource">

      <wsdl:types>
            <xsd:schema
                  elementFormDefault="qualified"

      targetNamespace="http://schemas.xmlsoap.org/ws/2004/09/mex">
      <xsd:include schemaLocation="WS-MetadataExchange-2004_09.xsd"/>
            </xsd:schema>
            <xsd:schema
            elementFormDefault="qualified"
            targetNamespace="http://docs.oasis-open.org/wsrf/rl-2">
      <xsd:include schemaLocation="WS-ResourceLifetime-1_2.xsd" />
            </xsd:schema>
            <xsd:schema
            elementFormDefault="qualified"
            targetNamespace="http://docs.oasis-open.org/wsrf/rp-2">
      <xsd:include schemaLocation="WS-ResourceProperties-1_2.xsd" />
            </xsd:schema>
            <xsd:schema
            elementFormDefault="qualified"
            targetNamespace="http://docs.oasis-open.org/wsrf/r-2">
            <xsd:include schemaLocation="WS-Resource-1_2.xsd" />
            </xsd:schema>
```

```
<xsd:schema
            elementFormDefault="qualified"
            targetNamespace="http://ws.apache.org/muse/test/sensor">
      <xsd:import namespace="http://docs.oasis-open.org/wsrf/rl-2" />
                  <!-- Operations defined by Sensor capability -->
                  <!--
                  <xsd:element name="Start"/>
                  <xsd:element name="StartResponse"/>

                  <xsd:element name="Stop"/>
                  <xsd:element name="StopResponse"/>
                  -->
            <!--  Custom faults defined by Sensor operations -->
                  <!--
                  <xsd:element name="StartFailedFault">
                        <xsd:complexType>
                        <xsd:complexContent>
                  <xsd:extension base="wsrf-bf:BaseFaultType" />
                              </xsd:complexContent>
                        </xsd:complexType>
                  </xsd:element>

                  <xsd:element name="StopFailedFault">
                        <xsd:complexType>
                        <xsd:complexContent>
                  <xsd:extension base="wsrf-bf:BaseFaultType" />
                              </xsd:complexContent>
                        </xsd:complexType>
                  </xsd:element>

       -->
      <!-- Properties defined by Sensor capability -->
      <!-- ------------------------------------- -->

            <xsd:element name="Temperature" type="xsd:integer"/>
            <xsd:element name="Humidity" type="xsd:integer"/>
            <xsd:element name="PAR" type="xsd:integer"/>
            <xsd:element name="TSR" type="xsd:integer"/>
            <xsd:element name="InternalVoltage" type="xsd:integer"/>
         <xsd:element name="InternalTemperature" type="xsd:integer"/>

            <!-- WSRP document for Sensor type -->

            <xsd:element name="SensorProperties">
                  <xsd:complexType>
                        <xsd:sequence>

            <!-- WS-RL ScheduledTermination properties -->

            <!-- <xsd:element ref="wsrf-rl:CurrentTime" />
            <xsd:element ref="wsrf-rl:TerminationTime" />
             -->
```

```
            <!-- Sensor properties -->

            <xsd:element ref="tns:Temperature" />
            <xsd:element ref="tns:Humidity" />
            <xsd:element ref="tns:PAR" />
            <xsd:element ref="tns:TSR" />
            <xsd:element ref="tns:InternalVoltage" />
            <xsd:element ref="tns:InternalTemperature" />


                        </xsd:sequence>
                  </xsd:complexType>
            </xsd:element>

            </xsd:schema>
      </wsdl:types>

      <!-- ======== Messages =========== -->

<wsdl:message name="GetMetadataMsg">
      <wsdl:part name="GetMetadataMsg" element="wsx:GetMetadata" />
</wsdl:message>

<wsdl:message name="GetMetadataResponseMsg">
   <wsdl:part name="GetMetadataResponseMsg" element="wsx:Metadata" />
</wsdl:message>

<wsdl:message name="DestroyRequest">
      <wsdl:part name="DestroyRequest" element="wsrf-rl:Destroy" />
</wsdl:message>

<wsdl:message name="DestroyResponse">
    <wsdl:part name="DestroyResponse" element="wsrf-
rl:DestroyResponse" />
</wsdl:message>

<wsdl:message name="ResourceNotDestroyedFault">
      <wsdl:part name="ResourceNotDestroyedFault" element="wsrf-
rl:ResourceNotDestroyedFault" />
</wsdl:message>

<wsdl:message name="ResourceUnknownFault">
            <wsdl:part name="ResourceUnknownFault" element="wsrf-
r:ResourceUnknownFault" />
</wsdl:message>

<wsdl:message name="ResourceUnavailableFault">
            <wsdl:part name="ResourceUnavailableFault" element="wsrf-
r:ResourceUnavailableFault" />
</wsdl:message>

<wsdl:message name="SetTerminationTimeRequest">
            <wsdl:part name="SetTerminationTimeRequest"
element="wsrf-rl:SetTerminationTime" />
</wsdl:message>
```

```
<wsdl:message name="SetTerminationTimeResponse">
            <wsdl:part name="SetTerminationTimeResponse"
element="wsrf-rl:SetTerminationTimeResponse" />
</wsdl:message>

<wsdl:message name="UnableToSetTerminationTimeFault">
            <wsdl:part name="UnableToSetTerminationTimeFault"
element="wsrf-rl:UnableToSetTerminationTimeFault" />
</wsdl:message>

<wsdl:message name="TerminationTimeChangeRejectedFault">
            <wsdl:part name="TerminationTimeChangeRejectedFault"
element="wsrf-rl:TerminationTimeChangeRejectedFault" />
</wsdl:message>

<wsdl:message name="GetResourcePropertyDocumentRequest">
            <wsdl:part name="GetResourcePropertyDocumentRequest"
element="wsrf-rp:GetResourcePropertyDocument"/>
</wsdl:message>

<wsdl:message name="GetResourcePropertyDocumentResponse">
      <wsdl:part name="GetResourcePropertyDocumentResponse"
element="wsrf-rp:GetResourcePropertyDocumentResponse"/>
</wsdl:message>

<wsdl:message name="GetResourcePropertyRequest">
      <wsdl:part name="GetResourcePropertyRequest" element="wsrf-
rp:GetResourceProperty" />
</wsdl:message>

<wsdl:message name="GetResourcePropertyResponse">
      <wsdl:part name="GetResourcePropertyResponse" element="wsrf-
rp:GetResourcePropertyResponse" />
</wsdl:message>

<wsdl:message name="InvalidResourcePropertyQNameFault">
            <wsdl:part name="InvalidResourcePropertyQNameFault"
element="wsrf-rp:InvalidResourcePropertyQNameFault" />
</wsdl:message>

<wsdl:message name="GetMultipleResourcePropertiesRequest">
      <wsdl:part name="GetMultipleResourcePropertiesRequest"
element="wsrf-rp:GetMultipleResourceProperties" />
</wsdl:message>

<wsdl:message name="GetMultipleResourcePropertiesResponse">
      <wsdl:part name="GetMultipleResourcePropertiesResponse"
element="wsrf-rp:GetMultipleResourcePropertiesResponse" />
</wsdl:message>
```

```
<!-- Custom Operations -->
<!--
<wsdl:message name="StartRequest">
      <wsdl:part name="StartRequest" element="tns:Start" />
</wsdl:message>

<wsdl:message name="StartResponse">
      <wsdl:part type="xsd:anyType" />
</wsdl:message>

<wsdl:message name="StartFailedFault"
 <wsdl:part name="StartFailedFault" element="tns:StartFailedFault" />
</wsdl:message>

<wsdl:message name="StopRequest">
      <wsdl:part name="StopRequest" element="tns:Stop" />
</wsdl:message>

<wsdl:message name="StopResponse">
      <wsdl:part type="xsd:anyType" />
</wsdl:message>

<wsdl:message name="StopFailedFault">
   <wsdl:part name="StopFailedFault" element="tns:StopFailedFault" />
</wsdl:message>
     -->


<wsdl:portType
            name="SensorPortType"
          wsrf-rp:ResourceProperties="tns:SensorProperties">
<wsdl:operation name="GetMetadata">
      <wsdl:input
wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata"
      name="GetMetadataMsg" message="tns:GetMetadataMsg"/>
      <wsdl:output
wsa:Action="http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadataResp
onse" name="GetMetadataResponseMsg"
message="tns:GetMetadataResponseMsg"/>
</wsdl:operation>
<wsdl:operation name="Destroy">
                  <wsdl:input wsa:Action="http://docs.oasis-
open.org/wsrf/rlw-2/ImmediateResourceTermination/DestroyRequest"
                              name="DestroyRequest"
message="tns:DestroyRequest" />
                  <wsdl:output wsa:Action="http://docs.oasis-
open.org/wsrf/rlw-2/ImmediateResourceTermination/DestroyResponse"
                               name="DestroyResponse"
message="tns:DestroyResponse" />
                  <wsdl:fault name="ResourceNotDestroyedFault"
message="tns:ResourceNotDestroyedFault" />
                  <wsdl:fault name="ResourceUnknownFault"
message="tns:ResourceUnknownFault" />
                  <wsdl:fault name="ResourceUnavailableFault"
message="tns:ResourceUnavailableFault"/>
</wsdl:operation>
```

