ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Επιστημονικοί Υπολογισμοί στον Παγκόσμιο Ιστό

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

Γιώργου Σαλούστρου

Βόλος, 09-10- 2008

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Επιστημονικοί Υπολογισμοί στον Παγκόσμιο Ιστό

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΓΙΩΡΓΟΥ ΣΑΛΟΥΣΤΡΟΥ

Επιβλέποντες : Βάβαλης Εμμανουήλ, Αναπληρωτής Καθηγητής Π.Θ.
Τσομπανοπούλου Γιώτα, Επίκουρος Καθηγήτρια Π.Θ.

Εγκρίθηκε από τη διμελή εξεταστική επιτροπή την 09-10-2008

(Υπογραφή)
.....
Βάβαλης Εμμανουήλ
Αν. Καθηγητής Παν. Θεσσαλίας

(Υπογραφή)
.....
Τσομπανοπούλου Γιώτα
Επίκουρος Καθηγήτρια Παν. Θεσσαλίας

Η σελίδα αυτή είναι σκόπιμα λευκή.

Πίνακας περιεχομένων

1 Εισαγωγή

Ο τομέας του scientific computing ασχολείται με τη δημιουργία μαθηματικών μοντέλων και τη χρήση τους στην ανάπτυξη εφαρμογών λογισμικού, με στόχο την επίλυση απαιτητικών, από άποψη υπολογιστικής ισχύος, επιστημονικών προβλημάτων και προβλημάτων Μηχανικής. Ιστορικό ορόσημο για τη θεμελίωση του τομέα θεωρείται το έτος 1947 με την επιστημονική μελέτη των John von Neumann και Herman Goldstine με τίτλο "Numerical Inverting of Matrices of High Order", η οποία είναι από τις πρώτες που ασχολείται με σφάλματα στρογγυλοποίησης. Ο τομέας αυτός έχει μακρά ιστορία και είναι προγενέστερος της Δικτύωσης των Ηλεκτρονικών Υπολογιστών και του World Wide Web που έχουν φέρει την επανάσταση της πληροφορίας, την οποία βιώνουμε στις μέρες μας. Εκτός από τη χρήση του δικτύου για την ανταλλαγή πληροφοριών, ενδιαφέρον παρουσιάζει και η μελέτη των δυνατοτήτων που προσφέρει για πρόσβαση σε υπολογιστική ισχύ.

Η υφιστάμενη κατάσταση στον τομέα του scientific computing, ως προς την οπτική του πώς κάποιος με ένα δεδομένο πρόβλημα μπορεί να εκτελέσει επιστημονικούς υπολογισμούς, δε διαφέρει σημαντικά από το παρελθόν. Τα βήματα που πρέπει να ακολουθηθούν είναι η εύρεση της κατάλληλης βιβλιοθήκης για το δεδομένο πρόβλημα, η λήψη και η εγκατάσταση τοπικά αυτής και στη συνέχεια η εκτέλεση των υπολογισμών σε κάποιο μηχάνημα. Η διαδικασία, όμως, αυτή παρουσιάζει κάποια προβλήματα.

Η επίλυση επιστημονικών προβλημάτων μπορεί να γίνει απαιτητική τόσο σε μνήμη όσο και σε επεξεργαστική ισχύ. Αυτό σημαίνει ότι, για να εκτελέσει κάποιος επιστημονικούς υπολογισμούς για την επίλυση κάποιου προβλήματος με μεγάλη πολυπλοκότητα, θα πρέπει να έχει στην κατοχή του ή να διαθέτει εξουσιοδοτημένη πρόσβαση σε ειδικό υλικό, το οποίο μπορεί να είναι είτε κάποιος supercomputer, είτε κάποια computational clusters. Αυτό συνεπάγεται κόστος τόσο για την αγορά του εξειδικευμένου υλικού, όσο και για τη διαχείριση και συντήρησή του. Επίσης, η ανάγκη για την εκτέλεση υπολογισμών μπορεί να

μην είναι ιδιαίτερα συχνή, οπότε το κόστος για την αγορά και συντήρηση εξοπλισμού, ο οποίος θα μένει ανενεργός για μεγάλο χρονικό διάστημα, κρίνεται περιττό.

Η χρήση εξειδικευμένων βιβλιοθηκών για scientific computing εγείρει επιπλέον προβλήματα κυρίως ως προς τις εξαρτήσεις λογισμικού που κάποιος μπορεί να συναντήσει κατά την εγκατάστασή τους. Για να χρησιμοποιήσει κάποιος μία βιβλιοθήκη, πρέπει, καταρχάς, να την αναζητήσει μόνος του και στη συνέχεια να επιλύσει τις εξαρτήσεις λογισμικού που μπορεί να προκύψουν, ανάλογα με τη στοχευόμενη πλατφόρμα (π.χ. εύρεση του κατάλληλου compiler, εύρεση και εγκατάσταση άλλων τυχόν εξαρτήσεων), οι οποίες δεν είναι προφανές ότι μπορούν εύκολα να λυθούν. Έτσι, δεν παρέχεται δυναμική και ομοιόμορφη πρόσβαση στην πληθώρα βιβλιοθηκών για scientific computing, οι οποίες περικλείουν τη γνώση και την εμπειρία της επιστημονικής κοινότητας.

1.1 Αντικείμενο διπλωματικής

Στα πλαίσια αυτής της εργασίας, προτείνουμε μια διαφορετική προσέγγιση για την εκτέλεση επιστημονικών υπολογισμών, η οποία δίνει λύση στα παραπάνω προβλήματα. Το μοντέλο, που εισάγουμε για την εκτέλεση επιστημονικών υπολογισμών, περιλαμβάνει την έκθεση μεθόδων (legacy code) για επιστημονικούς υπολογισμούς ως Web Services και την ανάπτυξη εφαρμογών για τη διενέργεια αυτών με τη χρήση επιστημονικών ροών εργασίας.

Επικεντρωθήκαμε στην έκθεση βιβλιοθηκών για scientific computing γραμμένων σε παραδοσιακές γλώσσες προγραμματισμού, συγκεκριμένα σε fortran77, ως Web Services. Με την παροχή βιβλιοθηκών, ως υπηρεσιών προσβάσιμων μέσω στάνταρντ πρωτοκόλλων του Διαδικτύου, όπως το HTTP και το SOAP, οι οποίες αποτελούν και τον κύριο κορμό του scientific computing, αλλάζει ριζικά η υφιστάμενη κατάσταση στον τρόπο με τον οποίο εκτελούμε επιστημονικούς υπολογισμούς.

Οι υπηρεσίες αυτές μπορούν να ανακαλυφθούν δυναμικά από μία εφαρμογή κατά το χρόνο εκτέλεσης μέσω κάποιου UDDI repository (βλ. 3.1.3). Επίσης, υιοθετούν ένα μοντέλο χαλαρά συνδεδεμένων συστημάτων, το οποίο επιτυγχάνει διαλειτουργικότητα (ανεξαρτησία γλώσσας-πλατφόρμας) σε ένα ανομοιογενές περιβάλλον, όπως είναι το Διαδίκτυο.

Η έκθεση της προγραμματιστικής λογικής τέτοιων βιβλιοθηκών ως Web Services επιτυγχάνει μεγάλη διαφάνεια για τον τελικό χρήστη, καθώς μπορεί να έχει πρόσβαση σε αυτές μέσω στάνταρντ πρωτοκόλλων του Διαδικτύου. Η ίδια υπηρεσία μπορεί, για παράδειγμα, να τρέχει τοπικά σε κάποιο server, σε κάποιο computational cluster ή σε grid περιβάλλον, λεπτομέρειες που δεν απασχολούν τον τελικό χρήστη, καθώς ο τρόπος πρόσβασης παραμένει ίδιος.

Με τη χρήση αυτούσιων βιβλιοθηκών επιστημονικού υπολογισμού αποφεύγουμε τον επανασχεδιασμό τους, ο οποίος έχει μεγάλο κόστος, και εκμεταλλευόμαστε την

αποδεδειγμένη υψηλή αποδοτικότητα. Επιπλέον, με τη χρήση ενός τέτοιου μοντέλου, ο τελικός χρήστης αποκτά πρόσβαση σε υπολογιστική ισχύ (cluster ή grid περιβάλλον) με διαφανή τρόπο.

Συνοψίζοντας, στην παρούσα εργασία ασχοληθήκαμε με τα παρακάτω ζητήματα:

1. Εκθέσαμε μεθόδους της LAPACK, γραμμένων σε fortran77, ως Web Services.
2. Σχεδιάσαμε ροές εργασίας με τη χρήση της γλώσσας BPEL.
3. Μελετήσαμε τις δυνατότητες για ανάπτυξη Web Services στην υποδομή του Hellas Grid - EGEE.

1.2 Οργάνωση κειμένου

Στο 2^{ο} κεφάλαιο γίνεται αναφορά σε σχετικές προσπάθειες που έχουν γίνει στην ανάπτυξη συστημάτων για scientific computing. Στο 3^{ο} κεφάλαιο περιγράφουμε τις τεχνολογίες που χρησιμοποιήσαμε, ενώ στο 4^{ο} τη δομή και την υλοποίηση του συστήματος. Τέλος, στο 5^{ο} κεφάλαιο παρουσιάζεται η μελέτη γύρω από την ανάπτυξη Web Services για την εκτέλεση υπολογισμών στο EGEE.

2 Σχετικές δραστηριότητες και εργασίες

Την τελευταία δεκαετία, η μεγάλη πρόοδος στη δικτύωση των ηλεκτρονικών υπολογιστών και η ανάπτυξη του World Wide Web έχουν οδηγήσει στην επανάσταση της πληροφορίας, την οποία βιώνουμε στις μέρες μας. Πέραν όμως της χρήσης του δικτύου για την ανταλλαγή πληροφοριών, πολλές ερευνητικές προσπάθειες έχουν επικεντρωθεί στην ανάπτυξη κατανεμημένων συστημάτων για την αξιοποίηση του δικτύου σε διαφανή πρόσβαση σε υπολογιστική ισχύ. Στόχος των παραπάνω είναι να παρέχουν στο χρήστη ένα πολύτιμο εργαλείο για την επίλυση προβλημάτων επιστημονικού υπολογισμού και Μηχανικής. Στην ενότητα 2.1 παρουσιάζονται ερευνητικές προσπάθειες σε ό, τι αφορά την κατασκευή περιβαλλόντων για επιστημονικούς υπολογισμούς (Problem Solving Environments) προσβάσιμων μέσω του δικτύου, στην ενότητα 2.2 εργασίες σχετικά με την ανάπτυξη συστημάτων για grid computing [32] και τέλος στην ενότητα 2.3 εργαλεία για τη σύνθεση επιστημονικών ροών εργασίας (scientific flows).

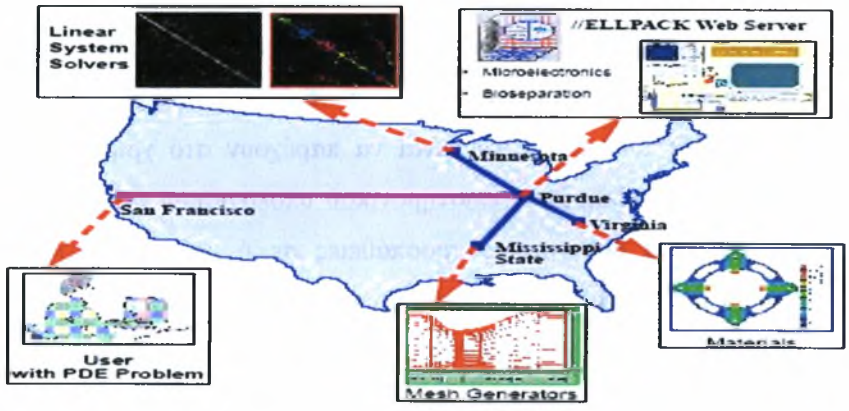
2.1 Problem Solving Environments

Τα περιβάλλοντα για επιστημονικούς υπολογισμούς (PSE's) σχεδιάστηκαν στο παρελθόν και συνεχίζουν να αναπτύσσονται, με σκοπό να εξυπηρετήσουν την ανάγκη ενσωμάτωσης διαφορετικών βιβλιοθηκών για scientific computing, παρέχοντας στο μη έμπειρο χρήστη διεπαφές υψηλού επιπέδου και την αλληλεπίδραση με αυτές. Τα χαρακτηριστικά στοιχεία στις διεπαφές αυτές περιλαμβάνουν μία γλώσσα υψηλού επιπέδου για τον ορισμό νέων διαδικασιών επίλυσης προβλημάτων, καθώς και γραφικές διεπαφές τόσο για την αναπαράσταση προβλημάτων, όσο και για την οπτικοποίηση των λύσεων.

Από τις αρχικές ερευνητικές προσπάθειες (1995) για την ανάπτυξη PSE's προσβάσιμων μέσω του δικτύου είναι το **WEB//ELLPACK** [9], όπως και ο απόγονός του **PELLPACK** [24] από το Πανεπιστήμιο του Purdue.

Τα παραπάνω συστήματα λειτουργούν ως PSE's για την επίλυση Μερικών Διαφορικών Εξισώσεων (ΜΔΕ). Τα περιβάλλοντα αυτά, ως PSE's, ενσωματώνουν μία πληθώρα βιβλιοθηκών για επίλυση ΜΔΕ, παρέχοντας μια διεπαφή, που τηρεί τα παραπάνω χαρακτηριστικά, δηλαδή μια γλώσσα υψηλού επιπέδου για την επίλυση και τον ορισμό προβλημάτων ΜΔΕ, καθώς και των σχημάτων επίλυσής τους. Επίσης, περιλαμβάνουν μια γραφική διεπαφή, η οποία βοηθάει τους χρήστες τόσο στον ορισμό προβλημάτων ΜΔΕ, όσο και στην οπτικοποίηση των λύσεών τους. Η εκτέλεση είναι δυνατή σε μια ποικιλία πλατφόρμων, ενώ δίνεται και η δυνατότητα τόσο της σειριακής όσο και της παράλληλης επίλυσης προβλημάτων.

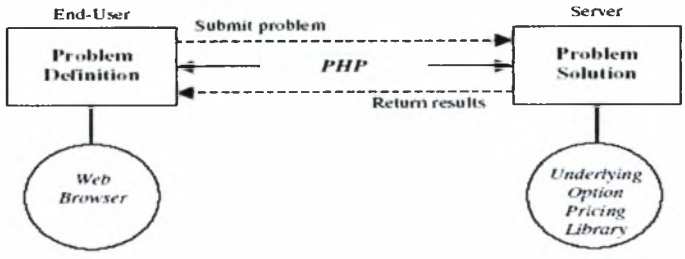
Η δυνατότητα πρόσβασης και στα δύο συστήματα για τις εφαρμογές πελάτη δε στηρίζεται σε στάνταρντ τεχνολογίες του Διαδικτύου, αλλά στην κατάλληλη ρύθμιση του X Window System [33], γεγονός που οφείλεται στο ότι αναπτύχθηκε σε μία περίοδο, όπου οι τεχνολογίες web ήταν στα αρχικά στάδια ανάπτυξής τους σε σχέση με νεότερες προσεγγίσεις. Στο σχήμα 2.1 παρουσιάζεται μία πανοραμική άποψη της αρχιτεκτονικής του Web//ELLPACK.



Σχήμα 2.1: Web//ELLPACK

Μία νεότερη προσέγγιση, σε σχέση με τα παραπάνω συστήματα, είναι το "A Web-Based Problem Solving Environment for Solution of Option Pricing Problems and Comparison of Methods" [4], το οποίο προέρχεται από το Πανεπιστήμιο Πατρών (2002).

Το παραπάνω σύστημα είναι μία διαδικτυακή πλατφόρμα, η οποία ενσωματώνει ποικίλες βιβλιοθήκες για option pricing problems [34]. Η σχεδίασή της έχει γίνει βάσει μίας ανοιχτής αρχιτεκτονικής βασισμένης σε WWW τεχνολογίες (html, php, http). Η εφαρμογή πελάτης έχει πρόσβαση στο σύστημα μέσω ενός web browser. Μέσα από το γραφικό περιβάλλον της web διεπαφής δίνεται ο ορισμός του προβλήματος, ενώ παρέχεται και η δυνατότητα οπτικής αναπαράστασης της εξόδου. Στο σχήμα 2.2 παρουσιάζεται η αρχιτεκτονική του συστήματος.



Σχήμα 2.2: Web-Based Problem Solving Environment

2.2 Systems for Grid Computing

Ο όρος Grid Computing χρησιμοποιήθηκε για πρώτη φόρα στις αρχές της δεκαετίας του 1990 στην εργασία των Ian Foster και Carl Kesselmans "The Grid: Blueprint for a new computing infrastructure", προκειμένου να δηλώσει ότι στο μέλλον η πρόσβαση σε υπολογιστική ισχύ θα γίνει τόσο απλή, όσο και η πρόσβαση στο δίκτυο ηλεκτρικής ενέργειας (ο ξένος όρος είναι electrical power grid).

