ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Τομέας Επικοινωνιών, Πληροφορικής και Συστημάτων Ηλεκτρονικής
Εργαστήριο Διαχείρισης και Βέλτιστου Σχεδιασμού Δικτύων

ΑΡΧΙΤΕΚΤΟΝΙΚΗ SOA ΜΕ ΧΡΗΣΗ JINI ΚΑΙ JMX ΜΕ ΕΦΑΡΜΟΓΗ ΣΕ ΔΙΑΧΕΙΡΙΣΗ ΔΙΚΤΥΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΧΡΙΣΤΟΦΟΡΟΣ Α. ΧΡΙΣΤΟΦΟΡΟΥ

Επιβλέπων: Βασίλης Μάγκλαρης
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2004

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Τομέας Επικοινωνιών, Πληροφορικής και Συστημάτων Ηλεκτρονικής
Εργαστήριο Διαχείρισης και Βέλτιστου Σχεδιασμού Δικτύων

ΑΡΧΙΤΕΚΤΟΝΙΚΗ SOA ΜΕ ΧΡΗΣΗ JINI ΚΑΙ JMX ΜΕ ΕΦΑΡΜΟΓΗ ΣΕ ΔΙΑΧΕΙΡΙΣΗ ΔΙΚΤΥΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΧΡΙΣΤΟΦΟΡΟΣ Α. ΧΡΙΣΤΟΦΟΡΟΥ

Επιβλέπον: Βασίλης Μάγκλαρης
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από τριμελή εξεταστική επιτροπή την .....

....

Αθήνα, Οκτώβριος 2004

.....
Χριστόφορος Α. Χριστοφόρου

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

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1
ΕΙΣΑΓΩΓΗ

Εδώ και αρκετά χρόνια, ο τομέας της δικτύωσης υπολογιστών και άλλων συσκευών, έχει βιώσει σημαντικές τεχνολογικές αλλαγές. Η συνεχής αύξηση στην ταχύτητα μετάδοσης δεδομένων, η ασταμάτητη εξάπλωση του παγκόσμιου ιστού (Internet) σε κάθε γωνιά του πλανήτη (ενσύρματα και ασύρματα), έχουν δημιουργήσει νέες ανάγκες μα και έχουν μεγαλώσει τις απαιτήσεις όσο αφορά την ασφάλεια, την αξιοπιστία και την ευστάθεια των συστημάτων.

Η συνεχής αύξηση στην απαίτηση υπολογιστικής ισχύος στις σύγχρονες εφαρμογές, καθώς και ο γιγαντισμός που χαρακτηρίζει τα σημερινά πληροφοριακά συστήματα και δίκτυα έχουν οδηγήσει σε αποκεντρωμένες προσεγγίσεις ανάλυσης και σχεδιασμού συστημάτων, όπου ο φόρτος εργασίας, ή η ευθύνη επόπτευσης, κατανέμεται σε ένα σύνολο γεωγραφικά διασκορπισμένων υπολογιστών και συσκευών. Οι υπολογιστές αυτοί που επικοινωνούν μεταξύ τους μέσω δικτύου, χρειάζονται ευέλικτους μηχανισμούς συντονισμού των εργασιών που εκτελούν παράλληλα. Επίσης γίνεται πλέον επιτακτική η ανάγκη για όσο το δυνατόν χαλαρότερη εξάρτηση μεταξύ των συστατικών μερών που συμμετέχουν, ώστε το όλο σύστημα να μην κινδυνεύει να αδρανοποιηθεί από το ενδεχόμενο απώλειας ενός μικρού υποσυστήματος.

Η διπλωματική αυτή εργασία, εξετάζει ένα σύνολο από τεχνολογίες και προσεγγίσεις σχεδιασμού κατανεμημένων συστημάτων, όσο αφορά την διαχείριση και την εκμετάλλευσή τους. Το μεγαλύτερο μέρος της ανάλυσης επικεντρώνεται σε τεχνολογίες που αφορούν την επικοινωνία μεταξύ συνεργαζόμενων οντοτήτων στο δίκτυο, και στην πλήρωση αναγκών που προέκυψαν ως συνέπεια της αύξησης της πολυπλοκότητας και του μεγέθους των σύγχρονων κατανεμημένων συστημάτων. Τέτοιες ανάγκες είναι η δυνατότητα δυναμικής προσθαφαίρεσης ή και αντικατάστασης τμημάτων (modules) του συστήματος, καθώς και η δυνατότητα της εκ του μακρόθεν αναβάθμισής τους μέσω δικτύου, χωρίς την ανάγκη της επιτόπου παρουσίας του διαχειριστή.

Σε επίπεδο υλοποίησης, από την παρούσα εργασία, ζητήθηκε η διασύνδεση πρακτόρων JMX, μέσω ενός συστήματος Jini, για την προσθήκη της δυνατότητας δυναμικής προσθαφαίρεσης πρακτόρων σε ένα κατανεμημένο σύστημα διαχείρισης δικτύων. Επίσης ζητήθηκε η υλοποίηση μεθόδου που να επιτρέπει την αποστολή κώδικα, από τον απομακρυσμένο διαχειριστή, στον πράκτορα, ώστε να μπορεί να αναβαθμίζει τις υπηρεσίες που προσφέρει στον διαχειριστή, χωρίς την ανάγκη τοπικής ρύθμισης του πράκτορα και χειρωνακτικής φόρτωσης κώδικα και βιβλιοθηκών.

Η αντιμετώπιση των πιο πάνω ζητημάτων έγινε με γενικό τρόπο, και δεν έχουν περιοριστεί στο ζήτημα της διαχείρισης δικτύων. Είναι δεδομένη η δυνατότητα χρήσης των εργαλείων που έχουν υλοποιηθεί και για την περίπτωση της διαχείρισης.

Για να καταστεί δυνατός ο έλεγχος της ορθής λειτουργίας, μα και η επίδειξη της λειτουργίας των εργαλείων που έχουν υλοποιηθεί, έχει κατασκευαστεί μια εφαρμογή πελάτης με ένα γραφικό περιβάλλον, που ασχολείται με την δυναμική ανακάλυψη πρακτόρων, την φόρτωση και παύση υπηρεσιών στους πράκτορες, καθώς και με την προσθήκη κώδικα από απομακρυσμένη τοποθεσία μέσω δικτύου.

Το παρόν κείμενο είναι δομημένο σε κεφάλαια ως εξής:

- **Κεφάλαιο 1**: Η παρούσα εισαγωγή
- **Κεφάλαιο 2**: Στο κεφάλαιο αυτό παρουσιάζονται οι τεχνολογίες JMX, JMX Remote API και JINI. Επίσης, επιχειρείται μια παρουσίαση των Service Oriented Αρχιτεκτονικών καθώς και των τεχνολογιών που εμπλέκονται σε αυτές. Οι Service Oriented Αρχιτεκτονικές αποτελούν μια εξέλιξη όσο αφορά την ανάλυση και τον σχεδιασμό κατανεμημένων συστημάτων.
- **Κεφάλαιο 3**: Στο κεφάλαιο αυτό γίνεται περιγραφή της αρχιτεκτονικής της εφαρμογής που έχει υλοποιηθεί καθώς και των δυνατοτήτων των συστατικών μερών της.
- **Κεφάλαιο 4**: Στο κεφάλαιο αυτό δίδονται τα τεχνικά χαρακτηριστικά των μερών που υλοποιήθηκαν καθώς και οι προϋποθέσεις για την λειτουργία τους.
- **Κεφάλαιο 5**: Στο κεφάλαιο αυτό γίνεται περιγραφή της εφαρμογής πελάτη που αναπτύχθηκε για έλεγχο και παρουσίαση του συστήματος.

ΚΕΦΑΛΑΙΟ 2
ΕΠΙΣΚΟΠΗΣΗ ΤΕΧΝΟΛΟΓΙΩΝ

2.1 Java Management Extensions

2.1.1 Εισαγωγή

Η τεχνολογία JMX, στη σημερινή της μορφή, αποτελεί την υλοποίηση των προτάσεων/προδιαγραφών που έχει θέσει η κοινότητα ανάπτυξης λογισμικού με τη χρήση τεχνολογίας JAVA για την ύπαρξη μιας ενιαίας πλατφόρμας διαχείρισης και δημιουργίας εφαρμογών διαχείρισης δικτύων, εφαρμογών και πόρων (hardware και software), με ένα τυποποιημένο και συγκεντρωτικό τρόπο. Το πρώτο μέρος των προδιαγραφών αυτών έχει οριστικοποιηθεί τον Οκτώβριο του 2002 και έχει κωδικοποιηθεί με την επωνυμία JSR 3 (Java Spesification Request 3). Το δεύτερο μέρος, οριστικοποιήθηκε τον Νοέμβριο του 2003 και έχει κωδικοποιηθεί με την επωνυμία JSR 160. Οι προδιαγραφές καθορίζουν μια σειρά από APIs (Application Programming Interfaces) τα οποία διαχωρίζουν τις λειτουργίες διαχείρισης σε επίπεδα δίνοντας τη δυνατότητα να συνδυαστεί η τεχνολογία JMX με υπάρχουσες τεχνολογίες διαχείρισης (όχι απαραίτητα συνδεδεμένες με την γλώσσα JAVA).

Η αρχιτεκτονική JMX μπορεί να διαιρεθεί σε τρία επίπεδα:

- Επίπεδο υπηρεσιών (Instrumentation Level)
- Επίπεδο πράκτορα – αντιπροσώπου (Agent Level)
- Επίπεδο κατανεμημένων υπηρεσιών (Distributed Services Level)

Το επίπεδο των κατανεμημένων υπηρεσιών (JSR 160) είναι πιο σύνθετο και συνδυάζει μια σειρά από άλλες τεχνολογίες κατανεμημένων συστημάτων και πρωτόκολλα διαχείρισης. Η χρήση διαδεδομένων και καλά εμπεδωμένων τεχνολογιών (π.χ. RMI), πρωτοκόλλων επικοινωνίας (π.χ. HTTP) καθώς και πρωτοκόλλων διαχείρισης (π.χ. SNMP) μας δίνει τη δυνατότητα να δημιουργήσουμε ένα δυναμικό περιβάλλον

κατανεμημένης – παράλληλης επεξεργασίας. Σε συνδυασμό με νέες ανερχόμενες τεχνολογίες (π.χ. JINI) μπορούν να κατασκευαστούν επεκτατά συστήματα, ανθεκτικά στην συνεχείς μεταβολές ενός δυναμικού δικτύου. Η δυνατότητα αυτή θα εξεταστεί στα πλαίσια αυτής της διπλωματικής εργασίας.

Πιο κάτω γίνεται μια παρουσίαση της αρχιτεκτονικής JMX, καθώς και περιγραφή των επιπέδων που την συναποτελούν, με ιδιαίτερη έμφαση στο επίπεδο κατανεμημένων υπηρεσιών.

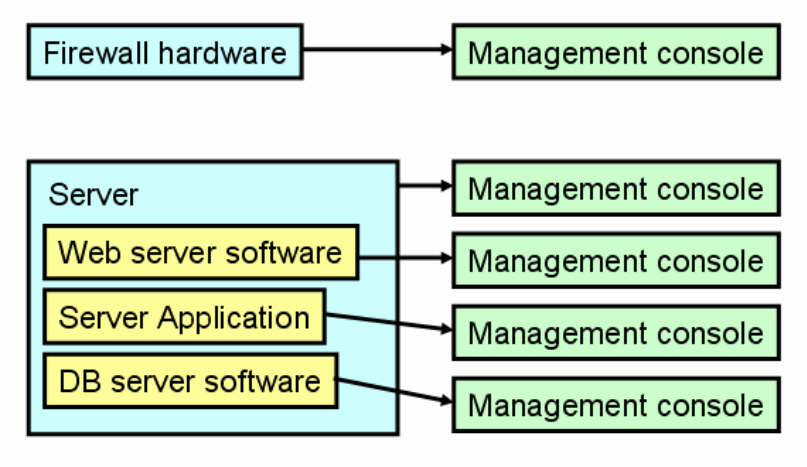
2.1.2 Πλεονεκτήματα της αρχιτεκτονικής JMX

Η τεχνολογία JMX προσφέρει μια ευέλικτη αρχιτεκτονική για την υλοποίηση κατανεμημένου λογισμικού διαχείρισης, μέσω ενός συστήματος πρακτόρων (agents) και αντιπροσώπων (proxies). Μέσω ενός δυναμικού σχήματος ανάπτυξης με τυποποιημένες πλέον διεπαφές (interfaces) προσφέρει τα μέσα για την διαχείριση εφαρμογών ή οντοτήτων JAVA, τη δημιουργία μεσισμικού διαχείρισης (middleware) αλλά και την ενσωμάτωση αυτών των εργαλείων σε υφιστάμενα συστήματα διαχείρισης. Σημαντικά χαρακτηριστικά είναι:

- Ένας πράκτορας JMX μπορεί να λειτουργήσει σχεδόν σε οποιαδήποτε συσκευή η οποία υποστηρίζει την τεχνολογία JAVA. Επίσης, η κάθε εφαρμογή JAVA, μπορεί με ελάχιστες μετατροπές (ουσιαστικά μια μικρή επέκταση) να «αποκαλύψει» μερικά από τα χαρακτηριστικά (attributes) και τη λειτουργικότητά της (methods) σαν ένα ή περισσότερα MBeans, εγγεγραμμένα σε ένα JMX πράκτορα. Συνολικά η επιβάρυνση μεταφράζεται στην ενσωμάτωση ενός JMX πράκτορα και στην ενθυλάκωση (wrapping) του κάθε διαχειριζόμενου πόρου σε ένα MBean (επίπεδο υπηρεσιών).
- Η τεχνολογία JMX παρέχει μια τυποποιημένη διεπαφή (interface) για τη διαχείριση συστημάτων, δικτύων και οντοτήτων JAVA. Ο JMX πράκτορας φανερώνει ένα συγκεκριμένο API μέσω του οποίου οι διαχειριστές αποκτούν πρόσβαση στους υπό διαχείριση πόρους. Αυτό σημαίνει ότι, ανεξαρτήτως κατασκευαστή, τεχνολογίας και πρωτοκόλλου επικοινωνίας του υπό διαχείριση

πόρου, ο διαχειριστή επικοινωνεί με το κάθε MBean (το οποίο δρα ως αντιπρόσωπος), μέσω της ίδιας διεπαφής. Αυτό επιτρέπει την ενσωμάτωση (μα και προσθαφαίρεση) σε ένα κέντρο διαχείρισης, πόρων διαφορετικής φύσης και τεχνολογίας (π.χ. μια εφαρμογή μπορεί να διαχειριστεί ένα router και ένα web server μέσω του ιδίου περιβάλλοντος διαχείρισης).

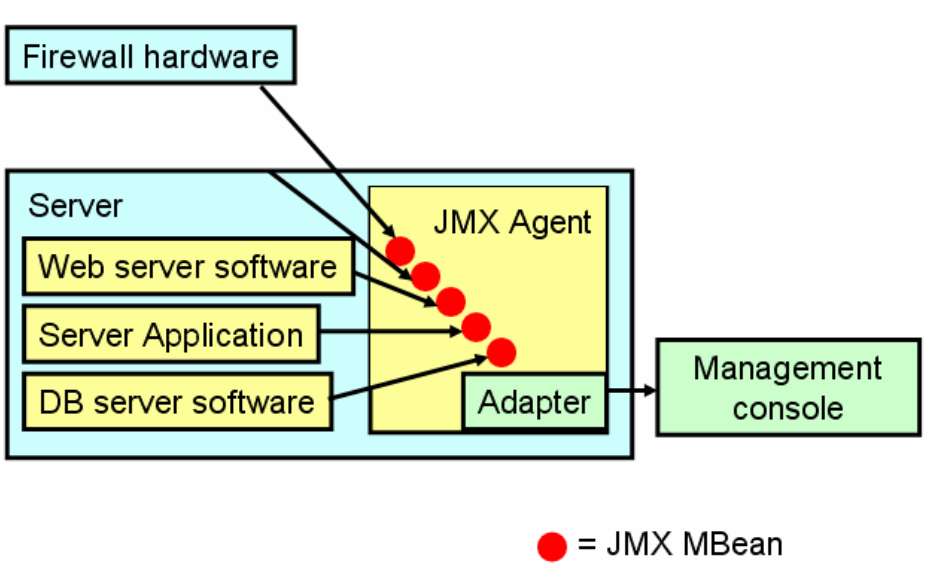
- Παρέχεται μια δυναμική και επεκτατή αρχιτεκτονική διαχείρισης, αφού δίνεται η δυνατότητα της προσθαφαίρεσης υπηρεσιών σε κάποιο πράκτορα JMX καθώς και η επέκταση της πλατφόρμας διαχείρισης με νέους πράκτορες, ανάλογα με τις ανάγκες που προκύπτουν. Η δυνατότητα προσθαφαίρεσης λειτουργικών τμημάτων (modules), εκτός από τους σχεδιαστές των εφαρμογών, επεκτείνονται και στους τελικούς χρήστες αφού η φόρτωση και ο τερματισμός υπηρεσιών μπορεί να γίνει σε πραγματικό χρόνο.



Σχήμα 2.1.1

Με τον όρο διαχειριζόμενοι πόροι αναφερόμαστε σε μια σειρά από στοιχεία τα οποία συναντούμε σε συστήματα και δίκτυα υπολογιστών. Το σύνολο αυτό περιλαμβάνει συστήματα υλικού (hardware – υπολογιστές, routers, switches, εκτυπωτές, αισθητήρες κ.τ.λ.) και συστήματα λογισμικού (εφαρμογές, λειτουργικά συστήματα, βάσεις δεδομένων, εξυπηρετητές – π.χ. web servers κ.τ.λ). Μέχρι στιγμής η κάθε απομακρυσμένη διαχειριζόμενη οντότητα, διέθετε ιδιόκτητη διεπαφή διαχείρισης καθώς και ιδιόκτητη κονσόλα διαχείρισης που υλοποιούσε αυτή τη διεπαφή. Με την τεχνολογία

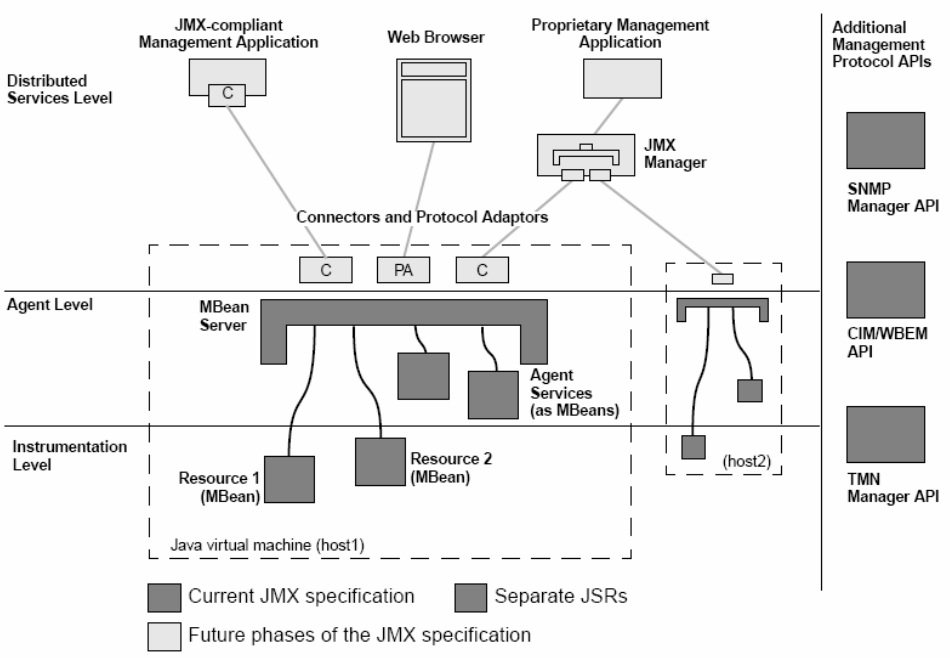
JMX πολλές διαφορετικές οντότητες μπορούν να τύχουν διαχείρισης από την ίδια κονσόλα. Η μεταβολή αυτή φαίνεται στα σχήματα Σχ.2.1.1 και Σχ.2.1.2



Σχ. 2.1.2

2.1.3 Επισκόπηση της αρχιτεκτονικής JMX

Όπως έχει αναφερθεί πιο πάνω, η αρχιτεκτονική JMX μπορεί να διαιρεθεί σε τρία επίπεδα. Η λειτουργική σημασία του κάθε επιπέδου αναλύεται πιο κάτω με τη βοήθεια του Σχ. 2.1.3:



Σχ.2.1.3

- **Επίπεδο υπηρεσιών (Instrumentation Level)**

  Σε αυτό το επίπεδο βρίσκονται τα MBeans. Τα MBeans (Managed Beans) είναι οντότητες JAVA οι οποίες δρουν ως αντιπρόσωποι των υπό διαχείριση πόρων (εφαρμογές, συσκευές, υπηρεσίες). Τα MBeans φανερώνουν στα ανώτερα επίπεδα τις διεπαφές μέσω των οποίων ο διαχειριστής μπορεί να επικοινωνήσει με τους υπό διαχείριση πόρους. Καθορίζουν τα χαρακτηριστικά (attributes) τα οποία μπορεί να διαβάσει και να μεταβάλει καθώς και τις λειτουργίες (methods) τις οποίες μπορεί να χρησιμοποιήσει ο (απομακρυσμένος συνήθως) διαχειριστής.

- **Επίπεδο πράκτορα – αντιπροσώπου (Agent Level)**

  Σε αυτό το επίπεδο βρίσκεται ο πυρήνας ενός JMX πράκτορα, ο οποίος είναι ο εξυπηρετητής των MBeans (MBean Server). Τα MBeans (δηλαδή οι υπό

διαχείριση πόροι), εγγράφονται στον εξυπηρετητή, ο οποίος με τη σειρά του δίνει τη δυνατότητα στους διαχειριστές να εκτελέσουν εργασίες πάνω σε αυτά. Ο εξυπηρετητής JMX διαθέτει μια σειρά από υπηρεσίες για την διαχείριση των MBeans, οι οποίες μπορούν να προσθαφαιρεθούν δυναμικά από το χρήστη αφού και αυτές με τη σειρά τους εγγράφονται στον εξυπηρετητή σαν διαχειριζόμενα Beans! Αυτές οι υπηρεσίες περιλαμβάνουν την παρακολούθηση χαρακτηριστικών των MBeans (monitor service), τη δημιουργία σχέσεων μεταξύ των εγγεγραμμένων MBeans (relation service), ασύγχρονη δημιουργία γεγονότων (timer service) και την διάδοση ειδοποιήσεων μεταξύ των MBeans και του διαχειριστή (notification service). Άλλη μια σημαντική υπηρεσία είναι η υπηρεσία δυναμικής φόρτωσης κλάσεων με την προσθήκη βιβλιοθηκών από το δίκτυο με δυναμικό τρόπο (m-let service). Αυτή η υπηρεσία εξετάζεται σχολαστικότερα στα πλαίσια αυτής της εργασίας.

- **Επίπεδο κατανεμημένων υπηρεσιών (Distributed Services Level)**
  Αυτό το επίπεδο περιλαμβάνει όλες τις μεθόδους με τις οποίες μπορεί μια εφαρμογή διαχείρισης να επικοινωνήσει με τον MBean Server και να χρησιμοποιήσει τις λειτουργίες του. Αυτή η επικοινωνία γίνεται με μια σειρά από πρωτόκολλα τα οποία προσφέρουν τη δυνατότητα χρήσης του JMX πράκτορα (δηλ. του MBean Server) από τοπικές και απομακρυσμένες εφαρμογές όχι απαραίτητα γραμμένες σε JAVA ή συμβατές με άλλα χαρακτηριστικά του πράκτορα (π.χ. πλατφόρμα hardware, λειτουργικό σύστημα κ.τ.λ.) .

2.1.4 Χειρισμός (Instrumentation) Πόρων με τη χρήση MBeans

Για να επιτευχθεί η διαχείριση πόρων χρησιμοποιώντας την τεχνολογία JMX, χρειάζεται πρώτα να γίνει δυνατή η επικοινωνία αυτών με τη γλώσσα JAVA. Η πρόσβαση σε αυτούς τους πόρους και η δυνατότητα χρήσης τους, υλοποιείται με αντικείμενα τα οποία είναι γνωστά ως MBeans (Managed Beans). Τα MBeans είναι αντικείμενα JAVA τα οποία ακολουθούν συγκεκριμένο προγραμματιστικό μοτίβο και υλοποιούν μια σειρά από

διεπαφές τις οποίες καθορίζουν οι προδιαγραφές. Αυτό διασφαλίζει ότι όλα τα MBeans προσφέρουν πρόσβαση στους υπό διαχείριση πόρους με ένα τυποποιημένο τρόπο.

Εφόσον υλοποιηθεί η πρόσβαση σε κάποιο πόρο μέσω ενός MBean, αυτός μπορεί να τύχει διαχείρισης μέσω ενός πράκτορα JMX. Τα MBeans μπορούν να εγγραφούν σε οποιοδήποτε πράκτορα συμβατό με τις προδιαγραφές JMX χωρίς πρότερη γνώση των ιδιαίτερων χαρακτηριστικών του.

Τα MBeans προσφέρουν ευελιξία, απλότητα και είναι εύκολα στην υλοποίηση. Αυτά τα χαρακτηριστικά δίνουν την δυνατότητα σε όσους αναπτύσσουν λογισμικό, υλικό και δίκτυα να προσθέσουν την δυνατότητα τοπικής και απομακρυσμένης διαχείρισης στα συστήματά τους με ένα τυποποιημένο τρόπο χωρίς να απαιτείται γνώση και κατανόηση πολύπλοκων εφαρμογών διαχείρισης. Επίσης υφιστάμενες εφαρμογές και συστήματα μπορούν να γίνουν διαχειρίσιμα με πολύ μικρές επεκτάσεις στων κώδικά τους. Η τεχνολογία JMX μπορεί να χρησιμοποιηθεί για τη διαχείριση πόρων πολλών διαφορετικών τύπων όπως π.χ. εφαρμογών, υπηρεσιών, συσκευών ή χρηστών. Το κάθε ένα από αυτά μπορεί απλά να επικοινωνεί με ένα αντικείμενο το οποίο συμφωνεί με τις προδιαγραφές ενός MBean. Ένα MBean μπορεί επίσης να ενθυλακώσει υφιστάμενο κώδικα ή αντικείμενα τα οποία λειτουργούν ως αντιπρόσωποι για κάποιο σύστημα.

Το επίπεδο υπηρεσιών καθορίζει επίσης και ένα μηχανισμό ειδοποιήσεων και γεγονότων. Αυτό ο μηχανισμός επιτρέπει στα MBeans να παράγουν ειδοποιήσεις και γεγονότα, τα οποία ακολούθως μπορούν να μεταδώσουν σε άλλα MBeans, σε άλλους JMX πράκτορες ή/και στις εφαρμογές διαχείρισης.

Κάθε υπό διαχείριση πόρος μπορεί να επικοινωνήσει/χρησιμοποιηθεί από ένα ή περισσότερα MBeans τα οποία μπορεί να είναι είτε Standard MBeans είτε Dynamic MBeans, τα οποία και προσφέρουν μεγαλύτερη ευελιξία κατά την εκτέλεση της εφαρμογής. Κάθε MBean οφείλει να υλοποιεί μια διεπαφή – η οποία καθορίζεται ως η διεπαφή διαχείρισης του – και την οποία φανερώνει ο MBean Server στις απομακρυσμένες εφαρμογές διαχείρισης.

Η διεπαφή διαχείρισης του κάθε MBean αποτελείται από:

- Τα ονόματα και τους τύπους των χαρακτηριστικών (attributes) των οποίων οι τιμές μπορούν να διαβαστούν ή/και να μεταβληθούν.
- Τα ονόματα, τις παραμέτρους και τους τύπους επιστροφής των μεθόδων (methods) οι οποίες μπορούν να κληθούν.
- Τους τύπους των ειδοποιήσεων και των γεγονότων που μπορούν να παραχθούν.

Τα χαρακτηριστικά του κάθε MBean είναι εσωτερικές οντότητες οι οποίες γίνονται προσβάσιμες μέσω των μεθόδων set και get. Οι λειτουργίες του, περιλαμβάνουν τις υπόλοιπες μεθόδους της διεπαφής διαχείρισης. Αν ένα MBean διαθέτει μεθόδους ή χαρακτηριστικά τα οποία δεν είναι δηλωμένα στην διεπαφή διαχείρισης, τότε αυτά δεν είναι προσβάσιμα από τον MBean Server, άρα ούτε και από τους απομακρυσμένους διαχειριστές. Σε ένα Standard MBean όλες αυτές οι μέθοδοι δηλώνονται στατικά στη διεπαφή διαχείρισης. Σε ένα Dynamic MBean, η διεπαφή διαχείρισης δηλώνεται σε πραγματικό χρόνο.

2.1.5 Αρχιτεκτονική και Λειτουργία ενός Πράκτορα JMX

Ένας πράκτορας JMX είναι ένας τυπικός πράκτορας διαχείρισης ο οποίος ασκεί άμεσο έλεγχο στους εγγεγραμμένους σε αυτόν πόρους, και που διαθέτει τα χαρακτηριστικά και τις λειτουργίες τους προς χρήση από εφαρμογές διαχείρισης. Οι υπό διαχείριση πόροι βρίσκονται συνήθως στο ίδιο στο ίδιο μηχάνημα με τον JMX πράκτορα χωρίς αυτό να είναι απαραίτητο.

Ο πυρήνας ενός πράκτορα JMX είναι ο MBean Server, ο εξυπηρετητής των διαχειριζόμενων αντικειμένων στον οποίο εγγράφονται τα MBeans. Ένας πράκτορας JMX προσφέρει επίσης μια σειρά από υπηρεσίες, χρήσιμες κατά τη διαχείριση των MBeans καθώς και ένα τουλάχιστον κανάλι επικοινωνίας για χρήση από απομακρυσμένες ή τοπικές εφαρμογές διαχείρισης.

Κατά την υλοποίηση ενός πράκτορα JMX, η πρότερη γνώση των πόρων που θα κληθεί να διαχειριστεί ή των λειτουργιών που αυτοί θα εκτελούν, δεν έχει καμία σημασία και καμία χρησιμότητα. Οι υπό διαχείριση πόροι μπορούν να εγγραφούν στον πράκτορα, φτάνει να συμμορφώνονται στο μοντέλο που καθορίζουν οι προδιαγραφές JMX. Παρομοίως η υλοποίηση του πράκτορα είναι ανεξάρτητη των λειτουργιών που εκτελούν οι εφαρμογές διαχείρισης.

Η εκτέλεση εργασιών πάνω στους εγγεγραμμένους πόρους μέσω του πράκτορα μπορεί να γίνει από όλες τις συνδεδεμένες σε αυτόν οντότητες. Δηλαδή η εγγραφή/διαγραφή MBeans, η ενεργοποίηση των υπηρεσιών του πράκτορα, καθώς και η κλήση μεθόδων των MBeans μπορεί να γίνει είτε από τον ίδιο τον πράκτορα, είτε από άλλο (εγγεγραμμένο) MBean είτε από μια συνδεδεμένη με τον πράκτορα εφαρμογή διαχείρισης.

Η διάκριση μεταξύ των εγγεγραμμένων πόρων γίνεται μέσω του μοναδικού για κάθε MBean ονόματος, το οποίο δηλώνεται κατά την εγγραφή. Το όνομα αυτό έχει την μορφή:

Domain:*attr1=value1,attr2=value2*

Με την ορθολογιστική χρήση των ονομάτων μπορούμε να κατηγοριοποιήσουμε τα MBean με βάση το Domain, και να δώσουμε μερικές εξ' αρχής πληροφορίες για τη λειτουργία τους μέσω των ζευγών *attr=value.*

2.1.6 Δυναμική Φόρτωση Κλάσεων

2.1.6.1 Εισαγωγή - Γενικά περί φόρτωσης κλάσεων

Σε κάθε περίπτωση, όταν ένα πρόγραμμα JAVA εκτελείται μέσα σε μια Java Virtual Machine, υπάρχει ένας μηχανισμός ο οποίος μεταφράζει, εξετάζει την ακεραιότητα και φορτώνει κλάσεις οι οποίες βρίσκονται σε μορφή ενδιάμεσου κώδικα (bytecodes). Το Java API επίσης, περιλαμβάνει μηχανισμό με τον οποίο μπορούν να φορτωθούν δηλώσεις κλάσεων δυναμικά και να ενσωματωθούν στο τρέχον πρόγραμμα ώστε να μπορούν να δημιουργηθούν αντικείμενα – στιγμιότυπα αυτών των κλάσεων. Όταν

μεταφράζονται (compile) τα προγράμματα Java οι ζητούμενες κλάσεις ή τα διάφορα πακέτα – βιβλιοθήκες, εντοπίζονται μέσα σε αρχεία σε μορφή bytecodes, τα οποία αρχεία βρίσκονται στους τοπικούς δίσκους, σε χώρους που καθορίζει η μεταβλητή περιβάλλοντος CLASSPATH.

Για κάθε κλάση η οποία υλοποιεί κάποιο interface, επεκτείνει κάποια άλλη κλάση ή χρησιμοποιεί άλλες κλάσεις εκτός των πρωτογενών τύπων (primitive types –String, Integer κ.τ.λ), όλες οι κλάσεις από τις οποίες εξαρτάται πρέπει να μπορούν να αναζητηθούν στον ίδιο χώρο. Κάθε κλάση, φορτώνεται από έναν φορτωτή κλάσεων (Classloader) ο οποίος εντοπίζει τα bytecodes, «ανακατασκευάζει» την κλάση από αυτά και δημιουργεί στιγμιότυπα της κλάσης. Οι classloaders είναι και αυτοί κλάσεις που φορτώνονται από άλλους classloaders σχηματίζοντας τελικά ένα νοητό δέντρο. Ο κάθε classloader έχει πρόσβαση στις κλάσεις που έχουν φορτώσει όλοι οι classloaders που βρίσκονται στο μονοπάτι που τον ενώνει με τη ρίζα του δέντρου. Όταν μια κλάση φορτωθεί μια φορά, παραμένει στην μνήμη ώστε να μπορούν να δημιουργηθούν νέα στιγμιότυπα χωρίς να γίνεται νέα αναζήτηση στο δίσκο.

Χαρακτηριστικό της γλώσσας Java είναι η δυνατότητα να αναζητεί τις δηλώσεις των κλάσεων και μέσω δικτύου, αφού μπορεί να «κατεβάσει» τα ζητούμενα bytecodes μέσω του πρωτοκόλλου HTTP. Γι αυτό το σκοπό έχει δημιουργηθεί ο URLClassLoader, ένας φορτωτής κλάσεων ο οποίος βρίσκεται στην βιβλιοθήκη java.net και ο οποίος έχει τη δυνατότητα να αναζητεί αρχεία class και jar σε ένα σύνολο από URL. Αυτός ο μηχανισμός δίνει στην περίπτωση JMX τη δυνατότητα στον χρήστη (απομακρυσμένη εφαρμογή διαχείρισης) να αποστείλει στον πράκτορα δηλώσεις κλάσεων τις οποίες δεν γνώριζε και τις οποίες δεν μπορεί να εντοπίσει στο δικό του δίσκο.

2.1.6.2 Υπηρεσία M-Let (Management Applet)

Η υπηρεσία m-let συγκαταλέγεται στις υπηρεσίες του επιπέδου πράκτορα των προδιαγραφών JMX και αφορά τη δυναμική φόρτωση κλάσεων. Η υπηρεσία m-let είναι στην ουσία ένας classloader ο οποίος έχει τη διαμόρφωση ενός MBean ώστε να μπορεί

να εγγραφεί σε ένα MBean Server. Ακολούθως μπορεί να φορτώσει κλάσεις οι οποίες βρίσκονται σε αρχεία αποθηκευμένα σε απομακρυσμένες τοποθεσίες.

Η πιο γνωστή λειτουργία της υπηρεσίας M-Let, είναι η δυνατότητα φόρτωσης κλάσεων και εγγραφής MBeans σύμφωνα με τις πληροφορίες που περιλαμβάνονται σε ένα αρχείο κειμένου. Ο χρήστης μπορεί να κατασκευάσει ένα αρχείο κειμένου το οποίο να περιλαμβάνει mlet tags, το καθένα από τα οποία να περιέχει όλες τις απαραίτητες πληροφορίες για την αρχικοποίηση ενός MBean. Τα mlet tags έχουν την εξής σύνταξη και ερμηνεία:

```
<MLET
CODE = class | OBJECT = serfile
ARCHIVE = "archiveList"
[CODEBASE = codebaseURL]
[NAME = mbeanname]
[VERSION = version]
>
[arglist]
</MLET>
```

όπου:

`CODE =` *class*

Αυτό το πεδίο καθορίζει το πλήρες όνομα της κλάσης του MBean που θα αρχικοποιηθεί, συμπεριλαμβανομένου και του πακέτου στο οποίο ανήκει. Το αρχείο .class της κλάσης πρέπει να μπορεί να εντοπιστεί σε ένα από τα αρχεία που καθορίζει το πεδίο ARCHIVE. Η παρουσία του πεδίου αυτού είναι υποχρεωτική δεδομένου ότι απουσιάζει το πεδίο `OBJECT` .

`OBJECT =` *serfile*

Το πεδίο αυτό καθορίζει το αρχείο `.ser` το οποίο περίχει μια σειριοποιημένη αναπαράσταση του MBean που θα εγγραφεί. Το αρχείο .class της κλάσης πρέπει να μπορεί να εντοπιστεί σε ένα από τα αρχεία που καθορίζει το πεδίο ARCHIVE. Η παρουσία του πεδίου αυτού είναι υποχρεωτική δεδομένου ότι απουσιάζει το πεδίο `CODE`.

`ARCHIVE = "`*archiveList*`"`

Το υποχρεωτικό αυτό πεδίο καθορίζει ένα σύνολο από αρχεία .jar τα οποία περιέχουν τα MBeans και οποιοδήποτε άλλο αρχείο ή κλάση χρειάζεται για την εγγραφή τους. Το αρχείο που δηλώνεται με τα πεδία CODE ή OBJECT πρέπει να περιλαμβάνεται σε ένα από αυτά τα αρχεία. Το πεδίο μπορεί να περιλαμβάνει περισσότερα από ένα αρχεία, τα οποία διαχωρίζονται μεταξύ τους με κόμμα (,)

και περικλείονται σε εισαγωγικά(«»). Όλα τα αρχεία πρέπει να είναι προσβάσιμα στην τοποθεσία που καθορίζει το πεδίο CODEBASE.

`CODEBASE =` *codebaseURL*

Το προαιρετικό αυτό πεδίο καθορίζει το URL όπου θα βρεθεί το MBean. Καθορίζει το directory στο οποίο θα βρεθούν τα αρχεία .jar που καθορίζει το πεδίο `ARCHIVE`. Αν δεν καθορίστεί αυτό το πεδίο τότε θεωρείται ότι τα αρχεία βρίσκονται στο ίδιο URL με το text αρχείο τα καθορίζει.

`NAME =` *mbeanname*

Προαιρετικό πεδίο το οποίο δηλώνει το objectname με το οποί θα εγγραφεί το MBean στον MBean Server. Ένα το objectname αρχίζει από (:) τότε το MBean εγγράφεται στο default domain.

`VERSION =` *version*

Το πεδίο αυτό είναι προαιρετικό και καθορίζει τον αριθμό της έκδοσης του MBean και των συσχετισμένων με αυτό .jar αρχείων. Με αυτό το πεδίο καθορίζεται το κατά πόσο τα αρχεία που θα φορτωθούν, θα αντικαταστήσουν αρχεία τα αοποία έχουν φορτωθεί με παλαιότερες κλήσεις της υπηρεσίας m-let.

arglist

Περιλαμβάνει μια ακολουθία από τις τιμές των παραμέτρων που χρειάζεται το MBean για να αρχικοποιηθεί. Οι παράμετροι περιγράφονται με tags τα οποία έχουν την πιο κάτω μορφή:

<`ARG TYPE=`*argumentType* `VALUE=`*value*>

όπου:

- *argumentType* είναι ο τύπος των δεδομένων που της συγκεκριμένης παραμέτρου.

Οι τύποι των παραμέτρων μπορούν να ανήκουν μόνο στους πρωτογρνείς ή στους βασικούς τύπους της Java (`java.lang.Boolean, java.lang.Byte, java.lang.Short, java.lang.Long, java.lang.Integer, java.lang.Float, java.lang.Double, java.lang.String`).

