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

Υλοποίηση Συστήματος για τη Διαχείριση Συμφωνιών Επιπέδου Υπηρεσιών με Χρήση Πολιτικών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αικατερίνη Χ. Μαραζοπούλου

Επιβλέπων: Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Απρίλιος 2008

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

Υλοποίηση Συστήματος για τη Διαχείριση Συμφωνιών Επιπέδου Υπηρεσιών με Χρήση Πολιτικών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αικατερίνη Χ. Μαραζοπούλου

Επιβλέπων: Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 4 Απριλίου 2008.

.....	.....	.....
Θεοδώρα Βαρβαρίγου	Βασίλειος Λούμος	Γεώργιος Στασινόπουλος
Καθηγήτρια Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Απρίλιος 2008.

.....

Αικατερίνη Χ. Μαραζοπούλου
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η τεχνολογία Πλέγματος γνωρίζει μεγάλη ανάπτυξη τα τελευταία χρόνια και έχουν γίνει σημαντικά βήματα για τη διείσδυση της τεχνολογίας αυτής στον επιχειρηματικό τομέα. Στην κατεύθυνση αυτή είναι ουσιαστική η συμβολή των Συμφωνιών Επιπέδου Υπηρεσιών (Service Level Agreements), καθώς η διασφάλιση ποιότητας υπηρεσιών που παρέχουν αποτελεί απαραίτητη προϋπόθεση για την εμπορική αξιοποίηση οποιασδήποτε τεχνολογίας.

Σκοπός της διπλωματικής αυτής εργασίας είναι η υλοποίηση ενός συστήματος διαχείρισης Συμφωνιών Επιπέδου Υπηρεσιών που βασίζεται στη χρήση πολιτικών. Οι πολιτικές ορίζονται από τον παροχέα υπηρεσιών και επιτρέπουν την προσαρμογή της συμπεριφοράς του απέναντι στους πελάτες, ανάλογα με το επιχειρηματικό μοντέλο που θέλει να ακολουθήσει.

Για την υλοποίηση του συστήματος αυτού χρησιμοποιήθηκε το μεσισμικό GRIA. Ειδικότερα επεκτάθηκε το πακέτο Service Provider Management ώστε να υποστηρίζει τη δημιουργία και τη διαχείριση των πολιτικών, καθώς και τη χρήση αυτών στη διαχείριση των SLAs.

Στην εργασία αυτή γίνεται αρχικά μια εισαγωγή στο Πλέγμα, στις Συμφωνίες Επιπέδου Υπηρεσιών και στο μεσισμικό GRIA. Ακολουθεί λεπτομερής περιγραφή της εγκατάστασης και ρύθμισης του λογισμικού που χρησιμοποιήθηκε. Τέλος περιγράφεται η μελέτη, ο σχεδιασμός και η υλοποίηση του συστήματος, τα διάφορα προβλήματα που αντιμετωπίστηκαν και δίνονται ιδέες για μελλοντικές εξελίξεις.

Λέξεις Κλειδιά — Πλέγμα, Συμφωνία Επιπέδου Υπηρεσιών, SLA, Πολιτική, μεσισμικό, GRIA, διαχείριση

Abstract

Grid computing is an emerging technology and during the last years important steps have been made towards a more business oriented Grid. Service Level Agreements constitute an important tool towards that direction, as the Quality of Service guarantees that they offer, are a vital factor for any commercial application.

The purpose of this thesis is the implementation of a system that manages Service Level Agreements (SLAs) through the use of policies. These policies are specified by the service provider and they allow him to adjust his attitude towards the clients depending on the business model he employs.

For the implementation of the aforementioned system, the GRIA middleware was used. To be more specific, the Service Provider Management was modified in order to support the creation and management of policies, as well as the use of policies in SLA management.

First we introduce the basic concepts of the Grid, the Service Level Agreements and the GRIA middleware. Then we describe in detail the installation and configuration of the software that was used. Finally, the design and implementation of the system are presented, as well as the problems that arose during the elaboration of this thesis and possible future work.

Keywords — Grid, Service Level Agreement, SLA, Policy, middleware, GRIA, Service Provider Management

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1

Εισαγωγή

Τα τελευταία χρόνια παρατηρείται μια στροφή προς τη χρήση κατανεμημένων υπολογιστικών πόρων, προκειμένου να αυξηθεί η απόδοση των εφαρμογών. Για την ομαλή χρήση των πόρων αυτών, είναι απαραίτητη η ύπαρξη συμφωνιών οι οποίες θα καθορίζουν τους όρους που διέπουν τη σχέση ανάμεσα σε αυτόν που προσφέρει μια υπηρεσία και αυτόν που τη χρησιμοποιεί. Η διαχείριση αυτών των συμφωνιών επιπέδου υπηρεσιών είναι μια διαδικασία πολύπλοκη, αφού απαιτείται να είναι σε μεγάλο βαθμό αυτοματοποιημένη, δηλαδή να μην απαιτεί ανθρώπινη παρέμβαση.

Σκοπός της διπλωματική αυτής εργασίας είναι η εισαγωγή πολιτικών για την αποτελεσματικότερη διαχείριση των συμφωνιών επιπέδου υπηρεσιών.

1.1 Οργάνωση του εγγράφου

Η εργασία αυτή αποτελείται από οκτώ κεφάλαια και δύο παραρτήματα.

Το παρόν κεφάλαιο αποτελεί μια εισαγωγή για την εργασία αυτή. Περιγράφεται συνοπτικά ο σκοπός της εργασίας και η δομή του εγγράφου.

Στο δεύτερο κεφάλαιο δίνεται αρχικά ένας ορισμός για το Πλέγμα και στη συνέχεια παρουσιάζονται ορισμένα ιστορικά στοιχεία. Ακολουθεί μια περιγραφή των χαρακτηριστικών και της αρχιτεκτονικής σχεδίασης των Πλεγμάτων.

Στο τρίτο κεφάλαιο γίνεται μια εισαγωγή στις συμφωνίες επιπέδου υπηρεσιών (SLAs), οι οποίες αποτελούν και τον πυρήνα αυτής της εργασίας.

Το τέταρτο κεφάλαιο περιλαμβάνει μια περιγραφή του μεσισμικού GRIA το οποίο χρησιμοποιήθηκε στα πλαίσια αυτής της εργασίας. Περιγράφεται επίσης αναλυτικά η αρχιτεκτονική του.

Στο πέμπτο κεφάλαιο περιγράφεται λεπτομερώς η διαδικασία εγκατάστασης του λογισμικού που χρησιμοποιήθηκε, καθως και οι ρυθμίσεις που χρειάστηκε να γίνουν.

Το έκτο κεφάλαιο περιλαμβάνει τη μελέτη και το σχεδιασμό του συστήματος. Παρουσιάζονται δηλαδή οι λειτουργικές και μη λειτουργικές απαιτήσεις του συστήματος.

Στο έβδομο κεφάλαιο περιγράφεται η υλοποίηση του συστήματος.

Τέλος στο όγδοο κεφάλαιο παρουσιάζονται τα συμπεράσματα που εξήχθησαν από την εργασία αυτή, καθώς επίσης και ορισμένες ιδέες για μελλοντικές εξελίξεις και βελτιώσεις του συστήματος.

Στο παράρτημα Α′ παρατίθεται μια λίστα των αρχείων κώδικα που δημιουργήθηκαν ή τροποποιήθηκαν για την εκπόνηση της εργασίας αυτής.

Στο παράρτημα Β′ δίνεται ο πηγαίος κώδικας για τις σημαντικότερες λειτουργίες του συστήματος που υλοποιήθηκε.

Ακολουθούν τέλος οι βιβλιογραφικές αναφορές.

Κεφάλαιο 2

Γενικά Στοιχεία για το Περιβάλλον Πλέγματος

Περιεχόμενα

Η χρήση ηλεκτρονικών υπολογιστών σε περιβάλλον Πλέγματος θεωρείται ολοένα και συχνότερα ως υποδομή νέας γενιάς και προσφέρει τη δυνατότητα πρόσβασης σε κατανεμημένους και ετερογενείς πόρους με στόχο την παροχή υπολογιστικής ισχύος σε εφαρμογές με υψηλές απαιτήσεις σε πόρους [1, 2, 3]. Στο κεφάλαιο αυτό παρατίθεται αρχικά ο ορισμός του Πλέγματος ("Grid") και στη συνέχεια παρουσιάζονται θέματα για το περιβάλλον Πλέγματος όπως ιστορικά στοιχεία, χαρακτηριστικά, ταξινόμηση σε κατηγορίες με βάση διάφορα κριτήρια και πλεονεκτήματα από τη χρήση αυτού.

2.1 Ορισμός του Πλέγματος

Η τεχνολογία των υπολογιστών και των τηλεπικοινωνιών γνωρίζει ραγδαία ανάπτυξη τα τελευταία χρόνια. Συνεχώς αναπτύσσονται εφαρμογές ιδιαίτερα απαιτητικές σε υπολογιστική ισχύ και χώρο αποθήκευσης δεδομένων. Παράλληλα, εντείνεται η ζήτηση για πρόσβαση σε πληροφορίες που βρίσκονται αποθηκευμένες σε διάφορα γεωγραφικά σημεία, γεγονός που καταδεικνύει την ανάγκη για τη διασύνδεση των κατανεμημένων πόρων και υποδομών με ηλεκτρονικά δίκτυα (electronic networks) και για ειδικευμένο ενδιάμεσο λογισμικό (middleware), με φιλικό προς το χρήστη περιβάλλον που θα αποκρύπτει 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 ενδιάμεσου λογισμικού, ονομάζεται Πλέγμα Υπολογιστικών Συστημάτων ή απλά Grid. Η ενοποίηση των δικτύων και του ενδιάμεσου λογισμικού σε μια ενιαία υποδομή με στόχο την κατανεμημένη αλλά ομοιογενή πρόσβαση στους πόρους του Grid αναφέρεται ως Ηλεκτρονική Υποδομή (e-Infrastructure).

Τα τελευταία χρόνια η ταχύτατη εξάπλωση του Διαδικτύου (Internet) σε συνδυασμό με τη διαθεσιμότητα δικτύων κορμού υψηλών ταχυτήτων και την 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 (στους οποίους περιλαμβάνονται υπολογιστική ισχύς, δεδομένα, υπηρεσίες, εργαλεία λογισμικού, επιστημονικά όργανα, κ.ά.) βρίσκονται κατανεμημένοι παγκοσμίως, δημιουργώντας έτσι 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 σε μια τεράστια ποικιλία από υπολογιστικούς πόρους που διατίθενται στο Διαδίκτυο. Το Grid είναι ένα είδος παράλληλου και κατανεμημένου συστήματος που δίνει τη δυνατότητα να μοιραζόμαστε, να επιλέγουμε και να συγκεντρώνουμε πόρους που κατανέμονται σε πολλούς διοικητικούς τομείς (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 υπερυπολογιστικού 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.

2.2 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 υπολογιστών. Όμως οι αρχικές ιδέες γύρω από το 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 στις 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 ο Ιστός (Web) που 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 της χρησιμοποίησής τους για την επίλυση προβλημάτων.

Υπάρχει πληθώρα τέτοιων προβλημάτων τα οποία μπορούν να επιλυθούν πολύ καλύτερα με τη βοήθεια της προηγμένης τεχνολογίας υπολογιστών. Κοινό χαρακτη-

ριστικό πολλών τέτοιων προγραμμάτων είναι οι υψηλές απαιτήσεις σε υπολογιστική ισχύ. Αυτό σημαίνει ότι όσο πιο μεγάλη υπολογιστική ισχύ διαθέτει κάποιος, τόσο πιο ακριβής είναι η απάντηση στο πρόβλημα που επιχειρεί να επιλύσει. Επιπλέον, ορισμένα προβλήματα δε μπορούν να επιλυθούν χωρίς επαρκή υπολογιστική ισχύ. Οι υπολογιστικά απαιτητικές εργασίες αποτελούν την κύρια δύναμη που δίνει ώθηση στην εξέλιξη των υπολογιστικών δομών.

Η ανάγκη για την επίλυση των προβλημάτων αυτών οδήγησε τους ανθρώπους στην ανάπτυξη των υπολογιστών και στη συνέχεια των υπερυπολογιστών. Οι μεγάλοι υπολογιστικοί πόροι της IBM της δεκαετίας του 1960 και οι υπερυπολογιστές Cray στις δεκαετίες 1970 και 1980 κυριαρχούσαν στα υπολογιστικά κέντρα για πάνω από δύο δεκαετίες. Στη συνέχεια επικράτησαν οι αρχιτεκτονικές συμμετρικής πολυεπεξεργασίας (SMP - Symmetric Multiprocessing) και μαζικής παράλληλης επεξεργασίας (MRR - Massive Parallel Processing) που περιελάμβαναν πολλούς επεξεργαστές οι οποίοι είχαν τη δυνατότητα να λειτουργούν παράλληλα. Για την αποτελεσματικότερη χρήση τους, οι προγραμματιστές εφαρμογών έπρεπε να δομήσουν τον κώδικά τους έτσι ώστε να επιτρέπει την παράλληλη εκτέλεσή του. Έπειτα εμφανίστηκαν τα clusters (συστάδες), 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 του 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 ρυθμοαπόδοση (throughput), τυχαίες συμφορήσεις του δικτύου (τυπικό για το TCP/IP) και σφάλματα δικτύου ήταν 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 τα οποία 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 καθοριζόταν μια τέτοια πολιτική, με την υπάρχουσα τεχνολογία δεν υπήρχαν τα απαραίτητα τεχνικά μέσα για τον αποτελεσματικό έλεγχό της και την επιβολή της.

Η ερευνητική δραστηριότητα στον τομέα των δικτύων υπολογιστών συνεχίστηκε, με αποτέλεσμα τη 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 και δρομολόγησης εργασιών. 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) και GLOBUS (1998). Το πρώτο βασίζεται στην ιδέα του εικονικού υπολογιστή (virtual computer), δηλαδή όλοι οι πόροι είναι συνδεδεμένοι μεταξύ τους και εμφανίζονται στο χρήστη ως μία εικονική μηχανή. Έχει όμως αρκετά μειονεκτήματα 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, τη μεταφορά δεδομένων, την ανακάλυψη πόρων και την εκτέλεση εργασιών. Λειτουργεί σε χαμηλό επίπεδο και είναι δομημένο σε επίπεδα υπηρεσιών, όπως φαίνεται και στο σχήμα 2.1.

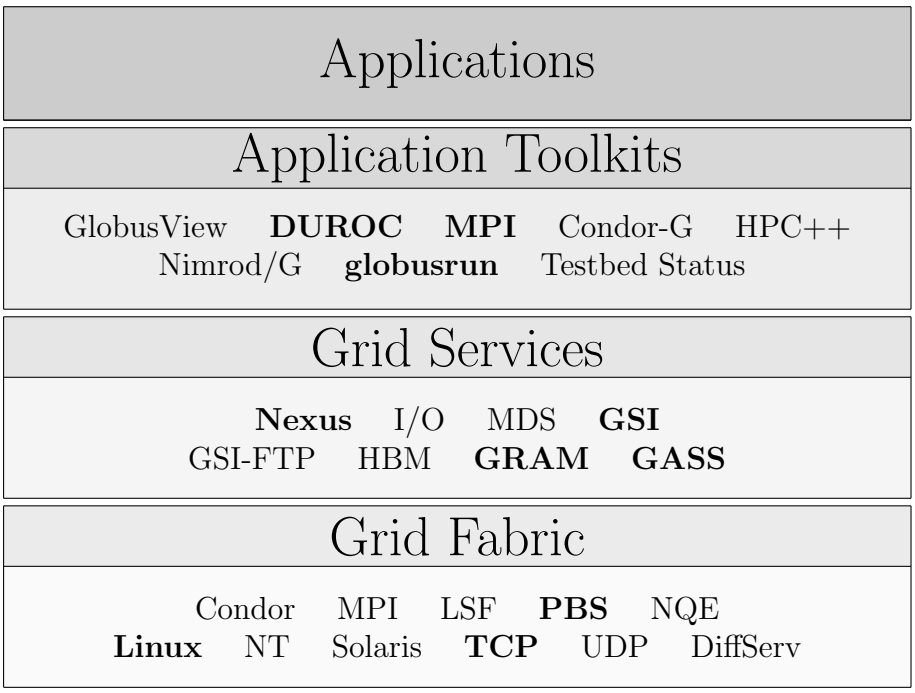
Η συνέχεια γίνεται με την ανάπτυξη των Web services (υπηρεσίες Ιστού) (2001). Πρόκειται για υπηρεσίες που είναι προσβάσιμες μέσω του Διαδικτύου και έχει επικρατήσει να χρησιμοποιούν συγκεκριμένες τεχνολογίες και πρωτόκολλα (XML, SOAP, WSDL).

Τέλος ακολουθεί η εγκαθίδρυση της Αρχιτεκτονικής Ανοιχτών Υπηρεσιών Grid (Open Grid Service Architecture - OGSA) (2002) ως κύριας αρχιτεκτονικής της Grid τεχνολογίας (όπως περιγράφεται στην ενότητα 2.8). Το πρότυπο αυτό είναι το πιο σημαντικό για τις τεχνολογίες Πλέγματος.

2.3 Χαρακτηριστικά των Πλεγμάτων

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

- Επιτρέπουν το διαμοιρασμό των πόρων σε πολλούς χρήστες διαφορετικών κοινοτήτων με ετερογενή πεδία εφαρμογών και οι οποίοι δε βρίσκονται στην ίδια



Σχήμα 2.1: Αρχιτεκτονική Globus

γεωγραφική περιοχή. Ένα Grid, ανάλογα με τις απαιτήσεις των εφαρμογών και τις υπάρχουσες δικτυακές υποδομές, μπορεί να στηρίζεται σε ένα τοπικό δίκτυο (LAN - Local Area Network), σε ένα μητροπολιτικό δίκτυο (MAN - Metropolitan Area Network), σε ένα εθνικής εμβέλειας δίκτυο (WAN - Wide Area Network) ή και σε διεθνούς κάλυψης δίκτυο όπως το Ευρωπαϊκό Ερευνητικό Δίκτυο GEANT και το Αμερικανικό Abilene.

- Απαιτούν ασφαλή πρόσβαση μέσω ενδιάμεσου λογισμικού (middleware) με έμφαση στο λογισμικό ανοικτού κώδικα (open source), όπως για παράδειγμα το GLOBUS. Τα Grids επεκτείνουν την φιλοσοφία του ανοικτού λογισμικού σε ανοικτά υπολογιστικά συστήματα, με περιορισμούς μόνο σε ό,τι αφορά στην ασφάλεια και τη 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).

- Ενοποιούν μέσω δικτύων Internet / Intranet υπολογιστικές, αποθηκευτικές και άλλες ηλεκτρονικές εγκαταστάσεις με ετερογενείς τεχνολογικές υλοποιήσεις, με στόχο την παροχή ολοκληρωμένων Ηλεκτρονικών Υπηρεσιών (eServices). Η ενοποίηση υλοποιείται με χρήση ενός επιπρόσθετου στρώματος ενδιάμεσου λογισμικού (middleware) που αναλαμβάνει το διαμοιρασμό των πόρων πάνω από το δίκτυο με τα παραπάνω χαρακτηριστικά.

2.4 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 [4]), 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 και το sitewide single sign-on, δηλαδή μοναδικό σημείο όπου ο χρήστης δίνει τα προσωπικά στοιχεία του (π.χ. username και password). Πάνω σε αυτά κτίστηκαν 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). 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 ενδιάμεσου λογισμικού και πρωτοκόλλων επικοινωνιών αποτέλεσαν τη βάση για την ανάπτυξη κατανεμημένων εφαρμογών και υπηρεσιών. Τα Grids 2ης γενιάς ουσιαστικά έδωσαν τα βασικά δομικά στοιχεία, αλλά η χρήση τους απαιτούσε σημαντική προσαρμογή στις ανάγκες του χρήστη (customization) ώστε να προσφέρει μια ολοκληρωμένη λύση. Οι ανεξάρτητες αυτές προσπάθειες χρήσης συστημάτων 2ης γενιάς περιείχαν πολλές «κατά απαίτηση» επεκτάσεις λογισμικού, πράγμα που κατέστησε την διαλειτουργικότητα (interopability) προβληματική.

- Λαμβάνοντας υπόψη, τόσο την εμπειρία από τις δύο πρώτες γενιές, όσο και την τεχνολογία των πολύ επιτυχημένων Web services, έχουν ξεκινήσει οι προσπάθειες για την 3η γενιά Grids (3G Grids), που βασίζονται στην Αρχιτεκτονική Ανοιχτών Υπηρεσιών Grid (όπως περιγράφεται στην ενότητα 2.8), 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 Grids αυξάνει τον ανταγωνισμό και επιταχύνει την επίτευξη διαλειτουργικότητας όχι μόνο μεταξύ εφαρμογών και εργαλειοθηκών, αλλά κυρίως μεταξύ διαφορετικών υλοποιήσεων βασικών υπηρεσιών του Grid.

2.5 Όροι για το Πλέγμα

2.5.1 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 επικεντρώνονταν στην επίλυση υπολογιστικά απαιτητικών προβλημάτων, η οποία διευκολυνόταν 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 υφίστανται επεξεργασία 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 κάποιον 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.

2.5.2 Μη υπολογιστικοί πόροι

Όπως είπαμε προηγουμένως δεν υπάρχουν μόνο υπολογιστικοί πόροι. 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, καθώς μπορεί να υπάρχουν προγράμματα που βρίσκονται σε συγκεκριμένες τοποθεσίες λόγω του ότι 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 των επεξεργαστών, των ταχυτήτων τους 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 εκτελεστεί 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 ρυθμοαπόδοση σε ένα χρήστη.

Αξιοπιστία Για παράδειγμα, υλικό όπως οι σκληροί δίσκοι χαρακτηρίζονται από το MTBF (Mean Time Between Failures, μέσος χρόνος μεταξύ αστοχιών).

Διαθεσιμότητα Μπορεί να μην είναι δυνατή κάθε στιγμή η σύνδεση με απομακρυσμένους πόρους, για παράδειγμα λόγω βλάβης.

Δυνατότητες Μια εφαρμογή λόγου χάρη, είναι δυνατόν να εκτελεί συγκεκριμένες λειτουργίες και αλγορίθμους που άλλες εφαρμογές δεν μπορούν να εκτελέσουν.

2.5.3 Εικονοποίηση (Virtualization)

Καθώς εξελισσόταν το Grid κατέστη σαφές ότι θα πρέπει να υπάρχει ένα στρώμα υπηρεσιών (service layer) το οποίο θα επιτρέπει 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 πώς αυτή θα εκτελεστεί (αν θα εκτελεστεί για παράδειγμα από έναν πανίσχυρο επεξεργαστή ή 10 πιο αδύναμους). Αυτό ονομάζεται εικονοποίηση (virtualization) και έχει τα εξής πλεονεκτήματα:

- Καλύτερη κατηγοριοποίηση και παρουσίαση των δυνατοτήτων των πόρων, 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.

2.5.4 Υπηρεσίες Ιστού (Web services)

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

Μία υπηρεσία Ιστού 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 Ιστού 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 διευκολύνεται η ανάπτυξη και η συντήρηση των υπηρεσιών.

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 του WS-Security και του WSRF (WS-Resource Framework). Το τελευταίο χρησιμοποιείται για τη δημιουργία υπηρεσιών Ιστού οι οποίες προσφέρουν στους μετέχοντες στο Grid πρόσβαση σε πόρους. Μία υπηρεσία Ιστού μπορεί να αντιστοιχεί σε πολλούς πόρους (WS-Resources), ενώ ένα πόρος μπορεί να είναι προσπελάσιμος από διάφορες υπηρεσίες Ιστού. 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 υπηρεσίας Ιστού.

2.5.5 Εικονικοί Οργανισμοί

Το ακριβές πρόβλημα το οποίο υποκινεί την ανάπτυξη του Grid είναι ο ελεγχόμενος και συντονισμένος διαμοιρασμός πόρων και η χρήση τους 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 διαμοιρασμός αυτός θα πρέπει να είναι ελεγχόμενος, με τους παρόχους και τους χρήστες των πόρων να ακολουθούν πρωτόκολλα τα οποία θα καθορίζουν με σαφήνεια τι 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 και για όσο διάστημα χρειαστεί, ένας Ε.Ο. ο οποίος θα καθορίζει το επίπεδο ασφάλειας μεταξύ των μελών του, παρέχοντας στη συνάρτηση (κάτω από προϋποθέσεις βέβαια) πρόσβαση στις αναγκαίες υπηρεσίες.

Επομένως ένας ορισμός του Ε.Ο. μπορεί να είναι ο εξής:

> Εικονικός Οργανισμός 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. Αυτό σημαίνει ότι οι Ε.Ο. μπορεί να επικαλύπτονται μεταξύ τους.

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. Ο όρος «σχέσεις διαμοιρασμού» αναφέρεται στο ποιοι πόροι διαμοιράζονται, στον 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 ζητημάτων χρονοπρογραμματισμού και ταυτόχρονης δέσμευσης ενός συνόλου πόρων Πολλά προβλήματα απαιτούν την ταυτόχρονη χρήση πόρων για να επιλυθούν. Για αυτόν το λόγο οι σχέσεις διαμοιρασμού πρέπει να μπορούν να συνδυαστούν για τη συντονισμένη χρήση πολλών πόρων. Για παράδειγμα μια διαμοιραζόμενη υπολογιστική μονάδα (πόρος 1) που επεξεργάζεται δεδομένα από μια διαμοιραζόμενη μονάδα αποθήκευσης (πόρος 2).

Μηχανισμοί Αποστολής Δικαιωμάτων (delegation) Πρόκειται για μηχανισμούς με τους οποίους ο χρήστης εξουσιοδοτεί τον διαμοιραζόμενο πόρο με τα δικαιώματά του. Για παράδειγμα μια υπολογιστική μονάδα μπορεί να εξουσιοδοτηθεί ώστε να έχει δικαίωμα πρόσβασης στα αρχεία της οντότητας που την χρησιμοποιεί.

Σχέσεις ομοτίμων Οι σχέσεις διαμοιρασμού πολύ συχνά δεν είναι απλά της μορφής πελάτη - εξυπηρετητη αλλά ομότιμων οντοτήτων, δηλαδή οι πάροχοι μπορεί να είναι και καταναλωτές.

Διαφορετική λειτουργία ενός πόρου Ο ίδιος πόρος μπορεί να χρησιμοποιηθεί με διαφορετικό τρόπο ανάλογα με τους περιορισμούς πρόσβασης και το 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 κάποιο άλλο Ε.Ο. να χρησιμοποιείται γενικά, παρέχοντας υπολογιστική ισχύ.

2.6 Ταξινόμηση των Πλεγμάτων

Η ταξινόμηση των συστημάτων Πλέγματος μπορεί να γίνει με διάφορα κριτήρια όπως οι εφαρμογές, η επαγγελματική αξία, η επιστημονική αποτελεσματικότητα και η αρχιτεκτονική. Μια από τις δημοφιλέστερες ταξινομήσεις σχετίζεται με το επίπεδο πολυπλοκότητας της οργάνωσης και την έκταση που καταλαμβάνει το όλο σύστημα. Βάσει αυτής διακρίνονται οι ακόλουθες κατηγορίες:

Departmental Grids Εγκαθίστανται συνήθως σε έναν τομέα ενός οργανισμού και 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 υλικό που χρησιμοποιείται δε παρουσιάζει απαραίτητα ομοιογένεια. Επίσης δεν υπάρχει μεγάλη ανάγκη για καθορισμό μιας πολιτικής πρόσβασης και για 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 αξιοποίησή τους.

Enterprise Grids 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 να συγκεντρώσουν τους πόρους τους που πριν ήταν διασκορπισμένοι σε διάφορα τμήματα τους. Στην περίπτωση αίτησης χρήσης πόρων από δύο ή περισσότερους τομείς της επιχείρησης εφαρμόζονται μηχανισμοί επίλυσης διενέξεων μέσω ενός συστήματος πολιτικών χρήσης (usage policies).

Partner Grids Είναι εγκαταστάσεις που ξεπερνούν το firewall μιας επιχείρησης και επιτρέπουν την 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 2.5.5). 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 τη λογιστική (accounting).

Open Grids Αναμένεται να εμφανιστούν στο μέλλον και αναφέρονται σε μια πλατφόρμα που αποτελείται από υποδομή, μεσισμικό (middleware) 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 μερισμό (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 σε αντάλλαγμα να έχουν κάποιο 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.

2.7 Αρχιτεκτονική σχεδίαση του Πλέγματος

Η διαδικασία της δημιουργίας μιας αρχιτεκτονικής Πλέγματος είναι παρόμοια με άλλες προσπάθειες κατασκευής ενός συστήματος, ειδικά όταν αναφερόμαστε σε κατανεμημένα συστήματα. Παρ' όλα αυτά, ορισμένοι παράμετροι διαδραματίζουν σημαντικότερο ρόλο στο σχεδιασμό ενός Grid συστήματος απ' ότι άλλοι. Αυτές οι παράμετροι 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

2.7.1 Απαιτήσεις ασφάλειας

Τυπικά δεδομένα επιχειρήσεων, επιχειρησιακές πρακτικές και επιχειρησιακοί πόροι πρέπει να προστατεύονται από καταστροφή, κλοπή και γενικά από ενέργειες που έχουν ως στόχο 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 σε 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 την 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, αφού πριν γίνει κάθε επικοινωνία θα πρέπει να υπάρξει συνεννόηση με το διαχειριστή του δικτύου, έτσι ώστε να ρυθμίσει το firewall (ανοίγοντας ίσως κάποιες θύρες) και τελικά να επιτραπεί η επικοινωνία.

Με τη μετάβαση της υποδομής του Grid στις τεχνολογίες των υπηρεσιών Ιστού (Web services) και XML, καθίσταται ευκολότερο για ένα firewall να επιβλέπει τα δεδομένα που περνούν. Επίσης με την εμφάνιση των WS-Agreements (ή SLAs - Service Level Agreements), 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.

2.7.2 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 Grids, όπως τα enterprise Grids, η ασφάλεια των δεδομένων είναι σχετικά εύκολο να εξασφαλιστεί. Σε πιο ανοιχτές αρχιτεκτονικές, όπως τα partner Grids, υπάρχουν τρία επίπεδα στα οποία πρέπει να εξασφαλιστεί ασφάλεια των δεδομένων:

- Σε ποιο κόμβο μπορεί να πραγματοποιηθεί μια εργασία. Μια εργασία μπορεί να πραγματοποιηθεί σε οποιονδήποτε αξιόπιστο κόμβο του Grid. Σε αυτή την περίπτωση μια λύση είναι η δημιουργία εικονικών οργανισμών που θα καθορίζουν ποιοι κόμβοι είναι αξιόπιστοι.