Τα συστήματα για grid computing έχουν ως στόχο το διαμοιρασμό υπολογιστικών πηγών, προκειμένου να παρέχουν διαφανή πρόσβαση σε υπολογιστική ισχύ, κρύβοντας από το χρήστη την ανομοιογένεια και μεταβλητότητα ενός grid περιβάλλοντος. Σε αντίθεση με τα παραδοσιακά παράλληλα συστήματα (computer clusters) και τους υπερυπολογιστές (super computers), τα συστήματα για grid computing υιοθετούν ένα μοντέλο χαλαρά συνδεδεμένων συστημάτων, όπου οι πόροι μπορεί να συνδέονται και μέσω ενός δικτύου, το οποίο όμως δε δίνει εγγυήσεις όπως το δημόσιο δίκτυο (internet), το υλικό (hardware) μπορεί να είναι ανομοιογενές, ενώ υπάρχει και η δυνατότητα δυναμικής προσθήκης νέων πόρων και χρηστών. Παρακάτω παρουσιάζονται ορισμένα συστήματα για Grid Computing.

Το **Netsolve** [12] είναι middleware υλοποιημένο σε C, το οποίο γεφυρώνει standard προγραμματιστικές διεπαφές με περιβάλλοντα επιστημονικού υπολογισμού, όπως το Matlab, Mathematica, Octave κ.λπ., τα οποία κυριαρχούν στην επιστημονική κοινότητα. Είναι σχεδιασμένο από το Πανεπιστήμιο του Tenesee, ενώ η πρώτη έκδοση παρουσιάστηκε το 2002.

Το Netsolve/Gridsolve αποτελείται από 3 οντότητες, όπως παρουσιάζεται στο σχήμα 2.4, τον Client, τον Agent και τον Server:

Ο **Client** πραγματοποιεί μια απομακρυσμένη κλήση και καλείται είτε από ένα περιβάλλον επιστημονικού υπολογισμού, όπως για παράδειγμα το MATLAB, είτε από ένα πρόγραμμα γραμμένο σε κάποια παραδοσιακή γλώσσα προγραμματισμού.

Ο **Server** εκτελεί μεθόδους για λογαριασμό των clients. Το υλικό του Server μπορεί να ποικίλει από ένα σύστημα ενός επεξεργαστή έως ένα σύστημα πολλών επεξεργαστών, ενώ και οι μέθοδοι που εκτελούνται σε αυτόν διαφέρουν σε πολυπλοκότητα.

Ο **Agent** είναι το κεντρικό σημείο του Netsolve. Διαθέτει μία λίστα με όλους τους διαθέσιμους servers, εκτελεί δέσμευση πόρων στις αιτήσεις των πελατών, ενώ, παράλληλα, εξασφαλίζει το διαμοιρασμό εργασίας ανάμεσα στους servers.

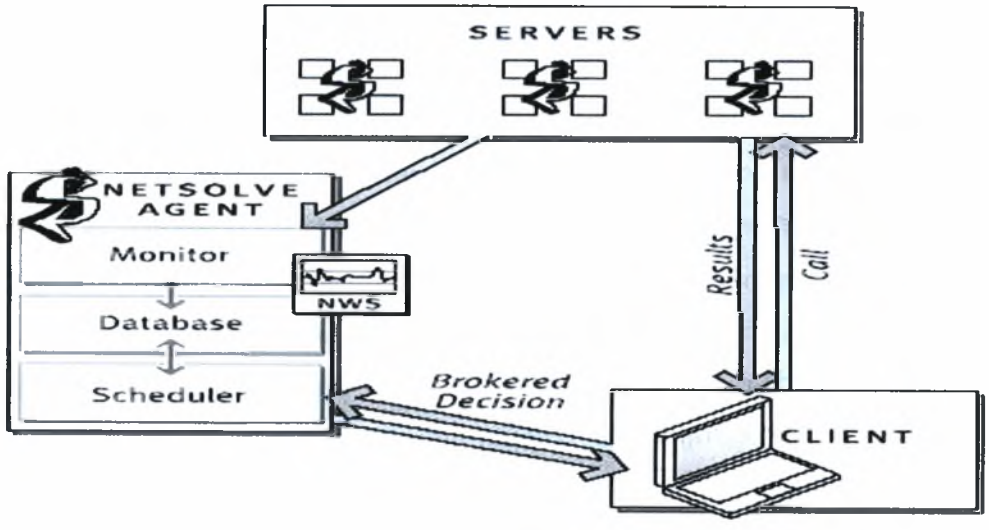
Ένα τυπικό σενάριο χρήσης του Netsolve είναι το εξής:

- Ο Client στέλνει μία αίτηση στον Agent για την αναζήτηση του κατάλληλου Server, όπου θα εκτελεστεί η επιθυμητή λειτουργία.

- Ο Agent επιστρέφει μία λίστα με τους διαθέσιμους Servers ταξινομημένους σε σειρά καταλληλότητας.

- Ο Client επιχειρεί να επικοινωνήσει με κάποιον Server από τη λίστα, ξεκινώντας από τον πρώτο σε καταλληλότητα. Στη συνέχεια, στέλνει τα δεδομένα εισόδου στον Server.

- Τελικά, ο Server εκτελεί τη λειτουργία για λογαριασμό του πελάτη και του επιστρέφει τα αποτελέσματα.

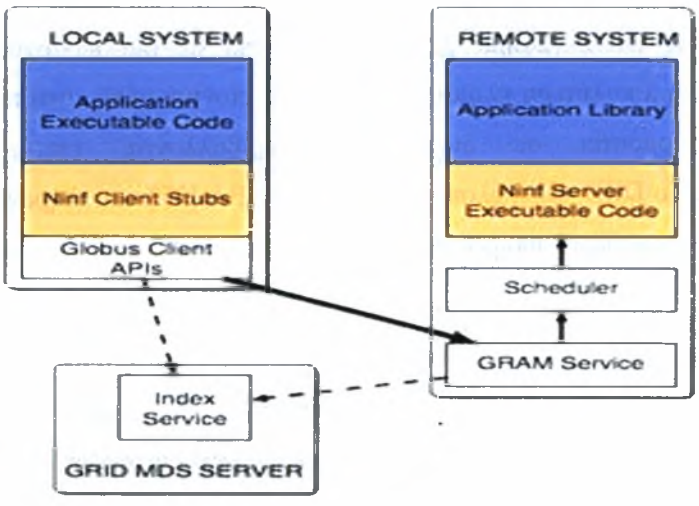


Σχήμα 2.3: Η αρχιτεκτονική του Netsolve/Gridsolve

Το **Ninf-G** [13], το οποίο προέρχεται από το National Institute of Advanced Industrial Science and Technology της Ιαπωνίας, είναι middleware για GridRPC [6]. Οι μέθοδοι του client API, που παρέχει το Ninf-G, μπορούν να κατηγοριοποιηθούν ως εξής:

- Μέθοδοι αρχικοποίησης και τερματισμού: οι μέθοδοι αρχικοποίησης και τερματισμού είναι παρόμοιες με αυτές του MPI (Message Passing Interface).
- Μέθοδοι χειρισμού function handles, οι οποίες επιτρέπουν τη δημιουργία και την καταστροφή function handles.
- Μέθοδοι κλήσεων RPC. Οι κλήσεις του GridRPC μπορούν να κατηγοριοποιηθούν με το συνδυασμό δύο ιδιοτήτων: αυτής της σύγχρονης συμπεριφοράς, καθώς και της σειράς κλήσης. Μια κλήση μπορεί να είναι είτε σύγχρονη είτε ασύγχρονη και να χρησιμοποιεί μία πληθώρα ορισμάτων ή μια στοίβα κλήσεων.
- Ασύγχρονοι μέθοδοι ελέγχου για GridRPC, οι οποίες απευθύνονται μόνο για ασύγχρονες κλήσεις.
- Ασύγχρονοι μέθοδοι αναμονής για GridRPC, οι οποίες επιτρέπουν στην εφαρμογή να προσαρμόσει, ανάλογα με τις απαιτήσεις της, την επιθυμητή συμπεριφορά της σε προηγούμενες κλήσεις που έχει κάνει για ασύγχρονο GridRPC.
- Μέθοδοι για αναφορά σφαλμάτων, οι οποίες επιστρέφουν κωδικούς και περιγραφές λαθών.
- Μέθοδοι για την κατασκευή μιας στοίβας ορισμάτων, οι οποίες επιτρέπουν στην εφαρμογή, κατά την εκτέλεσή της, να κατασκευάσει τη λίστα ορισμάτων για την κλήση μιας συνάρτησης.

Το Ninf-G παρέχει ένα Java client API και η υλοποίησή του έχει γίνει με τη χρήση του Java Commodity Grid Toolkit(JavaCoG kit) [31]. Στο σχήμα 2.4 παρουσιάζεται η αρχιτεκτονική λογισμικού.



Σχήμα 2.4: Η αρχιτεκτονική του Ninf-G

Το **DIET** project [5] είναι μία προσπάθεια από τα ερευνητικά κέντρα INRIA και INSA (2001). Στόχος του είναι η δημιουργία ενός middleware, το οποίο θα μπορεί να κλιμακώνεται, κατανέμοντας το πρόβλημα της δρομολόγησης των αιτήσεων για την εύρεση του κατάλληλου server ανάμεσα σε πολλαπλούς agents. Το σύστημα πραγματοποιεί τη δρομολόγηση, λαμβάνοντας υπόψη του την πληροφορία, που περιέχεται στην αίτηση του πελάτη (μέγεθος προβλήματος προς επίλυση, μέγεθος εμπλεκομένων δεδομένων), την απόδοση της στοχευόμενης πλατφόρμας (φόρτος server, διαθέσιμη μνήμη, απόδοση επικοινωνίας) και την τοπική διαθεσιμότητα των δεδομένων από προηγούμενους υπολογισμούς. Η δρομολόγηση αυτή κατανέμεται, χρησιμοποιώντας αρκετές συνεργαζόμενες ιεραρχίες συνδεδεμένες είτε στατικά είτε δυναμικά (σε peer-to-peer στυλ). Η διαχείριση των δεδομένων παρέχει τη δυνατότητα συγκράτησής τους από το σύστημα για μελλοντική χρήση.

Η σχεδίαση του DIET έχει γίνει με βάση τις εξής αρχές:

1) Κλιμάκωση: Καθώς ο αριθμός των αιτήσεων αυξάνει, ο agent γίνεται το σημείο συμφόρησης για ολόκληρο το σύστημα. Για την επίλυση αυτού του προβλήματος, είτε το μηχάνημα που φιλοξενεί τον agent πρέπει να είναι αρκετά ισχυρό, είτε να κατανεμηθεί η δρομολόγηση σε διάφορα μέρη.
2) Ευκολία βελτίωσης: Ο στόχος του DIET είναι να παρέχει ένα περιβάλλον λογισμικού, το οποίο θα μπορεί εύκολα να προσαρμοστεί στις ανάγκες του χρήστη. Τα API σε επίπεδο client, server και agent έχουν σχεδιαστεί με προσοχή, ώστε να χρησιμοποιηθούν είτε για την ενσωμάτωση μιας εφαρμογής στο grid είτε για την περαιτέρω βελτίωση του εργαλείου.
3) Ευκολία ανάπτυξης: Η ανάπτυξη ενός τέτοιου εργαλείου πρέπει να γίνει με τη χρήση υπαρχόντων middleware, τα οποία θα παρέχουν καλύτερη απόδοση σε μεγάλη κλίμακα (π.χ. CORBA, LDAP κ.τ.λ.).

Η αρχιτεκτονική του DIET (Σχήμα 2.5) έχει βασιστεί σε μία ιεραρχική προσέγγιση, προκειμένου να παρέχει καλύτερη κλιμάκωση. Η αρχιτεκτονική αυτή είναι πιο ευέλικτη και μπορεί να προσαρμοστεί σε διαφορετικά περιβάλλοντα, επιτυγχάνοντας και διαλειτουργικότητα. Το DIET είναι υλοποιημένο σε CORBA [35] και κληρονομεί από αυτήν τις προτυποποιημένες σταθερές υπηρεσίες, οι οποίες παρέχονται από διάφορες υλοποιήσεις της παραπάνω.

Το DIET αποτελείται από πολλά μέρη. Ο **client** είναι μια εφαρμογή, η οποία χρησιμοποιεί το σύστημα για να λύσει προβλήματα, χρησιμοποιώντας μία RPC προσέγγιση. Οι χρήστες μπορούν να έχουν πρόσβαση στο DIET, είτε μέσω πολλών διεπαφών, όπως web portals, PSE's, π.χ. το matlab, είτε από προγράμματα γραμμένα σε C ή C++. Ο SeD (**Server**

Daemon) παρέχει τη διεπαφή για τους servers υπολογισμού και μπορεί να προσφέρει οποιοδήποτε αριθμό υπηρεσιών υπολογισμού ανάλογα με την εφαρμογή. Ο SeD μπορεί να λειτουργήσει ως διεπαφή και ως μηχανισμός εκτέλεσης, είτε για ένα standalone μηχάνημα, είτε για ένα cluster, παρέχοντας υπηρεσίες παράδοσης εργασιών. Οι **agents** παρέχουν υψηλότερου επιπέδου υπηρεσίες, όπως για παράδειγμα δρομολόγησης και διαχείρισης δεδομένων. Αυτές οι υπηρεσίες κλιμακώνουν για το λόγο του ότι κατανέμονται ανάμεσα σε μια ιεραρχία από agents, οι οποίοι αποτελούνται από έναν master agent (MA), αρκετούς agents (A) και local agents(LA).

Το DIET παρέχει μία Peer-to-Peer επέκταση, στην οποία:

Ο client είναι ένας JXTA [36] peer. Όταν ψάχνει για μία δεδομένη υπηρεσία ανακαλύπτει έναν από τους MA(Master Agents) από τη JXTA διαφήμισή του, επιλέγει έναν από αυτούς, στέλνει μία αίτηση (ενθυλακωμένη σε ένα JXTA μήνυμα) γι' αυτήν την υπηρεσία και περιμένει για την απάντηση. Όταν έρθει η απάντηση, εξάγει τη λίστα με τους διαθέσιμους SeD's και στέλνει στον κατάλληλο SeD τον απομακρυσμένο υπολογισμό.

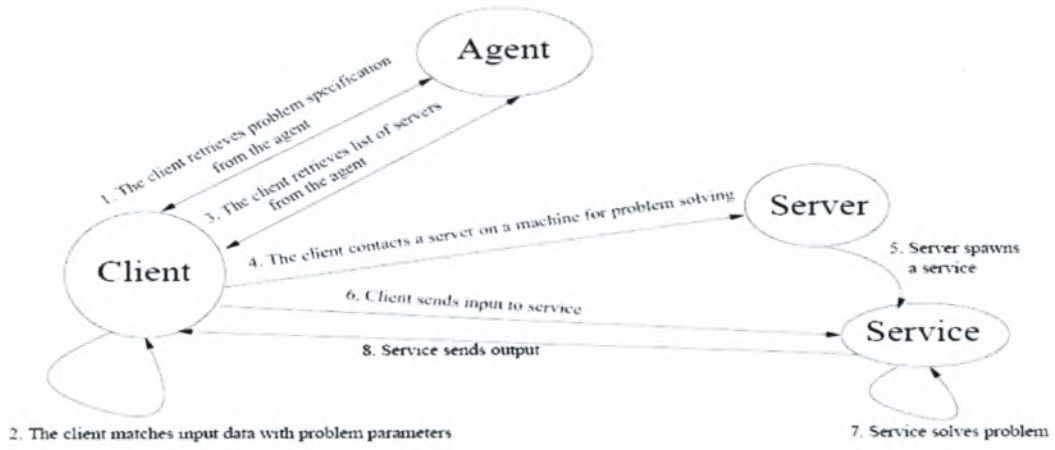
Ο SeD είναι κι αυτός με τη σειρά του ένας JXTA peer, που επιτρέπει στον client να στείλει αιτήσεις υπολογισμού, εκτελεί τον υπολογισμό και επιστρέφει τα αποτελέσματα (η επικοινωνία αυτή γίνεται JXTA μηνύματα).

Ο multi-MA αποτελείται από πολλούς MA (Master Agent), οι οποίοι τρέχουν σε μία δεδομένη στιγμή στο δίκτυο και είναι προσβάσιμοι από τον αρχικό MA. Οι MA συνδέονται δυναμικά μεταξύ τους, αφού αποτελούν JXTA peers.

Σχήμα 2.5: Η αρχιτεκτονική του DIET

Το **Gradsolve** [37], από το Πανεπιστήμιο του Tennessee (2003), είναι ένα σύστημα, το οποίο επιτρέπει την απομακρυσμένη κλήση MPI εφαρμογών πάνω από μία δυναμικά μεταβαλλόμενη Grid υποδομή.

Αναλυτικότερα, στον πυρήνα του GradSolve (Σχήμα 2.6) βρίσκεται μία XML βάση δεδομένων, η οποία αποτελείται από 4 tables για την αποθήκευση πληροφοριών: users, resources, applications και problems. Η επικοινωνία με τη βάση δεδομένων γίνεται με τη χρήση του XML-RPC [38] πρωτοκόλλου.