Το M-Let MBean αποτελεί επέκταση της κλάσης URLClassLoader η οποία έχει τη δυνατότητα να φορτώνει κλάσεις οι οποίες βρίσκονται σε ένα σύνολο από απομακρυσμένες τοποθεσίες, προσβάσιμες μέσω του πρωτοκόλλου HTTP. Το σύνολο των URL μπορεί να αφορά ένα δικτυακό directory (π.χ. http://codebase.ntua.gr/) όπου βρίσκονται αρχεία class, είτε να αναφέρεται σε βιβλιοθήκη κλάσεων τύπου jar (π.χ. http://codebase.ntua.gr/mybeans.jar) από την οποία μπορούν να αντληθούν οι ζητούμενες κλάσεις. Σε κάθε περίπτωση, η λίστα με τα URL με τα οποία επικοινωνεί το M-Let

MBean, μπορεί να μεταβληθεί με προσθήκες και διαγραφές. Αυτές οι λειτουργίες, δίνουν την δυνατότητα να χρησιμοποιηθεί σαν κλασσικός classloader, του οποίου το classpath επεκτείνεται δυναμικά, χωρίς να είναι απαραίτητη η διαδικασία της δημιουργίας αρχείου με mlet tags.

Ιδιαίτερη σημασία έχουν οι μέθοδοι addURL(java.lang.String url) και **addURL**(java.net.URL url) με τις οποίες μπορούμε προγραμματιστικά να προσθέσουμε βιβλιοθήκες και ακολούθως να φορτώσουμε MBeans δηλώνοντας την υπηρεσία m-let ως των classloader που θα αναλάβει την φόρτωση των κλάσεων.

Σε κάθε περίπτωση πάντως είναι απαραίτητη η ύπαρξη ενός http server ο οποίος να εξυπηρετεί τα αρχεία .jar και .class. Αυτό τον περιορισμό επιχειρήσαμε να άρουμε δημιουργώντας μια επέκταση της υπηρεσίας mlet η οποία επιτρέπει την αποστολή και φόρτωση κλάσεων στον πράκτορα JMX χωρίς την υποχρεωτική παρουσία εξυπηρετητή HTTP. Λεπτομέρειες και χαρακτηριστικά παρουσιάζονται στο κεφάλαιο Αρχιτεκτονική και Υλοποίηση.

2.2 JMX Remote API

2.2.1 Εισαγωγή

Το JMX Remote API, αποτελεί υλοποίηση του επιπέδου κατανεμημένων υπηρεσιών της τεχνολογίας JMX. Όπως έχει αναφερθεί, προσφέρει τα εργαλεία και τις διεπαφές που απαιτούνται ώστε να είναι χρησιμοποιήσιμος και διαχειρίσιμος ένας JMX πράκτορας. Βασικό χαρακτηριστικό είναι δυνατότητα απομακρυσμένης πρόσβασης στον MBean Server (και στα εγγεγραμμένα σ' αυτόν MBeans) χρησιμοποιώντας διάφορα πρωτόκολλα επικοινωνίας και μεθόδους απομακρυσμένης πρόσβασης (όχι απαραίτητα τεχνολογίες JAVA). Επίσης καθορίζονται οι διεπαφές και οι προδιαγραφές ώστε να μπορούν να προστεθούν υλοποιήσεις άλλων πρωτοκόλλων από το χρήστη. Τελικός στόχος του API είναι η απομακρυσμένη κλήση μεθόδων του JMX πράκτορα και η μεταφορά ειδοποιήσεων με διάφανο και ασφαλή τρόπο, ανεξάρτητο από το πρωτόκολλο επικοινωνίας.

2.2.2 Στοιχεία Αρχιτεκτονικής και Προγραμματιστικό μοντέλο

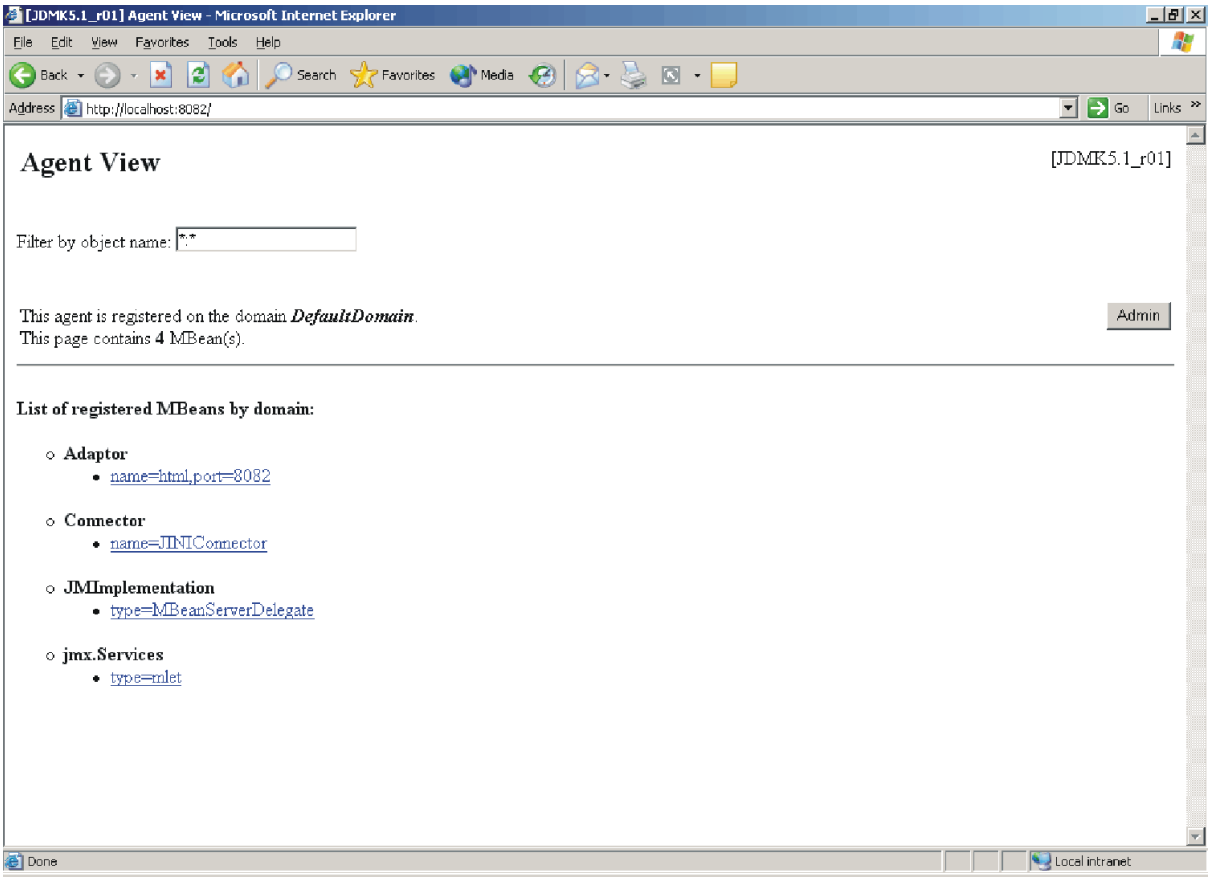
Η επικοινωνία των εφαρμογών διαχείρισης μπορεί να γίνει μέσω protocol adaptors ή μέσω connectors. Οι protocol adaptors δίνουν τη δυνατότητα στον JMX πράκτορα να μπορεί να επικοινωνήσει χρησιμοποιώντας συγκεκριμένο πρωτόκολλο (π.χ. HTTP) ενώ οι connectors ουσιαστικά έχουν διαμόρφωση εξυπηρετητή και περιμένουν αιτήσεις από απομακρυσμένους πελάτες μέσω μια μεθόδου απομακρυσμένης κλήσης υπηρεσιών (π.χ. RMI). Να σημειώσουμε ότι παρόλο που η ύπαρξη των protocol adapters προβλέπεται από το πρότυπο JMX, δεν προβλέπεται η υποχρεωτική υλοποίηση adaptors που να αφορούν συγκεκριμένα πρωτόκολλα, και οι υλοποιήσεις αυτών δεν αφορούν το πρότυπο. Για κάποια συγκεκριμένα πρωτόκολα – κυρίως πρωτόκολλα παράστασης πληροφορίας διαχείρισης – έχουν διαμορφωθεί και υλοποιηθεί ξεχωριστά JSR (Java Specification Request) σε μια διαδικασία ανεξάρτητη του JMX.

2.2.3 Protocol Adapters

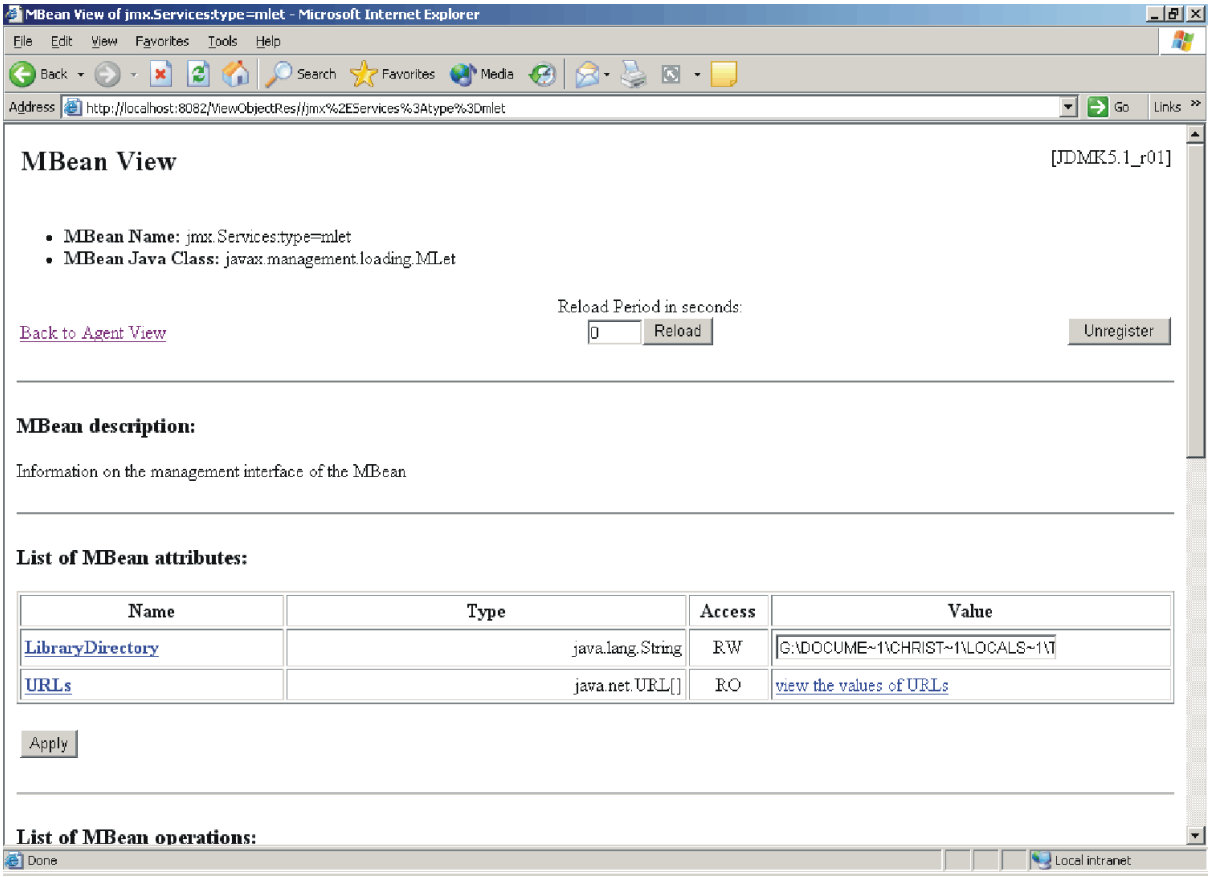
Ένας protocol adapter είναι ουσιαστικά ένα MBean το οποίο αφού εγγραφεί στον MBean Server αναμένει αιτήσεις σε συγκεκριμένη πόρτα επικοινωνίας και επικοινωνεί με συγκεκριμένο πρωτόκολλο. Παράδειγμα ενός τέτοιου adaptor είναι ο HTML adaptor ο οποίος αναμένει αιτήσεις HTTP στην πόρτα 80 και παράγει αποκρίσεις σε μορφή HTML. Για την χρήση της συγκεκριμένης μεθόδου επικοινωνίας μπορεί να χρησιμοποιηθεί οποιοδήποτε κοινό εργαλείο «μιλάει» το συγκεκριμένο πρωτόκολλο. Για το παράδειγμα του HTML adaptor, ένας απομακρυσμένος χρήστης μπορεί να καλέσει μεθόδους ενός JMX πράκτορα και να διαχειριστεί τους εγγεγραμμένους σ' αυτόν πόρους, χρησιμοποιώντας ένα απλό web browser. Σε κάθε περίπτωση πάντως το πρωτόκολλο επιβάλει τους εν γένει περιορισμούς που το χαρακτηρίζουν.

Ο HTML adaptor παρουσιάζει δύο τέτοιους σοβαρούς περιορισμούς. Πρώτον είναι το γεγονός ότι η μόνη μορφή δεδομένων που μπορεί να μεταφερθεί μέσω του HTTP πρωτοκόλλου είναι πρωτογενείς τύποι (string, integer, float κ.τ.λ.), αφαιρώντας την δυνατότητα της μεταφοράς πιο σύνθετων δομών ή αντικειμένων. Δηλαδή δεν μπορούν να κληθούν μεθόδοι των διαχειριζόμενων πόρων ή να διαβαστούν χαρακτηριστικά τα οποία δέχονται ως παράμετρο ή επιστρέφουν δεδομένα άλλου τύπου από τους πρωτογενείς τύπους δεδομένων (primitive data types). Ο δεύτερος σοβαρός περιορισμός οφείλεται στο σύγχρονο μοντέλο αιτήσεων/αποκρίσεων (request/response) που χρησιμοποιεί το HTTP πρωτόκολλο, το οποίο απαγορεύει την μεταφορά δεδομένων ή αιτήσεων από τον εξυπηρετητή στον πελάτη αν δεν έχει προηγηθεί σχετική αίτηση από τον πελάτη. Συνέπεια αυτού του χαρακτηριστικού είναι το ότι δεν μπορούν να σταλούν στον πελάτη ειδοποιήσεις ασύγχρονα παρά μόνο εφόσον ο ίδιος ο πελάτης τη ζητήσει. Το πρόβλημα αυτό έχει αντιμετωπισθεί με τη δημιουργία ενός notification mailbox, το οποίο συλλέγει όλα τα απομακρυσμένα γεγονότα που αφορούν ένα HTML πελάτη και τα οποία μεταφέρονται σε αυτόν όταν αυτός τα ζητήσει. Ο πελάτης οφείλει να αναζητεί περιοδικά τις ειδοποιήσεις που τον αφορούν.

Ο HTML adaptor αν και αρκετά διαδεδομένος, κυρίως όσον αφορά τον έλεγχο του JMX πράκτορα στο στάδιο της ανάπτυξης λογισμικού, δεν αποτελεί μέλος των προδιαγραφών JMX ούτε είναι υποχρεωτικό συστατικό της όποιας υλοποίησης. Μια υλοποίηση του HTML adaptor προσφέρει η ίδια η Sun, η οποία όμως διανέμεται με την βιβλιοθήκη com.sun.jdmk.* και όχι με τη βιβλιοθήκη javax.management.* ή javax.management.remote.* οι οποίες πρέπει να περιέχουν όλα όσα επιβάλει το πρότυπο JMX. Όπως έχει αναφερθεί και πιο πάνω, το πρότυπο JMX, προβλέπει την ύπαρξη protocol adaptors, χωρίς να επιβάλει την υλοποίηση τέτοιων για συγκεκριμένα πρωτόκολλά.



Σχ. 2.2.1 - *Η κεντρική όψη ενός JMX πράκτορα μέσω του HTML Adaptor της Sun. Διακρίνονται τα MBeans που είναι εγγεγραμμένα στον MBean Server καθώς και το domain του καθενός.*



Σχ.2.2.2- *Η όψη ενός MBean μέσω του HTML Adaptor της Sun. Στη συγκεκριμένη περίπτωση το MBean αποτελεί και μια από τις υπηρεσίες που προσφέρει το επίπεδο πράκτορα – η υπηρεσία M-Let τη δυναμική φόρτωση απομακρυσμένων βιβλιοθηκών κώδικα. Διακρίνονται ταχαρακτηριστικά του MBean, οι διαθέσιμες μέθοδοι, το όνομα της κλάσης στην οποία ανήκει το MBean, καθώς και το domain εγγραφής του.*

Πιο πάνω παρουσιάζονται screenshots της εικόνας του HTML adaptor της Sun. Παρουσιάζεται η μορφή με την οποία φανερώνονται στον χρήστη τα εγγεγραμμένα στον server MBeans καθώς και ο τρόπος με τον οποίο αποκαλύπτονται τα προς διαχείριση χαρακτηριστικά και μεθόδοι του κάθε διαχειριζόμενου πόρου (MBean).

Όπως έχει αναφερθεί πιο πάνω, οι protocol adapters μπορούν να αφορούν όχι μόνο πρωτόκολλα επικοινωνίας, αλλά και πρωτόκολλα παράσταση πληροφορίας διαχείρισης. Τέτοια πρωτόκολλα είναι όπως φαίνεται και στο Σχ. 2.1.3 τα SNMP (Simple Network Management Protocol), CIM/WBEM (Common Information Model/Web Based Enterprise Management) και TMN (Telecommunications Management Network).

2.2.4 Connectors

Οι connectors χρησιμοποιούνται για να συνδέσουν ένα JMX πράκτορα με απομακρυσμένες εφαρμογές διαχείρισης. Οι connectors έχουν διαμόρφωση πελάτη - εξυπηρετητή και έχουν ως σκοπό να φανερώσουν μια συγκεκριμένη διεπαφή στην πλευρά του πελάτη η οποία να αποκρύπτει την πολυπλοκότητα του μηχανισμού απομακρυσμένης κλήσης μεθόδων.

Συγκεκριμένα ένας connector αποτελείται από το server κομμάτι και το client κομμάτι. Το server κομμάτι είναι εγγεγραμμένο στον MBean Server και περιμένει αιτήσεις για συνδέσεις στον JMX πράκτορα από απομακρυσμένους πελάτες. Τα δύο μέρη επικοινωνούν μεταξύ τους χρησιμοποιώντας συγκεκριμένο πρωτόκολλο επικοινωνίας, ανταλλαγής μηνυμάτων ή κλήσης μεθόδων απομακρυσμένων αντικειμένων και ο κάθε connector αφορά συγκεκριμένο πρωτόκολλο. Στην πράξη όμως για μια εφαρμογή διαχείρισης, το πρωτόκολλο που χρησιμοποιείται παρασκηνιακά, είναι αδιάφορο αφού όλοι οι connectors οφείλουν τελικά να φανερώνουν στον πελάτη την ίδια διεπαφή (MBeanServerConection) η οποία είναι υποσύνολο (υπερκλάση) της διεπαφής που μπορεί να χρησιμοποιήσει ένας τοπικός χρήστης του πράκτορα (MBeanServer). Συνεπώς μια εφαρμογή διαχείρισης μπορεί να χρησιμοποιήσει οποιοδήποτε connector χωρίς αλλαγές αφού η απομακρυσμένη διεπαφή παραμένει σταθερή, και οι κλήσεις μεταφέρονται από άκρο σε άκρο με διάφανο τρόπο.

Το server κομμάτι ενός connector συνήθως διαθέτει μια διεύθυνση στο δίκτυο που επιτρέπει στους πελάτες επικοινωνήσουν μαζί του και να εγκαταστήσουν μια σύνδεση. Εναλλακτικά άλλοι connectors διαθέτουν αντικείμενα αντιπρόσωπους (proxies) με τα οποία εγκαθιστούν μια σύνδεση. Η επιλογή της μεθόδου σύνδεσης έχει να κάνει με την μέθοδο αναζήτησης και ανακάλυψης των connector servers μέσω του δικτύου.

Οι προδιαγραφές του JMX Remote API προβλέπουν την υλοποίηση connectors που να χρησιμοποιούν συγκεκριμένα πρωτόκολλα. Επίσης καθορίζουν την υποδομή και τις

παραμέτρους τις οποίες πρέπει να πληροί οποιοσδήποτε connector υλοποιηθεί από το χρήστη.

2.2.5 RMI Connector

Ο RMI Connector, όπως δηλώνει και το όνομά του χρησιμοποιεί ως μέθοδο επικοινωνίας το πρωτόκολλο RMI (Remote Method Invocation) και η παρουσία του είναι υποχρεωτική για κάθε υλοποίηση των προδιαγραφών JMX Remote API. Το RMI καθορίζει δύο μεθόδους μεταφορά κλήσεων μέσω δικτύου, το Java Remote Method Protocol (JRMP) και το Internet Inter – ORB Protocol (IIOP). Ο RMI Connector υποστηρίζει και τα δύο πρωτόκολλα. Μέσω JRMP ο RMI connector προσφέρει και ένα απλό μηχανισμό ασφάλειας και (authentication) για την εξασφάλιση σύνδεσης. Η ασφάλεια μπορεί να ενισχυθεί εάν χρησιμοποιηθούν sockets για την σύνδεση ώστε να μπορεί να χρησιμοποιηθεί μεταξύ client και server το πρωτόκολλο SSL (Secure Socket Layer).

2.2.6 Generic Connector

Ο Generic Connector αποτελεί προαιρετικό τμήμα του JMX Remote API και μπορεί να ρυθμιστεί εάν προστεθούν σε αυτόν τμήματα κώδικά τα οποία καθορίζουν:

- Το πρωτόκολλο μεταφοράς το οποίο θα χρησιμοποιηθεί για την αποστολή αιτήσεων από τον πελάτη στον εξυπηρετητή και για την αποστολή αποκρίσεων και ειδοποιήσεων από τον εξυπηρετητή στον πελάτη.
- Την μέθοδο με την οποία θα ενθυλακώνονται τα αντικείμενα που θα αποστέλλει ο πελάτης στον εξυπηρετητή και τα οποία θα φορτωθούν από τον φορτωτή κλάσεων του MBean στο οποίο θα εκτελεστεί η κάθε λειτουργία.

Μια περίπτωση συμπληρωμένου generic connector είναι ο JMXMP Connector (JMX Messaging Protocol) ο οποίος περιλαμβάνεται στην υλοποίηση της Sun για το JMX Remote API. Σε αυτή την περίπτωση το πρωτόκολλο μεταφοράς των αιτήσεων βασίζεται στο διαδεδομένο TCP και η μεταφορά των αντικειμένων γίνεται μέσω σειριοποίησης JAVA. Σε αυτή την περίπτωση υπάρχει η δυνατότητα για ακόμη περισσότερη ασφάλεια

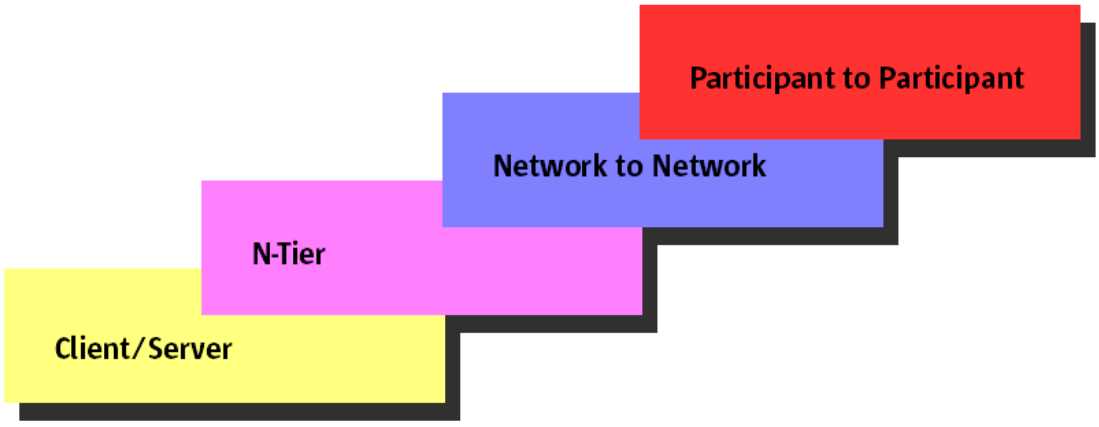
στην σύνδεση αφού μπορούν να χρησιμοποιηθούν οι τεχνολογίες Java Secure Socket Extension (JSSE), Java Authentication and Authorisation Service (JAAS) και Simple Authentication and Security Layer (SASL).

2.3 Τεχνολογία JINI

2.3.1 Εισαγωγή

Το Internet μας έχει δώσει τη δυνατότητα δημιουργίας δικτύων, τα οποία δεν διασυνδέουν πλέον μόνο υπολογιστές αλλά γενικότερα συσκευές (συμπεριλαμβανομένων των υπολογιστών) οι οποίες προσφέρουν υπηρεσίες μέσω του δικτύου (Web Services). Αν και η τεχνολογία Jini δεν ανήκει στην κατηγορία των web-services, εντούτοις μπορεί να επικοινωνήσει με αυτές τις υπηρεσίες ή ακόμα και να χρησιμοποιηθεί για την υλοποίηση τέτοιων υπηρεσιών. Ένα σύστημα Jini προσφέρει πρόσβαση σε υπηρεσίες μέσω δικτύου, ανεξάρτητα από λειτουργικό σύστημα, την πολυπλοκότητα του δικτύου ή την απόσταση μεταξύ των οντοτήτων που συνεργάζονται.

Η προσπάθεια παροχής και χρήσης υπηρεσιών μέσω δικτύου, είναι αρκετά παλιά, και έχουν εξελιχθεί προς αυτό αρκετές διαφορετικές προσεγγίσεις σχεδιασμού, οι οποίες φιλοδοξούν να λύσουν μια σειρά από προβλήματα που προκύπτουν. Ακόμα περισσότερο, προσπαθούν να θεραπεύσουν με αποτελεσματικό τρόπο μια σειρά από ανάγκες που προκύπτουν οι οποίες περιστρέφονται κυρίως γύρω από θέματα ασφάλειας και αξιοπιστίας των συστημάτων. Στο Σχ. 2.3.1 φαίνεται η ιεραρχία και η χρονική εξέλιξη αυτών των προσεγγίσεων.



Σχήμα 2.3.1

Κάθε γενιά συστημάτων χαρακτηρίζεται από τα προβλήματα που καλείται να αντιμετωπίσει καθώς κι από τις τεχνολογίες που αντιμετωπίζουν αυτά τα προβλήματα. Στα σημερινά συστήματα τα οποία ανήκουν κυρίως στη γενιά των multi-tier συστημάτων, όπου μια υπηρεσία υλοποιείται από ένα διατεταγμένο σύνολο υπολογιστών. Ο κάθε υπολογιστής μπορεί να επικοινωνεί μόνο με τα γειτονικά μέλη αυτής της ακολουθίας. Με αυτό τον τρόπο διαχωρίζεται ο πελάτης, από τον εξυπηρετητή, μα και ο εξυπηρετητής από ένα πιθανό πληροφοριακό σύστημα (βάση δεδομένων) που μπορεί να χρησιμοποιεί. Για την εγκατάσταση ενός τέτοιου συστήματος, έχουν γίνει μια σειρά από υποθέσεις-παραδοχές, οι οποίες εκ των πραγμάτων αποδεικνύεται ότι δεν ισχύουν. Έτσι ο σχεδιαστής ενός multi-tier συστήματος έχει την ψευδαίσθηση ότι:

- Το δίκτυο είναι αξιόπιστο.
- Η καθυστέρηση που προκαλεί το δίκτυο στην επικοινωνία είναι μηδενική ή αμελητέα.
- Το διαθέσιμο εύρος ζώνης είναι σταθερό και δεδομένο.
- Το δίκτυο είναι ασφαλές.
- Η τοπολογία του δικτύου δεν αλλάζει.
- Ο διαχειριστής του συστήματος είναι ένας και μοναδικός.
- Το κόστος μεταφοράς δεδομένων είναι μηδενικό.

Τα προβλήματα που ανακύπτουν σήμερα και τα οποία δεν αντιμετωπίζονται από το multi-tier μοντέλο, είναι τα εξής:

- Η εύρεση και διασύνδεση υπηρεσιών μέσω δικτύου χρησιμοποιώντας αυτοτελής οντότητες, με ένα ενιαίο κοινό σύστημα διευθυνσιοδότησης στο δίκτυο.
- Η δημιουργία αξιόπιστων υπηρεσιών συνθέτοντας μη αξιόπιστα μέρη, συμπεριλαμβανομένου και του μη αξιόπιστου δικτύου.
- Ο χειρισμός πολύ μεγάλων και πολύπλοκων δικτύων.
- Ο αναβάθμιση μιας υπηρεσίας ή τμήματος αυτής, χωρίς να είναι απαραίτητο να τεθεί εκτός λειτουργίας ολόκληρο το σύστημα, και να επανεκκινηθεί.

Όλα τα πιο πάνω φιλοδοξεί να αντιμετωπίσει μια προσέγγιση που επικεντρώνεται στις δύο οντότητες που αλληλεπιδρούν. Έτσι με μια αρχιτεκτονική *μετόχου-προς-μέτοχο* (participant-to-participant), μέσω της διασύνδεσης δικτύων, δίνεται η δυνατότητα σε οποιοδήποτε πελάτη βρίσκεται σε ένα δίκτυο, να αναζητήσει και να χρησιμοποιήσει υπηρεσίες που προσφέρονται από οποιονδήποτε μετέχει με ένα από τα συνδεδεμένα δίκτυα. Η έννοια του συνδεδεμένου δικτύου εδώ αφορά τη διασύνδεση κατανεμημένων συστημάτων τα οποία έχουν δική τους εσωτερική δομή και λειτουργικότητα και τα οποία είναι δομημένα πάνω σε κάποιο δίκτυο υπολογιστών. Έτσι η γενιά μέτοχος-προς-μέτοχο στην οποία ανήκει και η τεχνολογία Jini, προσφέρει τα πιο κάτω χαρακτηριστικά:

- Αντιμετωπίζει την ενδογενή έλλειψη αξιοπιστίας του δικτύου
- Αντιμετωπίζει την συνεχώς μεταβαλλόμενη τοπολογία του δικτύου
- Επιτρέπει την ύπαρξη πολλών διαχειριστών του συστήματος
- Επιτρέπει την αναβάθμιση τμημάτων του συστήματος χωρίς να επηρεάζονται τα υπόλοιπα μέρη.
- Λαμβάνει σοβαρά υπόψη την πιθανότητα αστοχίας του συστήματος κατά τον σχεδιασμό των αλληλεπιδράσεων μεταξύ μετόχων.

Αντιμετωπίζοντας τα πιο πάνω θέματα, η τεχνολογία Jini προσφέρει ένα απλό, γρήγορο και εύκολο τρόπο επικοινωνίας μεταξύ υπηρεσιών. Η μόνη ενέργεια που απαιτείται από τον πελάτη, είναι να εντοπίσει με δυναμικό τρόπο την υπηρεσία. Το βασικό πλεονέκτημα της αρχιτεκτονικής Jini είναι το ότι αποκρύπτει εντελώς από τα δύο άκρα μιας σύνδεσης, της λεπτομέρειες της επικοινωνίας. Τρεις διαφορετικές υλοποιήσεις μιας υπηρεσίας μπορούν να δέχονται εντολές μέσω XML, μέσω RMI είτε μέσω CORBA, χωρίς αυτό να επηρεάζει τον πελάτη, ή να απαιτεί οποιαδήποτε εξειδικευμένη ρύθμισή του. Ο ίδιος πελάτης μπορεί να χρησιμοποιήσει και τις τρεις υλοποιήσεις.

Μια υπηρεσία για να ενταχθεί σε ένα σύστημα Jini, διαφημίζει το εαυτό της, δημοσιεύοντας ένα αντικείμενο Java το οποίο υλοποιεί το API που διαθέτει η υπηρεσία στους πελάτες της. Οι λεπτομέρειες αυτής της υλοποίησης αφορούν μόνο την υπηρεσία και όχι τον πελάτη. Όταν ο πελάτης θέλει να χρησιμοποιήσει την υπηρεσία, αντιγράφει

το δημοσιευμένο αυτό αντικείμενο και το χρησιμοποιεί ως τοπικό αντιπρόσωπο της απομακρυσμένης υπηρεσίας.

Με αυτό το μηχανισμό, η τεχνολογία Jini δίνει τα εξής πλεονεκτήματα:

- Ο πελάτης σχεδιάζεται με βάση το API της υπηρεσίας αγνοώντας τις λεπτομέρειες της υλοποίησης ή της επικοινωνίας. Προσφέρεται δηλαδή ακόμα ένα επίπεδο αφαίρεσης στο σχεδιασμό.
- Δημιουργούνται εφαρμογές και υπηρεσίες που επικοινωνούν χωρίς την ανάγκη προεγκατεστημένων οδηγών και ρυθμίσεων.
- Κάθε αστοχία στο σύστημα επιδιορθώνεται αυτόματα χωρίς τη ανάγκη επαναρύθμισης του συστήματος.
- Αποσυνδέονται πελάτης και υπηρεσία σε τέτοιο βαθμό ώστε να μπορούν να αντικατασταθούν – είτε ο ένας είτε ο άλλος – χωρίς την ανάγκη παύσης του συστήματος.

2.3.2 Στόχοι του συστήματος

Η βασική ιδέα πίσω από ένα σύστημα Jini είναι η δημιουργία μιας κοινότητας η οποία να αποτελείται από κατανεμημένους χρήστες στο δίκτυο και από τους πόρους και τις υπηρεσίες που αυτοί χρησιμοποιούν. Ο συνολικός στόχος είναι να μετατραπεί το δίκτυο σε ένα ευέλικτο και εύκολα διαχειρίσιμο εργαλείο, μέσω του οποίο θα μπορούν να εντοπιστούν υπηρεσίες και πόροι από τους τελικούς χρήστες, είτε αυτοί είναι εφαρμογές, είτε είναι υπολογιστές. Με τον όρο πόροι ή υπηρεσίες αναφερόμαστε τόσο σε συσκευές hardware όσο και σε software οντότητες. Η σχεδίαση επικεντρώνεται στην αντιμετώπιση του δικτύου ως μια μεταβλητή οντότητα η οποία μοντελοποιεί με περισσότερη ακρίβεια τη φύση μιας κατανεμημένης ομάδας συνεργαζόμενων οντοτήτων προσθέτοντας την δυνατότητα της προσθαφαίρεσης χρηστών και υπηρεσιών.

Ένα σύστημα Jini αποτελείται από τα πιο κάτω μέρη:

- Ένα σύνολο από τμήματα - εργαλεία τα οποία σχηματίζουν την απαραίτητη υποδομή για την «ομοσπονδοποίηση» (ενοποίηση σε μια κοινότητα) των στοιχείων ενός κατανεμημένου συστήματος.

- Το προγραμματιστικό μοντέλο, ικανό να υποστηρίξει και να διευκολύνει την κατασκευή αξιόπιστων κατανεμημένων υπηρεσιών.
- Υπηρεσίες οι οποίες εφόσον ενταχθούν σε μια κοινότητα Jini, διαθέτουν τη λειτουργικότητά τους προς χρήση σε όλα τα μέλη της κοινότητας.

Παρόλο που τα πιο πάνω μέρη είναι διακριτά και διαχωρίσιμα, στην πράξη σχετίζονται πολύ μεταξύ τους. Η υποδομή της τεχνολογίας Jini, είναι ουσιαστικά ένα σύνολο από υπηρεσίες των οποίων η παρουσία είναι απαραίτητη. Αυτές οι υπηρεσίες, κατασκευάζονται σύμφωνα με το προγραμματιστικό μοντέλο καθώς και οι ενταγμένες σε αυτό υπηρεσίες.

Η τεχνολογία Jini, μπορεί σε τελικό στάδιο να εξυπηρετήσει μια σειρά από ανάγκες όπως:

- Δίνει τη δυνατότητα σε χρήστες να μοιράζονται υπηρεσίες και πόρους μέσω του δικτύου.
- Προσφέρει εύκολη πρόσβαση σε πόρους (οι οποίοι μπορούν να βρίσκονται οπουδήποτε στο δίκτυο) από οποιοδήποτε μέρος του δικτύου.
- Απλοποιεί τη διαδικασία κατασκευής, συντήρησης και σύνδεσης με το δίκτυο συσκευών, λογισμικού και τελικών χρηστών.

Το Jini αποτελεί επέκταση του περιβάλλοντος εφαρμογών της Java από τη μία μοναδική JVM, σε ένα σύνολο από JVM που επικοινωνούν χρησιμοποιώντας ένα δίκτυο υπολογιστών. Η γλώσσα Java διαθέτει την κατάλληλη υποδομή για μεταφορά κώδικα και δεδομένων μεταξύ μηχανημάτων που εκτελούν προγράμματα, ιδιότητα που τη χαρακτηρίζει σαν ένα ικανό περιβάλλον ανάπτυξης κατανεμημένων εφαρμογών. Η αυστηρή τυποποίηση του περιβάλλοντος ανάπτυξης εφαρμογών, με ενσωματωμένους μηχανισμούς ασφάλειας, επιτρέπει την αναγνώριση της κλάσης ενός αντικειμένου για εκτέλεση σε μια JVM ακόμα και αν το συγκεκριμένο αντικείμενο αρχικοποιήθηκε σε άλλο μηχάνημα. Αυτό προσφέρει ένα περιβάλλον με αυξημένη τη δυνατότητα κινητικότητας των αντικειμένων εντός του δικτύου, και την κλήση μεθόδων οπουδήποτε στο δίκτυο.

Η αρχιτεκτονική Jini εκμεταλλεύεται αυτά τα χαρακτηριστικά της Java, για να απλοποιήσει την ανάπτυξη ενός κατανεμημένου συστήματος. Πρακτικά, προσθέτει ένα μηχανισμό που επιτρέπει την ροή λειτουργικών τμημάτων, επεκτείνοντας τη δυνατότητα μετακίνησης και ανταλλαγής αντικειμένων σε ολόκληρο το κατανεμημένο σύστημα.

Η υποδομή Jini προσφέρει ένα μηχανισμό για εύκολη σύνδεση και αποσύνδεση χρηστών, υπηρεσιών και συσκευών σε ένα σύστημα. Η είσοδος και η έξοδος σε μια κοινότητα Jini, αντιμετωπίζονται ως φυσικά συμβάντα τα οποία εκτελούνται με αυτόματο τρόπο (Plug and Play). Συνεπώς προκύπτει ένα δυναμικό κατανεμημένο σύστημα, όπου η χειρωνακτική ρύθμιση του κάθε λειτουργικού τμήματος ώστε να επιτευχθεί σύνδεση με το σύστημα, απουσιάζει!

2.3.3 Υποθέσεις όσο αφορά το περιβάλλον λειτουργίας

Η τεχνολογία Jini ενοποιεί υπολογιστές και συσκευές σε μια κοινότητα την οποία ο τελικός χρήστης αντιμετωπίζει ως ένα ενιαίο σύστημα, συγκεντρωμένο σε ένα υπολογιστή. Θεωρείται ότι οι διάφορες λειτουργικές οντότητες επικοινωνούν μεταξύ τους μέσω ενός δικτύου ικανοποιητικής ταχύτητας αν και λαμβάνεται πάντοτε ως δεδομένο ότι το δίκτυο προκαλεί καθυστερήσεις και η μεταφορά εντολών και δεδομένων δεν μπορεί να θεωρηθεί ως άμεση.

Θεωρείται επίσης ότι η κάθε συσκευή που ενώνεται σε ένα σύστημα Jini διαθέτει σε κάποιο βαθμό, δική της μνήμη και υπολογιστική ισχύ. Σε περίπτωση που δεν ικανοποιούνται οι δύο αυτές συνθήκες, μια συσκευή μπορεί και πάλι να συνδεθεί στο σύστημα, αρκεί να μπορεί να χρησιμοποιηθεί από μία άλλη συσκευή με μνήμη και υπολογιστική ισχύ. Σε αυτή την περίπτωση, η υπόψη συσκευή εντάσσεται σε μια κοινότητα Jini δι' αντιπροσώπου (proxy).

Τέλος η υποδομή του Jini έχει ως επίκεντρο την τεχνολογία Java. Η αρχιτεκτονική της τεχνολογίας Jini αντλεί ένα μεγάλο μέρος από την απλότητα στο σχεδιασμό της, από το γεγονός – υπόθεση, ότι όλα τα λειτουργικά της τμήματα είναι υλοποιημένα σε Java. Η

δυνατότητα μεταφοράς και εκτέλεσης κώδικα μέσω δικτύου, αποτελεί θεμελιώδες χαρακτηριστικό του Jini. Παρόλα αυτά το Jini δεν περιορίζεται τόσο στην γλώσσα Java, όσο στο περιβάλλον λειτουργίας της Java (JVM). Συνεπώς μπορεί να χρησιμοποιηθεί οποιαδήποτε γλώσσα, αρκεί να μπορεί να μεταφραστεί σε συμβατά με Java bytecodes!

