Ατομική Διπλωματική Εργασία

ΣΧΕΔΙΑΣΜΟΣ ΚΑΙ ΥΛΟΠΟΙΗΣΗ MARKETPLACE ΛΟΓΙΣΜΙΚΟΥ ΒΑΣΙΣΜΕΝΟ ΣΤΟ REPOSITORY SONATYPE NEXUS ΓΙΑ ΤΟ CAMF

Αθανάσιος Τρύφωνος

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2016

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Σχεδιασμός Και Υλοποίηση Marketplace Λογισμικού Βασισμένο Στο Repository Sonatype Nexus Για Το CAMF

Αθανάσιος Τρύφωνος

Επιβλέπων Καθηγητής
Μάριος Δ. Δικαιάκος

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2016

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον καθηγητή κύριο Μάριο Δικαιάκο που μου έκανε την τιμή και με επέλεξε να συνεργαστούμε στα πλαίσια αυτής της Διπλωματικής εργασίας. Δουλεύοντας μαζί του διεύρυνα τις γνώσεις μου, όχι μόνο στο συγκεκριμένο θέμα που πραγματεύεται η εργασία μου αλλά και σε τομείς που θα μου φανούν χρήσιμοι στην μετέπειτα επαγγελματική μου σταδιοδρομία.

Επίσης θα ήθελα να ευχαριστήσω ιδιαίτερα το διδακτορικό φοιτητή Δημήτρη Τριχινά και τον δρ. Νικόλα Λουλλούδη για την πολύτιμη και καθοριστική καθοδήγηση που μου παρείχαν κατά τη διάρκεια εκπόνησης της Διπλωματικής μου εργασίας.

Τέλος θα ήθελα να ευχαριστήσω το οικογενειακό και στενό φιλικό περιβάλλον μου για την υποστήριξη που μου παρείχαν σε όλους τους τομείς καθόλη τη διάρκεια των σπουδών μου.

Περίληψη

Στα πλαίσια της διπλωματικής αυτής εργασίας αναπτύχθηκε το Cloud Software Marketplace, το οποίο είναι ένα plugin για την πλατφόρμα Eclipse, με σκοπό την προσθήκη λειτουργίας software repository στο CAMF.

Το CAMF είναι ένα Cloud Application Management Framework στο οποίο δημιουργούνται περιγραφές εφαρμογών Cloud μέσω του γραφικού περιβάλλοντος του για ανάπτυξή τους στους διάφορους παρόχους.

Πριν την εισαγωγή του plugin, η διαδικασία με την οποία οι χρήστες εισήγαγαν τις εφαρμογές τις οποίες θα αναπτύξουν στο Cloud, γινόταν με μη αυτοματοποιημένο τρόπο. Με την προσθήκη του Marketplace, οι εφαρμογές πλέον, φυλάσσονται σε ένα Nexus repository, με κάποια συγκεκριμένη δομή, στο οποίο ο χρήστης έχει τη δυνατότητα να κάνει αναζήτηση και να τις κατεβάσει χρησιμοποιώντας κλήσεις από το REST API που παρέχει το Nexus. Στη συνέχεια ο χρήστης, χρησιμοποιεί το γραφικό περιβάλλον διεπαφής του plugin για να ορίσει τις τιμές που θέλει να δώσει στις παραμέτρους ρύθμισης (configuration) της εφαρμογής και να την εισάγει στον editor από την παλέτα. Η περιγραφή αυτή μεταφράζεται σε γλώσσα προτύπου TOSCA και μπορεί να προωθηθεί στον πάροχο για ανάπτυξη.

Επίσης, το plugin παρέχει τη δυνατότητα και πάλι μέσω του γραφικού του περιβάλλοντος, στους developers και διαχειριστές του Marketplace, να δημιουργήσουν και να ανεβάσουν νέα εκθέματα λογισμικού στο Nexus repository ή ακόμα και να διαγράψουν εφαρμογές οι οποίες φυλάσσονται ήδη στο repository, κάνοντας και πάλι χρήση των κλήσεων του REST API του Nexus.

Τέλος, έχει ληφθεί πρόνοια υλοποίησης, σε μελλοντικό στάδιο, των λειτουργιών του Cloud Software Marketplace για διαφορετικού τύπου repository από το Nexus, όπως για παράδειγμα το Artifactory της jFrog και το Apache Archiva.

Αθανάσιος Τρύφωνος – Πανεπιστήμιο Κύπρου, Μάιος 2016

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Εισαγωγή

Ολοένα και περισσότερο στις μέρες μας, η χρήση των υπηρεσιών Cloud αυξάνεται. Από απλές εφαρμογές από απλούς καθημερινούς χρήστες, μέχρι τις πιο πολύπλοκες των μεγάλων πολυεθνικών και οργανισμών, οι υπηρεσίες Cloud παρέχουν τις κατάλληλες υποδομές στις οποίες οι εφαρμογές αυτές μπορούν να τρέξουν, καθόλη τη διάρκεια του κύκλου ζωής τους (από τη φάση της ανάπτυξης της εφαρμογής μέχρι τη φάση συντήρησης του έτοιμου προϊόντος) ελαχιστοποιώντας το κόστος λειτουργίας.
Παρόλο που το Cloud, με τις υπηρεσίες που προσφέρει έχει καταφέρει να διευκολύνει κατά πολύ τη ζωή των χρηστών του, δεν παύει να υστερεί σε κάποια σημεία, δυσκολεύοντας το έργο κυρίως των developers.
Ο ρόλος των developers είναι να αναπτύσσουν ποιοτικά καλό, εύκολα συντηρήσιμο και εύχρηστο λογισμικό. Στην εποχή του Cloud οι developers έπρεπε να προσαρμοστούν στο να αναπτύσσουν λογισμικό βάση των επιταγών του μοντέλου ανάπτυξης και περιγραφής εφαρμογών του Cloud. Ακόμη, η ύπαρξη πολλών και διαφορετικών παρόχων υπηρεσιών Cloud, που ο καθένας απαιτεί διαφορετικό μοντέλο και περιγραφή της αναπτυσσόμενης εφαρμογής, δυσκόλευε τη μεταφορά της εφαρμογής από τον ένα πάροχο στον άλλο (portability) και την επαναχρησιμοποίηση λογισμικού, κλειδώνοντας τον developer στη χρήση ενός συγκεκριμένου παρόχου. Έτσι οι developers, εστιάζουν

μεγάλο μέρος του χρόνου τους στο σχεδιασμό και ανάπτυξη εφαρμογών για έναν συγκεκριμένο πάροχο.

Για την επίλυση των προβλημάτων της φορητότητας των εφαρμογών από έναν πάροχο σε κάποιον άλλο και της επαναχρησιμοποίησης λογισμικού σε Cloud εφαρμογές, αναπτυχθήκαν αρκετά εργαλεία. Τα εργαλεία αυτά ονομάζονται *Cloud Application Management Frameworks* και σκοπός τους είναι να διευκολύνουν τους developers, με ελάχιστο κόπο και σε ελάχιστο χρόνο, μέσω αυτοματοποιημένων διαδικασιών να γράφουν περιγραφές Cloud εφαρμογών. Κύριο χαρακτηριστικό τους είναι η δυνατότητα ανάπτυξης των εφαρμογών σε σχεδόν όλους τους διαθέσιμους Cloud Providers, διευκολύνοντας έτσι και την επαναχρησιμοποίηση του λογισμικού.

Μερικά από τα πιο γνωστά Cloud Application Management Frameworks είναι το Juju Charms[6], το CloudFormation[7] της Amazon, το Service Mesh[8] και το Winery[1]. Όλα τα Application Management Frameworks (AMF), που αναφέρθηκαν πιο πριν, υποστηρίζουν περιγραφή των εφαρμογών Cloud μέσω αυτοματοποιημένων εργασιών χρησιμοποιώντας κάποιο γραφικό περιβάλλον διεπαφής, προσφέρουν ανεξαρτησία παρόχου, μπορουν δηλαδή να αναπτύσσουν εφαρμογές Cloud σε σχεδόν όλους τους παρόχους (πλην του CloudFormation που είναι αποκλειστικά για την Amazon) και παρέχουν κάποιο είδος repository, από το οποίο οι χρήστες μπορούν να χρησιμοποιήσουν έτοιμα εκθέματα λογισμικού στις περιγραφές τους ρυθμίζοντας τα βάση των αναγκών τους (επαναχρησιμοποίηση λογισμικού).

Ακόμα ένα Cloud Application Management Framework, ιδιαίτερα σημαντικό για την εκπόνηση αυτής της διπλωματικής, είναι το CAMF[2,4]. Το CAMF περιέχει όλα τα σημαντικά χαρακτηριστικά ενός Application Management Framework, όπως αυτά παρουσιάστηκαν εν συντομία πιο πάνω. Εκεί που υστερεί σε σύγκριση με τα υπόλοιπα παρόμοια προγράμματα του είδους του είναι η έλλειψη υποστήριξης software repository που επιτρέπει την εύκολη εισαγωγή και επαναχρησιμοποίηση λογισμικού στις περιγραφές εφαρμογών Cloud. Μέχρι τώρα η διαδικασία επαναχρησιμοποίησης λογισμικού στις περιγραφές εφαρμογών Cloud γινόταν με μη αυτοματοποιημένο τρόπο απαιτώντας αρκετά βήματα αλληλεπίδρασης με τον χρήστη.

Τέλος, ακόμα μία πολύ σημαντική έννοια για την εκπόνηση αυτής της διπλωματικής είναι η έννοια του software repository. Με τον όρο software repository, αναφερόμαστε σε ειδικού τύπου εξυπηρετητές αρχείων, στους οποίους φυλάσσονται εκθέματα λογισμικού ή συστατικά στοιχεία αυτών και προσφέρουν στους χρήστες δυνατότητες

αναζήτησης ευρετηρίου (search index), κατεβάσματος των εκθεμάτων και διαχείρισης του repository με ενέργειες όπως ανέβασμα νέου ή νέας έκδοσης εκθέματος λογισμικού και διαγραφής εκθεμάτων λογισμικού.

1.2 Συνεισφορά

Σκοπός αυτής της ΑΔΕ είναι η επέκταση της λειτουργικότητας του CAMF με τον σχεδιασμό και υλοποίηση ενός software marketplace, χρησιμοποιώντας το repository ανοιχτού κώδικα OSS Sonatype Nexus[3]. Η ενσωμάτωση και ενδοεπικοινωνία του software marketplace με το CAMF γίνεται μέσω της ολοκληρωμένης πλατφόρμας ανάπτυξης λογισμικού Eclipse, στην οποία το marketplace αναπτύσσεται σαν plugin. Η λειτουργικότητα του marketplace είναι σχεδιασμένη κατά τέτοιο τρόπο ώστε να υποστηρίζει δύο τύπους χρηστών: (i) Απλούς χρήστες και (ii) διαχειριστές repository και developers, με λειτουργίες όπως:

- Αναζήτηση εκθεμάτων λογισμικού βάση keywords
- Κατέβασμα του εκθέματος από το marketplace.
- Υποστήριξη αναζήτησης και κατεβάσματος εκθεμάτων από διάφορα repositories
- Ρύθμιση εκθέματος λογισμικού που θα χρησιμοποιηθεί (configuration)
- Δημιουργία νεών artifacts
- Ανέβασμα νέου artifact στο marketplace.
- Διαγραφή artifact από το marketplace.

οι οποίες συνοδεύονται από τα κατάλληλα μηνύματα σφαλμάτων και ενημέρωσης προόδου των εργασιών που επιτελεί ο χρήστης.

1.3 Δομή Εγγράφου

Στο Κεφάλαιο 2 παρουσιάζεται η μελέτη σχετικής εργασίας με εκτενή αναφορά στα χαρακτηριστικά και την αρχιτεκτονική του CAMF, σύντομη περιγραφή των χαρακτηριστικών παρόμοιου λογισμικού, ανάλυση της έννοιας software repository και σύγκριση των διαθέσιμων repository της αγοράς.

Στο Κεφάλαιο 3 παρουσιάζεται ο αρχιτεκτονικός σχεδιασμός του marketplace plugin, το μοντέλο το οποίο πρέπει να ακολουθούν τα artifacts τα οποία θα ανεβαίνουν στο repository και το API του plugin.
Στο Κεφάλαιο 4 παρουσιάζεται η λειτουργικότητα του plugin μέσω διαγραμματικών use case και αναλυτική περιγραφή τους και πώς το plugin χειρίζεται τις περιπτώσεις σφαλμάτων.

Στο Κεφάλαιο 5 γίνεται ο έλεγχος και η αξιολόγηση του plugin.

Και τέλος στο Κεφάλαιο 6 παρουσιάζονται τα συμπεράσματα και η μελλοντική δουλεία που μπορεί να γίνει για την βελτίωση και ανάπτυξη του plugin.

Κεφάλαιο 2

Background Section and Related Work

Σε αυτό το κεφάλαιο, γίνεται μια αναλυτική περιγραφή των λειτουργιών και της αρχιτεκτονικής του CAMF. Επιπρόσθετα, παρουσιάζεται το πρότυπο TOSCA, το οποίο παρέχει τη γλώσσα την οποία χρησιμοποιεί το CAMF και πολλά άλλα παρόμοια με το CAMF Application Management Frameworks (AMF) για το Cloud για την περιγραφή της τοπολογίας και της ενορχήστρωσης υπηρεσιών και εφαρμογών Cloud. Στη συνέχεια, παρουσιάζονται τα AMF και τα χαρακτηριστικά τους, τα οποία παρέχουν παρόμοια λειτουργικότητα με το CAMF και θεωρούνται ανταγωνιστές του. Τελειώνοντας το Κεφάλαιο 2, γίνεται μία σύγκριση των διαθέσιμων Software repositories, καθώς και γιατί επιλέχθηκε για τους σκοπούς της υλοποίησης του Software repository για το CAMF το λογισμικό ανοικτού κώδικα OSS Sonatype Nexus.

2.1 Cloud Application Management Framework (CAMF)

Με τον όρο Application Management Framework, αναφερόμαστε σε κομμάτια λογισμικού τα οποία εξειδικεύονται στο να παρέχουν στους χρήστες τη δυνατότητα να αναπτύσσουν και να δεσμεύουν στην υποδομή των παρόχων cloud (cloud providers), περιγραφές των υπολογιστικών πόρων που χρειάζονται για τις ανάγκες των εφαρμογών

τους. Η διαδικασία αυτή είναι κατά κύριο λόγο αυτοματοποιημένη και γίνεται εύκολα χρησιμοποιώντας κάποιο γραφικό περιβάλλον διεπαφής (GUI), αν υποστηρίζεται από το AMF, ή μέσω κάποιας γλώσσας μοντελοποίησης. Στη συνέχεια το AMF, παίρνοντας την περιγραφή αυτή της μοντελοποίησης από το χρήστη, αναλαμβάνει να την προωθήσει στον πάροχο για ανάπτυξη. Έτσι λοιπόν, οι χρήστες είναι σε θέση να εστιάζουν τις προσπάθειες τους στην ανάπτυξη και βελτίωση των εφαρμογών τους παρά στις αλλαγές που θα έπρεπε να κάνουν στην περιγραφή της εφαρμογής τους για την υποδομή συγκεκριμένου παρόχου cloud.

Το CAMF είναι ένα σύγχρονο Application Management Framework το οποίο ενσωματώνει όλα τα βασικά χαρακτηριστικά που περιέχει ένα AMF, όπως αυτά που περιεγράφησαν πιο πάνω.

Αναλυτικότερα, το CAMF είναι ένα AMF ανοικτού κώδικα, το οποίο είναι χτισμένο πάνω στην πλατφόρμα Eclipse σαν plugin. Αυτό σημαίνει ότι το CAMF εκμεταλλεύεται πλήρως τις δυνατότητες επέκτασης που παρέχει η πλατφόρμα του Eclipse δηλώνοντας και υλοποιώντας Extensions και Extension points.

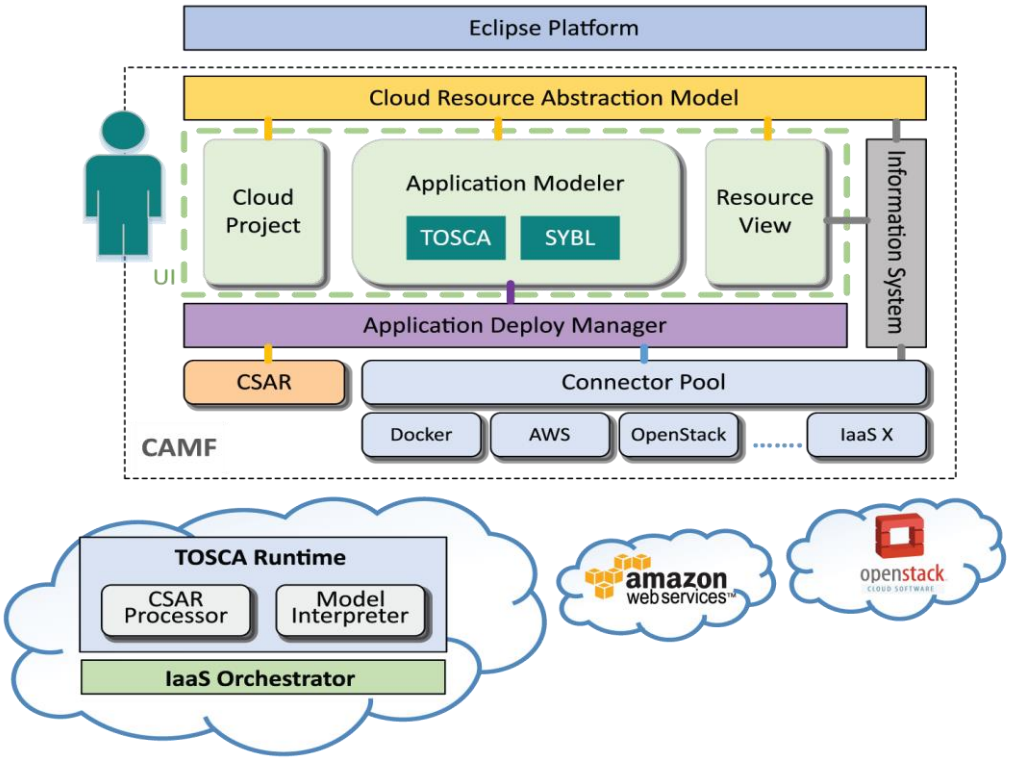
Το κύριο συστατικό του CAMF είναι το Application Modeling Tool, το οποίο διευκολύνει την περιγραφή εφαρμογών για cloud χρησιμοποιώντας το πρότυπο TOSCA[5]. Χρησιμοποιώντας την παλέτα, ο χρήστης μπορεί να εισάγει στην περιγραφή του, διάφορα συστατικά στοιχεία εφαρμογών cloud όπως deployments scripts, user applications, key pairs, virtual images κ.α. κάνοντας απλά drag and drop στον καμβά και το Application Modeling Tool είναι υπεύθυνο να κάνει τη γραφική αναπαράσταση του μοντέλου της περιγραφής ενώ ταυτόχρονα στο παρασκήνιο μεταφράζει την γραφική αυτή αναπαράσταση στη γλώσσα προτύπου TOSCA. Η μετάφραση της γραφικής αναπαράστασης σε πρότυπο TOSCA γίνεται με τη βοήθεια της βιβλιοθήκης του Eclipse, Graphiti, η οποία είναι βασισμένη στο Eclipse Modeling Framework (EMF)[11].

