ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ **Ε**ΠΙΚΟΙΝΩΝΙΩΝ, **Η**ΛΕΚΤΡΟΝΙΚΗΣ **Κ**ΑΙ **Σ**ΥΣΤΗΜΑΤΩΝ **Π**ΛΗΡΟΦΟΡΙΚΗΣ

Υλοποίηση Συστήματος Παροχής Αδειών Χρήσης Λογισμικού σε Περιβάλλον Πολυπλέγματος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΠΑΥΛΟΥ Δ. ΚΡΑΝΑ

Επιβλέπων : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Οκτώβριος 2009

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

Υλοποίηση Συστήματος Παροχής Αδειών Χρήσης Λογισμικού σε Περιβάλλον Πολυπλέγματος

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΠΑΥΛΟΥ Δ. ΚΡΑΝΑ

Επιβλέπων : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 6η Οκτωβρίου 2009.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Θεοδώρα Βαρβαρίγου	Νεκτάριος Κοζύρης	Συμεών Παπαβασιλείου
Καθηγήτρια Ε.Μ.Π.	Αν. Καθηγητής Ε.Μ.Π.	Επίκ. Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2009

(Υπογραφή)

.....

ΚΡΑΝΑΣ ΠΑΥΛΟΣ

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

Περίληψη

Ο σκοπός της διπλωματικής εργασίας ήταν η ανάπτυξη ενός συστήματος ασφαλούς διανομής αδειών χρήσης λογισμικού σε εφαρμογές πελατών που βρίσκονται σε ετερογενή δίκτυα και διαφορετικά υπολογιστικά συστήματα, με χρήση τεχνολογιών πολυπλέγματος, και ταυτόχρονη καταγραφή της χρήσης της εφαρμογής με απότερο σκοπό τη δυνατότητα κοστολόγησης και τιμολόγησης της χρήσης με βάση προσυμφωνημένων συμφωνιών και ποιότητας παροχής υπηρεσιών.

Συγκεκριμένα, έγινε μελέτη των υπάρχουσων αρχιτεκτονικών που επιλύουν το πρόβλημα παροχής αδειών χρήσης λογισμικού από κεντρικό κόμβο σε δικτυακούς χρήστες και καταγραφή των απαιτήσεων που το νέο σύστημα θα πρέπει να ικανοποιεί. Στη συνέχεια έγινε καταγραφή των δυνατοτήτων που παρέχουν οι τεχνολογίες πολυπλέγματος σε συνδιασμό με τη χρήση υπηρεσιοστρεφών αρχιτεκτονικών. Αφού καταλήξαμε στην επιλογή της χρήσης του μεσισμικού GRIA, έγινε η ανάλυση, ο σχεδιασμός και η υλοποίηση του προτεινόμενου συστήματος, καθώς επίσης και ο έλεγχος της ορθής λειτουργίας σε πραγματικό χρόνο με χρήση της εμπορικής εφαρμογής Mathematica και του δημοφιλούς εξυπηρετητή παροχής αδειών λογισμικού FLEXlm server.

Η αρχιτεκτονική που προτείνεται θέτει νέες προκλήσεις στον τρόπο με τον οποίο διανέμονοται οι άδειες λογισμικού σήμερα. Αφού επιλύει τους τεχνολογικούς περιορισμούς που τίθονται από το υπάρχον μοντέλο, κάνοντάς την ιδανική για χρήση εκεί που απαιτείται ύπαρξη ετερογενών δικτύων και ασφαλή επικοινωνία μεταξύ των συμβαλλομένων, θέτει νέα χαρακτηριστικά προς αξιοποίηση που προκύπτουν από τη χρήση τεχνολογιών πολυπλέγματος και τις διαρκώς αναπτυσσόμενες δυνατότητές τους.

Λέξεις Κλειδιά: <<Παροχή Αδειών Χρήσης Λογισμικού, FLEXlm, Τεχνολογίες Διαδικτύου, Διαδικτυακές Υπηρεσίες, Υπηρεσιοστραφές Αρχιτεκτονικές, Τεχνολογίες Πολυπλέγματος, GRIA middleware, security, Public Key Infranstructure, SLA, Java, Apache Tomcat Server >>

Abstract

The scope of this thesis was the delelopment of a new License Management System, which will guarantee the transportation of software licenses, in a secured way, between client applications that are installed in heterogeneous networks and different computing systems using GRID technologies. Furthermore, this system will be capable of monitoring costumer usage and the level of commitments from existing agreements and specified QoS.

More in detail, at first, the existing architectures of transporting software licenses across the network, using the concurrent license model was studied, and the requirements of the new system was analised. Moreover, the abilities that the GRID technologies can offer with the parallel use of service oriented architecture were examined. The proposed system was developed using the GRIA middleware, while its proper funcioning was testing using the commercial application 'Mathematica' and the FLEXlm License server.

The proposed architecture brings new challenges in the way the software licenses are transported nowdays. The usefulness of the system is the ability to access service licenses among heterogeneous networks, using secure communications. Furthermore, the use of GRID technoligies and its developing characteristics gives new benefits to be exploited.

Keywords: <<Software License Management, FLEXlm, Internet Technologies, Web Services, Service Oriented Architecture, GRID Technologies, GRIA middleware, Security, Public Key Infranstructure, SLA, Java, Apache Tomcat Server>>

Ευχαριστίες

Θα ήθελα να ευχαριστήσω την Καθηγήτρια Θεοδώρα Βαρβαρίγου για την ανάθεση της διπλωματικής εργασίας, την παρακολούθηση και την παροχής κατευθύνσεων για την υλοποίησή της.

Επίσης θα ήθελα να κάνω ιδιαίτερη αναφορά στον Κατσαρό Γρηγόριο, υποψήφιο διδάκτορα της σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, και πολύ καλό μου φίλο για την αμέριστη υπομονή και συμπαράστασή του, καθώς με τη συνεχή παρότρυνση από μέρους του συνέβαλε καθοριστικά στην ολοκήρωση αυτού του έργου χαράζοντας τις γενικές γραμμές διεκπεραίωσής του και παρέχοντάς μου πολύτιμες συμβουλές και βοήθεια καθ' όλη τη διάρκεια υλοποίησης του συστήματος.

Πίνακας περιεχομένων

Πίνακας εικόνων

1

Εισαγωγή

1.1 Παροχή Αδειών Χρήσης Λογισμικού

Με τον όρο παροχή αδειών χρήσης λογισμικού εννοείται η μεθοδολογία κατά την οποία εξασφαλίζεται η νόμιμη άδεια χρήσης μια εφαρμογής από έναν χρήστη, ο οποίος επιθυμεί να την εκκινήσει, με σεβασμό στα αποκλειστικά δικαιώματα των πνευματικών της δημιουργών. Για την παροχής της άδειας χρήσης έχουν κατά καιρούς προταθεί διάφορες αρχιτεκτονικές, ανάλογα με τις εκάστοτε τεχνολογίες και τις δυνατότητες που αυτές προσφέρουν. Η πιο σύγχρονη και ευρέως διαδεδομένη εμπορική αρχιτεκτονική καθιστά δυνατή την ταυτόχρονη εκτέλεση ενός τύπου εφαρμογής από απομακρυσμένους χρήστες, σε περιβάλλον τοπικού δικτύου και με χρήση ενός κεντρικού εξυπηρετητή, ο οποίος και αναλαμβάνει να παρέχει τις αντίστοιχες άδειες στους τελικούς χρήστες.

Το συγκεκριμένο μοντέλο παρέχει πληθώρα πλεονεκτημάτων, όπως είναι εμφανές από την περιγραφή του. Επιτρέπει σε έναν μεγάλο αριθμό χρηστών να εκτελούν εφαρμογές χωρίς την ανάγκη ύπαρξης ξεχωριστής άδειας χρήσης ανά μηχάνημα. Ένας κεντρικός εξυπηρετητής έχει συγκεντρωμένες το σύνολο των αδειών και τις διανέμει ανά χρήστη κατ' αίτηση. Ετσι εξασφαλίζεται ότι μια επιχείρηση θα προμηθευτεί μόνο τις απαραίτητες άδειες, ανάλογα με τις ανάγκες της, και όχι ανάλογα με τον αριθμό των χρηστών που δύναται να την εκτελέσσουν.

Παρόλα αυτά, η συγκεκριμένη υλοποίηση εμφανίζει σημαντικά μειονεκτήματα, κυρίως λόγω του περιορισμού της να βρίσκονται όλοι οι συμβαλλόμενοι σε ένα κοινό τοπικό δίκτυο. Κατά αυτόν τον τρόπο, δεν υπάρχει η δυνατότητα επέκτασης και χρήσης της αρχιτεκτονικής από επιχειρήσεις που βρίσκονται γεωγραφικά διασκορπισμένες, όπου και απαιτείται η χρήση του διαδικτύου. Οι υπάρχουσες υλοποιήσεις που έχουν προταθεί έως τώρα για να επιλύσουν αυτό το πρόβλημα είναι ευάλλωτες σε επιθέσεις από κακόβουλους χρήστες, καθώς δεν εγγυούνται την ασφάλεια των επικοινωνιών με βάση τα επιχειρηματικά πρότυπα για την προστασία τους. Επίσης, το μοντέλο κοστολόγησης ανά εφαρμογή εν χρήση θεωρείται πλέον ξεπερασμένο, και νέες πολιτικές τιμολόγησης θα μπορούσαν να προταθούν και να εφαρμοστούν με χρήση των δυνατοτήτων που παρέχουν πλέον οι νέες τεχνολογίες, παρέχοντας μεγαλύτερη ευελιξία στις επιχειρήσεις.

1.2 Αντικείμενο διπλωματικής

Αντικείμενο της διπλωματικής εργασίας είναι η ανάλυση των απαιτήσεων, ο σχεδιασμός και τελικά η υλοποίηση ενός αξιόπιστου συστήματος παροχής αδειών χρήσης λογισμικου που θα επιλύει τους περιορισμούς του υπάρχοντος μοντέλου, όπως περιγράφηκε στην παραπάνω υποενότητα και καθιστά δυνατή την παροχή τους με χρήση του διαδικτύου και με ασφαλή και αξιόπιστο τρόπο κατά τα επιχειρηματικά πρότυπα.

Ένα από τα χαρακτηριστικά του ισχύοντος μοντέλου είναι η απαίτηση όλοι οι χρήστες να βρίσκονται στο ίδιο τοπικό δίκτυο με τον εξυπηρετητή αδειών χρήσης λογισμικού. Η επικοινωνία γίνεται στο επίπεδο μεταφοράς (transport layer) με ανταλλαγή TCP πακέτων δεδομένων. Καθώς οι τεχνολογίες εξελίσσονται, άρχισε να γίνεται πολύ δημοφιλής η χρήση τους διαδικτύου και η χρήση κατανεμημένων συστημάτων και υπηρεσιών. Η ίδια η φύση των επιχειρήσεων μεταλλάχτηκε σημαντικά τα τελευταία χρόνια, επιτρέποντας τον γεωγραφικό κατακερματισμό τους. Κατά αντίστοιχο τρόπο, θα πρέπει και το μοντέλο παροχής αδειών χρήσης λογισμικού που θα χρησιμοποιείται να επιτρέπει τη διασυνδεσιμότητα χρηστών από διαφορετικά και εν γένη ετερογενή δίκτυα. Αυτό δε γίνεται αφικτό καθώς, όπως αναφέρθηκε, η επικοινωνία χρηστών και εξυπηρετητών γίνεται στο επίπεδο μεταφοράς. Γι' αυτό απαραίτητη είναι η ανάπτυξη ενός μηχανισμού που θα μεταφέρει την επικοινωνία από το επίπεδο μεταφοράς στο επίπεδο εφαρμογής (application layer). Έτσι κρίθηκε κατάλληλη η μελέτη των ιδιοτήτων που παρέχει η χρήση διαδικτυακών υπηρεσιών και πιο συγκεκριμένα η υιοθέτηση της υπηρεσιοστρεφής αρχιτεκτονικής κατά την ανάπτυξη διαδικτυακών

εφαρμογών, και ο μετέπειτα σχεδιασμός και υλοποίηση του συστήματος που προτείνεται κατά αυτόν τον τρόπο.

Η επικοινωνία στο επίπεδο εφαρμογής με χρήση του διαδικτύου για την ολοκλήρωση υπηρεσιών που παρέχονται από ετερογενή δίκτυα δημιουργεί μια άλλη πρόκληση. Οποιαδήποτε χρήση του διαδικτύου εγκυμονεί σημαντικά προβλήματα ασφάλειας, αφού εν γένη η χρήση του είναι εξ ορισμού ανασφαλής. Είναι γνωστό επίσης πως οι εμπορικές εφαρμογές διέπονται από ισχυρά επιχειρηματικά πρότυπα, τα οποία ορίζουν ακριβώς τις ελάχιστες απαιτήσεις ως προς την ασφάλεια στη μεταφορά των δεδομένων και στην πιστοποίηση των συμβαλλόμενων πλευρών. Το πρόβλημα σε αυτή την περίπτωση είναι πως το υπάρχον μοντέλο παροχής αδειών χρήσης λογισμικού δεν υλοποιεί κανένα μηχανισμό ασφαλείας. Οι χρήστες και οι εξυπηρετητές βρίσκονται στο ίδιο τοπικό δίκτυο, οι εξυπηρετητές έχουν αφημένες όλες τις θύρες τους ανοιχτές, και είναι ευθύνη του διαχειριστή του δικτύου να εγγυηθεί την ασφάλεια στο δίκτυο. Η συγκεκριμένη προσσέγγιση όμως σε περιβάλλον διαδικτύου είναι ανεπίτρεπτη και μη αποδεχτή από καμιά εμπορική εφαρμογή. Για αυτό το λόγο και αποφασίστηκε η χρήση τεχνολογιών δημοσίου κλειδιού (Public Key Infrastructure – PKI) που θα εγγυηθεί την ακεραιότητα των δεδομένων κατά τη μεταφορά τους στο διαδίκτυου, την κρυπτογράφησή τους και την πιστοποίηση των δύο συμβαλλόμενων κάθε φορά προσώπων, του παρόχου και του καταναλωτή.

Τέλος, όπως αναφέρθηκε στην προηγούμενη υποενότητα, ο τρόπος κοστολόγησης της χρήσης μιας εφαρμογής έχει κατά καιρούς αλλάξει, με στόχο την εκάστοτε προσαρμογή του στις ανάγκες των επιχειρήσεων. Έτσι αρχικά κοστολογούταν μια εφαρμογή ανάλογα με το πλήθος των τερματικών που ήταν σε θέση να μπορούν να την εκτελούν. Αφού παρατηρήθηκε πως κάτι τέτοιο ήταν ασύμφορο οικονομικά για μια επιχείρηση, καθώς πολύ λιγότεροι χρήστες κάθε φορά την εκτελούσαν, δημιουργήθηκε η ανάγκη για την ανάπτυξη του υπάρχοντος μοντέλου, σύμφωνα με το οποίο μια εφαρμογή κοστολογείται ανάλογα με τους χρήστες που είναι σε θέση να την εκτελούν ταυτόχρονα. Έτσι οι επιχειρήσεις είναι σε θέση να προσαρμόζουν τις απαιτήσεις τους ανάλογα με τις ιδιαίτερες ανάγκες της κάθε μίας ξεχωριστά. Όμως και αυτό το μοντέλο θεωρείται πλέον παρωχημένο. Το ιδανικό θα ήταν να τιμολογείται κάθε εφαρμογή με βάση την πραγματική της χρήση από τους χρήστες ενός οργανισμού, και αν είναι εφικτό, ανάλογα με το είδος της χρήσης της. Σε αυτό το σημείο έγινε μελέτη των δυνατοτήτων που παρέχουν οι τεχνολογίες πολυπλέγματος (GRID) στην καταγραφή της χρήσης μιας διαδικτυακής υπηρεσίας και στα πλεονεκτήματα που παρέχουν στις διαδικτυακές εφαρμογές. Έτσι, αποφασίστηκε να γίνει χρήση του μεσισμικού πολυπλέγματος GRIA, τα χαρακτηριστικά του οποίου θεωρούνται ιδανικά για εμπορικές

εφαρμογές που κάνουν χρήση του διαδικτύου, παρέχοντας ασφαλή επικοινωνία μεταξύ όλων των συνιστωσών τους, στα επιχειρηματικά πρότυπα, ενώ παράλληλα καταγράφουν ποικίλες μεταβλητές που αφορούν τη χρήση των πόρων και των υπηρεσιών τους, δίνοντας έτσι τη δυνατότητα για δυναμική και εξατομικευμένη κοστολόγηση σε κάθε εταιρία. Κατά αυτόν τον τρόπο, παρέχεται μεγαλύτερη ευελιξία στις επιχειρήσεις που επιθυμούν να αγοράσουν μια εφαρμογή, πληρώνοντας ανάλογα με τη χρήση τους, ενώ οι διανομείς πακέτων λογισμικού είναι σε θέση να διαφημίζουν διάφορα πακέτα της ίδιας εφαρμογής, ανταποκρινόμενα στις ανάγκες της κάθε εταιρίας και αυξάνοντας την ανταγωνιστηκότητά τους στην αγορά.

Μετά τη μελέτη των διαφόρων χαρακτηριστικών των τεχνολογιών που θα χρησιμοποιηθούν στη διπλωματική εργασία, έγινε καταγραφή και ανάλυση των απαιτήσεων εκ νέου, σχεδιάστηκε η νέα προτεινόμενη αρχιτεκτονική και υλοποιήθηκε με χρήση προγραμματιστικών εργαλείων που προσφέρουν διασυνδεσιμότητα μεταξύ ετερογενών δικτύων και διαφορετικών υπολογιστικών συστημάτων, στηριζόμενα στη γλώσσα προγραμματισμού Java, στον Apache Tomcat Server, και στο μεσισμικό πολυπλέγματος GRIA. Ο έλεγχος του συστήματος έγινε με χρήση της εμπορικής εφαρμογής Mathematica, ενώ ως εξυπηρετητής παροχής αδειών χρήσης λογισμικού υπήρξε ο FLEXlm server.

1.3 Οργάνωση κειμένου

Το κέιμενο της διπλωματικής εργασίας είναι οργανωμένο στα εξείς κεφάλαια. Στο κεφάλαιο 1 γίνεται μια σύντομη περιγραφή του αντικειμένου με το οποίο ασχολείται η διπλωματική εργασία, ορίζει το πρόβλημα και αναλύει το πώς σκοπεύεται να επιλυθεί. Στο κεφάλαιο 2 γίνεται παρουσίαση των πιο σημαντικών εννοιών και τεχνολογιών που θα χρησιμοποιηθούν εδώ, για την καλύτερη κατανόηση από τον αναγνώστη του προβλήματος του συστήματος που υλοποιήθηκε. Στο κεφάλαιο 3 γίνεται παρουσίαση της καινούριας αρχιτεκτονικής που προτείνεται και η περιγραφή των λειτουργιών που θα υλοποιηθούν με τη βοήθεια διαγραμμάτων ροής. Στο κεφάλαιο 4 αναλύονται τα προγραμματιστικά εργαλεία και οι τεχνολογίες που χρησιμοποιήθηκαν για την ανάπτυξη του προτεινόμενου μοντέλου, ενώ ταυτόχρονα αναλύονται τα επιμέρους συσταστικά και κλάσεις της κάθε οντότητας λογισμικού που το αποτελεί. Στο κεφάλαιο 5 γίνεται ο έλεγχος της ορθής λειτουργίας του συστήματος, αναφέροντας αρχικά τις διαδικασίες σωστής εγκατάστασης και στη συνέχεια παρουσιάζοντας περιπτώσεις χρήσης του σε πραγματικό χρόνο. Τέλος στο κεφάλαιο 6 αναφέρονται τα συμπεράσματα που καταλήξαμε μετά την υλοποίηση και τον έλεγχο ορθής λειτουργίας του και επικεντρωνόμαστε στις τεχνολογικές προκλήσεις που παρουσιάζονται με

τη καθιέρωση και χρήση του από τις επιχειρίσεις και τους οργανισμούς, ενώ ταυτόχρονα προτείνονται μελλοντικές επεκτάσεις και βελτιώσεις του συγκεκριμένου μοντέλου. Στο κεφάλαιο 7 παρατίθεται η σχετική βιβλιογραφία από την οποία βασίστηκε η συγκεκριμένη διπλωματική εργασία.

2

Περιγραφή Βασικών Εννοιών

Σε αυτό το κεφάλαιο θα γίνει μια αναλυτική περιγραφή των βασικών εννοιών, μοντέλων και τεχνικών που έχουν χρησιμοποιηθεί στην παρούσα διπλωματική εργασία. Απαραίτητο είναι να αναφερθεί πως όλα τα παραπάνω αποτελούν συνήθεις πρακτικές και έχουν προταθεί ήδη από τρίτους, συνεπώς δεν είναι αντικείμενο της διπλωματικής. Παρόλα αυτά, η βασική κατανόησή τους από τον αναγνώστη θεωρείται απαραίτητη.

2.1 Διαδικασία Παροχής Αδειών Λογισμικού

Στην παρούσα ενότητα θα αναφέρουμε τον ορισμό της παροχής άδειας λογισμικού με μια σύντομη ιστορική αναδρομή, ενώ παράλληλα θα γίνει αναφορά και στο μοντέλο που απαιτεί τη χρήση ενός FLEXlm Server.

2.1.1 Ορισμός της Παροχής Άδειας Λογισμικού

Η διαδικασία παροχής άδειας λογισμικού είναι μια συμφωνία μεταξύ του παρόχου του λογισμικού και του τελικού χρήστη. Η άδεια χρήσης λογισμικού επιτρέπει στον τελικό

χρήστη να χρησιμοποιήσει το εν λόγω λογισμικό σε περιπτώσεις η χρήση του τελευταίου χωρίς την αντίστοιχη άδεια επιφέρει ποινικές ευθύνες.

Με την παροχής άδειας χρήσης λογισμικού, ο πάροχος και πνευματικός ιδιοκτήτης του επιτρέπει να πραγματοποιηθούν συγκεκριμένες ενέργειες που καθορίζονται πλήρως πάνω στη συγκεκριμένη εφαρμογή από τον συμφωνημένο χρήστη. Η πιο συνηθισμένη ενέργεια που μπορεί να πραγματοποιηθεί είναι η εκτέλεση του συγκεκριμένου λογισμικού με σεβασμό στους όρους που έχουν καθοριστεί στην άδεια, αλλά και σε ορισμένες περιπτώσεις η αντιγραφή, τροποποίηση και διανομή του.

Είναι πλέον σύνηθες οι διάφορες εφαρμογές να λειτουργούν πλέον συχνότερα με άδειες χρήσης λογισμικού, παρά με την αγοραπωλησία. Οι ίδιες οι άδειες εξάλλου μπορούν να περιγράφουν διάφορους τύπους χρήσης. Έτσι υπάρχουν άδειες που ορίζουν τον αριθμό των μηχανημάτων στα οποία μπορεί να εκτελείται ταυτόχρονα μια εφαρμογή ή τον αριθμό των τελικών χρηστών που μπορούν να την χρησιμοποιούν την ίδια χρονική στιγμή. Υπάρχουν περιπτώσεις όπου ένα συγκεκριμένο αντίγραφο λογισμικού να απαιτεί μια άδεια χρήσης ανά επεξεργαστή όπου εκτελείται. Τέλος το πιο συνήθες είναι μία εφαρμογή να μπορεί να εκτελεστεί μόνο σε ένα μηχάνημα και να επιτρέπεται η αντιγραφή της σε άλλο μόνο για περιπτώσεις αποθήκευσης. Παρόλα αυτά, δίνεται η δυνατότητα να είναι αποθηκευμένη σε περισσότερα του ενός μηχανήματα, με την προυπόθεση όμως να εκτελείται μόνο σε ένα σε κάθε χρονική στιγμή .

2.1.2 Ιστορική Αναδρομή – Μεμονωμένα μη συνδεδεμένα συστήματα

Το χρονικό σημείο κατά το οποίο άλλαξε ριζικά ο τρόπος διανομής των αδειών λογισμικού, είναι η εμφάνιση και η ευρεία διάδοση των δικτύων, που είχε σαν αποτέλεσμα την κατάργηση των υπολογιστών ως μεμονομένες και αυτοδύναμες μονάδες. Ως τότε η διανομή των εφαρμογών γινόταν μόνο με κάποιο μέσο φυσικής αποθήκευσης [δισκέττα, δίσκο CD κτλ]. Τα μοντέλα παροχής αδειών λογισμικού αφορούσαν σαν αποτέλεσμα κάποιο συγκεκριμένο μηχάνημα, ως μεμονομένη και αυτοδύναμη οντότητα.

Άδεια με συστελλόμενο περιτύλιγμα (Shrink-wrapped)

Το πρώτο μοντέλο που χρησιμοποιήθηκε για διανομή άδειας λογισμικού ονομάστηκε *μοντέλο με συστελλόμενο περιτύλιγμα*. Σύμφωνα με το συγκεκριμένο μοντέλο, το λογισμικό διανέμεται σε ένα μέσο φυσικής αποθήκευσης, η οποία εσωκλείεται σε ένα περιτύλιγμα, βρίσκεται δηλαδή περιτυλιγμένη σε ένα χαρτί, ή ένα πλαστικό, ή βρίσκεται μέσα σε μια συσκευασία όπως πλαστικό κουτί, η δε άδεια είναι εκτυπωμένη σε ένα χαρτί, ή σε περιπτώσεις που είναι πολύ μεγάλη, βρίσκεται αποθηκευμένη μέσα στο φυσικό μέσο. Ο χρήστης θεωρείται ότι αποδέχεται τους όρους της άδειας χρήσης του συγκεκριμένου λογισμικού ταυτόχρονα με το άνοιγμα της συσκευασίας. Αυτές οι άδειες οδήγησαν μετά από λίγα χρόνια στις *ηλεκτρονικές άδειες με συστελλόμενο περιτύλιγμα* στις οποίες ο χρήστης δέχεται τους όρους που αναγράφονται απατώντας σε μια ερώτηση ναι/όχι η οποία και τους υποβάλεται ακριβώς την πρώτη φορά που γίνεται απόπειρα εκτέλεσής της.

Αυτό που πρέπει να σημειωθεί, είναι ότι δεδομένου ότι οι υπολογιστές δεν είναι συνδεδεμένοι σε κάποιο δίκτυο, δεν υπάρχει δυνατότητα παροχής της άδειας χρήσης εξ αποστάσεως. Το συγκεκριμένο μοντέλος παρέχει στο χρήστη τη δυνατότητα χρήσης της εφαρμογής επ' άπειρον, χωρίς κανέναν περιορισμό, από τη στιγμή της αγοράς της. Οι δε χρήστες δεν είναι κάτοχοι της συγκεκριμένης εφαρμογής, αλλά της άδειας χρήσης της.

Ένα από τα προβλήματα που προκύπτουν με το συγκεκριμένο μοντέλο, είναι η αποτυχία του να αποκλείσει τη χρήση του λογισμικού σε περισσότερα του ενός μηχανήματα την ίδια χρονική στιγμή. Συνήθως οι άδειες χρήσης του λογισμικού απαγορεύουν την αντιγραφή, αναδιανομή και εν τέλη την εκτέλεσή του από διαφορετικό χρήστη ταυτόχρονα. Παρόλα αυτά οι άδειες με συστελλόμενο περιτύλιημα δεν αποκλείουν κακόβουλη χρήση, με αποτέλεσμα πολλού χρήστες να προβαίνουν σε ενέργειες πειρατείας.

Σήμερα αυτός ο τρόπος παροχής άδειας τείνει να εγκαταληφθεί και να αντικατασταθεί από πιο έξυπνα μοντέλα. Παρόλα αυτά χρησιμοποιείται ακόμα σε περιπτώσεις λογισμικού που προορίζονται αποκλειστικά για μεμονωμένους υπολογιστές, όπως τα παιχνίδια.

Μοντέλο Παροχής Άδειας με Σειριακό Αριθμό

Ένα λίγο διαφορετικό μοντέλο είναι το *μοντέλο παροχής άδειας με σειριακό αριθμό*. Σύμφωνα με το συγκεκριμένο μοντέλο, απαιτείται η εισαγωγή ενός σειριακού αριθμού κατά την πρώτη προσπάθεια εκτέλεσης του λογισμικού. Ο σειριακός αριθμός αποτελείται από μια σύνθετη και μακροσκελής ακολουθία γραμμάτων, αριθμών και συμβόλων τον οποίο και προμηθεύεται ο τελικός χρήστης κατά την αγορά του προιόντος. Η ύπαρξη του σειριακού αριθμού εξασφαλίζει ότι το λογισμικό θα χρησιμοποιηθεί από τον αγοραστή του.

Αξίζει εδώ να σημειωθεί ότι τα πρώτα χρόνια χρήσης του εν λόγω μοντέλου, δεν υπήρχαν δίκτυα, με αποτέλεσμα οι υπολογιστές να μην είναι συνδεδεμένοι και να αποτελλούν αυτόνομες οντότητες, όπως ήδη αναφέρθηκε. Αυτό είχε σαν αποτέλεσμα η ύπαρξη του σειριακού αριθμού να μην είναι δυνατόν να αποκλείσει την αναδιανομή του λογισμικού, την αντιγραφή και ταυτόχρονη εκτέλεσή του από διαφορετικούς χρήστες. Παρόλα αυτά, με την εξέλιξη της τεχνολογίας των δικτύων, την ύπαρξη του διαδικτύου και τη δυνατότητα ύπαρξης μια κεντρικής βάσης δεδομένων η οποία και θα μπορεί να διαχειρίζεται τους σειριακούς αριθμούς, είναι δυνατόν να εξασφαλιστούν οι όροι της άδειας λογισμικού και να αποκλειστούν περιπτώσεις πειρατείας και γενικότερα παράνομης χρήσης του, γι' αυτό και το μοντέλο με το σειριακό αριθμό χρησιμοποιείται ακόμα και σήμερα ευρέως.

Μοντέλο παροχής άδειας Περιορισμένου Χρόνου

Σύμφωνα με το συγκεκριμένο μοντέλο, παρέχεται η δυνατότητα χρήσης του λογισμικού για ένα προκαθορισμένο και μόνο χρονικό διάστημα. Συνήθως η χρήση διαιρείται σε δύο μέρη. Ένα δοκιμαστικό μέρος (trial) κατά το οποίο ο χρήστης μπορεί να εκτελεί την εφαρμογή χωρίς κανένα κόστος για ένα συγκεκριμένο χρονικό διάστημα, και το δεύτερο μέρος, κατά το οποίο αν ο χρήστης είναι ευχαριστημένος από την εφαρμογή, αγοράζει την άδεια χρήσης που του παρέχει τη δυνατότητα να τη χρησιμοποιεί είτε για άπειρο χρονικό διάστημα ή για περιορισμένο. Στη δεύτερη περίπτωση, η εφαρμογή ελέγχει πόσος χρόνος έχει περάσει από την χρονική στιγμή που ενεργοποιήθηκε η άδεια με βάση το ρολόι του συστήματος στο οποίο εκτελείται, σύμφωνα βεβαίως με τους όρους της τελευταίας. Αν έχει περάσει το χρονικό διάστημα η άδεια ανακαλείται και η εφαρμογή δε μπορεί να εκτελεστεί.

Το μοντέλο αυτό είναι ευάλλωτο, καθώς οποιαδήποτε αλλαγή στο ρολόι του συστήματος και επαναφορά του σε προηγούμενη χρονική στιγμή, δίνει το δικαίωμα στο χρήστη να το

παρακάμψει και να μπορέσει να χρησιμοποιήσει την εφαρμογή επ' άπειρον. Παρόλα αυτά, έχουν υλοποιηθεί βελτιώσεις του συγκεκριμένου μοντέλου, κατά τις οποίες οποιαδήποτε μεταβολή στο ρολόι του συστήματος προκαλεί αυτόματη ανάκληση της άδειας χρήσης.

Μοντέλο Παροχής Άδειας Βασισμένο στα Χαρακτηριστικά

Ένα άλλο μοντέλος παροχής άδειας λογισμικού είναι το μοντέλο *βασισμένο στα χαρακτηριστικά*. Σύμφωνα με το συγκεκριμένο μοντέλο, το λογισμικό παρέχει μια πληθώρα δυνατοτήτων οι οποίες απευθύνονται σε τελικούς χρήστες με διαφορετικές απαιτήσεις. Έτσι μπορεί να ικανοποιεί ανάγκες αρχάριων χρηστών, οι οποίοι είναι ικανοποιημένοι με ένα minimum απαιτήσεων, να ικανοποιεί ανάγκες ενδιάμεσων χρηστών οι οποίοι και θέλουν να χρησιμοποιήσουνε περισσότερα χαρακτηριστικά του, αλλά και ανάγκες επαγγελματιών που πρέπει να χρησιμοποιήσουνε όλες τις δυνατότητες που παρέχει η εφαρμογή. Γι' αυτόν το λόγο το λογισμικό, είτε διανέμεται σε διαφορετικές εκδόσεις, η κάθε μία από τις οποίες καλύπτεται με διαφορετική άδεια χρήσης, είτε διανέμεται σε ένα μοναδικό πακέτο, με τον χρήστη όμως να αποφασίζει ποιες δυνατότητές του θα χρησιμοποιήσει, επιλέγοντας την κατάλληλη άδεια. Όσες περισσότερες δυνατότητες παρέχονται στο χρήστη, τόσο πιο ακριβά κοστίζει και η άδεια.

Το προφανές ελάττωμα της άδειας βασισμένης στα χαρακτηριστικά είναι η αδυναμία της να εξασφαλίζει αποκλειστική χρήση του λογισμικού σε περιπτώσεις μεμονωμένων υπολογιστών. Γι' αυτόν το λόγο και το συγκεκριμένο μοντέλο παροχής άδειας συνδιάζεται με άλλα για να αποτρέψει περιπτώσεις παράνομης χρήσης.

Μοντέλο Παροχής Άδειας Βασισμένο στην Τοποθεσία

Το *μοντέλο παροχής άδειας βασισμένο στην τοποθεσία* επιτρέπει την ταυτόχρονη χρήση του λογισμικού από πολλαπλούς χρήστες. Εφαρμόζεται συνήθως σε περιπτώσεις εταιριών ή άλλων οργανισμών. Το μοντέλο αυτό παρέχει οικονομικές ευκολίες σε εταιρίες να αγοράζουν πολλαπλές άδειες πιο οικονομικά. Έτσι, αντί να προμηθεύονται πολλές άδειες, μία για κάθε πρόσωπο που σχετίζεται με την εταιρία, αγοράζεται μία η οποία και δίνει τη δυνατότητα χρήσης της εφαρμογής σε όσους χρήστες ορίσει αυτή.

Μοντέλο Παροχής Άδειας Βασισμένο στο Υλικό

Η διασφάλιση ορθής χρήσης του λογισμικού σε μεμονομένους υπολογιστές μπορεί να επιτευχθεί χρησιμοποιώντας μια μικρή συσκευή που ονομάζεται dongle. Το dongle, το οποίο μπορεί να εισαχθεί σε μια κατάλληλη θύρα του υπολογιστή, παρέχει στην εφαρμογή έναν σειριακό αριθμό που χρειάζεται η τελευταία να επιβεβαιώσει για να μπορέσει να εκκινήσει. Δεδομένου ότι είναι πολύ δύσκολο να κατασκευαστή παρόμοιο dongle που να δίνει τον ίδιο αριθμό, και ότι μόνο ένα dongle παράγει τον συγκεκριμένο σειριακό αριθμό που απαιτείται από την εφαρμογή, είναι πρακτικά πολύ δύσκολο να εκτελεστεί ταυτόχρονα από πολλαπλούς χρήστες.

Μοντέλο Παροχής άδειας .NET

Το *μοντέλο παροχής άδειας με .NET* αποτελεί τμήμα της προγραμματιστικής πλατφόρμας της Microsoft με όνομα .NET Framework. Το μοντέλο αυτό εξαφασλίζει τη μοναδική χρήση ενός λογισμικού σε μεμονωμένο υπολογιστή, χωρίς να απαιτείται η ύπαρξη δικτύου και κεντρικής βάσης δεδομένων που να διαχειρίζεται τη διακίνηση των αδειών. Το λογισμικό παρέχεται μαζί με την πλατφόρμα .NET Framework. Η μοναδικότητα του χρήστη εφασφαλίζεται εξ ορισμού από τα χαρακτηριστικά του .NET Framework, οι δε προγραμματιστές της εφαρμογής μπορούν να την επεκτείνουν να υποστηρίζει ταυτόχρονα και άλλα μοντέλα παροχής αδειών. Το σημαντικότερο μειονέκτημα του συγκεκριμένου μοντέλου είναι η απαίτηση το λογισμικό να εκτελείται πάνω στην προκαθορισμένη πλατφόρμα της Microsoft, πράγμα που μειώνει τη διαλειτουργικότητά του, και τη δυνατότητα να εκτελείται σε διαφορετικά λειτουργικά συστήματα καθώς παρουσιάζονται προβλήματα συμβατότητας.

2.1.3 Ιστορική Αναδρομή – Υπολογιστικά συστήματα συνδεδεμένα στο διαδίκτυο

Καθώς οι τεχνολογίες δικτύων εξελίσσονταν, κάνοντας εφικτή την ταυτόχρονη σύνδεση πολλών υπολογιστών στο διαδίκτυο, ταυτόχρονη εξέλιξη παρατηρήθηκε και στον τρόπο παροχής των αδειών λογισμικού. Ο κύριος σκοπός της άδειας, δηλαδή η εξασφάλιση της ορθής χρήσης του λογισμικού και η προστασία της εφαρμογής από κακόβουλη και παράνομη χρήση είναι ευκολότερο να διασφαλιστεί. Παράλληλα όμως, το διαδίκτυο παρέχει

ευκολότερο και πιο ευέλικτο τρόπο είτε διανομής της άδειας, είτε πληρωμής της χρήσης της, είτε επιβολής των όρων της.

Συγκεκριμένα συστήματα σε διαφορετικά περιβάλλονται διαχειρίζονται τη διαδικασία παροχής άδειας με διαφορετικούς τρόπους και κάθε περιβάλλον έχει το δικό του τρόπο παραγωγής, έγκρισης και διανομής της άδειας. Οι τεχνικές παροχής της άδειας χρήσης του λογισμικού εξαρτώνται από τον τύπο της εφαρμογής και τους τελικούς χρήστες, από το κόστος της εφαρμογής, από το υλικό και διαδικτυακό περιβάλλον στο οποίο εκτελείται η εφαρμογή και από τους δράστες που συμμετέχουν στη διαδικασίας παροχής και διανομής της.

Με την εμφάνιση των τοπικών δικτύων οι περισσότερες εταιρίες και οργανισμοί διαθέτουν απλούς τερματικούς υπολογιστές συνδεδεμένους είτε απευθείας μαζί, είτε μέσω σταθμών εξυπηρετητών, μαζί με εκτυπωτές ή άλλες δικτυακές συσκευές σε ένα ενιαίο τοπικό δίκτυο. Τα τοπικά δίκτυα αυτά μπορεί να συνδέονται με άλλα τοπικά δίκτυα είτε απευθείας μεταξύ τους, είτε μέσω του διαδικτύου. Το γεγονός αυτό δίνει τη δυνατότητα κάθε κόμβος ενός δικτύου να μπορεί να αποκτήσει ένα λογισμικό ή άδεια χρήσης τους από κάποιον άλλον κόμβο που βρίσκεται στο ίδιο τοπικό δίκτυο είτε κάπου αλλού. Επίσης το λογισμικό μπορεί να βρίσκεται σε ένα μόνο υπολογιστικό κόμβο και να προσπελάζονται από οποιαδήποτε άλλη συσκευή.

Η ιδέα ότι ένας κόμβος ο οποίος αποτελεί έναν κεντρικό εξυπηρετητή μπορεί να προσπελάζεται από οποιονδήποτε άλλον, είτε στο ίδιο τοπικό δίκτυο, είτε αλλού, οδήγησε σε εντελώς διαφορετικές αρχιτεκτονικές διανομής αδειών λογισμικού. Επιπλέον τα διάφορα χαρακτηριστικά που αποκτά ένα υπολογιστικό σύστημα σε ένα δίκτυο αποτελούν επιπλέον στοιχεία ταυτοποίησης ενός χρήστη, άρα και εξασφάλισης της ορθής χρήσης μιας εφαρμογής. Τέτοια χαρακτηριστικά είναι το δικτυακό όνομα ενός κόμβου, η διεύθυνση IP του, όπως κυρίως η διεύθυνση MAC της κάρτας δικτύου του, η οποία και είναι μοναδική παγκοσμίως.

Όλα τα παραπάνω καθέστησαν δυνατό να αναπτυχθούν διαφορετικά μοντέλα διανομής αδειών λογισμικού. Τα κυριότερα από αυτά είναι το *μοντέλο παροχής με σύγχρονο τρόπο* και το *μοντέλο κλειδωμένο στον κόμβο*.

Μοντέλο Παροχής Άδειας Χρήσης Λογισμικού με Σύγχρονο Τρόπο (concurrent license model)

Πριν γίνει εφικτή η σύνδεση των υπολογιστών μιας εταιρίας ή ενός οργανισμού μεταξύ τους μέσω τοπικού δικτύου, κάθε υπολογιστής στην εταιρία στον οποίο θα έπρεπε να εκτελεστεί μια συγκερκιμένη εφαρμογή έπρεπε να την εγκασταστήσει, αποδεχόμενος την άδεια χρήσης της. Έχει παρατηρηθεί όμως ότι σε μια δεδομένη χρονική στιγμή μόνο ένα υποσύνολο από τα συνολικά εγκατεστημένα αντίγραφα μιας εφαρμογής εκτελούνται ταυτόχρονα. Χωρίς την ύπαρξη τοπικού δικτύου, το πρόβλημα εξασφάλισης της ορθής εκτέλεσης του λογισμικού λυνόταν με τα μοντέλα που αφορούν μεμονωμένους υπολογιστές. Αυτό έχει σαν αποτέλεσμα να ξοδεύονται αρκετά χρήματα για αγορά αδειών χωρίς αυτές να χρειάζονται, αφού οι εφαρμογές δεν εκτελούνται. Το ιδανικό σενάριο θα ήταν ο οργανισμός να προμηθεύεται τον αριθμό των αδειών που χρειάζεται, δηλαδή τον αριθμό των αντίγραφων της εφαρμογής που εκτελούνται ταυτόχρονα.

Η λύση σε αυτό το πρόβλημα δίνεται με το *μοντέλο παροχής άδειας χρήσης λογισμικού με σύγχρονο τρόπο*. Το μοντέλο αυτό απαιτεί έναν υπολογιστικό κόμβο να λειτουργεί σαν εξυπηρετητής και όλοι οι υπόλοιποι να μπορούν να επικοινωνούν μαζί του. Κάθε τερματικό συνδεδεμένο στο τοπικό δίκτυο έχει αποθηκευμένο ένα αντίγραφο της εφαρμογής, ενώ ο εξυπηρετητής έχει αποθηκευμένες το σύνολο των αδειών χρήσης της εφαρμογής. Όταν κάθε τερματικό-πελάτης επιθυμεί να εκτελέσει μια συγκεκριμένη εφαρμογή επικοινωνεί με τον εξυπηρετητή. Αυτός ελέγχει πόσοι πελάτες εκτελούν την επιθυμητή εφαρμογή τη συγκεκριμένη χρονική στιγμή. Αν είναι λιγότεροι από όσους περιγράφονται στην άδεια χρήσης, ο εξυπηρετητής δεσμεύει την άδεια για λογαριασμού του χρήστη. Αν είναι ίσοι με όσους απαιτεί η άδεια, ο εξυπηρετητής αρνείται την εκτέλεση της εφαρμογής. Με αυτό το μοντέλο, οι άδειες κατά κάποιον τρόπο *ρέουν* στο δίκτυο και δεσμεύονται δυναμικά από όποιον έχει την ανάγκη να κάνει χρήση του λογισμικού.

Για να λειτουργήσει αυτή η αρχιτεκτονική απαιτεί συνδέσεις TCP/IP πάνω στο δίκτυο μεταξύ του πελάτη και του εξυπηρετητή. Συχνά απαιτούνται αυτές οι συνδέσεις να παραμένουν ανοιχτές καθ' όλη τη διάρκεια εκτέλεσης της εφαρμογή, καθώς πολλά πρωτόκολλα απαιτούν την ανά χρονικά διαστήματα επιβεβαίωση από μεριάς εξυπηρετητή ότι ο πελάτης είναι ενεργός και εκτελεί κανονικά την εφαρμογή. Αν για κάποιο λόγο ο πελάτης είναι ανενεργός, ο εξυπηρετητής το αντιλαμβάνεται με αυτή τη μέθοδο, κλείνει τη σύνδεση και αποδεσμεύει την άδεια από το χρήστη ώστε να μπορεί να εκτελέσει την εφαρμογή κάποιος άλλος. Αυτή η πρακτική δημιουργεί το πρόβλημα της αυξημένης κίνησης πάνω στο δίκτυο, καθώς

πολλαπλές συνδέσεις παραμένουν ανοιχτές για μεγάλο χρονικό διάστημα, οι οποίες ανταλλάσουν πακέτα δεδομένων αρκετά συχνά. Το άλλο μεγάλο μειονέκτημα του μοντέλου παροχής με σύγχρονο τρόπο παρατηρείται σε μεγάλους οργανισμούς, όπου και ο εξυπηρετητής συνδέεται ταυτόχρονα με πάρα πολλούς πελάτες, διατηρώντας τις συνδέσεις του ανοιχτές για μεγάλο χρονικό διάστημα, πράγμα που μπορεί να οδηγήσει σε προβλήματα απόδοσης. Ακριβώς για το λόγο ότι υπάρχει μόναδικός εξυπηρετητής, αν για κάποιο από τους παραπάνω λόγους δε μπορεί να διαθέσει τις άδειες, τότε κανένας δε θα μπορέσει να εκτελέσει την εφαρμογή. Η λύση σε αυτό είναι η ύπαρξη εφεδρειών (backups) που μπορεί να επικοινωνήσει ο πελάτης αν ο πρώτος κεντρικός κόμβος δεν ανταποκρινεται.