2.3.4 Επισκόπηση Συστήματος

2.3.4.1 Βασικές αρχές

Σκοπός του Jini είναι η διασύνδεση συσκευών και τμημάτων λογισμικού σε ένα δυναμικό κατανεμημένο σύστημα. Η προκύπτουσα κοινότητα παρέχει απλότητα στην πρόσβαση και την διαχείριση, ενώ υποστηρίζει την παράλληλη χρήση πόρων από διάφορους χρήστες που προσφέρουν μεγάλα μονολιθικά συστήματα. Εν τέλει διατηρεί την ευελιξία και την ομοιογένεια που προσφέρει ένας προσωπικός υπολογιστής.

Ένα σύστημα Jini θεωρεί ότι τα μέλη του τηρούν μια σειρά από συμφωνημένες αρχές εμπιστοσύνης, διαχείρισης, αναγνώρισης και ότι ακολουθούν κοινή πολιτική σε θέματα ασφάλειας. Είναι δυνατό σε μια στρωματοποιημένη αρχιτεκτονική, να διασυνδεθούν μεταξύ τους κοινότητες Jini.

2.3.4.2 Υπηρεσίες (Services)

Θεμελιώδης έννοια της αρχιτεκτονικής Jini είναι αυτή της «υπηρεσίας». Ως υπηρεσία θεωρούμε μια οντότητα η οποία διαθέτει συγκεκριμένη λειτουργικότητα και η οποία μπορεί να χρησιμοποιηθεί από ένα άτομο, ένα πρόγραμμα ή μια άλλη υπηρεσία. Μια υπηρεσία μπορεί να εκτελεί υπολογισμούς, να αποθηκεύει δεδομένα, να λειτουργεί ως αντιπρόσωπος άλλης οντότητας, να είναι μια συσκευή ή οτιδήποτε άλλο.

Τα μέλη μιας κοινότητας Jini μπορούν από κοινού να αποκτήσουν πρόσβαση σε μια υπηρεσία. Μια κοινότητα Jini δεν πρέπει να αντιμετωπίζεται σαν ένα σύνολο από πελάτες και εξυπηρετητές ή σαν σύνολο χρηστών και προγραμμάτων. Αντίθετα σε μια κοινότητα Jini μπορούν να χρησιμοποιηθούν μια ακολουθία από υπηρεσίες, ο

συνδυασμός των οποίων να επιτυγχάνει μια συγκεκριμένη λειτουργία. Η δυναμική φύση ενός συστήματος Jini επιτρέπει την προσθαφαίρεση υπηρεσιών ανάλογα με τις ανάγκες και τη ζήτηση.

Το Jini παρέχει τους μηχανισμούς για κατασκευή υπηρεσιών, την αναζήτησή τους, την επικοινωνία με αυτές και τελικά τη χρήση τους σε ένα κατανεμημένο σύστημα. Υπηρεσίες μπορεί να είναι εκτυπωτές, οθόνες, δίσκοι, εφαρμογές, βάσεις δεδομένων ή ακόμα και οι ίδιοι οι χρήστες του συστήματος! Οι υπηρεσίες επικοινωνούν μεταξύ τους χρησιμοποιώντας ένα πρωτόκολλο – *service protocol* – το οποίο ουσιαστικά είναι ένα σύνολο από διεπαφές (interfaces), γραμμένες σε γλώσσα Java. Το σύνολο αυτό είναι επεκτατό, δηλαδή δίνεται η δυνατότητα για προσθήκη κι άλλων διεπαφών από τον χρήστη ανάλογα με τις υπηρεσίες που κατασκευάζει ή προσθέτει. Το ίδιο το Jini, καθορίζει μόνο ένα μικρό αριθμό τέτοιων διεπαφών, οι οποίες αφορούν κρίσιμες αλληλεπιδράσεις μέσα στο σύστημα, και συγκεκριμένα τις υπηρεσίες που συναποτελούν την υποδομή (π.χ. αναζήτηση).

2.3.4.3 Αναζήτηση Υπηρεσιών (Lookup Service)

Οι υπηρεσίες εντοπίζονται σε μια κοινότητα Jini, με τη χρήση της υπηρεσίας αναζήτησης (lookup service). Η υπηρεσία αναζήτησης αποτελεί κεντρικό μηχανισμό αρχικοποίησης του συστήματος και παρέχει το βασικό σημείο επαφής των υπηρεσιών με τους τελικούς χρήστες. Ακριβέστερα, η υπηρεσία αναζήτησης αντιστοιχεί τις διεπαφές που φανερώνουν τη λειτουργικότητα της κάθε υπηρεσίας, με σύνολα από αντικείμενα τα οποία υλοποιούν τις υπηρεσίες. Πρόσθετα η καταχώρηση πληροφοριακών εγγραφών (attributes), οι οποίες συσχετίζονται με την κάθε υπηρεσία, επιτρέπει την ακόμα πιο λεπτομερή αναζήτηση

Σε μια υπηρεσία αναζήτησης, μπορούν να εγγραφούν ως υπηρεσίες και άλλες υπηρεσίες αναζήτησης. Επιπλέον, μπορούν να εγγραφούν αντικείμενα τα οποία περικλείουν άλλες τεχνολογίες καταχώρησης και αναζήτησης υπηρεσιών (SLP, JNDI κ.τ.λ.), προσφέροντας τη δυνατότητα για διασύνδεση διαφορετικών συστημάτων. Αντίστοιχα, μια υπηρεσία

αναζήτησης Jini, μπορεί να καταχωρηθεί σε υπηρεσίες αναζήτησης άλλων τεχνολογιών προσφέροντας συμμετρική πρόσβαση μεταξύ συστημάτων.

Μια υπηρεσία για να προστεθεί σε μια υπηρεσία αναζήτησης, χρησιμοποιεί τα πρωτόκολλα ανακάλυψης (discovery) και εγγραφής (join). Αρχικά εντοπίζεται μια υπηρεσία αναζήτησης (χρησιμοποιώντας το πρωτόκολλο ανακάλυψης), και ακολούθως καταχωρείται σε αυτή (χρησιμοποιώντας το πρωτόκολλο εγγραφής).

2.3.4.4 Java Remote Method Invocation (RMI)

Η επικοινωνία μεταξύ υπηρεσιών επιτυγχάνεται μέσω του RMI. Η επικοινωνία αυτή από μόνη της, δεν επέχει θέσης υπηρεσίας στο σύστημα η οποία αναζητείται, εντοπίζεται και χρησιμοποιείται, αλλά είναι μέρος της υποδομής του Jini. Το RMI παρέχει τους μηχανισμούς τον εντοπισμό και την ενεργοποίηση απομακρυσμένων αντικειμένων.
Το RMI δεν αφορά μόνο την μεταφορά δεδομένων μεταξύ αντικειμένων, αλλά αφορά και την μεταφορά ολόκληρων αντικειμένων μέσω δικτύου, συμπεριλαμβανομένου του κώδικα που δηλώνει τις κλάσεις στις οποίες ανήκουν. Μεγάλο μέρος της απλότητας στη λειτουργία ενός συστήματος Jini, προκύπτει από την δυνατότητα κυκλοφορίας κώδικα μέσα στο δίκτυο. Το RMI αποτελεί μια επέκταση της γλώσσας Java, και αποτελεί ολοκληρωμένη λύση στο πρόβλημα κλήσης απομακρυσμένων μεθόδων.

2.3.4.5 Ασφάλεια (Security)

Το μοντέλο ασφάλειας της τεχνολογίας Jini βασίζεται στην έννοια του εντολέα (principal) και σε αυτή του *καταλόγου πρόσβασης* (access list). Με τον όρο εντολέας δεν αναφερόμαστε σε φυσικούς χρήστες αλλά στα αντικείμενα που τους αναπαριστούν στην κοινότητα. Ως εντολέας σε κάθε περίπτωση, ορίζεται ο καταναλωτής μιας υπηρεσίας, δηλαδή μια υπηρεσία παίρνει τη θέση του εντολέα, όταν κάνει χρήση μιας άλλης υπηρεσίας. Ακολούθως σε κάθε υπηρεσία, αντιστοιχεί ένα κατάλογος πρόσβασης που περιλαμβάνει τους εντολείς που έχουν δικαίωμα πρόσβασης και χρήσης. Οι υπηρεσίες μπορούν να κάνουν χρήση άλλων υπηρεσιών αλυσιδωτά, χρησιμοποιώντας τα δικαιώματα πρόσβασης του εντολέα στην αρχή της ακολουθίας.

2.3.4.6 Μίσθωση (Leasing)

Η πρόσβαση σε πολλές από τις υπηρεσίες ενός περιβάλλοντος Jini βασίζεται στην έννοια της μίσθωσης (lease). Μίσθωση σημαίνει εξασφαλισμένη πρόσβαση στην υπηρεσία για ένα συγκεκριμένο χρονικό διάστημα το οποίο όμως αφού λήξει, απαιτείται ανανέωση της μίσθωσης για τη συνέχεια. Ο χρήστης και ο παροχέας μιας υπηρεσίας, διαπραγματεύονται μια μίσθωση (lease) χρησιμοποιώντας το πρωτόκολλο υπηρεσίας (service protocol) ως εξής: Ο χρήστης αιτείται την υπηρεσία για συγκεκριμένο χρονικό διάστημα, και παραχωρείται πρόσβαση για κάποιο διάστημα, όχι απαραίτητα όσο ζήτησε ο χρήστης. Εάν η μίσθωση δεν ανανεωθεί προτού να λήξει, - είτε επειδή ο χρήστης δεν χρειάζεται πλέον την υπηρεσία, είτε λόγω πτώσης του δικτύου, είτε επειδή δεν επιτρέπεται η ανανέωση από τον παροχέα – ο χρήστης και ο παροχέας συμπεραίνουν ότι η υπηρεσία μπορεί να ελευθερωθεί.

Οι μισθώσεις σε μια υπηρεσία μπορούν να έχουν το στοιχείο της αποκλειστικότητας, δηλαδή να επιτρέπουν μόνο σε ένα χρήστη να χρησιμοποιεί την υπηρεσία σε μια δεδομένη στιγμή. Υπάρχει όμως και η δυνατότητα για παράλληλη χρήση μιας υπηρεσίας με την παραχώρηση περισσοτέρων μισθώσεων.

Η μίσθωση αφορά και τη χρήση των υπηρεσιών που αποτελούν την υποδομή του Jini. Συγκεκριμένα, η εγγραφή κάποιας υπηρεσίας σε μια υπηρεσία αναζήτησης (lookup service) γίνεται επίσης βάσει μίσθωσης. Αυτό το χαρακτηριστικό, είναι που επιτρέπει στον χρήστη του συστήματος, να έχει ενώπιόν του ανά πάσα στιγμή, μια ακριβή εικόνα ως προς το ποιες υπηρεσίες είναι άμεσα προσβάσιμες και ποιες όχι.

2.3.4.7 Συναλλαγές (Transactions)

Μια ακολουθία εργασιών, είτε εντός μιας υπηρεσίας, είτε κατανεμημένων σε περισσότερες από μια υπηρεσίες, μπορεί να ενθυλακωθεί σε μια συναλλαγή (transaction). Το Jini παρέχει εντός του πρωτοκόλλου υπηρεσιών τις απαραίτητες διεπαφές για την δέσμευση των μεταβολών και των αποτελεσμάτων, όταν συμπληρωθεί όλη η ακολουθία των εργασιών. Η υλοποίηση όμως των διεπαφών αυτών, εξαρτάται αποκλειστικά από την υλοποίηση της υπηρεσίας που τα χρησιμοποιεί.

2.3.4.8 Γεγονότα (Events)

Η αρχιτεκτονική Jini παρέχει υποστήριξη και για κατανεμημένα γεγονότα. Οποιοδήποτε αντικείμενο μπορεί να δώσει τη δυνατότητα σε άλλα αντικείμενα, να εγγραφούν σαν αποδέκτες ειδοποιήσεων όταν συμβούν συγκεκριμένα γεγονότα. Αυτό επιτρέπει την κατασκευή κατανεμημένων εφαρμογών που να βασίζονται σε ένα μοντέλο ειδοποιήσεων όσο αφορά θέματα συγχρονισμού και συνάφειας.

2.3.5 Επισκόπηση των συστατικών μερών του συστήματος

2.3.5.1 Υποδομή

Η υποδομή της τεχνολογίας Jini, καθορίζει τον ελάχιστο δυνατό πυρήνα της τεχνολογία. Ως υποδομή νοούνται τα πιο κάτω:

- Το κατανεμημένο σύστημα ασφάλειας, το οποίο είναι ενσωματωμένο στο RMI και το οποίο επεκτείνει το σύστημα ασφάλειας της Java.
- Τα πρωτόκολα ανακάλυψης και εγγραφής, και το πρωτόκολλο υπηρεσιών το οποίο επιτρέπει σε υπηρεσίες (hardware και software) να ανακαλύπτουν άλλες υπηρεσίες, να εντάσσονται στο σύστημα και να δημοσιεύουν τη λειτουργικότητά τους στα υπόλοιπα μέρη της κοινότητας.
- Η υπηρεσία αναζήτησης, η οποία συνιστά ένα ευρετήριο υπηρεσιών. Στην υπηρεσία αναζήτησης εγγράφονται αντικείμενα γραμμένα στην γλώσσα Java. Τα αντικείμενα αυτά μπορούν να αντιγραφούν στα πλαίσια μιας αναζήτησης και να λειτουργήσουν σαν τοπικοί αντιπρόσωποι (proxies) των υπηρεσιών που τα δημοσίευσαν.

Τα πρωτόκολλα ανακάλυψης και εγγραφής, καθορίζουν τον τρόπο με τον οποίο μια υπηρεσία εντάσσεται σε ένα σύστημα Jini. Η βασική μέθοδος με την οποία επικοινωνούν οι υπηρεσίες είναι το RMI το οποίο χρησιμοποιείται για την επικοινωνία των οντοτήτων που εγγράφονται στο σύστημα, με τις υπηρεσίες που αποτελούν την υποδομή – δηλαδή την υπηρεσία ανακάλυψης (lookup service). Οι υπηρεσίες μεταξύ τους μπορούν να επιλέξουν οποιοδήποτε άλλο πρωτόκολλο, ανάλογα με την υλοποίηση. Το κατανεμημένο

σύστημα ασφάλειας, ταυτοποιεί τις συνεργαζόμενες οντότητες και παραχωρεί δικαιώματα ώστε να ενεργήσουν, είτε για δικό τους λογαριασμό, είτε εκ μέρους κάποιας άλλης οντότητας. Η υπηρεσία αναζήτησης ανά πάσα στιγμή αντικατοπτρίζει την κατάσταση του συστήματος, και περιλαμβάνει τις ενεργές εγγεγραμμένες υπηρεσίες της κοινότητας, ώστε να μπορούν να τις ανακαλύψουν όλα τα μέλη της κοινότητας.

2.3.5.2 Προγραμματιστικό Μοντέλο

Οι υπηρεσίες και τα πρωτόκολλα της υποδομής, χρησιμοποιούν το προγραμματιστικό μοντέλο, και ταυτόχρονα το διαθέτουν προς χρήση στις υπόλοιπες υπηρεσίες του συστήματος. Οι υπηρεσίες εγγράφονται στην υπηρεσία αναζήτησης βάσει μίσθωσης, επιτρέποντας της έτσι να αντικατοπτρίζει με μεγαλύτερη ακρίβεια ποιες υπηρεσίες είναι διαθέσιμες τη δεδομένη στιγμή. Όταν εγγράφονται ή διαγράφονται υπηρεσίες στην υπηρεσία αναζήτησης, δημιουργούνται γεγονότα και εκπέμπονται ειδοποιήσεις σε όλα τα αντικείμενα στο σύστημα, τα οποία έχουν δηλώσει ενδιαφέρον.

Οι υπηρεσίες και τα πρωτόκολλα της υποδομής, υλοποιούν ένα σύνολο από διεπαφές, οι οποίες καθορίζουν τον τρόπο με τον οποίο επικοινωνούν οι υπηρεσίες με την υποδομή του συστήματος. Σε αυτές τις διεπαφές συμπεριλαμβάνονται:

- Η διεπαφή μίσθωσης (leasing interface) η οποία καθορίζει τον τρόπο με τον οποίο κατανέμονται και απελευθερώνονται πόροι (υπηρεσίες) στο σύστημα, βάση ενός μοντέλου που στηρίζεται σε χρονικούς περιορισμούς.
- Οι διεπαφές γένεσης γεγονότων και εκπομπής ειδοποιήσεων (event and notification interfaces), που αποτελούν επέκταση του μοντέλου ειδοποιήσεων των JavaBeans σε ένα κατανεμημένο περιβάλλον.
- Οι διεπαφές εκτέλεσης συναλλαγών (transaction interfaces), τα οποία επιτρέπουν σε οντότητες να χειρίζονται μια σειρά από ενέργειες σαν μια εντολή που είτε θα έχει συνολικό αποτέλεσμα, είτε δεν θα έχει κανένα αποτέλεσμα.

Η διεπαφή μίσθωσης, προσθέτει χρονικό περιορισμό, στην υπόθεση ότι μια εγγεγραμμένη υπηρεσία είναι ενεργή, αναγκάζοντάς την να ανανεώνει την εγγραφή της σε τακτά χρονικά διαστήματα. Με αυτό τον τρόπο αντιμετωπίζονται οι αστοχίες του

δικτύου, ή ακόμα και των μηχανημάτων που προσφέρουν τις υπηρεσίες, αφού σε περίπτωση σφάλματος δεν θα ανανεωθεί η μίσθωση και θα πάψει να παρουσιάζεται η υπηρεσία ως διαθέσιμη ενώ δεν είναι.

Οι διεπαφές γένεσης γεγονότων και αποστολής ειδοποιήσεων καταρχήν επεκτείνουν τον μηχανισμό ειδοποιήσεων των JavaBeans σε ένα σύστημα από Java Virtual Machines που επικοινωνούν μέσω δικτύου. Επιτρέπουν τον χειρισμό γεγονότων από απομακρυσμένα αντικείμενα ενώ διαθέτουν ένα μηχανισμό αξιόπιστης αποστολής και λήψης ειδοποιήσεων. Σοβαρά υπόψη στο σχεδιασμό λαμβάνεται και το γεγονός ότι μια ειδοποίηση μπορεί να καθυστερήσει σημαντικά να φθάσει στον αποδέκτη.

Οι διεπαφές συναλλαγών, αφορούν τον συντονισμό των μεταβολών κατάστασης στις εφαρμογές Jini. Συγκεκριμένα το πρωτόκολλο συναλλαγών, καθορίζει ότι οι ενέργειες ενός συνόλου κατανεμημένων αντικειμένων οριστικοποιούνται σε δύο βήματα. Το πρώτο βήμα είναι η φάση ψηφοφορίας (voting phase), όπου το κάθε αντικείμενο «ψηφίζει» κατά πόσο έχει ολοκληρώσει το μέρος της εργασίας που του αναλογεί, και δηλώνει ετοιμότητα να οριστικοποιήσει τα αποτελέσματα των ενεργειών του. Για παράδειγμα, μια υπηρεσία μεταβάλει μια εγγραφή σε μια βάση δεδομένων, και δηλώνει ετοιμότητα να οριστικοποιήσει τις αλλαγές. Σαν δεύτερο βήμα, ο συντονιστής της ενέργειας (ο πελάτης ή κάποιος διαχειριστής συναλλαγών – transaction manager), αποστέλλει στα κατανεμημένα αντικείμενα εντολή να δεσμεύσουν τις αλλαγές. Αν ένα μέρος της εργασίας αποτύχει να εκτελεστεί ή να ολοκληρωθεί, ο συντονιστής μπορεί να ζητήσει από τις υπηρεσίες να αναιρέσουν τις ενέργειές τους.

Σε ένα σύστημα Jini, η ορθή υλοποίηση των λειτουργιών εκτέλεσης συναλλαγών (κυρίως το μέρος της αναίρεσης ενεργειών), αφήνεται σε αυτόν που υλοποιεί την υπηρεσία ή τα απομακρυσμένα αντικείμενα χωρίς να υπάρχει η δυνατότητα για ενιαία αντιμετώπιση η έλεγχο των συστατικών του συστήματος. Σκοπός του πρωτοκόλλου συναλλαγών Jini, είναι ο συντονισμός των αλληλεπιδράσεων μεταξύ των οντοτήτων και όχι η επιβολή συγκεκριμένης μεθοδολογίας.

Μια υπηρεσία Jini, δεν είναι απαραίτητο να χρησιμοποιεί το προγραμματιστικό μοντέλο όσο αφορά τη λειτουργικότητα που προσφέρει στον πελάτη. Το προγραμματιστικό μοντέλο είναι δεσμευτικό, μόνο σε ότι έχει να κάνει με την επικοινωνία της υπηρεσίας με την υποδομή του συστήματος Jini. Για παράδειγμα, μια υπηρεσία, για να εγγραφεί σε μια υπηρεσία αναζήτησης, χρησιμοποιεί αναγκαστικά τις διαδικασίες της μίσθωσης και επικοινωνεί μέσω RMI. Το κατά πόσο όμως η υπηρεσία θα προσφέρει την λειτουργικότητά της στους πελάτες, βάσει του ιδίου μοντέλου μίσθωσης ή όχι, ή η μέθοδος επικοινωνίας είναι ζήτημα που αφορά αποκλειστικά τον παροχέα της υπηρεσίας και όχι το σύστημα Jini.

Ο δεσμός μεταξύ της υποδομής και της υλοποίησης του προγραμματιστικού μοντέλου στις υπηρεσίες, είναι το στοιχείο που χαρακτηρίζει μια κοινότητα υπηρεσιών και καταναλωτών ως σύστημα Jini, παρά σε μια απλή ομοσπονδία οντοτήτων. Η όλη διαδικασία απλοποιείται από την στενή σχέση που υπάρχει μεταξύ της υποδομής, του προγραμματιστικού μοντέλου και των υπηρεσίων.

2.3.5.3 Υπηρεσίες

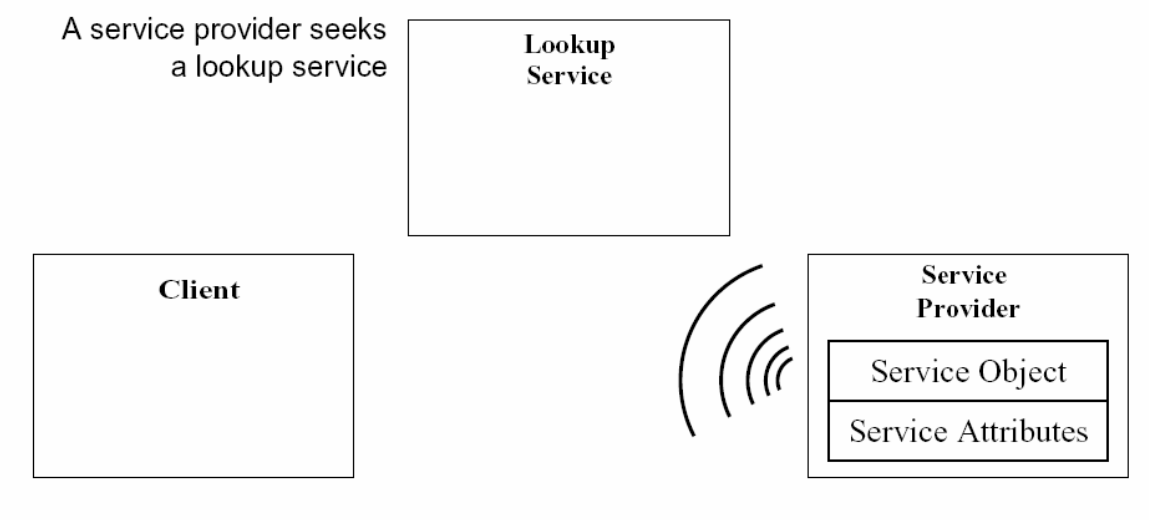
Η υποδομή και το προγραμματιστικό μοντέλο της τεχνολογίας Jini, έχουν ως σκοπό να δώσουν τη δυνατότητα σε υπηρεσίες, να διαθέτουν τη λειτουργικότητά τους μέσω δικτύου. Αυτές οι υπηρεσίες χρησιμοποιούν την υποδομή του Jini για να μπορούν να ανακαλύπτουν η μια την άλλη, να μπορούν να ανακοινώνουν την παρουσία τους σε άλλες υπηρεσίες και πελάτες και για να μπορούν να επικοινωνούν.

Οι υπηρεσίες είναι αντικείμενα Java, τα οποία διαθέτουν από μία διεπαφή που καθορίζει τις μεθόδους που μπορεί να καλέσει ή να χρησιμοποιήσει μια τρίτη οντότητα. Μια διεπαφή μπορεί να προορίζεται να χρησιμοποιηθεί από άλλα προγράμματα, ή μπορεί να προορίζεται για αλληλεπίδραση με κάποιο χρήστη. Οι διαθέσιμες μέθοδοι που μπορούν να χρησιμοποιηθούν για την κλήση μιας υπηρεσίες δηλώνονται από την σχετική διεπαφή και εξαρτώνται αποκλειστικά από το είδος της υπηρεσίας. Μια υπηρεσία μπορεί να υλοποιείται χρησιμοποιώντας άλλες διαθέσιμες υπηρεσίες του συστήματος.

2.3.6 Αρχιτεκτονική Υπηρεσιών

2.3.6.1 Πρωτόκολλα Ανακάλυψης και Αναζήτησης

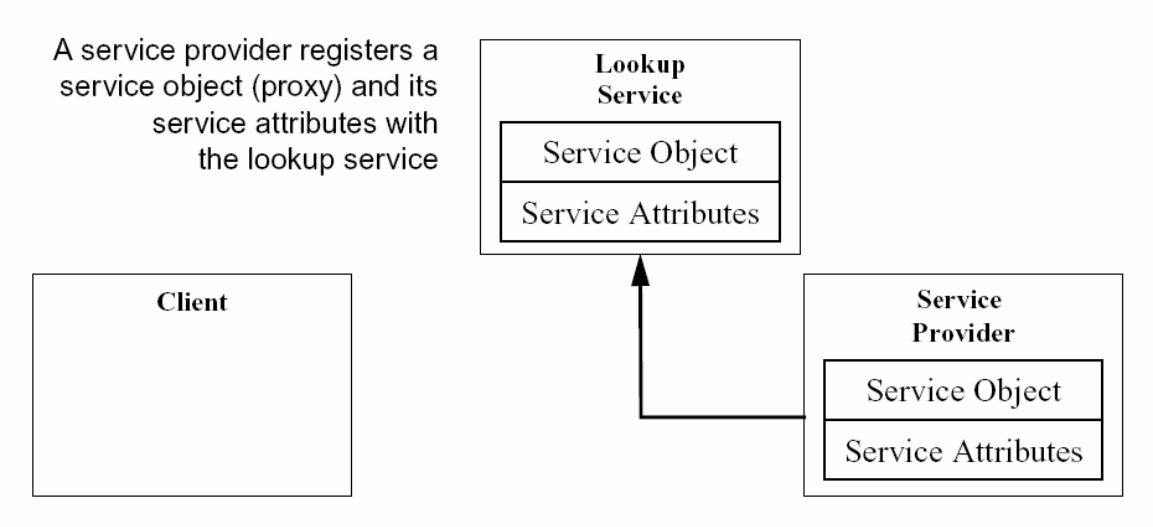
Την καρδία ενός συστήματος Jini, συνιστούν η τριάδα των πρωτοκόλλων ανακάλυψης (discovery), εγγραφής (join) και αναζήτησης (lookup). Τα πρωτόκολλα ανακάλυψης και εγγραφής, χρησιμοποιούνται όταν μια συσκευή ή μια υπηρεσία, πρωτοσυνδέονται σε ένα σύστημα Jini. Η ανακάλυψη συμβαίνει όταν μια υπηρεσία ψάχνει να βρει μια υπηρεσία αναζήτησης για να εγγραφεί. Αφού εντοπιστεί η υπηρεσία αναζήτησης, η υπηρεσία εγγράφεται σε αυτή. Η αναζήτηση συμβαίνει όταν ένας χρήστης ή μια εφαρμογή πελάτη, μέλος του συστήματος, προσπαθεί να εντοπίσει μια υπηρεσία, έχοντας υπόψη τη διεπαφή που αποκαλύπτει, καθώς και κάποια άλλα χαρακτηριστικά (attributes) που αυτή εγγράφει στην υπηρεσία αναζήτησης.



Σχ. 2.3.2 *– Ανακάλυψη (Discovery)*

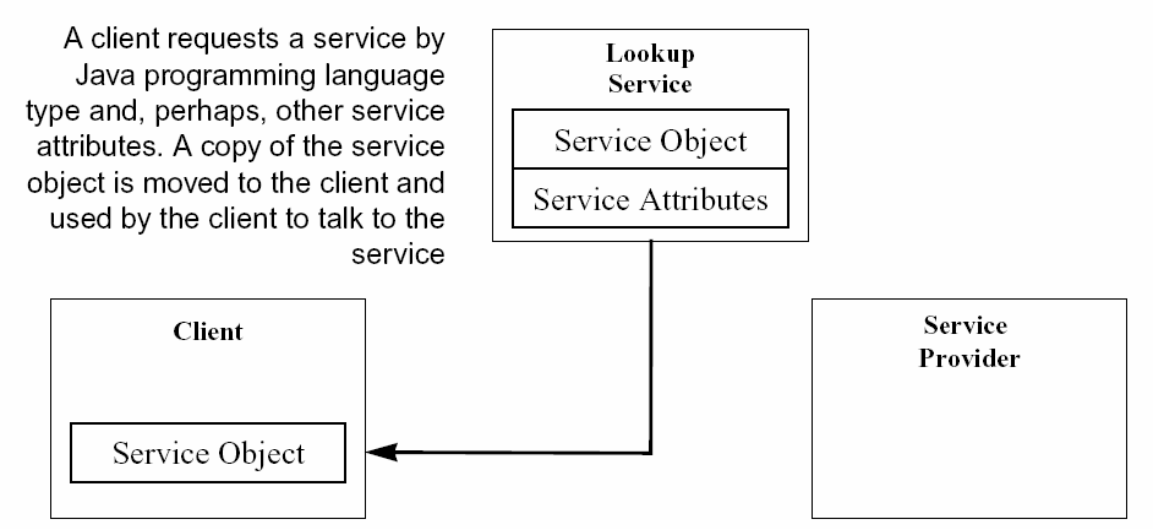
Μια υπηρεσία, εντάσσεται σε μια κοινότητα Jini, μέσω ανακάλυψης και εγγραφής. Αρχικά, ο παροχέας μιας υπηρεσίας, εντοπίζει μια ή περισσότερες υπηρεσίες αναζήτησης. Για να εντοπιστεί η υπηρεσία αναζήτησης, χρησιμοποιείται το πρωτόκολλο multicast, με το οποίο ανακοινώνεται στο τοπικό δίκτυο μια αίτηση προς όλες τις προσβάσιμες υπηρεσίες αναζήτησης να απαντήσουν δηλώνοντας την τοποθεσία τους. Ακολούθως ο παροχέας, αντιγράφει στην υπηρεσία αναζήτησης, ένα αντικείμενο Java

(αντικείμενο αντιπρόσωπος - service object) και μια σειρά από χαρακτηριστικά (attributes) που περιγράφουν την υπηρεσία, και αποκαλύπτουν συγκεκριμένες ιδιότητές της. Το αντικείμενο Java που αντιγράφεται, υλοποιεί την διεπαφή (interface) που αποκαλύπτει η υπηρεσία, που περιλαμβάνει τις μεθόδους που μπορούν να καλέσουν οι καταναλωτές της υπηρεσίας, ώστε να την χρησιμοποιήσουν.



Σχ. 2.3.3 *– Εγγραφή (Join)*

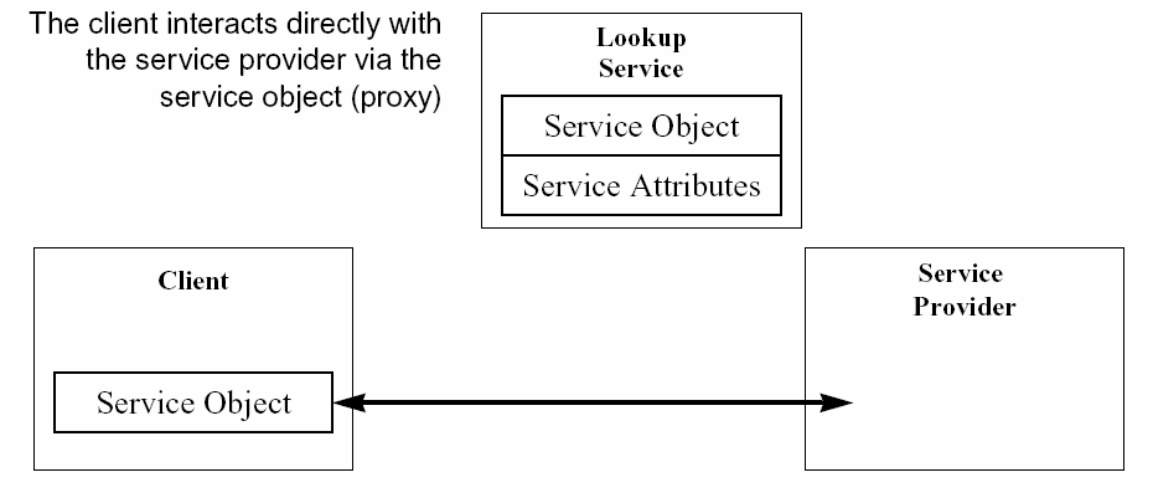
Μια υπηρεσία, πρέπει να είναι σε θέση να ανακαλύπτει μια υπηρεσία αναζήτησης, ή μπορεί να αναθέσει αυτή τη λειτουργία σε κάποιο τρίτο. Η υπηρεσία πλέον έχει εγγραφεί κανονικά στην κοινότητα και μπορεί να την εντοπίσει οποιαδήποτε άλλη οντότητα.



Σχ. 2.3.4 *– Αναζήτηση (Lookup)*

Μια εφαρμογή πελάτης (client) εντοπίζει την κατάλληλη υπηρεσία, εξετάζοντας τον τύπο της, δηλαδή εξετάζοντας την διεπαφή (Java interface) που αποκαλύπτει καθώς και μια σειρά από χαρακτηριστικά (attributes) που τη συνοδεύουν. Αφού εντοπιστεί η κατάλληλη υπηρεσία, η εφαρμογή πελάτης, αντιγράφει το αντικείμενο Java που έχει διαθέσει ο παροχέας της υπηρεσίας, ώστε να αποκτήσει ένα τοπικό αντιπρόσωπο (proxy) της υπηρεσίας.

Το τελευταίο στάδιο είναι η κλήση μεθόδων της υπηρεσίας από τον πελάτη όπως φαίνεται στο Σχ. 2.3.5 πιο κάτω:



Σχ. 2.3.5

Το πρωτόκολλο που χρησιμοποιείται για την επικοινωνία μεταξύ του service object και της υπηρεσίας, μπορεί να είναι οποιοδήποτε διαδεδομένο (XML, RMI, CORBA), ή προσωπικό πρωτόκολλο. Όπως έχει αναφερθεί, διαφορετικές υλοποιήσεις της ίδιας υπηρεσίας μπορούν να χρησιμοποιούν διαφορετικά πρωτόκολλα.

Η δυνατότητα μεταφοράς κώδικα και αντικειμένων από τον παροχέα μιας υπηρεσίας στην υπηρεσία αναζήτησης, κι από εκεί στον πελάτη προσφέρει απόλυτη ελευθερία ως προς την επιλογή της μεθόδου επικοινωνίας μεταξύ υπηρεσίας και πελάτη. Η μετακίνηση κώδικα διασφαλίζει ότι το αντικείμενο αντιπρόσωπος (service object) που διαθέτει κάθε φορά ο πελάτης είναι απόλυτα συγχρονισμένο με την υπηρεσία που εκπροσωπεί, αφού το αντικείμενο έχει τοποθετηθεί εκεί από την ίδια την υπηρεσία. Η μόνη πληροφορία που

χρησιμοποιεί ο πελάτης, είναι το γεγονός ότι το αντικείμενο αυτό υλοποιεί μια συγκεκριμένη διεπαφή. Οι υπόλοιπες λεπτομέρειες της υλοποίησης παραμένουν αδιάφορες.

Ο πελάτης αλληλεπιδρά με την υπηρεσία, μέσω ενός συνόλου από διεπαφές γραμμένων στην γλώσσα Java. Αυτές οι διεπαφές καθορίζουν το σύνολο των διαθέσιμων προς χρήση μεθόδων. Ο πελάτης ζητά από την υπηρεσία αναζήτησης να του επιστρέψει τις υπηρεσίες που υλοποιούν συγκεκριμένη διεπαφή. Με αυτό τον τρόπο διασφαλίζεται ότι ο πελάτης, γνωρίζει τον τρόπο με τον οποίο θα χρησιμοποιήσει την υπηρεσία, αφού γνωρίζει την διεπαφή που θα χρησιμοποιηθεί!

Στην υπηρεσία αναζήτησης, εκτός από το αντικείμενο αντιπρόσωπο της υπηρεσίας, μπορεί να φυλαχθεί υπό τη μορφή χαρακτηριστικού (attribute) και ένα γραφικό περιβάλλον χειρισμού της υπηρεσίας (Graphical User Interface). Αυτή η λειτουργία δίνει τη δυνατότητα να χρησιμοποιηθεί μια υπηρεσία κατευθείαν από τον χρήστη και όχι μέσω μια άλλης εφαρμογής. Ένα τέτοιο παράδειγμα μπορεί να είναι μια υπηρεσία εκτύπωσης μέσω δικτύου, όπου ο χρήστης επιλέγει το αρχείο και τις λεπτομέρειες της εκτύπωσης (έγχρωμη – μαυρόασπρη, draft κ.λ.π.) κατευθείαν μέσω ενός παραθύρου στην οθόνη του.

Στην περίπτωση όπου δεν μπορούν να εντοπιστούν υπηρεσίες αναζήτησης, ένας πελάτης μπορεί να χρησιμοποιήσει μια τεχνική που ονομάζεται peer lookup. Σε ένα τέτοιο σενάριο, ο πελάτης αποστέλλει μέσω δικτύου, multicast πακέτα, παρόμοια με αυτά που στέλλει η υπηρεσία αναζήτησης για να ζητήσει από τις υπηρεσίες να εγγραφούν. Ακολούθως οι υπηρεσίες, στέλλουν απαντήσεις στον πελάτη, επιχειρώντας να εγγραφούν σε αυτόν, σαν να ήταν υπηρεσία αναζήτησης. Ο πελάτης μπορεί ακολούθως να επιλέξει τις υπηρεσίες που χρειάζεται από της αιτήσεις που θα λάβει για εγγραφή, και να απορρίψει τις υπόλοιπες.

2.3.6.2 Υλοποίηση Υπηρεσιών

Μια υπηρεσία συνήθως υλοποιείται από ένα σύνολο από αντικείμενα, τα οποία βρίσκονται σε είτε στην ίδια Java Virtual Machine, είτε εντοπίζονται σε μια κοινή διεύθυνση στο δίκτυο (single address space). Αντικείμενα που ανήκουν σε διαφορετικά σύνολα, είναι εντελώς απομονωμένα το ένα από ο άλλο.

Μια υπηρεσία μπορεί να υλοποιείται είτε εξολοκλήρου, είτε μερικώς από εξειδικευμένο υλικό αντί λογισμικό. Σε τέτοια περίπτωση, παρεμβάλλεται το σχετικό λογισμικό που αναλαμβάνει την επικοινωνία και την σύνδεση με ένα σύστημα Jini.

Από την πλευρά του πελάτη, δεν γίνεται καμία διάκριση μεταξύ των υπηρεσιών που είναι υλοποιημένες ως αντικείμενα σε ένα απομακρυσμένο μηχάνημα, των υπηρεσιών που αντιγράφονται εξολοκλήρου τοπικά, ή των υπηρεσιών που προσφέρονται από συσκευές. Εξ όσον οι υπηρεσίες αυτές είναι διαθέσιμες στο δίκτυο, ο πελάτης τις αντιμετωπίζει ως αντικείμενα γραμμένα στην γλώσσα Java. Επίσης, πέραν του ζητήματος της ορθής λειτουργίας μιας υπηρεσίας, μια υλοποίηση μπορεί να αντικατασταθεί εξολοκλήρου από μια άλλη χωρίς να αντιληφθεί οτιδήποτε ο πελάτης.