- Ασφάλεια του χώρου 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 η αποθήκευση των δεδομένων.

- Ασφάλεια των δεδομένων κατά τη μεταφορά τους. Σήμερα ευτυχώς ο καθορισμός ασφαλών καναλιών μεταφοράς δεδομένων είναι δυνατός με τη χρήση των πρωτοκόλλων SSL (Secure Socket Layer) και WSS (Web Services Security). Έτσι είναι δυνατή η κρυπτογράφηση των δεδομένων που μεταφέρονται μεταξύ κόμβων.

2.7.3 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 εγκαθιστούσε σε υπολογιστές ή υπερυπολογιστές. Η ισχύς αυτή πολλές φορές έμενε ανεκμετάλλευτη, 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 αξιοποιούνταν.

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 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 μπορεί, για τις ώρες που οι απαιτήσεις είναι μέγιστες, να καταφεύγει σε προσφορές επεξεργαστικής ισχύς από τρίτους για να καλύψει τις βραχυπρόθεσμες ανάγκες της.

2.7.4 Αποθήκευση δεδομένων

Το Grid θεωρείται 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 τί θα αποθηκευτεί σε μια συγκεκριμένη τοποθεσία καθορίζεται από το σύστημα αποθήκευσης, τη διαθέσιμη χωρητικότητα, το ρυθμό μεταφοράς δεδομένων, το κόστος αποθήκευσης, το χρόνο ζωής των δεδομένων και φυσικά από την πολιτική.

Συνήθως τα πρωτόκολλα που χρησιμοποιούνται λειτουργούν με σχετικά αδιαφανή τρόπο. Αυτό σημαίνει ότι ο χρήστης δε γνωρίζει το μηχανισμό, με τον οποίο αποθηκεύεται ένα αρχείο του (αν για παράδειγμα αποθηκεύεται σε δίσκο ή ταινία). Αυτό που γνωρίζει είναι πως το πρωτόκολλο εγγυάται 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 ρυθμοαπόδοση και η διάρκεια ζωής των δεδομένων. Με βάση τα χαρακτηριστικά αυτά γίνεται και επιλογή μεταξύ 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 παρουσιάζουν αποτελεσματική κλιμάκωση, καθώς αυξάνεται η χωρητικότητα. Στην περίπτωση που χρησιμοποιείται δίσκος είναι σήμερα δύσκολο να δημιουργηθεί ένα οικονομικό σύστημα μεγάλης χωρητικότητας 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 ρυθμοαπόδοσης (throughput) αντιμετωπίζεται εύκολα στα Grid. Χάρη στην έμφυτη ικανότητα του Grid να κλιμακώνεται, υπάρχει η δυνατότητα να αυξηθεί η ρυθμοαπόδοση 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.

2.7.5 Διαθέσιμο εύρος ζώνης στο Διαδίκτυο

Αφού το Grid στηρίζεται στο Διαδίκτυο, το διαθέσιμο εύρος ζώνης (bandwidth) αποτελεί σημαντικό παράγοντα. Επομένως η ταχύτητα της σύνδεσης με το Διαδίκτυο μιας εφαρμογής που τρέχει πάνω σε Grid, είναι σημαντική. 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 στην 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 σήμερα, στην περίπτωση τουλάχιστον των επιστημονικών εφαρμογών, οι συνδέσεις με το Διαδίκτυο μπορεί να είναι της τάξης των μερικών δεκάδων Gigabit ανά δευτερόλεπτο (Gbps). Βέβαια η δυνατότητα μεταφοράς δεδομένων ενός υπολογιστή περιορίζεται στο 1 Gbps, αλλά και μόνο του αυτό σημαίνει δυνατότητα μεταφοράς 16 Gigabyte σε 16 δευτερόλεπτα. Παρ' όλα αυτά, στην 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 εν τέλει την εκτέλεση των εργασιών. Επειδή η ύπαρξη μεγάλου όγκου δεδομένων και πολλών χρηστών που τα διαχειρίζονται, όπως επίσης και η γεωγραφική διασπορά των πόρων και των χρηστών είναι χαρακτηριστικά του Grid, 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, η γεωγραφική θέση τους, ο αναμενόμενος όγκος δεδομένων, το απαιτούμενο εύρος ζώνης και η 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 πρέπει να ανανεώνεται και στην 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 στον οποίο υπάρχει ένα δεδομένο πάψει να λειτουργεί, τότε αυτό μπορεί να αναζητηθεί αυτόματα σε άλλον κόμβο.

- Για τη μεταφορά των δεδομένων μεταξύ του τόπου αποθήκευσης και του τόπου χρήσης τους χρειάζεται ένα μέσο το οποίο να παρέχει γρήγορη, αξιόπιστη και

ασφαλή μεταφορά.

- 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 μεταπληροφορίας 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.

2.7.6 Υπάρχοντες πόροι

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

Υπάρχουν μερικές κατηγορίες τέτοιων υπολογιστικών πόρων:

- Υπολογιστές γραφείου (desktop PCs), οι οποίοι υπάρχουν σχεδόν σε όλο τον κόσμο, 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).

- Clusters. Αυτά τα συστήματα είναι συνήθως διαθέσιμα όλο το εικοσιτετράωρο και είναι αξιόπιστα και ασφαλή. Η γνώση της δομής ενός cluster, των μηχανημάτων και των εξειδικευμένων πόρων που περιλαμβάνει, 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 ανεκμετάλλευτα για μεγάλα χρονικά διαστήματα. Πλέον είναι δυνατή η χρήση τους ως πόρων και είναι ιδιαίτερα χρήσιμα στην περίπτωση εργασιών που αξιοποιούν τις ειδικές δυνατότητες που διαθέτουν.

Σημαντικός είναι ο καθορισμός των εργασιών που 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.

2.7.7 Πόροι ειδικού σκοπού

Πολλοί οργανισμοί διαθέτουν πόρους οι οποίοι δημιουργήθηκαν ειδικά για την εκτέλεση μιας εργασίας ή μιας κατηγορίας εργασιών. 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 Ιστού και XML υπάρχει διαθέσιμη μια τεράστια ποικιλία βιβλιοθηκών, που επιτρέπουν τη δημιουργία κώδικα, που θα ανταποκρίνεται στις εκάστοτε ανάγκες. Ο κώδικας αυτός μπορεί να αποτελεί ένα resource wrapper ή minihosting περιβάλλον, επιτρέποντας την πρόσβαση στον πόρο μέσω των υπαρχόντων ιδιόκτητων διασυνδέσεων.

- Μια άλλη επιλογή είναι να εκτελείται σε ένα μηχάνημα μια υπηρεσία Grid, με την οποία θα επικοινωνεί αυτός που επιθυμεί την πρόσβαση στον πόρο. Αυτή η Grid υπηρεσία θα προωθεί μέσω ειδικού μηχανισμού τις απαραίτητες πληροφορίες στον πόρο. Η δεύτερη αυτή λύση προβλέπεται να είναι φθηνότερη και μάλλον θα προτιμηθεί στο μέλλον.

2.7.8 Μεταφερσιμότητα

Ένα σημαντικό ζήτημα στο σχεδιασμό κάθε Πλέγματος είναι η μετατροπή των εφαρμογών του προηγούμενου συστήματος έτσι ώστε πλέον να λειτουργούν στο περιβάλλον του Πλέγματος και να το αξιοποιούν. 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 θα αξιοποιεί τα ιδιαίτερα χαρακτηριστικά του. Παρ' όλα αυτά όμως, για να αξιοποιήσει η εφαρμογή καλύτερα το Grid σύστημα, είναι προτιμότερο να προσαρμόζεται ο ίδιος ο κώδικάς της ξαναγράφοντας μέρος του.

Υπάρχουν δύο ειδών συστατικά μέρη στο Πλέγμα: αυτά που χρησιμοποιούν τις υπηρεσίες άλλων και αυτά που παρέχουν μια υπηρεσία χωρίς να στηρίζονται σε άλλες υπηρεσίες του Πλέγματος. Και στις δύο περιπτώσεις για να δημιουργηθεί ένα τέτοιο στοιχείο, είναι απαραίτητο να λαμβάνονται συνεχώς υπόψη οι απαιτήσεις χρήσης και λογισμικού 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 Πλέγματος (ένας αλγόριθμος ίσως να μη χρειάζεται να είναι υπερβολικά αποδοτικός, αφού οι υπάρχοντες πόροι στο Πλέγμα μπορούν να καλύψουν τις απαιτήσεις ενός ίσως μη βελτιστοποιημένου αλγορίθμου).

Ορισμένες υπάρχουσες εφαρμογές μπορούν να εκτελεστούν σε ένα περιβάλλον Πλέγματος με τη βοήθεια διεπαφών, που κάνουν χρήση ήδη καθορισμένων προτύπων. Έτσι μειώνεται ο φόρτος εργασίας που χρειάζεται, για να γίνουν αυτές προσβάσιμες μέσα στο περιβάλλον Πλέγματος. 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 Πλέγματος.

2.7.9 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.

Είναι σημαντικό επομένως 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, η οποία όμως θα συνοδεύεται και από αντίστοιχα υψηλό επίπεδο ασφάλειας του συστήματος. Δυστυχώς όμως, μέχρι σήμερα, αποδεικνύεται ότι η 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.

2.8 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 [4], η οποία συντάχθηκε σε ένα κείμενο που περιγράφει τη δομή ενός Grid συστήματος και συγκεντρώνει όλα τα πρότυπα που χρησιμοποιεί η αρχιτεκτονική αυτή σε κάθε επίπεδό της.

2.8.1 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 διαφορετικότητα των πόρων που συμμετέχουν σε ένα Grid και ο δυναμικός τρόπος συνεργασίας τους απαιτεί ένα σύστημα διαχείρισης που θα προσαρμόζεται σύμφωνα με τις 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 αυτό είναι εφικτό. Το OGSA βασίζεται στη τεχνολογία των Web services (υπηρεσίες Ιστού), που περιγράφονται στην ενότητα 2.9.2.

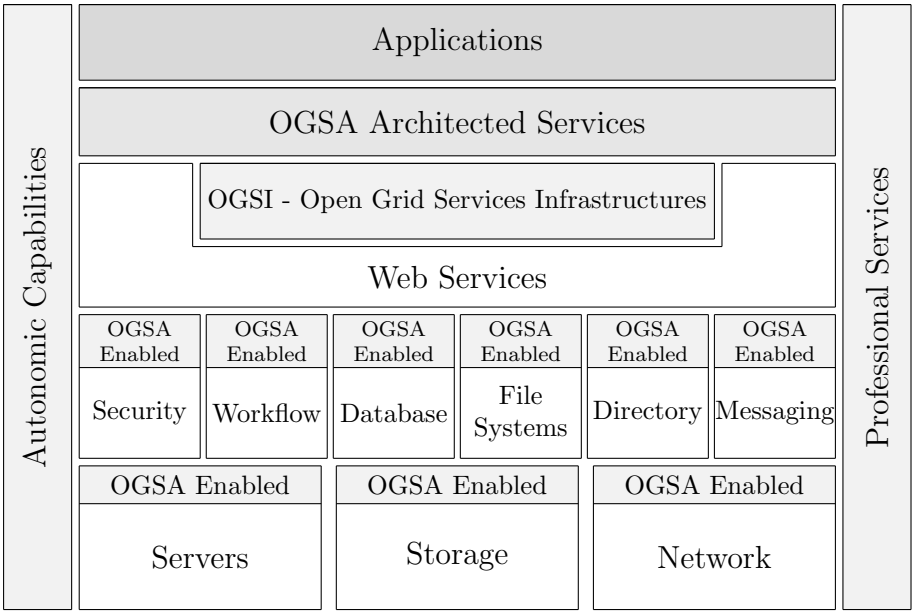
2.8.2 Περιγραφή της αρχιτεκτονικής

Το OGSA αποτελείται από τέσσερα βασικά επίπεδα (όπως φαίνεται και στο Σχήμα 2.2):

- Πόρους (φυσικούς και λογικούς)
- Υπηρεσίες Ιστού (Web services) συμπεριλαμβανομένης και της υποδομής Open Grid Services Infrastructure (OGSI) που έχει άμεση σχέση με τις υπηρεσίες
- Υπηρεσίες σχεδιασμένες από το OGSA
- Grid εφαρμογές

2.8.2.1 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 ο επεξεργαστής ενός υπολογιστή, μέχρι και το τμήμα υπολογιστών μιας μεγάλης εταιρείας. 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, κάποια ενδιάμεσα συστήματα (middleware) που, χρησιμοποιώντας τους φυσικούς πόρους, προσφέρουν στοιχειώδεις υπηρεσίες, όπως διαχείριση αρχείων ή βάσεων δεδομένων, στο παραπάνω επίπεδο.



Σχήμα 2.2: Αρχιτεκτονική του OGSA

2.8.2.2 Επίπεδο υπηρεσιών Ιστού

Μία πολύ βασική θεώρηση του OGSA είναι ότι όλοι οι πόροι του συστήματος αντιστοιχίζονται με υπηρεσίες. Έτσι η OGSI υποδομή (Open Grid Services Infrastructure) χρησιμοποιεί συγκεκριμένες, γνωστές υπηρεσίες Ιστού 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.

2.8.2.3 Επίπεδο υπηρεσιών Πλέγματος

Οι υπηρεσίες Ιστού και οι δυνατότητες του OGSI παρέχουν τη βασική υποδομή για το επόμενο επίπεδο, αυτό των υπηρεσιών Πλέγματος. Αυτή την εποχή υπάρχει μεγάλη δραστηριότητα προς την κατεύθυνση προσδιορισμού και προτυποποίησης τέτοιων υπηρεσιών, όπως υπηρεσίες υπολογισμού, πληροφοριών, πυρήνα. 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 (υπηρεσιοστρεφής αρχιτεκτονική), όπως περιγράφεται στην ενότητα 2.9.

2.8.2.4 Επίπεδο εφαρμογών Πλέγματος

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

2.9 Υπηρεσιοστρεφής Αρχιτεκτονική

Η προσαρμογή κάποιων υπηρεσιών σε μια εφαρμογή δεν είναι δύσκολη διαδικασία και επιπλέον αναμένεται να δώσει στην εφαρμογή πρόσθετα λειτουργικά χαρακτηριστικά. Αυτό όμως δεν είναι αρκετό για τη δημιουργία μιας υπηρεσιοστρεφούς αρχιτεκτονικής.

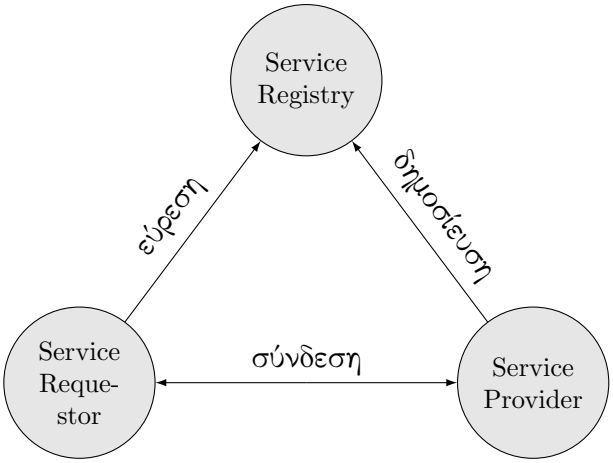
Το πρότυπο υπηρεσιοστρεφούς αρχιτεκτονικής (Service Oriented Architecture - SOA) είναι ένα σχεδιαστικό μοντέλο με κύριο χαρακτηριστικό την ενσωμάτωση λογικής εφαρμογών μέσα σε υπηρεσίες που θα αλληλεπιδρούν μέσω επικοινωνιακών πρωτοκόλλων. Βάσει αυτού, η υιοθέτηση σε μια εφαρμογή SOA δομής σημαίνει αυτόματα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

Σε επίπεδο σχεδιασμού συστημάτων, η χρήση της τεχνολογίας των Web services οδηγεί στην υιοθέτηση της λεγόμενης Service-Oriented αρχιτεκτονικής. Οι βασικοί ρόλοι και λειτουργίες στην αρχιτεκτονική αυτή παρουσιάζονται στο Σχήμα 2.3. Η αρχιτεκτονική αυτή υποδεικνύει μια σχέση εξυπηρετητή-πελάτη (server-client) ανάμεσα στον παροχέα υπηρεσιών (service provider, που παίζει το ρόλο του server) και τον αιτούντα (service-requestor, που παίζει το ρόλο του client). Ο πάροχος υπηρεσιών είναι αυτός που παρέχει την υπηρεσία δεχόμενος μηνύματα αιτήσεων από τους αιτούντες. 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 αιτών την υπηρεσία 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, πελάτη και εξυπηρετητή, σε επαφή.

2.9.1 Γενικά για τις υπηρεσίες

Τον τελευταίο καιρό έχει γίνει αρκετά μεγάλη συζήτηση γύρω από το θέμα των υπηρεσιών των εφαρμογών. Οι υπηρεσίες τείνουν να γίνουν τμήματα της εφαρμογής που αθροιστικά σχηματίζουν το περιβάλλον αυτής. Φυσικά δεν αποτελούν απλά ένα κομμάτι της εφαρμογής, αλλά έχουν χαρακτηριστικά που τις μετατρέπουν σε μέρος μιας αρχιτεκτονικής προσανατολισμένης στις υπηρεσίες (Service Oriented Architecture).

Ένα από τα χαρακτηριστικά αυτά είναι η αυτονομία από άλλες υπηρεσίες. Αυτό σημαίνει ότι κάθε υπηρεσία είναι υπεύθυνη για το δικό της εύρος λειτουργίας και έτσι περιορίζεται και εξειδικεύεται σε συγκεκριμένες επαγγελματικές χρήσεις. Αυτός ο σχεδιασμός έχει ως αποτέλεσμα τη δημιουργία ανεξάρτητων μονάδων, ελαστικά συν-



Σχήμα 2.3: Η δομή της υπηρεσιοστρεφούς αρχιτεκτονικής

δεδεμένων μεταξύ τους με κάποιο πρότυπο πλαίσιο επικοινωνίας. Εξαιτίας αυτής της ανεξαρτησίας των υπηρεσιών στο πλαίσιο αυτό, η προγραμματιστική λογική που κάθε υπηρεσία χρησιμοποιεί, δεν χρειάζεται να προσαρμόζεται σε συγκεκριμένη πλατφόρμα ή τεχνολογία. Είναι χαρακτηριστικό των υπηρεσιών ενός πλαισίου, ο πολυμορφισμός των μερών του με ταυτόχρονη ομαλή συνεργασία.

2.9.2 Υπηρεσίες Ιστού

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

- Επικοινωνία μέσω πρωτοκόλλων Internet (κυρίως HTTP)
- Αποστολή και λήψη δεδομένων δομημένων με XML

Κατά την ανάπτυξη του μοντέλου των Web services παρουσιάστηκε η ανάγκη για τη δημιουργία καινούργιων τεχνολογιών και προτύπων. Έτσι επινοήθηκε το SOAP που χρησιμοποιεί την XML για τη σύνταξη των μηνυμάτων του και το WSDL που είναι μια γλώσσα βασισμένη στην XML και η οποία χρησιμοποιείται για περιγραφή μιας υπηρεσίας Ιστού. Έτσι η υλοποίηση τέτοιων υπηρεσιών απαιτεί:

- Την περιγραφή της υπηρεσίας αναλυτικά, για παράδειγμα με ένα WSDL έγγραφο
- Την δυνατότητα μεταφοράς δεδομένων δομημένων με XML με χρήση του πρωτοκόλλου SOAP μέσω HTTP

Είναι συνηθισμένο μια υπηρεσία να λειτουργεί και ως πελάτης (client/requestor) και ως πάροχος (provider). Αναλόγως λοιπόν με τη δραστηριότητά της κάθε στιγμή, η υπηρεσία προσλαμβάνει τον κατάλληλο ρόλο.

2.9.3 Web Services Description Language (WSDL)

Οι υπηρεσίες Ιστού 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 [5], μια γλώσσα περιγραφής γνωστή ως Web Services Description Language (WSDL) η οποία βασίζεται στην XML. Ένα WSDL έγγραφο περιέχει όλες τις πληροφορίες σχετικά με τα δεδομένα εισόδου και εξόδου, τις μεθόδους και ό,τι άλλο χρειάζεται να γνωρίζει κάποιος για την ακριβή χρήση μιας διαδικτυακής υπηρεσίας.

2.9.4 Simple Object Access Protocol (SOAP)

Αν και αρχικά είχε θεωρηθεί ως η τεχνολογία που θα γεφυρώσει το κενό μεταξύ ανόμοιων πλατφόρμων βασισμένων σε RPC (Remote Procedure Call) επικοινωνία, το SOAP έχει εξελιχθεί στο πιο διαδεδομένο πρωτόκολλο επικοινωνίας για τη χρήση υπηρεσιών Ιστού. Είναι ένα πρωτόκολλο για ανταλλαγή μηνυμάτων βασισμένων στην XML. Ένα μήνυμα SOAP έχει μια συγκεκριμένη μορφή και χρησιμοποιείται για να περιγράψει δεδομένα όπως RPC κλήσεις. Το μήνυμα αυτό μεταφέρεται μεταξύ των υπηρεσιών και των εφαρμογών, χρησιμοποιώντας κυρίως το πρωτόκολλο HTTP. 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.

Η βασική δομή ενός μηνύματος SOAP 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 πρέπει να μεταφερθεί 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 μήνυμα τις απαιτούμενες πληροφορίες, σύμφωνα και με το WSDL της υπηρεσίας που θα κληθεί. Γενικά, μέσα στο «φάκελο» του SOAP μηνύματος μπορεί να μεταφέρθει οποιαδήποτε πληροφορία, αρκεί να είναι δομημένη σε XML και βέβαια ο παραλήπτης να γνωρίζει πώς να αποκωδικοποιήσει το μήνυμα αυτό.

2.10 Σύγκριση του Πλέγματος με άλλες τεχνολογίες

Η δομή του Grid αποτελεί εξέλιξη άλλων δομών και τεχνολογιών, όπως τα κα-

τανεμημένα συστήματα (distributed systems), ο Παγκόσμιος Ιστός (Web), οι υπολογισμοί ομοτίμων πόρων (peer-to-peer computing) και οι τεχνολογίες δυναμικής εικονικής οργάνωσης (dynamic virtualization). Παρακάτω παρουσιάζονται οι ομοιότητες και οι διαφορές της τεχνολογίας των συστημάτων Πλέγματος με κάθε μία από τις ανωτέρω τεχνολογίες:

Παγκόσμιος Ιστός Τα Grids, όπως και ο Παγκόσμιος Ιστός, κρύβουν όλη την πολυπλοκότητα από τους χρήστες. Σε αντίθεση όμως με το Web, το οποίο απλά χρησιμεύει για τη διευκόλυνση της επικοινωνίας, τα συστήματα Grid επιτρέπουν πλήρη συνεργασία, με σκοπό την επίτευξη κοινών επιχειρηματικών στόχων.

Ομότιμα Συστήματα Κοινό στοιχείο των ομότιμων συστημάτων και των συστημάτων τεχνολογίας Πλέγματος είναι ότι επιτρέπουν στους χρήστες να μοιράζονται αρχεία. Σε αντίθεση όμως με τα ομότιμα συστήματα, τα Grids επιτρέπουν συνεργασίες πολλών με πολλούς (many-to-many) και δεν περιορίζονται στην ανταλλαγή αρχείων, αλλά χρησιμεύουν για συναλλαγές κάθε είδους πόρων.

Κατανεμημένα Συστήματα και Συστάδες Τα Grids, όπως και τα κατανεμημένα συστήματα και τα clusters, ενοποιούν υπολογιστικούς πόρους. Όμως τα Grids μπορεί να είναι γεωγραφικά κατανεμημένα και ετερογενή, σε αντίθεση με τα κατανεμημένα συστήματα και τα clusters τα οποία απαιτούν γεωγραφική εγγύτητα και ομοιογένεια των συστημάτων.

Τεχνολογίες εικονικής οργάνωσης συστημάτων Τα Πλέγματα και οι τεχνολογίες εικονικής οργάνωσης συστημάτων (virtualization) βασίζονται στην εικονική οργάνωση πόρων. Σε αντίθεση όμως με αυτές — οι οποίες έχουν ως στόχο την οργάνωση μιας αυτόνομης εικονικής μηχανής — τα συστήματα Grid επιτρέπουν την εικονική οργάνωση μεγάλου αριθμού ανόμοιων και απομακρυσμένων πόρων.

2.11 Τα πλεονεκτήματα του Πλέγματος

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

Για να παρέχουν στους πελάτες τη λύση που χρειάζονται, οι επιχειρήσεις έχουν να λύσουν κυρίως προβλήματα που αφορούν στην ασφάλεια και στην ποιότητα των υπηρεσιών που παρέχουν στον πελάτη. 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 εταιρεία, ή ακόμη και σε ομάδες γεωγραφικά απομακρυσμένες σε σχέση με την εταιρεία. Το πιο κοινό τεχνολογικό στοιχείο που θα μπορούσε να χρησιμοποιηθεί ως τέτοιος πόρος είναι οι ηλεκτρονικοί υπολογιστές γραφείου (desktop PCs), οι οποίοι συνήθως δε λειτουργούν στο μέγιστο των δυνατοτήτων τους. Συνήθως το ποσοστό χρήσης της

υπολογιστικής τους ισχύος αντιστοιχεί στο 10% των δυνατοτήτων τους, ακόμα και στις πρωτεύουσες επαγγελματικές τους εφαρμογές.

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

Χαμηλότερο κόστος υπολογισμού Οι διάφορες πλατφόρμες Grid, ανάλογα με τον προσανατολισμό τους, δίνουν την δυνατότητα εκτέλεσης περισσότερων εργασιών με μικρότερες επενδύσεις σε διοίκηση και σε υλικό. Το Grid καταφέρνει να βελτιώσει τον λόγο απόδοση προς τιμή σε έναν οργανισμό, αλλά το αποτέλεσμα εξαρτάται 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. Αν συγκεκριμένα πρόκειται για εταιρεία, μπορεί να της δώσει τη δυνατότητα να κατακτήσει μεγαλύτερο μερίδιο αγοράς.

Καλύτερα αποτελέσματα Η αύξηση της ισχύος οδηγεί πολλές φορές σε αναθεώρηση των ερευνητικών προσανατολισμών και στη δοκιμή πολλά υποσχόμενων λύσεων, που στο παρελθόν μπορεί να 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.

2.12 Τομείς που ευνοούνται από την τεχνολογία Πλέγματος

Η τεχνολογία Grid έχει ήδη εφαρμοστεί σε ερευνητικό επίπεδο σε διάφορους τομείς και η χρήση της αναμένεται να διαδοθεί. 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 και 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. Βέβαια οι εφαρμογές αυτές απαιτούν ιδιαίτερα ισχυρούς υπολογιστικούς πόρους. Η υιοθέτηση της τεχνολογίας Πλέγματος 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 συμβάλλουν ιδιαίτερα στη μελέτη χρηματοοικονομικών αναλύσεων και στην επίλυση 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 παρέχοντες παρά καταναλωτές της Grid τεχνολογίας. Η δεύτερη κατηγορία, που σχετίζεται κατά κύριο λόγο με τη δημόσια έρευνα, θεωρείται κυρίως ως καταναλωτής της Grid τεχνολογίας. Τυπικά παραδείγματα εδώ, η NASA με τα διαστημικά της προγράμματα και η φυσική υψηλών ενεργειών, αντικείμενο έρευνας μεγάλων ινστιτούτων όπως το CERN. Άλλοι τομείς της επιστήμης, που ανήκουν στους συχνούς χρήστες της Grid τεχνολογίας, είναι 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. Στο κατασκευαστικό τομέα γίνεται σε μεγάλο βαθμό χρήση υπολογιστικών προσομοιώσεων. Τυπικές εφαρμογές που ευνοούνται στην περίπτωση αυτή είναι η δυναμική των ρευστών, ο σχεδιασμός με τη βοήθεια υπολογιστή, η ανάλυση πεπερασμένων στοιχείων και οι προσομοιώσεις συγκρούσεων.

Κεφάλαιο 3

Γενικά Στοιχεία για SLAs

Περιεχόμενα

Οι συμφωνίες επιπέδου υπηρεσιών έχουν πρωτεύοντα ρόλο σε ένα περιβάλλον Πλέγματος που βασίζεται στην υπηρεσιοστρεφή αρχιτεκτονική. Στο κεφάλαιο αυτό δίνεται αρχικά ένας ορισμός του SLA και γίνεται μια σύντομη ιστορική αναδρομή. Στη συνέχεια παρουσιάζεται η δομή και ο κύκλος ζωής ενός SLA. Τέλος αναλύεται ο ρόλος των SLAs στο Πλέγμα.

3.1 Ορισμός του SLA

Ένα SLA (Service Level Agreement ή Συμφωνία Επιπέδου Υπηρεσιών) είναι ένα συμβόλαιο ανάμεσα στον παροχέα υπηρεσιών και τον πελάτη, που καθορίζει τις λεπτομέρειες της υπηρεσίας που θα προσφερθεί.

Το SLA καθορίζει το γενικό πλαίσιο για την παροχή υπηρεσιών, δηλαδή τις υποχρεώσεις του παροχέα και του πελάτη. Σε ένα SLA προσδιορίζονται τα αποδεκτά όρια τιμών για χαρακτηριστικά όπως το επίπεδο διαθεσιμότητας ενός πόρου, η απόδοση και η διαθεσιμότητα μιας υπηρεσίας. Επιπλέον μπορεί να καθορίζει τη χρέωση του πελάτη για τις υπηρεσίες που χρησιμοποιεί καθώς και τις κυρώσεις που ενδέχεται να επιβάλλονται σε περίπτωση παραβίασης της συμφωνίας τόσο από την πλευρά του πελάτη, όσο και από την πλευρά του παροχέα υπηρεσιών.

3.2 Σύντομη ιστορική αναδρομή

Τα SLAs χρησιμοποιούνταν από τα τέλη της δεκαετίας του 1980 από τους παρόχους υπηρεσιών σταθερής τηλεφωνίας ως μέρος των συμβολαίων τους με τις επιχειρήσεις στις οποίες προσέφεραν τις υπηρεσίες τους. Αργότερα, τα τμήματα Information Technology (IT) των μεγάλων επιχειρήσεων άρχισαν να χρησιμοποιούν SLAs με τους πελάτες τους — δηλαδή με χρήστες σε άλλα τμήματα τις ίδιας επιχείρησης — ώστε να μπορούν να συγκρίνουν την ποιότητα υπηρεσιών που όντως παρέχουν με αυτή που έχουν δεσμευτεί να παρέχουν. Μέσα από τη διαδικασία αυτή άρχισαν να σκέφτονται το ενδεχόμενο να παρέχουν υπηρεσίες IT σε κάποια εξωτερική εταιρεία.