Για να δημιουργηθεί μια νέα περιγραφή εφαρμογών cloud στο CAMF, ο χρήστης πρέπει να ξεκινήσει ένα νέο cloud project στο Eclipse, με τον ίδιο τρόπο που ξεκινάει κάθε project στο Eclipse. Κάθε cloud project αποτελείται από τους εξής τέσσερις φακέλους: (i) Application Descriptions που είναι ο φάκελος που περιέχει την περιγραφή TOSCA, (ii) Application Submissions στον οποίο περιέχονται λεπτομέρειες σχετικά με το deployment των εφαρμογών, (iii) Artifacts που είναι ο φάκελος που περιέχει τα

αρχεία τα οποία αναφέρονται στην περιγραφή της εφαρμογής και τέλος (iv) Monitoring που περιέχει δεδομένα παρακολούθησης του συστήματος κατά τη φάση της ανάπτυξης.

Κατά τη δημιουργία της περιγραφής εφαρμογών, ο χρήστης έχει τη δυνατότητα να ορίσει πολιτικές ελαστικότητας για κάθε στοιχείο της περιγραφής του. Η εισαγωγή των πολιτικών ελαστικότητας γίνεται στο ειδικό tab στο view του CAMF με ονομασία Properties. Το CAMF υποστηρίζει δύο τύπους πολιτικών ελαστικότητας, σύμφωνα με τη γλώσσα SYBL. Οι τύποι πολιτικών αυτών είναι: (i) ο τύπος Περιορισμών ελαστικότητας που εκφράζει περιορισμούς σε σχέση με το κόστος, την απόδοση και άλλες μετρικές που αφορούν ποιοτικά τις εφαρμογές και (ii) ο τύπος Στρατηγικών ελαστικότητας που εκφράζει στρατηγικές που πρέπει να επιβάλλονται στο περιβάλλον εκτέλεσης όταν κάποιοι περιορισμοί παραβιαστούν.

Τέλος, όταν η περιγραφή του χρήστη έχει ολοκληρωθεί και είναι έτοιμη να υποβληθεί για ανάπτυξη, η περιγραφή TOSCA μαζί με όλα τα σχετικά αρχεία συμπιέζονται σε ένα αρχείο CSAR από τον CSAR Exporter και προωθούνται από το CAMF στο περιβάλλον επεξεργασίας TOSCA περιγραφών του παρόχου (TOSCA Container). Στο Σχήμα 2.1 φαίνεται η σχηματική αναπαράσταση της αρχιτεκτονικής του CAMF που περιγράφηκε πιο πάνω.



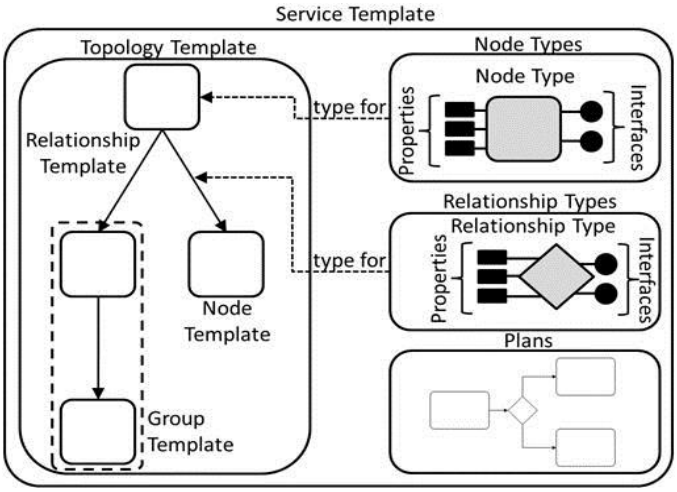
Σχήμα 2. 1 Αρχιτεκτονική CAMF

2.2 OASIS TOSCA

Ιδιαίτερη αναφορά στην προηγούμενη ενότητα έγινε στο πρότυπο TOSCA. Το TOSCA είναι ένα πρότυπο το οποίο παρέχει τη γλώσσα περιγραφής της δομής των συστατικών στοιχείων των εφαρμογών, των κύριων δράσεων διαχείρισης αυτών καθώς και των συσχετίσεων τους, χρησιμοποιώντας περιγραφές για την τοπολογία των υπηρεσιών (service topology).

Κύριες έννοιες στην ορολογία του TOSCA είναι (i) τα Topology Templates τα οποία χρησιμοποιούνται για να περιγράψουν τη δομή μιας υπηρεσίας και (ii) τα Plans τα οποία καθορίζουν ένα μοντέλο, βάση του οποίου δημιουργείται, τερματίζεται και διαχειρίζεται μια υπηρεσία καθόλη τη διάρκεια του κύκλου ζωής της. Στο Σχήμα 2.2 φαίνονται τα δομικά στοιχεία μιας περιγραφής TOSCA.

Τα Topology Templates αποτελούνται κατά κύριο λόγο από ένα σύνολο από κόμβους, τα Node Templates και τις συσχετίσεις τους, τα Relationship Templates τα οποία μαζί δημιουργούν έναν κατευθυνόμενο γράφο. Τα Node Templates καθορίζουν την ύπαρξη Node Types, τα οποία ορίζουν τις ιδιότητες των συστατικών στοιχείων καθώς και τις διαθέσιμες λειτουργίες για τη διαχείριση τους. Τα Node Types, επίσης υποδηλώνουν τις εξαρτήσεις και τις συνεισφορές κάθε συστατικού στοιχείου προς άλλα συστατικά στοιχεία (Requirements and Capabilities) καθώς και τα εκθέματα λογισμικού που η περιγραφή αυτή αναπτύσσει (Deployment Artifacts). Αντίστοιχα τα Relationship Templates καθορίζουν την ύπαρξη σχέσεων μεταξύ κόμβων και των ιδιοτήτων τους σε ένα Topology Template.



Σχήμα 2. 2 Δομικά στοιχεία μιας TOSCA περιγραφής

Δυνατά σημεία του TOSCA είναι η δυνατότητα που παρέχει για φορητότητα των εφαρμογών μεταξύ των υποστηριζόμενων παρόχων, καθώς όλη η απαιτούμενη λογική διαχείρισης του κύκλου ζωής των εφαρμογών αλλα και τα ίδια τα εκτελέσιμα αρχεία και scripts περιέχονται στο παραγόμενο CSAR.

2.3 Ανταγωνιστές του CAMF

Σε αυτό το σημείο αξίζει να δούμε συγκριτικά τα χαρακτηριστικά μερικών από τα διαθέσιμα AMF, που υπάρχουν στην αγορά, για δημιουργία Cloud εφαρμογών και υποστηρίζουν λειτουργικότητα παρόμοια με αυτή του CAMF. Πιο συγκεκριμένα θα παρουσιαστούν τα βασικά χαρακτηριστικά των Juju Charms, CloudFormation της Amazon, ServiceMesh της CSC και τέλος του Winery.

Χαρακτηριστικά	**Juju**	**CloudFormation**	**ServiceMesh**	**Winery**	**CAMF**
GUI	Yes	Yes	Yes	Yes	Yes
Provider Independent	Yes	No	Yes	Yes	Yes
Platform Independent	Yes	Yes	Yes	Yes	Yes
Elasticity Policies	No	Yes	No	No	Yes
Monitoring	Yes	Yes	Yes	No	Yes
Deployment	Yes	Yes	Yes	No	Yes
Marketplace	Yes	No	Yes	No	No

Πίνακας 2.1 Χαρακτηριστικά AMF

Όπως προκύπτει από τον Πίνακα 2.1 το σημείο στο οποίο υστερεί το CAMF έναντι των υπολοίπων AMF, είναι η έλλειψη υποστήριξης λειτουργιών marketplace, με τη βοήθεια των οποίων, κάποιος χρήστης θα μπορούσε να έχει πρόσβαση σε ένα pool, δικών του ή και άλλων developers για την ταχύτερη δημιουργία περιγραφών εφαρμογών για Cloud.

2.4 Software Repository

Με τον όρο Software repository, αναφερόμαστε σε ειδικούς τύπους εξυπηρετητών αρχείων οι οποίοι παρέχουν στους χρήστες την ικανότητα να αποθηκεύουν εκθέματα λογισμικού (artifacts) και να μπορούν να τα ανακτούν οποιαδήποτε στγμή (download capabilities).

Τα σύγχρονα software repositories, υποστηρίζουν δυνατότητες Version Control δίνοντας έτσι τη δυνατότητα στους πελάτες που τα χρησιμοποιούν, να μπορούν να ζητήσουν και να ανακτήσουν προηγούμενες εκδόσεις των αρχείων.

Ένα άλλο στοιχείο το οποίο τα διαφοροποιεί από τους κοινούς εξυπηρετητές αρχείων είναι το γεγονός ότι τα repositories, ακολουθούν κάποιο τύπο μορφοποίησης η οποία τους επιτρέπει να αποθηκεύουν και να εκθέτουν το περιεχόμενό τους σε διάφορα εργαλεία πελάτες. Όλα τα repositories, για κάθε στοιχείο το οποίο εμπεριέχουν, δημιουργούν αρχεία μεταδεδομένων, τα οποία είναι, συνήθως, σε μορφή XML και περιγράφουν το στοιχείο αυτό με δεδομένα του τύπου Group Id, Artifact Id, Version number (GAV) κλπ.

Ο πιο διαδεδομένος τύπος repository είναι τα repository τύπου maven. Ο λόγος για τη δημοφιλία τους, είναι η άμεση σχέση που έχουν με τη γλώσσα προγραμματισμού JAVA, καθώς το Maven είναι ένα εργαλείο που λειτουργεί σαν build automation tool σε μεγάλα JAVA Projects.

Όσον αφορά τη μορφοποίηση ενός maven repository, όπως όλα τα repositories, για κάθε artifact που φυλάσσεται σε αυτό, υπάρχει ένα XML αρχείο πολύ συγκεκριμένο, το Project Object Model (POM) το οποίο περιέχει τις περιγραφικές πληροφορίες του artifact και τυχόν εξαρτήσεις του από άλλα artifacts. Μέσα στο pom.xml αρχείο,

σημαντικό ρόλο για τη μοναδική αναγνώριση και indexing του artfact στο repository, παίζουν και παράμετροι GAV που αναφερθήκαν και πιο πάνω, ή αλλιώς στη γλώσσα των Maven repositories, συντεταγμένες στοιχείου, τις οποίες αποτελούν οι πληροφορίες group id, artifact id και version.

Με τον όρο group id αναφερόμαστε σε κάποια λογική ομάδα, στην οποία μπορεί να ανήκει κάποιο artifact. Μία τέτοια λογική ομάδα μπορεί να αποτελείται από περισσότερα του ενός artifacts. Παράδειγμα, όλα τα arifacts που δημιουργεί ο οργανισμός Apache Foundation ανήκουν κάτω από το group id org.apache. Ο όρος artifact id, αναφέρεται στο περιγραφικό όνομα του artifact και ο συνδυασμός group id και artifact id πρέπει να είναι μοναδικός για κάθε project. Το version αναφέρεται στον αριθμός της έκδοσης ενός συγκεκριμένου artifact και βάση αυτού του αριθμού μπορεί να γίνει η αναζήτηση προηγούμενων εκδόσεων ενός συγκεκριμένου artifact. Επιπλέον πληροφορίες που μπορεί να περιέχει το αρχείο pom.xml είναι ο τύπος του artifact π.χ. zip, jar, war κ.α., η ημερομηνία που το aritfact ανέβηκε στο repository και κάποιες φορές το όνομα του δημιουργού.

2.4.1 Sonatype Nexus

Το Sonatype Nexus[3] είναι ένα εργαλείο διαχείρισης Maven repository, στο οποίο μπορούν να φυλάσσονται binary artifacts διαφόρων τύπων αλλά και κώδικας. Τόσο τα artifacts τα ίδια, όσο και τα μεταδεδομένα που τα χαρακτηρίζουν, αποθηκεύονται στο file system του, προσδίδοντάς του έτσι ταχύτητα και αξιοπιστία, στις ενέργειες indexing και αναζήτησης. Παρέχει ένα γραφικό περιβάλλον διεπαφής, από το οποίο ο χρήστης μπορεί να διαχειριστεί το repository και τα περιεχόμενά του. Μπορεί εύκολα να ρυθμιστεί, χρησιμοποιώντας μια ομάδα XML αρχείων και οι δυνατότητές του μπορούν εύκολα να επεκταθούν, με τη χρήση plugins. Παρέχει στους developers, πρόσβαση στο API του με τη χρήση REST κλήσεων. Ακόμα, παρέχει τη δυνατότητα δημιουργίας λογικών repositories (shadow repositories), τα οποία λειτουργούν ως σύνδεσμοι στα πραγματικά repository. Μειονέκτημά του, η μη υποστήριξη αναζήτησης, σε artifacts που δεν ακολουθούν τη μορφοποίηση Maven.

2.4.2 jFrog Artifactory

Από την άλλη, το Artifactory[9] είναι ένα repository manager, γενικού σκοπού. Υποστηρίζει την φύλαξη αρχείων διαφόρων τύπων. Η φύλαξη των αρχείων στο Artifactory μπορεί να γίνει με δύο τρόπους, είτε στο file system, είτε σε μία βάση δεδομένων, της οποίας η χρήση δεν συνιστάται όταν το μέγεθος του repository γίνει αρκετά μεγάλο. Τα μεταδεδομένα για τα artifacts που φιλοξενεί, αποθηκεύονται σε μία βάση δεδομένων. Αυτό αυξάνει την ταχύτητα αναζήτησης και indexing όταν το repository είναι σχετικά μικρό σε μέγεθος, όμως χάνει από την αξιοπιστία του, γιατί πολύ βασικές του διεργασίες όπως η αναζήτηση, εξαρτώνται από την κατάσταση της βάσης δεδομένων. Οι ρυθμίσεις του είναι αποθηκευμένες σε μία βάση δεδομένων. Υποστηρίζει επέκταση των δυνατοτήτων του με τη χρήση Groovy plugin και παρέχει στους developers πρόσβαση στο API του με τη χρήση REST κλήσεων.

2.4.3 Apache Archiva

Ο τελευταίος repository manager που μελετήθηκε, είναι το Apache Archiva[10]. Το Archiva είναι λογισμικό ανοικτού κώδικα και υποστηρίζει artifacts που ακολουθούν τη μορφοποίηση maven. Τα artifacts όπως και τα μεταδεδομένα τους φυλάσσονται στο file system και η διαδικασίες αναζήτησης και indexing γίνονται χρησιμοποιώντας τη βιβλιοθήκη Lucene. Παρέχει γραφικό περιβάλλον διεπαφής, για τις ενέργειες διαχείρισης και όλες του οι ρυθμίσεις αποθηκεύονται στο local file system σε XML αρχεία. Μπορεί να εεκταθεί η λειτουργικότητα του με τη χρήση plugins, όμως δεν παρέχει πλήρης πρόσβαση στο API για developers. Επιτρέπει τη δημιουργία shadow repositories καθώς και την αναζήτηση artifacts που δεν ακολουθούν τη μορφοποίηση maven. Μειονέκτημα του είναι, ότι χρειάζεται μία βάση δεδομένων για να αποθηκεύσει τους χρήστες διαχειριστές, πράγμα που δημιουργεί single points of failure, καθώς η κρίσιμη διαδικασία διαχείρισης του repository εξαρτάται από την κατάσταση της βάσης δεδομένων.

2.4.4 Συγκριτικός πίνακας χαρακτηριστικών repositories

Έχοντας δει τα χαρακτηριστικά των repositories, μπορεί να δημιουργηθεί ένας συγκριτικός πίνακας των χαρακτηριστικών τους, από τον οποίο μπορούμε να αποφασίσουμε ποιο από τα παραπάνω repositories είναι καταλληλότερο για να

χρησιμοποιηθεί σαν το repository που θα φυλάσσονται τα software artifacts του Marketplace plugin.

Χαρακτηριστικά	Sonatype Nexus	Archiva	Artifactory
Indexing	Yes, filesystem	Yes, filesystem	Yes, database
Storage	Filesystem	Filesystem	Filesystem or database
Search	Yes	Yes, Lucene	Yes
Relies on database	No	Yes	Yes
Configuration	Yes, filesystem	Yes, filesystem	Yes, database
Non-Maven Support	Only storage	Yes	Yes
REST API	Yes	Partly	Yes

Πίνακας 2.2 Χαρακτηριστικά των Repositories

Έχοντας πλέον, τη συγκεντρωτική εικόνα του Πίνακα 2.2, κρίθηκε πως τα repositories Archiva και Artifactory προσθέτουν επιπλέον επίπεδο πολυπλοκότητας στην ανάπτυξη του plugin, αφού βασίζονται σε κάποια εξωτερική βάση δεδομένων για την εκτέλεση σημαντικών εργασιών. Αυτό εκτός του ότι αυξάνει το επίπεδο πολυπλοκότητας, δημιουργεί και ένα single point of failure, τη βάση δεδομένων καθώς χωρίς αυτή δεν μπορούν να λειτουργήσουν πλήρως τα repositories. Ένα επιπλέον μειονέκτημα του Archiva, είναι ότι δεν υποστηρίζει πλήρως API, πράγμα που δυσκολεύει τη διαδικασία ανάπτυξης λογισμικού με αυτό.

Για τους λόγους που περιεγράφησαν πιο πάνω για την ανάπτυξη του Marketplace plugin επιλέχθηκε το repository Nexus.

Κεφάλαιο 3

Σχεδιασμός plugin

3.1 Απαιτήσεις

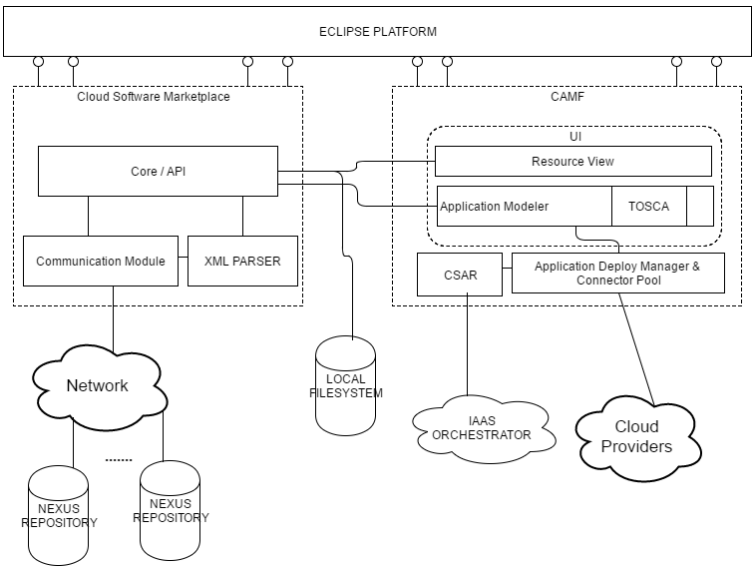
Οι λειτουργικές απαιτήσεις για την υλοποίηση του Cloud Software Marketplace plugin επιβάλλουν την ύπαρξη ενός Nexus repository σε έναν απομακρυσμένο ή τοπικό εξυπηρετητή ο οποίος θα φιλοξενεί τα software artifacts που θα χρησιμοποιούν οι χρήστες για τις περιγραφές των εφαρμογών τους. Το plugin θα πρέπει να είναι σε θέση να επικοινωνεί με τον εξυπηρετητή στον οποίο είναι εγκατεστημένο το repository και να μπορεί, μέσω HTTP αιτημάτων, να κάνει κλήσεις στο REST API του Nexus repository για να μπορεί να επιτελέσει τις ακόλουθες ενέργειες:

- Αναζήτηση των artifacts στο repository βάση keywords
- Download των artifacts από το repository
- Προβολή των artifacts που βρίσκονται στο ευρετήριο του repository
- Διάβασμα των μεταδεδομένων που περιγράφουν το artifact, είτε πρόκειται για μεταδεδομένα τα οποία το Nexus δημιουργεί από μόνο του, είτε για μεταδεδομένα που ορίζονται από το χρήστη κατά το ανέβασμα του artifact στο repository
- Δυνατότητα επιλογής του repository στο οποίο το Marketplace θα συνδέεται
- Δυνατότητα ενεργειών διαχείρισης του repository όπως upload και delete artifact
- Εμφάνιση μηνυμάτων λαθών σε περίπτωση αποτυχίας κάποιας ενέργειας κατά τη διάρκεια της επικοινωνίας του Marketplace με το repository και αντίστοιχα ενημερωτικά μηνύματα για την ολοκλήρωση κάποιας ενέργειας.