2.3.7 Προηγμένα Χαρακτηριστικά

2.3.7.1 Εγγραφή στην Υπηρεσία Αναζήτησης

Οι διεπαφές που αφορούν την υλοποίηση του πρωτοκόλλου εγγραφής (join protocol), δίνουν τη δυνατότητα καθορισμού συγκεκριμένων παραμέτρων, που επιτρέπουν την διάκριση μεταξύ των διαφόρων κοινοτήτων που λειτουργούν στο δίκτυο καθώς και την διάκριση μεταξύ των υπηρεσιών που λειτουργούν εντός της ίδιας κοινότητας. Επίσης, επειδή η διαδικασία εγγραφής (κυρίως σε μια υπηρεσία ανακάλυψης) μπορεί να εξελιχθεί σε μια αρκετά επίπονη, πολύπλοκη και διαρκή ενέργεια (λόγω της μίσθωσης), έχουν παρουσιαστεί στην έκδοση 2.0 του Jini δύο βοηθητικές κλάσεις οι οποίες μειώνουν δραματικά την έκταση του κώδικα που αναλαμβάνει αυτή την εργασία.

Groups

Κάθε κοινότητα JINI καθορίζεται από ένα όνομα ώστε να μπορεί να διακριθεί από μια άλλη κοινότητα. Το όνομα αυτό το επιλέγει ο σχεδιαστής της κοινότητας και είναι ένα απλό αλφαριθμητικό (string). Κάθε υπηρεσία αναζήτησης (lookup service), όταν αρχικοποιείται, της δίνεται ως παράμετρος ένας πίνακας από τέτοια group names. Ακολούθως, όταν μια υπηρεσία επιθυμεί να εγγραφεί σε μια υπηρεσία αναζήτησης, εξετάζει εάν το group στο οποίο θέλει να ενταχθεί, εξυπηρετείται από την συγκεκριμένη υπηρεσία αναζήτησης. Εάν ναι τότε εγγράφεται. Ανάλογος έλεγχος γίνεται και κατά την αναζήτηση υπηρεσιών, ώστε ένας πελάτης να ψάχνει εντός της κοινότητας που επιθυμεί τις υπηρεσίες που θέλει να ανακαλύψει. Μια υπηρεσία αναζήτησης μπορεί να είναι γενικού σκοπού, εάν δηλωθεί ότι ανήκει στο group "public" ή εάν της δοθεί το κενό string ως παράμετρος. Μαζί τα αρχεία που διανέμονται με το Jini, διατίθεται και μια υλοποίηση της υπηρεσίας αναζήτησης που ονομάζεται "reggie". Στην περίπτωση του reggie, μια υπηρεσία μπορεί να εγγραφεί ή να αναζητήσει άλλες υπηρεσίες, απλά και μόνο δηλώνοντας το group name. Υπάρχει πάντοτε η ευχέρεια για υλοποίήσεις που θα απαιτούν να προηγηθεί κάποιο authentication πριν εκτελεστεί οποιαδήποτε λειτουργία.

Entries

Έχει αναφερθεί αρκετές φορές πιο πάνω, ότι με το που εγγράφεται μια υπηρεσία σε μια υπηρεσία αναζήτησης, εκτός από το αντικείμενο αντιπρόσωπο (service object), καταχωρεί και μια σειρά από διευκρινιστικά χαρακτηριστικά (attributes). Τα attributes ουσιαστικά είναι αντικείμενα που επεκτείνουν την κλάση net.jini.core.entry.Entry και τα οποία περιέχουν μια σειρά από ζεύγη «attrName = value». Τα ονόματα και οι τιμές δεν είναι απαραίτητα τύπου string. Οι βασικές βιβλιοθήκες του Jini περιέχουν μια σειρά από τέτοια attributes, δίδεται όμως η δυνατότητα στον χρήστη πολύ εύκολα να δημιουργήσει δικά του attributes που να εξυπηρετούν ειδικές ανάγκες. Ο βασικός ρόλος αυτών των χαρακτηριστικών, είναι να ενισχύσουν την δυνατότητα ταυτοποίησης υπηρεσιών κατά τη διάρκεια της αναζήτησης. Χαρακτηριστικά παραδείγματα attributes είναι τα εξής:

- **net.jini.lookup.entry.Name:** Αυτό το χαρακτηριστικό περιέχει μόνο ένα ζεύγος τιμών και αφορά το όνομα της υπηρεσίας, ως γενική περιγραφή.

- **net.jini.lookup.entry.ServiceInfo:** Αυτό το χαρακτηριστικό περιλαμβάνει έξι πεδία τύπου string. Αυτά τα πεδία είναι manufacturer (το όνομα του κατασκευαστή της υπηρεσίας), model (το όνομα ή ο αριθμός του μοντέλου της υπηρεσίας), name (το όνομα της υπηρεσίας), serialNumber (ο σειριακός αριθμός της υπηρεσίας), vendor (ο εμπορικός διανομέας της υπηρεσίας) και version (η έκδοση της υπηρεσίας).

Κατά τη διάρκεια της αναζήτησης μια υπηρεσίας, δεν μπορούν να ληφθούν υπόψη πεδία τιμών συγκεκριμένων χαρακτηριστικών παρά μόνο ακριβής τιμές. Δηλαδή ο κάθε πελάτης οφείλει να αναζητήσει υπηρεσίες βάση των χαρακτηριστικών εκείνων για τα οποία διαθέτει την ακριβή τους τιμή, και ακολούθως, αφού αντιγράψει τα χαρακτηριστικά τοπικά, να επιλέξει τις κατάλληλες υπηρεσίες βάση πιο αυστηρών περιορισμών.

JoinManager

Η κάθε υπηρεσία σε μια κοινότητα Jini, έχει ως στόχο να δημοσιεύσει την λειτουργικότητα της με το να εγγραφεί σε τουλάχιστον μια υπηρεσία αναζήτησης. Η υπηρεσία λοιπόν πρέπει να είναι σε θέση να ανακαλύπτει υπηρεσίες αναζήτησης, τόσο μέσω του πρωτοκόλλου multicast, όσο και μέσω του πρωτοκόλλου unicast. Επίσης πρέπει να διαθέτει μια υλοποίηση του πρωτοκόλλου εγγραφής (join) ώστε να μπορεί να εγγραφεί στις υπηρεσίες αναζήτησης που επιθυμεί.

Για να παραμείνει εγγεγραμμένη σε μια υπηρεσία αναζήτησης, μια υπηρεσία θα πρέπει να φροντίζει για την τακτική ανανέωση της κάθε εγγραφής, λόγω της αναμενόμενης λήξης της κάθε μίσθωσης. Επίσης θα πρέπει να είναι σε θέση να διαχειριστεί τα χαρακτηριστικά (attributes) που καταχωρούνται σε κάθε εγγραφή, η τιμές των οποίων ενδέχεται να μεταβληθούν σε κάποιο χρονικό σημείο.

Για να αποφευχθεί η συγγραφή πολύπλοκου κώδικα που να αναλαμβάνει όλες αυτές τις εργασίες, έχει υλοποιηθεί η κλάση **net.jini.lookup.JoinManager** η οποία αναλαμβάνει να διαχειριστεί αποδοτικά, όλες τις πιο πάνω λειτουργίες, δηλ. την ανακάλυψη των

υπηρεσιών αναζήτησης, την εγγραφή σε αυτές, την ανανέωση των μισθώσεων και τη διαχείριση των attributes. Το μόνο που χρειάζεται, είναι να δηλωθούν τα groups που αφορούν την υπηρεσία, τα attributes που τη συνοδεύουν καθώς και το αντικείμενο αντιπρόσωπος (service object).

2.3.7.2 ServiceDiscoveryManager

Ο κάθε χρήστης υπηρεσιών σε μια κοινότητα Jini πρέπει με τη σειρά του να είναι σε θέση να ανακαλύψει υπηρεσίες αναζήτησης στο δίκτυο (χρησιμοποιώντας τόσο το multicast όσο και το unicast πρωτόκολλο), καθώς και να αναζητήσει εκεί τις υπηρεσίες που θέλει να χρησιμοποιήσει, σύμφωνα πάντα με την διεπαφή που αυτές υλοποιούν, και με τα χαρακτηριστικά που τις συνοδεύουν.

Η κλάση **net.jini.lookup.ServiceDiscoveryManager** είναι μια βοηθητική κλάση η οποία επιτρέπει την ανακάλυψη υπηρεσιών και την παρακολούθηση των μεταβολών σε αυτές κατά τη διάρκεια της χρήσης τους, αναλαμβάνοντας εξολοκλήρου τις διαδικασίες που περιγράφηκαν πιο πάνω.

Επειδή η συνεχής αποστολή αιτήσεων για ανακάλυψη υπηρεσιών (αναζήτησης ή άλλων) στο δίκτυο, μπορεί να αποδειχθεί ιδιαίτερα δαπανηρή, δίδεται η δυνατότητα από τον ServiceDiscoveryManager για την δημιουργία ενός τοπικού αντιγράφου (cache) του περιεχομένου της κάθε υπηρεσίας αναζήτησης, ώστε οι συχνές αναζητήσεις να μην μεταφράζονται σε συχνές απομακρυσμένες αιτήσεις.

Δίδεται επίσης η δυνατότητα για την εγκατάσταση ενός συστήματος ειδοποιήσεων, όσο αφορά γεγονότα ανακάλυψη μιας υπηρεσίας, μεταβολής στη διαθεσιμότητά της ή ακόμα και μεταβολής της τιμής κάποιου από τα χαρακτηριστικά που τη συνοδεύουν.

2.3.7.3 Jini Extensible Remote Invocation – JERI

Όπως έχει αναφερθεί στις προηγούμενες παραγράφους, δεν υπάρχει περιορισμός ως προς το πρωτόκολλο επικοινωνίας μεταξύ της υπηρεσίας και του αντιπροσώπου της στο μηχάνημα του πελάτη. Η Java μέχρι τώρα διέθετε μηχανισμούς που αφορούσαν μόνο την

δημιουργία αντιπροσώπων που χρησιμοποιούν RMI για να επικοινωνήσουν με απομακρυσμένα αντικείμενα.

Στην έκδοση 2.0 του Jini, έχουν προστεθεί τα εργαλεία, ώστε να είναι εφικτή η παραγωγή αντικειμένων αντιπροσώπων (proxies) σε πραγματικό χρόνο (on the fly) που να μπορούν όμως να χρησιμοποιούν οποιοδήποτε πρωτόκολλο μεταφοράς εντολών και δεδομένων. Τα πρωτόκολλα που μπορούν να χρησιμοποιηθούν, μπορούν να βασίζονται σε σύνδεση, σε ασύγχρονη αποστολή μηνυμάτων, να είναι πρωτόκολλα internet, ή ακόμα να μην στηρίζονται καθόλου στη στοίβα πρωτοκόλλων TCP/IP.

Αυτό το σκοπό εξυπηρετούν τα εργαλεία που αφορούν το Jini Extensible Remote Invocation. Προσφέρεται ένα ακόμα επίπεδο αφαίρεσης όσο αφορά τον προγραμματισμό και την έκδοση απομακρυσμένων (remote) αντικειμένων, αφού δεν χρησιμοποιούνται απευθείας τα API's του JERI. Η κάθε υπηρεσία έχει επαφή μόνο με το interface Exporter, αγνοώντας ποια υλοποίηση θα χρησιμοποιηθεί τελικά, αφού δίνεται η δυνατότητα να καθοριστεί η υλοποίηση (συνεπώς και το πρωτόκολλο επικοινωνίας) μέσω αρχείων ρυθμίσεων (configuration files). Με αυτό τον τρόπο η όλη επικοινωνία γίνεται με τρόπο διάφανο ως προς τα δύο συμμετέχοντα μέρη.

2.4 Service Oriented Architecture

2.4.1 Εισαγωγή

Για την ανάλυση, τον σχεδιασμό και την υλοποίηση κατανεμημένων συστημάτων μεγάλης κλίμακας, χρειάζεται να ακολουθηθεί συγκεκριμένη μεθοδολογία, ώστε τα διάφορα μέρη του συστήματος, να μπορούν να συνεργάζονται αποδοτικά για την εκτέλεση της κάθε λειτουργίας. Επίσης, είναι απαραίτητη η ύπαρξη τεχνολογιών οι οποίες να επιτρέπουν την επικοινωνία των συστατικών μερών του συστήματος, και ακόμα περισσότερο την συνεργασία ετερογενών και ασύμβατων συστημάτων. Δεδομένου ότι σήμερα υπάρχουν, έχουν δοκιμαστεί και έχουν θεμελιωθεί τέτοιες τεχνολογίες, έχει εισαχθεί η έννοια μιας αρχιτεκτονικής που να έχει ως επίκεντρο τις υπηρεσίες που συναποτελούν το σύστημα. Η προσέγγιση του προβλήματος, υλοποιώντας μια SOA (Service Oriented Architecture), μπορεί να μειώσει την πολυπλοκότητα του λογισμικού, να μειώσει τους χρόνους ανάπτυξης και ελέγχου, να διευκολύνει τη συντήρηση και τη διαχείριση, και τέλος να ενθαρρύνει την επαναχρησιμοποίηση κώδικα και παλαιότερων συστημάτων.

2.4.2 Ορισμός

Σκοπός της αρχιτεκτονικής αυτής είναι να συσχετίσει με δυναμικό τρόπο διάφορες λειτουργικές οντότητες σε ένα δίκτυο, μέσω καλά ορισμένων διεπαφών. Με τον όρο διεπαφή εννοούμε τα χαρακτηριστικά των «σημείων επαφής» μέσω των οποίων μπορεί να συνδεθεί και να επικοινωνήσει κάποια οντότητα με την υπηρεσία. Στην ιδανική περίπτωση, οι διεπαφές ορίζονται με ουδέτερο τρόπο, ανεξάρτητο από πλατφόρμα, λειτουργικό σύστημα ή γλώσσα προγραμματισμού.

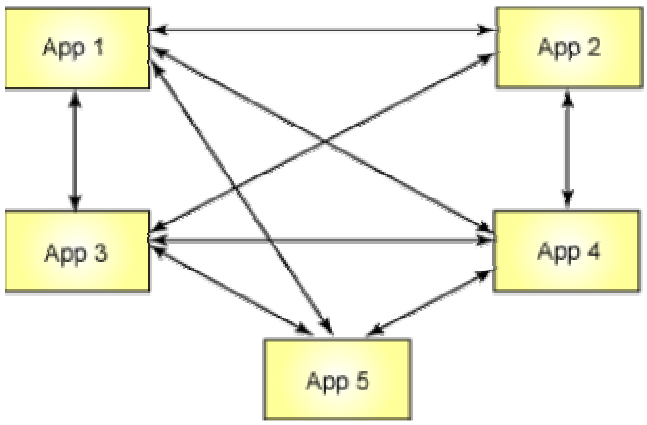
Για την υλοποίηση μιας αρχιτεκτονικής προσανατολισμένης στις υπηρεσίες, χρησιμοποιούνται υλοποιημένες υπηρεσίες και εφαρμογές, ενωμένες στο δίκτυο. Σε αυτές τις υπηρεσίες, δίνεται η δυνατότητα να δημοσιεύσουν τη λειτουργικότητά τους ώστε να μπορούν να τις ανακαλύψουν άλλες υπηρεσίες στο δίκτυο και τελικά να μπορούν να τις χρησιμοποιήσουν.

Συνοψίζοντας, μπορούμε να πούμε ότι πρόκειται για μια αρχιτεκτονική όπου η κάθε λειτουργία υλοποιείται από μια ανεξάρτητη και αυτοτελή υπηρεσία η οποία διαθέτει μια καλά ορισμένη διεπαφή, και όπου οι υπηρεσίες μπορούν να κληθούν με βάση κάποια καθορισμένη ακολουθία ώστε να σχηματίσουν μια διεργασία που εκτελεί κάποια εργασία.

2.4.3 Περιγραφή Προβλήματος

Σε ένα κατανεμημένο σύστημα, το κάθε συστατικό μέρος, πρέπει με κάποιο τρόπο να ενημερωθεί για την ύπαρξη των υπολοίπων μονάδων, και το που βρίσκονται στο δίκτυο ώστε να μπορεί να ενωθεί μαζί τους και να χρησιμοποιήσει τις υπηρεσίες τους. Η μέχρι τώρα προσέγγιση προέβλεπε ότι αυτές οι συνδέσεις γίνονται στατικά, δηλαδή στο κάθε κομμάτι του συστήματος γίνονταν χειρωνακτικά ρυθμίσεις ως προς το που θα βρει τι, και μέσω ποιας διεπαφής θα επιτευχθεί η επικοινωνία. Αυτό αριθμητικά μεταφράζεται στο ότι εάν ένα σύστημα διαθέτει n τμήματα κατανεμημένα στο δίκτυο, χρειάζεται να δημιουργηθούν χειρωνακτικά n(n-1) συνδέσεις. Σε περίπτωση δε που θα πρέπει να συνδεθεί ακόμα μια υπηρεσία στο σύστημα, χρειάζεται να δημιουργηθούν 2n νέες συνδέσεις, οι οποίες πρέπει να εγκατασταθούν, να δοκιμαστούν, να τεκμηριωθούν και ακόμη δυσκολότερα. να συντηρηθούν. Χρειάζεται επίσης αρκετές φορές να γίνουν αλλαγές στον κώδικα και των υφιστάμενων λειτουργικών τμημάτων του συστήματος. Η εικόνα αυτή αποτυπώνεται στο Σχ. 2.4.1, και είναι προφανές ότι με την αύξηση του αριθμού των μερών που συμμετέχουν, αυξάνεται τρομερά η πολυπλοκότητα του συστήματος. Η δε διασύνδεση συστημάτων πολλές φορές κρίνεται ως απαγορευτική.

Μέχρι στιγμής, έχουν αναπτυχθεί και θεμελιωθεί αρκετές τεχνολογίες οι οποίες κατά κύριο λόγο, αντιμετωπίζουν το πρόβλημα της διασύνδεσης ετερογενών συστημάτων, καθώς και τη δυνατότητα μεταφοράς ολόκληρων αντικειμένων μέσω δικτύου, και μεταξύ διαφόρων γλωσσών προγραμματισμού και λειτουργικών συστημάτων. Αυτό το σύνολο των τεχνολογιών, είναι γνωστό σήμερα σαν *web-services*, και θεωρείται ως το υπόβαθρο πάνω στο οποίο μπορεί να στηθεί ένα σύστημα SOA, αν και ως προσέγγιση σχεδιασμού συστημάτων, είναι παντελώς ανεξάρτητη από επιμέρους τεχνολογίες.



Σχ. 2.4.1

Ως web-services εννοούμε τα πρότυπα XML (eXtensible Markup Language), SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language), και UDDI (Universal Description, Discovery and Integration), τα οποία δίνουν την δυνατότητα για την περιγραφή διεπαφών και αντικειμένων με ουδέτερο τρόπο, ανεξάρτητο από πλατφόρμα υλοποίησης. Καθορίζουν επίσης μηχανισμούς για την ανταλλαγή μηνυμάτων μεταξύ προγραμμάτων στο δίκτυο, αντιμετωπίζουν όμως πολύ συγκεκριμένα θέματα διασύνδεσης εφαρμογών. Κατά την ανάπτυξη αυτών των τεχνολογιών φαίνεται ότι η κοινότητα λογισμικού έκανε κάποιες παραδοχές, και υπήρξαν προϋποθέσεις που έκαναν το εγχείρημα επιτυχημένο. Μερικές είναι:

- Η ωριμότητα τεχνολογιών δικτύου (π.χ. TCP/IP), εργαλείων μοντελοποίησης (π.χ. Ολοκληρωμένα περιβάλλοντα ανάπτυξης – IDE's, UML), πλατφόρμες όπου η διαλειτουργικότητα είναι θεμελιώδης παράμετρος (J2EE) και μεθοδολογίες (Object Oriented). Όλα αυτά συνιστούν την υποδομή που διευκολύνει την επικοινωνία μεταξύ μηχανημάτων και την υλοποίηση χαλαρά συζευγμένων (loose coupled) αλληλεπιδράσεων μεταξύ τους. Με τον όρο loose-coupling αναφερόμαστε κυρίως στην ουδετερότητα ως προς την πλατφόρμα και την γλώσσα προγραμματισμού, της δήλωσης των διεπαφών.
- Η αποδοχή και η ωρίμανση της ιδέας ότι σε ένα δικτυακό περιβάλλον είναι απαραίτητη η επίτευξη της διαλειτουργικότητας των εφαρμογών για να υπάρξει συνεργασία κατανεμημένων συστημάτων.

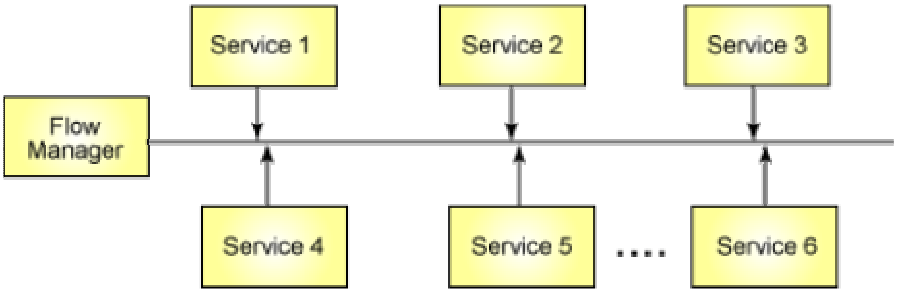
- Συναίνεση στο ότι για την επίτευξη χαμηλού κόστους διαλειτουργικότητας, χρειάζονται ανοικτά πρότυπα βασισμένα στο διαδίκτυο. Συνεπώς είναι απαραίτητη η συνεργασία των μεγάλων εταιριών του χώρου στην ανάπτυξη αυτών των προτύπων.

Μια SOA δεν περιορίζεται στα web-services αν και μάλλον παρέχουν τον πιο απλό τρόπο για μια τέτοια υλοποίηση. Συγκεκριμένα, επιτυγχάνεται η δήλωση διεπαφών με τη χρήση της WSDL και η ανταλλαγή μηνυμάτων και αντικειμένων χρησιμοποιώντας SOAP χωρίς να εισάγεται περιορισμός και εξάρτηση από κάποια συγκεκριμένη γλώσσα ή πλατφόρμα. Μπορούν όμως να χρησιμοποιηθούν και άλλοι τρόποι που επιτρέπουν την ανταλλαγή XML μηνυμάτων μέσω HTTP.

Επίσης τα web-services και η SOA, παρόλη την ανεξαρτησία που έχουν από κάθε γλώσσα προγραμματισμού, διαφαίνεται από τα χαρακτηριστικά τους ότι προκρίνουν την γλώσσα Java ως φυσική επιλογή για κάθε υλοποίηση. Η γλώσσα Java είναι μεταφέρσιμη σε πολλά διαφορετικά μηχανήματα και λειτουργικά συστήματα. Μπορούν να δηλωθούν interfaces με ένα καλά ορισμένο και πλήρη τρόπο και διαθέτει εργαλεία για σύνδεση και επικοινωνία μέσω πολλών διαφορετικών πρωτοκόλλων.

2.4.4 Συστατικά μιας SOA

Στην εικόνα του Σχ 2.4.1, έχοντας ως δεδομένη την υποστήριξη των τεχνολογιών που περιλαμβάνουν τα web-services, αλλά και τη γλώσσα Java, χρειάζεται μεταξύ των συστατικών μερών του συστήματος να εξασφαλιστεί αξιόπιστος έλεγχος της διαδικασίας ανταλλαγής μηνυμάτων ώστε το περιβάλλον να παρέχει ασφάλεια, ενιαία πολιτική σε θέματα ρυθμίσεων και δυνατότητα παρακολούθησης.

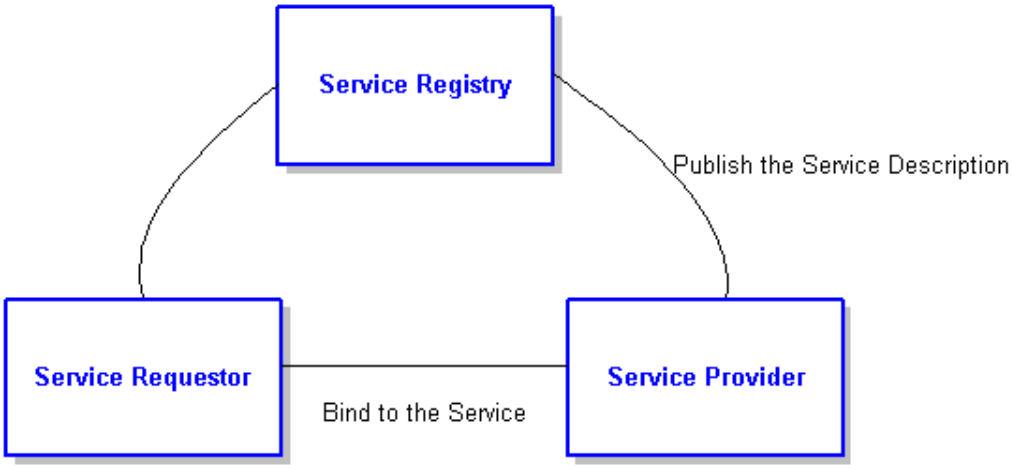


Σχ.2.4.2

Εντάσσεται δηλαδή στο σύστημα η έννοια του «διαύλου υπηρεσιών» (Service Bus), ο οποίος εκτελεί έλεγχο ροής των μηνυμάτων και πιθανότατα μετατροπή των μηνυμάτων ώστε να μπορούν να υλοποιηθούν διαφορετικά σχήματα ανταλλαγής μηνυμάτων. Η εικόνα του συστήματος τώρα μετατρέπεται σε αυτή του Σχ. 2.4.2.

Συγκρίνοντας τα σχήματα 2.4.1 και 2.4.2, παρατηρούμε αρχικά την μεγάλη διαφορά στον τρόπο σύνδεσης της κάθε υπηρεσίας με το σύστημα. Βλέπουμε ότι η κάθε υπηρεσία, αποκαλύπτει μόνο μία διεπαφή στις υπόλοιπες υπηρεσίες, και όλες οι συνδέσεις γίνονται σε ένα κεντρικό σημείο. Παρακάτω θα φανεί ότι αυτό το σημείο μπορεί να είναι περισσότερο από ένα μηχάνημα στο δίκτυο. Ο έλεγχος ροής μπορεί να είναι κατανεμημένος. Ο δίαυλος υπηρεσιών (service bus) δεν εκτελεί μόνο έλεγχο, ούτε αφορά μόνο την διακίνηση μηνυμάτων. Στον ορισμό της αρχιτεκτονικής πιο πάνω, περιγράφονται δύο λειτουργίες οι οποίες δεν έχουν ενταχθεί στην μέχρι τώρα περιγραφή.

Σημαντική προσθήκη της SOA σε σχέση με προηγούμενες προσεγγίσεις, είναι η δυνατότητα που δίνεται σε κάθε υπηρεσία να δημοσιεύσει τη λειτουργικότητά της, καθώς και η δυνατότητα να αναζητήσει και να ανακαλύψει άλλες υπηρεσίες τις οποίες χρειάζεται. Οι υπηρεσίες της δημοσίευσης και της αναζήτησης προσφέρονται από το service bus, και ουσιαστικά αποτελούν μέρος της υποδομής που απαιτείται για την υλοποίηση μιας SOA. Στο Σχήμα 2.4.3 φαίνεται η σχέση μεταξύ των διαφόρων τμημάτων της αρχιτεκτονικής και οι αλληλεπιδράσεις μεταξύ τους.



Σχ. 2.4.3

Όπως φαίνεται στο σχήμα 2.4.3, εισάγεται ακόμα μια λειτουργική μονάδα, το μητρώο υπηρεσιών (service registry). Όταν μια υπηρεσία ενώνεται με το σύστημα, οφείλει να εγγράφεται στο μητρώο υπηρεσιών. Η εγγραφή γίνεται είτε με δυναμικό και αυτόματο τρόπο, είτε χειρωνακτικά από τον διαχειριστή και συντηρητή του συστήματος. Στη δεύτερη περίπτωση, η προσθήκη αφορά την εγκατάσταση μιας μόνο σύνδεσης, στο μητρώο αφήνοντας τις υπόλοιπες υπηρεσίες ανέπαφες. Το μητρώο υπηρεσιών αποτελεί το κεντρικό αυτό σημείο σύνδεσης που αναφέρθηκε για το service-bus. Στο δίκτυο μπορούν να υπάρχουν εγκατεστημένα πολλά τέτοια μητρώα, αρκεί να είναι προσβάσιμα, είτε μέσω δυναμικής ανακάλυψης, είτε μέσω της εγκατάστασης ενός δικτύου συνδεδεμένων μητρώων.

Ως προς τη λειτουργία, μιας SOA, όπως φαίνεται στο Σχ. 2.4.3, σε κάθε εργασία εμπλέκονται τρεις οντότητες:

- Ο παροχέας μιας υπηρεσίας (service provider), οντότητα υπεύθυνη για την δημοσίευση της περιγραφής της υπηρεσίας στο μητρώο υπηρεσιών.
- Το μητρώο υπηρεσιών (service registry), το οποίο λειτουργεί ως ευρετήριο υπηρεσιών, και δίνει τη δυνατότητα σε κάθε συνδεδεμένη οντότητα να αναζητήσει και να ανακαλύψει υπηρεσίες που χρειάζεται να καταναλώσει.
- Ο αιτητής της υπηρεσίας (service requestor), οντότητα υπεύθυνη για την ανακάλυψη των υπηρεσιών που θα χρησιμοποιήσει και την σύνδεση με αυτές ώστε να μπορεί να τις καταναλώσει.

Οι σχέσεις αυτών των οντοτήτων αποκαλύπτουν ακολούθως μια σειρά από υπηρεσίες, οι οποίες αποτελούν ουσιαστικά την υποδομή πάνω στην οποία κτίζεται μια SOA. Βασικά συστατικά λοιπόν της αρχιτεκτονικής είναι οι υπηρεσίες (services), τα μηνύματα που ανταλλάσσονται (messages) και η δυναμική ανακάλυψη (dynamic discovery).

2.4.4.1 Υπηρεσίες

Μια υπηρεσία αποκαλύπτεται στο σύστημα σαν ένα κομμάτι με συγκεκριμένη λειτουργικότητα με τα εξής χαρακτηριστικά:

- Διαθέτει μία διεπαφή (interface) με την έννοια του αντικειμένου μέσω του οποίου επιτυγχάνεται η επικοινωνία με την υπηρεσία. Τα χαρακτηριστικά αυτής της διεπαφής ορίζονται με πλήρη και ουδέτερο τρόπο, κατανοητό σε όλη την έκταση του κατανεμημένου συστήματος.
- Μπορεί να εντοπιστεί με δυναμικό και αυτόματο τρόπο και μπορεί να κληθεί από οποιοδήποτε σημείο του συστήματος σαν τοπική υπηρεσία. Υπάρχει δηλαδή απόλυτη διαφάνεια ως προς την τοποθεσία στο δίκτυο.
- Είναι αυτοτελής και έχει τη διαμόρφωση «μαύρου κουτιού» αφού ο καταναλωτής της υπηρεσίας χρησιμοποιεί μόνο τη δημοσιευμένη διεπαφή, αγνοώντας της λεπτομέρειες της υλοποίησης.

2.4.4.2 Μηνύματα

Οι παροχείς και οι καταναλωτές υπηρεσιών, επικοινωνούν μεταξύ τους ανταλλάσσοντας μηνύματα. Η διεπαφή του κάθε συστατικού τμήματος καθορίζει τη συμπεριφορά του και τη μορφή των μηνυμάτων που αποδέχεται και που αποστέλλει. Τα μηνύματα αυτά πρέπει να είναι διατυπωμένα σε μια γλώσσα αναγνώσιμη από όλα τα συστατικά μέρη, ανεξαρτήτως των λεπτομερειών υλοποίησής τους. Για να υπάρχει δε η δυνατότητα προσθήκης νέων υπηρεσιών, αυτή η γλώσσα πρέπει να είναι ευέλικτη και επεκτάσιμη ώστε να μην περιορίζει τη δυνατότητα επικοινωνίας και να επιτρέπει τον σχεδιασμό μηνυμάτων με μορφή πρότερα άγνωστη. Σε αντίθετη περίπτωση για κάποια ουσιαστική αναβάθμιση, θα ήταν απαραίτητο να αλλάξει εντελώς ο τρόπος σύνταξης των μηνυμάτων, κάτι που μεταφράζεται σε επανασχεδιασμό όλων των διεπαφών του

συστήματος. Φυσική επιλογή για την σύνταξη αυτών των μηνυμάτων αποτελεί η γλώσσα XML, η οποία κληρονομείται από τα web-services. Η XML προσφέρει την απαραίτητη διαλειτουργικότητα, και την απαραίτητη επεκτασιμότητα.

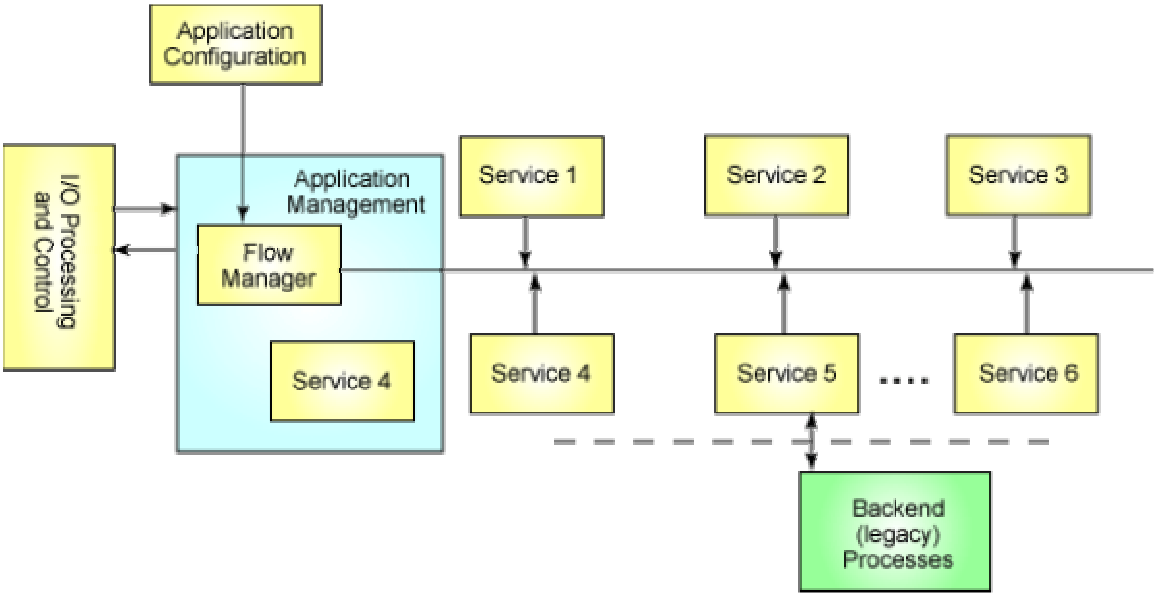
2.4.4.3 Δυναμική Ανακάλυψη

Η ουσιαστική προσθήκη της προσέγγισης SOA είναι η δυνατότητα δυναμικής ανακάλυψης υπηρεσιών. Η δυνατότητα αυτή προσφέρεται από το μητρώο υπηρεσιών, στο οποίο οφείλουν να εγγραφούν όλοι οι παροχείς υπηρεσιών, και στο οποίο απευθύνουν ερωτήσεις οι καταναλωτές υπηρεσιών για να τις εντοπίσουν. Η ύπαρξη του μητρώου υπηρεσιών προσδίδει στην αρχιτεκτονική τα πιο κάτω χαρακτηριστικά:

- Επεκτασιμότητα υπηρεσιών αφού μπορούν να προστεθούν δυναμικά υπηρεσίες
- Αποσυνδέει τους παροχείς υπηρεσιών από τους καταναλωτές, αφού οι συνδέσεις μεταξύ τους εγκαθίστανται μόνο κατά τη διάρκεια της χρήσης της υπηρεσίας και δεν είναι στατικές.
- Επιτρέπει την εύκολη αναβάθμιση υπηρεσιών χωρίς να χρειάζονται ξεχωριστές ενέργειες για την ενημέρωση των καταναλωτών.
- Επιτρέπει στους καταναλωτές υπηρεσιών να επιλέγουν ποιους παροχείς θα χρησιμοποιήσουν σε πραγματικό χρόνο (π.χ. ανάλογα με το φόρτο εργασίας στον καθένα).

2.4.5 Ανάπτυξη Εφαρμογών σε μια SOA

Μέχρι τώρα έχουν αναλυθεί τα συστατικά μιας SOA, καθώς και οι υπηρεσίες που συναποτελούν την υποδομή για την ανάπτυξή της. Δεν έχει γίνει όμως μέχρι τώρα λόγος για το πώς αναπτύσσονται εφαρμογές σε ένα περιβάλλον SOA, και το πώς εκτελούνται εργασίες οι οποίες συνήθως αποτελούνται από μια σειρά από μικρότερες διεργασίες.



Σχ. 2.4.4

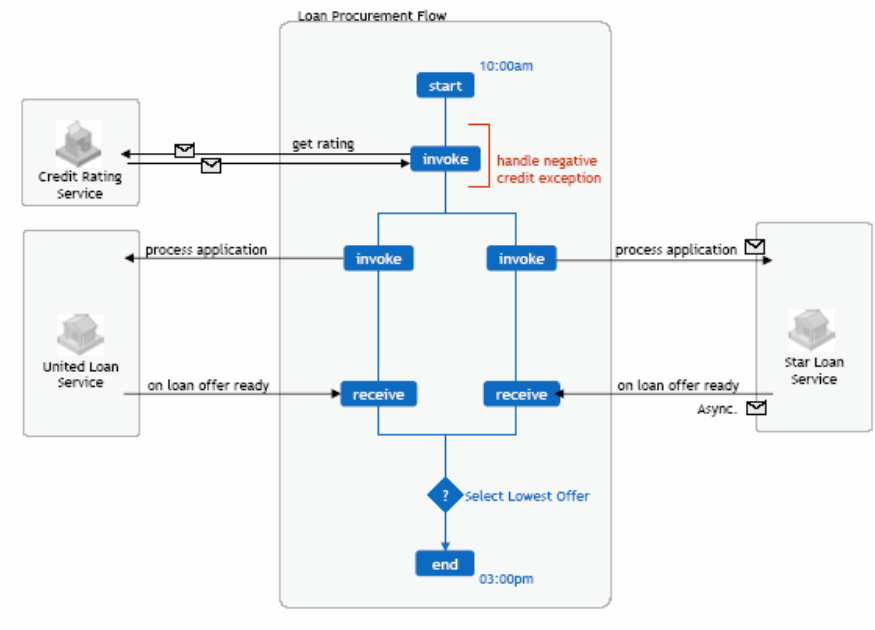
Στο Σχ. 2.4.2, έχει εξασφαλιστεί η επικοινωνία μεταξύ των υπηρεσιών και η δυναμική ανακάλυψή τους. Πρέπει λοιπόν να ενταχθεί και μια εφαρμογή χρήστης του όλου συστήματος, η οποία θα προσφέρει είσοδο δεδομένων, συντονισμένο χειρισμό υπηρεσιών για επεξεργασία και έξοδο αποτελεσμάτων. Η θέση αυτής της εφαρμογής φαίνεται στο Σχ. 2.4.4, το οποίο πλέον απεικονίζει ένα ολοκληρωμένο σύστημα βασισμένο σε SOA.

Η κάθε εφαρμογή, εκτελεί μια σειρά από εργασίες για να εκπληρώσει το σκοπό της. Η κάθε εργασία μπορεί να αφορά πρόσβαση σε βάσεις δεδομένων, επεξεργασία δεδομένων, να χειρίζεται μια συσκευή ή να έχει άλλη μορφή. Για να συμπληρωθεί μια εργασία, χρειάζεται να χειριστεί μια σειρά από υπηρεσίες, η συνδυασμένη λειτουργικότητα των οποίων παράγει το επιθυμητό αποτέλεσμα. Πυρήνας της εφαρμογής, όπως φαίνεται στο σχήμα 2.4.4, είναι ο διαχειριστής ροής (flow manager). Σε ένα σύστημα μπορούν να υπάρχουν πολλές τέτοιες εφαρμογές που αποτελούν τους ουσιαστικούς χρήστες ή καταναλωτές του συστήματος.

Ο διαχειριστής ροής επεξεργάζεται προκαθορισμένες ροές εκτέλεσης υπηρεσιών. Αυτές οι ροές μπορούν να προσθαφαιρεθούν ή να μεταβληθούν σε πραγματικό χρόνο. Η κάθε

εργασία σχεδιάζεται αρχικά ως ένα διάγραμμα ροής το οποίο παρουσιάζει την ακολουθία με την οποία πρέπει να εκτελεστεί η κάθε υπηρεσία για να παραχθεί το τελικό αποτέλεσμα. Συστατικά του διαγράμματος αυτού πρέπει να είναι υπηρεσίες που ήδη υπάρχουν στο σύστημα και οι οποίες μπορούν να ανακαλυφθούν δυναμικά. Ακολούθως ο flow manager δίνει τις σχετικές εντολές. Δεν υπάρχουν περιορισμοί για το διάγραμμα, αφού μπορούν να εκτελούνται λειτουργίες παράλληλα, δηλαδή αυτό μπορεί να έχει πολλούς κλάδους.