3.3 Η δομή των SLAs

Ένα SLA γενικά περιλαμβάνει λειτουργικούς και μη λειτουργικούς όρους. Οι λειτουργικοί όροι αναφέρονται στις ενέργειες που πρέπει να γίνουν προκειμένου να προσφερθεί η υπηρεσία στον πελάτη (για παράδειγμα ποιοι πόροι πρέπει να χρησιμοποιηθούν). Οι μη λειτουργικοί όροι περιγράφουν ουσιαστικά την ποιότητα της υπηρεσίας που θα προσφερθεί (για παράδειγμα ο χρόνος αντίδρασης του παροχέα υπηρεσιών σε μια αίτηση του πελάτη) [6].

3.3.1 Υπάρχουσες προδιαγραφές για SLAs

Στους κόλπους της ερευνητικής κοινότητας του Grid έχουν γίνει πολλές προσπάθειες για τον ορισμό της δομής και του περιεχομένου των SLAs. Οι δύο σημαντικότερες προσπάθειες είναι η προδιαγραφή WSLA (Web Service Level Agreement) της IBM [7] και η πιο πρόσφατη προδιαγραφή WS-Agreement του Global Grid Forum [8]. Η WSLA είναι μια γλώσσα βασισμένη στην XML (ορίζεται ως ένα XML schema), η οποία σχεδιάστηκε για την περιγραφή SLAs με ευέλικτο και εξατομικευμένο τρόπο. Το WS-Agreement είναι και αυτό μια XML γλώσσα που χρησιμοποιείται για την περιγραφή μιας συμφωνίας μεταξύ ενός παροχέα και ενός καταναλωτή υπηρεσιών, καθώς και για τον προσδιορισμό ενός πρωτοκόλλου για τη δημιουργία συμφωνιών με βάση κάποια πρότυπα συμφωνιών (templates).

Σύμφωνα με τα [9, 10] οι παραπάνω προδιαγραφές επικεντρώνονται στα τεχνικά ζητήματα των υπηρεσιών, δηλαδή στους λειτουργικούς όρους του SLA και δεν καλύπτουν ικανοποιητικά την επιχειρηματική πλεύρα του ζητήματος. Προτείνεται λοιπόν να δοθεί βάρος στους μη λειτουργικούς όρους των SLAs, που είναι σημαντικοί για τη δημιουργία μιας σχέσης εμπιστοσύνης ανάμεσα στον πελάτη και τον παροχέα.

3.4 Ο κύκλος ζωής ενός SLA

Η διαδικασία που ακολουθείται για να δημιουργηθεί ένα SLA είναι σε γενικές γραμμές η ακόλουθη:

Δημιουργία ενός SLA Template Αρχικά ο πάροχος υπηρεσιών δημιουργεί ορισμένα πρότυπα SLAs. Σκοπός είναι να δημιουργηθούν πρότυπα που θα καλύπτουν τις βασικές ανάγκες διαφόρων ομάδων χρηστών. Επίσης τα πρότυπα αυτά μπορούν να χρησιμεύσουν ως παραδείγματα για τη δημιουργία νέων προτύπων. Στο στάδιο αυτό, τα πρότυπα δεν είναι ορατά από τους πελάτες.

Δημοσίευση και Ανακάλυψη Ο πάροχος δημοσιεύει ένα πρότυπο, δηλαδή οι πελάτες μπορούν να το ανακαλύψουν και να δουν τους όρους του.

Διαπραγμάτευση Αφού ο πελάτης ανακαλύψει ένα πρότυπο SLA που τον ενδιαφέρει, μπορεί να διαπραγματευτεί τους όρους του SLA με τον παροχέα, ώστε να προσαρμοστούν αν είναι δυνατόν στις ανάγκες του πελάτη. Το στάδιο αυτό δεν είναι υποχρεωτικό, καθώς ο πελάτης μπορεί να συμφωνήσει αμέσως με ένα πρότυπο και να το υπογράψει. Για να γίνει η διαπραγμάτευση πρέπει να υπάρχει μια κοινή γλώσσα περιγραφής των όρων. Επίσης πρέπει να σημειωθεί ότι ο παροχέας υπηρεσιών είναι αυτός που υπογράφει τελευταίος τη συμφωνία, δηλαδή αυτός που έχει τον τελευταίο λόγο.

Υπογραφή Όταν οι δύο πλευρές καταλήξουν σε μια συμφωνία, τότε το SLA υπογράφεται και τίθεται σε ισχύ.

Παρακολούθηση και χρέωση Για κάθε SLA που έχει υπογραφεί πρέπει να παρακολουθείται σε πραγματικό χρόνο η χρήση που γίνεται, ώστε αφενός να ελέγχεται αν παραβιάζονται όροι του συμβολαίου και αφετέρου να υπολογίζονται οι κατάλληλες χρεώσεις.

«Κλείσιμο» ενός SLA Ο παροχέας υπηρεσιών ή ο πελάτης επιλέγει να τερματίσει τη χρήση ενός SLA. Στην περίπτωση αυτή, αποστέλλονται οι τελικοί λογαριασμοί στο χρήστη και ο πελάτης δεν μπορεί πλέον να χρησιμοποιήσει αυτό το SLA.

3.5 Ο ρόλος των SLAs στο Πλέγμα

Στον τομέα των συστημάτων Πλέγματος οι υπηρεσιοστρεφείς αρχιτεκτονικές αποκτούν όλο και μεγαλύτερη σημασία και προτιμώνται για την υλοποίηση νέων εμπορικών εφαρμογών. Αυτό σημαίνει ότι η παροχή και η κατανάλωση υπηρεσιών αποκτά πρωτεύοντα ρόλο στη συνεργασία μεταξύ διαφόρων οργανισμών. Προκειμένου να επιταχυνθεί η γρήγορη αλληλεπίδραση μεταξύ των οργανισμών, πρέπει η διαχείριση και η διάθεση των πόρων και των υπηρεσιών να γίνεται δυναμικά και αυτόματα. Όταν οι υπηρεσίες βρίσκονται σε διαφορετικούς τομείς διαχείρισης, η περιπλοκότητα της υποδομής διαχείρισης αυξάνεται.

Για να αντιμετωπιστεί αυτό το ζήτημα βασιζόμαστε σε ένα πλαίσιο διαχείρισης στο οποίο η εμπιστοσύνη του πελάτη κατοχυρώνεται μέσω ενός συμβολαίου με τον παροχέα των υπηρεσιών, δηλάδή μέσω ενός SLA. Στην περίπτωση της αντικειμενοστρεφούς αρχιτεκτονικής, κάθε υπηρεσία πρέπει να έχει ένα SLA.

Παρόλο που η αντικειμενοστρεφής αρχιτεκτονική και οι τεχνολογίες Πλέγματος έχουν αρχίσει να συνεργάζονται, υπάρχουν ορισμένα εμπόδια που πρέπει να ξεπεραστούν. Ένα ίσως από τα πιο σημαντικά είναι η εύρεση εφικτών επιχειρηματικών μοντέλων για τη χρήση πόρων και υπηρεσιών στα πλαίσια αυτής της αρχιτεκτονικής. Ενώ τα «παραδοσιακά» Πλέγματα θεωρούν δεδομένη την αμοιβαία συνεργασία μεταξύ των οργανισμών, σε ένα επιχειρηματικά βιώσιμο Πλέγμα η συνεργασία αυτή δεν είναι δεδομένη καθώς υπεισέρχεται ο εμπορικός ανταγωνισμός. Τα SLAs θα μπορούσαν να είναι το κλειδί για την επίλυση αυτού του προβλήματος. Προκειμένου να είναι χρήσιμα στα επιχειρηματικά Πλεγματα, τα SLAs πρέπει να επιλαμβάνονται τις ανάγκες όλων των εμπλεκόμενων μερών της εμπορικής σχέσης.

Κεφάλαιο 4

Το Μεσισμικό GRIA

Περιεχόμενα

Τα μεσισμικά (Middleware) για το Grid είναι στοίβες λογισμικού που σχεδιάστηκαν για να παρουσιάζουν με έναν ομοιόμορφο τρόπο υπολογιστικούς πόρους και πόρους δεδομένων που δεν έχουν ομοιογενή δομή, ώστε να μπορούν να είναι προσβάσιμοι από το λογισμικό του πελάτη χωρίς να είναι γνωστές εκ των προτέρων οι ρυθμίσεις του συστήματος [12].

Στο κεφάλαιο αυτό, παρουσιάζεται το μεσισμικό GRIA, το οποίο χρησιμοποιήθηκε για την εκπόνηση αυτής της εργασίας.

4.1 Ορισμός και χρήση του GRIA

Το GRIA (Grid Resources for Industrial Applications) είναι μια υπηρεσιοστρεφής (service-oriented) πλατφόρμα που σχεδιάστηκε για να υποστηρίζει B2B (Business to Business) συνεργασίες, παρέχοντας υπηρεσίες με ασφαλή, διαλειτουργικό και ευέλικτο τρόπο [13].

Πρόκειται για ένα open-source μεσισμικό το οποίο χρησιμοποιώντας επιχειρηματικά μοντέλα επιτρέπει στους παροχείς υπηρεσιών και τους χρήστες να ανακαλύψουν ο ένας τον άλλον και να διαπραγματευτούν τους όρους για πρόσβαση σε υπολογιστικούς πόρους. Εστιάζοντας σε επιχειρηματικές μεθοδολογίες και την αντίστοιχη σημειολογία, το GRIA δίνει τη δυνατότητα στους χρήστες να προβλέπουν τις υπολογιστικές τους ανάγκες πιο αποτελεσματικά από άποψη κόστους και να αναπτύσσουν νέα επιχειρηματικά μοντέλα για τις υπηρεσίες τους.

Το GRIA χρησιμοποιεί Grid Services, τα οποία αποτελούν μια εξειδικευμένη μορφή των Web services, και υποστηρίζουν δραστηριότητες με μεγάλη διάρκεια και αυξημένες απαιτήσεις πόρων. Οι δραστηριότητες αυτές μπορεί να απαιτούν τη συνδυαστική χρήση πόρων από διάφορους παροχείς υπηρεσιών και ενδέχεται να διαμοιράζουν τους πόρους αυτούς σε χρήστες από διαφορετικούς οργανισμούς. Τέτοιες δραστηριότητες συναντώνται συχνά σε επιχειρηματικούς τομείς όπως οι επιστήμες υγείας και τα οικονομικά, αλλά και στην ακαδημαϊκή έρευνα. Επίσης μπορεί να περιλαμβάνουν πόρους όλων των ειδών: πηγές δεδομένων (data sources), αποθήκευση (storage), εφαρμογές και επεξεργασία.

Το GRIA είναι σχεδιασμένο ώστε να υποστηρίζει τις βασικές απαιτήσεις των επιχειρηματικών εφαρμογών:

- Οι πάροχοι υπηρεσιών θα πρέπει να έχουν τη δυνατότητα να λειτουργούν ανεξάρτητα ο ένας από τον άλλο και να συναγωνίζονται για την παροχή υπηρεσιών προς τους πελάτες.
- Οι πελάτες θα πρέπει να έχουν τη δυνατότητα να ελέγχουν ποιες υπηρεσίες χρησιμοποιούν, πόσο και ποιος τις χρησιμοποιεί.
- Οι υπηρεσίες θα πρέπει να υπακούουν σε SLAs έτσι ώστε οι πάροχοι να γνωρίζουν ποιες υπηρεσίες οφείλουν να παρέχουν και οι πελάτες να γνωρίζουν ποιες υπηρεσίες τους προσφέρονται και σε ποια τιμή.

- Η παρεχόμενη ασφάλεια θα πρέπει να είναι επιπέδου εμπορικών εφαρμογών.
- Οι πάροχοι υπηρεσιών θα πρέπει να μπορούν να λειτουργούν μέσα στα όρια των σχετικών αδειών λογισμικού.
- Τα μηνύματα που ανταλλάσσονται θα πρέπει να ακολουθούν τα σχετικά πρότυπα.
- Το κόστος λειτουργίας θα πρέπει να είναι το ελάχιστο δυνατό.

4.2 Σύντομη ιστορική αναδρομή

Το GRIA project ξεκίνησε το Δεκέμβριο του 2001 με σαφή στόχο να εισάγει τη χρήση του Grid στον επιχειρηματικό και βιομηχανικό κόσμο. Ο στόχος αυτός διαχώρισε το GRIA από τις υπόλοιπες υποδομές για Grid που αναπτύσσονταν εκείνη την εποχή. Ως σημαντικότερα ζητήματα για τους χρήστες του επιχειρηματικού κόσμου προσδιορίστηκαν η ασφάλεια, τα επίπεδα υπηρεσιών (service levels) και η διαλειτουργικότητα.

Αρχικά το GRIA πρότεινε την ενσωμάτωση επιχειρηματικών μοντέλων και μεθοδολογιών στην ήδη υπάρχουσα πλατφόρμα Globus GT2 (Globus Toolkit 2). Εκείνη την περίοδο το Globus ήταν ένα από το πιο επιτυχημένα μεσισμικά για Grid, αλλά ήταν ακόμη προσανατολισμένο στις απαιτήσεις της ακαδημαϊκής κοινότητας. Η ενσωμάτωση όμως στο Globus παρουσίασε εξ αρχής πολλές δυσκολίες, καθώς αποδείχθηκε δύσκολη η υλοποίηση, η συντήρηση και η χρήση του λογισμικού.

Κατά τη διάρκεια της πρώτης φάσης του GRIA εμφανίστηκε το Open Grid Services Architecture (OGSA), οπότε κατέστη προφανές ότι η χρήση του GT2 δεν ήταν πλέον η κατάλληλη επιλογή, καθώς θα το αντικαθιστούσε σύντομα το GT3/OGSA. Ήταν εξίσου προφανές ότι θα χρειαζόταν αρκετός καιρός μέχρι το GT3 να θεωρηθεί ικανοποιητικά σταθερό ώστε να αποτελέσει τη βάση για επιχειρηματικά - εμπορικά συστήματα. Υπήρχαν βέβαια και άλλες πλατφόρμες μεσισμικών για Grid (όπως το UNICORE) που ήταν εκείνη την εποχή διαθέσιμες ή υπό ανάπτυξη, όμως δεν υπήρχε ομοφωνία ως προς τη μακροπρόθεσμη βιωσιμότητα ή την καταλληλότητα τους μετά την παρουσίαση του OGSA.

Στο σημείο αυτό (2002), το GRIA εγκατέλειψε την αρχική πρόταση για ενσωμάτωση στο GT2 και υλοποίησε μια ελαφριά (lightweighted) υποδομή για Grid χρησιμοποιώντας Web services προκειμένου να υποστηρίξει file-compute επεξεργασία σε ένα εμπορικό B2B πλαίσιο. Η χρήση Web services ήταν συνεπής προς τις απαιτήσεις του επιχειρηματικού κόσμου και προς την απόφαση της κοινότητας Grid να αναπτύξει Grid υπηρεσίες σα βελτίωση του μοντέλου των 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 παρόχου (supplier account) πριν από οποιαδήποτε τιμολόγηση ή εμπορική συναλλαγή. Η διαδικασία αυτή υποστήριζε επιπλέον διαπραγμάτευση Ποιότητας Υπηρεσιών (Quality of Service - QoS) και το επιπρόσθετο χαρακτηριστικό της προ-εγκεκριμένης λίστας παρόχων.

Η αξιολόγηση του συστήματος από τελικούς χρήστες, οδήγησε στον προσδιορισμό ποικίλων απαιτήσεων σχετικά με τη διασύνδεση χρήστη (user interface). Στο ένα άκρο ήταν οι χρήστες που απαιτούσαν διασύνδεση command-line, ενώ στο άλλο άκρο αυτοί που απαιτούσαν ένα εύχρηστο γραφικό περιβάλλον. Η έκδοση αυτή του GRIA χρησιμοποιούσε μια διασύνδεση τύπου 'wizard' η οποία δεν μπορούσε να ανταποκριθεί σε αυτές τις τόσο διαφορετικές απαιτήσεις. Για να υπάρχει μεγάλη ευελιξία, σχεδιάστηκε ένα client-side API, το οποίο επέτρεπε στους χρήστες να γράφουν τα δικά τους client-side προγράμματα για εφαρμογές χρηστών GRIA .

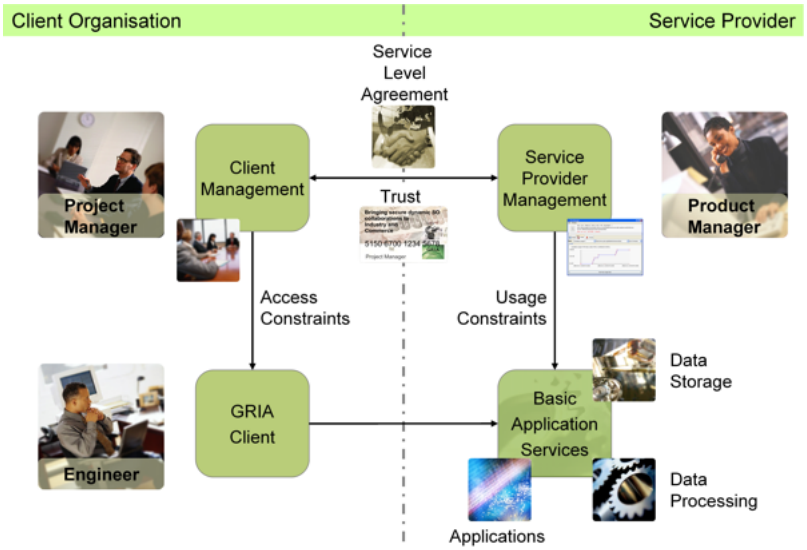
Το GRIA v2 που εκδόθηκε στις αρχές του 2004, παρείχε αυτό το client-side API μαζί με ένα μικρό αριθμό από graphical handlers για τους χρήστες που δεν ήθελαν να χρησιμοποιήσουν τη διασύνδεση command-line. Αυτό το API είχε υλοποιηθεί σε Java. Επίσης παρεχόταν μια συμπληρωματική διασύνδεση command-line προκειμένου οι χρήστες να μπορούν να γράφουν scripts που να εφαρμόζουν στο GRIA τις επιχειρηματικές μεθοδολογίες.

Η επόμενη έκδοση GRIA v3 είχε παρόμοια αρχιτεκτονική με την προηγούμενη έκδοση, αλλά με πληθώρα βελτιώσεων στην απόδοση και τη χρηστικότητα. Πιο συγκεκριμένα, περιελάμβανε ένα portal 'Enterprise Client' (υλοποιημένο με το Java API), ώστε ο τελικός χρήστης να χρειάζεται μόνο ένα συνήθη φυλλομετρητή (browser) για να έχει πλήρη πρόσβαση σε όλες τις λειτουργίες του GRIA [14].

Η επόμενη έκδοση GRIA v4 απλοποίησε ακόμα περισσότερο τη διαδικασία εγκατάστασης και χρήσης του μεσισμικού. 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. Παρ' όλα αυτά, το μοντέλο περιείχε ατέλειες και οι συμβιβασμοί που συντελέστηκαν κατά την υλοποίησή 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 υπερεκτιμούν. Από την πλευρά τους οι πάροχοι υπηρεσιών δεν μπορούσαν να καλύψουν τις υπερβολικές αυτές απαιτήσεις, γεγονός που αποτέλεσε στενωπό του μοντέλου.

Η τρέχουσα έκδοση του GRIA v5 αποτελεί την πλεόν 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 που είχε εισαχθεί στην προηγούμενη έκδοση, αντικαταστάθηκε από την υπηρεσία διαχείρισης SLAs η οποία μπορεί να ρυθμιστεί ώστε να παρακολουθεί, να περιορίζει και να χρεώνει ανάλογα με μετρικές χρήσης καθορισμένες από τον πάροχο υπηρεσιών. Τα SLAs επιτρέπουν τη διαπραγμάτευση των όρων του επιπέδου υπηρεσιών μεταξύ των παρόχων και των χρηστών, ώστε οι χρήστες να μπορούν να δίνουν ρεαλιστικές εκτιμήσεις των αναγκών τους.

Η ανάπτυξη της επόμενης έκδοσης GRIA v6 αναμένεται να ξεκινήσει την άνοιξη του 2008, αλλά το πρόγραμμα των εκδόσεων δεν έχει καθοριστεί ακόμα. Η έκδοση



Σχήμα 4.1: Αρχιτεκτονική του GRIA

αυτή θα θέσει ακόμα ψηλότερα τον πήχη στο θέμα της διαλειτουργικότητας στη χρήση των προτύπων και θα ενσωματώνει ακόμα ισχυρότερα μοντέλα εργασιακής ροής (workflow) και διαχείρισης, καθιστώντας εύκολη τη ρύθμιση ποικίλων υπηρεσιών ώστε να ικανοποιούν διαφορετικές επιχειρησιακές απαιτήσεις [13].

4.3 Αρχιτεκτονική

Το GRIA παρέχει τα εξής πακέτα λογισμικού, κάθε ένα από τα οποία απευθύνεται σε μια συγκεκριμένη ανάγκη μιας επιχείρησης:

- Basic Application Services
- Service Provider Management
- Client package
- Client Management
- OGSA-DAI application service
- Service Developers Toolkit

Στο σχήμα 4.1 φαίνονται οι υπηρεσίες του GRIA και οι μεταξύ τους αλληλεπιδράσεις.

4.3.1 Basic Application Services

Το πακέτο αυτό επιτρέπει σε έναν οργανισμό που διαθέτει εγκαταστάσεις cluster computing να παρέχει υπηρεσίες αποθήκευσης και επεξεργασίας δεδομένων. Για το σκοπό αυτό περιλαμβάνει μία υπηρεσία δεδομένων (Data Service) και μία υπηρεσία εργασιών (Job Service).

Η υπηρεσία δεδομένων δίνει τη δυνατότητα στους χρήστες να ανεβάζουν και να κατεβάζουν αρχεία δεδομένων από τον παροχέα υπηρεσιών και επιπλέον να μεταφέρουν δεδομένα μεταξύ των υπηρεσιών δεδομένων διαφορετικών παροχέων. Επίσης υποστηρίζεται διαχείριση των δικαιωμάτων πρόσβασης που έχουν αποδοθεί σε άλλους χρήστες ή σε άλλους παροχείς υπηρεσιών.

Η υπηρεσία εργασιών επιτρέπει σε απομακρυσμένους χρήστες να ξεκινούν, να παρακολουθούν ή να τερματίζουν υπολογιστικές εργασίες που εκτελούνται από τον παροχέα υπηρεσιών. Η υπηρεσία αυτή παίρνει δεδομένα εισόδου και αποθηκεύει δεδομένα εξόδου σε μια τοπική υπηρεσία δεδομένων. Η υπηρεσία εργασιών μπορεί να ρυθμιστεί ώστε να υποστηρίζει διάφορες εφαρμογές, οι οποίες έχουν επιλεγεί από τον παροχέα υπηρεσιών.

Οι υπηρεσίες εφαρμογών (application services) μπορούν να ρυθμιστούν ώστε να μπορούν να χρησιμοποιηθούν χωρίς ο πελάτης να έχει λογαριασμό στο σύστημα (unmanaged). Εναλλακτικά, μπορεί η χρήση τους να περιορίζεται σε εγγεγραμμένους πελάτες που έχουν υπογράψει κάποιο συμβόλαιο επιπέδου υπηρεσιών, οπότε και τις διαχειρίζεται το πακέτο Service Provider Management.

4.3.2 Service Provider Management

Το πακέτο αυτό επιτρέπει σε έναν πάροχο υπηρεσιών να υποστηρίζει SLAs και αν είναι απαραίτητο, να χρεώνει για τη χρήση υπηρεσιών. Αυτό γίνεται μέσω ενός απλού πρωτοκόλλου διαχείρισης που μπορεί να χρησιμοποιηθεί με υπηρεσίες εφαρμογών (για παράδειγμα με GRIA Basic Application Services). Το πακέτο αυτό περιλαμβάνει μία Υπηρεσία Λογαριασμών (Trade Account Service) και μια Υπηρεσία Διαχείρισης SLAs (SLA Management Service).

4.3.2.1 Trade Account Service

Η υπηρεσία Trade Account αναλαμβάνει τη δημιουργία και τη διαχείριση των λογαριασμών. Κάθε λογαριασμός αντιπροσωπεύει μια σχέση εμπιστοσύνης ανάμεσα στον παροχέα υπηρεσιών και τον πελάτη. Ο πελάτης είναι ο κάτοχος του ποσού που αντιστοιχεί στο λογαρισμό (budget-holder του λογαριασμού) και είναι υπεύθυνος για την πρόσβαση στο λογαριασμό όλων των υπηρεσιών. Ο πελάτης μπορεί να επιτρέψει σε άλλους να χρησιμοποιούν υπηρεσίες και να χρεώνουν το λογαριασμό του. Επίσης μπορεί να παρακολουθεί την κατάσταση του λογαριασμού του και τις χρεώσεις που έχουν γίνει ώστε να επαληθεύει την ορθότητά τους. Οι υπηρεσίες των οποίων η χρήση χρεώνεται, καταγράφουν τις κατάλληλες χρεώσεις στο λογαριασμό του πελάτη.

Ο παροχέας υπηρεσιών αναθέτει σε κάθε λογαριασμό ένα πιστωτικό όριο ώστε να περιορίσει το οικονομικό ρίσκο και καταγράφει τις χρεώσεις και τις πληρωμές που έχουν γίνει στο λογαριασμό αυτό. Η Υπηρεσία Λογαριασμών κρατάει συνεχώς στοιχεία για τα χρέη των πελατών (liability) και ελέγχει να μην ξεπερνούν το πιστωτικό όριο. Αν ξεπεραστεί το όριο αυτό, ο λογαριασμός συνεχίζει να καταγράφει τις χρεώσεις του πελάτη. Η κεντρική ιδέα γύρω από αυτό είναι να μπορεί μια άλλη υπηρεσία να ελέγξει το πιστωτικό όριο πριν δεσμευθεί να παρέχει τις υπηρεσίες της στο χρήστη.

Η Υπηρεσία Λογαριασμών δεν αντικαθιστά μια τραπεζική υπηρεσία. Καταγράφει ποσά που ο πελάτης οφείλει στον παροχέα υπηρεσιών και όχι πραγματικά χρηματικά ποσά. Δεν εγγυάται ότι αυτές οι εγγραφές θα είναι πλήρεις και 100% ακριβείς, καθώς αυτό δεν είναι εφικτό χωρίς αξιόπιστα δίκτυα. Συνεπώς ο παροχέας υπηρεσιών δεν μπορεί να θεωρήσει τα ποσά που καταγράφονται από την Υπηρεσία Λογαριασμών ως πραγματικά χρηματικά ποσά και να τα χρησιμοποιήσει για συναλλαγές. Μπορεί όμως με βάση αυτά να κόψει αποδείξεις που θα σταλούν στους πελάτες.

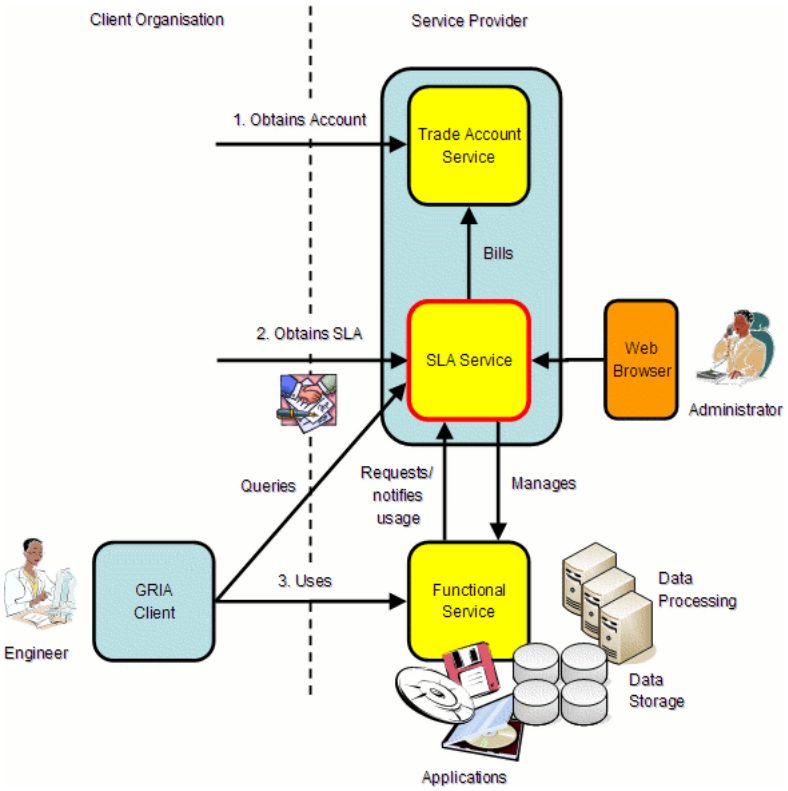
4.3.2.2 SLA Management Service

Η υπηρεσία αυτή δίνει τη δυνατότητα στο διαχειριστή υπηρεσιών να ορίσει τους διαθέσιμους πόρους (για παράδειγμα CPUs, εφαρμογές), να αναθέσει τμήματα των πόρων στους πελάτες μέσω SLAs και να χρεώσει για τη χρήση των πόρων με βάση την τιμολογιακή πολιτική όπως καθορίζεται στα SLAs . Στο σχήμα 4.2 παρουσιάζεται η Υπηρεσία Διαχείρισης SLAs .

4.3.2.3 Οι μετρικές (metrics)

Η υπηρεσία διαχείρισης SLAs του GRIA είναι σχεδιασμένη έτσι ώστε να παρουσιάζει μεγάλη ευελιξία. Ανακτά πληροφορίες σχετικά με τη χρήση των λειτουργικών υπηρεσιών, καταγράφει τη χρήση αυτή και προαιρετικά εφαρμόζει περιορισμούς ή χρεώνει για τη χρήση. Όμως διαφορετικές λειτουργικές υπηρεσίες κάνουν χρήση διαφορετικού είδους πόρων και συνεπώς πρέπει να «δηλώνουν» στην υπηρεσία διαχείρισης SLAs τη χρήση διαφορετικών μετρήσιμων ποσοτήτων. Μια υπηρεσία δεδομένων λόγου χάρη θα δηλώσει χρήση χώρου στο δίσκου, ενώ μια υπηρεσία εργασίας θα αναφέρει χρήση της κεντρικής μονάδας επεξεργασίας (CPU). Αυτές οι μετρήσιμες ποσότητες είναι γνωστές ως «μετρικές» (metrics) και αντιπροσωπεύονται από URIs. Η υπηρεσία SLA του GRIA δεν καταλαβαίνει τη σημασία αυτών των URIs, απλά καταγράφει τη χρήση τους και ενεργεί ανάλογα με το πώς έχει ρυθμιστεί.