Ακόμη, το Marketplace repository, θα πρέπει να μπορεί να επιτελεί ενέργειες οι οποίες είναι ανεξάρτητες από την επικοινωνία του με το repository όπως:

- Δημιουργία cache στο local file system του χρήστη όπου θα φυλάσσονται τα κατεβασμένα artifacts του χρήστη για πιο άμεση χρήση τους.
- Υλοποίηση drag and drop από την παλέτα του CAMF στον Application Model Editor
- Και τέλος δημιουργία TOSCA περιγραφής για το συγκεκριμένο artifact το οποίο ο χρήστης χρησιμοποιεί στην περιγραφή της εφαρμογής του στο CAMF.

3.2 Αρχιτεκτονική του plugin



Σχήμα 3. 1 High-Level Αρχιτεκτονική Cloud Software Marketplace plugin

Όπως προκύπτει και από τις απαιτήσεις, το plugin πρέπει να μπορεί να είναι σε θέση να επικοινωνεί με τα Nexus repositories. Τη δουλειά αυτή αναλαμβάνει να επιτελέσει το Communication Module. Το Communication Module, μέσω του διαδικτύου, στέλνει αιτήματα HTTP στο Nexus repository παραλήπτη. Τα HTTP αιτήματα, περιέχουν κλήσεις στο REST API του Nexus ανάλογα με την εργασία που ο χρήστης επιθυμεί να επιτελέσει (search, download κτλ.). Το Nexus όταν δεχτεί μια κλήση στο REST API του, ανάλογα με τη φύση του αιτήματος, αποστέλλει την κατάλληλη απάντηση, στον αποστολέα του αιτήματος, με όλες τις απαραίτητες πληροφορίες που ο χρήστης ζήτησε, μορφοποιημένες σε XML. Στη συνέχεια, το Communication Module λαμβάνει την απάντηση του Nexus και την παραδίδει στον XML Parser του plugin για να εξάγει τις πληροφορίες για το χρήστη.

Λόγω του γεγονότος ότι το Nexus repository, για κάθε μία από τις ενέργειες του χρήστη που επιτελεί, μορφοποιεί τα δεδομένα που στέλνει σε XML με διαφορετική δομή, ο parser πρέπει να μπορεί να μεταφράζει σωστά κάθε XML απάντηση του Nexus στις κατάλληλες πληροφορίες.

Σχήμα 3. 2 Παράδειγμα XML απάντησης από το NEXUS

Υπεύθυνος για το συντονισμό της όλης διαδικασίας είναι ο πυρήνας του Marketplace plugin, που γνωρίζει ανα πάσα στιγμή τι αίτημα θέλει να στείλει ο χρήστης στο repository και τι είδους απάντηση περιμένει για να επεξεργαστεί. Έτσι είναι σε θέση να ορίσει με ποιο τρόπο θα πρέπει το Communication Module να επικοινωνήσει με το repository και ποια μέθοδο επεξεργασίας θα χρησιμοποιήσει ο parser. Επίσης ο πυρήνας είναι υπεύθυνος για την αναγνώριση λαθών που προκύπτουν κατά τις διάφορες ενέργειες του χρήστη και για την εμφάνιση των κατάλληλων μηνυμάτων λάθους ή ενημέρωσης προόδου. Ακόμη είναι υπεύθυνος για την προβολή των artifacts που βρίσκονται αποθηκευμένα στην τοπική cache του Marketplace στο local file system του χρήστη. Η δουλειά του πυρήνα δεν σταματάει εκεί. Έχει επίσης την ευθύνη της προετοιμασίας των artifacts που θα χρησιμοποιήσει ο χρήστης για τη δημιουργία των περιγραφών του. Αυτό το κομμάτι περιλαμβάνει το configuration των artifacts και την εισαγωγή τους στο Resources View του CAMF όπου από εκεί θα γίνει η εισαγωγή τους στην περιγραφή με drag and drop. Τέλος, σημαντικό κομμάτι των εργασιών που επιτελεί ο πυρήνας είναι η δημιουργία TOSCA περιγραφών για τα artifacts που προέρχονται από το Marketplace, χρησιμοποιώντας τον TOSCA Application Modeler

του CAMF και της δημιουργίας διαγραμματικής αναπαράστασης του εισαγόμενου artifact στον application editor του CAMF.

3.3 Μοντελοποίηση των artifacts

Πολύ βασικό για την επιτυχή λειτουργία του Cloud Software Marketplace plugin είναι το μοντέλο που πρέπει να ακολουθούν τα artifacts, οι εφαρμογές του χρήστη δηλαδή, που φυλάσσονται στο Nexus Repository. Όπως είδαμε μέχρι στιγμής, το Nexus, κάθε φορά που κάποιο νέο artifact ανεβαίνει ανανεώνει το ευρετήριό του με πληροφορίες που αφορούν το νέο artifact για τη διευκόλυνση των ενεργειών που υποστηρίζει το repository. Επιπρόσθετα, είδαμε ότι στο Nexus repository μπορούν να φυλάσσονται artifacts διαφόρων τύπων όπως jar, exe, zip, shell scripts κτλ. Τι γίνεται όμως στην περίπτωση που ο χρήστης θέλει να χρησιμοποιήσει artifacts, τα οποία αποτελούνται από περισσότερα από ένα αρχεία, τα οποία έχουν κάποια σχέση το ένα με το άλλο για την ομαλή λειτουργία του artifact? Είναι ικανοποιητικά τα μεταδεδομένα που παράγει το Nexus για κάθε artifact ή χρειάζεται να γίνει κάτι άλλο πριν? Η απάντηση στα ερωτήματα αυτά είναι ο ορισμός μιας ενιαίας και γενικής δομής που πρέπει να ακολουθούν τα artifacts, έτσι ώστε να έχουν ίδια μεταχείριση, τόσο από το Nexus repository όσο και από το Marketplace plugin.

3.3.1 Δομή των artifacts

Για το σχεδιασμό της δομής των artifacts πρέπει να ληφθούν υπόψιν δύο βασικές προδιαγραφές, (i) πως χειρίζεται το Nexus τα artifacts και (ii) πως θα χρησιμοποιηθούν τα artifacts από το Marketplace plugin.

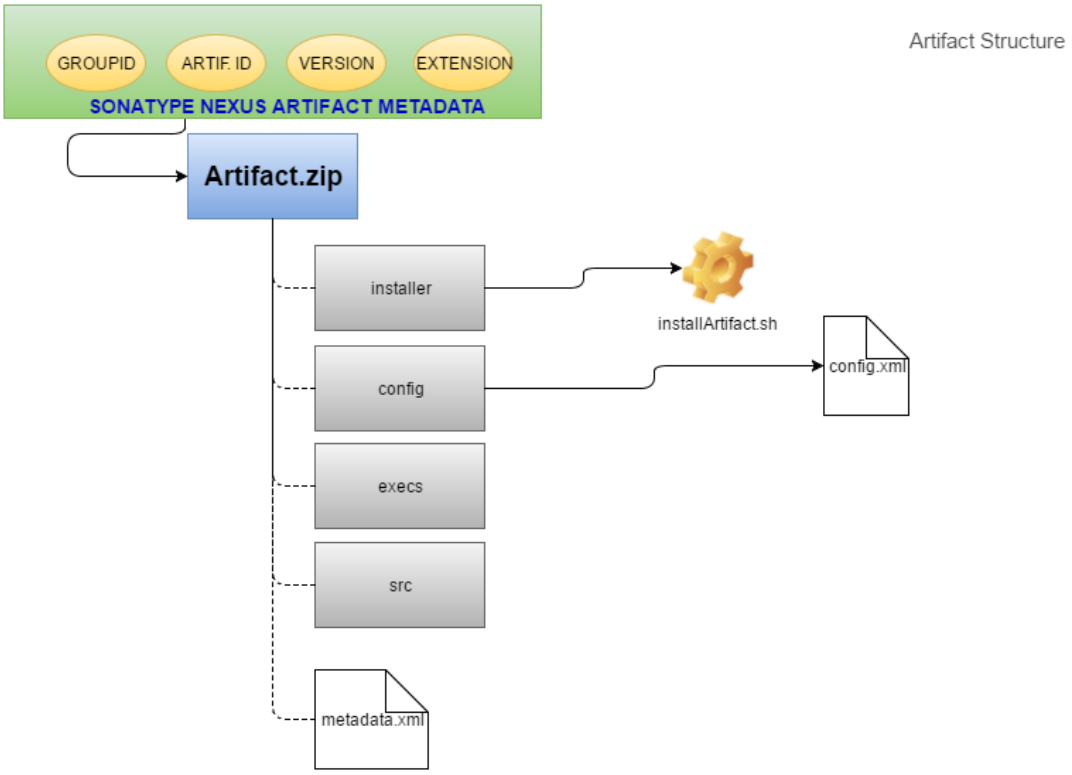
Όσον αφορά την πρώτη προδιαγραφή, το Nexus, για κάθε artifact που ανεβαίνει δημιουργεί μεταδεδομένα που το περιγράφουν. Στην περίπτωση που ένα artifact αποτελείται από περισσότερα αρχεία, για κάθε ένα από αυτά τα αρχεία θα δημιουργήσει μεταδεδομένα περιγραφής, ξεχωριστά για το καθένα. Αυτό είναι μη επιθυμητό, καθώς έτσι χάνεται η όποια σύνδεση έχουν τα αρχεία του artifact μεταξύ τους. Η προτεινόμενη λύση για να ξεπεραστεί αυτό το εμπόδιο είναι, με κάποιον τρόπο να γίνει ομαδοποίηση των αρχείων που είναι απαραίτητα για τη σωστή λειτουργία του artifact. Ο τρόπος με τον οποίο γίνεται η ομαδοποίηση είναι συμπιέζοντας όλα τα αρχεία που αποτελούν το

artifact σε ένα αρχείο zip. Το όνομα του αρχείου zip, πρέπει να ακολουθεί την εξής σύμβαση: *όνομα_artifact-version.zip.*

Με την ομαδοποίηση των αρχείων σε zip, δεν επιλύουμε πλήρως το πρόβλημα της μοντελοποίησης των artifacts. Ναι μεν πετύχαμε εν μέρει το σκοπό μας και ικανοποιήσαμε την πρώτη προδιαγραφή, πρέπει όμως να ικανοποιήσουμε και τη δεύτερη πολύ σημαντική προδιαγραφή, του πως θα χρησιμοποιηθεί το artifact αυτό από το plugin. Ο λόγος που υλοποιείται αυτό το plugin είναι για να διευκολύνει τους χρήστες να δημιουργούν σωστές περιγραφές εφαρμογών Cloud. Όταν η περιγραφή ολοκληρωθεί και δοθεί στον πάροχο για ανάπτυξη, θα πρέπει να υπάρχει η γνώση για το ποιο αρχείο είναι υπεύθυνο για την εγκατάσταση της εφαρμογής, αν το αρχείο περιέχει άλλα εκτελέσιμα ή κώδικα που θα πρέπει να γίνει compile, ποιες παραμέτρους του artifact μπορεί ο χρήστης να κάνει configure και μια γενική περιγραφή της δομής αυτής .

Συνοψίζοντας, η δομή την οποία πρέπει να έχουν τα artifacts (Σχήμα 3.3) είναι η ακόλουθη:

- Ομαδοποίηση όλων των συστατικών μερών του artifact σε ένα zip αρχείο με όνομα «*όνομα_artifact-version.zip»*
- Φάκελος installer που περιέχει το script αρχείο υπεύθυνο για την έναρξη λειτουργίας του artifact. Η ύπαρξη του είναι υποχρεωτική
- Φάκελος config που περιέχει xml αρχείο στο οποίο δηλώνονται υπό μορφή key –value pairs τις configurable παραμέτρους που δηλώνει ο developer του artifact ότι χρειάζεται να δηλωθούν για να λειτουργήσει το artifact. Η ύπαρξη του είναι υποχρεωτική
- Φάκελος execs που περιέχει επιπλέον binary files που ίσως να είναι απαραίτητα για τη λειτουργία του artifact. Η ύπαρξη του είναι προαιρετική.
- Φάκελος src που περιέχει τυχόν κώδικα που πρέπει να γίνει compile για τη λειτουργία του artifact. Η ύπαρξη του είναι προαιρετική.
- Αρχείο με το όνομα metadata.xml στο οποίο περιγράφεται η πιο πάνω δομή. Η ύπαρξή του είναι υποχρεωτική.



Σχήμα 3. 3 Δομή του artifact

```
<artifact>
    <groupId>wordpress</groupId>
    <artifactId>wordpress</artifactId>
    <version>1.0</version>
    <uploadDate></uploadDate>
    <os>Ubuntu</os>
    <installer>installWordpress.sh</installer>
    <config>config.xml</config>
    <binaries>0</binaries>
    <sources>0</sources>
</artifact>
```

Σχήμα 3. 4 Παράδειγμα: Αρχείο metadata.xml

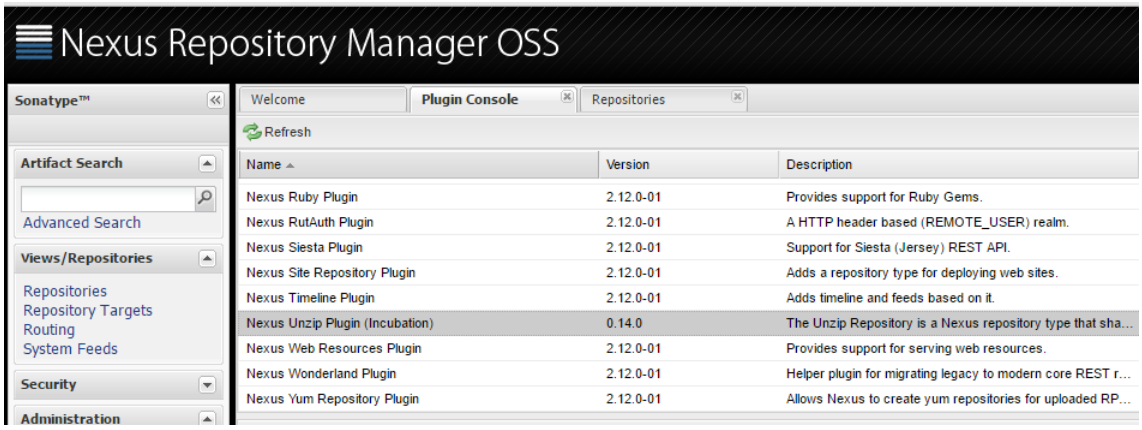
```xml
<properties>
    <property>
        <name>Database Name</name>
        <value></value>
    </property>
    <property>
        <name>Database User</name>
        <value></value>
    </property>
    <property>
        <name>Database Password</name>
        <value></value>
    </property>
    <property>
        <name>Site Name</name>
        <value></value>
    </property>
    <property>
        <name>Admin User</name>
        <value></value>
    </property>
    <property>
        <name>Admin Password</name>
        <value></value>
    </property>
    <property>
        <name>Admin Email</name>
        <value></value>
    </property>
</properties>
```

Σχήμα 3. 5 Παράδειγμα: Αρχείο xml configuration

3.3.2 Nexus Configuration

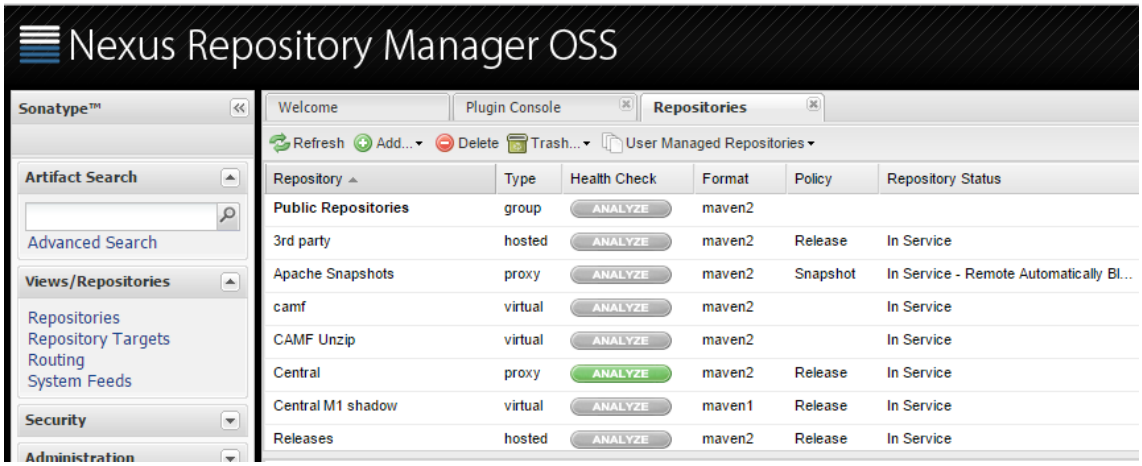
Η δομή που προτείνεται στην προηγούμενη ενότητα, αντιμετωπίζει ένα λειτουργικό πρόβλημα στην ομαλή λειτουργία του Marketplace plugin. Όταν ο χρήστης του Marketplace επιθυμεί να χρησιμοποιήσει κάποιο από τα artifacts που βρίσκονται στο Nexus, οι πληροφορίες που λαμβάνει από το Nexus για το artifact δεν αφορούν το ίδιο το artifact αλλά το ίδιο το zip αρχείο. Στην πραγματικότητα ο για να λειτουργήσει σωστά το Marketplace plugin, χρειάζεται τις πληροφορίες που βρίσκονται στο αρχείο metadata.xml (Σχήμα 3.4) , το οποίο είναι συμπιεσμένο στο zip αρχείο.

Τη λύση, εδώ, προσφέρει το ίδιο το Nexus, με τη δυνατότητα που έχει να επεκταθούν οι λειτουργίες του με την εγκατάσταση Nexus plugins και πιο συγκεκριμένα του Nexus Unzip Plugin (Σχήμα 3.6), το οποίο επιτρέπει την προσπέλαση του εσωτερικού zip και jar αρχείων, σαν να ήταν κανονικοί φάκελοι.