```
<wsdl:operation name="SetTerminationTime">
                  <wsdl:input wsa:Action="http://docs.oasis-
open.org/wsrf/rlw-
2/ScheduledResourceTermination/SetTerminationTimeRequest"
                              name="SetTerminationTimeRequest"
message="tns:SetTerminationTimeRequest" />
                  <wsdl:output wsa:Action="http://docs.oasis-
open.org/wsrf/rlw-
2/ScheduledResourceTermination/SetTerminationTimeResponse"
                               name="SetTerminationTimeResponse"
message="tns:SetTerminationTimeResponse" />
                  <wsdl:fault name="UnableToSetTerminationTimeFault"
message="tns:UnableToSetTerminationTimeFault" />
                  <wsdl:fault name="ResourceUnknownFault"
message="tns:ResourceUnknownFault" />
                  <wsdl:fault name="ResourceUnavailableFault"
message="tns:ResourceUnavailableFault"/>
                  <wsdl:fault
name="TerminationTimeChangeRejectedFault"
message="tns:TerminationTimeChangeRejectedFault" />
</wsdl:operation>
<wsdl:operation name="GetResourcePropertyDocument">
                  <wsdl:input wsa:Action="http://docs.oasis-
open.org/wsrf/rpw-
2/GetResourcePropertyDocument/GetResourcePropertyDocumentRequest"

name="GetResourcePropertyDocumentRequest"
message="tns:GetResourcePropertyDocumentRequest"/>
                  <wsdl:output wsa:Action="http://docs.oasis-
open.org/wsrf/rpw-
2/GetResourcePropertyDocument/GetResourcePropertyDocumentResponse"

name="GetResourcePropertyDocumentResponse"
message="tns:GetResourcePropertyDocumentResponse"/>
                  <wsdl:fault name="ResourceUnknownFault"
message="tns:ResourceUnknownFault"/>
                  <wsdl:fault name="ResourceUnavailableFault"
message="tns:ResourceUnavailableFault"/>
</wsdl:operation>
<wsdl:operation name="GetResourceProperty">
                  <wsdl:input wsa:Action="http://docs.oasis-
open.org/wsrf/rpw-2/GetResourceProperty/GetResourcePropertyRequest"
                              name="GetResourcePropertyRequest"
message="tns:GetResourcePropertyRequest" />
                  <wsdl:output wsa:Action="http://docs.oasis-
open.org/wsrf/rpw-2/GetResourceProperty/GetResourcePropertyResponse"
                               name="GetResourcePropertyResponse"
message="tns:GetResourcePropertyResponse" />
                  <wsdl:fault name="ResourceUnknownFault"
message="tns:ResourceUnknownFault"/>
                  <wsdl:fault name="ResourceUnavailableFault"
message="tns:ResourceUnavailableFault"/>
                  <wsdl:fault
name="InvalidResourcePropertyQNameFault"
message="tns:InvalidResourcePropertyQNameFault" />
</wsdl:operation>
```

```
<wsdl:operation name="GetMultipleResourceProperties">
                  <wsdl:input wsa:Action="http://docs.oasis-
open.org/wsrf/rpw-
2/GetMultipleResourceProperties/GetMultipleResourcePropertiesRequest"

name="GetMultipleResourcePropertiesRequest"
message="tns:GetMultipleResourcePropertiesRequest" />
                  <wsdl:output wsa:Action="http://docs.oasis-
open.org/wsrf/rpw-
2/GetMultipleResourceProperties/GetMultipleResourcePropertiesResponse
"

name="GetMultipleResourcePropertiesResponse"
message="tns:GetMultipleResourcePropertiesResponse" />
                  <wsdl:fault name="ResourceUnknownFault"
message="tns:ResourceUnknownFault"/>
                  <wsdl:fault name="ResourceUnavailableFault"
message="tns:ResourceUnavailableFault"/>
                  <wsdl:fault
name="InvalidResourcePropertyQNameFault"
message="tns:InvalidResourcePropertyQNameFault" />
</wsdl:operation>

<!--
<wsdl:operation name="Start">
            <wsdl:input
wsa:Action="http://ws.apache.org/muse/test/sensor/Start"
                                name="StartRequest"
message="tns:StartRequest" />
            <wsdl:output
wsa:Action="http://ws.apache.org/muse/test/sensor/StartResponse"
                                name="StartResponse"
message="tns:StartResponse" />
            <wsdl:fault name="StartFailedFault"
message="tns:StartFailedFault"/>
            <wsdl:fault name="ResourceUnknownFault"
message="tns:ResourceUnknownFault"/>
            <wsdl:fault name="ResourceUnavailableFault"
message="tns:ResourceUnavailableFault"/>
      </wsdl:operation>
      <wsdl:operation name="Stop">
            <wsdl:input
wsa:Action="http://ws.apache.org/muse/test/sensor/Stop"
                                name="StopRequest"
message="tns:StopRequest" />
            <wsdl:output
wsa:Action="http://ws.apache.org/muse/test/sensor/StopResponse"
                                name="StopResponse"
message="tns:StopResponse" />
            <wsdl:fault name="StopFailedFault"
message="tns:StopFailedFault"/>
            <wsdl:fault name="ResourceUnknownFault"
message="tns:ResourceUnknownFault"/>
            <wsdl:fault name="ResourceUnavailableFault"
message="tns:ResourceUnavailableFault"/>
</wsdl:operation>
 -->
</wsdl:portType>
```

```
<wsdl:binding name="SensorBinding" type="tns:SensorPortType">
<wsdl-soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http" />

<wsdl:operation name="GetMetadata">
            <wsdl-soap:operation soapAction="GetMetadata" />
            <wsdl:input>
            <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </wsdl:input>
            <wsdl:output>
            <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
            </wsdl:output>
</wsdl:operation>

<wsdl:operation name="Destroy">
            <wsdl-soap:operation soapAction="Destroy"/>
            <wsdl:input name="DestroyRequest">
            <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </wsdl:input>
            <wsdl:output name="DestroyResponse">
            <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </wsdl:output>
            <wsdl:fault name="ResourceNotDestroyedFault">
                  <wsdl-soap:fault use="encoded"
                        name="ResourceNotDestroyedFault"/>
            </wsdl:fault>
            <wsdl:fault name="ResourceUnknownFault">
                  <wsdl-soap:fault use="encoded"
                        name="ResourceUnknownFault"/>
            </wsdl:fault>
            <wsdl:fault name="ResourceUnavailableFault">
                  <wsdl-soap:fault use="encoded"
                        name="ResourceUnavailableFault"/>
            </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="SetTerminationTime">
            <wsdl-soap:operation soapAction="SetTerminationTime"/>
            <wsdl:input name="SetTerminationTimeRequest">
                  <wsdl-soap:body use="encoded"

      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </wsdl:input>
            <wsdl:output name="SetTerminationTimeResponse">
            <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </wsdl:output>
            <wsdl:fault name="UnableToSetTerminationTimeFault">
                  <wsdl-soap:fault use="encoded"
                        name="UnableToSetTerminationTimeFault"/>
            </wsdl:fault>
```

```
            <wsdl:fault name="ResourceUnknownFault">
                  <wsdl-soap:fault use="encoded"
                        name="ResourceUnknownFault"/>
            </wsdl:fault>
            <wsdl:fault name="ResourceUnavailableFault">
                  <wsdl-soap:fault
                        use="encoded"
                        name="ResourceUnavailableFault"/>
            </wsdl:fault>
            <wsdl:fault name="TerminationTimeChangeRejectedFault">
                  <wsdl-soap:fault
                        use="encoded"
                        name="TerminationTimeChangeRejectedFault"/>
            </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="GetResourcePropertyDocument">
            <wsdl-soap:operation
soapAction="GetResourcePropertyDocument"/>
            <wsdl:input name="GetResourcePropertyDocumentRequest">
                  <wsdl-soap:body use="encoded
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </wsdl:input>
            <wsdl:output name="GetResourcePropertyDocumentResponse">
                  <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </wsdl:output>
            <wsdl:fault name="ResourceUnknownFault">
                  <wsdl-soap:fault use="encoded"
                        name="ResourceUnknownFault"/>
            </wsdl:fault>
            <wsdl:fault name="ResourceUnavailableFault">
                  <wsdl-soap:fault use="encoded"
                        name="ResourceUnavailableFault"/>
            </wsdl:fault>
</wsdl:operation>
<wsdl:operation name="GetResourceProperty">
            <wsdl-soap:operation soapAction="GetResourceProperty"/>
            <wsdl:input name="GetResourcePropertyRequest">
                  <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </wsdl:input>
            <wsdl:output name="GetResourcePropertyResponse">
                  <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </wsdl:output>
            <wsdl:fault name="ResourceUnknownFault">
                  <wsdl-soap:fault use="encoded"
                        name="ResourceUnknownFault"/>
            </wsdl:fault>
            <wsdl:fault name="ResourceUnavailableFault">
                  <wsdl-soap:fault use="encoded"
                        name="ResourceUnavailableFault"/>
            </wsdl:fault>
            <wsdl:fault name="InvalidResourcePropertyQNameFault">
                  <wsdl-soap:fault use="encoded"
                        name="InvalidResourcePropertyQNameFault"/>
            </wsdl:fault>
</wsdl:operation>
```