Μοντέλο Παροχής Άδειας Λογισμικού Κλειδωμένης σε Κόμβο (node-locked license model)

Σε ένα δίκτυο οι άδειες μπορούν επίσης να είναι κλειδωμένες σε ένα συγκεκριμένο κόμβο-συσκευή, και μόνο αυτός ο κόμβος να μπορεί να χρησιμοποιήσει την εφαρμογή. Το συγκεκριμένο μοντέλος είναι γνωστό και ως *μοντέλο παροχής άδειας κλειδωμένης σε κόμβο*.

Μια τέτοια άδεια περιλαμβάνει πληροφορίες σχετικά με την ταυτότητα του κόμβου, όπως η στατική διεύθυνση IP του, το δικτυακό όνομα του υπολογιστή ή την διεύθυνση MAC της κάρτας δικτύου του.

Το συγκεκριμένο μοντέλο εξασφαλίζει ότι η εφαρμογή θα μπορεί να εκτελεστεί μόνο από το συγκεκριμένο δικτυακό κόμβο. Κατά αυτόν τον τρόπο δεν απαιτείται η ύπαρξη ενός κεντρικού εξυπηρετητή που να λειτουργεί σαν παροχέας άδειας. Παρόλα αυτά, συνήθως το μοντέλο εφαρμόζεται μόνο σε τοπικά δίκτυα, αφού τα περισσότερα χαρακτηριστικά παραμένουν μοναδικά μόνο μέσα στο συγκεκριμένο τοπικό δίκτυο.

2.2 FLEXlm Εφαρμογές

Σε αυτή την ενότητα θα γίνει περιγραφή του τρόπου παροχής αδειών λογισμικού με χρήση του FLEXlm εξυπηρετητή της Macrovision Corporation.

2.2.1 Εισαγωγή στον FLEXlm

Ο FLEXlm είναι ο πιο διάσιμος εξυπηρετητής διαχείρησης αδειών που εφαρμόζεται στην βιομηχανία λογισμικού. Είναι γνωστός για τη δυνατότητα να κάνει τις άδειες διαθέσιμες (ή όπως αλλιώς είναι γνωστό 'να ρέουν') σε οποιοδήποτε κόμβο ενός δικτύου, σε αντίθεση με το να είναι δεσμευμένες μόνιμα σε κάποιο μηχάνημα. Αυτή δυνατότητα οφελεί τόσο τους τελικούς χρήστες όσο και τους διαχειριστές των αδειών. Οι πρώτοι κάνουν πιο αποδοτική χρήση λιγότερων αδειών μοιράζοντάς τες πάνω στο δίκτυο, ενώ οι δεύτεροι μπορούν να ελέγχουν ανά πάσα στιγμή ποιος εκτελεί μια εφαρμογή και σε ποιο μηχάνημα.

Το FLEXlm προήλθε από τη συνεργασία δύο εταιριών: της GLOBEtrotter Software και της Highland Software το 1988. Το 1994 η GLOBEtrotter Software αγόρασε τα δικαιώματα της Highland Software πάνω στο προιόν. Η Highland Software συνέχιση να είναι ο μεταπωλητής του τρίτου μέρους του προιόντος. Εν τέλη η GLOBEtrotter Software αγοράστηκε από την Macrovision Corporation το 2000 όπου και μετανόμασε το προιόν από FLEXlm σε FLEXnet και πλέον το εμπορεύεται κάτω από την ονομασία FLEXnet Publisher.

Περιγράφοντας σε γενικές γραμμές τη λειτουργία του, ο FLEXlm είναι ένα διαχειρηστής παροχής αδειών λογισμικού της Macrovision Corporation, και ως τέτοιος ικανοποιεί όλες τις απαιτήσεις του προβλήματος παροχής αδειών, όπως περιγράφηκαν στην παραπάνω ενότητα. Χρησιμοποιείται κυρίως σε επιχειρησιακά περιβάλλοντα για να παράχει την άδεια εκτέλεσης μιας εφαρμογής από τους τελικούς χρήστες.

Η παροχή αδειών λογισμικού με χρήση δικτύου γίνεται με δύο τρόπους, όπως ήδη αναφέραμε. Ο πρώτος απαιτεί τη σύνδεση της άδειας με ένα συγκεκριμένο υπολογιστικό κόμβο [node-locked] και τη δυνατότητα εκτέλεσης της εφαρμογής με βάση τη συγκεκριμένη άδεια μόνο πάνω στον κόμβο αυτό. Η ταυτοποίηση γίνεται με βάση τα δικτυακά χαρακτηριστικά του τερματικού, όπως IP ή MAC διεύθυνση ή δικτυακό όνομα. Κατά το δεύτερο τρόπο υπάρχει ένας κεντρικός εξυπηρετητής, προσπελάσιμος από όλους τους

κόμβους του δικτύου, ο οποίος και έχει συγκεντρωμένες το σύνολο των αδειών. Όταν ένα τερματικό-πελάτης θέλει να εκτελέσει μια συγκεκριμένη εφαρμογή, επικοινωνεί πρώτα με τον εξυπηρετητή. Ο τελευταίος ελέγχει αν το σύνολο των χρηστών που εκτελεί τη συγκεκριμένη εφαρμογή είναι μικρότερο από όσους ορίζει η άδεια, και αν ναι, δεσμεύει μία για τον συγκεκριμένο χρήστη. Κατά αυτόν τον τρόπο έχουμε μια συνεχόμενη ροή αδειών λογισμικού πάνω στο δίκτυο. Όταν ο χρήστης τερματίσει την εκτέλεση της εφαρμογής, επικοινωνεί πάλι με τον εξυπηρετητή και αυτός με τη σειρά του αποδεσμεύει την αντίστοιχη άδεια. Έτσι παρέχεται η δυνατότητα σε έναν οργανισμό με εκατοντάδες περιστασιακούς χρήστες να αγοράσει μια εφαρμογή με 50 άδειες. Αυτό σημαίνει ότι κάθε χρήστης του οργανισμού μπορεί να εκτελέσει την εφαρμογή, αλλά μόνο 50 χρήστες μπορούν να την εκτελούν ταυτόχρονα ανά πάσα χρονική στιγμή. Χαρακτηριστικό του FLEXlm είναι ότι υποστηρίζει και τα δύο αυτά μοντέλα [3].

2.2.2 Συστατικά του FLEXlm

Όπως γίνεται κατανοητό από την παραπάνω υποπαράγραφο, τα συστατικά του FLEXlm εξαρτόνται από το είδος του μοντέλου παροχής αδειών που χρησιμοποιείται. Η κυριότερη διαφοράς τους εντοπίζεται στην απαίτηση για ύπαρξη ή όχι κεντρικού εξυπηρετητή.

Πιο συγκεκριμένα

- Μοντέλο κλειδωμένο σε κόμβο [node-locked]: Οι άδειες δεν παρέχονται από κάποιον εξυπηρετητή, αλλά είναι συνδεδεμένες με κάποιον υπολογιστικό κόμβο και είναι διαθέσιμες σε αυτόν άμεσα.
- Σύγχρονο μοντέλο [concurrent]: Οι άδειες παρέχονται από κάποιον κεντρικό εξυπηρετητή. Αυτός κρατάει κατάλληλα αρχεία άδειας (license files) που παρέχονται από τον πωλητή του λογισμικού και περιλαμβάνουν τις γραμμές SERVER, VENDOR και κατ' επιλογή USE_SERVER.

Συστατικά για το μοντέλο κλειδωμένο σε κόμβο [node-locked]

Για το μοντέλο κλειδωμένο σε κόμβο τα βασικά συστατικά που απαιτούνται από τον FLEXlm είναι δύο:

1. Η εφαρμογή που διαθέτει FLEXlm άδεια μαζί με τη στιατική βιβλιοθήκη του πελάτη.
2. Το αρχείο παροχής άδειας (license file).

Συστατικά για το σύγχρονο μοντέλο [concurrent]

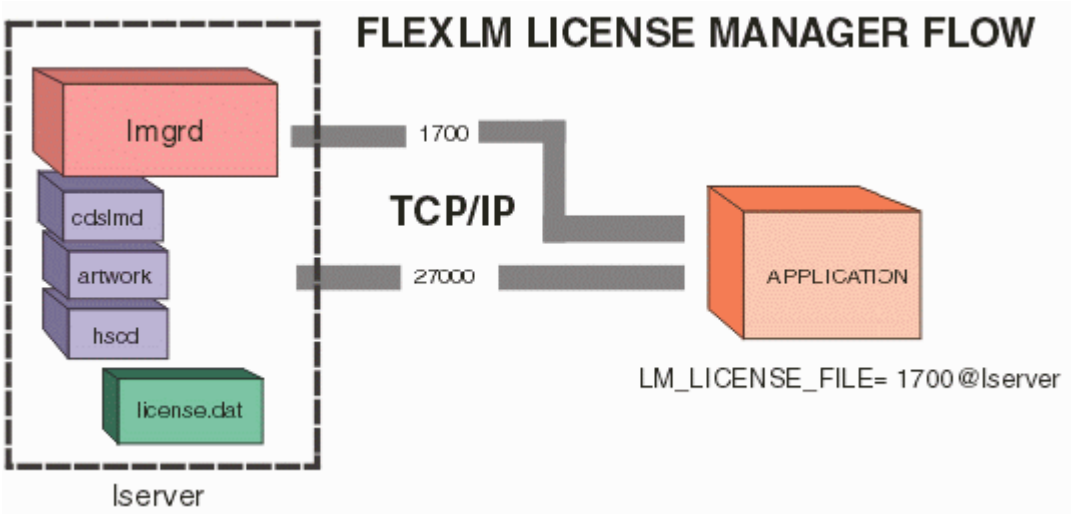
Για το σύγχρονο μοντέλο τα βασικά συστατικά που απαιτούνται αππό τον FLEXlm είναι τέσσερα:

1. Ο δαίμονας διαχείρισης άδειας (License Manager Daemon – lmgrd).
2. Ο δαίμονας πωλητής (Vendor daemon).
3. Το αρχείο παροχής άδειας (license file).
4. Η εφαρμογή που διαθέτει FLEXlm άδεια μαζί με τη στιατική βιβλιοθήκη του πελάτη.

Ο δαίμονας διαχείρισης άδειας μαζί με το δαίμονα πωλητή αποτελούν και τον εξυπηρετητή παροχής άδειας (license server).

Επιπρόσθετα με τα παραπάνω στοιχεία υπάρχουν και τρία προαιρετικά στοιχεία:

1. Το Debug Log File που είναι προσπελάσιμο από τον δαίμονα διαχείρισης άδειας.
2. Το Report Log File που είναι προσπελάσιμο από τον δαίμονα πωλητή για χρήση από τον FLEXnet manager.
3. Το End-User Administration Options File, ένα αρχείο με τις επιλογές διαχείρησης του τελικού χρήστη, το οποίο δημιουργείται και διατηρείται από τον τελικό χρήστη.



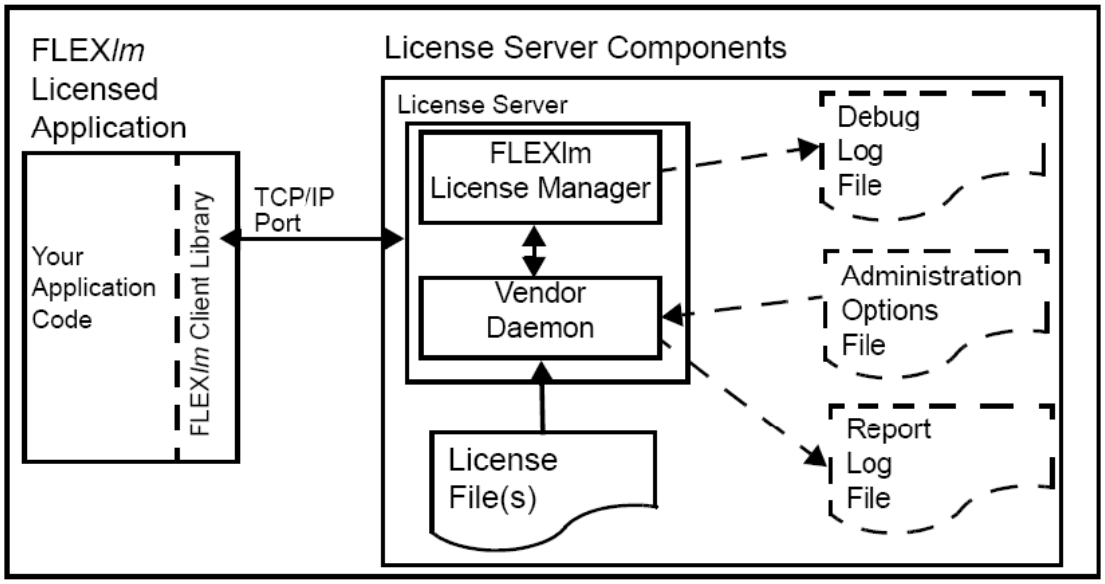
Εικόνα 2.1 –Συστατικά FLEXlm

Όπως έχει αναφερθεί, για το σύγχρονο μοντέλο απαιτείται η ύπαρξη ενός κεντρικού εξυπηρετητή διαχείρισης των αδειών λογισμικού. Ο license server μπορεί να βρίσκεται σε κάποιον ξεχωριστό κόμβο στο δίκτυο, μπορεί όμως να βρίσκεται και στο ίδιο μηχάνημα με την εφαρμογή-πελάτη. Τα τρία προαιρετικά αρχεία, Debug Log, Report Log και End-User Administration Options File, ρυθμίζεται από τον τελικό χρήστη της εφαρμογής ή αντίστοιχα από τον διαχειριστή του συνόλου των τελικών χρηστών.

2.2.3 Αναλυτική Περιγραφή των συστατικών του Σύγχρονου Μοντέλου

Στο παρόν θέμα της διπλωματικής εργασίας, θα γίνει χρήση του σύγχρονου μοντέλου παροχής αδειών λογισμικού. Για αυτόν το λόγο θα γίνει μια πιο αναλυτική περιγραφή των συστατικών που το αποτελούν, ώστε να έχει ο αναγνώστης μια πιο πλήρη εικόνα των δυνατοτήτων που παρέχει ο FLEXlm όταν χρησιμοποιείται κατά αυτόν το τρόπο.

Στο ακόλουθο σχήμα απεικονίζονται όλα τα στοιχεία που αποτελούν το σύγχρονο μοντέλο.



Εικόνα 2.2 - Διάγραμμα Οντοτήτων FLEXlm

Δαίμονας διαχείρισης αδειών (FLEXlm License Manager)

Ο δαίμονας διαχείρισης αδειών χειρίζεται την πρώτη επικοινωνία μεταξύ του χρήστη-πελάτη με τις FLEXlm εφαρμογές, προωθώντας τη σύνδεση μεταξύ τους στον κατάλληλο δαίμονα πωλητή που θα τις εξυπηρετήσει. Δουλειά του διαχειριστή αδειών είναι η εκκίνηση και ο τερματισμός των αντίστοιχων διεργασιών πωλητών. Μετά την πρώτη σύνδεση στον FLEXlm και την προώθηση της επικοινωνίας στον δαίμονα πωλητή, ο χρήστης-πελάτης δεν ξανασυνδέεται στον διαχειριστή.

Δαίμονας πωλητής αδειών (Vendor Daemon)

Στον FLEXlm, οι ρέουσες άδειες διαχειρίζονται από αντίστοιχες διεργασίες που τρέχουν πάνω στον εξυπηρετητή. Κάθε διεργασία είναι και ένα στιγμιότυπο του δαίμονα πωλητή που έχει ένα αδειοθετημένο [κατά FLEXlm] προιόν πάνω στο δίκτυο. Ο δαίμονας πωλητής καταγράφει ανά πάσα στιγμή το πόσες άδειες είναι δεσμευμένες και ποιος τις έχει δεσμεύσει.

Κάθε εταιρία λογισμικού που παρέχει τα προιόντα της με μοντέλο παροχής αδειών με σύγχονο τρόπο μέσω δικτύου, έχει κυκλοφορήσει άδειες συμβατές με FLEXlm. Αξίζει να αποσαφηνιστεί ότι η εταιρία παραμένει ο επίσημος πωλητής του λογισμικού, και όχι ο FLEXlm. Ο τελευταίος απλά παρέχει της άδειες μέσω της αντίστοιχης κατά προιόν διεργασίας πωλητή κατ' αίτηση του χρήστη πελάτη. Κάθε φορά που ο χρήστης επιθυμεί να

εκκινήσει μια εφαρμογή στέλνει μια αίτηση στον FLEXlm server η οποία εξυπηρετείται από τον δαίμονα διαχείρισης. Ο τελευταίος γνωστοποιεί στο χρήστη ποια είναι η κατάλληλη διεργασία πωλητή που θα τις εξυπηρετήσει και στη συνέχεια η επικοινωνία γίνεται με αυτήν η οποία και είναι υπεύθυνη να αποφανθεί αν είναι εφικτή η εκκίνηση της εφαρμογής ή όχι.

Τα δύο αυτά συσταστικά, ο License Manager και ο Vendor Daemon φαίνονται και στο ακόλουθο σχήμα.

Οι εφαρμογές που αδειοθετούνται μέσω του FLEXlm επικοινωνούνε με τον δαίμονα πωλητή μέσω TCP/IP συνδέσεων πάνω στο δίκτυο. Οι χρήστες-πελάτες και οι διεργασίες πωλητών (που βρίσκονται στον εξυπηρετητή – license server) μπορούν να βρίσκονται σε οποιοδήποτε σημείο ενός οποιαδήποτης κλιμακας τοπικού δικτύου. Επιπλέον η κίνηση που δημιουργείται πάνω στο δίκτυο μεταξύ των χρηστών –πελάτες της εφαρμογής- και του δαίμονα πωλητή δεν εξαρτάται από το είδος των υπολογιστικών συστημάτων που συμμετέχουν στην εν λόγω επικοινωνία, καθιστόντας την εφικτή για χρήση μεταξύ ετερογενών δικτύων. Αυτό σημαίνει πως μπορούν να υπάρχουν μηχανήματα με εντελώς διαφορετικά χαρακτηριστικά πάνω στο δίκτυο, τα οποία μπορεί να χρησιμοποιούν διαφορετικά λειτουργικά συστήματα (Windows και UNIX για παράδειγμα).

Αν για οποιοδήποτε λόγο ο δαίμονας πωλητής τερματίσει, τότε όλοι οι χρήστες που εξαρτώνται από αυτόν χάνουν τις άδειές τους (αν και αυτό δεν ισοδυναμεί με αυτόματο τερματισμό της εφαρμογής, καθώς έχει γίνει πρόβλεψη για δυνατότητα εκτέλεσής της για κάποιο σύντομο χρονικό διάστημα, ώστε να γίνεται εφικτή η αποθήκευση των δεδομένων που έχουν επεξεργαστεί). Οι τελικοί χρήστες συνήθως επαναδεσμεύουν τις άδειές τους όταν ο διαχειριστής αδειών επανακινήσει τον δαίμονα πωλητή, αν και υπάρχει η περίπτωση να αναγκαστούν να τερματίσουν την εφαρμογή αν ο πωλητής παραμένει μη προσπελάσιμος για αρκετό χρονικό διάστημα.

Αρχείο Παροχής Άδειας (License File)

Οι πληροφορίες σχετικά με την παροχή άδειας αποθηκεύονται σε ένα αρχείο κειμένου το οποίο ονομάζεται αρχείο παροχής άδειας (license file). Αυτό δημιουργείται από τον πωλητή του λογισμικού-εφαρμογής και εγκαταστήνεται και επεξεργάζεται από τον διαχειριστή αδειών. Περιλαμβάνει πληροφορίες σχετικά με τους υπολογιστικούς κόμβους που

λειτουργούν σαν εξυπηρετητές και τους δαίμονες πωλητές, ενώ περιέχει και τουλάχιστον μια γραμμή δεδομένων (που FEATURE ή INCREMENT γραμμή) για κάθε εφαρμογή που διαχειρίζεται ο FLEXlm server. Κάθε τέτοια γραμμή περιλαμβάνει ένα ηλεκτρονικό κλειδί ή ηλεκτρονική υπογραφή βασισμένο στα δεδομένα της γραμμής, τα hostids που ορίζονται στην(ις) SERVER γραμμή(ές) και διάφορα άλλα δεδομένα που σχετίζονται με τον δαίμονα πωλητή. Εδώ αξίζει να αναφέρουμε ότι για τις FLEXlm εφαρμογές που χρησιμοποιούνε το μοντέλο αδειοδότησης κλειδομένο στον κόμβο (node-locked) χρειάζεται μόνο να διαβάσουν το συγκεκριμένο αρχείο, χωρίς την παρεμβολή του εξυπηρετητή. Οι περισσότερες εφαρμογές έχουν μια προεπιλεγμένη τοποθεσία όπου αποθηκεύεται το αρχείο παροχής άδειας.

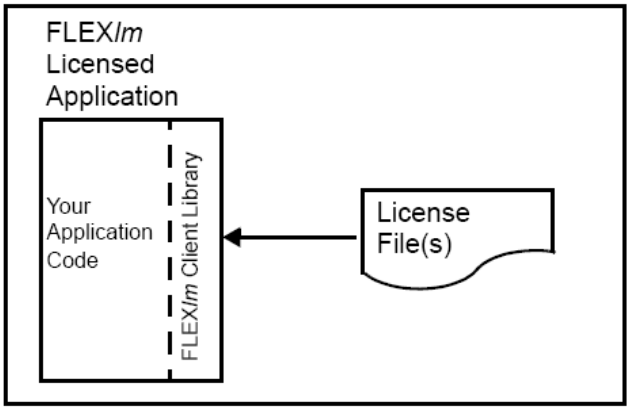
Παράδειγμα Αρχείου Παροχής Άδειας:

```
SERVER my_server 17007ea8 1700
VENDOR sampled
FEATURE f1 sampled 1.000 01-jan-2005 10 SIGN=9BFAC0316462
FEATURE f2 sampled 1.000 01-jan-2005 10 SIGN=1B9A308CC0F7
```

Η γραμμή SERVER ορίζει σαν εξυπηρετητή τον my_server με αναγνωριστικό 17007ea8 ο οποίος και ακούει στη θύρα 1700. Αντίστοιχα στη γραμμή VENDOR όρίζεται ότι η διεργασία του δαίμονα πωλητή θα είναι η sampled. Τέλος στις γραμμές FEATURE περιγράφονται οι άδειες που απαιτούνται για το προιόν. Εδώ υπάρχουν 10 άδειες για την εφαρμογή s1 με έκδοση 1.000 που θα εξυπηρετηθούν από τον πωλητή sampled, οι οποίες λήγουν την 01-01-2005, ενώ η υπογραφή πιστοποιεί το γνήσιο του προιόντος.

Η FLEXlm εφαρμογή

Η εφαρμογή που θα εκκινήσει με χρήση του FLEXlm συνδέεται με την αντίστοιχη βιβλιοθήκη χρήστη FLEXlm η οποία και παρέχει την επικοινωνία με τον εξυπηρετητή. Κατά τη διάρκεια της εκτέλεσης της εφαρμογής, η τελευταία επικοινωνεί με τον αντίστοιχο δαίμονα πελάτη για να κάνει αίτηση αδειοδότησης.



Εικόνα 2.3 – FLEXlm Εφαρμογή

Αρχείο Καταγραφής Σφαλμάτων (Debug Report File)

Το αρχείο καταγραφής σφαλμάτων περιλαμβάνει μηνύματα σφαλμάτων που εμφανίστηκαν κατά την εκτέλεση ή άλλα μηνύματα που σχετίζονται με την κατάσταση του συστήματος. Μερικά από αυτά τα μηνύματα περιλαμβάνουν πληροφορίες που σχετίζονται με τον δαίμονα διαχείρισης και άλλα σχετίζονται με τους δαίμονες πωλητές. Τα μηνύματα αυτά είναι χρήσιμα για την εύρεση σφαλμάτων στον κώδικα του license server και την επιδιόρθωσή τους

Αρχείο Επιλογών του Τελικού Χρήστη (End-User Administration Options File)

Το αρχείο επιλογών του τελικού χρήστη επιτρέπει τον τελικό χρήστη να ρυθμίσει διάφορες λειτουργικές παραμέτρους του FLEXlm, με σεβασμό στους κανόνες που ορίζονται στην άδεια και εξασφαλίζονται από τον δαίμονα πωλητή. Πιο συγκεκριμένα οι παράμετροι που μπορεί να ελέγξει ο χρήστης είναι:

- Να επιτρέψει τη χρήση επιπλέον χαρακτηριστικών της εφαρμογής.
- Να απαγορέψει τη χρήση συγκεκριμένων χαρακτηριστικών της εφαρμογής .
- Να κρατήσει ένα συγκεκριμένο αριθμό αδειών ως αποθεματικό για χρήση σε επίγουσες καταστάσεις.
- Να περιορίσει τον αριθμό των διαθέσιμων αδειών.
- Να ρυθμίσει το είδος και το πλήθος των μηνυμάτων σφαλμάτων που καταγράφονται από τον license server.

- Να επιτρέψει τη χρήση ενός αρχείου καταγραφής αναφορών.

Αρχείο Καταγραφής Αναφορών (Report Log File)

Το αρχείο καταγραφής αναφορών συγκεντρώνει πληροφορίες και στατιστικά από τη χρήση των προιόντων και δημιουργείται και επεξεργάζεται από τον δαίμονα πωλητή.

2.2.4 Εργαλεία Διαχείρησης του FLEXlm

Ο FLEXlm server περιέχει διάφορα εργαλεία στον διαχειριστή ώστε να τον βοηθήσει να ελέγξει διάφορες παραμέτρους που αφορούν τη διαδικασία παροχής άδειας λογισμικού στο δίκτυο. Οι λειτουργίες των κυριότερων εξ αυτών είναι οι εξείς:

Lmdiag: επιτρέπει τη διάγνωση προβλημάτων όταν δε μπορεί να ελεγχθεί μια άδεια. Αν ο έλεγχος πετύχει, το lmdiag το υποδεικνύει, αν όχι, καταγράφει το λόγο για τον οποίο απέτυχε.

Lmdown: επιτρέπει την απενεργοποίηση των επιλεγμένων FLEXlm servers

Lmhostid: επιστρέφει το host id της παρούσας πλατφόρμας

Lmreread: προκαλεί τον lmgrd να ξαναδιαβάσει το license file και να εκκινήσει καινούριους δαίμονες πωλητές, αν έχουν προστεθεί. Επιπλέον, όλοι οι υπάρχοντες πωλητές αναγκάζονται να ξαναδιαβάσουν το license file για να επαναρυθμίσουν τη λειτουργία τους με βάση τα δεδομένα που καταγράφονται στο συγκεκριμένο αρχείο.

Lmstat: βοηθά στον έλεγχο της κατάστασης όλης της δραστηριότητας που αφορά την παροχή άδειας με διάφορα στατιστικά που αφορούνε και τις διεργασίες που τρέχουν αλλά και τις ίδιες τις άδειες

2.2.5 Συνοπτική Περιγραφή της Διαδικασίας Απόκτησης Άδειας στο Σύγχρονο Μοντέλο (concurrent)

Έχοντας αναφέρει τα βασικά συστατικά του FLEXlm καθώς επίσης και αναλυτική περιγραφή των πιο σημαντικών από αυτών, ο αναγνώστης είναι σε θέση να γνωρίζει τη βασική λειτουργία του και τον τρόπο με τον οποίο δουλεύει. Ανακεφαλαιώνοντας, θα γίνει πλέον περιγραφή ολόκληρης της διαδικασίας απόκτησης άδειας στο σύγχρονο μοντέλο με χρήση του FLEXlm. Αυτή συνοψίζεται στα ακόλουθα βήματα:

1. Η FLEXlm βιβλιοθήκη του χρήστη-πελάτη της εφαρμογής εντοπίζει το license file το οποίο και περιλαμβάνει το δικτυακό αναγνωστιστικό του FLEXlm License Server καθώς επίσης και τον αριθμό της TCP/IP θύρας στη οποία ακούει ο δαίμονας διαχείρισης (τμήμα του License server).
2. Η FLEXlm εφαρμογή συνδέεται με τον δαίμονα διαχείρισης στην παραπάνω θύρα. Ο τελευταίος εντοπίζει τον δαίμονα πωλητή ο οποίος είναι υπεύθυνος για τη συγκεκριμένη εφαρμογή και στον οποίο θα πρέπει να απευθυνθεί η εφαρμογή.
3. Ο δαίμονας διαχείρισης εντοπίζει σε ποιον υπολογιστικό κόμβο και σε ποια TCP/IP θύρα βρίσκεται και ακούει ο πωλητής και γνωστοποιεί τα στοιχεία αυτά στην FLEXlm εφαρμογή. Απαραίτητο είναι εδώ να διευκρινιστεί ότι ο δαίμονας διαχείρισης έχει καθορίσει εξ αρχής ποιο μηχάνημα εκτελεί τον πωλητή, εδώ απλά τον εντοπίζει.
4. Η FLEXlm εφαρμογή πραγματοποιεί σύνδεση με τον αντίστοιχο δαίμονα πωλητή κσι του στέλνει αίτηση για παροχή άδειας.
5. Ο δαίμονας πωλητής ελέγχει αν υπάρχουν διαθέσιμες άδειες και στέλνει στην FLEXlm εφαρμογή την αποδοχή ή άρνηση στην αίτησή της.
6. Η FLEXlm βιβλιοθήκη του χρήστη-πελάτη δέχεται ή αρνείται την άδεια και η εφαρμογή εκκινεί ή όχι ανάλογα.

2.3 Υπηρεσιοστραφής Αρχιτεκτονική (Service Oriented Architecture – SOA)

Ένα από τα πιο σύνθετα προβλήματα κατά την ανάλυση και σχεδίαστη πληροφοριακών συστημάτων είναι η προσπάθεια να γίνουν όσο το δυνατόν πιο απλά, αλλά ταυτόχρονα λειτουργικά. Μια από τις πιο θεμελειώδεις αρχές, όπως έχει αναφέρει ο ίδιος ο Albert Einstein άλλωστε, είναι ότι τα πράγματα θα πρέπει να είναι όσο γίνεται απλά, αλλά όχι απλούστερα. Όπως όμως έχει δείξει η βιομηχανία λογισμικού, πολλές φορές αυτό δεν επιτυγχάνεται. Άλλες φορές τα πληροφοριακά συστήματα παραείναι απλοικά με αποτέλεσμα να μην ικανοποιούν τις ανάγκες για τις οποίες αναπτύχθηκαν, ενώ άλλες είναι τόσο σύνθετα που το κόστος ανάπτυξης και κυρίως συντήρησής τους καθιστάται απαγορευτικό, χωρίς να αναφέρουμε τη σχεδόν αδύνατη ολοκλήρωση δύο διαφορετικών τέτοιων συστημάτων [4].

2.3.1 Πραγματικές και Τεχνικές Εξαρτήσεις

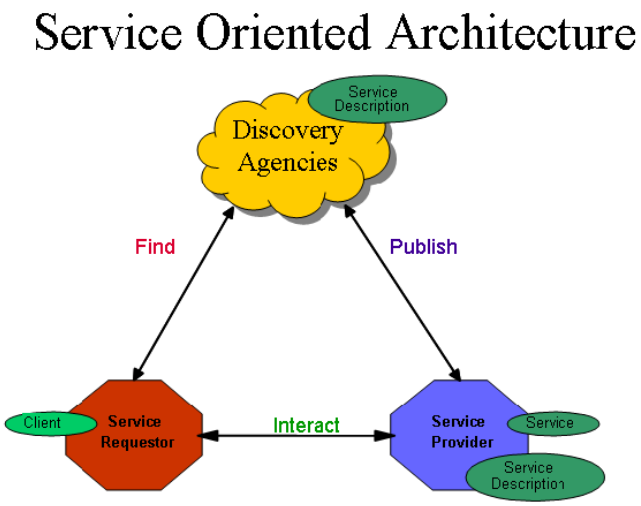
Δε χρειάζεται να ψάξει κανείς πολύ για να συναντήσει το πρόβλημα που έχει να αντιμετωπίσει. Όσο αναπτύσσονται προγράμματα λογισμικού, αναπόφεκτα εμφανίζονται οι ίδιες καταστάσεις και προβλήματα που πρέπει να επιλυθούν. Ιδανικά θα ήταν χρήσιμο να μπορεί να επαναχρησιμοποιηθεί μια λειτουργία ενός υπάρχον λογισμικού για να δώσει λύσεις σε μια κατάσταση, παρά να αναπτυχθεί το συγκεκριμένο κομμάτι του συστήματος από την αρχή. Μια πραγματική εξάρτηση είναι η κατάσταση κατά την οποία ένα σύστημα εξαρτάται από τη λειτουργικότητα που παρέχεται από κάποιο άλλο. Αν ο κόσμος περιείχε μόνο πραγματικές εξαρτήσεις, τότε η πρόταση του Einstein θα είχε ικανοποιηθεί εδώ και πολύ καιρό. Το πρόβλημα όμως είναι ότι υπάρχουν και τεχνικές εξαρτήσεις πέρα από τις πραγματικές.

Η διάκριση μεταξύ τεχνικών και πραγματικών εξαρτήσεων μπορεί να γίνει κατανοητή μέσω ενός χαρακτηριστικού προβλήματος. Ας υποθέσουμε ότι κάποιος κάνει ένα υπερατλαντικό ταξίδι. Ένα από τα πράγματα που θα χρειαστεί είναι προσαρμογείς για τους ακροδέκτες ηλεκτρικών συσκευών. Αυτό είναι απαραίτητο καθώς οι ακροδέκτες είναι διαφορετικοί ανάλογα με την ήπειρο και τη χώρα που κάποιος διαμένει. Η διάκριση μεταξύ πραγματικής και τεχνικής εξάρτησης είναι εμφανής. Πραγματική εξάρτηση αποτελεί η ανάγκη για

τροφοδοσία από ηλεκτρικό ρεύμα. Η ανάγκη εξάρτηση αυτή είναι επιλυμένη. Παντού υπάρχει παροχή ηλεκτρικού ρεύματος. Τεχνική εξάρτηση αποτελεί η ανάγκη για ταίριασμα μεταξύ των δύο διαφορετικών συσκευών. Του ακροδέκτη της ηλεκτρικής συσκευής εν προκειμένου, και του παρουχέα του ηλεκτρικού ρεύματος.

Παρότι φαίνεται ότι δε γίνεται να εξαφανίστεί εντελώς η ύπαρξη των τεχνικών εξαρτήσεων, το σίγουρο είναι ότι μπορεί να ελλατωθεί. Αν με κάποιο τρόπο γινόταν μειωθούν αυτές στο ελάχιστο, τότε η αρχή του Einstein θα μπορούσε να ικανοποιηθεί. Μεταφράζοντας κάπως την περίφημη αρχή που διατύπωσε, θα μπορούσε να ειπωθεί πως οι τεχνικές εξαρτήσεις θα πρέπει να μειώνονται στο ελάχιστο δυνατόν, ενώ αντίθετα οι πραγματικές εξαρτήσεις θα πρέπει να μείνουν ανεπηρέαστες.

2.3.2 Ορισμός της Υπηρεσιοστρεφούς Αρχιτεκτονικής (SOA)



Εικόνα 2.4 – Υπηρεσιοστρεφής Αρχιτεκτονική

Έχοντας κατανοήσει τη διάκριση μεταξύ τεχνικών και πραγματικών εξαρτήσεων, μπορεί να δοθεί ένας ορισμός της υπηρεσιοστραφούς αρχικτεκτονικής (service oriented architecture – SOA). Αυτή ορίζεται ως αρχιτεκτονική πρακτική της οποίας ο στόχος είναι επιτευχθούν οι λιγότερες δυνατές εξαρτήσεις μεταξύ δύο διαδραστικών οντοτήτων λογισμικού. Με τον όρο υπηρεσία ορίζεται ως ένα κομμάτι λογισμικού που έχει αναπτυχθεί από τον παροχέα της υπηρεσίας και το οποίο ικανοποιεί κάποιες απαιτήσεις ώστε να παρέχει τα επιθυμητά τελικά αποτελέσματα στον καταναλωτή της υπηρεσίας. Ο παροχέας και ο καταναλωτής των

υπηρεσιών είναι ρόλοι που παίζουν αυτά τα στοιχεία λογισμικού για λογαριασμό των ιδιοκτητών τους.

Αν και ο παραπάνω ορισμός ακούγεται πολύ αφηρημένος, στην πραγματικότητα η υπηρεσιοστραφής αρχιτεκτονική βρίσκεται παντού. Ενώ φαίνεται ίδιος με τη βασική αρχή του αντικειμενοστραφούς προγραμματισμού, έχει σημαντικές διαφορές. Κατά τον τελευταίο, συσχετιζόμενα δεδομένα και συναρτήσεις που τα επεξεργάζονται πρέπει να βρίσκονται κάτω από την ίδια οντότητα λογισμικού, σύμφωνα με την υπηρεσιοστραφής όμως αρχιτεκτονική αυτό δεν προκύπτει. Αντίθετα, μια υπηρεσία αποτελούν συνήθως μόνο οι συναρτήσεις που επεξεργάζονται τα δεδομένα, ή με βάση τον ορισμό που δόθηκε παραπάνω, μια υπηρεσία αποτελείται συγκεκριμένα τμήματα/συναρτήσεις λογισμικού που παρέχουν τα επιθυμητά αποτελέσματα στον καταναλωτή. Υπάρχει με άλλα λόγια διάκριση μεταξύ δεδομένων και επεξεργασίας. Στην ουσία τα αποτελέσματα που παρέχει μια υπηρεσία αποτελούν την αλλαγή στην κατάσταση του καταναλωτή, αν και μερικές φορές μπορούν να επιφέρουν αλλαγή στην κατάσταση του παροχέα ή και των δύο.

Ένας από τους κυριότερους λόγους για τον οποίο κάποιος τρίτος ανατήθεται για να παράγει την εργασία κάποιου άλλου είναι κυρίως γιατί είναι και πιο ειδικός για να παράγει το συγκεκριμένο έργο, άρα και να το παράγει καλύτερα και αποδοτικότερα. Συχνά άλλωστε είναι πιο εύκολο και πιο οικονομικό για κάποιον προγραμματιστή να χρησιμοποιήσει μια οντότητα λογισμικού που έχει αναπτύξει κάποιος άλλος από το να την αναπτύξει ο ίδιος από την αρχή. Πέρα από το τεχνικό κομμάτι και την εξειδικευμένη γνώση που μπορεί να κατέχει ο άλλος καλύτερα, το συγκεκριμένο τμήμα έχει χρησιμοποιηθεί και από άλλους καταναλωτές της υπηρεσίας, άρα και έχει ελεγχθεί στο ότι παράγει εγγυημένα τα επιθυμητά αποτελέσματα. Αυτό και μόνο γλιτώνει πολύ χρόνο από τον αρχικό προγραμματιστή που επιθυμεί να χρησιμοποιήσει τη συγκεκριμένη υπηρεσία. Έτσι προκύπτει αυτό που ονομάζεται στην τεχνολογία λογισμικού ως “διάκριση ευθυνών” και είναι από τις σημαντικότερες αρχές της επιστήμης λογισμικού.

Στην αρχή του κεφαλαίου αναφέρθηκε ότι η υπηρεσιοστραφής αρχιτεκτονική μειώνει στο ελάχιστο τις διάφορες τεχνικές εξαρτήσεις μεταξύ των διαδραστικών οντοτήτων λογισμικού. Αυτό γίνεται υιοθετώντας δύο αρχιτεκτονικούς περιορισμούς:

1. Υπάρχουν μικρές και πάντα διαθέσιμες διεπαφές μεταξύ όλων των οντοτήτων που συμμετέχουν. Μόνο αφηρημένα σχήματα περιγράφονται στις συγκεκριμένες διεπαφές. Οι διεπαφές επιπλέον είναι καθολικά διαθέσιμες για όλους τους παρόχους και καταναλωτές.

2. Υπάρχει ένα σχήμα με δυνατότητες επέκτασης το οποίο και περιγράφει το πώς θα ανταλλάσονται τα μηνύματα μεταξύ των διεπαφών. Καμία ή το πολύ ελάχιστη πληροφορία περιγράφεται που να είναι συσχετιζόμενη με καθ' αυτή τη λειτουργία του συστήματος. Το σχήμα άλλωστε ελλατώνει το λεξιλόγιο και τη δομή των μηνυμάτων. Το γεγονός ότι το σχήμα επίσης μπορεί να επεκταθεί επιτρέπει την ύπαρξη νέων εκδόσεων υπηρεσιών χωρίς να καταργούνται οι υπάρχουσες.

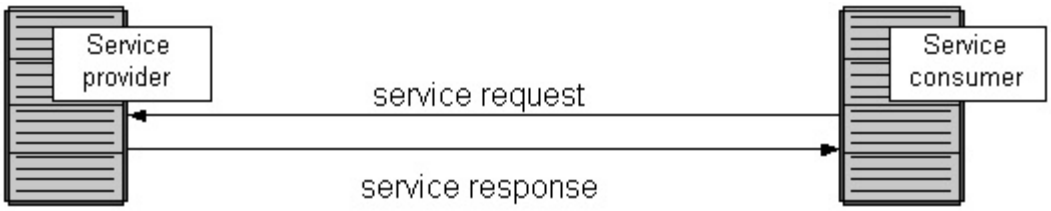
Όπως γίνεται κατανοητό, η ύπαρξη και δομή των διεπαφών είναι από τα σημαντικότερα μέρη. Αν οι διεπαφές δε λειτουργούν σωστά, τότε και ολόκληρο το σύστημα δε θα λειτουργήσει σωστά κατά τον ίδιο τρόπο. Επιπλέον είναι αρκετά χρονοβόρες για να σχεδιαστούν και να υλοποιηθούν σωστά, ενώ παράλληλα θεωρούνται μήτρες συνεχόμενης παραγωγής σφαλμάτων στα κατανεμημένα κατά αυτόν τον τρόπο συστήματα. Μια διεπαφή προδιαγράφει τη συμπεριφορά ενός συστήματος, για αυτόν ακριβώς το λόγο και είναι πολύ δύσκολο να αναπτυχθεί σωστά μεταξύ διαφορετικών πλατφόρμων και προγραμματιστικών γλωσσών. Οι απομακρυσμένες διεπαφές εξ' άλλου αποτελούν και το πιο αργό κομμάτι ενός κατανεμημένου συστήματος. Για όλους αυτούς τους λόγους είναι προτιμότερο να γίνεται επαναχρησιμοποίηση υπάρχοντων τέτοιων δομών, αντί να δημιουργούνται καινούριες κάθε φορά για κάθε νέα εφαρμογή.

Για να θεωρηθεί η αρχιτεκτονική ως υπηρεσιοσταφής είναι αναγκαίο να οριστεί η σημασιολογία των μηνυμάτων που θα ανταλλάστονται σε κάθε εφαρμογή. Αυτή γίνεται με σεβασμό σε τέσσερεις βασικές αρχές. Πρώτον τα μηνύματα πρέπει να ορίζουν τα δεδομένα προς επεξαργασία, αλλά σε καμία περίπτωση δεν πρέπει να ορίζουν το πώς θα επεξαργαστούν. Αυτό είναι αντικείμενο ευθύνης της υπηρεσίας. Δεύτερον οι παροχείς των υπηρεσιών θα πρέπει να είναι σε θέση να κατανοούν την αίτηση του καταναλωτή. Αυτό γίνεται υιοθετώντας μια συγκεκριμένη δομή και λεξιλόγιο, το οποίο είναι κατανοητό και γίνεται σεβαστό από όλες τις οντότητες που αλληλεπιδρούν σε αυτή την επικοινωνία. Τρίτον, απαραίτητη θεωρείται η δυνατότητα επέκτασης και αναβάθμισης. Αυτό εξασφαλίζει την προσαρμογή της υπηρεσίας σε οποιαδήποτε μεταβολή μπορεί υπάρξει στις απαιτήσεις που πρέπει να ικανοποιεί, συνεχίζοντας όμως να εξυπηρετεί καταναλωτές με διαφορετικές-παλιότερες- απαιτήσεις. Τέταρτον τέλος, η υπερησιοστρεφής αρχιτεκτονική θα πρέπει να έχει έναν μηχανισμό που θα δίνει τη δυνατότητα στους καταναλωτές να ανακαλύπτουν τους παρόχους των υπηρεσιών που τους καλύπτουν.

Πρακτικά η υπηρεσιοστραφής αρχιτεκτονική αποτελεί έναν τρόπο ένα τρόπο σχεδίασης ενός συστήματος λογισμικού ώστε αυτό να μπορεί να παρέχει υπηρεσίες είτε σε εφαρμογές οι

οποίες θεωρούμε ότι αποτελούν τον τελικό χρήστη, είτε σε άλλες υπηρεσίες μέσω δημοσιευμένων και γνωστών διεπαφών. Κατά αυτό τον τρόπο δίνεται ένας καλύτερος τρόπος να εκτεθούν οι λειτουργίες μια επιχείρησης άρα και ένας άριστος τρόπος να αναπτυχθούν εφαρμογές οι οποίες θα βασίζονται σε αυτές ακριβώς τις λειτουργίες. Τελικά η υπηρεσιοστρεφής αρχιτεκτονική δεν είναι απλά μια αυθαίρετη έννοια, αλλά μια εξαιρετικά σημαντική αυτή την εποχή αρχιτεκτονική εξαιτίας της ολοένα και αναδυόμενης χρήσης των διαδικτυακών υπηρεσιών [5].