Η χρήση των μετρικών καταγράφεται με δύο τρόπους: στιγμιαία και συγκεντρωτικά. Για κάποιες μετρικές, όπως για παράδειγμα η μεταφορά δεδομένων, η στιγμιαία μέτρηση εκφράζεται καλύτερα ως ρυθμός (ως bytes per second) και η συγκεντρωτική χρήση δεν έχει μονάδες χρόνου (bytes). Για άλλες μετρικές, όπως η CPU, η στιγμιαία μέτρηση είναι απλώς η ποσότητα που χρησιμοποιείται τη χρονική στιγμή της μέτρησης (για παράδειγμα 3 CPUs) και η συγκεντρωτική χρήση έχει και χρονική διάσταση (180 CPU.seconds). Η υπηρεσία SLA μπορεί να κάνει μετατροπές μεταξύ των δύο μορφών.



Σχήμα 4.2: Η Υπηρεσία Διαχείρισης SLAs στην αρχιτεκτονική του GRIA

4.3.2.4 SLA Templates

Ένα SLA Template (πρότυπο SLA) καθορίζει ποιες υπηρεσίες μπορεί να χρησιμοποιήσει ένας χρήστης, την ποσότητα των πόρων καθώς και το πόσο θα χρεώνεται και πόσο συχνά. Ένας πάροχος υπηρεσιών μπορεί να ορίσει και να εκδόσει πολλά διαφορετικά SLA Templates.

Ένα SLA Template μπορεί να βρίσκεται σε μια από τις εξής δύο καταστάσεις:

draft Στην κατάσταση αυτή είναι ορατό μόνο από το διαχειριστή της υπηρεσίας. Μπορούν να γίνουν αλλαγές στο template και μετά να δημοσιευτεί.

published Στην κατάσταση αυτή είναι ορατό από όσους προσδιορίζονται από τους κανόνες ελέγχου πρόσβασης (access control rules). Επίσης δεν μπορεί να γίνει καμία επεξεργασία σε ένα δημοσιευμένο template.

Και στις δύο παραπάνω καταστάσεις είναι δυνατή η αντιγραφή, η εξαγωγή και η διαγραφή, ενώ επιπλέον μπορούν να προσδιοριστούν οι κανόνες πρόσβασης.

Τα μέρη ενός SLA Template είναι τα εξής:

- Επιγραφή και Περιγραφή (Label και Description)
- Περίοδος χρέωσης (Billing Period)
- Τέλη εγγραφής, συνδρομής και Νόμισμα (Signing Fee, Subscription Fee, Currency)
  Το τέλος εγγραφής χρεώνεται στο λογαριασμό όταν συμφωνείται ένα καινούργιο SLA, ενώ το τέλος συνδρομής χρεώνεται στο τέλος κάθε περιόδου χρέωσης (επιπλέον των χρεώσεων που έγιναν λόγω της χρήσης πόρων).
- Περίοδος ισχύος (Validity Period)
- Επιτρεπόμενες Υπηρεσίες (Permitted Services)
  Αν η λίστα αυτή είναι κενή, τότε όλες οι λειτουργικές υπηρεσίες που ανήκουν στην ομάδα "sla-managed-services" μπορούν να χρησιμοποιηθούν με το SLA.
- Περιορισμοί (Constraints)
  Κάθε περιορισμός περιορίζει τη στιγμιαία ή τη συγκεντρωτική χρήση μιας μετρικής. Υπάρχουν δύο είδη περιορισμών: περιοδικός και αόριστος (indefinite). Ένας περιοδικός περιορισμός περιορίζει τη χρήση μιας μετρικής κατά τη διάρκεια μιας επαναλαμβανόμενης χρονικής περιόδου. Μπορεί να χρησιμοποιηθεί για να περιορίζει τη χρήση στο διάστημα μιας ημέρας, για παράδειγμα. Αντίθετα ένας αόριστος περιορισμός ισχύει από τη στιγμή που τίθεται σε ισχύ το SLA μέχρι την παρούσα χρονική στιγμή.
- Ιδιωτικοί Περιορισμοί (Private Constraints)
  Σε ορισμένες περιπτώσεις ένας πελάτης θέλει η συμφωνία του με τον πάροχο να γίνει με «επιχειρηματικούς όρους» και όχι με τεχνικούς. Δηλαδή ο πελάτης

θέλει να προσδιορίσει ποιο είναι το επιθυμητό αποτέλεσμα και όχι ακριβώς ποιοι πόροι πρέπει να χρησιμοποιηθούν ώστε να το επιτύχει αυτό. Το GRIA το πετυχαίνει αυτό με τη χρήση των ιδιωτικών περιορισμών.

- Όροι χρέωσης (Pricing Terms)
  Στο τέλος κάθε περιόδου χρέωσης εξετάζεται η χρήση ενός SLA και υπολογίζεται ο λογαριασμός. Οι χρεώσεις μπορεί να γίνουν για τη συγκεντρωτική χρήση μιας μετρικής ή για την αύξηση της μέτρησης μιας μετρικής κατά τη διάρκεια της περιόδου χρέωσης.

4.3.2.5 SLAs στο GRIA

Ο πάροχος υπηρεσιών δημοσιεύει ορισμένα SLA Templates. Ο πελάτης επιλέγει ένα από τα πρότυπα αυτά SLAs, το οποίο κρίνει ότι ικανοποιεί τις ανάγκες του, και είτε το υπογράφει αμέσως είτε εισέρχεται σε μια φάση διαπραγματεύσεων με τον πάροχο υπηρεσιών προκειμένου να τροποποιηθούν κάποιοι όροι του συμβολαίου, έτσι ώστε να ανταποκρίνεται καλύτερα στις απαιτήσεις του πελάτη. Αν επιτευχθεί συμφωνία μεταξύ των δύο μερών, υπογράφεται το SLA.

Στο GRIA ένα SLA μπορεί να βρίσκεται σε μια από τις ακόλουθες καταστάσεις:

active Σε αυτή την κατάσταση η υπηρεσία SLAs επεξεργάζεται αιτήσεις από τις λειτουργικές υπηρεσίες και αν δεν παραβιάζονται οι όροι του SLA επιτρέπει την εκκίνηση νέων δραστηριοτήτων.

suspended Όταν ένα SLA βρίσκεται σε αυτή την κατάσταση δεν μπορούν να ξεκινήσουν νέες δραστηριότητες που θα έχουν ως αποτέλεσμα τη χρήση πόρων.

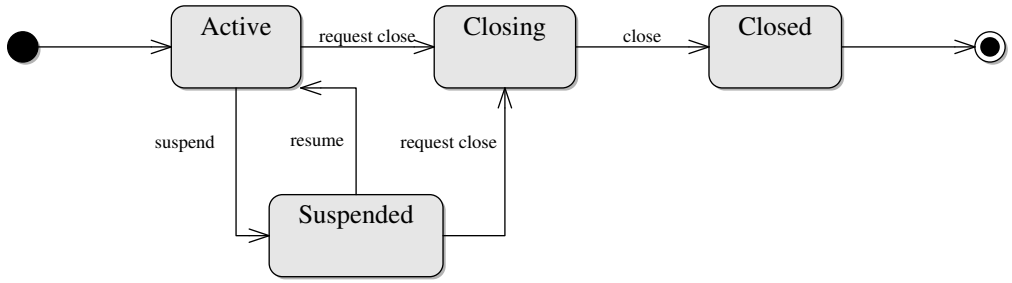
closing Όταν ένα SLA μπει σε αυτή την κατάσταση, όλες οι τρέχουσες δραστηριότητες ειδοποιούνται να καταστραφούν. Το SLA παραμένει σε αυτή την κατάσταση μέχρι να ολοκληρωθούν όλες οι δραστηριότητες και να σταλεί ο τελικός λογαριασμός. Δεν μπορούν να ξεκινήσουν νέες δραστηριότητες στην κατάσταση αυτή.

closed Σε αυτή την κατάσταση δεν μπορούν να ξεκινήσουν νέες δραστηριότητες.

Οι μεταβάσεις μεταξύ των καταστάσεων φαίνονται στο σχήμα 4.3.

4.3.3 Client package

Η εφαρμογή-πελάτη του GRIA δίνει τη δυνατότητα στο χρήστη να χρησιμοποιήσει διάφορες υπηρεσίες του GRIA όπως αποθήκευση δεδομένων, επεξεργασία κατά δέσμες (batch processing) και συστήματα βάσεων δεδομένων. Επίσης επιτρέπει την χρήση των υπηρεσιών διαχείρισης του GRIA που είναι απαραίτητες ώστε να γίνεται καταγραφή και χρέωση των άλλων υπηρεσιών.



Σχήμα 4.3: Μεταβάσεις μεταξύ των καταστάσεων ενός SLA

4.3.4 Client Management

Το πακέτο αυτό χρησιμοποιείται από έναν οργανισμό-πελάτη για δική του εσωτερική χρήση. Παρέχει μια υπηρεσία συνδρομών (membership service), μια υπηρεσία ιδιωτικών λογαριασμών (private account service) και μια υπηρεσία Registry.

Το Membership Service χρησιμοποιείται για να διαχειρίζεται ομάδες χρηστών. Οι διαχειριστές μπορούν να δημιουργήσουν νέες ομάδες και να καθορίσουν σε ποια ομάδα ανήκει ένας χρήστης.

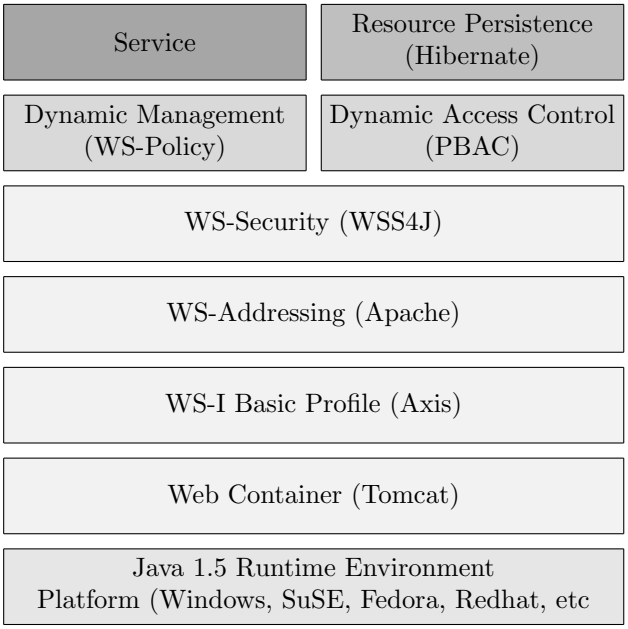
Το Private Account Service και το Registry Service παρέχουν προς το παρόν παρόμοια λειτουργικότητα. Και τα δύο επιτρέπουν τη δημιουργία registries και την πρόσθεση πόρων όπως trade accounts, SLAs, data stagers. Το Private Account Service περιλαμβάνει επιπλέον συναρτήσεις εξειδικευμένες στη διαχείριση λογαριασμών και SLAs , ενώ το Registry Service μπορεί να χρησιμοποιήσει μια βάση δεδομένων eXist ως πίσω άκρο (back-end).

4.3.5 OGSA-DAI application service

Το πακέτο αυτό έχει υλοποιηθεί με βάση το OGSA-DAI WS-I 2.2 [15]. Επιτρέπει στους πελάτες να έχουν πρόσβαση σε πόρους δεδομένων μέσω μιας υπηρεσίας GRIA, χρησιμοποιώντας τις βιβλιοθήκες του OGSA-DAI. Η υπηρεσία αυτή μπορεί να χρησιμοποιηθεί μόνη της για να παρέχει ελεύθερη (unmanaged) πρόσβαση στις βάσεις δεδομένων ή μαζί με το το πακέτο GRIA Service Provider Management για να αξιοποιήσει την υπηρεσία χρέωσης και SLAs που προσφέρει το πακέτο αυτό.

4.3.6 Service Developers Toolkit

Το πακέτο αυτό παρέχει τεκμηρίωση (documentation) και δείγματα κώδικα για τη δημιουργία Web services που λειτουργούν με τις υπηρεσίες και τους πελάτες του GRIA.



Σχήμα 4.4: Η υποδομή των Web services στο GRIA

4.3.7 Υποδομή των Web services

Στην έκδοση 5.1 του GRIA υλοποιείται η υποδομή των Web services που φαίνεται στο σχήμα 4.4

Η στοίβα τεχνολογιών πυρήνα (core technology stack) που χρησιμοποιεί το GRIA είναι η εξής:

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

4.3.8 Ασφάλεια

Το GRIA χρησιμοποιεί τις ακόλουθες τεχνολογίες ασφαλείας:

Ασφάλεια επιπέδου μεταφοράς (transport layer security) Η διασφάλιση της εμπιστευτικότητας των μηνυμάτων (message confidentiality) στο GRIA

επιτυγχάνεται με τη χρήση του πρωτοκόλλου μεταφοράς HTTPS. Επίσης υποστηρίζεται πιστοποίηση (authentication) τόσο του χρήστη όσο και της υπηρεσίας.

Ασφάλεια Επιπέδου Μηνύματος (message layer security) Παρέχεται με το WS-Security, το οποίο αποτελεί ένα σύνολο από επεκτάσεις SOAP που παρέχουν ακεραιότητα, εμπιστευτικότητα και πιστοποίηση επιπέδου μηνύματος.

SAML tokens Πρόκειται για ένα XML standard για την ανταλλαγή δεδομένων πιστοποίησης και εξουσιοδότησης μεταξύ ενός παρόχου ταυτότητας (identity provider) και ενός παρόχου υπηρεσιών (service provider) [16].

X.509 Πιστοποιητικά Πρόκειται για ένα πρότυπο της ITU-T για υποδομή δημόσιου κλειδιού (Public Key Infrastructure - PKI).

Process Based Acced Control (PBAC) Είναι ένας μηχανισμός δυναμικού ελέγχου πρόσβασης. Χρησιμοποιείται συχνά για έλεγχο πρόσβασης σε Web services. Τα Web services δημιουργούν πόρους και επιτρέπουν στους χρήστες να εκτελέσουν κάποιες λειτουργίες με αυτούς. Όταν ένας χρήστης προσπαθεί να προσπελάσει έναν πόρο καλώντας μια λειτουργία μέσω ενός Web service, το σύστημα PBAC ελέγχει αν ο χρήστης είναι εξουσιοδοτημένος να χρησιμοποιήσει τη ζητούμενη λειτουργία σε αυτόν τον πόρο.

Κεφάλαιο 5

Εγκατάσταση και Ρύθμιση του απαιτούμενου Λογισμικού

Περιεχόμενα

Στα πλαίσια αυτής της εργασίας χρησιμοποιήθηκε η έκδοση 5.1 του GRIA και πιο συγκεκριμένα τα πακέτα Basic Application Services, Service Provider Management και Client. Στο κεφάλαιο αυτό περιγράφεται αναλυτικά ο τρόπος εγκατάστασης και οι ρυθμίσεις που χρησιμοποιήθηκαν για κάθε ένα από αυτά. Επίσης ήταν απαραίτητη η τροποποίηση ορισμένων πακέτων του συστήματος, για αυτό περιγράφεται και η διαδικασία δημιουργίας των πακέτων από τον πηγαίο κώδικα

5.1 Προαπαιτούμενα

Για την εγκατάσταση των πακέτων του GRIA είναι απαραίτητα τα εξής:

- Apache Tomcat 5.5.x
- Java SDK 1.5.x
- Ένα από τα ακόλουθα λειτουργικά συστήματα: Windows XP, Windows 2003 Server, SuSE 9.3–10.2, Fedora Core 3, 4, 5. Επίσης είναι δυνατή η εγκατάσταση με μικρές τροποποιήσεις σε ένα από τα ακόλουθα λειτουργικά συστήματα: Red Hat, Debian, Ubuntu, Mandriva.
- Ένας φυλλομετρητής Ιστού (Web browser)
- Ένα πρόγραμμα για αποσυμπίεση συμπιεσμένων αρχείων

Στα πλαίσια της εργασίας αυτής χρησιμοποιήθηκαν οι εξής εκδόσεις των παραπάνω λογισμικών:

- Apache Tomcat 6.0.13
- Java SDK 1.5.0_12
- Το λειτουργικό σύστημα Windows Vista Business
- Mozilla Firefox 2.0.0.12
- Το πρόγραμμα WinRAR

Επιπλέον για τη δημιουργία μιας Αρχής Πιστοποίησης μπορεί να χρησιμοποιηθεί το πρόγραμμα XCA [17]. Για την παρούσα εργασία χρησιμοποιήθηκε η έκδοση 0.6.3 του προγράμματος.

Για τη δημιουργία και επεξεργασία των keystores χρησιμοποιήθηκε το πρόγραμμα KeyTool GUI και πιο συγκεκριμένα η έκδοση 1.7 του προγράμματος.

5.2 Εγκατάσταση του πακέτου Basic Application Services

Εκτός από τα προαπαιτούμενα της ενότητας 5.1, στην περίπτωση του πακέτου αυτού απαιτούνται επιπλεόν τα εξής:

- Μια υλοποίηση της γλώσσας προγραμματισμού Perl [18]. Επιλέχθηκε η υλοποίηση ActiveState Perl 5.8.8.822 [19].
- Η εφαρμογή ImageMagick που χρησιμοποιείται ως test application [20]. Χρησιμοποιήθηκε η έκδοση 6.3.6-Q8.

Για την εγκατάσταση του πακέτου αυτού πρέπει να γίνει deploy στον Tomcat το αρχείο gria-basic-app-services.war. Η παραμετροποίηση του πακέτου γίνεται μέσω της κεντρικής ιστοσελίδας της εφαρμογής που είναι διαθέσιμη στη διεύθυνση http://localhost:8080/gria-basic-app-services/. Την πρώτη φορά που ανοίγεται η ιστοσελίδα αυτή, εμφανίζεται ένα μήνυμα που ενημερώνει το χρήστη ότι το πακέτο δεν έχει ακόμα ρυθμιστεί καθώς και σχετικές οδηγίες. Συνοπτικά η διαδικασία που ακολουθείται είναι η εξής:

1. Αρχικά πρέπει να δημιουργηθεί ένας χρήστης στον Tomcat ο οποίος θα έχει το ρόλο gria_basic_app_services_admin. Για το σκοπό αυτό πρέπει να προστεθεί στο αρχείο $CATALINA_HOME/conf/tomcat-users.xml μια γραμμή κώδικα που ορίζει το νέο ρόλο και τουλάχιστον ένας χρήστης με το ρόλο αυτό. Για παράδειγμα:

   ```
   <?xml version='1.0' encoding='utf-8'?>
   <tomcat-users>
     <role rolename="gria_basic_app_services_admin"/>
     <user username="EnterAUserName"
           password="EnterThePassword"
           roles="gria_basic_app_services_admin"/>
   </tomcat-users>
   ```

2. Επιλογή του φακέλου στον οποίο θα αποθηκευτούν οι ρυθμίσεις για το πακέτο αυτό.

3. Εγκατάσταση και παραμετροποίηση ενός keystore το οποίο περιέχει το ιδιωτικό κλειδί της υπηρεσίας. Με αυτόν τον τρόπο, ο πελάτης μπορεί να επαληθεύσει ότι όντως χρησιμοποιεί την υπηρεσία που θέλει.

   Μέσω της ιστοσελίδας διαχείρισης του πακέτου μπορεί να δημιουργηθεί ένα keystore, δηλαδή ένας χώρος στον οποίο αποθηκεύονται τα κλειδιά για τις διάφορες υπηρεσίες. Στη συνέχεια το keystore που δημιουργήθηκε ανοίγεται με το πρόγραμμα KeyTool GUI προκειμένου για να παραχθεί μια αίτηση υπογραφής πιστοποιητικού (Certificate Signing Request). Πρόκειται για ένα αρχείο με

επέκταση .csr το οποίο πρέπει να υπογραφεί από μια Αρχή Πιστοποίησης (Certificate Authority) την οποία ο πελάτης εμπιστεύεται. Μια Αρχή Πιστοποίησης μπορεί να δημιουργηθεί με χρήση του προγράμματος XCA. Η αίτηση πιστοποίησης αποστέλλεται στην Αρχή Πιστοποίησης, η οποία στέλνει πίσω στον πελάτη που έκανε την αίτηση δύο αρχεία με επέκταση .crt: το πιστοποιητικό της ίδιας της Αρχής Πιστοποίησης και το νέο υπογεγραμμένο πιστοποιητικό. Τέλος (με τη βοήθεια του προγράμματος KeyTool GUI εισάγεται το υπογεγραμμένο πιστοποιητικό στο keystore και αυτό μπορεί να γίνει upload μέσω της ιστοσελίδας διαχείρισης.

4. Παραμετροποίηση του Tomcat έτσι ώστε να υποστηρίζει ασφαλείς συνδέσεις (HTTPS Connections). Για το σκοπό αυτό πρέπει εισαχθεί το ακόλουθο τμήμα κώδικα στο αρχείο $CATALINA_HOME/conf/server.xml, ώστε να ενεργοποιηθεί το SSL στον Tomcat.

```
<Connector
  port="8443" protocol="HTTP/1.1" SSLEnabled="true"
  keystoreFile=
    "C:\gria\basic-app-services\service-keystore.ks"
  keystorePass="EnterThePassword"
  keystoreType="JKS"
  minProcessors="5" maxProcessors="75"
  enableLookups="true" disableUploadTimeout="true"
  acceptCount="100" debug="0" scheme="https" secure="true"
  maxThreads="150"
  clientAuth="false" sslProtocol="TLS"/>
```

5. Καθορισμός της τοποθεσίας στην οποία θα αποθηκευτεί η βάση δεδομένων.

6. Καθορισμός του endpoint address για την υπηρεσία, δηλαδή το url με το οποίο θα είναι προσβάσιμη η υπηρεσία από το φυλλομετρητή.

Στη συνέχεια δημιουργήθηκε μια υπηρεσία Δεδομένων (Data Service) και μια υπηρεσία Εργασιών (Job Service).

Πρέπει να σημειωθεί ότι στην περίπτωση της υπηρεσίας Εργασιών τα platform scripts που είναι διαθέσιμα ως παραδείγματα (στο συμπιεσμένο αρχείο platform-scripts.tgz), δε λειτουργούν στο λειτουργικό σύστημα Windows Vista, οπότε έπρεπε να γίνουν κάποιες αλλαγές στα εξής αρχεία που περιέχονται στο φάκελο platform-scripts\rm_local:

- `getJobStatus.pl`
- `killJob.pl`
- `startJob.pl`

Πίνακας 5.1: Τιμές ID, Major, Minor Number για Windows

OS	ID	MAJOR	MINOR
Win32s	0	-	-
Windows 95	1	4	0
Windows 98	1	4	10
Windows Me	1	4	90
Windows NT 3.51	2	3	51
Windows NT 4	2	4	0
Windows 2000	2	5	0
Windows XP	2	5	1
Windows Server 2003	2	5	2
Windows Vista	2	6	0

Στα scripts αυτά υπάρχει η υπορουτίνα `checkOS` η οποία ελέγχει την έκδοση του λειτουργικού συστήματος. Πιο συγκεκριμένα στην υπορουτίνα αυτή καλείται η συνάρτηση `Win32::GetOSVersion()` η οποία επιστρέφει μία λίστα με τα εξής στοιχεία: (`STRING, MAJOR, MINOR, BUILD, ID`). Δηλαδή το `GetOSVersion[1]` αντιστοιχεί στο major version number του λειτουργικού συστήματος, το `GetOSVersion[2]` στο minor version number. Το `ID` είναι ένα ψηφίο που προσδιορίζει το λειτουργικό σύστημα. Οι τιμές που επιστρέφει η συνάρτηση για διάφορες εκδόσεις λειτουργικών συστημάτων Windows δίνονται στον πίνακα 5.1 [21].

Η συνθήκη που ελεγχόταν αρχικά ήταν η εξής:

```
if (( $os_version[1] >= 5) and ($os_version[2] > 0)){
   print "this platform can handle jobs\n";
} else {
   die " error, this platform cannot handle jobs\n";
}
```

Στην παραπάνω περίπτωση γίνονται δεκτά μόνο τα λειτουργικά Windows XP και Windows Server 2003, οπότε έγινε η εξής τροποποίηση ώστε να λαμβάνεται υπόψη και το λειτουργικό Windows Vista:

```
if ((( $os_version[1] >= 5) and ($os_version[2] > 0))
       or (($os_version[1]==6) and ($os_version[2]==0))){
   print "this platform can handle jobs\n";
} else {
   die " error, this platform cannot handle jobs\n";
}
```

5.3 Εγκατάσταση του πακέτου Service Provider Management

Η διαδικασία εγκατάστασης του πακέτου αυτού είναι παρόμοια με αυτή του πακέτου Basic Application Services. Στην περίπτωση αυτή δεν είναι απαραίτητη η υλοποίηση της Perl, ούτε και το πρόγραμμα ImageMagick.

Επίσης στα βήματα 1 και 4 ο κώδικας που πρέπει να εισαχθεί στα αντίστοιχα αρχεία διαφέρει ως προς το όνομα της υπηρεσίας. Δηλαδή το basic-app-services όπου εμφανίζεται πρέπει να αντικατασταθεί με το service-provider-mgt.

5.4 Εφαρμογή Client

Η εφαρμογή αυτή, απλά απαιτεί την αποσυμπίεση του αρχείου gria-client-cli-5.1.zip σε ένα νέο φάκελο. Στη συνέχεια απαιτείται η δημιουργία ενός ιδιωτικού κλειδιού και ενός πιστοποιητικού που θα υπογράψει μια Αρχή Πιστοποίησης την οποία θα εμπιστεύεται ο παροχέας υπηρεσιών. Η διαδικασία είναι παρόμοια με αυτή που ακολουθήθηκε στην εγκατάσταση του πακέτου basic application services στο βήμα 3. Για το άνοιγμα της εφαρμογής πελάτη, εκτελείται από γραμμή εντολών η εντολή `gridcli` μέσα από το φάκελο της εγκατάστασης.

5.5 Πηγαίος κώδικας του GRIA

Ο κώδικας βρίσκεται στο αρχείο gria-5.1-src.zip το οποίο και πρέπει να αποσυμπιεστεί. Στη συνέχεια, προκειμένου να γίνει build το project, απαιτείται το λογισμικό Apache Maven 2 [22]. Στην εργασία αυτή χρησιμοποιήθηκε η έκδοση 2.0.6 του προγράμματος.

Για να δημιουργηθούν τα πακέτα του GRIA από τον πηγαίο κώδικα, πρέπει να εκτελεστούν μέσα από το φάκελο στον οποίο αποσυμπιέστηκε ο κώδικας, οι εντολές:

```
cd base
mvn install
cd ../gria
mvn install
```

Οι παραπάνω εντολές δημιουργούν όλα τα πακέτα του GRIA, δηλαδή τα αντίστοιχα αρχεία .war. Αν ενδιαφέρει μόνο ένα πακέτο, τότε μπορεί η δεύτερη εντολή mvn install να εκτελεστεί μέσα από το φάκελο του πακέτου αυτού και όχι από το φάκελο gria.

Στη συγκεκριμένη περίπτωση, χρειάστηκε να κάνουμε αλλαγές μόνο στο πακέτο Service Provider Management, οπότε κάθε φορά κάναμε build μόνο το πακέτο αυτό. Πρέπει ακόμη να σημειωθεί, ότι επειδή κατά τη διαδικασία δημιουργίας του πακέτου αυτού από το maven κάποια tests δεν ήταν επιτυχή, χρησιμοποιούσαμε την εντολή `mvn -Dmaven.test.skip=true install` προκειμένου να γίνει δημιουργηθεί επιτυχώς.

Για την προβολή και την τροποποίηση του κώδικα χρησιμοποιήθηκε το πλαίσιο εργασίας Eclipse v3.2.2 [23] στο οποίο είχε προστεθεί το Maven Eclipse Plugin [24] προκειμένου να μπορούν τα projects του maven να είναι προσβάσιμα από το Eclipse.

Κεφάλαιο 6

Μελέτη και Σχεδιασμός του Συστήματος

Περιεχόμενα

Στο κεφάλαιο αυτό περιγράφονται οι λειτουργικές και μη λειτουργικές απαιτήσεις του συστήματος καθώς και οι κύριες σχεδιαστικές αποφάσεις που ελήφθησαν.

6.1 Ο ρόλος των πολιτικών

Η τρέχουσα υλοποίηση του GRIA δεν επιτρέπει την προσαρμογή της συμπεριφοράς του παροχέα απέναντι στους πελάτες. Σκοπός της εργασίας αυτής είναι η επέκταση του GRIA ώστε να υποστηρίζει διαχείριση των SLAs με χρήση πολιτικών (policies). Δηλαδή, ο παροχέας υπηρεσιών θα μπορεί να εφαρμόζει κάποια πολιτική στους πελάτες μιας συγκεκριμένης κατηγορίας. Για παράδειγμα, αν ένας πελάτης θεωρείται «καλός», με βάση κάποια κριτήρια που καθορίζονται από τον παροχέα υπηρεσιών, τότε θα μπορεί να έχει ορισμένα προνόμια, όπως μικρότερη χρέωση για κάποιες υπηρεσίες.

Οι πολιτικές καθορίζονται αυστηρά και μόνο από τον παροχέα υπηρεσιών. Ο πελάτης δεν έχει καμία πρόσβαση σε αυτές ώστε να τις τροποποιήσει ή να τις διαπραγματευτεί με τον παροχέα υπηρεσιών, όπως συμβαίνει στην περίπτωση των SLAs. Για την ακρίβεια, ο πελάτης δε γνωρίζει αν εφαρμόζεται κάποια πολιτική.

Πρέπει να σημειωθεί ότι οι πολιτικές δεν πρέπει σε καμία απολύτως περίπτωση να παραβιάζουν το SLA που έχει συμφωνηθεί με τον πελάτη με τρόπο που δεν τον ωφελεί. Δεν μπορεί για παράδειγμα ο παροχέας να εφαρμόσει μια πολιτική που θα χρεώνει τον πελάτη περισσότερο από όσο έχει συμφωνηθεί στο SLA. Οι πολιτικές χρησιμοποιούνται μόνο προς όφελος του πελάτη.