```
<wsdl:operation name="GetMultipleResourceProperties">
            <wsdl-soap:operation
soapAction="GetMultipleResourceProperties"/>
            <wsdl:input name="GetMultipleResourcePropertiesRequest">
                  <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </wsdl:input>
            <wsdl:output
name="GetMultipleResourcePropertiesResponse">
                  <wsdl-soap:body use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </wsdl:output>
            <wsdl:fault name="ResourceUnknownFault">
                  <wsdl-soap:fault
                        use="encoded"
                        name="ResourceUnknownFault"/>
            </wsdl:fault>
            <wsdl:fault name="ResourceUnavailableFault">
                  <wsdl-soap:fault
                        use="encoded"
                        name="ResourceUnavailableFault"/>
            </wsdl:fault>
            <wsdl:fault name="InvalidResourcePropertyQNameFault">
                  <wsdl-soap:fault
                        use="encoded"
                        name="InvalidResourcePropertyQNameFault"/>
            </wsdl:fault>
 </wsdl:operation>
<!--
<wsdl:operation name="Start">
            <wsdl-soap:operation soapAction="Start"/>
            <wsdl:input name="StartRequest">
                  <wsdl-soap:body
                        use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </wsdl:input>
            <wsdl:output name="StartResponse">
                  <wsdl-soap:body
                        use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </wsdl:output>
            <wsdl:fault name="StartFailedFault">
                  <wsdl-soap:fault
                        use="encoded"
                        name="StartFailedFault"/>
            </wsdl:fault>
            <wsdl:fault name="ResourceUnknownFault">
                  <wsdl-soap:fault
                        use="encoded"
                        name="ResourceUnknownFault"/>
            </wsdl:fault>
            <wsdl:fault name="ResourceUnavailableFault">
                  <wsdl-soap:fault
                        use="encoded"
                        name="ResourceUnavailableFault"/>
            </wsdl:fault>
</wsdl:operation>
```

```
<wsdl:operation name="Stop">
            <wsdl-soap:operation soapAction="Stop"/>
            <wsdl:input name="StopRequest">
                  <wsdl-soap:body
                        use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </wsdl:input>
            <wsdl:output name="StopResponse">
                  <wsdl-soap:body
                        use="encoded"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </wsdl:output>
            <wsdl:fault name="StopFailedFault">
                  <wsdl-soap:fault
                        use="encoded"
                        name="StopFailedFault"/>
            </wsdl:fault>
            <wsdl:fault name="ResourceUnknownFault">
                  <wsdl-soap:fault
                        use="encoded"
                        name="ResourceUnknownFault"/>
            </wsdl:fault>
            <wsdl:fault name="ResourceUnavailableFault">
                  <wsdl-soap:fault
                        use="encoded"
                        name="ResourceUnavailableFault"/>
            </wsdl:fault>
</wsdl:operation>
-->

</wsdl:binding>

<wsdl:service name="SensorService">
      <wsdl:port name="SensorPort" binding="tns:SensorBinding">
            <wsdl-soap:address
location="http://localhost:8080/SensorGrid/services/sensor"/>
      </wsdl:port>
      </wsdl:service>


</wsdl:definitions>
```

Από την στιγμή που έχουμε το WSDL αρχείο που περιγράφει το endpoint, χρησιμοποιούμε το εργαλείο wsdl2java για να παράγουμε τον απαραίτητο κώδικα για να κάνουμε implement και deploy την εφαρμογή μας. Αφού τρέξουμε την εντολή, η παρακάτω δομή θα εμφανιστεί στον τρέχον φάκελο.

```
/JavaSource
   ...
/WebContent
   /WEB-INF
      /services
            /muse
               /wsdl
                  ...
               /router-entries
                  ...
               /META-INF
                  /services.xml
               /muse.xml
   ...
/build.xml
```

- **/JavaSource** – περιέχει τον παραγόμενο κώδικα. Για κάθε capability, το namespace URI του κάθε capability αν είναι δυνατόν θα μετατραπεί σε όνομα πακέτου και θα δημιουργηθούν δύο .java αρχεία:
    - IMyCapability.java
    - MyCapability.java

Το πρώτο αρχείο είναι ένα interface που περιγράφει τις μεθόδους του capability. Το δεύτερο είναι ένα implementation του interface που περιέχει κώδικα που θα αλλαχτεί σύμφωνα με τις απαιτήσεις της εφαρμογής.

- **/WebContent** – περιέχει ένα web archive που είναι έτοιμο να γίνει deployed όταν ο παραγόμενος κώδικας γίνει compiled και τοποθετηθεί στο **/WebContent/WEB-INF/lib**. Όλη η παραπάνω διαδικασία του compilation και του πακεταρίσματος γίνεται αυτόματα από ένα Ant script, το οποίο μπορούμε πάντα να αλλάξουμε σύμφωνα με τις απαιτήσεις τις εφαρμογής μας.

- **/WebContent/WEB-INF/services/muse** – περιέχει όλες τις meta - πληροφορίες που αφορούν το endpoint.
    - **/wsdl** - το WSDL αρχείο.
    - **/router-entries** - προεπιλεγμένο, όταν ένα endpoint αρχίζει, θέλουμε τουλάχιστον ένα endpoint να υπάρχει για να μπορούμε να αλληλεπιδράσουμε με αυτό. Το εργαλείο θα δημιουργήσει ένα μονό endpoint persistence entry, το οποίο δεν είναι σημαντικό έως ότου να κάνουμε deploy το resource και να συνδεθούμε με αυτό.
    - **/META-INF/services.xml** – αυτό το αρχείο δημιουργείται σύμφωνα με τις λειτουργίες που είναι ορισμένες στο WSDL.
    - **/muse.xml** - αυτό το αρχείο δημιουργείται επίσης σύμφωνα με τις capabilities που βρίσκονται στο WSDL.
    -
- **/build.xml** - το *ant* build αρχείο που έχει από default ένα target για να κάνει compile ό,τι βρίσκεται στο **/JavaSource** πακέτο σε ένα *jar* αρχείο, να το αντιγράψει στο **/WebContent/WEB-INF/lib** και στην συνέχεια να πακετάρει το περιεχόμενο του **/WebContent** σε ένα WAR αρχείο το οποίο είναι έτοιμο να γίνει deployed στον J2EE server.

Πριν τρέξουμε το Ant θα πρέπει να κάνουμε τα ακόλουθα:

- **Να συμπληρώσουμε τις παραγόμενες μεθόδους σύμφωνα με τις απαιτήσεις της εφαρμογής** - ο παραγόμενος κώδικας στο **/JavaSource** περιέχει τον σκελετό των μεθόδων και δεν έχει implementation. Αυτές οι μέθοδοι είναι μαρκαρισμένες με το σχόλιο "TODO" και πετούνε `RuntimeException exceptions` αφού δεν έχουν implementation.

- **Να δώσουμε default τιμές για τους non-primitive τύπους** - αν στο WSDL αρχείο έχουν οριστεί ιδιότητες που δεν είναι primitive types, πρέπει να αρχικοποιηθούν αν το `minOccurs` είναι τουλάχιστον 1. Για τους primitive τύπους - το String δεν είναι primitive τύπος – η Java ορίζει λογικές default τιμές. Όμως για τους non-primitive τύπους η default τιμή είναι μηδέν. Όταν το resource αρχικοποιείται και μία non-primitive ιδιότητα πρέπει να συμβεί τουλάχιστον μία φορά, σύμφωνα με το schema και η default τιμή δεν έχει αλλάξει, το runtime θα εντοπίσει 0 περιστατικά τις ιδιότητας και θα πετάξει μία `BelowMinimum` runtime exception. Αυτήν η ενέργεια είναι προαιρετική και την εκτελούμε μόνο σε περίπτωση που έχουμε ορίσει non-primitive τύπους.

Στην δική μας περίπτωση εκτελούμε μόνο το πρώτο από τα ανωτέρω βήματα και τα δύο αρχεία IMyCapability.java και MyCapability.java μετά από την συμπλήρωση έχουν ως εξής:

Το IMyCapability.java το οποίο δεν τροποποιήθηκε:

```
//
// IMyCapability.java
// Thu May 24 15:56:37 EEST 2007
// Generated by the Apache Muse Code Generation Tool
//
package org.apache.ws.muse.test.sensor;
public interface IMyCapability
{
    String PREFIX = "tns";
    String NAMESPACE_URI = "http://ws.apache.org/muse/test/sensor";

    public int getTemperature();

    public void setTemperature(int param0);

    public int getInternalTemperature();

    public void setInternalTemperature(int param0);

    public int getInternalVoltage();

    public void setInternalVoltage(int param0);

    public int getPAR();
```

```
    public void setPAR(int param0);

    public int getHumidity();

    public void setHumidity(int param0);

    public int getTSR();


    public void setTSR(int param0);
}
```

Και το MyCapability.java στο οποίο συμπληρώσαμε τις μεθόδους σύμφωνα με τις απαιτήσεις τις εφαρμογής:

```
//
// MyCapability.java
// Thu May 24 15:56:37 EEST 2007
// Generated by the Apache Muse Code Generation Tool
//
package org.apache.ws.muse.test.sensor;
import javax.xml.namespace.QName;
import java.io.*;
import org.apache.muse.ws.addressing.soap.SoapFault;
import org.apache.muse.ws.resource.WsResource;
import
org.apache.muse.ws.resource.impl.AbstractWsResourceCapability;
public class MyCapability extends AbstractWsResourceCapability
implements IMyCapability
{
      private String resourceId = null;
      private String moteID = null;

    private static final QName[] _PROPERTIES = new QName[]
    {
        new QName(NAMESPACE_URI, "Temperature", PREFIX),
        new QName(NAMESPACE_URI, "InternalTemperature", PREFIX),
        new QName(NAMESPACE_URI, "InternalVoltage", PREFIX),
        new QName(NAMESPACE_URI, "PAR", PREFIX),
        new QName(NAMESPACE_URI, "Humidity", PREFIX),
        new QName(NAMESPACE_URI, "TSR", PREFIX)
    };
```

```
public QName[] getPropertyNames()
    {
        return _PROPERTIES;
    }

    @Override
    public void initializeCompleted() throws SoapFault {
      super.initializeCompleted();

      WsResource sensorResource = this.getWsResource();
      resourceId =
sensorResource.getEndpointReference().getParameterString(new
QName("http://ws.apache.org/muse/addressing", "ResourceId", "muse-
wsa"));

      moteID = resourceId.substring(15);

      System.out.println("My Resource ID = " + resourceId);
      this.getLog().info("My Resource ID = " + resourceId);
      this.getLog().info("My Mote ID = " + moteID);
    }
    private int _Temperature;
    private int _InternalTemperature;
    private int _InternalVoltage;
    private int _PAR;
    private int _Humidity;
    private int _TSR;
    public int getTemperature()
    {
      String str ="";
            try{
  FileReader f = new FileReader("C:\\cygwin\\home\\gollum\\
.DataCollector\\Legend" + moteID + "0.txt");
                    BufferedReader b = new BufferedReader(f);

                        try{
                              str = b.readLine();
                              System.out.println("String= " + str);
                        }
                        catch (IOException e)
                        {
                        }
            }
            catch (FileNotFoundException e)
            {
            System.err.println("There is no file named like
this...sorry...");
            }
            _Temperature = Integer.parseInt(str);
        return _Temperature;
    }
```

```
    public void setTemperature(int param0)
    {
        _Temperature = param0;
    }
    public int getInternalTemperature()
    {
      String str ="";
            try{

FileReader f = new FileReader("C:\\cygwin\\home\\gollum\\
.DataCollector\\Legend" + moteID + "4.txt");
                  BufferedReader b = new BufferedReader(f);

                        try{
                              str = b.readLine();
                              System.out.println("String= " + str);
                        }
                        catch (IOException e)
                        {
                        }
            }
            catch (FileNotFoundException e)
            {
            System.err.println("There is no file named like
this...sorry...");
            }
            _InternalTemperature = Integer.parseInt(str);
        return _InternalTemperature;
    }

    public void setInternalTemperature(int param0)
    {
        _InternalTemperature = param0;
    }
    public int getInternalVoltage()
    {
      String str ="";
            try{

FileReader f = new FileReader("C:\\cygwin\\home\\gollum\\
.DataCollector\\Legend" + moteID + "5.txt");
                  BufferedReader b = new BufferedReader(f);

                        try{
                              str = b.readLine();
                              System.out.println("String= " + str);
                        }
                        catch (IOException e)
                        {
                        }
            }
            catch (FileNotFoundException e)
            {
            System.err.println("There is no file named like
this...sorry...");
            }
            _InternalVoltage = Integer.parseInt(str);
        return _InternalVoltage;
    }
```

```
    public void setInternalVoltage(int param0)
    {

        _InternalVoltage = param0;
    }
    public int getPAR()
    {
      String str ="";
            try{

                  FileReader f = new
FileReader("C:\\cygwin\\home\\gollum\\.DataCollector\\Legend" +
moteID + "2.txt");
                  BufferedReader b = new BufferedReader(f);

                        try{
                              str = b.readLine();
                              System.out.println("String= " + str);
                        }
                        catch (IOException e)
                        {
                        }
            }
            catch (FileNotFoundException e)
            {
            System.err.println("There is no file named like
this...sorry...");
            }
            _PAR = Integer.parseInt(str);
           return _PAR;
    }
    public void setPAR(int param0)
    {
        _PAR = param0;
    }
    public int getHumidity()
    {
      String str ="";
            try{

FileReader f = new FileReader("C:\\cygwin\\home\\gollum\\
.DataCollector\\Legend" + moteID + "1.txt");
                  BufferedReader b = new BufferedReader(f);

                        try{
                              str = b.readLine();
                              System.out.println("String= " + str);
                        }
                        catch (IOException e)
                        {
                        }
            }
            catch (FileNotFoundException e)
```

```
{
            System.err.println("There is no file named like
this...sorry...");
            }
            _Humidity = Integer.parseInt(str);
            return _Humidity;
    }
    public void setHumidity(int param0)
    {
        _Humidity = param0;
    }
    public int getTSR()
    {
      String str ="";
            try{

                  FileReader f = new
FileReader("C:\\cygwin\\home\\gollum\\.DataCollector\\Legend" +
moteID + "3.txt");
                  BufferedReader b = new BufferedReader(f);

                        try{
                              str = b.readLine();
                              System.out.println("String= " +
str);
                        }
                        catch (IOException e)
                        {
                        }
            }
            catch (FileNotFoundException e)
            {
            System.err.println("There is no file named like
this...sorry...");
            }
            _TSR = Integer.parseInt(str);
        return _TSR;
    }
    public void setTSR(int param0)
    {
        _TSR = param0;
    }