Η ιδέα της SOA και η λειτουργία της φαίνονται ακολούθως:



Εικόνα 2.5 – Υπηρεσία SOA και Συνδέσεις

- Υπηρεσία: Μια λογική οντότητα. Ο ρόλος της προσδιορίζεται από μια ή περισσότερες δημοσιευμένες διεπαφές.
- Πάροχος υπηρεσίας: Η οντότητα λογισμικού που υλοποιεί μια προδιαγραφή της υπηρεσίας
- Καταναλωτής υπηρεσίας: Η οντότητα λογισμικού η οποία καλεί έναν πάροχο υπηρεσίας [6].

2.3.3 Επιπρόσθετα Χαρακτηριστικά

Υπάρχει ένα σύνολο επιπρόσθετων χαρακτηριστικών και περιορισμών που μπορούν να έχουν εφαρμογή στην υπηρεσιοστρεφής αρχιτεκτονική και προσφέρουν πλεονεκτήματα σχετικά με την επέκταση, την απόδοση και την αξιοπιστία της.

Stateless Υπηρεσία

Κάθε μήνυμα που στέλνει ο καταναλωτής στον παροχέα θα πρέπει να περιλαμβάνει όλη την απαιτούμενη πληροφορία για τον παροχέα ώστε να την επεξεργαστεί. Αυτός ο περιορισμός βοηθάει στην ευκολότερη επέκταση της υπηρεσίας από τον παροχέας, καθώς ο τελευταίος δεν είναι αναγκασμένος να αποθηκεύει πληροφορίες μεταξύ των αιτημάτων. Αυτό βοηθάει επίσης και στη μαζικότερη χρήση της υπηρεσίας από πολλαπλούς καταναλωτές ταυτοχρόνως, καθώς κάθε αίτημα μπορεί να θεωρηθεί μοναδικό, με την έννοια ότι για κάθε ένα υπάρχει ακριβώς μία απόκριση. Επιπλέον έχει ισχυριθεί ότι αυτή η πρακτική βοηθάει στον καλύτερο εντοπισμό σφαλμάτων επειδή είναι ξεκάθαρο για οποιοδήποτε αίτημα ποια θα είναι η απάντησή του. Τέλος δεν υπάρχει κάποια ενδιάμεση κατάσταση στην υπηρεσία να λάβει ο προγραμματιστής κατά νου, και αυτό δίνει δυνατότητα για άμεση ανάκαμψη όταν η υπηρεσία είναι για παράδειγμα ανενεργή, καθιστόντας την πιο αξιόπιστη.

Stateful Υπηρεσία

Οι συγκεκριμένες υπηρεσίες είναι σχεδόν απαραίτητο να χρησιμοποιούνται σε μια πληθώρα περιπτώσεων με πιο χαρακτηριστική την ανάγκη για ύπαρξη συνεδρίας (session) μεταξύ του καταναλωτή και του παροχέα. Οι κυριότερος λόγοι για ύπαρξη συνεδρίας είναι λόγοι απόδοσης. Για παράδειγμα, η συνεχόμενη αποστολή πιστοποιητικών ασφαλείας μεταξύ των μηνυμάτων των δύο αλληλεπιδρόμενων οντοτήτων είναι ένα σοβαρό πρόβλημα τόσο για τον πάροχο όσο και για τον καταναλωτή. Είναι πολύ πιο αποδοτικό να αντικατασταθούν τα πιστοποιητικά με κάποιο σύμβολο το οποίο το διαμοιράζονται τα δύο μέρη. Ένα άλλο παράδειγμα εφαρμογής είναι η ανάγκη για παροχή πελατοκεντρικής εξυπηρέτησης. Παρόλα αυτά, το συγκεκριμένο είδος υπηρεσίας έχει αρκετά μειονεκτήματα, καθώς μειώνεται η συνολική δυνατότητα επέκτασης της υπηρεσίας καθώς η τελευταία θα πρέπει να είναι σε θέση να αποθηκεύει το περιεχόμενο (context) του κάθε χρήστη. Αποτέλεσμα αυτού είναι και η δυσκολία από μεριάς καταναλωτή να επιλέξει διαφορετικό παροχέα [7].

2.3.4 Υπηρεσίες Υπηρεσιοστρεφούς Αρχιτεκτονικής

Ενώ είναι πολύ εύκολο για κάποιον να κατανοήσει το τι είναι μια υπηρεσία, παρόλα αυτά είναι πολύ δύσκολο να δοθεί κάποιος συγκεκριμένος και καθολικά αποδεκτός ορισμός. Το ζήτημα αυτό είναι ακόμα αντικείμενο μελέτης και αφορμή συζητήσεων στο "W3C Web Services Architecture Working Group". Παρά τις δυσκολίες εύρεσης ενός ικανοποιητικού

ορισμού, είναι γενικά αποδεκτό ότι μια υπηρεσία αποτελεί κομμάτι της υπηρεσιοστρεφούς αρχιτεκτονικής όταν ικανοιποιεί τουλάχιστον τους κάτωθι περιορισμούς:

1. Οι διεπαφές της είναι βασισμένες σε διαδικτυακά πρωτόκολλα, όπως το HTTP, FTP και SMTP.
2. Τα μηνύματα που ανταλλάσονται είναι XML [8] μορφής, εκτός από τα δυαδικά επισυναπτόμενα δεδομένα.

Υπάρχουν δύο ειδών διαδικτυακές υπηρεσίες, οι SOAP [9]και οι REST [10].

Υπηρεσίες SOAP

Μια SOAP υπηρεσία πρέπει να ικανοποιεί τις ακόλουθες απαιτήσεις

1. Όλα τα μηνύματα μεταφέρονται μέσω του SOAP πρωτοκόλλου, εκτός από τα δυαδικά επισυναπτόμενα δεδομένα. Αξίζει να αποσαφηνιστεί ότι τα SOAP μηνύματα μεταφέρονται μέσω διαδικτυακών πρωτοκόλλων, όπως ακριβώς απαιτεί ο ορισμός των διαδικτυακών υπηρεσιών.
2. Η περιγραφή της υπηρεσίας γίνεται με τη γλώσσα περιγραφής WSDL [11].

Οι SOAP υπηρεσίες είναι οι πιο διαδεδομένες στη βιομηχανία λογισμικού. Η διάδοσή τους είναι τόσο ευρή που συχνά ταυτίζουμε όλες τις υπηρεσίες με τις SOAP/WSDL-ed. Το κυριότερο πλεονέκτημα τους είναι η εξασφάλιση της μεταφοράς του μηνύματος διαμέσω ετερογενών δικτύων, χωρίς να ενδιαφέρει ποιο πρωτόκολλο κάνει πράγματι τη μεταφορά. Με άλλα λόγια το SOAP παίζει το ρόλο ενός φακέλου που μεταφέρει το περιεχόμενό του. Ένα ακόμα πλεονέκτημα του SOAP είναι ότι προσφέρει μπορεί να μεταφέρει μηνύματα με διάφορους τρόπους, από την απλή περίπτωση ερώτησης-απάντησης μέχρι περιπτώσεις εκπομπής (broadcast) και άλλες πιο σύνθετες.

Υπηρεσίες REST

Ο όρος REST υιοθετήθηκε για πρώτη φορά από τον Roy Fielding για να περιγράψει μια διαδικτυακή αρχιτεκτονική. Μια REST υπηρεσία είναι υπηρεσία υπηρεσιοστρεφής αρχιτεκτονικής, βασισμένη στους πόρους (resources). Σαν πόρος εννοείται οτιδήποτε έχει μια URI. Ο πόρος μπορεί να έχει από καμία μέχρι περισσότερες αναπαραστάσεις. Συνήθως αναφέρεται ότι ο πόρος δεν υπάρχει αν δεν υπάρχει τουλάχιστον μια αναπαράστασή του.

Μια REST υπηρεσία πρέπει να ικανοποιεί τις αντίστοιχες απαιτήσεις:

1. Οι διεπαφές είναι βασισμένες μόνο στο HTTP διαδικτυακό πρωτόκολλο. Αυτά που ορίζονται ως εκ τούτου είναι τα HTTP GET/DELETE/POST/PUT για ανάκτηση μια αναπαράστασης, διαγραφής, επεξαργασίας και δημιουργίας της αντίστοιχα.
2. Τα περισσότερα μηνύματα είναι XML όπως αυτή ορίζεται στο XML Schema 1.1 της W3C [12] ή το RELAX NG.
3. Απλά μηνύματα μπορούν να κωδικοποιηθούν με βάση την URL κωδικοποίηση.
4. Οι υπηρεσίες και οι πάροχοι των υπηρεσιών πρέπει να είναι πόροι (resources) ενώ ο καταναλωτής μπορεί να είναι πόρος.

Εν τέλη, αυτό που γίνεται κατανοητό από τα παραπάνω είναι ότι οι REST υπηρεσίες απαιτούν ελάχιστη από τη διαθέσιμη υποδομή, καθώς απαιτούν ανάπτυξη μόνο σύμφωνα με το HTTP πρωτόκολλο και το XML σχήμα, τεχνολογίες που πλέον υποστηρίζονται από τις περισσότερες γλώσσες προγραμματισμού και πλατφόρμες. Οι REST υπηρεσίες είναι απλές και αποδοτικές επειδή αφενός το HTTP είναι το πιο διαδεδομένο διαδικτυακό πρωτόκολλο, και αφετέρου γιατί είναι ικανοποιητικό για τις περισσότερες των εφαρμογών. Στις περισσότερες περιπτώσεις, η απλότητα του HTTP είναι προτιμότερη από την χρήση ενός επιπλέον στρώματος μεταφοράς, όπως το SOAP.

2.3.5 Κριτικές πάνω στην Υπηρεσιοστρεφή Αρχιτεκτονική

Μία από τις πιο σοβαρές κριτικές που έχει δεχθεί η συγκεκριμένη αρχιτεκτονική είναι ότι αποτελεί στην ουσία μια ακόμα προσπάθεια να δοθεί ορισμός στο τι ακριβώς είναι μια διαδικτυακή υπηρεσία. Για παράδειγμα πολλοί ισχυρίζονται ότι η SOA καταλήγει απλά να χρησιμοποιεί επιπλέον στρώματα λογισμικού για την επεξαργασία των XML μηνυμάτων. Ιδιαίτερα αν απουσιάζουν βιβλιοθήκες που επιτρέπουν την αποδοτική κληση διαδικασιών από απόσταση (Remote Procedure Call – RPC) οι εφαρμογές καταλήγουν να τρέχουν αργά και να καταναλώνουν πολύ περισσότερη υπολογιστική ισχύ, αυξάνοντας κατακόρυφα το κόστος. Πολλές από τις εφαρμογές που αναπτύσσονται με SOA πραγματικά υποφέρουν από τα παραπάνω, παρόλα αυτά πλέον η SOA μπορεί να υλοποιηθεί με χρήση τεχνολογιών (όπως το Java Business Integration – JBI) που δεν εξαρτόνται από τις remote procedure calls, ούτε απαιτούν μετάφραση των XML μηνυμάτων.

Όπως αναφέρθηκε παραπάνω, στις stateful υπηρεσίες δημιουργείται η απαίτηση να μοιράζεται ένα κοινό λογισμικό περιεχόμενο βασισμένο στον καταναλωτή (consumer-specific context) μεταξύ του τελευταίου και του παρόχου. Αυτό έχει σαν αποτέλεσμα να

μειώνει τη συνολική δυνατότητα επέκτασης της υπηρεσίας, ενώ καθιστά πολύ δύσκολο για τον καταναλωτή να επιλέξει διαφορετικό παροχέα. Έτσι πολλοί κατηγορούν τη SOA ότι περιορίζεται στην ουσία από τις εφαρμογές στις οποίες υλοποιείται.

Ένα από τα πιο μεγάλα προβλήματα που εμφανίζονται αναπτύσσοντας εφαρμογές βασισμένες στη SOA είναι επίσης η έλλειψη τεκμηριωμένου ελέγχου. Δεν υπάρχουν εργαλεία που θα μπορούν να παρέχουν τεκμηριωμένο έλεγχο για όλες τις υπηρεσίες που αλληλεπιδρούν σε μια τυπική αρχιτεκτονική. Αυτή η έλλειψη οριζόντιας εμπιστοσύνης μεταξύ του παροχέα και του καταναλωτή απαιτεί να γίνεται έλεγχος της υπηρεσίας και από τους δύο αλληλεπιδρόμενους σε καθημερινή βάση. Εκτός αυτού, είναι επιπλέον πολύ δύσκολο να υπάρχουν συγκεκριμένα επίπεδα ασφάλειας. Τα μοντέλα ασφάλειας που έχουν αναπτυχθεί για αυτό το σκοπό αποτυγχαίνουν καθώς μία υπηρεσία δημοσιευμένη παρέχει δεδομένα σε μια άλλη που θεωρεί ως έμπιστη, η οποία όμως με τη σειρά της λειτουργεί ως πάροχος σε τρίτους καταναλωτές με τους οποίους την πρώτη δεν την συνδέδουν κανένα πιστοποιητικό ασφάλειας. Καθώς βρισκόμαστε σε ένα ολοένα και πιο ολοκληρωμένο διαδικτυακό περιβάλλον, όπου εφαρμογές επικοινωνούν μεταξύ τους μέσω διεπαφών και αυτές με άλλες δια μέσου ετερογενών δικτύων, η ανάγκη να καθοριστούν σαφώς τα επίπεδα ασφαλείας είναι κάτι παραπάνω από κρίσιμη.

Ένα ακόμα ζήτημα είναι ο ορισμός των διαφόρων σχημάτων που περιγράφουν τα XML μηνύματα που ανταλλάσονται μεταξύ των δύο αλληλεπιδρόμενων οντοτήτων. Επειδή όλα τα WS-* σχήματα συνεχίζουν να αναπτύσσονται, υπάρχει το ρίσκο να δημιουργηθούνε πρότυπα που αλληλοαναιρούνται, καθιστώντας στην ουσία αδύνατη την οποιαδήποτε τεκμηρίωση των πρωτοκόλλων και σχημάτων που χρησιμοποιεί η SOA [13].

2.4 Τεχνολογίες Πολυπλέγματος (GRID)

2.4.1 Σύντομη Ιστορική Αναδρομή

Η πρώτη προσπάθεια σύνδεσης δύο διαφορετικών υπολογιστών σε κοινό δίκτυο έγινε στις αρχές της δεκαετίας του 1970, όπου και γεννήθηκε και η ιδέα χρησιμοποίησης αδέσμευτων πόρων, και η λειτουργία του προγόνου του σημερινού internet, του APRANET. Με την εγκατάσταση του πρώτου δικτύου με χρήση ethenet από την Xerox πραγματοποιήθηκε και η πρώτη ουσιαστική προσπάθεια για κατανεμημένη υπολογιστική εφαρμογή. Η πραγματική καινοτομία έγινε από την Apple και τον επιστήμονα Richard Crandall ο οποίος κατάφερε να βάλει αδρανείς δικτυακά συνδεδεμένους υπολογιστές να εργάζονται και να πραγματοποιούν υπολογισμούς συνδιάζοντας τα αποτελέσματα μεταξύ τους πάνω στο δίκτυο. Ήταν η γέννηση των κατανεμημένων υπολογιστικών συστημάτων.

Οι καταναμημένες υπηρεσίες άρχισαν να βρίσκουν ευρεία εφαρμογή με την ωρίμανση και εξάπλωση του διαδικτύου, τη δεκαετία του 1990. Τότε έγινε ένα από τα πιο φιλόδοξα ερευνητικά προγράμματα με όνομα SETI@home project. Η ιδέα ήταν να καταφέρει να συνδεθεί μέσω του διαδικτύου ένας πάρα πολύ μεγάλος αριθμός υπολογιστικών συστημάτων, ο οποίος θα μπορούσε να εκτελεί σαν ενιαία οντότητα πολύπλοκους υπολογισμούς. Πράγματι, πάνω από δύο εκατομμύρια άνθρωποι συμετείχαν αποδεικνύοντας ότι η τεχνολογία του κατανεμημένου υπολογισμού θα μπορούσε να επιταχύνει τα αποτελέσματα υπολογιστικών ερευνών, κρατώντας παράλληλα το κόστος σε πολύ χαμηλά επίπεδα [14].

Η δημοφιλία του Internet καθώς επίσης και η διαθεσιμότητα σε πολύ ισχυρούς υπολογιστές και τεχνολογίες διαδικτύου μεγάλης ταχύτητας, όπως επίσης και το πολύ χαμηλό κόστος όλων των παραπάνω, έχουν αλλάξει τον τρόπο με τον οποίο χρησιμοποιούνται οι υπολογιστές σήμερα. Αυτά τα χαρακτηριστικά που πλέον είναι διαθέσιμα σε όλους, προσφέρουν τη δυνατότητα χρήσης κατανεμημένων υπολογιστών σαν απλή και ενιαία υπολογιστική οντότητα, καταλήγοντας σε αυτό που σήμερα ονομάζεται ως τεχνολογίες πολυπλέγματος. Το πολυπλέγμα δίνει τη δυνατότητα κοινής χρήσης, επιλογής και συσσωμάτωσης μιας τεράστιας ποικιλίας υπολογιστικών πόρων, όπως οι super υπολογιστές, μηχανήματα αποθήκευσης, και συσκευές ειδικού τύπου που είναι γεωγραφικά κατανεμημένες και ανήκουν σε διαφορετικούς οργανισμούς και χρησιμοποιούνται για διαφορετικές

εφαρμογές και ανάγκες κάθε φορά. Οι τεχνολογίες πολυπλέγματος ξεκίνησαν σαν ερευνητικό πρόγραμμα που σα στόχο είχε να συνδέσει τελείως απομακρυσμένους super υπολογιστές, αλλά τώρα η χρήση τους έχει αλλάξει κατά πολύ.

2.4.2 Τι είναι οι Τεχνολογίες Πολυπλέγματος

Ο όρος τεχνολογίες πολυπλέγματος δόθηκε το 1998 από τους Ian Foster και Carl Kesselman για να περιγρέψει τα κατανεμημένα υπολογιστικά και αποθηκευτικά περιβάλλοντα. Ο όρος αυτός αποτελεί έναν υπολογιστικό πρότυπο που μοιάζει με ένα πολύπλεγμα ηλεκτρικής ισχύος, αποτελούμενο από πολλούς πόρους που παράγουν ισχύ σε ένα κοινό απόθεμα, που θα μπορούν να το χρησιμοποιούν οι πελάτες σύμφωνα με τις ανάγκες τους. Στόχος ήταν να παρέχει μεγάλη υπολογιστική ισχύ χωρίς την ανάγκη για εκατάσταση και συντήρηση πολύπλοκων υποδομών λογισμικού σε κάθε γεωγραφική περιοχή που υπήρχε η ανάγκη για εκτέλεση ακριβών σε υπολογιστική ισχύ εφαρμογών. Το όραμα ήταν η ύπαρξη μιας ενιαίας εικονικής υποδομής λογισμικού. Αυτό δεν έχει ακόμα πραγματοποιηθεί, παρόλα αυτά σήμερα μπορεί κάποιος να διακρίνει ολοένα και περισσότερα χαρακτηριστικά να αναπτύσσονται και να προτυτοποιούνται .

Εικόνα 2.6 – Ολοκλήρωση Μέσω Τεχνολογιών Πολυπλέγματος

Το μεγαλύτερο στοίχημα στις τεχνολογίες πολυπλέγματος είναι χωρίς αμφιβολία η απλοποίηση και βελτιστοποίηση των πληροφοριακών πόρων. Η απλοποίηση υλοποιείται από

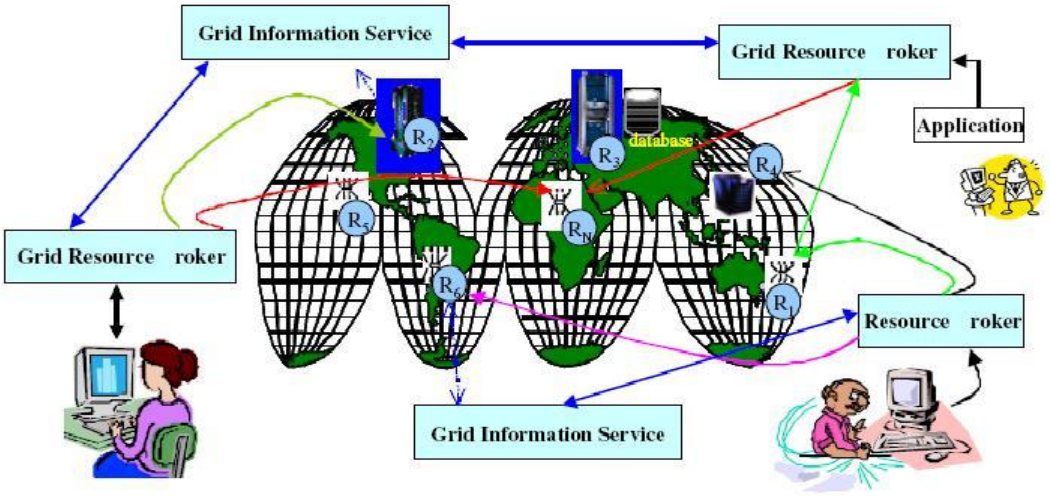
το γεγονός ότι οι χρήστες πλέον μπορούν να συγκεντρωθούν μόνο στα τμήματα λογισμικού που τους αφορούν, δηλαδή τις εφαρμογές-πελάτες που αναπτύσσουν, χωρίς να ενδιαφέρονται να συντηρούν μια πολύπλοκη και ολοκληρωμένη υπολογιστική υποδομή. Η βελτιστοποίηση υλοποιείται καθώς τα κέντρα αποθήκευσης δεδομένων δε χρειάζεται να εξαρτόνται από το μέγεθος των δεδομένων που αποθηκεύουν, που είχε σαν αποτέλεσμα τη χρήση πανάκριβων μηχανημάτων, αλλά μπορούν με ένα πολύ αποδοτικό τρόπο να τα μοιράζουν σε κατανεμημένους κόμβους. Πλέον, η παγκόσμια κατανεμημένη υποδομή που παρέχουν οι τεχνολογίες πολυπλέγματος είναι μια πραματικότητα σε εταιρίες κολοσσούς όπως η IBM, Shell, Phillips, NASA κτλ, όπου μπορούν να βάζουν εργασίες προς εκτέλεση οποιαδήποτε χρονική στιγμή, σε οποιοδήποτε μέρος, με τα δεδομένα να ρρέουν πάνω σε όλο το διαδίκτυο.

Ένα εύλογο ερώτημα που προκύπτει από τα παραπάνω είναι σε τι διαφέρουν οι τεχνολογίες πολυπλέγματος από οποιαδήποτε άλλη τεχνολογία κατανεμημένων συστημάτων. Είναι γεγονός ότι την παρούσα χρονική συγκηρία στην εξέλιξη των τεχνολογιών λογισμικού υπάρχει μια πληθώρα κατανεμημένων συστημάτων με παρεπλήσιες δυνατότητες. Παρόλα αυτά, κάποια χαρακτηριστικά του πολυπλέγματος το κάνουν να υπερέχει σημαντικά από τις υπόλοιπες τεχνολογίες. Ένα από αυτά είναι το γεγονός ότι κάθε κόμβος στο πολυπλέγμα δεν ελέγχεται από κάποια κεντρική υπλογιστική οντότητα, αλλά είναι αυτόνομος και έχει τη δική του κυριότητα. Ένα άλλο χαρακτηριστικό είναι η χρήση ανοιχτών προτύπων για τις πληροφορίες που ανταλλάσονται μεταξύ των κόμβων. Εξάλλου, οι τεχνολογίες πολυπλέγματος επιτρέπουν να γίνεται χρήση των υπηρεσιών που προσφέρουν κάτω από διαφορετικά πρότυπα ποιότητας υπηρεσιών (QoS) εξαρτόμενα κάθε φορά από το είδος της εφαρμογής που εξυπηρετούν [15].

2.4.3 Διάφορα Είδη Τεχνολογιών Πλέγματος

Ο κύριος στόχος για τον οποίο αναπτύχθηκαν οι τεχνολογίες πλέγματος ήταν η σύνδεση των μεταξύ τους διαφόρων διασκορπισμένων γεωγραφικών super υπολογιστικών συστημάτων, αλλά πλέον έχουν αναπτυχθεί ακόμα περισσότερο. Σήμερα βρίσκουν χρήση σε πληθώρα εφαρμογών που σκοπό έχουν να επιλύσουν εντελώς διαφορετικά προβλήματα. Στις τεχνολογίες πολυπλέγματος οι χρήστες μπορούν να επικοινωνούν με μια υπηρεσία μεσάζοντα για να επιλύσουν κάποιο πρόβλημα που χρειάζεται την εκτέλεση κάποιας οντότητας λογισμικού, η υπηρεσία μεσάζοντας με της σειρά της αναζητεί την κατάλληλη υπηρεσία για να προωθήσει την αίτηση προς επίλυση, οργανώνει το χρονοδιάγραμμα της εκτέλεσης και εν

τέλη υποβάλει την αίτηση για να εκτελεστεί η απαιτούμενη εργασία, η οποία θα χρησιμοποιεί καταναμεμένους υπολογιστικούς πόρους, όπως φαίνεται και στο ακόλουθο σχήμα.



Εικόνα 2.7 – Αλληλεπιδράσεις Οντοτήτων Τεχνολογιών Πλέγματος

Από τη μεριά του χρήστη οι τεχνολογίες πολυπλέγματος μπορούν να κατηγοριοποιηθούν και να παρέχουν τις ακόλουθες υπηρεσίες:

- Υπολογιστικές Υπηρεσίες: Κύριο μέλημά τους είναι η παροχή ασφαλών υπηρεσιών για εκτέλεση εφαρμογών σε κατανεμημένους πόρους ατομικά ή συλλογικά. Οι μεσάζοντες παρέχουν τις υπηρεσίες για για συλλογική χρήση των κατανεμημένων πόρων.
- Υπηρεσίες Αποθήκευσης Δεδομένων: Αυτές έχουν σα στόχο να παρέχουν ασφαλή πρόσβαση σε κατανεμημένα πακέτα δεδομένων και ασφαλή συντήρησή τους. Για καλύτερη απόδοση, συχνά χρησιμοποιούνται αντίγραφα δεδομένων, ευρετηριοποιούνται, ενώ συχνά διαφορετικά πακέτα δεδομένων βρίσκονται διασκορπισμένα σε διαφορετικά γεωγραφικά μέρη, δίνοντας την ψευδαίσθηση παρόλα αυτά μαζικής αποθήκευσης. Συνήθως τα δεδομένα αυτά επεξεργάζονται από τις υπολογιστικές υπηρεσίες.
- Υπηρεσίες Εφαρμογών: Αυτές είναι επικεντρωμένες στη διαχείριση εφαρμογών για στο να παρέχουν πρόσβαση σε απομακρυσμένες οντότητες λογισμικού και βιβλιοθήκες με διαφανές τρόπο. Το αντικειμενοστρεφές πρότυπο που αναλύθηκε στην προηγούμενη υποενότητα παίζει ένα βασικότατο ρόλο στον ορισμό των

υπηρεσιών εφαρμογών. Επιπλέον οι συγκεκριμένες υπηρεσίες αναπτύσσονται πάνω στις υπολογιστικές και αποθήκευσυς δεδομένων υπηρεσίες που παρέχει το πολυπλέγμα.

- Υπηρεσίες Πληροφοριών: Αυτές οι υπηρεσίες εξάγουν και παρουσίαζουν πληροφορία, όπως αυτή επεξεργάζεται μέσω των δεδομένων που παρέχουν τα παραπάνω είδη υπηρεσιών. Οι χαμηλού επιπέδου λεπτομέριες που χειρίζονται αυτές οι υπηρεσίες είναι και ο τρόπος με τον οποίο η πληροφορία παρουσιάζεται, αποθηκεύεται, προσπελάσσεται, διαμοιράζεται και συντηρείται.
- Υπηρεσίες Γνώσης: Σχετίζονται με τον τρόπο κατά τον οποίο η γνώση αποκτάται, χρησιμοποιείται, ανακαλείται, δημοσιεύεται και συντηρείται για να βοηθήσει τους χρήστες να επιτύχουν τους στόχους τους. Με τον όρο γνώση εννοείται η πληροφορία η οποία εφαρμόζεται για να επιλύσει ένα πρόβλημα ή να παρθεί μια απόφαση [16].

2.4.4 Οι Δυνατότητες των Τεχνολογιών Πολυπλέγματος

Οι τεχνολογίες πολυπλέγματος μπορούν να καλύψουν ένα μεγάλο εύρος απαιτήσεων και να επιλύσουν πολλά προβλήματα. Τα κυριότερα από τα προβλήματα που προσφέρουν λύση είναι τα ακόλουθα:

Αποδοτικότερη Εκμετάλλευση Πόρων

Η βασικότερη χρήση των τεχνολογιών πολυπλέγματος είναι η εκτέλεση μιας εφαρμογής σε ένα απομακρυσμένο περιβάλλον. Το μηχάνημα στο οποίο είναι επιθυμητό να εκτελεστεί η εφαρμογή μπορεί να είναι απασχολημένο ή να μην αρκούν οι διαθέσιμοι υπολογιστικοί του πόροι (ισχύς ή μνήμη). Έτσι δημιουργείται η απαίτηση η εφαρμογή να εκτελεστεί σε κάποιο άλλο απομακρυσμένο μηχάνημα πάνω στο πολυπλέγμα. Για να είναι δυνατόν αυτό, πρέπει να παρέχεται η δυνατότητα στην εφαρμογή να εκτελεστεί σε κάποιο απομακρυσμένο μηχάνημα, το οποίο φυσικά να πληρεί τις απαιτήσεις της. Επειδή έχει παρατηρηθεί ότι στις περισσότερες εταιρίες και οργανισμούς ένας πολύ μεγάλος αριθμός υπολογιστικών πόρων παραμένει αδρανής και μάλιστα τα περισσότερα μηχανήματα απασχολούνται λιγότερο από το 5% του συνολικού χρόνου που λειτουργούν. Κατά αυτό τον τρόπο, με τις τεχνολογές πολυπλέγματος μπορεί μια εφαρμογή να εκτελεστεί κάνοντας χρήση των αδρανών αυτών υπολογιστικών πόρων.

Παράλληλη Υπολογιστική Επεξεργασία

Από τις πιο ελκυστικές δυνατότητες που παρέχουν οι τεχνολογίες πολυπλέγματος είναι η παράλληλη επεξεργασία. Προγράμματα που βρίσκουν εφαρμογή σε διάφορους τομείς όπως η βιοιατρική, οικονομικά μοντέλα αναλύσεων, animation, επεξεργασία video και εικόνας είναι πολύ απαιτητικά σε υπολογιστική ισχύ. Το κοινό τους χαρακτηριστικό όμως είναι ότι χρησιμοποιούν αλγορίθμους οι οποίοι μπορούν να διαιρεθούν σε ανεξάρτητα κομμάτια και να εκτελεστούν παράλληλα. Κατά αυτό τον τρόπο μια εφαρμογή υψηλών απαιτήσεων σε υπολογιστική ισχύ μπορεί να διαιρεθεί σε μικρότερες υποεφαρμογές κάθε μία από τις οποίες εκτελείται σε διαφορετικό μηχάνημα του πολυπλέγματος και κάνοντας χρήση κοινών κατανεμημένων δεδομένων. Παρόλα τα εμπόδια όπως η περιορισμένη χωριτηκότητα του δικτύου, η καθυστέρηση που παρατηρείται όταν απαιτείται από μια υποεφαρμογή να προσπελάσει δεδομένα που επεξεργάζεται μια άλλη την ίδια χρονική στιγμή και άλλα γνωστά από τις τεχνολογίες των κατανεμημένων συστημάτων, το Grid μπορεί να μειώσει κατά πολύ το χρόνο εκτέλεσης μιας εργασίας.

Συνεργασία μεταξύ εικονικών πόρων

Μία ακόμη συνεισφορά των τεχνολογιών πολυπλέγματος είναι η δυνατότητα για συνεργασία μεταξύ ενός ευρύτερου κοινού. Παλιότερα κάτι ανάλογο συνέβαινε και με τα κατανεμημένα υπολογιστικά συστήματα, το Grid όμως επεκτείνει κατά πολύ τις δυνατότητες συνεργασίας αφού μπορεί να τις προσφέρει σε μεγαλύτερο κοινό, ενώ τα πρότυπα που διαθέτει κάνουν ετερογενή υπολογιστικά συστήματα να συμπεριφέρονται σαν ένα μεγάλο εικονικό σύστημα που διαθέτει μεγάλη ποικιλία εικονικών πόρων.

Το μοίρασμα των εικονικών αυτών πόρων περιλαμβάνει αρχικά δεδομένα, είτε με τη μορφή αρχείων είτε βάσεων δεδομένων. Το τελευταίο αποτελεί ένα πολυπλέγμα δεδομένων (data grid) και τα πλεονεκτήματα που προσφέρει είναι πολλά. Τα δεδομένα μπορούν να διαμοιράζονται στα συστήματα που αποτελούν το πολυπλέγμα κάνοντας δυνατή τη μετάδοση με μεγαλύτερες ταχύτητες μέσω ειδικών τεχνικών. Επίσης είναι μεγαλύτερες οι δυνατότητες ασφάλειας αφού σε ορισμένα σημεία του Grid κρατούνται αντίγραφα ασφαλείας (backup) των κρισιμότερων δεδομένων.

Εκτός από τα δεδομένα τα συστήματα του grid μπορούν να μοιράζονται και άλλου είδους πόρους όπως: ειδικό εξοπλισμό, λογισμικό, υπηρεσίες (services), άδειες, εύρος ζώνης σύνδεσης στο internet κ.α.

Ισορροπία Πόρων

Όπως αναφέρθηκε πριν, σε ένα υπολογιστικό πολυπλέγμα οι πόροι που συνεισφέρουν μεμονωμένα μηχανήματα ενοποιούνται και αντιμετωπίζονται ως εικονικοί πόροι διαθέσιμοι σε όλους. Με κατάλληλη χρήση αυτών των πόρων σε εφαρμογές πολυπλέγματος είναι δυνατή η ισορροπία στους πόρους αναθέτοντας εργασίες-grid (grid jobs) στα μηχανήματα με χαμηλή χρήση. Κάτι τέτοιο μπορεί να συμβάλει στην αντιμετώπιση υπερφόρτωσης ορισμένων μηχανημάτων είτε κατευθύνοντας όπως προαναφέρθηκε τις εργασίες σε μηχανήματα με χαμηλή χρήση, είτε αν όλα τα μηχανήματα ήδη χρησιμοποιούνται στο μέγιστο να μπαίνουν σε προτεραιότητα οι υψηλότερης σημασίας εργασίες και να αναστέλλονται οι μικρής σημασίας.

Αξιοπιστία

Η προσέγγιση στον τομέα της αξιοπιστίας από την τεχνολογία του grid βασίζεται σε φθηνά μηχανήματα απομακρυσμένα συνήθως μεταξύ τους. Έτσι αν σε μια τοποθεσία συμβεί απώλεια τροφοδοσίας ή άλλου είδους πρόβλημα, τα υπόλοιπα τμήματα του grid δεν θα επηρεαστούν και οι εργασίες θα αναπροσαρμοστούν αυτόματα για να εκτελεστούν στα υπόλοιπα μηχανήματα. Άλλωστε, αντίγραφα των υψηλής προτεραιότητας εργασιών είναι δυνατόν εκ των προτέρων να εκτελούνται σε περισσότερα του ενός μηχανήματα έτσι ώστε η πιθανή απώλεια ενός μηχανήματος να μην αποτελεί πρόβλημα. Κατά αυτόν τον τρόπο επιλύεται το πρόβλημα του υψηλού κόστους απόκτησης και συντήρησης ενός μηχανήματος με μεγάλη αξιοπιστία και τεράστια συστήματα ψήξης και εξασφάλισης συνεχούς τροφοδοσίας ώστε να μην υπάρχει κίνδυνος αστοχίας, Επιπλέον δεν υπάρχει κεντρικό σημείο αποτυχίας (central point of failure) καθώς οι τεχνολογίες πολυπλέγματος βασίζονται εξ ορισμού σε κατανεμημένους και απολύτως αυτόνομους υπολογιστικούς κόμβους.

2.4.5 Τα Είδη των Πόρων στις Τεχνολογίες Πολυπλέγματος

Όπως έχει αναφερθεί στις τεχνολογίες πολυπλέγματος, αυτόνομες υπολογιστικές μονάδες που λειτουργούν ως κόμβοι αντιμετωπίζονται σαν μια ενιαία υπολογιστική οντότητα, συνεισφέροντας υπολογιστικούς πόρους. Οι κυριότεροι εξ αυτών είναι οι ακόλουθοι.

Υπολογιστική Ισχύς

Ο πιο συνηθισμένος πόρος στις τεχνολογίες πολυπλέγχματος είναι η υπολογιστική ισχύς που παρέχουν οι επεξεργαστές των μηχανημάτων του. Οι επεξεργαστές αυτοί μπορεί να ποικίλουν σε αρχιτεκτονική, ταχύτητα, λογισμικό, μνήμη, αποθηκευτικό χώρο και δυνατότητες διασύνδεσης. Υπάρχουν πολλοί τρόποι για την χρήση των υπολογιστικών πόρων στις τεχνολογίες πολυπλέγματος. Ο πιο απλός είναι η εκτέλεση μίας υπάρχουσας εφαρμογής σε ένα διαφορετικό μηχάνημα. Ένας δεύτερος είναι η χρήση σε μία εφαρμογή που είναι σχεδιασμένη να μπορεί να χωριστεί σε μέρη που να εκτελούνται παράλληλα σε δύο ή περισσότερα μηχανήματα. Τέλος, ένας τρίτος τρόπος είναι σε εφαρμογές που χρειάζεται να εκτελεστούν πολλές φορές, κάτι που μπορεί να γίνει ταυτόχρονα σε διαφορετικά μηχανήματα.

Αποθηκευτικός Χώρος

Ο δεύτερος πιο συνηθισμένος χρησιμοποιούμενος πόρος στις τεχνολογίες πολυπλέγματος είναι ο αποθηκευτικός χώρος δεδομένων. Ένα Grid που παρέχει κυρίως το μοίρασμα του πόρου αυτού ονομάζεται grid δεδομένων (data grid). Κάθε μηχάνημα παρέχει μία ποσότητα αποθηκευτικού χώρου για χρήση από το πλέγμα, είτε μόνιμα, είτε προσωρινά. Ο αποθηκευτικός χώρος αυτός μπορεί να είναι βασική μνήμη ή δευτερεύων αποθηκευτικό μέσο όπως σκληρός δίσκος. Ο αποθηκευτικός χώρος μπορεί να χρησιμοποιηθεί με πολλούς τρόπους για να αυξήσει τη χωρητικότητα, τις επιδόσεις, το μοίρασμα και την αξιοπιστία των δεδομένων. Η χωρητικότητα μπορεί να αυξηθεί χρησιμοποιώντας τον χώρο αποθήκευσης σε πολλά. Κάθε αρχείο ή βάση δεδομένων μπορεί να χωρίζεται σε διάφορα αποθηκευτικά μέσα και μηχανήματα εξαλείφοντας το πρόβλημα των περιορισμών μέγιστου μεγέθους που υπαγορεύουν τα συνηθισμένα συστήματα αρχείων. Οι χρήστες των τεχνολογιών πολυπλέγματος βλέπουν τα δεδομένα στον ίδιο εικονικό χώρο παρά το ότι αυτά μπορεί να βρίσκονται διασκορπισμένα σε διάφορα σημεία του Grid. Στις βάσεις δεδομένων τα πλεονεκτήματα είναι ακόμα περισσότερα αφού πλέον αντί για πολλές μικρές βάσεις δεδομένων μπορούμε να έχουμε μία μεγαλύτερη και έτσι η πρόσβαση και η αναζήτηση σε αυτήν να είναι ευκολότερη. Επιπλέον, τα περισσότερο αναπτυγμένα συστήματα αρχείων μπορούν αυτόματα να κρατούν αντίγραφα ορισμένων δεδομένων ώστε να αυξάνεται η αξιοπιστία και οι επιδόσεις του συστήματος. Ένας προγραμματιστής μπορεί να καθορίσει σε συγκεκριμένες συσκευές αποθήκευσης να αποθηκεύσουν δεδομένα βάσει προτύπων χρήσης (usage patterns). Με βάση αυτά οι εργασίες θα εκτελούνται στα μηχανήματα που περιέχουν τα δεδομένα ή σε μηχανήματα που είναι άμεσα συνδεδεμένα με αυτά. Τέλος, οι επιδόσεις ενός συστήματος που υλοποιεί ένα τέτοιο σύστημα αρχείων μπορούν επίσης να αυξηθούν με

χρήση του χωρίσματος δεδομένων (data striping). Με την τεχνική αυτή με κατάλληλο χώρισμα των δεδομένων σε διάφορα αποθηκευτικά μέσα και μηχανήματα επιτυγχάνεται μεταφορά δεδομένων με μεγαλύτερους ρυθμούς από αυτούς που μπορεί να πετύχει οποιοδήποτε μεμονωμένο μέσο. Σε περιπτώσεις απαιτητικών εφαρμογών που είναι αναγκαίος ο υψηλός αυτός ρυθμός μετάδοσης δεδομένων, η δυνατότητα αυτή είναι πολύ σημαντική.

Επικοινωνίες

Η ραγδαία ανάπτυξη στις ταχύτητες των επικοινωνιών μεταξύ υπολογιστών κάνει σήμερα την υλοποίηση των τεχνολογιών πολυπλέγματος πολύ ευκολότερη σε σχέση με την εποχή που εμφανίστηκαν τα κατανεμημένα συστήματα. Είναι προφανές λοιπόν ότι ένας ακόμα σημαντικός πόρος είναι η χωρητικότητα επικοινωνιών (communication capacity). Σε αυτό συμπεριλαμβάνονται τόσο οι επικοινωνίες μέσα στο πλέγμα όσο και έξω από αυτό. Οι επικοινωνίες είναι πολύ σημαντικές για την αποστολή εργασιών και δεδομένων που συνδέονται με αυτές στα διάφορα σημεία του Grid. Κάποιες εργασίες απαιτούν πολύ μεγάλες ποσότητες δεδομένων για επεξεργασία, το μεγαλύτερο μέρος των οποίων δεν βρίσκεται στο μηχάνημα που τις εκτελεί. Επομένως, το διαθέσιμο εύρος ζώνης για τις επικοινωνίες αυτές είναι συχνά κρίσιμος πόρος που μπορεί να περιορίσει την επίδοση του grid.

Άδειες Χρήσης Λογισμικού

Στις τεχνολογίες πολυπλέγματος μπορεί να χρησιμοποιείται λογισμικό το οποίο είτε είναι πολύ ακριβό για να εγκατασταθεί σε όλα τα μηχανήματά του είτε η άδεια χρήσης του περιορίζει τον αριθμό των μηχανημάτων που μπορούν να το χρησιμοποιούν ταυτόχρονα. Χρησιμοποιώντας το Grid, οι εργασίες που απαιτούν τη χρήση του συγκεκριμένου software στέλνονται σε ένα μηχάνημα στο οποίο είναι εγκατεστημένο ενώ γίνεται έλεγχος ώστε να μην παραβιαστούν οι όροι των αδειών χρήσης. Επομένως το ίδιο το λογισμικό αποτελεί έναν ακόμα πόρο μέσα σε περιβάλλον πολυπλέγματος

2.4.6 Προυποθέσεις για Επαγγαλμετικές Τεχνολογίες Πολυπλέγματος (enterprise GRID)

Είναι προφανές ότι για να καθιστεί δυνατή η χρήση των τεχνολογιών πολυπλέγματος σε επίπεδο εταιρίας, είτε για να το χρησιμοποιήσει για δικούς της σκοπούς είτε σαν

ανεξάρτητος πάροχος διαδικτυακών υπολογιστικών υπηρεσιών, είναι απαραίτητο να πληρούνται μια σειρά από προυποθέσεις.

Προτυποποίηση

Το πιο βασικό χαρακτηριστικό που πρέπει να υλοποιούνε οι τεχνολογίες πολυπλέγματος είναι η αυστηρή προτυποποίηση. Σε ένα παγκόσμιο και αλληλεπιδρόμενο περιβάλλον, είναι απαραίτητο να υπάρχουν κάποια σταθερά πρότυπα, ώστε ο κάθε πελάτης να μπορεί να ξέρει πώς να κάνει χρήση μια υπηρεσίας. Ο οργανισμός που έχει αναλάβει την εποπτεία και την συνεργασία όλων την ερευνητικών εγχειρημάτων σχετικά με το Grid είναι το GGF (Global Grid Forum). Το εκάστοτε πολύπλεγμα και ειδικά αυτά που προορίζονται για επαγγελματικές εφαρμογές πρέπει να υιοθετούν τα υπάρχοντα πρότυπα και να συνεισφέρουν στη δημιουργία νέων. Αυτός είναι ο μόνος τρόπος να εξασφαλιστεί συμβατότητα ανάμεσα στους οργανισμούς και μεταξύ των εφαρμογών και υλοποιήσεων Grid.

Επεκτασιμότητα

Η εξέλιξη των τεχνολογιών πολυπλέγματος έχει προσανατολιστεί στην ενσωμάτωση των νέων τεχνολογιών για μέγιστη χρησιμοποίηση όλων των δυνατών υπολογιστικών πόρων. Η διαδικασία αυτή πρέπει να δημιουργήσει ένα νέο περιβάλλον που θα έχει παράλληλα δυνατότητες διαχείρισης από τους υπευθύνους της εταιρίας αλλά και δυνατότητες χρέωσης των πελατών. Στις μέρες μας, οι συναλλαγές μεταξύ των εταιριών γίνονται αποκλειστικά με ηλεκτρονικά μέσα, κάτι που φυσικά πρέπει να ενσωματώνει και το περιβάλλον του επαγγελματικού Grid.