Σχήμα 2.6: Η αρχιτεκτονική του Gradsolve

Το Gradsolve περιλαμβάνει 3 ανθρώπινες οντότητες:

- Ο **Administrator** είναι υπεύθυνος για τη διαχείριση των χρηστών και των πόρων του συστήματος. Αρχικοποιεί την XML βάση δεδομένων, δημιουργώντας εγγραφές για τους διαφορετικούς χρήστες (users), καθώς και το resource table, εισάγοντας πληροφορίες για τους πόρους του συστήματος.
- Ο **Library Writer** κάνει upload στο GradSolve την εφαρμογή του. Εκτός από την εφαρμογή περιλαμβάνεται και ένα IDL (Interface Description Language) αρχείο το οποίο περιέχει λεπτομέρειες για την κλήση της εφαρμογής. Η IDL γλώσσα είναι ειδικά σχεδιασμένη για το Gradsolve. Το σύστημα αρχικά μετατρέπει το IDL αρχείο σε XML και δημιουργεί ένα πρόγραμμα για την απομακρυσμένη κλήση της εφαρμογής. Το πρόγραμμα αυτό είναι υπεύθυνο για την αρχικοποίηση του παράλληλου περιβάλλοντος, την εισαγωγή δεδομένων από αρχεία εισόδου, την κλήση της παράλληλης ρουτίνας και τέλος την αποθήκευση των δεδομένων σε αρχεία εξόδου. Έπειτα, πραγματοποιείται η μεταγλώττιση του κώδικα της εφαρμογής, κάνοντας τις απαιτούμενες συνδέσεις με βιβλιοθήκες, όπως αυτές περιγράφονται στο IDL αρχείο, μεταφέρει τα εκτελέσιμα σε διαφορετικά resources στο Grid και αποθηκεύονται στην XML βάση δεδομένων οι θέσεις των εκτελέσιμων. Δίνεται η δυνατότητα στον library writer να δώσει πληροφορίες σχετικά με την

ελάχιστη απαίτηση πόρων, τον εκτιμούμενο χρόνο εκτέλεσης και περιγραφής της κατανομής των δεδομένων, που χρησιμοποιούνται από την εφαρμογή.

- Οι **End Users** μπορούν να λύσουν προβλήματα, χρησιμοποιώντας το Gradsolve, γράφοντας είτε ένα C είτε ένα Fortran πρόγραμμα. Η διαδικασία αυτή περιλαμβάνει την κλήση της ρουτίνας gradsolve(), στην οποία περνιούνται ως ορίσματα το όνομα της εφαρμογής, καθώς και οι παράμετροι που χρειάζονται από αυτήν.

Η κλήση της gradsolve() κινητοποιεί τον **Gradsolve Application Manager** (GAM). Ο GAM αρχικά ελέγχει αν ο χρήστης κατέχει πρόσβαση για την εκτέλεση εργασιών. Στη συνέχεια, προχωρά στη δέσμευση πόρων για την εργασία, χρησιμοποιώντας το table resources από την XML βάση δεδομένων και τελικά προχωρά στη δρομολόγηση.

Το **GridPsi** [2] είναι μία πλατφόρμα για επίλυση προβλημάτων ΜΔΕ. Το GridPsi παράγει αυτόματα κώδικα για matlab για την εύκολη ενσωμάτωσή του για Grid computing.

Αναλυτικότερα, στην πρώτη φάση η πλατφόρμα παρέχει στο χρήστη μία γραφική διεπαφή για τον ορισμό του προβλήματος μερικών διαφορικών εξισώσεων που έχει να επιλύσει. Στη δεύτερη φάση, παράγει αυτόματα κώδικα matlab για την επίλυση των 3D μοντέλων διαφορικών εξισώσεων για το πρόβλημα που έχει περιγράψει ο χρήστης. Τέλος, στην τρίτη φάση παρέχει ενσωμάτωση του matlab κώδικα με grid middleware (συγκεκριμένα του UNICORE [39]), το οποίο επιτρέπει την εκτέλεση στην grid υποδομή.

2.3 Εργαλεία για τη δημιουργία επιστημονικών ροών εργασίας

Ο όρος ροή εργασίας (workflow) είναι μία αφηρημένη έννοια, η οποία χρησιμοποιείται σε διάφορα επιστημονικά πεδία, προκειμένου να περιγράψει την απεικόνιση μία σειράς ατομικών ενεργειών για την περιγραφή μίας σύνθετης εργασίας. Στον τομέα των ηλεκτρονικών υπολογιστών, η έννοια της ροής εργασίας χρησιμοποιείται για την περιγραφή πολύπλοκης επεξεργασίας δεδομένων.

Με την ανάπτυξη των Web Services και των Grid Services [7], πρόεκυψε η ανάγκη ανάπτυξης γλωσσών, οι οποίες θα δίνουν τη δυνατότητα ελέγχου πάνω από το επίπεδο των υπηρεσιών. Οι γλώσσες για τη σύνθεση ροών εργασίας δίνουν τη δυνατότητα "ενορχήστρωσης" υπηρεσιών, προσφέροντας τη δυνατότητα περιγραφής διακριτών διεργασιών, καθώς και της αναπαράστασης των αλληλεπιδράσεων μεταξύ τους. Εργαλεία, που έχουν αναπτυχθεί για τη σύνθεση ροών εργασίας, είναι το Triana και το Taverna.

Το **Triana** [14] είναι ένα γραφικό PSE (Problem Solving Environment), το οποίο επιτρέπει στο χρήστη τη σύνθεση επιστημονικών ροών. Το Triana, αν και σχεδιάστηκε αρχικά για ανάλυση δεδομένων, λόγω της πληθώρας εργαλείων που προσφέρει μπορεί να χρησιμοποιηθεί για πολλούς σκοπούς. Επίσης, μπορεί να χρησιμοποιηθεί ως περιβάλλον για grid computing και δυναμικά να ανακαλύψει και να «ενορχηστρώσει» κατανεμημένες πηγές, όπως για παράδειγμα Web Services. Χρησιμοποιεί μια XML-based γλώσσα για την αναπαράσταση τόσο τον ορισμών όσο και των workflows.

Το Triana, για τη διαφήμιση, ανακάλυψη και επικοινωνία των υπηρεσιών, χρησιμοποιεί το GAP, μία προγραμματιστική διεπαφή υψηλού επιπέδου. Οι ροές εργασίας στο Triana αποτελούνται από δίκτυα Triana-GAP υπηρεσιών, οι οποίες μπορούν να διαφημιστούν, να ανακαλυφθούν και να επικοινωνήσουν, χρησιμοποιώντας αφηρημένες, υψηλού επιπέδου, κλήσεις, οι οποίες είναι ανεξάρτητες από τους χαμηλότερους μηχανισμούς, που χρησιμοποιούνται για την κατανομή εργασίας τόσο σε grid όσο και σε P2P δίκτυα.

Το **Taverna** [19] είναι ένα εργαλείο υλοποιημένο σε Java, το οποίο περιλαμβάνει ένα γραφικό περιβάλλον, που παρέχει στο χρήστη μία γραφική διεπαφή για τη σύνθεση και την εκτέλεση ροών εργασίας (workflows) Βιοπληροφορικής. Στο Taverna, κάθε ροή εργασίας αποτελείται από ένα γράφο από processors, όπου ο καθένας από αυτούς μετατρέπει ένα σετ δεδομένων εισόδου σε ένα σετ δεδομένων εξόδου. Αυτές οι ροές εργασίας είναι γραμμένες σε μία γλώσσα, η οποία ονομάζεται Sculf. Η Sculf είναι μία γλώσσα υψηλού επιπέδου βασισμένη στην XML, στην οποία κάθε βήμα επεξεργασίας της ροής εργασίας αποτελείται από μία ατομική εργασία. Μία ροή εργασίας στη γλώσσα Sculf αποτελείται από:

Processors: Ο processor είναι μία οντότητα, η οποία μετατρέπει ένα σετ δεδομένων εισόδου σε ένα σετ δεδομένων εξόδου. Ένας από τους κύριους διαθέσιμους processors είναι ο Arbitrary WSDL type, ο οποίος επιτρέπει τη λειτουργία κλήσης μίας Web Service.

Data links: Τα data links μεσολαβούν ανάμεσα στη ροή των δεδομένων σε μία πηγή και σε μία καταβόθρα. Η πηγή δεδομένων, όπως και η καταβόθρα, μπορεί να είναι η έξοδος από κάποιο processor ή η είσοδος σε μία ροή εργασίας.

Coordination Constraints: Η δομή αυτή ενώνει δύο processors και ελέγχει τη ροή εκτέλεσής τους. Αυτό το επίπεδο ελέγχου απαιτείται όταν υπάρχει μία διεργασία, στην οποία τα επιμέρους στάδιά της είναι αναγκαίο να εκτελεστούν με συγκεκριμένη σειρά, ενώ δεν υπάρχει απευθείας εξάρτηση δεδομένων μεταξύ τους.

3 Θεωρητικό και τεχνολογικό υπόβαθρο

Στο ακόλουθο κεφάλαιο αναφέρουμε συνοπτικά τις τεχνολογίες που χρησιμοποιήσαμε.

3.1 Web Services

Ως Web Service μπορούμε να ορίσουμε την εξαγωγή της προγραμματιστικής λογικής μίας εφαρμογής ως μιας υπηρεσίας προσβάσιμης μέσω στάνταρντ πρωτοκόλλων του διαδικτύου. Στόχος των Web Services είναι να δώσουν τη δυνατότητα δημιουργίας ενός κατανεμημένου περιβάλλοντος, όπου οποιοσδήποτε αριθμός εφαρμογών θα μπορεί να επικοινωνεί και να συνεργάζεται, μεταξύ και ανάμεσα, σε οργανισμούς, ανεξαρτήτως πλατφόρμας και γλώσσας. Στο σχήμα 3.1 παρουσιάζονται συνοπτικά οι τεχνολογίες, από τις οποίες αποτελούνται τα Web Services, καθώς και ο τρόπος αλληλεπίδρασής τους.

Ένα Web Service είναι ένα κομμάτι επιχειρηματικής λογικής (business logic), το οποίο εντοπίζεται κάπου στο Διαδίκτυο και είναι προσβάσιμο μέσω στάνταρντ πρωτοκόλλων του Διαδικτύου, όπως το HTTP ή το SMTP. Πολλές web τεχνολογίες (J2EE, CORBA) θα μπορούσαν να κατηγοριοποιηθούν ως Web Service τεχνολογίες, αλλά δεν είναι. Η μεγάλη διαφορά αυτών των τεχνολογιών με τα Web Services είναι ότι βασίζονται σε XML standards (εν αντιθέσει με ένα αντίστοιχο δυαδικό πρωτόκολλο επικοινωνίας), το οποίο υποστηρίζεται παγκοσμίως. Η XML παρέχει έναν, ανεξάρτητο πλατφόρμας, τρόπο αναπαράστασης των δεδομένων και η παγκόσμια υποστήριξή της διασφαλίζει ότι στο μέλλον κάθε νέα μεγάλη τεχνολογία λογισμικού θα ακολουθεί μία Web Service προσέγγιση.

Η τεχνολογία των Web Services έχει τα εξής χαρακτηριστικά:

1) XML based: Με τη χρήση της XML για την αναπαράσταση των δεδομένων, οι τεχνολογίες αυτές μπορούν να υποστηρίξουν τη διαλειτουργικότητα στον πυρήνα τους.

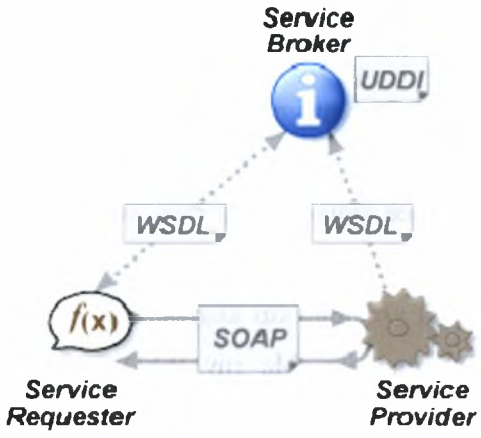
2) Loosely coupled: Ο καταναλωτής ενός Web Service δεν είναι "δεμένος" με αυτό το service απευθείας. Η διεπαφή του service μπορεί να αλλάξει με το χρόνο, χωρίς να στερεί τη δυνατότητα από τις εφαρμογές-πελάτες να αλληλεπιδρούν με αυτήν την υπηρεσία. Ένα ισχυρά συνδεδεμένο σύστημα, δομημένο στο μοντέλο client-server, συνεπάγεται ότι οι λειτουργίες του client και του server είναι ισχυρά δεμένες μεταξύ τους, υπονοώντας ότι τυχόν αλλαγές σε κάποια από τις διεπαφές συνεπάγεται αλλαγές και στην άλλη. Η υιοθέτηση ενός μοντέλου χαλαρής διασύνδεσης τείνει να μετατρέψει τη διαχείριση συστημάτων λογισμικού πιο εύκολη και επιτρέπει την απλούστερη ενσωμάτωση διαφορετικών συστημάτων.

3) Coarse-grained : Οι αντικειμενοστραφείς τεχνολογίες, όπως η Java, εκθέτουν τις μεθόδους τους μέσω ατομικών μεθόδων. Μία ατομική μέθοδος είναι μία καλώς ορισμένη λειτουργία, η οποία παρέχει μία σημαντική ικανότητα σε επίπεδο συνεργασίας. Για παράδειγμα, το χτίσιμο ενός Java προγράμματος από την αρχή απαιτεί τη δημιουργία πολλών ατομικών μεθόδων, οι οποίες, στη συνέχεια, συντίθενται σε μια σύνθετη υπηρεσία, προκειμένου να καταναλωθεί τελικώς είτε από μια εφαρμογή-πελάτη είτε από μία άλλη υπηρεσία. Η τεχνολογία των Web Services παρέχει ένα φυσικό τρόπο ορισμού coarse-gained υπηρεσιών.

4) Δυνατότητα για σύγχρονες και ασύγχρονες υπηρεσίες: Στις σύγχρονες κλήσεις, ο client μπλοκάρει και περιμένει για τη συνέχιση της εκτέλεσής του, έως ότου ολοκληρωθεί η κλήση της υπηρεσίας. Οι ασύγχρονες κλήσεις επιτρέπουν στον client, μετά την κλήση μίας υπηρεσίας, να εκτελέσει και άλλες λειτουργίες πριν την ολοκλήρωσή της. Στις σύγχρονες κλήσεις οι εφαρμογές-πελάτες παίρνουν τα αποτελέσματά τους αμέσως μετά την ολοκλήρωση της υπηρεσίας, ενώ στις ασύγχρονες κάποια χρονική στιγμή αργότερα. Η υποστήριξη για ασύγχρονες κλήσεις είναι κομβικός παράγοντας στη δημιουργία χαλαρά συνδεδεμένων συστημάτων.

5) Υποστήριξη για RPC: Η τεχνολογία των Web Services επιτρέπει σε εφαρμογές-πελάτες να πραγματοποιήσουν κλήσεις σε διαδικασίες, μεθόδους ή λειτουργίες σε απομακρυσμένα αντικείμενα, χρησιμοποιώντας ένα πρωτόκολλο βασισμένο στην XML.

6) Υποστήριξη για ανταλλαγή εγγράφων: Ένα από τα κύρια πλεονεκτήματα της XML είναι ο γενικός τρόπος αναπαράστασης που παρέχει όχι μόνο για δεδομένα, αλλά και για πολυσύνθετα έγγραφα. Αυτά τα έγγραφα μπορεί να είναι τόσο απλά, όσο η αναπαράσταση μιας διεύθυνσης ή τόσο πολυσύνθετα, όσο ένα ολόκληρο βιβλίο.



Σχήμα 3.1: Η αρχιτεκτονική τριών επιπέδων των Web Services

3.1.1 SOAP

Το SOAP (Simple Object Access Protocol) είναι ένα πρωτόκολλο επικοινωνίας, το οποίο καθορίζει τη μορφή των μηνυμάτων για την ανταλλαγή XML εγγράφων πάνω από μία ποικιλία πρωτοκόλλων μεταφοράς, όπως το HTTP, SMTP και FTP. Το SOAP παρέχει έναν πολύ απλό τρόπο για την πραγματοποίηση του RPC: ανταλλαγή XML εγγράφων.

Αποτελεί τον ακρογωνιαίο λίθο μίας Service Oriented Architecture. Ο σχεδιασμός του έγινε με βασικό στόχο να παρέχει στις εφαρμογές-πελάτες τη δυνατότητα να συνδέονται και να καλούν εύκολα απομακρυσμένες μεθόδους, με βασικό πρωτόκολλο μεταφοράς το HTTP, παρότι όπως αναφέραμε και παραπάνω μπορούν να χρησιμοποιηθούν και άλλα πρωτόκολλα για τη μεταφορά SOAP μηνυμάτων. Προγενέστερες τεχνολογίες του SOAP, όπως για παράδειγμα το DCOM [40], η CORBA [35] και το Java RMI [41], σχεδιάστηκαν με τους ίδιους στόχους, δηλαδή την εύκολη κλήση απομακρυσμένων μεθόδων και κατά συνέπεια την απλούστερη κατασκευή κατανεμημένων συστημάτων. Η κύρια διαφορά του SOAP είναι ότι, λόγω της XML αναπαράστασης των δεδομένων που χρησιμοποιεί, επιτυγχάνει διαλειτουργικότητα πλατφόρμας και γλώσσας. Έτσι, για παράδειγμα, μία εφαρμογή-πελάτης γραμμένη σε Perl, η οποία «τρέχει» σε ένα Mac PC, μπορεί να καλέσει μία απομακρυσμένη μέθοδο, η οποία «τρέχει» στο Microsoft SOAP server σε ένα μηχάνημα αρχιτεκτονικής IX86. Ένα ακόμα βασικό πλεονέκτημα του SOAP είναι η επεκτασιμότητά του. Η περιγραφή της δομής ενός SOAP μηνύματος γίνεται με τη χρήση ενός XML schema, το οποίο τροποποιείται ανάλογα με τις ανάγκες κάθε εφαρμογής.