}
```

Αφού έχουν ολοκληρωθεί τα παραπάνω βήματα δημιουργούμε το WAR αρχείο, απλά τρέχοντας το ant script. Το όνομα του αρχείου αυτού θα είναι ίδιο με τον φάκελο μέσα στον οποίο είναι το ant script. Έχουμε προσθέσει στο ant αρχείο μας να αντιγράφει το WAR αρχείο μετά από την δημιουργία του στον φάκελο του

webapps του Tomcat. Με αυτόν τον τρόπο έχουμε κάνει deploy την εφαρμογή μας στον Tomcat.

Όταν οι εφαρμογές Muse επεκτείνονται πάνω στον Apache Axis2, πρέπει να ταιριάζουν στο πρότυπο υπηρεσιών Axis2. Εντούτοις, επειδή το πρότυπο προγραμματισμού του Muse είναι ανεξάρτητο πλατφόρμας, τα περισσότερα από τα αρχεία και τους καταλόγους του axis2 θα είναι στατικά οποιοδήποτε και να είναι το είδος του resource που εφαρμόζουμε, το πλαίσιο του Muse περιλαμβάνει μια υπηρεσία Axis2 που θα διαχειριστεί όλους τους πόρους που είναι ορισμένοι στο muse.xml. Όταν η μηχανή Axis2 αρχίζει, θα φορτώσει την υπηρεσία Muse, η οποία θα διαβάσει το muse.xml και θα ξεκινήσει τα resources.

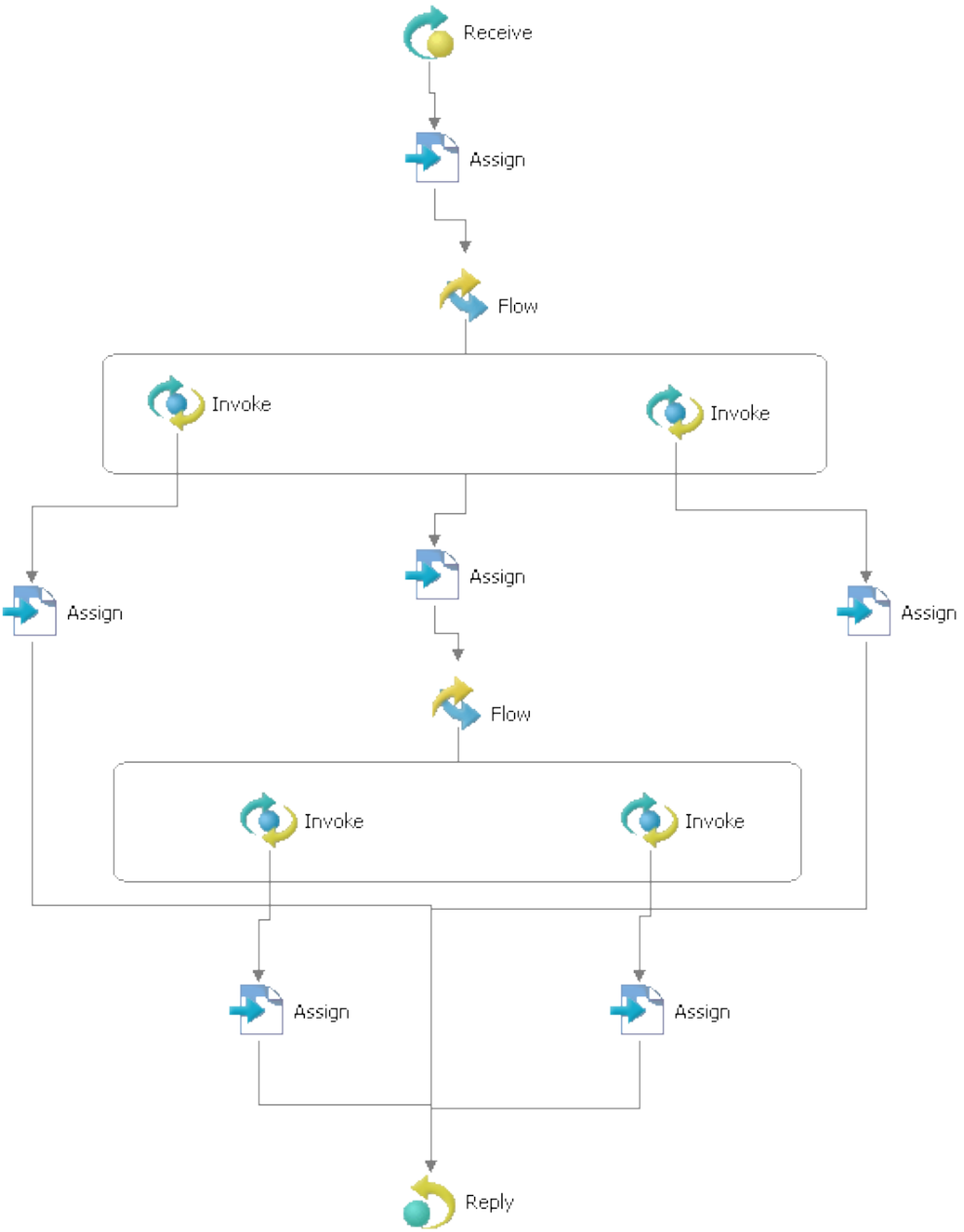
Έτσι μέχρι στιγμής παίρνουμε τις μετρήσεις μέσω του προγράμματος DataCollector και μέσω του Muse έχουμε μία Web service που μπορεί και διαβάζει τα statefull resources μας. Στην συνέχεια θέλουμε να δημιουργήσουμε μία ροή εργασίας χρησιμοποιώντας και το παραπάνω Web service. Αυτό θα το επιτύχουμε αν δημιουργήσουμε ένα BPEL αρχείο. Η διαδικασία αυτή λοιπόν έχει ως εξής.

Όπως έχει ήδη αναφερθεί η διαδικασία δημιουργίας ενός BPEL αρχείου είναι επιρρεπής σε λάθη. Για αυτόν τον λόγο θα χρησιμοποιήσουμε τον ActiveBPEL Designer και θα δημιουργήσουμε μία ροή εργασίας η οποία θα καλεί το ανωτέρω Web services σε δύο μηχανήματα από δύο φορές στο καθένα καθώς θέλουμε να πάρουμε τις μετρήσεις από δύο κόμβους αισθητήρων.

Το γραφικό διάγραμμα που φτιάξαμε για να δημιουργηθεί στον ActiveBpel Designer το BPEL αρχείο φαίνεται παρακάτω. Σε αυτό καλούμε πρώτα δύο φορές το Web service στον πρώτο μηχάνημα ταυτόχρονα μέσα σε μία "flow" συνθήκη, αφού πρώτα αρχικοποιήσουμε τις απαραίτητες μεταβλητές μέσω της copy λειτουργίας του assign. Στην συνέχεια, αφού εκτελεστούν οι δύο παραπάνω κλήσεις, θα καλέσουμε άλλες δύο φορές την ίδια υπηρεσία με διαφορετικό url, για να πάρουμε τις μετρήσεις των αισθητήρων από το δεύτερο μηχάνημα.

Η ροή εκκινείτε λαμβάνοντας ως όρισμα ένα String και μετά από την κλήση κάθε υπηρεσίας τα αποτελέσματα αντιγράφονται στην τελική μεταβλητή της ροής μέσω της λειτουργίας copy του assign.

Παρακάτω φαίνεται το γραφικό αυτό διάγραμμα.



Το αρχείο BPEL που δημιουργείται για το παραπάνω διάγραμμα είναι το εξής:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
BPEL Process Definition
Edited using ActiveBPEL(tm) Designer Version 3.1.0
(http://www.active-endpoints.com)
-->
<bpel:process xmlns:bpel="http://docs.oasis-
open.org/wsbpel/2.0/process/executable"
xmlns:ext="http://www.activebpel.org/2006/09/bpel/extension/query_han
dling" xmlns:ns="http://workflow.sensors.netmode.ntua.gr/xsd"
xmlns:ns1="http://workflow.sensors.netmode.ntua.gr"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
ext:createTargetXPath="yes" name="SensorBusinessProcess"
suppressJoinFailure="yes"
targetNamespace="http://SensorBusinessProcess">
   <bpel:extensions>
      <bpel:extension mustUnderstand="yes"
namespace="http://www.activebpel.org/2006/09/bpel/extension/query_han
dling"/>
   </bpel:extensions>
   <bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
location="WSDL/SensorBusinessProcess.wsdl"
namespace="http://workflow.sensors.netmode.ntua.gr"/>
   <bpel:partnerLinks>
      <bpel:partnerLink myRole="measurementsservice"
name="getMeasurementsLinkType01"
partnerLinkType="ns1:getMeasurementsLinkType01"/>
      <bpel:partnerLink name="SensorWorkflowServiceLinkType01"
partnerLinkType="ns1:SensorWorkflowServiceLinkType01"
partnerRole="sensorWorkflowservice01"/>
   </bpel:partnerLinks>
   <bpel:variables>
      <bpel:variable messageType="ns1:RequestInformation"
name="RequestInformation"/>
      <bpel:variable messageType="ns1:workflowResponse"
name="workflowResponse"/>
      <bpel:variable messageType="ns1:getPropertyMessage"
name="getPropertyMessage01"/>
      <bpel:variable messageType="ns1:getPropertyResponse"
name="getPropertyResponse01"/>
      <bpel:variable messageType="ns1:getPropertyMessage"
name="getPropertyMessage02"/>
      <bpel:variable messageType="ns1:getPropertyResponse"
name="getPropertyResponse02"/>
      <bpel:variable messageType="ns1:getPropertyMessage"
name="getPropertyMessage03"/>
      <bpel:variable messageType="ns1:getPropertyResponse"
name="getPropertyResponse03"/>
      <bpel:variable messageType="ns1:getPropertyMessage"
name="getPropertyMessage04"/>
      <bpel:variable messageType="ns1:getPropertyResponse"
name="getPropertyResponse04"/>
   </bpel:variables>
   <bpel:flow>
      <bpel:links>
```

```
            <bpel:link name="L9"/>
            <bpel:link name="L11"/>
            <bpel:link name="L1"/>
            <bpel:link name="L3"/>
            <bpel:link name="L5"/>
            <bpel:link name="L7"/>
            <bpel:link name="L2"/>
            <bpel:link name="L4"/>
            <bpel:link name="L6"/>
            <bpel:link name="L8"/>
            <bpel:link name="L10"/>
            <bpel:link name="L12"/>
         </bpel:links>
         <bpel:receive createInstance="yes" operation="getMeasurements"
partnerLink="getMeasurementsLinkType01"
portType="ns1:getMeasurementsPortType01"
variable="RequestInformation">
            <bpel:sources>
               <bpel:source linkName="L9"/>
            </bpel:sources>
         </bpel:receive>
         <bpel:reply operation="getMeasurements"
partnerLink="getMeasurementsLinkType01"
portType="ns1:getMeasurementsPortType01" variable="workflowResponse">
            <bpel:targets>
               <bpel:target linkName="L2"/>
               <bpel:target linkName="L4"/>
               <bpel:target linkName="L6"/>
               <bpel:target linkName="L8"/>
            </bpel:targets>
         </bpel:reply>
         <bpel:flow>
            <bpel:targets>
               <bpel:target linkName="L10"/>
            </bpel:targets>
            <bpel:sources>
               <bpel:source linkName="L11"/>
            </bpel:sources>
            <bpel:invoke inputVariable="getPropertyMessage01"
operation="getProperty" outputVariable="getPropertyResponse01"
partnerLink="SensorWorkflowServiceLinkType01"
portType="ns1:SensorWorkflowServicePT01">
               <bpel:sources>
                  <bpel:source linkName="L1"/>
               </bpel:sources>
            </bpel:invoke>
            <bpel:invoke inputVariable="getPropertyMessage02"
operation="getProperty" outputVariable="getPropertyResponse02"
partnerLink="SensorWorkflowServiceLinkType01"
portType="ns1:SensorWorkflowServicePT01">
               <bpel:sources>
                  <bpel:source linkName="L3"/>
               </bpel:sources>
            </bpel:invoke>
         </bpel:flow>
         <bpel:flow>
            <bpel:targets>
               <bpel:target linkName="L12"/>
            </bpel:targets>
            <bpel:invoke inputVariable="getPropertyMessage03"
operation="getProperty" outputVariable="getPropertyResponse03"
```

```
partnerLink="SensorWorkflowServiceLinkType01"
portType="ns1:SensorWorkflowServicePT01">
            <bpel:sources>
               <bpel:source linkName="L5"/>
            </bpel:sources>
         </bpel:invoke>
         <bpel:invoke inputVariable="getPropertyMessage04"
operation="getProperty" outputVariable="getPropertyResponse04"
partnerLink="SensorWorkflowServiceLinkType01"
portType="ns1:SensorWorkflowServicePT01">
            <bpel:sources>
               <bpel:source linkName="L7"/>
            </bpel:sources>
         </bpel:invoke>
      </bpel:flow>
      <bpel:assign>
         <bpel:targets>
            <bpel:target linkName="L1"/>
         </bpel:targets>
         <bpel:sources>
            <bpel:source linkName="L2"/>
         </bpel:sources>
         <bpel:copy>
            <bpel:from part="part1" variable="getPropertyResponse01">
               <bpel:query>ns:return</bpel:query>
            </bpel:from>
            <bpel:to part="response" variable="workflowResponse">
               <bpel:query>ns:local1Temp</bpel:query>
            </bpel:to>
         </bpel:copy>
      </bpel:assign>
      <bpel:assign>
         <bpel:targets>
            <bpel:target linkName="L3"/>
         </bpel:targets>
         <bpel:sources>
            <bpel:source linkName="L4"/>
         </bpel:sources>
         <bpel:copy>
            <bpel:from part="part1" variable="getPropertyResponse02">
               <bpel:query>ns:return</bpel:query>
            </bpel:from>
            <bpel:to part="response" variable="workflowResponse">
               <bpel:query>ns:local2Temp</bpel:query>
            </bpel:to>
         </bpel:copy>
      </bpel:assign>
      <bpel:assign>
         <bpel:targets>
            <bpel:target linkName="L5"/>
         </bpel:targets>
         <bpel:sources>
            <bpel:source linkName="L6"/>
         </bpel:sources>
         <bpel:copy>
            <bpel:from part="part1" variable="getPropertyResponse03">
               <bpel:query>ns:return</bpel:query>
            </bpel:from>
            <bpel:to part="response" variable="workflowResponse">
               <bpel:query>ns:sheep1Temp</bpel:query>
            </bpel:to>
```

```
            </bpel:copy>
         </bpel:assign>
         <bpel:assign>
            <bpel:targets>
               <bpel:target linkName="L7"/>
            </bpel:targets>
            <bpel:sources>
               <bpel:source linkName="L8"/>
            </bpel:sources>
            <bpel:copy>
               <bpel:from part="part1" variable="getPropertyResponse04">
                  <bpel:query>ns:return</bpel:query>
               </bpel:from>
               <bpel:to part="response" variable="workflowResponse">
                  <bpel:query>ns:sheep2Temp</bpel:query>
               </bpel:to>
            </bpel:copy>
         </bpel:assign>
         <bpel:assign>
            <bpel:targets>
               <bpel:target linkName="L9"/>
            </bpel:targets>
            <bpel:sources>
               <bpel:source linkName="L10"/>
            </bpel:sources>
            <bpel:copy>
               <bpel:from>
                  <bpel:literal>
                     <wsa:EndpointReference
xmlns:s="http://workflow.sensors.netmode.ntua.gr"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<wsa:Address
xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">http://s
heep.netmode.ntua.gr:8080/SensorGrid/services/SensorWorkflowService</
wsa:Address>
  <wsa:ServiceName PortName="SensorWorkflowServiceSOAP11port_http01"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">s:Sensor
GridProcess</wsa:ServiceName>
   </wsa:EndpointReference>
                  </bpel:literal>
               </bpel:from>
               <bpel:to partnerLink="SensorWorkflowServiceLinkType01"/>
            </bpel:copy>
            <bpel:copy>
               <bpel:from>
                  <bpel:literal>
                     <ns:getProperty>
   <ns:param0>SensorResource-1</ns:param0>
   <ns:param1>Temperature</ns:param1>
</ns:getProperty>
                  </bpel:literal>
               </bpel:from>
               <bpel:to part="part1" variable="getPropertyMessage01"/>
            </bpel:copy>
            <bpel:copy>
               <bpel:from>
                  <bpel:literal>
                     <ns:getProperty>
   <ns:param0>SensorResource-2</ns:param0>
```

```
   <ns:param1>Temperature</ns:param1>
</ns:getProperty>
                  </bpel:literal>
               </bpel:from>
               <bpel:to part="part1" variable="getPropertyMessage02"/>
            </bpel:copy>
            <bpel:copy>
               <bpel:from>
                  <bpel:literal>
                     <ns:getProperty>
   <ns:param0>SensorResource-1</ns:param0>
   <ns:param1>Temperature</ns:param1>
</ns:getProperty>
                  </bpel:literal>
               </bpel:from>
               <bpel:to part="part1" variable="getPropertyMessage03"/>
            </bpel:copy>
            <bpel:copy>
               <bpel:from>
                  <bpel:literal>
                     <ns:getProperty>
   <ns:param0>SensorResource-2</ns:param0>
   <ns:param1>Temperature</ns:param1>
</ns:getProperty>
                  </bpel:literal>
               </bpel:from>
               <bpel:to part="part1" variable="getPropertyMessage04"/>
            </bpel:copy>
         </bpel:assign>
         <bpel:assign>
            <bpel:targets>
               <bpel:target linkName="L11"/>
            </bpel:targets>
            <bpel:sources>
               <bpel:source linkName="L12"/>
            </bpel:sources>
            <bpel:copy>
               <bpel:from>
                  <bpel:literal>
                     <wsa:EndpointReference
xmlns:s="http://workflow.sensors.netmode.ntua.gr"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<wsa:Address
xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">http://l
ocalhost:8080/SensorGrid/services/SensorWorkflowService</wsa:Address>
  <wsa:ServiceName PortName="SensorWorkflowServiceSOAP11port_http01"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">s:Sensor
GridProcess</wsa:ServiceName>
   </wsa:EndpointReference>
                  </bpel:literal>
               </bpel:from>
               <bpel:to partnerLink="SensorWorkflowServiceLinkType01"/>
            </bpel:copy>
         </bpel:assign>
   </bpel:flow>
</bpel:process>
```

