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

Ανάλυση, Σχεδιασμός και Υλοποίηση Επεκτάσεων Λειτουργικότητας Επικοινωνίας Μεσολογισμικού Πλέγματος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ευάγγελος Σ. Μπέλλος

Επιβλέπων : Θεοδώρα Βαρβαρίγου

Αν. Καθηγήτρια Ε.Μ.Π.

Αθήνα, Ιούλιος 2007

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

Ανάλυση, Σχεδιασμός και Υλοποίηση Επεκτάσεων Λειτουργικότητας Επικοινωνίας Μεσολογισμικού Πλέγματος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ευάγγελος Σ. Μπέλλος

Επιβλέπων : Θεοδώρα Βαρβαρίγου

Αν. Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 24 Ιουλίου 2007.

.....	.....	.....
Θεοδώρα Βαρβαρίγου	Εμμανουήλ Πρωτονοτάριος	Ελευθέριος Καγιάφας

Αθήνα, Ιούλιος 2007

.....

Ευάγγελος Σ. Μπέλλος

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

Περίληψη

Η ανάπτυξη ετερογενών και κατανεμημένων περιβαλλόντων πλέγματος, καθιστά εφικτή την επίλυση υπολογιστικά εντατικών προβλημάτων με αξιόπιστο και οικονομικό τρόπο. Παράλληλα, εντείνει τη ζήτηση για ειδικευμένο μεσολογισμικό το οποίο θα επιτρέπει την πρόσβαση των χρηστών στους πόρους του πλέγματος, αποκρύπτοντας τις ανομοιόμορφες τεχνολογικές τους υλοποιήσεις.

Σκοπός της διπλωματικής εργασίας είναι η μελέτη των τεχνολογιών διαδικτυακών πυλών και μεσολογισμικού πλέγματος, που στοχεύουν στην παροχή ενιαίας, ευέλικτης και εύκολης πρόσβασης των χρηστών στους πόρους του πλέγματος από οποιοδήποτε σημείο.

Αρχικά γίνεται μια εισαγωγή στο υπολογιστικό πλέγμα, με έμφαση στα χαρακτηριστικά και την αρχιτεκτονική του. Στη συνέχεια εξετάζεται το μεσολογισμικό GRIA, ως το πλέον κατάλληλο για την παροχή της λειτουργικότητας και των υπηρεσιών του πλέγματος στον εμπορικό τομέα. Παρουσιάζεται η αρχιτεκτονική των διαδικτυακων πυλών πλέγματος καθώς και τα αποτελέσματα ερευνών σχετικά την κοινή λειτουργικότητα που οφείλουν να παρέχουν. Προκειμένου οι διαδικτυακές πύλες να επικοινωνούν με διαφανή τρόπο με τους ανομοιογενείς πόρους του πλέγματος, απαιτείται ένας μηχανισμός δυναμικής φόρτωσης των συστατικών στοιχείων που επιτρέπουν την πρόσβαση στη λειτουργικότητα ενός συγκεκριμένου μεσολογισμικού, μέσω μιας κοινής ενιαίας διεπαφής. Τα συστατικά αυτά στοιχεία ονομάζονται plug-ins και στα πλαίσια της παρούσας διπλωματικής εργασίας μελετήθηκε η χρήση τους για τη βέλτιστη επικοινωνία των διαδικτυακων πυλών με το μεσολογισμικό GRIA καθώς και τα επιπλέον λειτουργικα χαρακτηριστικα που προσφέρουν ως υπηρεσίες πλέγματος.

Σε συνέχεια των παραπάνω, έγινε μελέτη, σχεδιασμός και υλοποίηση plug-ins για την υποβολή, παρακολούθηση και έλεγχο των εργασιών στο μεσολογισμικό GRIA. Η προαναφερθείσα μελέτη και υλοποίηση ολοκληρώθηκε με τρόπο που να προσφέρει απόκρυψη των λεπτομερειών του συγκεκριμένου μεσολογισμικού, παρέχοντας στο χρήστη την κατάλληλη διεπαφή. Επιπλέον, η υφιστάμενη υλοποίηση επιτρέπει τη διαχείριση των Συμβολαίων Παροχής Υπηρεσιών και των λογαριασμών των χρηστών.

Λέξεις Κλειδιά

Πλέγμα, μεσολογισμικό, Grid Portal, GRIA, plug-in, Υπηρεσιοστρεφής Αρχιτεκτονική, Αρχιτεκτονική Ανοιχτών Υπηρεσιών, πόροι, εργασία, Συμβόλαιο Παροχής Υπηρεσιών, Trade Account, διεπαφή.

Abstract

The development of heterogeneous and distributed Grid environments renders the solution of computer-intensive problems viable in a reliable and economical way. Thus, there has arisen a need for specialized middleware, which will allow users access to Grid resources, while hiding their technologically disparate implementations.

The scope of this thesis is the study of the middleware and portal technology aiming to provide uniform, flexible and intuitive user access to Grid resources from anywhere.

In the beginning, there is an introduction to Grid Computing with emphasis on its main characteristics and its architecture. Then, the GRIA middleware is examined as the most suitable for providing Grid functionality and services to the commercial sector. The Grid Portal architecture is presented along with studies regarding the common functionality the Portals should support. In order to provide Grid Portals with transparent access to disparate Grid resources, there is a mechanism enabling the dynamic loading of components that provide access to the functionality of a specific Grid middleware, via a uniform interface. These components are called plug-ins and their use in optimizing communication between Grid Portals and the GRIA middleware, as well as their additional Grid Service functionality, were examined in the course of this thesis.

Based on the above, specific plug-ins have been analyzed, designed, implemented and deployed facilitating the submission, monitoring and control of jobs in the GRIA middleware. The aforementioned analysis was completed with regard to concealing middleware-specific details, while providing the necessary user interface. Furthermore, the current schema and implementation enables the management of Service Level Agreements and Trade Accounts.

Keywords

Grid, middleware, Grid Portal, GRIA, plug-in, Service Oriented Architecture, Open Grid Services Architecture, resources, job, Service Level Agreement, Trade Account, interface.

Πίνακας Περιεχομένων

Πίνακας Σχημάτων

Πίνακας Πινάκων

1 Εισαγωγή

Στο παρόν κεφάλαιο παρατίθεται αρχικά ο ορισμός του Πλέγματος (“Grid”) και στη συνέχεια αναλύεται η οργάνωση της παρούσης Διπλωματικής Εργασίας και περιγράφεται συνοπτικά το περιεχόìενο των κεφαλαίων.

1.1 Ορισμός του Πλέγματος

Η τεχνολογία των υπολογιστών και των τηλεπικοινωνιών γνωρίζει την κορύφωση τα τελευταία χρόνια καθώς αναπτύσσονται συνεχώς εφαρμογές που χαρακτηρίζονται από εξαιρετικές απαιτήσεις σε υπολογιστική ισχύ και χρόνο εκτέλεσης. Παράλληλα, εντείνεται η ζήτηση για πρόσβαση σε πληροφορίες σε κάθε γεωγραφικό σημείο ανεξάρτητα του μέσου γεγονός που αναδεικνύει κυρίαρχη την ανάγκη για τη διασύνδεση των κατανεμημένων πόρων και υποδομών με ηλεκτρονικά δίκτυα (electronic networks) και ειδικευμένο μεσολογισμικό (middleware) το οποίο θα επιτρέπει την εύκολη και φιλική πρόσβαση των χρηστών αποκρύπτοντας τις ετερογενείς τεχνολογικές υλοποιήσεις των προαναφερθέντων πόρων.

Το κατανεμημένο αυτό περιβάλλον που επιτρέπει το διαμοιρασμό και την από κοινού χρήση υπολογιστικών, αποθηκευτικών και άλλων πόρων, με τη συνδρομή του μεσολογισμικού, ονομάζεται *Πλέγμα Υπολογιστικών Συστημάτων* ή απλά Grid. Η ενοποίηση των δικτύων και του μεσολογισμικού σε μια ενιαία υποδομή με στόχο την κατανεμημένη αλλά ομογενή πρόσβαση στους πόρους του Grid αναφέρεται ως *Ηλεκτρονική Υποδομή* (e-Infrastructure).

Τα τελευταία χρόνια, η ραγδαία εξάπλωση του διαδικτύου (Internet) σε συνδυασμό με τη διαθεσιμότητα δικτύων κορμού υψηλών ταχυτήτων και τη τεχνολογική ανάπτυξη των υπολογιστών αλλά και του λογισμικού, έχουν δημιουργήσει μια νέα δυναμική στην κλασσική έννοια του όρου «υπολογιστικό περιβάλλον». Στην παρούσα φάση, σημαντικός αριθμός υπολογιστικών πόρων, οι οποίοι περιλαμβάνουν υπολογιστική ισχύ, δεδομένα, υπηρεσίες, εργαλεία λογισμικού, επιστημονικά όργανα, κ.α., βρίσκονται κατανεμημένοι σε παγκόσμιο επίπεδο, δημιουργώντας την ανάγκη για ασφαλή, ομοιόμορφη, αξιόπιστη και

απομακρυσμένη πρόσβαση μέσω δικτύων ώστε να αξιοποιηθούν ικανοποιητικά οι δυνατότητες που αυτοί παρέχουν.

Τα Πλέγματα (Grids) είναι μια προσέγγιση της σύστασης δυναμικά δομημένων περιβαλλόντων, χρησιμοποιώντας υπολογιστικούς πόρους που είναι διεσπαρμένοι τόσο γεωγραφικά όσο και οργανωτικά. Ο όρος Grid περιλαμβάνει το σύνολο της υποδομής (υλικό και λογισμικό) και των απαραίτητων υπηρεσιών για τη δημιουργία ενός ενιαίου (γεωγραφικά διεσπαρμένου) υπερ-υπολογιστικού περιβάλλοντος.

Καθώς υπάρχουν πολλοί ορισμοί που όμως αλληλοσυμπληρώνονται και δεν αλληλοαναιρούνται, αναφέρουμε στη συνέχεια τον τεχνικό ορισμό του Grid έτσι όπως δίνεται από την IBM:

Grid είναι η δυνατότητα, με τη χρήση ενός συνόλου από ανοικτά πρότυπα και πρωτόκολλα, της απόκτησης πρόσβασης σε εφαρμογές, δεδομένα, επεξεργαστική ισχύ, χώρο αποθήκευσης δεδομένων και μίας τεράστιας ποικιλίας από υπολογιστικούς πόρους που διατίθενται στο Internet. Το Grid είναι ένα είδος παράλληλου και κατανεμημένου συστήματος που δίνει τη δυνατότητα να μοιραζόμαστε, να επιλέγουμε και να συγκεντρώνουμε πόρους που κατανέμονται σε πολλαπλές administrative domains βασιζόμενοι στην διαθεσιμότητα των πόρων τους, την χωρητικότητα, την επίδοση, το κόστος και σε απαιτήσεις Ποιότητας Υπηρεσίας (Quality of Service) που καθορίζονται από το χρήστη.

Από τα παραπάνω, γίνεται φανερό ότι ο όρος Grid περιλαμβάνει το σύνολο της υποδομής, υλικό και λογισμικό, κατάλληλα διασυνδεόμενων μέσω δικτύων υψηλών ταχυτήτων, καθώς και των απαραίτητων υπηρεσιών για τη δημιουργία ενός ενιαίου υπερ-υπολογιστικού περιβάλλοντος, που αν και είναι γεωγραφικά διεσπαρμένο, εμφανίζεται με τρόπο διαφανή σε όλους τους χρήστες του. Αποτελεί ένα ενιαίο σύνολο υπολογιστικών πόρων, μια συμπαγή - αν και κατανεμημένη - υπολογιστική πλατφόρμα. Το Grid διασυνδέει ετερογενή υπολογιστικά περιβάλλοντα, με όμοια ή διαφορετική φιλοσοφία και υπηρεσίες, δημιουργώντας επιπλέον νέα σύνολα υπηρεσιών με αυξημένες υπολογιστικές δυνατότητες και νέους τρόπους αξιοποίησης των ποικίλων πόρων τους οποίους διαμοιράζει.

1.2 Οργάνωση του εγγράφου

Το παρόν έγγραφο αποτελείται από επτά (7) κεφάλαια και ένα παράρτημα. Στις ενότητες των κεφαλαίων αυτών παρουσιάζεται ουσιαστικά και με αναλυτικό τρόπο το αντικείμενο της διπλωματικής εργασίας και η διαδικασία εκπόνησης της.

Το κεφάλαιο 2 παρέχει γενικές πληροφορίες για τα περιβάλλοντα πλέγματος όπως ιστορικά στοιχεία, όρους, χαρακτηριστικά, ταξινόμηση και πλεονεκτήματα αυτών.

Το τρίτο (3) κεφάλαιο περιλαμβάνει πληροφορίες για το μεσολογισμικό GRIA. Αρχικά παρατίθεται ο ορισμός του και κάποια ιστορικά στοιχεία σχετικά με αυτό, ενώ στη συνέχεια γίνεται εκτενής αναφορά στην αρχιτεκτονική του.

Στο επόμενο κεφάλαιο (κεφάλαιο 4) παρουσιάζονται τα Grid Portals με έμφαση στη διαφανή διασύνδεση τους με τους πόρους του Grid. Επισημαίνεται η απαίτηση για τη χρήση plug-ins προκειμένου να διασυνδεθούν τα Portals με τα μεσολογισμικά του Grid. Τέλος παρουσιάζεται η αρχιτεκτονική των Grid Portals και αναλύεται η κοινή τους λειτουργικότητα.

Το κεφάλαιο 5 παρουσιάζει την μελέτη, το σχεδιασμό και την υλοποίηση συγκεκριμένων plug-ins για τη διασύνδεση των Grid Portals με το μεσολογισμικό GRIA. Παρουσιάζεται συνοπτικά η εγκατάσταση και ρύθμιση του λογισμικού που χρησιμοποιήθηκε, ενώ στη συνέχεια γίνεται αναλυτική περιγραφή των λειτουργιών των plug-ins και των αντίστοιχων διεπαφών τους. Στο τέλος του κεφαλαίου αναφέρονται τα προβλήματα που ανέκυψαν κατά την υλοποίηση των plug-ins και πως αυτά αντιμετωπίστηκαν.

Στο κεφάλαιο 6 παρουσιάζονται κάποια συμπεράσματα που εξήχθησαν κατά την εκπόνηση της διπλωματικής εργασίας καθώς και πιθανές μελλοντικές εξελίξεις και βελτιώσεις.

Το κεφάλαιο 7 περιλαμβάνει τις Βιβλιογραφικές Αναφορές, ενώ στο παράρτημα παρατίθεται ο πηγαίος κώδικας των υλοποιηθείσων διεπαφών και plug-ins.

2 *Γενικά για το περιβάλλον Πλέγματος*

Η χρήση ηλεκτρονικών υπολογιστών σε περιβάλλον πλέγματος [1] θεωρείται ολοένα και συχνότερα ως υποδομή νέας γενιάς με δυνατότητα παροχής κατανεμημένων και ετερογενών πόρων με στόχο την παροχή υπολογιστικής ισχύος σε εφαρμογές με υψηλές απαιτήσεις σε πόρους, με διαφανή τρόπο [2], [3]. Στο κεφάλαιο αυτό παρουσιάζονται θέματα για το περιβάλλον πλέγματος όπως ιστορικά στοιχεία, όροι, ταξινόμηση και πλεονεκτήματα από τη χρήση αυτού.

2.1 Ιστορικά στοιχεία - Από το Διαδίκτυο στο Πλέγμα

Η έρευνα για το Grid ξεκίνησε στους ακαδημαϊκούς χώρους - έστω και στα πλαίσια απλών συζητήσεων - πολλά χρόνια πριν ο επιχειρησιακός κόσμος αντιληφθεί τις δυνατότητες, που θα του πρόσφερε η μετάβαση σε ένα σύστημα κατανεμημένων υπολογισμών. Όμως οι αρχικές ιδέες που σχετίζονταν με τον όρο Grid, το αντιμετώπιζαν διαφορετικά απ' ότι οι σημερινές.

Πανεπιστήμια και ερευνητικά ιδρύματα είναι παραδοσιακά αυτοί που εισάγουν νέες ιδέες και προοπτικές στα θέματα που αφορούν τις υπολογιστικές υποδομές. Αυτό ισχύει και στην περίπτωση του διαδικτύου (Internet), πρώιμη μορφή του οποίου αποτελεί το *ARPANET* που αναπτύχθηκε στις ΗΠΑ στη δεκαετία του 1960 από την ερευνητική ομάδα ARPA (Advanced Research Projects Agency), και το δίκτυο (web) το οποίο αναπτύχθηκε στο ευρωπαϊκό ερευνητικό εργαστήριο του CERN. Στη συνέχεια η επιστημονική κοινότητα ήταν πάλι εκείνη η οποία πρώτη ανακάλυψε τα πλεονεκτήματα των νέων τεχνολογιών καθώς και τρόπου χρησιμοποίησής τους για την επίλυση προβλημάτων.

Υπάρχει πληθώρα τέτοιων προβλημάτων τα οποία μπορούν να επιλυθούν πολύ καλύτερα με τη βοήθεια της προηγμένης τεχνολογίας υπολογιστών. Πολλά από αυτά τα προβλήματα έχουν το κοινό χαρακτηριστικό ότι είναι πολύ απαιτητικά σε υπολογιστική ισχύ. Αυτό σημαίνει ότι όσο πιο μεγάλη υπολογιστική ισχύ διαθέτει κάποιος, τόσο πιο ακριβής είναι η απάντηση στο πρόβλημα του οποίου επιχειρείται η επίλυση. Ισχύει ακόμη ότι ορισμένα προβλήματα δε μπορούν καν να επιλυθούν χωρίς επαρκή υπολογιστική δύναμη. Τα

τελευταία 40 χρόνια της εξέλιξης των υπολογιστικών δομών, το θέμα των υπολογιστικά απαιτητικών εργασιών αποτελεί παράγοντα που δίνει ώθηση στην εξέλιξη αυτή.

Η ανάγκη για την επίλυση των προβλημάτων αυτών οδήγησε στην ανάπτυξη των υπολογιστών και στη συνέχεια των υπερ-υπολογιστών. Οι μεγάλοι υπολογιστικοί πόροι της IBM της δεκαετίας του 1960 και οι υπερ-υπολογιστές Cray στις δεκαετίες 1970 και 1980 κυριαρχούσαν στα υπολογιστικά κέντρα για πάνω από δύο δεκαετίες. Στη συνέχεια επικράτησαν οι αρχιτεκτονικές συμμετρικής πολυεπεξεργασίας SMP και μαζικής παράλληλης επεξεργασίας MPP που περιλάμβαναν πολλούς επεξεργαστές που είχαν τη δυνατότητα να λειτουργούν παράλληλα και για την αποτελεσματικότερη χρήση τους οι προγραμματιστές εφαρμογών έπρεπε να δομήσουν των κώδικά τους έτσι ώστε να επιτρέπει την παράλληλη εκτέλεσή του. Έπειτα εμφανίστηκαν τα clusters, δηλαδή πολλοί υπολογιστές διασυνδεδεμένοι μεταξύ τους και κατάφεραν να διαδοθούν χάρη στο χαμηλότερο κόστος τους. Έκαναν χρήση προγραμματιστικών μοντέλων όπως το MPI (Message Passing Interface) και το PVM (Parallel Virtual Machine), κατανεμημένου συστήματος αρχείων όπως το NFS (Network File System) και γρήγορων δικτυακών συνδέσεων όπως Ethernet και Myrinet, έτσι ώστε να αποφεύγεται η μείωση της απόδοσης λόγω αργής επικοινωνίας μεταξύ των υπολογιστών που αποτελούν το cluster.

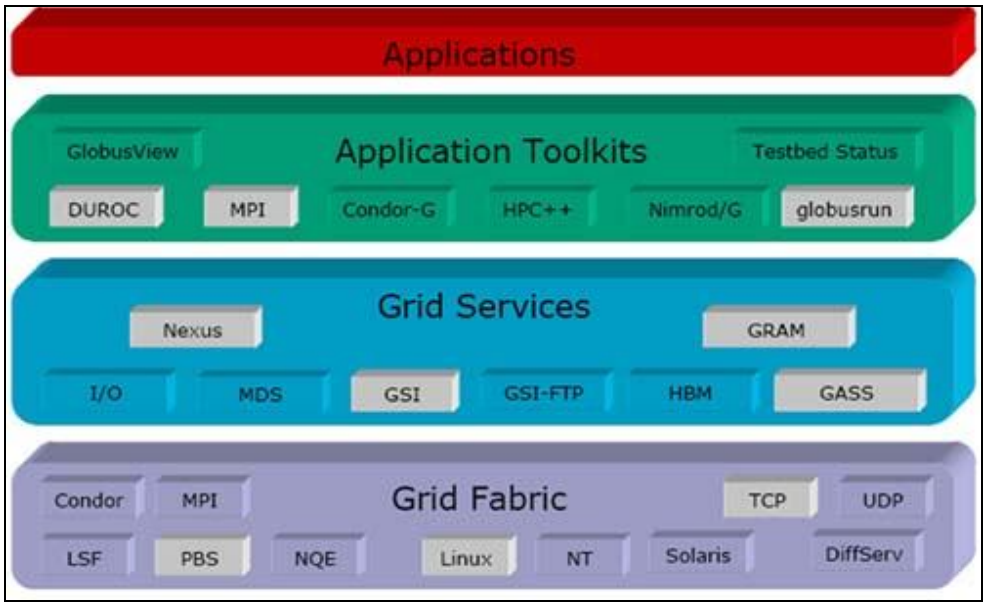
Η αρχιτεκτονική των cluster συστημάτων παρουσίαζε ορισμένα βασικά προβλήματα στην περίπτωση της επικοινωνίας μεταξύ clusters διαφορετικών ιδρυμάτων:

- Τα clusters τυπικά αποτελούνται από ίδιες ή παρόμοιες μηχανές. Δύο clusters διαφορετικών αρχιτεκτονικών ήταν δύσκολο να ενωθούν σε ένα ενιαίο σύστημα. Διαφορές στους compilers, τα εργαλεία, τις βιβλιοθήκες και τη δομή των αρχείων δύο διαφορετικών λειτουργικών συστημάτων θα αποτελούσαν προβλήματα για την ενοποίηση.
- Τυπικές παράμετροι για δικτυακές συνδέσεις μεγάλων αποστάσεων απεδείχθησαν μη ικανοποιητικές για την τεχνολογία των clusters. Καθυστέρηση, χαμηλή ρυθμαπόδοση (throughput), τυχαίες συμφορήσεις του δικτύου (τυπικό για το TCP/IP) και σφάλματα δικτύου ήταν όλοι παράγοντες που συνέβαλαν σε αυτό.
- Δεν υπήρχε η απαιτούμενη τεχνολογία για να αντιμετωπιστούν τα προβλήματα ασφαλείας που προέκυπταν. Συναρτήσεις με χαμηλή ασφάλεια, που ήταν κατάλληλες για εσωτερικά συστήματα που συνήθως προστατεύονταν από firewalls, ήταν τελείως ακατάλληλες για περιβάλλον ανοικτού δικτύου.
- Θέματα διαχείρισης και πολιτικής έδωσαν επίσης μια νέα διάσταση στο πρόβλημα. Τα διάφορα ιδρύματα έπρεπε να καθορίσουν τις συνθήκες κάτω από τις οποίες θα μοιράζονταν τους πόρους τους μέσω του δικτύου. Ακόμη όμως και αν καθορίζονταν μια τέτοια πολιτική, με την υπάρχουσα τεχνολογία δεν υπήρχαν τα απαραίτητα τεχνικά μέσα για τον αποτελεσματικό έλεγχό και επιβολή της.

Η ερευνητική δραστηριότητα στον τομέα των δικτύων υπολογιστών συνεχίστηκε, με αποτέλεσμα την δημιουργία του *NSFNET* [1986], δικτύου στα 56Kbps που συνέδεε τα πέντε NSF κέντρα υπερ-υπολογιστών. Ως συνέχεια και εξέλιξη αυτών των τεχνολογιών

μπορούμε να θεωρήσουμε το πρόγραμμα Condor [1988] του πανεπιστημίου του Wisconsin. Το σύστημα αυτό είναι ένας 'διαχειριστής φόρτου εργασίας' (workload manager), με δυνατότητες παρακολούθησης και διαχείρισης πόρων, δρομολόγησης εργασιών και αποτελεί το πρώτο πρόγραμμα με κατεύθυνση προς την αξιοποίηση των Grid υπηρεσιών.

Η ανάπτυξη δικτύων υψηλών ταχυτήτων και η ανάγκη για μεγάλη επεξεργαστική ισχύ οδήγησε σε έντονη ερευνητική δραστηριότητα στον τομέα των Grid τεχνολογιών. Η έρευνα αυτή κατέληξε σε ενδιαφέροντα αποτελέσματα με πιο σημαντικά τα προγράμματα LEGION [1993], SRB [1997], GLOBUS [1998]. Το πρώτο βασίζεται στην ιδέα του 'εικονικού υπολογιστή (virtual computer): όλοι οι πόροι συνδεδεμένοι μεταξύ τους, εμφανίζονται στον χρήστη ως μία εικονική μηχανή, με αρκετά μειονεκτήματα όμως, όπως η πολύπλοκη υλοποίηση και η μικρή αποδοτικότητα. Το SRB (Storage Resource Broker) ήταν μια πλατφόρμα διαχείρισης αποθηκευτικών πόρων που βοήθησε πολύ στην ανάπτυξη των Grid τεχνολογιών, αφού αντιμετώπισε τα προβλήματα μεταφοράς δεδομένων σε Grid περιβάλλον. Τέλος, το πιο διαδεδομένο σύστημα διαχείρισης Grid υπηρεσιών είναι το GLOBUS, που αναπτύχθηκε στο Argonne National Lab στο πανεπιστήμιο του Berkeley. Το GLOBUS προτυποποίησε πρωτοκόλλα για την ασφάλεια, μεταφορά δεδομένων, ανακάλυψη πόρων και εκτέλεση εργασιών. Λειτουργεί σε χαμηλό επίπεδο και είναι ανεπτυγμένο σε επίπεδα υπηρεσιών, όπως φαίνεται παρακάτω και στο Σχήμα 1:



Σχήμα 1: Αρχιτεκτονική Globus

Η συνέχεια γίνεται με την ανάπτυξη των Web Services [2001] , υπηρεσιών που είναι προσβάσιμες μέσω του διαδικτύου και χρησιμοποιούν συγκεκριμένα πρωτόκολλα περιγραφής (XML, SOAP, WSDL) και τέλος με την εγκαθίδρυση της Αρχιτεκτονικής Ανοιχτών Υπηρεσιών Grid (Open Grid Service Architecture - OGSA) [2002] ως κύριας αρχιτεκτονικής της Grid τεχνολογίας (όπως περιγράφεται στην ενότητα 2.7). Το πρότυπο αυτό είναι το πιο σημαντικό για τις τεχνολογίες του Grid, αφού περιγράφει τις δυνατότητες του συστήματος αυτού να αναλύσει την λειτουργία του σε όλα τα επίπεδά του.

2.2 Χαρακτηριστικά

Όπως έχει ήδη αναφερθεί, τα Grids ενοποιούν μέσω ηλεκτρονικών δικτύων υπολογιστικούς, αποθηκευτικούς και άλλους πόρους κατανεμημένους σε τοπική, εθνική και διεθνή κλίμακα. Βάσει αυτού, διακρίνονται με τα εξής χαρακτηριστικά:

- Επιτρέπουν το διαμοιρασμό των πόρων σε πολλαπλούς χρήστες διαφορετικών κοινοτήτων με ετερογενή πεδία εφαρμογών και γεωγραφική κατανομή. Ένα Grid μπορεί να στηρίζεται είτε σε ένα τοπικό δίκτυο (campus LAN), είτε σε μητροπολιτικό δίκτυο MAN, σε εθνικής εμβέλειας δίκτυο (WAN) ή και διεθνούς κάλυψης δίκτυο όπως το Ευρωπαϊκό Ερευνητικό Δίκτυο GEANT και το Αμερικανικό Abilene ανάλογα με τις απαιτήσεις των εφαρμογών και τις υπάρχουσες δικτυακές υποδομές.
- Ενοποιούν μέσω δικτύων Internet / Intranet υπολογιστικές, αποθηκευτικές και άλλες ηλεκτρονικές εγκαταστάσεις με ετερογενείς τεχνολογικές υλοποιήσεις με στόχο την παροχή ολοκληρωμένων Ηλεκτρονικών Υπηρεσιών (e-Services). Η ενοποίηση υλοποιείται με χρήση ενός επιπρόσθετου στρώματος μεσολογισμικού που αναλαμβάνει το διαμοιρασμό των πόρων πάνω από το δίκτυο με τα παραπάνω χαρακτηριστικά.
- Απαιτούν ασφαλή πρόσβαση μέσω μεσολογισμικού με έμφαση στο λογισμικό ανοικτού κώδικα - open source (π.χ. GLOBUS). Τα Grids επεκτείνουν την φιλοσοφία του ανοικτού λογισμικού σε ανοικτά υπολογιστικά συστήματα, με περιορισμούς μόνο όσο αφορά την ασφάλεια και την διαθεσιμότητα πόρων για την κάλυψη συγκεκριμένων αναγκών.
- Παρουσιάζουν μεγάλη δυνατότητα κλιμάκωσης, με ιδιαίτερα περιορισμένη αρχική επένδυση. Οι αρχιτεκτονικές Grid μπορεί να αποτελέσουν σημαντικό εργαλείο για την υπέρβαση του ψηφιακού χάσματος στον κόσμο, σε μια ήπειρο, σε μία χώρα (κέντρο - περιφέρεια), σε έναν οργανισμό (campus).

2.3 Οι Γενιές των Συστημάτων Πλέγματος

Βάσει της ιστορίας και της εξέλιξης των Grids, διακρίνονται οι ακόλουθες γενιές αυτών:

- Σύμφωνα με τον Charlie Catlett, Πρόεδρο του Global Grid Forum - που πλέον ονομάζεται Open Grid Forum [13] - η πρώτη γενιά Grids (1st Generation Grids ή ***1G Grids***) ουσιαστικά αποτελούνταν από τοπικούς "μετα-υπολογιστές" (metacomputers) με βασικές λειτουργίες όπως το κατανεμημένο σύστημα αρχείων (sitewide single sign-on), δηλαδή μοναδικό σημείο όπου ο χρήστης δίνει τα προσωπικά στοιχεία του (π.χ. user/password), πάνω στις οποίες χτίστηκαν νέες

κατανεμημένες εφαρμογές με ειδικά προσαρμοσμένα δικτυακά πρωτόκολλα. Με την υλοποίηση Gigabit test-beds τα 1G Grids επεκτάθηκαν και έγινε προσπάθεια δημιουργίας "μετα-κέντρων" (metacenters), τα οποία διερεύνησαν θέματα ολοκλήρωσης μεταξύ διαφορετικών κέντρων. Γενικά τα Grids πρώτης γενιάς ήταν εντελώς προσαρμοσμένα στα συγκεκριμένα πειράματα και αποτέλεσαν απόδειξη της ιδέας (proof-of-concept).

- Τα συστήματα 2ης γενιάς Grids, (***2G Grids***), ξεκίνησαν με προγράμματα όπως το Condor, το I-WAY (που αποτέλεσε την αρχή του Globus) και το Legion (που αποτέλεσε την αρχή του Avaki), όπου νέες υπηρεσίες μεσολογισμικού και πρωτοκόλλων επικοινωνιών αποτέλεσαν τη βάση για την ανάπτυξη κατανεμημένων εφαρμογών και υπηρεσιών. Τα Grids 2ης γενιάς ουσιαστικά διαμόρφωσαν τα βασικά δομικά στοιχεία, αλλά η χρήση τους απαιτούσε εκτενές "customization" και ποικίλες εργασίες για να καλυφθούν σημαντικά κενά. Οι ανεξάρτητες αυτές προσπάθειες χρήσης συστημάτων 2ης γενιάς που περιείχαν πολλές "κατά απαίτηση" επεκτάσεις λογισμικού, κατέστησε την διαλειτουργικότητα προβληματική.
- Λαμβάνοντας υπόψη, τόσο την πρότερη εμπειρία από τις 2 πρώτες γενιές, όσο και τις τεχνολογίες των πολύ επιτυχημένων υπηρεσιών web, έχουν ξεκινήσει οι προσπάθειες για την 3η γενιά Grids (***3G Grids***), που βασίζονται στην Αρχιτεκτονική Ανοιχτών Υπηρεσιών Grid (όπως περιγράφεται στην ενότητα 2.7), όπου μια σειρά από προδιαγραφές κοινών και ανοιχτών διεπαφών υποστηρίζουν τη διαλειτουργικότητα ανεξάρτητα ανεπτυγμένων υπηρεσιών. Η πρόσφατα εκδοθείσα προδιαγραφή Open Grid Services Infrastructure - OGSI) είναι ο θεμέλιος λίθος της παραπάνω αρχιτεκτονικής. Με την εισαγωγή προτυποποιημένων τεχνικών προδιαγραφών, η 3η γενιά Grid θα επιταχύνει τον ανταγωνισμό και την επίτευξη διαλειτουργικότητας όχι μόνο μεταξύ εφαρμογών και εργαλειοθηκών, αλλά κυρίως μεταξύ διαφορετικών υλοποιήσεων βασικών υπηρεσιών του Grid.

2.4 Όροι για το Πλέγμα

2.4.1 Το meta-computing

Το meta-computing εμφανίστηκε ως η προσπάθεια για την αποδοτική σύνδεση και συγκέντρωση της υπολογιστικής δύναμης που βρισκόταν σε διάφορα μέρη του κόσμου. Ο όρος προέρχεται από το γεγονός ότι στα ακαδημαϊκά συστήματα η γνώση για το είδος και την ισχύ των μηχανημάτων αποθηκεύονταν σε έναν κεντρικό meta-υπολογιστή. Οι προσπάθειες συγκεντρώνονταν στην επίλυση υπολογιστικά απαιτητικών προβλημάτων, η οποία διευκολύνονταν από τη συνδυασμένη υπολογιστική ισχύ πολλών επεξεργαστών. Για

κάποιο διάστημα το meta-computing θεωρούνταν συνώνυμο του Grid-computing. Όμως τα διάφορα πληροφοριακά συστήματα δε χρησιμοποιούν μόνο επεξεργαστές. Κάθε εργασία έχει είσοδο και έξοδο. Υπάρχουν δεδομένα τα οποία επεξεργάζονται και τα οποία ίσως χρειάζεται να αποθηκευτούν στο τέλος και μάλιστα σε χώρο εκτός του υπολογιστή όπου γίνεται η επεξεργασία. Μπορεί το σύστημα να χρειαστεί να συνεργαστεί με κάποια βάση δεδομένων ή με εξειδικευμένο υλικό (hardware) το οποίο ανήκει σε κάποιο άλλο οργανισμό από αυτόν που θέλει να εκτελέσει την εργασία. Εν τέλει είναι δυνατόν οι πόροι να μην είναι μόνο υπολογιστικοί, αλλά και πόροι λογισμικού, εξειδικευμένο υλικό κλπ. Βλέπουμε λοιπόν ότι το meta-computing αντιμετώπισε μόνο ένα μέρος του προβλήματος.

2.4.2 Μη υπολογιστικοί πόροι

Όπως είπαμε προηγουμένως οι πόροι μπορεί να μην είναι μόνο υπολογιστικοί. Γενικά ως πόρος θεωρείται οποιοδήποτε στοιχείο της δικτυωμένης υποδομής το οποίο διατίθεται προς χρήση μέσω καθορισμένων πρωτοκόλλων Grid. Έτσι ακόμη και τα δίκτυα θεωρούνται ως πόροι οι οποίοι προσφέρουν εύρος ζώνης (bandwidth) για τη μεταφορά δεδομένων. Οι χώροι αποθήκευσης δεδομένων και οι βάσεις δεδομένων θεωρούνται ως πόροι όταν χρησιμοποιούν τυποποιημένες διασυνδέσεις για να επιτρέψουν σε εφαρμογές Grid να διατηρήσουν δεδομένα. Επίσης είναι δυνατόν να υπάρχουν πόροι λογισμικού, γιατί μπορεί να υπάρχουν προγράμματα που βρίσκονται σε συγκεκριμένες τοποθεσίες και μπορούν να τρέξουν μόνο σε εξειδικευμένο υλικό ή τίθεται θέμα του ποιος είναι εξουσιοδοτημένος να τα χρησιμοποιήσει. Κάθε πόρος διαθέτει ορισμένα χαρακτηριστικά τα οποία τον καθιστούν μοναδικό. Τα κυριότερα είναι:

- *Απόδοση*. Είναι δυνατόν υπολογιστικοί κόμβοι να διαφέρουν ως προς τον αριθμό και την ταχύτητα των επεξεργαστών, όπως επίσης και την ποσότητα και ταχύτητα της τοπικής μνήμης. Από την άλλη δίαυλοι δικτύου μπορεί να διαφέρουν ως προς το εύρος ζώνης και την καθυστέρηση.
- *Αρχιτεκτονική*. Για παράδειγμα, αναλόγως της αρχιτεκτονικής ενός επεξεργαστή μπορεί να είναι ή να μην είναι δυνατόν να τρέξει ένα πρόγραμμα σε αυτόν.
- *Ποιότητα υπηρεσίας* (Quality of Service). Μπορεί για κάποιο χρονικό διάστημα ένα δίκτυο να μην είναι σε θέση να εγγυηθεί την ελάχιστη διαθέσιμη ρυθμαπόδοση σε ένα χρήστη.
- *Αξιοπιστία*. Για παράδειγμα, υλικό όπως οι σκληροί δίσκοι χαρακτηρίζονται από το TTF (μέσος χρόνος δραστηριότητας χωρίς σφάλμα).
- *Διαθεσιμότητα*. Εξαιτίας βλάβης της επικοινωνίας στην περίπτωση των απομακρυσμένων πόρων, η σύνδεση με αυτούς μπορεί να μην είναι δυνατή συνεχώς.
- *Δυνατότητες*. Για παράδειγμα, κάθε εφαρμογή συνήθως είναι δυνατόν να εκτελεί συγκεκριμένες λειτουργίες και αλγορίθμους, που καμία άλλη εφαρμογή να μη μπορεί.

2.4.3 Εικονοποίηση

Καθώς εξελισσόταν το Grid κατέστη σαφές ότι θα πρέπει να υπάρχει ένα στρώμα υπηρεσιών (service layer) το οποίο θα επέτρεπε στους χρήστες και τις εφαρμογές τους να αιτούνται τη χρήση πόρων χωρίς να γνωρίζουν την ακριβή δομή αυτών. Για παράδειγμα, κάποιος χρήστης θα μπορούσε να ζητήσει μέσω της αντίστοιχης υπηρεσίας την εκτέλεση μιας εργασίας χωρίς να χρειάζεται να ξέρει πως αυτή θα εκτελεστεί (αν θα εκτελεστεί για παράδειγμα από έναν πανίσχυρο επεξεργαστή ή 10 πιο αδύναμους). Το γεγονός αυτό ονομάζεται εικονοποίηση (virtualization) και έχει τα εξής πλεονεκτήματα:

- Καλύτερη κατηγοριοποίηση και παρουσίαση των δυνατοτήτων που βασίζεται περισσότερο στις πραγματικές ανάγκες και όχι σε φυσικούς περιορισμούς (χρειάζομαι 60 επεξεργαστές για μια ώρα, δε με νοιάζει σε πόσα δωμάτια βρίσκονται).
- Πιο αποτελεσματική χρήση των συνηθισμένων πόρων.
- Κοινή πρόσβαση σε σπάνιους πόρους με μοναδικές ικανότητες όπως εξειδικευμένες εφαρμογές και υλικό (hardware).
- Δε χρειάζεται να καθορίζει η εφαρμογή τον τρόπο με τον οποίο η εργασία θα κατανέμεται στους πόρους.

2.4.4 Υπηρεσίες δικτύου

Η προσπάθεια επικοινωνίας με απομακρυσμένους πόρους και χρήσης αυτών, το οποίο αποτελεί βασικό στοιχείο του Grid computing, στηρίχθηκε κατά καιρούς σε διάφορες τεχνολογίες όπως RPC (Remote Procedure Call), CORBA, COM/DCOM, και RMI. Για την επίτευξη όμως αποτελεσματικής επικοινωνίας μεταξύ διαφορετικών συστημάτων χρειάστηκε η εμφάνιση μιας νέας τεχνολογίας με το όνομα ‘υπηρεσίες δικτύου (Web Services).

Μία υπηρεσία δικτύου είναι μια καλά καθορισμένη συνάρτηση η οποία είναι προσπελάσιμη μέσω του διαδικτύου. Η επικοινωνία με αυτή γίνεται μέσω της τεχνολογίας XML και του πρωτοκόλλου SOAP που βασίζεται σε αυτή. Η τεχνολογία των υπηρεσιών δικτύου καθορίζει μια κοινή πλατφόρμα επικοινωνίας με την οποία μπορούν να δημιουργηθούν διασυνδέσεις σε διάφορες συναρτήσεις. Η ύπαρξη ενός κοινού συντακτικού (XML) επιτρέπει στις υπηρεσίες να απαλλαχθούν από τις τυπικές ενέργειες που χρειάζονται για την επικοινωνία. Αυτές τις αναλαμβάνουν πλέον οι εκάστοτε εξυπηρετητές που φιλοξενούν τις υπηρεσίες. Έτσι οι υπηρεσίες μπορούν πλέον να γραφτούν εύκολα, να αντικατασταθούν, να αναπτυχθούν και να συντηρηθούν.