Ασφάλεια

Ένα πολύ σημαντικό ζήτημα αποτελεί και η ασφάλεια στη χρήση των τεχνολογιών πολυπλέγματος. Όπως είναι γνωστό, οποιοδήποτε σύστημα γίνεται χρήση του διαδικτύου είναι ευάλλωτο σε επιθέσεις. Ο τομέας της ασφάλειας αναπτύχθηκε για να μπορούν να προβλέπονται και να αντιμετωπίζονται οι οποιεσδήποτε τέτοιου είδους επιθέσεις. Στο Grid, η ασφάλεια παίζει ακόμα πιο σημαντικό ρόλο, καθώς διακινούνται ευαίσθητα ή σημαντικά δεδομένα, όπως οι πληροφορίες ιατρικού ιστορικού ενός ασθενούς αν πρόκειται για ιατρική εφαρμογή, πληροφορίες που θεωρούνται περιουσία μιας ανταγωνιστικής εταιρίας, ή απόρρητα κρατικά δεδομένα. Έτσι απαραίτητη είναι η ταυτοποίηση των χρηστών αλλά και η σαφής εξουσιοδότησή τους για προσπέλαση δεδομένων. Σύμφωνα με την ταυτοποίηση επιτρέπεται μόνο σε πιστοποιημένους χρήστες να έχουν πρόσβαση σε συγκεκριμένες υπηρεσίες και υπολογιστικούς πόρους. Επιπρόσθετα όμως, καθώς βρισκόμαστε σε ένα

κατανεμημένο περιβάλλον όπου οι πόροι, οι οποίοι ανήκουν σε διαφορετικούς εικονικούς οργανισμούς, συνδιαλλέγονται και αλληλεπιδρούν μεταξύ τους, απαραίτητο είναι κάθε φορά ένας χρήστης ή μια υπηρεσία –υπολογιστικός πόρος- να μπορεί να προσπελάει τα δεδομένα τα οποία είναι εξουσιοδοτημένη και μόνο αυτά. Σε κάθε αντίθετη περίπτωση, θα αμφισβητηθεί η τεχνολογία και οι υπηρεσίες του πολυπλέγματος που παρέχονται και τελικά θα απορριφθούν από τον επαγγελματικό χώρο.

Χρηστικότητα

Τέλος, έχει πολύ μεγάλη σημασία και ο τομέας της χρηστικότητας. Από τη μέχρι τώρα εμπειρία σε τεχνολογίες Grid, έχουμε διαπιστώσει πως οι εφαρμογές που υπάρχουν μέχρι σήμερα, είναι αρκετά δύσχρηστες και συνήθως παρουσιάζουν πολλά προβλήματα διότι βρίσκονται ακόμη σε φάση εξέλιξης. Για να μπορεί να λειτουργήσει ένα εταιρικό πολύπλεγμα, θα πρέπει να είναι σταθερό και χωρίς προβλήματα. Αυτά προϋποθέτουν την απασχόληση ενός ειδικού σε θέματα Grid. Ακόμα, θα πρέπει να περιλαμβάνει ένα εύχρηστο γραφικό περιβάλλον. Ο εκάστοτε πελάτης, δεν έχει καμία υποχρέωση γνώσεων περί πολυπλέγματος. Γνωρίζει μόνο τις ανάγκες και τις απαιτήσεις που έχει η δουλειά του και μέσα από αυτό το περιβάλλον πρέπει να μπορεί να καταχωρεί τις δουλείες του με ευκολία ανάλογη των αγορών μέσω Internet.

2.5 Το Μεσισμικό (middleware) GRIA

Τα μεσισμικά (middlewares) για τις τεχνολογίες πολυπλέγματος είναι στοίβες λογισμικού που σχεδιάστηκαν για να παρουσίασουν με έναν ομοιόμορφο τρόπο υπολογιστικούς πόρους και πόρους δεδομένων που δεν έχουν ομοιογενή μορφή ώστε να μπορούν να είναι προσβάσιμοι από το λογισμικό του χρήστη-πελάτη, χωρίς να είναι εκ των προτέρων γνωστές οι ρυθμίσεις του συστήματος. Στη συγκεκριμένη ενότητα θα γίνει μια παρουσίαση του μεσισμικού GRIA [17] το οποίο χρησιμοποιήθηκε για την εκπόμηση της παρούσας διπλωματικής εργασίας.

2.5.1 Τι είναι το GRIA

Το GRIA (Grid Resources for Industrial Applications) είναι μία υπηρεσιοστρεφής πλατφόρμα που σχεδιάστηκε για να υποστηρίζει το B2B (Business to Business) μοντέλο συνεργασιών, παρέχοντας υπηρεσίες με ασφαλή διαλειτουργικό και ευέλικτο τρόπο. Πρόκειται για ένα μεσισμικό ανοιχτού κώδικα το οποίο χρησιμοποιώντας επιχειρηματικά μοντέλα επιτρέπει στους παροχείς υπηρεσιών και του καταναλωτές να ανακαλύπτουν ο ένας τον άλλον και να διαπραγματεύονται τους όρους για πρόσβαση σε υπολογιστικούς πόρους. 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.

2.5.2 Επιχειρηματική Απεικόνιση του GRIA (Business Perspective)

Οι υπηρεσίες του GRIA υποστηρίζουν λειτουργίες οι οποίες είναι εφικτό να απαιτούν τη χρήση πόρων από διαφορετικούς παροχείς πάνω στο πολυπλέγμα, ενώ ταυτόχρονα να τους διαμοιράζονται με χρήστες που ανήκουν σε διαφορετικούς οργανισμούς και επιχειρήσεις. Τέτοιες λειτουργίες είναι απαραίτητες και χρησιμοποιούνται συχνά σε διάφορους τομείς, όπως στη λογιστική, στη μηχανική, στη βιιατρική, στις υπηρεσίες ιατρικής περίθαλψης και φυσικά στην ακαδημαική κοινότητα. Συχνά αφορούν πόρους όλων των ειδών: δεδομένα, πόρους για αποθήκευση, εφαρμογές και υπολογιστική ισχύ.

Στα πολυπλέγματα που χρησιμοποιούνται για ακαδημαικούς σκοπούς, το μοίρασμα των πόρων γίνεται με χρήση των υπηρεσιών παρόχων, που συμφωνούν να μοιράσουν τους πόρους τους για να δημιουργήσουνε μια κοινή υπολογιστική οντότητα η οποία και ικανοποιεί τις

απαιτήσεις μιας κοινότητας χρηστών. Για να γίνει αυτό εφικτό απαιτείται από τον πάροχο των υπηρεσιών και την κοινότητα να εγγραφούν σε έναν εικονικό οργανισμό (virtual organisation) και να συμφωνήσουν να υπακούουν σε κοινές πολιτικές, ώστε κατά αυτόν τον τρόπο να διαχειρίζονται και να εξασφαλίζουν πρόσβαση σε διαμοιρασμένους πόρους. Αυτός όμως ο διακανονισμός είναι πολύ ακριβός να εγκατασταθεί και να λειτουργήσει, ενώ παράλληλα δεν ικανοποιεί τις ανάγκες μια επιχείρησης καθώς δεν εξασφαλίζει στους συμμετέχοντες να διαπραγματεύονται τις τιμές ή την ποιότητα των υπηρεσιών που προσφέρουν, στα πλαίσια του επιχειρησιακού ανταγωνισμού.

Το GRIA έχει σχεδιαστεί για να επιλύει ακριβώς αυτό το πρόβλημα, παρέχοντας υπηρεσίες που πληρούν τις απαιτήσεις μια επιχείρησης.

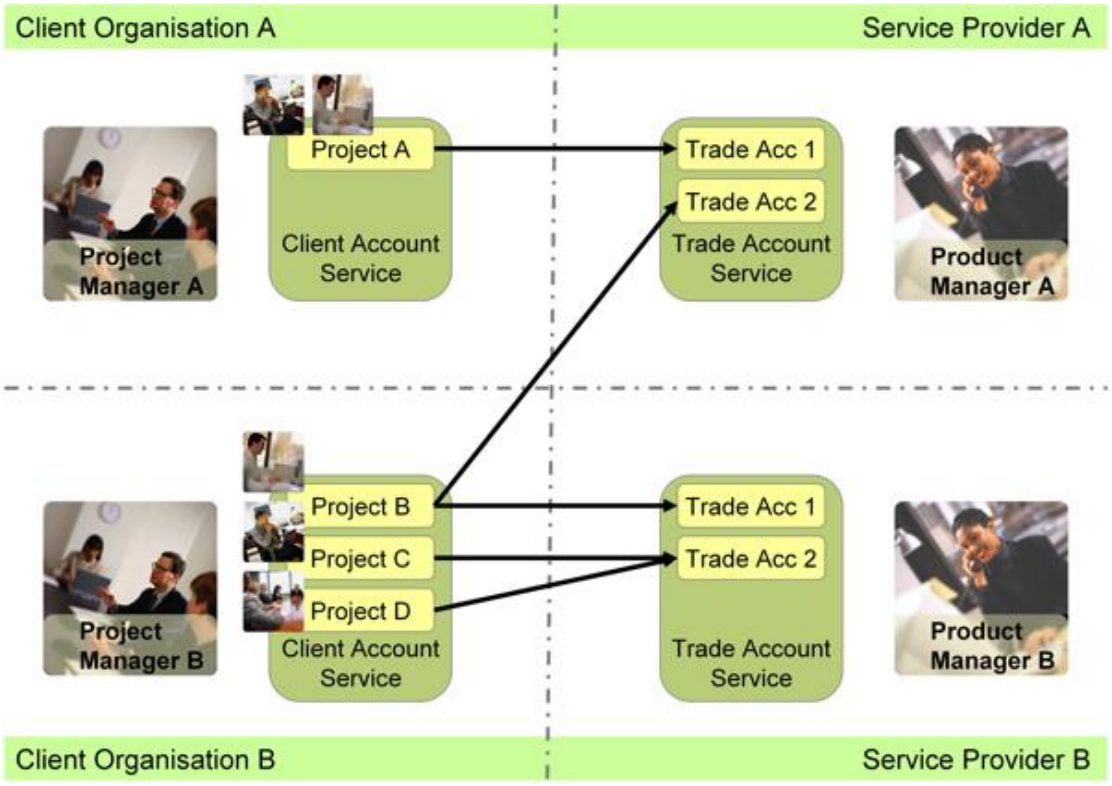
- Οι πάροχοι των υπηρεσιών είναι ικανοί να λειτουργούν ανεξάρτητα ο ένας από τον άλλον, και να ανταγωνίζονται ώστε να παρέχουν πληρωμένες υπηρεσίες στους καταναλωτές.
- Οι καταναλωτές έχουν τη δυνατότητα να ελέγχουν τις υπηρεσίες που καταναλώνουν, πόσο πολύ έχουν χρεωθεί και από ποιον.
- Οι υπηρεσίες ελέγχονται μέσω συμφωνιών επιπέδου υπηρεσιών (service level agreements) έτσι ώστε οι πάροχοι αφενός να γνωρίζουν τι είδους υπηρεσίες πρέπει να πάρεχουν και αφετέρου οι καταναλωτές το τι είδους τους προσφέρονται και με πιο αντίτιμο.
- Η ασφάλεια ικανοποιεί τις εμπορικές ανάγκες και πρότυπα (standards).
- Οι πάροχοι προσφέρουν υπηρεσίες σεβόμενοι τις άδειες λογισμικού των εφαρμογών που χρησιμοποιούν.
- Τα μηνύματα που ανταλλάσονται ικανοποιούν προκαθορισμένα πρότυπα.
- Το επιχειρισιακό κόστος ελαχιστοποιείται.

Για όλα τα παραπάνω, το μεσισμικό GRIA χρησιμοποιεί μια αποκεντρωμένη διαχειριστική προσέγγιση με ελάχιστη αλληλοεξάρτηση μεταξύ των ιστιοχώρων. Κάθε ιστιοχώρος που προσφέρει υπηρεσίες GRIA είναι ανεξάρτητος και ελεύθερος να επιλέξει ποιους χρήστες θα εμπιστευτεί και υπό ποιες προυποθέσεις, ενώ υιοθετεί τις δικές του πολιτικές και αποφασίζει αυτόνομα για το ποιες εφαρμογές θα υποστηρίξει. Οι ιστιοχώροι μπορεί να επικοινωνούν μεταξύ τους, αλλά αυτό συμβαίνει μόνο εξαιτίας των κοινών χρηστών τους, οι οποίοι είναι υπεύθυνοι να διαχειριστούν τις εξαρτήσεις που προκύπτουν. Δεν υπάρχουν κοινές συμφωνίες

και εικονικοί οργανισμού, αν και οι χρήστες μπορούν να επικοινωνούν με χρήση εικονικών οργανισμών εαν το επιθυμούν.

Αξίζει να αναλυθεί περισσότερο ο τρόπος με τον οποίο υποστηρίζει το GRIA όλα τα παραπάνω. Όταν ο καταναλωτής επιθυμεί να αγοράσει μια υπηρεσία (ή καλύτερα τη χρήση μιας υπηρεσίας) από κάποιον πάροχο, πρώτα πρέπει αν ανοίξουν έναν λογαριασμό συνδιαλλαγής. Αυτός ο λογαριασμός αντιπροσωπεύει μια σχέση εμπιστοσύνης μεταξύ τους, βασισμένη στην πρόθεση του καταναλωτή να πληρώσει για την προσφερόμενη υπηρεσία. Οι δύο πλευρές μπορούν να ορίσουν πιστωτικό όριο για τη χρήση της υπηρεσίας που αντιπροσωπεύει τη μέγιστη δυνατή ποσότητα υπηρεσίας που ο παροχέας είναι διατεθιμένος να προσφέρει προτού πληρωθεί ή την μέγιστη δυνατή ποσότητα που είναι ικανός να πληρώσει.

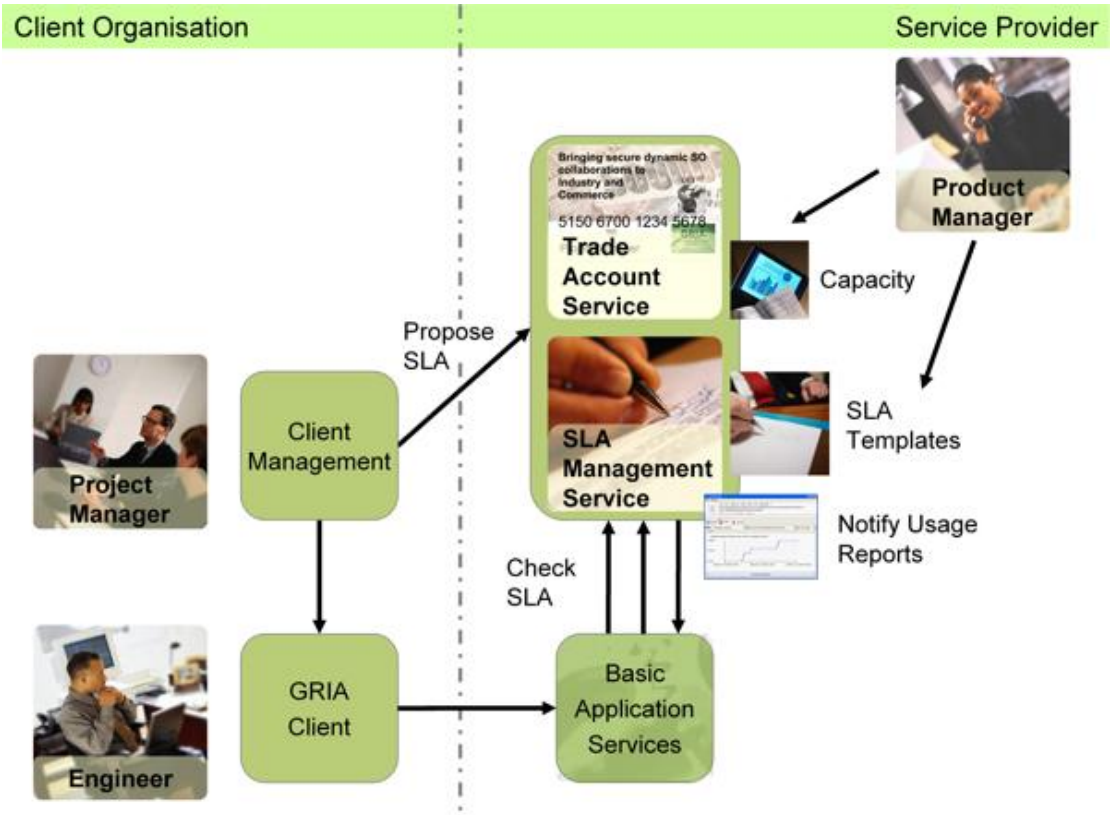
Ένας τυπικός οργανισμός καταναλωτή μπορεί να έχει αρκετές εξαρτήσεις μεταξύ διαφορετικών παροχέων, ο καθένας από τους οποίους χρησιμοποιείται από διαφορετικό εξουσιοδοτημένο χρήστη, όπως φαίνεται και στο ακόλουθο σχήμα.



Εικόνα 2.8 – Διαχείριση Επιχειρισιακών Σχέσεων στο GRIA

Απαραίτητο είναι να τονιστεί ότι ο λογαριασμός συνδιαλλαγής δεν είναι κάποιος τραπεζικός λογαριασμός. Καταγράφει απλά τα ποσά που χρωστάει ο καταναλωτής για τη χρήση μιας υπηρεσίας. Ο πάροχος δε μπορεί να πάρει λεφτά από τον λογαριασμό αυτόν ή να χειριστεί το ποσό ώστε να ξεπληρώσει άλλους, αλλά μπορεί να εκδόσει ένα τιμολόγιο και να το στείλει στους καταναλωτές. Ο τελευταίος μπορεί να ζητήσει τις λεπτομέρειες της χρήσης του ώστε να ελέγξει για τυχόν σφάλματα, πράγμα που διασφαλίζεται από το GRIA καθώς καταγράφονται τα μηνύματα που επεξεργάζονται οι υπηρεσίες της. Όταν ο καταναλωτής πληρώσει το λογαριασμό του, αυτό αναφέρεται ώστε να μην χρεωθεί ξανά για ίδια χρήση.

Το GRIA επιτρέπει στους παρόχους και καταναλωτές να ανταλλαξουν πόρους (αποθηκευτικό χώρο, δεδομένα, υπολογιστική ισχύ και εφαρμογές) μέσω της χρήσης των πιστωτικών συμφωνιών επιπέδου υπηρεσιών (service level agreements – SLA). Το SLA περιγράφει την ποιότητα της υπηρεσίας (QoS) και άλλες δεσμεύσεις του παρόχου καθώς και τις οικονομικές επιβαρύνσεις του πελάτη για τη χρήση της υπηρεσίας. Το GRIA επιτρέπει στους παρόχους να διαφημίζει SLAs που προτάθηκαν από τους καταναλωτές κατά τη διάρκεια της διαπραγματευσης.



Εικόνα 2.9 – Διαχείριση με Χρήση Συμφωνιών Επιπέδου Υπηρεσιών (SLA)

Οι υπηρεσίες του GRIA δημιουργούνε αναφορές σχετικά με τη χρήση τους, οι οποίες είναι είτε ποσοτικές (δεδομένα που μεταφέρθηκαν, χρόνος επεξεργασίας) είτε ποιοτικές (κατάσταση υπηρεσίας λόγω σφαλμάτων). Αυτές τις αναφορές χρησιμοποιεί το GRIA για να καταγράψει τη χρήση της υπηρεσίας που έχει κάνει ο χρήστης. Οι απαραίτητες αποφάσεις που λαμβάνει το μεσισμικό με βάση τις συγκεκριμένες αναφορές είναι:

- Αν πρέπει να δεχτεί ένα συγκεκριμένο SLA όταν προτείνεται από έναν καταναλωτή.
- Αν μια υπηρεσία καλύπτεται από υπάρχον SLA για δεδομένα QoS και αν αυτή μπορεί να εκτελεστεί με βάση τη χωρητικότητα του παροχέα και τις προοδιαγραφές που ορίζει τοι συμφωνημένο SLA.
- Αν το SLA πρέπει να ακυρωθεί όταν η χωρητικότητα του παροχέα εξαντλείται και πώς να ελλατώσει το φόρτο εργασίας για να λύσει το πρόβλημα αυτό.
- Τι είδους QoS παρέχεται στον καταναλωτή και πόσο πρέπει να χρεώσει για αυτό.

Σε ότι αφορά τις απαιτήσεις για ασφάλεια, το GRIA έχει σχεδιαστεί για να διατηρεί την ανεξαρτησία μεταξύ των διαφορετικών εμπλεκομένων ιστιοχώρων με το λειτουργικό κόστος παράλληλα στο ελάχιστο. Υιοθετούνται οι μηχανισμοί ασφαλείας που ικανοποιούν τα πρότυπα της υπηρεσιοστρεφούς αρχιτεκτονικής για να διασφαλίσει τους χρήστες και τους ρόλους τους καθώς προσπελάσσουν τις υπηρεσίες. Αξιοσημείωτο εξάλλου είναι ότι το GRIA δε δίνει ευθύ πρόσβαση ούτε στους εξουσιοδοτημένους χρήστες να προσπελάνε τις υπηρεσίες και να εκτελούνε εντολές.. Αντίθετα πρέπει να κάνουν αίτηση ώστε η υπηρεσία να εκτελέσει εντολές εκ μέρους τους. Αυτό σημαίνει ότι οι πάροχοι δε χρειάζεται να εγκαταστήσουν λογαριασμούς για κάθε χρήστη, που ισοδυναμεί με μεγάλο λειτουργικό κόστος για πολλά ακαδημαικά πολυπλέγματα. Επίσης μεινώνει το επίπεδο εμπιστοσύνης που πρέπει να έχουν εξασφαλίσουν οι πάροχοι στους χρήστες τους, δίνοντας τους τη δυνατότητα να παραμένουν ανεξάρτητοι μεταξύ τους ακόμα και όταν επικοινωνούν με άλλους πάροχους εκ μέρους των πρώτων. Επίσης εξασφαλίζει ότι οι εφαρμογές που θα εκτελεστούν μέσω των παρεχόμενων υπηρεσιών θα είναι αδειοθετημένοι.

Ένα ακόμα πλεονέκτημα του GRIA είναι ότι δυνατότητα δυναμικής διαχείρισης των συμφωνιών επιπέδου υπηρεσιών, επιτρέποντας να οριστούν και να συμφωνηθούν νέες για κάθε προσπέλαση υπηρεσιών, συναρτήσεων, δεδομένων ή απαίτηση για υπολογιστική ισχύ. Η δυναμική διαχείριση των SLAs επιτρέπει στο χρήστη να ελέγχει τις διασυνδέσεις με τις υπηρεσίες που μπορεί να ανήκουν σε διαφορετικούς παρόχους και να διαμοιράζονται τα αποτελέσματα με άλλους χρήστες όταν αυτό απαιτείται. Επίσης υποστηρίζεται η ολοκλήρωση με άλλους πιστοποιημένους από το χρήστη ιστιοχώρους και συγκεκριμένες

πολιτικές καθορίζουν ποιοι ιστιοχώροι μπορούν να θεωρηθούν έμπιστοι και να προσπελλάσονται . Αυτό καθιστά πολύ εύκολο να εγκαταστηθούν επιχειρησιακές συνδέσεις και να παρέχονται υπηρεσές σε νέους καταναλωτές άμεσα, χωρίς την ύπαρξη χρονοβόρων διαδικασιών. Τέλος, κρατώντας τους παρόχους ανεξάρτητους μεταξύ τους, μεταβιβάζεται η ευθύνη στον καθένα να εξασφαλίσει την ασφάλεια που απαιτείται. Έτσι, αντίθετα με πολλά ακαδημαικά GRIDs, δεν υπάρχει κεντρικό σημείο αποτυχίας (center point of failure) και άμα κάποιος ιστιοχώρος του GRIA υποστεί προβλήματα ασφάλειας, αυτό λύνεται τοπικά, καθώς οι υπόλοιποι ιστιόχωροι δε χρειάζεται να απενεργοποιηθούνε. Έτσι, η επιτυγχάνεται υψηλού βαθμού ανεκτικότητα, πράγμα απαραίτητο σε επιχειρηματικές εφαρμογές [18].

Τέλος, οι επιχειρηματικές εφαρμογές πρέπει να υποστηρίζουν την εύκολη διασυνδεσιμότητα μεταξύ των παρόχων και των καταναλωτών. Αυτό απαιτεί σαφώς καθορισμένα πρότυπα πάνω στα οποία θα στηριχθεί η ανάπτυξη των υπηρεσιών και των εφαρμογών-πελατών που θα τις χρησιμοποιούν. Το GRIA, βασισμένο στην υπηρεσιοστρεφή αρχιτεκτονική, κάνει βασικών εξ αυτών των προτύπων, τα σημαντικότερα από τα οποία είναι WS-I Basic Profile [19], WS-Security [20], WS-Federation [21], WS-Addressing [22], Web Service Resource Framework (WSRF) [23] και WS-Notification [24].

3

Ανάλυση Απαιτήσεων Συστήματος

Στο συγκεκριμένο κεφάλαιο θα γίνει αναλυτική περιγραφή των απαιτήσεων του συστήματος που έχει υλοποιηθεί στην παρούσα διπλωματική εργασία για την παροχή αδειών χρήσης λογισμικού. Θα περιγραφεί η αρχιτεκτονική του συστήματος σε αντιδιαστολή με την υπάρχουσα αρχιτεκτονική που υιοθετείται από τα σύγχρονα μοντέλα παροχής αδειών σήμερα, θα αναλυθούνε οι επιπλέον λειτουργίες που προστίθονται, ενώ όλο το κείμενο θα τεκμηριωθεί και από αντίστοιχα διαγράμματα οντοτήτων και ροής δεδομένων για την καλύτερη κατανόηση από τον αναγνώστη.

3.1 Αρχιτεκτονική

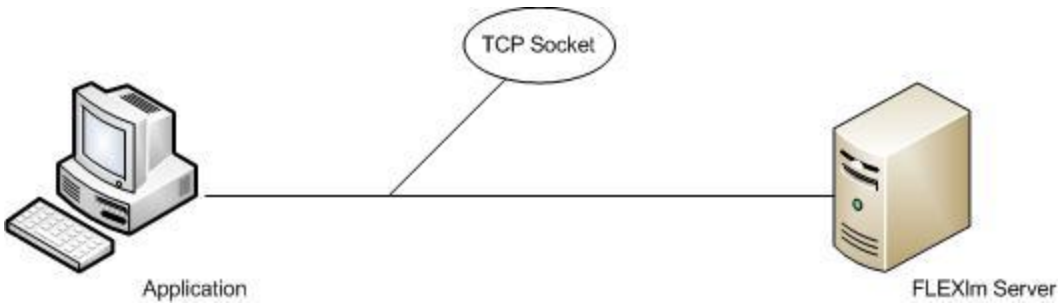
Τα μοντέλα παροχής αδειών λογισμικού με σύγχρονο τρόπο και χρήση ενός κεντρικού εξυπηρετητή, όπως παρουσιάστηκαν στο κεφάλαιο 2, εδραιώνουν την επικοινωνία μεταξύ εφαρμογής πελάτη και εξυπηρετητή στο χαμηλό επίπεδο (επίπεδο μεταφοράς TCP), με αποτέλεσμα να θέτονται περιορισμοί ως προς τη χρήση τους καθώς η τελευταία είναι εφικτή μόνο σε περιβάλλον ενός τοπικού δικτύου μιας επιχείρησης ή οργανισμού. Κατά αυτόν τον τρόπο δεν είναι δυνατός ο έλεγχος της κίνησης στο δίκτυο μεταξύ πελάτη και εξυπηρετητή, ενώ ταυτόχρονα ο τελευταίος είναι αναγκεσμένος να έχε εκτεθημένες όλες τις θύρες του για να μπορέσει να ανοίξει δυναμικά συνδέσεις με τους αιτούμενους κόμβους.

Θέλοντας να εξαλείψουμε τα παραπάνω μειονεκτήματα και να προσδώσουμε στο μοντέλο επιπλέον χαρακτηριστικά που θα το κάνουν πιο ελκυστικό για επιχειρησιακές εφαρμογές, αναπτύξαμε την παρακάτω αρχιτετκτονική που θα περιγραφεί στη συνέχεια, κάνοντας χρήση τεχνολογιών διαδικτύου και μεσισμικού πολυπλέγματος.

Η αξιοπιστία του νέου συστήματος παροχής αδειών που περιγράφουμε, εξασφαλίζεται μεταφέροντας πλέον την επικοινωνία από το επίπεδο μεταφοράς στο επίπεδο εφαρμογών. Αυτό δίνει τη δυνατότητα για ανάπτυξη νέων οντοτήτων λογισμικού που θα βελτιώσουν τον έλεγχο στην προσπέλαση των εφαρμογών, ενώ παράλληλα γίνεται εφικτή η υλοποίηση νέων λειτουργιών που θα προσθέσουν νέες και ελκυστικότερες λύσεις για την εκτέλεση εφαρμογών από πολλαπλούς χρήστες. Το γεγονός ότι η επικοινωνία γίνεται στο επίπεδο εφαρμογών επιτρέπει τη χρήση πιστοποιητικών για την αναγνώριση των χρηστών και τον έλεγχο για το ποιος είναι εξουσιοδοτημένος να προσπελάσει τι, πράγμα που επιτρέπει τη διασυνδεσιμότητα μεταξύ ετερογενών δικτύων και υπολογιστικών συστημάτων αυξάνοντας παράλληλα και το επίπεδο ασφάλειας του συνολικού συστήματος.

3.1.1 Η υπάρχουσα αρχιτεκτονική του σύγχρονου μοντέλου παροχής αδειών

Η υπάρχουσα αρχιτεκτονική του σύγχρονου μοντέλου παροχής αδειών λογισμικού με χρήση ενός κεντρικού εξυπηρετητή, που για τις ανάγκες της διπλωματικής εργασίας θα είναι ο FLEXlm server της Macrovision Corporation, έχει εκτενώς αναλυθεί στο προηγούμενο κεφάλαιο, γι' αυτό και στην παρούσα ενότητα θα γίνει μια απλή αναφορά. Η αρχιτεκτονική αποτελείται από δύο κεντρικές οντότητες: την εφαρμογή-πελάτη και τον εξυπηρετητή.

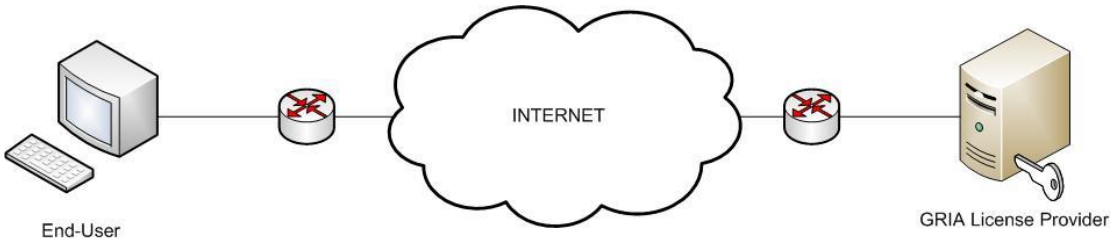


Εικόνα 3.1 – Διάγραμμα Συνιστωσών Υπάρχουσας Αρχιτεκτονικής

Οι δύο αυτές οντότητες βρίσκονται πάνω στο ίδιο τοπικό δίκτυο και επικοινωνούν μεταξύ τους μέσω TCP sockets πάνω στο επίπεδο μεταφοράς. Οποτεδήποτε υπάρχει η ανάγκη από τον πελάτη να εκτελέσει μια συγκεκριμένη εφαρμογή, ανοίγει ένα socket και ανταλλάσει μηνύματα με τον FLEXlm εξυπηρετητή, ώστε ο τελευταίος πάλι μέσω socket στο TCP διάστρωμα να του παραδόσει την άδεια για εκτέλεση της εφαρμογής.

3.1.2 Η προτεινόμενη αρχιτεκτονική

Η νέα αρχιτεκτονική που προτείνεται στην παρούσα διπλωματική εργασία αποτελείται από δύο κύριες οντότητες, τον End-User και τον License Provider. Αυτές παίζουν το ρόλο της εφαρμογής-πελάτη και του εξυπηρετητή αντίστοιχα, σύμφωνα με το υπάρχον μοντέλο. Η κύρια διαφορά τους είναι ότι η επικοινωνία δε γίνεται πλέον με TCP μηνύματα στο διάστρωμα μεταφοράς, αλλά στο επίπεδο εφαρμογών με χρήση του διαδικτύου και του HTTP πρωτοκόλλου, σύμφωνα με όσα ορίζει το υπηρεσιοστρεφές μοντέλο (OSA) και τα πρότυπα του GRIA.



Εικόνα 3.2 – Διάγραμμα Συνιστωσών Προτεινόμενης Αρχιτεκτονικής

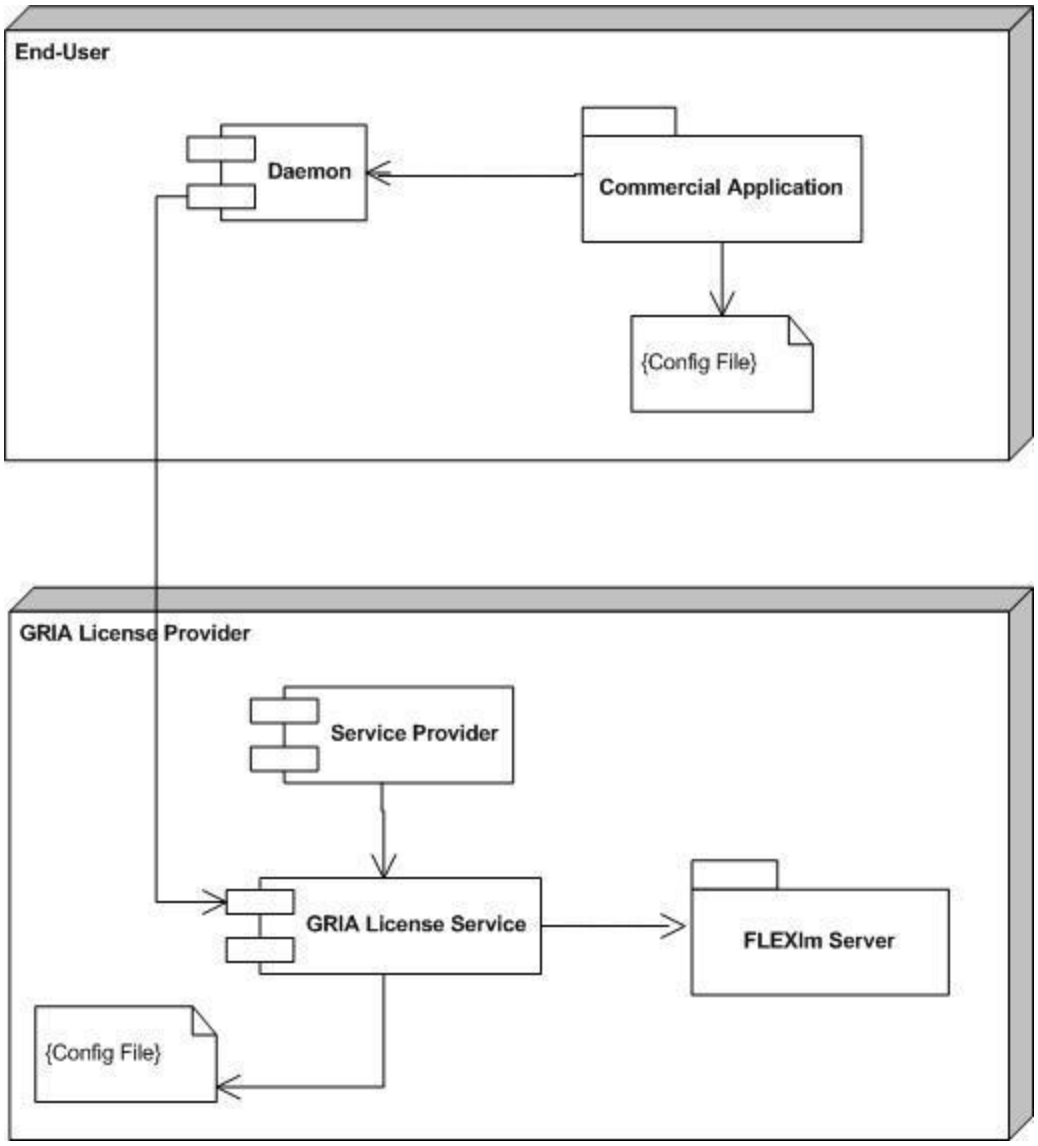
Παρατηρούμε από το παραπάνω σχήμα, ότι ο End User πλέον επικοινωνεί μέσω του διαδικτύου με κατάλληλα μηνύματα με τον πιστοποιημένο παροχέα GRIA υπηρεσιών ώστε να αποκτήσει την απαιτούμενη άδεια λογισμικού για να εκκινήσει την εφαρμογή. Εδώ πρέπει να σημειωθεί πως η εφαρμογή-πελάτης εμπεριέχεται στον End-User, ενώ ο FLEXlm είναι κομμάτι του GRIA License Provider. Τα δύο αυτά στοιχεία εξακολουθούν να λειτουργούν όπως και πριν, παρεμβαίνονται όμως οι κατάλληλες οντότητες λογισμικού που μεταφέρουν την TCP κίνηση στο διαδίκτυο, παρέχοντας παράλληλα και τις απαιτούμενες λειτουργίες που θα εξασφαλήσουν επιχειρησιακή ολοκλήρωση.

Ο End-User αποτελείται από δύο οντότητες, την εφαρμογή-πελάτη και τον daemon, μια εφαρμογή καταναλωτή υπηρεσιών πολυπλέγματος, η οποία και είναι υπεύθυνη για την επικοινωνία με την αντίστοιχη διαδικτυακή υπηρεσία. Οι δύο αυτές οντότητες επικοινωνούν μεταξύ τους με TCP μηνύματα στο διάστρωμα μεταφοράς, ενώ ο εντοπισμός του deamon από την εφαρμογή γίνεται με χρήση ενός αρχείου (configuration file) όπου και έχει αποθηκευμένη την IP διεύθυνση του daemon. Αξίζει να σημειώσουμε ότι και οι δύο βρίσκονται στο ίδιο τοπικό δίκτυο, ενώ μπορεί να συνυπάρχουν στο ίδιο υπολογιστικό σύστημα. Ανάλογα με τις ανάγκες της κάθε επιχείρησης, μπορεί να υπάρχουν ένας ή περισσότεροι daemon εγκατεστημένοι ή ακόμα και κάθε υπολογιστικός κόμβος που πρόκειται να εκτελέσει μια συγκεκριμένη εφαρμογή να έχει εγκατεστημένο το δικό του daemon. Κατά αυτόν τον τρόπο επιτρέπεται στην εφαρμογή να αναζητήσει μια δεύτερη τέτοια λογισμική οντότητα στην περίπτωση που η πρώτη δεν είναι διαθέσιμη, συμβάλλοντας στη μη ύπαρξη ενός κεντρικού σημείου αποτυχίας (central point of failure).

Ο GRIA License Provider αποτελείται από τον FLEXlm server και λοιπές διαδικτυακές υπηρεσίες που ακολουθούν το υπηρεσιοστρεφές πρότυπο και κάνουν χρήση των τεχνολογιών του πολυπλέγματος GRIA. Και εδώ, ο FLEXlm λειτουργεί όπως και στο υπάρχον μοντέλο παροχής αδειών, ανταλλάσοντας TCP μηνύματα μέσω sockets που ανοίγει με κάποια άλλη οντότητα στο ίδιο τοπικό δίκτυο. Ο ρόλος του διαμεσολαβητή έχει ανατεθεί σε αυτό το τμήμα στο License Service, μια διαδικτυακή υπηρεσία που έχει αναπτυχθεί με χρήση των βιβλιοθηκών του μεσισμικού GRIA. Ο FLEXlm αποτελεί στην ουσία πόρο (resource) του GRID ο οποίος προσπελάσσεται μέσω του τελευταίου, δίνοντας τη δυνατότητα για εκτέλεση της επιθυμητής εφαρμογής. Η ίδια η εφαρμογή δηλαδή δεν είναι μέρος του GRID, παρόλα αυτά για να μπορέσει να εκτελεστεί απαιτείται δέσμευση κάποιου πόρου, δέσμευση η οποία μπορεί να μετρηθεί και να χρεωθεί με βάση τις δυνατότητες για επιχειρησιακή ανάπτυξη λογισμικού που παρέχει το GRIA. Ο GRIA License Provider αποτελείται επίσης από δύο άλλες ονότητες λογισμικού, το Trade Account Service, και το SLA Service. Αυτές επικοινωνούν με το License Service, επιτρέποντας την ποσοτική μέτρηση της υπηρεσίας, την εξαγωγή στατιστικών και αναφορών που αφορούν τη χρήση της, και εν τέλη τη χρέωση στον πιστοποιημένο χρήστη που την καταναλώνει. Όλες οι οντότητες που αποτελούν διαδικτυακές υπηρεσίες επικοινωνούν εδώ μέσω SOAP μηνυμάτων, τα οποία είναι κρυπτογραφημένα με χρήση τεχνολογίας ιδιωτικού/δημόσιου κλειδιού (Public Key Infrastructure - PKI). Αντίθετα ο FLEXlm server επικοινωνεί μόνο με τον GRIA License Provider μέσω TCP sockets. Αυτό δεν αποτελεί πρόβλημα ασφάλειας, καθώς βρίσκεται στο ίδιο τοπικό δίκτυο του παρόχου της υπηρεσίας και δεν είναι απευθείας προσπελάσιμος από την εφαρμογή-πελάτη, αλλά πάντα

δια μέσου της διαδικτυακής υπηρεσίας, η οποία και αναλαμβάνει την πιστοποιήση και εξουσιοδότηση να παράλαβει την άδεια λογισμικού εκ μέρους της εφαρμογής-πελάτη.

Το διάγραμμα οντοτήτων της προτεινόμενης αρχιτεκτονικής φαίνεται στο ακόλουθο σχήμα:



Εικόνα 3.3 – Διάγραμμα Οντοτήτων Προτεινόμενης Αρχιτεκτονικής

Η επικοινωνία μεταξύ End-User και GRIA License Provider γίνεται μέσω του διαδικτύου. Αυτό απαιτεί χρήση του HTTP πρωτοκόλλου όπως αναφέρθηκε παραπάνω και η επικοινωνία γίνεται στο διάστρωμα εφαρμογών. Επιπλέον, αξίζει να τονιστεί στον αναγνώστη ότι τα δύο άκρα της συγκεκριμένης αρχιτεκτονικής αποτελούν τον καταναλωτή και τον πάροχο διαδικτυακών υπηρεσιών αποτέλεσμα του οποίου είναι, σύμφωνα με την υπηρεσιοστρεφή

αρχιτεκτονική, το HTTP να μεταφέρει σαφώς καθορισμένα και προτυποποιημένα SOAP μηνύματα. Το ότι η επικοινωνία γίνεται σε τόσο υψηλό επίπεδο και η χρήση σαφώς ορισμένων προτύπων επιτρέπει τη διασύνδεση δύο ή περισσότερων ετερογενών δικτύων μεταξύ τους. Αυτό είναι πολύ σημαντικό από επιχειρηματική σκοπιά, καθώς δεν απαιτείτα ο πάροχος και ο κάθε καταναλωτής να έχουν συμφωνήσει εξ αρχής στο τι τεχνολογίες θα χρησιμοποιήσουν και επιτρέπεται δυνητικά στον καθένα να χρησιμοποιήσει το συγκεκριμένο μοντέλο. Το γεγονός παρόλα αυτά της έκθεσης του παροχέα αδειών στο διαδίκτυο θα μπορούσε να αποτελεί σημαντικό πρόβλημα ασφάλειας. Αυτό δεν υφίσταται καθώς εξ σχεδιασμού της αρχιτεκτονικής ο FLEXlm δεν έχει εκθέσει όλες τις πόρτες του στους τρίτους όπως στο υπάρχον μοντέλο, αλλά η πρόσβαση γίνεται μόνο με μια προκαθορισμένη και γνωστή θύρα (πχ. 80, 8080, 8443). Επιπλέον γίνεται χρήση τεχνολογιών ιδιωτικού/δημοσίου κλειδιού για την πιστοποίηση του καταναλωτή και του παροχέα, πράγμα που επιτρέπει την ασφαλή πιστοποίηση του καθενός από τα δύο συμβαλλόμενα μέρη και την κρυπτογράφηση της μεταξύ τους κίνησης στο διαδίκτυο. Έτσι είναι αδύνατο σε κάποιον τρίτον να υποκλέψει μία άδεια λογισμικού και να τη χρησιμοποιήσει στη θέση του αιτούμενου. Τέλος, το σύστημα που προτείνεται, εξασφαλίζει ότι μόνο οι εξουσιοδοτημένοι χρήστες θα μπορούν να προσπελάσουν τη διαδικτυακή υπηρεσία και να προμηθευτούνε την επιθυμητή άδεια εκτέλεσης της εφαρμογής τους με κατάλληλες ρυθμίσεις στην κεντρική σελίδα διαχείρισης της υπηρεσίας που παρέχει το μεσισμικό GRIA. Κάτι τέτοιο στο υπάρχον σύγχρονο μοντέλο παροχής αδειών είναι αρκετά δύσκολο τεχνικά, καθώς απαιτείται κάθε φορά κατάλληλη ρύθμιση στο firewall που βρίσκεται μπροστά στον FLEXlm server ή την εγκατάσταση και επαναρύθμιση ενός proxy server έτσι ώστε να διασφαλίζεται ότι μόνο οι επιθυμητοί χρήστες θα είναι ικανοί να τον προσπελλάσουν. Αν υποθέσουμε ότι σε ένα επιχειρησιακό μοντέλο οι χρήστες μιας εφαρμογής αλλάζουν διαρκώς, αυτό απαιτεί συνεχή ρύθμιση των παραπάνω, που όπως σημειώθηκε, είναι αρκετά δύσκολο τεχνικά και χρονοβόρο.