Στο Σχ. 2.4.5 παρουσιάζεται ένα απλό σενάριο στο οποίο συμμετέχουν τρεις υπηρεσίες, όπου παρουσιάζεται η ανάγκη για παράλληλη κλήση κάποιων υπηρεσιών. Στο σενάριο καλείται αρχικά με σύγχρονο τρόπο μια διεργασία, από την οποία αναμένεται ένα αποτέλεσμα. Σε αυτή την περίπτωση χρειάζεται πρόνοια για την αντιμετώπιση λαθών που μπορεί να εμφανιστούν είτε στην επικοινωνία, είτε στην απόκριση που θα δοθεί.



Σχ. 2.4.5 *– Διάγραμμα ροής*

Ακολούθως καλούνται δύο άλλες υπηρεσίες οι οποίες μπορούν να εκτελούνται ταυτόχρονα αφού είναι ανεξάρτητες. Στο συγκεκριμένο παράδειγμα, η απόκριση της κάθε υπηρεσίας αναμένεται να καθυστερήσει αρκετά. Πρέπει λοιπόν να προβλέπεται ένας μηχανισμός ώστε οι αποκρίσεις να μπορούν να λαμβάνονται ασύγχρονα, ίσως με

την δημιουργία κάποιου γεγονότος ή κάποιας ειδοποίησης. Στο τέλος, αφού ληφθούν τα απαραίτητα αποτελέσματα, γίνεται κάποια επιλογή με βάση τη σύγκριση των αποτελεσμάτων. Παρατηρούμε στο πιο πάνω σχήμα, με ποιο τρόπο ενορχηστρώνονται οι διάφορες λειτουργίες, και πως μπορούν να χρησιμοποιηθούν οι προσφερόμενες υπηρεσίες. Σε ένα διαφορετικό σχήμα, θα μπορούσε να γίνει αναζήτηση όλων των υπηρεσιών κάποιας συγκεκριμένης μορφής (π.χ. αναζήτηση προσφοράς για παροχή δανείου) και να ερωτηθούν όλες οι υπηρεσίες που θα ανακαλυφθούν.

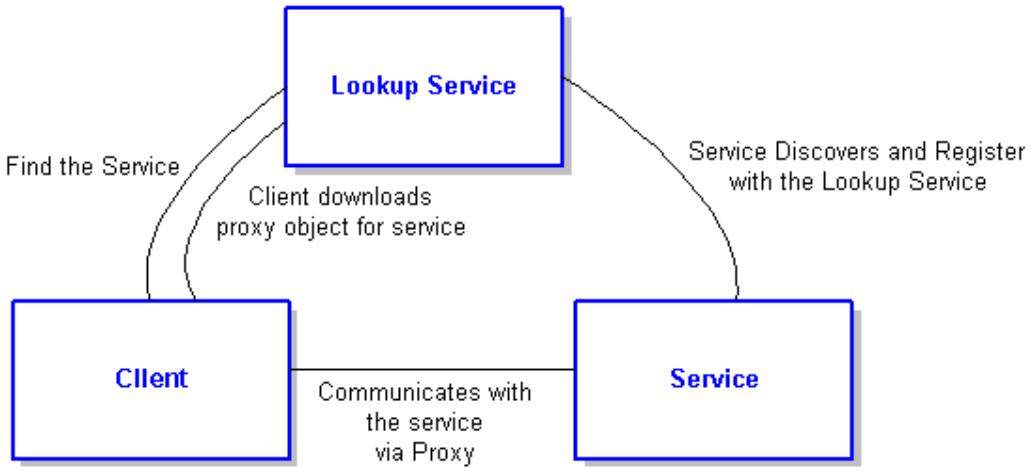
Αρχικά το διάγραμμα ροής που καθορίζει την ακολουθία με την οποία καλούνται οι υπηρεσίες και εκτελούνται οι διάφορες λειτουργίες, αναπαρίσταται χρησιμοποιώντας μια κατάλληλη γλώσσα. Ακολούθως ο flow manager διαβάζει την αναπαράσταση και παράγει τις κατάλληλες κλήσεις στον κατάλληλο χρόνο. Μια τέτοια γλώσσα αναπαράστασης διαγραμμάτων ροής, είναι η BPEL (Business Process Execution Language), η οποία φυσικά είναι μια αναπαράσταση XML. Παρουσίαση της σύνταξης BPEL θα γίνει πιο κάτω. Το XML κείμενο μπορεί να αλλάξει χωρίς να χρειάζεται να ενοχληθούν οι εγγεγραμμένες υπηρεσίες. Επίσης, σε περίπτωση υψηλού φόρτου εργασίας, μπορούν να προστεθούν σε πραγματικό χρόνο στιγμιότυπα υπηρεσιών με την ίδια λειτουργικότητα, ώστε να επιτευχθεί παράλληλη επεξεργασία για καλύτερη απόδοση.

Η δυνατότητα προσθαφαίρεσης υπηρεσιών, και η δυναμική ανακάλυψή τους, μπορεί να προσφέρει ένα μοντέλο για παράλληλη επεξεργασία, ώστε να μπορεί να σχηματιστεί ένα Grid. Σε ένα Grid, αυτό που συνήθως αποτελεί υπηρεσία, είναι η διάθεση χρόνου εκμετάλλευσης της υπολογιστικής ισχύος ενός μηχανήματος. Η συνήθης πρακτική για την δημιουργία ενός Grid, αφορούσε είτε την κατασκευή μιας συστοιχίας υπολογιστών αφιερωμένων στην παράλληλη επεξεργασία, είτε στην εγκατάσταση συγκεκριμένου software στα συνδεδεμένα στο δίκτυο μηχανήματα. Στην πρώτη περίπτωση το σύστημα είναι συγκεντρωμένο και μπορεί να εποπτεύεται σχετικά εύκολα. Η δεύτερη περίπτωση όμως αφορά χιλιάδες υπολογιστές απλωμένους σε όλο τον πλανήτη, όπου πολύ εύκολα ένα μηχάνημα τίθεται εκτός λειτουργίας, ή αποσυνδέεται, αποσύροντας τις υπηρεσίες του από το Grid. Σε αυτή την περίπτωση, η δυνατότητα που προσφέρει μια Service

Oriented αρχιτεκτονική, για δυναμική προσθαφαίρεση και ανακάλυψη υπηρεσιών γίνεται ιδιαίτερα ελκυστική, όσο αφορά την κατασκευή συστημάτων, προσαρμόσιμων στις αλλαγές.

2.4.6 Σύγκριση με Jini

Παρατηρώντας το Σχ. 2.4.3, μπορούμε να δούμε ότι οι οντότητες που αλληλεπιδρούν σε μια SOA αρχιτεκτονική, υπάρχουν σε πλήρη αντιστοιχία και στην τεχνολογία Jini. Επίσης υπάρχει πλήρης αντιστοιχία όσο αφορά τον ρόλο που επιτελεί η κάθε οντότητα.



Σχ. 2.4.6

Συγκρίνοντας την service oriented αρχιτεκτονική με τη μορφή ενός συστήματος βασισμένου στην τεχνολογία Jini, προκύπτει το συμπέρασμα ότι μια κοινότητα Jini, αποτελεί υλοποίηση του service oriented μοντέλου. Το Jini προσφέρει τις υπηρεσίες της δυναμικής προσθαφαίρεσης υπηρεσιών, της αναζήτησης και της ανακάλυψης τους και αναλαμβάνει ολοκληρωτικά τα θέματα επικοινωνίας μεταξύ των οντοτήτων που λειτουργούν εντός του συστήματος.

Οι περιορισμοί που προέρχονται από μια τέτοια υλοποίηση, αφορούν την απαίτηση για υλοποίηση του κορμού του συστήματος εξολοκλήρου σε γλώσσα Java, αν και σε τελική ανάλυση η απαιτούμενη διαλειτουργικότητα επιτυγχάνεται.

2.4.7 BPEL (Business Process Execution Language)

2.4.7.1 Εισαγωγή

Σε πραγματικές συνθήκες, μια υπηρεσία που προσφέρεται μέσω δικτύου, συνήθως απαιτεί τη συνεργασία άλλων υπηρεσιών για να παράξει το απαιτούμενο αποτέλεσμα. Στο παράδειγμα του Σχ. 2.4.5, βλέπουμε το διάγραμμα ροής μιας υπηρεσίας έγκρισης δανείου. Για να απαντήσει το σύστημα εάν και εφόσον εγκρίνεται το δάνειο που έχει ζητήσει ο πελάτης, πρέπει να χρησιμοποιήσει τρεις άλλες υπηρεσίες, οι οποίες μπορεί να προσφέρονται από το σύστημα του ιδίου οργανισμού, είτε από συστήματα άλλων οργανισμών. Το διάγραμμα ροής του σχήματος, βρίσκεται «προγραμματισμένο» στον Flow Manager όπως αυτός παρουσιάζεται στο Σχ. 2.4.4.

Στο παράδειγμα, χρησιμοποιείται μια μάλλον «τοπική» υπηρεσία από την οποία αντλείται το υπόλοιπο και η οικονομική κατάσταση του πελάτη. Ακολούθως ερωτούνται δύο υπηρεσίες έγκρισης δανείων, οι οποίες προφανώς ανήκουν στα δίκτυα δύο διαφορετικών τραπεζών. Αυτές οι υπηρεσίες αποκαλύπτονται υπό την μορφή web-services, ώστε να μπορούν να είναι προσβάσιμες από ετερογενή συστήματα. Από εδώ προκύπτει η σχέση των web-services με την αρχιτεκτονική SOA, καθώς και των επιμέρους τεχνολογιών με την BPEL.

Συγκεκριμένα οι υπηρεσίες σε μια SOA αρχιτεκτονική, σύμφωνα με την υφιστάμενη υποδομή στο δίκτυο, και την μέχρι τώρα εμπειρία στη διασύνδεση συστημάτων, πρέπει να αποκαλύπτονται σαν web-services, οι διεπαφές των υπηρεσιών να δηλώνονται με τη χρήση της WSDL και επικοινωνία να γίνεται με ανταλλαγή XML μηνυμάτων (π.χ. SOAP). Ακολούθως, η ενορχήστρωση (orchestration), των υπηρεσιών για την παραγωγή του επιθυμητού αποτελέσματος, εκτελείται από ένα Flow Manager σύμφωνα πάντα με ένα διάγραμμα ροής. Αυτό το διάγραμμα ροής, είναι διατυπωμένο σε μία XML αναπαράσταση: την BPEL!

2.4.7.2 Ορισμός και περιγραφή της BPEL

Η BPEL προέκυψε από την ανάγκη ύπαρξης ενός κοινού προτύπου για την σύνθεση υπηρεσιών, και πιο συγκεκριμένα για την σύνθεση web-services. Οι πρώτες προσπάθειες έγιναν με την ανάπτυξη των προτύπων WSFL (Web Service Flow Language) και XLANG (Web Services for Business Process Design) από τις εταιρίες IBM και Microsoft αντίστοιχα. Δεδομένου όμως ότι για να είναι πραγματικά χρήσιμη μια τέτοια τεχνολογία, πρέπει να την χαρακτηρίζει η διαλειτουργικότητα, και αφού η προηγούμενη πείρα υποδεικνύει την χρήση ανοικτών προτύπων, οι εταιρείες συνεργάστηκαν για την δημιουργία ενός νέου κοινού προτύπου. Έτσι εμφανίστηκε η έκδοση 1.0. τον Αύγουστο του 2002, οπότε και κατατέθηκε στον οργανισμό ανοικτού λογισμικού OASIS, για την συμπλήρωση των προδιαγραφών, την ολοκλήρωση και την έκδοση της γλώσσας σαν ανοικτό πρότυπο. Το τελικό πρότυπο αναμένεται να εκδοθεί τον Νοέμβριο του 2004 και συνιστά την συνένωση και την πλήρη αντικατάσταση των προτύπων WSFL και XLANG.

Σαν διαδικασία μοντελοποίησης διαδικασιών, η BPEL κληρονομεί την εμπειρία που υπάρχει από την ανάλυση των παραδοσιακών διαγραμμάτων ροής, ενώ χρησιμοποιεί πολλές έννοιες και δυνατότητες από τις εκτελέσιμες γλώσσες προγραμματισμού. Το λεξικό που χρησιμοποιεί σαν γλώσσα, καθώς και οι εργασίες που ορίζει, έχουν ως υπόβαθρο την δήλωση της γλώσσας XML, καθώς και τις περιγραφικές δυνατότητες της WSDL.

Η BPEL έχει τα πιο κάτω χαρακτηριστικά και δυνατότητες:

- Δηλώνει διεργασίες (business processes) με την μορφή συντονισμένων αλληλεπιδράσεων ενός συνόλου από web-services.
- Περιγράφει και δηλώνει τόσο εκτελέσιμες διαδικασίες, όσο και αφηρημένες διεργασίες (abstract processes) που προσφέρονται εξωτερικά και εκτελούνται ανεξάρτητα.
    - Οι εκτελέσιμες διαδικασίες, ορίζονται εξολοκλήρου και περιγράφεται με κάθε λεπτομέρεια η λειτουργικότητα και οι λεπτομέρειες της υλοποίησης. Αυτό σημαίνει ότι η BPEL, περιέχει μηχανισμούς για εκτέλεση πράξεων

(και ανάθεση τιμών), για εκτέλεση επαναλήψεων (for/while), για ελέγχους συνθηκών (if – Boolean statements), για παράλληλη εκτέλεση συναρτήσεων (σε αντιστοιχία με τα Threads) και για είσοδο-έξοδο (input – output). Αυτά τα χαρακτηριστικά είναι που κληρονομούνται από τις εκτελέσιμες γλώσσες προγραμματισμού, και που δηλώνουν την BPEL σαν κάτι περισσότερο από μια γραμματική παράστασης δεδομένων (όπως η απλή XML ή τα μηνύματα SOAP.

- Οι αφηρημένες διεργασίες ουσιαστικά, δεν ορίζονται με βάση τις λεπτομέρειες υλοποίησης, η οποίες μας είναι αδιάφορες. Εκείνο που ορίζεται είναι η διεπαφή (interface) που θα χρησιμοποιηθεί, καθώς και τα μηνύματα που θα ανταλλαγούν ώστε να μεταφερθούν τα δεδομένα εισόδου, και τα αποτελέσματα. Ουσιαστικά, πρόκειται για τον μηχανισμό επικοινωνίας του Flow Manager με απομακρυσμένες υπηρεσίες.

- Επιτρέπει την δημιουργία σύνθετων web-services.

2.4.7.3 Γραμματική BPEL κειμένου

Το XML κείμενο μιας διεργασίας BPEL είναι πολύ στενά συνδεδεμένο με μια σειρά από δηλώσεις WSDL. Για να γίνει κατανοητό, πρέπει αυτός οι δηλώσεις να προηγηθούν της παρουσίασης των συστατικών του BPEL κειμένου.

Συστατικά WSDL

Η WSDL (Web Services Description Language) αποσκοπεί στο να περιγράψει με ουδέτερο (platform independent) τρόπο, τις διεπαφές (interfaces) των υπηρεσιών και τα μηνύματα (messages) που δέχονται ως μέρος μιας αίτησης και αποστέλλουν ως απόκριση.

Κάθε υπηρεσία, διαθέτει μια σειρά από συναρτήσεις-μεθόδους, τις οποίες μπορεί ο καταναλωτής να καλέσει, ώστε να μπορεί να χρησιμοποιήσει την υπηρεσία. Η κάθε μέθοδος, δέχεται κάποια δεδομένα ως παράμετρο για επεξεργασία, και επιστρέφει

κάποια δεδομένα ως απόκριση. Αυτά τα δεδομένα μπορεί να είναι απλοί τύποι (String, Integer), πιο σύνθετοι τύποι (δομές) ή ακόμα και αντικείμενα (objects).

Ένα τυπικό κείμενο WSDL είναι το πιο κάτω και αφορά δηλώσεις για την διάθεση μιας υπηρεσίας διεκπεραίωσης παραγγελιών.

```
<definitions targetNamespace="http://manufacturing.org/wsdl/purchase"
      xmlns:sns="http://manufacturing.org/xsd/purchase"
      xmlns:pos="http://manufacturing.org/wsdl/purchase"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">

<import namespace="http://manufacturing.org/xsd/purchase"
      location="http://manufacturing.org/xsd/purchase.xsd"/>

<message name="POMessage">
      <part name="customerInfo" type="sns:customerInfo"/>
      <part name="purchaseOrder" type="sns:purchaseOrder"/>
</message>

<message name="InvMessage">
      <part name="IVC" type="sns:Invoice"/>
</message>

<!-- portTypes supported by the purchase order process -->
<portType name="purchaseOrderPT">
      <operation name="sendPurchaseOrder">
            <input message="pos:POMessage"/>
            <output message="pos:InvMessage"/>
            <fault name="cannotCompleteOrder"
                  message="pos:orderFaultType"/>
      </operation>
</portType>

<portType name="invoiceCallbackPT">
      <operation name="sendInvoice">
            <input message="pos:InvMessage"/>
      </operation>
</portType>


<plnk:partnerLinkType name="purchasingLT">
      <plnk:role name="purchaseService">
            <plnk:portType name="pos:purchaseOrderPT"/>
      </plnk:role>
</plnk:partnerLinkType>

<plnk:partnerLinkType name="invoicingLT">
      <plnk:role name="invoiceService">
            <plnk:portType name="pos:computePricePT"/>
      </plnk:role>
```

```
      <plnk:role name="invoiceRequester">
            <plnk:portType name="pos:invoiceCallbackPT"/>
      </plnk:role>
</plnk:partnerLinkType>

</definitions>
```

Υπόδειγμα WSDL

Τα σημαντικά συστατικά μιας WSDL δήλωσης, είναι τα <message>,<portType> και <partnerLinkType>. Οι δηλώσεις αυτές χρησιμοποιούνται από την BPEL και επεξηγούνται πιο κάτω.

Με ένα <message> tag δηλώνεται μια δομή δεδομένων, η οποία μπορεί να χρησιμοποιηθεί σαν τύπος μεταβλητής για είσοδο και έξοδο δεδομένων. Για παράδειγμα η δήλωση

```
<message name="POMessage">
      <part name="customerInfo" type="sns:customerInfo"/>
      <part name="purchaseOrder" type="sns:purchaseOrder"/>
</message>
```

αναφέρεται σε μια δομή (message), που ονομάζεται «POMessage», και έχει δύο πεδία (part). Το ένα πεδίο ονομάζεται "customerInfo" και είναι τύπου "sns:customerInfo" ενώ το άλλο ονομάζεται t "purchaseOrder" και είναι τύπου "sns:purchaseOrder". Οι τύποι δεδομένων είναι δηλωμένοι σε ένα αρχείο, που εντοπίζεται εκεί που ορίζεται το namespace «sns».

Με ένα <portType> tag δηλώνεται μια διεπαφή (interface) η οποία με τη σειρά της περιέχει μεθόδους (operations) που διαθέτουν είσοδο για τις παραμέτρους (input) και έξοδο για τα αποτελέσματα (output). Για παράδειγμα η δήλωση

```
<portType name="purchaseOrderPT">
      <operation name="sendPurchaseOrder">
            <input message="pos:POMessage"/>
            <output message="pos:InvMessage"/>
            <fault name="cannotCompleteOrder"
                  message="pos:orderFaultType"/>
      </operation>
</portType>
```

αναφέρεται σε μια διεπαφή με όνομα "purchaseOrderPT", που περιεχει μια μέθοδο με όνομα "sendPurchaseOrder". Η μέθοδος παίρνει ως παράμετρο μια δομή τύπου "pos:POMessage" και επιστρέφει μια δομή τύπου "pos:InvMessage". Σε περίπτωση λάθους κατά την εκτέλεση επιστρέφεται μια δομή τύπου "pos:orderFaultType".

Τέλος δηλώνονται οι τύποι των συνδέσεων μεταξύ συνεργαζόμενων υπηρεσιών μέσω ενός <partnerLinkType> tag. Για την εγκατάσταση μιας τέτοιας σύνδεσης, δηλώνεται ο ρόλος του κάθε συνεργαζόμενου άκρου καθώς και η διεπαφή που το κάθε άκρο αποκαλύπτει. Για παράδειγμα η δήλωση

```
<plnk:partnerLinkType name="invoicingLT">
      <plnk:role name="invoiceService">
            <plnk:portType name="pos:computePricePT"/>
      </plnk:role>
      <plnk:role name="invoiceRequester">
            <plnk:portType name="pos:invoiceCallbackPT"/>
      </plnk:role>
</plnk:partnerLinkType>
```

αναφέρεται σε ένα τύπο σύνδεσης που ονομάζεται "invoicingLT" και στην οποία συμμετέχουν δύο μέρη. Το άκρο της υπηρεσίας ("invoiceService") αποκαλύπτει μια διεπαφή τύπου "pos:computePricePT" ενώ το άκρο του καταναλωτή ("invoiceRequester") αποκαλύπτει μια διεπαφή τύπου "pos:invoiceCallbackPT". Ο λόγος για τον οποίο ο καταναλωτής αποκαλύπτει μια διεπαφή στην υπηρεσία, είναι για να δώσει τη δυνατότητα σε μια υπηρεσία που χρειάζεται πολύ χρόνο για να παράξει αποτέλεσμα, να στείλει την απόκρισή της ασύγχρονα, χωρίς να χρειάζεται να μπλοκάρει ο χρήστης, ενώ βρίσκεται σε αναμονή.

Δομή του BPEL κειμένου

Η βασική δομή ενός κειμένου BPEL, φαίνεται πιο κάτω:

```
<process name="ncname" targetNamespace="uri"
      queryLanguage="anyURI"?
      expressionLanguage="anyURI"?
      suppressJoinFailure="yes|no"?
      enableInstanceCompensation="yes|no"?
      abstractProcess="yes|no"?
      xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">

<partnerLinks>?
      <!-- Note: At least one role must be specified. -->
      <partnerLink name="ncname" partnerLinkType="qname"
            myRole="ncname"? partnerRole="ncname"?>+
      </partnerLink>
</partnerLinks>

<partners>?
      <partner name="ncname">+
            <partnerLink name="ncname"/>+
      </partner>
</partners>

<variables>?
      <variable name="ncname" messageType="qname"?
            type="qname"? element="qname"?/>+
</variables>

<correlationSets>?
      <correlationSet name="ncname" properties="qname-list"/>+
</correlationSets>

<faultHandlers>?
      <!-- Note: There must be at least one fault handler or default. -->
      <catch faultName="qname"? faultVariable="ncname"?>*
            activity
      </catch>
      <catchAll>?
            activity
      </catchAll>
</faultHandlers>

<compensationHandler>?
      activity
</compensationHandler>

<eventHandlers>?
<!-- Note: There must be at least one onMessage or onAlarm handler. -->
      <onMessage partnerLink="ncname" portType="qname"
      operation="ncname" variable="ncname"?>
            <correlations>?
                  <correlation set="ncname" initiate="yes|no"?>+
            <correlations>
```

```
            activity
      </onMessage>
      <onAlarm for="duration-expr"? until="deadline-expr"?>*
            activity
      </onAlarm>
</eventHandlers>

      activity
</process>
```

Βασική Δομή BPEL διεργασίας

Τα attributes που βρίσκονται το άνοιγμα του κειμένου, καθορίζουν τον τρόπο με τον οποίο θα αναγνωσθεί και θα εκτελεστεί η διεργασία. Η ιδιαίτερή τους σημασία είναι η εξής:

- queryLanguage: Σε αυτό το πεδίο καθορίζεται η XML γλώσσα με την οποία γίνεται η επιλογή κόμβων ή ιδιοτήτων κατά την ανάθεση τιμών. Η default τιμή αφορά την γλώσσα XPath1.0 και το URI που χρησιμοποιείται είναι: http://www.w3.org/TR/1999/REC-xpath-19991116
- expressionLanguage: Σε αυτό το πεδίο δηλώνεται η γλώσσα στην οποία είναι διατυπωμένη η διεργασία (process). Και εδώ η default τιμή αφορά την γλώσσα XPath1.0 και το URI που χρησιμοποιείται είναι: http://www.w3.org/TR/1999/REC-xpath-19991116
- supressJoinFaillure: Καθορίζει αν η αποτυχία εκπλήρωσης των περιορισμών που προκύπτουν από τα σημεία όπου συνδέονται δύο κλάδοι στο διάγραμμα ροής, θα εκπέμψει κάποιο σφάλμα, η αν αυτό το σφάλμα θα απορροφηθεί. Τα σημεία σύνδεσης δύο κλάδων αφορούν θέματα συγχρονισμού παράλληλων ροών εκτέλεσης, όταν π.χ. είναι απαραίτητος ο υπολογισμός ενός μεγέθους σε μια ροή πριν την συνέχεια της εκτέλεσης μιας άλλης παράλληλης.
- enableInstanceCompensation: Σε αυτό το πεδίο δηλώνεται αν οι ενέργειες ενός στιγμιοτύπου της διεργασίας, μπορούν να αναιρεθούν συνολικά, με μέσα που παρέχει το λειτουργικό σύστημα. Η default τιμή είναι “no”.
- abstractProcess: Αυτό το πεδίο δηλώνει αν η διεργασία που ακολουθεί στο κείμενο είναι αφηρημένη ή εκτελέσιμη. Με τον όρο αφηρημένη, εννοούμε ότι η διεργασία απλά καλεί μεθόδους άλλων διεργασιών ή υπηρεσιών, χωρίς να περιέχει δικό της εκτελέσιμο κώδικα.

Πιο κάτω στο κείμενο, ακολουθούν μια σειρά από άλλες δηλώσεις, οι οποίες έχουν την εξής σημασία:

<partnerLinks> ... </partnerLinks>

Εδώ δηλώνονται τα web-services τα οποία θα χρησιμοποιήσει η διεργασία, για να παράξει το ζητούμενο αποτέλεσμα. Οι δηλώσεις ενθυλακώνονται στο partnerLinks tag και έχουν την πιο κάτω μορφή:

```
<partnerLink name="invoicing" partnerLinkType="lns:invoicingLT"
      myRole="invoiceRequester" partnerRole="invoiceService"/>
```

Για κάθε «σύνδεση» με μια άλλη υπηρεσία, πρέπει να δηλωθεί ένα όνομα (`invoicing`), πρέπει να δηλωθεί το είδος της «σύνδεσης» (`lns:invoicingLT`) όπως αυτό έχει ονομαστεί στο WSDL αρχείο όπου ορίζεται, και πρέπει τέλος, να δηλωθεί ποιο από τους δύο ρόλους (υπηρεσία ή καταναλωτής) αναλαμβάνει το κάθε εμπλεκόμενο μέρος (myRole-partnerRole).

<variables>... </variables>
Μέσα στα variables tags δηλώνονται όλες οι μεταβλητές που θα χρησιμοποιηθούν κατά την εκτέλεση της διεργασίας. Οι δηλώσεις έχουν την πιο κάτω μορφή:

```
<variable name="PO" messageType="lns:POMessage"/>
```

Η δήλωση θυμίζει εκτελέσιμη γλώσσα προγραμματισμού. Για κάθε μεταβλητή δηλώνεται το όνομά της και ο τύπος των δεδομένων που θα αποθηκεύσει. Οι τύποι δεδομένων δηλώνονται επίσης στα συνημμένα WSDL αρχεία.

<correlationSets> ... </correlationSets>
Η BPEL μπορεί να μοντελοποιήσει δύο είδη αλληλεπιδράσεων:

- Αλληλεπιδράσεις σύντομες που δεν διατηρούν την κατάστασή τους (stateless interactions)
- Ασύγχρονες αλληλεπιδράσεις με μεγάλη διάρκεια που διατηρούν την κατάστασή τους (long running, asynchronous interactions)

Τα correlationSets παρέχουν την υποδομή για την υλοποίηση αλληλεπιδράσεων που διατηρούν την κατάστασή τους. Συγκεκριμένα, αποθηκεύουν τα δεδομένα που χρειάζονται ώστε να διατηρηθεί η κατάσταση της κάθε στιγμιοτύπου και ακολούθως προσφέρουν τον μηχανισμό για δρομολόγηση των μηνυμάτων που φτάνουν ασύγχρονα στη διεργασία, προς τον σωστό αποδέκτη – δηλαδή το σωστό στιγμιότυπο της διεργασίας. Με τα corellationSets μπορούμε να δημιουργήσουμε συνόδους (sessions).

Για την δημιουργία ενός correlationSet, πρέπει πρώτα να δηλωθούν μια σειρά από properties τα οποία θα χρησιμοποιηθούν για τον διαχωρισμό των συνόδων (sessions) μεταξύ τους. Ένα property ορίζεται όπως πιο κάτω:

<bpws:propertyname="..." type="..."/>
Δηλώνεται ένα property με όνομα που ισχύει για ολόκληρο το κείμενο BPEL καθώς και ο τύπος του.

<bpws:propertyAliaspropertyName="..." messageType="..."
part="..." query="..."/>
Υπάρχει η δυνατότητα, ένα property να αντιστοιχηθεί με ένα πεδίο ενός WSDL μηνύματος, ώστε να του ανατεθεί τιμή όταν ανταλλαγεί ένα τέτοιο μήνυμα.

Ακολούθως, δημιουργείται ένα correlationSet με δηλώσεις της μορφής:

<correlationSetname="..." properties="..."/>

Σε αυτή τη δήλωση ομαδοποιούνται διάφορα properties, τα οποία εφόσον τους ανατεθεί τιμή, την διατηρούν καθόλη τη διάρκεια μιας συνόδου (session). Η σύνοδος αρχικοποιείται με μια δήλωση της μορφής:

<correlations> <correlation set="PurchaseOrder" initiate="yes"/>
</correlations>

η οποία ενθυλακώνεται σε κάποιο από τα activity tags που θα περιγραφούν πιο κάτω. Από αυτό το σημείο και μετά το κάθε εισερχόμενο μήνυμα, δρομολογείται στο στιγμιότυπο της διεργασίας που ανήκει στην κατάλληλη σύνοδο.

<faultHandlers> ... </faultHandlers>
Οι faultHandlers είναι τμήματα κώδικα τα οποία εκτελούνται όταν προκύψει κάποιο σφάλμα κατά την εκτέλεση. Μπορεί να υπάρχει οποιοσδήποτε αριθμός από faultHandlers που να αντιμετωπίζει το κάθε πιθανό σφάλμα ξεχωριστά. Συγκεκριμένα, τα πιθανά σφάλματα που μπορεί να προκύψουν δηλώνονται στα αρχεία WSDL και πιο συγκεκριμένα στα portTypes. Για κάθε fault γίνεται η εξής δήλωση:

```
<catch faultName="lns:cannotCompleteOrder"
       faultVariable="POFault">
      (activities)*
</catch>
```

Η μορφή θυμίζει πολύ τα try/catch της Java. Εδώ δηλώνεται το όνομα του σφάλματος που μπορεί να προκύψει, καθώς και η μεταβλητή όπου θα φυλαχτούν οι τιμές που θα επιστραφούν από την κληθείσα μέθοδο που εξέπεμψε το σφάλμα.

<compensationHandlers> ... </compensationHandlers>
Εντός ενός compensationHandler tag μπορεί να καθοριστεί η δραστηριότητα που θα εκτελεστεί σε περίπτωση που κριθεί επιθυμητό, να αναιρεθούν τα αποτελέσματα ενός μπλοκ κώδικα το οποίο όμως έχει εκτελεστεί πλήρως. Για κάθε μπλοκ κώδικα μπορεί να οριστεί μόνο ένας compensationHandler, ο οποίο μπορεί να κληθεί όταν σε κάποιο σημείο του προγράμματος προκύψει σφάλμα κατά την εκτέλεση.

<eventHandlers> ... </eventHandlers>
Οι eventHandlers καθορίζουν τμήματα κώδικα, τα οποία εκτελούνται εάν προκύψουν συγκεκριμένα γεγονότα. Ένα είδος eventHandler είναι και οι compensationHandlers, οι οποίοι ενεργοποιούνται κυρίως όταν προκύψουν σφάλματα. Εκτός από τους compensationHandlers υπάρχουν ακόμα δύο είδη γεγονότων που μπορεί να χειριστεί η BPEL.

- Με το tag onMessage μπορεί να δηλωθεί δραστηριότητα, η οποία θα εκτελεστεί σε περίπτωση που ληφθεί ασύγχρονα ένα μήνυμα, όσο τρέχει το μπλοκ εντολών στο οποίο έχει δηλωθεί ο eventHandler.
- Με το tag onAlarm μπορεί να δηλωθεί δραστηριότητα, η οποία θα εκτελεστεί εάν παρέλθει ένα συγκεκριμένο χρονικό διάστημα και εφόσον εκτελείται ακόμα το

μπλοκ εντολών στο οποίο έχει δηλωθεί ο eventHandler. Μπορούν να δηλωθούν περισσότεροι από ένας onAlarm handlers.

Στο κείμενο που περιγράφει την δομή πιο πάνω, περιέχεται η λέξη *activity.* Στη θέση αυτής της λέξης προστίθενται οι διάφορες δραστηριότητες τις οποίες πρέπει να φέρει σε πέρας η διεργασία. Τέτοιες δραστηριότητες είναι η λήψη και αποστολή μηνυμάτων, η λήψη αποφάσεων, η κλήση μεθόδων κ.τ.λ. Σε αυτή τη θέση μπορούν να μπουν οποιαδήποτε από τα παρακάτω tags:

• <receive> • <reply> • <invoke> • <assign> • <throw> • <terminate> • <wait> • <empty>	• <sequence> • <switch> • <while> • <pick> • <flow> • <scope> • <compensate>

Παρακάτω εξηγείται η σύνταξη αυτών των δραστηριοτήτων:

<receive>

Με αυτό τον τρόπο δηλώνουμε ότι αναμένεται η άφιξη ενός μηνύματος, απαραίτητου για την εκτέλεση του υπολοίπου της διεργασίας. Με το receive tag, συνήθως αρχικοποιείται μια διεργασία, αφού χρησιμοποιείται για την λήψη και αποθήκευση της αίτησης του χρήστη της διεργασίας που δηλώνουμε. Η δομή της δήλωσης είναι η εξής:

```
<receive partnerLink="ncname" portType="qname" operation="ncname"
      variable="ncname"? createInstance="yes|no"?
      standard-attributes>
      standard-elements
      <correlations>?
            <correlation set="ncname" initiate="yes|no"?>+
      </correlations>
</receive>
```

Για την λήψη ενός μηνύματος, δηλώνεται η «σύνδεση» (partnerLink) μέσω της οποίας αναμένεται να φθάσει, και η μέθοδος (operation) της παρούσας διεργασίας – δηλ του παραλήπτη και όχι του αποστολέα του μηνύματος - η οποία θα κληθεί ώστε να παραδοθεί το μήνυμα. Επίσης ορίζεται η μεταβλητή στην οποία θα φυλαχτεί το μήνυμα,

κάτι που αποκαλύπτει και τη μορφή του μηνύματος, αφού τα πεδία που συνθέτουν την μεταβλητή έχουν οριστεί πιο πάνω. Ακολούθως μπορούν να οριστούν κάποια link (αφορούν θέματα συγχρονισμού και θα εξηγηθούν πιο κάτω). Σε ένα τέτοιο γεγονός μπορούν να εκκινηθούν (ή να συμπληρωθούν) μια σειρά από correlations τα οποία, όπως αναφέρθηκε πιο πάνω, ορίζουν τα χαρακτηριστικά μιας συνόδου (session). Σημασία πρέπει επίσης να δοθεί στην δυνατότητα να δημιουργηθεί ξεχωριστό στιγμιότυπο (κάτι σαν Thread) της διεργασίας (process) , μέσω της ιδιότητας createInstance.

<reply>

Η εντολή <reply> αποτελεί συμπληρωματική λειτουργία του <receive> που έχει περιγραφεί πιο πάνω, και έχει αντίστοιχη σημασία. Με το reply tag αποστέλλουμε ένα μήνυμα σε κάποιο άλλο web-service, μέσω ενός partnerLink το οποίο έχει οριστεί πιο πάνω. Συνήθως αποτελεί και το τελευταίο τμήμα μιας διεργασίας, αφού αποστέλλει την απόκριση στον χρήστη της διεργασίας. Η δήλωση είναι η εξής:

```
<reply partnerLink="ncname" portType="qname" operation="ncname"
      variable="ncname"? faultName="qname"?
      standard-attributes>
      standard-elements
      <correlations>?
            <correlation set="ncname" initiate="yes|no"?>+
      </correlations>
</reply>
```

Οι ιδιότητες εδώ έχουν την ίδια σημασία με την εντολή <receive>, με τη διαφορά ότι η μέθοδος αναφέρεται σε κλήση που θα γίνει από τη διεργασία στην απομακρυσμένη υπηρεσία – δέκτη του μηνύματος. Νέο στοιχείο εδώ είναι και η δήλωση της ιδιότητας faultName, που αναφέρεται σε σφάλμα που μπορεί να εκπέμψει η απομακρυσμένη υπηρεσία και το οποίο αφού ληφθεί, θα κληθεί ο αντίστοιχος χειριστής (faultHandler). Για το συγκεκριμένο partnerLink, τα πιθανά σφάλματα που μπορεί να προκύψουν ορίζονται στη δήλωση του αντίστοιχου portType στο σχετικό WSDL κείμενο.

<invoke>

Η εντολή invoke μας επιτρέπει να καλέσουμε μια από τις μεθόδους που αποκαλύπτει μια απομακρυσμένη υπηρεσία, όπως αυτή έχει δηλωθεί στο WSDL κείμενο, περνώντας

παραμέτρους ως είσοδο, και λαμβάνοντας δεδομένα ως έξοδο. Συγκεκριμένα η δήλωση είναι:

```
<invoke partnerLink="ncname" portType="qname" operation="ncname"
      inputVariable="ncname"? outputVariable="ncname"?
      standard-attributes>
      standard-elements
      <correlations>?
            <correlation set="ncname" initiate="yes|no"?
            pattern="in|out|out-in"/>+
      </correlations>
      <catch faultName="qname" faultVariable="ncname"?>*
            activity
      </catch>
      <catchAll>?
            activity
      </catchAll>
      <compensationHandler>?
            activity
      </compensationHandler>
</invoke>
```

Εδώ βλέπουμε ότι καθορίζεται η υπηρεσία (partnerLink), η μέθοδος (operation), τα δεδομένα που θα σταλούν ως παράμετροι (inputVariable) και τα δεδομένα που αναμένονται ως απόκριση (outputVariable). Το portType δεν είναι απαραίτητο να δηλωθεί αφού έχει καθοριστεί αλλού. Και εδώ μπορεί να ξεκινήσει ή να σταματήσει μια σύνοδος με παρόμοιο τρόπο. Σε αυτή την εντολή, μπορεί να καθοριστεί και η συμπεριφορά της διεργασίας, σε περίπτωση που ληφθεί σφάλμα κατά την κλήση της απομακρυσμένης μεθόδου. Αυτό μπορεί να γίνει είτε ξεχωριστά για το κάθε πιθανό σφάλμα που μπορεί να εκπεμφθεί (εντολή <catch>), ή μπορούν να αντιμετωπιστούν όλα με συνολικό τρόπο (εντολή <catchAll>). Επίσης μπορεί να δηλωθεί κι ένας χειριστής (compensationHandler) ο οποίος θα αναλάβει να αναιρέσει τη δραστηριότητα της εντολής <invoke> εάν και εφόσον κληθεί.

<assign>

Η εντολή αυτή αναλαμβάνει να αναθέσει τιμές σε μεταβλητές κατά τη διάρκεια της εκτέλεσης της διεργασίας. Χρησιμοποιείται για την αντιγραφή δεδομένων μεταξύ των μεταβλητών. Η γραμματική της εντολής είναι:

```
<assign standard-attributes>
      standard-elements
      <copy>+
            <from variable=" " part=" "/>
            <to variable=" " part=" "/>
      </copy>
</assign>
```

Μπορούν να ενθυλακωθούν στην εντολή περισσότερες από μια αναθέσεις τιμών. Σε κάθε from και το καθορίζεται η μεταβλητή (variable) και το πεδίο (part) στο οποίο βρίσκονται ή θα αντιγραφούν τα δεδομένα.

<throw>

Με αυτή την εντολή εκπέμπεται ένα σφάλμα (fault) μέσα από τη διεργασία κι όχι από κάποια κλήση μεθόδου. Η γραμματική της εντολής είναι:

```
<throw faultName="qname" faultVariable="ncname"? standard-attributes>
      standard-elements
</throw>
```

<terminate>

Αυτή η εντολή απλά τερματίζει τη διεργασία στο συγκεκριμένο σημείο.