Το γεγονός ότι οι πρώτες υπηρεσίες υστερούσαν σε αξιοπιστία, ασφάλεια και απόδοση, όπως επίσης και σε λειτουργικότητα υψηλότερου επιπέδου, είχε ως αποτέλεσμα την εμφάνιση διαφόρων τεχνολογιών που έλυναν τα προβλήματα αυτά. Έτσι οδηγηθήκαμε στην εμφάνιση των WS-Security και του WSRF (WS-Resource Framework). Το τελευταίο

χρησιμοποιείται για τη δημιουργία υπηρεσιών δικτύου οι οποίες προσφέρουν στους μετέχοντες στο Grid πρόσβαση σε πόρους. Μία υπηρεσία δικτύου μπορεί να αντιστοιχεί σε πολλούς πόρους (WS-Resources), ενώ ένα πόρος μπορεί να είναι προσπελάσιμος από διάφορες υπηρεσίες δικτύου. Οι πόροι αυτοί είναι πλέον δυναμικοί, δηλαδή οι ιδιότητές τους μπορούν να αλλάζουν. Μπορούν να δημιουργούνται και να καταστρέφονται κατ' απαίτηση και η πρόσβαση σε αυτούς είναι δυνατή μόνο μέσω μιας δικτυακής υπηρεσίας.

2.4.5 Εικονικοί Οργανισμοί

Το ακριβές πρόβλημα το οποίο υποκινεί την ανάπτυξη του Grid είναι ο ελεγχόμενος και συντονισμένος διαμοιρασμός και χρήση πόρων για την επίλυση προβλημάτων στο πλαίσιο δυναμικών Εικονικών Οργανισμών, Ε.Ο. (Virtual Organizations-VOs) [2]. Ο παραπάνω διαμοιρασμός αφορά όχι μόνο στην ανταλλαγή δεδομένων αλλά επίσης στην άμεση πρόσβαση σε οντότητες Grid (υπολογιστικές μονάδες, υπηρεσίες, λογισμικό, δεδομένα και άλλους πόρους), που συνδέονται μεταξύ τους σύμφωνα με ένα πλαίσιο εμπιστοσύνης. Επιπλέον, ο προαναφερθείς διαμοιρασμός θα πρέπει να είναι ελεγχόμενος, με τους παρόχους και τους χρήστες των πόρων να ακολουθούν πρωτόκολλα τα οποία θα καθορίζουν με σαφήνεια τι θα πρέπει να μοιραστεί, ποιος επιτρέπεται να διαμοιράσει και ποιες είναι οι συνθήκες κάτω από τις οποίες πραγματοποιείται ο διαμοιρασμός. Τα μέλη ενός Ε.Ο. επιδιώκουν την επίτευξη ενός κοινού στόχου και λόγω αυτού οι διάφορες οντότητες μπορούν να γίνονται μέλη ή να αποχωρούν από έναν Ε.Ο. δυναμικά και ενώ το σύστημα βρίσκεται σε λειτουργία. Για παράδειγμα ένα πρόγραμμα εκτελεί μια συνάρτηση, η οποία για την εκτέλεσή της χρειάζεται πρόσβαση σε διάφορες υπηρεσίες, που βρίσκονται σε διάφορα μέρη. Είναι δυνατόν στην περίπτωση αυτή να δημιουργηθεί δυναμικά για όσο διάστημα χρειαστεί ένας Ε.Ο., ο οποίος θα καθορίζει το επίπεδο ασφάλειας μεταξύ των μελών του, παρέχοντας (κάτω από προϋποθέσεις βέβαια) στη συνάρτηση, πρόσβαση στις αναγκαίες υπηρεσίες.

Επομένως ένας ορισμός του Ε.Ο. μπορεί να είναι: το σύνολο των οντοτήτων που συνδέονται προσωρινά για ένα κοινό σκοπό ή για την εκτέλεση μιας κοινής εργασίας. Ένα Grid μπορεί να περιλαμβάνει πολλούς Ε.Ο., ενώ μια οντότητα μπορεί να είναι μέλος πολλών Ε.Ο. ταυτόχρονα με αποτέλεσμα να υπάρχει δυνατότητα επικάλυψης.

Βάσει των παραπάνω, υπάρχουν πολλές απαιτήσεις για την επιθυμητή λειτουργία των Ε.Ο., όπως :

- *Ευέλικτες σχέσεις διαμοιρασμού*: Οι σχέσεις διαμοιρασμού μπορούν να μεταβάλλονται δυναμικά στον χρόνο, εννοώντας το ποιοι πόροι διαμοιράζονται, τον τύπο την πρόσβασης που επιτρέπεται, και σε ποιους συμμετέχοντες επιτρέπεται η χρήση του πόρου. Αυτές οι σχέσεις δεν ονομάζουν απαραίτητα ένα σύνολο συμμετεχόντων πόρων, αλλά τις περισσότερες φορές καθορίζονται εμμέσως από τις πολιτικές πρόσβασης του πόρου. Για παράδειγμα ένας οργανισμός μπορεί να επιτρέψει την πρόσβαση σε ένα πόρο σε οποιονδήποτε μπορεί να αποδείξει ότι είναι πελάτης.

- *Μηχανισμοί εντοπισμού*: Η δυναμική φύση των σχέσεων διαμοιρασμού απαιτεί την ύπαρξη μηχανισμών για την ανακάλυψη και τον προσδιορισμό των σχέσεων που υπάρχουν σε μια συγκεκριμένη χρονική στιγμή. Για παράδειγμα ένας νέος συμμετέχων σε έναν Ε.Ο. πρέπει να έχει την δυνατότητα να εντοπίσει τους πόρους στους οποίους του επιτρέπεται η πρόσβαση, την ποιότητα υπηρεσίας (QoS) που παρέχουν αυτοί οι πόροι κτλ.
- *Επίλυση θεμάτων χρονοπρογραμματισμού και ταυτόχρονης δέσμευσης ενός συνόλου πόρων*: Πολλά προβλήματα απαιτούν την ταυτόχρονη χρήση πόρων για να επιλυθούν. Γι' αυτόν τον λόγο οι σχέσεις διαμοιρασμού πρέπει να μπορούν να συνδυαστούν για την συντονισμένη χρήση πολλών πόρων που βρίσκονται σε κάποιους οργανισμούς. Για παράδειγμα μια διαμοιραζόμενη υπολογιστική μονάδα (πόρος 1) που επεξεργάζεται δεδομένα από μια διαμοιραζόμενη μονάδα αποθήκευσης (πόρος 2).
- *Μηχανισμοί Αποστολής Δικαιωμάτων* (delegation): Μηχανισμοί με τους οποίους ο χρήστης εξουσιοδοτεί τον διαμοιραζόμενο πόρο με τα δικαιώματά του. Για παράδειγμα μια υπολογιστική μονάδα να εξουσιοδοτηθεί ώστε να έχει δικαίωμα πρόσβασης στα αρχεία της οντότητας που την χρησιμοποιεί.
- *Σχέσεις ομότιμων*: Οι σχέσεις διαμοιρασμού πολύ συχνά δεν είναι απλά της μορφής πελάτης - εξυπηρετητής αλλά ομότιμων οντοτήτων, δηλαδή οι παροχείς μπορεί να είναι και καταναλωτές.
- *Διαφορετική λειτουργία ενός πόρου*: Ο ίδιος πόρος μπορεί να χρησιμοποιηθεί με διαφορετικό τρόπο ανάλογα με τους περιορισμούς πρόσβασης και τον σκοπό για τον οποίο γίνεται η διαμοίραση. Για παράδειγμα ένας υπολογιστής σε έναν Ε.Ο. μπορεί να χρησιμοποιείται για την εκτέλεση ενός συγκεκριμένου λογισμικού και σε κάποιον άλλον Ε.Ο. να χρησιμοποιείται γενικά, παρέχοντας υπολογιστική ισχύ.

2.5 Ταξινόμηση του Πλέγματος

Υπάρχουν διάφορες ταξινομήσεις Grid συστημάτων σε επίπεδο εφαρμογών, επαγγελματικής αξίας, επιστημονικής αποτελεσματικότητας και αρχιτεκτονικής. Μια από τις δημοφιλέστερες ταξινομήσεις σχετίζεται με το επίπεδο πολυπλοκότητας της οργάνωσης και της έκτασης που καταλαμβάνει το όλο σύστημα. Βάσει αυτής διακρίνονται οι ακόλουθες κατηγορίες:

- *Departmental Grids*, τα οποία εγκαθίστανται συνήθως σε ένα τομέα ενός οργανισμού και συνήθως προορίζονται για να εξυπηρετήσουν ένα και μόνο σκοπό. Εξωτερικά προστατεύονται με firewall κάτι το οποίο ταιριάζει στην περίπτωση αφού συνήθως η όποια επικοινωνία γίνεται εσωτερικά στον τομέα. Η περίπτωση αυτή

είναι παρόμοια με αυτή των clusters. Υπάρχει όμως η διαφορά ότι το hardware που χρησιμοποιείται δεν παρουσιάζει απαραίτητα ομοιογένεια. Επίσης υπάρχει μικρή μόνο ανάγκη για τον καθορισμό πολιτικής πρόσβασης και για την επίβλεψη της πρόσβασης. Τελικά ο σκοπός των departmental Grids είναι να συγκεντρώσουν και εικονοποιήσουν τους πόρους του τομέα έτσι ώστε να επιτευχθεί η καλύτερη δυνατή χρήση τους.

- *Enterprise Grids*, τα οποία ανήκουν σε μία οργανωμένη οντότητα όπως μια επιχείρηση και χρησιμοποιούνται για πολλούς σκοπούς. Υπάρχει η δυνατότητα επικοινωνίας με τον εξωτερικό κόσμο, η οποία δεν είναι αμφίδρομη καθώς οι πόροι καθαυτοί δε μετακινούνται έξω από τα όρια του firewall. Τα enterprise grids χρησιμοποιούνται από οργανισμούς που θέλουν με κάποιο τρόπο να συγκεντρώσουν τους πόρους τους που πριν ήταν διασκορπισμένοι σε διάφορα τμήματα τους. Στην περίπτωση αίτησης χρήσης πόρων από δύο ή και περισσότεροι τομείς της επιχείρησης εφαρμόζονται μηχανισμοί επίλυσης τέτοιων διενέξεων μέσω ενός συστήματος πολιτικών χρήσης (usage policies).
- *Partner Grids*, τα οποία είναι εγκαταστάσεις που ξεπερνούν το firewall μιας επιχείρησης και επιτρέπουν τη κοινή χρήση των πόρων από διάφορους οργανισμούς. Η ανάγκη για τα partner grids προέκυψε από την ανάγκη συνεργασίας μεταξύ διάφορων επιχειρησιακών οντοτήτων για την επίτευξη ενός κοινού στόχου. Η έννοια των partner grids δε θα πρέπει να συγχέεται με αυτή των Εικονικών Οργανισμών (όπως αναφέρθηκε στην παράγραφο 2.4.5). Ένας Ε.Ο. μπορεί να περιλαμβάνει πολλά partner grids και ένα partner grid πολλούς Ε.Ο.. Η διάθεση πόρων στο εξωτερικό περιβάλλον ενός οργανισμού απαιτεί την ύπαρξη προτύπων και πολιτικών για την ασφάλεια, την επίβλεψη των διάφορων ενεργειών και το accounting.
- *Open Grids*, τα οποία αναμένεται να εμφανιστούν στο μέλλον και αναφέρονται σε ì ια πλατφόρμα που αποτελείται από υποδομή, μεσολογισμικό και εφαρμογές που χρησιμοποιούνται από κοινού από διάφορους ανεξάρτητους οργανισμούς. Υπό φυσιολογικές συνθήκες ένα open grid δεν είναι αφιερωμένο στην επίτευξη ενός και μόνο στόχου. Αντίθετα οι συμμετέχοντες χρησιμοποιούν το σύστημα για να επιτύχουν τους δικούς τους ανεξάρτητους στόχους, άλλες φορές δουλεύοντας μεμονωμένα και άλλες φορές συμμετέχοντας σε εικονικούς οργανισμούς. Το grid θα προσφέρει τους πόρους και τις απαραίτητες υπηρεσίες για το έργο, καθώς και ένα μέσο για το sharing και την επικοινωνία μεταξύ των συνεργαζομένων. Οντότητες που δε χρειάζονται πλέον τους πόρους τους, θα μπορούν να τους αξιοποιούν διαθέτοντάς τους σε ένα από τα υπάρχοντα grid και αποκτώντας σε αντάλλαγμα οικονομικό κέρδος. Το open Grid (συχνά αναφέρεται και ως παγκόσμιο Grid) θα

αποτελέσει φυσικό επακόλουθο της παράλληλης ύπαρξης διάφορων enterprise grids και partner grids που θα επικοινωνούν μεταξύ τους με τη χρήση των ίδιων πρωτοκόλλων.

2.6 Αρχιτεκτονική Σχεδίαση του Πλέγματος

Η διαδικασία της δημιουργίας μιας αρχιτεκτονικής Grid είναι παρόμοια με άλλες προσπάθειες κατασκευής ενός συστήματος, ειδικά όταν αναφερόμαστε σε κατανεμημένα συστήματα. Παρόλα αυτά, ορισμένες παράμετροι διαδραματίζουν σημαντικότερο ρόλο στο σχεδιασμό ενός Grid συστήματος απ' ότι άλλες. Οι παράμετροι αυτές είναι:

- Απαιτήσεις ασφάλειας
- Ευαισθησία δεδομένων
- Μέγιστες απαιτήσεις σε επεξεργαστική ισχύ
- Αποθήκευση δεδομένων
- Διαθέσιμο εύρος ζώνης στο διαδίκτυο
- Υπάρχοντες πόροι
- Πόροι ειδικού σκοπού
- Μεταφερσιμότητα
- Δυνατότητες συνεργασίας

2.6.1 Απαιτήσεις ασφάλειας

Τυπικά δεδομένα επιχειρήσεων, επιχειρησιακές πρακτικές και επιχειρησιακοί πόροι πρέπει να προστατεύονται από καταστροφή, κλοπή και γενικά από ενέργειες που στοχεύουν να βλάψουν το σύστημα. Σχεδόν όλες οι σύγχρονες εταιρείες σχεδιάζουν κανόνες ασφάλειας, έτσι ώστε να επιτρέπουν στους υπαλλήλους τους να έχουν πρόσβαση στα δεδομένα της εταιρείας και ταυτόχρονα να αποτρέπουν την πρόσβαση από οποιονδήποτε θέλει να βλάψει τα συμφέροντά της.

Κατά καιρούς έχουν εμφανιστεί διάφορες λύσεις. Τα firewalls ήταν μια γρήγορη και εύκολη προσέγγιση, η οποία όμως υστερούσε σε ευελιξία. Τα firewalls επιτρέπουν μόνο τη κυκλοφορία δεδομένων, η οποία έχει εγκριθεί από το διαχειριστή του δικτύου. Έτσι κάθε δραστηριότητα για την οποία δεν έχει δοθεί ειδική έγκριση θεωρείται επίθεση προς το σύστημα. Ενώ όμως το σύστημα προστατεύεται, δημιουργείται πρόβλημα απόδοσης σε επικοινωνίες διαφόρων ειδών, αφού πριν γίνει κάθε επικοινωνία θα πρέπει να υπάρξει συνεννόηση με το διαχειριστή του δικτύου, έτσι ώστε να ρυθμίσει το firewall (ανοίγοντας ίσως και κάποιες πόρτες) και τελικά να επιτραπεί η επικοινωνία.

Με τη μετάβαση της υποδομής του Grid στις τεχνολογίες των υπηρεσιών δικτύου (Web Services) και XML, καθίσταται ευκολότερο για ένα firewall να επιβλέπει τα δεδομένα που περνούν. Επίσης με την εμφάνιση των WS-Agreements (ή SLA - Service Level Agreement), επιτρέπεται η διαπραγμάτευση για την πραγματοποίηση ή μη της επικοινωνίας, να γίνεται

με παρόμοιο τρόπο για όλων των ειδών τις υπηρεσίες. Σε αυτή την περίπτωση μπορεί πλέον το firewall να διατηρεί ανοικτές διασυνδέσεις, με τις οποίες θα συνδέονται οι διάφορες εφαρμογές, για να πραγματοποιήσουν τη διαπραγμάτευση με το σύστημα, αποκτώντας τελικά εξουσιοδοτημένη πρόσβαση σε υπηρεσίες και δεδομένα.

2.6.2 Ευαισθησία δεδομένων

Στην περίπτωση που τα διαχειριζόμενα δεδομένα είναι ευαίσθητα, το περιβάλλον του Grid είναι πολύ δύσκολο να παρέχει αρκετή ασφάλεια. Για παράδειγμα, είναι πολύ εύκολο για κάποιον να παρεμβληθεί στην επικοινωνία και να αλλάξει τα μεταφερόμενα δεδομένα. Στην περίπτωση όμως των σχετικά κλειστών Grid, όπως τα enterprise Grids, η ασφάλεια των δεδομένων είναι σχετικά εύκολο να εξασφαλιστεί. Σε πιο ανοιχτές αρχιτεκτονικές όπως τα partner Grids υπάρχουν τρία (3) επίπεδα στα οποία πρέπει να εξασφαλιστεί ασφάλεια των δεδομένων:

- Μια εργασία μπορεί να πραγματοποιηθεί σε οποιονδήποτε εξουσιοδοτημένο κόμβο του Grid. Σε αυτή την περίπτωση μια λύση είναι η δημιουργία εικονικών οργανισμών που θα καθορίζουν ποιοι κόμβοι είναι αξιόπιστοι.
- Ο χώρος στον οποίο αποθηκεύονται τα δεδομένα ή κρατούνται αντίγραφά τους. Ο χώρος που αποθηκεύονται τα αυθεντικά δεδομένα είναι εύκολο να ασφαλιστεί, αφού συνήθως ανήκει στον οργανισμό τον οποίο τα δεδομένα αφορούν περισσότερο. Στη περίπτωση των χώρων που κρατούνται αντίγραφα, μπορεί πάλι να δημιουργηθεί εικονικός οργανισμός που θα καθορίζει σε ποιους κόμβους επιτρέπεται να αποθηκευτούν τα δεδομένα.
- Τέλος σημαντικό είναι να υπάρχει ασφάλεια των δεδομένων κατά τη μεταφορά τους. Σήμερα ευτυχώς ο καθορισμός ασφαλών καναλιών μεταφοράς δεδομένων είναι δυνατός με τη χρήση των προτύπων SSL (Secure Socket Layer) και WSS (Web Services Security). Έτσι είναι δυνατή η κρυπτογράφηση των δεδομένων που μεταφέρονται μεταξύ κόμβων.

2.6.3 Μέγιστες απαιτήσεις σε επεξεργαστική ισχύ

Πριν την εμφάνιση του Grid, όταν μια επιχείρηση ήθελε να πραγματοποιήσει μια εργασία αγόραζε επεξεργαστική ισχύ με τη μορφή επεξεργαστών που εγκαθίσταντο σε υπολογιστές ή υπερ-υπολογιστές. Η ισχύς αυτή πολλές φορές ξεπερνούσε την απαιτούμενη, αφού οι απαιτήσεις για επεξεργαστική ισχύ συγκεκριμένες ώρες της ημέρας μπορούσαν να είναι πολύ μικρότερες. Επίσης μπορεί να απευθυνόταν μόνο στις ανάγκες της συγκεκριμένης εργασίας και να μην ήταν δυνατό να χρησιμοποιηθεί και για άλλες εφαρμογές της επιχείρησης. Έτσι η πλεονάζουσα ισχύς δεν αξιοποιείτο.

Με την εμφάνιση όμως του Grid κατέστη δυνατό η επιχείρηση να χρησιμοποιεί την ισχύ για περισσότερες εφαρμογές, όπως επίσης να διαθέτει επεξεργαστική ισχύ σε τρίτους

αποκομίζοντας και κάποιο κέρδος. Επιπλέον, η επιχείρηση έχει τη δυνατότητα να μην αγοράσει επεξεργαστική ισχύ που θα ανταποκρίνεται στις μέγιστες απαιτήσεις της, εφόσον βέβαια αυτές οι απαιτήσεις δεν υφίστανται για μεγάλο διάστημα της ημέρας. Αντίθετα έχει τη δυνατότητα, για τις ώρες που οι απαιτήσεις φτάνουν στο maximum, να καταφεύγει σε προσφορές επεξεργαστικής ισχύς από τρίτους για να καλύψει τις βραχυπρόθεσμες απαιτήσεις της.

2.6.4 Αποθήκευση δεδομένων

Το Grid θεωρείται ως επίπεδη αρχιτεκτονική από την άποψη ότι όλοι οι κόμβοι είναι ισοδύναμοι. Η αποθήκευση των δεδομένων ακολουθεί την ίδια λογική και επομένως οποιοδήποτε δεδομένο μπορεί να τοποθετηθεί οπουδήποτε. Το τι θα αποθηκευτεί σε μια συγκεκριμένη τοποθεσία καθορίζεται από το σύστημα αποθήκευσης, τη διαθέσιμη χωρητικότητα, το ρυθμό δεδομένων, το κόστος αποθήκευσης, το χρόνο ζωής των δεδομένων και φυσικά την πολιτική.

Συνήθως τα πρωτόκολλα που χρησιμοποιούνται εδώ και που διαχειρίζονται την αποθήκευση των δεδομένων σύμφωνα με την απαίτηση του χρήστη λειτουργούν με σχετικά αδιαφανή τρόπο. Αυτό σημαίνει ότι ο χρήστης δεν γνωρίζει το μηχανισμό, με τον οποίο αποθηκεύεται ένα αρχείο του (αν πχ. αποθηκεύεται σε σκληρό δίσκο ή ταινία). Αυτό που γνωρίζει είναι ότι έχει κάνει μια συμφωνία με το πρωτόκολλο, η οποία του υπόσχεται ότι το αρχείο θα αποθηκευτεί και θα διατηρηθεί στο χώρο αποθήκευσης.

Από τα κριτήρια που αναφέρθηκαν παραπάνω τα σημαντικότερα είναι η χωρητικότητα, ο χρόνος απόκρισης, η ρυθμαπόδοση (throughput) και η διάρκεια ζωής των δεδομένων. Με βάση τα χαρακτηριστικά αυτά γίνεται και επιλογή μεταξύ σκληρού δίσκου ή ταινίας:

- Η χωρητικότητα είναι ο συνολικός χώρος ενός συστήματος αποθήκευσης και θα πρέπει να ανταποκρίνεται στις μέγιστες απαιτήσεις για αποθηκευτικό χώρο. Ορισμένα συστήματα αποθήκευσης δεν κλιμακώνουν αποτελεσματικά, καθώς αυξάνεται η χωρητικότητα. Στη περίπτωση που χρησιμοποιείται σκληρός δίσκος είναι δύσκολο να δημιουργηθεί ένα οικονομικό σύστημα μεγάλης αποθήκευσης για αποθήκευση δεδομένων για μεγάλο χρονικό διάστημα.
- Ο χρόνος απόκρισης συνήθως καθορίζει το μέσο που θα χρησιμοποιηθεί. Όταν το ζητούμενο είναι μικρός χρόνος απόκρισης, τότε είναι αδύνατο να χρησιμοποιηθούν συστήματα βασισμένα σε ταινία, ενώ όταν το ζητούμενο είναι η βέλτιστη απόδοση τότε τα συστήματα ταινίας αποδεικνύονται πιο οικονομικά.
- Το πρόβλημα της ρυθμαπόδοσης αντιμετωπίζεται εύκολα στα Grids. Χάρη στην έμφυτη ικανότητα του Grid να κλιμακώνεται υπάρχει η δυνατότητα να αυξηθεί η ρυθμαπόδοση χρησιμοποιώντας πολλά στιγμιότυπα του ζητούμενου αντικειμένου, έτσι ώστε να επιτευχθεί αθροιστικά ο ρυθμός παράδοσης των δεδομένων. Επομένως είναι δυνατόν να χρησιμοποιηθούν σχετικά αργά μέσα όπως οι ταινίες αποθήκευσης.

- Τέλος ο χρόνος ζωής των δεδομένων ευνοεί συνήθως τα συστήματα ταινίας

Το σημαντικό στην περίπτωση του Grid είναι ότι με τη χρήση των κατάλληλων πρωτοκόλλων και προτύπων τα διάφορα συστήματα αποθήκευσης αντιμετωπίζονται με ενιαίο τρόπο.

2.6.5 Διαθέσιμο εύρος ζώνης στο διαδίκτυο

Αφού το Grid στηρίζεται στο διαδίκτυο, το διαθέσιμο εύρος ζώνης (bandwidth) αποτελεί σημαντικό παράγοντα. Επομένως το μέγεθος της σύνδεσης με το internet μιας εφαρμογής που τρέχει πάνω σε Grid, είναι σημαντικό. Όταν κάποιος αγοράζει πρόσβαση σε πόρους, πρέπει να υπάρχει αρκετό εύρος ζώνης, για να προωθηθεί η εργασία μέσω του διαδικτύου. Μια εφαρμογή που τρέχει πάνω σε Grid θα πρέπει να μεταφέρει μαζί της το περιβάλλον λειτουργίας της, γεγονός που στη περίπτωση που τρέχουν πολλές μικρές εφαρμογές μαζί, καθιστά το μέγεθος των μεταφερόμενων δεδομένων (εφαρμογή, βιβλιοθήκες, αρχεία) αρκετά μεγάλο. Μάλιστα, στην περίπτωση που τα αποτελέσματα μιας έστω και μικρής εφαρμογής παράγουν μεγάλο όγκο αποτελεσμάτων, τα πράγματα χειροτερεύουν.

Ευτυχώς στις μέρες μας, τουλάχιστον στην περίπτωση των επιστημονικών εφαρμογών, οι συνδέσεις με το διαδίκτυο μπορεί να είναι και της τάξης των μερικών δεκάδων Gigabit/sec. Παρόλα αυτά στη καθόλου ασυνήθιστη περίπτωση που 100 κόμβοι επιχειρούν ταυτόχρονα να εκτελέσουν μια εργασία με σχετικά μεγάλο όγκο δεδομένων, ακόμη και το εύρος ζώνης των 10 Gbps μπορεί να αποδειχθεί ανεπαρκές καθυστερώντας έτσι την εκτέλεση των εργασιών. Επειδή είναι χαρακτηριστικό του Grid να υπάρχουν μεγάλοι όγκοι δεδομένων και χρήστες που τους διαχειρίζονται να είναι γεωγραφικά κατανεμημένοι, πρέπει να λαμβάνονται υπόψη τα παρακάτω:

- Σχεδιασμός μιας αρχιτεκτονικής δεδομένων που να εξυπηρετεί τις ανάγκες. Θα πρέπει να λαμβάνονται υπόψη τα είδη των συνδέσεων των διαφόρων χρηστών, που βρίσκονται, ποιος είναι ο αναμενόμενος όγκος δεδομένων και ποιο το απαιτούμενο εύρος ζώνης και ποιότητα υπηρεσίας (QoS).
- Θα πρέπει συχνά να γίνεται προσωρινή αποθήκευση επιπλέον αντιγράφων των δεδομένων (caching), για να βελτιστοποιείται ο χρόνος πρόσβασης. Ερωτήσεις που θα πρέπει να απαντηθούν είναι αν το αντίγραφο είναι ενημερωμένο, πόσο συχνά να ανανεώνεται και στη περίπτωση του κατανεμημένου συστήματος caching, ποιο αντίγραφο είναι πλησιέστερο στο χρήστη
- Για αξιοπιστία μπορεί να χρησιμοποιηθεί πλεονασμός δεδομένων έτσι ώστε να εξασφαλίζεται ότι τα δεδομένα είναι διαρκώς διαθέσιμα. Έτσι αν ένας κόμβος που έχει το δεδομένο πάψει να λειτουργεί, τότε αυτό μπορεί να αναζητηθεί αυτόματα σε άλλον.

- Για τη μεταφορά των δεδομένων μεταξύ των τόπων αποθήκευσης και χρήσης τους χρειάζεται ένα μέσο, το οποίο να παρέχει γρήγορη, αξιόπιστη και ασφαλή μεταφορά.
- Επίσης πρέπει να καθορίζεται σε ποιον ανήκουν τα δεδομένα, ποιοι μπορούν να τα δουν και ποιοι να τα τροποποιήσουν ή να τα σβήσουν.
- Ίσως να χρειάζονται επιπλέον δεδομένα μετα-πληροφορίας που θα περιγράφουν τη δομή του συστήματος και των δεδομένων αυτού. Αυτά τα δεδομένα έχουν παρόμοιες ανάγκες με τα υπόλοιπα, όπως να είναι προσβάσιμα και αξιόπιστα.

2.6.6 Υπάρχοντες πόροι

Η δυνατότητα της επαναχρησιμοποίησης υπαρχόντων πόρων, συνήθως υπολογιστών, ήταν από τους κύριους λόγους που οδήγησαν στην αποδοχή της τεχνολογίας του Grid. Σημαντικό για ένα Grid είναι η δημιουργία ενός είδους ευρετηρίου πόρων καθώς και της ισχύς τους.

Υπάρχουν μερικές κατηγορίες τέτοιων υπολογιστικών πόρων:

- Υπολογιστές γραφείου (desktop PCs), οι οποίοι υπάρχουν σχεδόν σε όλο τον κόσμο και όχι μόνο σε επιχειρήσεις, αλλά και σε απλά νοικοκυριά. Συνήθως αυτοί οι πόροι είναι διαθέσιμοι τις νυχτερινές ώρες και χαρακτηρίζονται από υψηλό κόστος συντήρησης ανά μηχάνημα. Ακόμη και έτσι όμως το πλήθος τους, σε συνδυασμό με το ότι παραμένουν ανενεργοί για 12 με 16 ώρες τη μέρα, τους καθιστά ένα ισχυρό εργαλείο για εργασίες (συνήθως συμπληρωματικές) που γίνονται νυχτερινές ώρες.
- Clusters. Αυτά τα συστήματα είναι συνήθως διαθέσιμα 24 ώρες το εικοσιτετράωρο και είναι αξιόπιστα και ασφαλή. Η γνώση της δομής ενός cluster, τι μηχανήματα περιλαμβάνει και ποιους εξειδικευμένους πόρους είναι πολύτιμη για ένα Grid σύστημα. Έτσι με τη χρήση του κατάλληλου συστήματος (scheduler) μπορούν να αντιστοιχιστούν οι εργασίες στους πόρους που είναι καταλληλότεροι για την εκτέλεσή τους.
- Συστήματα συμμετρικής πολυεπεξεργασίας (SMP). Αυτά τα συστήματα μέχρι την εμφάνιση της Grid τεχνολογίας παρέμεναν αναξιοποίητα για μεγάλα χρονικά διαστήματα. Πλέον είναι δυνατή η χρήση τους ως πόροι και είναι ιδιαίτερα χρήσιμα στην περίπτωση εργασιών που εκμεταλλεύονται ακριβώς τις ειδικές δυνατότητες που διαθέτουν.

Σημαντικό είναι να καθοριστεί ποια είδη εργασιών εκμεταλλεύονται καλύτερα τις ειδικές ικανότητες κάθε συστήματος. Όταν δημιουργηθεί ένα ευρετήριο των πόρων που είναι διαθέσιμοι, της ζήτησης που μπορεί να καλυφθεί με τη διάθεση αυτών στο Grid και των προτεραιοτήτων των εργασιών, μπορεί κάποιος να διαπιστώσει, εάν η υπάρχουσα

υποδομή επαρκεί για να καλύψει τις ανάγκες του ή αν χρειάζεται να ξοδευτούν επιπλέον χρήματα για την ενίσχυσή της.

2.6.7 Πόροι ειδικού σκοπού

Πολλοί οργανισμοί διαθέτουν πόρους οι οποίοι δημιουργήθηκαν ειδικά για την εκτέλεση μιας εργασίας ή για μια κατηγορία από εργασίες. Αυτοί οι πόροι μπορεί να είναι ειδικά μηχανήματα, όπως συσκευές μετρήσεων, ειδικού σκοπού λογισμικό και ειδικού σκοπού υλικό. Παραδοσιακά αυτοί οι πόροι θα γίνονταν προσβάσιμοι μέσω ειδικών πρωτοκόλλων ή ειδικά διαμορφωμένων δικτύων.

Με την έλευση του Grid computing εμφανίστηκαν διάφορα εργαλεία, που επιτρέπουν τη δημιουργία νέων υπηρεσιών κατάλληλων για σύνδεση με αυτούς τους πόρους. Βέβαια τα εργαλεία αυτά δεν έχουν φτάσει στο επίπεδο ωριμότητας που απαιτείται, ώστε να είναι δυνατό αυτό για όλους τους πόρους. Για τους ειδικούς πόρους που αυτό δεν είναι δυνατό μέχρι στιγμής, υπάρχουν δύο λύσεις:

- Χάρη στις τεχνολογίες των υπηρεσιών δικτύου και XML υπάρχει διαθέσιμη μια τεράστια ποικιλία βιβλιοθηκών, που επιτρέπουν τη δημιουργία κώδικα, που θα ανταποκρίνεται στις εκάστοτε ανάγκες. Ο κώδικας αυτό μπορεί να αποτελεί ένα resource wrapper ή minihosting περιβάλλον, επιτρέποντας την πρόσβαση στον πόρο μέσω των υπαρχόντων ιδιόκτητων διασυνδέσεων.
- Μια άλλη επιλογή είναι να τρέχει σε ένα μηχάνημα μια υπηρεσία Grid, με την οποία θα επικοινωνεί αυτός που επιθυμεί την πρόσβαση στον πόρο. Η Grid υπηρεσία αυτή θα προωθεί μέσω ειδικού μηχανισμού τις απαραίτητες πληροφορίες στον πόρο. Η δεύτερη αυτή λύση προβλέπεται να είναι φθηνότερη και μάλλον θα προτιμηθεί στο μέλλον.

2.6.8 Μεταφερσιμότητα

Ένα σημαντικό ζήτημα στο σχεδιασμό κάθε Grid είναι η μετατροπή των εφαρμογών του προηγούμενου συστήματος έτσι ώστε πλέον να λειτουργούν στο περιβάλλον του Grid και να το αξιοποιούν όσο το δυνατόν αποδοτικότερα. Μια ιδέα η οποία αποτελεί γρήγορη και εύκολη λύση είναι να γραφτεί για την εφαρμογή ένας αντίστοιχος Grid wrapper, κώδικας ο οποίος θα επιτρέπει τη λειτουργία της στο νέο περιβάλλον και να αξιοποιεί τα ιδιαίτερα χαρακτηριστικά του. Παρόλα αυτά όμως, για να αξιοποιήσει η εφαρμογή καλύτερα το Grid σύστημα, πρέπει να προσαρμόζεται ο ίδιος ο κώδικάς της ξαναγράφοντας μέρος του.

Υπάρχουν δύο ειδών συστατικά μέρη στο Grid: αυτά που χρησιμοποιούν τις υπηρεσίες άλλων και αυτά που παρέχουν μια υπηρεσία χωρίς να στηρίζονται σε άλλες υπηρεσίες του Grid. Και στις δύο περιπτώσεις για να δημιουργηθεί ένα τέτοιο στοιχείο, είναι απαραίτητο να λαμβάνονται συνεχώς υπόψη οι χρηστικές και λογισμικές απαιτήσεις του προηγούμενου συστήματος. Για απαιτήσεις, όπως η ασφάλεια, χρειάζεται νέος σχεδιασμός. Από την άλλη ορισμένες απαιτήσεις (όπως απαιτήσεις απόδοσης) γίνονται λιγότερο περιοριστικές στο

περιβάλλον του Grid (ένας αλγόριθμος ίσως να μη χρειάζεται να είναι υπερβολικά αποδοτικός, αφού οι υπάρχοντες πόροι στο Grid μπορούν να καλύψουν τις απαιτήσεις ενός μη βελτιστοποιημένου αλγορίθμου).

Ορισμένες υπάρχουσες εφαρμογές μπορούν να εκτελεστούν σε ένα περιβάλλον Grid με τη βοήθεια διεπαφών, που κάνουν χρήση ήδη καθορισμένων προτύπων. Έτσι μειώνεται ο φόρτος εργασίας που χρειάζεται, για να γίνουν αυτές προσβάσιμες μέσα στο Grid περιβάλλον. Υπάρχουν όμως και άλλες εφαρμογές, η μεταφορά των οποίων δεν είναι δυνατή ακόμη και με την παραπάνω μέθοδο. Σε αυτή την περίπτωση χρειάζεται για κάθε εφαρμογή να γραφτεί μια ειδική διεπαφή, που θα στηρίζεται στην τεχνολογία Grid.

2.6.9 Δυνατότητες συνεργασίας

Ένα από τα σημαντικότερα ζητήματα κατά το σχεδιασμό ενός Grid συστήματος, είναι οι δυνατότητες που αυτό θα προσφέρει στους χρήστες του για συνεργασία με άλλους χρήστες και οντότητες του συστήματος. Εξάλλου βασικός στόχος του Grid είναι η επίτευξη κοινών στόχων, όπως η επίλυση ενός πολύπλοκου προβλήματος, με τη συνεργασία πολλών παραγόντων, πόρων κλπ.

Σημαντικό επομένως είναι να υπάρχει η δυνατότητα καθορισμού πολιτικών διαπραγμάτευσης για την επίτευξη της συνεργασίας. Πρέπει να καθορίζονται πρωτόκολλα συνεργασίας και κανονισμοί, οι οποίοι θα εξασφαλίζουν την ασφάλεια της όλης διαδικασίας. Πρέπει να δίνεται η δυνατότητα δημιουργίας δυναμικών εικονικών οργανισμών και δυναμικής διαχείρισής τους. Η έννοια του εικονικού οργανισμού είναι καθοριστική, καθώς σύμφωνα με τον ορισμό του, οι μετέχοντες σε έναν εικονικό οργανισμό συνεργάζονται μεταξύ τους για να εξυπηρετήσουν τα κοινά τους συμφέροντα.

Η ιδανική περίπτωση είναι το Grid να περιλαμβάνει αυξημένες δυνατότητες συνεργασίας, συνοδευόμενες όμως και από αντίστοιχο υψηλό επίπεδο ασφάλειας του συστήματος. Δυστυχώς, μέχρι σήμερα, αποδεικνύεται ότι η ταυτόχρονη επίτευξη των στόχων αυτών παρουσιάζει σημαντικές δυσκολίες. Αυτές αναμένεται να ξεπεραστούν, ως ένα βαθμό, στο μέλλον, με την εμφάνιση νέων βελτιωμένων πρωτοκόλλων ασφάλειας και συνεργασίας.

2.7 Αρχιτεκτονική Ανοιχτών Υπηρεσιών Πλέγματος

Οι συντονισμένες μελέτες του *Open Grid Forum* (OGF) - ενός οργανισμού αποτελούμενου από χρήστες, προγραμματιστές, ερευνητές με σκοπό την ανάπτυξη προτύπων για την υποστήριξη των Grid τεχνολογιών - κατέληξε το 2002 στην Αρχιτεκτονική Ανοιχτών Υπηρεσιών - *Open Grid Service Architecture* [8], η οποία συντάχθηκε σε ένα κείμενο που περιγράφει την δομή ενός Grid συστήματος και συγκεντρώνει όλα τα πρότυπα που χρησιμοποιεί η αρχιτεκτονική αυτή σε κάθε επίπεδο της.

2.7.1 Στόχοι της Αρχιτεκτονικής

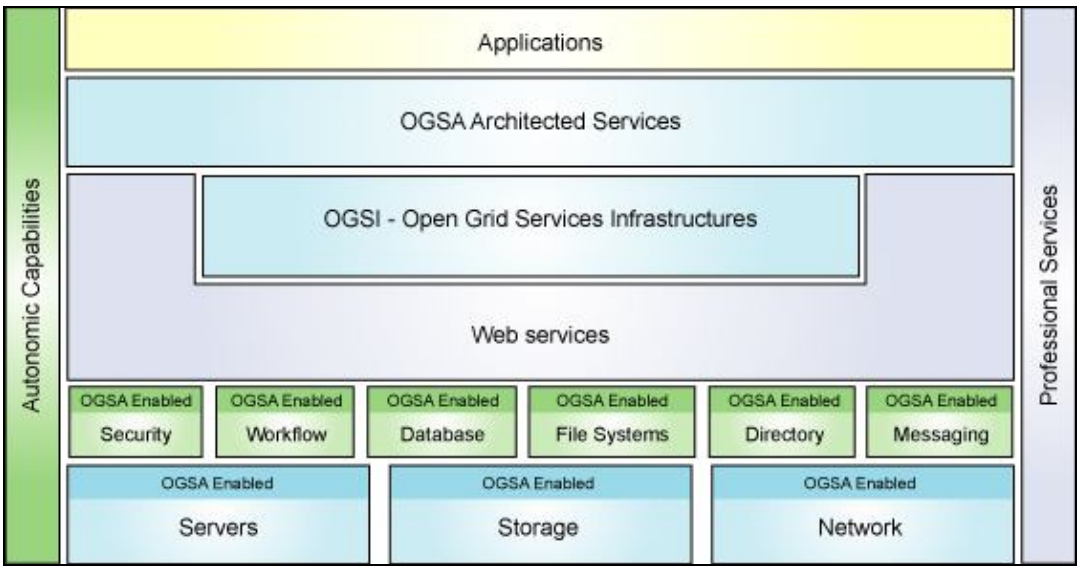
Οι κύριοι στόχοι του OGSA είναι οι ακόλουθοι:

- να διαχειρίζεται πόρους απομακρυσμένων ετερογενών συστημάτων
- να αποδίδει σημαντική ποιότητα στις υπηρεσίες που παρέχει (Quality of Service)
- να παρέχει τις βάσεις για αυτόνομες λύσεις διαχείρισης. Η ποικιλομορφία των πόρων που συμμετέχουν σε ένα Grid και ο δυναμικός τρόπος συνεργασίας τους απαιτεί ένα σύστημα διαχείρισης που θα προσαρμόζεται στις εκάστοτε ανάγκες.
- να καθορίζει γνωστά πρότυπα και πρωτόκολλα λειτουργίας. Η δυναμική εισαγωγή και συνεργασία ετερογενών συστημάτων στο δίκτυο του Grid, βασίζεται στην υιοθέτηση γνωστών, ευρείας χρήσης προτύπων.
- να εκμεταλλεύεται υπάρχουσες τεχνολογίες και να τις προσαρμόζει στο Grid, αν είναι εφικτό. Το OGSA βασίζεται στη τεχνολογία των Web Services (υπηρεσίες διαδικτύου) όπως αυτές αναφέρονται στην ενότητα 2.8.2.