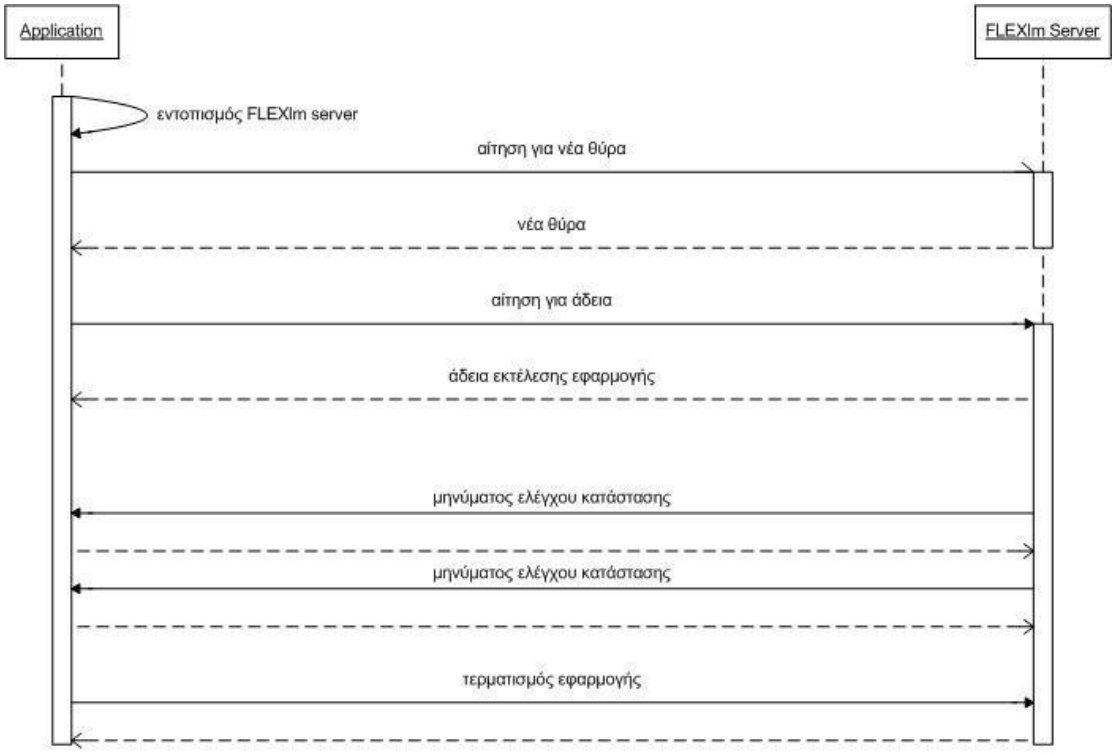
3.2 Περιγραφή Λειτουργιών

Για να μπορέσουν να γίνουν κατανοητές οι λειτουργίες της αρχιτεκτονικής που προτείνεται στην παρούσα διπλωματική και να αναδειχθούν οι διαφορές της με την υπάρχουσα, θα γίνει μια σύντομη περιγραφή της λειτουργίας του σύγχρονου μοντέλου παροχής αδειών λογισμικού όπως υπάρχει σήμερα, και στη συνέχεια αναλυτική περιγραφή του προτεινόμενου μοντέλου.

3.2.1 Λειτουργία Υπάρχουσας Αρχιτεκτονικής του Σύγχρονου Μοντέλου

Η διαδικασίας απόκτησης άδειας χρήσης λογισμικού στο σημερινό σύγχρονο μοντέλο παροχής ακολουθεί την παρακάτω διαδικασία.

Η εφαρμογή πελάτης διαβάζει από ένα license file την IP διεύθυνση και τη θύρα που περιμένει αιτήσεις ο εξυπηρετητής και προσπαθεί να συνδεθεί. Ο FLEXlm server βρίσκεται σε διαρκεί αναμονή αιτήσεων προς όλες τις θύρες που έχει ανοιχτές ανά εφαρμογή που εξυπηρετεί, σύμφωνα με το configuration file του, όπως αναφέρθηκε και στο δεύτερο κεφάλαιο. Η εφαρμογή ξεκινά την εκτέλεσή της δημιουργώντας ένα socket στη συγκεκριμένη IP διεύθυνση και θύρα που διάβασε στο αρχείο. Στη συνέχεια λαμβάνει από την απόκριση του μηνύματος από τον εξυπηρετητή τη δεύτερη προς σύνδεση θύρα. Έτσι δημιουργεί ένα δεύτερο socket στην αντίστοιχη πλέον θύρα του εξυπηρετητή. Αφού ανταλλάξουν οι δύο ονότητες μηνύματα αναγνώρισης και γίνει ο απαραίτητος έλεγχος από μεριά FLEXlm ότι υπάρχουν διαθέσιμες άδειες για εκτέλεση, αποστέλεται η άδεια και η εφαρμογή εκκινεί. Όσο η εφαρμογεί εκτελείται, το socket παραμένει ανοιχτό και ανταλάσσονται διαρκώς μηνύματα μεταξύ πελάτη και εξυπηρετητή προκειμένου ο τελευταίος να διαπιστώση την κατάσταση του πελάτη και να αποδεσμεύσει την άδεια σε περίπτωση που αυτός είναι για κάποιο λόγο ανενεργός. Όταν η εφαρμογή τερματίσει, αποστέλεται κατάλληλο μήνυμα και το socket κλείνει. Τα παραπάνω φαίνονται αναλυτικότερα στο ακόλουθο διάγραμμα ροής δεδομένων.



Εικόνα 3.4 – Διάγραμμα Ροής Υπάρχουσας Αρχιτεκτονικής

3.2.2 Λειτουργία Προτεινόμενης Αρχιτεκτονικής

Η προτεινόμενη αρχιτεκτονική όπως αναφέρθηκε αποτελείται από δύο βασικές οντότητες, τον End-User και τον GRIA License Provider, και αυτές με τη σειρά τους αποτελούνται από την εφαρμογή-πελάτη και τον daemon, και το Licenser Service, τον FLEXlm server και τις διαδικτυακές υπηρεσίες του GRIA οι οποίες και αναλαμβάνουν την καταγραφή της χρήσης της υπηρεσίας από τον καταναλωτή. Η διαδικασία παροχής άδειας στο προτεινόμενο μοντέλο έχει ως εξής:

Ο χρήστης εκκινεί την εφαρμογή του, η οποία αναζητεί στο license file τη διεύθυνση IP και θύρα του FLEXlm server. Στο συγκεκριμένο αρχείο έχουν αποθηκευτεί τα δικτυακά στοιχεία του daemon. Έτσι η εφαρμογή νομίζει ότι αποστέλει μήνυμα αίτησης στον FLEXlm server, αλλά το μήνυμα στην ουσία το παραλαμβάνει ο daemon. Κατά αυτόν τον τρόπο δε γίνεται καμία παρέμβαση στην υλοποίηση της εφαρμογής που θέλουμε να εκτελέσουμε.

Ο daemon, για κάθε αίτηση δημιουργεί μια καινούρια διεργασία για εξυπηρέτηση της αίτησης. Δημιουργεί τα κατάλληλα input/output streams για επικοινωνία πάνω στο TCP στρώμα με την εφαρμογή, και μέσω του GRIA client που υλοποιεί δημιουργεί SOAP πακέτα για επικοινωνία με το License Service. Εδώ πρέπει να τονιστεί ότι τα μηνύματα αυτά είναι κρυπτογραφημένα με τεχνολογία δημόσιου/ιδιωτικού κλειδιού, και έτσι ο daemon αναγνωρίζει ποιος χρήστης έκανε αίτηση και στέλνει τα αντίστοιχα κλειδιά του στο service, τα οποία μπορεί να εντοπίσει με βάση τα δικτυακά χαρακτηριστικά του χρήστη και μια βάση δεδομένων. Στην απλή περίπτωση που εξετάζουμε εδώ θεωρούμε ότι κάθε εφαρμογή έχει εγκατεστημένο το δικό της daemon άρα είναι συγκεκριμένο το κλειδί που πρέπει να ενσωματωθεί και να κρυπτογραφήσει κατάλληλα το μήνυμα και δε χρειάζεται ο εντοπισμός του.

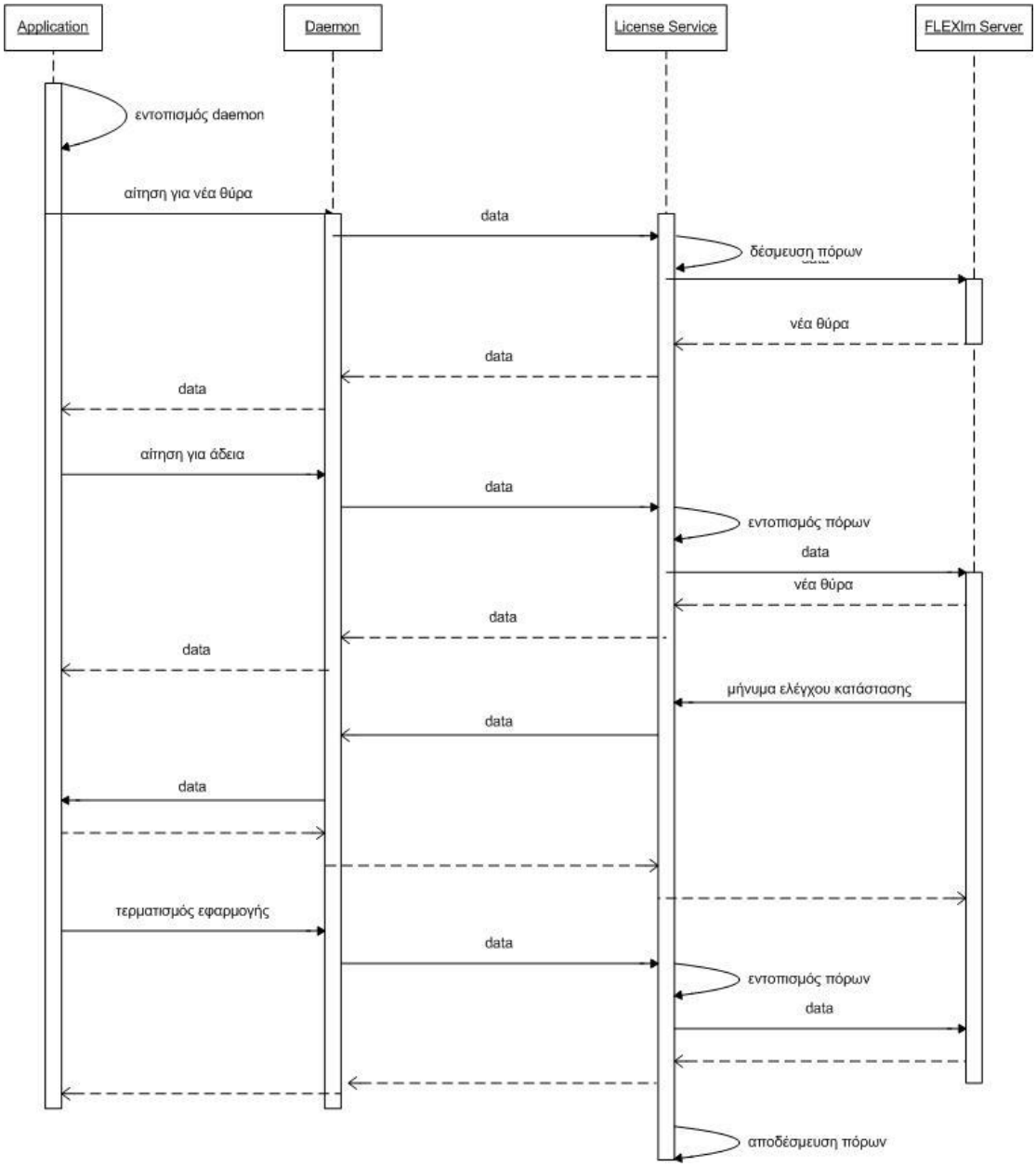
Το License Service παραλαμβάνει την αίτηση του daemon για προσπέλαση του FLEXlm server. Σε αυτό το σημείο είναι δουλειά του μεσισμικού πολυπλέγματος GRIA να αρχικοποιήσει τους πόρους που θα διαθέσει για εκτέλεση της υπηρεσίας από τον καταναλωτή. Στην αρχή γίνεται έλεγχος με το Trade Account Service για το αν ο χρήστης έχει δημιουργήσει κατάλληλο λογαριασμό προς χρέωση. Στη συνέχεια και εφόσον πιστοποιηθεί ο χρήστης και είναι εξουσιοδοτημένος γίνεται χρήση του συμφωνημένου SLA για καταγραφή και χρέωση της χρήσης του και δεσμεύονται οι αντίστοιχοι πόροι στο πολυπλέγμα, με σεβασμό στους όρους και τους περιορισμούς που θέτονται στο SLA για την εξυπηρέτηση της αίτησής του. Τότε και μόνο τότε, το License Service ξεκινάει την επικοινωνία με το FLEXlm server που γνωρίζει μέσω ενός αρχείου ρυθμήσεων και του αποστέλεται η αίτηση για λήψη της άδειας χρήσης της επιθυμητής εφαρμογής. Το Service ανοίγει ένα session το οποίο και διατηρεί ανοιχτό καθόλη την εκτέλεση της εφαρμογής και το κλείνει όταν αυτή τερματίσει ή όταν διαπιστωθεί ότι ο χρήστης είναι ανενεργός. Στο session αποθηκεύονται νέα input/output streams για την επικοινωνία του License Service με τον FLEXlm, η οποία γίνεται όπως έχει αναφερθεί μέσω TCP μηνυμάτων πάνω στο διάστρωμα μεταφοράς.

Ο FLEXlm server παραλαμβάνει το αίτημα του χρήστη, διαμέσου του service που λειτουργεί εκ μέρους του τελευταίου και εκτελεί τις ενέργειες που θα έκανε και στο υπάρχον μοντέλο παροχής αδειών. Κατά αυτόν τον τρόπο νομίζει ότι επικοινωνεί απευθείας με την εφαρμογή-πελάτη. Έτσι δεν υπάρχει απαίτηση για παρέμβαση στην υλοποίησή του. Κατά τα γνωστά, αποστέλει μέσω TCP μηνύματος τη νέα θύρα για σύνδεση, την οποία και αποστέλει το License Service πίσω στον daemon –καταναλωτή της υπηρεσίας- και τελικά στην εφαρμογή πελάτη.

Η εφαρμογή πελάτης αποστέλει νέο μήνυμα για σύνδεση στη νέα θύρα η οποία μεταφέρεται μέσω του daemon στο License Service. Σε αυτό το σημείο γίνεται έλεγχος πάλι από το μεσισμικό πολυπλέγματος GRIA για την πιστοποίηση και εξουσιοδότηση του χρήστη να εκτελέσει τη συγκεκριμένη υπηρεσία. Διαπιστώνεται ότι υπάρχουν ήδη δεσμευμένοι πόροι στο πολυπλέγμα για αυτόν, επομένως προωθεί την αίτηση στο License Service, με το SLA service να συνεχίζει να καταγράφει τη χρήση της υπηρεσίας. Η τελευταία εντοπίζει τους δεσμευμένους για τον χρήστη πόρους, άρα και το session που έχει ανοιχτεί για την εξυπηρέτησή του, και χρησιμοποιώντας τα ήδη υπάρχοντα streams συνεχίζει να προωθεί τα μηνύματα στον FLEXlm server.

Όταν η εφαρμογή-πελάτης τερματίσει, αποστέλεται κατάλληλο μήνυμα στον FLEXlm server, αυτό αναγνωρίζεται από το License Service, και αφού το προωθήσει για τις κατάλληλες ενέργειες, κλείνει το socket και το session που έχει ανοίξει, ενώ παράλληλα αποδεσμεύει τους πόρους που είχε κρατήσει για τον χρήστη. Το Service Management Provider component του GRIA έχει καταγράψει όλη τη χρήση και ο διαχειριστής της υπηρεσίας μπορεί μετά από κάποιο χρονικό διάστημα να στείλει στο χρήστη τιμολόγιο με τη χρέωση που του έχει γίνει με βάση το συμφωνημένο SLA.

Θεωρώντας ότι οι λειτουργίες του μεσισμικού GRIA που αφορούν την καταγραφή της χρήσης της υπηρεσίας έχουν αναπτυχθεί σε προηγούμενο κεφάλαιο και μπορούν να αναζητηθούν σε διάφορες πηγές, ακολουθεί το διάγραμμα ροής δεδομένων της προτεινόμενης αρχιτεκτονικής, όπου παραλείπονται οι συγκεκριμένες οντότητες για την απλούστευσή του. Επειδή επιπλέον ο τρόπος με τον οποίο δημιουργούνται, χρησιμοποιούνται και εντοπίζονται οι πόροι που δεσμεύονται από τη διαδικτυακή υπηρεσία πολυπλέγματος δεν είναι αντικείμενο της παρούσης ενότητας και θα αναλυθούνε εκτενώς σε επόμενο κεφάλαιο, παραλείπονται και αυτοί. Το διάγραμμα ροής δεδομένων που περιλαμβάνει τον FLEXlm server, την εφαρμογή-πελάτη, καθώς επίσης και το daemon και το License Service, φαίνεται ακολούθως:



Εικόνα 3.5 – Διάγραμμα Ροής Προτεινόμενης Αρχιτεκτονικής

4

Σχεδιασμός συστήματος

Σε αυτό το κεφάλαιο θα γίνει αναφορά στην υλοποίηση της προτεινόμενης αρχιτεκτονικής για το σύγχρονο μοντέλο παροχής αδειών λογισμικού, όπως αυτή περιγράφηκε στην προηγούμενη ενότητα. Αρχικά θα περιγραφούν οι τεχνολογίες που επιλέχθηκαν για την ανάπτυξη του συστήματος και τα πλεονεκτήματά τους σε σύγκριση με άλλες, και στη συνέχεια το διάγραμμα των κλάσεων και η περιγραφή των λειτουργιών όλων των συνιστωσών του συστήματος.

4.1 Πλατφόρμες και προγραμματιστικά εργαλεία

Για την ορθή εγκατάσταση, λειτουργία και έλεγχο του συστήματος που υλοποιήθηκε για να αντικαταστήσει το σημερινό σύγχρονο μοντέλο παροχής αδειών λογισμικού, έγινε χρήση διαφόρων τεχνολογιών, πλατφορμών και προγραμματιστικών εργαλείων. Έτσι έγινε χρήση της Java σαν η κύρια γλώσσα προγραμματισμού των κλάσεων και των συναρτήσεών τους, ο Apache Tomcat Server σαν εξυπηρετητής των διαδικτυακών υπηρεσιών, το μεσισμικό GRIA που ήδη έχει αναφερθεί εκτενώς σε προηγούμενα κεφάλαια, και μια σειρά άλλες τεχνολογίες η κάθε μία από τις οποίες πρόσθεσε ιδιαίτερα χαρακτηριστικά και δυνατότητες στο σύστημα, όπως οι τεχνολογία κρυπτογράφησης ιδιωτικού/δημοσίου κλειδιού, η τεχνολογία hibernate για την αποθήκευση αντικειμένων και μεταβλητών στη διάρκεια ενός session, ενώ η

ανάπτυξη έγινε με τη βοήθεια του προγραμματιστικού εργαλείου Eclipse με χρήση του Maven Project.

4.1.1 Η Γλώσσα Προγραμματισμού Java

Η Java [25] είναι μια από τις πιο διαδεδομένες αντικειμενοστραφείς γλώσσες προγραμματισμού που σχεδιάστηκε αρχικά από τη Sun Microsystems και κυκλοφόρησε για πρώτη φορά το 1995 σαν ένα στοιχείο του πυρήνα της Java πλατφόρμας της Sun. Η σύνταξή της προέρχεται κατά κύριο μέρος από τις γλώσσες προγραμματισμού C και C++ αλλά έχει πιο απλό αντικειμενοστραφές μοντέλο και λιγότερες ευκολίες χαμηλού επιπέδου. Οι Java εφαρμογές μεταγλωττίζονται τυπικά σε δυαδικό κώδικα ο οποίος μπορεί να εκτελεστεί σε οποιαδήποτε εικονική μηχανή της Java (Java Virtual Machine – JVM) ανεξαρτήτως της αρχιτεκτονικής του υπολογιστικού συστήματος που εκτελείται. Ο ίδιος ο δημιουργός της Java, James Gosling, υποσχέθηκε μια γλώσσα "Write Once, Run Anywhere" που σημαίνει ότι οποιοδήποτε λογισμικό που έχει αναπτυχθεί με Java μπορεί να εκτελεστεί οπουδήποτε, χωρίς την παραμικρή αλλαγή.

Οι αρχικοί μεταγλωττιστές της Java, οι εικονικές μηχανές και οι βιβλιοθήκες κλάσεων αναπτύχθηκαν από τη Sun το 1995. Από το 2007 και σε συνεργασία με τις προδιαγραφές της Java Community Process η Sun έκανε διαθέσιμο το μεγαλύτερο μέρος της τεχνολογίας της ως ελεύθερο λογισμικό κάτω από την άδεια General Public License (GNU) [26].

Οι πρωτεύοντες στόχοι κατά τη δημιουργία της Java ήταν οι ακόλουθοι:

1. Έπρεπε να υιοθετεί όλες τις αρχές του αντικειμενοστραφούς προγραμματισμού.
2. Έπρεπε να επιτρέπει τη δυνατότητα να εκτελείται η ίδια εφαρμογή σε οποιαδήποτε αρχιτεκτονική υπολογιστικού συστήματος.
3. Έπρεπε να περιλαμβάνει βιβλιοθήκες κλάσεων που να επιτρέπουν και να υποστηρίζουν σε ικανοποιητικό βαθμό τη χρήση δικτύων
4. Έπρεπε να σχεδιαστεί ώστε να μπορεί να εκτελεί εφαρμογές από απομακρυσμένες πηγές με ασφάλεια.
5. Έπρεπε τέλος να είναι όσο γίνεται πιο φιλική στο χρήστη υιοθετώντας όλα τα θετικά στοιχεία των υπολοίπων γλωσσών προγραμματισμού.

Όλες οι παραπάνω προδιαγραφές υποδουλώνουν τα πλεονεκτήματα που απορρέουν από τη χρήση της Java σαν κύρια γλώσσα ανάπτυξης λογισμικού και τους λόγους για τους οποίους επιλέχθηκε για την υλοποίηση της προτεινόμενης αρχιτεκτονικής που εξετάζεται στην παρούσα διπλωματική εργασία. Έτσι πιο αναλυτικά, εξαιτίας της κατασκευής της ως αντικειμενοστραφής γλώσσα προγραμματισμού παρέχει απλότητα στο σχεδιασμό, διαχωρίζει τον κώδικα σε ξεχωριστές και αυτόνομες μονάδες, και καθιστά εφικτές τις μικρές αλλαγές στον κώδικα του συστήματος σε πολύ σύντομο χρονικό διάστημα χωρίς ταυτόχρονα να επηρεάζει τη συνολική λειτουργία του. Ένα άλλο σημαντικό στοιχείο της συγκεκριμένης γλώσσας προγραμματισμού είναι η δυνατότητα επέκτασης του κώδικα προσθέτοντας νέες λειτουργίες χωρίς ιδιαίτερη δυσκολία, η ευκολία στον εντοπισμό και διόρθωση των σφαλμάτων που προκύπτουν κατά την ανάπτυξη και συντήρηση της εφαρμογής, ενώ τέλος παρέχεται η δυνατότητα επαναχρησιμοποίησης κλάσεων και μεθόδων σε διαφορετικά προγράμματα. Όλα τα παραπάνω είναι σημαντικά χαρακτηριστικά στη βιομηχανία λογισμικού και οδηγούνε στη δημιουργία ευέλικτων και επεκτάσιμων προγραμμάτων και εφαρμογών.

Η ανεξαρτησία πλατφόρμας που παρέχει η Java είναι ένα άλλο σημαντικότατο χαρακτηριστικό της συγκεκριμένης γλώσσας. Δίνει τη δυνατότητα ώστε κάθε οντότητα λογισμικού γραμμένη σε Java να μπορεί να εκτελεστεί πάντα και κατά τον ίδιο τρόπο σε οποιαδήποτε πλατφόρμα υλικού ή λειτουργικού συστήματος. Κατά αυτόν τον τρόπο εξασφαλίζεται και η λειτουργία της αρχιτεκτονικής που περιγράφουμε και η οποία θέτει σαν μία από τις κυριότερες απαιτήσεις της τη δυνατότητα διασυνδεσιμότητας ετερογενών δικτύων που αποτελούνται από διαφορετικά υπολογιστικά συστήματα. Επιπλέον, μέσω των βιβλιοθηκών που παρέχει είναι πλέον η πιο διαδεδομένη γλώσσα για ανάπτυξη διαδικτυακών εφαρμογών και υπηρεσιών με πολλά πλεονεκτήματα αναφορικά με τη γρήγορη και ασφαλή διαδικτυακή επικοινωνία, βασικότατα χαρακτηριστικά στην παροχή αδειών λογισμικού με χρήση δικτύων, όπως και αναφέρθηκε στο δεύτερο κεφάλαιο.

Εναλλακτικά, αντί για τη χρήση της Java σαν κύρια γλώσσα ανάπτυξης του συστήματος θα μπορούσε να χρησιμοποιηθεί κάποια άλλη αντικειμενοστραφής γλώσσα προγραμματισμού που θα διατηρούσε τα περισσότερα από τα θετικά στοιχεία της Java. Οι άλλες δύο επιλογές ήταν η C++ και η C# της προγραμματιστικής πλατφόρμας .NET της Microsoft [27]. Η πρώτη απορρίφθηκε καθώς είναι σχεδιασμένη για εφαρμογές χαμηλότερου επιπέδου, επιτρέποντας τη χρήση δεικτών και διευθύνσεων πάνω στη μνήμη αλλά υστερεί στις διαδικτυακές εφαρμογές. Η δεύτερη από την άλλη είναι και αυτή μια δικτυακή γλώσσα προγραμματισμού, παρόλα αυτά περιορίζει κατά πολύ τον προγραμματιστή και τον χρήστη του συστήματος στη

χρήση ενός συγκεκριμένου λειτουργικού συστήματος και αρχιτεκτονικής, πράγμα αντίθετο των απαιτήσεων του συστήματος που υλοποιείται την παρούσα διπλωματική. Για όλους αυτούς τους λόγους θεωρήθηκε η Java σαν η καλύτερη δυνατή επιλογή για τον προγραμματισμό των οντοτήτων λογισμικού του.

Στην παρούσα υλοποίηση έγινε χρήση της έκδοσης 1.6

4.1.2 Ο Apache Tomcat Υποδοχέας

Ο Apache Tomcat [28] υποδοχέας (container) είναι ένας ανοιχτού λογισμικού δικτυακός υποδοχέας – εξυπηρετητής εφαρμογών- που αναπτύχθηκε από την Apache Software Foundation. Υλοποιεί τα πρότυπα Java Servlets και Java Server Pages (JSPs) όπως αυτές προσδιορίζονται από την Sun Microsystems παρέχοντας ένα περιβάλλον για εκτέλεση Java κώδικα σε συνεργασία με ένα δικτυακό εξυπηρετητή. Διαθέτει εργαλεία για τη ρύθμιση των παραμέτρων του και τη διαχείρισή του, ενώ ταυτόχρονα μπορεί να ρυθμιστεί με κατάλληλη επεξεργασία των XML αρχείων του. Επιπλέον με κατάλληλες ρυθμίσεις επιτρέπει την ασφαλή επικοινωνία μεταξύ πελάτη και εξυπηρετητή με χρήση ζευγαριού ιδιωτικού/δημοσίου κλειδιού και κρυπτογράφηση στο επίπεδο μεταφοράς με χρήση του πρωτοκόλλου HTTPs. Ο Tomcat περιλαμβάνει το δικό του εσωτερικό HTTP εξυπηρετητή, ενώ δεν πρέπει να μπερδεύεται με τον Apache Web Server που είναι ένας καθαρά HTTP εξυπηρετητής γραμμένος σε γλώσσα C.

Ο Tomcat θεωρείται ο πιο διαδεδομένος και αξιόπιστος διαδικτυακός υποδοχέας και χρησιμοποιείται ευρέως στην ανάπτυξη διαδικτυακών εφαρμογών. Στην υλοποίηση φιλοξενεί όλη την οντότητα λογισμικού GRIA Service Provider, δηλαδή τις διαδικτυακές υπηρεσίες του μεσισμικού πολυπλέγματος GRIA, καθώς επίσης και την υπηρεσία License Service. Η έκδοση που χρησιμοποιήθηκε είναι η 6.0.

4.1.3 Hibernate

Το Hibernate [29] είναι ένα ισχυρό και υψηλής απόδοσης εργαλείο που επιτρέπει σε αντικείμενα κλάσεων να χρησιμοποιούνε υπηρεσίες ερωτημάτων σχεσιακών βάσεων δεδομένων. Επιτρέπει την ανάπτυξη persistent κλάσεων ακολουθώντας το

αντικειμενοστραφές μοντέλο με όλα τα χαρακτηριστικά του, όπως η κληρονομικότητα, ο πολυμορφισμός, η συσχέτιση, η σύνθεση και η συλλογή. Αντίθετα με πολλά άλλα persistence εργαλεία, το Hibernate δεν αποκρύπτει από τον προγραμματιστή τις δυνατότητες που παρέχει η SQL γλώσσα, αλλά μέσω της φορητής SQL επέκτασης που εμπεριέχει επιτρέπει τη χρήση αντίστοιχων σχεσιακών ερωτημάτων για την αποθήκευση, εξαγωγή και επεξεργασία δεδομένων. Στην ουσία αποτελεί ένα ενδιάμεσο στρώμα μεταξύ της βάσης δεδομένων και του στρώματος επιχειρησιακής λογικής (business logic) της εφαρμογής, μεταφέροντας δεδομένα από τη βάση σε συνεκτικά αντικείμενα (data beans) με άμεσο και αποδοτικό τρόπο, επιτρέποντας παράλληλα την εκτέλεση ερωτημάτων σχεσιακών βάσεων σε αυτά πριν την υποβολή (submit) τους πίσω στη βάση.

Για τους παραπάνω λόγους, το Hibernate κρίθηκε κατάλληλο για την αποθήκευση μεταβλητών κατά τη διάρκεια των συνεδρίων (session) μεταξύ του License Service και του daemon. Κρίθηκε προτιμότερο από τη χρήση μια βάσης δεδομένων για την ευκολότερη υλοποίηση, τη μεγαλύτερη ασφάλεια και την αποδοτικότερη ανάκτηση των δεδομένων αυτών, χαρακτηριστικά απαραίτητα για περιβάλλοντα διαδικτύου.

4.1.4 Maven Project

Το Maven Project [30] είναι προϊόν της εταιρίας Apache Software Foundation. Είναι ένα εργαλείο ανοιχτού κώδικα για τη διαχείριση και ανάπτυξη ενός προγράμματος λογισμικού. Βασίζεται στην ιδέα της ανάπτυξης μέσω μοντέλων αντικειμένων και βοηθάει στο χτίσιμο, την τεκμηρίωση και τον έλεγχο ενός έργου μέσω μιας κεντρικής πληροφορίας. Είναι χτισμένο σε Java και επιτρέπει την ανάπτυξη Java εφαρμογών.

Ένα από τα κύρια χαρακτηριστικά του είναι η διατήρηση του βασικού κύκλου ζωής της ανάπτυξης ενός προγράμματος λογισμικού σύμφωνα με τις αρχές της βιομηχανίας λογισμικού. Έτσι περιλαμβάνει όλα τα στάδια της ζωής της: από τον συντακτικό έλεγχο και τη μεταγλώττιση, μέχρι την εγκατάσταση σε έναν εξυπηρετητή και την τεκμηρίωση. Όλες οι πληροφορίες που αφορούν το πρόγραμμα βρίσκονται συγκεντρωμένες σε ένα κεντρικό αρχείο πληροφορίας με κατάληξη pom, μέσα στο οποίο ορίζονται το όνομα του προγράμματος και η έκδοσή του, οι διάφορες εξωτερικές βιβλιοθήκες που είναι απαραίτητες για να εκτελεστεί το είδος ελέγχου που θα πραγματοποιηθεί πριν την εγκατάσταση κα. Το Maven επιτρέπει επιπλέον την οργάνωση ολόκληρου του συστήματος σε οντότητες

λογισμικού, έχοντας όλα τα χαρακτηριστικά των αντικειμενοστραφή μοντέλων, παρέχοντας δυνατότητες κληρονομικότητας, πολυμορφισμού κτλ.

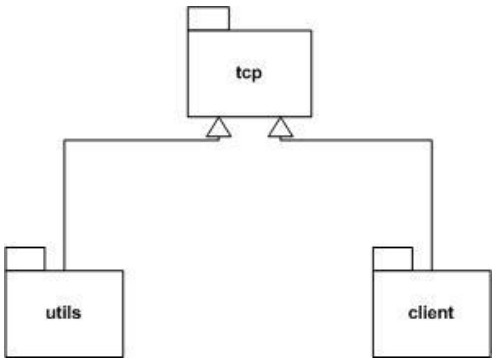
Στο σύστημα που αναπτύσσουμε κρίθηκε απαραίτητο για την καλύτερη οργάνωση του κώδικα σε προτυποποιημένη ιεραρχική δομή και την ευκολία που παρέχει για την άμεση μεταγλώττισή του και δημιουργία war αρχείων.

4.2 Περιγραφή Κλάσεων

Σε αυτή την υποενότητα θα γίνει μια αναλυτική περιγραφή των οντοτήτων που αποτελούν το κάθε συστατικό του συστήματος που υλοποιήθηκε. Όπως έχει ήδη αναφερθεί, το σύστημα αποτελείται από δύο κύριες συνιστώσες, πάνω στις οποίες στηρίχθηκε η ανάπτυξή του. Από τη μία υπάρχει το κομμάτι που αντιστοιχεί στην εφαρμογή-πελάτη, τον daemon, και από την άλλη το κομμάτι που αντιστοιχεί στον FLEXlm server, και είναι η GRIA License Service. Εδώ θα αναλυθούν οι κλάσεις σαν ξεχωριστές οντότητες που αποτελούν το κάθε ένα από τα δύο αυτά συστατικά.

4.2.1 Daemon

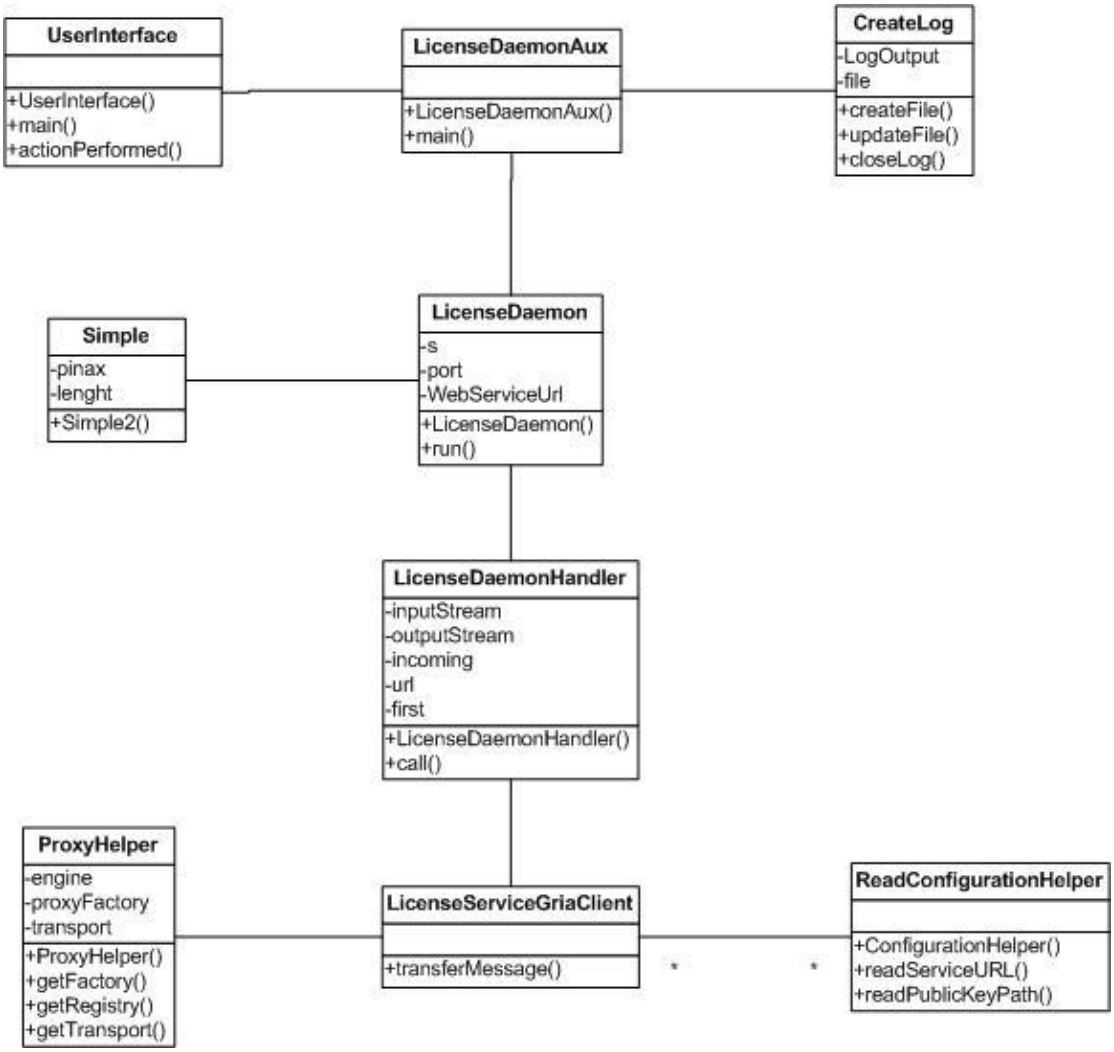
Αρχικά η οντότητα daemon είναι οργανωμένη σε packages, όπως φαίνονται στο ακόλουθο σχήμα:



Εικόνα 4.1 – Διάγραμμα Συνιστωσών του Daemon

Το package utils περιλαμβάνει την κλάση ProxyHelper, το package client την LicenseServiceGriaClient, ενώ το tcp περιλαμβάνει τις κλάσεις CreateLog, LicenseDaemon, LicenseDaemonAux, LicenseDaemonHandler, Simple2 και UserInterface. Το διάγραμμα

κλάσσεων, όπου αναγράφονται οι σημαντικότερες μεταβλητές και φαίνονται οι μέθοδοι που υλοποιούνται καθώς επίσης ο τρόπος που επικοινωνούν τα αντικείμενα αυτών φαίνεται ακολούθως:



Εικόνα 4.2 – Διάγραμμα Κλάσεων του Daemon

Η αναλυτική λειτουργία της κάθε κλάσης περιγράφεται παρακάτω:

UserInterface

Η UserInterface κλάση περιλαμβάνει το GUI αρχικοποίησης του daemon. Μέσω ενός γραφικού περιβάλλοντος ο διαχειριστής μπορεί να αρχικοποιήση τον daemon δίνοντάς του το URI του GRIA License Service που είναι υπεύθυνο για την παροχή της άδειας εκτέλεσης της εφαρμογής που επιθυμείται και ένα εύρος θυρών [ports] στις οποίες ακούει ο server.

Επιπλέον περιέχει ένα στοιχείο όπου θα αναγράφονται τα διάφορα μηνύματα που προκύπτουν καθ' όλη τη διάρκεια εκτέλεσής του.

LicenseDaemonAux

Η κλάση αυτή είναι η κεντρική κλάση η οποία περιλαμβάνει και τη main μέθοδο. Εκκινείται είτε από τη γραμμή εντολών με πέρασμα των κατάλληλων παραμέτρων, είτε από την UserInterface σε περίπτωση εκκίνησης του daemon από γραφική διεπαφή. Δέχεται τρία ορίσματα, το URI της διαδικτυακής υπηρεσίας και ένα εύρος [αρχικό και τελικό σημείο] θυρών στις οποίες ακούει ο εξηπυρετητής. Με την κλήση της, δημιουργεί τόσα νήματα (threads) της κλάσης LicenseDaemon, όσο και το εύρος των θυρών που δόθηκε σαν παράμετρος αρχικοποίησης.

CreateLog

Η CreateLog αρχικοποιείται κατά την εκκίνηση του daemon, από την LicenseDaemonAux, και καταγράφει ένα αρχείο μηνυμάτων, ανάλογα με τις ενέργειες και τα σφάλματα που λαμβάνουν χώρα κατά την εκτέλεσή του.

Simple

Βοηθητική κλάση αντικείμενα της οποίας δημιουργούνται σε κάθε μεταφορά πακέτων από τον daemon στην εφαρμογή-πελάτη. Στην ουσία περιλαμβάνει έναν πίνακα από bytes ο οποίος και περιέχει την ακολουθία των bytes που στέλνει ο FLEXlm server στην εφαρμογή.

LicenseDaemon

Η κλάση αυτή αρχικοποιείται από την LicenseDaemonAux, δημιουργώντας ένα νήμα για κάθε θύρα που δέχεται αιτήσεις ο FLEXlm server. Ανοίγοντας μια σύνδεση, περιμένει κάποια εφαρμογή-πελάτη να συνδεθεί με τον daemon, και στη συνέχεια για κάθε κλήση δημιουργεί ένα νέο νήμα της κλάσης LicenseDaemonHandler η οποία και αναλαμβάνει να εξυπηρετήση την αίτηση.

LicenseDaemonHandler

Η LicenseDaemonHandler αναλαμβάνει να εξυπηρετήσει τα μηνύματα της εφαρμογής-πελάτης προς τον FLEXlm server και αντίστροφα. Σε κάθε κλήση της δημιουργεί ένα αντικείμενο της κλάσης LicenseServiceGriaClient και καλεί την αντίστοιχη μέθοδο για να μεταφέρει τα δεδομένα που έχει συλλέξει από την εφαρμογή και να τα στείλει στη

διαδικτυακή υπηρεσία. Επιπλέον ανοίγει τα κατάλληλα streamings για την επικοινωνία με την εφαρμογή μέσω της σύνδεσης που υπάρχει.

LicenseServiceGriaClient

Εδώ βρίσκεται ο κώδικας ο οποίος υλοποιεί τον καταναλωτή της διαδικτυακής υπηρεσίας. Αφού αρχικοποιηθεί με τα κατάλληλα ορίσματα που περιλαμβάνουν το URI της επιθυμητής υπηρεσίας και μια ακολουθία από bytes που αντιπροσωπεύουν το μήνυμα της εφαρμογής προς τον FLEXlm server, θέτει τα κατάλληλα πιστοποιητικά για την ταυτοποίηση του χρήστη και καλεί τη διαδικτυακή υπηρεσία πολυπλέγματος με χρήση της βοηθητικής κλάσης ProxyHelper.

ProxyHelper

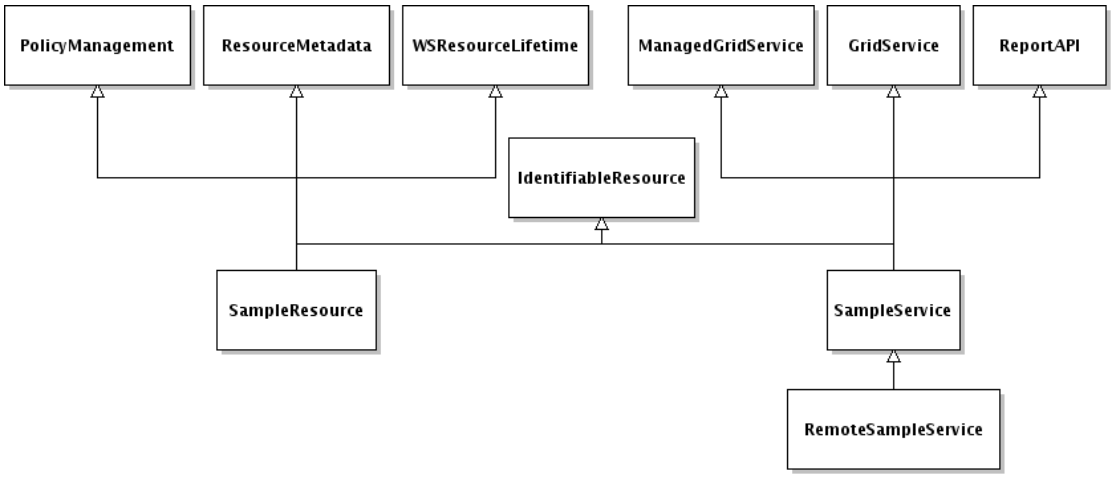
Πρόκειται για μια βοηθητική κλάση, η οποία κάνει τις κατάλληλες αρχικοποιήσεις για τη σύνδεση ενός καταναλωτή με τον πάροχο μια διαδικτυακής υπηρεσίας που κάνει χρήση του μεσισμικού πολυπλέγματος GRIA 5.3. Κρίθηκε σκόπιμο να διαχωριστεί από την LicenseServiceGriaClient καθώς η τελευταία αποτελεί έναν γενικό καταναλωτή διαδικτυακών υπηρεσιών. Δεν ενδιαφέρει δηλαδή σε το τι είδους υπηρεσία πρόκειται να γίνει η κλίση προς εξυπηρέτηση. Αντίθετα η ProxyHelper δε διαθέτει καμία πληροφορία σχετικά με τη λειτουργία της υπηρεσίας που πρόκειται να κλιθή. Απλώς αρχικοποιεί τα αντικείμενα που είναι απαραίτητα για την κλίση της. Με άλλα λόγια, στην LicenseServiceGriaClient ασχολείται με το λειτουργικό κομμάτι της υπηρεσίας (business) ενώ αντίθετα η ProxyHelper με το καθαρά τεχνικό.