Βέβαια από τη διαδικασία αυτή ωφελείται και ο πάροχος υπηρεσιών, καθώς μπορεί να χρησιμοποιήσει τις πολιτικές για διαφημιστικούς και προωθητικούς σκοπούς. Επιπλέον μπορεί να δώσει ένα κίνητρο στους χρήστες να είναι «καλοί» πελάτες.

6.2 Η μορφή των πολιτικών

Λαμβάνοντας υπόψη τα παραπάνω καθώς και τη μορφή των SLAs του GRIA, καταλήξαμε ότι μια πολιτική θα πρέπει να προβλέπει τα εξής:

Αντιμετώπιση μιας παραβίασης του SLA. Στην έκδοση 5.1 του GRIA αν παραβιαστεί κάποιος όρος του SLA διακόπτεται αμέσως κατάλληλος αριθμός των εργασιών που χρησιμοποιούν τη μετρική του περιορισμού που παραβιάστηκε, ώστε η χρήση να επανέλθει μέσα στο όρια που προσδιορίζονται από το συμβόλαιο. Θεωρήσαμε ότι θα έπρεπε να υπάρχει δυνατότητα διαφορετικής αντίδρασης. Για παράδειγμα, εφόσον ο πάροχος έχει διαθέσιμους πόρους, ο χρήστης να ενημερώνεται ότι παραβίασε κάποιον όρο του συμβολαίου του, αλλά να του επιτρέπεται να συνεχίσει να χρησιμοποιεί τους αντίστοιχους πόρους.

Έκπτωση βάσει των παραβιάσεων του SLA. Το πλήθος των παραβιάσεων αποτελεί ένα μέτρο του πόσο καλός είναι ένας πελάτης. Αν δεν παραβιάζει συχνά τους όρους του SLA του, ο πάροχος μπορεί να τον ανταμοίψει με μειώσεις στους λογαριασμούς που του αποστέλλονται.

Εφαρμογή έκπτωσης ανάλογα με τη χρήση των υπηρεσιών. Εάν ένας χρήστης έχει χρησιμοποιήσει αρκετά το τελευταίο διάστημα τις υπηρεσίες που προσφέρει ένας πάροχος (οπότε έχει αρκετά έσοδα από τον πελάτη αυτό), τότε ο πάροχος μπορεί να προσφέρει τις υπηρεσίες του σε καλύτερη τιμή για το συγκεκριμένο πελάτη.

6.3 Πού εφαρμόζονται οι πολιτικές

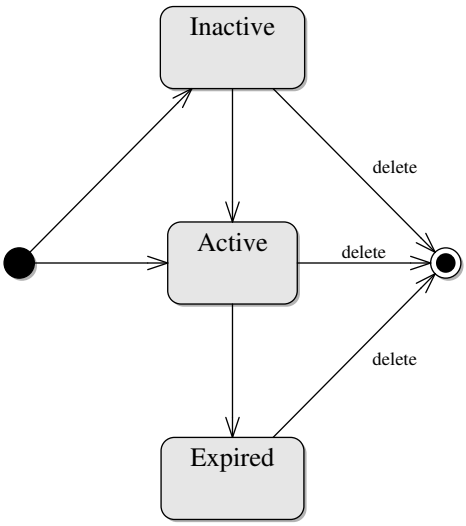
Μια πολιτική προφανώς εφαρμόζεται τελικά σε ένα SLA. Όμως ένας πάροχος υπηρεσιών μπορεί να διαχειρίζεται πολύ μεγάλο αριθμό SLAs οπότε η σύνδεση του κάθε ενός από αυτά ξεχωριστά με μία πολιτική, θα ήταν μια χρονοβόρα διαδικασία. Μια πιο πρακτική σκέψη, που ανταποκρίνεται στις απαιτήσεις μιας επιχείρησης, είναι η σύνδεση μιας πολιτικής με ένα SLA Template. Στην περίπτωση αυτή, η πολιτική εφαρμόζεται σε κάθε SLA που προέρχεται από αυτό το template. Οπότε η σύνδεση μιας πολιτικής με ένα SLA και η εφαρμογή της σε αυτό προσδιορίζεται μέσω της σύνδεσης της πολιτικής με το αντίστοιχο SLA Template. Στη συνέχεια, ο όρος «σύνδεση μιας πολιτικής με ένα SLA» αναφέρεται στην σύνδεση της πολιτικής με το SLA Template από το οποίο δημιουργήθηκε το SLA.

Εκτός από τη πρακτική ευκολία που προσφέρει η σύνδεση πολιτικών με τα SLA Templates είναι συνεπής και με τη φιλοσοφία των πολιτικών. Πιο συγκεκριμένα, το γεγονός ότι ο κάθε πάροχος έχει τη δυνατότητα να κάνει διαθέσιμα διαφορετικά SLA Templates ανά πελάτη εισάγει ήδη μια μορφή κατηγοριοποίησης μεταξύ των πελατών. Έχει συνεπώς νόημα να διατηρήσουμε την κατηγοριοποίηση αυτή και στην εφαρμογή πολιτικών.

Μια πολιτική μπορεί να συνδεθεί με ένα SLA Template ανεξάρτητα με την κατάσταση στην οποία βρίσκεται αυτό (draft ή published). Επίσης πρέπει να σημειωθεί ότι η σχέση μεταξύ πολιτικών και SLA Templates είναι ένα προς πολλά. Δηλαδή κάθε χρονική στιγμή μια πολιτική μπορεί να συνδέεται με ένα ή περισσότερα SLA Templates ενώ ένα SLA Template μπορεί να είναι συσχετισμένο με μία μόνο πολιτική. Κατ' επέκταση, αφού ένα SLA προέρχεται από ένα μόνο SLA Template, κάθε στιγμή ένα SLA μπορεί να είναι συσχετισμένο με μία μόνο πολιτική.

6.4 Δυνατές Καταστάσεις μιας πολιτικής

Η κατάσταση μια πολιτικής καθορίζεται σε σχέση με το χρόνο έναρξης και το χρόνο λήξης της πολιτικής. Κάθε χρονική στιγμή που προηγείται του χρόνου έναρξης, η πολιτική είναι ανενεργή (inactive), δηλαδή δεν έχει καμία επίδραση στα SLAs με τα οποία έχει συσχετιστεί, όμως κάποια στιγμή στο μέλλον πρόκειται να ενεργοποιηθεί. Κάθε χρονική στιγμή που βρίσκεται μεταξύ του χρόνου έναρξης και του χρόνου λήξης, η πολιτική είναι ενεργή (active), δηλαδή οι όροι της εφαρμόζονται στα SLAs με τα οποία είναι συνδεδεμένη. Ο χρόνος λήξης καθορίζει τη μετάβαση από την ενεργή στη



Σχήμα 6.1: Διάγραμμα καταστάσεων για τις πολιτικές

ληγμένη (expired) κατάσταση. Στην κατάσταση αυτή, η πολιτική δεν έχει πάλι καμία επίδραση στα SLAs και δεν πρόκειται να ενεργοποιηθεί στο μέλλον.

Οι καταστάσεις μιας πολιτικής καθώς και οι δυνατές μεταβάσεις μεταξύ των καταστάσεων φαίνονται αναλυτικά στο UML διάγραμμα καταστάσεων του σχήματος 6.1. Όπως φαίνεται και στο σχήμα, μια πολιτική όταν δημιουργείται μπορεί να βρίσκεται στην ανενεργή ή στην ενεργή κατάσταση. Αυτό είναι λογικό, καθώς δεν έχει νόημα να δημιουργηθεί μια πολιτική με χρόνο λήξης στο παρελθόν. Επίσης μια πολιτική μπορεί σε οποιαδήποτε κατάσταση και αν βρίσκεται να διαγραφεί και να μεταβεί στην τελική κατάσταση.

Κεφάλαιο 7

Υλοποίηση του Συστήματος

Περιεχόμενα

Στο κεφάλαιο αυτό παρουσιάζονται λεπτομέρειες για την υλοποίηση του συστήματος. Πιο συγκεκριμένα, στα πλαίσια της εργασίας αυτής, προστέθηκε κώδικας στο πακέτο Service Provider Management ώστε να δημιουργηθεί η υποδομή για τη χρήση πολιτικών. Επίσης τροποποιήθηκαν τμήματα κώδικα, ώστε να μπορεί μια πολιτική να συνδέεται με ένα SLA Template και να εφαρμόζεται σε όλα τα SLAs που δημιουργήθηκαν από το template αυτό.

7.1 Δομή ενός έγκυρου αρχείου πολιτικής

Οι πολιτικές είναι αρχεία xml που πρέπει να έχουν τη δομή που παρουσιάζεται στη συνέχεια.

- Το root element πρέπει να είναι `<policyTemplate>`.
- Πρέπει να υπάρχει μια ετικέτα (`<label>`) για την πολιτική, καθώς επίσης και μια περιγραφή (`<description>`).
- Ο χρόνος έναρξης και λήξης της πολιτικής (`<startTime>` και `<endTime>` αντίστοιχα) καθορίζονται εισάγοντας το έτος, το μήνα, την ημέρα, την ώρα και τα λεπτά. Σημειώνεται ότι ο μήνας παίρνει ακέραιες τιμές στο διάστημα $[0, 11]$.
- Για να καθοριστεί η αντίδραση του παρόχου σε μια παραβίαση του SLA υπάρχει μια λίστα από παραβιάσεις (`<violations>`). Η λίστα αυτή μπορεί να είναι κενή ή να περιλαμβάνει μία ή περισσότερες παραβιάσεις.
- Κάθε παραβίαση περιγράφεται από ένα στοιχείο (`<violation>`) το οποίο περιλαμβάνει τα εξής:
  - Μία μετρική (`<metric>`). Χρησιμοποιήθηκε η ίδια ακριβώς μορφή για τις μετρικές με τα SLA Templates του GRIA.
  - Έναν τύπος περιορισμού (`<type>`). Λαμβάνει την τιμή `INSTANTANEOUS` ή `CUMULATIVE` και αναφέρεται στον τύπο του περιορισμού (constraint) του SLA Template.
  - Μία ενέργεια (`<action>`) η οποία (τουλάχιστον προς το παρόν) μπορεί να λαμβάνει μία από τις τιμές `kill` και `notify`. Προσδιορίζει την ενέργεια που θα γίνει αν παραβιαστεί ένας περιορισμός με τη μετρική και τον τύπο που προσδιορίστηκαν παραπάνω.

  Κάθε στοιχείο `<violation>` της πολιτικής αναφέρεται ουσιαστικά σε ένα στοιχείο `<constraint>` των SLAs. Δηλαδή ένα στοιχείο `<violation>` αντιστοιχεί σε παραβίαση ενός περιορισμού του SLA με την ίδια μετρική και τον ίδιο τύπο. Κάθε στοιχείο `<violation>` πρέπει να είναι μοναδικό, δηλαδή να μην προσδιορίζονται διαφορετικές ενέργειες για παραβίαση του ίδιου περιορισμού. Αυτό σημαίνει ότι η μοναδικότητα μιας παραβίασης είναι συνώνυμη με τη μοναδικότητα του ζεύγους μετρικής - τύπου.

- Ακολουθεί ένα στοιχείο `<history>` στο οποίο προσδιορίζονται διάφορα στοιχεία του παρελθόντος του χρήστη, τα οποία θα ληφθούν υπόψη για την εφαρμογή εκπτώσεων. Περιλαμβάνει τα εξής:

  - Ένα στοιχείο `<totalViolations>` που προσδιορίζει τα όρια μέσα στο οποία πρέπει να βρίσκεται ο αριθμός των παραβιάσεων που έχει κάνει ο χρήστης, προκειμένου να εφαρμοστεί κάποια έκπτωση. Οι παραβιάσεις αυτές αναφέρονται στις πιο πρόσφατες περιόδους χρέωσης. Ο αριθμός των περιόδων αυτών καθορίζεται από την ιδιότητα (attribute) `numOfBillingPeriods`. Το στοιχείο `<totalViolations>` περιλαμβάνει μια λίστα (πιθανώς κενή) από διαστήματα (Ranges), τα οποία ορίζονται παρακάτω.
  - Ένα στοιχείο `<usage>`που προσδιορίζει τα όρια μέσα στα οποία πρέπει να βρίσκεται η χρήση που έχει κάνει ο πελάτης, προκειμένου να εφαρμοστεί κάποια έκπτωση. Το πεδίο αυτό αναφέρεται ουσιαστικά στο ποσό που πλήρωσε ο χρήστης στις πιο πρόσφατες περιόδους χρέωσης. Ο αριθμός των περιόδων αυτών καθορίζεται από την ιδιότητα (attribute) `<numOfBillingPeriods>`. Το στοιχείο `<usage>` περιλαμβάνει μια λίστα (πιθανώς κενή) από διαστήματα (Ranges).

- Ένα διάστημα `<range>` αντιστοιχεί σε ένα διάστημα αριθμών. Η δομή είναι η εξής:

  - Το κάτω άκρο προσδιορίζεται από το `<lowerBound>`, το οποίο πρέπει να είναι ένας μη αρνητικός αριθμός. Το διάστημα είναι κλειστό στο άκρο αυτό.
  - Το άνω άκρο προσδιορίζεται από το `<upperBound>`, το οποίο προφανώς πρέπει να είναι μεγαλύτερο ή ίσο του κάτω άκρου. Το διάστημα είναι ανοικτό στο άκρο αυτό. Επίσης αν το άνω άκρο είναι -1 τότε θεωρείται ότι ισοδυναμεί με το ∞.
  - Αν η μετρούμενη ποσότητα είναι μέσα στο διάστημα που προσδιορίζεται από τα παραπάνω πεδία, τότε εφαρμόζεται μια έκπτωση που ορίζεται από το `<discount>`. Το στοιχείο αυτό εκφράζει ποσοστό επί τοις εκατό (%), οπότε προφανώς παίρνει τιμές από 0 έως 100.

  Δηλαδή το στοιχείο

```
<totalViolations numOfBillingPeriods="3">
   <range>
      <lowerBound>0</lowerBound>
      <upperBound>5</upperBound>
      <discount>10</discount>
   </range>
</totalViolations>
```

προσδιορίζει ότι αν το πλήθος των παραβιάσεων του ισχύοντος SLA τις τελευταίες 3 περιόδους χρέωσης ανήκει στο διάστημα $[0, 5)$, τότε θα εφαρμοστεί έκπτωση 10%. Οι περίοδοι χρέωσης αρχίζουν να μετρώνται από τη στιγμή που εφαρμόζεται η πολιτική στο SLA. Μετά την παροδο των τριών περιόδων χρέωσης, ο μετρητής των παραβιάσεων μηδενίζεται και η διαδικασία επαναλαμβάνεται για τις επόμενες τρεις περιόδους χρέωσης. Δηλαδή στο συγκεκριμένο παράδειγμα η έκπτωση εφαρμόζεται (αν οι παραβιάσεις του χρήστη είναι εντός των ορίων που προσδιορίζονται) κάθε τρεις περιόδους χρέωσης.

Παρακάτω δίνεται ένα παράδειγμα ενός έγκυρου αρχείου πολιτικής:

```
<?xml version="1.0" encoding="UTF-8"?>
<policyTemplate>
  <label>The label of the policy</label>
  <description>The description of the policy</description>

  <startTime>
    <year>2006</year>
    <month>11</month>
    <dayOfMonth>10</dayOfMonth>
    <hour>0</hour>
    <minute>52</minute>
  </startTime>

  <endTime>
    <year>2008</year>
    <month>3</month>
    <dayOfMonth>6</dayOfMonth>
    <hour>21</hour>
    <minute>57</minute>
  </endTime>

  <!-- A possibly empty list of violations-->
  <violations>

    <!-- A violation has a metric, a type and an action -->
    <violation>
      <metric type="RESOURCE">
        <uri>http://www.gria.org/sla/metric/resource/disc</uri>

        <description>
          <description>disc space</description>
          <instantaneous>amount of disc space</instantaneous>
        </description>
```

```
     <units type="BINARY">
        <instantaneous>B</instantaneous>
      </units>
    </metric>
    <type>INSTANTANEOUS</type>
    <action>notify</action>
  </violation>

  <violation>
    <metric type="RESOURCE">
      <uri>http://www.gria.org/sla/metric/resource/disc</uri>

      <description>
        <description>disc space</description>
        <instantaneous>amount of disc space</instantaneous>
      </description>

     <units type="BINARY">
        <instantaneous>B</instantaneous>
      </units>
    </metric>
    <type>CUMULATIVE</type>
    <action>notify</action>
  </violation>

</violations>

<!-- Take into consideration what the user has done
     in the past, to apply a discount -->
<history>
  <!-- according to the total violations the user made
       recently, apply a discount
       A possibly empty list of ranges -->
  <totalViolations numOfBillingPeriods="3">
    <range>
      <lowerBound>0</lowerBound>
      <upperBound>5</upperBound>
      <discount>10</discount>
    </range>
  </totalViolations>

  <!-- according to how much the user has payed recently,
```

```
        apply a discount
        A possibly empty list of ranges -->
    <usage numOfBillingPeriods="2">
      <range>
        <lowerBound>0</lowerBound>
        <upperBound>10</upperBound>
        <discount>2</discount>
      </range>

      <range>
        <lowerBound>10</lowerBound>
        <upperBound>1000</upperBound>
        <discount>10</discount>
      </range>

    </usage>

  </history>

</policyTemplate>
```

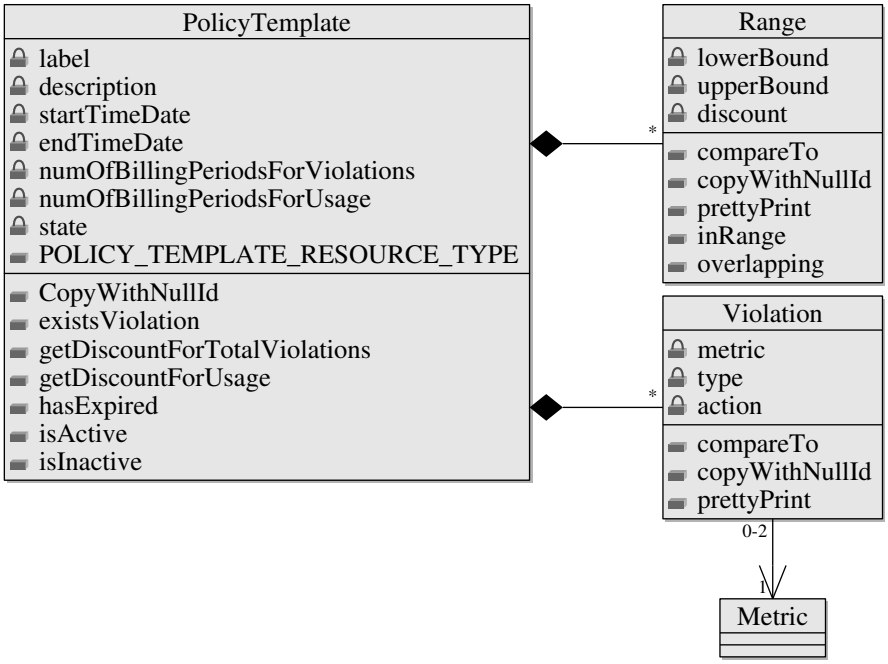
7.2 Υλοποίηση των πολιτικών στο GRIA

Όταν μία πολιτική διαβαστεί από ένα xml αρχείο, πρέπει να δημιουργηθούν ένα νέο αντικείμενο που θα περιέχει τις πληροφορίες της πολιτικής και η αντίστοιχη καταχώρηση στη βάση δεδομένων.

Στο GRIA η αντιστοίχιση των δομών της Java στη βάση δεδομένων γίνεται με χρήση του Hibernate. Πρόκειται για ένα open-source λογισμικό που αντιστοιχίζει ένα αντικειμενοστραφές μοντέλο πεδίου (object-oriented domain model) σε μια σχεσιακή βάση δεδομένων. Οπότε, για τη δημιουργία και χρήση πολιτικών ήταν απαραίτητη η δημιουργία ενός νέου αντικειμένου που θα αναπαριστά τις πολιτικές καθώς και των αντίστοιχων αρχείων του hibernate.

Αναλυτικότερα, δημιουργήθηκαν οι κλάσεις `PolicyTemplate`, `Violation` και `Range`, κατ' αντιστοιχία των όσων περιγράφηκαν στην ενότητα 7.1. Στο σχήμα 7.1 δίνεται το UML διάγραμμα κλάσεων για τις δομές αυτές. Σημειώνεται ότι για λόγους απλούστευσης του σχήματος, δεν εμφανίζονται οι μέθοδοι setters και getters των κλάσεων.

Στο σημείο αυτό αξίζει να σχολιαστεί η πολλαπλότητα της σχέσης μεταξύ των μετρικών και των παραβιάσεων. Μια παραβίαση έχει μία ακριβώς μετρική και μια μετρική μπορεί να υπάρχει σε δύο το πολύ παραβιάσεις. Αυτό είναι κατανοητό αν αναλογιστεί κανείς την ανάγκη για μοναδικότητα των παραβιάσεων όπως περιγράφηκε στην ενότητα 7.1 (μοναδικότητα του ζεύγους μετρικής - τύπου). Ο τύπος μπορεί να πάρει δύο διακριτές τιμές (`CUMULATIVE`, `INSTANTANEOUS`), οπότε για μια δεδομένη



Σχήμα 7.1: Διάγραμμα κλάσεων για τις πολιτικές

μετρική, υπάρχουν δύο διαφορετικά ζευγάρια μετρικής - τύπου, συνεπώς υπάρχουν δύο μόνο διαφορετικές παραβιάσεις.

Επιπλέον, όπως αναφέρθηκε στην ενότητα 7.1, ορισμένα πεδία των κλάσεων μπορούν να λάβουν συγκεκριμένες μόνο τιμές. Για παράδειγμα στην κλάση `Violation` η μεταβλητή `type` μπορεί να λάβει μία από τις ακόλουθες δύο τιμές: `INSTANTANEOUS` ή `CUMULATIVE`. Για να είναι ευκολότερος ο έλεγχος της εγκυρότητας των πεδίων αυτών όταν δίνονται ως είσοδο από το χρήστη, καθώς και η προσθήκη περισσότερων δυνατών τιμών στο μέλλον, ορίστηκε για κάθε τέτοια μεταβλητή ένας αντίστοιχος τύπος. Στον πίνακα 7.1 εμφανίζονται συγκεντρωτικά οι μεταβλητές αυτές, οι τιμές που μπορούν να λάβουν και οι τύποι που ορίστηκαν.

7.3 Διαχείριση πολιτικών

Η διαχείριση των πολιτικών γίνεται μέσω μιας ιστοσελίδας που είναι μέρος της δικτυακής εφαρμογής του πακέτου Service Provider Management. Η πρόσβαση στη σελίδα διαχείρισης των πολιτικών γίνεται μέσω ενός συνδέσμου με την επιγραφή Policy Manager, ο οποίος έχει τοποθετηθεί στην μπάρα πλοήγησης που βρίσκεται στο πάνω μέρος της δικτυακής εφαρμογής. Η ιστοσελίδα διαχείρισης των πολιτικών επιτρέπει τη δημιουργία, αντιγραφή και διαγραφή πολιτικών. Ακόμη ο χρήστης μπορεί να δει μια λίστα με όλες τις διαθέσιμες πολιτικές όπως επίσης και τις λεπτομέρειες μιας

Πίνακας 7.1: Τύποι Μεταβλητών

Variable	Class	Values	User Defined Types

`state`	`PolicyTemplate`	`inactive`
		`active`
		`expired`
		`closed`
			`PolicyTemplateState`

`type`	`Violation`	`INSTANTANEOUS` `CUMULATIVE`	`UsageType`
`action`	`Violation`	`kill` `notify`	`ActionType`

επιλεγμένης πολιτικής.

Για τη διαχείριση των πολιτικών δημιουργήθηκε ένα νέο πακέτο στο φάκελο sla/service/src/java, το `uk.ac.soton.itinnovation.grid.service.policy`. Το πακέτο αυτό περιλαμβάνει την κλάση `MyPolicyAdmin` η οποία ουσιαστικά αναλαμβάνει τη διαχείριση των πολιτικών μέσω της παραπάνω ιστοσελίδας. Επίσης στο πακέτο αυτό βρίσκεται η κλάση `PolicyServiceImpl` η οποία είναι υπεύθυνη για τη διαχείριση των πολιτικών στο επίπεδο των πόρων. Δηλαδή, για τη δημιουργία μιας πολιτικής από ένα αρχείο xml και την προσθήκη της στη βάση δεδομένων, τη διαγραφή μιας πολιτικής από τη βάση, την ενεργοποίηση καθώς και τον έλεγχο της κατάστασης μιας πολιτικής.

Η διαφορετική λειτουργία των κλάσεων αυτών μπορεί να γίνει περισσότερο κατανοητή με ένα παράδειγμα. Αν ο χρήστης επιλέξει λόγου χάρη τη διαγραφή μιας πολιτικής, η κλάση `MyPolicyAdmin` εμφανίζει τα σχετικά μηνύματα επιβεβαίωσης και καλεί μεθόδους της κλάσης `PolicyServiceImpl` προκειμένου να γίνει ουσιαστικά η διαγραφή της πολιτικής, δηλαδή να αφαιρεθεί από τη βάση δεδομένων. Τέλος η κλάση `MyPolicyAdmin` ανανεώνει το γραφικό περιβάλλον (δεν εμφανίζεται πλέον η πολιτική στη λίστα με τις διαθέσιμες πολιτικές).

Πρέπει να σημειωθεί ότι κατά τη δημιουργία μιας πολιτικής από ένα αρχείο xml γίνεται έλεγχος της εγκυρότητας όλων των πεδίων. Εκτός από τους απλούς ελέγχους εγκυρότητας των τύπων και των τιμών, ελέγχεται επιπλέον τα διαστήματα που μπορεί να περιέχει το `<totalViolations>` να μην επικαλλύπτονται. Όμοια για την περίπτωση του `<usage>`. Επίσης ελέγχεται κάθε `<violation>` να είναι μοναδικό, δηλαδή να μην υπάρχουν δύο παραβιάσεις με τα ίδια `<metric>` και `<type>`.

Επιπλέον, δεν επιτρέπεται η διαγραφή μιας πολιτικής αν είναι συνδεδεμένη με ένα SLA Template. Εδώ πρέπει να σημειωθεί ότι το GRIA δεν εμποδίζει τη διαγραφή ενός SLA Template όταν υπάρχουν SLAs που έχουν δημιουργηθεί από αυτό. Η αντιμετώπιση αυτή όμως δημιουργεί περιπλοκές στη λειτουργία των πολιτικών καθώς το SLA Template είναι ο συνδετικός κρίκος μεταξύ των πολιτικών και των SLAs. Για παράδειγμα αν διαγραφεί ένα SLA Template τότε δεν υπάρχει τρόπος τα SLAs που

έχουν δημιουργηθεί από αυτό, να συνδεθούν με μια πολιτική. Οπότε τροποποιήθηκε ο κώδικας του συστήματος, ώστε να μην επιτρέπεται η διαγραφή ενός SLA Template αν υπάρχουν SLAs που έχουν δημιουργηθεί από αυτό.

7.4 Σύνδεση μιας πολιτικής με ένα SLA

Ο χρήστης μπορεί να συνδέσει μια πολιτική με ένα SLA Template μέσω της υπάρχουσας ιστοσελίδας διαχείρισης των SLAs του Service Provider Management. Η ιστοσελίδα αυτή έχει τροποποιηθεί ώστε δίπλα σε κάθε SLA Template να εμφανίζεται μια λίστα με τις διαθέσιμες πολιτικές καθώς και τα απαραίτητα κουμπιά για τη σύνδεση και την αποσύνδεση μιας πολιτικής με το SLA Template. Όταν μια πολιτική συσχετισθεί με ένα SLA Template, τότε αμέσως συνδέεται και με όλα τα SLAs που έχουν προέλθει από αυτό το template. Για να είναι δυνατή αυτή η λειτουργία, προστέθηκε μια μεταβλητή `policyTemplate` στην κλάση SLATemplate καθώς και μια μεταβλητή `slaTemplate` στην κλάση SLA που δηλώνει το template από το οποίο προήλθε το SLA.

Οι σχέσεις μεταξύ των κλάσεων αυτών φαίνονται στο UML διάγραμμα κλάσεων του σχήματος 7.2. Για λόγους απλούστευσης του σχήματος, δεν εμφανίζονται οι μέθοδοι setters και getters των αντικειμένων.