Στην τεχνική κριτική του SOAP, κύριο μειονέκτημά του, σε σχέση με παρεμφερείς τεχνολογίες, είναι ότι, λόγω της χρήσης της XML, το μέγεθος των μηνυμάτων είναι μεγαλύτερο. Αυτό είναι το trade-off του SOAP, προκειμένου να επιτυγχάνει διαλειτουργικότητα. Παρόλα αυτά, για περιβάλλοντα που αλλάζουν συχνά και δυναμικά, το SOAP αποτελεί την τεχνολογική λύση για την εύκολη διαχείρισή τους.

3.1.2 Web Services Description Language (WSDL)

Η WSDL είναι μία τεχνολογία βασισμένη στην XML, η οποία μαζί με το SOAP αποτελούν τα κύρια συστατικά μίας Service Oriented Architecture. Συνοπτικά, η WSDL είναι η συμφωνία ανάμεσα στον service requestor και στον service provider για τον τρόπο κλήσης της υπηρεσίας, όπως για παράδειγμα μία διεπαφή σε Java. Κύριο χαρακτηριστικό της WSDL είναι ότι παρέχει ανεξαρτησία γλώσσας και πλατφόρμας και επίσης χρησιμοποιείται κυρίως για να περιγράφει SOAP υπηρεσίες. Με την WSDL μία εφαρμογή-πελάτης μπορεί να ανακαλύψει μία υπηρεσία και με τη χρήση εργαλείων να παράγει αυτόματα κώδικα για την κλήση της υπηρεσίας, επιτρέποντας έτσι την ευκολότερη ενσωμάτωση νέων εφαρμογών.

Τα βασικά στοιχεία που περιέχονται σε ένα WSDL έγγραφο είναι:

- Περιγραφή της διεπαφής με όλες τις διαθέσιμες υπηρεσίες.
- Πληροφορία για τους τύπους δεδομένων των εισερχόμενων και εξερχόμενων μηνυμάτων.
- Πληροφορία για το πρωτόκολλο μεταφοράς που θα χρησιμοποιηθεί.
- Διεύθυνση για τον εντοπισμό της υπηρεσίας

3.1.3 Universal Description, Discovery and Integration (UDDI)

Το UDDI είναι ένα ευρετήριο ανεξάρτητο πλατφόρμας, βασισμένο στην XML, για τη διαφήμιση και εύρεση υπηρεσιών. Αποτελεί μία ανοιχτή πρόταση της βιομηχανίας, η οποία επιτρέπει στις εταιρίες τη δημοσίευση των υπηρεσιών τους, την ανακάλυψη νέων υπηρεσιών, ενώ ορίζει και τον τρόπο με τον οποίο οι εφαρμογές λογισμικού αλληλεπιδρούν πάνω από το Διαδίκτυο. Η καταχώρηση μιας εταιρίας στο UDDI αποτελείται από τα εξής τρία μέρη:

- **White Pages**: Περιέχουν πληροφορίες επικοινωνίας με την εταιρία, όπως διευθύνσεις κ.τ.λ.
- **Yellow Pages**: Κατηγοριοποίηση της εταιρίας με βάση στάνταρντ ταξονομίες.
- **Green Pages**: Τεχνικές πληροφορίες σχετικά με τις παραχωρούμενες πληροφορίες κάθε εταιρίας.

Το UDDI έχει σχεδιαστεί έτσι ώστε να ακολουθεί τις τεχνολογίες των Web Services. Η αλληλεπίδραση εφαρμογών με αυτό γίνεται μέσω SOAP μηνυμάτων για την πρόσβαση σε WSDL έγγραφα, τα οποία απαιτούνται για την αλληλεπίδραση με τις υπηρεσίες τις οποίες έχει καταχωρημένες στο ευρετήριό του.

3.2 Web Services Orchestration - BPEL

Η BPEL (Business Process Execution Language) είναι μία XML - based γλώσσα για την «ενορχήστρωση» Web Services. Ο σχεδιασμός της BPEL έγινε με γνώμονα την ανάγκη ορισμού μίας διαφορετικής γλώσσας για την περιγραφή ροών εργασίας.

Οι ροές εργασίας στην BPEL περιγράφουν τη λογική μίας μακροπρόθεσμης διαδικασίας, η οποία αποτελείται από επιμέρους modules για τις βραχυπρόθεσμες εργασίες. Η διεπαφή για την κλήση των επιμέρους modules είναι αυτή των Web Services, δηλαδή για την επικοινωνία της διαδικασίας με ένα module χρησιμοποιείται η ανταλλαγή SOAP μηνυμάτων. Η BPEL περιέχει δομές ελέγχου για την περιγραφή της λογικής της διαδικασίας, όπως επίσης και τη δυνατότητα ορισμού data types. Γι' αυτό αναφέρεται ως «γλώσσα ενορχήστρωσης», λόγω της δυνατότητας του κεντρικοποιημένου ελέγχου που προσφέρει κατά τον ορισμό και την εκτέλεση της διαδικασίας.

Το XML αρχείο με την περιγραφή της ροής της εργασίας δίνεται ως είσοδος στην BPEL engine, η οποία είναι υπεύθυνη για την εκτέλεσή του.

3.3 Η βιβλιοθήκη LAPACK

Η LAPACK (Linear Algebra Package), για αριθμητική ανάλυση γραμμένη σε fortran77, παρέχει ρουτίνες για την επίλυση γραμμικών εξισώσεων, προβλημάτων ελαχίστων τετραγώνων, προβλημάτων ιδιοτιμών κ.τ.λ.

Η LAPACK μπορεί να θεωρηθεί ως απόγονος της LINPACK, η οποία είχε σχεδιαστεί για να εκτελείται σε υπολογιστικά συστήματα κοινής μνήμης. Η LAPACK βασίζεται στη βιβλιοθήκη BLAS (Basic Linear Algebra Subprograms). Η BLAS περιέχει, με τη σειρά της, ρουτίνες για βασικές πράξεις γραμμικής άλγεβρας, όπως πολλαπλασιασμό διανυσμάτων, πολλαπλασιασμό αριθμού με διάνυσμα κ.τ.λ. και διαιρείται σε 3 επίπεδα: την BLAS επιπέδου 1, επιπέδου 2 και επιπέδου 3. Η κατάλληλη ρύθμιση της βιβλιοθήκης BLAS είναι κύριος παράγοντας για την αποδοτική λειτουργία της LAPACK.

3.4 Web Service Technologies

Στον τομέα των Web Services τεχνολογιών υπάρχει διαθέσιμη μια πληθώρα εργαλείων, τα οποία μελετήσαμε για τις ανάγκες της εργασίας.

Ως Web Services Resource Framework (WSRF) ορίζουμε το σύστημα, το οποίο είναι υπεύθυνο για τη φιλοξενία και τον κύκλο ζωής μίας Web Service. Υπάρχουν διαθέσιμα αρκετά WSRF, που έχουν κατασκευαστεί από μεγάλες εταιρίες στο χώρο της Πληροφορικής,

τα οποία όμως δεν είναι δωρεάν. Μία άδειά τους απαιτεί πολλές χιλιάδες ευρώ, ενώ το setup και το administration πρέπει να γίνουν από εξειδικευμένο προσωπικό. Εργαλεία που ανήκουν σε αυτήν την κατηγορία, και τα οποία μελετήσαμε, είναι ο Websphere application server από την IBM, ο Weblogic application server από την BEA, καθώς και κάποιες άλλες πλατφόρμες από την Sun. Τα παραπάνω προϊόντα παρέχουν ολοκληρωμένες λύσεις για μία Service Oriented Architecture, καθώς παρέχουν web server, WSRF, UDDI registries κ.λπ.

Από τα frameworks, που είναι διαθέσιμα δωρεάν και τα οποία μελετήσαμε, ένα είναι το application server Jboss, το οποίο παρέχει υποστήριξη για Web Services και παρέχεται κάτω από την GNU license, καθώς και το Glassfish application server, το οποία παρέχεται κάτω από την CDDL (Common Distribution and Development License) και την GPL (General Public License). Από τα παραπάνω εργαλεία επιλέξαμε τον glassfish application server. Οι λόγοι που μας οδήγησαν σε αυτήν την επιλογή είναι ότι ο glassfish παρέχει πολύ καλή υποστήριξη μέσα από την ενεργή κοινότητά του. Η εγκατάστασή του είναι απλή, ενώ η φόρτωση εφαρμογών και η διαχείριση του application server γίνονται μέσα από web based γραφικά περιβάλλοντα (δίνεται ακόμα και η δυνατότητα διαχείρισης μιας cluster διάταξης). Η metro stack, που περιέχει, παρέχει τα τελευταία API's για Web Services, ενώ παρέχει plug-ins για περιβάλλοντα ανάπτυξης, όπως το eclipse και το netbeans.

Εκτός από τα WSRF, μελετήσαμε και τα δωρεάν UDDI registries και πιο συγκεκριμένα το jUDDI, το οποίο είναι open-source υλοποιημένο σε Java, καθώς και το freeEbxml registry.

4 Σχεδίαση Συστήματος

Στο κεφάλαιο περιγράφεται αναλυτικά η αρχιτεκτονική του συστήματος και τα βήματα που ακολουθήθηκαν για την έκθεση μεθόδων της LAPACK (βιβλιοθήκη αριθμητικής ανάλυσης, η οποία είναι υλοποιημένη σε fortran 77) ως Web Services και η σύνθεση scientific flows με τη χρήση της γλώσσας BPEL.

4.1 Έκθεση μεθόδων της LAPACK ως Web Services

Η LAPACK είναι βιβλιοθήκη αριθμητικής ανάλυσης, υλοποιημένη σε fortran77, η οποία περιέχει μεθόδους για την επίλυση γραμμικών συστημάτων εξισώσεων, προβλημάτων ελαχίστων τετραγώνων, προβλημάτων ιδιοτιμών κ.λπ. Η LAPACK αποτελεί ένα ευρέως χρησιμοποιούμενο εργαλείο, όπου έχει συγκεντρωθεί η γνώση και η εμπειρία της επιστημονικής κοινότητας από τα πολλά έτη παρουσίας της (η πρώτη έκδοσή της παρουσιάστηκε το 1995). Εκτός από την κλασική (Fortran) έκδοση, υπάρχουν εκδόσεις της LAPACK σε ορισμένες άλλες γλώσσες προγραμματισμού, όπως η C και η Java, οι οποίες όμως δεν έχουν προκύψει από επανασχεδιασμό της βιβλιοθήκης γι' αυτές τις γλώσσες αλλά είναι αποτέλεσμα εργαλείων μεταγλώττισης κώδικα fortran77 σε αυτές.

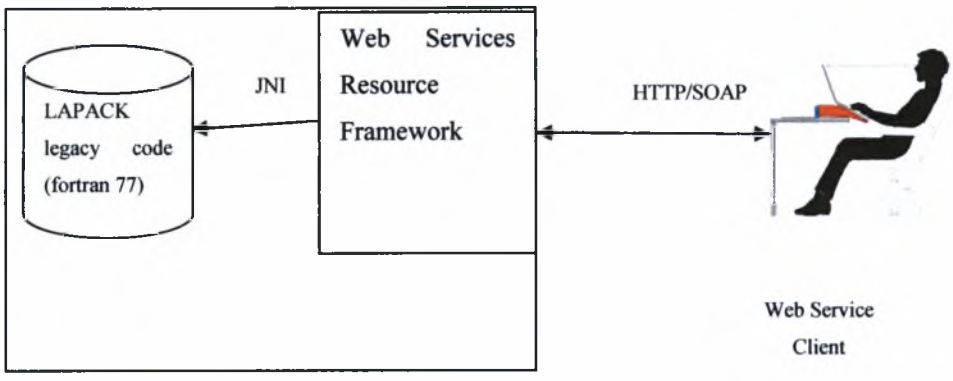
Βασικός στόχος ήταν η χρήση της fortran77 έκδοσης της LAPACK, ώστε να αποδειχθεί ότι είναι εφικτή η εξαγωγή της προγραμματιστικής λογικής legacy code ως Web Services με τη χρήση των υπαρχόντων εργαλείων. Εκτός αυτού, η συγκεκριμένη έκδοση της LAPACK είναι πιο αποδοτική και ακριβής από την Java (γλώσσα στην οποία είναι υλοποιημένο το WSRF [21]) έκδοσή της.

Το σύστημά μας (Σχήμα 4.1) αποτελείται από τα εξής δύο βασικά μέρη:

Web Service Client: Οποιαδήποτε οντότητα, η οποία αιτείται τη χρήση μίας υπηρεσίας, μέσα από τη χρήση standard πρωτοκόλλων του Διαδικτύου, όπως το http και το SOAP. Ο Web Service Client, εκτός των μεμονωμένων κλήσεων σε μία υπηρεσία, μπορεί να είναι ένα scientific flow, το οποίο αιτείται σειριακές ή/και παράλληλες κλήσεις σε διαφορετικές

υπηρεσίες των οποίων τα αποτελέσματα συνδυάζει για την επίλυση προβλημάτων επιστημονικού υπολογισμού.

Web Services Resource Framework (WSRF): Ο glassfish application server αποτελεί το WSRF (Web Services Resource Framework) του συστήματός μας. Είναι το κύριο κομμάτι, αφού είναι υπεύθυνο για τη φιλοξενία και τον κύκλο ζωής των υπηρεσιών μας και καθορίζει τη διαθεσιμότητά τους, καθώς και το χρόνο απόκρισης του συστήματος πέραν αυτού που απαιτείται για την πραγματοποίηση των υπολογισμών.



Σχήμα 4.1: Περιγραφή του συστήματος για την εξαγωγή legacy code ως Web Services

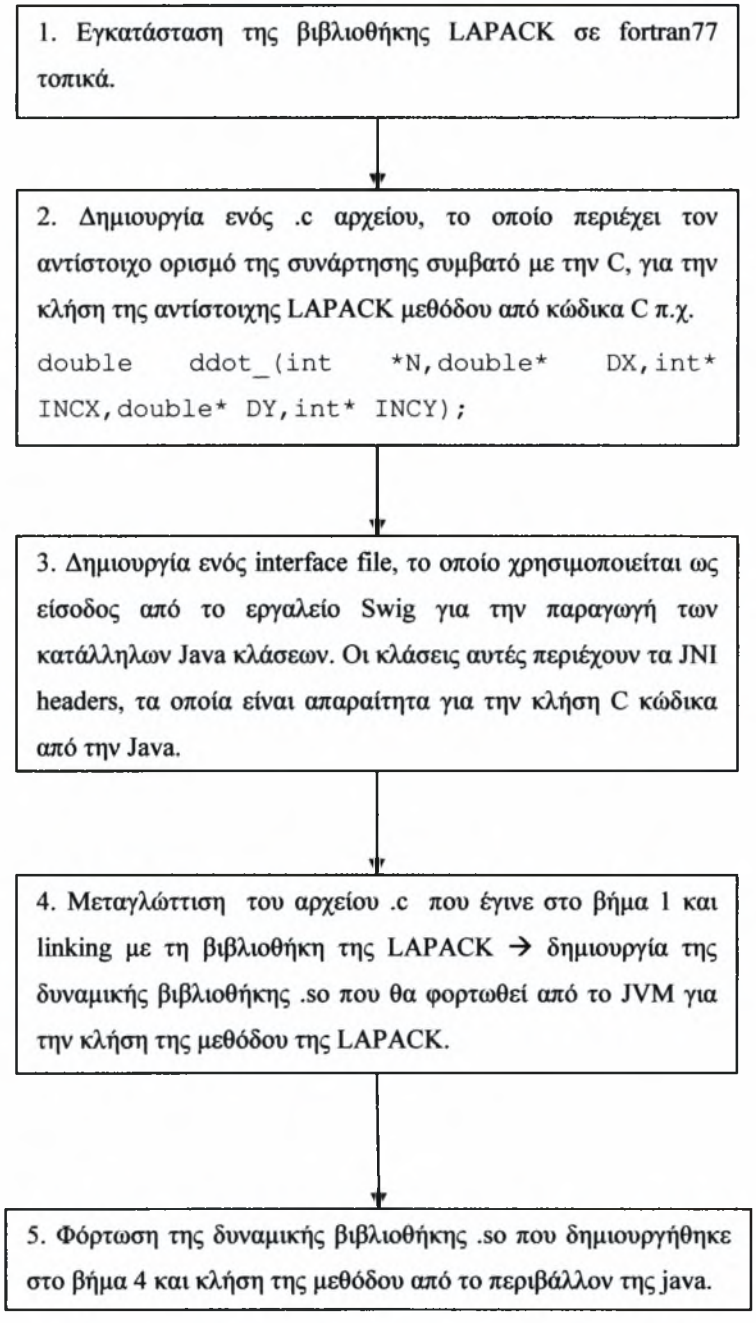
4.2 Ενσωμάτωση της LAPACK στο WSRF

Το WSRF, το οποίο χρησιμοποιήσαμε (glassfish application server), είναι υλοποιημένο σε Java, οπότε το αρχικό πρόβλημα ήταν η ενσωμάτωση ρουτινών της LAPACK γραμμένων σε fortran77 σε αυτό. Για το λόγο αυτό έγινε χρήση της τεχνολογίας του Java Native Interface (JNI) και του εργαλείου Swig.