<wait>

Η εντολή wait σταματά την εκτέλεση του προγράμματος για ένα καθορισμένο χρονικό διάστημα, ή μέχρι να λήξει μια προθεσμία. Η γραμματική είναι:

```
<wait (for="duration-expr" | until="deadline-expr")
      standard-attributes>
      standard-elements
</wait>
```

<empty>

Η εντολή <empty> απλά δεν εκτελεί καμία λειτουργία. Χρησιμοποιείται για να δηλώσει κενά μπλοκ εντολών (π.χ. ένα catch στο οποίο δεν θέλουμε να γίνει καμία ενέργεια). Η γραμματική της είναι:

```
<empty standard-attributes>
      standard-elements
</empty>
```

<sequence>

Σε ένα sequence tag ενθυλακώνεται ένα σύνολο εντολών, οι οποίες πρόκειται να εκτελεστούν ακολουθιακά με την σειρά που εμφανίζονται. Μέσα σε ένα sequence tag ενθυλακώνεται και όλη η δραστηριότητα της διεργασίας. Η σύνταξη της εντολής είναι:

```
<sequence standard-attributes>
      standard-elements
      activity+
</sequence>
```

<switch>

Η εντολή switch συμπεριλαμβάνεται στα δυναμικά και εκτελέσιμα χαρακτηριστικά της BPEL και έχει την ίδια σημασία και λειτουργικότητα με αυτή που συναντάμε στις κλασσικές εκτελέσιμες γλώσσες προγραμματισμού. Συγκεκριμένα, η εντολή επιτρέπει την εκτέλεση ενός από μια σειρά μπλοκ εντολών, σύμφωνα με την τιμή κάποιων εκφράσεων. Η γραμματική είναι η εξής:

```
<switch standard-attributes>
      standard-elements
      <case condition="bool-expr">+
            activity
      </case>
      <otherwise>?
            activity
      </otherwise>
</switch>
```

Παρατηρούμε ότι δίνεται και η δυνατότητα της «γενικης περίπτωσης» μέσω του tag otherwise, το οποίο εκτελείται σε περίπτωση που δεν ικανοποιείται καμία από τις πιο πάνω συνθήκες (case).

<while>

Η εντολή while επίσης προσφέρει την ίδια λειτουργικότητα με αυτή που συναντάμε στις κλασσικές γλώσσες προγραμματισμού. Συγκεκριμένα εκτελεί επαναληπτικά ένα μπλοκ εντολών όσο είναι αληθής κάποια Boolean συνθήκη. Η σύνταξη της εντολής είναι:

```
<while condition="bool-expr" standard-attributes>
      standard-elements
      activity
</while>
```

<pick>

Η δομή pick προσθέτει στην BPEL περισσότερο έλεγχο όσο αφορά την ασύγχρονη μεταφορά μηνυμάτων μεταξύ των συνεργαζόμενων υπηρεσιών και της διεργασίας, η οποία μπορεί να αντιδράσει ανάλογα με το ποια γεγονότα θα συμβούν πρώτα, ή ακόμα να θέσει χρονικούς περιορισμούς (timeouts). Συγκεκριμένα, σε ένα pick tag ενθυλακώνονται μια σειρά από χειριστές γεγονότων που χειρίζονται είτε την άφιξη μηνύματος συγκεκριμένου τύπου, είτε το γεγονός της παρέλευσης συγκεκριμένου χρονικού διαστήματος ή συγκεκριμένης προθεσμίας. Ακολούθως εκτελείται το μπλοκ εντολών που αφορά το γεγονός που θα συμβεί πρώτο. Η σύνταξη της δομής είναι:

```
<pick createInstance="yes|no"? standard-attributes>
      standard-elements
      <onMessage partnerLink="ncname" portType="qname"
            operation="ncname" variable="ncname"?>+
            <correlations>?
                  <correlation set="ncname" initiate="yes|no"?>+
            </correlations>
            activity
      </onMessage>
      <onAlarm (for="duration-expr" | until="deadline-expr")>*
            activity
      </onAlarm>
</pick>
```

<flow>

Η δομή flow ενθυλακώνει μια ομάδα εντολών, οι οποίες πρόκειται να εκτελεστούν ταυτόχρονα. Με αυτό τον τρόπο μπορούν να μοντελοποιηθούν διαγράμματα ροής τα οποία διαθέτουν πολλούς παράλληλους κλάδους. Οι ροές εντολών που εκτελούνται ταυτόχρονα, μπορούν να συγχρονιστούν και να αλλάξουν δεδομένα, με την δήλωση και χρήση κάποιων link (η λειτουργία των οποίων θα εξηγηθεί παρακάτω). Η γραμματική της δομής είναι:

```
<flow standard-attributes>
      standard-elements
      <links>?
            <link name="ncname">+
      </links>
      activity+
</flow>
```

<scope>

Σε ένα scope tag ενθυλακώνεται μια ακολουθία εντολών, οι οποίες συνιστούν ένα μπλοκ εντολών για το οποίο μπορούν να δηλωθούν ξεχωριστά μεταβλητές, χειριστές σφαλμάτων (faultHandlers), και χειριστές αναίρεσης εντολών (compensationHandlers). Ουσιαστικά, σε ένα scope μπορεί να βρίσκεται δηλωμένη μια πλήρησς διεργασία, με τη διαφορά ότι δεν μπορούν να οριστούν partnerLinks αποκλειστικά γι αυτό το μπλοκ εντολών. Συγκεκριμένα η μορφή ενός scope είναι η εξής:

```
<scope variableAccessSerializable="yes|no" standard-attributes>
      standard-elements
      <variables>?
            ...
      </variables>
      <correlationSets>?
            ...
      </correlationSets>
      <faultHandlers>?
            ...
      </faultHandlers>
      <compensationHandler>?
            ...
      </compensationHandler>
      <eventHandlers>?
            ...
      </eventHandlers>
      activity
</scope>
```

Γενικότερα όλη η διεργασία χωρίζεται νοητά σε scopes, στα οποία αντιστοιχούν (ισχύουν) κάποιες μεταβλητές, για τα οποία δηλώνονται χειριστές σφαλμάτων (faultHandler), και των οποίων οι ενέργειες μπορούν να αναιρεθούν από τον αντίστοιχο χειριστή αναίρεσης (compensasionHandler)

<compensate>

Η εντολή compensate καλεί τον χειριστή αναίρεσης κάποιου μπλοκ εντολών (scope), όπως αυτό έχει δηλωθεί. Η σύνταξη της εντολής είναι η εξής:

```
<compensate scope="ncname"? standard-attributes>
      standard-elements
</compensate>
```

Σημειώνεται ότι αυτή η εντολή μπορεί να χρησιμοποιηθεί μόνο μέσα σε κάποιο faultHandler ή μέσα σε κάποιο άλλο compensationHandler.

Στην παρουσίαση της γραμματικής των πιο πάνω εντολών, βλέπουμε να υπάρχει η δυνατότητα για προσθήκη standard-attributes, και standard-elements. Ως standars-attributes εννοούμε τα:

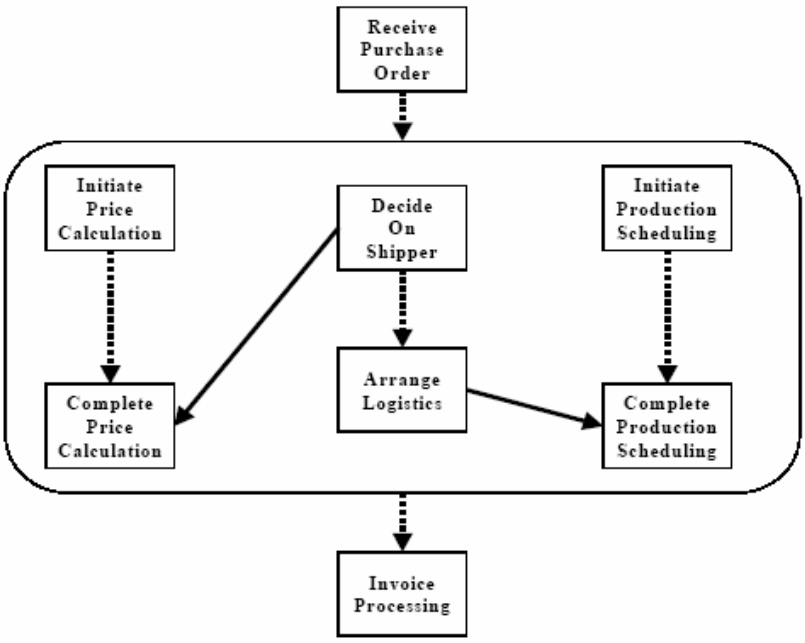
```
name="ncname"?
joinCondition="bool-expr"?
suppressJoinFailure="yes|no"?
```

Οι πιο πάνω ιδιότητες είναι προαιρετικές και έχουν την εξής σημασία:

- name: Δίνει τη δυνατότητα να δοθεί ένα μοναδικό όνομα στην κάθε εντολή, μπλοκ ή δομή του προγράμματος.
- joinCondition=”bool-expr”: Καθορίζει την λογική πράξη που θα εκτελεστεί σχετικά με την εκπλήρωση των περιορισμών που προκύπτουν από τα διάφορα links που καταλήγουν σε αυτή την εντολή. Η default τιμή είναι OR.
- suppressJoinFailure="yes|no": Καθορίζει αν η αποτυχία εκπλήρωσης των περιορισμών σύμφωνα με την joinCondition θα εκπέμψει κάποιο σφάλμα, ή αν δεν θα ληφθεί υπόψη.

Τα standard-elements που αναφέρονται πιο πάνω είναι τα εξής:

```
<target linkName="ncname"/>*
<source linkName="ncname" transitionCondition="bool-expr"?/>*
```



Σχ. 2.4.7

Οι δηλώσεις source και target αφορούν την άφιξη και τον προορισμό των συνδέσεων (links) που ενώνουν και συγχρονίζουν παράλληλες ροές εκτέλεσης. Στο πιο πάνω σχήμα φαίνεται ένα παράδειγμα όπου οι παράλληλες ροές είναι αλληλοεξαρτώμενες.

Με τα διακεκομμένα βέλη δηλώνεται η ακολουθία των εργασιών που θα εκτελεστούν. Τα βέλη με τις συνεχόμενες μαύρες γραμμές δηλώνουν ότι για να προχωρήσει η εκτέλεση πέρα από το σημείο που καταλήγει το βέλος, θα πρέπει να έχει ολοκληρωθεί η εκτέλεση των όσων προηγούνται του σημείου εκκίνησης του βέλους. Αυτά τα βέλη σε BPEL, δηλώνονται και ονομάζονται με δηλώσεις

```
<link name="ncname"></link>
```

και τοποθετούνται στο διάγραμμα με δηλώσεις

```
<target linkName="ncname"/>
<source linkName="ncname"/>
```

2.4.8 BPELJ

Η BPEL, όπως έχει περιγραφεί πιο πάνω, στοχεύει στον συντονισμό – ή καλύτερα τον προγραμματισμό- αυτόνομων λειτουργικών μονάδων και διεργασιών. Δεν φιλοδοξεί να αποτελέσει ακόμα μια γλώσσας γενικού σκοπού. Αντίθετα, έχει τεθεί ως προϋπόθεση ότι η BPEL θα συνεργάζεται με άλλες εκτελέσιμες γλώσσες προγραμματισμού, οι οποίες και θα υλοποιούν τις αυτόνομες λειτουργικές μονάδες που θα κληθεί να συντονίσει.

Μια από τις γλώσσες υλοποίησης εφαρμογών δικτύου, με έμφυτο το γνώρισμα της διαλειτουργικότητας, είναι η Java. HBPEL όπως έχει περιγραφεί, απαιτεί από τις υπηρεσίες που καλείται να συντονίσει, να αποκαλυφθούν με την μορφή web-services, δηλαδή να αποκαλύψουν ένα interface δηλωμένο σε WSDL, με την επικοινωνία να επιτυγχάνεται μέσω SOAP μηνυμάτων. Επίσης η BPEL διαθέτει εργαλεία για τον σχεδιασμό εκτελέσιμων διεργασιων, μέσω μιας XML αναπαράστασης, η οποία όμως κληρονομεί τους περιορισμούς που επιβάλει η XML ως ο κοινός παρονομαστής μεταξύ των διαφόρων λειτουργικών συστημάτων και γλωσσών προγραμματισμού.

Έχοντας αυτά υπόψη, οι εταιρείες IBM και BEA που συμμετέχουν στο σχεδιασμό του προτύπου της BPEL, έχουν προτείνει ένα συνδυασμό της BPEL με τη γλώσσα Java, την BPELJ. Σκοπός της συνένωσης, είναι να κατανεμηθούν καθήκοντα στην κάθε γλώσσα, σύμφωνα με τις αρετές τις κάθε μιας. Συγκεκριμένα στον πιο κάτω πίνακα φαίνονται οι λειτουργίες στις οποίες υπερτερεί η κάθε γλώσσα χωριστά.

BPEL	***Java***
• Συντονισμός αυτοτελών λειτουργικών μονάδων και συνδυασμός της λειτουργικότητάς τους για την παραγωγή σύνθετων διεργασιών • Διατήρηση πολλών ταυτόχρονων ροών εκτέλεσης που απαιτούν μεγάλα χρονικά διαστήματα για να συμπληρωθούν • Επιλεκτική αναίρεση των ενεργειών σε περίπτωση αποτυχίας συμπλήρωσης μιας ροής εκτέλεσης • Καθορισμός δέσμης ενεργειών με έναρξη εκτέλεσης σε συγκεκριμένη χρονική στιγμή και με συγκεκριμένη σειρά • Αποστολή μηνυμάτων σε web-services • Παραλαβή ενός μηνύματος συγκεκριμένου τύπου, λαμβανομένου από ένα σύνολο πιθανών αναμενόμενων τύπων μηνυμάτων. • Δρομολόγηση μηνυμάτων εισερχόμενων στην κατάλληλη διεργασία αποδέκτη.	• Υπολογισμός των τιμών που θα τοποθετηθούν σε μεταβλητές ή σε μηνύματα που θα αποσταλούν • Δημιουργία μηνυμάτων που θα αποσταλούν σε κάποιο web-service χρησιμοποιώντας δεδομένα από κάποιο άλλο μήνυμα ή κάποια άλλη μεταβλητή • Αποδόμηση ενός εισερχόμενου μηνύματος, εύρεση σημαντικών δεδομένων, μετατροπή και εισαγωγή τους σε άλλα μηνύματα. • Υπολογισμός τιμών που θα χρησιμοποιηθούν για τον έλεγχο της ροής της διεργασίας (έλεγχος συνθήκης, βρόγχοι επανάληψης) • Εκτέλεση εργασιών χωρίς την ανάγκη δημιουργίας ξεχωριστών wev-services

Η BPELJ επιτρέπει στην Java και την BPEL, να συνεργαστούν, επιτρέποντας σε τμήματα κώδικα Java (*Java snippets*), να ενσωματωθούν στον κώδικα της BPEL. Τα snippets είναι μικρά block κώδικα Java, τα οποία μπορούν να χρησιμοποιηθούν για εργασιές όπως είναι η αρχικοποίηση μεταβλητών, ο έλεγχος συνθηκών, υλοποίηση βρόγχων επανάληψης, δημιουργία μηνυμάτων προς web-services κ.τ.λ.

Η BPELJ δεν περιλαμβάνει επεκτάσεις οι οποίες είναι απαραίτητες σε όλες τις BPELJ διεργασίες. Συνεπώς, η κάθε BPEL διεργασία αποτελεί ταυτόχρονα μια έγκυρη BPELJ διεργασία.

Εκτός από τη δυνατότητα εκτέλεσης υπολογισμών μέσω Java κώδικα, δίδεται και η δυνατότητα μακροχρόνιας αλληλεπίδρασης BPEL διεργασιών με J2EE εφαρμογές. Προστίθεται η δυνατότητα δημιουργίας partner links όπου οι διεπαφές θα είναι ορισμένες στη γλώσσα Java και όχι στην σε WSDL δίνοντας τη δυνατότητα απευθείας αλληλεπίδρασης με απομακρυσμένα αντικείμενα Java.

Θεμελιώδης διαφορά μεταξύ μιας διεπαφής WSDL και μιας διεπαφής Java, είναι ότι οι Java μέθοδοι μπορούν να δεχθούν σαν παράμετρο και να επιστρέψουν αυθαίρετα αντικείμενα σε αντίθεση με τα XML μηνύματα. Για να μπορεί να γίνει εκμετάλλευση της δύναμης της γλώσσας Java, μια διεργασία θα πρέπει να είναι σε θέση να ετοιμάσει αντικείμενα Java, για να τα στείλει υπό τη μορφή παραμέτρων, καθώς και να παραλάβει αντικείμενα Java ως τιμές επιστροφής. Σε κάθε περίπτωση ο κώδικας Java, έχει πρόσβαση σε όλες τις BPEL μεταβλητές που βρίσκονται στη σφαίρα επιρροής του (scope).

Πιο κάτω φαίνεται ο τρόπος δήλωσης μεταβλητών Java, ή δήλωση Java Partner Links καθώς και η δήλωση ενός snippet που μπορεί να τοποθετηθεί οπουδήποτε στο BPEL κείμενο:

Java Partner Link

```
<partnerLink name="rateLookup"
        partnerLinkType="bpelj:com.mycom.ejb.RateLookup"/>
```

Variable

```
<variable name="jmsMessage" type="bpelj:javax.jms.TextMessage"/>
```

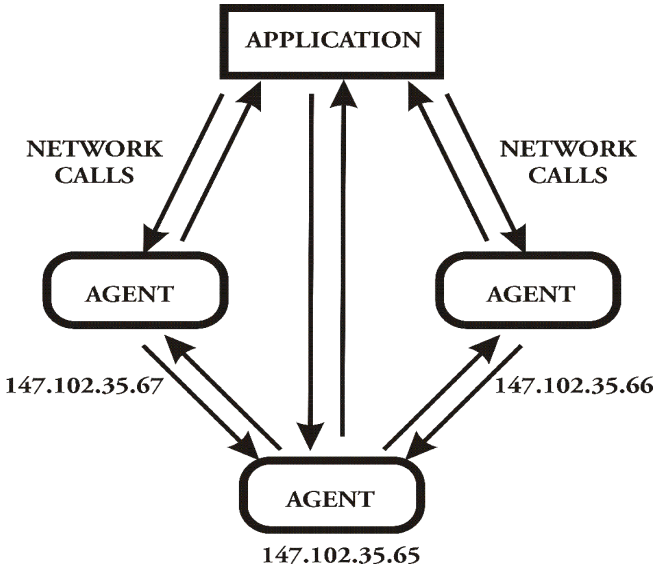
Java Snippet

```
<bpelj:snippet name="Calculate Total">
        <bpelj:code>
                response.setTaxRate(taxRate);
                response.setDiscount(discount.getRate());
                float subtotal = response.getSubtotal();
                subtotal = subtotal * (1 – discount.getRate());
                response.setSubtotal(subtotal);
                float taxes = subtotal * taxRate;
                float total = subtotal + taxes;
                response.setTax(taxes);
                response.setTotal(total);
                // Prepare the text message to be sent in the next activity.
                jmsMessage =
                p_inquiryTopic.getSession().createTextMessage(response);
        </bpelj:code>
</bpelj:snippet>
```

ΚΕΦΑΛΑΙΟ 3
ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΕΦΑΡΜΟΓΗΣ

3.1 Εισαγωγή

Η συνεχής αύξηση στην απαίτηση υπολογιστικής ισχύος στις σύγχρονες εφαρμογές, καθώς και ο γιγαντισμός που χαρακτηρίζει τα σημερινά πληροφοριακά συστήματα και δίκτυα έχουν οδηγήσει σε αποκεντρωμένες προσεγγίσεις ανάλυσης και σχεδιασμού συστημάτων, όπου ο φόρτος εργασίας, ή η ευθύνη επόπτευσης, κατανέμεται σε ένα σύνολο γεωγραφικά διασκορπισμένων υπολογιστών και συσκευών. Οι υπολογιστές αυτοί επικοινωνούν μεταξύ τους μέσω δικτύου.



Σχήμα 3.1.1

Σε ένα τέτοιο κατανεμημένο σύστημα, υπάρχει πάντα η εφαρμογή πελάτης, και υπάρχουν και οι κατανεμημένες οντότητες (πράκτορες) οι οποίες αναλαμβάνουν την παρακολούθηση συσκευών, τη λήψη μετρήσεων, την αποθήκευση και ανάκληση δεδομένων, ή την επεξεργασία δεδομένων. Η επικοινωνία μεταξύ της εφαρμογής και των πρακτόρων γίνεται μέσω δικτύου με εγκατάσταση σχετικών συνδέσεων.

Βασικές προϋποθέσεις για την ομαλή και αποδοτική λειτουργία ενός κατανεμημένου συστήματος, είναι η δυνατότητα εγκατάστασης συνδέσεων μεταξύ τμημάτων με γρήγορο και εύκολο τρόπο, η ύπαρξη απλής διαδικασίας αναβάθμισης των λειτουργικών τμημάτων, η δυνατότητα ανακάλυψης της ζητούμενης λειτουργικότητας σε πραγματικό χρόνο καθώς και η γρήγορη και εύκολη προσαρμογή του συστήματος σε αλλαγές στη δομή του, λόγω αστοχίας του δικτύου ή κάποιου τμήματός του.

Η ανάγκη για πλήρωση των πιο πάνω προϋποθέσεων, γίνεται ακόμα πιο επιτακτική όσο μεγαλώνει το μέγεθος και η πολυπλοκότητα ενός κατανεμημένου συστήματος, καθώς εντάσσονται σε αυτό λειτουργικές μονάδες υλοποιημένες σε εντελώς διαφορετικές πλατφόρμες. Ως κοινός παρονομαστής των διαφόρων τεχνολογιών ανάπτυξης λογισμικού που υπάρχουν, έχει αναπτυχθεί η XML παράσταση δεδομένων και εντολών, με βάση την οποία έχουν αναπτυχθεί οι τεχνολογίες που συνθέτουν τα ευρέως διαδεδομένα web-services. Αφού έχει επιτευχθεί η διασύνδεση των λειτουργικών τμημάτων με αποδοτικό τρόπο, εμφανίστηκε η ανάγκη για πιο αποδοτική ενορχήστρωση της λειτουργίας των επιμέρους τμημάτων. Αυτό το κενό ήρθαν να καλύψουν οι Service Oriented αρχιτεκτονικές, που υιοθετούν ένα κεντρικό σημείο διασύνδεσης το οποίο αναλαμβάνει εξολοκλήρου τον συντονισμό (service bus).

Η παρούσα εργασία, εκμεταλλεύεται την προσέγγιση των SOA αρχιτεκτονικών, για την ανάπτυξη κατανεμημένων συστημάτων βασισμένων στη γλώσσα Java. Συγκεκριμένα, τα ζητήματα αυτόματης διασύνδεσης, αυτόματης ανακάλυψης, προσαρμογής στις μεταβολές αντιμετωπίζονται χρησιμοποιώντας την τεχνολογία Jini η οποία καλύπτει με ένα πλήρη τρόπο αυτό το κενό. Το ζήτημα της αναβάθμισης λειτουργικών τμημάτων και αυτό της διακίνησης κώδικα, αντιμετωπίζεται με τη χρήση της τεχνολογίας JMX που μας επιτρέπει την εύκολη έναρξη και παύση τμημάτων κώδικα.

3.2 Χαρακτηριστικά του συστήματος

Το σύστημα που έχει αναπτυχθεί και δοκιμαστεί, αφορά τη δυνατότητα ανάπτυξης SOA αρχιτεκτονικών με τη χρήση των τεχνολογιών Jini και JMX. Αν και τα χαρακτηριστικά που αναφέρονται πιο κάτω αποτελούν ουσιαστικά προϋποθέσεις για την ύπαρξη μιας SOA αρχιτεκτονικής, εντούτοις γίνεται ξεχωριστή μνεία, γιατί η πλήρωση αυτών των ιδιοτήτων θα εξεταστεί συνολικά από την εφαρμογή πελάτη που έχει αναπτυχθεί.

Επεκτασιμότητα: Το πλαίσιο λειτουργίας όπως έχει αναπτυχθεί, επιτρέπει την επέκταση του συστήματος με νέους πράκτορες (agents) οι οποίοι γίνονται άμεσα αντιληπτοί. Επίσης υπάρχει η δυνατότητα προσθήκης ή και αντικατάστασης υπηρεσιών εντός του κάθε πράκτορα κάτι που επιτρέπει την εύκολη αναβάθμιση του συστήματος. Συνολικά το σύστημα είναι δυναμικό και επεκτακτό χωρίς ουσιαστικό περιορισμό ως προς το πόσες οντότητες μπορούν να συμμετέχουν σε αυτό.

Αντοχή και προσαρμογή στις μεταβολές: Η δυνατότητα ανακάλυψης κάθε νέου πράκτορα άμεσα, αλλά και η δυνατότητα εκπομπής ειδοποιήσεων όταν αυτός πάψει να είναι διαθέσιμος, προσδίδουν στο σύστημα το χαρακτηριστικό της προσαρμοστικότητας στις αλλαγές και της αντοχής σε αυτές. Τέτοια γεγονότα θεωρούνται πλέον δεδομένο ότι μπορούν να συμβούν σε ένα δυναμικό και μη αξιόπιστο δίκτυο και αντιμετωπίζονται σαν φυσικές διεργασίες στο περιβάλλον που λειτουργεί το σύστημα. Επίσης υπάρχει η δυνατότητα εκπομπής ειδοποιήσεων κατά την εγγραφή και διαγραφή υπηρεσιών σε ένα πράκτορα, ώστε η εφαρμογή πελάτης να γνωρίζει ποιες υπηρεσίες είναι διαθέσιμες κάθε στιγμή.

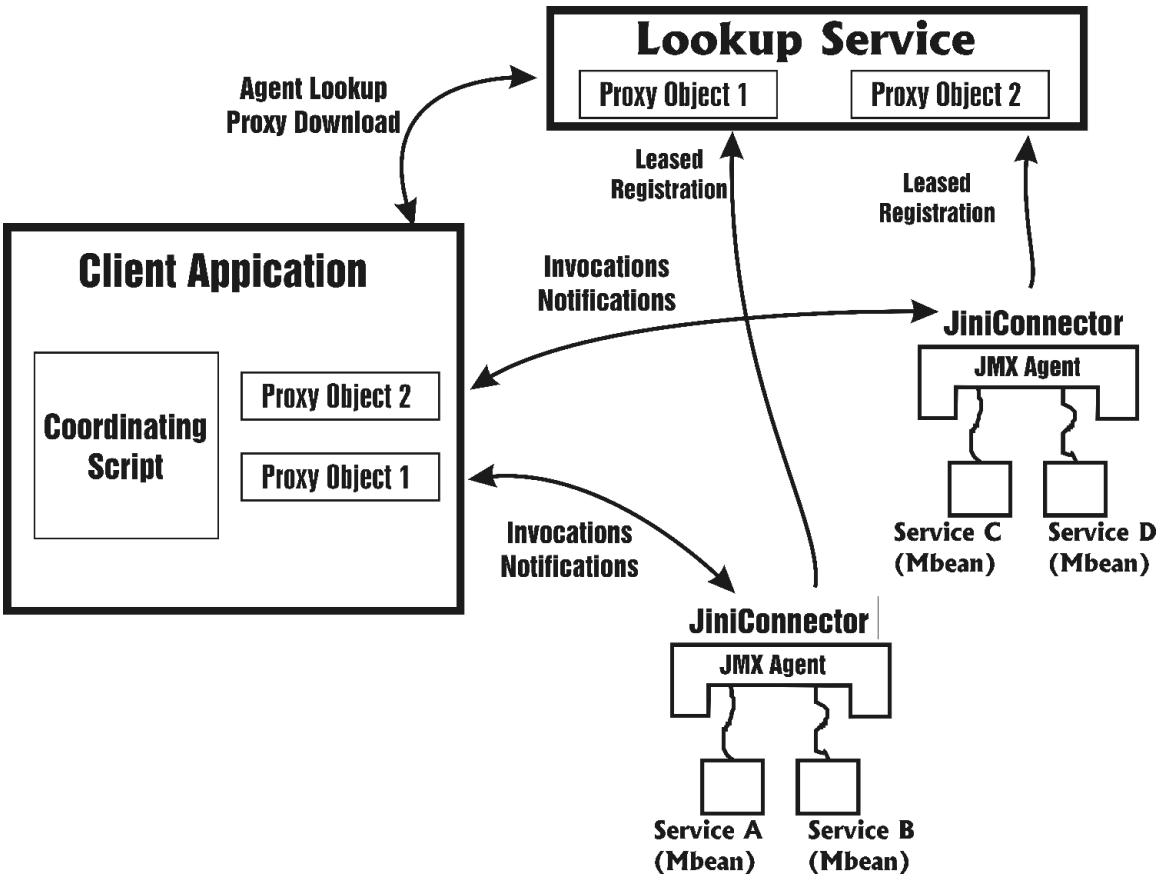
Διαφάνεια στην επικοινωνία: Η μέθοδος χρήσης των πρακτόρων μέσω αντικειμένου αντιπροσώπου, προσθέτει ακόμα ένα επίπεδο αφαίρεσης στην πλευρά της εφαρμογής πελάτη. Η εφαρμογή πελάτης χρησιμοποιεί τον κάθε πράκτορα μέσω του αντικειμένου αντιπροσώπου (proxy), σαν τοπικό αντικείμενο, αγνοώντας την μέθοδο με την οποία επικοινωνούν μεταξύ τους τα απομακρυσμένα αντικείμενα. Αυτό διευρύνει

της επιλογές ως προς την υλοποίηση υπηρεσιών, χωρίς να εμπλέκεται ο πελάτης στην διαδικασία.

Ανεξαρτησία από πλατφόρμα λειτουργίας: Το όλο σύστημα έχει αναπτυχθεί χρησιμοποιώντας τη γλώσσα Java, η οποία είναι ανεξάρτητη πλατφόρμας λειτουργίας. Το σύστημα μπορεί να λειτουργήσει σε οποιοδήποτε μηχάνημα υπάρχει εγκατεστημένη Java Virtual Machine.

3.3 Αρχιτεκτονική του συστήματος

Η αρχιτεκτονική του συστήματος παρουσιάζεται πιο κάτω στο Σχήμα 3.3.1 όπου διακρίνονται και τα βασικά συστατικά του. Πιο κάτω επεξηγείται ο ρόλος και η σημασία τους στη λειτουργία του όλου συστήματος.



Σχήμα 3.3.1 *– Αρχιτεκτονική του συστήματος*

Σύμφωνα με το πιο πάνω σχήμα, τα βασικά συστατικά της εφαρμογής είναι τα εξής:

- Υπηρεσία Αναζήτησης (Lookup Service)
- Πράκτορες JMX (JMX Agents)
- Jini Connectors – Proxy Objects
- MBeans
- Εφαρμογή Πελάτης (Client Application)

Το σύστημα αποτελείται από τους JMX συμβατούς πράκτορες, εντός των οποίων λειτουργεί ένα σύνολο από MBeans. Τα MBeans αποτελούν τις υπηρεσίες που μπορεί να χρησιμοποιήσει ο απομακρυσμένος πελάτης, και μπορούν να εγγραφούν και να διαγραφούν από τον πελάτη. Η ανακάλυψη, ο χειρισμός, η ενορχήστρωση της λειτουργίας και η διαχείριση αυτών των υπηρεσιών είναι οι στόχοι της όλης δομής του συστήματος. Αυτές οι υπηρεσίες μπορούν να εκτελούν οποιαδήποτε λειτουργία. Στο σενάριο που εξετάζει η παρούσα εργασία, αυτά τα MBeans εκτελούν ερωτήματα SNMP.

Οι πράκτορες, αφού αρχικοποιηθούν, ενεργοποιούν τον JiniConnector. Ο JiniConnector δίνει τη δυνατότητα σε απομακρυσμένου πελάτες να ανακαλύψουν και να επικοινωνήσουν με τους πράκτορες. Με την εκκίνησή του ο JiniConnector, εντοπίζει όλες τις διαθέσιμες υπηρεσίες αναζήτησης μέσω multicast μηνυμάτων. Ακολούθως εγγράφεται σε αυτές, καταχωρώντας εκεί ένα αντικείμενο αντιπρόσωπο (proxy object) το οποίο θα χρησιμοποιήσει ακολούθως η κάθε εφαρμογή πελάτης.

Όταν μια εφαρμογή πελάτης ξεκινήσει, εντοπίζει με τη σειρά της όλες τις διαθέσιμες υπηρεσίες αναζήτησης μέσω multicast μηνυμάτων. Εκεί αναζητεί όλους τους εγγεγραμμένους σε αυτές πράκτορες JMX, και ακολούθως αντιγράφει από εκεί τα αντικείμενα αντιπροσώπους των πρακτόρων που έχουν τοποθετήσει εκεί οι αντίστοιχοι JiniConnectors.

Η εφαρμογή πελάτης, αφού αποκτήσει τα αντικείμενα αντιπροσώπους, μπορεί να χρησιμοποιήσει τον κάθε πράκτορα σαν τοπικό αντικείμενο. Η εφαρμογή πελάτης ακολούθως ζητά από τον κάθε πράκτορα κατάλογο με όλα τα εγγεγραμμένα σε αυτόν

MBeans μαζί με την περιγραφή τους. Με αυτό τον τρόπο μπορεί η εφαρμογή πελάτης να κατασκευάσει ένα τοπικό μητρώο όλων των διαθέσιμων υπηρεσιών στο σύστημα.

Ακολούθως η εφαρμογή πελάτης, μπορεί να καλέσει τις λειτουργίες της κάθε υπηρεσίας με συγκεκριμένη σειρά, σύμφωνα πάντα με ένα σενάριο το οποίο περιλαμβάνεται στην εφαρμογή, ώστε να εκτελεστούν διάφορες σύνθετες εργασίες.

Στα πλαίσια αυτής της εργασίας, έχουν υλοποιηθεί, ο JiniConnector, και ένα δείγμα εφαρμογής πελάτη με την οποία επιδεικνύεται η λειτουργία του συστήματος και εκτελούνται σύνθετες εργασίες. Επίσης έχει υλοποιηθεί μια υπηρεσία (MBean) που επιτρέπει την διακίνηση και φόρτωση κώδικα από την εφαρμογή πελάτη στους πράκτορες με απομακρυσμένες κλήσεις, χωρίς την ανάγκη τοπικής διαχείρισης των πρακτόρων.

3.4 JINI Connector για JMX Πράκτορες

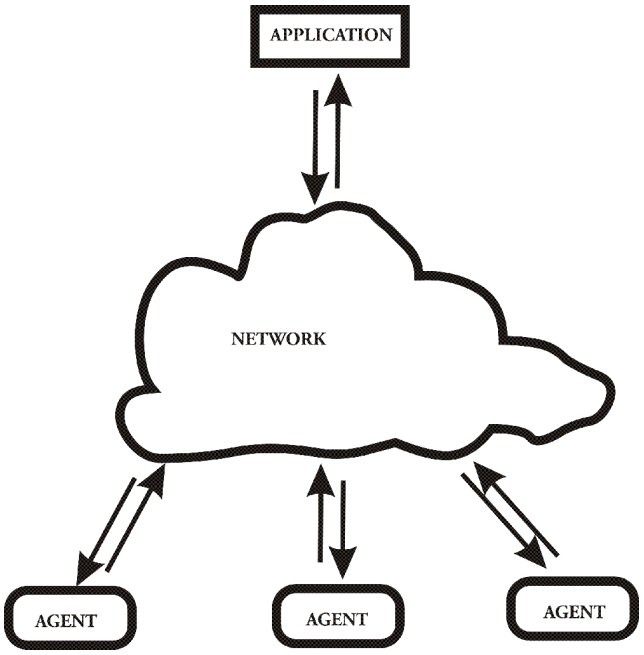
3.4.1 Υφιστάμενη Κατάσταση

Στο σενάριο που εξετάζει η παρούσα εργασία, χρησιμοποιούνται πράκτορες συμβατοί με το πρότυπο JMX (Java Management eXtension) για μια εφαρμογή διαχείρισης δικτύων. Η εφαρμογή αρχικοποιεί και εγγράφει στους JMX πράκτορες κατάλληλα MBeans τα οποία εκτελούν ερωτήσεις SNMP (Simple Network Management Network) και παρακολουθούν συγκεκριμένα μεγέθη σε υπολογιστές του δικτύου (π.χ. throughput σε κάποιο interface ενός router). Η επικοινωνία μεταξύ της εφαρμογής και των πρακτόρων, καθώς και των πρακτόρων μεταξύ τους, γίνεται με την ανταλλαγή μηνυμάτων SOAP (Simple Object Access Protocol) παρότι όλα τα μέρη του συστήματος είναι υλοποιημένα στη γλώσσα Java. Ο περιορισμός που εισάγεται με τις συνδέσεις SOAP, είναι ότι η εφαρμογή πελάτης, πρέπει να γνωρίζει εκ των προτέρων την τοποθεσία του κάθε πράκτορα στο δίκτυο (IP) και να εγκαταστήσει ειδική σύνδεση μαζί του για να επιτευχθεί η επικοινωνία. Επίσης δεν υπάρχει τρόπος εύκολου εντοπισμού γεγονότων όπως είναι η

αστοχία του δικτύου – κάτι που μπορεί να καταστήσει ένα πράκτορα μη προσβάσιμο για τον πελάτη.

Ένα ακόμα πρόβλημα είναι ότι για να επεκταθεί το σύστημα με την προσθήκη περισσότερων πρακτόρων, πρέπει να εγκατασταθούν χειρωνακτικά οι συνδέσεις αφού η εφαρμογή πελάτης, χρειάζεται το IP του πράκτορα για να επικοινωνήσει μαζί του. Επίσης σε περίπτωση όπου χρειάζεται να διασφαλιστεί επικοινωνία μεταξύ πρακτόρων, οι σχετικές συνδέσεις πρέπει επίσης να εγκατασταθούν χειρωνακτικά και πάλι θα είναι στατικές – δηλ, σε περίπτωση απώλειας ενός πράκτορα, κινδυνεύει να αδρανοποιηθεί ένα μεγάλο μέρος του συστήματος.

Χειρωνακτικά πρέπει να γίνει και η σχετική ενημέρωση όλων των μερών του συστήματος σε περίπτωση που ένας πράκτορας χρειαστεί να αλλάξει τοποθεσία στο δίκτυο, δηλ να αλλάξει IP.



Σχήμα 3.4.1

3.4.2 Περιγραφή του JINIConnector

Το ζητούμενο είναι ο σχεδιασμός και η υλοποίηση ενός ευέλικτου συστήματος επικοινωνίας μεταξύ των πρακτόρων και της εφαρμογής πελάτη καθώς και μεταξύ πρακτόρων με χρήση της τεχνολογίας Jini. Θέλουμε τις συνδέσεις μεταξύ των κατανεμημένων τμημάτων της εφαρμογής, να εγκαθίστανται με αυτόματο τρόπο, και να υπάρχει διαφάνεια από την πλευρά του πελάτη, ως προς την τοποθεσία του κάθε πράκτορα στο δίκτυο. Θέλουμε δηλαδή η εικόνα του σχήματος 3.1.1, να μεταβληθεί στην εικόνα του σχήματος 3.4.1.

Όταν μια εφαρμογή πελάτης ενώνεται πρώτη φορά με το δίκτυο, θα πρέπει με αυτόματο τρόπο να μπορεί να εντοπίσει και να επικοινωνήσει με όλους τους διαθέσιμους πράκτορες που μπορεί να χρησιμοποιήσει. Σε άλλη περίπτωση όταν προστίθεται ένας πράκτορας στο σύστημα, πρέπει να μπορεί να ενημερώσει τα υπόλοιπα μέρη του συστήματος για την ύπαρξη και τη διαθεσιμότητά του. Τέλος, εάν μια οντότητα χρειάζεται τις υπηρεσίες ενός πράκτορα με συγκεκριμένα χαρακτηριστικά, θα πρέπει να μπορεί να τον αναζητήσει στο δίκτυο, να τον εντοπίσει και να επικοινωνήσει μαζί του. Το σύνολο αυτών των λειτουργιών, μπορεί να παρομοιαστεί με τη λειτουργία Plug ‘n’ Play που συναντάμε στη σύνδεση περιφερειακών σε υπολογιστές.

Ο εκάστοτε πελάτης μιας υπηρεσίας (είτε μια εφαρμογή, είτε κάποιος πράκτορας) δεν θα πρέπει να επηρεάζεται, ούτε από την θέση της υπηρεσίας στο δίκτυο, μα ούτε και από την μορφή του δικτύου που παρεμβάλλεται (Internet, LAN, Ethernet συνδέσεις κ.τ.λ.). Η επικοινωνία δηλαδή πρέπει να είναι διαφανής.