Σχήμα 3. 6 Nexus Unzip Plugin

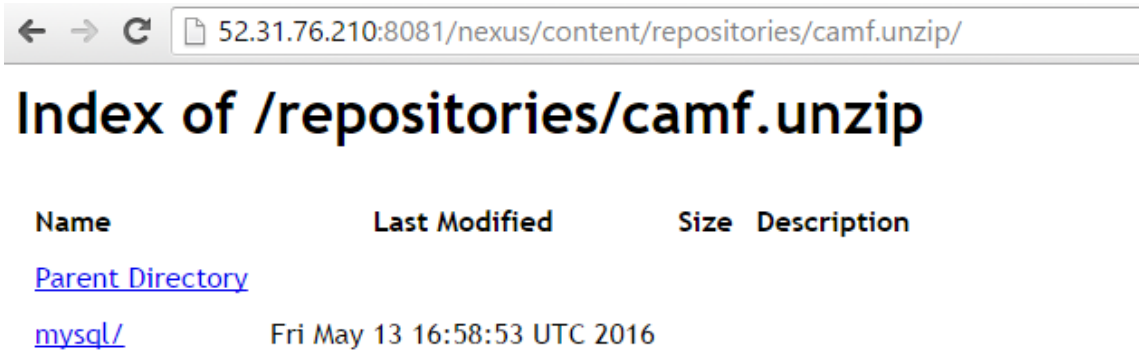
Ο τρόπος με τον οποίο γίνεται αυτό είναι με τη δημιουργία ενός Shadow Repository (Σχήμα 3.7), δηλαδή ενός repository, στο οποίο στο ίδιο δεν φυλάσσονται artifacts, αλλά παρουσιάζει τα artifacts που φυλάσσονται στο master repository στο οποίο έχει πρόσβαση και με την προσθήκη των δυνατοτήτων από το unzip plugin στο configuration του shadow repository (Σχήμα 3.8), θα μπορεί πλέον να κάνει περιήγηση στο εσωτερικό zip αρχείων που βρίσκονται στο Nexus (Σχήμα 3.9).



Σχήμα 3. 7 Το repository CAMF Unzip είναι το shadow repository το repository releases και έχει τις δυνατότητες από το unzip plugin

```
<repository>
  <id>camf.unzip</id>
  <name>Camf Unzip</name>
  <providerRole>org.eclipse.tycho.nexus.internal.plugin.UnzipRepository</providerRole>
  <providerHint>org.eclipse.tycho.nexus.plugin.DefaultUnzipRepository</providerHint>
  <localStatus>IN_SERVICE</localStatus>
  <notFoundCacheActive>true</notFoundCacheActive>
  <notFoundCacheTTL>15</notFoundCacheTTL>
  <userManaged>true</userManaged>
  <exposed>true</exposed>
  <browseable>true</browseable>
  <writePolicy>READ_ONLY</writePolicy>
  <searchable>true</searchable>
  <localStorage>
    <provider>file</provider>
  </localStorage>
  <externalConfiguration>
    <masterRepositoryId>public</masterRepositoryId>
    <useVirtualVersion>true</useVirtualVersion>
    <synchronizeAtStartup>false</synchronizeAtStartup>
  </externalConfiguration>
</repository>
```

Σχήμα 3. 8 Configuration CAMF Unzip Shadow Repository



Σχήμα 3. 9 Περιήγηση στο εσωτερικό του Shadow repository CAMF UNZIP. Το zip artifact mysql φαινεται σαν φάκελος στον οποίο ο χρήστης μπορεί να περιηγηθεί στο εσωτερικό του.

3.3.3 Υποθέσεις

Σε αυτό το σημείο και μέχρι το τέλος αυτού του εγγράφου, θεωρείται ότι ισχύουν οι εξής υποθέσεις για τη φύση των artifacts τα οποία φυλάσσονται στο Nexus repository και χρησιμοποιούνται από το Marketplace:

i. Τα artifacts τα οποία χρησιμοποιεί ο χρήστης στις περιγραφές των εφαρμογών του ακολουθούν την προτεινόμενη δομή.
ii. Τα artifacts λειτουργούν σωστά, σύμφωνα με τις απαιτήσεις του developer.
iii. Δεν εμπεριέχεται κακόβουλο λογισμικό στα artifacts.

Στην ενοτητα του Κεφαλαίου 6, 6.2 Μελλοντική εργασία, προτείνεται η υλοποίηση ενός συστήματος ελέγχου της ορθότητας και καθαρότητας των artifacts στα repositorires.

3.4 Φάση σχεδιασμού

Αφού ορίστηκε η αρχιτεκτονική του plugin, σχεδιάστηκε το μοντέλο που πρέπει να ακολουθούν τα artifacts και ρυθμίστηκε κατάλληλα το Nexus, στη συνέχεια πρέπει να οριστούν ποιες λειτουργικότητες πρέπει να επιτελεί κάθε module του plugin. Όπως φαίνεται και από το (Σχήμα 3.1), τα 3 βασικά modules του plugin είναι (i) ο πυρήνας / API του plugin που καθορίζει το σύνολο των ενεργειών που μπορεί να επιτελέσει ο χρήστης με το Marketplace, (ii) το communication module που υλοποιεί τις μεθόδους επικοινωνίας του plugin με το Nexus και τέλος (iii) ο XML parser που μεταφράζει τις απαντήσεις του Nexus σε δεδομένα χρήσιμα για τη λειτουργία plugin.

3.4.1 Το API του plugin

Στην ενότητα αυτή θα παρουσιαστεί το σύνολο των κυρίως λειτουργιών του plugin, την περιγραφή της λειτουργίας και αν υπάρχει με ποια κλήση του REST API του Nexus γίνεται αντιστοίχιση. Στο Παράρτημα Β παρουσιάζεται η υλοποίηση του API σε JAVA.

Μέθοδος	**Παράμετροι**	**Επιστρέφει**	**NEXUS REST API**	**Περιγραφή**
keywordSearch()	String	ArrayList <Artifacts>	/service/ local/ lucene/ search	Στέλνει HTTP GET αίτημα στο REST API του NEXUS. Επιστρέφει λίστα με τα artifacts που αντιστοιχούν στην παράμετρο string της

				αναζήτησης
downloadArtifact()	String, String	void	/nexus/ content	Στέλνει HTTP GET αίτημα στο REST API του NEXUS για κατέβασμα του αρχείου που δηλώνεται στην πρώτη παράμετρο και αποθηκεύεται στο local file system φάκελο που είναι η δεύτερη παράμετρος
uploadArtifact()	Map<String, String>	void	/service/ local/ artifact/ maven/ content	Στέλνει HTTP POST αίτημα στο REST API του NEXUS. Ανεβάζει το αρχείο στο repository. Το αρχείο δηλώνεται στις παραμέτρους Map<String, String>
indexRepository()	-	ArrayList <Artifacts>	/service/ local/ repositories/ public/ index_content/	Στέλνει HTTP GET αίτημα στο REST API του NEXUS. Επιστρέφει μία

				λίστα με το συνολικό περιεχόμενο του repository
deleteArtifact()	Artifacts	Void	content/ repositories/	Στέλνει HTTP DELETE αίτημα στο REST API του NEXUS. Διαγράφει το artifact που ορίζεται στην παράμετρο Artifacts από το repository
pingRepository()	String	String	service/ local/status	Στέλνει HTTP GET αίτημα στο REST API του NEXUS. Ελέγχει αν το repository που καθορίζεται στην παράμετρο είναι online. Επιστρέφει την απάντηση του repository.

Πίνακας 3. 1 Το API του plugin

3.4.2 Communication Module

Το δεύτερο συστατικό στοιχείο του Marketplace plugin είναι το Communication Module, το οποίο είναι υπεύθυνο για να στέλνει τα κατάλληλα HTTP αιτήματα, που

ορίζει ο πυρήνας, στο REST API του NEXUS. Στον Πίνακα 3.2 φαίνονται οι λειτουργίες του Communication Module

Μέθοδος	**Παράμετροι**	**Επιστρέφει**	**Περιγραφή**
CloudHttpGetRequest()	String	String	Στέλνει HTTP GET αίτημα στο URL που ορίζεται στην παράμετρο και επιστρέφει την απάντηση του εξυπηρετητή σαν String
CloudHttpPostRequest()	String	String	Στέλνει HTTP POST αίτημα στο URL που ορίζεται στην παράμετρο και επιστρέφει την απάντηση του εξυπηρετητή σαν String
CloudHttpDeleteRequest()	String	String	Στέλνει HTTP DELETE αίτημα στο URL που ορίζεται στην παράμετρο και επιστρέφει την απάντηση του εξυπηρετητή σαν String
authenticate()	String, String	String	Προσθέτει σε αιτήματα HTTP POST τις παραμέτρους

			username και password κωδικοποιημένες σε Base64 για σκοπούς αναγνώρισης του Marketplace Manager από το Nexus
addPart()	Map<String, String>	Void	Εισάγει παραμέτρους σε HTTP POST αιτήματα. Οι παράμετροι είναι κωδικοποιημένοι σε μορφή key-value pair μέσα στη δομή Map.

Πίνακας 3. 2 Το Communication Module

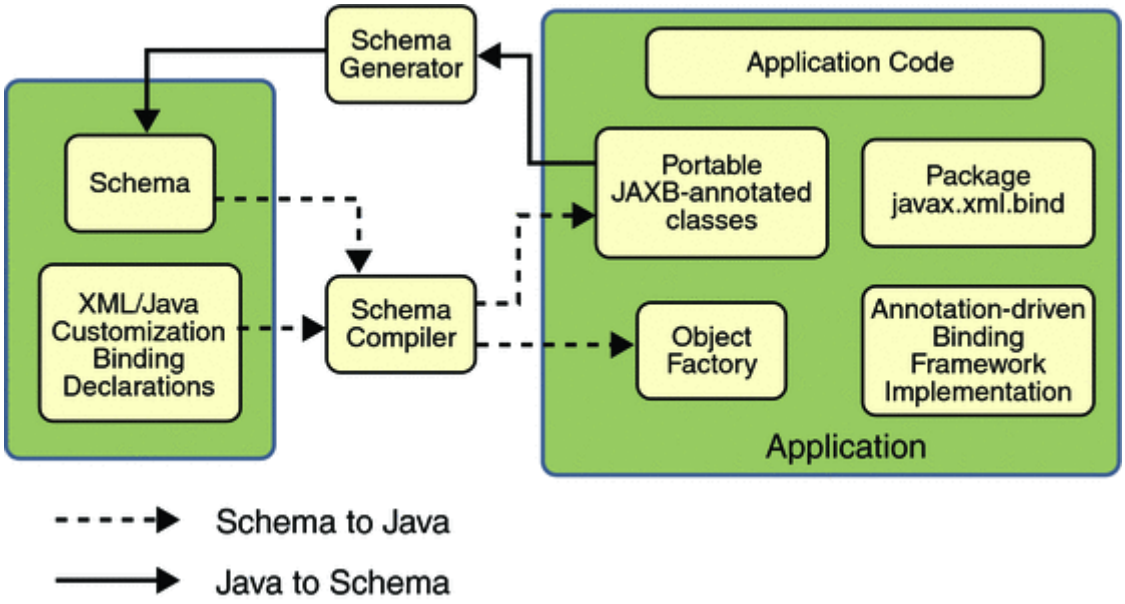
3.4.3 XML Parser

Ο XML Parser, είναι το τρίτο και τελευταίο συστατικό στοιχείο του Marketplace. Δουλειά του είναι, όπως αναφέρθηκε και προηγουμένως η μετάφραση των XML απαντήσεων από το Nexus, σε δεδομένα που χρειάζεται το plugin. Για το σχεδιασμό του Parser χρησιμοποιήθηκαν δύο τρόποι οι οποίοι αναλύονται στις επόμενες δύο ενότητες. Ο πρώτος τρόπος, είναι με τη χρήση του API της JAVA Java Architecture for XML Binding (JAXB) και ο δεύτερος με την ανάλυση τη χρήση του Document Object Model.

3.4.3.1 JAXB

Η αρχιτεκτονική της υλοποίησης ενός XML Parser με χρήση JAXB[12] (Σχήμα 3.10) αποτελείται από τα εξής συστατικά στοιχεία:

- Τον schema compiler: ο οποίος συνδέει ένα XML schema με μία κλάση JAVA της οποίας τα χαρακτηριστικά συμφωνούν με τα χαρακτηριστικά που περιέχονται στο XML schema
- Τον schema generator: ο οποίος αντιστοιχεί τα χαρακτηριστικά κλάσεων JAVA με χαρακτηριστικά που πρέπει να υπάρχουν στο XML schema (κάνει την αντίστροφη δουλειά από τον schema compiler)
- Το binding runtime framework το οποίο κατά τη διάρκεια της εκτέλεσης χρησιμοποιεί τα δύο προηγούμενα συστατικά και παρέχει δυνατότητες γραψίματος (marshalling), διαβάσματος (unmarshalling) από και προς XML σχήματα και δυνατότητα χειρισμού των δεδομένων XML



Σχήμα 3. 10 Αρχιτεκτονική JAXB

Οι κλάσεις απαντήσεων του Nexus repository οι οποίες χρησιμοποιούν το JAXB είναι οι ακόλουθες:

Κλάση	**Σκοπός**
Status	Διαβάζει την απάντηση από την κλήση API pingRepository().
Artifact	Διαβάζει το περιεχόμενο της απάντησης της κλήσης API keywordSearch(). Αφορά το αρχείο metadata.xml που

	βρίσκεται στο εσωτερικού του zipped artefact.
Properties	Διαβάζει το περιεχόμενο του αρχείου xml configuration στο οποίο αναγράφονται οι παράμετροι για τις οποίες ο χρήστης πρέπει να προσδώσει τιμές.

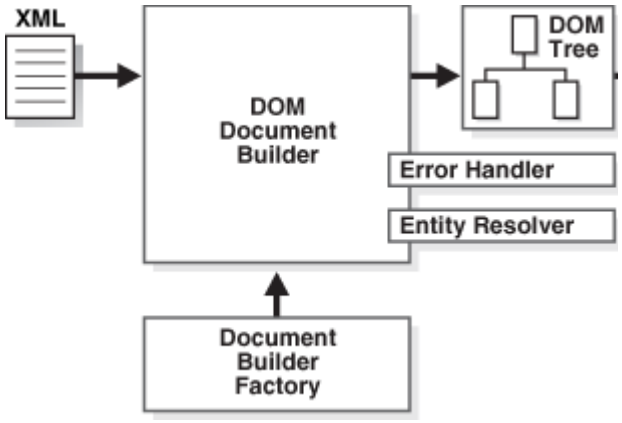
Πίνακας 3.3 JAXB κλάσεις απαντήσεων Nexus repository

3.4.3.1 XML DOM Parser

Ο άλλος τρόπος που χρησιμοποιήθηκε για το parsing των απαντήσεων XML είναι παράγοντας το DOM Tree της XML απάντησης. Η JAVA περιέχει built-in βιβλιοθήκες με τις οποίες μπορεί να το καταφέρει αυτό.

Αρχικά ορίζεται ένα αντικείμενο της κλάσης DocumentBuilder, το οποίο, χρησιμοποιώντας τη μέθοδο parse, διαβάζει το string το οποίο παίρνει σαν παράμετρο και επιστρέφει ένα αντικείμενου της κλάσης Document, που αντιπροσωπεύει το DOM του XML.

Στη συνέχεια χρησιμοποιώντας αντικείμενα των κλάσεων Node και Nodelist αποκτάται πρόσβαση για διάβασμα ή γράψιμο στα διάφορα στοιχεία που ορίζονται στο XML.



Σχήμα 3. 11 Βήματα DOM XML Parsing

Οι μέθοδοι που υλοποιούν το XML DOM Parsing είναι οι ακόλουθες:

Μέθοδος	Παράμετρος	Επιστρέφει	Περιγραφή
xmlParseIndex()	String	ArrayList<String>	Επιστρέφει μια λίστα από strings που αντιπροσωπεύουν

			τις πληροφορίες που απαντάει το Nexus στην κλήση indexRepository()
xmlParseKeywordSearch()	String	ArrayList <Artifacts>	Επιστρέφει μία λίστα από artifacts που αντιπροσωπεύουν τις πληροφορίες που απαντάει το Nexus στην κλήση keywordSearch()

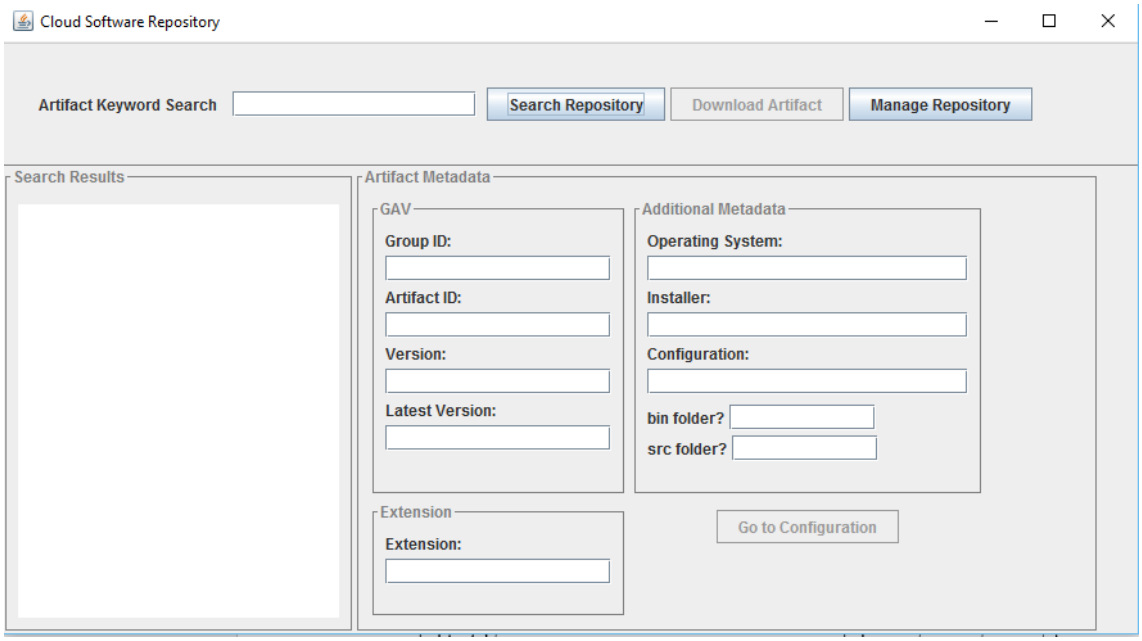
Πίνακας 3.4 Μέθοδοι XML DOM Parsing

Κεφάλαιο 4

Υλοποίηση

4.1 Υλοποίηση stand alone εφαρμογής

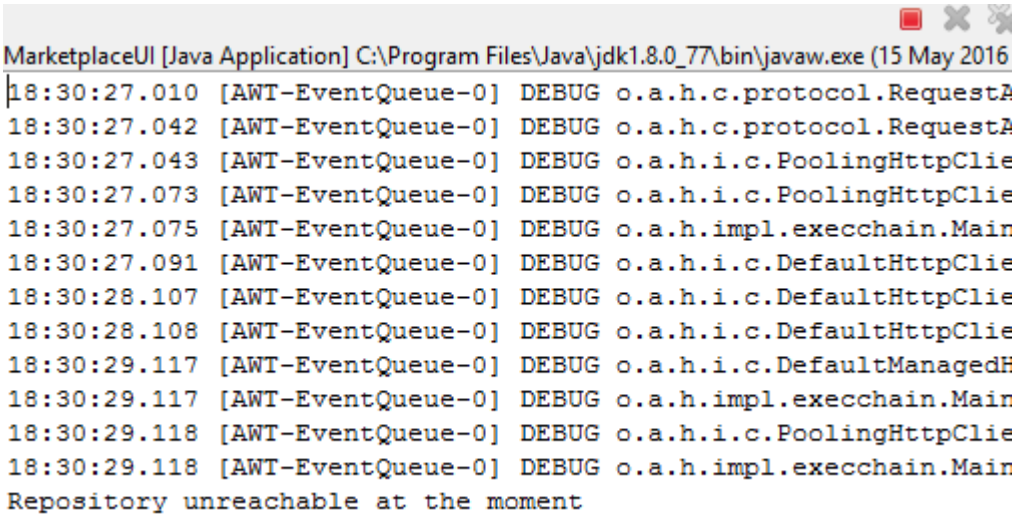
Πριν την υλοποίηση του Marketplace plugin, για σκοπούς επίδειξης και ελέγχου της λειτουργικότητας, έγινε υλοποίηση της stand alone εφαρμογής. Η stand alone εφαρμογή εμπεριέχει όλη τη λειτουργικότητα που ορίστηκε στην ενότητα 3.1 Απαιτήσεις του Κεφαλαίου 3 εκτός από τις λειτουργίες που έχουν σχέση με την περιγραφή της εφαρμογής στο CAMF και αποτελείται από δύο διαφορετικές οθόνες που εξομοιώνουν τη χρήση του plugin από τους δύο τύπους χρηστών που υποστηρίζονται από το Marketplace, τους απλούς χρήστες και τους διαχειριστές.