2.7.2 Περιγραφή της Αρχιτεκτονικής

Το OGSA αποτελείται από τέσσερα βασικά επίπεδα (όπως φαίνεται και στο σχήμα που ακολουθεί – Σχήμα 2) :

- τους πόρους (φυσικούς και λογικούς)
- τις υπηρεσίες διαδικτύου (Web Services) συμπεριλαμβανομένου και της υποδομής Open Grid Services Infrastructure (OGSI) που έχει άμεση σχέση με τις υπηρεσίες
- τις υπηρεσίες σχεδιασμένες από το OGSA
- τις Grid εφαρμογές



Σχήμα 2: Αρχιτεκτονική του OGSA

2.7.2.1 Επίπεδο φυσικών και λογικών πόρων

Αυτό είναι το πιο χαμηλό επίπεδο από τα τέσσερα και έχει να κάνει με τους πόρους που συμμετέχουν στο Grid. Η έννοια του ‘πόρου’ στο OGSA είναι πολύ σημαντική και δεν είναι συγκεκριμένη. Ως πόρος (resource) σε ένα Grid μπορεί να θεωρηθεί από τον επεξεργαστή ενός υπολογιστή, μέχρι το τμήμα υπολογιστών μιας μεγάλης εταιρίας. Επίσης εκτός από υπολογιστικές μονάδες, συμμετέχουν και μονάδες αποθήκευσης, βάσεις δεδομένων, δικτυακές υποδομές κλπ. Αυτά που αναφέρθηκαν παραπάνω είναι κυρίως οι φυσικοί πόροι. Εκτός από αυτούς έχουμε και τους λογικούς πόρους, κάποια ενδιάμεσα συστήματα (middleware) που χρησιμοποιώντας τους φυσικούς πόρους, προσφέρουν στοιχειώδεις υπηρεσίες, όπως διαχείριση αρχείων ή βάσεων δεδομένων, στο παραπάνω επίπεδο.

2.7.2.2 Επίπεδο υπηρεσιών διαδικτύου (Web Services)

Μία πολύ βασική θεώρηση του OGSA είναι ότι όλοι οι πόροι του συστήματος αντιστοιχίζονται με υπηρεσίες. Έτσι η OGSI υποδομή χρησιμοποιεί συγκεκριμένες, γνωστές υπηρεσίες δικτύου και πρωτόκολλα όπως XML και WSDL για να δημιουργήσει επαφές και διασυνδέσεις κάθε Grid υπηρεσίας με τους πόρους του συστήματος. Η OGSI επεκτείνει τις δυνατότητες των δικτυακών υπηρεσιών έτσι ώστε να παρέχει δυναμικές και αξιόπιστες υπηρεσίες όπως απαιτείται για τη μοντελοποίηση των πόρων.

2.7.2.3 Επίπεδο Grid υπηρεσιών

Οι υπηρεσίες διαδικτύου και οι δυνατότητες του OGSI παρέχουν τη βασική υποδομή για το επόμενο επίπεδο, των Grid υπηρεσιών. Αυτή την εποχή υπάρχει μεγάλη δραστηριότητα προς την κατεύθυνση προσδιορισμού και προτυποποίησης τέτοιων υπηρεσιών, όπως υπηρεσίες υπολογισμού, πληροφοριών, πυρήνα, κ.α. Καθώς οι υλοποιήσεις αυτών των υπηρεσιών θα αρχίσουν να εμφανίζονται, η OGSA αρχιτεκτονική θα γίνεται όλο και πιο χρήσιμη, βασιζόμενη πάντα στις υπηρεσίες. Πρόκειται δηλαδή για Service Oriented Architecture, όπως περιγράφεται στην ενότητα 2.8.

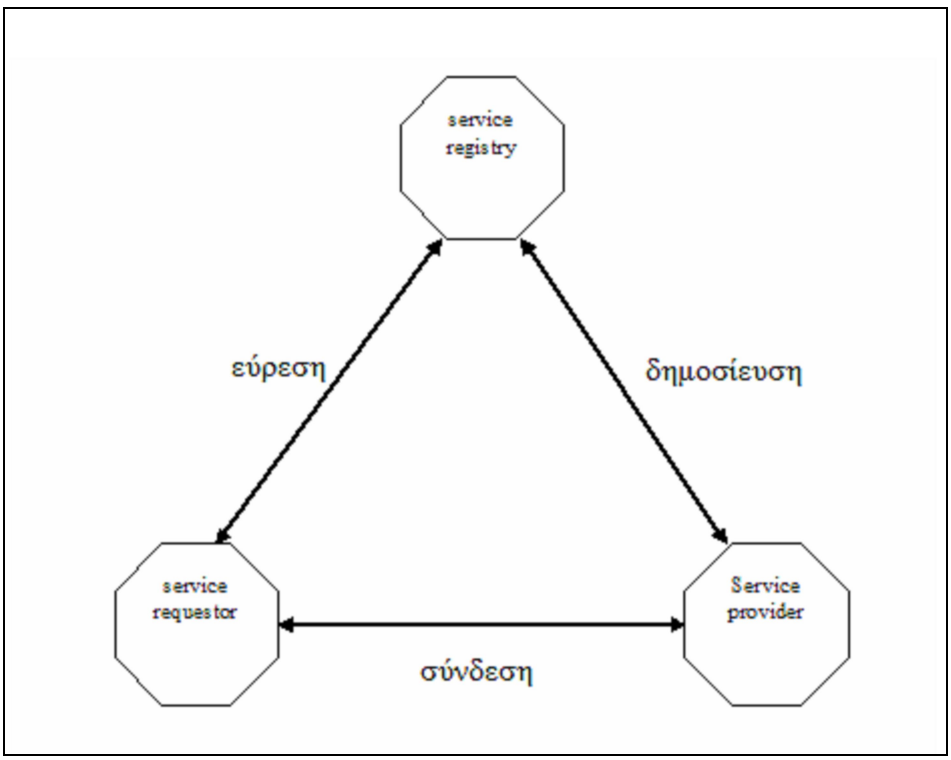
2.7.2.4 Επίπεδο εφαρμογών Grid

Με το πέρασμα του χρόνου και όσο οι Grid υπηρεσίες θα αναπτύσσονται, καινούργιες εφαρμογές βασισμένες στην Grid τεχνολογία θα κάνουν την εμφάνισή τους. Αυτές οι εφαρμογές θα χρησιμοποιούν μια ή και περισσότερες Grid υπηρεσίες του προηγούμενου επιπέδου.

2.8 *Υπηρεσιοστρεφής Αρχιτεκτονική*

Η προσαρμογή κάποιων υπηρεσιών σε μια εφαρμογή δεν είναι μια δύσκολη διαδικασία και επιπρόσθετα αναμένεται να αποδώσει στην εφαρμογή πρόσθετα λειτουργικά χαρακτηριστικά. Τα χαρακτηριστικά αυτά όμως δεν αρκουν για να δημιουργήσουν μια υπηρεσιοστρεφή αρχιτεκτονική (*Service Oriented Architecture* – SOA).

Το πρότυπο υπηρεσιοστρεφούς αρχιτεκτονικής είναι ένα σχεδιαστικό μοντέλο με κύριο χαρακτηριστικό την ενσωμάτωση λογικής εφαρμογών μέσα σε υπηρεσίες που θα αλληλεπιδρούν μέσω επικοινωνιακών πρωτοκόλλων. Βάσει αυτού, η υιοθέτηση σε μια εφαρμογή SOA δομής σημαίνει αυτόματα την αποδοχή κάποιων σχεδιαστικών αρχών και πρόσθετων τεχνολογιών ως βασικού τμήματος του τεχνικού περιβάλλοντος της.



Σχήμα 3: Δομή Service Oriented Αρχιτεκτονικής

Σε επίπεδο σχεδιασμού συστημάτων, η χρήση της τεχνολογίας των Web Services οδηγεί στην υιοθέτηση της υπηρεσιοστρεφους αρχιτεκτονικής. Οι βασικοί ρόλοι και λειτουργίες στην αρχιτεκτονική αυτή παρουσιάζονται στο Σχήμα 3. Η αρχιτεκτονική αυτή υποδεικνύει μια σχέση εξυπηρετητή-πελάτη (server-client) ανάμεσα στον προμηθευτή υπηρεσιών (service provider, που παίζει το ρόλο του server) και τον ζητώντα την υπηρεσία (service-requestor, που παίζει το ρόλο του client). Ο service-provider είναι αυτός που παρέχει την υπηρεσία δεχόμενος μηνύματα κλήσεις από τους requestors. Είναι επίσης υπεύθυνος για τη δημιουργία της περιγραφής της υπηρεσίας (service description) και τη δημοσίευσή της σε κάποιο κατάλογο-οδηγό υπηρεσιών (Universal Description, Discovery and Integration-UDDI). Ο service requestor αναζητά την υπηρεσία και την περιγραφή της σε κάποιο κατάλογο υπηρεσιών (service registry) και στη συνέχεια καλεί κατάλληλα την επιθυμητή υπηρεσία. Ο κατάλογος υπηρεσιών φέρνει ουσιαστικά τις δύο πλευρές, client και server σε

επαφή. Η συνέχεια αφορά μόνο τις δύο άλλες συμμετέχουσες μονάδες (service requestor και service provider).

2.8.1 Γενικά για τις υπηρεσίες

Τον τελευταίο καιρό έχει γίνει αρκετά μεγάλη συζήτηση γύρω από το θέμα των υπηρεσιών και των εφαρμογών. Οι υπηρεσίες που αθροιστικά σχηματίζουν το περιβάλλον μιας εφαρμογής τείνουν να γίνουν τμήματα ίδιας της εφαρμογής. Φυσικά δεν αποτελούν απλά ένα κομμάτι της εφαρμογής, αλλά διαθέτουν χαρακτηριστικά που τις μετατρέπουν σε μέρος μιας υπηρεσιοστρεφούς αρχιτεκτονικής.

Ένα από τα χαρακτηριστικά αυτά είναι η αυτονομία από άλλες υπηρεσίες. Αυτό σημαίνει ότι κάθε υπηρεσία είναι υπεύθυνη για το δικό της εύρος λειτουργίας, περιορίζοντας έτσι και εξειδικεύοντάς την σε συγκεκριμένες επαγγελματικές χρήσεις. Αυτός ο σχεδιασμός έχει ως αποτέλεσμα τη δημιουργία ανεξάρτητων μονάδων, ελαστικά συνδεδεμένων μεταξύ τους με κάποιο πρότυπο πλαίσιο επικοινωνίας. Εξαιτίας αυτής της ανεξαρτησίας που απολαμβάνουν οι υπηρεσίες στο πλαίσιο αυτό, η προγραμματιστική λογική που κάθε υπηρεσία χρησιμοποιεί, δεν χρειάζεται να προσαρμόζεται σε συγκεκριμένη πλατφόρμα ή τεχνολογία. Χαρακτηριστικό των υπηρεσιών ενός πλαισίου είναι ο πολυμορφισμός των μερών του με ταυτόχρονη ομαλή συνεργασία.

2.8.2 Διαδικτυακές Υπηρεσίες

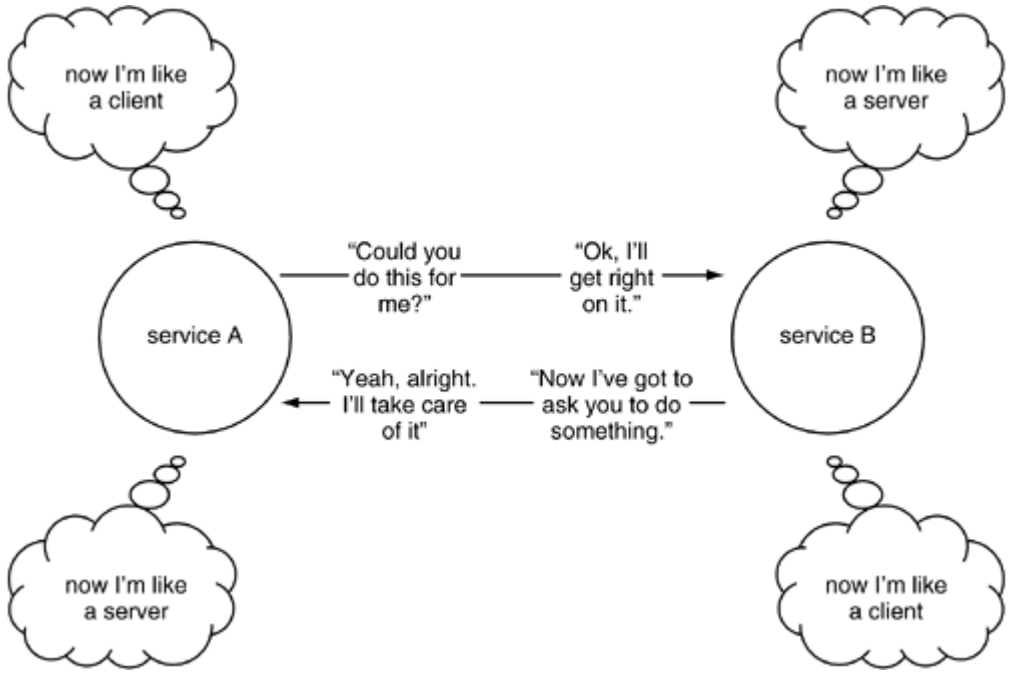
Ο πιο ευρέως διαδεδομένος και επιτυχημένος τύπος υπηρεσιών είναι οι διαδικτυακές υπηρεσίες XML Services γνωστές ως Web Services. Αυτός ο τύπος υπηρεσίας έχει δύο βασικές προαπαιτήσεις:

- επικοινωνεί μέσω πρωτοκόλλου Internet (κυρίως HTTP)
- στέλνει και δέχεται δεδομένα μέσα από XML αρχεία

Η ευρεία αποδοχή του μοντέλου των Web Services είχε ως αποτέλεσμα την ανάγκη πρόσθετων τεχνολογιών που βασίζονται σε αυτές και τη δημιουργία καινούργιων προτύπων. Έτσι η “παραγωγή” τέτοιων υπηρεσιών απαιτεί:

- τη περιγραφή της υπηρεσίας αναλυτικά, τουλάχιστον με ένα WSDL έγγραφο
- τη δυνατότητα μεταφοράς ενός XML εγγράφου χρησιμοποιώντας SOAP μέσω HTTP.

Επιπρόσθετα είναι συνηθισμένο μια υπηρεσία να λειτουργεί και ως πελάτης (client/requestor) και ως πάροχος (provider) υπηρεσίας. Αναλόγως λοιπόν με τη δραστηριότητα της υπηρεσίας κάθε στιγμή, μετατρέπεται από το ένα στο άλλο. Στο παρακάτω σχήμα φαίνεται ένα παράδειγμα αυτής της συμπεριφοράς:



Σχήμα 4: Ανταλλαγή ρόλων μεταξύ Web services κατά τη διάρκεια μιας επικοινωνίας

2.8.3 Web Services Description Language (WSDL)

Οι Web Services χρειάζεται να ορίζονται με συγκεκριμένο τρόπο έτσι ώστε να μπορούν να εντοπιστούν και να χρησιμοποιηθούν από άλλες υπηρεσίες και εφαρμογές. Για αυτό το σκοπό δημιουργήθηκε από τον οργανισμό W3C, μια γλώσσα περιγραφής γνωστή ως *Web Services Description Language* (WSDL).

Η γλώσσα αυτή είναι μια XML δομή εγγράφου, που περιέχει όλες τις πληροφορίες σχετικά με τα δεδομένα εισόδου και εξόδου, τις μεθόδους και ότι άλλο χρειάζεται να γνωρίζει κάποιος για την ακριβή χρήση μιας διαδικτυακής υπηρεσίας.

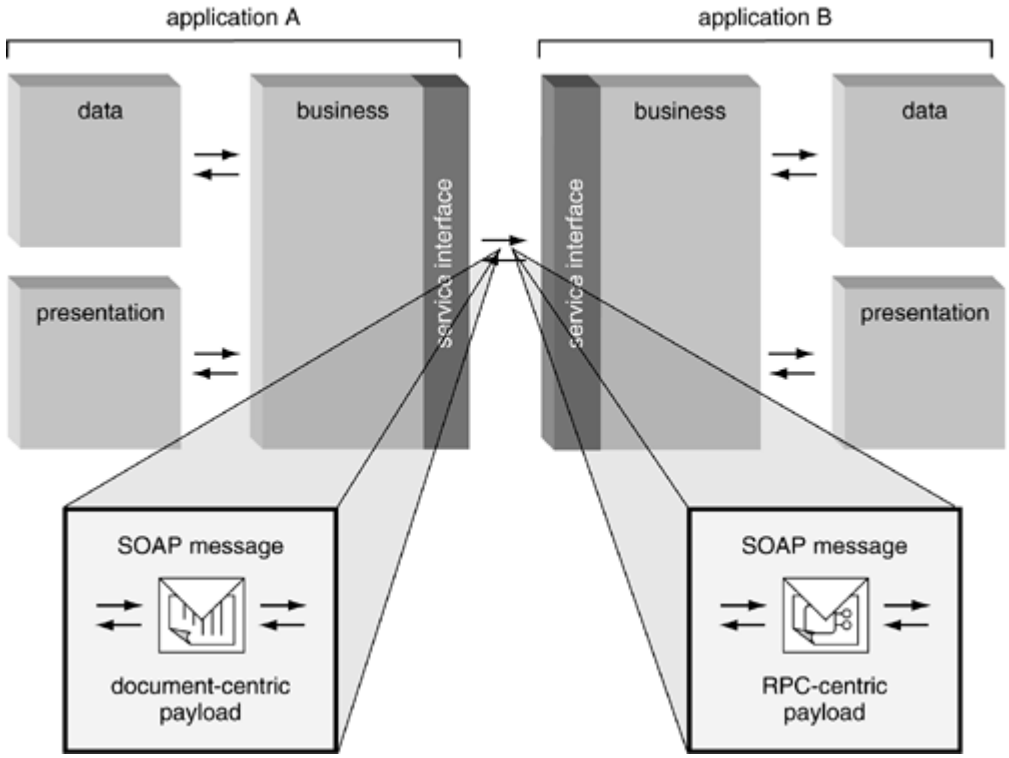
2.8.4 Simple Object Access Protocol (SOAP)

Αν και αρχικά είχε θεωρηθεί ως η τεχνολογία που θα γεφυρώσει το κενό μεταξύ ανόμοιων πλατφορμών βασισμένων σε RPC επικοινωνία, το SOAP έχει εξελιχθεί στο πλέον ευρέως χρησιμοποιούμενο πρότυπο επικοινωνίας για τη χρήση XML Υπηρεσιών Διαδικτύου (Web Services). Μετά από αυτή την κατάσταση γίνεται πολλές φορές η παράφραση του ακρωνύμου από Simple Object Access Protocol σε Service Oriented Architecture (or Application) Protocol.

Το πρωτόκολλο SOAP διαμορφώνει ένα πρότυπο μήνυμα που αποτελείται από ένα XML έγγραφο ικανό να περιγράψει δεδομένα όπως RPC κλήσεις κ.α. Το μήνυμα αυτό μεταφέρεται μεταξύ των υπηρεσιών και των εφαρμογών, χρησιμοποιώντας κυρίως το HTTP πρωτόκολλο δικτύου. Με τον τρόπο αυτό ολοκληρώνεται το πλαίσιο λειτουργίας και

επικοινωνίας στην SOA δομή, αφού με την βοήθεια της περιγραφής WSDL είναι εφικτή η επικοινωνία και συνεργασία οποιωνδήποτε υπηρεσιών στο δίκτυο.

Στο παρακάτω σχήμα (Σχήμα 5) βλέπουμε τη χρήση του πρωτοκόλλου σε εφαρμογές είτε για προτυποποιημένη RPC επικοινωνία είτε για γενική χρήση μεταφοράς μηνύματος.