Η τεχνολογία Jini αντιμετωπίζει αποτελεσματικά τους περιορισμούς που περιγράφονται πιο πάνω, και πληρεί τις προδιαγραφές που τίθενται. Συγκεκριμένα, όπως περιγράφεται και στην επισκόπηση της τεχνολογίας, εγκαθίσταται μια υπηρεσία αναζήτησης, η οποία επιτρέπει την αναζήτηση και την ανακάλυψη υπηρεσιών όπως περιγράφεται πιο πάνω. Επίσης, η τεχνολογία Jini εξασφαλίζει ότι η επικοινωνία μιας οντότητας με ένα απομακρυσμένο πράκτορα, γίνεται μέσω ενός τοπικού αντικειμένου αντιπροσώπου

(proxy). Με αυτό τον τρόπο ο πελάτης αγνοεί εντελώς τόσο την τοποθεσία του πράκτορα στο δίκτυο, όσο και την μέθοδο με την οποία επικοινωνούν ο αντιπρόσωπος με τον πράκτορα. Δίδεται λοιπόν η δυνατότητα στον πελάτη να χρησιμοποιήσει την κάθε υπηρεσία σαν τοπική και όχι σαν απομακρυσμένη.

Για την λύση του προβλήματος της συνδεσιμότητας, έχει υλοποιηθεί ένας JINI Connector για πράκτορες JMX. Αν και υπάρχει το δεδομένο των πρακτόρων JMX που εκτελούν λειτουργίες διαχείρισης δικτύου, ο connector που έχει υλοποιηθεί είναι γενικού σκοπού, και μπορεί να χρησιμοποιηθεί για οποιοδήποτε πράκτορα συμβατό με τις προδιαγραφές JMX. Έχει δοθεί σημασία, στη δυνατότητα καθορισμού των χαρακτηριστικών του connector υπό μορφή παραμέτρων, ώστε αυτός να μην περιορίζεται στη λειτουργία του κατανεμημένου Network Manager. Ταυτόχρονα, υπάρχει πρόνοια ώστε να μπορούν να χρησιμοποιηθούν όλες οι δυνατότητες της τεχνολογίας Jini όσον αφορά τον σχηματισμό κοινοτήτων πρακτόρων με ειδικά και συγκεκριμένα χαρακτηριστικά.

Ο JINIConnector είναι ουσιαστικά ένα MBean το οποίο μπορεί να εγγραφεί σε ένα οποιοδήποτε MBean Server. Αφού εγγραφεί, με κατάλληλη εντολή από τον πράκτορα, αρχίζει η λειτουργία της ανακάλυψης υπηρεσιών αναζήτησης (lookup services) τα οποία όμως ανήκουν σε συγκεκριμένο group, ανήκουν δηλαδή σε συγκεκριμένη κοινότητα. Ακολούθως ο JINIConnector, μισθώνει την εγγραφή ενός αντικειμένου JMXConnector σε κάθε μια από τις υπηρεσίες αναζήτησης που εντοπίζει, καταχωρώντας παράλληλα και μια σειρά από διευκρινιστικά attributes. Αυτά τα attributes δίνουν την δυνατότητα σε εφαρμογές πελάτες, να ξεχωρίσουν του JMX πράκτορες με βάση κάποια ειδικά χαρακτηριστικά της λειτουργίας που προσφέρουν. Το αντικείμενο JMXConnector, πρόκειται να λειτουργήσει ως αντιπρόσωπος του πράκτορα (proxy) στις απομακρυσμένες JVM. Με αυτό τον τρόπο, ο πράκτορας μπορεί να εντοπιστεί από άλλους πράκτορες, και από εφαρμογές πελάτες π.χ. απομακρυσμένους διαχειριστές. Ο JINIConnector αναλαμβάνει να ανανεώνει τακτικά την εγγραφή του στις υπηρεσίες αναζήτησης (leasing).

3.5 Υπηρεσία eXteneded Management appLet (XMLet)

3.5.1 Υφιστάμενη Κατάσταση

Όπως περιγράφηκε στην παρουσίαση της τεχνολογίας JMX, μια από τις υπηρεσίες που προσφέρει ένας πράκτορας, είναι αυτή της δυναμικής φόρτωση κλάσεων από απομακρυσμένες τοποθεσίες. Τη δυνατότητα αυτή δίνει η υπηρεσία MLet (Management Applet), η οποία διατηρεί μια βιβλιοθήκη από URL's, από τα οποία έχει τη δυνατότητα να αντιγράψει, τα απαραίτητα .class και .jar αρχεία, ώστε να φορτώσει τις κλάσεις που περιέχονται σε αυτά.

Η αντιγραφή των αρχείων από την υπηρεσία MLet όμως, γίνεται μέσω του πρωτοκόλλου HTTP, κάτι που συνεπάγεται ότι τα αρχεία αυτά πρέπει να εξυπηρετούνται από ένα web-server. Όταν μια εφαρμογή διαχείρισης συνδεδεμένη με ένα πράκτορα JMX χρειάζεται να φορτώσει ένα MBean στον πράκτορα, πρέπει να κλάση αυτού, και όλες οι κλάσεις από τις οποίες εξαρτάται, να είναι ή να γίνουν γνωστές σε κάποιο Classloader της JVM (Java Virtual Machine) που είναι φορτωμένος ο πράκτορας. Αν δεν υπάρχουν τοπικά αντίγραφα των δηλώσεων των κλάσεων, τότε η εφαρμογή διαχείρισης, θα πρέπει είτε να εκκινήσει ένα δικό της web-server που θα εξυπηρετήσει τα αρχεία προς αποστολή, είτε να γνωρίζει εκ των προτέρων μια τοποθεσία (URL) στο δίκτυο όπου αυτά βρίσκονται.

Σε μεγάλα κατανεμημένα συστήματα όπου οι πράκτορες χρησιμοποιούνται για φόρτωση εφαρμογών (application servers), η δημιουργία/εγκατάσταση κάποιου codebase (βιβλιοθήκη κλάσεων που εξυπηρετείται από web server) μπορεί να αποτελεί αποδοτική λύση στο θέμα της διακίνησης κώδικα. Σε μεγάλα συστήματα αυτό μπορεί να αποτελεί και εγγύηση για την ασφάλεια και τη εγκυρότητα του κώδικα που διατίθεται. Για μικρές εφαρμογές όμως, και ειδικότερα για εφαρμογές διαχείρισης, η ανάγκη εγκατάστασης http server, απλά για την φόρτωση ενός MBean που διαχειρίζεται π.χ. ένα switch, μπορεί να αποτελεί πρόβλημα και να προσθέτει ένα ακόμα επίπεδο πολυπλοκότητας στην εφαρμογή. Θα ήταν πολύ πιο πρακτικό να υπήρχε η δυνατότητα να αποσταλεί ένα jar

αρχείο που να περιέχει όλες τις απαιτούμενες κλάσεις, και να μπορεί ο πράκτορας με τη σειρά του να τις φορτώσει από το .jar αρχείο.

3.5.2 Περιγραφή της υπηρεσίας XMLet

Όπως περιγράφεται και στο κεφάλαιο που αφορά το JMX Remote API, η επικοινωνία μιας JMX συμβατής εφαρμογής με ένα απομακρυσμένο πράκτορα, γίνεται μέσω ενός αντικειμένου αντιπροσώπου. Το αντικείμενο αντιπρόσωπος, επικοινωνεί με κάποιο connector εγκατεστημένο στον πράκτορα χρησιμοποιώντας μια μέθοδο απομακρυσμένης κλήσης μεθόδων (RPC-Remote Procedure Call). Συγκεκριμένα το JMX Remote API, προβλέπει connectors που χρησιμοποιούν RMI (JRMP ή IIOB) και δίδεται η δυνατότητα για δημιουργία και άλλων μεθόδων ανταλλαγής μηνυμάτων.

Για την καταλληλότητα μιας μεθόδου RPC, πρέπει να πληρούνται δύο προϋποθέσεις. Πρώτον είναι ο ορισμός ενός πρωτοκόλλου ανταλλαγής μηνυμάτων, που θα αναλάβει την μεταφορά κλήσεων – αιτήσεων και αποκρίσεων απομακρυσμένων μεθόδων. Κατά την μεταφορά κλήσεων και αποκρίσεων μεθόδων, μεταφέρονται δεδομένα και αρχικοποιημένα αντικείμενα, είτε ως παράμετροι, είτε ως τιμές επιστροφής. Δεύτερη λοιπόν προϋπόθεση είναι υλοποίηση μεθόδου που να επιτρέπει την μεταφορά των αντικειμένων αυτών, μέσω του δικτύου, με τρόπο ώστε να μπορούν να ανασυσταθούν τα αντικείμενα αυτά στα δύο άκρα της σύνδεσης.

Δεδομένου λοιπόν του ότι ο μηχανισμός μεταφοράς αντικειμένων υπάρχει, μια πρώτη σκέψη είναι να μεταφερθεί το jar αρχείο, ενθυλακωμένο σε ένα μεταφερόμενο αντικείμενο. Ακολούθως πρέπει να υλοποιηθεί ο μηχανισμός για εξαγωγή του jar αρχείου από το αντικείμενο, και την ανασύσταση των κλάσεων που περιέχονται σε αυτό.

Η φόρτωση των κλάσεων σε μια JVM μπορεί να γίνει μόνο από κάποιο Classloader. Συνεπώς, πρέπει να φροντίσουμε ότι το jar αρχείο θα γίνει προσβάσιμο σε ένα classloader ο οποίος και θα λάβει την εντολή να φορτώσει τις σχετικές κλάσεις και να εγγράψει τα απεσταλέντα MBeans.

Όπως έχει αναφερθεί, η υπηρεσία MLet, αποτελεί επέκταση της κλάσης URLClassLoader, η οποία έχει τη δυνατότητα να φορτώσει κλάσεις, αφού πρώτα αντιγράψει της δηλώσεις τους (bytecodes) μέσω HTTP από απομακρυσμένες τοποθεσίες. Δίδεται επίσης η δυνατότητα από τον πράκτορα, να δηλωθεί η υπηρεσία MLet, σαν ο classloader που θα φορτώσει την κλάση ενός MBean, προτού αυτό εγγραφεί στον MBean Server. Για παράδειγμα, μπορούμε να φορτώσουμε ένα MBean με μια εντολή της μορφής:

createMBean(String className, ObjectName name, ObjectName loaderName)

Σαν loaderName δηλώνεται ένα στιγμιότυπο της υπηρεσίας MLet, το οποίο έχει πρόσβαση στα bytecodes που δηλώνουν την κλάση className. Δεδομένου ότι έχουμε έτοιμο ένα classloader με διαμόρφωση MBean, μπορούμε να τον χρησιμοποιήσουμε σαν βάση για την υλοποίηση μιας υπηρεσίας, που θα μας επιτρέπει την απευθείας αποστολή του jar αρχείου, και τον χειρισμό του όταν αυτό φτάσει στον προορισμό του.

Για την αντιμετώπιση του προβλήματος δημιουργήθηκε μια επέκταση (υποκλάση) της υπηρεσίας MLet στην οποία προστέθηκε η δυνατότητα να λαμβάνει υπό μορφή ενός πίνακα από bytes (byte[]) ένα αρχείο jar, από το οποίο να μπορεί να φορτώνει κλάσεις. Στο MBean που έχει προκύψει, έχουν προστεθεί οι κατάλληλες μέθοδοι, που επιτρέπουν την προσθήκη jar αρχείων στις βιβλιοθήκες που διαχειρίζεται σαν classloader. Έχουν υλοποιηθεί επίσης αντικείμενα που μας επιτρέπουν την μεταφορά των jar αρχείων υπό τη μορφή απλών bytes μέσα από τις RMI κλήσεις των μεθόδων του XMLet MBean.

3.6 SNMPProxy και SNMPUtil MBeans

3.6.1 Σκοπός

Το πλαίσιο λειτουργίας που έχει υλοποιηθεί, είχε ως αρχικό στόχο την εφαρμογή σε σεναρίων που αφορούν τη διαχείριση δικτύων. Χρησιμοποιώντας τον JiniConnector, μια

εφαρμογή πελάτης μπορεί να επικοινωνήσει με ένα πράκτορα JMX, να εντοπίσει όλα τα MBeans που αυτός περιέχει, και να τα χρησιμοποιήσει με βάση κάποιο προκαθορισμένο σενάριο. Πιο συγκεκριμένα, τα MBeans αποτελούν τις υπηρεσίες του SOA συστήματος που έχει αναπτυχθεί, και ο πελάτης μπορεί να μετατρέψει ένα διάγραμμα ροής, σε μια ακολουθία από κλήσεις μεθόδων, διοχετεύοντας παράλληλα τα αποτελέσματα της μιας υπηρεσίας ως είσοδο σε μια επόμενη.

Για να μπορέσει να εκτελέσει η εφαρμογή πελάτης το σενάριο, χρειάζεται πρώτα να εντοπίσει στο σύστημα, όλες τις απαιτούμενες υπηρεσίες που θα αναλάβουν την επιμέρους επεξεργασία δεδομένων ή την επιμέρους εκτέλεση εργασιών. Σε περίπτωση που αυτές δεν βρεθούν, ή ακόμα εάν δεν βρεθεί ο απαιτούμενος αριθμός στιγμιότυπων μιας υπηρεσίας, η εφαρμογή πελάτης μπορεί να φορτώσει τις απαιτούμενες υπηρεσίες στους πράκτορες, πριν εκκινήσει η εκτέλεση του σεναρίου. Αυτή τη δυνατότητα μας τη δίνει η τεχνολογία JMX σε συνδυασμό και με την υπηρεσία XMLet που έχει υλοποιηθεί και περιγράφεται πιο πάνω.

Ένα μέρος της διαχείρισης συσκευών δικτύου αφορά την λήψη διαφόρων μετρήσεων και τον υπολογισμό στατιστικών που αφορούν τις συσκευές και τους διαύλους μεταφοράς δεδομένων. Η πιο διαδεδομένη μέθοδος λήψης των απαραίτητων ποσοτήτων είναι τα SNMP queries (Simple Network Management Protocol). Ακολούθως, χρησιμοποιώντας συγκεκριμένες μαθηματικές σχέσεις, καταλήγουμε στις απαραίτητες στατιστικές που μας παρουσιάζουν την κατάσταση του δικτύου. Για την κάλυψη αυτού του κενού, έχουν υλοποιηθεί τα SNMPProxy και SNMPUtil MBeans, ώστε να μπορεί να εκτελεστεί από την εφαρμογή πελάτη ένα σενάριο διαχείρισης δικτύου.

3.6.2 SNMPProxy

Το SNMPProxy MBean, προσφέρει μεθόδους που αφορούν την εκτέλεση SNMP ερωτήσεων σε συσκευές δικτύου. Συγκεκριμένα παρέχει τις απαραίτητες μεθόδους, ώστε να μπορούμε να αντλήσουμε οποιοδήποτε μετρητή από την MIB (Management Information Base) μιας SNMP aware συσκευής στο δίκτυο. Επίσης προσφέρει μεθόδους

που αφορούν την σάρωση ενός τμήματος του δέντρου της MIB, ώστε να μπορούν να αντληθούν πολλές τιμές μαζί σε περίπτωση που αυτό είναι επιθυμητό.

Όσο αφορά τη σάρωση τμήματος του δέντρου, έχουν υλοποιηθεί μέθοδοι που αφορούν την σάρωση συγκεκριμένων τμημάτων της MIB όπως είναι το RouteTable ή ακόμα και του πίνακα αντιστοιχίας διευθύνσεων του πρωτοκόλλου ARP (Address Resolution Protocol).

3.6.3 SNMPUtil

Το SNMPUtil MBean, προσφέρει βοηθητικές μεθόδους, που έχουν να κάνουν κυρίως με την εκτέλεση αριθμητικών πράξεων και τον υπολογισμό στατιστικών. Τα δεδομένα που επεξεργάζονται οι μέθοδοι του SNMPUtil MBean, συνήθως προέρχονται από ερωτήματα SNMP του SNMPProxy MBean, τα οποία η εφαρμογή πελάτης διοχετεύει κατάλληλα.

Οι διαθέσιμες μέθοδοι, αφορούν κυρίως υπολογισμούς της μορφής

$$(X(t1)- X(t0))/(t1-t0)$$

όπου X είναι οι τιμές συγκεκριμένου μετρητή που λαμβάνονται σε δύο διαφορετικές χρονικές στιγμές.

Επίσης έχει υλοποιηθεί μέθοδος που προκαλεί την κατάλληλη καθυστέρηση ώστε να ληφθούν οι διακριτές τιμές και να υπολογιστεί η στατιστική.

ΚΕΦΑΛΑΙΟ 4
ΥΛΟΠΟΙΗΣΗ ΕΦΑΡΜΟΓΗΣ

4.1 Εισαγωγή

Όπως περιγράφηκε στο κεφάλαιο που αφορούσε την αρχιτεκτονική του συστήματος, οι πράκτορες της εφαρμογής είναι πράκτορες συμβατοί με το πρότυπο JMX (JMX Agents) οι οποίοι περιέχουν MBeans που εκτελούν συγκεκριμένες λειτουργίες. Οι πράκτορες αυτοί έχουν σχεδιαστεί ώστε με το που εκκινούν, να αρχικοποιείται, να εγγράφεται και να εκκινεί και ο JINIConnector που έχει υλοποιηθεί. Επίσης ο κάθε πράκτορας, διαθέτει τοπικά αντίγραφα (μέσω του ορισμού του classpath) όλων των κλάσεων που απαιτούνται για την λειτουργία της υπηρεσίας XMLet (που επίσης έχει υλοποιηθεί στα πλαίσια αυτής της εργασίας), έτσι ώστε να μπορεί η εφαρμογή πελάτης να εκκινήσει στιγμιότυπά της, και να μπορεί να αποστείλει και να φορτώσει κώδικα, πρότερα άγνωστο στη JVM του πράκτορα.

Για να λειτουργήσει το όλο σύστημα εγκαθίσταται και λειτουργεί και μια υπηρεσία αναζήτησης (lookup service). Ως υπηρεσία αναζήτησης, έχει χρησιμοποιηθεί η αντίστοιχη υλοποίηση της Sun (ονομάζεται Reggie), η οποία διανέμεται με τις βιβλιοθήκες του JINI 2.0. Μπορεί να χρησιμοποιηθεί οποιαδήποτε υλοποίηση υπηρεσίας αναζήτησης, αρκεί να ακολουθεί το πρότυπο της τεχνολογίας Jini.

Επίσης, τόσο για την λειτουργία της υπηρεσίας αναζήτησης, όσο και για την λειτουργία των πρακτόρων του συστήματος, έχει εγκατασταθεί ένας μικρός web-server (ClassServer που περιλαμβάνεται στο Jini 2.0), στον οποίο βρίσκονται διαθέσιμα όλα τα αρχεία που χρειάζονται για να λειτουργήσουν οι κλήσεις RMI μεταξύ των λειτουργικών τμημάτων του συστήματος. Περαιτέρω εξηγήσεις θα δοθούν στη συνέχεια.

Τέλος, η εφαρμογή πελάτης που εξετάζει και παρουσιάζει τις δυνατότητες του όλου συστήματος θα παρουσιαστεί με την μορφή screenshots στο επόμενο κεφάλαιο.

4.2 Υλοποίηση του JINIConnector

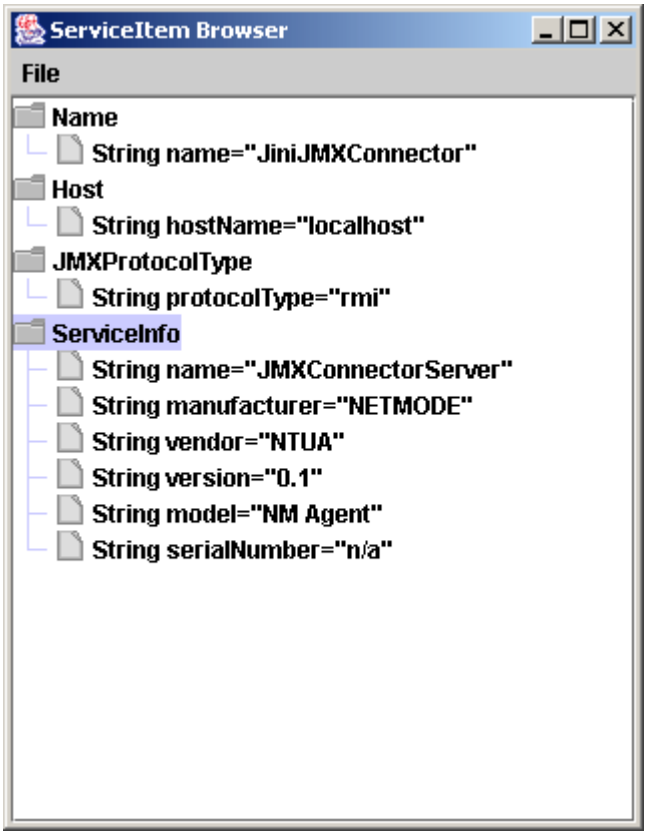
Όπως αναφέρθηκε, ο JINIConnector, έχει την διαμόρφωση ενός MBean το οποίο μπορεί να εγγραφεί σε οποιοδήποτε MBean Server, δίνοντας του τη δυνατότητα να ενταχθεί σε μια κοινότητα JINI. Ο πιο σύνθετος constructor του JINIConnector παίρνει τα πιο κάτω ορίσματα:

public JINIConnector(String set_groups[], String entryName, Entry[] entries)

Σαν πρώτο όρισμα, ο constructor, παίρνει ένα πίνακα από strings, ο οποίος περιέχει τα ονόματα των groups στα οποία επιθυμεί να ενταχθεί ο JMX πράκτορας. Αυτή η πληροφορία θα χρησιμοποιηθεί, αφού εντοπιστούν οι υπηρεσίες αναζήτησης, για να επιλέξει ο connector σε ποιες από αυτές θα εγγραφεί και ποιες θα αγνοήσει ώστε να ενταχθεί στις κοινότητες Jini που απαιτείται. Σε περίπτωση που δηλωθεί null σε αυτή την παράμετρο, τότε ο connector εγγράφει τον πράκτορα στο group “public”.

Σαν δεύτερο όρισμα, ο constructor, παίρνει ένα string, το οποίο καθορίζει το όνομα του πράκτορα, όπως αυτό θα καταχωρηθεί σαν ένα Name attribute σε κάθε υπηρεσία αναζήτησης. Αυτό το όνομα δεν έχει συγκεκριμένο ρόλο να παίξει, απλά χρησιμοποιείται σαν διευκρινιστικό ή σαν αναγνωριστικό για τον πράκτορα. Σε περίπτωση που δεν καθοριστεί όνομα (δηλαδή, αν δηλωθεί null) τότε ο πράκτορας παίρνει το όνομα “JMXAgent”.

Σαν τρίτο όρισμα, ο constructor, παίρνει ένα πίνακα ο οποίος περιέχει αντικείμενα τύπου Entry, τα οποία και αποτελούν τα attributes που επιθυμεί να εγγράψει ο πράκτορας σε κάθε υπηρεσία αναζήτησης. Αυτό το πεδίο είναι σημαντικό, αφού ο πράκτορας μπορεί να εγγράψει μία σειρά από χαρακτηριστικά τα οποία θα τον καθορίζουν σε ενδεχόμενη αναζήτηση πελατών, και θα δίνουν ικανοποιητική πληροφορία, ως προς το τι μπορεί να προσφέρει ο πράκτορας.



Σχήμα 3.2.2 *– Όψη των attributes που εγγράφει ο JINIConnector στο lookup service. Τα attributes Host και JMXProtocolType, τοποθετούνται υποχρεωτικά. Η τιμή του attribute Name καθορίζεται στον constructor, ενώ το το attribute ServiceInfo το έχει τοποθετήσει ο χρήστης.*

Ειδικά για την καταχώρηση σε υπηρεσίες αναζήτησης πρακτόρων JMX, η κοινότητα Jini, υλοποίησε τρία νέα χαρακτηριστικά (attributes), τα οποία πρέπει να συνοδεύουν πάντοτε μια εγγραφή αντικειμένου JMXConnector. Αυτά τα χαρακτηριστικά μαζί με την σημασία τους είναι:

- **Host**: υποχρεωτικό χαρακτηριστικό το οποίο περιέχει την διεύθυνση του πράκτορα στο δίκτυο είτε σε μορφή http://agent.ntua.gr είτε σαν διεύθυνση IP. Σε κάθε εγγραφή μπορεί να υπάρχει μόνο ένα χαρακτηριστικό τύπου Host.
- **JMXProtocolType**: υποχρεωτικό χαρακτηριστικό το οποίο δηλώνει το πρωτόκολλο επικοινωνίας που χρησιμοποιεί ο JMXConnector. Τα πρωτόκολλα που ανήκουν μέχρι τώρα στο πρότυπο είναι τα «rmi» για JRMP συνδέσεις, «iiop» για IIOP συνδέσει και «jmxmp» σε περίπτωση που χρησιμοποιείται ο προαιρετικός JMXMP generic connector. Σε κάθε εγγραφή μπορεί να υπάρχει μόνο ένα χαρακτηριστικό τύπου JMXProtocolType.

- **JMXProperty**: προαιρετικό χαρακτηριστικό το οποίο δηλώνει κάποια ιδιότητα του πράκτορα και την αντίστοιχη τιμή της. Το όνομα της ιδιότητας μπορεί να έχει τη μορφή παρόμοια με τα system properties της Java (jmx.connection.timeout). Η τιμή που καταχωρείται στην υπηρεσία αναζήτησης, είναι απαραίτητα τύπου string, κάτι που όμως δεν περιορίζει τη δυνατότητα ορισμού αριθμητικών μεγεθών αφού η μετατροπή μεταξύ αλφαριθμητικών και αριθμητικών τύπων είναι πλέον κάτι το τετριμμένο. Σε κάθε εγγραφή μπορούν να υπάρχουν απεριόριστος αριθμός από JMXProperty χαρακτηριστικά.

Ο JINIConnector φροντίζει να εγγραφεί ένα χαρακτηριστικό τύπου Host με την κατάλληλη τιμή την οποία εξασφαλίζει από το σύστημα. Επίσης φροντίζει για την εγγραφή και ενός χαρακτηριστικού τύπου JMXProtocolType, το οποίο στην περίπτωσή μας, έχει πάντα την τιμή «rmi», αφού χρησιμοποιείται ο RMIConnector που προσφέρει το πρότυπο JMX. Ο χρήστης του JINIConnector δεν χρειάζεται να καθορίσει αυτά τα δύο χαρακτηριστικά. Μπορεί όμως να καθορίσει ως παράμετρο στον constructor, όσα JMXProperty χαρακτηριστικά χρειάζεται, καθώς και οποιοδήποτε άλλο χαρακτηριστικό υπάρχει ή έχει υλοποιήσει ο ίδιος.

Αφού αρχικοποιηθεί και εγγραφεί στον MBean Server ένα JINIConnector MBean, ο πράκτορας ενεργοποιεί τον connector καλώντας την μέθοδο

public boolean enableConnections()

Η μέθοδος αυτή, αναλαμβάνει να δημιουργήσει ένα στιγμιότυπο JMXConnector, συνδεδεμένο με τον MBeanServer. Ακολούθως, αρχικοποιείται ένα στιγμιότυπο της βοηθητικής κλάσης JoinManager, η οποία αναλαμβάνει να χειριστεί με αποδοτικό τρόπο, όλα τα θέματα που αφορούν την ανακάλυψη κατάλληλων υπηρεσιών αναζήτησης και την εγγραφή σε αυτές. Ο JoinManager αναλαμβάνει επιπλέον την τακτική ανανέωση των μισθώσεων σε αυτές τις υπηρεσίες, κρατώντας τον πράκτορα συνδεδεμένο με την κοινότητα Jini, και διαθέσιμο σε απομακρυσμένες εφαρμογές.

Όπως κάθε MBean, έτσι και ο JINIConnector διαθέτει ένα management interface, το οποίο επιτρέπει σε τοπικούς και απομακρυσμένους διαχειριστές του πράκτορα, να

καλέσουν μεθόδους του. Το interface αυτό είναι περιέχει μόνο μεθόδους που αφορούν την εκκίνηση και την παύση του connector, καθώς και την αλλαγή του ονόματος του πράκτορα, και των jinni groups που αυτός εγγράφεται. Οι μέθοδοι είναι οι εξής:

```
public String getEntryName();
public void setEntryName(String name);
public String[] getGroups();
public void setGroups(String[] groups);
public boolean enableConnections();
public boolean disableConnections();
```

JINIConnectorMBean interface

4.2.1 Codebase

Όπως σε κάθε περίπτωση που χρησιμοποιείται RMI για την συνεργασία απομακρυσμένων αντικειμένων, έτσι και εδώ, απαιτείται κάποιες κλάσεις να είναι διαθέσιμες και ένα συγκεκριμένο URL, ώστε να μπορούν να αντιγραφούν από απομακρυσμένους χρήστες.

Η επικοινωνία μεταξύ του JINIConnector και της υπηρεσίας αναζήτησης γίνεται μέσω RMI συνεπώς, όλες οι κλάσεις από τις οποίες εξαρτάται το αντικείμενο αντιπρόσωπος (service object – στην περίπτωσή μας ο JMXConnector) καθώς και όλες οι κλάσεις των attributes που θα εγγραφούν στο lookup service, πρέπει να είναι διαθέσιμες σε κάποιο URL, όπου θα εξυπηρετούνται από κάποιο http server. Αυτό δίνει την δυνατότητα στο lookup service καθώς και αργότερα στον πελάτη, να αντιγράψει (να αποσειριοποιήσει) όλα τα αντικείμενα που χρειάζεται για να επιτευχθεί η επικοινωνία μεταξύ των μερών.

Η δήλωση του URL γίνεται με την ανάθεση τιμής στο system property java.rmi.codebase.

4.2.2 RMISecurityManager

Για να μπορεί να είναι εφικτή η μεταφορά κώδικα μεταξύ μηχανημάτων, αλλά και για να μπορεί να εκτελεστεί ο κώδικας που έχει αντιγραφεί από απομακρυσμένες τοποθεσίες,

χρειάζεται να αρχικοποιηθεί ένας SecurityManager, ο οποίος θα αναλάβει να εξετάσει το κατά πόσο είναι δηλωμένες ως ασφαλείς οι τοποθεσίες από τις οποίες προέρχεται ο «ξένος» κώδικας. Ακολούθως εκχωρεί δικαιώματα για την εκτέλεσή του και την παραχώρηση σε αυτόν πόρων του συστήματος.

Για να αντιμετωπιστεί το θέμα της ασφάλειας, χρειάζεται να αρχικοποιηθεί ένα στιγμιότυπο της κλάσης RMISecurityManager, το οποίο εξετάζει κάθε λειτουργία, κατά πόσο έχει το δικαίωμα να εκτελεστεί, σύμφωνα με τους περιορισμούς που δηλώνονται σε ένα text αρχείο. Η τοποθεσία του αρχείου αυτού τοποθετείται στο system property java.security.policy.

Μερικές σημαντικές δηλώσεις στο αρχείο για να μπορεί να λειτουργήσει ο JINIConnector είναι οι πιο κάτω:

- Επιτρέπουν την χρήση του πρωτοκόλλου multicast για την ανακάλυψη υπηρεσιών αναζήτησης στο δίκτυο. Τα IP "224.0.1.85" και "224.0.1.84", έχουν δεσμευθεί από την IANA ως οι αντίστοιχες request και announcement διευθύνσεις του πρωτοκόλλου multicast, για εφαρμογές Jini.

  permission java.net.SocketPermission "224.0.1.85", "connect,accept";
  permission java.net.SocketPermission "224.0.1.84", "connect,accept";

- Επιτρέπει την αντιγραφή κώδικα από τις αντίστοιχες τοποθεσίες, ώστε να μπορεί να χρησιμοποιηθεί το πρωτόκολλο unicast για τον εντοπισμό υπηρεσιών αναζήτησης καθώς επίσης και την λειτουργία του RMI.

  permission java.net.SocketPermission "*:1024-", "connect,accept";

- Επιτρέπει τη δήλωση κάποιου http codebase για τις ανάγκες μεταφοράς κώδικα, όπως έχει περιγραφεί πιο πάνω.

  permission java.net.SocketPermission "*:80", "connect";

4.3 Υλοποίηση της υπηρεσίας XMLet

Η υπηρεσία XMLet έχει τη διαμόρφωση ενός MBean ικανού να εγγραφεί σε οποιοδήποτε MBeanServer, και στον οποίο δίνει τη δυνατότητα δυναμικής φόρτωση κλάσεων με απευθείας αποστολή του jar αρχείου που περιέχει τα σχετικά αρχεία.

Η υπηρεσία XMLet αποτελεί υποκλάση του MLet MBean, και διατηρεί ακέραια όλη τη λειτουργικότητά του. Όλες οι μέθοδοι που περιλαμβάνει η υπηρεσία MLet, καθώς και η δυνατότητα φόρτωσης κλάσεων μέσω URL και μέσω περιγραφής από text αρχεία παραμένουν ακέραια και μπορούν να χρησιμοποιηθούν χωρίς αλλαγές. Επίσης έχουν διατηρηθεί άθικτοι και οι constructors του MBean.

Στην επέκταση της υπηρεσίας MLet που έχει υλοποιηθεί, έχουν προστεθεί οι μέθοδοι

public void **loadMBeanUsingJars**(String objectName,String className,
Vector classEntries, Object[] params, String[] signature)

public void **loadMBeanUsingJars**(String objectName,String className,
Vector classEntries)

public void **loadMBeanUsingXMLet**(String objectName,String className)

public void **loadMBeanUsingXMLet**(String objectName,String className,
Object[] params,String[] signature)

public void **addJar**(XMLetJarCarrier jc)

public void **addJar**(Vector jcv)

Οι μέθοδοι **loadMBeanUsingJars** και **addJar**, βρίσκονται σε αντιστοιχία με τις μεθόδους **getMBeansFromURL** και **addURL** του MLet. Η μέθοδος **loadMBeanUsingXMLet** είναι βοηθητική και μπορεί να χρησιμοποιηθεί για την

φόρτωση MBeans που βρίσκονται σε κάποιο από τα jar αρχεία που έχουν ήδη αποσταλεί στον XMLet ClassLoader, είτε μέσω της **loadMBeanUsingJars** είτε μέσω της **addJar**.

Συγκεκριμένα, όπως έχει αναλυθεί και στην παρουσίαση της υπηρεσίας MLet, ένας classloader, αφού φορτώσει μια κλάση από κάποια τοποθεσία, είναι πλέον σε θέση να φορτώσει οποιαδήποτε κλάση βρίσκεται στην ίδια τοποθεσία. Με παρόμοιο τρόπο, αφού μεταφερθεί ένα jar αρχείο στην υπηρεσία XMlet, μπορεί ακολούθως να φορτωθεί οποιαδήποτε κλάση βρίσκεται στο αρχείο αυτό, χωρίς την επανάληψη της μεταφοράς του αρχείου. Στην περίπτωση που αποστέλλεται σαν όρισμα ένα Vector, αυτό περιέχει αντικείμενα τύπου XMLetJarCarrier.

Για να αντιμετωπισθεί το ζήτημα της μεταφοράς των αρχείων τύπου jar, χωρίς την παρέμβαση του πρωτοκόλλου http, έχει υλοποιηθεί η κλάση XMLetJarCarrier, η οποία λειτουργεί ως φορέας των καθαρών byte (raw bytes) του αρχείου, και μπορεί να μεταφερθεί σαν όρισμα μιας RPC κλήσης.

Η παραγωγή των XMLetJarCarrier, δημιουργήθηκε η βοηθητική κλάση XMLetJarCarrierFactory. Χρησιμοποιώντας την μέθοδο

public XMLetJarCarrier getInstance(File file)

επιστρέφεται ένα αντικείμενο XMLetJarCarrier, το οποίο ενθυλακώνει το αρχείο jar σε μορφή απλών bytes, αγνοώντας την εσωτερική του δομή. Κατά την διάρκεια της μετατροπής γίνεται έλεγχος ότι το αρχείου είναι τύπου jar, και ότι έχει διαβαστεί ακέραιο το αρχείο. Κάθε XMLetJarCarrier αντικείμενο μπορεί να περιέχει μόνο ένα αρχείο. Σε περίπτωση που χρειάζεται να αποσταλούν περισσότερα από ένα αρχεία, μπορούν να χρησιμοποιηθούν οι μέθοδοι που δέχονται ως παράμετρο ένα Vector από αντικείμενα XMLetJarCarrier. Ο μοναδικός περιορισμός που προκύπτει από αυτή την προσέγγιση είναι ότι το αντικείμενο XMLetJarCarrier πρέπει να είναι γνωστό και στα δύο άκρα της σύνδεσης.

Το XMLet MBean, αφού λάβει τα αντικείμενα XMLetJarCarrier, μπορεί με κλήση της μεθόδου

public byte[] getByteData();

να διαβάσει τα δεδομένα του αρχείου, και δημιουργώντας ένα JarInputStream να διαβάσει μια-μια τις κλάσεις που περιέχονται. Σε κάθε περίπτωση, τα αρχεία διαβάζονται και τα bytecodes όλων των κλάσεων φυλάγονται στην μνήμη, πριν την αρχικοποίηση οποιουδήποτε αντικειμένου.

Αν κληθεί η **loadMBeanUsingJars**, τότε αρχικοποιείται και εγγράφεται στον MBean Server το σχετικό MBean. Σε περίπτωση που κληθεί η **addJar**, τότε απλά προσθέτουμε βιβλιοθήκες κλάσεων στον XMLet ClassLoader ώστε να μπορούμε να τον χρησιμοποιήσουμε αργότερα χρησιμοποιώντας εντολές της μορφής

public void **loadMBeanUsingXMLet**(String objectName,String className)

Δεδομένου ότι το XMLet MBean κληρονομεί ακέραιη την λειτουργικότητα του MLet, οι βιβλιοθήκες κλάσεων στις οποίες έχει πρόσβαση, μπορούν να επεκταθούν τόσο με την προσθήκη jar αρχείων, όσο και με την προσθήκη URL.

4.4 Υλοποίηση των SNMPProxy και SNMPUtil MBean

4.4.1 SNMPProxy

Όπως έχει αναφερθεί, το MBean αυτό παρέχει μεθόδους που εκτελούν ερωτήματα SNMP. Αρχικά στο MBean αυτό έχουν υλοποιηθεί οι μέθοδοι:

public SNMPResult **snmpGet**(String agent, Integer port, String oid);

public SNMPWalkResult **snmpWalk**(String agent, Integer port, String oid);

Αυτές οι μέθοδοι, αντιστοιχούν στις γνωστές εντολές snmpget και snmpwalk των SNMP εργαλείων που υλοποιήθηκαν στο Carnegie Mellon University (CMU). Η πρώτη μέθοδος

δέχεται ως όρισμα τη DNS διεύθυνση του μηχανήματος που θα ερωτηθεί, την πόρτα στην οποία θα γίνει η ερώτηση (συνήθως χρησιμοποιείται η τυποποιημένη θύρα 161) καθώς και το object id του μετρητή που θα ερωτηθεί. Η μέθοδος αυτή επιστρέφει ένα αντικείμενο τύπου SNMPResult. Το αντικείμενο αυτό περιέχει το object id που ερωτήθηκε, την τιμή που προέκυψε από την ερώτηση, καθώς και την τιμή του μετρητή system.sysUpTime (Object Id = 1.3.6.1.2.1.1.3.0), ώστε να μπορεί να χρησιμοποιηθεί η τιμή που επιστράφηκε στον υπολογισμό ρυθμών και στατιστικών σε σχέση με το χρόνο.

Η δεύτερη μέθοδος δέχεται τα ίδια ορίσματα με την αντίστοιχη get εκδοχή, με την διαφορά ότι το τρίτο όρισμα αποτελεί την ρίζα του υποδέντρου της MIB που θα σαρωθεί. Η συνάρτηση αυτή επιστρέφει ένα αντικείμενο SNMPWalkResult, το οποίο ουσιαστικά περιέχει ένα HashMap με τα ζεύγη «ObjectID,Value», όπως αυτά επιστρέφονται από την σάρωση.

Ειδικά για την περίπτωση του Route Table της συσκευής, έχει υλοποιηθεί η μέθοδος

public SNMPWalkResult **getRouteTable**(String agent);

η οποία δέχεται ως όρισμα μόνο την DNS διεύθυνση ή το IP του μηχανήματος που θα ερωτηθεί. Η μέθοδος εκτελεί μία snmpwalk ερώτηση στο μηχάνημα (στη θύρα 161) για την σάρωση του υποδέντρου ip.ipRouteTable.ipRouteEntry.ipRouteNextHop (Object Id = 1.3.6.1.2.1.4.21.1.7) και επιστρέφει ένα αντικείμενο SNMPWalkResult, όπως αυτό έχει εξηγηθεί πιο πάνω.