Το JNI είναι η τεχνολογία που επιτρέπει στο Java Virtual Machine (JVM) να καλέσει ρουτίνες/βιβλιοθήκες υλοποιημένες σε διαφορετικές γλώσσες προγραμματισμού, όπως για παράδειγμα σε C ή C++. Στην περίπτωσή μας, για να μπορέσουμε μέσα από τον glassfish application server να καλέσουμε κώδικα σε fortran77, πρόβλημα το οποίο ανάγεται στην κλήση fortran κώδικα από περιβάλλον Java, ακολουθήσαμε τη διαδρομή Java→ C→ fortran. Αναλυτικότερα:

Αρχικά, υλοποιήσαμε wrapper συναρτήσεις σε C για τις αντίστοιχες ρουτίνες της LAPACK σε fortran77, και έπειτα, με τη χρήση του JNI, πραγματοποιήσαμε κλήση σε αυτές. Για τη διαδικασία αυτή χρησιμοποιήσαμε το εργαλείο Swig, το οποίο μέσα από τη χρήση ενός interface file (*.i), στο οποίο περιγράφονται οι συναρτήσεις της C διεπαφής, παράγει τις απαραίτητες Java κλάσεις, οι οποίες είναι αναγκαίες για την κλήση των μεθόδων υλοποιημένων σε C.

Στο παρακάτω σχήμα παραθέτουμε αριθμητικά τα βήματα που ακολουθήσαμε κατά τη διαδικασία αυτή:



Σχήμα 4.2: Βήματα για την κλήση μεθόδων της LAPACK από το JVM.

4.3 Υλοποίηση των Web Services

Έπειτα από τη διαδικασία που περιγράφηκε αναλυτικά στην παράγραφο 4.1, υλοποιήσαμε τις αντίστοιχες Web Services μεθόδων της LAPACK. Για την απόδειξη του σεναρίου, υλοποιήσαμε δύο υπηρεσίες την DDOT, η οποία υπολογίζει το εσωτερικό γινόμενο δύο διανυσμάτων διάστασης n, και την DGBSV, η οποία υπολογίζει τη λύση του συστήματος γραμμικών εξισώσεων Ax=b, όπου ο A είναι πίνακας ζώνης. Κάποιος μπορεί κάλλιστα, ακολουθώντας ακριβώς την ίδια διαδικασία, να εκθέσει οποιαδήποτε μέθοδο της LAPACK, όπως και ολόκληρη τη βιβλιοθήκη, ως Web Services.

Η προσέγγιση, που ακολουθήσαμε για την ανάπτυξη των υπηρεσιών, ήταν bottom-up, δηλαδή ξεκινήσαμε από την υλοποίηση της υπηρεσίας σε Java και έπειτα με τη χρήση εργαλείων δημιουργήσαμε τα stubs που απαιτούνται για την κλήση της, όπως και τα WSDL με τις περιγραφές των υπηρεσιών.

Ο glassfish application server κατά την εκκίνησή του φορτώνει τις αντίστοιχες δυναμικές βιβλιοθήκες (.so), οι οποίες περιέχουν τον κώδικα των μεθόδων της LAPACK και αναμένει εισερχόμενες συνδέσεις. Κατά την άφιξη ενός SOAP request για κάποια από τις υπηρεσίες μας, γίνεται επεξεργασία του μηνύματος και πραγματοποιείται κλήση της αντίστοιχης μεθόδου.

Στο παράρτημα Α περιέχεται ο κώδικας σε Java των services, ενώ στο παράρτημα Β τα WSDL's, τα οποία περιέχουν πληροφορίες για τον τρόπο κλήσης τους.

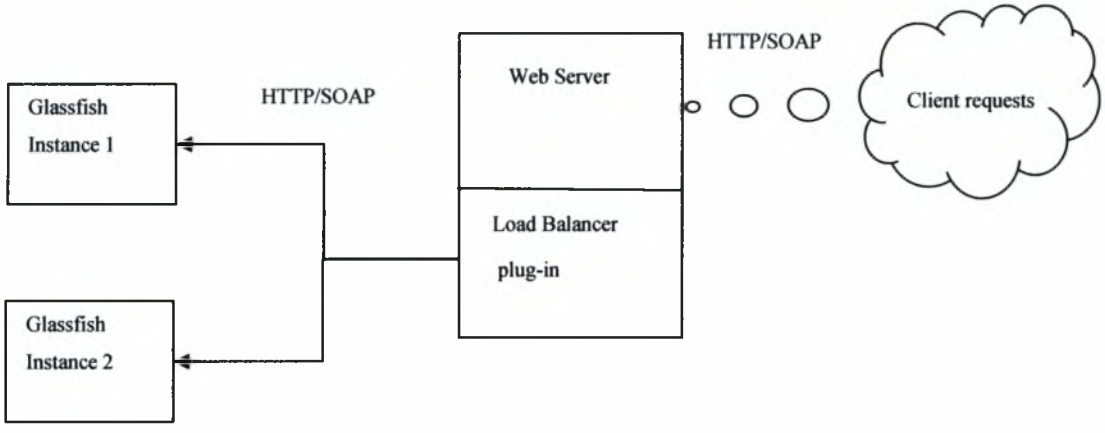
4.4 Αρχιτεκτονική Συστήματος

Για υψηλότερη διαθεσιμότητα, σε περίπτωση βλάβης, και γρηγορότερη απόκριση του συστήματος, σε περίπτωση μεγάλου φόρτου εργασίας, υλοποιήθηκε μία cluster διάταξη για το WSRF, το οποίο φιλοξενεί τις υπηρεσίες.

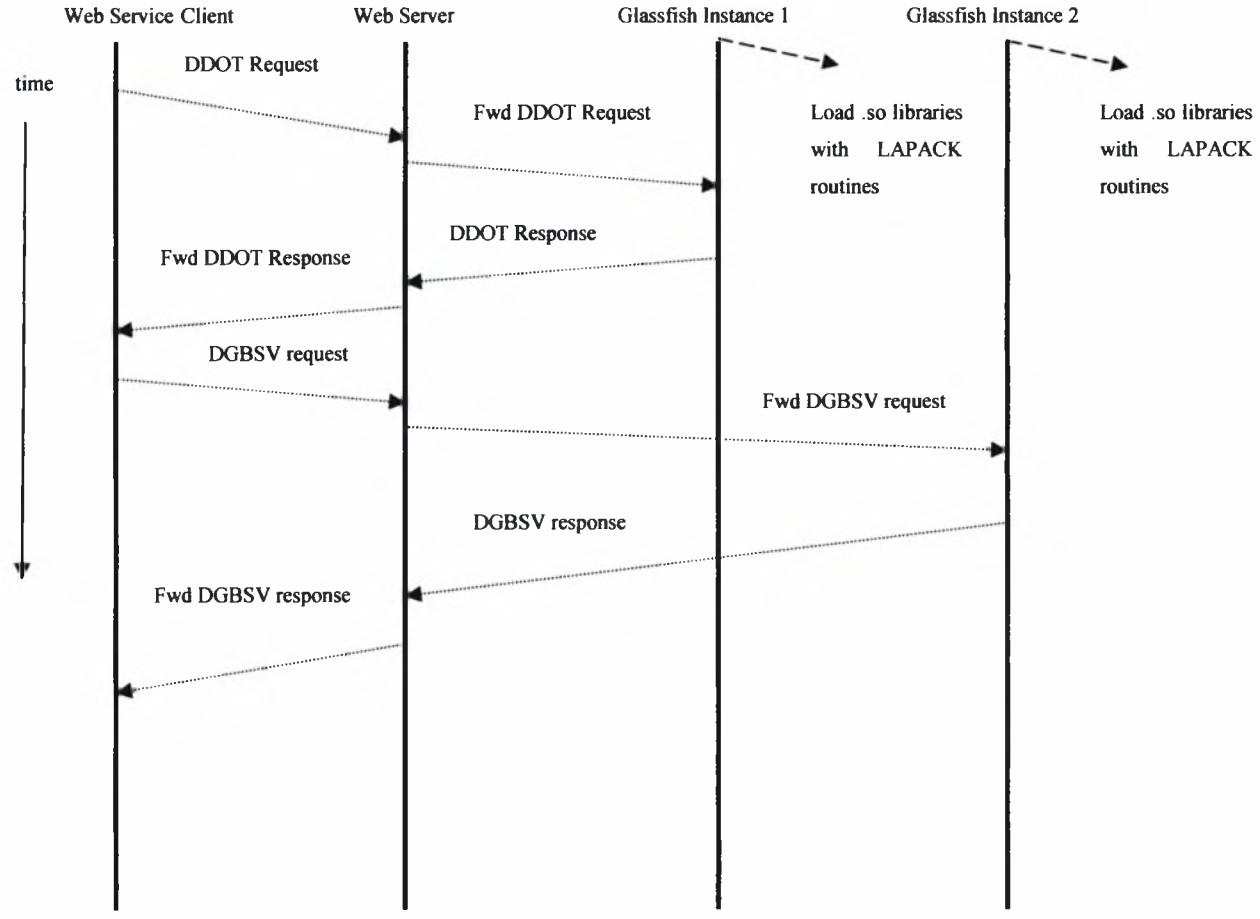
Αναλυτικότερα, το σύστημά μας (Σχήμα 4.3) αποτελείται από έναν web server και δύο WSRF instances. Ο web server, που χρησιμοποιήσαμε για τις ανάγκες μας, ήταν ο Sun Java System Web Server 6.1. Τα SOAP requests, με τη χρησιμοποίηση ως πρωτοκόλλου μεταφοράς του HTTP, αρχικά φθάνουν στον web server. Ο web server με τη σειρά του κατανέμει το φόρτο εργασίας ανάμεσα στα δύο instances, στα οποία γίνεται και η εκτέλεση

των services, προωθώντας το SOAP request στο αντίστοιχο instance. Μόλις φθάσει το SOAP request στο κατάλληλο instance εκτελείται η υπηρεσία, το SOAP response στέλνεται στον web server και τελικά αυτός το προωθεί στην εφαρμογή-πελάτη. Στο σχήμα 4.4 δίνεται ένα παράδειγμα εκτέλεσης.

Η πολιτική που ακολουθεί ο web server για τη δρομολόγηση αυτή είναι πολιτική round robbin με βάρη. Οι ρυθμίσεις του load balancer έχουν γίνει με ίσα ποσοστά κατανομής φορτίου και στα δύο instances (δίνεται η δυνατότητα ρύθμισης του ποσοστού από τον administator). Για το σύστημα που κατασκευάσαμε χρησιμοποιήσαμε δύο instances, αλλά κάποιος θα μπορούσε να χρησιμοποιήσει και περισσότερα, αναλόγως του φόρτου εργασίας και της διαθεσιμότητας που θέλει να επιτύχει.



Σχήμα 4.3: Περιγραφή του συστήματος



Σχήμα 4.3 : Παράδειγμα εκτέλεσης

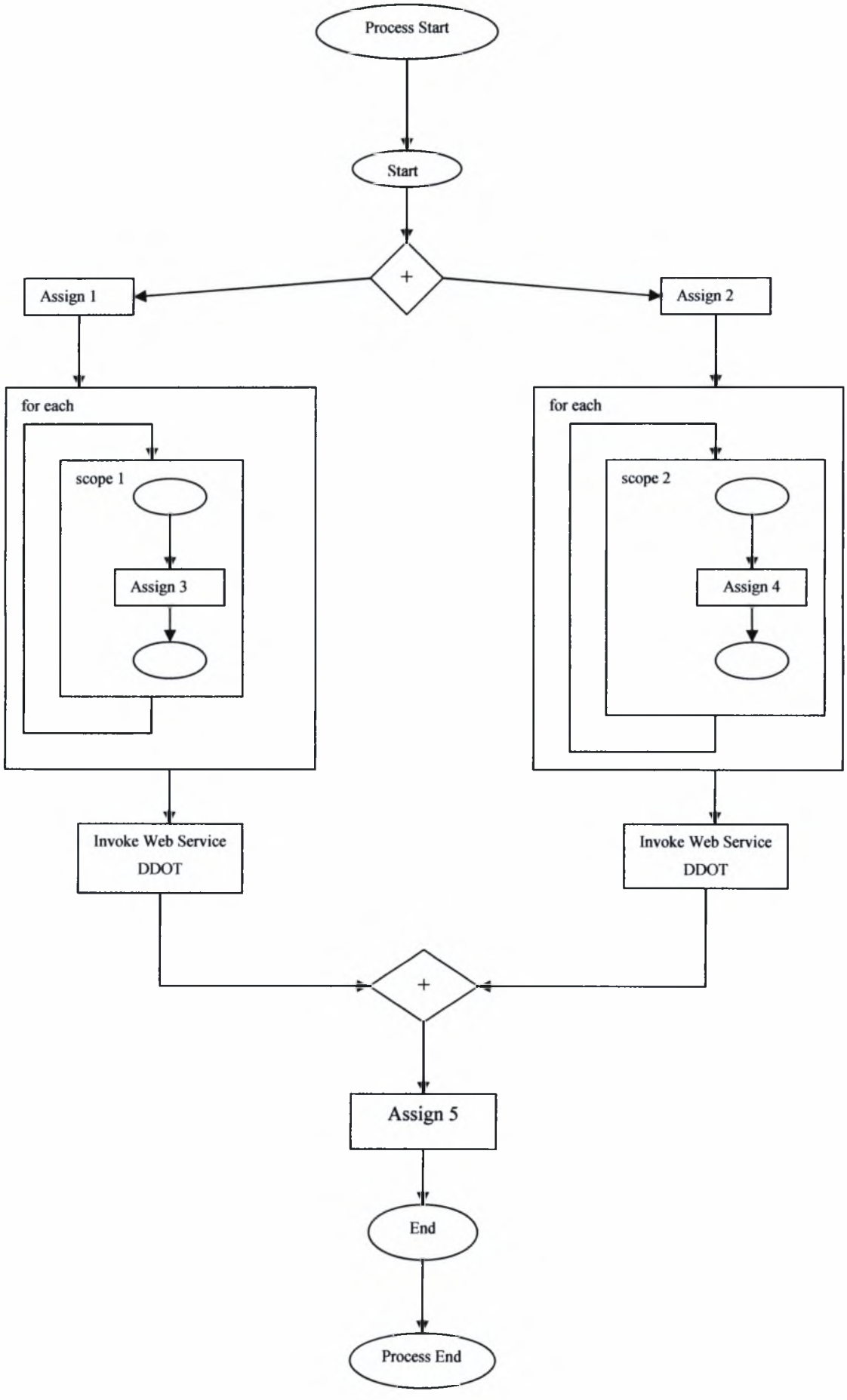
4.5 Σχεδίαση διεργασιών με τη γλώσσα WS-BPEL

Η BPEL είναι μία XML-based γλώσσα για τη δημιουργία flows. Σε XML περιγράφεται μία ροή εργασίας, η οποία αποτελείται από μία σειρά σειριακών ή/και παράλληλων κλήσεων σε Web Services ή σε τοπικά java components, των οποίων τα ενδιάμεσα αποτελέσματα μπορούν να συνδυαστούν, προκειμένου να αποτελέσουν είσοδο σε επόμενες κλήσεις. Η XML περιγραφή της ροής δίνεται ως είσοδο στην BPEL engine, η οποία με τη σειρά της την εκτελεί.

Με τη χρήση της γλώσσας BPEL και των services DDOT και DGBSV υλοποιήσαμε δύο ροές εργασίας: μία για τον παράλληλο υπολογισμό του εσωτερικού γινομένου δύο διανυσμάτων και μία ροή για την επίλυση ενός γραμμικού συστήματος εξισώσεων με τη χρήση της DGBSV.

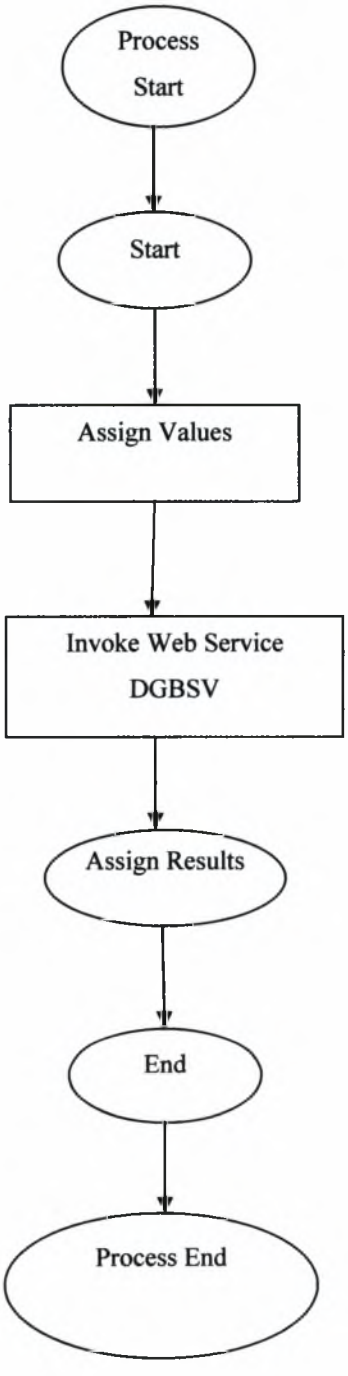
Η ροή εργασίας για το παράλληλο υπολογισμό του DDOT παίρνει ως είσοδο δύο διανύσματα, έστω A και B. Στη συνέχεια, μοιράζει τα διανύσματα σε δύο ίσα μέρη το καθένα και έπειτα εκτελεί παράλληλη κλήση της υπηρεσίας DDOT για το καθένα από αυτά. Δηλαδή

εκτελεί την κλήση DDOT (A[0, n/2], B[0, n/2]) και παράλληλα την DDOT (A[n/2,n], B[n/2, n]). Στη συνέχεια, μετά το πέρας των δύο κλήσεων αθροίζει τα αποτελέσματα και παράγει το τελικό ζητούμενο. Εδώ να σημειώσουμε ότι η ροή εργασίας που αναπτύξαμε είναι, επίσης, ένα Web Service, το οποίο το κάναμε deploy στον application servers μας.