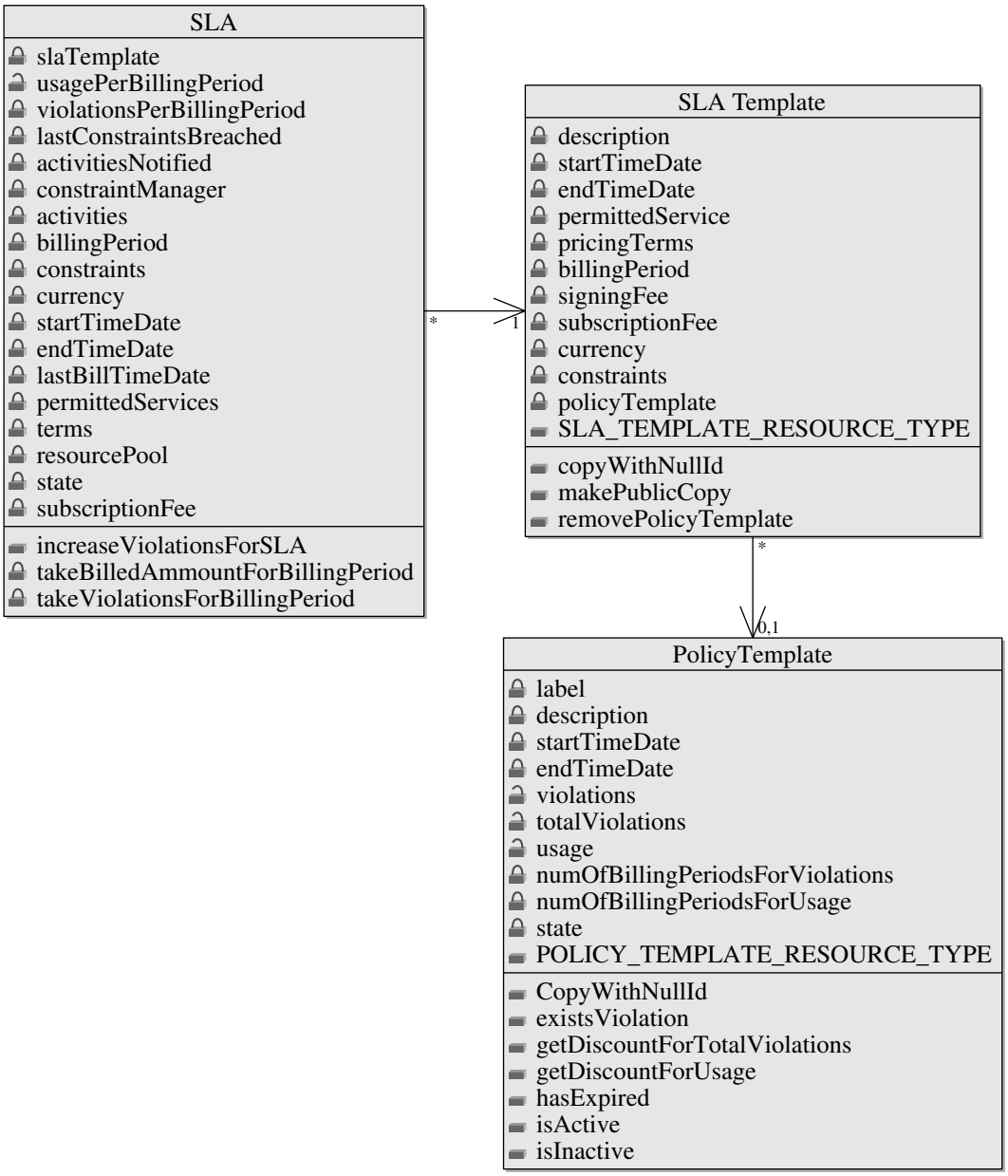
7.5 Εφαρμογή μιας πολιτικής σε ένα SLA

Όταν μια ενεργή πολιτική είναι συνδεδεμένη με ένα SLA τότε προφανώς πρέπει οι όροι της να εφαρμοστούν. Πρώτα από όλα, κάθε φορά που παραβιάζεται ένας περιορισμός του SLA πρέπει να ελέγχεται τι είδους ενέργεια προβλέπει η πολιτική. Αν δεν υπάρχει πρόβλεψη, τότε εκτελείται η προκαθορισμένη ενέργεια, δηλαδή διακόπτεται ο απαραίτητος αριθμός δραστηριοτήτων (που χρησιμοποιούν τη μετρική που προσδιορίζεται στον περιορισμό που παραβιάστηκε), προκειμένου η χρήση της μετρικής να είναι μέσα στα επιτρεπτά όρια. Επίσης πρέπει να μετρώνται οι παραβιάσεις και η χρήση που γίνονται σε κάθε περίοδο χρέωσης και όταν περάσει ο προβλεπόμενος αριθμός περιόδων χρέωσης να υπολογίζεται η έκπτωση που θα εφαρμοστεί.

7.5.1 Μέτρηση παραβιάσεων

Στο GRIA υπάρχει η κλάση `ConstraintManagerImpl` η οποία ελέγχει ποιοι περιορισμοί έχουν παραβιαστεί. Ο έλεγχος αυτός γίνεται σε δύο επίπεδα:

- Όταν ο χρήστης θέλει να ξεκινήσει μια νέα δραστηριότητα η οποία κάνει χρήση μετρικών, ελέγχεται αν η δραστηριότητα αυτή παραβιάζει κάποιον όρο του SLA. Για παράδειγμα, έστω ότι στο SLA προβλέπεται ότι ο χρήστης δεν μπορεί να χρησιμοποιήσει περισσότερα από 2MB χώρου αποθήκευσης. Αν ο χρήστης προσπαθήσει να ανεβάσει ένα αρχείο μεγαλύτερου μεγέθους, τότε ο περιορισμός



Σχήμα 7.2: Διάγραμμα κλάσεων για SLAs, SLA Templates και πολιτικές

αυτός θα παραβιαστεί. Πριν ξεκινήσει λοιπόν η δραστηριότητα αυτή, εντοπίζεται ότι θα υπάρξει παραβίαση και δεν επιτρέπεται η εκκίνηση της δραστηριότητας. Ο έλεγχος αυτός γίνεται στη μέθοδο `startActivity` της κλάσης που προαναφέρθηκε.

- Δεν είναι δυνατόν όμως να ελεγθούν έτσι όλοι οι περιορισμοί. Για παράδειγμα στην περίπτωση που στο SLA υπάρχει ένας περιορισμός για το χρόνο που μπορεί να χρησιμοποιήσει ο χρήστης τη CPU. Για να ελεγχθεί πότε θα παραβιαστεί ο περιορισμός αυτός, πρέπει να ελέγχεται συνεχώς η χρήση της CPU και όταν περάσει το επιτρεπόμενο όριο, να ληφθούν τα κατάλληλα μέτρα. Το διάστημα σταθμοσκόπησης (polling interval) μπορεί να οριστεί από το διαχειριστή του συστήματος. Η προεπιλεγμένη τιμή (η οποία χρησιμοποιήθηκε σε αυτή την εργασία) είναι 20 δευτερόλεπτα. Η διαδικασία αυτή γίνεται στη μέθοδο `getActionsForSLA` της κλάσης `ConstraintManagerImpl`.

Οι μέθοδοι που αναφέρθηκαν παραπάνω ανίχνευουν τις παραβιάσεις. Τροποποιήθηκαν λοιπόν κατάλληλα, ώστε παράλληλα με τον εντοπισμό να γίνεται και καταμέτρηση των παραβιάσεων ενός χρήστη. Αναλυτικότερα, στις μεθόδους αυτές κατασκευαζόταν ένα σύνολο που περιείχε όλους τους περιορισμούς που είχαν παραβιαστεί. Προφανώς το μέγεθος του συνόλου αυτού αντιστοιχεί στο πλήθος των παραβιάσεων.

Οι παραβιάσεις αυτές έπρεπε να αποθηκεύονται και στη βάση δεδομένων. Για το σκοπό αυτό προστέθηκε η μεταβλητή `List<Integer> violationsPerBP` στην κλάση που αναπαριστά τα SLAs. Πρόκειται για μια λίστα ακεραίων, κάθε στοιχείο της οποίας αντιστοιχεί στο πλήθος των παραβιάσεων που έγινε σε μια περίοδο χρέωσης. Δηλαδή το πρώτο στοιχείο της λίστας αντιστοιχεί στις παραβιάσεις που έγιναν την πρώτη περίοδο χρέωσης από τη στιγμή που ενεργοποιήθηκε η πολιτική. Η λίστα αυτή, μετά την πάροδο του πλήθους των περιόδων χρέωσης που προσδιορίζεται από την πολιτική για τη μέτρηση παραβιάσεων, αδειάζει, ώστε να εξοικονομείται χώρος. Βέβαια με τη λογική αυτή, θα μπορούσε να είχε χρησιμοποιηθεί απλά ένας ακέραιος αριθμός που θα αύξανε συνεχώς και θα μηδενιζόταν μετά την πάροδο των απαραίτητων περιόδων χρέωσης. Η αναπαράσταση αυτή είναι περισσότερο αποδοτική, όμως με τη μορφή της λίστας αποθηκεύεται περισσότερη πληροφορία (η κατανομή των παραβιάσεων στο χρόνο). Η πληροφορία αυτή θα μπορούσε στο μελλον να χρησιμοποιηθεί για πιο εξελιγμένες μορφές πολιτικής.

Επίσης πρέπει να σημειωθεί ότι η μέτρηση των παραβιάσεων ενός SLA δε γίνεται συνεχώς, αλλά μόνο όταν είναι συνδεδεμένο με μια ενεργή πολιτική. Άλλωστε μόνο στην περίπτωση αυτή η μέτρηση έχει νόημα.

Όμως η μέτρηση των παραβιάσεων με τον τρόπο αυτό δε λαμβάνει υπόψη την περίπτωση που η παραβίαση ενός όρου του SLA δεν οδηγεί σε τερματισμό της δραστηριότητας, αλλά σε ειδοποίηση του χρήστη. Αν δηλαδή ο χρήστης παραβιάσει έναν όρο του SLA του για τον οποίο προβλέπεται από την πολιτική η ενέργεια `notify`, ο χρήστης ενημερώνεται σχετικά, όμως η δραστηριότητα συνεχίζεται κανονικά. Αυτό σημαίνει ότι την επόμενη φορά που θα ελεγχθεί αν έχει γίνει κάποια παραβιάση του SLA, η μέθοδος `getActionsForSLA` θα εντοπίσει την παραβίαση ξανά, καθώς η

χρήση της μετρικής θα είναι εκτός των επιτρεπτών ορίων. Δηλαδή κάθε παραβίαση θα προστίθεται ξανά στο συνολικό αριθμό παραβιάσεων κάθε φορά που θα καλείται η `getActionsForSLA` (δηλαδή κάθε 20 δευτερόλεπτα), για όσο χρονικό διάστημα επιτρέπεται από την πολιτική να «παραβιάζει» ο χρήστης το SLA του.

Για την αντιμετώπιση του προβλήματος αυτού προσθέσαμε στην κλάση `SLA` μία μεταβλητή `Set<Constraint> lastBreached`. Πρόκειται για ένα σύνολο στο οποίο αποθηκεύονται οι περιορισμοί που είχαν παραβιαστεί την τελευταία φορά που κλήθηκε η `getActionsForSLA`. Οπότε την επόμενη φορά που θα κληθεί η συνάρτηση αυτή, θα κατασκευάσει ένα νέο σύνολο (`breached`) με τους περιορισμούς που εκείνη τη στιγμή είναι παραβιασμένοι και συγκρίνοντάς το με το σύνολο `lastBreached` μπορούν να εντοπιστούν οι νέες παραβιάσεις, οι οποίες και πρέπει να μετρηθούν.

Πιο αναλυτικά, η τομή των δύο συνόλων $breached \cap lastBreached$ περιλαμβάνει τους περιορισμούς οι οποίοι ηταν παραβιασμένοι και την προηγούμενη φορά που έγινε έλεγχος, αλλά και τώρα. Οι παραβιάσεις αυτές δεν πρέπει να μετρηθούν εκ νέου. Αν από το σύνολο των νέων περιορισμών που παραβιάστηκαν, αφαιρεθούν οι περιορισμοί που ήταν παραβιασμένοι και την περασμένη φορά, τότε βρίσκονται οι περιορισμοί που τώρα παραβιάστηκαν για πρώτη φορά, οι οποίοι και πρέπει να μετρηθούν. Δηλαδή οι παραβιάσεις πρέπει να αυξηθούν κατά αριθμό ίσο με το πλήθος των στοιχείων του συνόλου $breached - breached \cap lastBreached$. Στο τέλος, το σύνολο `lastBreached` ανανεώνεται ώστε να περιέχει τα στοιχεία του συνόλου `breached`. Η διαδικασία αυτή σκιαγραφείται στον αλγόριθμο 1.

Αλγόριθμος 1 Αλγόριθμος για σωστή μετρηση παραβιάσεων

```
if policyTemplate != null then
  Set aSet = breached − breached ∩ lastBreached
  numOfNewViolations = aSet.size()
  if numOfNewViolations ≠ 0 then
    Αύξηση των παραβιάσεων του SLA
  end if
  lastBreached = breached
end if
```

Για να λειτουργήσει ο αλγόριθμος αυτός σωστά, κάθε φορά που καλείται η μέθοδος `startActivity` και μια δραστηριότητα παραβιάζει έναν περιορισμό, ο περιορισμός αυτός προστίθεται στο σύνολο `lastBreached` του SLA.

7.5.2 Αντιμετώπιση παραβιάσεων

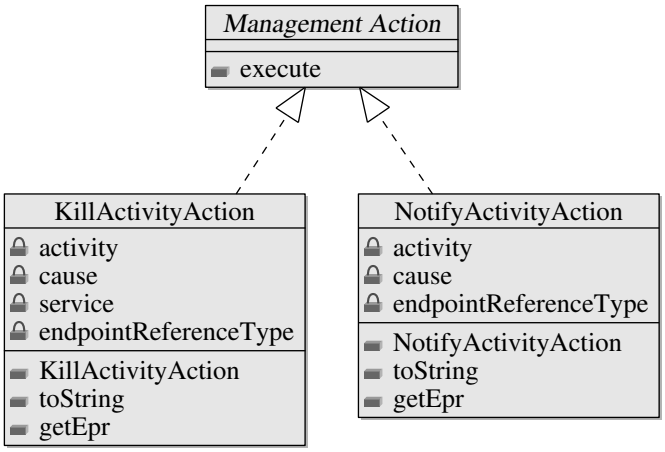
Όταν είναι συνδεδεμένη μια ενεργή πολιτική με ένα SLA πρέπει για κάθε παραβίαση να ελέγχεται η ενέργεια που προσδιορίζει η πολιτική αυτή για τη συγκεκριμένη παραβίαση. Οπότε ο κώδικας τροποποιήθηκε ώστε να ενεργεί με κατάλληλο τρόπο (kill, notify) ανάλογα με το τι προβλέπει η πολιτική. Ο αλγόριθμος 2 περιγράφει σε γενικές γραμμές πώς γίνεται η επιλογή της κατάλληλης ενέργειας.

Αλγόριθμος 2 Αλγόριθμος για την επιλογή κατάλληλης ενέργειας

```
if policyTemplate == null then
  Δεν υπάρχει συνδεδεμένη πολιτική.
  Εκτελείται η προκαθορισμένη ενέργεια kill.
else
  for all constraints that have breached do
    if η πολιτική προσδιορίζει κάτι για αυτή την παραβίαση then
      if η ενέργεια που προσδιορίζει η πολιτική είναι kill then
        Διακοπή των δραστηριοτήτων.
      else if η ενέργεια που προσδιορίζει η πολιτική είναι notify then
        Συνέχιση της δραστηριότητας.
        Ειδοποίηση του χρήστη.
      end if
    else
      Η πολιτική δεν προβλέπει κάτι για αυτή την παραβίαση.
      Εκτελείται η προκαθορισμένη ενέργεια kill.
    end if
  end for
end if
```

Όμως με την προσθήκη αυτής της λειτουργίας, προκύπτει το εξής πρόβλημα: Αν ο χρήστης παραβιάσει έναν όρο του SLA του για τον οποίο προβλέπεται από την πολιτική η ενέργεια `notify`, ο χρήστης ενημερώνεται σχετικά, όμως η δραστηριότητα συνεχίζεται κανονικά. Αυτό σημαίνει ότι την επόμενη φορά που θα ελεγχθεί αν έχει γίνει κάποια παραβιάση του SLA, η μέθοδος `getActionsForSLA` θα εντοπίσει την παραβίαση ξανά, αφού η χρήση της μετρικής θα είναι εκτός των επιτρεπτών ορίων. Αυτό σημαίνει ότι ο χρήστης θα λαμβάνει συνεχώς ειδοποιήσεις για όσο χρονικό διάστημα του επιτρέπεται από την πολιτική να «παραβιάζει» το SLA του.

Η συμπεριφορά αυτή είναι προφανώς ανεπιθύμητη. Το ιδανικό θα ήταν ο χρήστης να λαμβάνει μία μόνο ειδοποίηση για κάθε δραστηριότητά του που παραβιάζει έναν όρο του SLA. Το πρόβλημα αυτό αντιμετωπίστηκε όπως και το αντίστοιχο πρόβλημα με τη μέτρηση των παραβιάσεων (στην ενότητα 7.5.1). Δηλαδή προσθέσαμε στην κλάση `SLA` τη μεταβλητή `Set<Activity> lastNotified`. Πρόκειται για ένα σύνολο στο οποίο αποθηκεύονται οι δραστηριότητες για τις οποίες είχε σταλεί ειδοποίηση την τελευταία φορά που κλήθηκε η συνάρτηση `getActionsForSLA`. Την επόμενη φορά που θα κληθεί η συνάρτηση αυτή, θα κατασκευαστεί ένα νέο σύνολο (`activitiesNotified`) με τις δραστηριότητες που αυτή τη φορά θα έπρεπε να ειδοποιηθούν και συγκρίνοντάς το `lastNotified` μπορούν να εντοπιστούν οι νέες δραστηριότητες, οι οποίες και πρέπει να ειδοποιηθούν. Προφανώς κάθε νέα δραστηριότητα που ειδοποιείται απο τη μέθοδο `startActivity` πρέπει να εισάγεται στο σύνολο `lastNotified` του αντίστοιχου SLA.



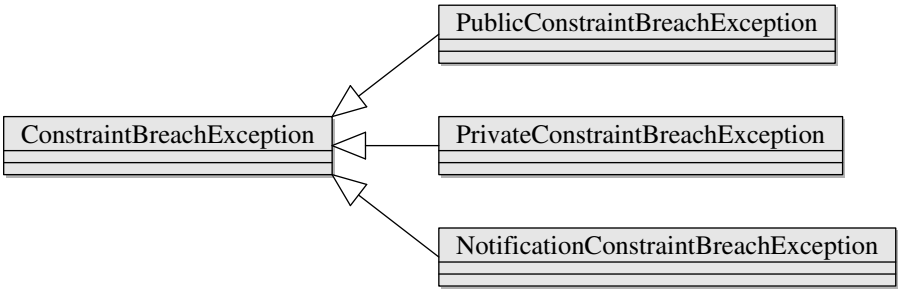
Σχήμα 7.3: Διάγραμμα κλάσεων για τις ενέργειες

Εδώ πρέπει να σημειωθεί ότι οι μέθοδοι `startActivity` και `getActionsForSLA` αντιμετωπίζουν με διαφορετικό τρόπο τις παραβιάσεις. Η `startActivity` δεν επιτρέπει σε μια δραστηριότητα να ξεκινήσει και αν έχει παραβιαστεί δημόσιος περιορισμός δημιουργεί μια εξαίρεση `PublicConstraintBreachException`, ενώ αν έχει παραβιαστεί ιδιωτικός περιορισμός μια εξαίρεση `PrivateConstraintBreachException`. Τις εξαιρέσεις αυτές τις χειρίζονται κατάλληλα οι μέθοδοι που καλούν τη `startActivity`. Αντίθετα η `getActionsForSLA` δημιουργεί μια λίστα με όλες τις ενέργειες που πρέπει να γίνουν (κάθε ενέργεια αντιστοιχεί σε μια δραστηριότητα που παραβιάζει έναν περιορισμό του SLA) και στη συνέχεια εκτελεί τις ενέργειες αυτές (η προκαθορισμένη ενέργεια ορίζεται στην κλάση `KillActivityAction`).

Η διαφορετική αυτή αντιμετώπιση έπρεπε να διατηρηθεί και στην περίπτωση που προβλέπεται αποστολή ειδοποίησης στο χρήστη για μια παραβίαση. Έτσι ορίστηκε ένας νέος τύπος εξαίρεσης με το όνομα `NotificationConstraintBreachException` και η `startActivity` δημιουργεί μια εξαίρεση τέτοιου τύπου όταν παραβιάζεται ένας περιορισμός και πρέπει να σταλεί ειδοποίηση. Επίσης δημιουργήθηκε ένας νέος τύπος ενέργειας `NotifyActivityAction` ώστε να προστίθεται στη λίστα με τις ενέργειες στη μέθοδο `getActionsForSLA`. Στα σχήματα 7.3 και 7.4 δίνονται τα διαγράμματα κλάσεων για τις ενέργειες και τις εξαιρέσεις αντίστοιχα.

7.5.3 Μέτρηση της χρήσης

Το μέγεθος που χρησιμοποιήθηκε στην εργασία αυτή ως μέτρο της χρήσης είναι το συνολικό ποσό που καλείται να πληρώσει ο χρήστης στο τέλος κάθε περιόδου χρέωσης. Όπως συμβαίνει και με τις συνολικές παραβιάσεις, το ποσό αυτό πρέπει να αποθηκεύεται και στη βάση δεδομένων. Για το σκοπό αυτό προστέθηκε η μεταβλητή `List<BigDecimal> usagePerBP` στην κλάση `SLA`. Πρόκειται για μια λίστα αριθμών,



Σχήμα 7.4: Διάγραμμα κλάσεων για τις εξαιρέσεις

κάθε στοιχείο της οποίας είναι το ποσό που αντιστοιχεί στις υπηρεσίες που χρησιμοποίησε ο πελάτης σε μια περίοδο χρέωσης. Η λίστα αυτή, μετά την πάροδο του πλήθους των περιόδων χρέωσης που προσδιορίζεται από την πολιτική για τη μέτρηση της χρήσης, αδειάζει, ώστε να εξοικονομείται χώρος.

Η χρέωση των SLAs στο τέλος κάθε περιόδου χρέωσης γίνεται στη μέθοδο `sendBills` της κλάσης `SLA`. Η μέθοδος αυτή υπολογίζει το ποσό του λογαριασμού που θα σταλεί στο χρήστη. Ο κώδικας της μεθόδου τροποποιήθηκε έτσι ώστε το ποσό αυτό να προστίθεται απλά στη λίστα `usagePerBP`.

7.5.4 Εφαρμογή έκπτωσης

Όταν υπολογίζεται ο λογαριασμός για ένα SLA στο τέλος κάθε περιόδου χρέωσης (στη μέθοδο `sendBills` της κλάσης `SLA`), χρησιμοποιούμε την πληροφορία για τις παραβιάσεις και τη χρήση για να υπολογίσουμε την έκπτωση που πρέπει να εφαρμοστεί.

Στην περίπτωση που πρέπει να εφαρμοστεί έκπτωση λόγω συνολικών παραβιάσεων και λόγω της χρήσης, οι εκπτώσεις αυτές εφαρμόζονται διαδοχικά. Το τελικό ποσό μετά την εφαρμογή της έκπτωσης δίνεται από τον τύπο:

$$x = (1 - d_1\%) \cdot (1 - d_2\%) \cdot x'$$

όπου x' το ποσό πριν την εφαρμογή της έκπτωσης, x το ποσό μετά την εφαρμογή της έκπτωσης, d_1 και d_2 τα δύο ποσοστά επί τοις εκατό των εκπτώσεων. Προφανώς η σειρά εφαρμογής των εκπτώσεων δεν επηρεάζει το αποτέλεσμα.

Μια άλλη επιλογή θα ήταν οι εκπτώσεις να προστίθενται, δηλαδή έκπτωση 20% λόγω των παραβιασεων και έκπτωση 30% λόγω της χρήσης να έχει συνολική έκπτωση 50%. Η προσέγγιση αυτή είναι μη πρακτική, καθώς θα μπορούσε να προκύψει εύκολα έκπτωση μεγαλύτερη από 100%!

Κεφάλαιο 8

Συμπεράσματα - Μελλοντικές Εξελίξεις

Οι συμφωνίες επιπέδου υπηρεσιών είναι ζωτικής σημασίας στο περιβάλλον Πλέγματος, ιδιαίτερα για τη διείσδυση της τεχνολογίας αυτής στον επιχειρηματικό τομέα. Η χρήση πολιτικών καθιστά αποτελεσματικότερη τη διαχείρισή τους, αφού προσφέρει στον παροχέα υπηρεσιών ένα ακόμη επίπεδο ελέγχου των SLAs. Οι πολιτικές προσφέρουν ένα ευέλικτο μοντέλο για την εφαρμογή της επιθυμητής επιχειρηματικής στρατηγικής.

Η μορφή που ορίστηκε για τις πολιτικές στα πλαίσια αυτής της εργασίας είναι σχετικά απλή και λαμβάνει υπόψη της βασικές απαιτήσεις ενός επιχειρηματικού μοντέλου. Η μορφή των πολιτικών θα μπορούσε να εξελιχθεί, ώστε να ανταποκρίνεται καλύτερα στις ανάγκες του επιχειρηματικού κόσμου. Για παράδειγμα θα μπορούσαν να συγκεντρώνονται πληρέστερα στατιστικά στοιχεία για τη χρήση των πόρων και των SLAs και να δίνεται η δυνατότητα στον παροχέα υπηρεσιών να ρυθμίζει καλύτερα τη συμπεριφορά του απέναντι στους πελάτες. Επίσης θα μπορούσαν να οριστούν νέες ενέργειες, πέρα από τις `kill` και `notify`, ώστε να μπορεί ο πάροχος υπηρεσιών να προσαρμόζει σε μεγαλύτερο βαθμό την αντίδρασή του σε ορισμένα γεγονότα όπως η παραβίαση όρων του SLA. Για παράδειγμα θα μπορούσε να οριστεί μια ενέργεια που θα αναλαμβάνει τη μείωση της χρήσης ενός πόρου αν παραβιαστεί ένας περιορισμός που κάνει χρήση της αντίστοιχης μετρικής. Ή ακόμα μια ενέργεια που θα ειδοποιεί το χρήστη ότι έχει κάνει μια παραβίαση, αλλά δε θα του επιτρέπει να συνεχίζει επ΄ άπειρον τη χρήση των πόρων αυτών. Μετά την πάροδο ορισμένου χρονικού διαστήματος, αν ο χρήστης δεν έχει φροντίσει να επαναφέρει τη χρήση των πόρων μέσα στο όρια που προβλέπονται από το SLA, ο πάροχος τερματίζει τις κατάλληλες δραστηριότητες.

Στο σύστημα που υλοποιήθηκε για την εργασία αυτή, σε κάθε SLA μπορεί να εφαρμόζεται κάθε στιγμή μία μόνο πολιτική. Αυτό θα μπορούσε να διευρυνθεί, ώστε σε ένα SLA να μπορούν να εφαρμόζονται πολλές πολιτικές ταυτόχρονα. Στην περίπτωση αυτή, όροι διαφορετικών πολιτικών μπορεί να έρχονται σε σύγκρουση μεταξύ τους, οπότε είναι απαραίτητος ο ορισμός μιας ιεραρχίας πολιτικών που θα καθορίζει ποια πολιτική έχει προτεραιότητα και ποιοι όροι θα εφαρμοστούν τελικά. Στην κορυφή

της ιεραρχίας αυτής, θα μπορούσε για παράδειγμα να υπάρχει μια προκαθορισμένη πολιτική που θα ισχύει για όλα τα SLAs. Ο διαχειριστής του συστήματος θα μπορεί να εφαρμόζει σε συγκεκριμένα SLAs άλλες πολιτικές οι οποίες θα παρακάμπτουν την προκαθορισμένη πολιτική ή μερικούς όρους της μόνο. Αυτή η ιεραρχική ταξινόμηση των πολιτικών θα προσφέρει στον πάροχο ακόμα μεγαλύτερη ευελιξία στη διαχείριση των SLAs.

Ένα ακόμη βήμα που θα μπορούσε να γίνει είναι να αναπτυχθεί ένα σύστημα που με αυτόματο τρόπο θα επιλέγει την εφαρμογή της κατάλληλης πολιτικής προκειμένου να αυξηθεί το κέρδος του παροχέα υπηρεσιών. Βεβαια αυτό προϋποθέτει την ύπαρξη ενός επιχειρηματικού μοντέλου το οποίο θα ορίζει τι είναι μεγιστοποίηση του κέρδους και θα παρέχει τις απαραίτητες πληροφορίες για την ανάπτυξη ενός αλγορίθμου για την επιλογή της σωστής πολιτικής.

Παράρτημα Α΄

Αρχεία που Δημιουργήθηκαν ή Τροποποιήθηκαν

Α΄.1 Αρχεία που δημιουργήθηκαν

Όλα τα αρχεία που δημιουργήθηκαν ανήκουν στο πακέτο του Service Provider Management. Παρακάτω παρατίθεται μια λίστα με τα αρχεία αυτά. Σημειώνεται ότι δίνεται ολόκληρο το μονοπάτι του αρχείου που δημιουργήθηκε μέσα στο φάκελο που περιέχει τον κώδικα του Service Provider Management (δηλαδή το φάκελο gria-5.1-src/service-provider-mgt).

- `sla\common\src\java\uk.ac.soton.itinnovation.grid.service.types\PolicyTemplate.java`
- `sla\service\src\java\uk.ac.soton.itinnovation.grid.service.types\PolicyTemplate.hbm.xml`
- `sla\common\src\java\uk.ac.soton.itinnovation.grid.service.types\PolicyTemplateState.java`
- `sla\common\src\java\uk.ac.soton.itinnovation.grid.service.types\ActionType.java`
- `sla\common\src\java\uk.ac.soton.itinnovation.grid.service.types\Range.java`
- `sla\service\src\java\uk.ac.soton.itinnovation.grid.service.types\Range.hbm.xml`
- `sla\common\src\java\uk.ac.soton.itinnovation.grid.service.types\Violation.java`
- `sla\service\src\java\uk.ac.soton.itinnovation.grid.service.types\Violation.hbm.xml`

- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla\PolicyServiceImpl.java
- sla\service\admin\src\main\webapps\sla-admin\uploadPolicyFile.jsp
- sla\service\admin\src\main\webapps\sla-admin\policyTemplate.jsp
- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla\MyPolicyAdmin.java
- sla\service\src\resources\policy-template-policy.xml
- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla.bizfacade\PolicyTemplateFacade.java
- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla.bizfacade\PolicyTemplateFacadeImpl.java
- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla.dao\PolicyTemplateDAO.java
- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla.dao.hibernate\PolicyTemplateDAOImpl.java
- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla.NotifyActivityAction.java
- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla.NotificationConstraintBreachException.java

Α΄.2 Αρχεία που τροποποιήθηκαν

Παρακάτω παρατίθεται μια λίστα με τα αρχεία του πακέτου Service Provider Management που τροποποιήθηκαν. Σημειώνεται ότι δίνεται ολόκληρο το μονοπάτι του αρχείου που τροποποιήθηκε μέσα στο φάκελο που περιέχει τον κώδικα του Service Provider Management (δηλαδή το φάκελο gria-5.1-src/service-provider-mgt).

- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla\ConstraintManagerImpl.java
- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla\ConstraintManager.java
- sla\service\admin\src\main\webapps\sla-admin\index.html

- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla.bizfacade\BizfacadeFactory.java
- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla.bizfacade.spring\SpringBizfacadeFactory.java
- release\src\main\webapp\WEB-INF\classes\applicationContext.xml
- release\src\main\webapp\WEB-INF\classes\hibernate.cfg.xml
- sla\service\admin\src\main\webapps\sla-admin\editor.js
- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla.dao\DAOFactory.java
- sla\common\src\java\uk.ac.soton.itinnovation.grid.service.types\SLATemplate.java
- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.types\SLATemplate.hbm.xml
- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla.webadmin\SLAAdmin.java
- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla\SLA.java
- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla.bizfacade\SLAFacadeImpl.java
- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla\ResourceWithUsageByMetric.hbm.xml
- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla\SLAServiceImpl.java
- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla\ResourcePool.java
- sla\service\src\java\uk.ac.soton.itinnovation.grid.service.sla\UsageMessageHandler.java

Επίσης τροποποιήθηκαν τα εξής αρχεία (δεν ανήκουν στο πακέτο του Service Provider Management):

- platform-scripts\rm_local\startJob.pl
- platform-scripts\rm_local\killJob.pl
- platform-scripts\rm_local\getJobStatus.pl

Παράρτημα Β΄

Πηγαίος Κώδικας

Περιεχόμενα

Στο παράρτημα αυτό παρατίθεται ο πηγαίος κώδικας των σημαντικότερων κλάσεων που υλοποιήθηκαν καθώς και τα βασικότερα τμήματα κώδικα που τροποποιήθηκαν.