ReadConfigurationHelper

Η κλάση αυτή περιλαμβάνει δύο μεθόδους για το πέρασμα των παραμέτρων ρυθμίσεων από το διαχειριστή του καταναλωτή. Ο τελευταίος θέτει μέσω ενός αρχείου ρυθμίσεων την ακριβής διεύθυνση της επιθυμητής διαδικτυακής υπηρεσίας του παρόχου, και τη διαδρομή στο δίσκο του δημόσιου κλειδιού του, για την κατάλληλη πιστοποίηση, και η κλάση με τις δύο αυτές μεθόδους θέτει τις τιμές αυτές στις αντίστοιχες μεταβλητές του daemon.

4.2.2 GRIA License Service

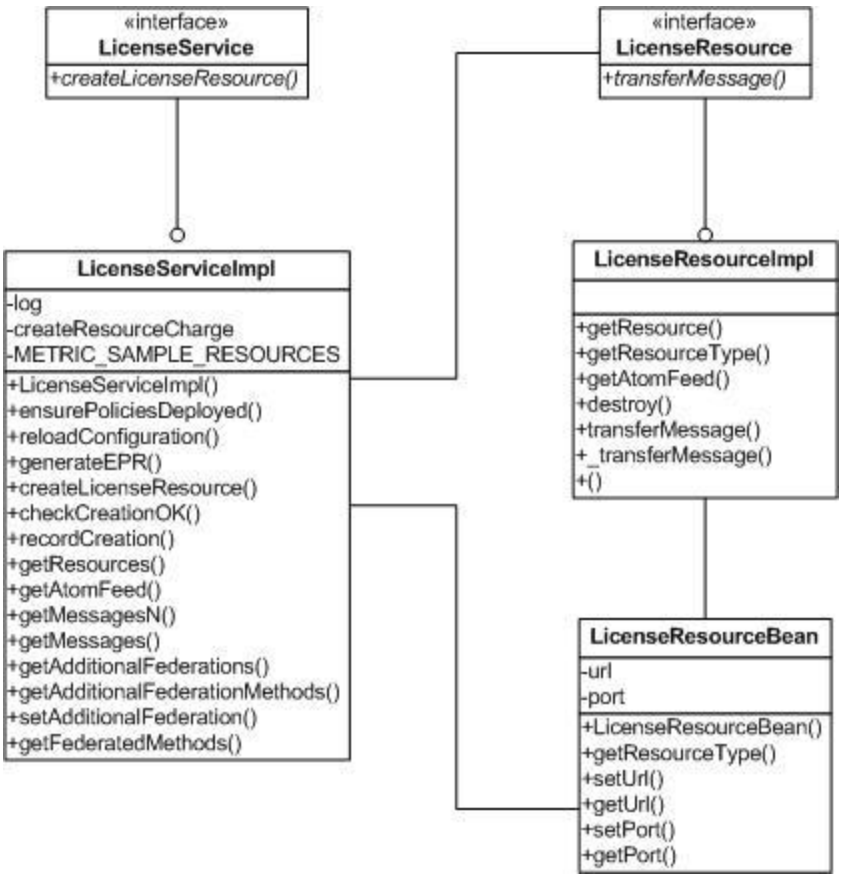
Η ανάπτυξη του GRIA License Service, της διαδικτυακής υπηρεσίας ποπλυπλέγματος δηλαδή που βρίσκεται στο κομμάτι του FLEXlm server, έγινε ακολουθώντας το πρότυπο του μεσισμικού GRIA όπως αυτό ορίζεται από την 5.3 έκδοσή του. Για την υλοποίηση χρησιμοποιήθηκαν οι αντίστοιχες βιβλιοθήκες [31]. Το σχηματικό διάγραμμα των διεπαφών που χρησιμοποιήθηκαν για την ανάπτυξη της υπηρεσίας φαίνεται στο ακόλουθο σχήμα:



Εικόνα 4.3 – Σχηματικό Διάγραμμα των Διεπαφών του License Service

Στο πρώτο επίπεδο φαίνονται οι διεπαφές (interfaces) που αποτελούν πρότυπα της υπηρεσιοστρεφούς αρχιτεκτονικής και του μεσισμικού GRIA. Αυτά σχετίζονται με την επιχειρισιακή λογική που παρέχει το GRIA και δίνει τη δυνατότητα καταγραφής της χρήσης των υπηρεσιών που υλοποιούνται με βάση το συγκεκριμένο μεσισμικό, ώστε να καταστεί δυνατή η μετέπειτα χρέωση. Άλλα από αυτά σχετίζονται με τον κύκλο ζωής και τη διαχείριση των πόρων που δεσμεύονται για την εξυπηρέτηση της κάθε κλήσης της υπηρεσίας. Στο κάτω επίπεδο φαίνονται οι διεπαφές που σχετίζονται με αυτή καθ' αυτή την υπηρεσία. Βλέπουμε εδώ μια διάκριση στις διεπαφές με κατάληξη Service και Resource. Οι πρώτες ορίζουν τις μεθόρους που δε χρειάζονται δέσμευση πόρων, και αφορούν κυρίως λειτουργίες όπως η ανακάλυψη της συνόδου που υπάρχει μεταξύ παρόχου και του καταναλωτή ή τη δημιουργία και τη δέσμευση πόρων, ενώ οι δεύτερες σχετίζονται με τις μεθόδους οι οποίες κάνουν χρήση ήδη δεσμευμένων πόρων.

Αναλυτικότερα οι κλάσεις οι οποίες αναπτύχθηκαν για την υλοποίηση της συγκεκριμένης διαδικτυακής υπηρεσίας φαίνονται στο ακόλουθο σχήμα:



Εικόνα 4.4 – Σχηματικό Κλάσεων του License Service

Οι τρεις κυριότερες κλάσεις περιγράφονται ακολούθως:

License ResourceBean

Σε αυτή την κλάση περιγράφονται οι μεταβλητές που αποθηκεύονται κατά τη διάρκεια της συνόδου μεταξύ παρόχου και καταναλωτή δεσμεύοντας τους αντίστοιχους πόρους στο πολυπλέγμα. Η κλάση κληρονομεί την GridResource για αυτό το σκοπό. Περιλαμβάνει τις getteres/setters αντίστοιχες μεθόδους. Η μεταβλητές αποθηκεύονται με χρήση hibernate.

License ServiceImpl

Εδώ πέρα περιγράφονται οι στατικές μέθοδοι που παρέχονται στον καταναλωτή για την κλήση της διαδικτυακής υπηρεσίας. Όλες οι μέθοδοι που περιγράφονται εδώ δε

χρησιμοποιούνε δεσμευμένους πόρους για την εκτέλεσή τους. Οι περισσότερες αποτελούν τμήμα του μεσισμικού GRIA για το λόγο αυτό και δεν αναλύονται περαιτέρω. Οι λειτουργίες τους σχετίσχονται με τη δημιουργία και δέσμευση πόρων σε κάποιον καταναλωτή, την καταγραφή των πόρων που χρησιμοποιούνται, την εύρεση των ήδη δεσμευμένων πόρων, την αποστολή του αντίστοιχου ERP κτλ.

License ResourceImpl

Η κλάση αυτή υλοποιεί τις μεθόδους που υλοποιούν λειτουργίες που απαιτούν την ύπαρξη δεσμευμένων πόρων στο πολυπλέγμα. Οι κυριότερες μέθοδοι σχετίζονται με την εύρεση των πόρων του αντίστοιχου αιτούμενου κάθε φορά καταναλωτή, την καταστροφή και αποδέσμευσή τους. Επιπλέον στη συγκεκριμένη κλάση υλοποιείται η μέθοδος η οποία σχετίζεται με τη μεταφορά των πακέτων επικοινωνίας μεταξύ του daemon και του FLEXlm server, και αντίστροφα. Σαν ορίσματα δέχεται τη θύρα που περιμένει αιτήσεις ο FLEXlm server και μια ακολουθία από bytes που αποτελεί και το μήνυμα της εφαρμογής-πελάτη προς τον τελευταίο. Επιστρέφει το μήνυμα απόκριση του FLEXlm server σε μορφή πάλι ακολουθίας bytes.

5

Έλεγχος

Στο συγκεκριμένο κεφάλαιο θα γίνει περιγραφή του ελέγχου και της εκτέλεσης του συστήματος, για να τεκμηριωθεί η αρχική μας υπόθεση, ότι είναι δυνατή η υλοποίηση ενός συστήματος το οποίο θα μπορεί να κατανέμει άδειες παροχής λογισμικού μεταξύ διαφορετικών και ετερογενών δικτύων, με απόλυτη ασφάλεια και δυνατότητας καταγραφής της χρήσης της εφαρμογής ώστε να είναι εφικτό η μετέπειτα τιμολόγηση και χρέωση της υπηρεσίας.

Στην αρχή θα γίνει αναφορά σχετικά με την εγκατάσταση του συστήματος, και εν συνεχεία θα παρουσιαστούν σενάρια εκτέλεσής του.

5.1 Οδηγός Εγκατάστασης

Για την ορθή λειτουργία του συστήματος παροχής αδειών λογισμικού, τόσο από την πλευρά του τελικού χρήστη όσο και από την πλευρά του παρόχου της διαδικτυακής υπηρεσίας, απαιτείται η κατάλληλη εγκατάσταση του περιβάλλοντος, δηλαδή των πλατφορμών και των τεχνολογιών που παρουσιάστηκαν στα προηγούμενα κεφάλαια, καθώς και η εγκαστάσταση των οντοτήτων που αποτελούν το νέο αυτό σύστημα.

5.1.1 Εγκατάσταση Περιβάλλοντος

5.1.1.1 Εγκατάσταση Περιβάλλοντος Εκτέλεσης Java (JRE)

Το αρχείο που χρησιμοποιείται για την εγκατάσταση του JRE βρίσκεται από την επίσημη ιστοσελίδα στην τοποθεσία http://java.sun.com και η έκδοση που χρησιμοποιήθηκε είναι η 1.6.0_11. Ανάλογα με το επιθυμητό λειτουργικό σύστημα εκτελείται το αντίστοιχο δυαδικό (binary) αρχείο και ακολουθούνται οι οδηγίες εγκατάστασης. Κατά αυτό τον τρόπο και βασιζόμενοι στην ιδιότητα της Java να εκτελείται σε οποιαδήποτε υπολογιστική πλατφόρμα και λειτουργικό σύστημα εξασφαλίζεται διασυνδεσιμότητα μεταξύ ετερογενών συστημάτων. Μετά την εγκατάσταση απαραίτητος είναι ο ορισμός μεταβλητών περιβάλλοντος, όπως αυτές ορίζονται από την επίσημη ιστοσελίδα. Πιο συγκεκριμένα:

1. JAVA_HOME: Η διαδρομή στο δίσκο που βρίσκεται αποθηκευμένο το Jdk
2. JRE_HOME: Η διαδρομή στο δίσκο που βρίσκεται αποθυκευμένο το jre
3. PATH: Προστίθεται στο path του συστήματος η διαδρομή που βρίσκονται τα προς εκτελέσημα αρχεία (java, javac, κτλ
4. CLASSPATH: Η διαδρομή στο δίσκο που βρίσκονται εγκατεστημένες όλες οι βιβλιοθήκες και τα jar αρχεία που θα χρησιμοποιηθούν από το σύστημα

5.1.1.2 Εγκατάσταση του Tomcat Server

Το αρχείο για την εγκατάσταση του Tomcat βρίσκεται στην επίσημη ιστοσελίδα του, στην τοποθεσία http://tomcat.apache.org . Στην παρούσα εφαρμογή έγινε χρήση της έκδοσης 6. Ανάλογα με το επιθυμητό λειτουργικό σύστημα εκτελείται το αντίστοιχο δυαδικό (binary) αρχείο και ακολουθούνται οι οδηγίες εγκατάστασης. Κατά αυτό τον τρόπο και πάλι ο πάροχος της διαδικυτακής υπηρεσίας είναι ελεύθερος να επιλέξει το λειτουργικό σύστημα που επιθυμεί να έχει. Έτσι, εξασφαλίζεται διασυνδεσιμότητα μεταξύ ετερογενών συστημάτων, καθώς δε δεσμεύει τον τελικό χρήστη η υπάρξη ορισμένου τύπου παρόχων.

Ο πάροχος, μετά την εγκατάσταση του tomcat θα πρέπει να επικεφθεί τον ιστοχώρο http://127.0.0.1:8080 για να επιβεβαιώσει τη σωστή του εγκατάσταση.

Ο πάροχος επίσης μπορεί επιπλέον να ορίσει τη μεταβλητή περιβάλλοντος CATALINA_HOME που δείχνει τη διαδρομή στον δίσκο που είναι αποθηκευμένος ο εξυπηρετητής.

5.1.1.3 Δημιουργία Δημόσιου Κλειδιού

Όπως έχει τονιστεί, ένα από τα πιο σημαντικά χαρακτηριστικά των επιχειριματικών εφαρμογών είναι η ασφάλεια που εξασφαλίζουν στην επικοινωνία μεταξύ των εμπλεκομένων, των παρόχων και των καταναλωτών διαδικτυακών υπηρεσιών. Στο σύστημα που υλοποιήθηκε σε αυτή τη διπλωματική εργασία έγινε χρήση της τεχνολογίας δημοσίου κλειδιού (Public Key Infrastructure – PKI) για την κρυπτογράφηση των ανταλλασόμενων μηνυμάτων σε επίπεδο μεταφοράς (transport layer). Για να λειτουργήσει σωστά επομένως το σύστημα, πρέπει ο πάροχος να προμηθευτεί ένα ζευγάρι δημοσίου/ιδιωτικού κλειδιού, την υπογραφή του από μια έμπιστη Certification Authority (CA) και να ακολουθήση η κατάλληλη εγκατάτασή του. Σε περίπτωση που δεν είναι δυνατό να προμηθευτεί ένα αντίστοιχο, θα πρέπει να δημιουργήσει ένα. Ένα κατάλληλο εργαλείο για αυτό είναι το KeyTool GUI που παρέχεται δωρεάν από το GRIA στον ιστοχώρο http://www.gria.org/downloads#keytool-gui . Με χρήση αυτού του εργαλείου και ακολουθώνας τις οδηγίες χρήσης του είναι δυνατή η δημιουργία του επιθυμητού ζευγαριού. Σε περίπτωση που δεν είναι υπάρχει δυνατότητα κατάλληλης υπογραφής του από μια γνωστή και παγκοσμίως έμπιστη αρχή, μπορεί ο πάροχος να δημιουργήσει μια δικιά του με χρήση του εργαλείου XCA που παρέχεται δωρεάν στον ιστοχώρο http://sourceforge.net/projects/xca/ από τη SourceForge. Απαραίτητη είναι η εισαγωγή της συγκεκριμένης αρχής στο keystore που θα χρησιμοποιηθεί. Μετά τη δημιουργία του ζευγαριού, ο διαχειριστής της διαδικτυακής υπηρεσίας πρέπει να τροποποιήσει το αρχείο server.xml που βρίσκεται στον υποφάκελο conf του tomcat θέτοντας τις αντίστοιχες τιμές στις παραμέτρους που βρίσκονται κάτω από το node Connector και ορίζοντας ρητά τη διαδρομή στον τοπικό δίσκο του keystore που μόλις δημιουργήθηκε προς χρήση. Απαραίτητο είναι να διευκρινιστεί πως αν δε γίνει χρήση μιας γνωστής αρχής για την πιστοποίηση των κλειδιών του παρόχου, σε κάθε πρώτη σύνδεση του χρήστη-καταναλωτή της διαδικτυακής υπηρεσίας θα πρέπει να γίνει αποδοχή της συγκεκριμένης αρχής πιστοποίησης.

Εικόνα 5.1 – Τμήμα Αρχείου Ρυθμίσεων του Tomcat

5.1.1.4 Εγκατάσταση μεσισμικού GRIA

Για να καθιστεί δυνατό η χρήση όλων των λειτουργιών που παρέχει το μεσισμικό GRIA απαραίτητο είναι να γίνει εγκατάσταση των διαδικτυακών υπηρεσιών που προσφέρει. Ο πάροχος πρέπει να επισκευτεί την ιστοσελίδα http://www.gria.org/downloads/downloads-5.3 για να προμηθευτεί τις αντίστοιχες οντότητες λογισμικού. Έγινε χρήση των πακέτων Basic Application Services v5.3 και Service Provider Management v5.3. Από την κεντρική σελίδα διαχείρισης του tomcat http://localhost:8080/manager/html επιλέγεται από το κατάλληλο υπομενού το αντίστοιχο αρχείο war για εγκατάσταση. Στη συνέχεια για κάθε ένα από τα δύο αυτά πακέτα θα πρέπει να ακολουθηθεί η εξής διαδιασία:

1. Επιλέγεται το αντίστοιχο πακέτο κάνοντας κλικ πάνω στην εγγραφή του με τα προγράμματα που τρέχουν στον εξυπηρετητή.
2. Εισάγονται στο αρχείο tomcat-users.xml του υποφακέλου /conf του tomcat τα χαρακτηριστικά του κάθε υπεύθυνου διαχειριστή, δηλαδή ο ρόλος του, ο ψευδώνυμο (username) και ο κωδικός πρόσβασης (password)
3. Στη συνέχεια ο διαχειριστής οφείλει να επισκεφτεί τον κατάλογο admin προκειμένου να ρυθμίζει τις επιμέρους παράμετρους του κάθε πακέτου.
4. Στο keystore setup δηλώνεται το keystore που θα χρησιμοποιηθεί για την κρυπτογράφηση και την πιστοποίηση των μηνυμάτων της διαδικτυακής υπηρεσίας σε επίπεδο εφαρμογής [κρυπτογράφηση φακέλων, σύμφωνα με τα πρότυπα της υπηρεσιοστρεφούς αρχικτονικής] και την πιστοποίηση των εμπλεκόμενων χρηστών.

Συνίσταται το keystore να είναι το ίδιο με το keystore που χρησιμοποιεί ο tomcat για την κρυπτογράφηση των πακέτων σε επίπεδο μεταφοράς.

5. Εγκατάσταση κρυπτογραφημένης μεταφοράς. Εδώ ενεργοποιείται η δυνατότητα κρυπτογραφημένης μεταφοράς μηνυμάτων σε επίπεδο μεταφοράς, με χρήση https πρωτοκόλλου.
6. Δηλώνεται η διαδρομή στο δίσκο που θα αποθηκεύονται πληροφορίες σχετικά με τη χρήση των υπηρεσιών με χρήση τοπικής βάσης δεδομένων.
7. Γίνεται ρύθνιση του endpoint reference των διαδικτυακών υπηρεσιών, που σχετίζονται με το keystore και τη διαδικτυακή ονοματολογία του εξυπηρετητή, όπως αυτή ορίζεται και στα προς χρήση κλειδιά.

Οι παραπάνω ρυθμίσεις πρέπει να γίνουν και στα δύο πακέτα λογισμικού. Από εκεί και πέρα, απαιτείται ξεχωριστή ρύθμιση των ιδιοτήτων της κάθε οντότητας, ανάλογα με τις διαδικτυακές υπηρεσίες που θα καλεστεί να καταγράφει. Αυτό είναι τμήμα της εγκατάστασης της εφαρμογής. Η παραπάνω διαδικασία αναφέρεται αναλυτικά και κατά την εγκατάσταση των πακέτων, αλλά και στο επίσημο ιστοχώρο του GRIA.

5.1.2 Εγκατάσταση Νέου Συστήματος

Σε αυτή την ενότητα θα γίνει περιγραφή της διαδικασίας εγκατάστασης του συστήματος, τόσο σε ό,τι αφορά τον daemon και την πλευρά του τελικού χρήστη, όσο και την πλευρά της υπηρεσίας και του εξυπηρετητή αδειών.

5.1.2.1 Εγκατάσταση Daemon

Η ονότητα λογισμικού που αποτελεί τον daemon βρίσκεται στο αρχείο GRIALicenseDeamon.zip. Το συγκεκριμένο αρχείο περιλαμβάνει τα start.bat, GRIALicenseDeamon.jar, και config.sys. Ο διαχειριστής πρέπει αν εξάγει τα συγκεκριμένα αρχεία σε οποιοδήποτε φάκελο. Το αρχείο config.sys περιλαμβάνει τις ρυμίσεις του daemon. Στην πρώτη γραμμή αναγράφεται το URI της διαδικτυακής υπηρεσίας που πρόκειται να καταναλωθεί για την παροχή της άδειας λογισμικού, ανάλογα με τις απαιτήσεις του παρόχου. Στη δεύτερη γραμμή αναγράφεται η διαδρομή στο δίσκο που βρίσκεται αποθηκευμένο το δημόσιο κλειδί του χρήστη. Ένα παράδειγμα ενός τέτοιου αρχείου είναι το ακόλουθο:

```
https://147.102.19.107:8443/devkit/services/GRIALicenseService
C:\keystores\home\service-keystore.ks
```

Απαραίτητο είναι το αρχείο config.sys να βρίσκεται στον ίδιο φάκελο με το GRIALicenseDeamon.jar. Αν ο διαχειριστής του daemon επιθυμεί, μπορεί να τοποθετήσει τα bat αρχείου σε οποιαδήποτε άλλη διαδρομή επιθυμεί, και με κατάλληλη τροποποίησή του να δείξει το GRIALicenseDeamon.jar αρχείο προς εκτέλεση.

Επιπλέον, ανάλογα με την εφαρμογή, θα πρέπει ο διαχειριστής να εντοπίσει το αντίστοιχο license file, και να τροποποιήσει αντίστοιχα τη γραμμή που αναγράφεται στον FLEXlm server, δηλώνοντας την IP διεύθυνση του Daemon, ανάλογα με το τερματικό που έχει γίνει εγκατάσταση.

5.1.2.2 Εγκατάσταση του GRIA License Service

Η διαδικτυακή υπηρεσία που υλοποιήθηκε στο συγκεκριμένο σύστημα είναι πακεταρισμένη σε ένα war αρχείο. Ο διαχειριστής του παρόχου της υπηρεσίας πρέπει να συνδεθεί στην κεντρική σελίδα διαχείρισης του εξυπηρετητή tomcat http://localhost:8080/manager/html και από το αντίστοιχο υπομενού να ανεβάσει το συγκεκριμένο war αρχείο.

Μετά την εγκατάσταση του αρχείου, απαιτούνται επιπρόσθετες ενέργειες για την πλήρη ρύθμισή του. Ο διαχειριστής πρέπει να ακολουθήσει τα βήματα που περιγράφηκαν στην υποενότητα 5.1.1.4, και αφορούν τη δήλωση του keystore που θα χρησιμοποιηθεί, την ενεργοποίηση της κρυπογραφημένης μεταφοράς πακέτων, τον ορισμό της διαδρομής στο δίσκο που θα χρησιμοποιηθεί για την τοπική βάση δεδομένων και τέλος τον καθορισμό του endpoint reference της διαδικτυακής υπηρεσίας. Σε αυτό το σημείο θα πρέπει να τονιστεί πως είναι επιθυμητό τα keystore της GRIA License Service και των υπηρεσιών του μεσισμικού GRIA να είναι ταυτόσημα ή ιδανικά, όλες οι υπηρεσίες να είναι εγκατεστημένες στον ίδιο δικυτακό κόμβο/ Κατά αυτόν τον τρόπο εξοικονομείται χρόνος για την κατάλληλη ρύθμισή τους καθιστόντας ευκολότητερη την ολοκλήρωση της διαδικτυακής υπηρεσίας με το GRIA. Αφού γίνει η απαιτούμενη εγκατάσταση, όπως αυτή περιγράφηκε, ο διαχειριστής θα πρέπει να πάει στη σελίδα Access Control για να ενημερώσει τον τρόπο με τον οποίο τα PBAC 2 controls θα έχουν τη δυνατότητα για προσπέλαση των πόρων. Αρκεί να κατεβάσει το xml αρχείο LicenseResourceType.xml με τις πολιτικές (policies) που ορίζουν τις προσβάσεις και να προσθέσει τις αντίστοιχες εγγραφές που θα επιτρέψουν την προσπέλαση των συσχετιζόμενων μεθόδων (transferMessage, getServerURL) από τον καταναλωτή της διαδικτυακής υπηρεσίας.

```
<!-- Generic operations available for most resource types -->
<operation name="getEPR"/>
<operation name="getServiceMatchPattern"/>
<operation name="setLabel"/>
<operation name="destroy"/>

<operation name="transferMessage">
    <process-role name="owner"/>
</operation>
<operation name="getServerURL">
    <process-role name="owner"/>
</operation>

<process-role name="owner"/>
<state name='ready'/>
```

Εικόνα 5.2 – Τμήμα Αρχείου LicenseResourceType.xml

Μετά την επεξεργασία του αρχείου, ο διαχειριστής θα πρέπει να το ανεβάσει στο αντίστοιχο σημείο.

Τέλος, απαραίτητη είναι η ύπαρξη ενός αρχείου ρυθμίσεων με όνομα LicenseService.config στον υποφάκελο /conf του εξυπηρετητή Tomcat. Στο αρχείο αυτό γράφεται στην πρώτη γραμμή η διεύθυνση IP του FLEXlm server που θα αναλάβει να παρέχει την άδεια εκτέλεσης του αιτούμενου λογισμικού.

5.2 Εκτέλεση

Σε αυτή την υποενότητα θα περιγράψουμε τη διαδικασία εκτέλεσης του συστήματος διαχείρησης αδειών χρήσης λογισμικού που προτείνεται σε αυτή τη διπλωματική εργασία. Για την επαλήθευση της ορθότητας του συστήματος θα γίνει χρήση της εμπορικής εφαρμογής Mathematica και του FLEXlm server ως εξυπηρετητή αδειών λογισμικού. Τα δύο αυτά προγράμματα είναι ρυθμισμένα ώστε να λειτουργούνε κανονικά σε περιβάλλον τοπικού δικτύου, επικοινωνόντας σε μια προκαθορισμένη θύρα, με βάση τις αρχές και το πρωτόκολλο που περιγράφηκαν σε προηγούμενα κεφάλαια. Η διασυνεδσιμότητά τους και η ορθή λειτουργία του σεναρίου αυτού είναι αποδεδειγμένη και ευρέως διαδεδομένη στον

επιχειρηματικό κόσμο, γι΄αυτό και δε θα εξετάσουμε την ορθότητα της λειτουργίας του, και θα τη θεωρήσουμε σα δεδομένη.

Σε αυτό το κεφάλαιο θα επικεντρωθούμε στην ορθή λειτουργία του συστήματος που υλοποιήθηκε, λαμβάνοντας υπ' όψιν δύο σενάρια. Το πρώτο και πιο απλό σενάριο αφορά την απλή εκτέλεσή του με ασφαλή τρόπο σε ένα δίκτυο, χωρίς την καταγραφή της χρήσης της υπηρεσίας από τον πάροχο. Στο δεύτερο σενάριο θα επικεντρωθούμε στο αν παρέχεται η δυνατότητα παράλληλης καταγραφής της χρήσης της διαδικτυακής υπηρεσίας. Δεν ενδιαφέρει τη διπλωματική εργασία να μελετήσει μια αποδοτική και συμφέρουσα συμφωνία χρήσης, ούτε να ελέγξει το αν η τελευταία τηρείται από τον πάροχο. Αρκεί να αποδειχθεί ότι είναι δυνατή η καταγραφή της χρήσης. Πριν γίνει η παράθεση του ελέγχου των δύο σεναρίων, απαραίτητη είναι η περιγραφή του περιβάλλοντος εκτέλεσης.

5.2.1 Ρυθμίσεις Ελέγχου Εκτέλεσης

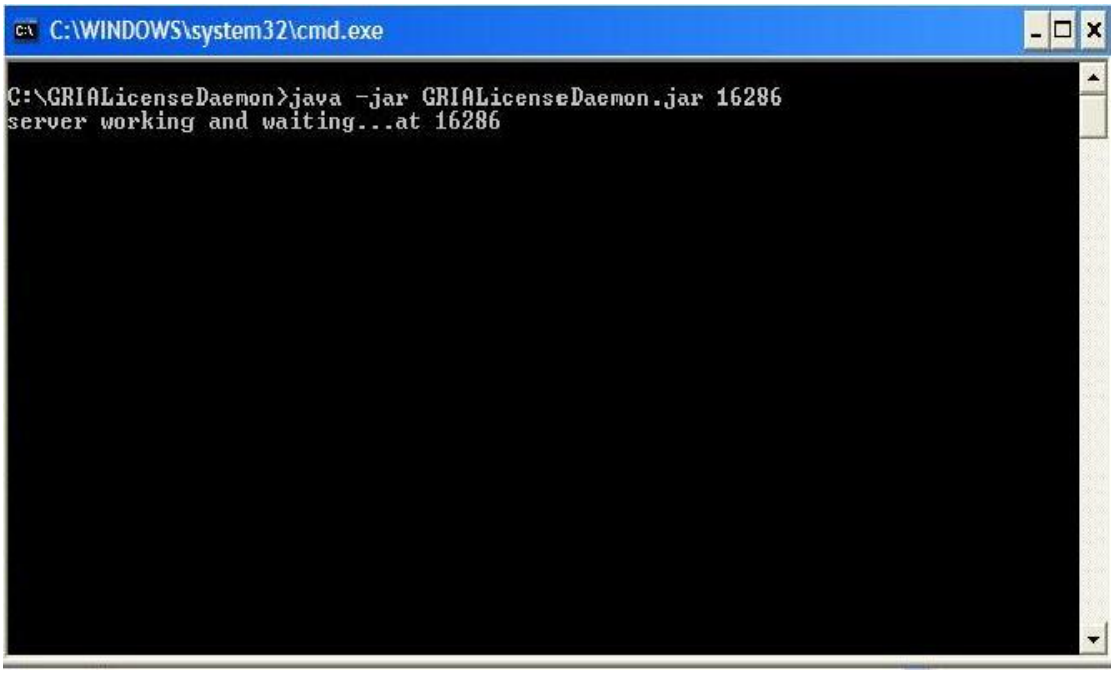
Για την εκτέλεση των παραπάνω σεναρίων έγινε χρήση ενός FLEXlm server ρυθμισμένου στο να παρέχει άδειες χρήσης της εμπορικής εφαρμογής Mathematica, ο οποίος βρίσκεται στην ηλεκτρονική διεύθυνση IP 147.102.246.1. Η διαδικτυακή υπηρεσία του παρόχου φιλοξενήθηκε σε έναν εξυπηρετητή Tomcat στη διεύθυνση 147.102.19.107. Αντίστοχα ο daemon θα εκτελεστεί από τον κόμβο 147.102.19.106. Για απλούστευση του ελέγχου, χωρίς να μειώσουμε τη γενικότητα της υλοποίησης, η εμπορική εφαρμογή Mathematica θα εκκινήσει από τον ίδιο κόμβο με τον daemon. Άλλωστε, όπως έχει αναφερθεί, είναι επιθυμητό το σενάριο να υπάρχει ένας daemon εγκατεστημένος σε κάθε μηχάνημα που πρόκειται να εκτελέσει την εφαρμογή, για να υπάρχει λιγότερη κίνηση στο δίκτυο. Επιπλέον, μέσω της κονσόλας διαχείρισης των Microsoft Windows [mmc] εφαρμόσαμε πολιτική μπλοκαρίσματος όλης της TCP κίνησης από και προς τον κόμβο 147.102.19.106 [κόμβος εκτέλεσης του daemon και της εφαρμογής] εκτός από τα πακέτα που έχουν σαν πηγή/προορισμό τις γνωστές πόρτες (80,8080,8443 κτλ). Κατά αυτό τον τρόπο εφασφαλίζεται ότι όλη κίνηση που μπορεί να δημιουργηθεί κατά την εκκίνηση του Mathematica θα μπορεί να περάσει στο δίκτυο μόνο διαμέσου του daemon, ο οποίος και θα στείλει πακέτα δεδομένων στις γνωστές γνωστές παγκόσμιες θύρες που περιμένουν αιτήσεις οι διαδικτυακές υπηρεσίες.

5.2.2 Εκτέλεση Συστήματος Χωρίς την Καταγραφή Χρήσης της Υπηρεσίας

Για την εκτέλεση του συγκεκριμένου σεναρίου, απαραίτητη είναι η ρύθμιση της διαδικτυακής υπηρεσίας ώστε να αποδέχεται όλες τις αιτήσεις, χωρίς την καταγραφή της χρήσης τους. Ο διαχειριστής του παρόχου θα πρέπει να επισκεφτεί τη σελίδα διαχείρισης της υπηρεσίας και να τη θέσει ως free.

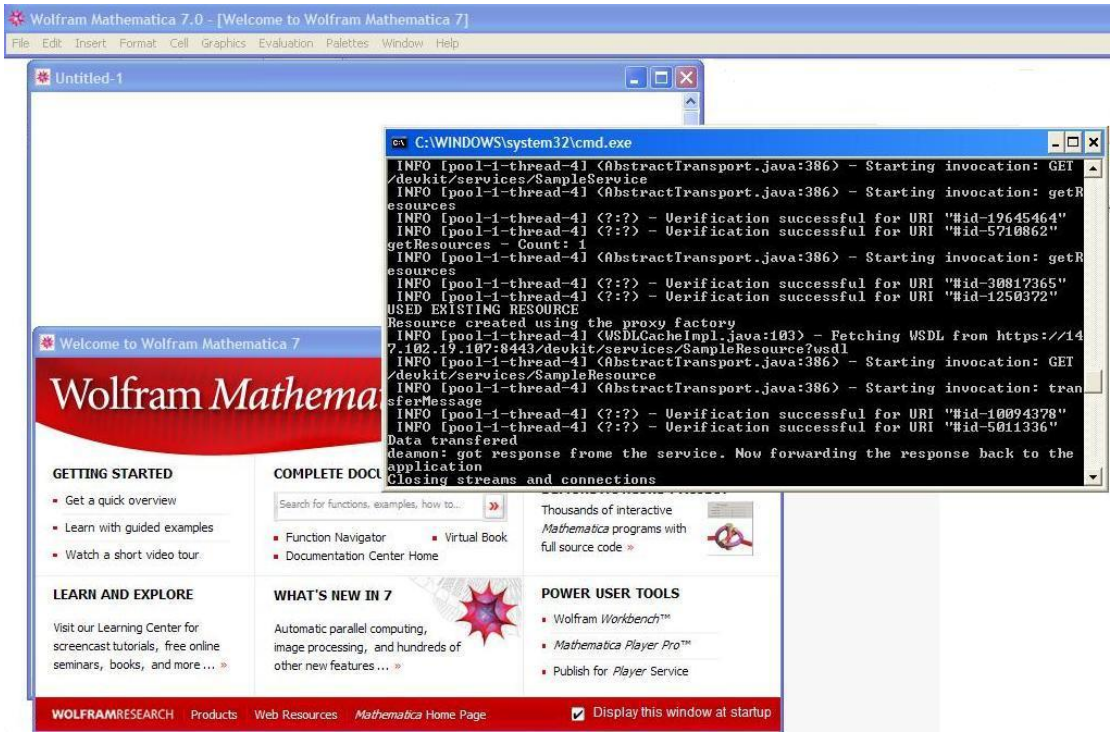
Εικόνα 5.3 – Δωρεάν Παροχή της Διαδικτυακής Υπηρεσίας

Αντίστοιχα ο διαχειριστής από την πλευρά του καταναλωτή εκκινεί τον daemon, ο οποίος και αρχικοποιείται και περιμένει αιτήσεις στην προκαθορισμένη θύρα του, από οποιονδήποτε χρήστη στο αντίστοιχο τοπικό του δίκτυο επιθυμεί να εκκινήσει την εφαρμογή του.



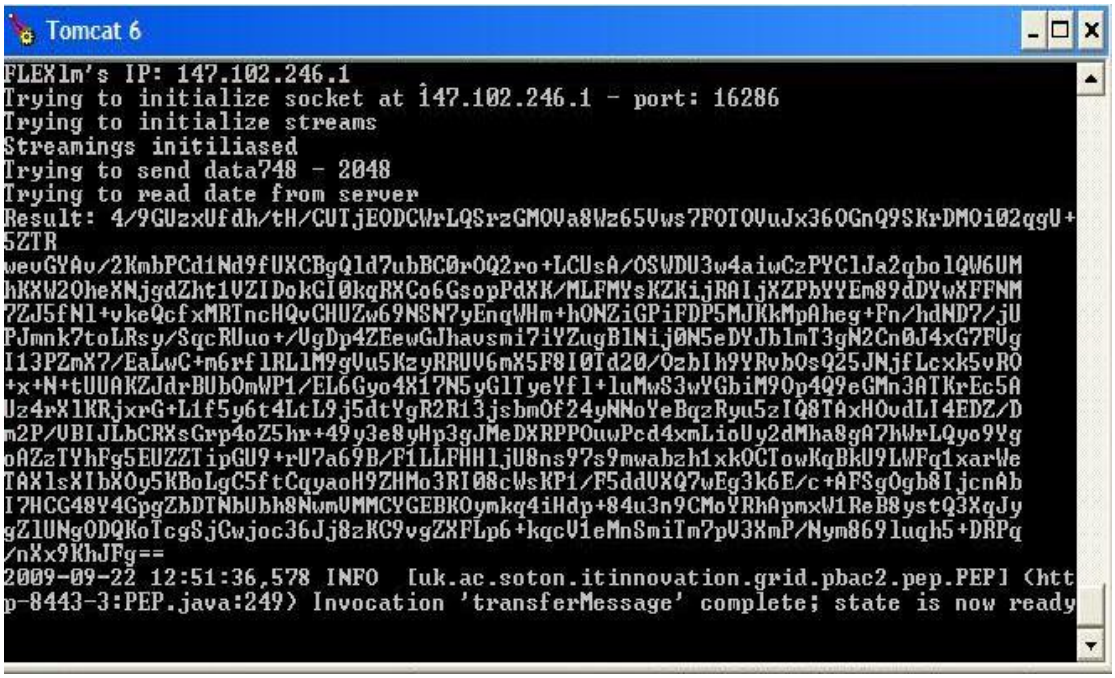
Εικόνα 5.4 – Εκκίνηση Daemon

Ο χρήστης της εφαρμογής εκκινεί με τη σειρά του το Mathematica, και αφού παρατηρηθούν οι ανταλλαγές των πακέτων μεταξύ του daemon και του GRIA License Service, το Mathematica ξεκινάει.



Εικόνα 5.5 – Εκκίνηση Mathematica

Ενώ αντίστοιχα στην κονσόλα του Tomcat Server παρατηρούμε επίσης την αλληλουχία των μηνυμάτων που ανταλλάσονται μεταξύ του daemon και του FLEXlm server.



Εικόνα 5.6 – Κονσόλα Tomcat Server

Παράλληλα, παρατηρούμε τη δέσμευση των πόρων από το μεσισμικό GRIA για την εκτέλεση της διαδικτυακής υπηρεσίας με βάση το περιεχόμενο (context) του καταναλωτή. Από την κονσόλα διαχείρισης της υπηρεσίας παρατηρούμε τον δεσμευμένο πόρο με όνομα ntua107!, το όνομα δηλαδή που δώσαμε στον καταναλωτή της.

Εικόνα 5.7 – Δεσμευμένοι Πόροι στο GRIA

Συνεχίζουμε την εκτέλεση της αφαρμογής και δοκιμάζουμε διάφορες περιπτώσεις. Εκκινούνται δύο ή περισσότερα στιγμιότυπά της ταυτόχρονα, τερματίζονται κατά τυχαία σειρά, επανεκκινούνται πάλι, και σε κάθε περίπτωση παρατηρούμε την εφαρμογή να λειτουργεί και να τερματίζει κανονικά, τον daemon να μην κολλάει, αλλά λόγο των παράλληλων διεργασιών που χρησιμοποιεί κατά αίτηση να τις εξυπηρετεί κανονικά, ενώ επίσης η διαδικτυακή υπηρεσία να μην εμφανίζει σφάματα και να μην τερματίζει, αλλά να συνεχίζει να εξυπηρετεί κάθε αίτηση του καταναλωτή και να πρωθεί και να επιστρέφει τα μηνύματα προς και από τον FLEXlm server. Ακόμα, επειδή ο χρήστης που χρησιμοποιεί την εφαρμογή είναι κάθε φορά ο ίδιος, το μεσισμικό GRIA τον αναγνωρίζει και δε δεσμεύει νέους πόρους κάθε φορά που προσπελάει την υπηρεσία. Έτσι μία φορά δεσμεύονται οι πόροι και παραμένουν δεσμευμένοι για όλη τη διάρκεια ζωής της υπηρεσίας, ή μέχρι να αποφασίσει ο διαχειριστής να τους αποδεσμεύσει. Κατά αυτό τον τρόπο δίνεται η δυνατότητα να κρατιούνται πληροφορίες για την χρήση της υπηρεσίας, αφού δεν καταστρέφονται σε κάθε τερματισμό λειτουργίας της μεθόδου που προσπελαύνει ο χρήστης.

Βλέπουμε λοιπόν ότι επαληθεύεται η αρχική υπόθεση, ότι είναι δυνατή η υλοποίηση ενός συστήματος παροχής αδειών χρήσης λογισμικού, το οποίο θα επεκτείνει το υπάρχον σύγχρονο δικτυακό μοντέλο και θα καταρρίπτοντας πολλούς από τους περιορισμούς που το τελευταίο θέτει στους χρήστες λόγω της αρχιτεκτονικής του. Το προτεινόμενο σύστημα λειτουργεί άριστα σε ξεχωριστά δίκτυα, επιτρέποντας τη διασυνδεσιμότητα ετερογενών δικτύων και διαφορετικών υπολογιστικών συστημάτων, ενώ ταυτόχρονα η χρήση της τεχνολογίας δημοσίου κλειδιού (Public Key Infrastructure – PKI) εξασφαλίζει την απαιτούμενη ασφάλεια των επικοινωνιών.

5.2.3 Εκτέλεση Συστήματος Με Καταγραφή Χρήσης της Υπηρεσίας

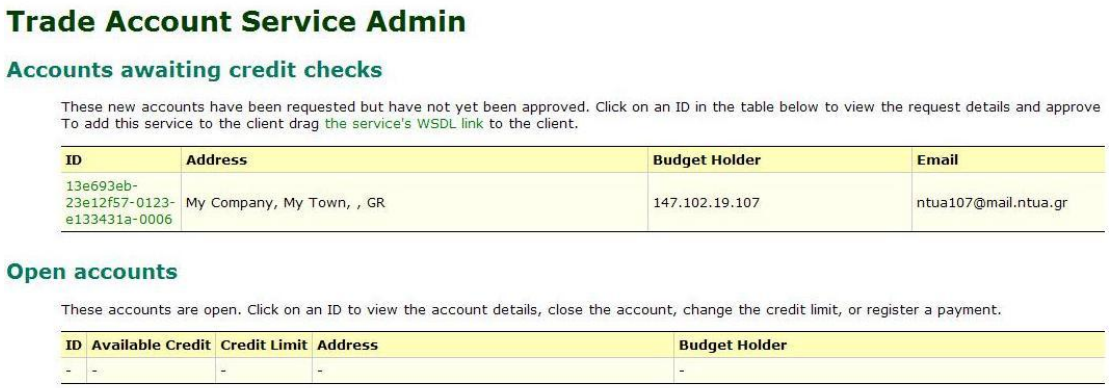
Για την εκτέλεση του συγκεκριμένου σεναρίου, απαραίτητη είναι η ρύθμιση της διαδικτυακής υπηρεσίας ώστε να αποδέχεται μόνο αιτήσεις οι οποίες συμφωνούν με κάποια προαπαιτούμενα, ώστε να είναι ικανή η καταγραφή της χρήσης τους. Το πακέτο service provider του GRIA παρέχει τη δυνατότητα εποπτεία της χρήσης μέσω συμφωνιών επιπέδου υπηρεσιών (SLAs), την καταγραφή τους σε κάποιον λογαριασμό συναλλαγής, ή τον συνδιασμό και των δύο. Στο παρόν σενάριο δεν ενδιαφέρει η εκ βαθέως ανάλυση της διαδικασίας αυτής, αλλά μόνο η απόδειξη ότι το υπάρχον σύστημα μπορεί να υλοποιήσει

κάποιον από τους παραπάνω μηχανισμούς, οι οποίοι όπως έχει αναφερθεί εξασφαλίζουν την καταγραφή της χρήσης της υπηρεσίας, με σκοπό την επακόλουθη κοστολόγισή τους και αποστολή αντίστοιχου τιμολογίου στον καταναλωτή. Επομένως, απαραίτητη είναι η κατάλληλη ρύθμιση του GRIA License Service από τον διαχειριστή του παρόχου. Ο τελευταίος πρέπει να επισκευτεί τη σελίδα της κονσόλας διαχείρισης της υπηρεσίας και να να ορίσει την αντίστοιχη υπηρεσία που θα τη διαχειρίζεται. Για τις ανάγκες του σεναρίου επιλέχθηκε η Trade Account Service.