Σχήμα 4.4: Parallel DDOT flow

Η επόμενη ροή εργασίας που σχεδιάσαμε είναι για την επίλυση ενός γραμμικού συστήματος εξισώσεων Ax=b, όπου ο A είναι πίνακας ζώνης. Η ροή εργασίας στέλνει μόνο τα μη μηδενικά στοιχεία του πίνακα. Η υπηρεσία DGBSV μετατρέπει τον πίνακα σε μορφή συμβατή με την fortran77 (δηλαδή αλλάζει τον τρόπο αποθήκευσης από γραμμές σε στήλες) και πραγματοποιεί την κλήση στη μέθοδο DGBSV της LAPACK. Στη συνέχεια, επιστρέφει το διάνυσμα b στην εφαρμογή-πελάτη.



Σχήμα 4.5: DGBSV flow

4.6 Σύνοψη

Η χρήση των Web Services αλλάζει τον τρόπο με τον οποίο κάνουμε επιστημονικούς υπολογισμούς και προσφέρει πολλές νέες δυνατότητες. Οποιαδήποτε οντότητα μέσα στο Διαδίκτυο μπορεί να έχει ομοιόμορφη πρόσβαση τόσο σε υλικό για την εκτέλεση υπολογισμών, όσο και σε βιβλιοθήκες.

Η τεχνολογία των Web Services παρέχει μεγάλη διαφάνεια στον τελικό χρήστη. Έτσι, μία υπηρεσία που εκτελεί επιστημονικούς υπολογισμούς μπορεί να τρέχει είτε τοπικά σε ένα απομακρυσμένο server, είτε σε κάποιο computational cluster ή ακόμα και σε ένα grid περιβάλλον. Σε καθεμία από τις παραπάνω περιπτώσεις εξάγουν τη λογική των υπηρεσιών, κρύβοντας τις υπόλοιπες λεπτομέρειες.

Στο επόμενο κεφάλαιο παραθέτουμε μία περιγραφή της υφιστάμενης κατάστασης του Hellas Grid, καθώς και την έρευνα που κάναμε γύρω από τη δυνατότητα ανάπτυξης Web Services, οι οποίες για την εκτέλεση των υπολογισμών θα χρησιμοποιούν την υποδομή του Hellas Grid.

5 Hellas Grid / EGEE

Στο ακόλουθο κεφάλαιο γίνεται μία σύντομη περιγραφή της υποδομής του Hellas Grid, το οποίο αποτελεί υποσύνολο του EGEE. Στη συνέχεια, αναφέρουμε τα προβλήματα που υπάρχουν στην ανάπτυξη εφαρμογών στο Hellas Grid και τέλος περιγράφουμε αναλυτικά τη διαδικασία ανάπτυξης Web Services, οι οποίες δρουν ως ενδιάμεσος ανάμεσα στο χρήστη και στην υποδομή για την αποστολή εργασιών.

5.1 Η υποδομή του Hellas Grid

Το EGEE (Enabling Grids for E-sciencE) είναι ένα ερευνητικό πρόγραμμα, το οποίο χρηματοδοτείται από την Ευρωπαϊκή Ένωση και ενώνει 70 ιδρύματα σε 27 χώρες. Στόχος του είναι η κατασκευή μιας Grid υποδομής για τον ευρωπαϊκό τομέα έρευνας. Αυτή τη στιγμή αποτελείται από 41.000 CPU's κατανεμημένες σε 250 sites, τα οποία αποτελούνται από ερευνητικά κέντρα, πανεπιστήμια, εταιρίες και τρίτους φορείς. Αν και το υλικό ποικίλει, στην υποδομή του EGEE χρησιμοποιείται η διανομή του Scientific Linux, ενώ το middleware του EGEE είναι το glite. Το Hellas Grid αποτελεί υποσύνολο του EGEE.

Η υποδομή του Hellas Grid αποτελείται από 6 υπολογιστικές συστοιχίες (clusters) με συνολική ισχύ 768 CPUs (384-dual), καθώς και 90 TB αποθηκευτικού χώρου, 30 με τη μορφή δίσκων και 60 με τη μορφή βιβλιοθηκών ταινιών (tape libraries). Οι συστοιχίες, που αποτελούν τους κόμβους του Grid, φιλοξενούνται σε ιδρύματα στην Αθήνα (3), τη Θεσσαλονίκη(1), την Πάτρα (1) και το Ηράκλειο Κρήτης (1).

Σχήμα 5.1: Η υποδομή του Hellas Grid.

5.2 Πρόσβαση στην υποδομή

Για την απόκτηση πρόσβασης στην υποδομή απαιτείται, αρχικά, η έκδοση ενός ψηφιακού πιστοποιητικού X.509. Η απόκτηση του παραπάνω γίνεται αρχικά με τη συμπλήρωση μίας on-line αίτησης και έπειτα την πιστοποίηση της ταυτότητας του χρήστη από την αρμόδια αρχή πιστοποίησης ανά περιοχή. Με την απόκτηση του πιστοποιητικού αυτού, ο χρήστης εισέρχεται στο Virtual Organization του Southern Eastern Europe. Με το πιστοποιητικό αυτό έχει πρόσβαση στην υποδομή όχι μόνο του Hellas Grid, αλλά και του EGEE (περίπου 4.100 CPU's κατανεμημένες σε όλη την Ευρώπη).

Μετά την ολοκλήρωση της παραπάνω διαδικασίας, για την αλληλεπίδραση με την υποδομή υπάρχουν δύο τρόποι. Ο πρώτος είναι η απόκτηση απομακρυσμένης πρόσβασης (για παράδειγμα ssh) σε κάποιο υπολογιστικό σύστημα, το οποίο παρέχεται από το Hellas Grid, και στο οποίο είναι εγκατεστημένο το User Interface του glite, και ο δεύτερος είναι η εγκατάσταση του glite τοπικά, διαδικασία μάλλον πολύπλοκη για ένα μη πεπειραμένο χρήστη. Οι λόγοι, για τους οποίους η διαδικασία εγκατάστασης του glite είναι πολύπλοκη, είναι ότι είναι απαραίτητο να επιλυθούν πολλές εξαρτήσεις λογισμικού (απαιτείται ειδική διανομή Linux με το όνομα Scientific Linux), και δεν υπάρχει έκδοση του glite διαθέσιμη για άλλες πλατφόρμες (π.χ. Windows, Mac).

Το User Interface του glite αυτή τη στιγμή αποτελείται από ένα σύνολο εργαλείων γραμμής εντολών. Τα εργαλεία αυτά παρέχουν τη λειτουργικότητα για την αποστολή εργασιών και την

ανάκτηση των αποτελεσμάτων. Η απομακρυσμένη εργασία του χρήστη αποτελείται από ένα εκτελέσιμο αρχείο (.out) και ένα jdl αρχείο. Το jdl (job description language) αρχείο περιέχει πληροφορίες για την απομακρυσμένη εργασία, όπως το όνομα του αρχείου εισόδου του προγράμματός μας, του αρχείου εξόδου, την πολυπλοκότητα των υπολογισμών (άνω και κάτω όρια) και άλλες πληροφορίες.

```
Requirements = (other.GlueCEStateStatus == (Production;)
Rank = (-other.GlueCEStateEstimatedResponseTime);
Executable = poisson_cg;
StdOutput = stdout.log;
StdError = stderr.log;
InputSandbox={poisson_cg};
OutputSandbox = {stdout.log, stderr.log};
```

Σχήμα 5.2: Παράδειγμα ενός jdl αρχείου

Ένα τυπικό σενάριο χρήσης του Hellas Grid για την αποστολή μιας εργασίας είναι το παρακάτω:

-Ο χρήστης με τη βοήθεια της εντολής voms-proxy-init επικοινωνεί με τον VOMS server για την έκδοση ενός προσωρινού πιστοποιητικού, το οποίο είναι έγκυρο για μία διάρκεια ωρών, που καθορίζεται από τον ίδιο. Οι λόγοι, για την έκδοση αυτού του προσωρινού πιστοποιητικού, είναι ότι αυτό χρησιμοποιείται από οντότητες του Grid, ως αντιπροσώπου του χρήστη για τη διαπραγμάτευση πόρων, γεγονός που το κάνει ευάλωτο σε θέματα ασφάλειας, λόγω του ότι μετακινείται πάνω από δημόσιο δίκτυο.

-Στη συνέχεια, με τη χρήση της εντολής glite-wms-job-submit, ο χρήστης στέλνει την εργασία του (*.out), την οποία θέλει να εκτελέσει απομακρυσμένα, καθώς και το jdl file, το οποίο περιέχει την περιγραφή της εργασίας, καθώς και τυχόν αρχεία εισόδου του προγράμματος. Κατά το στάδιο αυτό, υπάρχει επικοινωνία με τον Workload Manager (λεπτομέρειες αναφέρουμε παρακάτω) και έχουμε μεταφορά των παραπάνω αρχείων από την πλευρά του πελάτη στον WMS. Μετά την επιτυχημένη παράδοση της εργασίας μας, παίρνουμε ως αποτέλεσμα ένα μοναδικό αναγνωριστικό (job id), το οποίο πρέπει να χρησιμοποιήσουμε αργότερα για την ενημέρωση της κατάστασης της εργασίας μας.

-Με την εντολή glite-wms-job-status, ο χρήστης ελέγχει περιοδικά την κατάσταση της εργασίας που έχει στείλει στο Hellas Grid προς εκτέλεση. Οι πιθανές καταστάσεις μια εργασίας είναι Ready, Scheduled, Submitted, Running, Finished και Failed.

Έπειτα από τη μετάβαση της κατάστασης μίας εργασίας σε finished, ο χρήστης μπορεί να επικοινωνήσει με τον WMS και να λάβει πίσω τα αποτελέσματά του μέσω της εντολής glite-wms-job-output.

Όπως βλέπουμε, η ανάπτυξη εφαρμογών για το Hellas Grid με τα παραπάνω εργαλεία είναι δύσχρηστη. Η λειτουργικότητα που μας παρέχει το User Interface του glite είναι περιορισμένη. Από την υποδομή του Hellas Grid απουσιάζουν τα κατάλληλα API's σε επίπεδο εφαρμογής, τα οποία θα κρύβουν τις παραπάνω λεπτομέρειες και θα κάνουν απλή την ανάπτυξη εφαρμογών σε αυτό, έτσι ώστε να διαδοθεί ευρέως στην επιστημονική κοινότητα.

5.3 Ανάπτυξη υπηρεσιών στο Hellas Grid

Έχοντας αναφέρει παραπάνω τα προβλήματα σχετικά με την ανάπτυξη εφαρμογών πάνω στην υποδομή του Hellas Grid, κύρια ιδέα μας ήταν η ανάπτυξη Web Services για scientific computing, οι οποίες για την πραγματοποίηση των υπολογισμών θα χρησιμοποιούν την υποδομή αυτή.

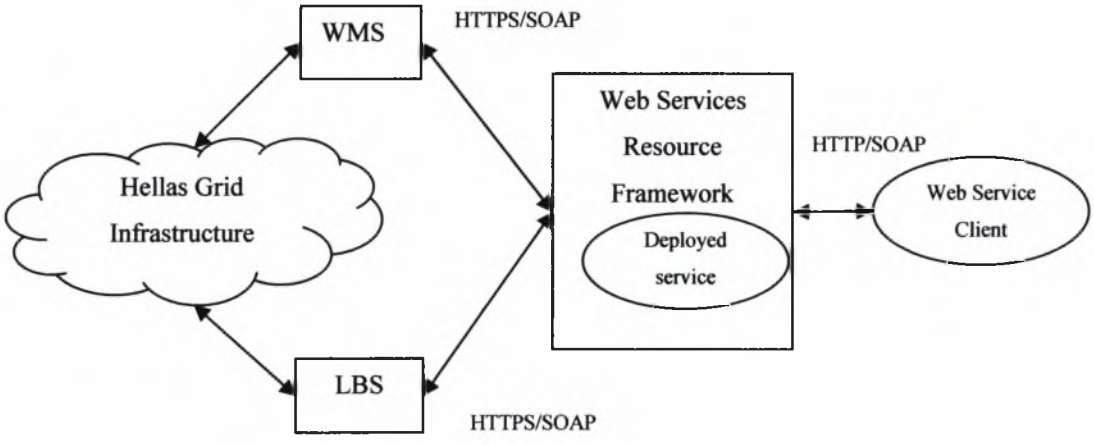
Με την προσέγγιση αυτή, για τις εφαρμογές-πελάτες αυτών των Web Services, η χρήση του Hellas Grid γίνεται τόσο απλή, όσο και η κλήση μιας Web Service με τα πλεονεκτήματα, τα οποία συνεπάγεται αυτό (διαλειτουργικότητα, ανεξαρτησία πλατφόρμας). Επίσης, καθίσταται εφικτή η σύνθεση scientific flows, είτε σε BPEL είτε με τη χρήση κάποιου άλλου εργαλείου για το σκοπό αυτό, όπως το Taverna, των οποίων ορισμένα τμήματα θα τρέχουν τοπικά και άλλα, πιο απαιτητικά, θα χρησιμοποιούν την υποδομή του Hellas Grid με την κλήση μιας Web Service.

5.4 Σχεδίαση και υλοποίηση των υπηρεσιών

Για την αποστολή μιας εργασίας στο Hellas Grid απαιτούνται τα ακόλουθα βήματα:

1) Επικοινωνία με τον VOMS server για την έκδοση ενός προσωρινού πιστοποιητικού.
2) Αποστολή του εκτελέσιμου της εργασίας μας (*.out), τυχόν αρχείων εισόδου για το πρόγραμμά μας και του jdl αρχείου, το οποίο περιέχει πληροφορίες για την εργασία.
3) Επικοινωνία με το Workload Management System (WMS). Το WMS παρέχει ένα API υλοποιημένο ως secure Web Services. Το API αυτό παρέχει υπηρεσίες για την αποστολή εργασιών και την επιστροφή αποτελεσμάτων. Το WSDL της υπηρεσίας αυτής βρίσκεται στο Παράρτημα Β.
4) Έπειτα από την παράδοση της εργασίας, απαιτείται η επικοινωνία με την Logging And Bookkeeping Service. Η υπηρεσία αυτή είναι υλοποιημένη ως secure Web Services και παρέχει μεθόδους για την ενημέρωση της κατάστασης της εργασίας μας. Το WSDL της υπηρεσίας αυτής βρίσκεται, επίσης, στο Παράρτημα Β.

Επειδή η παράδοση εργασιών και η επιστροφή των αποτελεσμάτων είναι ασύγχρονες, πρέπει να ακολουθηθεί η εξής διαδικασία: Αμέσως μετά την παράδοση της εργασίας, μέσα από την υπηρεσία Job Submit του WMS, επιστρέφεται ένα μοναδικό id για την εργασία αυτή. Με τη χρήση αυτού του id και την περιοδική επικοινωνία με την Logging and Bookkeeping service ενημερωνόμαστε για την κατάσταση της εργασίας μας. Όταν μεταβεί η κατάσταση της εργασίας μας σε finished, επικοινωνούμε με τον WMS για την ανάκτηση των αποτελεσμάτων και τα αποτελέσματα επιστρέφονται στην εφαρμογή-πελάτη. Στο σχήμα 5.2 παρουσιάζεται η αρχιτεκτονική των services.



Σχήμα 5.2 Η προτεινόμενη αρχιτεκτονική των services

Για τη δημιουργία μιας τέτοιας υπηρεσίας, αρκεί η ανάπτυξη μιας Java μεθόδου με την παραπάνω λειτουργικότητα. Μία τέτοια μέθοδος μπορεί είτε να κληθεί ως τοπικό component ενός scientific flow, είτε να ενσωματωθεί στο WSRF και να είναι διαθέσιμο ως Web Service.