Σχήμα 5: Χρήση SOAP Πρωτοκόλλου σε εφαρμογές

Το πρωτόκολλο SOAP καθορίζει ένα XML έγγραφο με μια συγκεκριμένη δομή που μέσα του θα εμπεριέχονται οι πληροφορίες. Η βασική δομή ενός τέτοιου εγγράφου είναι η παρακάτω:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
     …
   </soap:Body>
</soap:Envelope>
```

Εσωτερικά του “σώματος” (Body) του εγγράφου, ενσωματώνεται η πληροφορία που θέλουμε να μεταφέρουμε από τη μία υπηρεσία στην άλλη. Η τεχνολογία που χρησιμοποιείται για απομακρυσμένη κλήση υπηρεσιών είναι η SOAP-RPC, (επέκταση της τεχνολογίας Remote Procedure Call) η οποία εισάγει στο SOAP έγγραφο τις απαιτούμενες πληροφορίες, σύμφωνα και με το WSDL της υπηρεσίας που θα κληθεί και στέλνει το μήνυμα. Επιπλέον, όπως φαίνεται και στο σχήμα 5, εκτός από την χρήση που περιγράφηκε, το SOAP μπορεί να μεταφέρει οποιαδήποτε άλλη πληροφορία ενσωματωμένη σε XML δομή, μέσα στο “φάκελο” του SOAP μηνύματος, αρκεί βέβαια ο παραλήπτης να γνωρίζει πώς να αποκωδικοποιήσει το έγγραφο αυτό.

2.9 Σύγκριση Πλέγματος με άλλες τεχνολογίες

Ο σχεδιασμός του Grid έχει βασιστεί σε ένα ανοιχτό σύνολο προτύπων και πρωτοκόλλων τα οποία επιτρέπουν την επικοινωνία μεταξύ ετερογενών και γεωγραφικά κατανεμημένων συστημάτων, όπως για παράδειγμα το Open Grid Services Architecture (OGSA) (για το τελευταίο βλ. ενότητα 2.7).

Η δομή του Grid θεωρούμε ότι αποτελεί εξέλιξη άλλων δομών και τεχνολογιών, όπως τα κατανεμημένα συστήματα (distributed systems), ο παγκόσμιος ιστός (Web), οι υπολογισμοί ομότιμων πόρων (peer-to-peer ή P2P computing) και οι τεχνολογίες δυναμικής εικονικής οργάνωσης (dynamic virtualization). Λόγω αυτού, η τεχνολογία των συστημάτων Grid παρουσιάζει πολλές ομοιότητες με άλλες τεχνολογίες (στις οποίες βασίστηκε) αλλά και πολλές αντιθέσεις. Αναλυτικότερα:

- Όπως και στο Web τα Grids κρύβουν όλη την πολυπλοκότητα, και οι χρήστες απολαμβάνουν μια απλή και υλοποιημένη υπηρεσία. Σε αντίθεση όμως με το Web το οποίο απλά χρησιμεύει για τη διευκόλυνση της επικοινωνίας, τα συστήματα Grid επιτρέπουν πλήρη συνεργασία, με στόχο την επίτευξη κοινών επιχειρηματικών στόχων.
- Όπως και τα P2P συστήματα τα συστήματα Grid επιτρέπουν στους χρήστες να μοιράζονται αρχεία. Σε αντίθεση όμως με τα P2P συστήματα, επιτρέπουν συνεργασίες πολλών με πολλούς (many-to-many) και δεν περιορίζονται στα αρχεία, αλλά χρησιμεύουν για συναλλαγές κάθε είδους πόρων.
- Όπως και τα κατανεμημένα συστήματα (ή clusters) τα Grids ενοποιούν υπολογιστικούς πόρους. Σε αντίθεση με τα κατανεμημένα συστήματα τα οποία χρειάζονται γεωγραφική εγγύτητα και ομοιογένεια των συστημάτων, τα συστήματα Grid μπορεί να είναι γεωγραφικά κατανεμημένα και ετερογενή.
- Όπως και οι τεχνολογίες εικονικής οργάνωσης συστημάτων (virtualization) τα Grids βασίζονται στην εικονική οργάνωση (Ε.Ο.) πόρων. Σε αντίθεση με τις τεχνολογίες εικονικής οργάνωσης συστημάτων οι οποίες έχουν στόχο την οργάνωση και λειτουργία μια αυτόνομης εικονικής μηχανής, τα συστήματα Grid επιτρέπουν την εικονική οργάνωση απέραντων, ανόμοιων, απομακρυσμένων πόρων.

2.10 Τα πλεονεκτήματα του Πλέγματος

Πολλές εταιρίες επιδιώκουν να αξιοποιήσουν τα πλεονεκτήματα της σημερινής δομής του Grid όσον αφορά τα οικονομικά και τα αποδοτικά οφέλη, χωρίς να περιορίζονται σε ένα σύστημα, που δεν θα αυξάνεται σύμφωνα με τις ανάγκες τους.

Για να παρέχουν στους πελάτες την λύση που χρειάζονται, οι επιχειρήσεις έχουν να λύσουν κυρίως προβλήματα που αφορούν την ασφάλεια και την ποιότητα των υπηρεσιών

που παρέχονται στο πελάτη. Χρησιμοποιώντας τα πρότυπα που έχουν αναπτυχθεί, οι πελάτες τους έχουν τη δυνατότητα να κινηθούν σύμφωνα με την συμβατότητα με τις υπάρχουσες νέες τεχνολογίες Grid, αλλά και να υιοθετήσουν καινούργιες καθώς αυτές εξελίσσονται, προσφέροντας νέα πλεονεκτήματα και ευκολίες.

Η πλατφόρμα του Grid μπορεί να συνδυάζει τους περισσευούμενους πόρους σε ένα εταιρικό δίκτυο δημιουργώντας ένα ισχυρό Grid που μπορεί να μοιραστεί και να χρησιμοποιηθεί από ομάδες στην ίδια την εταιρία, ή ακόμη και σε γεωγραφικά απομακρυσμένες σε σχέση με την εταιρία ομάδες. Το πιο κοινό τεχνολογικό στοιχείο που θα μπορούσε να χρησιμοποιηθεί για αυτούς τους πόρους είναι οι ηλεκτρονικοί υπολογιστές γραφείου (desktop PCs), οι οποίοι συνήθως λειτουργούν χαμηλότερα από τις δυνατότητές τους. Το ποσοστό χρήσης της υπολογιστικής τους ισχύς κυμαίνεται στο 10% των δυνατοτήτων τους ακόμα και στις πρωτεύουσες επαγγελματικές τους εφαρμογές. Οργανώνοντας την υπολογιστική υποδομή σε επίπεδα, μπορεί να την χρησιμοποιησουν σαν έσοδο διαθέτοντας την για απαιτητικές εφαρμογές. Αυτό αποτελεί άμεσο όφελος για εταιρίες, που επιθυμούν να υλοποιήσουν τεχνολογίες Grid χωρίς να περιορίζονται από μελλοντικές εξελίξεις και προοπτικές.

Συνοψίζοντας τα κύρια πλεονεκτήματα της τεχνολογίας Grid, κυρίως όσον αφορά την επιχειρηματική εφαρμογή της είναι:

- *Χαμηλότερο κόστος υπολογισμού*, καθώς η σχέση απόδοσης / τιμής είναι διαφορετική. Οι διάφορες πλατφόρμες Grid, ανάλογα με τον προσανατολισμό τους, δίνουν την δυνατότητα εκτέλεσης περισσότερων εργασιών με μικρότερες επενδύσεις σε διοίκηση και σε υλικό σε σχέση με τις συνηθισμένες που βασίζονται αποκλειστικά σε υλικό εξοπλισμό. Το Grid καταφέρνει να βελτιώσει τον λόγο απόδοση προς τιμή σε έναν οργανισμό σε συνάρτηση πάντα από τις διαστάσεις του οργανισμού αυτού.
- *Γρηγορότερα ερευνητικά αποτελέσματα*. Το πλεόνασμα ισχύος που προκύπτει από την πλατφόρμα του Grid μπορεί να δώσει σε έναν οργανισμό την δυνατότητα για γρηγορότερα αποτελέσματα. Αυτό μάλιστα, αν πρόκειται για εταιρία, μπορεί να της δώσει τη δυνατότητα να αυξήσει το πλήθος των εργασιών και να κατακτήσει μεγαλύτερο μερίδιο αγοράς.
- *Καλύτερα αποτελέσματα*. Η αύξηση της ισχύος οδηγεί πολλές φορές σε αναθεώρηση των ερευνητικών προσανατολισμών ώστε να εξετασθούν πολλά υποσχόμενες λύσεις, που προηγουμένως είχαν απορριφθεί λόγω περιορισμών σε χρόνο και χρήμα. Επίσης η ισχύς αυτή μπορεί να βοηθήσει ώστε να προκύψουν αποτελέσματα καλύτερης ποιότητας επιτρέποντας μεγαλύτερες αναλύσεις και δοκιμές.

2.11 Τομείς που ευνοούνται από την τεχνολογία Πλέγματος

Η τεχνολογία Grid έχει ήδη εφαρμοστεί σε ερευνητικό επίπεδο σε διάφορους τομείς και αναμένεται να υιοθετηθεί για την επίτευξη των στόχων που οριοθετούνται σε κάθε τομέα. Αναλυτικότερα:

- Στον τομέα της ιατρικής, της βιολογίας και της γενετικής η τεχνολογία Grid μπορεί να παρέχει σημαντικές υπηρεσίες στη διάγνωση νοσημάτων και στην αξιόπιστη διάγνωση μέσω της επεξεργασίας ιατρικών εικόνων. Εφαρμογές, όπως η διάγνωση με τη βοήθεια ιατρικών απεικονίσεων και η ανάλυση πρωτεϊνών μέσω βιομοριακών μεθόδων, βασίζονται σε μεγάλο βαθμό στην αυτοματοποιημένη συγκέντρωση και επεξεργασία μεγάλου όγκου δεδομένων σε πραγματικό χρόνο, καθώς και στην διανομή των αποτελεσμάτων της επεξεργασίας σε γεωγραφικά διασπαρμένες τοποθεσίες. Το πρόβλημα απαίτησης σε υπολογιστικούς πόρους γίνεται ακόμα εντονότερο αν ληφθεί υπόψη ότι για να επιτευχθεί η επεξεργασία με τη βοήθεια του μεγάλου όγκου των παραγόμενων δεδομένων σε συνθήκες πραγματικού χρόνου αυτά θα πρέπει να συσχετιστούν με επίσης μεγάλο όγκο ιστορικών δεδομένων. Η υιοθέτηση της τεχνολογίας Grid μπορεί να δώσει στις εφαρμογές αυτές τους πόρους που χρειάζονται έτσι ώστε να παράγουν χρήσιμα αποτελέσματα, γρήγορα και αποδοτικά.
- Οι αγορές χρήματος είναι από τους νέους τομείς που έχουν αρχίσει να υιοθετούν την τεχνολογία Grid. Τράπεζες, ασφαλιστικές εταιρείες, χρηματιστηριακές επιχειρήσεις και αναλυτές, όλοι έχουν τη δυνατότητα να βελτιώσουν την ποιότητα των υπηρεσιών που παρέχουν κάνοντας χρήση της τεχνολογίας Grid. Οι τεχνολογίες Grid μπορούν να παρέχουν μεγάλες υπηρεσίες στη μελέτη χρηματοοικονομικών αναλύσεων και επίλυσης σχετικών προβλημάτων όπως ταυτοποίηση ρίσκου σε επενδυτικά σχέδια, διαδικασίες, απαιτήσεις και κόστη, κατηγοριοποίηση, ποσοτικοποίηση, διαχείριση και ανίχνευση ρίσκου.
- Ο επιστημονικός και ο εκπαιδευτικός κλάδος αποτελούν αυτή τη στιγμή τους κύριους χρήστες της Grid τεχνολογίας. Εδώ υπάρχουν δύο κατηγορίες χρηστών του Grid. Η πρώτη περιλαμβάνει ερευνητικά ινστιτούτα και τμήματα που διεξάγουν έρευνα πάνω στην επιστήμη των υπολογιστών. Η κατηγορία αυτή δεν υιοθετεί τις περισσότερες φορές τις εμπορικές λύσεις αλλά δημιουργεί τις δικές της. Τα μέλη της θεωρούν τους εαυτούς τους κυρίως ως παρέχοντες παρά ως καταναλωτές της Grid τεχνολογίας. Η δεύτερη κατηγορία, που σχετίζεται κυρίως με τη δημόσια έρευνα, θεωρείται κυρίως ως καταναλωτής της Grid τεχνολογίας. Τυπικά παραδείγματα εδώ, τα διαστημικά προγράμματα της NASA και η φυσική υψηλών ενεργειών, αντικείμενο έρευνας μεγάλων ινστιτούτων όπως το CERN. Άλλοι τομείς της επιστήμης, που ανήκουν στους συχνούς χρήστες της Grid τεχνολογίας, αποτελούν η χημεία και η μηχανική που περιλαμβάνουν

εφαρμογές υπολογιστικά απαιτητικές ή εφαρμογές που στηρίζονται στην εξόρυξη δεδομένων (data mining).

- Ο βιομηχανικός και κατασκευαστικός τομέας αποτελεί αποδεδειγμένα ένα χώρο, όπου η εφαρμογή της Grid τεχνολογίας μπορεί να αποδειχθεί εξαιρετικά ωφέλιμη. Στη αυτοκινητοβιομηχανία, ο σχεδιασμός αυτοκινήτων και φορτηγών απαιτεί γρήγορους και βελτιστοποιημένους πόρους για να επιταχυνθεί. Στον κατασκευαστικό τομέα γίνεται σε μεγάλο βαθμό η χρήση υπολογιστικών προσομοιώσεων. Τυπικές εφαρμογές που ευνοούνται στην περίπτωση αυτή είναι η δυναμική των ρευστών, ο σχεδιασμός με τη βοήθεια υπολογιστή, ανάλυση πεπερασμένων στοιχείων και προσομοιώσεις συγκρούσεων.

3 *Το μεσολογισμικό GRIA*

Τα μεσολογισμικά (middleware) για το Grid είναι στοίβες λογισμικού, σχεδιασμένες για να παρουσιάζουν ανόμοιους υπολογιστικούς πόρους με ομοιόμορφο τρόπο, έτσι ώστε κάποιο λογισμικό-πελάτης (client software) να μπορεί να έχει πρόσβαση σε αυτούς χωρίς να γνωρίζει a priori τη διαμόρφωση των συστημάτων απ' τα οποία προέρχονται [14]. Το κεφάλαιο αυτό παρουσιάζει θέματα σχετικά με το μεσολογισμικό GRIA, όπως ιστορικά στοιχεία, αρχιτεκτονική και ασφάλεια.

3.1 Ορισμός και χρήση του GRIA

Καθώς το Grid εξελισσόταν υιοθέτησε υπηρεσιοστρεφείς αρχιτεκτονικές οι οποίες θεμελιώθηκαν με το Open Grid Services Architecture (OGSA). Το GRIA (Grid Resources for Industrial Applications) είναι ένα open source μεσολογισμικό, του οποίου η υποδομή είναι υπηρεσιοστρεφής και ως βασικό στόχο έχει την υποστήριξη business-to-business (B2B) συνεργασιών, παρέχοντας υπηρεσίες δια μέσου των ορίων των οργανισμών, με ασφαλή, διαλειτουργικό και ευέλικτο τρόπο. Το GRIA στην ουσία έθεσε το Grid στην υπηρεσία του επιχειρηματικού τομέα, εστιάζοντας σε επιχειρηματικά και εμπορικά μοντέλα.

Το GRIA χρησιμοποιεί Grid Services, τα οποία αποτελούν εξειδίκευση των Web Services, υποστηρίζοντας μακρόβιες και απαιτητικές σε πόρους δραστηριότητες. Οι δραστηριότητες αυτές μπορεί να χρειαστούν συνδυασμένους πόρους πολλαπλών παροχών υπηρεσιών Grid και ίσως διαμοιράζουν αυτούς τους πόρους μεταξύ χρηστών διαφορετικών οργανισμών. Τέτοιες δραστηριότητες απαντώνται σε επιχειρηματικούς τομείς όπως οι επιστήμες υγείας, η παραγωγή media, η εφαρμοσμένη μηχανική και η ακαδημαϊκή έρευνα, ενώ μπορεί να εμπεριέχουν πόρους όλων των ειδών:

- πηγές δεδομένων (data sources)
- αποθήκευση δεδομένων
- εφαρμογές
- επεξεργασία

Το GRIA χρησιμοποιεί επιχειρηματικά μοντέλα, μεθοδολογίες και σημειολογία προκειμένου να επιτρέπει στους παρόχους υπηρεσιών και στους χρήστες να ανακαλύπτει ο ένας τον άλλον και να διαπραγματεύονται τους όρους για πρόσβαση σε υψηλής αξίας υπολογιστικούς πόρους [5]. Εστιάζοντας σε επιχειρηματικές μεθοδολογίες και την αντίστοιχη σημειολογία, το GRIA επιτρέπει στους χρήστες να ικανοποιούν τις υπολογιστικές τους ανάγκες με αποδοτικότητα κόστους και να αναπτύσσουν νέα μοντέλα για τις υπηρεσίες τους.

Συγκεκριμένα το GRIA έχει σχεδιαστεί για να καλύπτει τις ακόλουθες ανάγκες του επιχειρηματικού τομέα:

- Οι πάροχοι υπηρεσιών να λειτουργούν ανεξάρτητα και να συναγωνίζονται για την παροχή υπηρεσιών.
- Οι πελάτες να μπορούν να ελέγχουν ποιες υπηρεσίες θα καταναλώσουν, σε τι ποσότητα και από ποιόν.
- Οι υπηρεσίες να υπόκεινται σε Συμβόλαια Παροχής Υπηρεσιών (Service Level Agreements ή SLAs), ώστε οι πάροχοι να γνωρίζουν τι υπηρεσίες οφείλουν να προσφέρουν και οι πελάτες να γνωρίζουν τι υπηρεσίες θα λάβουν και σε ποια τιμή.
- Να παρέχεται ασφάλεια επιπέδου εμπορικών εφαρμογών.
- Τα λειτουργικά κόστη να ελαχιστοποιούνται.

3.2 Ιστορικά Στοιχεία

Το GRIA project ξεκίνησε το Δεκέμβριο του 2001 με τον σαφή αλλά δυσυπέρβλητο στόχο να θέσει το Grid στην υπηρεσία του βιομηχανικού και του επιχειρηματικού τομέα. Αυτός ο στόχος διαχώρισε το GRIA από την πληθώρα των ακαδημαϊκών υποδομών για το Grid που αναπτύσσονταν εκείνη την εποχή από την κοινότητα του Grid. Ως κυριότερα ζητήματα για τους χρήστες του επιχειρηματικού τομέα προσδιορίστηκαν:

- η ασφάλεια,
- τα επίπεδα υπηρεσιών (service levels)
- η διαλειτουργικότητα (interoperability).

Με τη βοήθεια αναλυτικών case studies που διεξήχθησαν από τους συνεργάτες-μέλη του GRIA προσδιορίστηκαν οι προδιαγραφές και εκτιμήθηκαν τα αποτελέσματα του GRIA project. Αρχικά προτάθηκε η ενσωμάτωση επιχειρηματικών μοντέλων και μεθοδολογιών στην ήδη υπάρχουσα πλατφόρμα Globus Toolkit 2 (GT2). Το Globus αποτελούσε εκείνη την περίοδο το πιο επιτυχημένο και δημοφιλές μεσολογισμικό για το Grid, αλλά ήταν ακόμα προσανατολισμένο στις απαιτήσεις της ακαδημαϊκής κοινότητας η οποία είχε πρωτοστατήσει στην έρευνα για το Grid. Η ενσωμάτωση όμως στο Globus παρουσίασε εξ' αρχής πολλές δυσκολίες, καθώς το λογισμικό αποδείχθηκε ιδιαίτερα δύσκολο στην υλοποίηση, συντήρηση και χρήση.

Κατά τη διάρκεια της πρώτης φάσης του GRIA παρουσιάστηκε το Open Grid Services Architecture (OGSA), όποτε κατέστη προφανές ότι η χρήση του GT2 δεν ήταν ενδεδειγμένη, καθώς αυτό θα αντικαθίστονταν σύντομα από το GT3/OGSA. Ήταν επίσης προφανές ότι θα περνούσε αρκετός καιρός μέχρι το GT3 να γίνει αρκετά σταθερό και

“ώριμο” ώστε να αποτελέσει τη βάση για επιχειρηματικά-εμπορικά συστήματα. Άλλες πλατφόρμες μεσολογισμικών για Grid (όπως το UNICORE) ήταν εκείνη την εποχή υπό ανάπτυξη, όμως δεν υπήρχε ομοφωνία ως προς τη βιωσιμότητα και τη χρησιμότητα τους μετά την παρουσίαση του OGSA.

Σε αυτό το σημείο (2002), το GRIA εγκατέλειψε την πρόταση για ενσωμάτωση στο GT2 και υλοποίησε μία “ελαφριά” (lightweight) υποδομή για Grid χρησιμοποιώντας Web Services προκειμένου να υποστηρίξει file-compute επεξεργασία σε εμπορικό B2B πλαίσιο. Η χρήση Web Services ήταν συνεπής προς τις απαιτήσεις του επιχειρηματικού τομέα και συνεπής προς την απόφαση της κοινότητας Grid να αναπτύξει Grid Services σαν βελτίωση του μοντέλου Web Services.

Η πρώτη πλήρης έκδοση του GRIA, το 2003, εισήγαγε λογιστική της χρήσης (accounting of usage) και τιμολόγηση. Το οικονομικό μοντέλο αντικατόπτριζε τις επιχειρηματικές πρακτικές, απαιτώντας τη δημιουργία λογαριασμού πάροχου (supplier account) πριν από οποιαδήποτε τιμολόγηση ή εμπορική συναλλαγή. Η “λεπτή” αυτή διαδικασία υποστήριζε επίσης διαπραγμάτευση Ποιότητας Υπηρεσίας (Quality of Service - QoS) και το επιπλέον χαρακτηριστικό της “προ-εγκεκριμένης λίστας παρόχων”.

Η αξιολόγηση του project από τελικούς χρήστες, οδήγησε στον προσδιορισμό ποικίλων απαιτήσεων σχετικά με τη διασύνδεση χρήστη (user interface). Στο ένα άκρο ήταν οι χρήστες οι οποίοι απαιτούσαν διασύνδεση command-line ενώ στο άλλο αυτοί που απαιτούσαν εύχρηστο γραφικό περιβάλλον. Η υπάρχουσα έκδοση του GRIA χρησιμοποιούσε διασύνδεση τύπου “wizard” η οποία αδυνατούσε να εξυπηρετήσει αυτές τις ριζικά αντίθετες απαιτήσεις. Προκειμένου να παρέχει μέγιστη ευελιξία, σχεδιάστηκε ένα client-side API, το οποίο επέτρεπε στους χρήστες να γράφουν τα δικά τους client-side προγράμματα για εφαρμογές χρηστών GRIA.

Το GRIA v2 που εκδόθηκε στις αρχές του 2004, παρείχε το προαναφερθέν API για τους έμπειρους χρήστες μαζί με έναν μικρό αριθμό από graphical handlers για τους χρήστες που δεν επιθυμούσαν να χρησιμοποιήσουν τη διασύνδεση command-line. Το API είχε υλοποιηθεί σε Java και μια συμπληρωματική διασύνδεση command-line παρέχονταν προκειμένου οι χρήστες να μπορούν να γράφουν scripts που να εφαρμόζουν τις επιχειρηματικές μεθοδολογίες στο GRIA.

Το GRIA v3 είναι αρχιτεκτονικά παρόμοιο με το GRIA v2, αλλά με πληθώρα βελτιώσεων τόσο στην απόδοση όσο και στη χρηστικότητα. Συγκεκριμένα η έκδοση αυτή περιλαμβάνει ένα portal “Enterprise Client” (υλοποιημένο με το Java API) ώστε από την πλευρά του χρήστη να απαιτείται μόνο ένας συνήθης φυλλομετρητής (browser) για πλήρη πρόσβαση στη λειτουργικότητα του GRIA [4].

Η επόμενη έκδοση GRIA v4 απλοποίησε ακόμα περισσότερο τη διαδικασία εγκατάστασης και χρήσης της λειτουργικότητας του μεσολογισμικού. Η έκδοση αυτή παρείχε πιο εύρωστη υποδομή για Grid βασισμένη στις αρχές του e-Commerce. Υποστήριζε δηλαδή μια λεπτομερή επιχειρηματική μέθοδο για την απόκτηση και χρέωση των υπολογιστικών πόρων. Η έκδοση αυτή γνώρισε μεγάλη επιτυχία και το μοντέλο λογιστικής/QoS που χρησιμοποιούσε αποδείχθηκε ότι πληρούσε σε ικανοποιητικό βαθμό τις απαιτήσεις των τελικών χρηστών. Παρόλα αυτά το μοντέλο περιείχε ατέλειες και οι συμβιβασμοί που συντελέστηκαν κατά την υλοποίηση του δεν ήταν μακροπρόθεσμα αποδεκτοί. Καθώς το βάρος της ευθύνης για την ορθή και ακριβή πρόβλεψη των απαιτήσεων QoS βάραινε τους

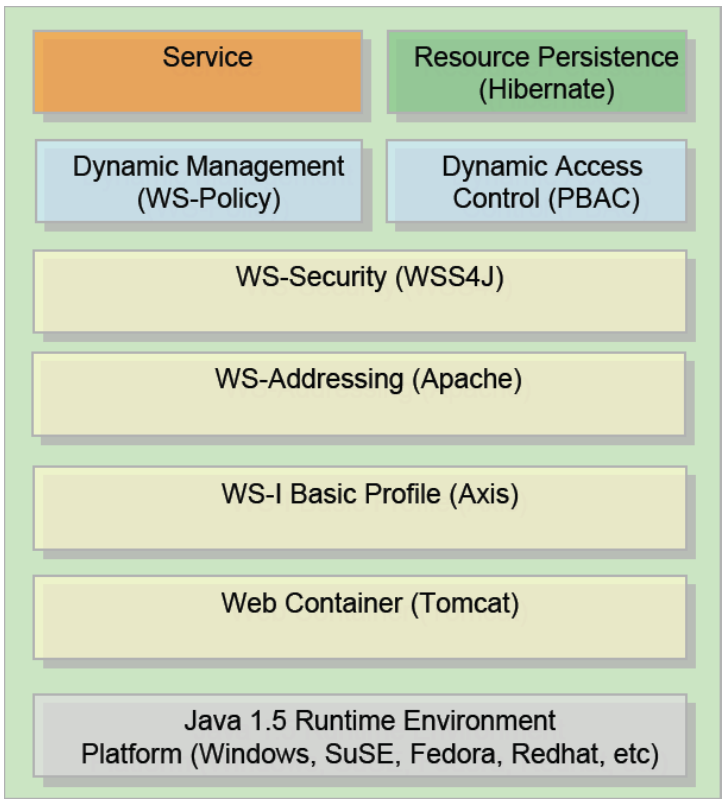
χρήστες, αυτοί είχαν την τάση να τις υπερ-εκτιμούν. Από την πλευρά τους οι πάροχοι υπηρεσιών δεν μπορούσαν να καλύψουν τις “φουσκωμένες” αυτές απαιτήσεις γεγονός που αποτέλεσε στενωπό του μοντέλου.

Η τρέχουσα έκδοση του GRIA v5 αποτελεί την πλέον σταθερή και λειτουργική. Παρέχει αρθρωτή και ευέλικτη υποδομή διαχείρισης, ενώ το λογιστικό μοντέλο έχει επεκταθεί προκειμένου να επιτρέπει την τοπική διαχείριση των χρηστών από τους οργανισμούς στους οποίους ανήκουν. Η πολιτική της υπηρεσίας QoS σταθερών όρων που είχε εισάγει η έκδοση 4, αντικαταστάθηκε από την υπηρεσία διαχείρισης SLAs (Service Level Agreement) η οποία μπορεί να ρυθμιστεί ώστε να παρακολουθεί, να περιορίζει και να χρεώνει ανάλογα με “μετρικές” χρήσης καθορισμένες από των πάροχο υπηρεσιών. Τα SLAs επιτρέπουν την διαπραγμάτευση των όρων του QoS μεταξύ των παρόχων και των χρηστών υπηρεσιών, ώστε οι χρήστες να μπορούν δίνουν ρεαλιστικές εκτιμήσεις των αναγκών τους [5].

3.3 Αρχιτεκτονική

3.3.1 Υποδομή των Web Services

Στην τρέχουσα έκδοση του GRIA (GRIA 5.1) υλοποιείται η ακόλουθη υποδομή των Web Services:



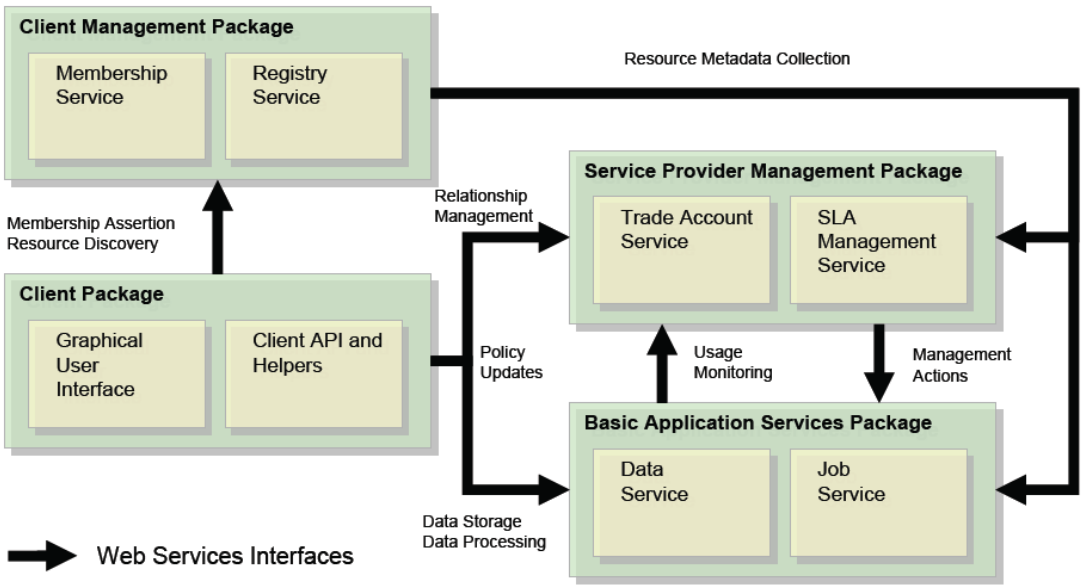
Σχήμα 6: Υποδομή των Web Services του GRIA

Η στοίβα των τεχνολογιών πυρήνα (core) που χρησιμοποιεί το GRIA είναι η ακόλουθη:

- Tomcat 5.5
- Axis 1.4
- WS-Addressing
- WSS4J 1.5
- Hibernate

3.3.2 Υπηρεσίες του GRIA και οι μεταξύ τους Αλληλεπιδράσεις

Όπως διαφαίνεται και στο σχήμα 7, οι υπηρεσίες του GRIA είναι ομαδοποιημένες σε πακέτα, σχεδιασμένα με την υπηρεσιοστρεφή αρχιτεκτονική προκειμένου να υποστηρίζουν ευέλικτη διασύνδεση και επεκτασιμότητα.



Σχήμα 7: Υπηρεσίες του GRIA και οι μεταξύ τους Αλληλεπιδράσεις

Το GRIA διαθέτει τα ακόλουθα 6 πακέτα:

- Basic Application Services Package
- Service Provider Management Package
- Client Package
- Client Management Package
- OGSA-DAI Application Services
- Service Developers' Toolkit

3.3.2.1 Basic Application Services Package

Το πακέτο αυτό επιτρέπει στον πάροχο υπηρεσιών που διαθέτει εγκαταστάσεις cluster computing να παρέχει αποθήκευση και επεξεργασία δεδομένων σε χρήστες που

εμπιστεύεται. Για το σκοπό αυτό στο πακέτο περιλαμβάνονται ένα Job Service και ένα Data Service.

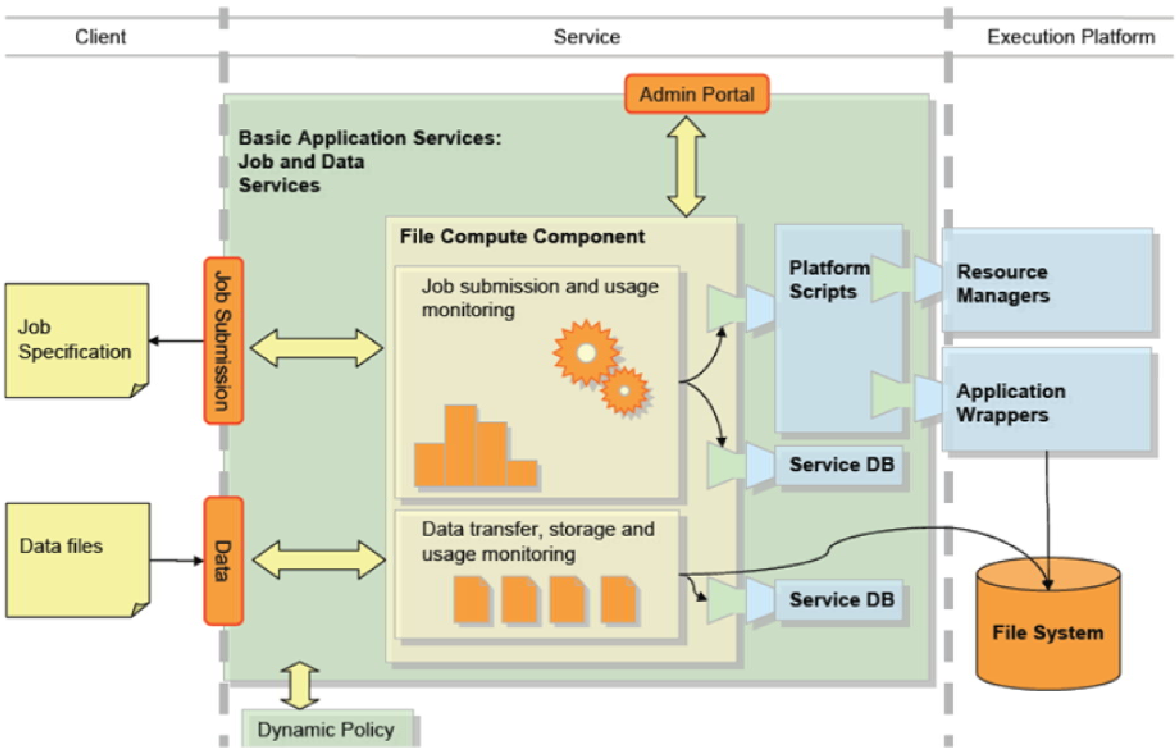
Όσον αφορά το Job Service ισχύουν τα εξής:

- Παρέχει διεπαφή σε ετερογενείς πλατφόρμες εκτέλεσης.
- Υποστηρίζει διαχείριση κύκλου ζωής ενός job (ανακάλυψη, υποβολή, εκτέλεση, παρακολούθηση) από τον απομακρυσμένο χρήστη.
- Μπορεί να ρυθμιστεί από τον πάροχο να υποστηρίζει πολλαπλές εφαρμογές.
- Παρέχει ευέλικτη ολοκλήρωση διαφορετικών διαχειριστών πόρων (όπως LSF, Condor) βασιζόμενο σε scripts.
- Παρέχει ευέλικτη ολοκλήρωση batch εφαρμογών μέσω απλών wrapper scripts.

Ενώ το Data Service:

- Παρέχει διεπαφή σε data stagers βασισμένους σε αρχεία.
- Υποστηρίζει βασικές λειτουργίες αποθήκευσης και μεταφοράς (μέσω SOAP, FTP και HTTP), επιτρέποντας στους απομακρυσμένους χρήστες να “κατεβάζουν” και να “ανεβάζουν” αρχεία δεδομένων από και προς τον πάροχο και να τα μεταφέρον μεταξύ Data Services φιλοξενούμενα από διαφορετικούς παρόχους.
- Παρέχει μια λογική χαρτογράφηση των Endpoint References (EPRs) σε φυσικά ονόματα.

Τέλος το πακέτο αυτό υποστηρίζει διαχείριση βασιζόμενη στα SLA χρησιμοποιώντας παρακολούθηση χρήσης των υπηρεσιών εφαρμογών. Αυτό επιτυγχάνεται καθώς τόσο το Job Service όσο και το Data Service παρακολουθούν (monitor) τις μετρικές χρήσης και δημιουργούν αναφορές για το την υπηρεσία SLA. Οι παραπάνω λειτουργίες παρουσιάζονται στο σχήμα 8.



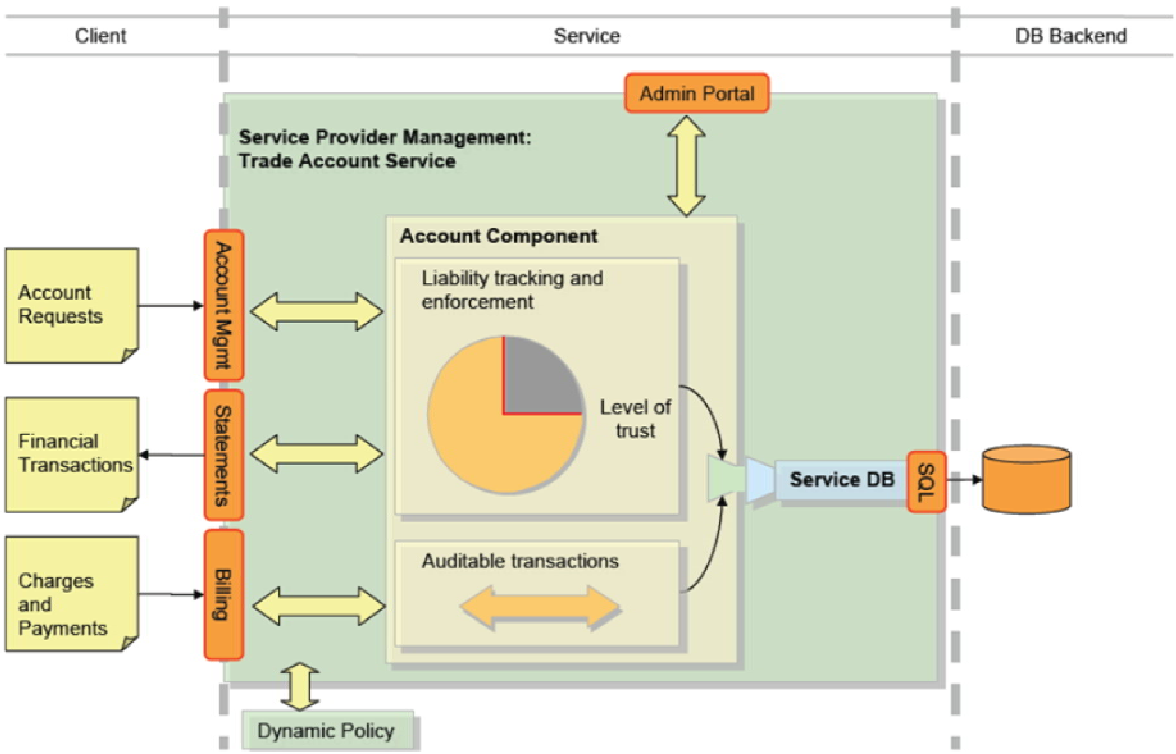
Σχήμα 8: Basic Application Services

3.3.2.2 Service Provider Management Package:

Το πακέτο αυτό παρέχει προαιρετική υποστήριξη για διαχείριση υπηρεσιών με βάση τα SLA καθώς και χρέωση βασισμένη σε ένα απλό πρωτόκολλο διαχείρισης για υπηρεσίες εφαρμογών τόσο του GRIA όσο και μη ενεχόμενων. Για την υλοποίηση των παραπάνω το πακέτο περιλαμβάνει ένα *Trade Account Service* και ένα SLA Management Service.

Όσον αφορά το Trade Account Service ισχύει ότι:

- Επιτρέπει σε έναν πάροχο υπηρεσιών να διαχειρίζεται τις σχέσεις του με τους πελάτες. Οι σχέσεις αυτές (και ως εκ τούτου και η ρίζα της εμπιστοσύνης για την παροχή της υπηρεσίας) αντιπροσωπεύονται από ένα trade account.