Εικόνα 5.8 – Σελίδα Διαχείρισης της Διαδικτυακής Υπηρεσίας

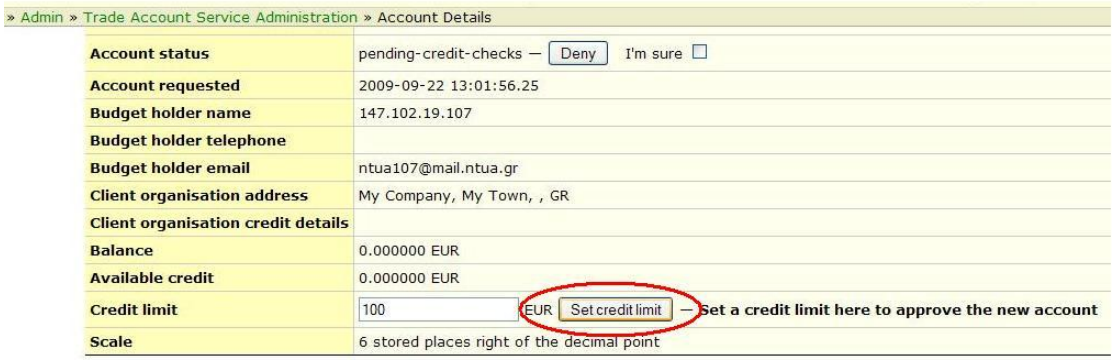
Στη συνέχεια ο καταναλωτής κάνει αίτηση για δημιουργία ενός νέου λογαριασμού συναλλαγής, ο οποίος και θα καταγράφει τη χρήση της υπηρεσίας. Αυτό μπορεί να γίνει αυτοματοποιημένα, με εκτέλεση του αντίστοιχου bat αρχείου που αναφέρθηκε σε προηγούμενο κεφάλαιο. Εδώ αξίζει να σημειωθεί ότι ο σκοπός της διπλωματικής δεν είναι να περιγράψει ένα σύστημα διαχείρισης της χρήσης διαδικτυακών υπηρεσιών, γι' αυτό και δεν ενσωματώθηκε στο προτεινόμενο σύστημα. Η ύπαρξή του όμως είναι απαραίτητη για τη δημιουργία ενός λογαριασμού, και αυτός είναι ο λόγος που παραθέτεται στο παράρτημα.

Μετά την εκτέλεση του συγκεκριμένου bat, δημιουργείται ένας νέος trade account του χρήστη, ο οποίος σύμφωνα με τις προδιαγραφές του GRIA βρίσκεται σε κατάσταση "awaiting credit checks".



Εικόνα 5.9 – Λογαριασμός Συναλλαγής: awaiting credid checks

Ο διαχειριστής του παρόχου, οφείλε σε αυτό το σημείο να κάνει τους απαραίτητους ελέγχους για την πιστοποίηση της πιστολητικής ικανότητας του καταναλωτή και της φερεγγυότητάς του, εφόσον αναφερόμαστε σε περιβάλλον διεπιχειρησιακών σχέσεων, και αφού έρθει σε επαφή μαζί του και συμφωνήσουν στο πιστωτικό όριο της χρήσης της διαδικτυακής υπηρεσίας, ανοίγει το λογαριασμό.



Εικόνα 5.10 – Άνοιγμα Λογαριασμού Συναλλαγής

Ο λογαριασμός πλέον μεταβαίνει σε κατάσταση open.

Open accounts

These accounts are open. Click on an ID to view the account details, close the account, change the credit limit, or register a payment.

ID	Available Credit	Credit Limit	Address	Budget Holder
13e693eb-23e12f57-0123-e133431a-0006	100.000000	100 EUR	My Company, My Town, , GR	147.102.19.107

Accounts being closed

These accounts are in the process of being closed. No new allocations may be made, but payment may be still outstanding. They need to be moved to payment is settled.

ID	Balance	Credit Limit	Address	Budget Holder
-	-	-	-	-

Εικόνα 5.11 – Λογαριασμός Συναλλαγής: open

Πλέον, είμαστε σε θέση να εκκινήσουμε εκ νέου τον daemon. Αφού ακολουθήσουν οι οθόνες που περιγράφηκαν στην προηγούμενη υποενότητα, ο χρήστης εκκινεί την εφαρμογή Mathematica, η οποία και επικοινωνεί με τον daemon. Ο τελευταίος καλεί την υπηρεσία GRIA License Service, η οποία προτού δεσμεύσει τους απαιτούμενους πόρους κάνει έλεγχο για να διαπιστώσει αν υπάρχει ανοιχτός λογαριασμός που να ανήκει στο χρήστη ο οποίος έκανε την αίτηση για προσπέλαση μιας από τις μεθόδους του. Ο λογαριασμός έχει ρυθμιστεί από τον διαχειριστή του παρόχου στην κατάσταση ‘open’, και η υπηρεσία δεσμεύει κανονικά τους απαιτούμενου πόρους και προχωράει στην εκτέλεση της αιτούμενης μεθόδου της. Κατά τη δέσμευση των πόρων χρεώνεται μία μονάδα ο λογαριασμός. Εξ ορισμού η αρχικοποίηση των πόρων για την εξυπηρέτηση του χρήστη κοστίζει μία μονάδα. Με κατάλληλη προσθήκη κώδικα, σύμφωνα με την επίσημη σελίδα του GRIA, μπορεί η διαδικτυακή υπηρεσία να τροποποιηθεί και να χρεώνει μία μονάδα κάθε κλήση της. Κατά αυτόν τον τρόπο, σε κάθε πακέτο που μεταφέρεται διαμέσου της υπηρεσίας στο FLEXlm server, ο καταναλωτής χρεώνεται 1 μονάδα. Αναλογιζόμενοι ότι σύμφωνα με το πρωτόκολλο λειτουργίας του FLEXlm server, υπάρχει μια ανά χρονικά διαστήματα επικοινωνία μεταξύ της εφαρμογής και του FLEXlm server ώστε να διαπιστωθεί κατά πόσο η εφαρμογή είναι σε λειτουργία και δεν έχει τερματίσει, συμπεραίνουμε ότι όσο μεγαλύτερη είναι η χρήση της, τόσα περισσότερα πακέτα θα μετακινούνται στο διαδίκτυο, άρα τόσες περισσότερες φορές θα γίνει προσπέλαση της υπηρεσίας, με αποτέλεσμα να αυξάνεται η χρέωση του χρήστη ανά χρήση. Βλέπουμε λοιπόν ότι μετά από διαδοχικές εκκινήσεις του Mathematica και λειτουργίας του για κάποιο χρονικό διάστημα, έχουν κοστολογηθεί στο χρήστη αντίστοιχες μονάδες.

Open accounts

These accounts are open. Click on an ID to view the account details, close the account, change the credit limit, or register a payment.

ID	Available Credit	Credit Limit	Address	Budget Holder
13e693eb-23e12f57-0123-e133431a-0006	100.000000	78 EUR	My Company, My Town, , GR	147.102.19.107

Accounts being closed

These accounts are in the process of being closed. No new allocations may be made, but payment may be still outstanding. They need to be moved to payment is settled.

ID	Balance	Credit Limit	Address	Budget Holder
-	-	-	-	-

Εικόνα 5.12 – Χρεωμένος Λογαριασμός Συναλλαγής

Παρατηρούμε δηλαδή πως η προτεινόμενη αρχιτεκτονική πράγματι, εξαφαλίζει δυνατότητες χρέωσης της χρήσης της διαδικτυακή υπηρεσίας από τον καταναλωτή. Έτσι, εκτός των δυνατοτήτων που αποδείχθηκαν στο προηγούμενο σενάριο και αφορούν την παροχή αδειών χρήσης λογισμικού από συμβαλλόμενα μέρη που βρίσκονται σε γεωγραφικά απομακρυσμένες περιοχές, εξασφαλίζοντας ασφαλή κατά τα επιχειρηματικά πρότυπα επικοινωνία, το σύστημα παρέχει επιπλέον τη δυνατότητα καλύτερης τιμολόγησης ενός εμπορικού λογισμικού, βασιζόμενη στη καθέ αυτή χρήση του, και όχι στα υποκείμενα που έχουν τη δυνατότητα εκτέλεσής του.

6

Επίλογος

Στο συγκεκριμένο κεφάλαιο θα γίνει αρχικά μια ανακεφαλαίωση των αποτελεσμάτων της διπλωματικής εργασίας και μια παρουσίαση των συμπερασμάτων που εξήχθησαν κατά τη διάρκεια εκπόνησής της και με την ολοκλήρωσή της. Στη συνέχεια θα γίνει αναφορά σε μελλοντικές επεκτάσεις που η παρούσα εργασία και το σύστημα, που αναπτύχθηκε στα πλαίσια αυτής, μπορούν να έχουν.

6.1 Σύνοψη και συμπεράσματα

6.1.1 Σύνοψη

Η παρούσα διπλωματική εργασία παρουσίασε εκτενώς και με σαφήνεια ένα νέο μοντέλο παροχής αδειών χρήσης λογισμικού, βασισμένο στο μοντέλο παροχής με σύγχρονο τρόπο που είναι ευρέως διαδεδομένο σήμερα, αλλά με χρήση διαδικτύου παρέχοντας τη δυνατότητα για συνδεσιμότητα μεταξύ ετερογενών δικτύων με ασφαλή τρόπο, με βάση τα σύγχρονα επιχειρηματικά πρότυπα. Επιπλέον, με χρήση τεχνολογιών πολυπλέγματος κατέστει δυνατή η καταγραφή της χρήσης των υπηρεσιών από τον τελικό χρήστη, δίνοντας τη δυνατότητα για νέες πολιτικές κοστολόγησης των εφαρμογών που χρησιμοποιούνε οι χρήστες ενός οργανισμού, πράγμα που οδηγεί σε μεγαλύτερη ευελιξία τους τελευταίους στην επιλογή του

απαιτούμενου λογισμικού, αλλά και σε μεγαλύτερο ανταγωνισμό από τους παρόχους και δημιουργούς των αντίστοιχων εφαρμογων.

Για την αποτελεσματική λειτουργία και διεκπεραίωση των παραπάνω στόχων χρησιμοποιήθηκαν τεχνολογίες αιχμής, όπως η υιοθέτηση του υπηρεσιοστρεφούς μοντέλου ανάπτυξης διαδικτυακών εφαρμογών και των τεχνολογιών πολυπλέγματος. Κάθε τεχνολογία, πλατφόρμα ανάπτυξης, γλώσσα προγραμματισμού και περιβάλλον εκτέλεσης επιλέχθηκε προσεκτικά και με σύνεση ώστε να πληρεί τις προυποθέσεις και τις ανάγκες του σχεδιασμού και των στόχων που έπρεπε να επιτελεί το νέο μοντέλο παροχής αδειών χρήσης λογισμικού, έτσι όπως ορίστηκαν σε προηγούμενα κεφάλαια. Για γλώσσα προγραμματισμού επιλέχθηκε η Java, ως πλατφόρμα ανάπτυξης της εφαρμογής ο eclipse σε συνδιασμό με το apache maven project, έγινε χρήση του μεσισμικού GRIA για να παρέχει τις δυνατότητες του πολυπλέγματος, η ασφάλεια στις επικοινωνίες έγινε με χρήση τεχνολογιών δημοσίου κλειδιού (PKI) [32] ενώ ως εξυπηρετητής που φιλοξενεί τις διαδικτυακές υπηρεσίες επιλέχθηκε ο Apache Tomcat.

Μετά την ολοκλήρωση της υλοποίησης του προτεινόμενου συστήματος έγινε η εγκατάστασή του για τον έλεγχο της ορθής λειτουργίας του και την ικανοποίηση των προδιαγεγραμμένων απαιτήσεων. Σαν εφαρμογή δοκιμής επιλέχθηκε το εμπορικό λογισμικό Mathematica, ενώ σαν εξυπηρετητής παροχών αδειών χρήσης του ο FLEXlm server. Στην αρχή έγινε δοκιμή και έλεγχος για το αν επικοινωνούνε όλες οι οντότητες λογισμικού. Αφού διαπιστώθηκε η σωστή και αδιάκοπη επικοινωνία μεταξύ τους, η εφαρμογή-πελάτης εκκίνησε και μετά από κάποιο χρονικό διάστημα τερμάτισε κανονικά. Στη συνέχεια εγίναν δοκιμές προσπαθώντας να εκκινήσουμε το mathematica παολλαπλές φορές, έγινε έλεγχος ότι για οποιαδήποτε εκκίνηση και τυχαίο τερματισμό και ο daemon αλλά και το GRIA License Service ανταποκρίνονται κανονικά. Τέλος, από τη φόρμα διαχείρισης της οντότητας service provider του GRIA έγινε η διαπίστωση ότι οι πόροι που δεσμεύονται από την εφαρμογή πελάτη καταγράφονται κανονικά, και πιστοποιούν τον κάθε χρήστη μέσω του ειδικού συμφωνημένου ονόματος που έχει πιστοποιηθεί από μια ανεξάρτητη και έμπιστη από όλους αρχή πιστοποίησης. Έτσι με χρήση και ορισμό συγκεκριμένων SLAs στα πρότυπα του GRIA είναι δυνατή η τιμολόγηση με βάση την ποιότητα της χρήσης της υπηρεσίας και της εφαρμογής.

6.1.2 Συμπεράσματα

Η ανάγκη των εταιριών πώλησης και διανομής λογισμικού για την εξασφάλιση της γνησιότητας των αντιγράφων λογισμικού που διαθέτουν οι πελάτες τους, οδήγησε στη δημιουργία των ηλεκτρονικών αδειών χρήσης του αντίστοιχου λογισμικού. Με την ανάπτυξη της τεχνολογίας και των δυνατοτήτων που αυτή παρείχε, οι εταιρίες αφεύραν νέους τρόπους διανομής των αδειών τους, ενώ παράλληλα η ανάπτυξη των επιχειρήσεων και των οργανισμών απαιτούσαν πιο αποδοτικούς τρόπους παροχής τους. Ο πιο διαδεδομένος σήμερα τρόπος προυποθέτει την ύπαρξη ενός εξυπηρετητή αδειών και εφαρμογών-πελατών στο ίδιο τοπικό δίκτυο. Παρόλα αυτά, η ευρής ανάπτυξη και χρήση του διαδικτύου δημιούργησε πλέον την ανάγκη για ασφαλή διασυνδεσιμότητα μεταξύ ετερογενών δικτύων και τη σπουδαιότητα της υλοποίησης ενός συστήματος παροχής αδειών με το οποίο θα εξασφαλίζεται η μεγαλύτερη δυνατή αξιοπιστία και ευελιξία της διαδικασίας παροχής της, θα επιτρέπεται ο έλεγχος της επικοινωνίας μεταξύ πελάτη και εξυπηρετητή, θα παρέχεται η μέγιστη δυνατή ασφάλεια στη μεταφορά των δεδομένων και θα καταγράφεται η χρήση της του συστήματος.

Ο μηχανισμός αυτός, ο οποίος περιγράφηκε στο παρόν έγγραφο, υλοποιήθηκε με σκοπό να πληρεί όλες αυτές τις προυποθέσεις. Μεταφέροντας την επικοινωνία απο το επίπεδο μεταφοράς στο επίπεδο εφαρμογής κατέστει δυνατή η διασυνδεσιμότητα μεταξύ δύο διαφορετικών τοπικών δικτύων, επιτρέποντας τη χρήση του από γεωγραφικά διασκορπισμένους οργανισμούς. Η χρήση του επιπέδου εφαρμογής επιπρόσθετα απαιτεί την ύπαρξη οντότητων λογισμικού που θα αναλάβουν τη μεταξύ τους επικοινωνία. Αυτό προσθέτει επιχειρησιακή λογική στις δύο επικοινωνούντες συνιστώσες του συστήματος, άρα και καλύτερο έλεγχο στην κίνηση που δημιουργείται και δυνατότητες εφαρμογής συγκεκριμένων πολιτικών, ανάλογα με την εταιρία. Με τη τεχνολογία δημοσίου κλειδιού εξασφαλίστηκε η μέγιστη δυνατή ασφάλεια και προστασία των μεταφερόμενων δεδομένων με βάση τα επιχειρηματικά πρότυπα, ενώ ως τώρα αυτό ήταν ευθύνη του διαχειριστή δικτύου. Η υιοθέτηση για την υλοποίηση του συστήματος εργαλείων όπως η γλώσσα προγραμματισμού Java, ο Tomcat Server, και το υπηρεσιοστρεφές μοντέλο εφαρμογών διαδικτύου με τα πρότυπα που ορίζει εξασφάλισαν τη διασυνδεσιμότητά του μεταξύ ετερογενών δικτύων και διαφορετικών υπολογιστικών συστημάτων. Αυτό άλλωστε ήταν μια από τις κυριότερες απαιτήσεις. Τέλος η δυνατότητα καταγραφής της χρήσης των διαδικτυακών υπηρεσιών που παρέχουν οι τεχνολογίες πολυπλέγματος επιτρέπουν την εκ νέου σχεδίαση διαφορετικών μοντέλων παροχής αδειών χρήσης λογισμικού, βασισμένο σε διαφορετικά κριτήρια και ιδιότητες.

Η χρήση και αξιοποίηση του προτεινόμενου συστήματος θα δημιουργήσει νέες προκλήσεις στο επιστημονικό πεδίο της παροχής αδειών λογισμικού. Ένας από τους στόχους που είχαν τεθεί εξ αρχής ήταν η δυνατότητα ποιοτικής κοστολόγησης της χρήσης μια εφαρμογής. Το τελευταίο είναι πολύ σημαντικό καθώς θα προσδώσει περισσότερη ευελιξία στην εταιρία χρήστη μιας εφαρμογής λογισμικού σε ό,τι αφορά τον προγραμματισμό, την παραγγελία, την προμήθεια και τέλος τη χρήση της συγκεκριμένης εφαρμογής, καθστόντας την αποδοτικότερη σε σχέση με το κόστος της. Από την άλλη, οι εταιρίες διανομείς και πωλειτές εφαρμογών λογισμικών μπορούν να προσφέρουν διαφορετικά πακέτα υπηρεσιών και εφαρμογών, βασισμένα στη χρήση, τα οποία να καλύπτουν διαφορετικές ανάγκες επιχειρήσεων. Κατά αυτό τον τρόπο ενισχύεται η ανταγωνιστηκότητα μεταξύ τους. Καθώς συναντάμε σε ολοένα και περισσότερα προιόντα το γεγονός να τιμολογούνται ανάλογα με τη χρήση τους και όχι ανάλογα με την αγορά τους, ο συγγραφέας της παρούσας διπλωματικής εργασίας θεωρεί ότι αντίστοιχο τείνει να συμβεί και με τη βιομηχανία λογισμικού. Έτσι, συστήματα τα οποία θα επιλύουν αυτές τις απαιτήσεις για καταγραφή της χρήσης σε ένα ασφαλες διαδικτυακό περιβάλλον, κατά τα επιχειρηματικά πρότυπα συνδεσιμότητας ετερογενών δικτυακών συνιστωσών, φαντάζουν επίκαιρα και με μεγάλα περιθώρια ανάπτυξης.

6.2 Μελλοντικές επεκτάσεις

Ο μηχανισμός παροχής αδειών χρήσης λογισμικού που περιγράφηκε στην παρούσα διπλωματική εργασία αποτελεί μια λύση που λειτουργεί ολοκληρωμένα και ανεξάρτητα, ωστόσο επιδέχεται επεκτάσεις και συμπληρώσεις.

Όπως αναφέραμε, η χρήση του μεσισμικού πολυπλέγματος GRIA, και πιο συγκεκριμένα οι οντότητες λογισμικού που παρέχονται μέσω του πακέτου GRIA Service Provider καθιστούν δυνατή την καταγραφή και τον έλεγχο της χρήσης των διαδικτυακών υπηρεσιών. Αυτό γίνεται με τη δέσμευση της υπηρεσίας από την Υπηρεσία Διαχείρησης των Συμφωνιών Επιπέδου Υπηρεσιών (Service Level Agreement Management Service). Ο πάροχος διαπραγματεύται με τον καταναλωτή μια συγκεκριμένη συμφωνία που ορίζει τους περιορισμούς και ελάχιστες απαιτήσεις για τη χρήση της υπηρεσίας, καθώς επίσης και τις μεταβλητές και παραμέτρους χρήσης που θα χρεώνονται, και πόσο. Στη συνέχεια, δημιουργείται ένας λογαριασμός συναλλαγής για κάθε καταλανωτή της διαδικτυακής

υπηρεσίας, και εκεί καταγράφεται η κοστολόγηση της χρήσης της υπηρεσίας. Όλος αυτός ο μηχανισμός γίνεται στη συγκεκριμένη υλοποίηση χειροκίνητα από το διαχειριστή της υπηρεσίας. Αυτός φροντίζει να έρθει σε επαφή με τους συνεργαζόμενος πελάτες-καταναλωτές, στη συνέχεια αποφασίζουν ένα αποδεκτό SLA, και γίνεται η εκχώρηση ενός λογαριασμού συναλλαγής (trade account) στον καθέναν. Ο διαχειριστής είναι και πάλι υπεύθυνος για τη ενεργοποίηση (activation) του λογαριασμού, για τον έλεγχο της χωρητικότητάς του, για την αποδέσμευσή του και γενικώς οποιαδήποτε διαχείριση απαιτείται. Από τη στιγμή που ρυθμιστούν τα ζητούμενα από το διαχειριστή, η διαδικτυακή υπηρεσία μπορεί να καταναλώνεται από την εφαρμογή-πελάτη και η χρήση της να καταγράφεται αυτόματα, όπως ορίζεται από το μεσισμικό πολυπλέγματος GRIA.

Σε ένα πιο αυτοματοποιημένο μοντέλο, θα ήταν δυνατή η υλοποίηση μιας οντότητας λογισμικού από την πλευρά του daemon, η οποία θα μηχανογραφούσε την διαδικασία συμφωνίας του κοινά αποδεκτού SLA και των υπόλοιπων απαιτούμενων βημάτων. Το GRIA δίνει δυνατότητα για εύρεση και πλοήγηση στα χαρακτηριστικά των διαθέσιμων για κάθε υπηρεσία SLAs και την εντολή για δημιουργία ενός καινούριου trade account. Ο ρόλος του διαχειριστή ή υπευθύνου πωλήσεων του παρόχου θα εξακολουθούσε να υπάρχει. Ο διαχειριστής παραμένει υπεύθυνος να ενεργοποιεί τα νέα trade accounts, η όλη η διαδικασία όμως της συμφωνίας του SLA και της δημιουργίας του trade account μπορεί να αυτοματοποιηθεί. Επιπρόσθετα, ο daemon, πριν τη κλίση της διαδικτυακής υπηρεσίας για την απόκτηση της άδειας χρήσης μιας εφαρμογής, θα πρέπει να κάνει έλεγχο ότι υπάρχει ενεργοποιημένο trade account και συμφωνημένο SLA. Σε αντίθετη περίπτωση θα επιστρέφει ανάλογο σφάλμα. Τέλος, πριν την κλίση της υπηρεσίας, το GRIA δίνει τη δυνατότητα στον client-καταναλωτή της να ελέγξει τη διαθεσιμότητά της, είτε από πλευράς καταναλωτή και ύπαρξη απαιτούμενου ποσού στο λογαριασμό συναλλαγής του, είτε από πλευράς παρόχου και ικανοποίηση των ελάχιστων απαιτήσεων που ορίζονται στο SLA. Βέβαια, με την παρούσα υλοποίηση ο καταναλωτής καλεί την υπηρεσία, και ο έλεγχος αυτός γίνεται πριν την εκτέλεση της επιθυμητής μεθόδου. Σε περίπτωση που δεν επιτρέπεται η εκτέλεση με βάση το SLA, η μέθοδος δεν εκτελείται και ενημερώνεται ο καταναλωτής με αντίστοιχο σφάλμα. Παρόλα αυτά, θα ήταν προτιμότερο να γίνει ο έλεγχος πριν την κλήση της υπηρεσίας και ενημέρωση του καταναλωτή εξ αρχής για να αποφευχθεί η περιττή κίνηση στο διαδίκτυο, πράγμα που μπορεί να σημαίνει και καθυστέρηση (delay).

7

Βιβλιογραφία

[1] IPR – Helpdesk, European Commission DG Enterprise

[2] Ivana Dusparic, Pervasive Application Rights Management Architecture, University of Dublin, July 2005

[3] FLEXlm End Users Guide Version 9.5, Macrovision Corporation, August 2004

[4] http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html

[5] Paul Gustavson, Tram Chase, Larry Root, Karl Crosson, Moving towards a Service – Oriented Architecture (SOA) for Distributed Computing Simulation Environments, SimVentions,Inc.

[6] Alan Brown, Simon Johnston, Kevin Kelly, Service – Oriented Architecture and Component – Based Development to Build Web Service Applications, Rational Software Corporation, 2002

[7] Cover Pages: Stateful Web Services, http://xml.coverpages.org/statefulWebServices.html

[8] W3Schools: XML Tutorial, http://www.w3schools.com/xml/default.asp

[9] W3Schools: SOAP Tutorial, http://www.w3schools.com/soap/default.asp

[10] Representational State Transefer from Wikipedia http://en.wikipedia.org/wiki/Representational_State_Transfer

[11] Web Services Description Language (WSDL) W3C Note 15 March 2001 http://www.w3.org/TR/wsdl

[12] W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures W3C Candidate Recommendation 30 April 2009, http://www.w3.org/TR/xmlschema11-1/

[13] Service-oriented architecture, from wikipedia,

http://en.wikipedia.org/wiki/Service-oriented_architecture

[14] Grid computing, from wikipedia, http://en.wikipedia.org/wiki/Grid_computing

[15] What is the Grid? from GridPedia, http://www.gridipedia.eu/aboutgrid.html

[16] Mark Baker, Rajkumar Buyya and Domenico Laforenza, Grids and Grid technologies for wide-area distributed computing Softw. Pract. Exper. 2002; (in press) (DOI: 10.1002/spe.488)

[17] Service Oriented Collaborations for Industry and Commerce - GRIA,

http://www.gria.org/

[18] A Business Perspective - GRIA, http://www.gria.org/about-gria/a-business-perspective

[19] Basic Profile Version 1.0 Final Material,

http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html

[20] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

[21] Scaleable Relationship Management using WS-Federation,

http://www.gria.org/about-gria/relationship-to-standards#federation

[22] Web Services Addressing (WS-Addressing) W3C Member Submission 10 August 2004 http://www.w3.org/Submission/ws-addressing/

[23] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[24] Usage Monitoring with WS-Notification

http://www.gria.org/about-gria/relationship-to-standards#notification

[25] The Source for Java Developers, http://java.sun.com/

[26] GNU Operating System, http://www.gnu.org/licenses/

[27] What is Microsoft .NET Framework http://www.microsoft.com/NET/

[28] Apache Tomcat, http://tomcat.apache.org/

[29] Hibernate, https://www.hibernate.org/

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

[31] GRIA master 5.3.1 API http://archive.gria.org/javadocs/5.3.1/

[32] Public key infrastructure, from Wikipedia,

http://en.wikipedia.org/wiki/Public_key_infrastructure

[33] Miguel L. Bote-Lorenzo, Yannis A. Dimitriadis, and Eduardo Gomez Sanchez , Grid Characteristics and Uses: a Grid Definition, Santiago de Compostela, Spain, Feb. 2004

[34] Thomas Erl, Introduction to Web Services Technology: SOA, SOAP, WSDL and UDDI, September 2004

[35] Ian Foster, Carl Kesselman and Steven Tuecke, “The anatomy of the grid”, Supercomputer Applications, 2001.

[36] The Grid ,Blueprint for a New Computing Infrastructure Edited by Ian Foster and Carl Kesselman,1998.

[37] What is the Grid? A three point Checklist by Ian Foster, 2002.

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

Παράρτημα

A. Daemon

Tcp. CreateLog.java

```
package tcp;
import java.io.*;
import java.util.Date;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import javax.swing.JTextArea;
public class CreateLog{
	protected Writer LogOutput ;
	File file;
	public void createFile()throws IOException{
		file = new File("c://LicenseDaemonHistory.log");
		LogOutput = new BufferedWriter(new FileWriter(file,
				true));
	}
	public void updateFile(String str, int port) throws IOException{
		LogOutput.write('\n');
		LogOutput.write("-------------------------------");
		LogOutput.write('\n');
		DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy
HH:mm:ss");

				Date date = new Date();
				LogOutput.write(dateFormat.format(date));
				LogOutput.write('\n');
```

```
                    LogOutput.write(" the ip "+str+ " used the daemon at port:
"+port );
                                        LogOutput.write('\n');
                    System.out.println("The LicenseDaemonHistory is
        updated!!"); }
                                public void updateFile(String str, int port,
JTextArea area)
                                throws IOException{
                                        LogOutput.write('\n');
                                        LogOutput.write("----------------------
----------");
                                        LogOutput.write('\n');
                                        DateFormat dateFormat = new
SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
                                                Date date = new
Date();

        LogOutput.write(dateFormat.format(date));

        LogOutput.write('\n');

        LogOutput.write("the ip "+str+ " used the daemon at port: "+port );

        LogOutput.write('\n');
                                                area.append("The
LicenseDaemonHistory is updated!!"+'\n'); }

        public void closeLog(){

        try{

                LogOutput.close();}

        catch(IOException ex){}
```

}

}

Tcp.LicenseDaemon.java

```
package tcp;

import java.net.*;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.io.IOException;
import javax.swing.JTextArea;



public class LicenseDaemon extends Thread {

	ServerSocket serverSocket = null;
	int port;
	CreateLog log;
	JTextArea area = null;
	boolean SocketCreated = false;

	public LicenseDaemon(int i, CreateLog logfile) {
		port = i;
		log = logfile;
	}
```

```
public LicenseDaemon(int i, CreateLog logfile, JTextArea area1) {

        port = i;
        log = logfile;
        area = area1;
}

public void run()
{
        Socket incomingCall = null;
        Simple2 sim2 = null;

        try
        {       //WAITING CONNECTIONS FROM MATHEMATICA
                serverSocket = new ServerSocket(port);
        } catch (IOException e1) {
                System.out.println(e1.getMessage());
        }

        try
        {
                if (area == null)
                        System.out.println("server working and waiting...at " + port);
                else
                        area.append("server working and waiting...at " + port + '\n');


                //USING EXECUTOR TO HANDLE MULTIPLE REQUESTS
                ExecutorService executor =
Executors.newFixedThreadPool(Integer.MAX_VALUE);
                int counter = 0;
                for (;;) {
```

```
                    for (;;) {
                        // MATHEMATICA TRIES TO CONNECT
                        incomingCall = serverSocket.accept();
                        counter++;
                        Callable <Simple2> deamonHandler = new
LicenseDaemonHandler(incomingCall, counter, port, area);
                        Future<Simple2> submit =
executor.submit(deamonHandler);
                        Future<Simple2> f = submit;
                        sim2 = f.get();

                        if (sim2.valid == true)
                            continue;
                        else
                            break;
                    }

                }


            } catch (Exception e) {
                try {
                    serverSocket.close();
                    incomingCall.close();

                } catch (IOException e1) {
                    System.out.println("Error trying to close connections. " +
e.getMessage());
                }
                e.printStackTrace();
            }
```

```
	}
}
```

Tcp. LicenseDaemonAux.java

```
package tcp;
import java.io.*;
import javax.swing.JTextArea;

public class LicenseDaemonAux {
	public int flag;
	public static void main(String[] args)
	{
		LicenseDaemonAux daemonAux = new LicenseDaemonAux(args[0]);

	}


	public LicenseDaemonAux(String from)
	{
		CreateLog log = new CreateLog();

		LicenseDaemon l = new LicenseDaemon(Integer.parseInt(from), log);
		l.start();
	}


	public LicenseDaemonAux(String from, String to)
	{
```

```
		CreateLog log = new CreateLog();

		if (from.equals(null) && to.equals(null))
		{
			for (int m = 27000; m < 27011;m++)
			{
				LicenseDaemon l = new LicenseDaemon(m, log);
				l.start();
			}
		}
		else
		{
			int i = Integer.parseInt(from);
			int j = Integer.parseInt(to);
			if (i <= j)
			{
				for (int k=i; k <= j; k++)
				{
					LicenseDaemon daemon = new LicenseDaemon(k,
log);
					daemon.start();
				}
			}
			if (i > j)
				System.out.println("wrong range!");
		}
	}

	public LicenseDaemonAux(String from, String to, JTextArea area)
	{
		CreateLog log = new CreateLog();
```

```
                if (from.equals(null) && to.equals(null))
                {
                        for (int m = 27000; m < 27011;m++)
                        {
                                LicenseDaemon daemon = new LicenseDaemon(m, log,
area);
                                daemon.start();}
                        }
                else
                {
                        int i = Integer.parseInt(from);
                        int j = Integer.parseInt(to);
                        if (i <= j)
                        {
                                for (int k = i; k <= j; k++)
                                {
                                        LicenseDaemon licenseDaemon = new
LicenseDaemon(k, log, area);
                                        licenseDaemon.start();}
                        }
                        if (i > j)
                        {
                                System.out.println("wrong range!");
                                area.append("wrong range!");
                        }

                }

        }

}
```

Tcp. LicenseDaemonHandler.java

```
package tcp;

import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.concurrent.Callable;
import javax.swing.JTextArea;

import tcp.client.LicenseServiceGriaClient;;

class LicenseDaemonHandler extends Thread implements Callable<Simple2> {
	Socket incomingCall;
	int counter;
	String url;
	String ip;
	int port;
	InputStream inputStream;
	OutputStream outputStream;
	private int secondport;
	private boolean valid = false;
	Simple sim = new Simple(null, counter);
	Simple2 sim2;
	JTextArea area;

	public LicenseDaemonHandler(Socket i, int c, int port)
	{
		incomingCall = i;
```

```
		counter = c;
		this.port = port;
	}

	public LicenseDaemonHandler(Socket i, int c, int port, JTextArea area1)
	{
		incomingCall = i;
		counter = c;
		this.port = port;
		area = area1;
	}

	public Simple2 call()
	{
		try {
			int i = this.counter;
			byte[] message = new byte[2048];

			if(area == null)
				System.out.println("Server @"+ port + ":application number " + i + " running...");
			else
				area.append("Server @"+ port + ": application number " + i + " running..."+'\n');


			//CREATING SERVICE CLIENT TO FORWARD STREAMS TO THE SERVICE
			System.out.println("Creating service Client");
			LicenseServiceGriaClient griaClient = new LicenseServiceGriaClient();
```

```
                System.out.println("deamon: initializing in/output streams");
                inputStream = incomingCall.getInputStream();
                outputStream = incomingCall.getOutputStream();


                int len = 0;

                //Reading Couple of Message

                System.out.println("deamon: Couple of messages: " + counter + "\nwaiting to read message from application");
                len = inputStream.read(message);
                System.out.println("deamon: message read. Lenght: " + String.valueOf(len) + ". Calling deamon2 to forward the message");
                sim = griaClient.transferMessage(port, message, len);
                System.out.println("deamon: got response frome the service. Now forwarding the response back to the application");
                outputStream.write(sim.pinax, 0, sim.length);

                System.out.println("Closing streams and connections");
                inputStream.close();
                outputStream.close();
                incomingCall.close();
                counter++;

        }  catch (Exception e){
                e.printStackTrace();
                valid = true;
                if(area == null)
                        System.out.println("Server @" + port + ": Authorization failed.");
                else
```

```
                    area.append("Server @" + port + ":       Authorization
failed."+'\n');

                    try {

                        inputStream.close();
                        outputStream.close();
                        incomingCall.close();
                        counter++;

                    } catch (IOException e1) {
                        e1.printStackTrace();
                    }
                }

                sim2 = new Simple2(port, valid);
                return sim2;
            }

}
```

Tcp. NetworkInfo.java

```
package tcp;

import java.net.InetAddress;
import java.io.InputStream;
import java.io.BufferedInputStream;
import java.io.IOException;
import java.text.ParseException;
import java.util.StringTokenizer;
```

```
public final class NetworkInfo {

	public static String getMacAddress() throws IOException {
		try {
			return parseMacAddress(runIpConfigCommand());
		} catch(ParseException ex) {
			ex.printStackTrace();
			throw new IOException(ex.getMessage());
		}
	}

	private final static String parseMacAddress(String ipConfigResponse) throws
ParseException {
		String localHost = null;
		try {
			localHost = InetAddress.getLocalHost().getHostAddress();
		} catch(java.net.UnknownHostException ex) {
			ex.printStackTrace();
			throw new ParseException(ex.getMessage(), 0);
		}
		StringTokenizer tokenizer = new
		StringTokenizer(ipConfigResponse, "\n");
		String lastMacAddress = null;
		while(tokenizer.hasMoreTokens()) {
			String line = tokenizer.nextToken().trim();
			// see if line contains IP address
			if(line.endsWith(localHost) && lastMacAddress != null) {
				return lastMacAddress;
			}

			// see if line contains MAC address
			int macAddressPosition = line.indexOf(":");
```

```
                    if(macAddressPosition <= 0) continue;
                    String macAddressCandidate =
                            line.substring(macAddressPosition + 1).trim();
                    if(isMacAddress(macAddressCandidate)) {
                            lastMacAddress = macAddressCandidate;
                            continue;
                    }

            }
            ParseException ex = new ParseException("cannot read MAC      address
from [" + ipConfigResponse + "]", 0);
                                    ex.printStackTrace();
                    throw ex;
    }

    private final static boolean isMacAddress(String
                    macAddressCandidate) {
            if(macAddressCandidate.length() != 17) return false;
            return true;
    }

    private final static String runIpConfigCommand() throws
    IOException {
            Process p = Runtime.getRuntime().exec("ipconfig /all");
            InputStream stdoutStream = new
            BufferedInputStream(p.getInputStream());
            StringBuffer buffer= new StringBuffer();
            for (;;) {
                    int c = stdoutStream.read();
                    if (c == -1) break;
                    buffer.append((char)c);
            }
```

```
                String outputText = buffer.toString();
                stdoutStream.close();
                return outputText;
        }
}
```

Tcp. Simple2.java

```
package tcp;
public class Simple2 {
        public int port2;
        public boolean valid;
        public Simple2(int p2, boolean flag){
                port2 = p2 ;
                valid = flag;
        }
}
```

Tcp. Simple.java

```
package tcp;
public class Simple {
        public byte [] pinax;
        public int length;
        public Simple(byte[] p, int l){
                pinax = p ;
                length = l ;
        }
```

```
}
```

Tcp. UserInterface.java

```
package tcp;
import javax.swing.*;
import java.awt.event.*;

public class UserInterface extends JFrame implements ActionListener{

	private static final long serialVersionUID = 1L;
	boolean choices=false;
	String Ip;
	String From=null;
	String To=null;
	String Standalone=null;
	int i=0;

	//JTextFields
	JTextField standalone=new JTextField(4);
	JTextField from=new JTextField(4);
	JTextField to=new JTextField(4);

	//JButtons & JRadioButtons
	ButtonGroup choice =new ButtonGroup();
	JButton start=new JButton("START!");
	JButton stop=new JButton("STOP!");
	JButton clear=new JButton("CLEAR!");
	JRadioButton RangePorts= new JRadioButton ("Range", true);
	JRadioButton StandPort =new JRadioButton("Standalone Port:    ");
```

```
JTextArea area=new JTextArea(15, 33);
JScrollPane scroll;
JLabel toLabel=new JLabel("To: ");
JLabel fromLabel=new JLabel("From: ");
JPanel pane;

public UserInterface()
{
        super("Server's Settings");
        setSize(400, 550);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        pane= new JPanel();
        JLabel range= new JLabel(" Range of     Ports of License Daemon: ");
        JLabel allign= new JLabel("       ");
        standalone.setEditable(false);
        ButtonGroup ranges=new ButtonGroup();
        ranges.add(RangePorts);
        ranges.add(StandPort);
        RangePorts.addActionListener(this);
        StandPort.addActionListener(this);
        stop.addActionListener(this);
        area.setLineWrap(true);
        area.setEditable(false);
        scroll=new                                          JScrollPane(area,
ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
        start.addActionListener(this);
        clear.addActionListener(this);
        pane.add(range);
        pane.add(allign);
        pane.add(allign);
        pane.add(allign);
```

```
        pane.add(RangePorts);
        pane.add(fromLabel);
        pane.add(from);
        pane.add(toLabel);
        pane.add(to);
        pane.add(StandPort);
        pane.add(standalone);
        pane.add(scroll);
        pane.add(start);
        pane.add(stop);
        pane.add(clear);
        setContentPane(pane);
        setVisible(true);
}

public void actionPerformed(ActionEvent evt){

        if (evt.getSource()==RangePorts){
                from.setEditable(true);
                to.setEditable(true);
                standalone.setEditable(false);
        }

        if (evt.getSource()==StandPort){
                standalone.setEditable(true);
                from.setEditable(false);
                to.setEditable(false);

        }

        if (evt.getSource()==start) {
```

```
                    if (RangePorts.isSelected()){
                        From=from.getText();
                        To=to.getText();

                        if((!From.equals(""))&&(!To.equals(""))) {
                            LicenseDaemonAux a = new
LicenseDaemonAux(From, To, area);
                            start.setEnabled(false);}
                    }

                    Standalone=standalone.getText();
                    if((!Standalone.equals(""))){
                        LicenseDaemonAux a=new LicenseDaemonAux(Standalone,
Standalone, area);
                        start.setEnabled(false);
                    }

                }

                if(evt.getSource()==clear){
                    area.setText("");
                }

                if (evt.getSource()==stop) {
                    System.exit(0);
                }
        }

        public static void main(String[] args){
                UserInterface u=new UserInterface();
        }
}
```

Tcp.client. LicenseServiceGriaClient.java

```
package tcp.client;

import sample.SampleService;
import sun.misc.BASE64Decoder;

import java.rmi.RemoteException;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;

import org.apache.axis.message.addressing.EndpointReferenceType;
import org.apache.axis.utils.XMLUtils;
import org.apache.log4j.Logger;
import org.w3c.dom.Document;
import java.rmi.RemoteException;
import java.util.ArrayList;

import javax.xml.messaging.Endpoint;
import javax.xml.ws.Dispatch;

import java.rmi.RemoteException;
import java.util.ArrayList;

import javax.xml.messaging.Endpoint;
import javax.xml.ws.Dispatch;

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

```
import uk.ac.soton.ecs.iam.grid.client.swing.SwingInputHandler;
import uk.ac.soton.ecs.iam.grid.comms.client.AxisTransport;
import uk.ac.soton.ecs.iam.grid.comms.client.RemoteDataService;
import uk.ac.soton.itinnovation.grid.client.engine.impl.DefaultAttributeSelector;
import uk.ac.soton.itinnovation.grid.client.engine.impl.DefaultFederationSelector;
import uk.ac.soton.itinnovation.grid.client.engine.impl.DefaultInvocationEngine;
import uk.ac.soton.itinnovation.grid.client.engine.impl.LocalRegistry;
import uk.ac.soton.itinnovation.grid.client.plugins.GridClientPluginManager;
import uk.ac.soton.itinnovation.grid.client.proxy.HelperProxyFactory;
import uk.ac.soton.itinnovation.grid.client.proxy.InvocationEngineProxyFactory;
import uk.ac.soton.itinnovation.grid.client.proxy.ProxyFactory;
import uk.ac.soton.itinnovation.grid.comms.client.ITInnovSocketFactory;
import uk.ac.soton.itinnovation.grid.comms.client.InteractiveX509TrustManager;
import uk.ac.soton.itinnovation.grid.comms.client.SAMLTokenCache;
import uk.ac.soton.itinnovation.grid.comms.client.WSDLCache;
import uk.ac.soton.itinnovation.grid.comms.client.WSDLCacheImpl;
import uk.ac.soton.itinnovation.grid.service.types.SimpleRegistry;
import uk.ac.soton.itinnovation.grid.types.ConversationID;
import uk.ac.soton.itinnovation.grid.types.MatchPattern;
import uk.ac.soton.itinnovation.grid.types.SubjectDescription;
import uk.ac.soton.itinnovation.grid.utils.CertificateTrustValidator;
import uk.ac.soton.itinnovation.grid.utils.IdentityProvider;
import uk.ac.soton.itinnovation.grid.utils.TestIdentityProvider;
import uk.ac.soton.itinnovation.grid.utils.KeystoreIdentityProvider;
import uk.ac.soton.itinnovation.grid.utils.Transport;


import java.io.IOException;

import sample.SampleResource;
import sample.SampleService;
import tcp.Simple;
```

```
import tcp.utils.*;


public class LicenseServiceGriaClient {

	ProxyFactory proxyFactory = null;
	DefaultInvocationEngine engine = null;
	AxisTransport transport = null;

	public LicenseServiceGriaClient() {}

	@SuppressWarnings("deprecation")
	public synchronized Simple transferMessage(int port, byte[] data, int len)
	{
		Simple result = new Simple(new byte[0], 0);

		ProxyHelper proxyHelper = new ProxyHelper();
		this.proxyFactory = proxyHelper.getFactory();
		this.transport = proxyHelper.getTransport();


		EndpointReferenceType serviceEPR = ConversationID.getEPR(new
ReadConfigurationHelper().readServiceUrl());

		System.out.println("Trying to create proxy to the Sample Service");
		SampleService sampleService = proxyFactory.createProxy(serviceEPR,
SampleService.class);
		System.out.println("Proxy created succesfully!\n");



		try {
```

```
//Get a Proxy to the corresponding resource
SubjectDescription subject = new SubjectDescription(
transport.getSubjectCert());
SubjectDescription issuer = new SubjectDescription(
transport.getIssuerCert());
MatchPattern pattern = new MatchPattern(subject, issuer);

int count = (sampleService.getResources()).length;
System.out.println("getResources - Count: " + count);

EndpointReferenceType resourceERP = null;

if(count == 0)
    resourceERP =
sampleService.createSampleResource("managed user", pattern);
else
{
    EndpointReferenceType [] resourceERPs =
sampleService.getResources();
    resourceERP = resourceERPs[0];
    System.out.println("USED EXISTING RESOURCE");
}

SampleResource sampleResource =
proxyFactory.createProxy(resourceERP, SampleResource.class);

System.out.println("Resource created using the proxy factory");

String message = sampleResource.transferMessage(port, data, len);
System.out.println("Data transfered");
```

```
                BASE64Decoder en1 = new BASE64Decoder();
                byte [] array = new byte[message.length()];
                try {
                        array = en1.decodeBuffer(message);
                } catch (IOException e) {
                        e.printStackTrace();
                }
                result = new Simple(array, array.length);