Το WSDL αρχείο από το οποίο δημιουργήσαμε το BPEL είναι το ακόλουθο:

```
<wsdl:definitions
xmlns:axis2="http://workflow.sensors.netmode.ntua.gr"
      xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
      xmlns:ns0="http://workflow.sensors.netmode.ntua.gr/xsd"
      xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
      xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
      xmlns:ns1="http://org.apache.axis2/xsd"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
      targetNamespace="http://workflow.sensors.netmode.ntua.gr">
      <wsdl:documentation>SensorWorkflowService</wsdl:documentation>
      <wsdl:types>
            <xs:schema

      xmlns:ns="http://workflow.sensors.netmode.ntua.gr/xsd"
                  attributeFormDefault="qualified"
elementFormDefault="qualified"

      targetNamespace="http://workflow.sensors.netmode.ntua.gr/xsd">
                  <xs:element name="getProperty">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="param0"
nillable="true"
                                          type="xs:string" />
                                    <xs:element name="param1"
nillable="true"
                                          type="xs:string" />
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>
                  <xs:element name="getPropertyResponse">
                        <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="return"
nillable="true"
                                          type="xs:int" />
                              </xs:sequence>
                        </xs:complexType>
                  </xs:element>

                  <xs:element name="workflowResponsElement">
                        <xs:complexType>
                              <xs:all>
                                    <xs:element name="local1Temp"
nillable="true"
                                          type="xs:int" />
                                    <xs:element name="local2Temp"
nillable="true"
                                          type="xs:int" />
                                    <xs:element name="sheep1Temp"
nillable="true"
                                          type="xs:int" />
                                    <xs:element name="sheep2Temp"
nillable="true"
```

```
                                                type="xs:int" />
                                    </xs:all>
                              </xs:complexType>
                        </xs:element>

                  </xs:schema>
            </wsdl:types>

            <wsdl:message name="getPropertyMessage">
                  <wsdl:part name="part1" element="ns0:getProperty" />
            </wsdl:message>

            <wsdl:message name="getPropertyResponse">
                  <wsdl:part name="part1" element="ns0:getPropertyResponse"
/>
            </wsdl:message>

            <wsdl:message name="RequestInformation">
                  <wsdl:part name="start" type="xs:string" />
            </wsdl:message>

            <wsdl:message name="workflowResponse">
                  <wsdl:part name="response"
element="ns0:workflowResponsElement"/>
            </wsdl:message>

            <wsdl:portType name="SensorWorkflowServicePT01">
                  <wsdl:operation name="getProperty">
                        <wsdl:input

            xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
                              message="axis2:getPropertyMessage"
wsaw:Action="urn:getProperty" />
                        <wsdl:output message="axis2:getPropertyResponse" />
                  </wsdl:operation>
            </wsdl:portType>



            <!-- <wsdl:portType name="SensorWorkflowServicePT02">
                  <wsdl:operation name="getProperty">
                        <wsdl:input

            xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
                              message="axis2:getPropertyMessage"
wsaw:Action="urn:getProperty" />
                        <wsdl:output message="axis2:getPropertyResponse" />
                  </wsdl:operation>
            </wsdl:portType>
             -->
            <wsdl:portType name="getMeasurementsPortType01">
                  <wsdl:operation name="getMeasurements">
                        <wsdl:input message="axis2:RequestInformation" />
                        <wsdl:output message="axis2:workflowResponse" />
                  </wsdl:operation>
            </wsdl:portType>

            <wsdl:binding name="SensorWorkflowServiceSOAP11Binding01"
                  type="axis2:SensorWorkflowServicePT01">
                  <soap:binding
transport="http://schemas.xmlsoap.org/soap/http"
```

```
                  style="document" />
            <wsdl:operation name="getProperty">
                  <soap:operation soapAction="urn:getProperty"
                        style="document" />
                  <wsdl:input>
                        <soap:body use="literal" />
                  </wsdl:input>
                  <wsdl:output>
                        <soap:body use="literal" />
                  </wsdl:output>
            </wsdl:operation>
      </wsdl:binding>
      <wsdl:binding name="SensorWorkflowServiceSOAP12Binding01"
            type="axis2:SensorWorkflowServicePT01">
            <soap12:binding
transport="http://schemas.xmlsoap.org/soap/http"
                  style="document" />
            <wsdl:operation name="getProperty">
                  <soap12:operation soapAction="urn:getProperty"
                        style="document" />
                  <wsdl:input>
                        <soap12:body use="literal" />
                  </wsdl:input>
                  <wsdl:output>
                        <soap12:body use="literal" />
                  </wsdl:output>
            </wsdl:operation>
      </wsdl:binding>

      <!-- <wsdl:binding name="SensorWorkflowServiceSOAP11Binding02"
            type="axis2:SensorWorkflowServicePT02">
            <soap:binding
transport="http://schemas.xmlsoap.org/soap/http"
                  style="document" />
            <wsdl:operation name="getProperty">
                  <soap:operation soapAction="urn:getProperty"
                        style="document" />
                  <wsdl:input>
                        <soap:body use="literal" />
                  </wsdl:input>
                  <wsdl:output>
                        <soap:body use="literal" />
                  </wsdl:output>
            </wsdl:operation>
      </wsdl:binding>
      <wsdl:binding name="SensorWorkflowServiceSOAP12Binding02"
            type="axis2:SensorWorkflowServicePT02">
            <soap12:binding
transport="http://schemas.xmlsoap.org/soap/http"
                  style="document" />
            <wsdl:operation name="getProperty">
                  <soap12:operation soapAction="urn:getProperty"
                        style="document" />
                  <wsdl:input>
                        <soap12:body use="literal" />
                  </wsdl:input>
                  <wsdl:output>
                        <soap12:body use="literal" />
                  </wsdl:output>
            </wsdl:operation>
      </wsdl:binding>
```

```
        -->
      <wsdl:service name="SensorGridProcess">
            <wsdl:port name="SensorWorkflowServiceSOAP11port_http01"

      binding="axis2:SensorWorkflowServiceSOAP11Binding01">
                  <soap:address

      location="http://localhost:8080/SensorGrid/services/SensorWorkf
lowService" />
            </wsdl:port>
            <wsdl:port name="SensorWorkflowServiceSOAP12port_http01"

      binding="axis2:SensorWorkflowServiceSOAP12Binding01">
                  <soap12:address

      location="http://localhost:8080/SensorGrid/services/SensorWorkf
lowService" />
            </wsdl:port>

            <!-- <wsdl:port
name="SensorWorkflowServiceSOAP11port_http02"

      binding="axis2:SensorWorkflowServiceSOAP11Binding02">
                  <soap:address

      location="http://sheep.netmode.ntua.gr:8080/SensorGrid/services
/SensorWorkflowService" />
            </wsdl:port>
            <wsdl:port name="SensorWorkflowServiceSOAP12port_http02"

      binding="axis2:SensorWorkflowServiceSOAP12Binding02">
                  <soap12:address

      location="http://sheep.netomde.ntua.gr:8080/SensorGrid/services
/SensorWorkflowService" />
            </wsdl:port>
             -->
      </wsdl:service>


      <plnk:partnerLinkType name="SensorWorkflowServiceLinkType01">
            <plnk:role name="sensorWorkflowservice01"
                  portType="axis2:SensorWorkflowServicePT01" />
      </plnk:partnerLinkType>

      <!-- <plnk:partnerLinkType
name="SensorWorkflowServiceLinkType02">
            <plnk:role name="sensorWorkflowservice02"
                  portType="axis2:SensorWorkflowServicePT02" />
      </plnk:partnerLinkType>
       -->
      <plnk:partnerLinkType name="getMeasurementsLinkType01">
            <plnk:role name="measurementsservice"
                  portType="axis2:getMeasurementsPortType01" />
      </plnk:partnerLinkType>

</wsdl:definitions>
```