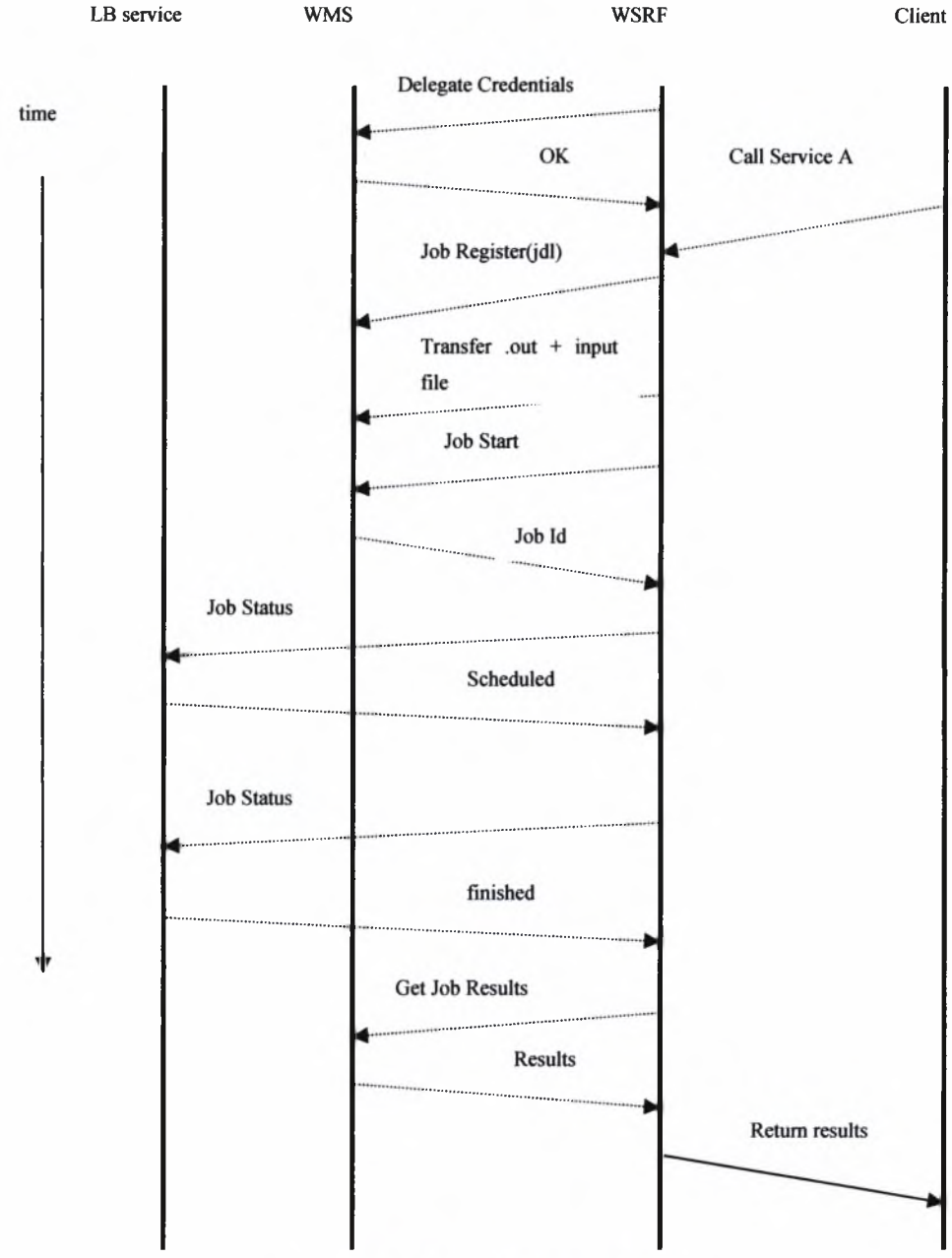
Για την ανάπτυξη, σε Java, κώδικα για την κλήση των υπηρεσιών του WMS και της LB service, χρησιμοποιήσαμε το εργαλείο wsdl2java, που παρέχεται από το apache axis, το οποίο παράγει τα απαραίτητα stubs, που απαιτούνται για την κλήση των υπηρεσιών, σε java. Όσο για την ασφαλή επικοινωνία, χρησιμοποιήσαμε τμήματα του JGlobus Toolkit. Το JGlobus Toolkit είναι ένα API υλοποιημένο σε Java και παρέχει διάφορες υλοποιήσεις πρωτοκόλλων που χρησιμοποιούνται σε Grid περιβάλλοντα. Το κομμάτι που χρησιμοποιήσαμε εμείς είναι το GSI (Grid Security Infrastructure), το οποίο είναι πρωτόκολλο ασφαλείας με αμοιβαία ταυτοποίηση των οντοτήτων. Κάποιος θα μπορούσε να χρησιμοποιήσει και κάποια δική του υλοποίηση για τη διαδικασία αυτή.

Για την υπηρεσία WMS υπάρχει διαθέσιμο ένα Java API, το οποίο παρέχεται από το EGEE και έχει δημιουργηθεί με τον παραπάνω τρόπο. Για την υπηρεσία Logging and Bookkeeping service δεν υπάρχει κάτι αντίστοιχο.

Παρακάτω ακολουθεί ένα παράδειγμα εκτέλεσης. Το WSRF στην αρχικοποίηση εκτελεί μία φορά τη διαδικασία "Delegate Credentials". Η διαδικασία αυτή είναι η μεταφορά του προσωρινού πιστοποιητικού στον WMS, ο οποίος ελέγχει την ορθότητά του. Το εκτελέσιμο, το οποίο, για παράδειγμα, μπορεί να είναι η διαδικασία της LAPACK DDOT, όπως και το jdl αρχείο είναι προσβάσιμο στο WSRF. Με την αίτηση της εφαρμογής-πελάτη, το WSRF μεταφέρει στον WMS τα αρχεία (*.out), το jdl, καθώς και το αρχείο εισόδου με τις παραμέτρους, το οποίο δημιουργείται βάσει της αιτήσεως της εφαρμογής-πελάτη. Μετά την επιτυχή παράδοση της εργασίας, επιστρέφεται ένα μοναδικό job id, το οποίο την κωδικοποιεί μοναδικά.

Για να πάρουμε τα αποτελέσματα πίσω από τον WMS, πρέπει πρώτα να μεταβεί η εργασία μας σε κατάσταση finished. Για το λόγο αυτό, επικοινωνούμε περιοδικά με την Logging and Bookkeeping Service, χρησιμοποιώντας ως παράμετρο το job id για την κατάσταση της εργασίας μας.

Μόλις μεταβεί η εργασία μας σε κατάσταση finished, επικοινωνούμε με τον WMS, προκειμένου να πάρουμε πίσω τα αποτελέσματα, τα οποία περικλείουμε σε ένα SOAP μήνυμα και τα επιστρέφουμε στην εφαρμογή-πελάτη.



Σχήμα 5.3 Παράδειγμα εκτέλεσης

5.5 Προβλήματα

Τα προβλήματα που αντιμετωπίσαμε, με αποτέλεσμα να μην κατορθώσουμε να υλοποιήσουμε ολοκληρωτικά μία τέτοια υπηρεσία για την απόδειξη του παραπάνω σεναρίου, είναι ότι κατά την προσπάθεια μεταφοράς των αρχείων από την εφαρμογή μας στον WMS server, ενώ επιτυγχάναμε ασφαλή σύνδεση ssl, δε μας επέτρεπε την αποστολή του αρχείου. Αναλυτικότερα, κατά την κλήση της Web Service του WMS getSandboxDestURI (JobId) μας επιστρέφονταν δύο url για τη μεταφορά των απαραίτητων αρχείων (.out και αρχείο εισόδου) για τα πρωτόκολλα που υποστηρίζει ο WMS, τα οποία ήταν το https και το gsiftp. Το gsiftp

είναι το κλασικό ftp, που χρησιμοποιεί ως πρωτόκολλο ασφαλείας το GSI (Grid Security Infrastructure), το οποίο περιλαμβάνει ssl επικοινωνία με την αμοιβαία αναγνώριση και των δύο πλευρών. Δοκιμάσαμε και τα δύο πρωτόκολλα, τόσο δηλαδή τις μεθόδους GET και POST του http, όσο και το gsiftp (με τη χρήση του JavaCog Kit), αλλά, ενώ επιτυγχάναμε εγκαθίδρυση ασφαλούς επικοινωνίας, η απέναντι πλευρά μας απαγόρευε τη μεταφορά των αρχείων. Επίσης, αξίζει να αναφέρουμε ότι το WSDL για την Logging and Bookkeeping Service, το οποίο βρήκαμε από την επίσημη web τοποθεσία του EGEE, αναφερόταν σε παλαιότερη έκδοση της υπηρεσίας με αυτήν που χρησιμοποιείται, με αποτέλεσμα από τις κλήσεις των υπηρεσιών να απουσιάζουν ορίσματα στα SOAP requests και responses. Η εκσφαλμάτωση έγινε από εμάς.

Επικοινωνήσαμε και ενημερώσαμε την ομάδα της τεχνικής υποστήριξης του Hellas Grid για το συγκεκριμένο πρόβλημα, στο οποίο, όμως, δε μας πρόσφεραν λύση.

Παρόλα αυτά, κατορθώσαμε να υλοποιήσουμε μία demo υπηρεσία. Η υπηρεσία αυτή λειτούργησε διότι αυτό που εκτελούσε ήταν η εντολή/echo, η οποία απλά τυπώνει μία συμβολοσειρά (δε χρειαζόταν μεταφορά εκτελέσιμου στον WMS) και, επίσης, επικοινωνήσαμε με την υπηρεσία LB service, προκειμένου να ενημερωθούμε για την κατάσταση της εργασίας.

5.6 Σύνοψη

Το EGEE, και κατ' επέκταση το Hellas Grid, το οποίο είναι υποσύνολό του, είναι μία πολλά υποσχόμενη τεχνολογία. Από τη μελέτη που κάναμε, διαπιστώσαμε ότι το ένα από τα προβλήματα, που παρουσιάζει, είναι η δυσκολία στη χρήση και στην ανάπτυξη εφαρμογών που θα εκμεταλλεύονται αυτήν την τεράστια υπολογιστική ισχύ.

Πιο συγκεκριμένα, την παρούσα χρονική στιγμή λείπουν τα απαραίτητα εργαλεία που θα κάνουν ευκολότερη τη χρήση και την ανάπτυξη εφαρμογών σε αυτήν. Προς τα εκεί μπορούν να στραφούν ερευνητικές προσπάθειες για την ανάπτυξη εργαλείων και API's, που κρύβουν από το χρήστη την πολυπλοκότητα και την ευμεταβλητότητα ενός Grid περιβάλλοντος και θα το διαδώσουν ευρέως στην επιστημονική κοινότητα.

Βιβλιογραφία

[1] Andreas Fischer: "An Execution Environment for Mathematical Services based on WSRF and WS-BPEL". Johannes Kepler University, Austria, December 2005.

[2] Zhou Jun, Yukio Umetani: "A Problem Solving Environment for Automatic Matlab 3D Finite Element Code Generation and Simplified Grid Computing". In e-Science and Grid Computing, Second IEEE International Conference on Volume, December 2006.

[3] Minas D. Koulisianis, George K. Tsolis, Thedore S. Papatheodorou: "A Web-Based Problem Solving Environment for Solution of Option Pricing Problems and Comparison of Methods". In Computational Science – ICCS 2002, pages 673-682.

[4] Paul Chew, Nikos Chrisochoides et al.: "Computational Science Simulations based on Web Services". In Computational Science – ICCS 2003, page 721.

[5] E. Caron, F. Desprez: "Diet: A Scalable Toolbox to build Network Enabled Servers on the Grid". In International Journal of High Performance Computing Applications, Vol. 20, No. 3, 2006, pages 335-352.

[6] Jack Dongarra, Henri Casanova et al.: "GridRPC: A Remote Procedure Call API for Grid Computing". Grid Computing – Grid 2002, pages 274 – 278.

[7] Ian Foster, Carl Kesselman, Jeffrey M. Nick, Steven Tuecke: "Grid Services for Distributed System Integration". Computer Archive, Vol. 35, No. 6, June 2002, pages 37-47.

[8] http://icl.cs.utk.edu/netsolve/, NetSolve / GridSolve project homepage.

[9] Sanjiva Weerawarana, Elias N. Houstis, John R. Rice et al.: "Web//ELLPACK : A Networked Computing Service on the World Wide Web".

[10] Michael Tomas, Conrad Steenberg, Frank Van Lingen, Harvey Newman et al.: "JClarens: A Java Framework for Developing and Deploying Web Services for Grid Computing". In IEEE International Conference on Web Services, July 2005.

[11] Keith Seymour, Asim YarKhan, Sudesh Agrawal, Jack Dongarra: "Netsolve: Grid Enabling Scientific Computing Environments". In Grid Computing and New Frontiers of High Performance Computing, 2005.

[12] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, S. Matsuoka: "Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing". In Journal of Grid Computing, Volume 1, March 2003.

[13] Hidemoto Nakada, Yoshio Tanaka: "The Design and Implementation of a Fault-Tolerant RPC system: Ninf-C". In International Conference on High Performance Computing and Grid in Asia Pacific Region, 2004.

[14] Matthew Shields, Ian Taylor: "Programming Scientific and Distributed Workflow with Triana Services". In Concurency and Computation: Practise and Experience, Volume 18, Pages 1021-1037, December 2005.

[15] E. Laure, S.M. Fischer, A. Frohner, C. Grandi et al.: "Programming the Grid with gLite". In Computational Methods in Science and Technology 12(1), Pages 33-45, 2006.

[16] Madhusudhan Govindaraju, Aleksander Slominski, Venkatesh Chopella et al.: "Requirements for and Evaluation of RMI Protocols for Scientific Computing". In Supercomputing, ACM/IEEE 2000.

[17] Ian Foster: "Service-Oriented Science". In Science, Volume 308, 6 May 2005.

[18] Sriram Krishnan, Karan Bhatia: "SOA's for Scientific Applications: Experiences and Challenges". In Proceedings of the Third International Conference on e-Science and Grid-Computing, 2007.

[19] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin et al.: "Taverna: a tool for the composition and enactment of bioinformatics workflows". In Journal Bioinformatics, Pages 3045-3054, 2004.

[20] Diego Puppin, Nicolla Tonelloto and Domenico Laforenza: "Using Web Services to Run Distributed Numerical Applications". In Recent Advances in Parallel Virtual Machine and Message Passing Interface, Pages 207-214, November 2004.

[21] Karl Czajkowski, Donald F. Ferguson, Ian Foster, Jeffrey Frey et al.: "The WS-Resource Framework". WSRF v1.2 specifications, January 2004.

[22] Frank Deremer, Hans Kron: "Programming in the large Versus Programming in the small". Proceedings of the international conference on Reliable software, 1975.

[23] Michael Blow, Yaron Goland, Matthias Kloppmann et al.: "BPELJ: BPEL for Java". A Joint White Paper by BEA and IBM, March 2004.

[24] '

[25] http://www.netlib.org/lapack/, LAPACK web page.

[26] https://glassfish.dev.java.net/, Glassfish application server web page.

[27] http://www.hellasgrid.gr/, Hellas Grid web page.

[28] http://public.eu-egee.org/, EGEE web page.

[29] http://trinity.datamat.it/projects/EGEE/wiki/wiki.php, WMProxy API Documentation.

[30] http://ws.apache.org/axis/, Apache axis web page.

[31] http://wiki.cogkit.org/index.php/Main_Page, JavaCoG kit web page.

[32] http://en.wikipedia.org/wiki/Grid_computing, Grid Computing definition.

[33] http://en.wikipedia.org/wiki/X_window_system, the X Window System definition.

[34] http://www.investorwords.com/3488/option_price.html, option price definition.

[35] http://en.wikipedia.org/wiki/CORBA, CORBA definition.

[36] https://jxta.dev.java.net/, JXTA project home page.

[37] Sathish S. Vadhiyar, Jack J. Dongarra: "GrADSolve - A Grid-based RPC system for Remote Invocation of Parallel Software". Journal of Parallel and Distributed Computing , July 2003.

[38] http://www.xmlrpc.com/, the XML-RPC home page.

[39] http://www.unicore.eu/, Unicore home page.

[40] http://technet.microsoft.com/en-us/library/cc722925.aspx, DCOM technical overview.

[41] http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp, Java RMI.