        } catch (RemoteException e) {
                e.printStackTrace();
        }

        return result;
    }
}
```

Tcp.utils. ReadConfigurationHelper.java

```
package tcp.utils;

import java.io.*;
import java.lang.String;

public class ReadConfigurationHelper {

    public ReadConfigurationHelper() {

    }
```

```
	public String readServiceUrl()
	{
		String result = null;

		try
		{
			FileInputStream inputstream = new FileInputStream("config.sys");
			DataInputStream datastream = new DataInputStream(inputstream);
			BufferedReader bufferin = new BufferedReader(new
InputStreamReader(datastream));

			if((result  = bufferin.readLine())!=null)
				datastream.close();
		}
		catch(IOException ex) {
			System.out.println(ex.getMessage());
		}

		return result;
	}

	public String readPublicKeyPath()
	{
		String result = null;

		try
		{
			FileInputStream inputstream = new FileInputStream("config.sys");
			DataInputStream datastream = new DataInputStream(inputstream);
			BufferedReader bufferin = new BufferedReader(new
InputStreamReader(datastream));
```

```
                    if((result  = bufferin.readLine())!=null)
                            if((result  = bufferin.readLine())!=null)
                                    datastream.close();

            }
            catch(IOException ex) {
                    System.out.println(ex.getMessage());
            }

            return result;
        }

}
```

Tcp.utils. ProxyHelper.java

```
package tcp.utils;

import java.rmi.RemoteException;
import java.util.ArrayList;

import javax.xml.messaging.Endpoint;
import javax.xml.ws.Dispatch;

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

import uk.ac.soton.ecs.iam.grid.client.swing.SwingInputHandler;
import uk.ac.soton.ecs.iam.grid.comms.client.AxisTransport;
import uk.ac.soton.ecs.iam.grid.comms.client.RemoteDataService;
import uk.ac.soton.itinnovation.grid.client.engine.impl.DefaultAttributeSelector;
import uk.ac.soton.itinnovation.grid.client.engine.impl.DefaultFederationSelector;
```

```
import uk.ac.soton.itinnovation.grid.client.engine.impl.DefaultInvocationEngine;
import uk.ac.soton.itinnovation.grid.client.engine.impl.LocalRegistry;
import uk.ac.soton.itinnovation.grid.client.plugins.GridClientPluginManager;
import uk.ac.soton.itinnovation.grid.client.proxy.HelperProxyFactory;
import uk.ac.soton.itinnovation.grid.client.proxy.InvocationEngineProxyFactory;
import uk.ac.soton.itinnovation.grid.client.proxy.ProxyFactory;
import uk.ac.soton.itinnovation.grid.comms.client.ITInnovSocketFactory;
import uk.ac.soton.itinnovation.grid.comms.client.InteractiveX509TrustManager;
import uk.ac.soton.itinnovation.grid.comms.client.SAMLTokenCache;
import uk.ac.soton.itinnovation.grid.comms.client.WSDLCache;
import uk.ac.soton.itinnovation.grid.comms.client.WSDLCacheImpl;
import uk.ac.soton.itinnovation.grid.service.types.SimpleRegistry;
import uk.ac.soton.itinnovation.grid.types.ConversationID;
import uk.ac.soton.itinnovation.grid.types.MatchPattern;
import uk.ac.soton.itinnovation.grid.types.SubjectDescription;
import uk.ac.soton.itinnovation.grid.utils.CertificateTrustValidator;
import uk.ac.soton.itinnovation.grid.utils.IdentityProvider;
import uk.ac.soton.itinnovation.grid.utils.TestIdentityProvider;
import uk.ac.soton.itinnovation.grid.utils.KeystoreIdentityProvider;
import uk.ac.soton.itinnovation.grid.utils.Transport;

import sample.SampleResource;
import sample.SampleService;


public class ProxyHelper {

        private ProxyFactory proxyFactory = null;
        private DefaultInvocationEngine engine = null;
        private AxisTransport transport = null;
```

```
@SuppressWarnings("deprecation")
public ProxyHelper()
{
        System.out.println("Starting ...");
        // Create my identity
        KeystoreIdentityProvider idp = new KeystoreIdentityProvider(new ReadConfigurationHelper().readPublicKeyPath(), "service", "admin".toCharArray());
        System.out.println("KeystoreIdentifier created");

        // Create transport object which invokes operations
        transport = new AxisTransport(idp);
        System.out.println("Kyestore added to AxisTransport");

        // We need a cache to store our wsdl files
        WSDLCache cache = new WSDLCacheImpl(transport);
        System.out.println("WSDLCache created");

        /* When we communicate with services we need to choose which
         * services we trust, we do this using a TrustValidator
         * This example uses a graphical validator InteractiveX509TrustManager
         * but a non interative TestIdentityProvider.TRUST_EVERYONE
         * can be used but only for testing. It should NOT be used a
         * real deployment.
         */
        CertificateTrustValidator manager = new InteractiveX509TrustManager(new SwingInputHandler());
        transport.setCertificateTrustValidator(manager);
        ITInnovSocketFactory.setHandlers(idp, manager);


        // Define a registry where we will store EndpointReference's of services and resources
```

```
ArrayList<SimpleRegistry> registries = new ArrayList<SimpleRegistry>();
registries.add(new LocalRegistry());

// Define a cache for any tokens we get from Membership Groups
SAMLTokenCache tokenCache = new SAMLTokenCache();

// Load the GRIA plugins.
GridClientPluginManager pluginManager = new GridClientPluginManager();
pluginManager.loadPlugins();



// Define an engine which uses the transport to invoke operations
engine = new DefaultInvocationEngine(transport);

// Add the registry to the engine
engine.setSelectedRegistries(registries);

// Tell the engine how to find tokens by defining an
//attribute selector, which stores tokens in the token cache
engine.setAttributeSelector(new DefaultAttributeSelector(tokenCache));

// Tell the engine to use a federation selector, which finds
//management context (SLAs, Trade Accounts) when invoking an operation.
engine.setFederationSelector(new DefaultFederationSelector());

/* The proxy factory creates proxy object from
 * EndpointReference's which help us communicate with
 * the services. The helper proxy factory adds some additional
 * useful methods
 */
ProxyFactory rawProxyFactory = new
InvocationEngineProxyFactory(engine, cache);
```

```
                proxyFactory = new HelperProxyFactory(rawProxyFactory,
pluginManager.getHelperRegistry());
                engine.setProxyFactory(proxyFactory);

        }

        public ProxyFactory getFactory() {
                return proxyFactory;
        }

        public SimpleRegistry getRegistry() {
                return engine.getSelectedRegistries().get(0);
        }

        public AxisTransport getTransport() {
                return transport;
        }

}
```

B. GRIA License Service

LicenseService.java

```
package license;

import uk.ac.soton.itinnovation.grid.types.MatchPattern;
import uk.ac.soton.itinnovation.grid.comms.wsdl.SoapBindingName;
import uk.ac.soton.itinnovation.grid.comms.wsdl.WebService;
import uk.ac.soton.itinnovation.grid.comms.wsdl.WebMethod;
import uk.ac.soton.itinnovation.grid.types.GridService;
import java.rmi.RemoteException;
import uk.ac.soton.itinnovation.grid.comms.reportapi.ReportAPI;
import uk.ac.soton.itinnovation.grid.comms.management.ManagedGridService;
import org.apache.axis.message.addressing.EndpointReferenceType;


/** The public interface for the licenseService. This includes all service operations which don't
 * take a context (i.e., they operation on the service itself rather than on a particular resource).
 *
 * Extending {@link ManagedGridService} allows the service administrator to add
 * to, remove from, and query the service's list of trusted account services. When spending
 * money, a user must have access to an account at one of these.
 *
 * Extending {@link ReportAPI} allows the SLA service to get usage reports from us.
 *
 * Extending {@link GridService} provides operations to get our identity and query for resources.
 */
@WebService(targetNamespace = "http://www.it-innovation.soton.ac.uk/2006/grid/license",
```

```
        name = "licenseService",
        serviceName = "licenseService")
@SoapBindingName("licenseServiceSoapBinding")
public interface licenseService extends ManagedGridService, ReportAPI, GridService {
    /** A unique string to represent the type of the service itself. The PBAC policy with
     * this name is used to protect operations on the service which have no context.
     * @see uk.ac.soton.itinnovation.grid.types.ConversationID#getResourceType
     */
    final String license_SERVICE_RESOURCE_TYPE = "http://license/2006/licenseServiceResourceType";

    /**
     * Create a new resource. Calling this may require an account or an SLA to be specified
     * in the SOAP header (or may require nothing at all), depending on how
     * the service has been configured.
     * <p>
     * On success, the EPR of the new resource is returned. Anyone who matches the 'owner'
     * rule can access the resource as its owner. Typically, this rule just matches the
     * identity of the person creating the resource.
     *
     * @param label A label for the new resource
     * @param owner A pattern to allow access by the new resource's owner
     * @return the EPR of the new resource
     */
    @WebMethod(usesBillingHeader = true)
    EndpointReferenceType createlicenseResource(String label, MatchPattern owner) throws RemoteException;

    /* Add additional operations here if they take no context (they act on the service as
     * a whole, rather that on a particular resource). Contextualised operations should be
```

```
        * added to {@link license} instead.
        */

}
```

LicenseResource.java

```
package license;

import uk.ac.soton.itinnovation.grid.types.IdentifiableResource;
import uk.ac.soton.itinnovation.grid.comms.wsdl.WebMethod;
import uk.ac.soton.itinnovation.grid.types.ResourceMetadata;
import uk.ac.soton.itinnovation.grid.comms.wsrf.WSResourceLifetime;
import uk.ac.soton.itinnovation.grid.comms.wsdl.SoapBindingName;
import uk.ac.soton.itinnovation.grid.comms.wsdl.WebService;
import uk.ac.soton.itinnovation.grid.types.PolicyManagement;

import java.rmi.RemoteException;

import java.lang.*;
import java.net.*;



/** The public interface for the license objects.
 * Each method in this interface is a SOAP operation on the licenseResource service.
 * @see licenseService
 * @see license.client.licenseConversation
 */
@WebService(targetNamespace = "http://www.it-innovation.soton.ac.uk/2006/grid/license",
                    name = "licenseResource",
```

```
        wsdlLocation = "license/licenseresource.wsdl",
        serviceName = "licenseResource")
@SoapBindingName("licenseResourceSoapBinding")
public interface licenseResource extends PolicyManagement, WSResourceLifetime, ResourceMetadata, IdentifiableResource {
    /** A unique string to represent a type of resource accessed through the service. The PBAC policy with
     * this name is used to protect operations on the service which quote a context with this type.
     * When discovering resources, they have this type in their metadata.
     * @see uk.ac.soton.itinnovation.grid.types.ConversationID#getResourceType
     */
    final String license_RESOURCE_TYPE = "http://license/2006/licenseResourceType";

    /* Add additional operations here. You must also add them to the PBAC resource policy
     * (license-policy.xml).
     * After changing the policy, you must undeploy the old one using the web interface.
     * Returning the main admin screen will then automatically deploy the new version.
     */

    @WebMethod
    int check() throws RemoteException;

    @WebMethod
    String speak() throws RemoteException;

    @WebMethod
    int add(int a) throws RemoteException;

    @WebMethod
```

```
    byte[] addBytes(byte[] array) throws RemoteException;

    @WebMethod
    int addServerConnection() throws RemoteException;

    @WebMethod
    String transferMessage(int port, byte[] message, int len) throws RemoteException;

    @WebMethod
    String getServerURL() throws RemoteException;


}
```

LicenseServiceImpl.java

```
package license.impl;

import java.util.List;
import license.licenseResource;          // SOAP operations on license resources
import license.licenseService;           // SOAP operations on the service
import license.licenseResourceBean;      // Bean holding resource state

// Client imports - we act as a client to the account and SLA services
import uk.ac.soton.ecs.iam.grid.comms.client.Conversation;
import uk.ac.soton.ecs.iam.grid.comms.client.TradeAccountConversation;
import uk.ac.soton.ecs.iam.grid.comms.client.SLAConversation;
import uk.ac.soton.itinnovation.grid.service.types.Constraint;
import uk.ac.soton.itinnovation.grid.service.types.UsageReport;
import uk.ac.soton.itinnovation.grid.service.types.Metric;
import uk.ac.soton.itinnovation.grid.service.types.UsageType;
```

```
import uk.ac.soton.itinnovation.grid.service.types.Bound;

// Standard Java imports
import java.util.Date;
import java.math.BigDecimal;
import java.util.GregorianCalendar;
import org.apache.log4j.Logger;
import org.w3c.dom.Document;
import java.rmi.RemoteException;

// Standard Grid service imports
import java.util.Arrays;
import uk.ac.soton.ecs.iam.grid.utils.ImplementationFactory;
import uk.ac.soton.itinnovation.grid.gridservit.context.helpers.AdditionalContextHelper;
import org.apache.axis.message.addressing.AttributedURI;
import org.apache.axis.message.addressing.EndpointReferenceType;
import uk.ac.soton.itinnovation.grid.comms.webadmin.WebAdmin;
import uk.ac.soton.itinnovation.grid.service.utils.gridservit.GridServiceLite;
import
uk.ac.soton.itinnovation.grid.service.utils.trustedaccounts.TrustedManagementServices;
import uk.ac.soton.itinnovation.grid.types.IDType;
import uk.ac.soton.itinnovation.grid.utils.AtomUtils;
import uk.ac.soton.itinnovation.grid.types.ConversationID;
import uk.ac.soton.itinnovation.grid.types.GridFailureException;
import uk.ac.soton.itinnovation.grid.service.utils.reportapi.ReportAPIHelper;
import uk.ac.soton.itinnovation.grid.service.types.wsn.NotificationMessage;
import uk.ac.soton.itinnovation.grid.service.types.GridResource;

// Access control
import uk.ac.soton.itinnovation.grid.types.MatchPattern;
import uk.ac.soton.itinnovation.grid.types.PolicyRule;
import uk.ac.soton.itinnovation.grid.pbac2.pdp.PDP;
```

```
import uk.ac.soton.itinnovation.grid.pbac2.pdp.PBACUtils;
import uk.ac.soton.itinnovation.grid.pbac2.pep.PEPServiceResource;

// Hibernate persistence

/** The implementation of licenseService. The mapping from the interface to this class is in the
 * implementationfactory.properties file.
 */
@PEPServiceResource(singletonResource = "service:license.licenseService",
                    resourceType = licenseService.license_SERVICE_RESOURCE_TYPE)
public class licenseServiceImpl extends GridServiceLite implements licenseService, WebAdmin {

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

        /** When creating a new resource using an account, the user is billed
         * for this amount. If billing fails, the request to create the
         * resource is rejected. This would normally be made configurable.
         */
        private BigDecimal createResourceCharge = new BigDecimal(12.50);

        /** A unique identifier for the number-of-license-resources metric.
         * An SLA service can be used to limit the number of license
         * resources which a user can have.
         */
        static final Metric METRIC_license_RESOURCES = new Metric("http://license/2006/metrics/license-resource");

        /** Usage reports for the SLA service are queued here. */
        ReportAPIHelper pullPoint = new ReportAPIHelper();
```

```
public licenseServiceImpl() {
        super("licenseService");

        try {
                reloadConfiguration();
        } catch (Exception ex) {
                // If we can't read the config, just get the user to set it up correctly
                log.warn("Failed to load configuration", ex);
        }
}

public void ensurePoliciesDeployed() throws RemoteException {
        /* This is called by the web administration code when viewing the main admin page.
         * Deploy any PBAC policies and resources we need.
         */
        // This policy protects operations in the licenseService interface

pbacUtils.ensureDeployed(licenseService.license_SERVICE_RESOURCE_TYPE, "license-service-policy.xml");


groupUtils.ensureGroupDeployed(TrustedManagementServices.MANAGEMENT_SERVICES_GROUP, new PolicyRule[] {
                        new PolicyRule(new MatchPattern("special:this-service"), PDP.GROUP_MEMBER_ROLE)});


        // This policy protects operations in the license interface
        pbacUtils.ensureDeployed(licenseResource.license_RESOURCE_TYPE, "license-policy.xml");
```

```
        // This object represents the service itself. It will be the only resource of this type.
        pbacUtils.ensureServiceResource(this, new PolicyRule[] {
                new PolicyRule(MatchPattern.createAnyonePattern(), "world"),
                new PolicyRule(new MatchPattern(TrustedManagementServices.MANAGEMENT_SERVICES_GROUP),
TrustedManagementServices.MANAGEMENT_ROLE)});

    }

    /** Reload the configuration. Can be used by the admin tool to reload the configuration
     * without restarting tomcat.
     * @throws Exception if the configuration is not yet valid
     */
    public void reloadConfiguration() throws Exception {
        /* Read your configuration files here... */
    }

    protected EndpointReferenceType generateEPR(GridResource bean) {
        EndpointReferenceType EPR = super.generateEPR(bean);
        ConversationID.setParent(EPR, thisServiceAddress);

        if (bean instanceof licenseResourceBean) {
            licenseResourceImpl impl = (licenseResourceImpl) ImplementationFactory.getSingletonInstance(licenseResource.class);
            try {
                EPR.setAddress(new AttributedURI("" + impl.getSoapEndpoint()));
            } catch (org.apache.axis.types.URI.MalformedURIException ex) {
                throw new RuntimeException(ex);
```

```
                }
            }
            return EPR;
        }

        public EndpointReferenceType createlicenseResource(String label, MatchPattern
owner) throws RemoteException {
            // Check the arguments are OK
            if (label == null)
                throw new IllegalArgumentException("label is null!");
            if (owner == null)
                throw new IllegalArgumentException("owner is null!");

            /* Ensure that the pattern matches the caller. This is just a sanity check
             * (to prevent the user from creating resources they can't access).
             */
            PBACUtils.validatePattern(owner, getCurrentUser());

            /* Get a proxy to the managing resource (SLA or account)
             * specified by the client, and check that it is suitable (we
             * trust the service and the user can use the resource).
             * Returns null if the user asked to use the service for free,
             * and this is allowed.
             */
            Conversation managingConversation = management.getBillingInfo(

        AdditionalContextHelper.getSingleHeader(ConversationID.BILLING_INFO),
                        getCurrentUser());

            /* Create the new resource object (a Java bean) and store the
             * managing resource EPR and the user's chosen label on it.
             *
```

```
                * At this point, managingConversation is known to be trusted,
                * although the user may not actually have enough money or
                * usage to create the new object.
                */
                licenseResourceBean resource = new licenseResourceBean(managingConversation, label);

                /* Save with hibernate, which will assign us a resource ID. This ID is needed by
                * checkCreationOK() and newProcess().
                */
                EndpointReferenceType epr = null;
                boolean success = false;
                addHibernatedObject(resource);
                try {
                        String resourceID = resource.getResourceID();

                        // Now make the new resource known to PBAC. The resource starts in the
                        // 'locked' state, which means that incoming requests will be queued until
                        // we unlock it.
                        //
                        // We do this before telling the SLA or account services about the resource
                        // to avoid a race condition where they could try to invoke a method on it
                        // (e.g., 'destroy') and get an InvalidResourceIDException.
                        pdp.newProcess(resource.getResourceType(), resourceID);
                        try {
                                // Does the user have enough money or usage to create this resource?
                                checkCreationOK(resource, managingConversation);
```

```
                    // Any expensive setup can be done here. If an
                    // exception is thrown, the new resource is
                    // destroyed and the user is not charged.

                    // Bill the user for creation of the resource.
                    recordCreation(resource, managingConversation, new
IDType(getCurrentUser()));

                    /* Perform any additional setup here:
                     * - The user's account/SLA is valid and they have already
been charged.
                     * - If an exception is thrown at this point the resource is
destroyed
                     *   (but the user doesn't get their money back!)
                     */

                    /* Finalise creation of the PBAC resource */
                    pdp.addPolicyRule(resourceID, new PolicyRule(new
MatchPattern(owner), "owner"));
                    pdp.signal(resourceID, "init");
                    success = true;

                    /* Additional setup can also be performed here,
                     * although it is best not to. The difference
                     * is that an exception thrown here will not
                     * destroy the resource; the user can find it
                     * using getResources() even though the creation
                     * operation will appear to have failed.
                     */

                    return generateEPR(resource);
```

```
                } finally {
                        // If we didn't send the 'init' signal then PBAC will forget about us.
                        // Otherwise, operations on the resource can now be executed.
                        pdp.unlock(resourceID);
                }
        } finally {
                if (!success) {
                        log.warn("Failed to init resource - deleting");
                        if (managingConversation instanceof SLAConversation && epr != null) {
                                // Inform the SLA service that we are not using the resource after all
                                pullPoint.addInstantaneousUsageReport(epr, Metric.CURRENT_ACTIVITIES, 0);
                        }
                        deleteHibernatedObject(resource);
                }
        }
    }

    /** Initialise a new resource.
     * If this method throws an exception, then new resource will be destroyed.
     * @param resource the new resource, already persisted with hibernate
     * @throws GridFailureException if the resource should be destroyed
     */
    private void checkCreationOK(licenseResourceBean resource, Conversation managingConversation) throws RemoteException {
        if (managingConversation instanceof TradeAccountConversation) {
            // Account
            TradeAccountConversation account = (TradeAccountConversation) managingConversation;
```

```
                account.checkCreditAvailable(createResourceCharge, "EUR");

            } else if (managingConversation instanceof SLAConversation) {
                // SLA
                GregorianCalendar now = new GregorianCalendar();
                SLAConversation sla = (SLAConversation) managingConversation;
                EndpointReferenceType epr = ConversationID.getEPR(thisServiceAddress, resource.getResourceID(), null);
                sla.startActivity(epr,
                        new Constraint[]{
                            // SLA must allow a new activity
                            new Constraint(Metric.CURRENT_ACTIVITIES, UsageType.INSTANTANEOUS, Bound.EQ, 1, now),
                            // SLA must allow a new license resource
                            new Constraint(METRIC_license_RESOURCES, UsageType.INSTANTANEOUS, Bound.EQ, 1, now),
                        },
                        new UsageReport[] {
                            new UsageReport(epr, Metric.CURRENT_ACTIVITIES, UsageType.INSTANTANEOUS, now, null, 1),
                            new UsageReport(epr, METRIC_license_RESOURCES, UsageType.INSTANTANEOUS, now, null, 1)
                        });
            }
        }

        /** A new resource has been created. Notify management service.
         * If an exception is thrown here, the resource will still exist and be publically
         * accessible.
         */
```

```
        private void recordCreation(licenseResourceBean resource, Conversation managingConversation, IDType user) throws RemoteException {
                if (managingConversation instanceof TradeAccountConversation) {
                        // Account
                        TradeAccountConversation account = (TradeAccountConversation) managingConversation;
                        account.bill(user, createResourceCharge, "EUR", resource.getResourceID(), "Created license resource");

                } else if (managingConversation instanceof SLAConversation) {
                        // SLA
                        SLAConversation sla = (SLAConversation) managingConversation;
                        // Record any extra resource usage here (beyond what was added in startActivity())

                        //EndpointReferenceType epr = ConversationID.getEPR(thisServiceAddress, resource.getResourceID(), null);
                        //pullPoint.addInstantaneousUsageReport(epr, ...);
                }
        }

        public EndpointReferenceType[] getResources() throws GridFailureException {
                return getResources(licenseResource.license_RESOURCE_TYPE, licenseResourceBean.class);
        }

        /** Return notices from the service as an Atom feed.
         * These notices are displayed on the main admin page and can also be polled
         * by the administrator using a news aggregator. The feed should be used to alert the
         * admin to configuration problems and any other important issues that come up.
         */
        public Document getAtomFeed(String atomFeed, String serviceBase) {
```

```
        Document doc = AtomUtils.getEmptyAtomFeed("license Service", "license
Service",
                                                    atomFeed, serviceBase);

        String configProblem = null;

        /* Warn the admin if the service hasn't been configured to trust any account
         * services.
         */
        if (getTrustedAccountServices().length == 0) {
                configProblem = "No trusted management services have been
chosen.";
        }

        if (configProblem != null) {
                AtomUtils.addEntry(doc,
                                thisServiceAddress + "/atom/not-configured",
                                new Date(),
                                configProblem,
                                AtomUtils.divFromXML(doc,
                                        "<div xmlns='" + AtomUtils.XHTML_NS +
"'>" +
                                        "<p>" + configProblem + "</p>" +
                                        "<p>Use the <a href='" + serviceBase +
"'>service " +
                                        "administration tool</a> to set the
configuration.</p>" +
                                        "</div>"));
        }

        return doc;
    }
```

```
	public NotificationMessage[] getMessagesN(int MaximumNumber){
		return pullPoint.getMessagesN(MaximumNumber);
	}

	public NotificationMessage[] getMessages(){
		return pullPoint.getMessages();
	}

	public List<EndpointReferenceType> getAdditionalFederations() {
		return management.getAdditionalFederations();
	}

	public List<String> getAdditionalFederationMethods() {
		return management.getAdditionalFederationMethods();
	}

	public void setAdditionalFederation(EndpointReferenceType[] federations, String[]
methods) throws RemoteException{
		management.setAdditionalFederation(federations,methods);
	}

	public List<String> getFederatedMethods() {
		return Arrays.asList(new String[]{});
	}


}
```

LicenseResourceImpl.java

```
package license.impl;

import license.licenseResource;          // SOAP operations on license resources
import license.licenseService;           // SOAP operations on the service
import license.licenseResourceBean;      // Bean holding resource state

// Client imports - we act as a client to the account and SLA services
import uk.ac.soton.ecs.iam.grid.comms.client.StorableInStateRepository;
import uk.ac.soton.ecs.iam.grid.comms.client.TradeAccountConversation;
import uk.ac.soton.ecs.iam.grid.comms.client.SLAConversation;
import uk.ac.soton.itinnovation.grid.service.types.Metric;

// Standard Java imports
import java.math.BigDecimal;
import org.apache.log4j.Logger;
import org.w3c.dom.Document;
import java.rmi.RemoteException;
import java.io.*;
import java.net.*;
import java.util.*;

// Standard Grid service imports
import org.apache.axis.message.addressing.EndpointReferenceType;
import uk.ac.soton.itinnovation.grid.service.utils.gridservit.GridServiceLite;
import uk.ac.soton.itinnovation.grid.utils.AtomUtils;
import uk.ac.soton.itinnovation.grid.types.ConversationID;
import uk.ac.soton.itinnovation.grid.types.GridErrorException;
import uk.ac.soton.itinnovation.grid.service.types.GridResource;
import uk.ac.soton.ecs.iam.grid.utils.ImplementationFactory;

// Hibernate persistence
import org.hibernate.Session;
```

```
import org.hibernate.Transaction;

import sun.misc.BASE64Encoder;

/** The implementation of licenseService. The mapping from the interface to this class is in the
 * implementationfactory.properties file.
 */
public class licenseResourceImpl extends GridServiceLite implements licenseResource {
	private static Logger log = Logger.getLogger(licenseResourceImpl.class);
	private licenseServiceImpl service = (licenseServiceImpl) ImplementationFactory.getSingletonInstance(licenseService.class);

	public licenseResourceImpl() {
		super("licenseResource");
	}

	/** Get a licenseResourceBean that was stored using hibernate.
	 * Create a hibernate session automatically. This is not suitable
	 * for beans containing lazy collections, which require the session
	 * to still be open when the collection is accessed.
	 * @throws GridErrorException if the resource does not exist
	 */
	private licenseResourceBean getResource(String resourceID) {
		Session session = factory.openSession();
		try {
			return getResource(session, resourceID);
		} finally {
			session.close();
		}
	}
```

```
	@Override
	protected Class<? extends GridResource> getResourceType(String resourceID) {
		return licenseResourceBean.class;
	}

	/** Get a licenseResourceBean that was stored using hibernate.
	 * Use this version if you already have a session (to prevent deadlock) or if
	 * the session must remain open when accessing the bean (because it contains
	 * lazy collections that are only fetched from the database when accessed).
	 * @param session hibernate session to use
	 * @throws GridErrorException if the resource does not exist
	 */
	private licenseResourceBean getResource(Session session, String resourceID) {
		licenseResourceBean resource = (licenseResourceBean) session.get(licenseResourceBean.class, resourceID);
		if (resource == null)
			throw new GridErrorException("Can't find resource '" + resourceID + "'!");
		return resource;
	}

	public void destroy() throws RemoteException {
		String conversationID = getConversationFromContext();

		// Look up the resource
		licenseResourceBean resource;
		try {
			resource = getResource(conversationID);
		} catch (GridErrorException ex) {
			log.error("Failed to get resource! Removing from PBAC anyway.", ex);
			pdp.signal(conversationID, "destroy");
```

```
                throw ex;
            }

            // Notify the SLA manager that usage has finished
            StorableInStateRepository managingConversation =
getManagingConversation(resource);
            if (managingConversation instanceof SLAConversation) {
                // Our use of resources within the SLA drops to zero
                EndpointReferenceType epr =
ConversationID.getEPR(thisServiceAddress, resource.getResourceID(), null);

                service.pullPoint.addInstantaneousUsageReport(epr,
licenseServiceImpl.METRIC_license_RESOURCES, 0.0);
                service.pullPoint.addInstantaneousUsageReport(epr,
Metric.CURRENT_ACTIVITIES, 0.0);
            }

            /* Do any clean-up required here. If an exception is thrown here
             * the resource will still exist and be accessible to the user.
             */

            // Signal to PBAC that the resource is finished
            pdp.signal(conversationID, "destroy");

            // Remove the resource from hibernate
            deleteHibernatedObject(resource);
        }

        /** Used by licenseServiceImpl to when returning EPRs. */
        URL getSoapEndpoint() {
            return thisServiceAddress;
        }
```

```
        /** Return notices from the service as an Atom feed.
         * These notices are displayed on the main admin page and can also be polled
         * by the administrator using a news aggregator. The feed should be used to alert the
         * admin to configuration problems and any other important issues that come up.
         */
        public Document getAtomFeed(String atomFeed, String serviceBase) {
                Document doc = AtomUtils.getEmptyAtomFeed("license Resource", "license Resource",
                                                                atomFeed, serviceBase);
                return doc;
        }


        public int check() throws RemoteException {
                String conversationID = getConversationFromContext();

                Session session = factory.openSession();
                try {
                        licenseResourceBean resource = getResource(session, conversationID);

                        Transaction tx = session.beginTransaction();
                        int newValue = resource.getCheckCounter() + 1;
                        resource.setCheckCounter(newValue);
                        tx.commit();

                        return newValue;
                } finally {
                        session.close();
                }
        }
```

```
	public int add(int a) throws RemoteException {
		String conversationID = getConversationFromContext();

		Session session = factory.openSession();
		try {
			licenseResourceBean resource = getResource(session,
conversationID);

			Transaction tx = session.beginTransaction();
			int newValue = resource.getCheckCounter() + a;
			resource.setCheckCounter(newValue);
			tx.commit();

			return newValue;
		} finally {
			session.close();
		}
	}

	public byte[] addBytes(byte[] array) throws RemoteException
	{
		String conversationID = getConversationFromContext();

		Session session = factory.openSession();
		try {
			licenseResourceBean resource = getResource(session,
conversationID);

			Transaction tx = session.beginTransaction();

			int value = 0;
```

```
            for (int i = 0; i < 4; i++) {
               int shift = (4 - 1 - i) * 8;
               value += (array  [i] & 0x000000FF) << shift;
            }


                        int newValue = resource.getAdded() + value;
                        resource.setAdded(newValue);

                        byte[] b = new byte[4];
            for (int i = 0; i < 4; i++) {
               int offset = (b.length - 1 - i) * 8;
               b[i] = (byte) ((newValue >>> offset) & 0xFF);
            }

                        tx.commit();

                        return b;
               } finally {
                        session.close();
               }
      }

      public String speak() throws RemoteException
      {
            String conversationID = getConversationFromContext();

            Session session = factory.openSession();
            try {
                  licenseResourceBean resource = getResource(session,
conversationID);
```

```
                    Transaction tx = session.beginTransaction();
                    String newValue = resource.getAlreadyTold() + "lew k kamia blakeia!!! ";
                    resource.setAlreadyTold(newValue);
                    tx.commit();


                    return newValue;
                } finally {
                    session.close();
                }
        }

        public int addServerConnection() throws RemoteException
        {
                String conversationID = getConversationFromContext();

                Session session = factory.openSession();
                try {
                    licenseResourceBean resource = getResource(session, conversationID);

                    Transaction tx = session.beginTransaction();
                    int newValue = resource.getAddedUsingServer();

                    newValue = newValue + getRequestedValue();

                    resource.setAddedUsingServer(newValue);
                    tx.commit();

                    return newValue;
                } finally {
```

```
                session.close();
        }
}



private int getRequestedValue()
{
        Socket clientSocket = null;
        InputStream in = null;
        OutputStream out = null;
        int value = 0;


        try {
                System.out.println("Trying to initialize socket");
                clientSocket = new Socket("127.0.0.1", 27000);

                System.out.println("Trying to initialize streams");
                in = clientSocket.getInputStream();
                out = clientSocket.getOutputStream();
                System.out.println("Streamings initiliased");

                out.flush();
                byte[] array = new byte[2048];

                System.out.println("Trying to read data");
                in.read(array);
                System.out.println("Data Read");

                for (int i = 0; i < 4; i++) {
                        int shift = (4 - 1 - i) * 8;
```

```
                    value += (array[i] & 0x000000FF) << shift;
                }

                System.out.println("Result: " + value);
            } catch(IOException ex) {
                System.out.println(ex.getMessage());
            }
            finally{
                try{
                    clientSocket.close();
                } catch(IOException ex) {
                    System.out.println("Cannot close socket. " +
ex.getMessage());
                }
            }

        return value;
        }

        public String transferMessage(int port, byte[] message, int len) throws
RemoteException
        {
            String result = null;
            String conversationID = getConversationFromContext();

            Session session = factory.openSession();

            try {

                licenseResourceBean resource = getResource(session,
conversationID);
```

```
                Transaction tx = session.beginTransaction();

                resource.setPort(port);

                String url = _getUrl();

                if((url == null) || (url == ""))
                {
                        System.out.println("ERROR: Could not find FLEXlm IP");
                }

                result = _transferMessage(url, port, message, len);

                resource.setUrl(url);

                tx.commit();

        } finally {
                session.close();
        }

        return result;
}

private String _transferMessage(String url, int port, byte[] message, int len)
{
        Socket clientSocket = null;
        InputStream in = null;
        OutputStream out = null;
        String result = null;

        try {
```

```
                System.out.println("Trying to initialize socket at " + url + " - port: " +
port);
                clientSocket = new Socket(url, port);

                System.out.println("Trying to initialize streams");
                in = clientSocket.getInputStream();
                out = clientSocket.getOutputStream();
                System.out.println("Streamings initiliased");

                out.flush();

                System.out.println("Trying to send data" + len + " - " +
message.length);
                out.write(message, 0, len);

                byte[] array = new byte[2048];
                System.out.println("Trying to read date from server");
                int count = in.read(array);

                byte[] array8bit = new byte[count];
                for(int j = 0; j < count; j++){
                        array8bit[j] = array[j];
                }
                BASE64Encoder en = new BASE64Encoder();
                result = en.encode(array8bit);

                System.out.println("Result: " + result);

        } catch(IOException ex) {
                System.out.println(ex.getMessage());
        } finally{
                try{
```

```
                        clientSocket.close();
                } catch(IOException ex) {
                        System.out.println("Cannot close socket. " +
ex.getMessage());
                }
        }

    return result;
        }

        public String getServerURL() throws RemoteException {
                String conversationID = getConversationFromContext();

                Session session = factory.openSession();
                try {

                        licenseResourceBean resource = getResource(session,
conversationID);

                        Transaction tx = session.beginTransaction();

                        _getUrl();

                        String url = resource.getUrl();

                        if((url == null) || (url == ""))
                        {
                                url = _getUrl();
                        }

                        if((url == null) || (url == ""))
                        {
```

```
                System.out.println("ERROR: Could not find FLEXlm IP");
            }
            else
            {
                System.out.println("FLEXlm's IP: " + url);
            }

            resource.setUrl(url);

            tx.commit();

            return url;

        } finally {
            session.close();
        }
    }

    private String _getUrl()
    {
        String result = null;

        try
        {
            FileInputStream inputstream = new FileInputStream("C:\\Tomcat
6.0\\conf\\LicenseService.config");
            DataInputStream datastream = new DataInputStream(inputstream);
            BufferedReader bufferin = new BufferedReader(new
InputStreamReader(datastream));

            if((result  = bufferin.readLine())!=null)
                datastream.close();
```

```
                }
                catch(IOException ex) {
                        System.out.println(ex.getMessage());
                }

                for(int i=0; i<50; i++)
                        System.out.println("FLEXlm's IP: " + result + " and port is " +
result);

                return result;

        }

}
```

LicenseResourceBean.java

```
package license;

import uk.ac.soton.ecs.iam.grid.comms.client.Conversation;
import uk.ac.soton.itinnovation.grid.service.types.GridResource;

import java.lang.*;
import java.net.*;
import java.io.*;

/** This is a Java bean representing a resource. New resources are created using
 * the "createlicenseResource" SOAP operation. They are persisted using hibernate.
 */
public class licenseResourceBean extends GridResource {
        private int checkCounter = 0;
```

```
private int added = 0;
private int addedUsingServer = 0;
private String alreadyTold = "";

private String url = "";
private int port = 0;

/* Constructor used by hibernate */
public licenseResourceBean() {
}

/* Constructor used by service */
public licenseResourceBean(Conversation managingConversation, String label) {
        setLabel(label);

        if (managingConversation != null)
                setManagementResource(managingConversation.getEndpointRef());
}

public String getResourceType() {
        return licenseResource.license_RESOURCE_TYPE;
}

/* Add getters and setters for your resource's fields here.
 * You'll also need to add them to the .hbm.xml file to get
 * them persisted.
 */


public Integer getCheckCounter() {
        return checkCounter;
}
```

```
public void setCheckCounter(Integer checkCounter) {
	if (checkCounter == null)
		checkCounter = 0;
	this.checkCounter = checkCounter;
}

public String getAlreadyTold()
{
	return alreadyTold;
}

public void setAlreadyTold(String str)
{
	if(alreadyTold == null)
		alreadyTold = "";
	this.alreadyTold = str;
}

public Integer getAdded()
{
	return added;
}

public void setAdded(Integer added)
{
	if (added == null)
		added = 0;
	this.added = added;
}

public Integer getAddedUsingServer()
```

```
{
	return addedUsingServer;
}

public void setAddedUsingServer(Integer addedUsingServer)
{
	if(addedUsingServer == null)
		addedUsingServer = 0;
	this.addedUsingServer = addedUsingServer;
}

public String getUrl()
{
	return this.url;
}

public void setUrl(String url)
{
	if (url == null)
		this.url = "";
	this.url = url;
}

public Integer getPort()
{
	return this.port;
}

public void setPort(Integer port)
{
	if (port == null)
		port = 0;
```

```
            this.port = port;
        }

}
```

LicenseResourceBean.hbm.xml

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

<hibernate-mapping>
        <class name="license.licenseResourceBean"
table="licenseService_licenseResource">
            <id name="resourceID">
                <generator class="uuid">
                    <param name="separator">-</param>
                </generator>
            </id>
            <property name="label"/>
            <property name="checkCounter"/>
            <property name="added"/>
            <property name="alreadyTold"/>
            <property name="addedUsingServer"/>
            <property name="url"/>
            <property name="port"/>
            <property name="managementResource"
type="uk.ac.soton.itinnovation.grid.service.types.EPRUserType" length="2048"/>
    </class>
</hibernate-mapping>
```

Γ CreateNewTradeAccount

CreateNewTradeAccount.java

```
import java.rmi.RemoteException;
import java.util.ArrayList;

import javax.xml.messaging.Endpoint;
import javax.xml.ws.Dispatch;

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

import uk.ac.soton.ecs.iam.grid.client.swing.SwingInputHandler;
import uk.ac.soton.ecs.iam.grid.comms.client.AxisTransport;
import uk.ac.soton.ecs.iam.grid.comms.client.RemoteDataService;
import uk.ac.soton.itinnovation.grid.client.engine.impl.DefaultAttributeSelector;
import uk.ac.soton.itinnovation.grid.client.engine.impl.DefaultFederationSelector;
import uk.ac.soton.itinnovation.grid.client.engine.impl.DefaultInvocationEngine;
import uk.ac.soton.itinnovation.grid.client.engine.impl.LocalRegistry;
import uk.ac.soton.itinnovation.grid.client.plugins.GridClientPluginManager;
import uk.ac.soton.itinnovation.grid.client.proxy.HelperProxyFactory;
import uk.ac.soton.itinnovation.grid.client.proxy.InvocationEngineProxyFactory;
import uk.ac.soton.itinnovation.grid.client.proxy.ProxyFactory;
import uk.ac.soton.itinnovation.grid.comms.account.TradeAccountResource;
import uk.ac.soton.itinnovation.grid.comms.account.TradeAccountService;
import uk.ac.soton.itinnovation.grid.comms.client.ITInnovSocketFactory;
import uk.ac.soton.itinnovation.grid.comms.client.InteractiveX509TrustManager;
import uk.ac.soton.itinnovation.grid.comms.client.SAMLTokenCache;
import uk.ac.soton.itinnovation.grid.comms.client.WSDLCache;
import uk.ac.soton.itinnovation.grid.comms.client.WSDLCacheImpl;
import uk.ac.soton.itinnovation.grid.comms.sla.SLAResource;
```

```
import uk.ac.soton.itinnovation.grid.service.types.AddressType;
import uk.ac.soton.itinnovation.grid.service.types.SimpleRegistry;
import uk.ac.soton.itinnovation.grid.types.ConversationID;
import uk.ac.soton.itinnovation.grid.types.MatchPattern;
import uk.ac.soton.itinnovation.grid.types.MatchRule;
import uk.ac.soton.itinnovation.grid.types.SubjectDescription;
import uk.ac.soton.itinnovation.grid.utils.CertificateTrustValidator;
import uk.ac.soton.itinnovation.grid.utils.IdentityProvider;
import uk.ac.soton.itinnovation.grid.utils.TestIdentityProvider;
import uk.ac.soton.itinnovation.grid.utils.KeystoreIdentityProvider;
import uk.ac.soton.itinnovation.grid.utils.Transport;

import license.licenseResource;
import license.licenseService;



public class licenseClient {

        ProxyFactory proxyFactory = null;
        DefaultInvocationEngine engine = null;
        AxisTransport transport = null;


        public static void main(String[] args) {

                licenseClient client = new licenseClient();
        }

        @SuppressWarnings("deprecation")
        public licenseClient()
        {
```

```
System.out.println("Starting ...");
// Create my identity
KeystoreIdentityProvider idp = new KeystoreIdentityProvider(new ReadConfigurationHelper().readPublicKeyPath(), "service", "admin".toCharArray());
System.out.println("KeystoreIdentifier created");

// Create transport object which invokes operations
transport = new AxisTransport(idp);
System.out.println("Kyestore added to AxisTransport");

// We need a cache to store our wsdl files
WSDLCache cache = new WSDLCacheImpl(transport);
System.out.println("WSDLCache created");

/* When we communicate with services we need to choose which
 * services we trust, we do this using a TrustValidator
 * This example uses a graphical validator InteractiveX509TrustManager
 * but a non interative TestIdentityProvider.TRUST_EVERYONE
 * can be used but only for testing. It should NOT be used a
 * real deployment.
 */
CertificateTrustValidator manager = new InteractiveX509TrustManager(new SwingInputHandler());
transport.setCertificateTrustValidator(manager);
ITInnovSocketFactory.setHandlers(idp, manager);


// Define a registry where we will store EndpointReference's of services and resources
ArrayList<SimpleRegistry> registries = new ArrayList<SimpleRegistry>();
registries.add(new LocalRegistry());
```

```
// Define a cache for any tokens we get from Membership Groups
SAMLTokenCache tokenCache = new SAMLTokenCache();

// Load the GRIA plugins.
GridClientPluginManager pluginManager = new GridClientPluginManager();
pluginManager.loadPlugins();



// Define an engine which uses the transport to invoke operations
engine = new DefaultInvocationEngine(transport);

// Add the registry to the engine
engine.setSelectedRegistries(registries);

// Tell the engine how to find tokens by defining an
//attribute selector, which stores tokens in the token cache
engine.setAttributeSelector(new DefaultAttributeSelector(tokenCache));

// Tell the engine to use a federation selector, which finds
//management context (SLAs, Trade Accounts) when invoking an operation.
engine.setFederationSelector(new DefaultFederationSelector());

/* The proxy factory creates proxy object from
 * EndpointReference's which help us communicate with
 * the services. The helper proxy factory adds some additional
 * useful methods
 */
ProxyFactory rawProxyFactory = new
InvocationEngineProxyFactory(engine, cache);
proxyFactory = new HelperProxyFactory(rawProxyFactory,
pluginManager.getHelperRegistry());
engine.setProxyFactory(proxyFactory);
```

```
//OPEN AN ACCOUNT


TradeAccountService tradeAccountService = null;
try
{
        //Get a Proxy to the corresponding resource
        SubjectDescription subjectaccount = new SubjectDescription(
transport.getSubjectCert());
        SubjectDescription issueraccount = new SubjectDescription(
transport.getIssuerCert());
        MatchPattern pattern = new MatchPattern(subjectaccount,
issueraccount);

        String urlAccountService = "https://147.102.19.107:8443/gria-
service-provider-mgt/services/TradeAccountService";
        EndpointReferenceType serviceAccountService =
ConversationID.getEPR(urlAccountService);
        System.out.println("Trying to create proxy to license Account
Service");
        tradeAccountService =
proxyFactory.createProxy(serviceAccountService, TradeAccountService.class);
        System.out.println("Account service Proxy created succesfully!\n");

        int countaccount = (tradeAccountService.getResources()).length;
        System.out.println("getResources Account- Count: " +
countaccount);

        EndpointReferenceType resourceAccountERP = null;
```

```
                resourceAccountERP =
tradeAccountService.openAccount("user107", "45634535", "user107@mail.ntua.gr", new
AddressType(), "user107", "user107", new MatchRule(pattern,"budget-holder",false));

            } catch (RemoteException e) {
                e.printStackTrace();
            }


        }


}
```