Στην συνέχεια δημιουργούμε τον deployment descriptor και κάνουμε deploy την ροή στον φάκελο "bpr" του Tomcat. Από την σελίδα του BpelAdmin της ActiveBpel Engine μπορούμε να βεβαιωθούμε ότι έχει γίνει σωστά το deploy και αν όχι να ελέγξουμε τα λάθη στο link του log. Ο deployment descriptor φαίνεται παρακάτω:

```
<?xml version="1.0" encoding="UTF-8"?>
<process xmlns="http://schemas.active-
endpoints.com/pdd/2006/08/pdd.xsd"
xmlns:bpelns="http://SensorBusinessProcess"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
location="bpel/SensorBusinessProcess/SensorBusinessProcess.bpel"
name="bpelns:SensorBusinessProcess">
   <partnerLinks>
      <partnerLink name="SensorWorkflowServiceLinkType01">
         <partnerRole endpointReference="dynamic"
invokeHandler="default:Address"/>
      </partnerLink>
      <partnerLink name="getMeasurementsLinkType01">
         <myRole allowedRoles="" binding="MSG"
service="getMeasurementsLinkType01Service"/>
      </partnerLink>
   </partnerLinks>
   <references>
      <wsdl
location="project:/SensorBusinessProcess/WSDL/SensorBusinessProcess.w
sdl" namespace="http://workflow.sensors.netmode.ntua.gr"/>
   </references>
</process>
```