Σχήμα 4. 1 Οθόνη Χρήστη Stand Alone εφαρμογής

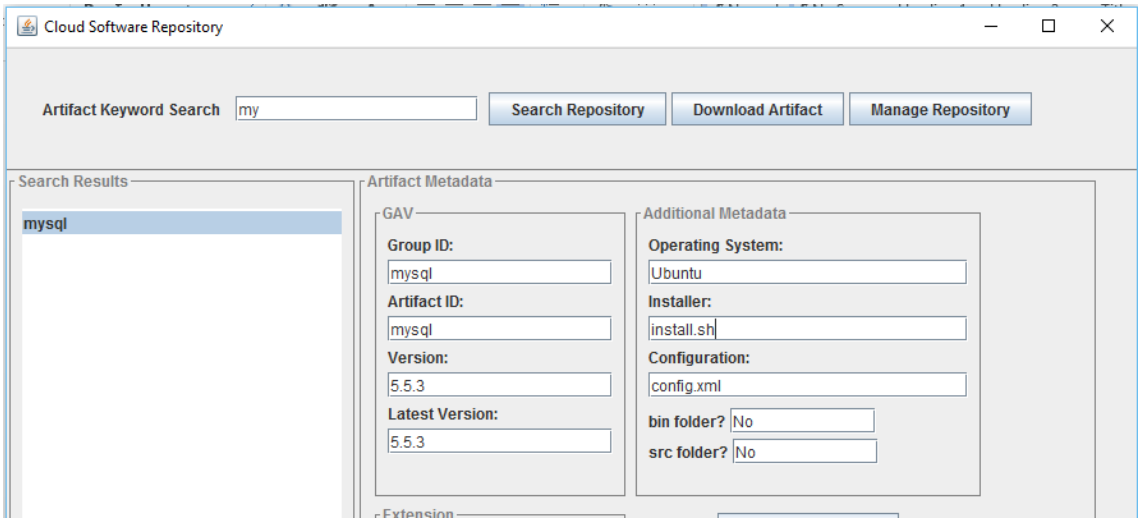
Το γραφικό περιβάλλον διεπαφής της οθόνης του χρήστη (Σχήμα 4.1), επιτρέπει στους χρήστες της εφαρμογής να επιτελέσουν τις λειτουργίες που τους επιτρέπεται. Οι ενέργειες που μπορεί να κάνει ένας χρήστης είναι η αναζήτηση artifacts από το repository βάση keywords και το κατέβασμα artifacts από το repository, σε φάκελο στο

local file system του χρήστη, που χρησιμοποιείται σαν cache για τα artifacts τα οποία κατεβαίνουν από το repository. Επίσης η stand alone εφαρμογή είναι σε θέση να παρουσιάζει τυχόν σφάλματα που προκύπτουν κατά τη διάρκεια της χρήσης της, στην κονσόλα του Eclipse (Σχήμα 4.2).

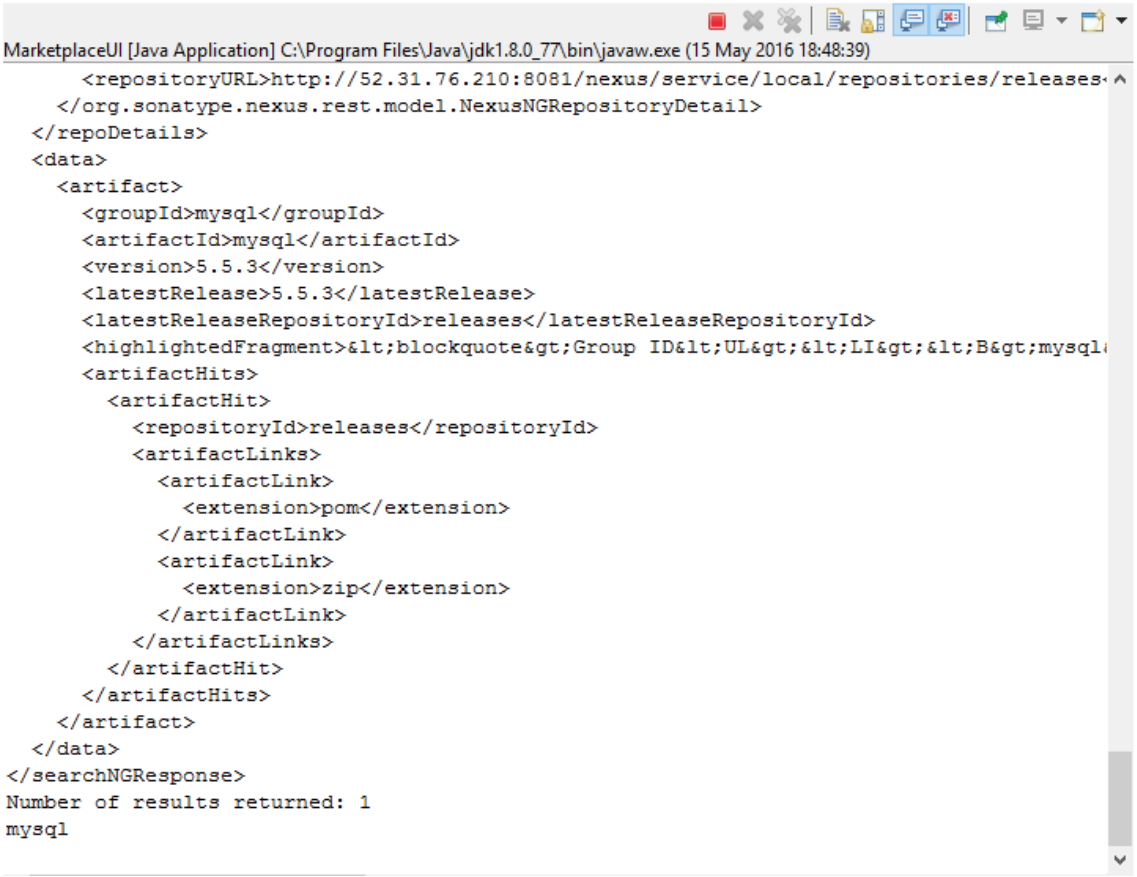


Σχήμα 4. 2 Εμφάνιση μηνύματος λάθους στο χρήστη. Το repository δεν είναι προσβάσιμο αυτή τη στιγμή.

Ο χρήστης πληκτρολογεί μερικώς ή πλήρως, το όνομα του artifact που επιθυμεί να χρησιμοποιήσει, στο Text box Artifact Keyword Search και ακολούθως πατάει το κουμπί Search Repository. Στη συνέχεια η εφαρμογή χρησιμοποιώντας την API κλήση στη μέθοδο, keywordSearch(), εμφανίζει στην κονσόλα του Eclipse τα αποτελέσματα της επικοινωνίας της εφαρμογής με το Nexus (XML απάντηση) (Σχήμα 4.4) και στο γραφικό περιβάλλον διεπαφής εμφανίζει, επιπλέον πληροφορίες που αφορούν το artifact και βρίσκονται εντός του zipped artifact στο αρχείο metadata.xml (Σχήμα 4.3).



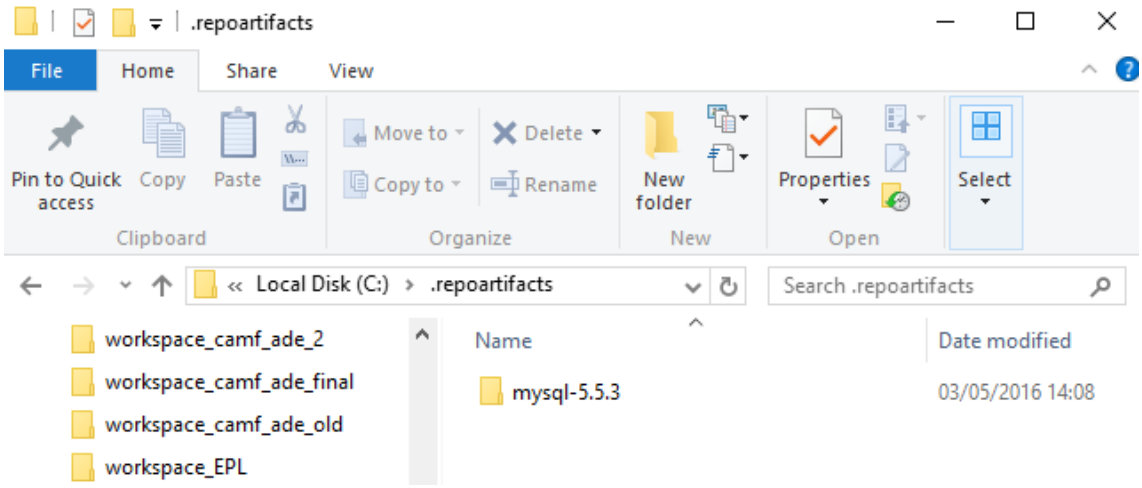
Σχήμα 4. 3 Αποτελέσματα αναζήτησης και εμφάνιση πληροφορίων σχετικές με το artifact



Σχήμα 4. 4 Αποτελέσματα αναζήτησης στην κονσόλα του Eclipse

Ακολούθως, πατώντας το κουμπί Download Artifact, η εφαρμογή κάνοντας κλήση στο API, στη μέθοδο downloadArtifact(), κατεβάζει το επιλεγμένο artifact από τη λίστα στο

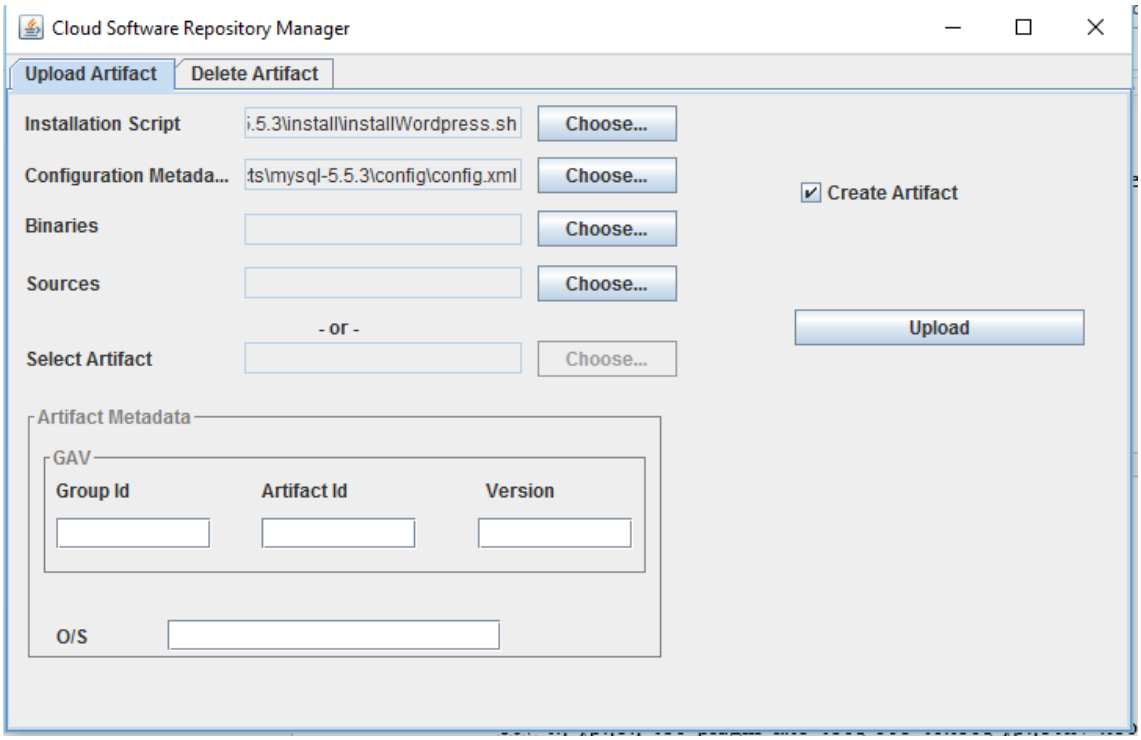
φάκελο .repoartifacts του local file system του χρήστη, που λειτουργεί σαν cache για τα κατεβασμένα artifacts (Σχήμα 4.5).



Σχήμα 4. 5 Το artifact στο local file system

Αντίστοιχα, η οθόνη του διαχειριστή του repository, αποτελείται από δύο tabs, τα οποία αντιπροσωπεύουν τις ενέργειες τις οποίες μπορεί να επιτελέσει, την ενέργεια upload artifact και την ενέργεια delete artifact.

Στο tab upload artifact, ο διαχειριστής μπορεί να επιλέξει να ανεβάσει artifact στο repository με δύο τρόπους. Είτε με επιλογή έτοιμου artifact σε μορφή zip, που να ακολουθεί τη δομή που παρουσιάστηκε στο Κεφάλαιο 3, είτε με το να επιλέξει τα συστατικά στοιχεία τα οποία αποτελούν το artifact του μέσω του γραφικού περιβάλλοντος διεπαφής και επιλέγοντας το check box create artifact και στη συνέχεια πατώντας το κουμπί upload, να ανεβάσει το artifact στο repository. Αξίζει να αναφερθεί εδώ ότι αν ο χρήστης επιλέξει το δεύτερο τρόπο για να κάνει upload, το zip αρχείο που αντιπροσωπεύει το artifact δημιουργείται από μόνο του, ακολουθώντας πιστά το μοντέλο του Κεφαλαίου 3.



Σχήμα 4. 6 Upload Artifact

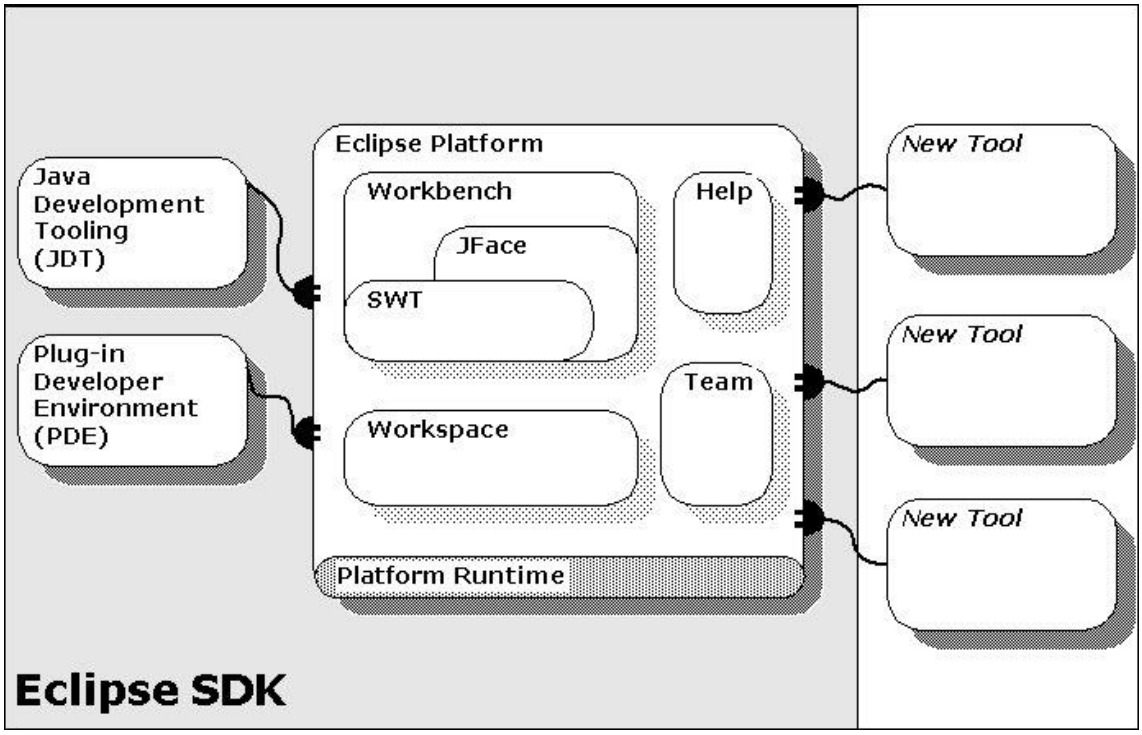
Τέλος, στο tab Delete Artifact, ο διαχειριστής μπορεί να διαγράψει ένα artifact το οποίο βρίσκεται στο repository του, επιλέγοντας το artifact από τη λίστα και πατώντας το κουμπί διαγραφή.

Σχήμα 4. 7 Delete Artifact

4.2 Ενσωμάτωση στο Eclipse

Όπως είναι φυσικό, το επόμενο βήμα της φάσης υλοποίησης, είναι η ενσωμάτωση του Marketplace, υπο μορφή plugin στο Eclipse. Το Eclipse είναι βασισμένο γύρω από το concept των plugins, όπου τα plugins είναι δομημένα κομμάτια κώδικα ή δεδομένων που συνεισφέρουν στη λειτουργικότητα του συστήματος (Σχήμα 4.8). Η συνεισφορά

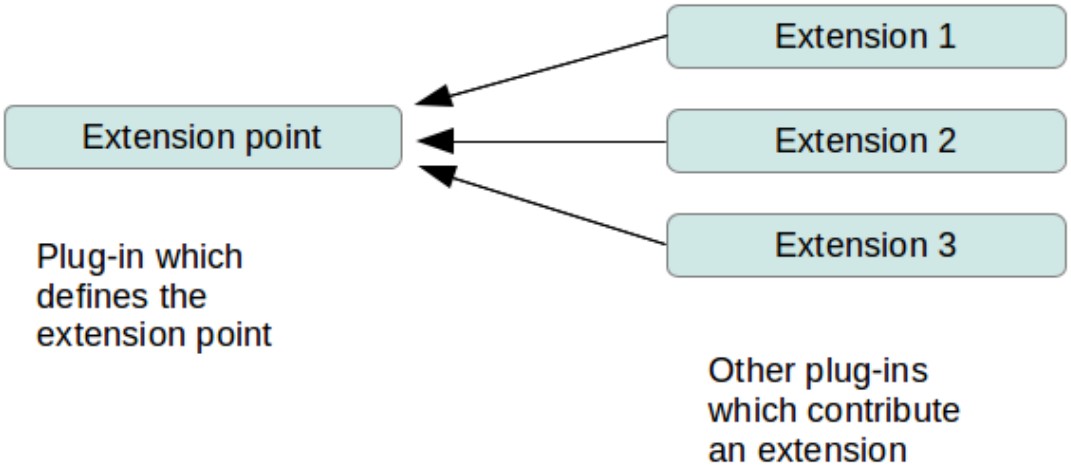
αυτή γίνεται μέσω του μηχανισμού των Extension και Extension Points[11], ενώ η δημιουργία γραφικου περιβάλλοντος διεπαφής για το plugin γίνεται με τη δημιουργία των Views.