Β′.1 Η κλάση PolicyTemplate.java

```
package uk.ac.soton.itinnovation.grid.service.types;

public class PolicyTemplate extends GridResource {

  private String label;
  private String description;
  //these define the policy's validity period
  private Date startTimeDate;
  private Date endTimeDate;
  private Violation[] violations = new Violation[] {};
  private Range[] totalViolations = new Range[] {};
  private Range[] usage = new Range[]{};
  private Integer numOfBillingPeriodsForUsage;
  private Integer numOfBillingPeriodsForViolations;
  private PolicyTemplateState state;
  public static final String POLICY_TEMPLATE_RESOURCE_TYPE =
   "http://www.it-innovation.soton.ac.uk/
    grid/resource/policy-template";

  private Logger logger = Logger.getLogger(getClass());

  public PolicyTemplate() {}

  //setters and getters
  public String getResourceType() {
    return POLICY_TEMPLATE_RESOURCE_TYPE;
  }

  public String getLabel() {
    return this.label;
  }

  public void setLabel(String argLabel) {
    this.label = argLabel;
  }

  public String getDescription() {
    return this.description;
  }

  public void setDescription(String argDescription) {
    this.description = argDescription;
```

```
}

public PolicyTemplateState getState (){
  return this.state;
}

public void setState (PolicyTemplateState state_) {
  this.state = state_;
}

public Violation[] getViolations() {
  return this.violations;
}

public void setViolations(Violation[] argViolation)
 throws GridFailureException {
  if (argViolation == null)
    argViolation = new Violation[]{};
  this.violations = argViolation;
}

public Integer getNumOfBillingPeriodsForViolations() {
  return this.numOfBillingPeriodsForViolations;
}

public void setNumOfBillingPeriodsForViolations (Integer num) {
  this.numOfBillingPeriodsForViolations = num;
}

public Integer getNumOfBillingPeriodsForUsage() {
  return this.numOfBillingPeriodsForUsage;
}

public void setNumOfBillingPeriodsForUsage (Integer num) {
  this.numOfBillingPeriodsForUsage = num;
}

public Calendar getStartTime() {
  if (this.startTimeDate == null) {
    return null;
  } else {
    GregorianCalendar gc =  new GregorianCalendar();
    gc.setTime(this.startTimeDate);
    return gc;
  }
}

public void setStartTime(Calendar argStartTime) {
  if (argStartTime == null) {
```

```
    this.startTimeDate = null;
  } else {
    this.startTimeDate =
     ((GregorianCalendar) argStartTime).getTime();
  }
}

public Calendar getEndTime() {
  if (this.endTimeDate == null) {
    return null;
  } else {
    GregorianCalendar gc =  new GregorianCalendar();
    gc.setTime(this.endTimeDate);
    return gc;
  }
}

public void setEndTime(Calendar argEndTime) {
  if (argEndTime == null) {
    this.endTimeDate = null;
  } else {
    this.endTimeDate = ((GregorianCalendar) argEndTime).getTime();
  }
}

public Range[] getTotalViolations(){
  return this.totalViolations;
}

public void setTotalViolations(Range[] totalViolations_)
 throws GridFailureException{
  if (totalViolations_ == null)
    totalViolations_ = new Range[]{};
  this.totalViolations = totalViolations_;
}

public Range[] getUsage(){
  return this.usage;
}

public void setUsage(Range[] usage_) throws GridFailureException{
  if (usage_ == null)
    usage_ = new Range[]{};
  this.usage = usage_;
}
//end of setters and getters

/**
 * Returns a copy of the Policy Template that will be
```

```
     * independent of the original when hibernated.
     * More precisely, the IDs of the template, of the
     * violations and of the ranges are set to null.
     * The metrics (used by the violations) are shared.
     */
    public PolicyTemplate copyWithNullId() {

      PolicyTemplate copy = new PolicyTemplate();
      copy.setLabel(this.getLabel());
      copy.setDescription(this.getDescription());
      copy.setStartTime((GregorianCalendar) this.getStartTime());
      copy.setEndTime((GregorianCalendar) this.getEndTime());
      copy.setNumOfBillingPeriodsForUsage
          (this.getNumOfBillingPeriodsForUsage());
      copy.setNumOfBillingPeriodsForViolations
          (this.getNumOfBillingPeriodsForViolations());
      copy.setState(this.getState());

      List<Violation> newViolations = new ArrayList<Violation>();
      for (Violation v: this.getViolations()) {
        newViolations.add(v.copyWithNullId());
      }
      try {
        copy.setViolations(newViolations.toArray(new Violation[]{}));
      } catch (GridFailureException ex) {
        ex.printStackTrace();
      }

      List<Range> newTotalViolations = new ArrayList<Range>();
      for (Range r: this.getTotalViolations()) {
        newTotalViolations.add(r.copyWithNullId());
      }
      try {
        copy.setTotalViolations(
         newTotalViolations.toArray(new Range[]{}));
      } catch (GridFailureException ex) {
        ex.printStackTrace();
      }

      List<Range> newUsage = new ArrayList<Range>();
      for (Range r: this.getUsage()) {
        newUsage.add(r.copyWithNullId());
      }
      try {
        copy.setUsage(newUsage.toArray(new Range[]{}));
      } catch (GridFailureException ex) {
        ex.printStackTrace();
      }
      return copy;
```

```
}

/**
 * Given a metric's uri and a constraint type,
 * finds if there is a matching violation
 * in this policy template.
 * <p>
 * If a violation is found returns the type of the
 * action that needs to be taken, else returns null.
 * @param metricURI The URI of the metric
 * @param type The type of the constraint breached
 * @return The action that needs to be taken
 */
public String existsViolation (String metricURI, String type ){
  String action = null;
  Violation[] violations = this.getViolations();
  logger.info("metric URI = "+metricURI+" type = "+type+"\n");

  for (int i=0;i<violations.length;i++){
    if (violations[i].getMetric().getURI().equals(metricURI)
        && violations[i].getType().equals(type)){
      action = violations[i].getAction();
      return action;
      //by definition, the metric's uri and the type define
      //unambiguously a violation so,
      //if one is found, it's the only one.
    }
  }
  return action;
}

/**
 * Finds the discount that should be applied, given the
 * number of violations.
 * @param violations The number of violations done
 * @return The discount that will be applied
 */
public Double getDiscountForViolations(Integer violations){

  for (int i=0;i<totalViolations.length;i++){
    if (totalViolations[i].inRange(violations.doubleValue())){
      return totalViolations[i].getDiscount();
    }
  }
  return 0.0;
}

/**
 * Finds the discount that should be applied, given the
```

```
 * amount that was billed to the user.
 * @param amount The amount that is billed to the user
 * @return The discount that will be applied
 */
public Double getDiscountForUsage(BigDecimal amount){

  for (int i=0;i<usage.length;i++){
    if (usage[i].inRange(amount.doubleValue())){
      return usage[i].getDiscount();
    }
  }
  return 0.0;
}

/**
 * Checks if this policy template has expired.
 * @return True if the policy template has expired.
 */
public boolean hasExpired (){
  GregorianCalendar now = new GregorianCalendar();
  if (now.getTimeInMillis()>=this.endTimeDate.getTime()) {
    return true;
  }
  else return false;
}

/**
 * Checks if this policy template is inactive.
 * @return True if the policy template is inactive.
 */
public boolean isInactive (){
  GregorianCalendar now = new GregorianCalendar();
  if (now.getTimeInMillis()<=this.startTimeDate.getTime()) {
    return true;
  }
  else return false;
}

/**
 * Checks the state of this policy template
 * @return -1 if this policy template is inactive
 *          0 if it's active
 *          1 if it has expired
 */
public int isActive (){
  GregorianCalendar now = new GregorianCalendar();
  if (now.getTimeInMillis()<=this.startTimeDate.getTime()){
    return -1;
  }
```

```
    else if (now.getTimeInMillis()>=this.endTimeDate.getTime()){
      return 1;
    }
    else return 0;
  }
}
```

B′.2 Το αρχείο PolicyTemplate.hbm.xml

```
<?xml version="1.0"?>
<!--
////////////////////////////////////////////////////////////////////
//                                                                //
//  Created By:            Katerina Marazopoulou                  //
//  Created Date:          2008-03-10                             //
//                                                                //
////////////////////////////////////////////////////////////////////
-->

<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping
 package="uk.ac.soton.itinnovation.grid.service.types">
  <class name="PolicyTemplate" table="PolicyManager_PolicyTemplate">
    <id name="resourceID">
      <generator class="uuid">
        <param name="separator">-</param>
      </generator>
    </id>

    <property name="label"/>
    <property name="description"/>
    <property name="startTimeDate" access='field'/>
    <property name="endTimeDate" access='field'/>
    <property name="numOfBillingPeriodsForUsage"/>
    <property name="numOfBillingPeriodsForViolations"/>

    <property name="state">
      <type name=
      "uk.ac.soton.itinnovation.grid.service.types.EnumUserType">
        <param name="enumClassName">
          uk.ac.soton.itinnovation.grid.service.types.PolicyTemplateState
        </param>
      </type>
    </property>
```

```
    <array name="violations" cascade="save-update">
      <key column="PolicyTemplate_id"/>
      <index column="idx" />
      <one-to-many
        class="uk.ac.soton.itinnovation.grid.service.types.Violation"/>
    </array>

    <array name="totalViolations" cascade="save-update">
      <key column="PolicyTemplate_id"/>
      <index column="idx" />
      <many-to-many
        class="uk.ac.soton.itinnovation.grid.service.types.Range"/>
    </array>

    <array name="usage" cascade="save-update">
      <key column="PolicyTemplate_id"/>
      <index column="idx" />
      <many-to-many
        class="uk.ac.soton.itinnovation.grid.service.types.Range"/>
    </array>
  </class>
</hibernate-mapping>
```

B′.3 Η κλάση MyPolicyAdmin.java

```
package uk.ac.soton.itinnovation.grid.service.policy;

/**
 * Manages the policies through the web-application.
 */
public class MyPolicyAdmin {
  private HttpServletRequest request = null;

  private static Logger log = Logger.getLogger(MyPolicyAdmin.class);
  private Logger thesisLog = Logger.getLogger("thesis");
  private PolicyServiceImpl policyService = new PolicyServiceImpl();
  private PDP pdp =
   (PDP) ImplementationFactory.getSingletonInstance(PDP.class);

  private int editorIDCount = 0;
  private int editorSubIDCount = -1;
  private String editorID()
   {return editorIDCount + (editorSubIDCount == -1 ? "" : ","
    + editorSubIDCount);}

  public MyPolicyAdmin(HttpServletRequest request) throws Stop {
    this.request = request;
  }
```

```
/** Checks whether this request is a POST request that asks
 * to upload a new policy, copy or delete an existing policy.
 * @return A success message, or null if no action was performed
 */
public String processPOST()
 throws Stop, GridFailureException, FileUploadException,
 IOException, XPathExpressionException {
  log.info("PolicyAdmin.processPOST()");
  if (FileUpload.isMultipartContent(request)) {
    return uploadFileData();
  }
  if(request.getParameter("finish.policy.template") != null){
    String resourceID =
     request.getParameter("finish.policy.template.resourceID");
    try{
      deletePt(resourceID);
    } catch (GridFailureException e){
      throw new Stop("Could not delete Policy template '"
                     + resourceID + "': " + e.getMessage());
    }
    return "Policy template '" + resourceID
          + "' was successfully deleted.";
  }
  if (request.getParameter("copy.policy.template") != null) {
    String resourceID =
     request.getParameter("copy.policy.template.resourceID");
    try {
      copyPt(resourceID);
    } catch (GridFailureException e){
      throw new Stop("Error copying Policy template: "
                     + e.getMessage());
    }
    return "Policy template successfully copied.";
  }
  return null;
}

/**
 * Copies the policy template with the given id into a
 * new one. The new policy template has a new id and a new label.
 * The new id is created automatically and the label's new value
 * is given by the user.
 * @param The id of the policy template that will be copied
 */
private void copyPt(String resourceID)
 throws GridFailureException {
  String label =
   request.getParameter("copy.policy.template.label");
```

```
  BizfacadeFactory fac = BizfacadeFactory.instance();
  PolicyTemplateFacade policyTemplateFacade =
   fac.getPolicyTemplateFacade();
  String newID =
   policyTemplateFacade.copyTemplate(resourceID, label);
}

private String uploadFileData()
 throws Stop, GridFailureException, FileUploadException,
 IOException, XPathExpressionException {
  // Check that we have a file upload request
  DiskFileUpload upload = new DiskFileUpload();
  List items = upload.parseRequest(request);

  FileItem templateFile = null;
  String   fileType = null;

  for (int i = 0; i < items.size(); i++) {
    FileItem item = (FileItem) items.get(i);
    if ( item.isFormField()) {
      if (item.getFieldName().equals("add.policy.template"))
        fileType = item.getFieldName();
    }else{
      if (templateFile != null)
        throw new RuntimeException("Multiple files uploaded: "
                    + item.getFieldName()
                    + " and " + templateFile.getFieldName());
      templateFile = item;
    }
  }
  if (templateFile == null ) {
    throw new Stop("No form data in submission (this may be due "
                  +"to getting logged out before submitting).");
  }
  if (templateFile.getSize() == 0){
    throw new Stop("Specified file not found or could not be opened."
                  +"<br>Please specify a valid xml file to upload.");
  }
  Document xmlDoc = FileUtils.parseXML(templateFile.getInputStream());
  if (fileType.equals("add.policy.template")){
    try {
      policyService.createPolicyTemplateFromDoc(xmlDoc);
    } catch (GridFailureException gex) {
      throw new Stop("Error with Policy template upload: ", gex);
    }
    return "Policy Template added";
  }
  return "";
}
```

```
/**
 * Deletes a policy template. First checks if the policy template is
 * associated with an sla template and if not it deletes the policy
 * template.
 * @param resourceId The id of the policy template to be deleted
 * @throws GridFailureException
 */
private void deletePt(String resourceId) throws GridFailureException
{
  SessionFactory factory = SingletonSessionFactory.getFactory();
  Session hibSession = factory.openSession();

  try
  {
    Transaction tx = hibSession.beginTransaction();

    PolicyTemplate pt =
     (PolicyTemplate) hibSession.get(PolicyTemplate.class, resourceId);

    List result =
     hibSession.createQuery("select slaTemplate.id "
                           +"from SLATemplate slaTemplate "
                           + "where slaTemplate.policyTemplate.id "
                           + "= :ptID")
                            .setString("ptID", resourceId).list();

    if (result.size()>0){
      throw (new GridFailureException
       ("Policy Template in use. Cannot delete it"));
    }

    for (Violation v: pt.getViolations()){
      hibSession.delete(v);
    }

    pt.setViolations(new Violation[0]);

    for (Range r: pt.getTotalViolations())
      hibSession.delete(r);

    pt.setTotalViolations(new Range[0]);

    for (Range r: pt.getUsage())
      hibSession.delete(r);

    pt.setUsage(new Range[0]);

    tx.commit();
```

```
  }
  finally
  {
    hibSession.close();
  }
  policyService.deletePolicyTemplate(resourceId);
}

/**
 * Shows the list with the uploaded policy templates in the
 * management webpage.
 * @throws IOException
 */
public void showPolicyTemplateList(Writer out) throws IOException
{
  String[] PolicyTresources = pdp.getResources
   (PolicyTemplate.POLICY_TEMPLATE_RESOURCE_TYPE, null, null);
  String[] states = new String[PolicyTresources.length];
  for (int i=0 ; i<PolicyTresources.length ; ++i)
    states[i] = pdp.getProcessState(PolicyTresources[i]);

  SessionFactory factory = SingletonSessionFactory.getFactory();
  Session hibSession = factory.openSession();

  try
  {
    out.write("<table><tr><th>Policy Template ID</th><th>Label</th>"
              + "<th>State</th><th>Actions</th></tr>");
    int shown = 0;
    for (int i=0 ; i<PolicyTresources.length ; ++i)
    {
      shown++;
      String resource = PolicyTresources[i];
      String state = states[i];

      PolicyTemplate pt =
       (PolicyTemplate) hibSession.get(PolicyTemplate.class, resource);

      if(pt==null){
        out.write("<tr><td colspan='4'>Could not find Policy "
                  + "Template " + resource + "</td></tr>") ;
        continue;
      }

      String resourceID = pt.getResourceID();

      //check and change status
      PolicyServiceImpl policyService = new PolicyServiceImpl();
      policyService.checkPtState(pt);
```

```
        hibSession.saveOrUpdate(pt);

        thesisLog.info("state type = "+pt.getState());
        thesisLog.info("state pdp = "+state);

        out.write("<tr id='row." + editorID() + "'>\n");
        //click on the resource id to see the template's details.
        out.write("<td><a href=\"policyTemplate.jsp?resource="
                  + resourceID + "\">" + resourceID + "</a></td>");
        out.write("<td>"+pt.getLabel()+"</td>");
        out.write("<td>"+pt.getState()+"</td>");
        out.write("<td>");
        out.write("<a class='button' href='javascript:copyPt(\""
                  + editorID() + "\", \"" + resourceID
                  + "\", \"Copy of " + pt.getLabel()
                  + "\")'>Copy</a> ");
        out.write("<a class='button' "
                  + "href='javascript:showDeletePtWarning(\""
                  + editorID() + "\", \"" + resourceID
                  + "\")'>Delete</a> ");
        out.write("</td>");
        out.write("</tr>\n");

        editorIDCount++;
      }
      if (shown == 0) {
        out.write("<tr><td colspan='4'>"
                  +"No Policy Templates</td></tr>");
      }
      out.write("</table>\n");

      editorIDCount++;
    }
    finally
    {
      hibSession.close();
    }
}

/**
 * Prints a violation
 * @param v The violation to print
 * @return The string with html result
 */
private String makeViolationHTML(Violation v){
  String output=v.prettyPrint(true);
  return "<tr><td>"  + v.getMetric().getURI() + "</td><td>"
   + output + "</td><td>" + v.getAction() + "</td></tr>";
}
```

```
private String makeRangeHTML(Range r){
  String output=r.prettyPrint(true);
  return output;
}

/**
 *
 * @param policyID The id of the policy template to print
 * @return The string with the html result
 * @throws IOException
 * @throws Stop
 */
public String printPolicyTemplate(String policyID)
 throws IOException, Stop{
  Session hibSession =
   SingletonSessionFactory.openThreadLocalSession();
  String output = "";
  String state = pdp.getProcessState(policyID);
  try{
    PolicyTemplate pt =
     policyService.getPolicyTemplate(hibSession, policyID);

    PolicyServiceImpl policyService = new PolicyServiceImpl();
    policyService.checkPtState(pt);
    thesisLog.info("state type = "+pt.getState());
    thesisLog.info("state pdp = "+state);

    hibSession.saveOrUpdate(pt);
    output+= "<h2>" + pt.getLabel() + " (" + state + ")</h2>";
    output+= "<div class='section'>\n";
    output+= "<table>\n";
    output+= "<tr><th>Field</th><th>Value</th></tr>\n";
    output+= "<tr><th>Resource ID</th><td>"
            + pt.getResourceID() + "</td></tr>\n";
    output+= "<tr><th>Description</th><td>";
    output+=
     (pt.getDescription() != null ? pt.getDescription() : "")
            + "</td></tr>\n";
    output+= "<tr><th>Start Time</th><td>";
    output+=
     (pt.getStartTime() != null ? pt.getStartTime().getTime() : "")
            + "</td></tr>\n";
    output+= "<tr><th>End Time</th><td>";
    output+=
     (pt.getEndTime() != null ? pt.getEndTime().getTime() : "")
            + "</td></tr>\n";
    output+= "<tr><th>State</th><td>";
    output+= pt.getState() + "</td></tr>\n";
```

```
output+= "</table>\n";

String outputV = "";
try{
  outputV+="<table>"
    +"<tr><th colspan='3'>Actions upon Violations</th></tr>"
      +"<tr><th>Metric</th><th>Type</th><th>Action</th></tr>";
  TreeSet<Violation> sortedViolations = new TreeSet<Violation>();
  sortedViolations.addAll(Arrays.asList(pt.getViolations()));
  int shown = 0;
  for (Violation v : sortedViolations) {
    shown++;
    outputV+= makeViolationHTML(v);
  }
  if (shown == 0) {
    outputV+="<tr><td>No Rules for Actions upon Violations"
            +"</td></tr>";
  }
  outputV+="</table>";
}catch(Exception e){
  throw new
   GridFailureException("Error in displaying violations", e);
}
output+=outputV;

String outputTV = "";
try{
  outputTV+="<table>"
    +"<tr><th colspan='2'>Total violations for SLA (last "
    + pt.getNumOfBillingPeriodsForViolations()
    +" billing periods)</th></tr>";
  TreeSet<Range> sortedTerms = new TreeSet<Range>();
  sortedTerms.addAll(Arrays.asList(pt.getTotalViolations()));
  int shown = 0;
  for (Range r : sortedTerms) {
    shown++;
    outputTV+= "<tr><td>"+makeRangeHTML(r)+"</td></tr>";
  }
  if (shown == 0) {
    outputTV+="<tr><td>No Total Violations term</td></tr>";
  }
  outputTV+="</table>";
}catch(Exception e){
  throw new GridFailureException("Error in "
           +"displaying pricing terms", e);
}
log.info("Kate: outputTV = "+outputTV);
output+=outputTV;
```

```
        String outputUsage = "";
        try{
          outputUsage+="<table>"
                       +"<tr><th>Usage (last "
                       +pt.getNumOfBillingPeriodsForUsage()
                       +" billing periods)</th></tr>";
          TreeSet<Range> sortedUsage = new TreeSet<Range>();
          sortedUsage.addAll(Arrays.asList(pt.getUsage()));
          int shown = 0;
          for (Range r : sortedUsage) {
            shown++;
            outputUsage+= "<tr><td>"+makeRangeHTML(r)+"</td></tr>";
          }
          if (shown == 0) {
            outputUsage+="<tr><td>No usage</td></tr>";
          }
          outputUsage+="</table>";
        }catch(Exception e){
          throw new GridFailureException("Error in displaying"
                                         + "permitted services", e);
        }
        output+=outputUsage;

      } finally {
        SingletonSessionFactory.closeThreadLocalSession();
      }

      output += "</div>";
      return output;
    }
}
```