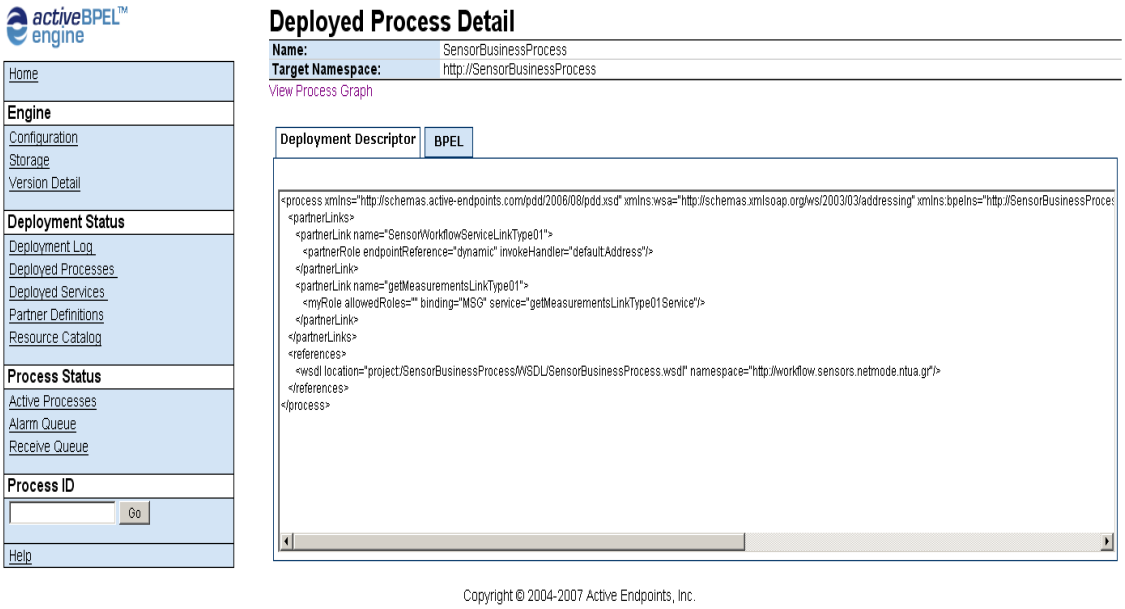
Για να ενεργοποιήσουμε την ροή εργασίας που δημιουργήσαμε πρέπει να την καλέσουμε μέσω ενός προγράμματος πελάτη. Αυτό φαίνεται παρακάτω:

```
package gr.ntua.netmode.sensors.workflow.sequential.client;

import
gr.ntua.netmode.sensors.workflow.GetMeasurementsLinkType01ServiceBind
ingStub;
import
gr.ntua.netmode.sensors.workflow.GetMeasurementsLinkType01ServiceLoca
tor;
import gr.ntua.netmode.sensors.workflow.xsd.WorkflowResponsElement;

public class SequentialWorkflowClient {

      /**
       * @param args
       */
      public static void main(String[] args) {

            try {
                  GetMeasurementsLinkType01ServiceLocator locator =
new GetMeasurementsLinkType01ServiceLocator();
                  GetMeasurementsLinkType01ServiceBindingStub stub =
(GetMeasurementsLinkType01ServiceBindingStub) locator
```

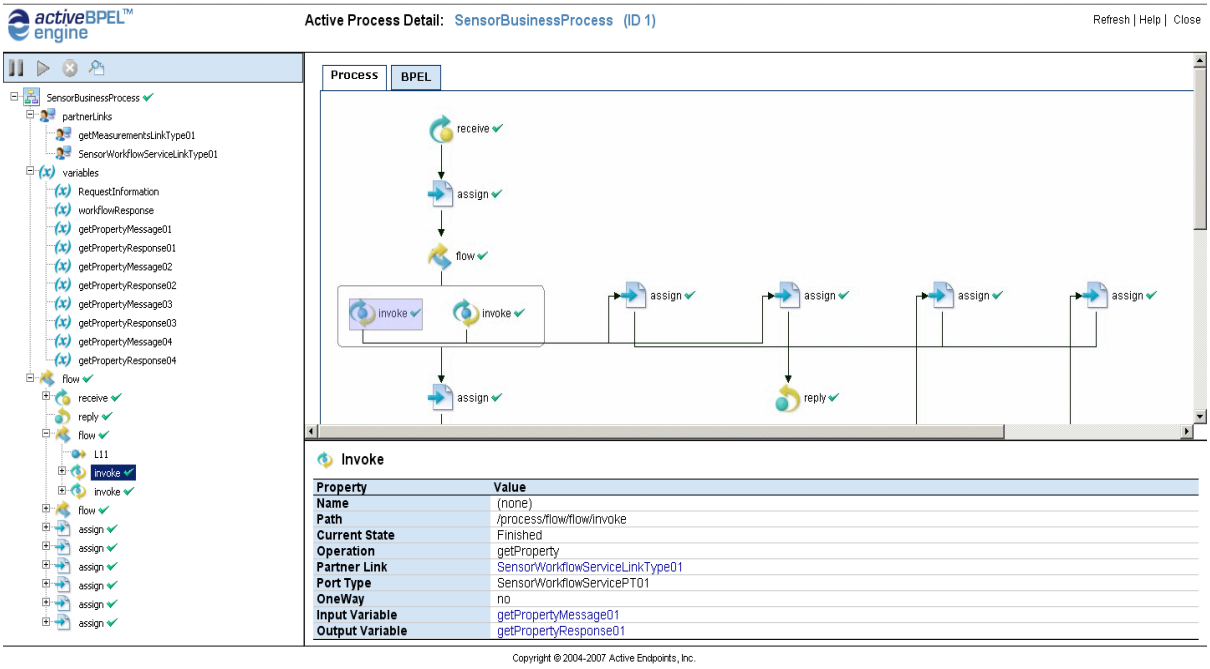
```
        .getgetMeasurementsLinkType01ServicePort();

                    WorkflowResponsElement workflowRE =
stub.getMeasurements("start");

                    System.out.println("Local1Temp " +
workflowRE.getLocal1Temp());
                    System.out.println("Local2Temp " +
workflowRE.getLocal2Temp());
                    System.out.println("Sheep1Temp " +
workflowRE.getSheep1Temp());
                    System.out.println("Sheep2Temp " +
workflowRE.getSheep2Temp());

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

Με αυτόν τον τρόπο τρέχουμε την ροή εργασίας που δημιουργήσαμε και παίρνουμε τα αποτελέσματα που ζητήσαμε.

Παρακάτω φαίνεται η διασύνδεση της κονσόλας της ActiveBPEL Engine των deployed διαδικασιών:



Ενώ στο ακόλουθο σχήμα απεικονίζεται η διασύνδεση της Active Bpel Engine για μία ενεργή διαδικασία, η οποία έχει εκτελεστεί επιτυχώς.



ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] www.apache.org/wsrf (WSRF)

[2] http://docs.oasis-open.org/wsrf/ (WSRF)

[3] http://developer.capeclear.com (BPEL)

[4] http://www.active-endpoints.com/ (BPEL)

[5] http://www.moteiv.com/products-tmotesky.php (tmotes)

[6] http://www.ceid.upatras.gr/courses/katanemhmena/wiki (TinyOS, nesC)

[7] www.wikipedia.com (Ad-hoc networks)

[8] http://w3.antd.nist.gov/wahn_ssn.shtml (Ad-hoc networks)

[9] www.it.uom.gr/project/ (WSDL)

[10] [E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, 2001] Web Services Description Language (WSDL) 1.1 (WSDL)

[11] http://en.wikipedia.org/wiki/Grid_computing (Grids)

[12] http://gridcafe.web.cern.ch/gridcafe/Gridhistory/gridlike.html (Grids)

[13] http://ws.apache.org/axis/ (Axis)

[14] http://ws.apache.org/juddi/ (JUDDI)

[15] www.descriptor.com (JUDDI)

[16] Peter Boon, 2004: Accessing resources using Web services technology (Compares OGSI and WSRF) (Web services)

[17] Steve Graham, Simeon Simeonov, Toufic Boubez, Doug Davis, Glen Daniels, Yuichi Nakamura, Ryo Neyama: Building Web Services With Java, Making Sense of XML, SOAP, WSDL and UDDI (Web services)

[18] *[Fremantle, P., Weerawarana, S., and Khalaf, R. Enterprise Services, Communications of the ACM, October 2002/Vol.45.No 10, pp.77-82.]* (Web services)

[19] http://www.go-online.gr/ (Web services)

[20] IBM: Self-Study Guide – Websphere Studio Application Developer And Web Services, ibm.com/redbooks

[21]Ethan Cerami: Web Services Essentials, O'Reilly, 2002