- Επιτρέπει στον πάροχο να καθορίζει το όριο της ευθύνης (liability) που είναι διατεθειμένος να ρισκάρει απέναντι σε έναν συγκεκριμένο πελάτη.
- Καταγράφει όλες τις αλλαγές του liability στους λογαριασμούς, δηλαδή καταγράφει χρεώσεις και πληρωμές μετα από καθορισμένο χρονικό διάστημα.
- Επιτρέπει στους πελάτες να παρακολουθούν και να ελέγχουν τους λογαριασμούς τους (μέσω της έκδοσης αναλυτικών λογαριασμών).
- Δεν υπόκεινται σε οικονομικό κανονισμό καθώς δεν αντικαθιστά την τράπεζα.



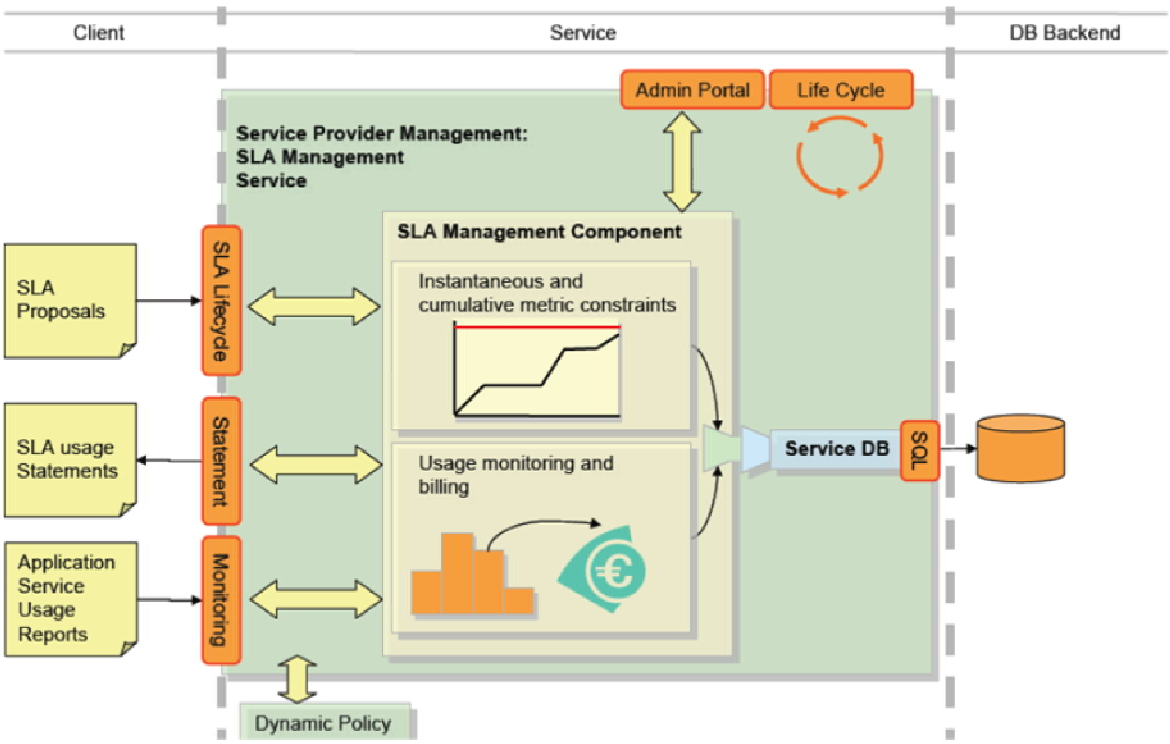
Σχήμα 9: Trade Account Service

Ενώ το SLA Management Service:

- Επιτρέπει στον πάροχο υπηρεσιών να καθορίζει εφαρμογές και όρους Ποιότητας Υπηρεσίας (QoS) στα προσφερόμενα SLAs
- Επιτρέπει στον πελάτη να αποκτήσει υπηρεσίες εφαρμογών από ένα πάροχο υπηρεσιών, μέσω της πρότασης κάποιου SLA βασισμένου στα προσφερόμενα.
- Λειτουργεί σε 3 επίπεδα:

- παρακολούθηση της χρήσης των υπηρεσιών εφαρμογών
- περιορισμού της χρήσης με βάση τους όρους QoS του SLA και τη διαθεσιμότητα
- χρέωση για τη χρήση, βασισμένη στην συνδρομή και τους όρους τιμολόγησης της χρήσης κάποιου πόρου

- Ορίζει τους περιορισμούς και τις αναφορές σε όρους μετρικών (δηλαδή ενός συστήματος παραμέτρων που μετρώνται), οι οποίες καθορίζονται από τον πάροχο υπηρεσιών πάνω στις υπηρεσίες εφαρμογών που παρέχει. Οι μετρικές μπορεί να είναι αθροιστικές ή στιγμιαίες.



Σχήμα 10: SLA Management Service

3.3.2.3 Client Package

Το πακέτο αυτό το οποίο επιτρέπει στο χρήστη να αποκτήσει πρόσβαση στις υπηρεσίες διαχείρισης και εφαρμογών του GRIA μέσω εφαρμογών desktop. Αποτελείται από ένα επεκτάσιμο graphical user interface (GUI) για GRIA και άλλες μη ενεχόμενες υπηρεσίες. Επιπλέον περιλαμβάνει ένα client Java API toolkit που συντελεί στην ολοκλήρωση (integration) των εφαρμογών του πελάτη. Το API αυτό βασίζεται στη διεπαφή Web Service και χρησιμοποιεί δυναμικούς πληρεξούσιους διακομιστές (proxies).

3.3.2.4 Client Management Package

Το πακέτο αυτό οποίο παρέχει προαιρετική υποστήριξή για διαχείριση των χρηστών υπηρεσιών σε επίπεδο οργανισμού (για εσωτερική δηλαδή χρήση), με συγκεντρωτικό έλεγχο και παρακολούθηση (monitoring) της απόκτησης και χρήσης υπηρεσιών. Παρέχει ένα Membership Service, ένα Private Account Service και ένα Registry Service.

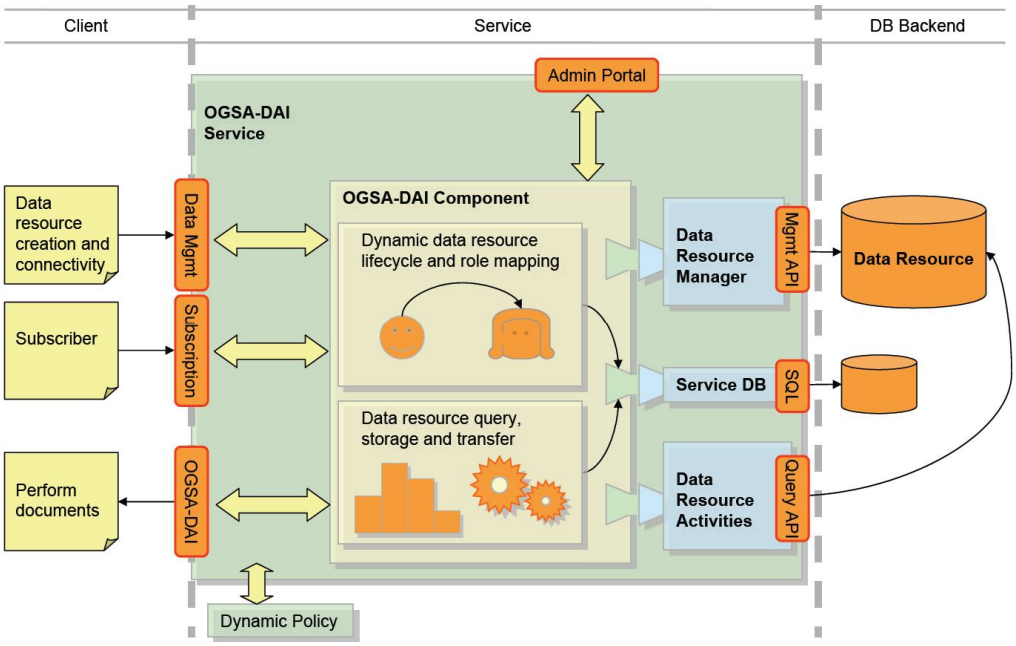
Το Memebership Service χρησιμοποιείται για να διαχειρίζεται ομάδες χρηστών. Οι managers μπορούν να δημιουργήσουν νέες ομάδες και να καθορίσουν ποιοι χρήστες είναι μέλη ποιάς ομάδας. Τα μέλη μιας ομάδας μπορούν να χρησιμοποιήσουν την υπηρεσία για να αποκτήσουν ένα SAML token (βλ. ενότητα 3.4.2), αποδεικνύοντας έτσι ότι είναι μέλη της. Αυτό το token μπορεί να χρησιμοποιηθεί σε άλλες υπηρεσίες συμπεριλαμβανομένων και υπηρεσιών που "τρέχουν" σε άλλα domains.

Το Private Account Service και το Registry Service παρέχουν - για την ώρα - παρόμοια λειτουργικότητα. Και τα δύο επιτρέπουν τη δημιουργία registries και την πρόσθεση πόρων (όπως trade accounts, SLAs και data stagers) σε αυτές. Το Private Account Service περιέχει επιπλέον λειτουργίες, ειδικές στη διαχείριση trade accounts και SLAs, ενώ το Registry Service μπορεί να χρησιμοποιήσει μια βάση δεδομένων eXist σαν οπισθοφυλακή (back-end).

3.3.2.5 OGSA-DAI Application Services

Το πακέτο αυτό επιτρέπει στους παρόχους υπηρεσιών να "προμηθεύουν" σύνθετες υπηρεσίες δεδομένων. Αναλυτικότερα:

- Χρησιμοποιούνται για την παροχή νέων ή ήδη υπάρχοντων data resources.
- Υποστηρίζουν σχεσιακούς, XML και κατά παραγγελία data resources.
- Παρέχουν ευέλικτη αρχιτεκτονική για ολοκλήρωση νέων τύπων data resource, χρησιμοποιώντας Διαχείριση Πόρων Δεδομένων (Data Resource Management) και Δραστηριότητες OGSA-DAI.
- Βασίζονται στην υπηρεσία OGSA-DAI του Πανεπιστημίου του Εδιμβούργου.
- Παρέχουν πρόσβαση στα data resources βάσει συνδρομών ρόλων σύμφωνα με την ακόλουθη διαδικασία:
    - Οι πάροχοι υπηρεσιών καθορίζουν ρόλους σε ήδη υπάρχοντα συστήματα διαχείρισης δεδομένων
    - Οι χρήστες με τη σειρά τους γίνονται συνδρομητές των ρόλων αυτών, γεγονός που τους επιτρέπει να έχουν πρόσβαση σε μία όψη των data resources. Αξίζει να σημειωθεί εδώ ότι η χαρτογράφηση μεταξύ πελατών και ρόλων είναι δυναμική.
- Υποστηρίζουν διαχείριση βασιζόμενη στα SLA διαμέσου της παρακολούθησης (monitoring) της χρήσης των υπηρεσιών εφαρμογών. Συγκεκριμένα:
    - παρακολουθούν τη χρήση μετρικών και δημιουργούν αναφορές για το SLA Service.
    - οι βασικές μετρικές περιλαμβάνουν των αριθμών των συνδρομών που υποστηρίζει κάθε ρόλος, καθώς και των αριθμό των συνδρομητών.
    - μπορούν να επεκταθούν για να συμπεριλάβουν περαιτέρω μετρικές όπως η διάρκεια μιας συναλλαγής.
- Διαθέτουν το σημαντικό πλεονέκτημα της απλής ρύθμισης (configuration) μέσω του web management portal του GRIA όπως διαπιστώνεται και από το σχήμα 11.

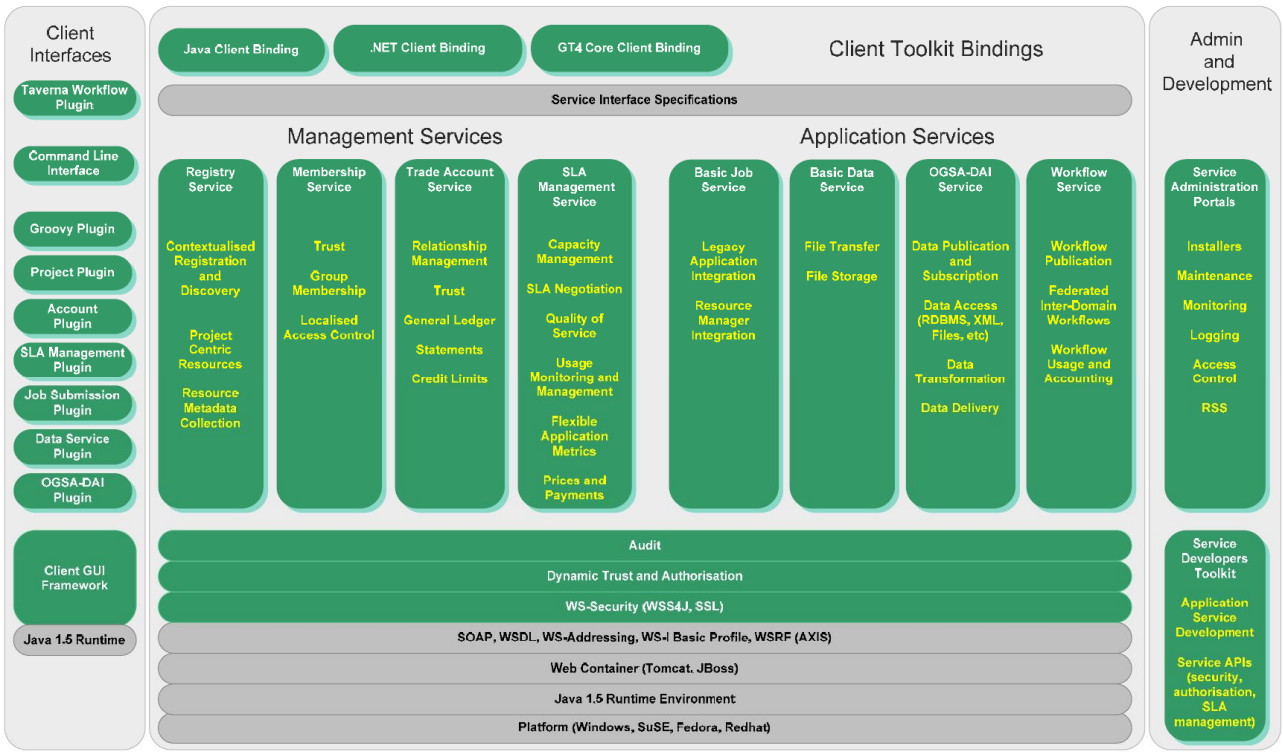


Σχήμα 11: OGSA-DAI Service

3.3.2.6 Service Developers' Toolkit

Το πακέτο αυτό επιτρέπει την ενοποίηση (integration) των υπηρεσιών εφαρμογών του χρήστη με την ασφάλεια και τις δυνατότητες διαχείρισης του GRIA.

Λαμβάνοντας υπόψη την παραπάνω ανάλυση, συγκεντρωτικά η αρχιτεκτονική της τρέχουσας έκδοσης του GRIA παρουσιάζεται στο σχήμα 12.



Σχήμα 12: Συγκεντρωτική Αρχιτεκτονική του GRIA 5.1

3.4 Ασφάλεια

3.4.1 Μια αποδεδειγμένη προσέγγιση

Η προσέγγιση του GRIA ως προς την ασφάλεια (security) έχει σχεδιαστεί κατά κύριο λόγο για να διατηρήσει την ανεξαρτησία μεταξύ των sites κρατώντας παράλληλα τα λειτουργικά κόστη σε χαμηλό επίπεδο. Το GRIA 5 έχει ενσωματωμένους μηχανισμούς ασφάλειας των Web Services προκειμένου να επιβεβαιώνει την ταυτότητα και τους ρόλους των χρηστών που επιχειρούν να αποκτήσουν πρόσβαση σε κάποια υπηρεσία. Όπως οι περισσότερες εφαρμογές e-Commerce, αλλά σε αντίθεση με κάποια άλλα Grids, το GRIA δεν παρέχει ούτε καν στους πιστοποιημένους χρήστες απευθείας δυνατότητα χρήσης δικών τους εντολών πάνω σε υπολογιστικούς κόμβους. Αντίθετα, οι χρήστες οφείλουν να ζητήσουν από την υπηρεσία να “τρέξει” εντολές εκ μέρους τους και by default οι μόνες εντολές που υποστηρίζονται είναι αυτές που έχει προσδιορίσει ο πάροχος της υπηρεσίας ως υπηρεσίες εφαρμογών.

Αυτό σημαίνει ότι οι πάροχοι υπηρεσιών δεν χρειάζεται να δημιουργήσουν τοπικούς λογαριασμούς για κάθε χρήστη (που θα οδηγούσε σε σημαντικό λειτουργικό κόστος). Επίσης περιορίζει την εμπιστοσύνη που πρέπει να δείξουν οι πάροχοι στους χρήστες τους, επιτρέποντας τους να παραμένουν ανεξάρτητοι ο ένας από τον άλλον ακόμα και όταν αλληλεπιδρούν με άλλους παρόχους εκ μέρους κοινών χρηστών. Τέλος, η παραπάνω προσέγγιση συνεπάγεται ότι οι πάροχοι υπηρεσιών μπορούν να εξασφαλίσουν ότι οι εφαρμογές που χρησιμοποιούνται από το site τους διαθέτουν άδεια χρήσης, αποφεύγοντας έτσι το ρίσκο νομικών μέτρων εναντίον τους από τους προμηθευτές λογισμικού.

Το GRIA 5 διαθέτει στοιχεία δυναμικής διαχείρισης πολιτικής ώστε να προσδιορίζει και να επιβάλλει πολιτικές για την πρόσβαση σε υπηρεσίες, σε διαδικασίες διαχείρισης πελατών (consumer management), όπως επίσης και σε κάθε ξεχωριστή μονάδα δεδομένων και επεξεργασίας. Αυτή η δυναμική πολιτική επιτρέπει στους χρήστες να καταστρώσουν αλληλεπιδράσεις μεταξύ υπηρεσιών χειριζόμενων από εντελώς ανεξάρτητους παρόχους και αν χρειαστεί να μοιράζονται τα αποτελέσματα με άλλους χρήστες. Το πακέτο διαχείρισης πελατών (consumer management package) του GRIA 5 υποστηρίζει επίσης ενοποίηση με την πιστοποίηση ενός “home site”, ώστε οι πολιτικές υπηρεσιών να μπορούν να καθορίζουν ποια sites θα εμπιστεύονται για να πιστοποιούν τους χρήστες έναντι σε κάθε όρο της εκάστοτε πολιτικής. Αυτό καθίστα πολύ εύκολή τη διαδικασία εγκατάστασης επιχειρηματικών σχέσεων, με αποτέλεσμα να μπορεί ξεκινήσει η παροχή υπηρεσιών σε νέους χρήστες μέσα σε λίγα μόλις λεπτά.

Τέλος, διατηρώντας την ανεξαρτησία μεταξύ των παρόχων υπηρεσιών, το GRIA τους επιτρέπει να είναι υπεύθυνοι για την δική τους ασφάλεια. Αντίθετα με πολλά ακαδημαϊκά Grids δεν υπάρχουν μοναδικά σημεία αποτυχίας (failure) και εάν ένα GRIA site υποστεί παραβίαση ασφαλείας, το πρόβλημα μπορεί να αντιμετωπιστεί τοπικά και τα υπόλοιπα sites δεν χρειάζεται να σταματήσουν τη λειτουργία τους. Έτσι, η προσέγγιση χαμηλής εξάρτησης που υιοθετεί το GRIA εξασφαλίζει υψηλό βαθμό ανοχής στις εισβολές, το οποίο είναι θεμελιώδες για τους επιχειρηματίες παρόχους υπηρεσιών Grid.

3.4.2 Τεχνολογίες ασφαλείας που υιοθετεί το GRIA

Το GRIA 5 χρησιμοποιεί ποικίλους μηχανισμούς ασφαλείας και σε πολλά επίπεδα. Συγκεκριμένα υιοθετεί:

- ασφάλεια επιπέδου μεταφοράς, μέσω κρυπτογραφημένων Secure Socket Layer (SSL) ή Transport Layer Security (TLS) συνδέσεων.
- ασφάλεια επιπέδου μηνύματος, μέσω του WS-Security, το οποίο αποτελεί ένα σετ από επεκτάσεις SOAP που παρέχουν ακεραιότητα, εμπιστευτικότητα και πιστοποίηση επιπέδου μηνύματος.
- SAML, που αποτελεί ένα XML standard για την ανταλλαγή δεδομένων πιστοποίησης και εξουσιοδότησης μεταξύ ενός παρόχου ταυτότητας (identity provider) και ενός παρόχου υπηρεσιών (service provider).
- X.509 πιστοποιητικά, δηλαδή πιστοποιητικά υποδομής κοινού κλειδιού όπως αυτή έχει οριστεί από την ITU-T.
- πολιτικές δυναμικής πρόσβασης επιπέδου resource (PBAC).

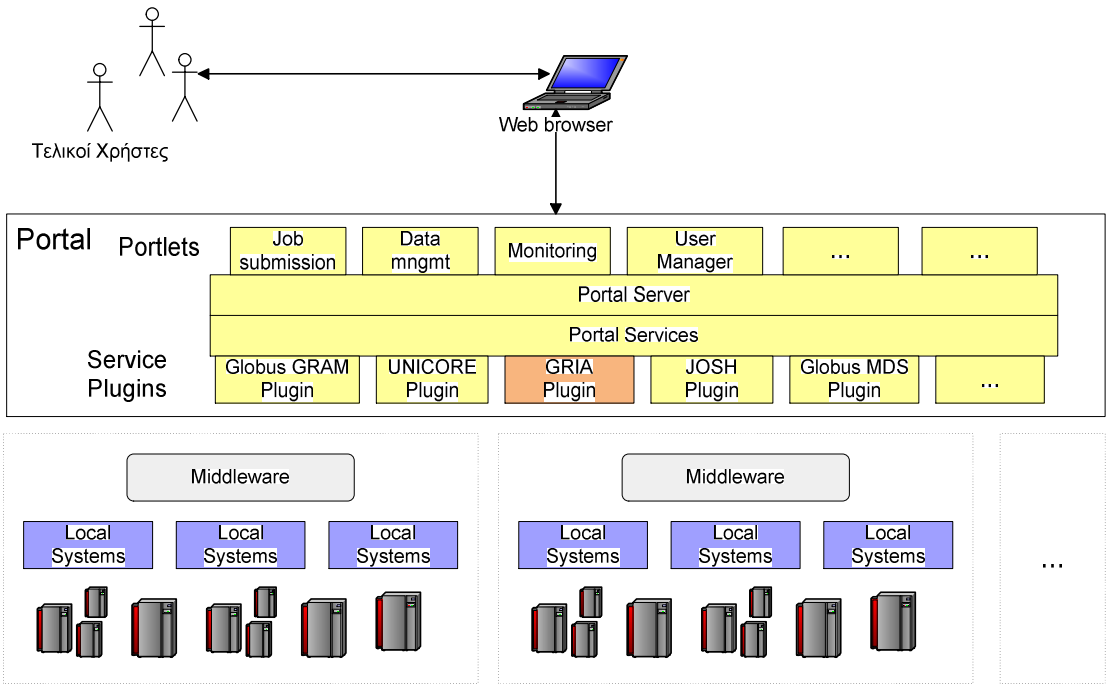
4 Grid Portals

Με στόχο την παροχή ενιαίας, ευέλικτης και απρόσκοπτης πρόσβασης των τελικών χρηστών στους πόρους του Grid από οποιοδήποτε σημείο χρησιμοποιούνται ευρέως portals. Προκειμένου τα portals να επικοινωνούν με διαφανή τρόπο με τους ανομοιογενείς πόρους του Grid, απαιτήθηκε ένας μηχανισμός δυναμικής φόρτωσης των συστατικών στοιχείων (components) που επιτρέπουν την πρόσβαση στη λειτουργικότητα ενός συγκεκριμένου μεσολογισμικού Grid, μέσω μιας κοινής ενιαίας διεπαφής. Τα components αυτά ονομάζονται plug-ins. Στα πλαίσια της διπλωματικής εργασίας υλοποιήθηκαν plug-ins για την επικοινωνία των Grid Portals με το μεσολογισμικό GRIA. Στο κεφάλαιο αυτό παρουσιάζονται γενικά στοιχεία σχετικά με την αρχιτεκτονική των Grid Portals και την κοινή τους λειτουργικότητα.

4.1 Αρχιτεκτονική

Οι τελικοί χρήστες του Grid χρειάζονται ένα περιβάλλον το οποίο οφείλει από τη μία να απλοποιεί τη χρήση των Grid και από την άλλη να διευκολύνει τη διασύνδεση (plug-in) νέων υπηρεσιών. Στους χρήστες οφείλει να παρέχεται μια φιλική και εύκολη στη χρήση διεπαφή, επιτρέποντας τους να αλληλεπιδρούν με τις υπηρεσίες αυτές μέσω ενός συνήθους φυλλομετρητή (browser).

Τα συστατικά στοιχεία ενός portal ονομάζονται portlets. Κάθε portlet αποτελεί μία “αυτόνομη” διαδικτυακή εφαρμογή, ενώ ο ρόλος του portal είναι η συγκέντρωση, ενοποίηση και αποτελεσματική παρουσίαση τους. Τα portlets επικοινωνούν (μέσω του server του portal) με τις καθοριζόμενες από το χρήστη υπηρεσίες, επιλέγοντας δυναμικά το plug-in για την πρόσβαση στο κατάλληλο μεσολογισμικό. Η αρχιτεκτονική ενός τέτοιου portal φαίνεται στο σχήμα 13.



Σχήμα 13: Αρχιτεκτονική των Grid Portals

Οι τελικοί χρήστες έχουν τη δυνατότητα, μέσω του portal, να χρησιμοποιούν τα portlets για να αποκτήσουν πρόσβαση στην απαιτούμενη λειτουργικότητα (όπως job submission, data management, infrastructure monitoring κ.α.) [16]. Τα αιτήματα των χρηστών μεταφράζονται μέσω των Portal Services και των Plug-ins σε εντολές και δεδομένα κατανοητά από το ένα συγκεκριμένο μεσολογισμικό (όπως το UNICORE, το GRMS και το GRIA).

Η παραπάνω αρχιτεκτονική μπορεί να χρησιμοποιηθεί βάσει 2 θεμελιωδών σεναρίων χρήσης.

Σύμφωνα με το 1^{ο} σενάριο:

- Ένας τελικός χρήστης κάνει log-in στο portal και του παρέχεται η βασική λειτουργικότητα που είναι εξουσιοδοτημένος να χρησιμοποιήσει.
- Ο χρήστης επιλέγει τη λειτουργία που θέλει να χρησιμοποιήσει (όπως job submission, resource monitoring κ.λπ.).
- Το κοινό portlet για τη λειτουργία που επέλεξε ο χρήστης φορτώνεται.
- Ο χρήστης εκτελεί την επιλεγμένη λειτουργία χρησιμοποιώντας μόνο τη γενική λειτουργικότητα που του παρέχει το portal (π.χ. προετοιμάζει και κάνει submit ένα job σε ένα συγκεκριμένο site).
- Εφόσον ο χρήστης έχει ολοκληρώσει τη λειτουργία που επιθυμούσε, δε χρειάζεται να ασχοληθεί με το υποκείμενο μεσολογισμικό, τα τοπικά συστήματα κ.ο.κ.

Συμφωνά με το 2^{ο} σενάριο:

- Ένας τελικός χρήστης κάνει log-in στο portal και του παρέχεται η βασική λειτουργικότητα που είναι εξουσιοδοτημένος να χρησιμοποιήσει.

- Ο χρήστης επιλέγει τη λειτουργία που θέλει να χρησιμοποιήσει (όπως job submission, resource monitoring κ.λπ.).
- Το κοινό portlet για τη λειτουργία που επέλεξε ο χρήστης φορτώνεται.
- Ο χρήστης επιλέγει σε ποιο site επιθυμεί να εκτελέσει την επιλεγμένη λειτουργία.
- Φορτώνεται ένας σύνδεσμος (link) σε ένα portlet εξειδικευμένο για το μεσολογισμικό που χρησιμοποιείται από το επιλεγμένο site.
- Ο χρήστης χρησιμοποιεί το εξειδικευμένο αυτό portlet για να εκτελέσει τη λειτουργία βάσει της πλήρους λειτουργικότητας που του παρέχει το μεσολογισμικό του επιλεγμένου site.

4.2 Μελέτη Κοινών Τεχνικών Απαιτήσεων και Λειτουργικότητας

Κατόπιν εξονυχιστικών μελετών [17] και λαμβάνοντας υπόψη τις κοινές τεχνικές και επιχειρηματικές απαιτήσεις της κοινότητας του Grid προσδιορίστηκε η κοινή λειτουργικότητα που οφείλει να υλοποιείται από τα Grid Portals προκειμένου να καλύπτονται οι βασικές ανάγκες με τον αποδοτικότερο τρόπο.

4.2.1 Κοινές Τεχνικές Απαιτήσεις

Οι προσδιορισμένες κοινές απαιτήσεις για τα portals μπορούν να οργανωθούν στις τρεις ακόλουθες κατηγορίες:

- Administration
- Διαχείριση Δεδομένων
- Διαχείριση Εργασιών (Jobs)

Βάσει αυτής της οργάνωσης, οι Κοινές Τεχνικές Απαιτήσεις παρουσιάζονται στον πινάκα 1 και αναλύονται στις ακόλουθες ενότητες.

Κατηγορία	**Κοινή Τεχνική Απαίτηση**
Administration	Ασφάλεια (Security)
	Διαχείριση Χρηστών
	Accounting

Διαχείρισ Δεδομένων	Διαχείριση Αρχειων
	Πρόσβαση στις Βάσεις Δεδομένων
Διαχείριση Εργασιών	Υποβολή Εργασιών (Job Submission)
	Παρακολούθηση & Έλεγχος Εργασιών (Job Monitoring & Control)
	Οπτικοποίηση Εργασιών (Job Visualisation)

Πίνακας 1: Κοινές Τεχνικές Απαιτήσεις των Grid Portals

4.2.1.1 Ασφάλεια

Αυτή η κοινή τεχνική απαίτηση καλύπτει τους υποκείμενους μηχανισμούς πιστοποίησης και εξουσιοδότησης που είναι απαραίτητη για τα Grid Portals. Αυτή η τεχνική απαίτηση περιλαμβάνει (αλλά δεν περιορίζεται σε):

- Διαδικασίες login και logout
- Πρόσβαση σε περιεχόμενο που φιλοξενείται από το portal
- Πρόσβαση σε εξωτερικό περιεχόμενο ή πόρους που γίνονται προσβάσιμοι από το portal
- Ενοποίηση με μη ενεχόμενες υπηρεσίες ασφαλείας

Υπάρχει πληθώρα επιχειρηματικών αναγκών για ασφάλεια στο portal, ανάλογα με την περίπτωση χρήσης (use case) που εξετάζεται. Πρώτη και κυριότερη είναι η ανάγκη πιστοποίησης των χρηστών του portal. Σημαντική είναι επίσης και η ανάγκη παροχής πιστοποίησης “single sing-on” για εξωτερικούς πόρους προσβάσιμους από τους χρήστες. Η μέθοδος “single sing-on” επιτρέπει στο χρήστη να πιστοποιεί μόνο μία φορά την ταυτότητα του και να αποκτά πρόσβαση σε πόρους διαφορετικών συστημάτων (που φυσιολογικά θα απαιτούσαν εκ νέου πιστοποίηση).

4.2.1.2 Διαχείριση Χρηστών

Αυτή η κοινή τεχνική απαίτηση καλύπτει την ανάγκη για διαδικτυακές διεπαφές (web interfaces) που να διαχειρίζονται τους λογαριασμούς των χρηστών και των ομάδων χρηστών, καθώς και την πρόσβαση τους σε περιεχόμενο ή πόρους. Περιλαμβάνεται επίσης η ανάγκη των χρηστών να διαχειρίζονται τις προσωπικές τους πληροφορίες και να βλέπουν τις πληροφορίες των άλλων χρηστών

Η επιχειρηματική ανάγκη αυτής της απαίτησης περιλαμβάνει την εύκολη και αποδοτική διαχείριση των λογαριασμών χρηστών του portal από τις ίδιες τις επιχειρήσεις. Οι χρήστες μπορεί να είναι υπάλληλοι μιας επιχείρησης, πελάτες ή το ευρύ κοινό.

4.2.1.3 Accounting

Αυτή η τεχνική απαίτηση καλύπτει την ανάγκη των διεπαφών του διαδικτύου για προβολή λογιστικών (accounting) πληροφοριών σχετικά με τη χρήση των πόρων. Περιλαμβάνει επίσης την ανάγκη online χρέωσης και πληρωμής για χρήση πόρων.

Οι επιχειρηματικές ανάγκες που καλύπτονται από αυτή την απαίτηση περιλαμβάνουν την ανάγκη να υπάρχουν δράστες (actors) οι οποίοι να προσδιορίζονται από τις επιχειρήσεις και να μπορούν να εκτελέσουν πληρωμές για τη χρήση πόρων που αφορούν τα portals. Οι δράστες μπορεί να έχουν τη δυνατότητα κερδοφορίας από τη χρήση των πόρων.

4.2.1.4 Διαχείριση Αρχείων

Αυτή η τεχνική απαίτηση καλύπτει τη λειτουργικότητα που αφόρα την ανάγκη:

- “ανεβάσματος” (upload) αρχείων σε μια διαδικτυακή διεπαφή.
- “κατεβάσματος” (download) αρχείων από μια διαδικτυακή διεπαφή.
- πρόσβαση (προβολή, επεξεργασία, διαγραφή ή οποιαδήποτε άλλη χρήση) σε αρχεία μιας διαδικτυακής διεπαφής.

4.2.1.5 Πρόσβαση στις Βάσεις Δεδομένων

Αυτή η τεχνική απαίτηση καλύπτει την ανάγκη πρόσβασης σε δεδομένα διαφορετικών και ετερογενών πηγών. Γενικά, αυτό περιλαμβάνει την ανάγκη εκτέλεσης ενεργειών σε ποίκιλες βάσεις δεδομένων (σχεσιακού τύπου, αντικειμενοστραφείς, βασισμένες σε XML κ.α.) τις οποίες διαχειρίζονται μια ή περισσότερες μη ενεχόμενες οντότητες. Μπορεί όμως και να περιλαμβάνει την ανάγκη πρόσβασης σε δομημένα δεδομένα που περιέχονται σε αρχεία συγκεκριμένου τύπου.

Η πρόκληση που προκύπτει από αυτή την τεχνική απαίτηση είναι να παρέχεται η δυνατότητα στους χρήστες να έχουν πρόσβαση σε δεδομένα από πολλαπλές πηγές, χωρίς να ενδιαφέρονται για τους υποκείμενους μηχανισμούς ασφαλείας, τα πρωτόκολλα και τον τύπο των αποθηκευμένων δεδομένων.

4.2.1.6 Υποβολή Εργασιών

Αυτή η τεχνική απαίτηση καλύπτει την ανάγκη για διεπαφές διαδικτύου οι οποίες θα επιτρέπουν στο χρήστη να υποβάλλει εργασίες προς εκτέλεση σε υπολογιστικούς πόρους.

Μια άλλη λειτουργικότητα που καλύπτεται είναι η διαπραγμάτευση των SLAs, η οποία μπορεί να θεωρηθεί σαν ειδική λειτουργικότητα υποβολής εργασιών.

Η βασική πρόκληση που ανακύπτει από αυτή την απαίτηση είναι ο σχεδιασμός ενός portal με βασική λειτουργικότητα υποβολής εργασιών. Το portal αυτό θα πρέπει να είναι παραμετροποιήσιμο και να διαθέτει δυνατότητες επέκτασης έτσι ώστε να μπορεί να ανταπεξέλθει στην πληθώρα αναγκών της κοινότητας του Grid.

4.2.1.7 Παρακολούθηση και Έλεγχος Εργασιών

Αυτή η τεχνική απαίτηση καλύπτει την ανάγκη παρακολούθησης (monitoring) και ελέγχου των εργασιών καθώς αυτές εκτελούνται και συγκεκριμένα:

- Προβολή της κατάστασης (status) της εργασίας.
- παύση, συνέχιση ή ακύρωση των εργασιών καθώς αυτές εκτελούνται.
- προβολή της ιστορίας εργασιών.

Η βασική πρόκληση που ανακύπτει από αυτή τη απαίτηση είναι ο σχεδιασμός ενός γενικού portal παρακολούθησης και ελέγχου εργασιών, το οποίο θα έχει τη δυνατότητα να υλοποιεί την προαναφερθείσα λειτουργικότητα σε υποβληθείσες εργασίες. Σε συνδυασμό με το Accounting Portal, μπορεί να διασυνδεθεί η εκτέλεση κάθε εργασίας με τις πληροφορίες χρέωσης της.

4.2.1.8 Οπτικοποίηση των Εργασιών

Αυτή η τεχνική απαίτηση καλύπτει την ανάγκη για παρουσίαση online οπτικοποιήσεων των αποτελεσμάτων των εργασιών, όπως π.χ. αποτελέσματα προσομοίωσης.

Η βασική πρόκληση που ανακύπτει από εδώ είναι ο σχεδιασμός ειδικευμένων portal οπτικοποίησης εργασιών διαφορετικών ειδών. Τα online εργαλεία οπτικοποίησης αποτελούν σημαντική επιχειρηματική ανάγκη για πολλούς χρήστες οι οποίοι επιθυμούν να δουν το “χοντρικό” αποτέλεσμα υποβολής μιας εργασίας και να αποφασίσουν εάν υπάρχει η ανάγκη επανάληψης της εκτέλεσης της, χωρίς να “κατεβάσουν” τεράστια αρχεία σε μέγεθος.

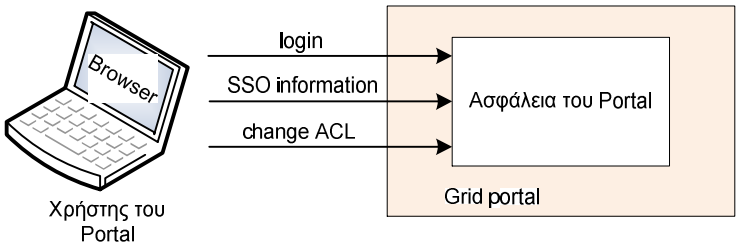
4.2.2 Κοινή Λειτουργικότητα

Εδώ παρουσιάζεται η κοινή λειτουργικότητα των Grid Portals όπως αυτή προέκυψε από την ανάλυση των κοινών τεχνικών και επιχειρηματιών απαιτήσεων της προηγούμενης ενότητας. Τα Domain diagrams (όπου αυτά χρησιμοποιούνται), παρουσιάζουν τη σχέση ανάμεσα στην κοινή λειτουργικότητα και τις κοινές τεχνικές απαιτήσεις.

4.2.2.1 Κοινή Λειτουργικότητα Ασφάλειας

4.2.2.1.1 Περιγραφή

Αυτή η κοινή λειτουργικότητα καλύπτει τους υποκείμενους μηχανισμούς πιστοποίησης και εξουσιοδότησης, όπως αυτοί προσδιορίστηκαν από την κοινή τεχνική απαίτηση ασφάλειας (βλ. ενότητα 4.2.1.1). Το σχήμα 13 παρουσιάζει αυτή την κοινή λειτουργικότητα:



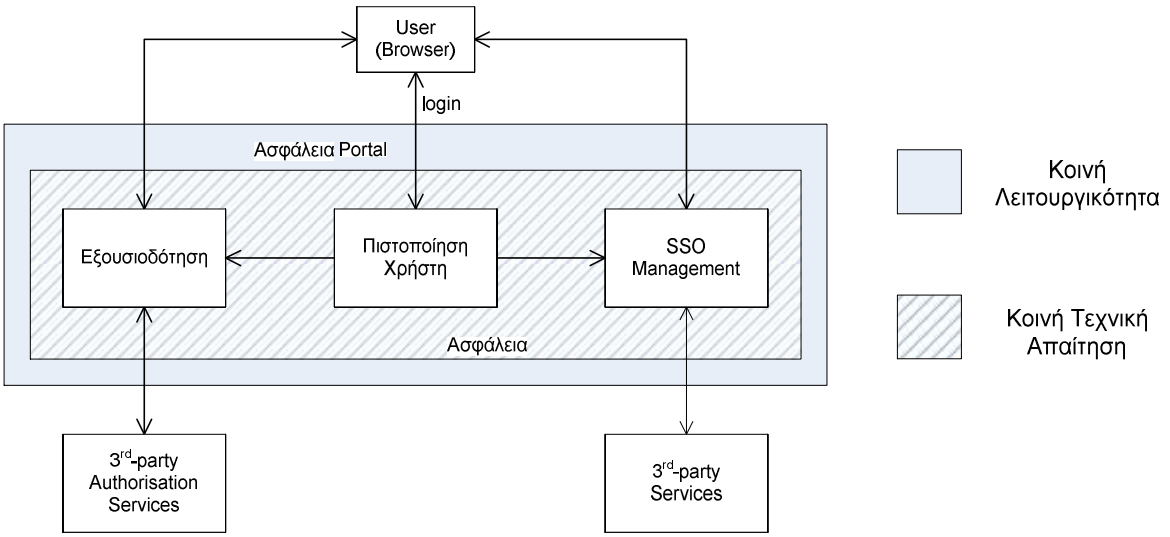
Σχήμα 14: Κοινή Λειτουργικότητα Ασφαλείας

Η κοινή λειτουργικότητα επιτρέπει σε ένα χρήστη:

- Να πιστοποιήσει την ταυτότητα του (login)
- Να εξουσιοδοτήσει λειτουργίες χρηστών στο portal χρησιμοποιώντας μη ενεχόμενα συστήματα εξουσιοδότησης ή Role Based Access Control (RBAC) βασιζόμενο σε μηχανισμούς του portal.
- Να χρησιμοποιήσει ολοκληρωμένες μη ενεχόμενες υπηρεσίες ασφαλείας.
- Να εκτελέσει όλες τις ακόλουθες λειτουργίες χρησιμοποιώντας τον μηχανισμό “Single Sign-On” (SSO).

4.2.2.1.2 Σχεδιαστικό Μόρφημα (Design Pattern)

Αρχικά παρουσιάζεται στο σχήμα 15 το domain diagram αυτού του μορφήματος.

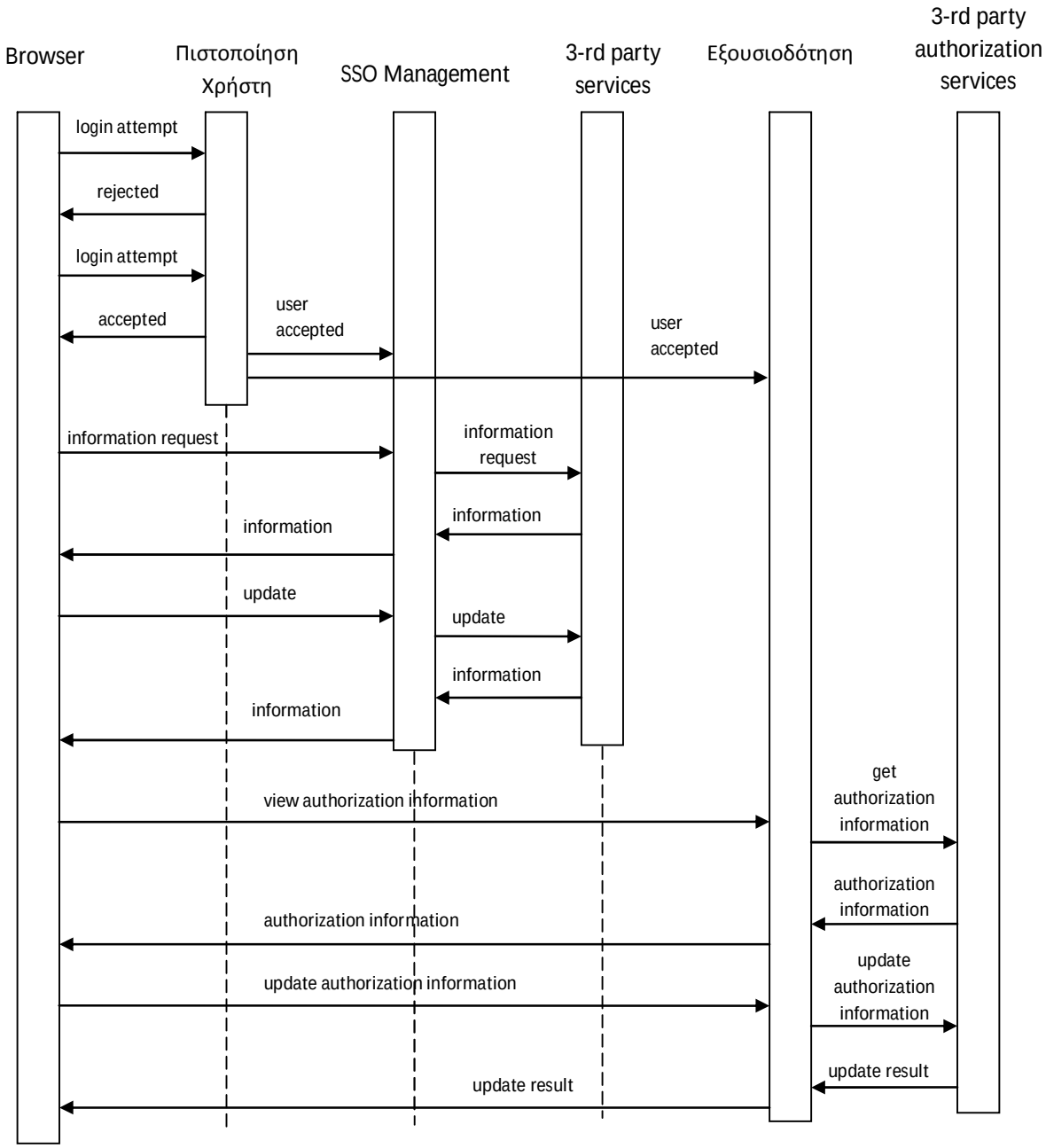


Σχήμα 15: Διάγραμμα Domain για το μόρφημα Ασφάλειας

Το σχεδιαστικό αυτό μόρφημα αποτελείται από τα ακόλουθα στοιχεία (components):

- Πιστοποίηση Χρήστη: επιτρέπει στο χρήστη να κάνει login στο portal. Αποτελεί πάντα το σημείο εισόδου.
- SSO Management: επιτρέπει στο χρήστη να παρέχει όλες τις απαραίτητες πληροφορίες για Single Sign-On (όπως username και password)
- Εξουσιοδότηση: χρησιμοποιείται για να μεταβάλλει τις άδειες πρόσβασης σε αντικεῖ ενα του portal και του Grid. Αυτό το component μπορεί να χρησιμοποιηθεί μαζί με τα εσωτερικά components πιστοποίησης του portal ή μαζί με κάποια εξωτερική υπηρεσία που προσφέρει πιο εξεζητημένες και λεπτομερείς λύσεις.

Στη συνέχεια παρουσιάζεται το ακολουθιακό διάγραμμα του μορφήματος.

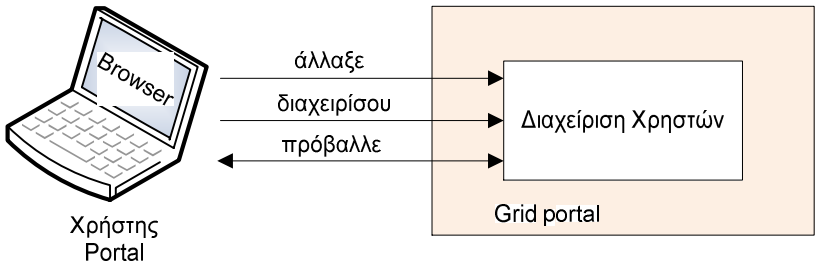


Σχήμα 16: Ακολουθιακό Διάγραμμα για το μόρφημα Ασφάλειας

4.2.2.2 Κοινή Λειτουργικότητα Διαχείρισης Χρηστών

4.2.2.2.1 Περιγραφή