Σχήμα 4. 8 Η αρχιτεκτονική της πλατφόρμας Eclipse

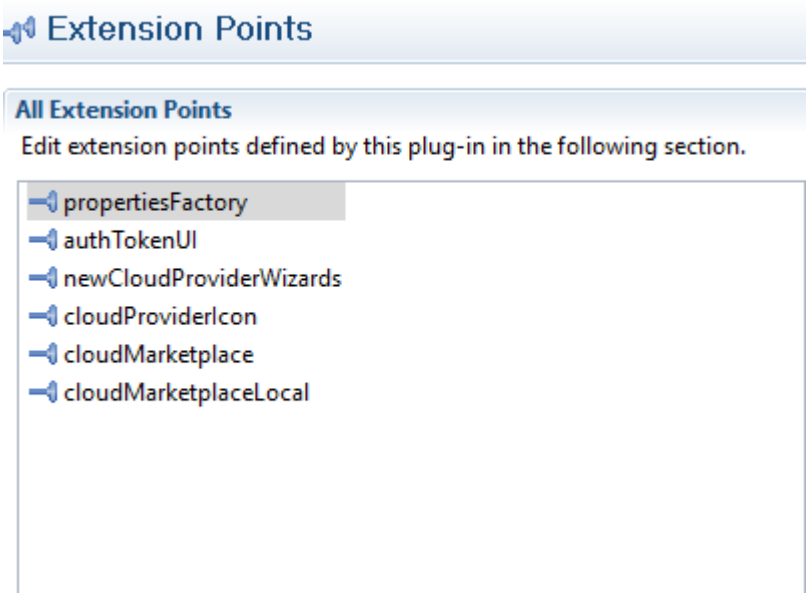
4.2.1 Extensions και Extension points

Όπως αναφέρθηκε και προηγουμένως, η συνεισφορά στη λειτουργικότητα στο Eclipse γίνεται με τη δήλωση Extension και Extension Points. Για να μπορέσουμε να αντιληφθούμε καλύτερα τις έννοιες των Extensions και Extension points μπορεί να χρησιμοποιηθεί ο εξής παραλληλισμός από την καθημερινή μας ζωή: Το Extension Point μπορεί να θεωρηθεί σαν μια πρίζα (socket) πάνω στο οποίο μπορούν να συνδεθούν πολλά και διαφορετικά καλώδια (plugs) τα οποία είναι σχεδιασμένα για το socket στο οποίο θα ενωθούν. Δηλώνοντας λοιπόν, κάποιος, και υλοποιώντας Extensions και Extension points στο Eclipse μπορεί να επεκτείνει τη λειτουργία του. Ο συνηθισμένος τρόπος με τον οποίο γίνεται ο ορισμός των Extension points είναι μέσω XML περιγραφής καθώς και καθορισμός των λειτουργιών που θα υλοποιεί το Extension με JAVA Interfaces.



Σχήμα 4. 9 Extension και extension points στο Eclipse

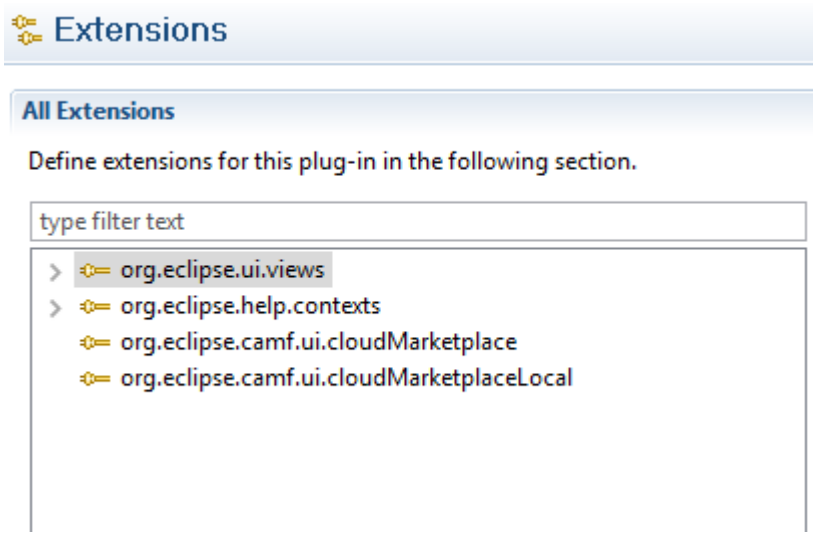
Για τις ανάγκες υλοποίησης του Marketplace plugin, δηλώθηκαν δύο Extension Points στο μανιφέστο του user interface του CAMF Plugin, που επιτρέπουν την επέκταση των λειτουργιών του, με το plugin Marketplace. Ο λόγος για τον οποίο δηλώθηκαν δύο Extension Points, είναι ότι το Marketplace, στην πραγματικότητα είναι δυο plugins. Το ένα αφορά την πρόσβαση στην cache στο local file system (cloudMarketplaceLocal), ενώ το δεύτερο εμπεριέχει αυτή καθαυτή τη λειτουργικότητα του Marketplace (cloudMarketplace) (Σχήμα 4.10).



Σχήμα 4. 10 Τα extension points cloudMarketplace και cloudMarketplaceLocal που επεκτείνουν τη λειτουργικότητα του CAMF

Επιπλέον, στο μανιφέστο περιγραφής του project του Marketplace, δηλωθήκαν τα Extension Points στα οποία το Marketplace plugin συνεισφέρει με λειτουργικότητα,

μέσα στα οποία βρίσκονται και τα Extension Points τα οποία δηλωθήκαν στο μανιφέστο CAMF UI plugin (Σχήμα 4.11).

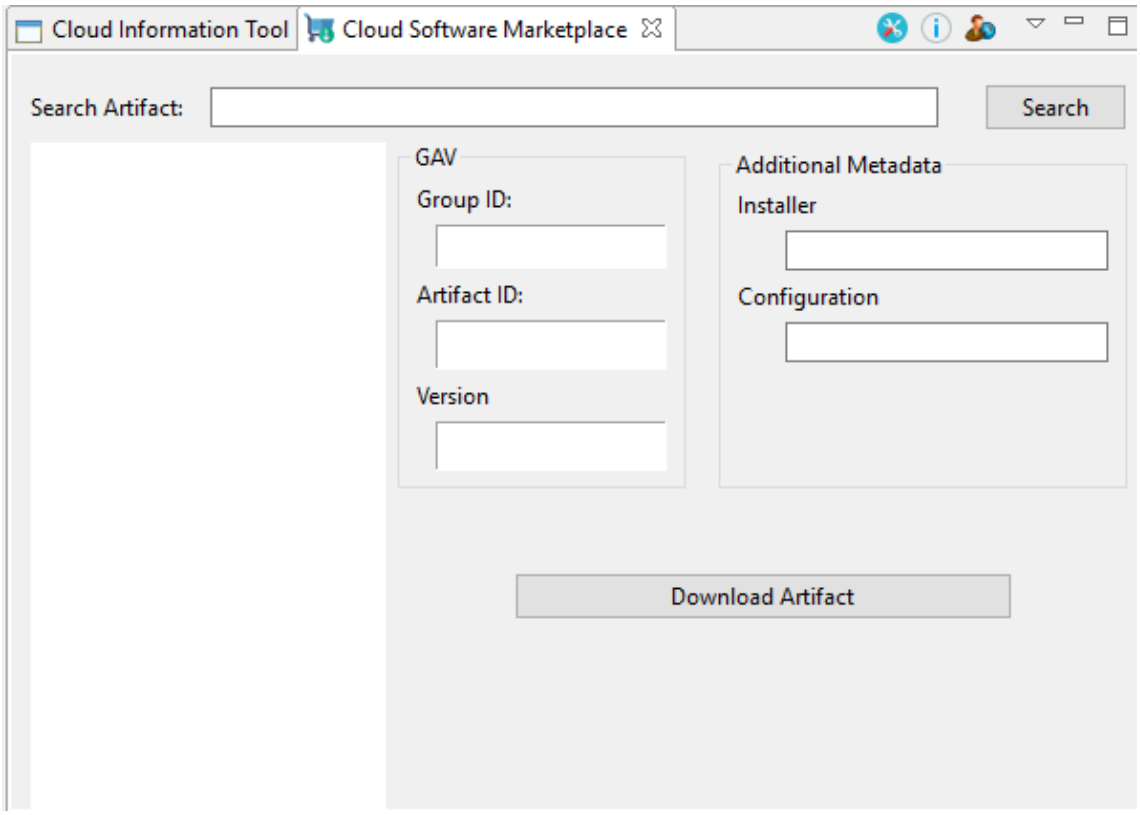


Σχήμα 4. 11 Extension Points στα οποία συνεισφέρει το Marketplace

4.2.2 Views

Αφού ορίστηκαν τα Extension Points και τα Extensions στα οποία το Marketplace plugin συνεισφέρει, το επόμενο βήμα είναι ο σχεδιασμός των views, μέσω των οποίων, το plugin θα αλληλεπιδρά με το χρήστη. Τα views που αποτελούν το Marketplace είναι τρία: (i) το view του χρήστη, (ii) το view του διαχειριστή του repository και (iii) το view της cache.

4.2.2.1 Το View του χρήστη

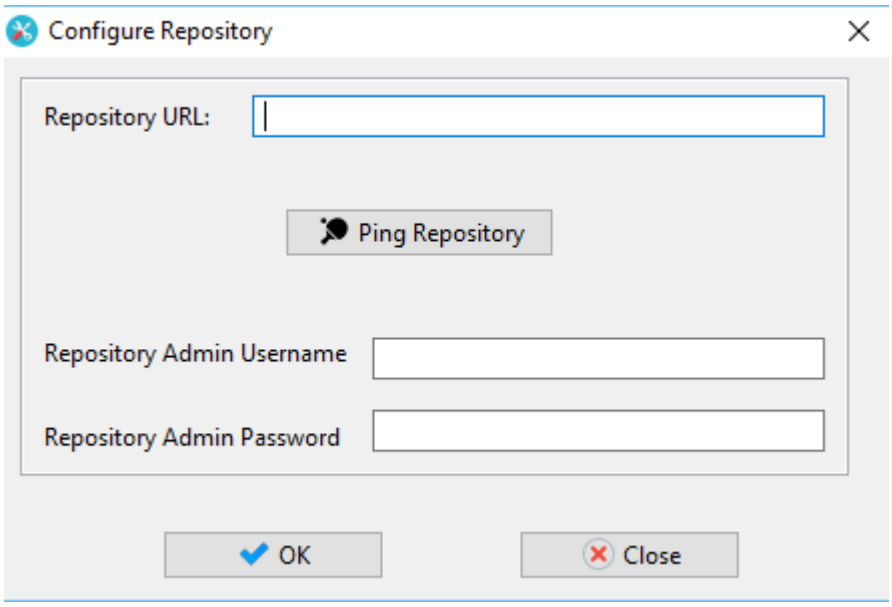


Σχήμα 4. 12 Το View του χρήστη

Όπως παρατηρούμε και από το Σχήμα 4.12 το view του χρήστη είναι σχεδόν πανομοιότυπο με την οθόνη του χρήστη στη stand alone εφαρμογή. Η διαφορά είναι ότι, τώρα, το γραφικό περιβάλλον διεπαφής του Marketplace είναι ενσωματωμένο στο Eclipse σαν plugin. Η λειτουργικότητα που παρέχει το view στους χρήστες είναι ακριβώς ίδια με αυτή της stand alone εφαρμογής. Οι λειτουργίες που υποστηρίζει είναι αναζήτηση των artifacts που φυλάσσονται σε ένα Nexus repository, προβολή των κυρίως μεταδεδομένων που αφορούν το συγκεκριμένο artifact, στα κατάλληλα πεδία, και κατέβασμα του artifact στο local file system.

Πατώντας στο κουμπί ρύθμισης του repository (Σχήμα 4.13), ο χρήστης μπορεί να ορίσει το Nexus repository το οποίο θέλει να χρησιμοποιήσει για τη λειτουργία του Marketplace, παρέχοντας το URL του repository στην ακόλουθη μορφή: *http(s)://repositoryurl:repositoryport*. Σε περίπτωση που ο χρήστης δώσει URL το οποίο δεν ακολουθεί τη συγκεκριμένη μορφή, ενημερώνεται ότι το URL που έδωσε δεν είναι μορφοποιημένο σωστά. Ακόμη στο παράθυρο του repository configuration, έχει τη δυνατότητα να ελέγξει αν το repository του οποίου έδωσε το URL είναι online, με το κουμπί Ping Repository. Σε αντίθετη περίπτωση, ενημερώνεται και πάλι με αντίστοιχο μήνυμα λάθους. Στα πεδία username και password μπορεί να εισάγει το όνομα χρήστη

και κωδικό για να γίνει η ταυτοποίηση του σαν διαχειριστή συστήματος και να προβεί σε ενέργειες διαχείρισης από το view του διαχειριστή.

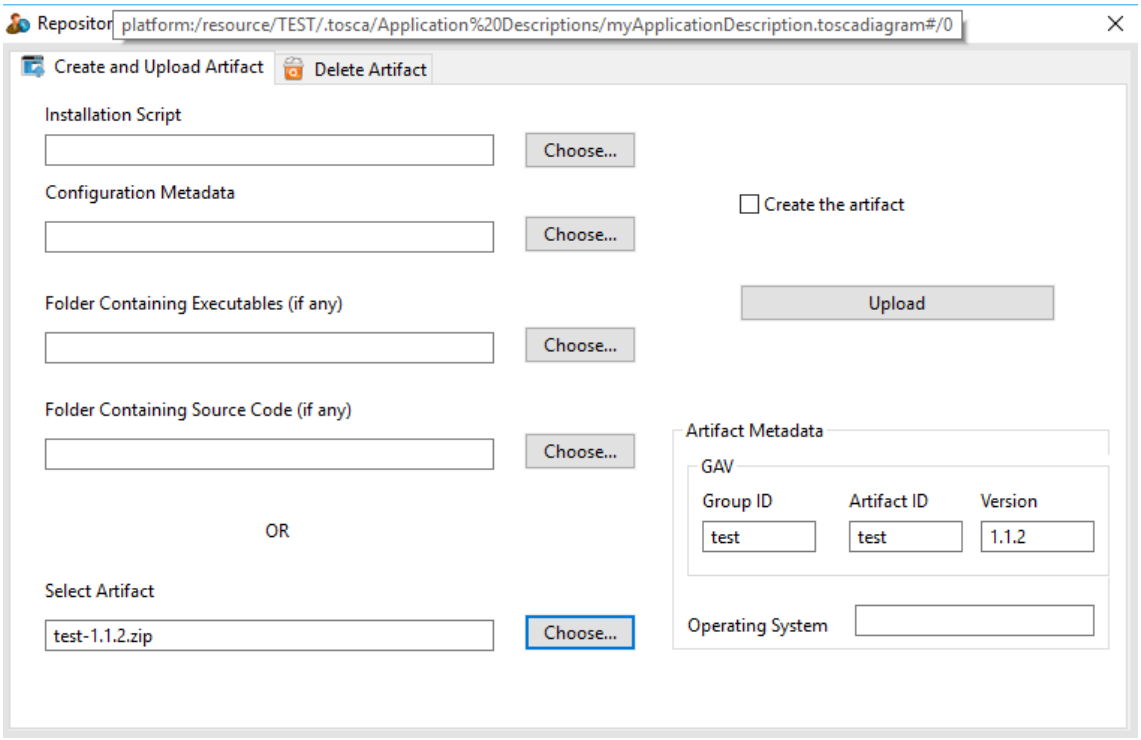


Σχήμα 4. 13 Διάλογος ρύθμισης repository

Τέλος κάθε φορά που ανοίγει το view του χρήστη, γίνεται έλεγχος αν υπάρχει ο φάκελος που αντιπροσωπεύει την local cache του Marketplace, .repoartifacts. Σε περίπτωση που δεν υπάρχει, το Marketplace, τον δημιουργεί, στο root folder του συστήματος (κάτω από το C:// στα Windows, κάτω από το /home/ στα Linux).

4.2.2.2 Το View του διαχειριστή

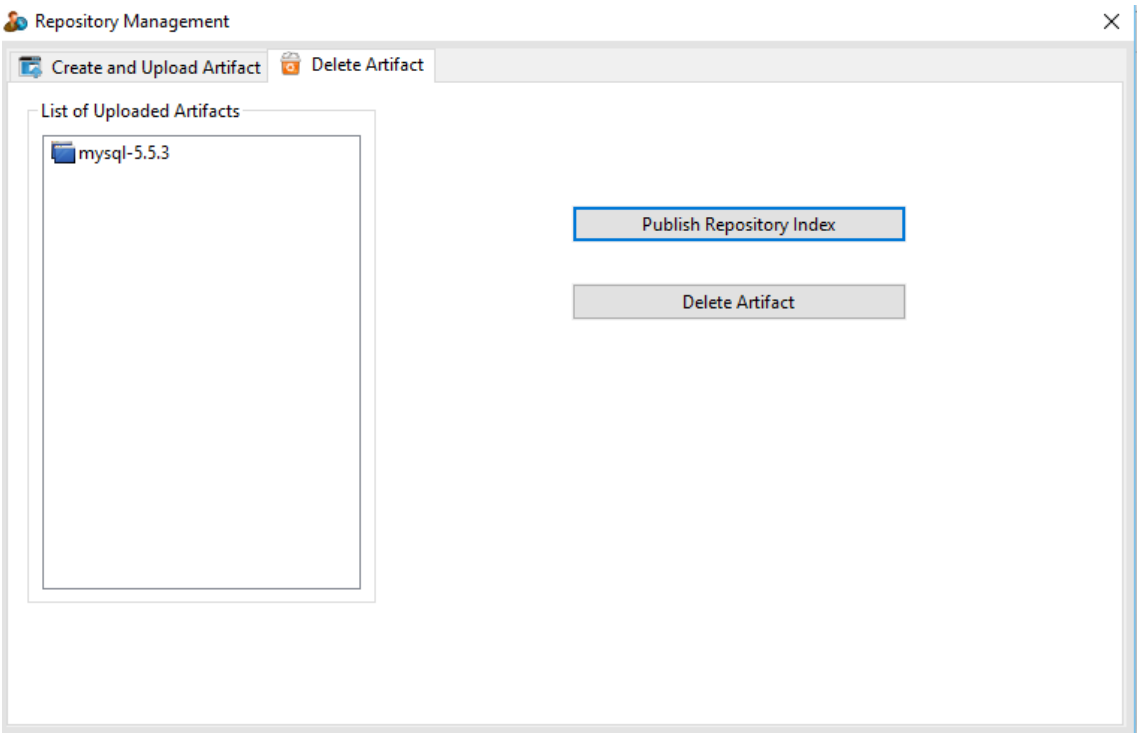
Όπως και στη stand alone εφαρμογή, έτσι και στο plugin, το view του διαχειριστή, επιτελεί τις εργασίες διαχείρισης του Nexus Repository. Η λειτουργικότητα που υποστηρίζει είναι ακριβώς η ίδια με τη λειτουργικότητα διαχείρισης στη stand alone εφαρμογή, με τις ενέργειες upload artifact και delete artifact από το repository. Στο plugin view του διαχειριστή, κατά τη διαδικασία του upload έχει προστεθεί η λειτουργία αυτόματου εντοπισμού των GAV παραμέτρων από το όνομα του artifact (Σχήμα 4.14), όταν ο χρήστης ανεβάζει έτοιμα artifacts. Στην περίπτωση που ο χρήστης δημιουργεί το artifact, επιλέγοντας μέσω του γραφικού περιβάλλοντος διεπαφής τα συστατικά του στοιχεία, τότε αυτά μαζεύονται στο φάκελο .repoartifacts, συμπιέζονται στο αρχείο zip με τη δομή που περιγράφηκε στην ενότητα 3.3.1 και γίνονται upload στο repository. Αξίζει να υπενθυμίσουμε σε αυτό το σημείο, τις υποθέσεις από την ενότητα 3.3.3 σχετικά με την ορθότητα των artifacts.