4.4.2 SNMPUtil

Το MBean αυτό διαθέτει μόνο μια μέθοδο η οποία εκτελεί υπολογισμό ρυθμών. Συγκεκριμένα διαθέτει την εντολή

public Integer **calculateRate**(SNMPResult res_t1,SNMPResult res_t0);

η οποία δέχεται ως παράμετρο δύο SNMPResult αντικείμενα, με την πληροφορία που βρίσκει σε αυτά κάνει υπολογισμούς χρησιμοποιώντας τη σχέση

$$(X(t1)- X(t0))/(t1-t0)$$

ΚΕΦΑΛΑΙΟ 5
ΕΦΑΡΜΟΓΗ ΠΕΛΑΤΗΣ

Εισαγωγή

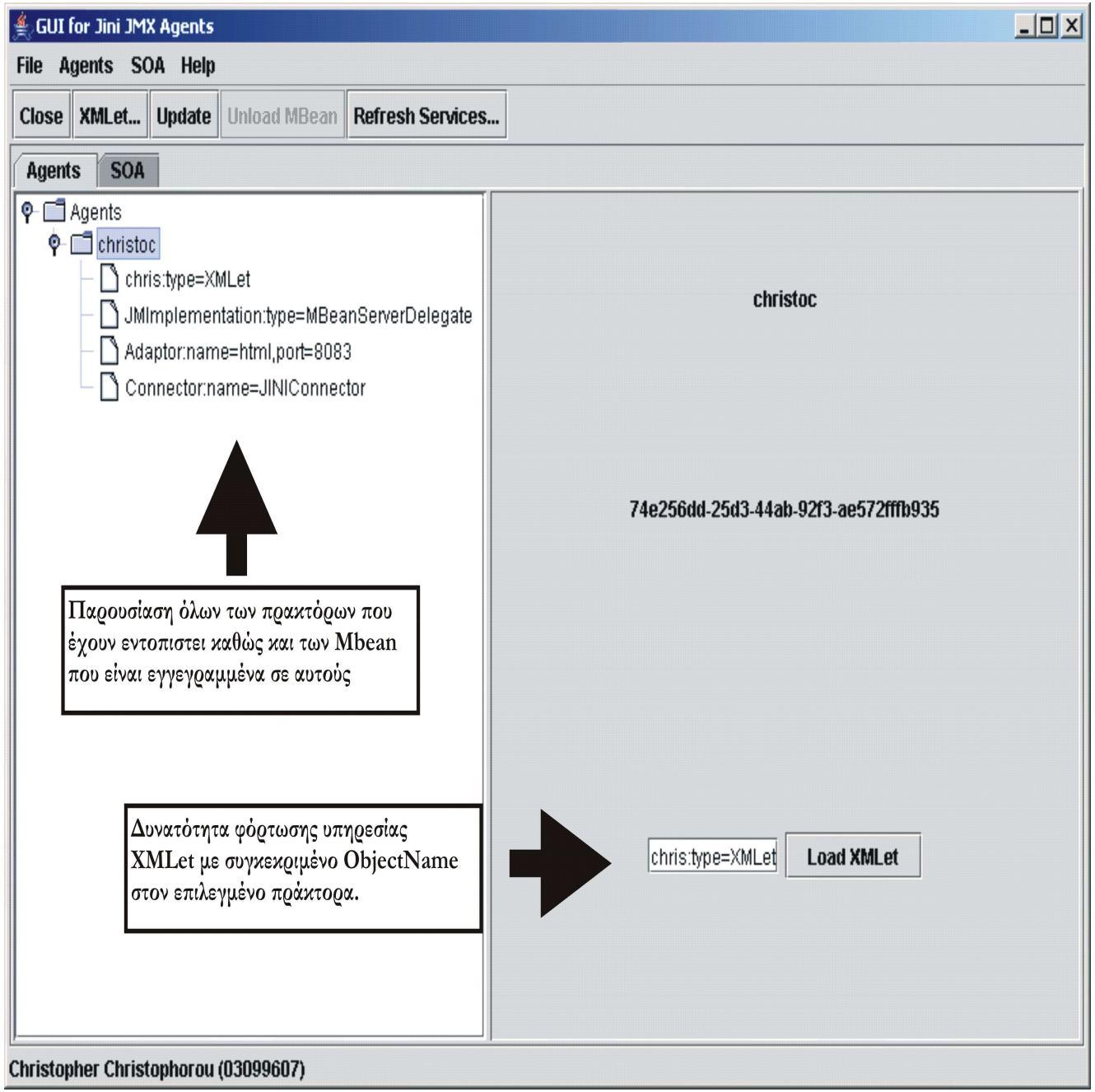
Για τον έλεγχο και την επίδειξη της λειτουργίας του συστήματος, έχει σχεδιαστεί μια απλή εφαρμογή πελάτης με γραφικό interface (GUI), το οποίο παρουσιάζει συνοπτικά όλες τις δυνατότητες του συστήματος όπως αυτό έχει αναπτυχθεί. Το γραφικό περιβάλλον έχει αναπτυχθεί με τη χρήση των Java Swing Components.

Με το που εκκινεί η εφαρμογή πελάτης, ενεργοποιεί τον μηχανισμό ανακάλυψης των υπηρεσιών αναζήτησης, και ακολούθως αρχίζει την αναζήτηση εγγεγραμμένων πρακτόρων JMX. Η λειτουργία αυτή επιτυγχάνεται με την χρήση της βοηθητικής κλάσης Service Discovery Manager όπως αυτή έχει περιγραφή στην παράγραφο των προηγμένων χαρακτηριστικών της υπηρεσίας Jini.

Ακολούθως η εφαρμογή πελάτης, παρουσιάζει σε δενδρική μορφή, όλους τους πράκτορες με τους οποίους απέκτησε επαφή, καθώς και τις υπηρεσίες MBeans που ο καθένας περιέχει.

Σε κάποια άλλη όψη της εφαρμογής, μπορούμε να δούμε το μητρώο των διαθέσιμων υπηρεσιών, και ακολούθως τις λειτουργίες και τα χαρακτηριστικά που διαθέτει η κάθε υπηρεσία χωριστά. Δίδεται επίσης η δυνατότητα κατασκευής απλών σεναρίων λειτουργίας (script) υπό την μορφή αλληλουχίας ενεργειών, ώστε να επιδειχθεί η δυνατότητα χρήσης της κάθε υπηρεσίας με ένα ευέλικτο και αποτελεσματικό τρόπο.

Ακολουθεί μια σειρά από screenshots της εφαρμογής πελάτη, με τις κατάλληλες επεξηγήσεις.

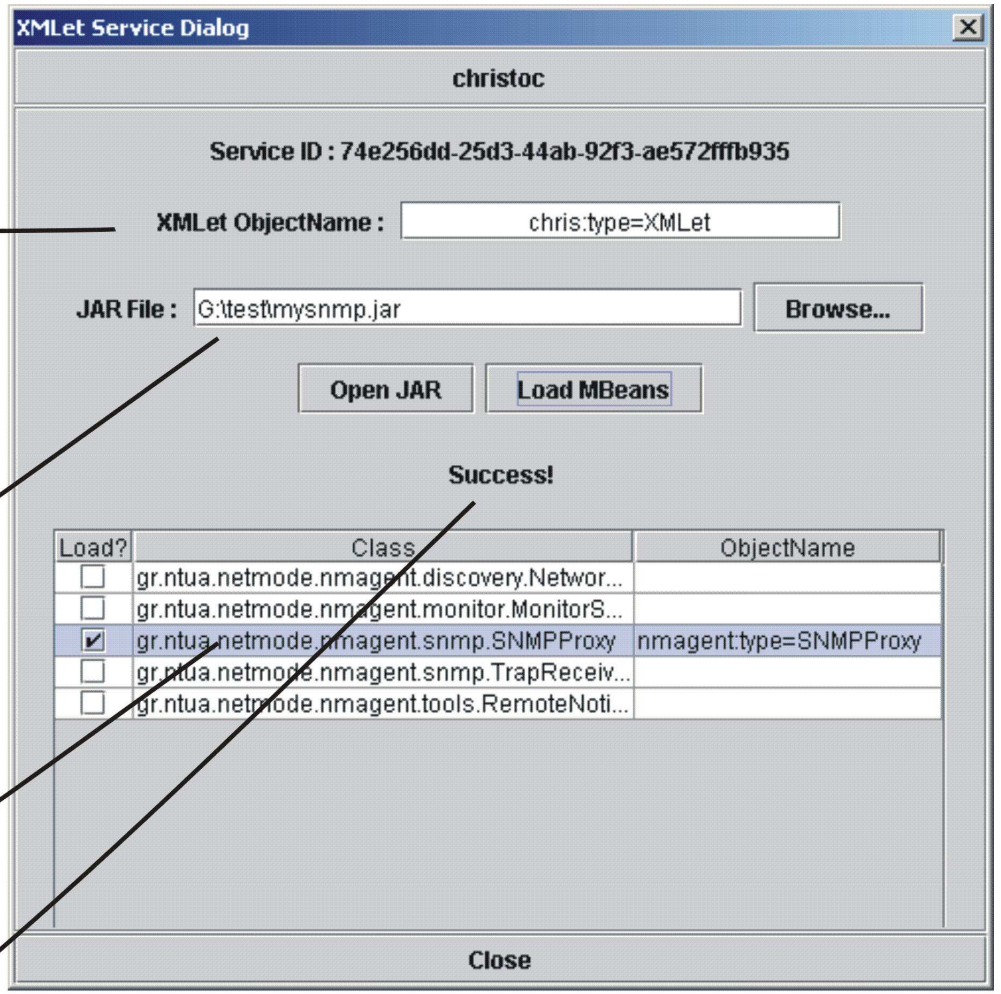

GUI for Jini JMX Agents
File Agents SOA Help
Close XMLet... Update Unload MBean Refresh Services...
Agents SOA
Agents
christoc
chris:type=XMLet
JMImplementation:type=MBeanServerDelegate
Adaptor:name=html,port=8083
Connector:name=JINIConnector
Παρουσίαση όλων των πρακτόρων που έχουν εντοπιστει καθώς και των Mbean που είναι εγγεγραμμένα σε αυτούς
Δυνατότητα φόρτωσης υπηρεσίας XMLet με συγκεκριμένο ObjectName στον επιλεγμένο πράκτορα.
christoc
74e256dd-25d3-44ab-92f3-ae572fffb935
chris:type=XMLet
Load XMLet
Christopher Christophorou (03099607)

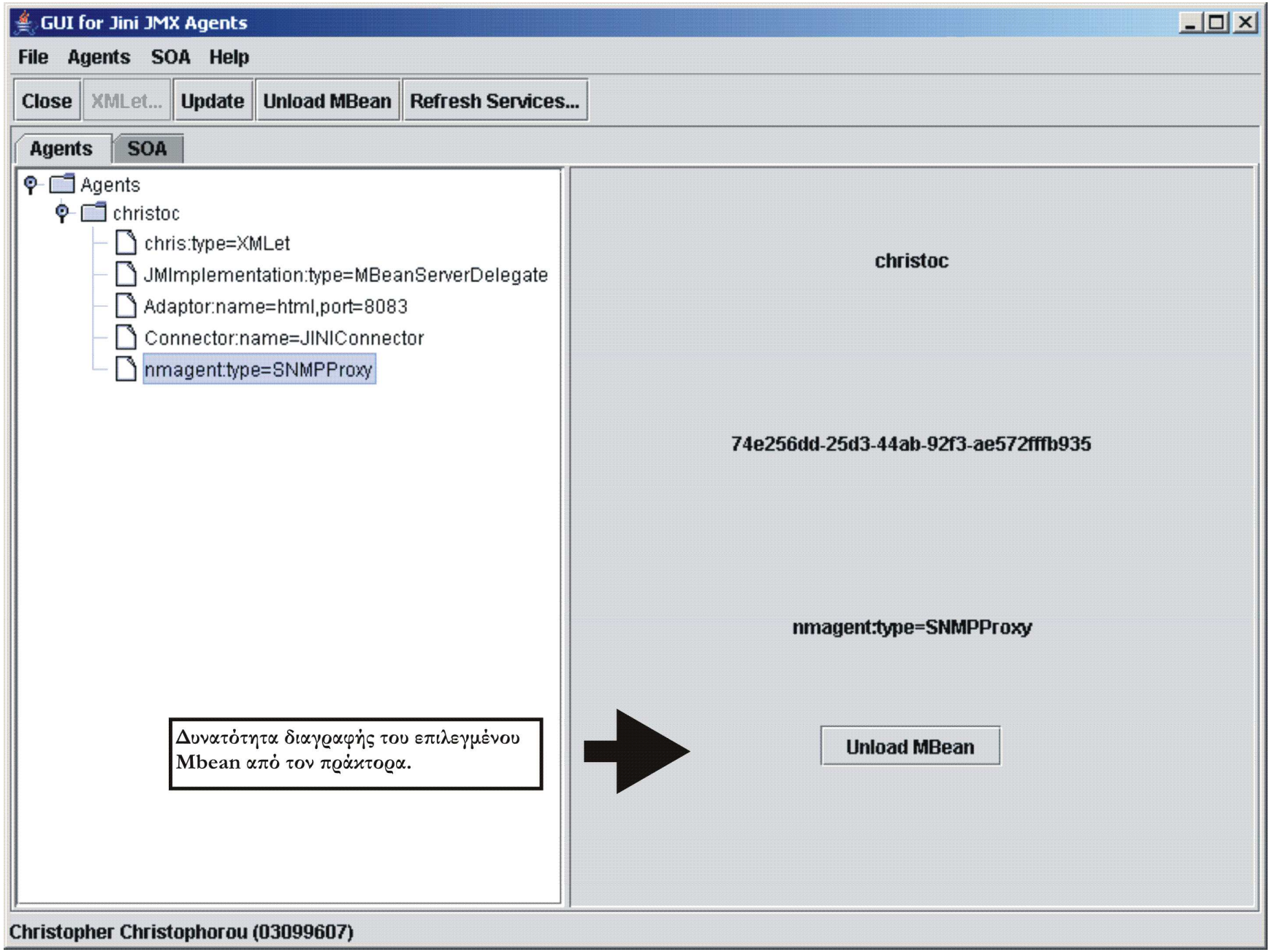
Καθορισμός του στιγμιότυπου της υπηρεσίας XMLet που θα χρησιμοποιηθεί

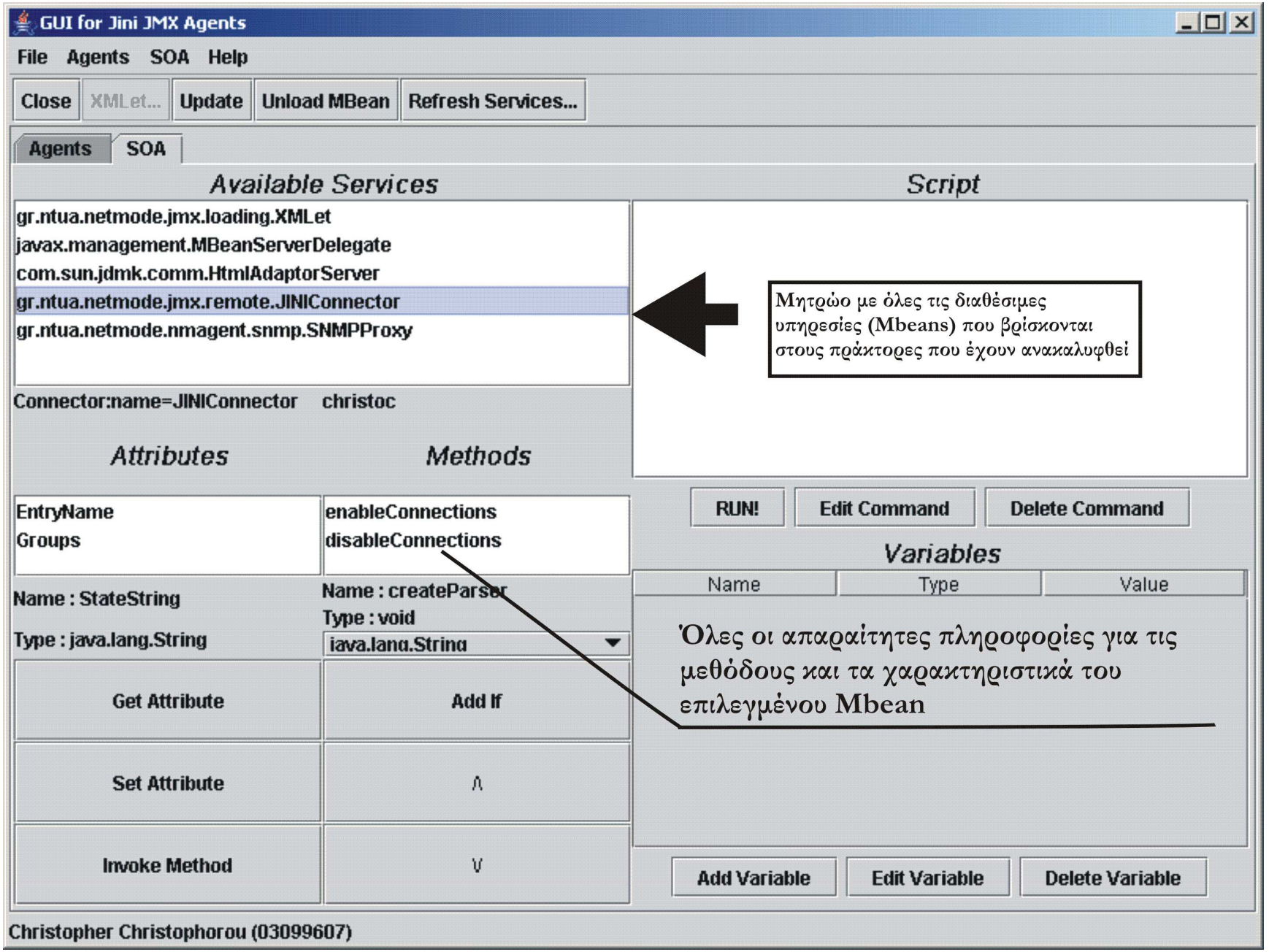
Επιλογή jar αρχείου - από τοπικό δίσκο - που περιέχει τα Mbeans που θέλουμε να στείλουμε στον πράκτορα

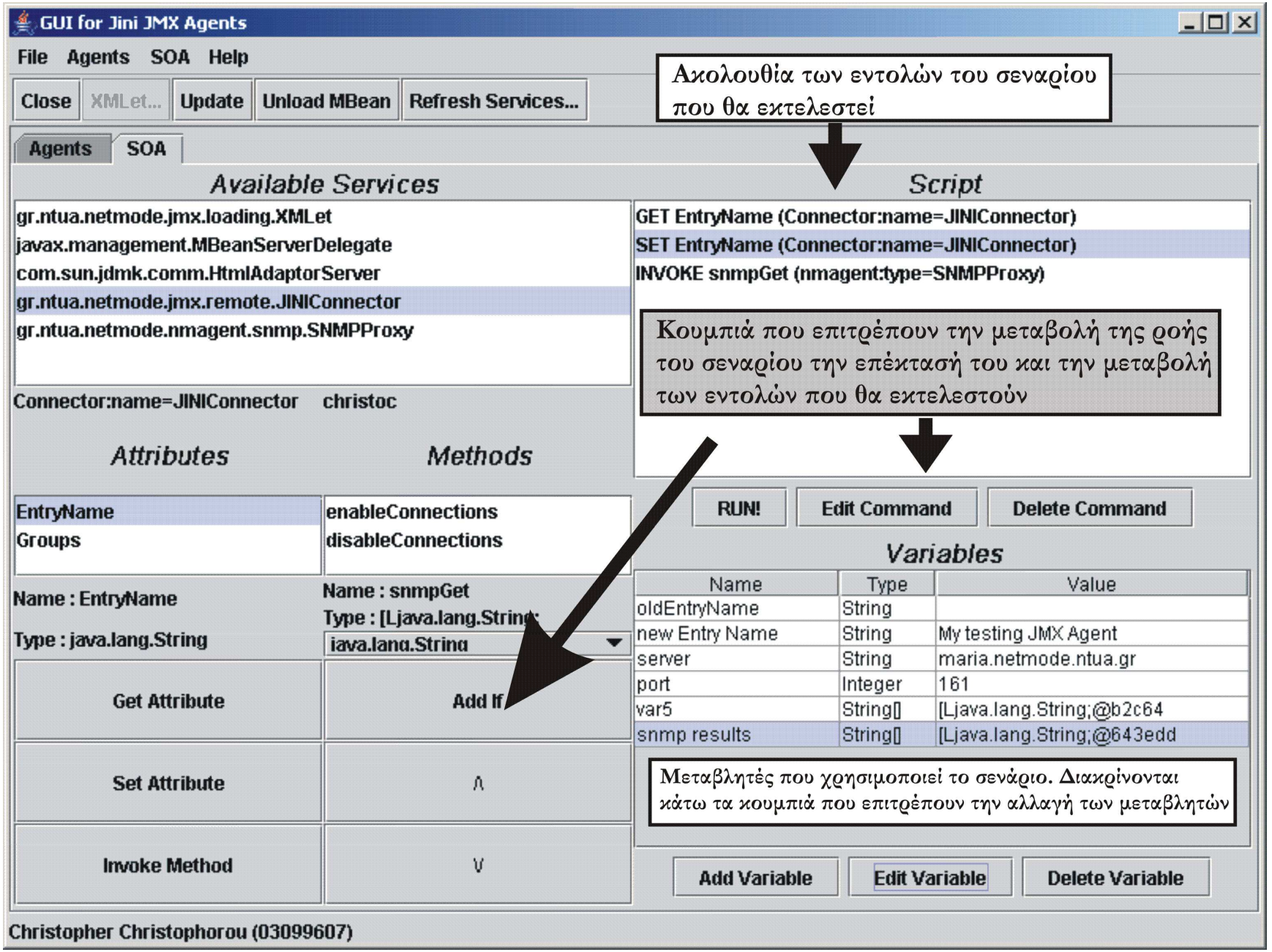
Επιλογή των Mbeans που θέλουμε να εγγραφούν στον πράκτορα και καθορισμός του ObjectName.

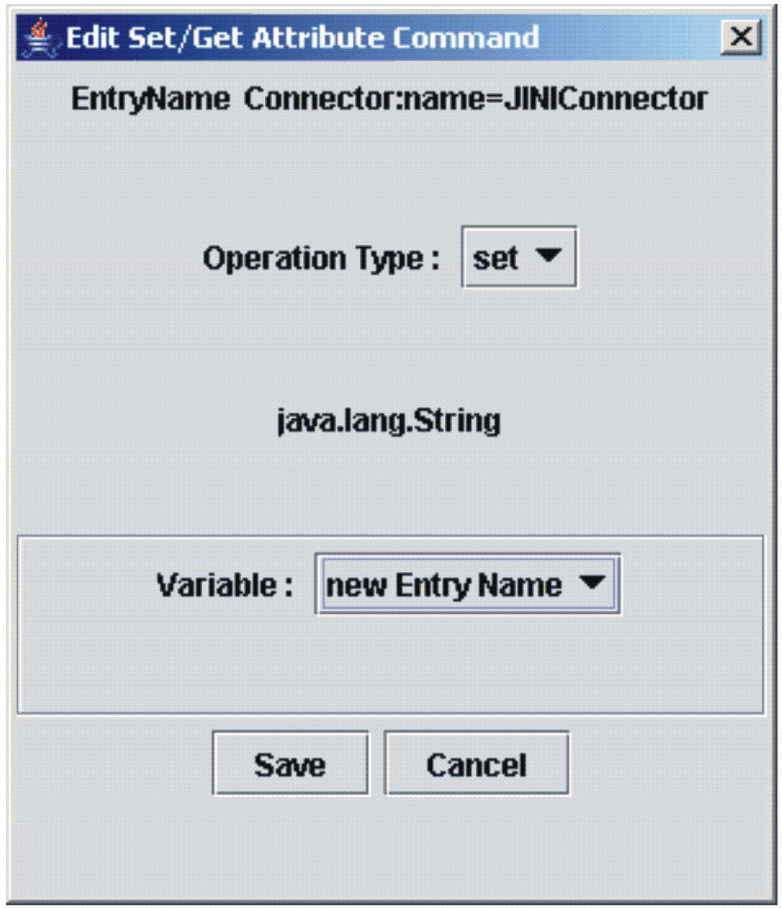
Ενδειξη για την επιτυχή εγγραφή των Mbeans

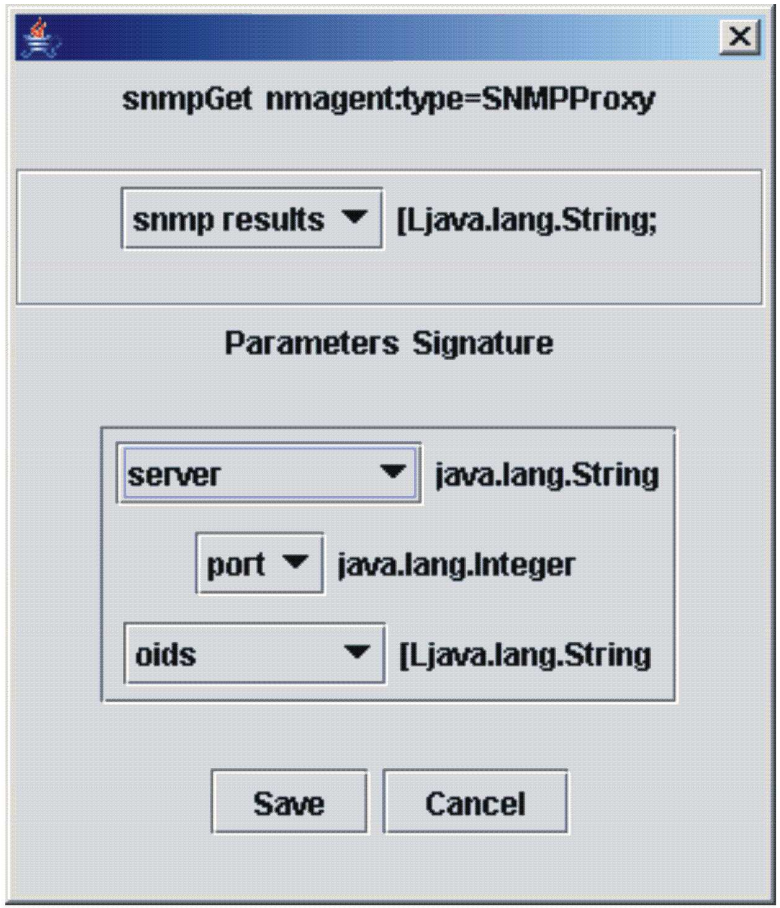



GUI for Jini JMX Agents
File
Agents
SOA
Help
Close
XMLet...
Update
Unload MBean
Refresh Services...
Agents
SOA
Agents
christoc
chris:type=XMLet
JMImplementation:type=MBeanServerDelegate
Adaptor:name=html,port=8083
Connector:name=JINIConnector
nmagent:type=SNMPProxy
christoc
74e256dd-25d3-44ab-92f3-ae572fffb935
nmagent:type=SNMPProxy
Unload MBean
Δυνατότητα διαγραφής του επιλεγμένου Mbean από τον πράκτορα.
Christopher Christophorou (03099607)

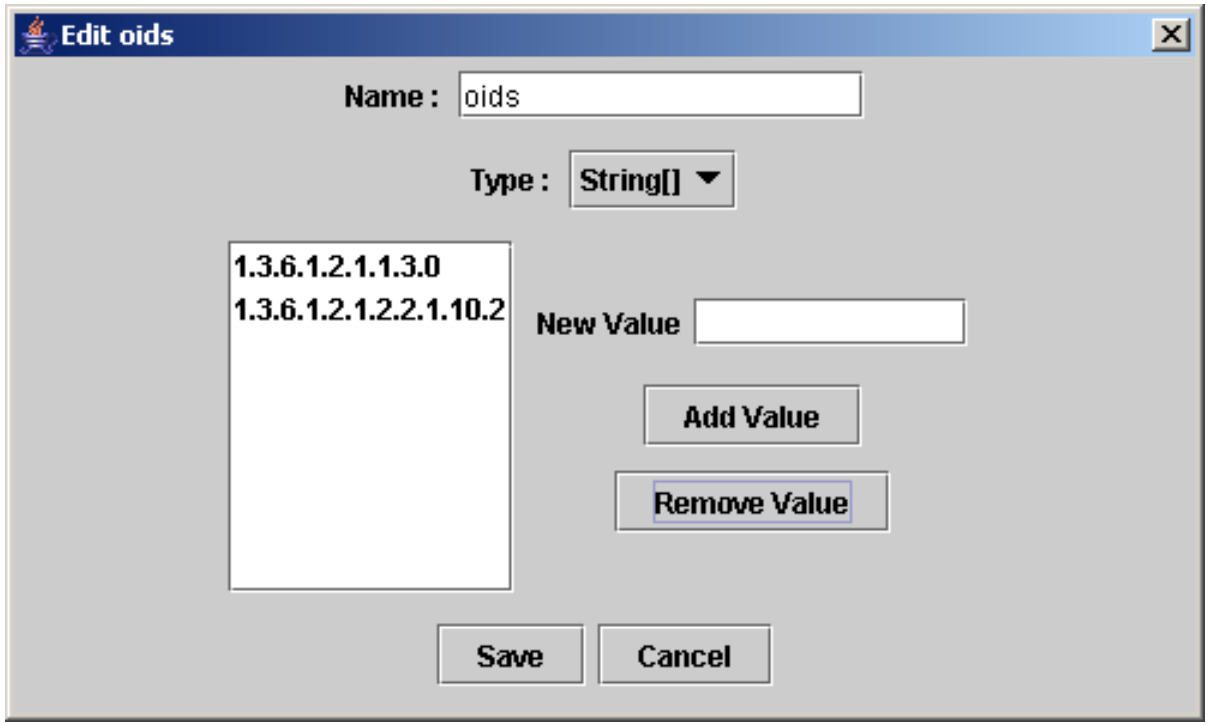

GUI for Jini JMX Agents
File Agents SOA Help
Close XMLet... Update Unload MBean Refresh Services...
Agents SOA
Available Services
gr.ntua.netmode.jmx.loading.XMLet
javax.management.MBeanServerDelegate
com.sun.jdmk.comm.HtmlAdaptorServer
gr.ntua.netmode.jmx.remote.JINIConnector
gr.ntua.netmode.nmagent.snmp.SNMPProxy
Connector:name=JINIConnector christoc
Attributes
Methods
EntryName
Groups
enableConnections
disableConnections
Name : StateString
Type : java.lang.String
Name : createParser
Type : void
java.lang.String
Get Attribute
Add If
Set Attribute
Invoke Method
Script
Μητρώο με όλες τις διαθέσιμες υπηρεσίες (Mbeans) που βρίσκονται στους πράκτορες που έχουν ανακαλυφθεί
RUN!
Edit Command
Delete Command
Variables
Name
Type
Value
Όλες οι απαραίτητες πληροφορίες για τις μεθόδους και τα χαρακτηριστικά του επιλεγμένου Mbean
Add Variable
Edit Variable
Delete Variable
Christopher Christophorou (03099607)


GUI for Jini JMX Agents
File Agents SOA Help
Close XMLet... Update Unload MBean Refresh Services...
Agents SOA
Available Services
gr.ntua.netmode.jmx.loading.XMLet
javax.management.MBeanServerDelegate
com.sun.jdmk.comm.HtmlAdaptorServer
gr.ntua.netmode.jmx.remote.JINIConnector
gr.ntua.netmode.nmagent.snmp.SNMPProxy
Connector:name=JINIConnector christoc
Attributes
Methods
EntryName
Groups
enableConnections
disableConnections
Name : EntryName
Type : java.lang.String
Name : snmpGet
Type : [Ljava.lang.String
Get Attribute
Add If
Set Attribute
Invoke Method
Christopher Christophorou (03099607)
Ακολουθία των εντολών του σεναρίου που θα εκτελεστεί
Script
GET EntryName (Connector:name=JINIConnector)
SET EntryName (Connector:name=JINIConnector)
INVOKE snmpGet (nmagent:type=SNMPProxy)
Κουμπιά που επιτρέπουν την μεταβολή της ροής του σεναρίου την επέκτασή του και την μεταβολή των εντολών που θα εκτελεστούν
RUN! Edit Command Delete Command
Variables
Name Type Value
oldEntryName String
new Entry Name String My testing JMX Agent
server String maria.netmode.ntua.gr
port Integer 161
var5 String[] [Ljava.lang.String;@b2c64
snmp results String[] [Ljava.lang.String;@643edd
Μεταβλητές που χρησιμοποιεί το σενάριο. Διακρίνονται κάτω τα κουμπιά που επιτρέπουν την αλλαγή των μεταβλητών
Add Variable Edit Variable Delete Variable



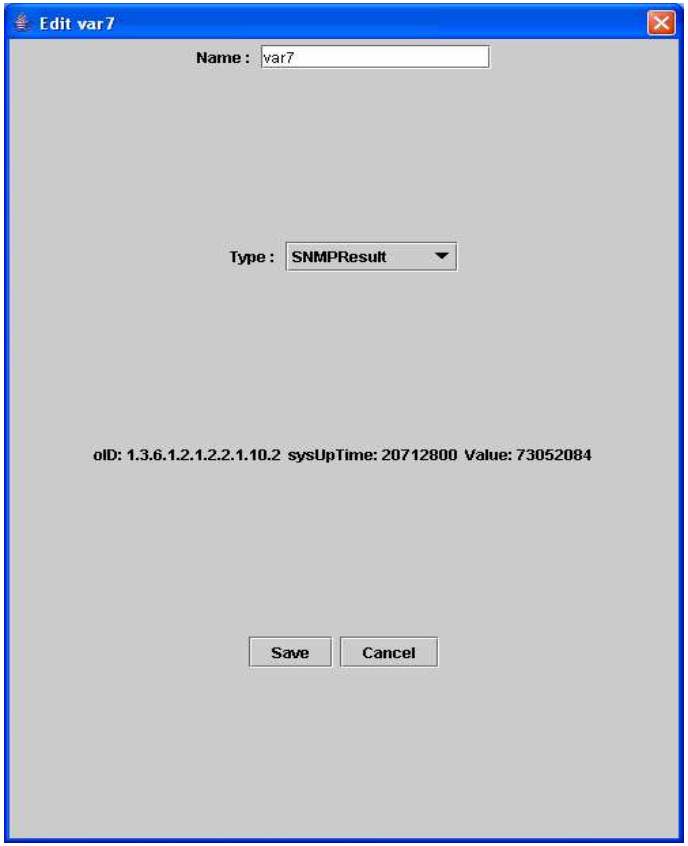


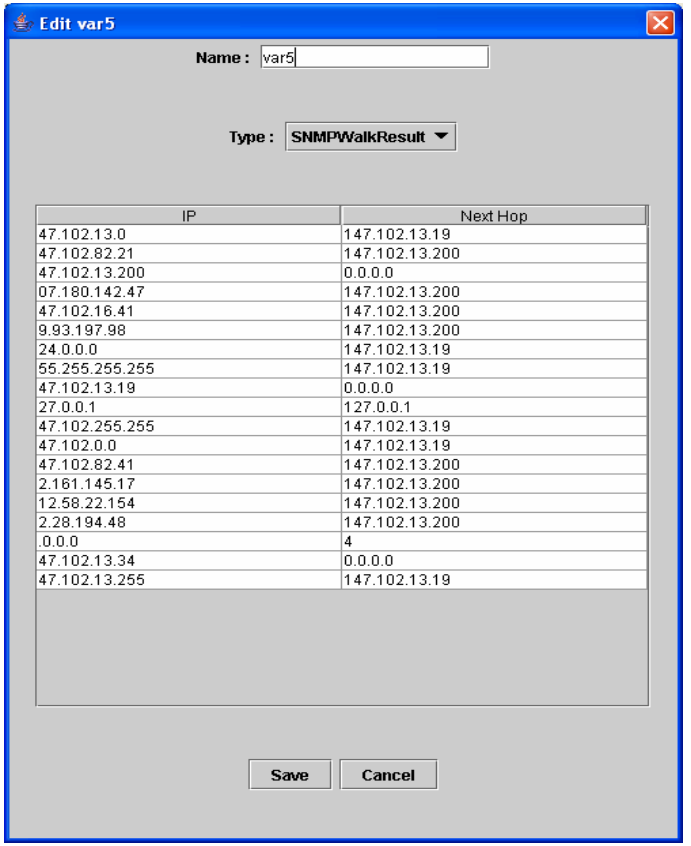
Παράθυρα διαλόγου αλλαγής παραμέτρων εκτέλεσης εντολών
Το παράθυρο αριστερά αφορά εντολές ανάγνωσης ή εγγραφής χαρακτηριστικών
Το παράθυρο δεξιά αφορά την κλήση μεθόδου μιας υπηρεσίας



Παράθυρο αλλαγής των χαρακτηριστικών μιας μεταβλητής. Μπορεί να οριστεί το όνομα της μεταβλητής, ο τύπος της (String, String[] ή Integer) και να καθοριστεί η αρχική τιμή.

Στην πιο πάνω περίπτωση, έχουμε μια μεταβλητή τύπου String[], στην οποία τοποθετούμε μια σειρά από object id's. Η συγκεκριμένη μεταβλητή θα δοθέι ως παράμετρος σε ένα SNMPProxy MBean. Πιο πάνω ζητούμε τις παραμέτρους system.sysUpTime και interfaces.ifTable.ifEntry.ifInOctets για το interface 2.





Χρησιμοποιώντας το ίδιο παράθυρο διαλόγου, μπορούμε να δούμε τις τιμές επιστροφής μιας κλήσης μεθόδου.

Στην πιο πάνω περίπτωση, έχουμε μεταβλητές τύπου SNMPResult και SNMPWalkResult, όπου μας έχουν επιστραφεί τα αποτελέσματα ενός SNMP ερωτήματος στο μηχάνημα maria.netmode.ntua.gr καθώς και το RouteTable του μηχανήματος. Χρησιμοποιούμε αυτή την πληροφορία για να υπολογίσουμε το throughput στο συγκεκριμένο interface.

ΚΕΦΑΛΑΙΟ 6
ΒΙΒΛΙΟΓΡΑΦΙΑ - ΑΝΑΦΟΡΕΣ

1. **Σχεδίαση και Υλοποίηση Κατανεμημένης Εφαρμογής Διαχείρισης Δικτύου με χρήση JMX και XML**
   Καφαντάρης Παναγιώτης, Λένης Άγγελος
   Εργαστήριο Διαχείρισης και Βέλτιστου Σχεδιασμού Δικτύων – Ε.Μ.Π.

2. **Java Management Extensions Instrumentation and Agent Specification v 1.2**
   (Sun Microsystems)
   http://java.sun.com/products/JavaManagement/

3. **Java Management Extensions (JMX) Technology Overview**
   (JMX Remote API version 1.0.1 – February 2004)
   http://java.sun.com/products/JavaManagement/

4. **Java Management Extensions**
   Tal Cohen IBM Labs in Haifa
   http://webcourse.cs.technion.ac.il/236606/ Spring2003/ho/WCFiles/236606 Lecture13.ppt

5. **Java Management Extensions 1.2.1 API Specification**
   http://java.sun.com/products/JavaManagement/

6. **MX4J - Open Source Java Management Extensions**
   http://mx4j.sourceforge.net/

7. **The ClassLoader – Java Distributed Computing**
   http://skaiste.elekta.lt/Books/O'Reilly/Bookshelfs/books/javaenterprise/dist/ch02_03.htm

8. **Reading files from JARs and Getting Started with the JMX**
   http://java.sun.com/developer/JDCTechTips/2003/tt0122.html

9. **ClassLoader tips**
   http://www.brainopolis.com/roller/trackback/kduffey/Weblog/classloader_tips_part_1

10. **Jini™ Architecture Specification**
    www.jini.org/nonav/standards/ davis/doc/specs/html/jini-spec.html

11. **Jini™ Network Technology – An Executive Overview**
    (Sun Microsystems - February 2001)
    http://wwws.sun.com/software/jini/whitepapers/jini-execoverview.pdf

12. **New to SOA and Web Services**
    http://www-106.ibm.com/developerworks/webservices/newto/

13. **Service Oriented Architecture Using Jini**
Ron Dearing, Cysive Inc, - 06 Jan 2003
http://www.searchwebservices.com/originalContent/0,289142,sid26_gci871817,00.html

14. **What is Service Oriented Architecture**
Hao He – 30 September 2003
http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html

15. **Migrating to Service Oriented Architecture Parts 1-2**
Kishore Channabasavaiah, Kerrie Holley, Edward M. Tuggle, Jr., - 16 December 2003
ftp://www6.software.ibm.com/software/developer/library/ws-migratesoa.pdf

16. **Service-Oriented Architecture Explained**
Sayed Hashimi - 08/18/2003
http://www.ondotnet.com/pub/a/dotnet/2003/08/18/soa_explained.html

17. **Business Process Execution Language for Web Services (BPEL4WS)**
Version 1.1 - 5 May 2003
http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

18. **BPELJ: BPEL for Java**
A Joint White Paper by BEA and IBM - March 2004
http://ftpna2.bea.com/pub/downloads/ws-bpelj.pdf

19. **BPEL for Programmers and Architects**
Paul Brown, Maciej Szefler - 2003-12-16
http://blog.fivesight.com/prb/space/BPEL/BPEL4ProgArchies.pdf