B′.4 Η κλάση PolicyServiceImpl.java

```
package uk.ac.soton.itinnovation.grid.service.policy;

public class PolicyServiceImpl{

  private Logger log = Logger.getLogger(getClass());
  private Logger thesisLog = Logger.getLogger("thesis");
  private PDP pdp =
   (PDP) ImplementationFactory.getSingletonInstance(PDP.class);

  public PolicyServiceImpl (){
    try {
      ensurePoliciesDeployed();
    } catch (GridFailureException e) {
      e.printStackTrace();
```

```
  }
}

public void ensurePoliciesDeployed() throws GridFailureException {
  PBACUtils.ensureDeployed(
   PolicyTemplate.POLICY_TEMPLATE_RESOURCE_TYPE,
   "policy-template-policy.xml");
}

public String createPolicyTemplateFromDoc(Document xmlDoc)
 throws GridFailureException {
  PolicyTemplate pt = null;
  Element root = xmlDoc.getDocumentElement();
  try {
    pt = AxisUtils.elementToObject(root, PolicyTemplate.class);
  } catch (RuntimeException rex) {
    try {
      pt = getOldPolicyTemplateFromDoc(xmlDoc);
    } catch (XPathExpressionException xex) {
      throw new
       GridFailureException("Error parsing Policy template", xex);
    }
  }
  PolicyTemplateFacade ptFacade =
   BizfacadeFactory.instance().getPolicyTemplateFacade();
  String resourceID = ptFacade.saveNewTemplate(pt);
  return resourceID;
}

/**
 * Parses an xml file and creates a new Policy Template.
 * @param xmlDoc The xml file to parse
 * @return The Policy Template created from the xml file
 * @throws GridFailureException
 * @throws XPathExpressionException
 */
private PolicyTemplate getOldPolicyTemplateFromDoc(Document xmlDoc)
 throws GridFailureException, XPathExpressionException {

  if(!xmlDoc.getDocumentElement().getTagName()
     .equals("policyTemplate"))
    throw new GridFailureException(
        + "Incorrect Policy Template document."
        + " Document root element must be 'policyTemplate' not '"
        + xmlDoc.getDocumentElement().getTagName()+"'");

  PolicyTemplate policyt = new PolicyTemplate();

  log.info("Making PolicyTemplate from xml document.");
```

```
XPathFactory  Xfactory=XPathFactory.newInstance();
XPath xPath = Xfactory.newXPath();
String label    = getTextContent(xPath, "/policyTemplate/label",
                  xmlDoc,XPathConstants.NODE);
String desc     = getTextContent(xPath,
                  "/policyTemplate/description",
                  xmlDoc,XPathConstants.NODE);
policyt.setLabel(label);
policyt.setDescription(desc);
GregorianCalendar startTime = new GregorianCalendar(
    Integer.parseInt(getTextContent(xPath,
                     "/policyTemplate/startTime/year",
                     xmlDoc,XPathConstants.NODE)),
    Integer.parseInt(getTextContent(xPath,
                     "/policyTemplate/startTime/month",
                     xmlDoc,XPathConstants.NODE)),
    Integer.parseInt(getTextContent(xPath,
                     "/policyTemplate/startTime/dayOfMonth",
                     xmlDoc,XPathConstants.NODE)),
    Integer.parseInt(getTextContent(xPath,
                     "/policyTemplate/startTime/hour",
                     xmlDoc,XPathConstants.NODE)),
    Integer.parseInt(getTextContent(xPath,
                     "/policyTemplate/startTime/minute",
                     xmlDoc,XPathConstants.NODE))
  );
startTime.setLenient(false);
policyt.setStartTime(startTime);

GregorianCalendar endTime = new GregorianCalendar(
    Integer.parseInt(getTextContent(xPath,
                     "/policyTemplate/endTime/year",
                     xmlDoc,XPathConstants.NODE)),
    Integer.parseInt(getTextContent(xPath,
                     "/policyTemplate/endTime/month",
                     xmlDoc,XPathConstants.NODE)),
    Integer.parseInt(getTextContent(xPath,
                     "/policyTemplate/endTime/dayOfMonth",
                     xmlDoc,XPathConstants.NODE)),
    Integer.parseInt(getTextContent(xPath,
                     "/policyTemplate/endTime/hour",
                     xmlDoc,XPathConstants.NODE)),
    Integer.parseInt(getTextContent(xPath,
                     "/policyTemplate/endTime/minute",
                     xmlDoc,XPathConstants.NODE))
  );
endTime.setLenient(false);
policyt.setEndTime(endTime);
```

```
//check if the start time is < than the end time
if (policyt.getStartTime().compareTo(policyt.getEndTime())>=0){
  throw new
   GridFailureException("Start time must be before the end time.");
}
//check if the policy template is expired when created.
GregorianCalendar now = new GregorianCalendar();
if (now.compareTo(policyt.getEndTime())>=0){
  throw new
   GridFailureException("The policy template is already expired.");
}

//Add Violations
ArrayList<Violation> vList = new ArrayList<Violation>();

NodeList vNodeList =
 (NodeList)xPath.evaluate("/policyTemplate/violations/violation",
                          xmlDoc,XPathConstants.NODESET);
vList = getViolationsFromNodeList(vNodeList);

policyt.setViolations(vList.toArray(new Violation[]{}));

//Add range for total violations
    ArrayList<Range> totalViolationsList = new ArrayList<Range>();

NodeList tvNodeList =
 (NodeList)xPath.evaluate(
                "/policyTemplate/history/totalViolations/range",
                 xmlDoc,XPathConstants.NODESET);
totalViolationsList = getRangesFromNodeList(tvNodeList);

policyt.setTotalViolations(totalViolationsList.toArray(new Range[]{}));

//Set billing periods
String numOfBillingPeriodsStrForViolations =
 (String)(xPath.evaluate("/policyTemplate/history/totalViolations"
                         +"/@numOfBillingPeriods",xmlDoc,
                         XPathConstants.STRING));
Integer numOfBillingPeriodsForViolations =
 new Integer(numOfBillingPeriodsStrForViolations);
policyt.setNumOfBillingPeriodsForViolations
 (numOfBillingPeriodsForViolations);

//Add range for usage
    ArrayList<Range> usageList = new ArrayList<Range>();

NodeList usageNodeList =
 (NodeList)xPath.evaluate("/policyTemplate/history/usage/range",
```

```
                                    xmlDoc, XPathConstants.NODESET);
    usageList = getRangesFromNodeList(usageNodeList);

    policyt.setUsage(usageList.toArray(new Range[]{}));

    //Set billing periods
    String numOfBillingPeriodsStr =
     (String)(xPath.evaluate("/policyTemplate/history/usage/"
                             +"@numOfBillingPeriods",
                             xmlDoc, XPathConstants.STRING));
    Integer numOfBillingPeriods = new Integer(numOfBillingPeriodsStr);
    policyt.setNumOfBillingPeriodsForUsage(numOfBillingPeriods);

    log.info("makePolicyTemplate " +policyt);
    return policyt;
  }

  private String getTextContent(XPath xPath,
   String query, Document xmlDoc, QName type) {
    try {
      return XMLUtils.getChildCharacterData(
       (Element) xPath.evaluate(query, xmlDoc, type));
    } catch (Exception ex) {
      throw new RuntimeException(ex);
    }
  }

  private String getTextContent(XPath xPath,
   String query, Element element, QName type) {
    try {
      return XMLUtils.getChildCharacterData(
       (Element) xPath.evaluate(query, element, type));
    } catch (Exception ex) {
      throw new RuntimeException(ex);
    }
  }

  /**
   * Creates a list of Ranges from a collection of nodes
   * that in the xml file represent a list of ranges. In
   * a policy's xml file, there are two different list of
   * ranges: <totalViolations> and <usage>.
   * @param rNodeList
   * @return
   * @throws GridFailureException
   * @throws XPathExpressionException
   */
  private ArrayList<Range> getRangesFromNodeList(NodeList rNodeList)
   throws GridFailureException, XPathExpressionException {
```

```
XPathFactory  Xfactory=XPathFactory.newInstance();
XPath xPath = Xfactory.newXPath();
ArrayList<Range> rList = new ArrayList<Range>();
for(int i=0 ; i < rNodeList.getLength() ; i++) {

  Element element = (Element) rNodeList.item(i);
  Double lowerBound =
   Double.parseDouble(getTextContent(xPath, "lowerBound",
                                     element,XPathConstants.NODE));
  Double upperBound =
   Double.parseDouble(getTextContent(xPath, "upperBound",
                                     element,XPathConstants.NODE));
  Double discount =
   Double.parseDouble(getTextContent(xPath, "discount",
                                     element,XPathConstants.NODE));

  //check if discount is between 0-100
  if (discount<0 || discount>100){
    throw new
     GridFailureException
      ("Discount values must be between 0 and 100.");
  }

  //check if lower bound is >=0
  if (lowerBound<0){
    throw new
     GridFailureException("Lower bound must not be negative.");
  }

  //check if lower bound is <= than upper bound
  if (upperBound!=-1 && lowerBound>upperBound){
    throw new
     GridFailureException
      ("Lower bound must be <= than the upper bound.");
  }

  Range r = new Range();
  r.setLowerBound(lowerBound);
  r.setUpperBound(upperBound);
  r.setDiscount(discount);

  rList.add((Range)r);

  for (int j=0;j<i;j++){
    if (rList.get(i).overlapping(rList.get(j))){
      throw new
       GridFailureException("Ranges must not be overlapping.");
    }
  }
```

```
    }
    return rList;
  }

  /**
   * Creates a list of Violations from the <violations> element
   * of the policy's xml file.
   * @param violationNodeList The NodeList that
   *        represents the <violations> element
   * @return A ArrayList with the Violations
   * @throws GridFailureException
   * @throws XPathExpressionException
   */
  private ArrayList<Violation> getViolationsFromNodeList
   (NodeList violationNodeList)
   throws GridFailureException, XPathExpressionException {
    XPathFactory  Xfactory=XPathFactory.newInstance();
    XPath xPath = Xfactory.newXPath();
    ArrayList<Violation> vList = new ArrayList<Violation>();
    for(int i=0 ; i < violationNodeList.getLength() ; i++) {

      Element element = (Element) violationNodeList.item(i);
      String action = getTextContent(xPath,"action",
                                     element, XPathConstants.NODE);
      String type = getTextContent(xPath,"type", element,
                                   XPathConstants.NODE);
      Violation v = new Violation();
      v.setMetric(getMetricFromDoc((NodeList) xPath.evaluate("metric",
                                   element, XPathConstants.NODESET)));
      v.setAction(action);
          v.setType(type);
          vList.add(v);

      for (int j=0;j<i;j++){
        if (vList.get(i).getMetric().getURI().equals
         (vList.get(j).getMetric().getURI() )
         && vList.get(i).getType().equals(vList.get(j).getType()) ){
          throw new
           GridFailureException("Different violations on the "
                                +"same metric and with the same "
                                +"type are not allowed.");
        }
      }
    }

    return vList;
  }

  /**
```

```
 * Creates a Metric from a collection of nodes that represent
 * a metric in an xml document (namely the <metric> element).
 * @param elements The nodelist that represents the metric
 * @return The Metric created
 * @throws XPathExpressionException
 */
private Metric getMetricFromDoc(NodeList elements)
 throws XPathExpressionException {
  Element element = (Element) elements.item(0);
  XPathFactory Xfactory = XPathFactory.newInstance();
  XPath xPath = Xfactory.newXPath();

  Metric metric = new Metric();
  String uri = getTextContent
   (xPath, "uri", element, XPathConstants.NODE);
  String description = getTextContent
   (xPath, "description/description",
    element, XPathConstants.NODE);
  String pluralDescription = getTextContent
   (xPath, "description/plural",
    element, XPathConstants.NODE);
  String instantaneousDescription = getTextContent
   (xPath, "description/instantaneous",element,XPathConstants.NODE);
  String cumulativeDescription = getTextContent
   (xPath, "description/cumulative", element, XPathConstants.NODE);
  String instantaneousUnits = getTextContent
   (xPath, "units/instantaneous", element, XPathConstants.NODE);
  String cumulativeUnits = getTextContent
   (xPath, "units/cumulative", element, XPathConstants.NODE);
  String metricType = (String)
   (xPath.evaluate("@type", element,
                   XPathConstants.STRING));
  String unitType = (String)
   (xPath.evaluate("units/@type", element,
                   XPathConstants.STRING));

  metric.setURI(uri);
  metric.setDescription(description);
  metric.setPluralDescription(pluralDescription);
  metric.setInstantaneousDescription(instantaneousDescription);
  metric.setCumulativeDescription(cumulativeDescription);
  metric.setInstantaneousUnits(instantaneousUnits);
  metric.setCumulativeUnits(cumulativeUnits);
  metric.setType(metricType);
  metric.setUnitType(unitType);

  return metric;
}
```

```
/**
 * Retrieves the policy template with the specified id.
 * @param session
 * @param resourceID The id of the policy template to retrieve.
 * @return The policy template with the specified id
 * @throws GridFailureException
 */
public PolicyTemplate getPolicyTemplate(Session session,
                                        String resourceID)
 throws GridFailureException {
  PolicyTemplate pt =
   (PolicyTemplate) session.get(PolicyTemplate.class, resourceID);
  if(pt == null)
    throw new
     GridFailureException("Could not find Policy Template "
                          + resourceID);
  return pt;
}

/**
 * Changes the policy template's state from inactive to active.
 * @param resourceID The id of the policy template to activate
 * @throws GridFailureException
 */
public void activatePolicyTemplate(String resourceID)
 throws GridFailureException{
  try{
    pdp.lockForAdmin(resourceID,"activate");
    //pdp.signal(resourceID, "activate");

    Session session = SingletonSessionFactory.openThreadLocalSession();

    try{
      pdp.signal(resourceID, "activate");
      Transaction tx = session.beginTransaction();
      PolicyTemplate pt = getPolicyTemplateInternal(resourceID);
      thesisLog.info("3 setting state of policy template to active");
      pt.setState(PolicyTemplateState.ACTIVE);
      session.saveOrUpdate(pt);
      tx.commit();
    }
    catch(GridFailureException e) {
      if(session.getTransaction() != null)
        session.getTransaction().rollback();
      throw e;
    }
    catch(Throwable e) {
      if(session.getTransaction() != null)
        session.getTransaction().rollback();
```

```
        throw new GridErrorException("", e);
      }
      finally {
        SingletonSessionFactory.closeThreadLocalSession();
      }
    }finally{
      pdp.unlock(resourceID);
    }
  }

  /**
   * Changes the policy template's state from active to expired.
   * @param resourceID The id of the policy template
   * @throws GridFailureException
   */
  public void expiredPolicyTemplate(String resourceID)
   throws GridFailureException{
    try{
      pdp.lockForAdmin(resourceID,"expires");
      //pdp.signal(resourceID, "expires");

      Session session =
       SingletonSessionFactory.openThreadLocalSession();

      try{
        pdp.signal(resourceID, "expires");
        Transaction tx = session.beginTransaction();
        PolicyTemplate pt = getPolicyTemplateInternal(resourceID);
        pt.setState(PolicyTemplateState.EXPIRED);
        session.saveOrUpdate(pt);
        tx.commit();
      }
      catch(GridFailureException e) {
        if(session.getTransaction() != null)
          session.getTransaction().rollback();
        throw e;
      }
      catch(Throwable e) {
        if(session.getTransaction() != null)
          session.getTransaction().rollback();
        throw new GridErrorException("", e);
      }
      finally {
        SingletonSessionFactory.closeThreadLocalSession();
      }
    }finally{
      pdp.unlock(resourceID);
    }
  }
```

```
/**
 * Deletes the policy template with the given id.
 * @param resourceID The id of the policy template
 *        that will be deleted
 * @throws GridFailureException
 */
public void deletePolicyTemplate(String resourceID)
 throws GridFailureException{
  try{
    pdp.lockForAdmin(resourceID,"finish");
    pdp.signal(resourceID, "finish");
  }finally{
    pdp.unlock(resourceID);
  }
}

/**
 * Requires a Hibernate session and an open transaction.
 */
private PolicyTemplate getPolicyTemplateInternal(String resourceId) {
  PolicyTemplateDAO policyTempalteDao =
   DAOFactory.instance().getPolicyTemplateDAO();
  PolicyTemplate pt =
   policyTempalteDao.findByResourceId(resourceId);
  if (pt == null) {
    throw new GridErrorException("Can't find policy template'"
                                 + resourceId + "'!");
  }
  return pt;
}

/**
 * Checks if the state of the policy template should
 * be changed. This happens if the state is ACTIVE
 * but the template's end time has passed or if the
 * state is INACTIVE but the current time is after
 * the template's start time.
 * @param pt The policy template to check
 */
public void checkPtState (PolicyTemplate pt){
  //should be transitioned to expired
  if (pt.getState().equals(PolicyTemplateState.ACTIVE)
      && pt.hasExpired()){
    try {
      this.expiredPolicyTemplate(pt.getResourceID());
    } catch (GridFailureException e) {
      log.info("Could not set pt to expired state");
      thesisLog.info("could not set pt to expired state");
```

```
        e.printStackTrace();
      }
    }
    //is inactive and should be transitioned to active
    else if (pt.getState().equals(PolicyTemplateState.INACTIVE)
            && !pt.isInactive()){
      try {
        this.activatePolicyTemplate(pt.getResourceID());
      } catch (GridFailureException e) {
        log.info("Could not set pt to active state");
        thesisLog.info("could not set pt to active state");
        e.printStackTrace();
      }
    }
  }
}
```

Β′.5 Η κλάση SLA.java

Β′.5.1 Η μέθοδος sendBills

```
//This is the method that finally bills the account
//it is called at the end of each billing period
//check here if the policy template specifies a discount
public void sendBills(List<PeriodBill> bills)
 throws GridFailureException {
   SLAServiceImpl slaService =
    (SLAServiceImpl) ImplementationFactory.getSingletonInstance
                                          (SLAServiceSOAP.class);
   String resourceID = this.getResourceID();
   TradeAccountConversation account =
    slaService.getAccount(this.getAccount());

   //Katerina Marazopoulou: code added
   //check if the policy template specifies a discount
   PolicyTemplate pt = slat.getPolicyTemplate();

   if(pt!=null){
     //every time we use a policy template,
     //we must check if it has expired
      PolicyServiceImpl policyService = new PolicyServiceImpl();
      policyService.checkPtState(pt);

      if (!pt.getState().equals(PolicyTemplateState.ACTIVE)){
         pt = null; //the pt is expired or inactive, so ignore it
      }
   }
```

```
Double discount = 1.0;  //total  discount,
                        //to be  multiplied directly  with the amount
Double discount1 = 0.0; //%discount due   to violations
Double discount2 = 0.0; //%discount due   to usage
if (pt!=null){ //there  is a policy template
   //compute   the   usage for   this billing period
   BigDecimal curPrice  =  BigDecimal.ZERO;
   for   (PeriodBill bill: bills) {
      thesisBill.info("Bill: " + bill);
      curPrice = curPrice.add(bill.getPrice());
   }
   this.getUsagePerBP().add(curPrice);

   if (pt.getTotalViolations()!=null){
      //there  is a policy template and tells something
      //about total violations

      //take into consideration  the   sla's total violations
      if (pt.getNumOfBillingPeriodsForViolations().equals
                              (this.violationsPerBP.size())){
         //it's time to apply the discount
         discount1   =
          pt.getDiscountForViolations(this.takeViolationsForBP());
         //we clear the list  so we can   start counting
         //violations for  the   next set of specified
         //billing   periods
         violationsPerBP.clear();
      }

      //after  computing   the   discount,
      //create the next element  for   the   violation's list
      this.getViolationsPerBP().add(new   Integer(0));
   }

   if (pt.getUsage()!=null){
      //there  is a policy template and tells something
      //about total amount billed to   the   user

      //take into consideration  the   total amount
      //billed the   last x billing periods
      if (pt.getNumOfBillingPeriodsForUsage().equals
                                 (usagePerBP.size())){
         discount2   =
          pt.getDiscountForUsage(this.takeBilledAmountForBP());
         usagePerBP.clear();
      }
   }
   //find the total discount
   discount = (100   -  discount1)*(100   -  discount2)/10000;
```

```
    }
    //Katerina Marazopoulou: end of  code added

    if (account != null) {
       try   {
          for   (PeriodBill bill: bills) {
             logger.debug("Bill:  " +  bill);
             BigDecimal price = bill.getPrice();
             //Katerina Marazopoulou: code added
             //compute   the   final bill,
             //taking the discount   into consideration
             price =  price.multiply(new BigDecimal(discount));
             //Katerina Marazopoulou: end of  code added
             String message = "Period "
                            + isoDate.format(bill.getStart().getTime())
                            + " to "
                            +   isoDate.format(bill.getEnd().getTime())
                            +   ":\n"
                            +   bill.getDescription();
             account.bill(null, price,  bill.getCurrency(),
                          resourceID,   message);
             this.setLastBillTime(bill.getEnd());
          }
       }  catch (RemoteException ex) {
          if (AxisUtils.remoteExceptionIs(ex,
            ActionCurrentlyUnavailableException.class))   {
             // then   the   account  may   be closed   or suspended
             //etc and we   must suspend the SLA
             String msg = "Cannot bill  to account " +
                ConversationID.getURLReferenceFromEPR(this.getAccount())
                + "   because  bill action is unavailable.    SLA is  "
                +  this.getResourceID();
             throw new   GridFailureException(msg,  ex);
          }  else {
             throw new   GridErrorException("Failed to bill account "
              + ConversationID.getURLReferenceFromEPR(this.getAccount()), ex);
          }
       }
    }
}
```

B′.6 Η κλάση ConstraintManagerImpl.java

B′.6.1 Η μέθοδος getActionsForSLA

```
private List<ManagementAction> getActionsForSLA(SLA sla) {
   UsageReportFacade usageFacade =
    BizfacadeFactory.instance().getUsageReportFacade();
```

```
List<ManagementAction> actions = new ArrayList<ManagementAction>();
Set<Activity> activitiesToHandle = new HashSet<Activity>();
Map<SLA, List<Activity>> activitiesBySLA =
 new HashMap<SLA, List<Activity>>();
Metric metric;
double amount;

GregorianCalendar now = new GregorianCalendar();

// find which (if any) of the SLA's constraints have been breached
Set<Constraint> breached = new HashSet<Constraint>();

for (Constraint constraint: sla.getConstraints()) {
   metric = constraint.getMetric();
   try {

      if (constraint.type == UsageType.INSTANTANEOUS) {
         amount = usageFacade.getRate(sla.getId(),
                                      metric,
                                      now.getTimeInMillis());
      } else {  // UsageType.CUMULATIVE
         amount = usageFacade.getUsageInPeriod(sla.getId(),
                     metric,
                     ((GregorianCalendar) constraint.
                     findTimeSpan(now)[0]).getTimeInMillis(),
                     now.getTimeInMillis());
      }
      if (constraint.hasBreach(amount)) {
         breached.add(constraint);
      }
   } catch (IllegalArgumentException ex) {
      // means that the constraint is not valid at this time
      //or we have had no usage on that metric: just ignore
   }
}

//Katerina Marazopoulou: code added

SLATemplate slat = sla.getSlat();
PolicyTemplate pt = null;

pt = slat.getPolicyTemplate();
if (pt!=null){
   //every time we use a policy template,
   //we must check if it has expired
   PolicyServiceImpl policyService = new PolicyServiceImpl();
   policyService.checkPtState(pt);

   if (!pt.getState().equals(PolicyTemplateState.ACTIVE)){
```

```
            pt = null; //the pt is expired or inactive, so ignore it
        }
    }

    if(pt!=null){ //there is a policy template
        //count violations
        Set<Constraint> a = new HashSet<Constraint>();
        a.addAll(sla.getLastBreached());
        //a contains the violations that were breached the last time
        Set<Constraint> b = new HashSet<Constraint>();
        b.addAll(breached);
        //b contains the violations that were breached this time
        a.retainAll(breached);
        //now a is the intersection of the two sets
        b.removeAll(a);
        //now b contains only the new constraints
        int numOfNewViolations = b.size();

        if (numOfNewViolations>0){
            sla.increaseViolationsForSLA(numOfNewViolations);
        }
    }

    //renew the last breached set of the sla
    //whether there is a policy or not
    try {
        Session currentSession =
         SingletonSessionFactory.getFactory().getCurrentSession();
        sla.setLastBreached(breached);
        currentSession.saveOrUpdate(sla);
    } catch (GridFailureException e) {
        e.printStackTrace();
    }

    //a set that will contain the activities that need to be notified
    Set<Activity> checkIfToNotify = new HashSet<Activity>();
    boolean notificationFound = false;
    //Katerina Marazopoulou: end of code added

    for (Constraint constraint: breached) {
        metric = constraint.getMetric();

        String debug = "";

        Iterator<Activity> iter = sla.iterateActivities();

        if (constraint.type == UsageType.CUMULATIVE) {
            // we must kill all activities in the SLA with
            //a non-zero rate for the constrained metric
```

```
    while (iter.hasNext()) {
        Activity activity = iter.next();
        if (usageFacade.getRate(activity.getId(),
                                metric,
                                now.getTimeInMillis()) > 0) {
            activitiesToHandle.add(activity);
            debug += "\n" + activity.getResourceID();
        }
    }
} else {  // it limits rate
    // We must kill enough activities in the SLA with
    // a non-zero rate for the constrained metric.
    // Possible strategies:
    // 1. Sort by activity start time and add
    // the newest to the kill set till we have enough.
    // 2. Sort by usage of the metric and add
    // the smallest to the kill set till we have enough.
    // 3. ...

    // This is choice 1:

    // choose sufficient 'candidates' to bring the rate down
    //double slaRate = sla.getRate(metric, now);
    double slaRate = usageFacade.getRate(sla.getId(),
                        metric, now.getTimeInMillis());

    while (iter.hasNext()) {
        Activity activity = iter.next();
        double activityRate = usageFacade.getRate(activity.getId(),
                                    metric, now.getTimeInMillis());

        if (activityRate > 0) {
            activitiesToHandle.add(activity);
            debug += "\n" + activity.getResourceID();
            slaRate -= activityRate;
            if (!constraint.hasBreach(slaRate)) {
                break;
            }
        }
    }
}

//Katerina Marazopoulou: code added

if (pt==null){ // no policy template associated,
                //do whatever you were doing before policies
    for (Activity activity: activitiesToHandle) {
        actions.add(new KillActivityAction(activity));
    }
```

```
    }
    else { //there is a policy template

        String action =
         pt.existsViolation(metric.getURI(), constraint.getType());
        if (action!=null){ //violation exists
            if (action.equals("kill")){
                for (Activity activity: activitiesToHandle) {
                    actions.add(new KillActivityAction(activity));
                }
            }
            else if (action.equals("notify")){
                checkIfToNotify.addAll(activitiesToHandle);
                notificationFound = true;
            }
        }
        else {//do whatever you were doing before
            for (Activity activity: activitiesToHandle) {
                actions.add(new KillActivityAction(activity));
            }
        }
    }
} //end of constraint for-loop

if(notificationFound){ //there are activities to notify
    Set<Activity> act1 = new HashSet<Activity>();
    act1.addAll(sla.getActivitiesNotified());
    //act1 contains the activities that were notified the last time
    Set<Activity> act2 = new HashSet<Activity>();
    act2.addAll(checkIfToNotify);
    //act2 contains the activities that need to be notified this time
    act1.retainAll(act2);
    //now act1 is the intersection of the two sets
    act2.removeAll(act1);
    //now act2 contains only the new activities
    int numOfNewToNotify = act2.size();

    if (numOfNewToNotify>0){//there are new activities to notify
        for (Activity activity: act2){
            actions.add(new NotifyActivityAction(activity));
        }
    }
}

//renew the activitiesNotified set
//whether there is a policy template or not
sla.setActivitiesNotified(activitiesToHandle);
//Katerina Marazopoulou: end of code added
```

```
    return actions;
}
```

Β′.6.2 Η μέθοδος startActivity

```
//Katerina Marazopoulou: changed so it throws
//NotificationConstraintBreachException
public void startActivity(SLA sla, Set<Constraint> actConstraints)
 throws PublicConstraintBreachException,
        PrivateConstraintBreachException,
        NotificationConstraintBreachException {
   UsageReportFacade usageFacade =
    BizfacadeFactory.instance().getUsageReportFacade();
   Set<Bound> applicableBounds =
    new HashSet<Bound>(Arrays.asList(
     new Bound[] {Bound.GT, Bound.GE, Bound.EQ}));
   Metric metric;
   GregorianCalendar now = new GregorianCalendar();
  GregorianCalendar startTime;
   GregorianCalendar[] timeSpan;
   double usage, rate;
  Set<Constraint> pConstraints = new HashSet<Constraint>();
  Set<Constraint> breached = new HashSet<Constraint>();

   for (Constraint actConstraint: actConstraints) {
      if (!sla.constrains(actConstraint.getMetric())) {
         continue;  // we don't constrain this metric
      }
      if (!applicableBounds.contains(actConstraint.bound)) {
         continue;
      }
      if (!actConstraint.isIndefinite()) {
         continue;
      }
      pConstraints.add(actConstraint);
  }

   // Our strategy is to check the constraints as they are
   // now and not to look into the future.  So if a minimum
   // rate is required, we check if we have that rate
   // capacity now.  If a minimum usage is required and we
   // have an indefinite constraint on the SLA, we check if
   // the constraint can accomodate the usage taking into
   // account activities up to now.  If a minimum usage is
   // required and the SLA constraint is periodic or fixed
   // then we can make no decision on that basis as we don't
   // know when the usage will occur.

  for (Constraint pConstraint: pConstraints) {
```

```
metric = pConstraint.getMetric();
  startTime = (GregorianCalendar) pConstraint.getStartTime();
  // this start time will always be the right one as we only
  //allow pConstraint to be an indefinite constraint

  for (Constraint slaConstraint: sla.getConstraints()) {
     if (slaConstraint.getMetric().equals(metric)) {
        //So pConstraint and slaConstraint constrain the same metric
        try {
           timeSpan = slaConstraint.findTimeSpan(startTime);
           if (pConstraint.type == UsageType.INSTANTANEOUS
               && pConstraint.hasBreach(0)) {
              if (slaConstraint.type == UsageType.INSTANTANEOUS) {
                 rate = usageFacade.getRate(sla.getId(), metric,
                                    startTime.getTimeInMillis());
                 // basically we need to check if the rateLimit
                 //in pConstraint is less than the rateLimit in
                 //slaConstraint minus this rate
                 if (!slaConstraint.canAccommodate(pConstraint,
                                                   rate)) {
                    breached.add(slaConstraint);
                 }
              } else if (slaConstraint.type ==
                         UsageType.CUMULATIVE) {
                 // We don't know when the usage is
                 // required so we can only reject if we
                 // have 0 usage left at the start time of
                 // the proposed constraint.
                 usage = usageFacade.getUsageInPeriod(sla.getId(),
                       metric,
                       timeSpan[0].getTimeInMillis(),
                       startTime.getTimeInMillis());
                 if (slaConstraint.getLimit() - usage <= 0) {
                    breached.add(slaConstraint);
                 }
              }
           }
           if (pConstraint.type == UsageType.CUMULATIVE
               && pConstraint.hasBreach(0)) {
              if (slaConstraint.type == UsageType.CUMULATIVE) {
                 if (slaConstraint.isIndefinite()) {
                    //check remaining usage in the constraint period
                    //(but the slaConstraint is indefinite so the
                    //"constraint period" is from the beginning)
                    usage = usageFacade.getUsage(sla.getId(), metric,
                                         startTime.getTimeInMillis());
                    //basically we need to check if the usageLimit
                    //in pConstraint is less than the usageLimit
                    //in slaConstraint minus this usage
```

```
                        if (!slaConstraint.canAccommodate(pConstraint,
                                                          usage)) {
                            breached.add(slaConstraint);
                        }
                    } else {
                        // We don't know when the usage is
                        // required so we can only reject if we
                        // have 0 usage left at the start time of
                        // the proposed constraint.
                        usage = usageFacade.getUsageInPeriod(sla.getId(),
                                metric,
                                timeSpan[0].getTimeInMillis(),
                                startTime.getTimeInMillis());
                        if (slaConstraint.getLimit() - usage <= 0) {
                            breached.add(slaConstraint);
                        }
                    }
                } else if (slaConstraint.type ==
                                UsageType.INSTANTANEOUS) {
                    //Non-zero usage is required,
                    // reject if the rate limit is 0
                    if (slaConstraint.getLimit() == 0.0) {
                        breached.add(slaConstraint);
                    }
                }
            }
        } catch (IllegalArgumentException ex) {
        }
    }
}

//Katerina Marazopoulou: code added

SLATemplate slat = sla.getSlat();
PolicyTemplate pt = null;

pt = slat.getPolicyTemplate();

if (pt!=null){
 //every time we use a policy template,
 //we must check if it has expired
 PolicyServiceImpl policyService = new PolicyServiceImpl();
    policyService.checkPtState(pt);

    if (!pt.getState().equals(PolicyTemplateState.ACTIVE)){
       pt = null; //the pt is expired or inactive, so ignore it
    }
}
```

```
boolean publicBreach = false;
boolean found = false;

String pubMsg = "From startActivity public: "
          +"The proposed activity breaches these constraints:";
String privMsg = "From startActivity private: "
          +"The proposed activity breaches these constraints:";
String notification = "From startActivity Notification:";

if (breached.size()>0){

   //renew the breached set
   sla.getLastBreached().addAll(breached);

   if (pt==null){ // no policy template associated,
                  //do whatever you were doing before policies
      pubMsg += ("\n no policy template associate. Kill them all");
      for (Constraint bCon: breached) {
         if (bCon.isPrivate()) {
            privMsg += "\n" + bCon.prettyPrint();
         } else {
            publicBreach = true;
            pubMsg += "\n" + bCon.prettyPrint();
         }
      }
   //If any of the breached constraints are public
   //then we throw a public breach exception
   // hide the private breaches if at all possible
      if (publicBreach) {
         throw new PublicConstraintBreachException(pubMsg);
      } else {
         throw new PrivateConstraintBreachException(privMsg);
      }
   }
   else { //there is a policy template
      pubMsg += ("\n there is a policy template");

      //there is a policy template, so start counting the violations
      sla.increaseViolationsForSLA(breached.size());

      for (Constraint bCon: breached) {

         String action = pt.existsViolation(bCon.getMetric().getURI(),
                                            bCon.getType());
         if (action!=null){ //violation exists
            if (action.equals("kill")){
               pubMsg += ("\n policy says to kill");
```

```
                    if (bCon.isPrivate()) {
                        privMsg += "\n" + bCon.prettyPrint();
                    } else {
                        publicBreach = true;
                        pubMsg += "\n" + bCon.prettyPrint();
                    }
                }
                else if (action.equals("notify")){
                    notification += "\n****This is a notification! "
                                +"The following constraint was breached: "
                                + bCon.prettyPrint();
                    found = true;
                }
            }
            else {//do whatever you were doing before
                pubMsg += ("\n policy says nothing for this constraint,"
                        + " so I'll kill it");

                if (bCon.isPrivate()) {
                    privMsg += "\n" + bCon.prettyPrint();
                } else {
                    publicBreach = true;
                    pubMsg += "\n" + bCon.prettyPrint();
                }
            }
        }
    }

    //If any of the breached constraints are public
    //then we throw a public breach exception
    //hide the private breaches if at all possible
    if (found){
        throw new NotificationConstraintBreachException(notification);
    }
    else if (publicBreach) {
        throw new PublicConstraintBreachException(pubMsg);
    } else {
        throw new PrivateConstraintBreachException(privMsg);
    }
  }
  //Katerina Marazopoulou: end of code added
}
```

Βιβλιογραφία

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

[2] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the Grid: Enabling scalable virtual organization. *International Journal of Supercomputer Applications*, 15(3):200–222, 2001.

[3] W. Leinberger and V. Kumar. Information Power Grid: The new frontier in parallel computing? *IEEE Concurrency*, 7(4):75–84, 1999.

[4] Open Grid Forum. Open Grid Services Architecture (OGSA) [version 1.5]. http://www.ogf.org/documents/GFD.30.pdf.

[5] The World Wide Web Consortium (W3C). http://www.w3.org/.

[6] Jun Yan, Jianying Zhang, Jian Lin, Mohan B. Chhetri, Suk K. Goh, and Ryszard Kowalczyk. Towards autonomous service level agreement negotiation for adaptive service composition. In *10th International Conference on Computer Supported Cooperative Work in Design*, May 2006.

[7] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef. Web Service on demand: WSLA-driven automated management. *IBM Systems Journal, Special Issue on Utility Computing*, 43(1):136–158, March 2004.

[8] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Web Services Agreement Specification (WS-Agreement). Technical report, Global Grid Forum (GGF), May 2004. Available at http://www.gridforum.org/Meetings/GGF11/Documents/draft-ggf-graap-agreement.pdf.

[9] Philipp Masche, Paul Mckee, and Bryce Mitchell. The increasing role of Service Level Agreements in B2B systems. In *WEBIST 2006, Proceedings of the Second International Conference on Web Information Systems and Technologies*, April 2006.

[10] Bryce Mitchell and Paul Mckee. *SLAs A Key Commercial Tool.* IOS Press Amsterdam, 2005. ISBN: 1-58603-563-0.

[11] NextGRID: Architecture for next generation grids. http://www.nextgrid.org/.

[12] UD National HPC Service. http://www.csar.cfs.ac.uk/user_information/grid/grid-middleware.shtml.

[13] GRIA. http://www.gria.org/.

[14] Mike Surridge and Steve Taylor. Experiences with GRIA - Industrial applications on a Web Services Grid. In *First International Conference on e-Science and Grid Computing*, December 2005.

[15] The OGSA-DAI Project. http://www.ogsadai.org.uk/.

[16] Security Assertion Markup Language (SAML). http://en.wikipedia.org/wiki/SAML.

[17] XCA. http://xca.sourceforge.net/.

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

[19] ActiveState Perl. http://www.activestate.com/Products/activeperl/.

[20] ImageMagick binaries. ftp://gd.tuwien.ac.at/pub/graphics/ImageMagick/binaries/.

[21] Perl 5.10.0 documentation. http://perldoc.perl.org/Win32.html.

[22] Apache Maven Project. http://maven.apache.org/.

[23] Eclipse. http://www.eclipse.org/.

[24] Maven Eclipse Plugin. http://maven.apache.org/plugins/maven-eclipse-plugin/.