Σχήμα 4. 14 View του διαχειριστή. Οι παράμετροι GAV έχουν συμπληρωθεί αυτόματα βάση του ονόματος του artifact

Όσον αφορά το tab του delete artifact (Σχήμα 4.15) δεν υπάρχουν ουσιαστικές διαφορές από τη stand alone εφαρμογή, πέραν των αλλαγών που έγιναν στο σχεδιασμό, κατά τα άλλα η λειτουργικότητα παραμένει ακριβώς η ίδια.

Βασική προϋπόθεση για την σωστή λειτουργία του View του διαχειριστή, είναι να έχει δώσει ο χρήστης τα διαπιστευτήρια του στο παράθυρο διαλόγου του Σχήματος 4.13, έτσι ώστε να μπορεί να γίνει η ταυτοποίηση του σαν διαχειριστή του repository και να προβεί σε ενέργειες διαχείρισης.



Σχήμα 4. 15 Tab delete artifact

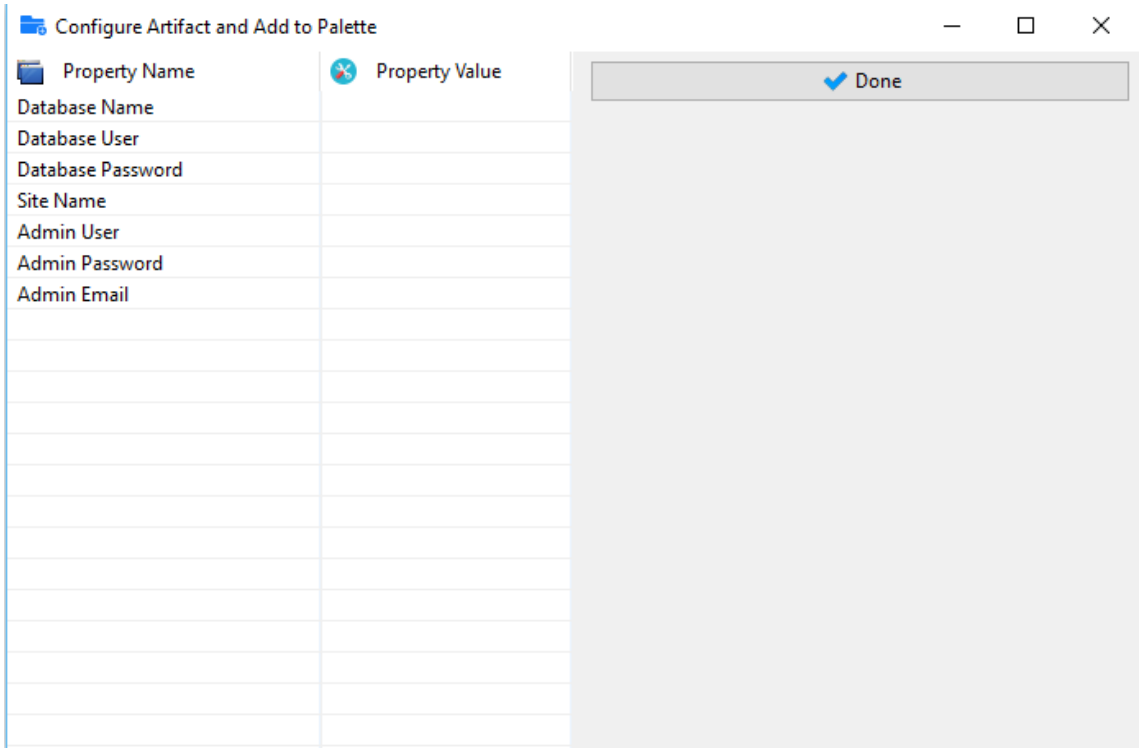
4.2.2.3 Το View του local file system

Το μόνο view που δεν είχε υλοποιηθεί στη stand alone εφαρμογή είναι το view του local file system. Σε αυτό το view, παρουσιάζονται τα περιεχόμενα του φακέλου .repoartifacts, τα οποία αντιστοιχούν στα artifacts που έχει κατεβάσει μέχρι στιγμής ο χρήστης από το Nexus repository.

Όταν κατεβαίνουν τα zipped artifacts από το Nexus repository, αποσυμπιέζονται και καταλήγουν σε αυτόν το φάκελο. Όταν ο χρήστης θέλει να χρησιμοποιήσει κάποιο artifact στην περιγραφή εφαρμογών που δημιουργεί, αρχικά επιλέγει το artifact από το view (Σχήμα 4.16) και εν συνεχεία πατώντας το κουμπί configuration που βρίσκεται στο action bar του view, προσδίδει τιμές στις configurable ιδιότητες του artifact. Αυτό επιτυγχάνεται επειδή το view είναι σε θέση να διαβάσει το XML configuration αρχείο, κάνοντας χρήση της JAXB Parser κλάσης Properties(Σχήμα 4.17).

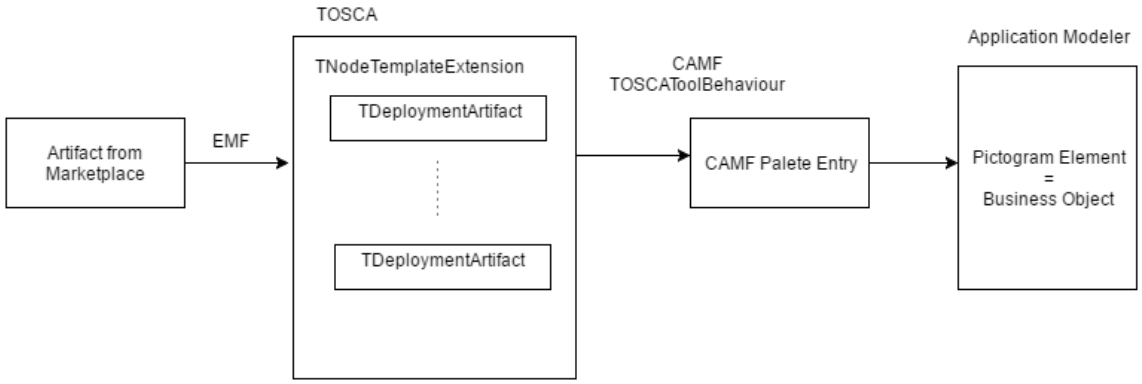
Όταν ο χρήστης τελειώσει με το configuration, δημιουργείται το αρχείο config.cfg που περιέχει τις τιμές που έδωσε ο χρήστης και το artifact τοποθετείται στην παλέτα του CAMF και είναι έτοιμο να χρησιμοποιηθεί.

Σχήμα 4. 16 Το view του local file system που περιέχει artifacts που έχει κατεβάσει μέχρι στιγμής ο χρήστης.



Σχήμα 4. 17 Παράθυρο διαλόγου configuration artifact, με τις παραμέτρους που πρέπει να δώσει τιμές ο χρήστης.

4.2.3 Αλλαγές και προσθήκες στο CAMF



Σχήμα 4. 18 Διαδικασία εισαγωγής Artifact στο CAMF

Έχοντας ενσωματώσει το Marketplace σαν plugin στο Eclipse, έπρεπε να προστεθούν τα τελευταία κομμάτια λειτουργικότητας του και που αφορούν λειτουργικότητα που έχει σχέση με το CAMF.

Πιο συγκεκριμένα, μετά την ολοκλήρωση του configuration κάποιου artifact, που θα χρησιμοποιηθεί στην περιγραφή μιας Cloud εφαρμογής, θα πρέπει να γίνεται μετατροπή του σε αντικείμενο TNodeTemplateExtension του CAMF. Το αντικείμενο TNodeTemplateExtension είναι η προγραμματιστική αναπαράσταση των Node Templates στο TOSCA. Το CAMF για να δημιουργήσει αντικείμενα TNodeTemplateExtension ή οποιαδήποτε άλλα αντικείμενα που αντιστοιχούν στη γλώσσα προτύπου TOSCA χρησιμοποιεί το Eclipse Modeling Framework (EMF).

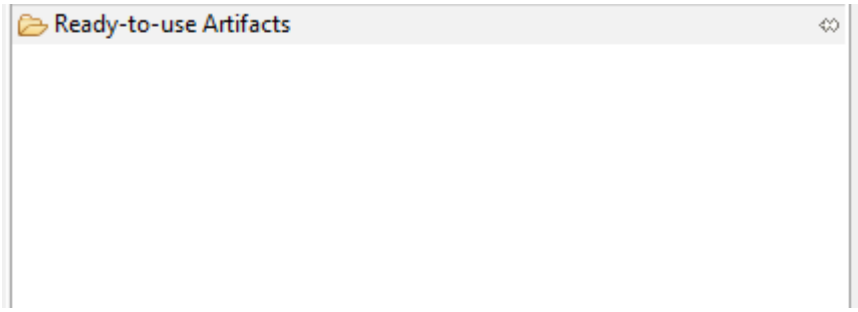
Το EMF είναι ένα modelling framework, το οποίο χρησιμοποιώντας κάποιο δομημένο μοντέλο δεδομένων σε XML, παράγει ένα σύνολο από Java κλάσεις που αναπαριστούν το μοντέλο προγραμματιστικά και ένα σύνολο από adapter κλάσεις που παρέχουν πρόσβαση για προβολή και επεξεργασία του μοντέλου.

Στη συνέχεια, για κάθε αρχείο που σχετίζεται με το artifact του χρήστη, δημιουργείται ένα αντικείμενο TDeploymentArtifact το οποίο αντιπροσωπεύει το Deployment Artifact στο TOSCA, πάλι χρησιμοποιώντας το EMF. Για κάθε TDeploymentArtifact δηλώνεται το χαρακτηριστικό artifactType να είναι RUA, που σημαίνει ότι αυτό το TDeploymentArtifact προήλθε από τη διαδικασία εισαγωγής μέσω του Marketplace, και το χαρακτηριστικό artifactName που αντιπροσωπεύει το όνομα του αρχείου που εισάγεται στην περιγραφή. Σε αυτό το σημείο να αναφερθεί ότι, όταν δημιουργηθούν το αντικείμενα TDeploymentArtifacts που περικλείονται από το TNodeTemplateExtension, ολά τα σχετικά αρχεία που σχετίζονται με το artifact αντιγράφονται κάτω από το φάκελο Atifacts/Applications του CMAF Project στο οποίο δημιουργείται η περιγραφή.

Για να επιτευχθεί η πιο πάνω διαδικασία, προστέθηκε η κλάση CreateReadyToUseArtifactFeature.java στο πακέτο κλάσεων του CAMF org.eclpse.camf.tosca.editor.features, η οποία χρησιμοποιώντας το προκαθορισμένο EMF του CAMF, παράγει την TOSCA περιγραφή για το artifact με τη δημιουργία των αντικειμένων TNodeTemplateExtension και TDeploymentArtifact.

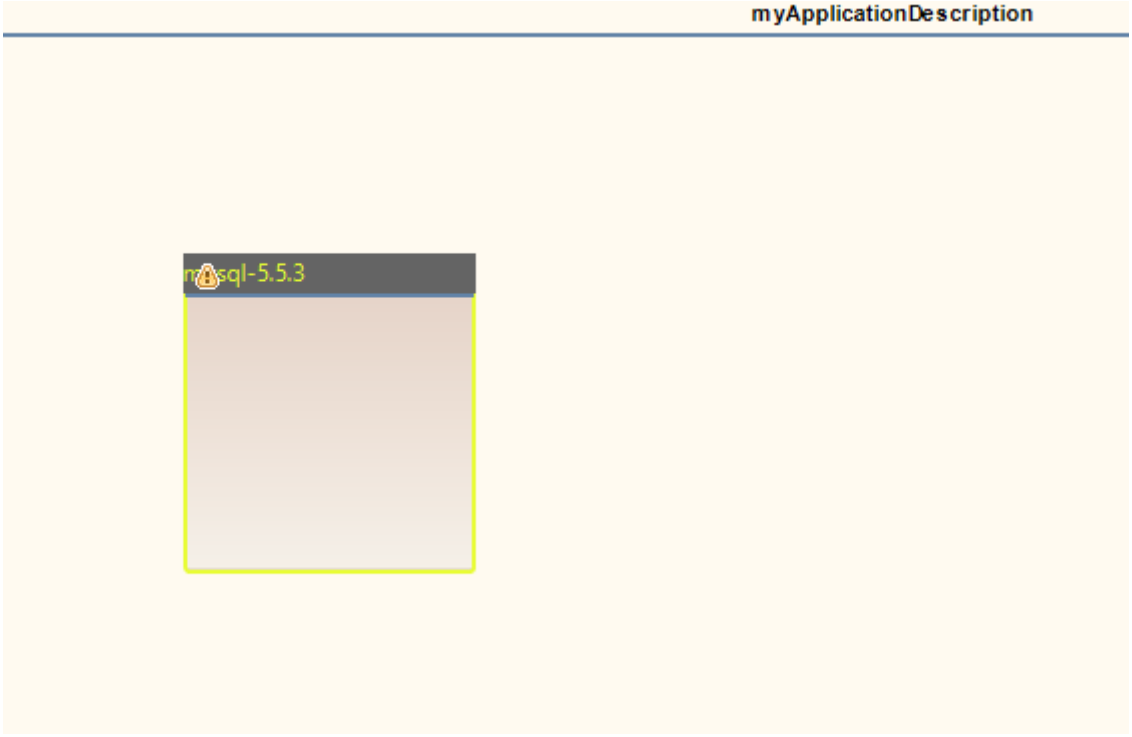
Στη συνέχεια για να μπορεί να χρησιμοποιηθεί από τους χρήστες, η περιγραφή TOSCA αυτή, εισάγεται σαν στοιχείο της παλέτας του CAMF, όπου από εκεί οι χρήστες μπορούν να το εισάγουν στις περιγραφές τους κάνοντας απλά drag and drop στον

Application Modeler. Για να ξεχωρίζουν τα artifacts που προέρχονται από το Marketplace από τα υπόλοιπα στοιχεία περιγραφών του CAMF δηλώθηκε στην κλάση του, ToscaToolBehaviorProvider, που είναι υπεύθυνη για τη διαχείριση των εργαλείων της παλέτας του CAMF, μέσω της μεθόδου addRepositoryCompartment() η εισαγωγή ξεχωριστού τμήματος με το όνομα Ready-to-use artifacts (Σχήμα 4.19). Η μέθοδος αυτή, είναι υπεύθυνη, επίσης στο να εντοπίζει όλα τα artifact που έχει ετοιμάσει ο χρήστης για χρήση και τα τοποθετεί στην παλέτα.



Σχήμα 4. 19 Το τμήμα παλέτας του CAMF για τα Artifacts του Marketplace

Τέλος για τη σχηματική παρουσίαση των artifacts στον Application Modeler, δημιουργήθηκε η κλάση AddReadyToUseArtifactFeature, της οποίας κύρια δουλειά είναι η δημιουργία των σχηματικών αναπαραστάσεων της περιγραφής TOSCA των artifacts του Marketplace. Για να επιτευχθεί αυτό, μέσω της μεθόδου canAdd της κλάσης, γίνεται έλεγχος αν το στοιχείο που τοποθετήθηκε ή θα τοποθετεί είναι στιγμιότυπο της κλάσης CreateReadyToUseArtifactFeature και αν ναι, τότε εισάγει στον Modeler το σχήμα που καθορίζεται στη μέθοδο add τη κλάσης.



Σχήμα 4. 20 Σχηματική αναπαράσταση artifact στον Modeler

4.3 Σενάρια Χρήσης

Για καλύτερη κατανόηση των λειτουργιών του Marketplace plugin, σε αυτή την ενότητα παρουσιάζονται διαγραμματικά σε UML και με συνοδευτικά επεξηγηματικά screenshots, τα βασικά σενάρια χρήσης. Τα σενάρια χρήσης έχουν χωριστεί σε δύο κατηγορίες:

- Σενάρια χρήσης του χρήστη
- Σενάρια χρήσης του διαχειριστή.

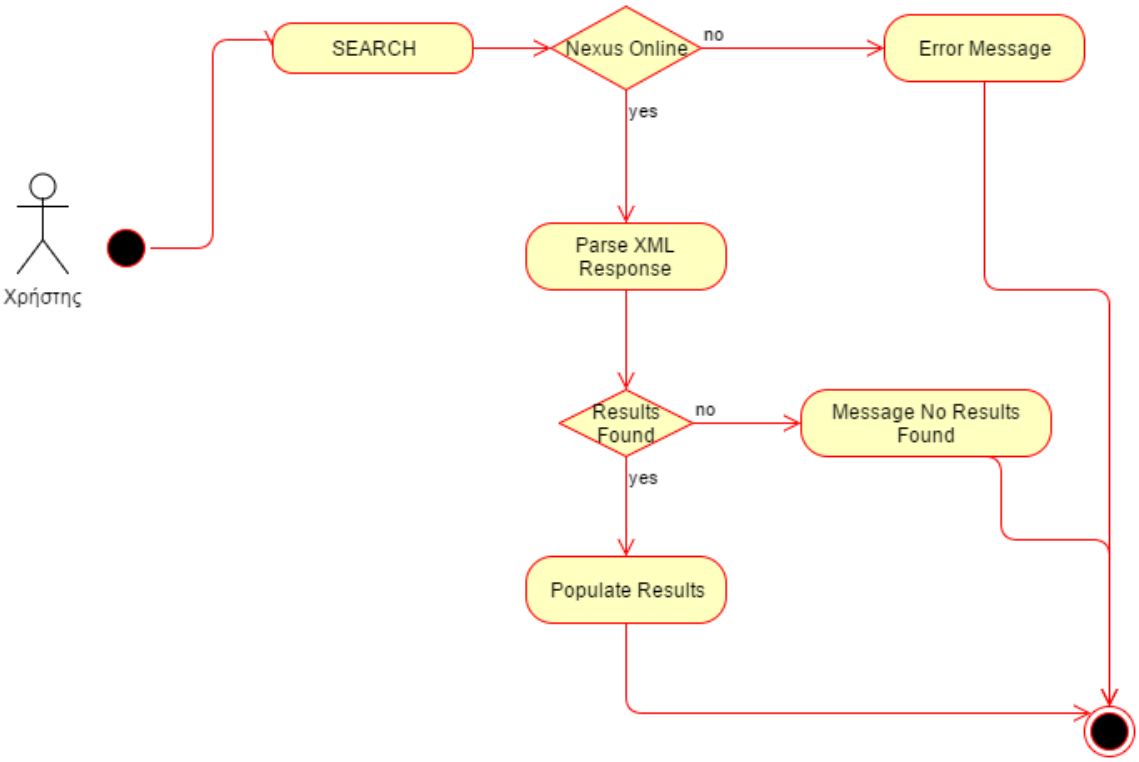
4.3.1 Σενάρια Χρήσης Χρήστη

Χρήστης, στο περιβάλλον χρήσης του Marketplace plugin, είναι αυτός, που χρησιμοποιεί το εργαλείο, για να επιτελέσει τις ενέργειες search aartifact από το repository, download artifact από το repository, configuration του artifact και εισαγωγή του στην παλέτα και τέλος χρήση του artifact στην περιγραφή της Cloud εφαρμογής του. Για κάθε μια από τις ενέργειες αυτές παρουσιάζεται το σενάριο χρήσης της. Πιο συγκεκριμένα, στην περιγραφή σεναρίων για το χρήστη που ακολουθεί, υποθέτουμε ότι

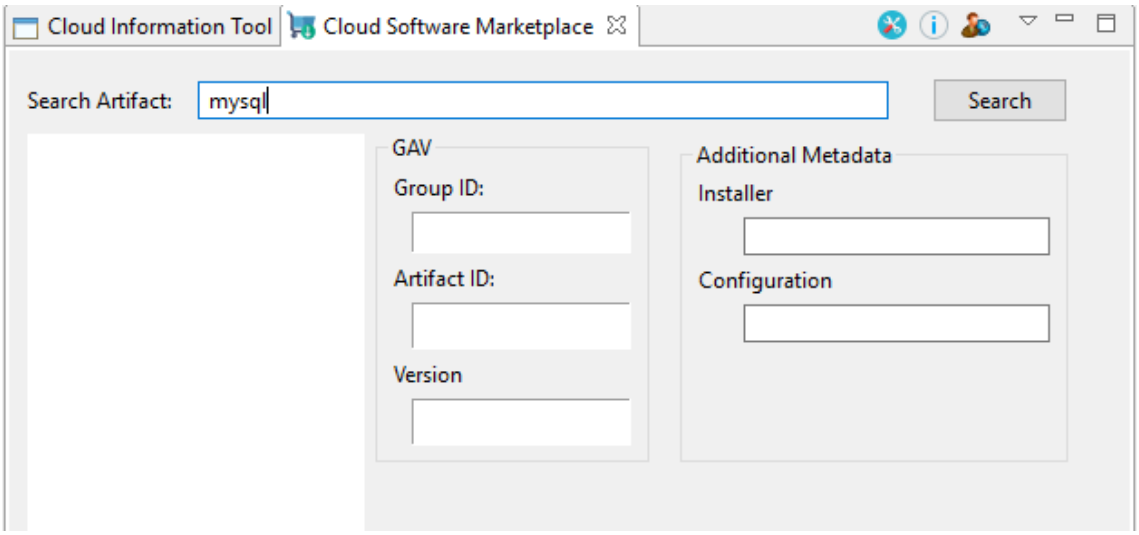
ο χρήστης επιθυμεί να δημιουργήσει μια περιγραφή Cloud εφαρμογής στην οποία θα κάνει χρήση μίας βάσης δεδομένων MySQL, χρησιμοποιώντας το Marketplace.