Αυτή η κοινή λειτουργικότητα καλύπτει τις ανάγκες για διεπαφές διαδικτύου οι οποίες να διαχειρίζονται τους λογαριασμούς των χρηστών του portal όπως αυτές προσδιορίστηκαν από την κοινή τεχνική απαίτηση διαχείρισης χρηστών (βλ. ενότητα 4.2.1.2).



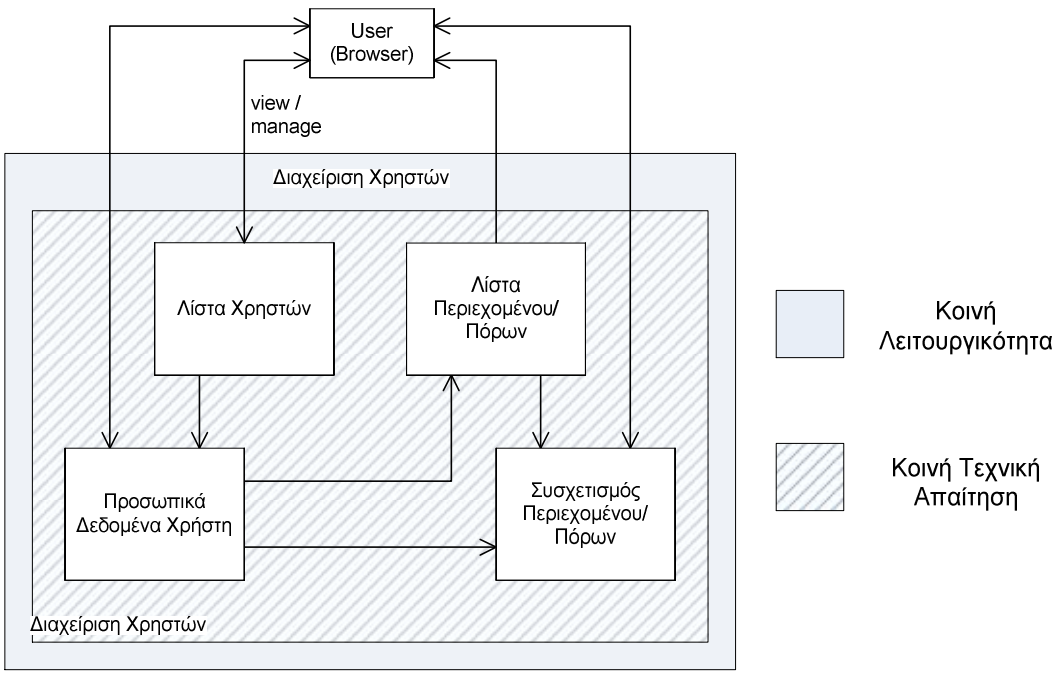
Σχήμα 17: Κοινή Λειτουργικότητα Διαχείρισης Χρηστών

Η κοινή λειτουργικότητα επιτρέπει στο χρήστη:

- Να διαχειρίζεται τον προσωπικό του λογαριασμό στο portal και να προσωποποιεί την πρόσβαση σε περιεχόμενο και πόρους.
- Να διαχειρίζεται τα προσωπικά του στοιχεία.
- Να βλέπει τα προσωπικά στοιχεία άλλων χρηστών (μόνο για εξουσιοδοτημένους χρήστες).

4.2.2.2.2 Σχεδιαστικό Μόρφημα

Στο παρακάτω σχήμα παρουσιάζεται το domain diagram αυτού του μορφήματος.

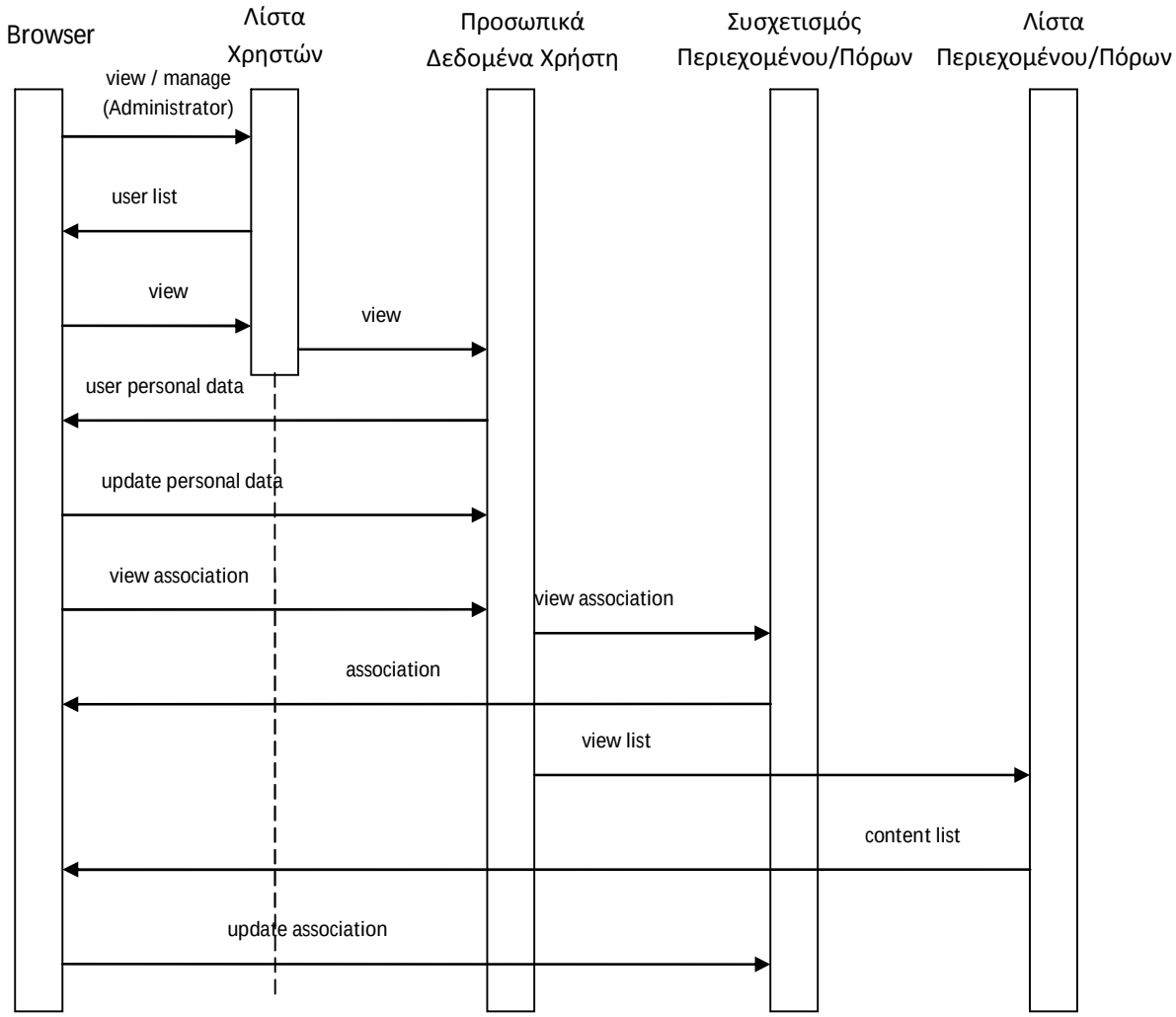


Σχήμα 18: Διάγραμμα Domain για το μόρφημα Διαχείρισης Χρηστών

Όπως παρατηρούμε αποτελείται από τα ακόλουθα στοιχειά (components):

- Προσωπικά Δεδομένα Χρήστη: επιτρέπει στους χρήστες να δουν και να αλλάξουν τις προσωπικές τους πληροφορίες, όπως το όνομα και η διεύθυνση του e-mail.
- Λίστα Χρηστών: επιτρέπει στους χρήστες ή στους διαχειριστές (administrators) να δουν και να διαχειριστούν λογαριασμούς χρηστών
- Συσχετισμός Περιεχομένου/Πόρων: επιτρέπει στους χρήστες ή τους διαχειριστές να αλλάξουν τους συσχετισμούς ανάμεσα στους λογαριασμούς χρηστών και το περιεχόμενο/πόρους. Για παράδειγμα ένας χρήστης μπορεί να επιλέξει ποια components θα χρησιμοποιήσει και πως θα είναι διαμορφωμένη η σελίδα του στο portal (προσωποποίηση της όψης του portals).
- Λίστα Περιεχομένου/Πόρων: επιτρέπει στους χρήστες να δουν τους διαθέσιμους πόρους του portal (δηλαδή τα portlets) και ποιους μπορούν να χρησιμοποιήσουν.

Στο σχήμα 19 παρουσιάζεται το αντίστοιχο ακολουθιακό διάγραμμα.



Σχήμα 19: Ακολουθιακό Διάγραμμα για το μόρφημα Διαχείρισης Χρηστών

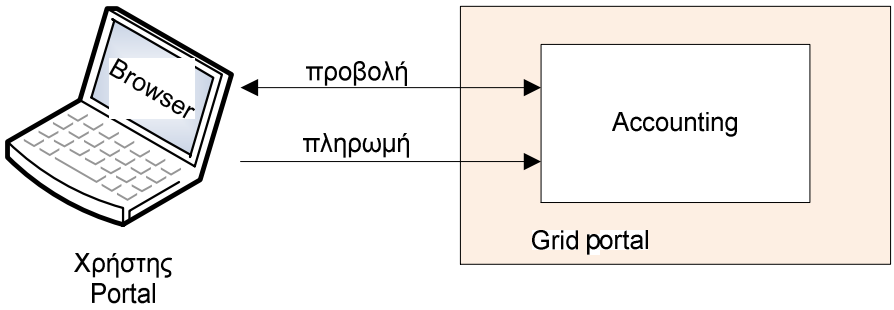
Όλες οι λειτουργίες που εκτελούνται από το μόρφημα αυτό μπορούν να διαιρεθούν σε δύο βασικές ροές:

- Η πρώτη αφορά τη διαχείριση των προσωπικών στοιχείων και συσχετισμών πόρων του χρήστη κυρίως από τους κατόχους των λογαριασμών.
- Η δεύτερη αποτελείται από τα ίδια components όπως και η πρώτη και από την επιπλέον δυνατότητα να προβληθούν ή να τροποποιηθούν λογαριασμοί χρηστών (από όσους διαθέτουν την κατάλληλη εξουσιοδότηση). Σε αυτή την περίπτωση, το πρώτο βήμα είναι η επιλογή ενός λογαριασμού από τη λίστα και έπειτα η αλλαγή των δικαιωμάτων του επιλεγμένου χρήστη.

4.2.2.3 Κοινή Λειτουργικότητα Accounting

4.2.2.3.1 Περιγραφή

Αυτή η κοινή λειτουργικότητα καλύπτει τις ανάγκες για διεπαφές διαδικτύου που να προβάλλουν λογιστικές πληροφορίες σχετικά τους πόρους, όπως αυτές προσδιορίστηκαν από την κοινή τεχνική απαίτηση accounting (βλ. ενότητα 4.2.1.3).



Σχήμα 20: Κοινή Λειτουργικότητα Accounting

Η κοινή λειτουργικότητα επιτρέπει στο χρήστη:

- Να δει λογιστικές πληροφορίες σχετικά με τους πόρους.
- Να δει πληροφορίες χρέωσης σχετικά με πόρους που χρησιμοποίησε σε δεδομένες χρονικές περιόδους
- Να κάνει online πληρωμές για πόρους που χρησιμοποίησε

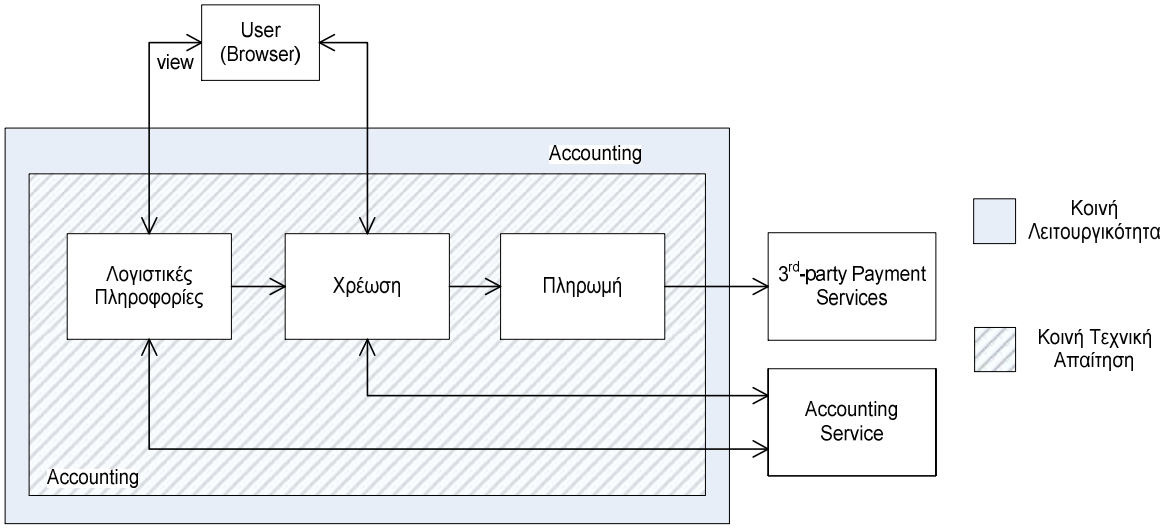
4.2.2.3.2 Σχεδιαστικό Μόρφημα

Το μόρφημα αυτό αποτελείται από τα παρακάτω στοιχειά (components):

- Λογιστικές Πληροφορίες: επιτρέπει στο χρήστη να δει λογιστικές πληροφορίες σχετικά με τους διαθέσιμους πόρους
- Χρέωση: επιτρέπει στο χρήστη να δει λεπτομερείς λογιστικές πληροφορίες. Είναι δυνατόν να ελεγχθούν οι παρεχόμενες πληροφορίες για διαφορετικές χρονικές περιόδους κατά πόρο ή κατά εργασία.

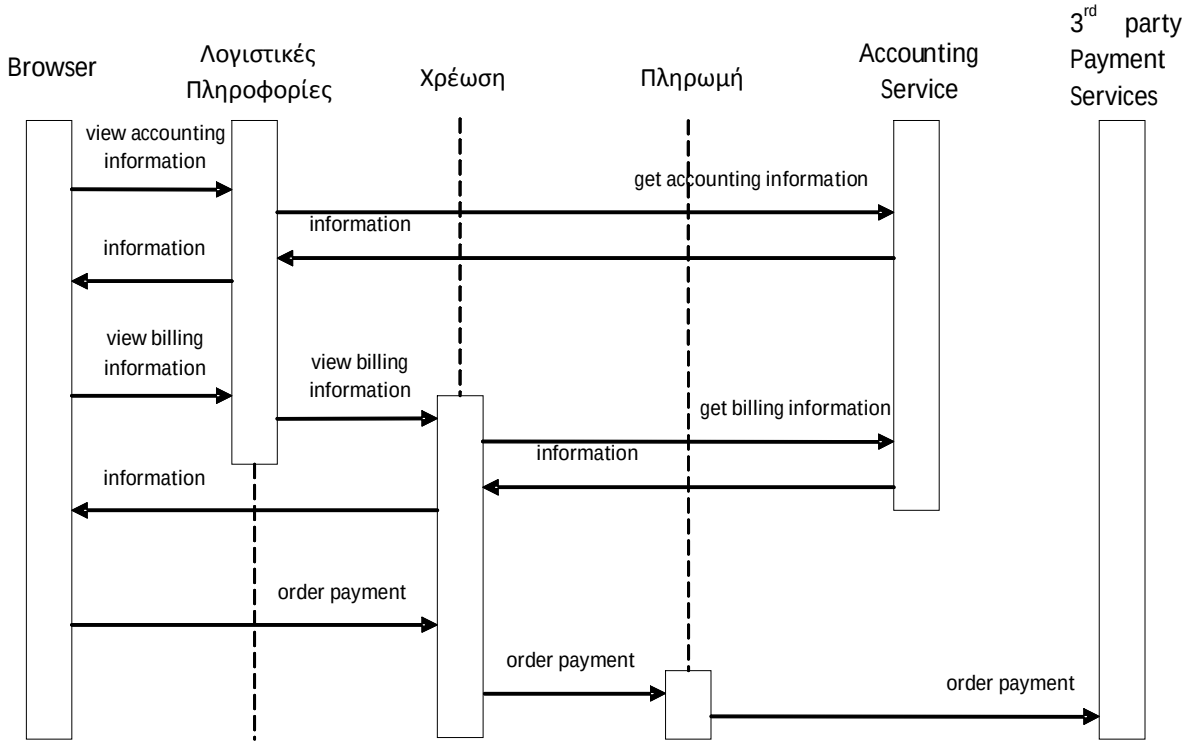
- Πληρωμή: επιτρέπει στο χρήστη να χρησιμοποιήσει μη ενεχόμενες υπηρεσίες πληρωμής (π.χ. για να εκτελέσει μια πληρωμή μέσω πιστωτικής κάρτας). Τα δεδομένα που συλλέγονται από το component Χρέωση και οι πληροφορίες του χρήστη παρουσιάζονται προς έγκριση και διορθώσει, πριν εκτελεστεί η πληρωμή.

Βάσει των προηγούμενων το domain diagram του μορφήματος θα είναι το ακόλουθο.



Σχήμα 21: Διάγραμμα Domain για το μόρφημα Accounting

Ενώ το αντίστοιχο ακολουθιακό διάγραμμα παρουσιάζεται στο σχήμα 22.

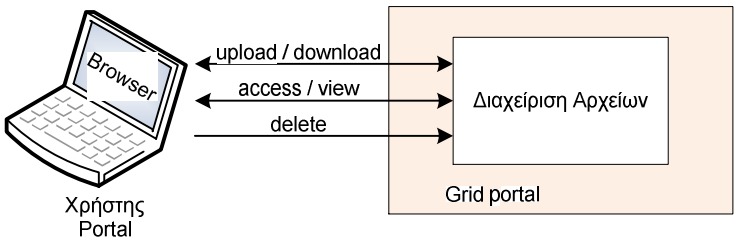


Σχήμα 22: Ακολουθιακό Διάγραμμα για το μόρφημα Accounting

4.2.2.4 Κοινή Λειτουργικότητα Διαχείρισης Αρχείων

4.2.2.4.1 Περιγραφή

Αυτή η κοινή λειτουργικότητα καλύπτει τις ανάγκες upload, download και πρόσβασης σε αρχεία, όπως αυτές προσδιορίστηκαν από την κοινή τεχνική απαίτηση διαχείρισης αρχείων (βλ. ενότητα 4.2.1.4).



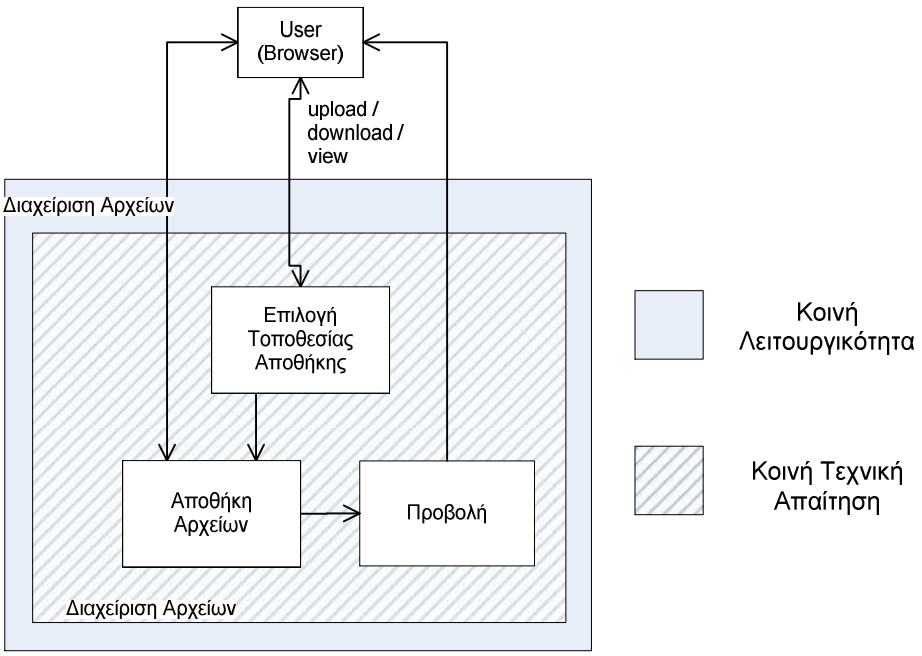
Σχήμα 23: Κοινή Λειτουργικότητα Accounting

Η κοινή αυτή λειτουργικότητα επιτρέπει στο χρήστη:

- Να “ανεβάζει” αρχεία σε αποθήκες (repositories) Grid χρησιμοποιώντας διεπαφές διαδικτύου.
- Να “κατεβάζει” αρχεία από αποθήκες Grid χρησιμοποιώντας διεπαφές διαδικτύου.
- Να έχει πρόσβαση σε αρχεία που βρίσκονται σε αποθήκες Grid και να μπορεί να αλλάζει τις ιδιότητες τους και τα δικαιώματα χρήσης τους.
- Να διαγράφει αρχεία από αποθήκες Grid.

4.2.2.4.2 Σχεδιαστικό Μόρφημα

Στο ακόλουθο σχήμα παρουσιάζεται το domain diagram αυτού του μορφήματος.

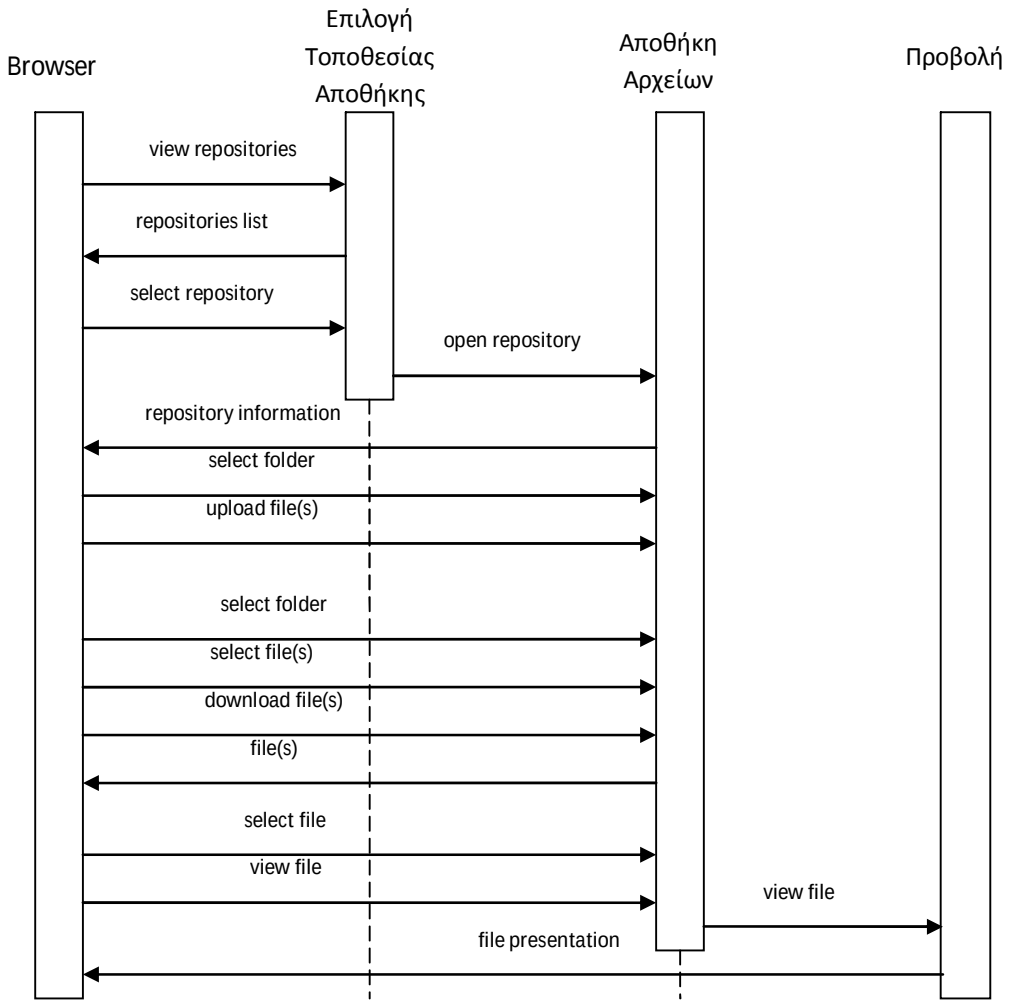


Σχήμα 24: Διάγραμμα Domain για το μόρφημα Διαχείρισης Αρχείων

Όπως παρατηρούμε το μόρφημα αποτελείται από τα ακόλουθα στοιχεία (components):

- Επιλογή Τοποθεσίας Αποθήκης: επιτρέπει στο χρήστη να επιλέξει την τοποθεσία της αποθήκης Grid με την οποία επιθυμεί να δουλέψει. Αυτό μπορεί να γίνει είτε απευθείας μέσω URL είτε μέσω επιλογής από λίστα διαθέσιμων τοποθεσιών. Μπορεί επίσης είτε να χρησιμοποιηθεί είτε η ιστορία των τοποθεσιών του χρήστη είτε να κληθεί μια εξωτερική λίστα πόρων.
- Αποθήκη Αρχειων: επιτρέπει στο χρήστη να δει τη δομή καταλόγου μίας ορισμένης αποθήκης αρχείων και να επιλέξει αρχεία και καταλόγους των οποίων τις ιδιότητες επιθυμεί να αλλάξει.
- Προβολή: παρουσιάζει στο portal αρχεία διαφορετικού τύπου από την αποθήκη Grid.

Στο σχήμα 25 παρουσιάζεται το αντίστοιχο ακολουθιακό διάγραμμα του μορφήματος.



Σχήμα 25: Ακολουθιακό Διάγραμμα για το μόρφημα Διαχείριση Αρχείων

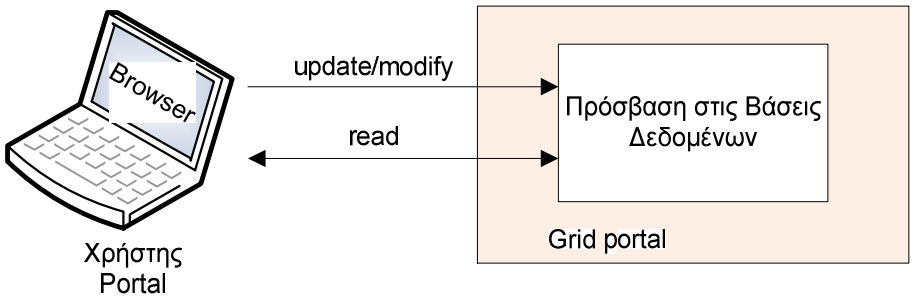
Το σημείο εισόδου του μορφήματος είναι πάντα η Επιλογή Τοποθεσίας Αποθήκης. Αφού ο χρήστης έχει επιλέξει επιτυχώς μια υπάρχουσα αποθήκη Grid, αυτή ανοίγεται μέσα στο component Αποθήκη Αρχείων. Ο χρήστης μπορεί τώρα να πλοηγηθεί μέσα στη δομή

καταλόγου της αποθήκης και να επιλέξει αρχεία ή καταλόγους, πάνω στα οποία επιθυμεί να εκτελέσει κάποιες λειτουργίες.

4.2.2.5 Κοινή Λειτουργικότητα Πρόσβασης σε Βάσεις Δεδομένων

4.2.2.5.1 Περιγραφή

Αυτή η κοινή λειτουργικότητα καλύπτει τις ανάγκες πρόσβασης σε δεδομένα από πολλαπλές και ετερογενείς πηγές, όπως αυτές προσδιορίστηκαν από την κοινή τεχνική απαίτηση πρόσβασης βάσεων δεδομένων (βλ. ενότητα 4.2.1.5).



Σχήμα 26: Κοινή Λειτουργικότητα Πρόσβασης σε Βάσεις Δεδομένων

Η κοινή λειτουργικότητα επιτρέπει στο χρήστη:

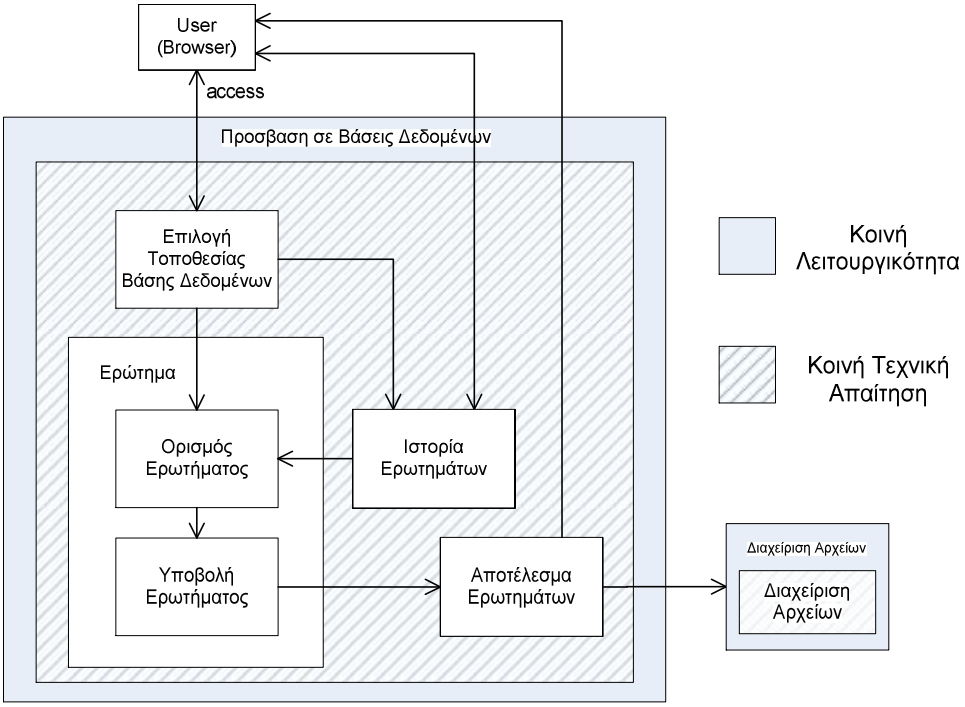
- Να διαβάζει και να ενημερώνει/μεταβάλλει ακόμη και ετερογενή δεδομένα από πολλαπλές πηγές χωρίς να ενδιαφέρεται για τους υποκείμενους μηχανισμούς ασφαλείας, τα πρωτόκολλα και τους τύπους των αποθηκευμένων δεδομένων.

4.2.2.5.2 Σχεδιαστικό Μόρφημα

Το μόρφημα αυτό αποτελείται από τα παρακάτω στοιχεία (components):

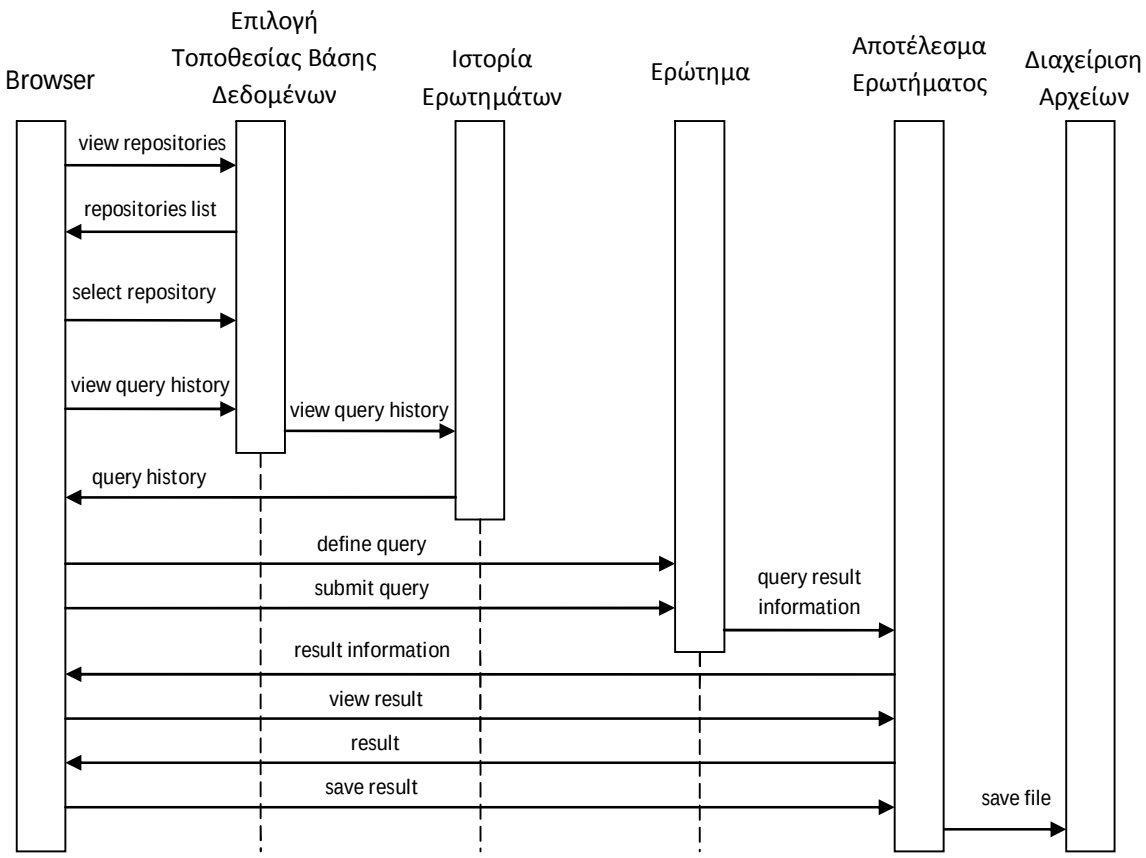
- Επιλογή Τοποθεσίας Βάσης Δεδομένων: επιτρέπει στο χρήστη να επιλέξει μια τοποθεσία αποθήκης βάσης δεδομένων με την οποία επιθυμεί να δουλέψει. Αυτό μπορεί να επιτευχθεί είτε απευθείας μέσω URL, είτε μέσω μιας λίστας διαθέσιμων τοποθεσιών. Προκειμένου να συνταχθεί αυτή η λίστα μπορεί να χρησιμοποιηθεί η ιστορία των τοποθεσιών του χρήστη.
- Ερωτήματα (Queries) τα αποτελούνται από δύο υπο-στοιχεία (sub-components):
    - Ορισμός Ερωτημάτων: επιτρέπει στο χρήστη να ορίσει ή να μεταβάλλει ένα ερώτημα το οποίο υποβάλλεται στο επόμενο υπο-στοιχείο.
    - Υποβολή Ερωτημάτων: υποβάλλει ένα ερώτημα και εμφανίζει progress bar (εφόσον έχει νόημα).
- Ιστορία Ερωτημάτων: επιτρέπει στο χρήστη να επιλέξει ένα ερώτημα (σε οποιαδήποτε μορφή) από μία λίστα ερωτημάτων που έγιναν στο παρελθόν.
- Αποτέλεσμα Ερωτήματος: επιτρέπει στο χρήστη να δει την κατάσταση στην οποία βρίσκεται ένα ολοκληρωμένο ερώτημα. Επίσης του επιτρέπει να δει τυχόν μηνύματα σφάλματος ή να ανακτήσει τα αποτελέσματα του ερωτήματος (εφόσον υπάρχουν).

Στο ακόλουθο σχήμα παρουσιάζεται το διάγραμμα domain αυτού του μορφήματος.



Σχήμα 27: Διάγραμμα Domain για το μόρφημα Πρόσβασης σε Βάσεις Δεδομένων

Το αντίστοιχο ακολουθιακό διάγραμμα του μορφήματος έχει ως εξής:

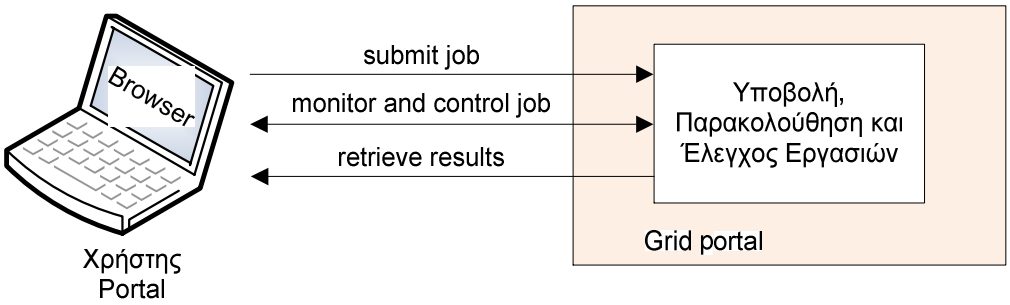


Σχήμα 28: Ακολουθιακό Διάγραμμα για το μόρφημα Πρόσβασης σε Βάσεις Δεδομένων

4.2.2.6 Κοινή Λειτουργικότητα Υποβολής, Παρακολούθησης και Ελέγχου Εργασιών

4.2.2.6.1 Περιγραφή

Αυτή η κοινή λειτουργικότητα καλύπτει τις ανάγκες εκτέλεσης, παρακολούθησης και ελέγχου των υπολογιστικών εργασιών, όπως αυτές προσδιορίστηκαν από την κοινές τεχνικές απαιτήσεις υποβολής, παρακολούθησης και ελέγχου εργασιών (βλ. ενότητες 4.2.1.6 - 4.2.1.7).



Σχήμα 29: Κοινή Λειτουργικότητα Υποβολής, Παρακολούθησης και Ελέγχου Εργασιών

Η κοινή λειτουργικότητα επιτρέπει στο χρήστη:

- Να εκκινήσει την εκτέλεση μιας υπολογιστικής εργασίας (job) επιλέγοντας τον τύπο της, τα δεδομένα εισόδου της (input data) και τις παραμέτρους της. Ο χρήστης θα πρέπει επίσης να μπορεί να αποφασίζει που θα εκτελεστεί η εργασία είτε επιλέγοντας ένα συγκεκριμένο μηχάνημα από μια λίστα διαθέσιμων πόρων, είτε θέτοντας διάφορα κριτήρια απόδοσης.
- Να βλέπει την ιστορία και την κατάσταση (finished, suspended, cancelled ή failed) ενός ή περισσότερων υποβληθείσων εργασιών. Επίσης ο χρήστης έχει τη δυνατότητα να παρακολουθεί και να ελέγχει (δηλαδή να ακυρώνει, να κάνει suspend ή resume) την εκτέλεση εργασιών οι οποίες δεν έχουν τερματίσει ακόμα.
- Να διαπραγματεύεται SLAs με τον ίδιο τρόπο με τον οποίο υποβάλλει υπολογιστικές εργασίες.

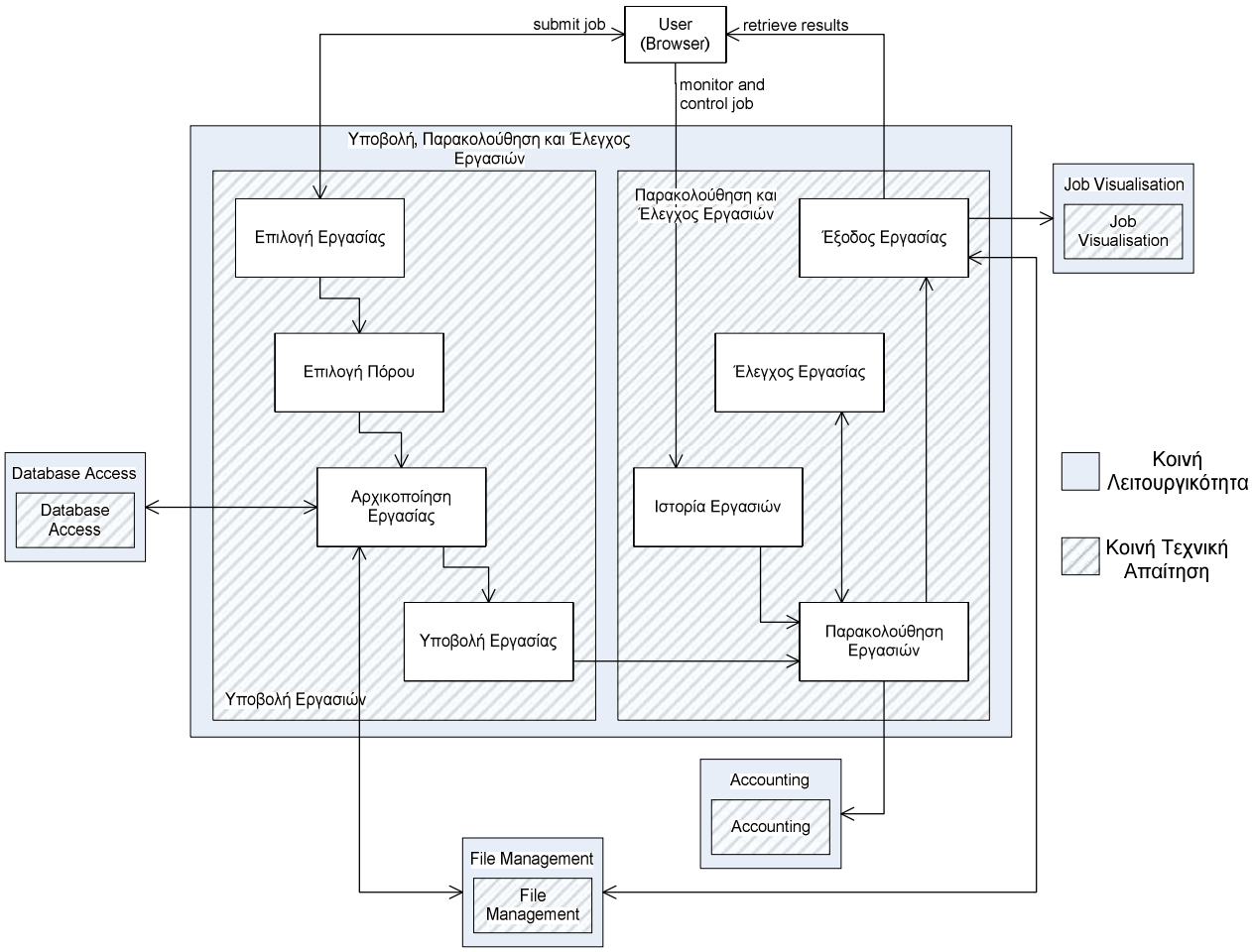
4.2.2.6.2 Σχεδιαστικό Μόρφημα

Το μόρφημα αυτό αποτελείται από τα εξής στοιχεία (components):

- Επιλογή Εφαρμογής: επιτρέπει στο χρήστη να επιλέξει μια εφαρμογή προς εκτέλεση στο Grid από μία λίστα διαθέσιμων εφαρμογών .
- Επιλογή Πόρου: επιτρέπει στο χρήστη να επιλέξει έναν υπολογιστικό πόρο στον οποίο θα εκτελεστεί η επιλεγμένη εργασία. Η συγκεκριμένη εφαρμογή που απαιτείται για την εκτέλεση της εργασίας θα πρέπει να είναι ήδη εγκατεστημένη στους πόρους που εμφανίζονται εδώ.
- Αρχικοποίηση Εργασίας: είναι η φάση στην οποία ο χρήστης διαμορφώνει τις διάφορες παραμέτρους της εργασίας και προσδιορίζει τα δεδομένα εισόδου της.
- Υποβολή Εργασίας: υποβάλλει την καθορισμένη εργασία στον καθορισμένο υπολογιστικό πόρο.

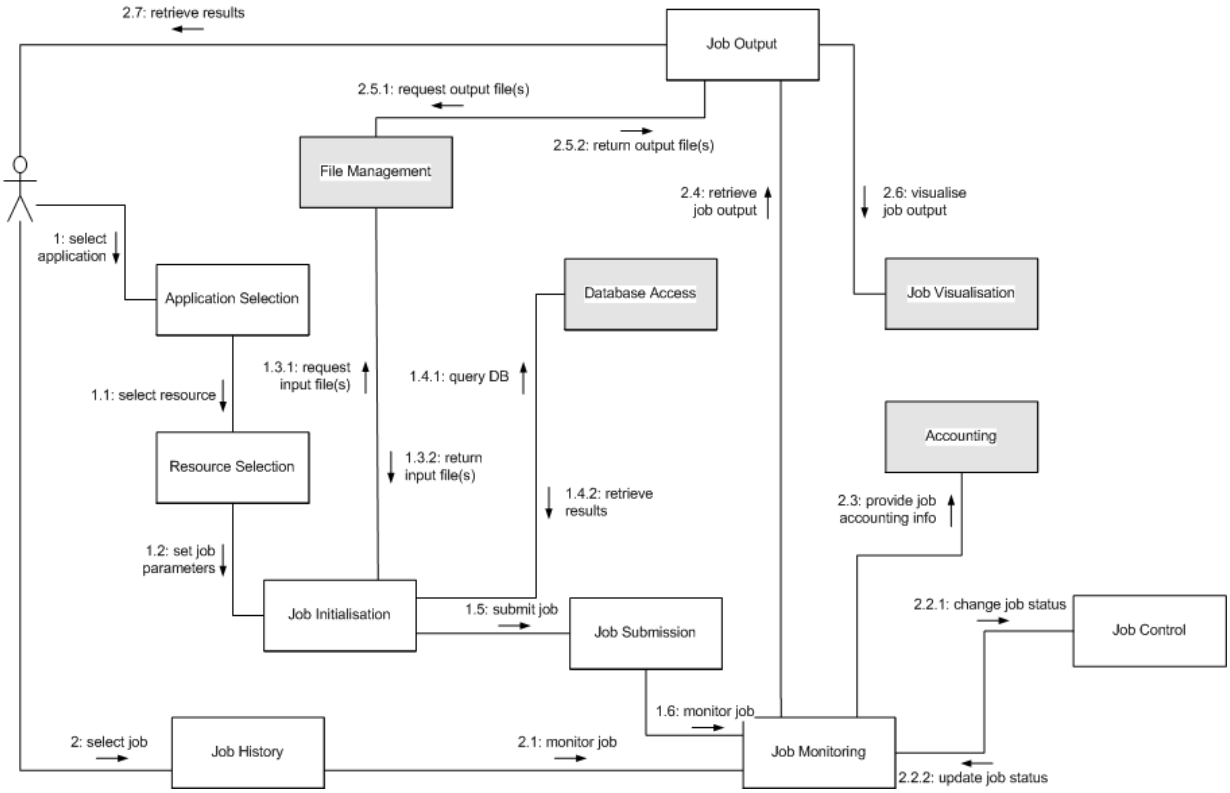
- Ιστορία Εργασιών: παρουσιάζει την ιστορία των υποβληθείσων εργασιών, μαζί με τις λεπτομέρειες υποβολής τους, επιτρέποντας στο χρήστη να επιλέξει μια από αυτές για παρακολούθηση.
- Παρακολούθηση (monitoring) Εργασίας: παρουσιάζει την κατάσταση μιας συγκεκριμένης υπολογιστικής εργασίας και πληροφορίες σχετικά με τους πόρους που αυτή χρησιμοποιεί (δηλαδή το ποσοστό ολοκλήρωσης της, τον εκτιμούμενο χρόνο που απομένει, τη χρήση της CPU, τη χρήση της μνήμης και του δίσκου κ.α.).
- Έλεγχος (Control) Εργασίας: επιτρέπει τον έλεγχο μιας συγκεκριμένης εργασίας, αν η κατάσταση της δεν είναι finished, failed ή cancelled. Αν μια εργασία "τρέχει", τότε αυτή μπορεί να διακοπεί προσωρινά (suspend) ή να ακυρωθεί. Αν μια εργασία είναι σε κατάσταση suspended τότε μπορεί να συνεχίσει τη λειτουργία της (resume) ή να ακυρωθεί.
- Έξοδος (Output) Εργασίας: επιτρέπει στο χρήστη να συλλέξει τα δεδομένα εξόδου μιας τερματισμένης (finished) εργασίας.

Ακολουθεί το διάγραμμα domain του συγκεκριμένου μορφήματος.



Σχήμα 30: Διάγραμμα Domain για το μόρφημα Υποβολής, Παρακολούθησης και Ελέγχου Εργασιών

Το σχήμα 31 παρουσιάζει το συνεργατικό διάγραμμα του μορφήματός Υποβολής, Παρακολούθησης και Ελέγχου Εργασιών.



Σχήμα 31: Συνεργατικό Διάγραμμα για το μόρφημα Υποβολής, Παρακολούθησης και Ελέγχου Εργασιών

Για μια επιτυχημένη υποβολή εργασίας τα components συνεργάζονται με τον ακόλουθο τρόπο:

Επιλογή Εφαρμογής → Επιλογή Πόρου →Αρχικοποίηση Εργασίας → Υποβολή Εργασίας

Αρχικά, πρέπει να επιλεγεί ο συγκεκριμένος τύπος εργασίας και στη συνέχεια πρέπει να επιλεγεί ένας πόρος που να τον υποστηρίζει. Οι λεπτομέρειες και τα δεδομένα εισόδου μιας εργασίας οφείλουν να καθοριστούν προτού αυτή υποβληθεί.

Με σκοπό τον προσδιορισμό των δεδομένων εισόδου, το component Αρχικοποίηση Εργασίας μπορεί να αλληλεπιδράσει με το component Διαχείρισης Αρχείων, το οποίο θα επιτρέψει στο χρήστη να “ανεβάσει” ένα νέο αρχείο εισόδου ή να υποδείξει ένα ήδη υπάρχον.

Στην περίπτωση που επιλεγεί ως τύπος εργασίας “διαπραγμάτευσή SLA”, στο component Επιλογή Εφαρμογής ακολουθείται παρόμοια διαδικασία. Τα επόμενα βήματα περιλαμβάνουν:

- επιλογή ενός πόρου (πάροχου υπηρεσιών) με τον οποίο θα γίνει η διαπραγμάτευση του SLA
- αρχικοποίηση της διαπραγμάτευσης είτε μεταβάλλοντας κάποιες παραμέτρους ενός SLA template είτε καθορίζοντας απευθείας ένα XML αρχείο στο οποίο περιέχονται οι ζητούμενες παράμετροι.

Αμέσως μετά την υποβολή μιας εργασίας, το portal μεταφέρεται στην παρακολούθηση μιας συγκεκριμένης εργασίας:

Υποβολή Εργασίας → Παρακολούθηση Εργασίας

Με αυτό τον τρόπο ο χρήστης δε θα χρειαστεί να επιλέξει την εργασία από μια λίστα υποβληθέντων εργασιών στο component Ιστορία Εργασιών. Αυτή η λειτουργία θα επιτρέψει την παρακολούθηση της προόδου μιας εργασίας και θα ειδοποιήσει το χρήστη όταν αυτή έχει ολοκληρωθεί ή έχει αποτύχει.

Σε περίπτωση που μια υποβληθείσα εργασία χρειάζεται παρακολούθηση, τα components συνεργάζονται με τον ακόλουθο τρόπο:

Ιστορία Εργασιών → Παρακολούθηση Εργασιών

Στο component Ιστορία Εργασιών ο χρήστης μπορεί να επιλέξει την εργασία που τον ενδιαφέρει από μια λίστα υποβληθείσων εργασιών και ύστερα να διαπιστώσει την κατάσταση της.

Με σκοπό την πρόσβαση σε λογιστικές πληροφορίες για μια συγκεκριμένη τερματισμένη εργασία, το component Παρακολούθηση Εργασιών μπορεί να καλέσει το Accounting component, το οποίο θα παρουσιάσει τις ζητούμενες πληροφορίες.

Εάν μια εργασία δεν έχει ολοκληρωθεί ακόμα ούτε έχει αποτύχει, τότε ενεργοποιείται η λειτουργικότητα του έλεγχου εργασίας, η οποία επιτρέπει την μεταβολή της κατάστασης της εργασίας.

Παρακολούθηση Εργασίας ← → Έλεγχος Εργασίας

Μετά την ολοκλήρωση των διαδικασιών ελέγχου εργασίας, ενεργοποιείται και πάλι το component Παρακολούθηση Εργασίας.

Μετα τον τερματισμό μιας εργασίας, τα δεδομένα εξόδου μπορούν να παρουσιαστούν και να “κατεβούν” στον υπολογιστή του χρήστη:

Παρακολούθηση Εργασίας →Έξοδος Εργασίας → Διαχείριση Αρχείων

Το component Έξοδος Εργασίας μπορεί να προσπελαστεί μέσω του component Παρακολούθηση Εργασιών όταν μια εργασία έχει τερματίσει. Το component Έξοδος Εργασίας παρέχει έναν σύνδεσμο προς ένα συγκεκριμένο αρχείο ή φάκελο, όπου βρίσκονται τα δεδομένα. Προκειμένου να προσπελαστούν τα δεδομένα εξόδου καλείται το component Διαχείριση Αρχείων. Εναλλακτικά τα δεδομένα θα μπορούσαν να προσπελαστούν απευθείας μέσω ενός συνδέσμου που θα παρέχει το component Έξοδος Εργασίας.

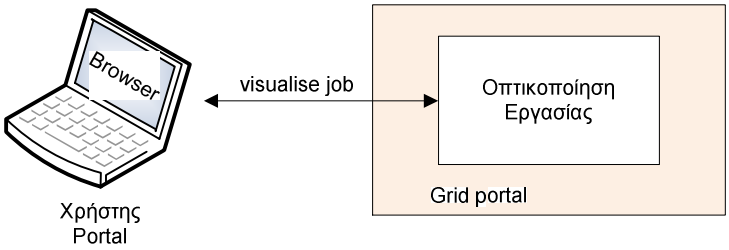
Μια άλλη πιθανή λειτουργικότητα είναι η οπτικοποίηση των δεδομένων εξόδου μέσω της κλήσης του component Οπτικοποίηση Εργασίας.

Στην περίπτωση της διαπραγμάτευσης SLA, η εργασία μπορεί ομοίως να παρακολουθείται, ενώ το component Έξοδος Εργασίας θα παρέχει την απάντηση του παρόχου υπηρεσιών (δηλαδή εάν ένα SLA έγινε αποδεκτό ή όχι).

4.2.2.7 Κοινή Λειτουργικότητα Οπτικοποίησης Εργασιών

4.2.2.7.1 Περιγραφή

Αυτή η κοινή λειτουργικότητα καλύπτει τις ανάγκες παρουσίασης online οπτικοποιήσεων των αποτελεσμάτων εκτέλεσης εργασιών, όπως αυτές προσδιορίστηκαν από την κοινή τεχνική απαίτηση οπτικοποίησης εργασιών (βλ. ενότητα 4.2.1.8).

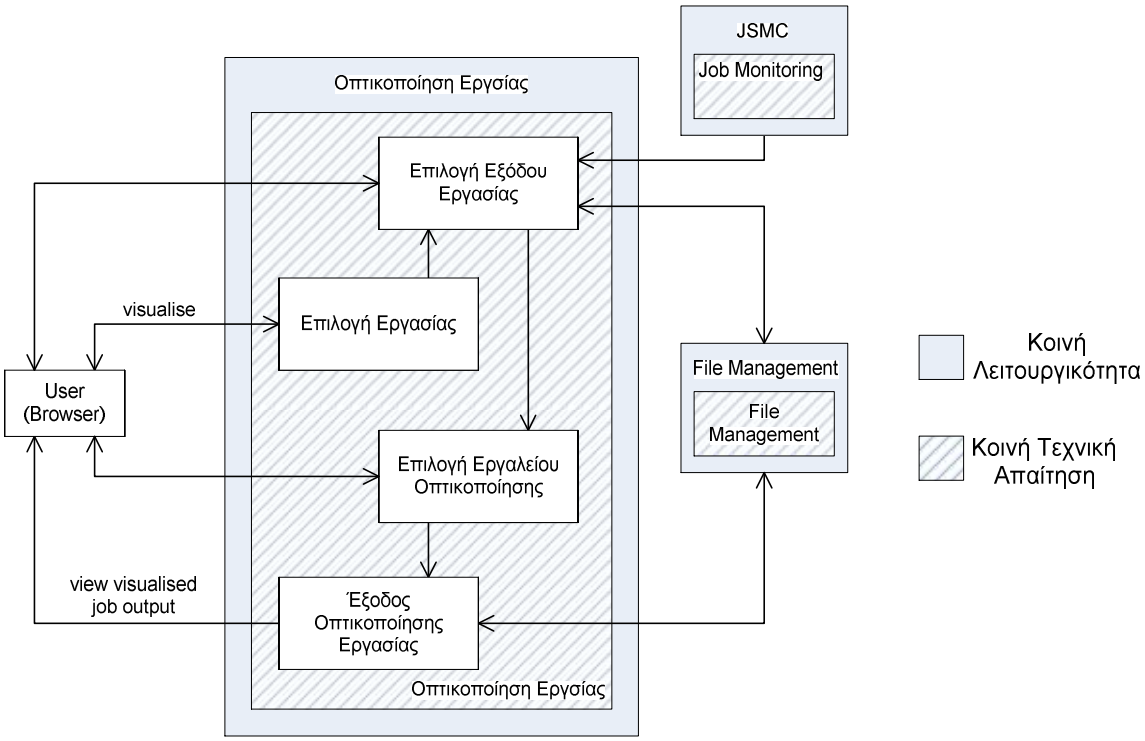


Σχήμα 32: Κοινή Λειτουργικότητα Υποβολής, Παρακολούθησης και Ελέγχου Εργασιών

Καθώς η μορφή της οπτικοποίησης μπορεί να παρουσιάζει μεγάλη ποικιλομορφία ανάλογα με τις εκάστοτε ανάγκες, ένα κοινό component μπορεί να παρέχει συνδέσμους εκτός του portal, ή εναλλακτικά, να μπορεί να εκκινήσει μια εφαρμογή και να την παρουσιάσει μέσα από το περιβάλλον του portal.

4.2.2.7.2 Σχεδιαστικό Μόρφημα

Το διάγραμμα domain του σχεδιαστικού μορφήματος παρουσιάζεται στο σχήμα 33.

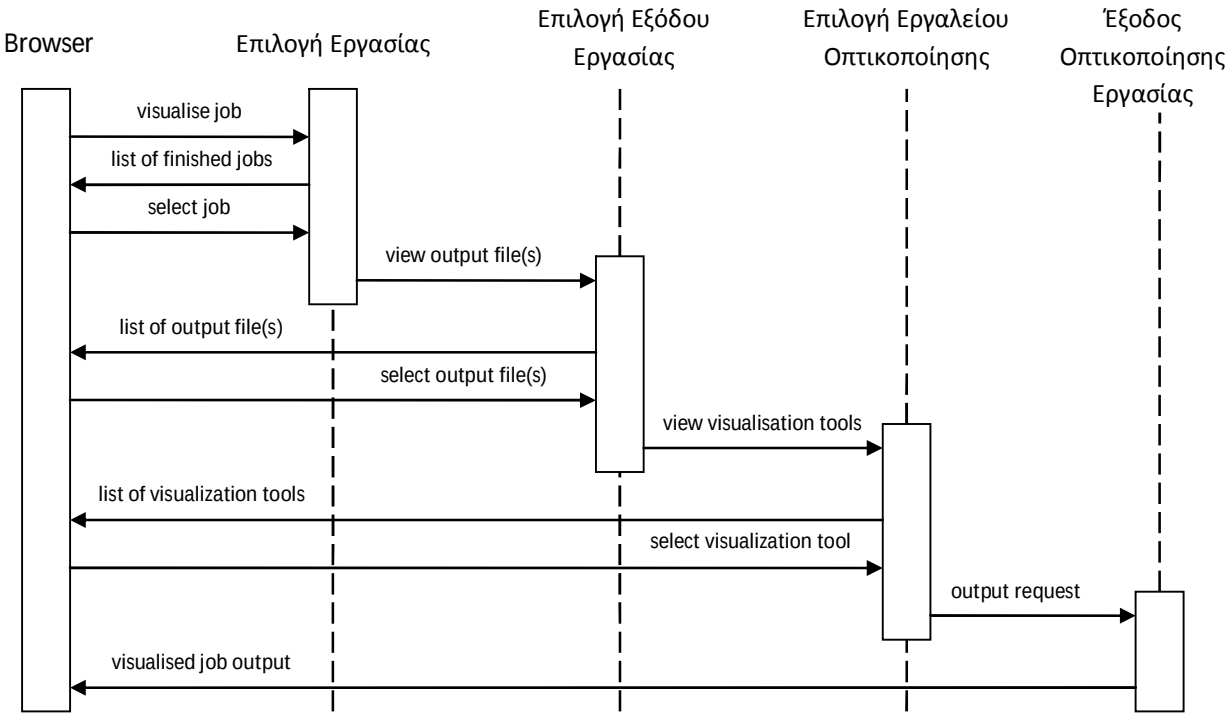


Σχήμα 33: Διάγραμμα Domain για το μόρφημα Οπτικοποίησης Εργασιών

Το μόρφημα αυτό αποτελείται από τα εξής στοιχεία (components):

- Επιλογή Εργασίας: παρουσιάζει μια λίστα των τερματισμένων εργασιών, τα αποτελέσματα των οποίων μπορούν να οπτικοποιηθούν.
- Επιλογή Εξόδου Εργασίας: παρουσιάζει και επιλέγει το αρχείο εξόδου μιας ορισμένης εργασίας, το οποίο θα οπτικοποιηθεί.
- Επιλογή Εργαλείου Οπτικοποίησης: επιλέγει τον τύπου του εργαλείου οπτικοποίησης που θα χρησιμοποιηθεί. Περισσότερα του ενός εργαλεία μπορεί να είναι κατάλληλα.
- Έξοδος Οπτικοποίησης Εργασίας: παρουσιάζει το αποτέλεσμα της οπτικοποίησης.

Ακολουθεί το ακολουθιακό διάγραμμα του μορφήματος.



Σχήμα 34: Ακολουθιακό Διάγραμμα για το μόρφημα Οπτικοποίηση Εργασιών

5 *Μελέτη, σχεδιασμός και υλοποίηση των plug-ins*

Κατά την εκπόνηση της διπλωματικής εργασίας, υλοποιήθηκαν plug-ins για την επικοινωνία των Grid Portals με το μεσολογισμικό GRIA. Η σχεδίαση και η υλοποίηση των plug-ins βασίστηκε στην κοινή λειτουργικότητα των portals όπως αυτή ορίστηκε στο κεφάλαιο 4, ενώ για την επικοινωνία με το GRIA χρησιμοποιήθηκε το Java client API του (βλ. κεφάλαιο 3). Οι διαδικασίες δοκιμής και ελέγχου των plug-ins απαίτησαν την εγκατάσταση του πακέτου Basic Application Services (βλ. ενότητα 3.3.2.1), καθώς και του πακέτου Service Provider Management (βλ. ενότητα 3.3.2.2) του GRIA. Το κεφάλαιο αυτό ασχολείται με τις διαδικασίες εγκατάστασης και ρύθμισης των πακέτων αυτών, όπως επίσης με τις λεπτομέρειες της διαδικασίας υλοποίησης των plug-ins και τα προβλήματα που αντιμετωπίστηκαν κατά τη διαδικασία αυτή.

5.1 Εγκατάσταση και ρύθμιση πακέτων GRIA

Η ανάπτυξη του προαναφερθέντος plug-in δεν θα ήταν εφικτή δίχως την ύπαρξη κάποιου παρόχου υπηρεσιών (service provider), του οποίου οι υπηρεσίες θα καταναλώνονταν. Για το σκοπό αυτό απαιτήθηκε η εγκατάσταση και παραμετροποίηση ορισμένων πακέτων του GRIA, βάσει των οποίων δημιουργήθηκε ένας υποτυπώδης πάροχος υπηρεσιών. Συγκεκριμένα εγκαταστάθηκαν τα πακέτα:

- Basic Applications Services, το οποίο παρείχε τη βασική υποστήριξη για τη διαχείριση εργασιών και δεδομένων.
- Service Provider Management, το οποίο παρείχε επιπλέον υποστήριξη για τη διαχείριση υπηρεσιών με βάση τα SLA.

Προαπαιτούμενα για την εγκατάσταση των πακέτων αυτών είναι:

- Ένα εκ των λειτουργικών συστημάτων:
  - SuSE 9.3 – 10.2
  - Fedora Core 3, 4 ή 5
  - Windows XP ή Windows 2003 Server
- Ο Apache Tomcat 5.5.x
- Το Java Software Development Kit 1.5.x

Ως λειτουργικό σύστημα επιλέχθηκαν τα Windows XP, όποτε η εγκατάσταση του Java SDK και του Tomcat τελέστηκε απρόσκοπτα, ακολουθώντας τις οδηγίες των προγραμμάτων εγκατάστασης.

5.1.1 Εγκατάσταση Basic Application Services

Προαπαιτούμενα για την εγκατάσταση αυτού του πακέτου:

- Η γλώσσα προγραμματισμού Perl [18]
- Η εφαρμογή ImageMagick [19], που χρησιμοποιείται ως test application.
- Η εφαρμογή KeyTool GUI, που χρησιμοποιείται για τη δημιουργία ενός keystore
- Η εφαρμογή XCA, που χρησιμοποιείται για την εγκατάσταση μιας υποτυπώδους certificate authority, η οποία θα υπογράψει τα πιστοποιητικά μας

Κατά την εγκατάσταση μας:

- επιλέχθηκε η υλοποίηση ActiveState Perl 5.8.8.820 από την ιστοσελίδα http://www.activestate.com/Products/ActivePerl/
- χρησιμοποιήθηκε η έκδοση 6.3.4-9-Q8 της εφαρμογή ImageMagick από το ftp://gd.tuwien.ac.at/pub/graphics/ImageMagick/binaries/
- χρησιμοποιήθηκε η έκδοση 1.7 της εφαρμογής KeyTool GUI από την ιστοσελίδα www.gria.org/downloads
- χρησιμοποιήθηκε η έκδοση 0.6.3 της εφαρμογής XCA από την ιστοσελίδα http://sourceforge.net/projects/xca

Στη συνέχεια έγινε deploy στον Tomcat το αρχείο gria-basic-app-services.war και η παραμετροποίηση του πακέτου έγινε μέσω της βασικής ιστοσελίδας διαχείρισης http://<servername>:8080/gria-service-provider-mgt, η οποία παρέχει πρόσβαση σε όλες τις επιμέρους υπηρεσίες που περιλαμβάνονται μέσα στο .war αρχείο. Την πρώτη φόρα που επισκεπτόμαστε την παραπάνω ιστοσελίδα, οι υπηρεσίες είναι απροσπέλαστες και το σύστημα θα καθοδηγήσει ως προς την ορθή παραμετροποίηση η οποία περιλαμβάνει:

- Τον ορισμό ενός νέου ρόλου χρήστη στον Tomcat, μέσω της μεταβολής του αρχείου ${**CATALINA_HOME**}/conf/tomcat-users.xml και της προσθήκης σε αυτό μιας γραμμής της μορφής:
  ```
  <role rolename="gria_basic_app_services_admin"/>
  ```
  Και μιας άλλης της μορφής:
  ```
  <user username="?" password="?"
              roles="gria_basic_app_services_admin"/>
  ```
- Τον ορισμό του καταλόγου εγκατάστασης του GRIA.
- Την εγκατάσταση και παραμετροποίηση ενός keystore, για τη δημιουργία ιδιωτικού κλειδιού και πιστοποιητικού για τον server. Ακολουθώντας τις οδηγίες δημιουργούμε (μέσω του KeyTool GUI) ένα πιστοποιητικό .csr, το οποίο υπογράφουμε χρησιμοποιώντας την υποτυπώδη certificate authority που δημιουργήσαμε προηγουμένως. Ολοκληρώνουμε τη διαδικασία εισάγοντας το υπογεγραμμένο πιστοποιητικό μας στο keystore και κάνοντας το “upload” στην ιστοσελίδα διαχείρισης.

- Την παραμετροποίηση του Tomcat, ώστε να υποστηρίζει Transport Layer Security. Αυτό επιτυγχάνεται εισάγοντας τις παρακάτω γραμμές στο αρχείο ${**CATALINA_HOME**}/conf/server.xml:

```
<Connector
    port="8443"
    keystoreFile="c:\gria\service-provider-mgt\
                          config\service-keystore.ks"
    keystorePass="?"
    keystoreType="JKS"
    minProcessors="5" maxProcessors="75"
    enableLookups="true" disableUploadTimeout="true"
    acceptCount="100" debug="0" scheme="https" secure="true"
    clientAuth="false" sslProtocol="TLS"/>
```

- Τον ορισμό της τοποθεσίας της Βάσης Δεδομένων.
- Τον καθορισμό της διεύθυνσης ακραίου σημείου (endpoint) των υπηρεσιών μας.

5.1.2 Εγκατάσταση Service Provider Management

Η εγκατάσταση του πακέτου αυτού είναι παρόμοια με την εγκατάσταση των Basic Application Services. Οι διαφορές εντοπίζονται στα εξής:

- Το Service Provider Management δεν προαπαιτεί τη γλώσσα προγραμματισμού Perl και την εφαρμογή ImageMagick
- Στο βήμα καθορισμού νέου ρόλου για τον Tomcat, οι γραμμές που πρέπει να προστεθούν στο ${**CATALINA_HOME**}/conf/tomcat-users.xml είναι οι ακόλουθες:

```
<role rolename="gria_service_provider_mgt_admin"/>

<user username="?" password="?"
  roles=" gria_service_provider_mgt_admin"/>
```

5.2 Λεπτομέρειες υλοποίησης των plug-ins

Λαμβάνοντας υπόψη την κοινή λειτουργικότητα των Grid Portals υλοποιήθηκαν δύο διεπαφές:

- JobPlugin Interface
- SLAPlugin Interface

Κάθε μία από αυτές περιλαμβάνει τις απολύτως απαραίτητες λειτουργίες, και οφείλουν να υλοποιούνται (implement) από κάθε plug-in το οποίο σκοπεύει να εκτελέσει τις αντίστοιχες λειτουργίες για κάποιο μεσολογισμικό. Συνεπώς τα plug-ins που αναπτύχθηκαν για το GRIA:

- GRIAJobPlugin
- GRIASLAPlugin

υλοποιούν το καθένα την αντίστοιχη διεπαφή. Στις ακόλουθες ενότητες αναλύονται τα υλοποιηθέντα interfaces και plug-ins. Σημειωτέον ότι για την ανάπτυξη τους χρησιμοποιήθηκε η γλώσσα προγραμματισμού Java 1.5, ώστε να αξιοποιηθεί το Java client API που παρέχεται από το GRIA.

5.2.1 JobPlugin Interface

Η διεπαφή αυτή περιλαμβάνει τις βασικές λειτουργίες για την υποβολή, παρακολούθηση και έλεγχο των εργασιών (όπως αυτή ορίζεται από την αντίστοιχη κοινή λειτουργικότητα, βλ. ενότητα 4.2.2.6). Στον πίνακα 2 παρουσιάζονται οι μέθοδοι που χρησιμοποιεί η διεπαφή, ακολουθούμενες από τα ορίσματα που απαιτούν και μια σύντομη περιγραφή τους. Ο πηγαίος κώδικας της διεπαφής παρουσιάζεται στο παράρτημα.

Μέθοδος	**Ορίσματα**	**Περιγραφή**
discoverApplications	• String[] JobServiceURL – πινάκας με τα URLs κάποιων Job Services • return String[] – πίνακας με τα ονόματα των ανακαλυφθέντων εφαρμογών ή κατάλληλο μήνυμα σφάλματος	Ανακαλύπτει τις εφαρμογές που υποστηρίζονται από κάποια Job Services.
findResources	• String ApplicationURL – το όνομα μιας εφαρμογής • String[] JobServiceURL – πινάκας με τα URLs των JobServices • return String[] – πίνακας με τα URLs των Job Services που υποστηρίζουν την εφαρμογή ή κατάλληλο μήνυμα σφάλματος	Βρίσκει τα Job Services που υποστηρίζουν μια συγκεκριμένη εφαρμογή.
submitJob	• String SLAConversationID – το ID του SLA που θα χρησιμοποιηθεί για τη χρέωση της εργασίας • String JobServiceURL – το Job Service στο οποίο θα υποβληθεί η εργασία • String ApplicationURL – η εφαρμογή που θα εκτελέσει η εργασία • String jobDescription – η περιγραφή μιας εργασίας • String[] data – πίνακας που περιλαμβάνει τις τοποθεσίες στο δίσκο του χρήστη, των δεδομένων εισόδου της εργασίας • String[] parameters – οι παράμετροι μιας εργασίας • return String – το ID της νέο-υποβληθείσας εργασίας ή κατάλληλο μήνυμα σφάλματος	Υποβάλλει μια εργασία σε κάποιο Job Service.
checkJob	• String JobConversationID – το ID της εργασίας, της οποίας την κατάσταση θέλουμε να διαπιστώσουμε	Διαπιστώνει την κατάσταση μιας εργασίας.

	• return String – κατάσταση της εργασίας (βλ. ενότητα 5.3) ή κατάλληλο μήνυμα σφάλματος	
cancelJob	• String JobConversationID – το ID της εργασίας, την οποία επιθυμούμε να τερματίσουμε • return Boolean – true αν ακυρώθηκε επιτυχώς και false σε αντίθετη περίπτωση	Ακυρώνει μια εργασία
getJobHistory	• return String[][] – πίνακας με στοιχεία για τις υποβληθείσες εργασίες (όπως αυτά περιλαμβάνονται στο repository)	Επιστρέφει την ιστορία των υποβληθείσων εργασιών που περιλαμβάνονται στο repository.
discoverJobs	• String[] JobServiceURL – πίνακας με τα Job Services στα οποία θα ψάξουμε για εργασίες • return String[][] – πίνακας με στοιχεία για τις ανακαλυφθείσες εργασίες	Ανακαλύπτει τις υποβληθείσες εργασίες σε κάποια Job Services
getJobOutputs	• String JobConversationID – το ID της εργασίας της οποίας θέλουμε να λάβουμε τα δεδομένα εξόδου • return String[] – η τοποθεσία στο σκληρό δίσκο του χρήστη που περιλαμβάνει τα job outputs	Λαμβάνει την τοποθεσία των αποτελεσμάτων (outputs) μιας εργασίας στο σκληρό δίσκο του χρήστη
storeJobOutputs	• String JobConversationID – το ID της εργασίας της οποίας θέλουμε να λάβουμε τα δεδομένα εξόδου • String[] OutputLocation – η τοποθεσία στο σκληρό δίσκο, στην οποία επιθυμούμε την αποθήκευση των job outputs • return Boolean – true αν τα δεδομένα αποθηκεύτηκαν επιτυχώς και false σε αντίθετη περίπτωση	Αποθηκεύει τα δεδομένα εξόδου μιας εργασίας σε μια συγκεκριμένη τοποθεσία
endJob	• String JobConversationID – το ID της εργασίας την οποίας θέλουμε κάνουμε finalize • return Boolean – true αν το finalization στέφθηκε με επιτυχία και false σε αντίθετη περίπτωση	Κάνει finalize μια εργασία

Πίνακας 2: Μέθοδοι και ορίσματα της διεπαφής JobPlugin

5.2.2 SLAPlugin Interface

Η διεπαφή αυτή περιλαμβάνει τις βασικές λειτουργίες για τη διαχείριση των SLAs και των Trade Accounts (υποσύνολο της κοινής λειτουργικότητας Accounting, βλ. ενότητα 4.2.2.3). Στην περίπτωση των SLAs παρεκκλίναμε ελαφρώς από την κοινή λειτουργικότητα, καθώς η θεώρηση της διαπραγμάτευσης SLA ως υποπερίπτωση του Job Submission δεν κρίθηκε ότι συμβάλλει στην ευχρηστία και την ευελιξία του συστήματος (βλ. ενότητα 5.3). Στον πίνακα 3 παρουσιάζονται οι μέθοδοι που χρησιμοποιεί η διεπαφή, ακολουθούμενες από τα ορίσματα που απαιτούν και μια σύντομη περιγραφή τους. Ο πηγαίος κώδικας της διεπαφής παρουσιάζεται στο παράρτημα.

Μέθοδος	**Ορίσματα**	**Περιγραφή**
discoverSLATemplates	• String[] SLAServiceURL – πίνακας με τα URLs κάποιων SLA Services • return String[][] – πίνακας με τα στοιχεία των ανακαλυφθέντων SLA Templates ή κατάλληλο μήνυμα σφάλματος	Ανακαλύπτει τα SLA Templates που υποστηρίζονται από κάποια SLA Services.
getSLATemplate	• String SLATemplateID – το ID του SLA Template, του οποίου θέλουμε να επιστραφούν οι όροι • return String[][] – πίνακας με τους όρους του ζητούμενου Template ή κατάλληλο μήνυμα σφάλματος	Ανακτά τους όρους ενός SLA Template
proposeSLA	• String SLATemplateID – το ID του SLA Template το οποίο θα χρησιμοποιήσουμε ως βάση της διαπραγμάτευσης • String SLAServiceURL – το SLA Service στο οποίο θα προταθεί το SLA • String Label – ο τίτλος του SLA • return String – το ID του SLA (εφόσον η πρόταση έγινε αποδεκτή) ή κατάλληλο μήνυμα σφάλματος	Προτείνει ένα SLA (βάσει κάποιου Template) σε ένα SLA Service. Εφόσον δεν έχουμε ρητή αναφορά σε κάποιο λογαριασμό, το GRIA διατρέχει το repository, προκειμένου να εντοπίσει το πιο κατάλληλο Trade Account για τη χρέωση του SLA
proposeSLAwithAccount	• String SLATemplateID – το ID του SLA Template το οποίο θα χρησιμοποιήσουμε ως βάση της διαπραγμάτευσης • String AccountID – Το ID του Trade Account στον οποίο θα χρεώνεται το SLA. • String SLAServiceURL – το SLA Service στο οποίο θα προταθεί το SLA • String Label – ο τίτλος του SLA • return String – το ID του SLA (εφόσον η πρόταση έγινε αποδεκτή) ή κατάλληλο μήνυμα σφάλματος	Προτείνει ένα SLA (βάσει κάποιου Template) σε ένα SLA Service. Το SLA χρεώνεται στο λογαριασμό που παρέχεται ως όρισμα

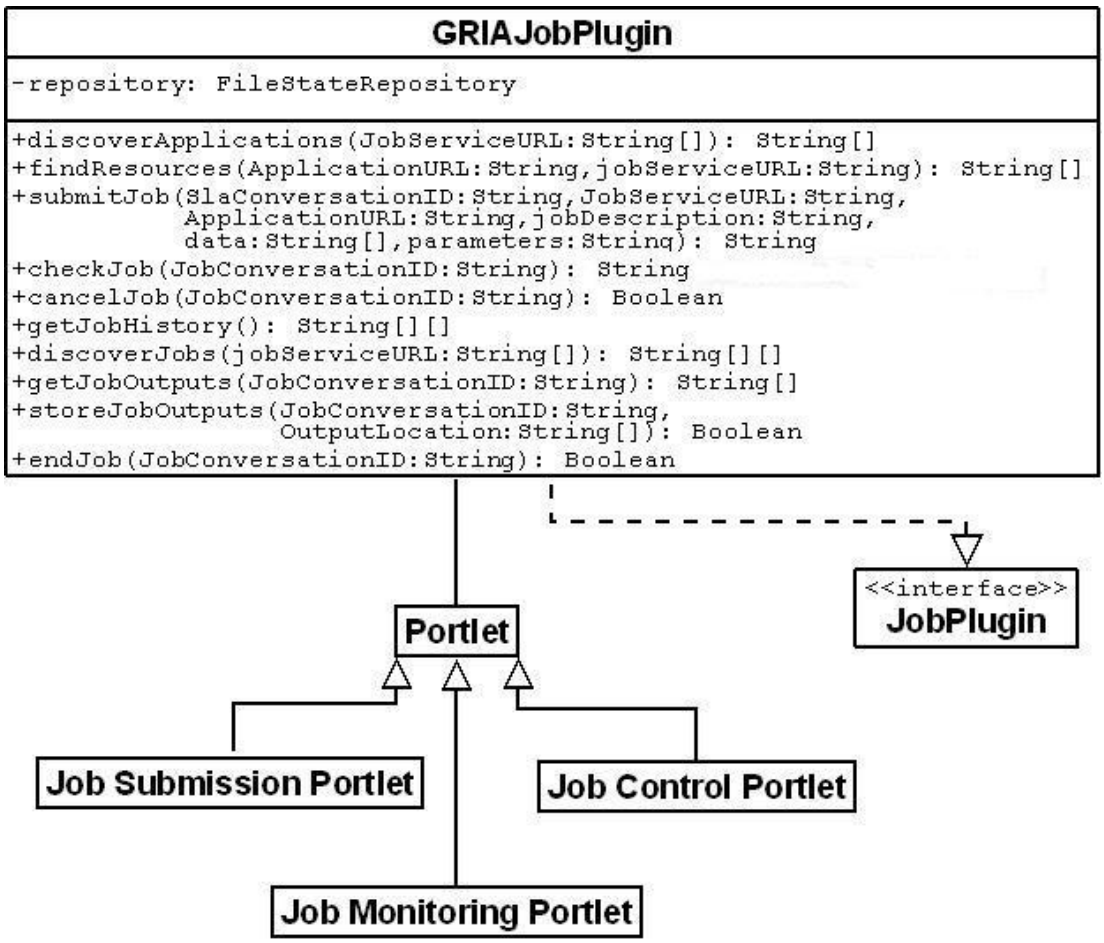
checkSLA	• String SLAConversationID – το ID του SLA του οποίου την κατάσταση επιθυμούμε να ελέγξουμε • return String – την κατάσταση του SLA ή κατάλληλο μήνυμα σφάλματος	Διαπιστώνει την κατάσταση ενός SLA
closeSLA	• String SLAConversationID – το ID του SLA, το οποίο επιθυμούμε “κλείσουμε” (τερματίσουμε) • return Boolean – true αν έκλεισε επιτυχώς και false σε αντίθετη περίπτωση	Κλείνει ένα SLA
getSLAHistory	• return String[][] – πίνακας με στοιχεία για τα SLAs που έχουν χρησιμοποιηθεί στο παρελθόν (όπως αυτά περιλαμβάνονται στο repository)	Επιστρέφει την ιστορία των SLAs που περιλαμβάνονται στο repository.
discoverSLAs	• String[] SLAServiceURL – πίνακας με τα SLA Services στα οποία θα ψάξουμε για SLAs • return String[][] – πίνακας με στοιχεία για τα ανακαλυφθέντα SLA	Ανακαλύπτει SLAs από κάποια SLA Services
createAccount	• String TradeAccountServiceURL – το URL του Account Service στο οποίο θέλουμε να δημιουργήσουμε τον λογαριασμό • String[] AccountParameters – πίνακας με τις παραμέτρους του λογαριασμού • String[] ClientAddress – πινάκας με τη διεύθυνση του χρήστη του λογαριασμού • return String – το ID του Account ή κατάλληλο μήνυμα σφάλματος	Δημιουργεί ένα Λογαριασμό σε κάποιο Trade Account Service
checkAccount	• String TradeAccountID – το ID του Account του οποίου την κατάσταση επιθυμούμε να ελέγξουμε • return String – την κατάσταση του Account ή κατάλληλο μήνυμα σφάλματος	Διαπιστώνει την κατάσταση ενός Λογαριασμού
closeAccount	• String TradeAccountID – το ID του Λογαριασμού, τον οποίο επιθυμούμε “κλείσουμε” (τερματίσουμε) • return Boolean – true αν έκλεισε επιτυχώς και false σε αντίθετη περίπτωση	Κλείνει έναν Λογαριασμό

getTradeAccountHistory	• return String[][] – πίνακας με στοιχεία για τα Accounts που έχουν χρησιμοποιηθεί στο παρελθόν (όπως αυτά περιλαμβάνονται στο repository)	Επιστρέφει την ιστορία των Λογαριασμών που περιλαμβάνονται στο repository.
discoverTradeAccounts	• String[] TradeAccountServiceURL – πίνακας με τα Account Services στα οποία θα ψάξουμε για Λογαριασμούς • return String[][] – πίνακας με στοιχεία για τους ανακαλυφθέντες λογαριασμούς	Ανακαλύπτει Λογαριασμούς από κάποια Trade Account Services

Πίνακας 3: Μέθοδοι και ορίσματα της διεπαφής SLAPlugin

5.2.3 GRIAJobPlugin

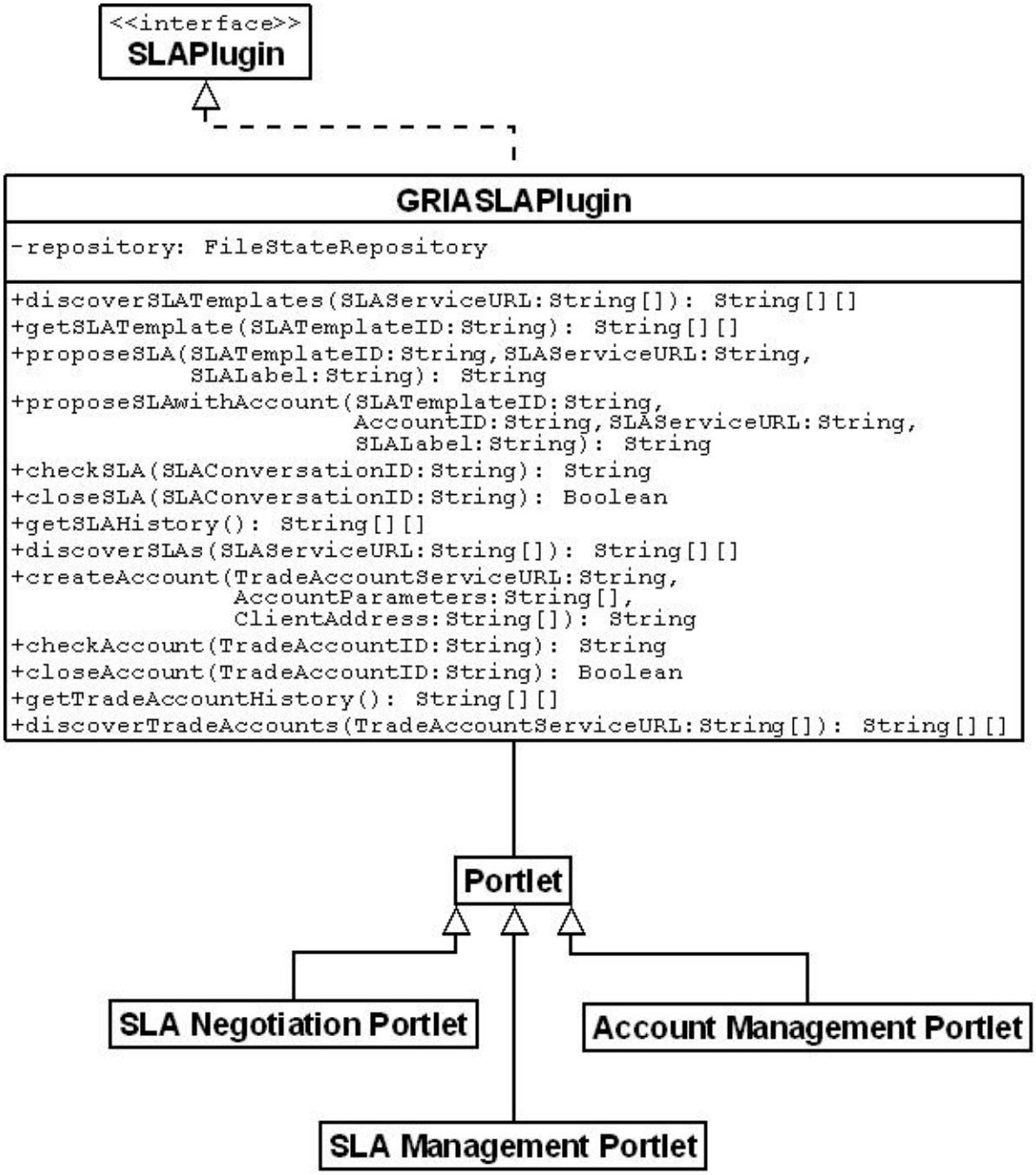
Το plug-in αυτό επιτρέπει σε ένα portal να έχει πρόσβαση στη λειτουργικότητα του μεσολογισμικού GRIA που αφορά της Εργασίες. Κάνει implement τη διεπαφή JobPlugin, με αποτέλεσμα να υλοποιεί όλες τις μεθόδους της. Κάνει εκτενή χρήση του client GRIA API προκειμένου να επικοινωνεί με το GRIA. Στο σχήμα 35 φαίνεται το διάγραμμα κλάσης του GRIAJobPlugin. Ο πηγαίος κώδικας του plug-in παρουσιάζεται στο παράρτημα.



Σχήμα 35: Διάγραμμα κλάσης GRIAJobPlugin

5.2.4 GRIASLAPlugin

Το plug-in αυτό επιτρέπει σε ένα portal να έχει πρόσβαση στη λειτουργικότητα του μεσολογισμικού GRIA που αφορά τα SLAs και τα Trade Accounts. Κάνει implement τη διεπαφή SLAPlugin, με αποτέλεσμα να υλοποιεί όλες τις μεθόδους της. Κάνει επίσης εκτενή χρήση του client GRIA API προκειμένου να επικοινωνεί με το GRIA. Στο σχήμα 36 φαίνεται το διάγραμμα κλάσης του GRIASLAPlugin. Ο πηγαίος κώδικας της διεπαφής παρουσιάζεται στο παράρτημα



Σχήμα 36: Διάγραμμα κλάσης GRIASLAPlugin

5.3 Προβλήματα και Λύσεις

Κατά το σχεδιασμό και την υλοποίηση των ανωτέρω plug-ins αντιμετωπίστηκαν ποικίλα προβλήματα και δυσκολίες οι οποίες οδήγησαν στην αναθεώρηση των προδιαγραφών και σε ορισμένες περιπτώσεις στην παρέκκλιση από την κοινή λειτουργικότητα. Αναλυτικά:

- Η χρήση από το GRIA της τοπικής αποθήκης δεδομένων (repository), προκάλεσε πολλά προβλήματα. Αφ' ενός, η τοπική αποθήκη αποτελεί μια βολική και εύχρηστη λύση για την πρόσβαση στα δεδομένα των χρηστών ώστε να αποφεύγεται η συνεχής επικοινωνία με τα Services. Αφ' ετέρου όμως, η τοπική της φύση την καθιστά επιρρεπή σε σφάλματα και ανακρίβειες. Το γεγονός, δηλαδή, ότι η αποθήκη δεν είναι πάντα up-to-date με το εκάστοτε Service, μπορεί να οδηγήσει σε αντικρουόμενες πληροφορίες και τελικά σε εξαιρέσεις. Παραδείγματος χάρη, εάν ο administrator (ή οποιοσδήποτε άλλος χρήστης είναι επαρκώς εξουσιοδοτημένος) ακυρώσει μια εργασία ή μεταβάλλει τα privileges ενός χρήστη από κάποια απομακρυσμένη τοποθεσία, τότε η αποθήκη δεν έχει τρόπο να γνωρίζει τις αλλαγές αυτές. Ο μόνος τρόπος να παραμένει (σχεδόν) πάντα ενήμερη η αποθήκη, είναι τα αιτήματα του χρήστη να μην αντιμετωπίζονται τοπικά, αλλά να προωθούνται απευθείας στα κατάλληλα services και με βάση το αποτέλεσμα να ενημερώνεται και η αποθήκη. Αυτό όμως δεν είναι πάντα επιθυμητό ούτε ιδιαίτερα αποδοτικό, καθώς καταναλώνεται εύρος ζώνης για συνεχή επικοινωνία με τα Services. Κατά την υλοποίηση των plug-ins, υιοθετήθηκε μια ενδιάμεση πρακτική που επιχειρεί να συνδυάσει τα πλεονεκτήματα και των δύο φιλοσοφιών. Αναπτύχθηκαν ξεχωριστές μέθοδοι για ανάκτηση της τοπικής ιστορίας και για "ανακάλυψη" υπηρεσιών από τα εκάστοτε Services. Συγκεκριμένα αναπτύχθηκαν οι εξής μεθοδοι:
    - getJobHistory() και discoverJobs()
    - getSLAHistory() και discoverSLAs()
    - getTradeAccountHistory() και discoverTradeAccount()

  Όποτε ο χρήστης δεν επιθυμεί να εκτελέσει κάποια κρίσιμη ενέργεια, τότε η τοπική ιστορία που φυλάσσεται στην αποθήκη είναι επαρκής. Όταν όμως ο χρήστης επιθυμεί να διαχειριστεί εργασίες (submit, cancel ή απλά check) ή να μεταβάλλει λογαριασμούς και SLAs, θα πρέπει πρώτα να χρησιμοποιήσει μεθόδους "ανακάλυψης", οι οποίες θα ενημερώσουν κατάλληλα την αποθήκη, εξασφαλίζοντας ότι οι πληροφορίες που θα ληφθούν θα είναι όσο το δυνατόν πιο πρόσφατες. Το μειονέκτημα αυτής της πρακτικής είναι ότι η "ανακάλυψη" υπηρεσιών απαιτεί από τον χρήστη να γνωρίζει τα URLs των Services, σε αντίθεση με την ανάκτηση της τοπικής ιστορίας.
- Το GRIA δεν υποστηρίζει suspend και resume εργασιών, με αποτέλεσμα να μην αναπτυχθούν μέθοδοι για αυτές τις λειτουργίες, παρεκκλίνοντας έτσι από την κοινή λειτουργικότητα που ορίστηκε στο κεφάλαιο 4.
- Ο μηχανισμός που χρησιμοποιεί το GRIA για να ενημερώνει το χρήστη για την κατάσταση μιας εργασίας αποτέλεσε πηγή αρκετών προβλημάτων. Κι αυτό διότι παρόλο που το GRIA διαθέτει κάποιο μηχανισμό επιστροφής της τρέχουσας κατάστασης της εργασίας (μέσω της κλάσης JobStatus), αυτός αποδείχτηκε εντελώς ανεπαρκής. Συγκεκριμένα επέστρεφε μόνο δύο τιμές:
    - ready
    - job-submitted

  Προκειμένου λοιπόν ο χρήστης να ενημερώνεται σωστά για την κατάσταση μιας εργασίας, όφειλαν να χρησιμοποιηθούν και άλλα κριτήρια για τον προσδιορισμό της.

Έτσι το GRIAJobPlugin (και συγκεκριμένα η μέθοδος checkJob()) υιοθέτησε τη χρήση 5 καταστάσεων, οι οποίες προσδιορίζονται βάσει ορισμένων συνθηκών. Στον πίνακα 4 παρουσιάζεται η χαρτογράφηση (mapping) των καταστάσεων του GRIA σε καταστάσεις του plug-in και οι συνθήκες που οφείλουν να πληρούνται σε κάθε κατάσταση.

Κατάσταση GRIA	**Κατάσταση plug-in**	**Συνθήκες υπό τις οποίες χαρτογραφούνται οι καταστάσεις του GRIA σε καταστάσεις του plug-in**
	UNAUTHORIZED	όταν ο χρήστης δε διαθέτει επαρκείς ρόλους πρόσβασης στην συγκεκριμένη εργασία
job-submitted	RUNNING	όταν η μέθοδος getInProgress() της κλάσης JobStatus επιστρέφει true
ready	FINISHED	όταν η μέθοδος getInProgress() επιστρέφει false και η μέθοδος getExitStatus() επιστρέφει 0
ready	CANCELLED OR FAILED	όταν η μέθοδος getInProgress() επιστρέφει false και η μέθοδος getExitStatus() δεν επιστρέφει 0
	FINALIZED	όταν έχει κληθεί η μέθοδος finish() της κλάσης Conversation

Πίνακας 4: Χαρτογράφηση των καταστάσεων GRIA σε καταστάσεις του GRIAJobPlugin

- Το GRIA δεν είναι καθόλου ευέλικτο όσον αφορά το SLA Negotiation. Δηλαδή, εάν κάποιος χρήστης επιχειρήσει να μεταβάλλει έστω και λίγο τους όρους ενός SLA προτύπου (template) πριν κάνει proposal, τότε τις περισσότερες φορές οδηγούμαστε σε απόρριψη του. Ακόμα και όταν δεν απορρίπτεται το proposal του χρήστη, το SLA που δημιουργείται “αγνοεί” τις αλλαγές του χρήστη και περιλαμβάνει ανέπαφους τους όρους του template. Το γεγονός αυτό οδήγησε σε περαιτέρω παρέκκλιση από την κοινή λειτουργικότητα, καθώς το plug-in δεν επιτρέπει στο χρήστη να αλλάξει κάποιο template, παρά μόνο να επιλέξει κάποιο template που προσεγγίζει τις ανάγκες του. Το γεγονός αυτό αντικατοπτρίζεται στις μεθόδους proposeSLA() και proposeSLAwithAccount() του GRIASLAPlugin.

6 *Συμπεράσματα - Μελλοντικές Εξελίξεις*

Η εξέλιξη των τεχνολογιών του Grid υπήρξε θεαματική τα τελευταία χρόνια. Από μια άσημη τεχνολογία σχετιζόμενη μονό με επιστημονικές και ακαδημαϊκές εφαρμογές, μετατράπηκε σε τεχνολογία αιχμής με σημαντική διείσδυση στην αγορά. Τα Grids βελτιώνουν την απόδοση των εφαρμογών, αυξάνουν την παραγωγικότητα και προωθούν τη συνεργασία, επιτρέποντας στις επιχειρήσεις να επιτυγχάνουν καλύτερα οικονομικά αποτελέσματα και να παρέχουν καλύτερες υπηρεσίες στους πελάτες τους.

Η τεχνολογία του Grid παρέχει επίσης τη δυνατότητα αποθήκευσης, κατανομής και ανάλυσης μεγαλύτερης ποσότητας δεδομένων, εξασφαλίζοντας την απρόσκοπτη πρόσβαση στις πληροφορίες, γεγονός που μπορεί να βελτιώσει σημαντικά τη διαδικασία λήψης αποφάσεων και κατά συνέπεια την παραγωγικότητα. Με τη βοήθεια μεσολογισμικών όπως το GRIA και κάνοντας χρήση της λειτουργικότητας των Grid Portals, τα Grids βελτιώνουν τη χρησιμοποίηση των πόρων και μειώνουν το κόστος, ενώ διατηρούν μια ευέλικτή δομή που μπορεί να αντεπεξέλθει στις συχνές αλλαγές των επιχειρηματικών αναγκών, χωρίς να χάνουν την αξιοπιστία και την ασφάλεια τους.

Συνεπώς, το Grid Computing έχει πλέον εδραιώσει τη θέση του ως ένα πολλά υποσχόμενο μοντέλο, ικανό να παρέχει δυναμικές, ανθεκτικές και cost-effective υποδομές που μπορούν να προωθήσουν τόσο τη συνεργασία όσο και την καινοτομία στον επιχειρηματικό τομέα. Οι εξελίξεις στον τομέα του Grid Computing, οδηγούν σε ολοένα και αυξανόμενη υιοθέτηση του από εμπορικούς οργανισμούς, οδηγούμενους από την επιθυμία των πελατών τους να προσφέρουν καλύτερα αποκρινόμενες υπηρεσιοστρεφείς αρχιτεκτονικές [23].

Η αισιοδοξία και ο ενθουσιασμός τόσο της επιστημονικής όσο και της επιχειρηματικής κοινότητας, έχει δώσει τεράστια ώθηση στην ανάπτυξη του Grid. Ήδη μεγάλες εταιρείες παρέχουν Grid λύσεις, ενώ ακόμα και η αρχιτεκτονική των υπολογιστών έχει αρχίσει να προσαρμόζεται ώστε να εξυπηρετεί καλύτερα το περιβάλλον πολυεπεξεργασίας που το Grid απαιτεί. Οι προοπτικές που γεννιούνται από την εξάπλωση του Grid είναι ιδιαίτερα ευοίωνες. Όπως όλα δείχνουν το μέλλον ανήκει στο Grid.

7 Βιβλιογραφικές Αναφορές

[1] I. Foster and C. Kesselman, "*The Grid: Blueprint for a Future Computing Infrastructure*", Morgan Kaufmann Publishers, USA, 1999.

[2] I. Foster, C. Kesselman, S. Tuecke, "*The Anatomy of the Grid: Enabling Scalable Virtual Organizations*" International Journal Supercomputer Applications, Vol. 15, No. 3, 2001.

[3] W. Leinberger, V. Kumar, "*Information Power Grid: The new frontier in parallel computing?*" IEEE Concur., Vol. 7, No. 4, pp. 75-84, October-December 1999.

[4] M. Surridge, S. Taylor, D. De Roure, and E. Zaluska, "*Experiences with GRIA-Industrial Applications on a Web Services Grid*", in Proceedings of the First International Conference on e-Science and Grid Computing, pp. 98-105. IEEE Press, 2005

[5] GRIA, Grid Resources for Industrial Applications, www.gria.org

[6] Padgett, J., K. Djemame, and P. Dew, "Grid-based SLA Management", Lecture Notes in Computer Science, pp. 1282-1291, 2005

[7] R. Buyya, D. Abramson, S. Venugopal, "*The Grid Economy*", Proceedings of the IEEE Volume 93, Issue 3, pp. 698-714, Mar 2005

[8] Open Grid Services Architecture (OGSA), www.ggf.org/documents/GFD.30.pdf

[9] WSRF, The Web Services Resource Framework (WSRF) v1.2, http://www.oasis-open.org/committees/download.php/17833/wsrf-1.2-os.zip

[10] Pawel Plaszczak and Richard Wellner,Jr., Grid Computing – The Savvy's Manager Guide, Morgan Kaufmann Publishers, 2006, ISBN-13:978-0-1274-2503-0

[11] K. Czajkowski and I. Foster and C. Kesselman and V. Sander and S. Tuecke SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems. Proceedings of the 8th Workshop on Job Scheduling Strategies for Parallel Processing, 2002.

[12] The Business Grid: Providing Transactional Business Processes via Grid Services Frank Leymannand, Kai Guntzel.

[13] Open Grid Forum, www.ogf.org

[14] UK National HPC Service, http://www.csar.cfs.ac.uk/

[15] The Open Middleware Infrastructure Institute, http://www.omii.ac.uk

[16] HPC-Europa, “Interim Report on Generic User Portals”, 2005

[17] Stathis Karanastasis, Vassiliki Andronikou, Piotr Grabowski, Dominik Tarnawczyk, Helene Huard, Frederic Magoules, “Portals Cluster Report On Generic Components”

[18] The Perl Directory, http://www.perl.org/

[19] ImageMagick, http://www.imagemagick.org/

[20] Andreas Menychtas, Konstantinos Dolkas, Dimosthenis Kyriazis and Theodora Varvarigou, “Building Portals to access Grid Middleware”

[21] Konstantinos Tserpes, Dimosthenis Kyriazis, Andreas Menychtas and Theodora Varvarigou, “An Open Architecture for QoS Information in Business Grids”

HPC-Europa, http://www.hpc-europa.org

[22] Elias Kourpas, “Grid Computing: Past, Present and Future An Innovation Perspective”, 2006

Παράρτημα

Πηγαίος κώδικάς του interface JobPlugin.java:

```
public interface JobPlugin {

  public String[] discoverApplications(String[] JobServiceURL);

  public String[] findResources(String ApplicationURL,
           String[] JobServiceURL);

  public String submitJob(String SLAConversationID,
      String JobServiceURL, String ApplicationURL,
      String jobDescription, String[] data,
      String[] parameters);

  public String checkJob(String JobConversationID);

  public Boolean cancelJob(String JobConversationID);

  public String[][] getJobHistory();

  public String[][] discoverJobs(String[] JobServiceURL);

  public String[] getJobOutputs(String JobConversationID);

  public Boolean storeJobOutputs(String JobConversationID,
              String[] OutputLocation);

  public Boolean endJob(String JobConversationID);
}
```

Πηγαίος κώδικάς του GRIAJobPlugin.java:

```
import java.io.File;
import java.io.IOException;
import java.rmi.RemoteException;
import java.util.*;

import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.xml.namespace.QName;

import org.apache.axis.message.addressing.*;
import org.apache.axis.types.URI;

import uk.ac.soton.ecs.iam.grid.client.staterepos.FileStateRepository;
import uk.ac.soton.ecs.iam.grid.comms.client.*;
```

```
import uk.ac.soton.itinnovation.grid.service.types.JobStatus;
import uk.ac.soton.itinnovation.grid.types.ConversationID;
import uk.ac.soton.itinnovation.grid.types.SubjectDescription;
import uk.ac.soton.itinnovation.grid.utils.ResourceTypeRegistry;


public class GRIAJobPlugin implements JobPlugin{

  private FileStateRepository repository;

  public GRIAJobPlugin (String FileRepositoryName)
  {
    try {
      repository = new FileStateRepository(FileRepositoryName);
    } catch (IOException e) {
      e.printStackTrace();
    }
  }


  public String[] discoverApplications(String[] JobServiceURL)
  {
    HashSet ApplicationSet = new HashSet();
    RemoteJobService[] jobService =
          new RemoteJobService[JobServiceURL.length];

    try {
      for (int i=0;i<JobServiceURL.length;i++)
      {
        jobService[i]= (RemoteJobService)
        repository.getOrCreateObject(
        RemoteJobService.class,ConversationID.getEPR(JobServiceURL[i]));
        ApplicationSet.addAll(
        Arrays.asList(jobService[i].getApplications()));
      }
      return (String[]) ApplicationSet.toArray(
                      new String[ApplicationSet.size()]);
    } catch (IOException e) {
      return new String[] {"error",e.getMessage()};
    }
  }

  public String[] findResources(String ApplicationURL,
                                        String[] JobServiceURL)
  {
    RemoteJobService[] jobService =
          new RemoteJobService[JobServiceURL.length];
    HashSet ResourceSet= new HashSet();

    try {
      for (int i=0;i<JobServiceURL.length;i++)
      {
        jobService[i]= (RemoteJobService)
        repository.getOrCreateObject(
        RemoteJobService.class,ConversationID.getEPR(JobServiceURL[i]));
        if ((Arrays.asList(
              jobService[i].getApplications())).contains(ApplicationURL))
          ResourceSet.add(JobServiceURL[i]);
      }
      return (String[]) ResourceSet.toArray(
```

```
                                    new String[ResourceSet.size()]);
  } catch (IOException e) {
    return new String[] {"error",e.getMessage()};
  }
}


public String submitJob(String SLAConversationID, String JobServiceURL,
                        String ApplicationURL,String jobDescription,
                        String[] data,String[] parameters)
{
  try {

    EndpointReferenceType slaServiceEPR = ConversationID.getEPR(
           "https://192.168.1.65:8443/gria-service-provider-
                                          mgt/services/SLAService");
    RemoteSLAService slaService = (RemoteSLAService)
    repository.getOrCreateObject(
                   RemoteSLAService.class,slaServiceEPR);

    EndpointReferenceType[] eprs = slaService.getResources();

    SLAConversation sla=null;

    for (EndpointReferenceType tempeprs: eprs)
      if (ConversationID.getConversationFromEPR(tempeprs)
                                                .equals(SLAConversationID))
        sla = repository.getOrCreateObject(
                                     SLAConversation.class, tempeprs);

    if (sla==null)
      return "error: no such SLA";
    if (!sla.checkUser(new SubjectDescription(
                               new AxisTransport().getSubjectCert())))
      if (sla.getValidRoles().length==0)
        return "unauthorized access to SLA";

    if (!sla.~~getSLAState~~().equalsIgnoreCase("active"))
      return "given SLA cannot be accessed because it is: " +
                                                    sla.~~getSLAState~~();
          EndpointReference jobServiceEPR =
                        new EndpointReference(JobServiceURL);
          RemoteJobService jobService = (RemoteJobService)
          repository.getOrCreateObject(
                          RemoteJobService.class,jobServiceEPR);

          JobConversation job = jobService.createJob(
                 ApplicationURL, sla.getEndpointRef(), jobDescription);

          DataConversation[] inputs = job.getInputs();

          if (inputs.length!=data.length)
           return "failed (incorrect number of inputs)";

          for(int i=0; i<data.length; i++) {
           inputs[i].save(new DataHandler(
                                     new FileDataSource(data[i])));
          }

          job.submitJob(null, parameters);
```

```
            return ConversationID.getConversationFromEPR(job.getEPR());

          } catch (URI.MalformedURIException e) {
            return "failed: " + e.getMessage();
        } catch (RemoteException e) {
            return "failed: " + e.getMessage();
        }
}

public String checkJob(String JobConversationID)
{
  try {
    JobConversation job = null;

    JobConversation[] allJobs =
              repository.getConversationsByType(JobConversation.class);
    for (JobConversation tempJob: allJobs)
      if (ConversationID.getConversationFromEPR(
                  tempJob.getEndpointRef()).equals(JobConversationID))
      {
        job=tempJob;
        break;
      }

    if (job==null)
      return "error: wrongID";

    if (job.getValidRoles().length==0)
      return "unauthorized";
    if (job.isFinished())
      return "finalized";

    JobStatus currentStatus = job.checkJob();
    Boolean InProgress = currentStatus.getInProgress();
    int ExitStatus = currentStatus.getExitStatus();

    if (InProgress)
      return "running";
    else if (!InProgress && ExitStatus==0)
      return "finished";
    else// (ExitStatus!=0)
      return "cancelled or failed";

  } catch (IOException e) {
    return "error: " + e.getMessage();
  }
}

public Boolean cancelJob(String JobConversationID)
{
  try {
    JobConversation[] allJobs =
              repository.getConversationsByType(JobConversation.class);

    JobConversation job=null;
    for (JobConversation tempJob: allJobs)
      if (ConversationID.getConversationFromEPR(
                  tempJob.getEndpointRef()).equals(JobConversationID))
          {
            job=tempJob;
            break;
```

```
        }
    if (job==null)
      return false;
    if (job.getValidRoles().length==0)
      return false;
    if (!job.isFinished() && job.checkJob().getInProgress())
    {
      job.destroy();
      return true;
    }
    else
      return false;
  } catch (IOException e) {
    return false;
  }
}

public String[][] discoverJobs(String[] JobServiceURL){

  try {
      EndpointReferenceType[] JobServiceEPR =
                  new EndpointReferenceType[JobServiceURL.length];
      RemoteJobService[] JobService =
                        new RemoteJobService[JobServiceURL.length];
      EndpointReferenceType[][] eprs =
                  new EndpointReferenceType[JobServiceURL.length][];

      ResourceTypeRegistry.registerResourceType(
          "http://www.it-innovation.soton.ac.uk/grid/resource/job",
           JobConversation.class);

      for (int i=0;i<JobServiceURL.length;i++)
      {
        JobServiceEPR[i] = ConversationID.getEPR(JobServiceURL[i]);
        JobService[i]= (RemoteJobService) repository.getOrCreateObject(
                              RemoteJobService.class,JobServiceEPR[i]);
        eprs[i] = new
              EndpointReferenceType[JobService[i].getResources().length];
        eprs[i] = JobService[i].getResources();

      }

      Vector jobVector = new Vector();

      if (eprs.length==0)
        return new String[][] {{"error","no Jobs found"}};

      for (int i=0;i<eprs.length;i++)
        for (int j=0;j<eprs[i].length;j++)
        {
          String resourceType =
                             ConversationID.getResourceType(eprs[i][j]);

            if (resourceType.equals(
              "http://www.it-innovation.soton.ac.uk/grid/resource/job"))
            {
              repository.getOrCreateObject(
                            JobConversation.class, eprs[i][j]);
              String JobID =
                    ConversationID.getConversationFromEPR(eprs[i][j]);
              String JobURI = ConversationID.getMeta(eprs[i][j],
```

```
            new QName("http://it-innovation.soton.ac.uk/2005/grid/job",
                                                    "applicationuri"));
            String myJobService = ConversationID.getParent(
                                                  eprs[i][j]).toString();
            String[] jobSLA = ConversationID.getManagingResource(
                                                 eprs[i][j]).split("#");
            String JobStatus = checkJob(JobID);
            String permission = "authorized";
            if (JobStatus.equalsIgnoreCase("UNAUTHORIZED"))
              permission = "unauthorized";
            jobVector.addElement(new String[] {JobID,JobURI,
                                 myJobService, jobSLA[0],
                             jobSLA[1], JobStatus,permission});
          }
        }

      if (jobVector.size()==0)
        return new String[][] {{"no Jobs found"}};
      String[][] discoveredJobs =
      new String[jobVector.size()][((String[])jobVector.get(0)).length];
      Object[] jobTemp = jobVector.toArray();

      for (int i=0;i<jobTemp.length;i++)
        for (int j=0;j<discoveredJobs[i].length;j++)
          discoveredJobs[i][j]=((String[]) jobTemp[i])[j];

      return discoveredJobs;
    } catch (RemoteException e) {
      return new String[][] {{"error",e.getMessage()}};
    }
  }

public String[][] getJobHistory(){
  try {
      JobConversation[] allJobs =
           repository.getConversationsByType(JobConversation.class);

      String[][] jobHistory= new String[allJobs.length][6];

      int i=0;
      if (allJobs.length==0)
        return new String[][] {};
      for (JobConversation tempJob: allJobs)
      {
        EndpointReferenceType tempJobEPR = tempJob.getEndpointRef();
        jobHistory[i][0] =
                  ConversationID.getConversationFromEPR(tempJobEPR);
        if (checkJob(jobHistory[i][0]).equalsIgnoreCase("unauthorized"))
          jobHistory[i][1] = "unknown";
        else
          jobHistory[i][1] = tempJob.checkJob().getApplicationURI();
        jobHistory[i][2] = checkJob(jobHistory[i][0]);
        jobHistory[i][3] = ConversationID.getURLFromEPR(tempJobEPR);
        jobHistory[i][4] = ConversationID.getManagingResource(
                                              tempJobEPR).split("#")[1];
        jobHistory[i][5] = ConversationID.getManagingResource(
                                              tempJobEPR).split("#")[0];
        i++;
      }
      return jobHistory;
    } catch (IOException e) {
```

```
      return new String[][] {{"error",e.getMessage()}};
    }
}

public String[] getJobOutputs(String JobConversationID){

  try {
    JobConversation[] allJobs =
                repository.getConversationsByType(JobConversation.class);


    JobConversation job=null;

    for (JobConversation tempJob: allJobs)
      if (ConversationID.getConversationFromEPR(
          tempJob.getEndpointRef()).equals(JobConversationID))

          {
            job=tempJob;
            break;
          }
    if (job==null)
      return new String[] {"error", "cannot find job"};
    if (job.getValidRoles().length==0)
      return new String[] {"error", "unauthorized to get outputs"};
    if (!checkJob(JobConversationID).equals("finished"))
      return new String[] {"error","not yet finished"};
    DataConversation[] outputs = job.getOutputs();
    String[] finalOutputs;
    if (outputs.length==0)
      return new String[] {"job has no outputs"};
    else
    {
      finalOutputs=new String[outputs.length];
      for (int i=0;i<outputs.length;i++)
        finalOutputs[i]=outputs[i].read().getName();
    }
    return finalOutputs;
  } catch (IOException e) {
    return new String[] {"error",e.getMessage()};
  }


}

public Boolean storeJobOutputs(String JobConversationID,
                                          String[] OutputLocation){

  try {
    JobConversation[] allJobs =
                repository.getConversationsByType(JobConversation.class);

    JobConversation job=null;

    for (JobConversation tempJob: allJobs)
      if (ConversationID.getConversationFromEPR(
                    tempJob.getEndpointRef()).equals(JobConversationID))
      {
        job=tempJob;
        break;
      }
```

```
      if (job==null)
        return false;
      if (!checkJob(JobConversationID).equals("finished"))
        return false;

      DataConversation[] outputs = job.getOutputs();
      if (outputs.length==0)
        return false;
      else
        for (int i=0;i<outputs.length;i++)
          outputs[i].read(new File(OutputLocation[i]));
      return true;
    } catch (IOException e) {
      return false;
    }
  }

  public Boolean endJob(String JobConversationID){

    try {
      JobConversation[] allJobs =
                repository.getConversationsByType(JobConversation.class);


      JobConversation job=null;

      for (JobConversation tempJob: allJobs)
        if (ConversationID.getConversationFromEPR(
                    tempJob.getEndpointRef()).equals(JobConversationID))

            {
              job=tempJob;
              break;
            }
      if (job==null)
        return false;
      if (job.getValidRoles().length==0)
        return false;
      if (checkJob(JobConversationID).equals("finished") ||
            checkJob(JobConversationID).equals("canceled or failed"))
      {
        job.finish();
        return true;
      }
      else
      {
        job.destroy();
        return true;
      }
    } catch (IOException e) {
      return false;
    }
  }
}
```

Πηγαίος κώδικάς του interface SLAPlugin.java:

```
public interface SLAPlugin {
```

```
  public String[][] discoverSLATemplates(String[] SLAServiceURL);

  public String[][] getSLATemplate(String SLATemplateID);

  public String proposeSLA(String SLATemplateID,
                                String SLAServiceURL, String SLALabel);

  public String proposeSLAwithAccount(String SLATemplateID,
           String AccountID, String SLAServiceURL, String SLALabel);

  public String checkSLA(String SLAConversationID);

  public Boolean closeSLA (String SLAConversationID);

  public String[][] getSLAHistory();

  public String[][] discoverSLAs(String[] SLAServiceURL);

  public String createAccount (String TradeAccountServiceURL,
              String[] AccountParameters, String[] ClientAddress);

  public String checkAccount (String TradeAccountID);

  public Boolean closeAccount (String TradeAccountID);

  public String[][] getTradeAccountHistory();

  public String[][] discoverTradeAccounts(String[]
                                       TradeAccountServiceURL);

}
```

Πηγαίος κώδικάς του GRIASLAPlugin.java:

```
import java.io.IOException;
import java.rmi.RemoteException;
import java.util.*;

import org.apache.axis.AxisFault;
import org.apache.axis.message.addressing.EndpointReferenceType;

import uk.ac.soton.ecs.iam.grid.client.staterepos.FileStateRepository;
import uk.ac.soton.ecs.iam.grid.comms.client.*;

import uk.ac.soton.itinnovation.grid.service.types.*;

import uk.ac.soton.itinnovation.grid.types.ConversationID;
import uk.ac.soton.itinnovation.grid.types.SubjectDescription;


public class GRIASLAPlugin implements SLAPlugin{

  FileStateRepository repository;

  public GRIASLAPlugin (String FileRepositoryName)
  {
    try {
```

```
    repository = new FileStateRepository(FileRepositoryName);
  } catch (IOException e) {
    e.printStackTrace();
  }
}


public String[][] discoverSLATemplates(String[] SLAServiceURL)
{
  try {
    EndpointReferenceType[] slaServiceEPR =
              new EndpointReferenceType[SLAServiceURL.length];
    RemoteSLAService[] slaService =
              new RemoteSLAService[SLAServiceURL.length];
    EndpointReferenceType[][] eprs =
              new EndpointReferenceType[SLAServiceURL.length][];
    for (int i=0;i<SLAServiceURL.length;i++)
    {
      slaServiceEPR[i] = ConversationID.getEPR(SLAServiceURL[i]);
      slaService[i]= (RemoteSLAService) repository.getOrCreateObject(
                           RemoteSLAService.class,slaServiceEPR[i]);
      eprs[i] = new EndpointReferenceType[
                                  slaService[i].getResources().length];
      eprs[i] = slaService[i].getResources();
    }

    Vector slaTemplateVector = new Vector();

    for (int i=0;i<eprs.length;i++)
      for (int j=0;j<eprs[i].length;j++)
      {
        String resourceType =
                           ConversationID.getResourceType(eprs[i][j]);
          String resourceStatus =
                          ConversationID.getResourceStatus(eprs[i][j]);
          if (resourceType.equals(
            "http://www.it-innovation.soton.ac.uk/grid/resource/
                    sla-template") && resourceStatus.equals("published"))
          {
            repository.getOrCreateObject(
                          SLATemplateConversation.class, eprs[i][j]);
            slaTemplateVector.addElement(
            new String[] {ConversationID.getConversationFromEPR(
                eprs[i][j]),ConversationID.getLabel(eprs[i][j]),
                ConversationID.getParent(eprs[i][j]).toString()});
          }
      }


    if (slaTemplateVector.size()==0)
      return new String[][] {{"error","no SLA Templates found"}};
    String[][] slaTemplate = new String[slaTemplateVector.size()]
                         [((String[])slaTemplateVector.get(0)).length];
    Object[] slaTemp = slaTemplateVector.toArray();

    for (int i=0;i<slaTemp.length;i++)
      for (int j=0;j<slaTemplate[i].length;j++)
        slaTemplate[i][j]=((String[]) slaTemp[i])[j];

    return slaTemplate;
  } catch (IOException e) {
```

```
    return new String[][] {{"error",e.getMessage()}};
  }
}

public String[][] getSLATemplate(String SLATemplateID)
{
  try {

    SLATemplateConversation[] allSLATemplates =
      repository.getConversationsByType(SLATemplateConversation.class);


    SLATemplateConversation slaTemplateConv=null;
    for (SLATemplateConversation tempslaTemplateConv: allSLATemplates)
      if (ConversationID.getConversationFromEPR(
          tempslaTemplateConv.getEndpointRef()).equals(SLATemplateID))

          {
            slaTemplateConv=tempslaTemplateConv;
            break;
          }

    String[][] slaTerms= new String[7][];

    if (slaTemplateConv==null)
      return new String[][] {{"error","no SLA Templates found"}};

    SLATemplate slaTemp = slaTemplateConv.getSLATemplate();

    Constraint[] constraint = slaTemp.getConstraints();
    PricingTerm[] pricingTerm = slaTemp.getPricingTerms();
    PermittedService[] permServ = slaTemp.getPermittedServices();

    slaTerms[0]=new String[] {"Currency",slaTemp.getCurrency()};
    slaTerms[1]=slaTemp.getSubscriptionFee()==null ? new String[]
        {"No subscription fee"} : new String[] {"Subscription fee: " +
                        slaTemp.getSubscriptionFee().toPlainString()};
    slaTerms[2]=slaTemp.getSigningFee()==null ? new String[]
         {"No signing fee"} : new String[] {"Signing fee: " +
                        slaTemp.getSigningFee().toPlainString()};
    slaTerms[3]=new String[] {"Billing Period: " +
                     slaTemp.getBillingPeriod().prettyPrint()};

    slaTerms[4] = new String[permServ.length+1];
    slaTerms[5] = new String[constraint.length+1];
    slaTerms[6] = new String[pricingTerm.length+1];

    slaTerms[4][0]="Permitted Services: ";

    for(int i=1; i<permServ.length+1; i++) {
              slaTerms[4][i] = permServ[i-1].getURL();
          }
    slaTerms[5][0]="Constraints: ";
    for(int i=1; i<constraint.length+1; i++) {
              slaTerms[5][i] = constraint[i-1].prettyPrint();
          }
    slaTerms[6][0]="Pricing Terms: ";
    for(int i=1; i<pricingTerm.length+1; i++) {
              slaTerms[6][i] = pricingTerm[i-1].prettyPrint();
          }
```

```
    return slaTerms;
  } catch (IOException e) {
    return new String[][] {{"error",e.getMessage()}};
  }
}

public String proposeSLA(String SLATemplateID,
                         String SLAServiceURL, String SLALabel)
{
  try {

    SLATemplateConversation[] allSLATemplates =
           repository.getConversationsByType(
                        SLATemplateConversation.class);
    SLATemplateConversation slaTemplateConv=null;
    for (SLATemplateConversation tempslaTemplateConv: allSLATemplates)
      if (ConversationID.getConversationFromEPR(
           tempslaTemplateConv.getEndpointRef()).equals(SLATemplateID))

         {
           slaTemplateConv=tempslaTemplateConv;
           break;
         }

    if (slaTemplateConv==null)
      return "error: no such template id";

    EndpointReferenceType slaServiceEPR =
                           ConversationID.getEPR(SLAServiceURL);
    RemoteSLAService slaService= (RemoteSLAService)
          repository.getOrCreateObject(
              RemoteSLAService.class,slaServiceEPR);
    SLATemplateConversation slaTemplateConversation =
      repository.getOrCreateObject(
        SLATemplateConversation.class,slaTemplateConv.getEndpointRef());

    SLATemplate slaTemplate = slaTemplateConversation.getSLATemplate();
    SLAProposal slaProposal = new SLAProposal();
    slaProposal.setSlaTemplate(slaTemplate);
    slaProposal.setStartTime(new GregorianCalendar());
    EndpointReferenceType epr = slaService.createSLA(
                                              slaProposal, SLALabel);

    return ConversationID.getConversationFromEPR(epr);
  }catch (AxisFault axisFault){
    return "error: " + axisFault.getFaultReason();
  }
  catch (IOException e) {
    return "error: " + e.getMessage();
  }
}

public String proposeSLAwithAccount(String SLATemplateID,
             String AccountID, String SLAServiceURL, String SLALabel)
{
  try {

    TradeAccountConversation[] allTradeAccountConv =
      repository.getConversationsByType(TradeAccountConversation.class);
    TradeAccountConversation tradeAccountConv=null;
```

```
    for (TradeAccountConversation tempTradeAccountConv:
                                                    allTradeAccountConv)
      if (ConversationID.getConversationFromEPR(
            tempTradeAccountConv.getEndpointRef()).equals(AccountID))
      {
        tradeAccountConv=tempTradeAccountConv;
        break;
      }

    if (tradeAccountConv==null)
      return "error: no such account";
    if (checkAccount(AccountID).equals(
                                "unauthorized to access account"))
      return "error: unauthorized to access account";
    if (!tradeAccountConv.getAccountStatus().equals("open"))
      return "account unavailable";
    if (!tradeAccountConv.checkUser(new SubjectDescription(
                                new AxisTransport().getSubjectCert())))
      return "unauthorized to use account";

    SLATemplateConversation[] allSLATemplates =
      repository.getConversationsByType(SLATemplateConversation.class);
    SLATemplateConversation slaTemplateConv=null;
    for (SLATemplateConversation tempslaTemplateConv: allSLATemplates)
      if (ConversationID.getConversationFromEPR(
          tempslaTemplateConv.getEndpointRef()).equals(SLATemplateID))

          {
            slaTemplateConv=tempslaTemplateConv;
            break;
          }
    if (slaTemplateConv==null)
      return "error: no such template id";

    EndpointReferenceType slaServiceEPR =
                        ConversationID.getEPR(SLAServiceURL);
    RemoteSLAService slaService= (RemoteSLAService)
        repository.getOrCreateObject(
            RemoteSLAService.class,slaServiceEPR);
    SLATemplateConversation slaTemplateConversation =
      repository.getOrCreateObject(
        SLATemplateConversation.class,slaTemplateConv.getEndpointRef());

    SLATemplate slaTemplate = slaTemplateConversation.getSLATemplate();
    SLAProposal slaProposal = new SLAProposal();
    slaProposal.setSlaTemplate(slaTemplate);
    slaProposal.setStartTime(new GregorianCalendar());

    SLAConversation slaConv = slaService.createSLA(
          tradeAccountConv.getEndpointRef(),slaProposal, SLALabel);

    return ConversationID.getConversationFromEPR(
                                            slaConv.getEndpointRef());
  }catch (AxisFault axisFault){
    return "error: " + axisFault.getFaultReason();
  }
  catch (IOException e) {
    return "error: " + e.getMessage();
  }
}
```

```
public String checkSLA(String SLAConversationID)
{

try {
    SLAConversation[] allSLAs =
            repository.getConversationsByType(SLAConversation.class);

    SLAConversation sla=null;

    for (SLAConversation tempSLA: allSLAs)
        if (ConversationID.getConversationFromEPR(
                tempSLA.getEndpointRef()).equals(SLAConversationID))
        {
          sla=tempSLA;
          break;
        }

    if (sla==null)
      return "error: wrongID";
    if (!sla.checkUser(new SubjectDescription(
                    new AxisTransport().getSubjectCert())))
      if (sla.getValidRoles().length==0)
        return "UNAUTHORIZED";
    if (sla.isFinished())
      return "finalized";
    else
      return sla.getSLAState();

  } catch (IOException e) {
    return "error: " + e.getMessage();
  }
}

public Boolean closeSLA (String SLAConversationID)
{
  try {

    SLAConversation[] allSLAs =
                repository.getConversationsByType(SLAConversation.class);

    SLAConversation sla=null;

    for (SLAConversation tempSLA: allSLAs)
      if (ConversationID.getConversationFromEPR(
            tempSLA.getEndpointRef()).equals(SLAConversationID))
      {
        sla=tempSLA;
        break;
      }
    if (sla==null)
    {
      System.out.println("null");
      return false;
    }
    if (!sla.checkUser(new SubjectDescription(
                          new AxisTransport().getSubjectCert())))
    {
      if (sla.getValidRoles().length==0)
        {
          System.out.println("certificate");
          return false;
```

```
        }
    }

    if (sla.getSLAState().equalsIgnoreCase("closed") ||
                    sla.getSLAState().equalsIgnoreCase("closing"))
      return true;
    else
      sla.closeSLA();

    return true;
  } catch (IOException e) {
    return false;
  }
}

public String[][] getSLAHistory(){

  SLAConversation[] allSLAs =
              repository.getConversationsByType(SLAConversation.class);

  String[][] slaHistory= new String[allSLAs.length][4];

  int i=0;
  if (allSLAs.length==0)
    return new String[][] {};
  for (SLAConversation tempSLA: allSLAs)
  {
      EndpointReferenceType tempSLAEPR = tempSLA.getEndpointRef();
      slaHistory[i][0] = ConversationID.getConversationFromEPR(
                                                        tempSLAEPR);
      slaHistory[i][1] = ConversationID.getLabel(tempSLAEPR);
      slaHistory[i][2] = checkSLA(slaHistory[i][0]);
      if (!slaHistory[i][2].equalsIgnoreCase("active"))
        slaHistory[i][3] = "unauthorized";
      else
      {
        if (!slaHistory[i][2].equals("unauthorized to check"))
          slaHistory[i][3] = "authorized";
        else
          slaHistory[i][3] = "unauthorized";
      }
      i++;
  }
  return slaHistory;
}

public String[][] discoverSLAs(String[] SLAServiceURL){

try {
    EndpointReferenceType[] slaServiceEPR =
                  new EndpointReferenceType[SLAServiceURL.length];
    RemoteSLAService[] slaService =
                  new RemoteSLAService[SLAServiceURL.length];
    EndpointReferenceType[][] eprs =
                  new EndpointReferenceType[SLAServiceURL.length][];
    for (int i=0;i<SLAServiceURL.length;i++)
    {
      slaServiceEPR[i] = ConversationID.getEPR(SLAServiceURL[i]);
      slaService[i]= (RemoteSLAService) repository.getOrCreateObject(
                            RemoteSLAService.class,slaServiceEPR[i]);
      eprs[i] =
```

```
        new EndpointReferenceType[slaService[i].getResources().length];
      eprs[i] = slaService[i].getResources();
    }
    Vector slaVector = new Vector();

    if (eprs.length==0)
      return new String[][] {{"no SLAs found"}};

    for (int i=0;i<eprs.length;i++)
      for (int j=0;j<eprs[i].length;j++)
      {
        String resourceType =
                  ConversationID.getResourceType(eprs[i][j]);

          if (resourceType.equals(
            "http://www.it-innovation.soton.ac.uk/grid/resource/sla"))
          {
            repository.getOrCreateObject(
                                    SLAConversation.class, eprs[i][j]);
            String SLAID =
                      ConversationID.getConversationFromEPR(eprs[i][j]);
            String SLALabel = ConversationID.getLabel(eprs[i][j]);
            String SLAService = ConversationID.getParent(
                                           eprs[i][j]).toString();
            String SLAStatus = checkSLA(SLAID);
            String permission = "authorized";
            if (SLAStatus.equals("UNAUTHORIZED"))
              permission = "unauthorized";
            slaVector.addElement(new String[] {
                   SLAID,SLALabel, SLAService, SLAStatus,permission});
          }
      }

    if (slaVector.size()==0)
      return new String[][] {{"no SLAs found"}};
    String[][] discoveredSLA =
      new String[slaVector.size()][((String[])slaVector.get(0)).length];
    Object[] slaTemp = slaVector.toArray();

    for (int i=0;i<slaTemp.length;i++)
      for (int j=0;j<discoveredSLA[i].length;j++)
        discoveredSLA[i][j]=((String[]) slaTemp[i])[j];

    return discoveredSLA;
  } catch (RemoteException e) {
    return new String[][] {{"error",e.getMessage()}};
  }
}

public String createAccount (String TradeAccountServiceURL,
                     String[] AccountParameters, String[] ClientAddress)
{
  try {
    EndpointReferenceType tradeAccountServiceEPR =
      ConversationID.getEPR(TradeAccountServiceURL);

    RemoteTradeAccountService tradeAccountService =
      (RemoteTradeAccountService) repository.getOrCreateObject(
          RemoteTradeAccountService.class,tradeAccountServiceEPR);

    AddressType clientAddress = new AddressType();
```

```
    clientAddress.setCity(ClientAddress[0]);
    clientAddress.setCountryCode(ClientAddress[1]);
    clientAddress.setPostalCode(ClientAddress[2]);
    clientAddress.setRegion(ClientAddress[3]);
    clientAddress.setState(ClientAddress[4]);
    clientAddress.setStreet(ClientAddress[5]);
    clientAddress.setTown(ClientAddress[6]);

    TradeAccountConversation tradeAccount =
      tradeAccountService.openAccount(
          AccountParameters[0],AccountParameters[1],
            AccountParameters[2],clientAddress,
              AccountParameters[3],AccountParameters[4]);
    return ConversationID.getConversationFromEPR(tradeAccount.getEPR());
  } catch (IOException e) {
    return "error: " + e.getMessage();
  }
}

public String checkAccount (String TradeAccountID)
{
  try {

    TradeAccountConversation[] allTradeAccountConv =
      repository.getConversationsByType(TradeAccountConversation.class);
    TradeAccountConversation tradeAccountConv=null;
    for (TradeAccountConversation tempTradeAccountConv:
                                                  allTradeAccountConv)
      if (ConversationID.getConversationFromEPR(
        tempTradeAccountConv.getEndpointRef()).equals(TradeAccountID))
      {
        tradeAccountConv=tempTradeAccountConv;
        break;
      }

    if (tradeAccountConv==null)
      return "error: no such account exists";

    if (!(tradeAccountConv.getAccountStatus().equals(
                                    "pending-credit-checks") ||
        tradeAccountConv.getAccountStatus().equals("denied") ||
        tradeAccountConv.getAccountStatus().equals(
                                    "account-usage-finished") ||
        tradeAccountConv.getAccountStatus().equals("suspended")))
        if (!tradeAccountConv.checkUser(new SubjectDescription(
                              new AxisTransport().getSubjectCert())))
          return "unauthorized";

    return tradeAccountConv.getAccountStatus();
  } catch (IOException e) {
    return "error: " + e.getMessage();
  }
}

public Boolean closeAccount (String TradeAccountID)
{
  try {
    TradeAccountConversation[] allTradeAccountConv =
        repository.getConversationsByType(
            TradeAccountConversation.class);
    TradeAccountConversation tradeAccountConv=null;
```

```
    for (TradeAccountConversation tempTradeAccountConv:
                                                allTradeAccountConv)
      if (ConversationID.getConversationFromEPR(
          tempTradeAccountConv.getEndpointRef()).equals(TradeAccountID))
      {
        tradeAccountConv=tempTradeAccountConv;
        break;
      }

    if (tradeAccountConv==null)
      return false;
    if ((tradeAccountConv.getAccountStatus().equals(
                                        "pending-credit-checks") ||
            tradeAccountConv.getAccountStatus().equals("denied") ||
            tradeAccountConv.getAccountStatus().equals(
                                        "account-usage-finished")))
      return false;
    if (!tradeAccountConv.checkUser(new SubjectDescription(
                                new AxisTransport().getSubjectCert())))
      return false;

    tradeAccountConv.closeAccount();
    return true;
  } catch (IOException e) {
    return false;
  }
}

public String[][] getTradeAccountHistory(){

  TradeAccountConversation[] allTAs =
    repository.getConversationsByType(TradeAccountConversation.class);

  String[][] taHistory= new String[allTAs.length][4];

  int i=0;
  if (allTAs.length==0)
    return new String[][] {};
  for (TradeAccountConversation tempSLA: allTAs)
  {
      EndpointReferenceType tempTAEPR = tempSLA.getEndpointRef();
      taHistory[i][0]= ConversationID.getConversationFromEPR(tempTAEPR);
      taHistory[i][1] = ConversationID.getLabel(tempTAEPR);
      taHistory[i][2] = checkAccount(taHistory[i][0]);
      if (!taHistory[i][2].equalsIgnoreCase("open"))
        taHistory[i][3] = "unauthorized";
      else
      {
        if (!taHistory[i][2].equals("unauthorized"))
          taHistory[i][3] = "authorized";
        else
          taHistory[i][3] = "unauthorized";
      }
      i++;
  }
  return taHistory;
}

public String[][] discoverTradeAccounts(String[] TradeAccountServiceURL)
{
  try {
```

```
        EndpointReferenceType[] TAServiceEPR = new
                 EndpointReferenceType[TradeAccountServiceURL.length];
        RemoteTradeAccountService[] TAService = new
                 RemoteTradeAccountService[TradeAccountServiceURL.length];
        EndpointReferenceType[][] eprs = new
                 EndpointReferenceType[TradeAccountServiceURL.length][];
        for (int i=0;i<TradeAccountServiceURL.length;i++)
        {
          TAServiceEPR[i] =
                 ConversationID.getEPR(TradeAccountServiceURL[i]);
          TAService[i]= (RemoteTradeAccountService)
                repository.getOrCreateObject(
                     RemoteTradeAccountService.class,TAServiceEPR[i]);
          eprs[i]=new EndpointReferenceType[
                                TAService[i].getResources().length];
          eprs[i] = TAService[i].getResources();
        }

        Vector TAVector = new Vector();

        for (int i=0;i<eprs.length;i++)
          for (int j=0;j<eprs[i].length;j++)
          {
            String resourceType= ConversationID.getResourceType(eprs[i][j]);
              if (resourceType.equals(
                         "http://www.it-innovation.soton.ac.uk/
                                        grid/resource/trade-account"))
              {
                repository.getOrCreateObject(
                              TradeAccountConversation.class, eprs[i][j]);
                String TAID = ConversationID.getConversationFromEPR(
                                                          eprs[i][j]);
                String TALabel = ConversationID.getLabel(eprs[i][j]);
                String myTAService = ConversationID.getParent(
                                                   eprs[i][j]).toString();
                String TAStatus = checkAccount(TAID);
                String permission = "authorized";
                if (TAStatus.equalsIgnoreCase("UNAUTHORIZED"))
                  permission = "unauthorized";
                TAVector.addElement(new String[] {TAID,TALabel,
                                  myTAService, TAStatus,permission});
              }
          }

        if (TAVector.size()==0)
          return new String[][] {{"no accounts present"}};
        String[][] TradeAccount = new String[TAVector.size()]
                                  [((String[])TAVector.get(0)).length];
        Object[] TATemp = TAVector.toArray();

        for (int i=0;i<TATemp.length;i++)
          for (int j=0;j<TradeAccount[i].length;j++)
            TradeAccount[i][j]=((String[]) TATemp[i])[j];

        return TradeAccount;

      } catch (IOException e) {
        return new String[][] {{"error",e.getMessage()}};
      }
    }
  }
```