ΠΑΡΑΡΤΗΜΑ Α

DDOT Java implementation

```
package server;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

@WebService()
public class DdotService {


    public static boolean loaded=false;

/**
     * Web Service operation
     */
    @WebMethod(operationName = "ddot")
    public double ddot(@WebParam(name = "N")
    int N, @WebParam(name = "MatrixA")
    double[] MatrixA, @WebParam(name = "INCX")
    int INCX, @WebParam(name = "MatrixB")
    double[] MatrixB, @WebParam(name = "INCY")
    int INCY) {

    double result;

    if(!loaded){
       try{
          System.loadLibrary("ddot");
           loaded=true;
           } catch(UnsatisfiedLinkError exc){}
       }


        System.out.println("Σ ="+N+" INCX="+INCX+" INCY="+INCY);

        for(int i=0;i<MatrixA.length;i++){
            System.out.println("MatrixA["+i+"]="+MatrixA[i]);
```

```
        }

        for(int i=0;i<MatrixB.length;i++){
            System.out.println("MatrixB["+i+"]="+MatrixB[i]);
        }



        SWIGTYPE_p_int n=ddot.new_intArray(1);
        SWIGTYPE_p_double a=ddot.new_doubleArray(N);
        SWIGTYPE_p_int incx=ddot.new_intArray(1);
        SWIGTYPE_p_double b=ddot.new_doubleArray(N);
        SWIGTYPE_p_int incy=ddot.new_intArray(1);


        ddot.intArray_setitem(n, 0, N);
        ddot.intArray_setitem(incx, 0, INCX);
        ddot.intArray_setitem(incy, 0, INCY);

        for(int i=0;i<N;i++){

                ddot.doubleArray_setitem(a, i, MatrixA[i]);
                ddot.doubleArray_setitem(b, i, MatrixB[i]);
        }

            result=ddot.ddot_(n, a, incx, b, incy) ;
            return result;


    }

       public static int max(int a, int b){
                if(a>=b){return a;}
                else return b;
            }

            public static int min(int a,int b){

                if(a<=b){return a;}
                else return b;

               }}
```

DGBSV Java implementation

```
package server;
```

```
public class DGBSV {


        static{
                try{
                System.load("/home/gesalous/lapack_test/dgbsv/libdgbsv.so");
              }
catch(Exception exc){System.out.println(exc.getMessage());}


        }


        public static void main(String[] args){


        double[]      A={-0.23,-6.98,2.54,2.46,2.56,-3.66,-2.73,2.46,-4.78,-2.13,4.07,-
3.82};
        double[] B={4.42,27.13,-6.14,10.50};

        dgbsv(4,1,2,1,A,5,B,4);



        }


    public static void dgbsv(int N,int KL,int KU,int NRHS,double[] A,int LDAB,double[]
B,int LDB){

            /*memory allocation*/
            double[] AB=new double[LDAB*N];
            int[] IPIV=new int[N];
            int INFO = 0;

            /*end of memory allocation*/

            /*pointers declarations*/
            SWIGTYPE_p_int n=dgbsv.new_intArray(1);
            SWIGTYPE_p_int kl=dgbsv.new_intArray(1);
            SWIGTYPE_p_int ku=dgbsv.new_intArray(1);
            SWIGTYPE_p_int nrhs=dgbsv.new_intArray(1);
            SWIGTYPE_p_double ab=dgbsv.new_doubleArray(LDAB*N);
            SWIGTYPE_p_int ldab=dgbsv.new_intArray(1);
            SWIGTYPE_p_int ipiv=dgbsv.new_intArray(N);
            SWIGTYPE_p_double b=dgbsv.new_doubleArray(LDB*NRHS);
            SWIGTYPE_p_int ldb=dgbsv.new_intArray(1);
            SWIGTYPE_p_int info =dgbsv.new_intArray(1);
            /*end of pointer declarations*/

            int i_pos,j_pos;
            int APosition=0;
            double result;




              for(int j=1;j<=N;j++){

                j_pos=j;

                for(int m=max(1,j-KU);m<=min(N,j+KL);m++){

                        i_pos=m;
  AB[ (((KL+KU+1+i_pos-j_pos))+(((j_pos)-1)*LDAB))-1]=A[APosition];
                    APosition++;
                }



       }    /*end of umarshall*/
```

```
        /*    for(int i=0;i<AB.length;i++){
                System.out.print(" Ξ'Ξ' "+AB[i]);
        }*/

        /*Initialize pointers*/
        dgbsv.intArray_setitem(n, 0, N);
        dgbsv.intArray_setitem(kl, 0, KL);
        dgbsv.intArray_setitem(ku, 0, KU);
        dgbsv.intArray_setitem(nrhs, 0, NRHS);

        for(int i=0;i<AB.length;i++){

            dgbsv.doubleArray_setitem(ab, i, AB[i]);
    }

        dgbsv.intArray_setitem(ldab, 0, LDAB);
        for(int i=0;i<IPIV.length;i++){

            dgbsv.intArray_setitem(ipiv, i, IPIV[i]);
    }
           for(int i=0;i<B.length;i++){

            dgbsv.doubleArray_setitem(b, i, B[i]);
    }

           dgbsv.intArray_setitem(ldb, 0, LDB);
           dgbsv.intArray_setitem(info, 0, INFO);


dgbsv.dgbsv_(n, kl, ku, nrhs, ab,ldab, ipiv, b, ldb, info);

        System.out.println("info "+dgbsv.intArray_getitem(info, 0));

        System.out.println("B ");
        for(int i=0;i<B.length;i++){
        System.out.print(dgbsv.doubleArray_getitem(b, i)+" ");
       /* System.out.print(B[i]+" ");*/


}
       System.out.println("IPIV ");
        for(int i=0;i<B.length;i++){
        System.out.print(dgbsv.intArray_getitem(ipiv, i)+" ");


}
        System.out.println("AB ");
        for(int i=0;i<AB.length;i++){
        System.out.print(dgbsv.doubleArray_getitem(ab, i)+" ");


}




    }/*end of dgbsv*/


            public static int max(int a, int b){
                if(a>=b){return a;}
                else return b;
            }

            public static int min(int a,int b){

                if(a<=b){return a;}
                else return b;

              }

   }/*end of class DGBSV*/
```

ПАРАРТНМА В

DDOT WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI
2.1.2-hudson-182-RC1. -->
<!-- Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI
2.1.2-hudson-182-RC1. -->
<definitions        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd"    xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"          xmlns:tns="http://server/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"   xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://server/" name="DdotServiceService">
<types>
<xsd:schema>
<xsd:import                                                    namespace="http://server/"
schemaLocation="http://10.64.13.10:8081/Ddot/DdotServiceService?xsd=1"/>
</xsd:schema>
</types>
<message name="ddot">
<part name="parameters" element="tns:ddot"/>
</message>
<message name="ddotResponse">
<part name="parameters" element="tns:ddotResponse"/>
</message>
<portType name="DdotService">
<operation name="ddot">
<input message="tns:ddot"/>
<output message="tns:ddotResponse"/>
</operation>
</portType>
<binding name="DdotServicePortBinding" type="tns:DdotService">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
<operation name="ddot">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
</binding>
```

```
<service name="DdotServiceService">
<port name="DdotServicePort" binding="tns:DdotServicePortBinding">
<soap:address location="http://10.64.13.10:8081/Ddot/DdotServiceService"/>
</port>
</service>
</definitions>
```

DGBSV WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI
2.1.2-hudson-182-RC1. -->
<!-- Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI
2.1.2-hudson-182-RC1. -->
<definitions       xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd     xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"          xmlns:tns="http://server/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"   xmlns="http://schemas.xmlsoap.org/wsdl/"
targetNamespace="http://server/" name="DGBSVService">
<types>
<xsd:schema>
<xsd:import                                                    namespace="http://server/"
schemaLocation="http://10.64.13.10:8081/Ddot/DGBSVService?xsd=1"/>
</xsd:schema>
</types>
<message name="dgbsv">
<part name="parameters" element="tns:dgbsv"/>
</message>
<message name="dgbsvResponse">
<part name="parameters" element="tns:dgbsvResponse"/>
</message>
<portType name="DGBSV">
<operation name="dgbsv">
<input message="tns:dgbsv"/>
<output message="tns:dgbsvResponse"/>
</operation>
</portType>
<binding name="DGBSVPortBinding" type="tns:DGBSV">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
<operation name="dgbsv">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
</binding>
<service name="DGBSVService">
<port name="DGBSVPort" binding="tns:DGBSVPortBinding">
<soap:address location="http://10.64.13.10:8081/Ddot/DGBSVService"/>
</port>
</service>
</definitions>
```

Parallel DDOT WSDL

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

<definitions
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"              name="SynchronousSample"
targetNamespace="http://localhost/SynchronousSample/SynchronousSample"
    xmlns:tns="http://localhost/SynchronousSample/SynchronousSample"
    xmlns:ns="http://xml.netbeans.org/schema/SynchronousSample"
    xmlns:plink="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
    xmlns:bpws="http://docs.oasis-open.org/wsbpel/2.0/varprop"
xmlns:ns0="http://server/">

    <types>
        <xsd:schema
targetNamespace="http://localhost/SynchronousSample/SynchronousSample">
            <xsd:import    namespace="http://xml.netbeans.org/schema/SynchronousSample"
schemaLocation="SynchronousSample.xsd"/>
            <xsd:import                                          namespace="http://server/"
schemaLocation="Partners/DdotServiceService/localhost_8080/Ddot/DdotServiceService.xsd
_1.xsd"/>
        </xsd:schema>
    </types>

    <message name="requestMessage">
        <part name="inputType" type="ns0:ddot"/>
    </message>

    <message name="responseMessage">
        <part name="resultType" type="ns0:ddotResponse"/>
    </message>

    <portType name="portType1">
        <operation name="operation1">
            <input name="input1" message="tns:requestMessage"/>
            <output name="output1" message="tns:responseMessage"/>
        </operation>
    </portType>

    <binding name="binding1" type="tns:portType1">
        <soap:binding                  transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>
        <operation name="operation1">
            <input name="input1">
                <soap:body use="literal"/>
            </input>
            <output name="output1">
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>

    <service name="service1">
        <port name="port1" binding="tns:binding1">
            <documentation/>
```

```
            <soap:address location="http://localhost:18181/SynchronousSample"/>
        </port>
    </service>

    <plink:partnerLinkType name="partnerlinktype1">
        <plink:role name="partnerlinktyperole1" portType="tns:portType1"/>
    </plink:partnerLinkType>
</definitions>
```

Parallel DDOT flow

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

targetNamespace="http://enterprise.netbeans.org/bpel/SynchronousSample/SynchronousSamp
le_1"
         xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:bpws="http://docs.oasis-open.org/wsbpel/2.0/process/executable"

xmlns:wsdlNS="http://enterprise.netbeans.org/bpel/SynchronousSample/SynchronousSample_
1"
         xmlns:ns1="http://localhost/SynchronousSample/SynchronousSample"
         xmlns:ns2="http://xml.netbeans.org/schema/SynchronousSample"
xmlns:ns3="http://server/">

    <import namespace="http://xml.netbeans.org/schema/SynchronousSample"
            location="SynchronousSample.xsd"
            importType="http://www.w3.org/2001/XMLSchema"/>
    <import namespace="http://localhost/SynchronousSample/SynchronousSample"
            location="SynchronousSample.wsdl"
            importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import                                                    namespace="http://server/"
location="Partners/DdotServiceService/DdotServiceService.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/"/>
    <partnerLinks>
        <partnerLink              name="Ddot"              partnerLinkType="ns3:Ddot"
partnerRole="DdotProvider"/>
        <partnerLink name="SynchronousSample"
                     partnerLinkType="ns1:partnerlinktype1"
                     myRole="partnerlinktyperole1"/>
    </partnerLinks>

    <variables>

        <variable name="DdotOut1" messageType="ns3:ddotResponse"/>
        <variable name="DdotIn1" messageType="ns3:ddot"/>

        <variable name="DdotOut" messageType="ns3:ddotResponse"/>
        <variable name="DdotIn" messageType="ns3:ddot"/>
        <variable name="output" message Type="ns1:response Message"/>
        <variable name="input" message Type="ns1:request Message"/>
    </variables>

    <sequence>
        <receive name="start"
                 partner Link="Synchronous Sample"
                 operation="operation1"
                 portType="ns1:portType1"
                 variable="input"
                 create Instance="yes">
        </receive>
        <flow name="Flow1">
            <sequence name="Sequence1">
                <assign name="Assign1">
                    <copy>
                        <from>   ( $inputVar.inputType/N div 2 )   </from>
                        <to>$DdotIn.parameters/N</to>
```

```
                    </copy>
                    <copy>
                        <from>$inputVar.inputType/INCX</from>
                        <to>$DdotIn.parameters/INCX</to>
                    </copy>
                    <copy>
                        <from>$inputVar.inputType/INCY</from>
                        <to>$DdotIn.parameters/INCY</to>
                    </copy>
                </assign>
                <forEach name="ForEach1" parallel="no" counterName="counter1">
                    <startCounterValue>1</startCounterValue>
                    <finalCounterValue>         (   $inputVar.inputType/N   div   2   )
</finalCounterValue>
                    <scope name="Scope1">
                        <assign name="Assign3">
                            <copy>
                                <from>$inputVar.inputType/MatrixA[$counter1]</from>
                                <to>$DdotIn.parameters/MatrixA[$counter1]</to>
                            </copy>
                            <copy>
                                <from>$inputVar.inputType/MatrixB[$counter1]</from>
                                <to>$DdotIn.parameters/MatrixB[$counter1]</to>
                            </copy>
                        </assign>
                    </scope>
                </forEach>
                <invoke     name="Invoke1"     partnerLink="Ddot"     operation="ddot"
portType="ns3:DdotService" inputVariable="DdotIn" outputVariable="DdotOut"/>
            </sequence>
            <sequence name="Sequence2">
                <assign name="Assign2">
                    <copy>
                        <from>   ( $inputVar.inputType/N div 2 )   </from>
                        <to>$DdotIn1.parameters/N</to>
                    </copy>
                    <copy>
                        <from>$inputVar.inputType/INCX</from>
                        <to>$DdotIn1.parameters/INCX</to>
                    </copy>
                    <copy>
                        <from>$inputVar.inputType/INCY</from>
                        <to>$DdotIn1.parameters/INCY</to>
                    </copy>
                </assign>
                <forEach name="ForEach2" parallel="no" counterName="counter2">
                    <startCounterValue>    (   ( $inputVar.inputType/N div 2 )   + 1 )
</startCounterValue>

<finalCounterValue>number($inputVar.inputType/N)</finalCounterValue>
                    <scope name="Scope2">
                        <sequence name="Sequence3">
                            <assign name="Assign4">
                                <copy>

<from>$inputVar.inputType/MatrixA[$counter2]</from>
                                        <to>$DdotIn1.parameters/MatrixA[ ( $counter2 -   (
$inputVar.inputType/N div 2 )  ) ]</to>
                                </copy>
                                <copy>

<from>$inputVar.inputType/MatrixB[$counter2]</from>
                                        <to>$DdotIn1.parameters/MatrixB[ ( $counter2 -   (
$inputVar.inputType/N div 2 )  ) ]</to>
                                </copy>
                            </assign>
                        </sequence>
                    </scope>
                </forEach>
                <invoke     name="Invoke2"     partnerLink="Ddot"     operation="ddot"
portType="ns3:DdotService" inputVariable="DdotIn1" outputVariable="DdotOut1"/>
            </sequence>
        </flow>
        <assign name="Assign5">
            <copy>
```

```
                <from>    ( $DdotOut1.parameters/return + $DdotOut.parameters/return )
</from>
                <to>$outputVar.resultType/return</to>
            </copy>
        </assign>
        <reply name="end"
               partnerLink="SynchronousSample"
               operation="operation1"
               portType="ns1:portType1"
               variable="outputVar"/>
    </sequence>
</process>
```

DGBSV flow

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


<process name="SynchronousSample"


targetNamespace="http://enterprise.netbeans.org/bpel/SynchronousSample/SynchronousSamp
le_1"
         xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:bpws="http://docs.oasis-open.org/wsbpel/2.0/process/executable"


xmlns:wsdlNS="http://enterprise.netbeans.org/bpel/SynchronousSample/SynchronousSample_
1"
         xmlns:ns1="http://localhost/SynchronousSample/SynchronousSample"
         xmlns:ns2="http://xml.netbeans.org/schema/SynchronousSample"
xmlns:ns3="http://server/">


    <import namespace="http://xml.netbeans.org/schema/SynchronousSample"
            location="SynchronousSample.xsd"
            importType="http://www.w3.org/2001/XMLSchema"/>
    <import namespace="http://localhost/SynchronousSample/SynchronousSample"
            location="SynchronousSample.wsdl"
            importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import                                                  namespace="http://server/"
location="Partners/DGBSVService/DGBSVService.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/"/>
    <partnerLinks>
        <partnerLink              name="DGBSV"              partnerLinkType="ns3:DGBSV"
partnerRole="DGBSVProvider"/>
        <partnerLink name="SynchronousSample"
                     partnerLinkType="ns1:partnerlinktype1"
                     myRole="partnerlinktyperole1"/>
    </partnerLinks>
```

```
<variables>
    <variable name="DgbsvOut" messageType="ns3:dgbsvResponse"/>
    <variable name="DgbsvIn" messageType="ns3:dgbsv"/>
    <variable name="outputVar" messageType="ns1:responseMessage"/>
    <variable name="inputVar" messageType="ns1:requestMessage"/>
</variables>

<sequence>
    <receive name="start"
             partnerLink="SynchronousSample"
             operation="operation1"
             portType="ns1:portType1"
             variable="inputVar"
             createInstance="yes">
    </receive>
    <assign name="Assign1">
        <copy>
            <from>$inputVar.inputType/N</from>
            <to>$DgbsvIn.parameters/N</to>
        </copy>
        <copy>
            <from>$inputVar.inputType/KL</from>
            <to>$DgbsvIn.parameters/KL</to>
        </copy>
        <copy>
            <from>$inputVar.inputType/KU</from>
            <to>$DgbsvIn.parameters/KU</to>
        </copy>
        <copy>
            <from>$inputVar.inputType/NRHS</from>
            <to>$DgbsvIn.parameters/NRHS</to>
        </copy>
        <copy>
            <from>$inputVar.inputType/A</from>
            <to>$DgbsvIn.parameters/A</to>
        </copy>
        <copy>
            <from>$inputVar.inputType/LDAB</from>
            <to>$DgbsvIn.parameters/LDAB</to>
        </copy>
        <copy>
            <from>$inputVar.inputType/B</from>
```

```
                <to>$DgbsvIn.parameters/B</to>
            </copy>
            <copy>
                <from>$inputVar.inputType/LDB</from>
                <to>$DgbsvIn.parameters/LDB</to>
            </copy>
        </assign>
        <invoke       name="Invoke1"       partnerLink="DGBSV"       operation="dgbsv"
portType="ns3:DGBSV" inputVariable="DgbsvIn" outputVariable="DgbsvOut"/>
        <assign name="Assign2">
            <copy>
                <from>$DgbsvOut.parameters/return</from>
                <to>$outputVar.resultType/return</to>
            </copy>
        </assign>
        <reply name="end"
               partnerLink="SynchronousSample"
               operation="operation1"
               portType="ns1:portType1"
               variable="outputVar"/>
    </sequence>
</process>
```