4.3.1.1 Σενάριο Χρήσης Αναζήτησης

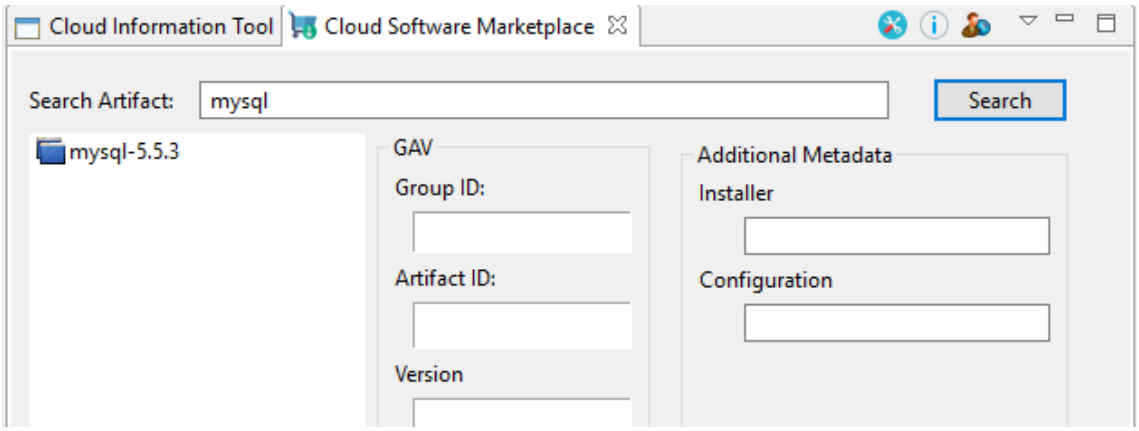
Αρχικά ο χρήστης θα πρέπει να κάνει χρήση της ενέργειας αναζήτηση που προσφέρει το Marketplace plugin για να εντοπίσει το artifact που τον ενδιαφέρει που στη συγκεκριμένη περίπτωση είναι η MySQL.



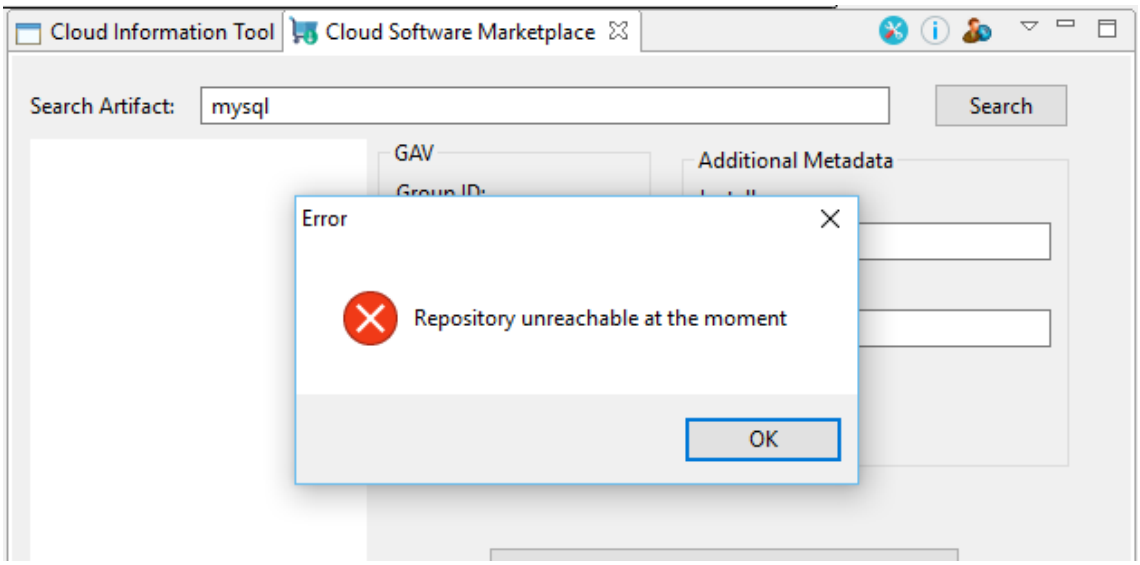
Σχήμα 4. 21 Διάγραμμα καταστάσεων σεναρίου χρήσης αναζήτηση



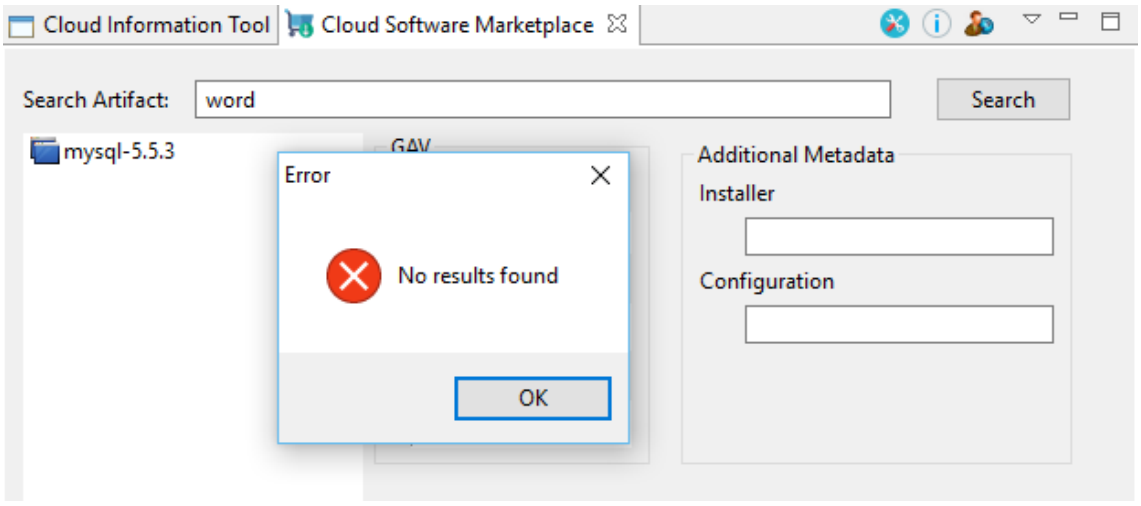
Σχήμα 4. 22 Εισαγωγή ονόματος artifact που οχρήστης επιθυμεί να αναζητήσει



Σχήμα 4. 23 Με το πάτημα του κουμπιού Search αν υπάρχουν αποτελέσματα δημοσιεύονται στη λίστα



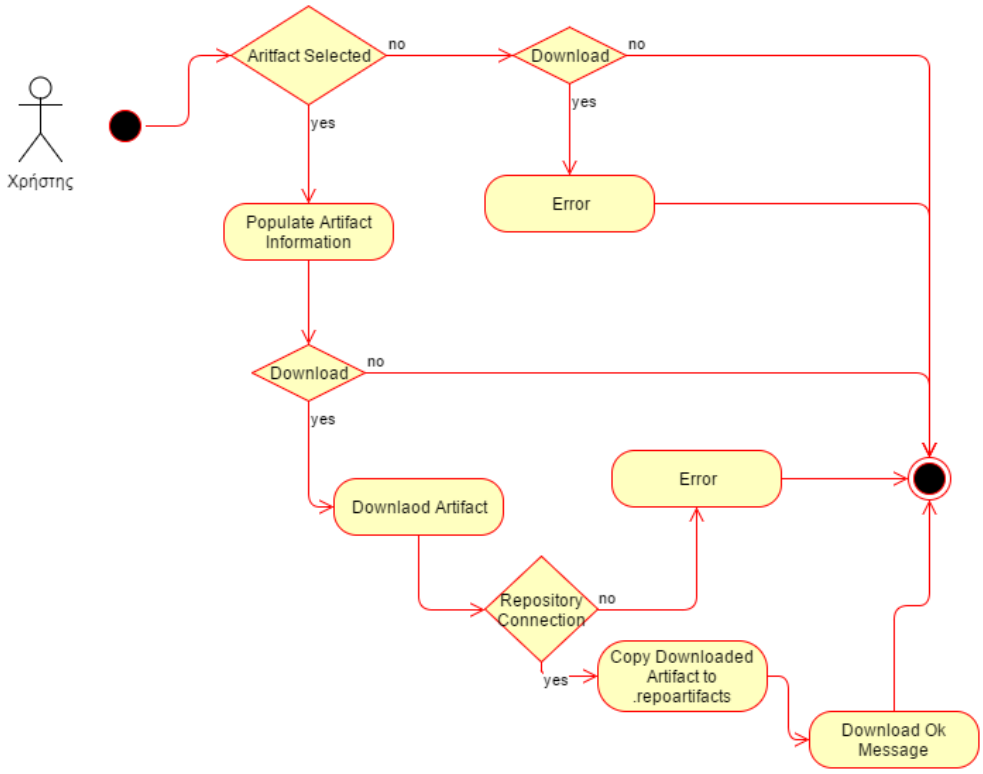
Σχήμα 4. 24 Σε περίπτωση που δεν βρεθεί το repository εμφανίζει μήνυμα λάθους.



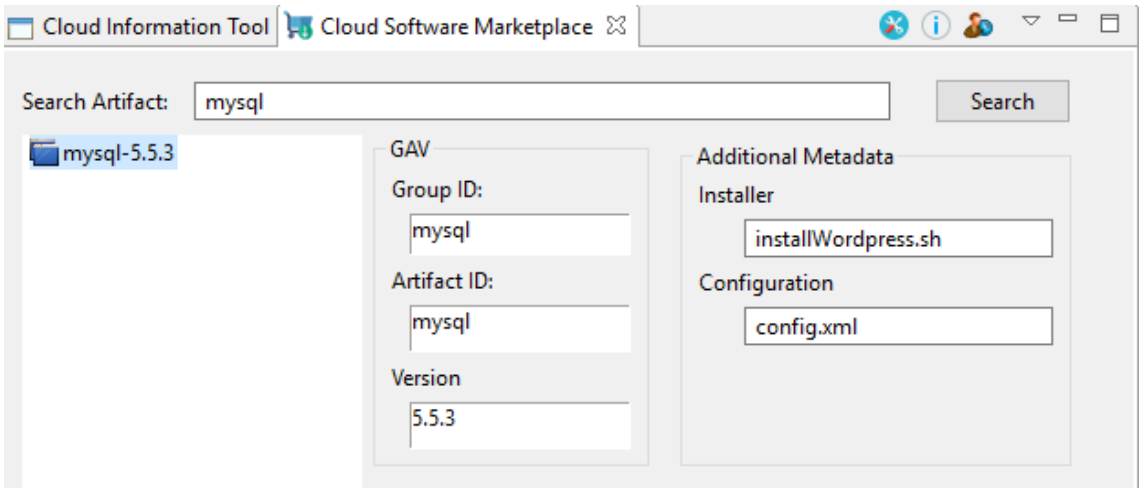
Σχήμα 4. 25Στην περίπτωση που η αναζήτηση δεν επιστρέψει κάποιο αποτέλεσμα εμφανίζεται το μήνυμα λάθους No results found

4.3.1.2 Σενάριο Χρήσης Download

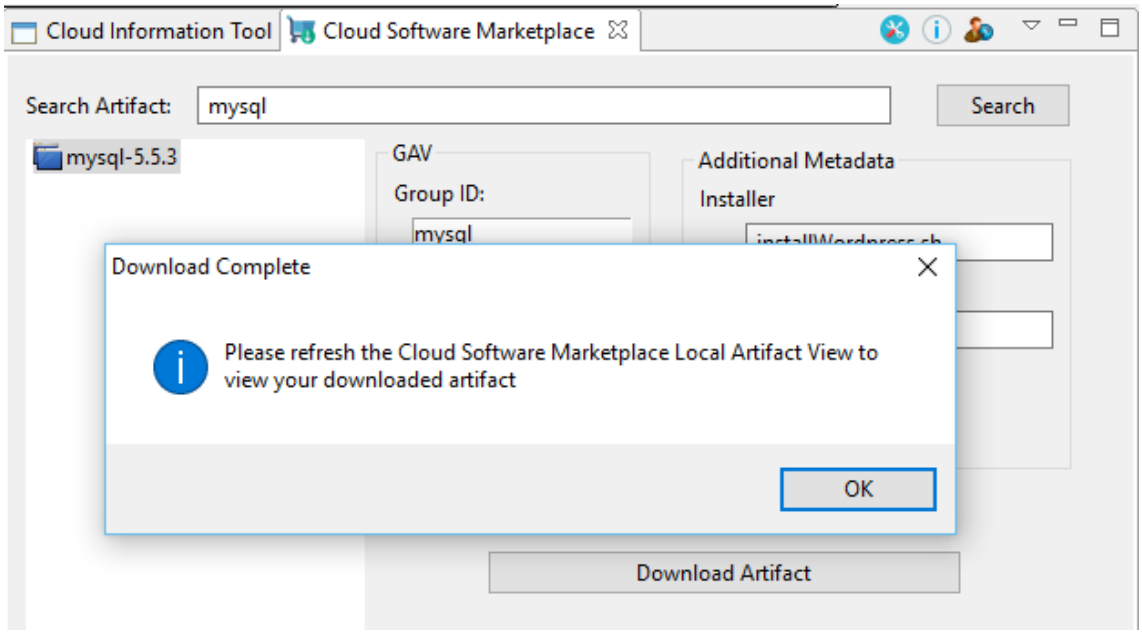
Σε αυτό το σενάριο αφού ο χρήστης έχει βρει το artifact που θα χρησιμοποιήσει το επόμενο στάδιο είναι να το κατεβάσει.



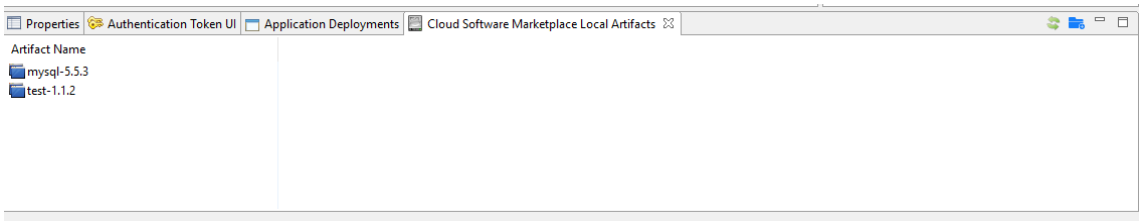
Σχήμα 4. 26 Διάγραμμα καταστάσεων σεναρίου χρήσης Download



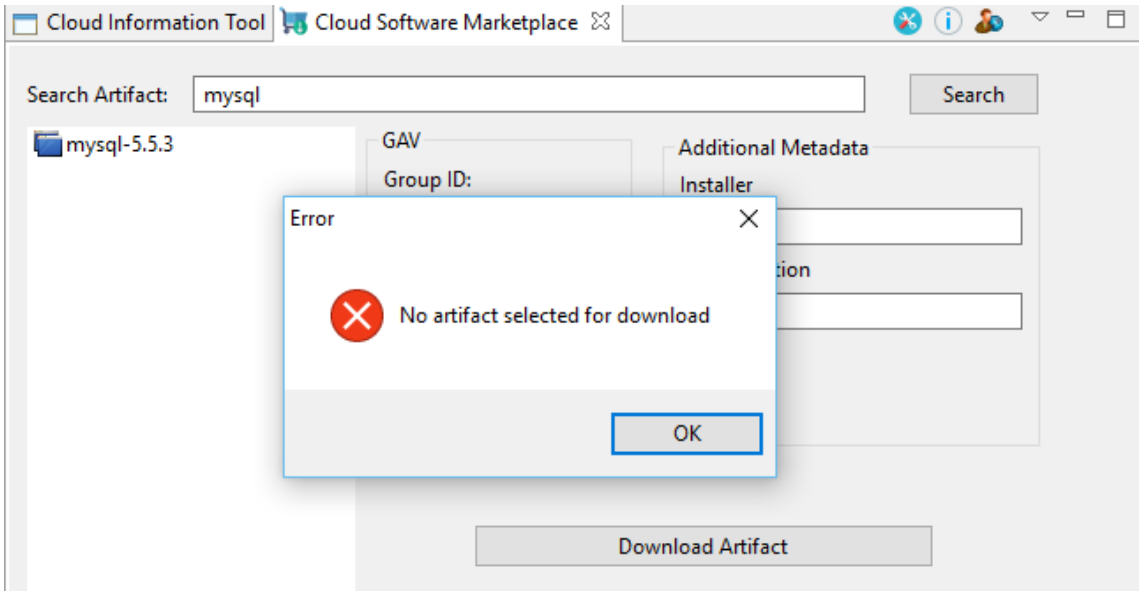
Σχήμα 4. 27 Εμφάνιση πληροφοριών επιλεγμένου artifact



Σχήμα 4. 28 Έχοντας επιλεγμένο το artifact από τη λίστα, με το πάτημα του κουμπιού Download το artifact θα κατεβεί στο φάκελο .repoartifacts και ο χρήστης θα ενημερωθεί με το κατάλληλο μήνυμα ότι η διαδικασία download έχει ολοκληρωθεί επιτυχώς.



Σχήμα 4. 29 Χρησιμοποιώντας το view όπου εμφανίζονται τα artifacts στο local file system, ο χρήστης μπορεί να εντοπίσει το artifact που κατέβασε.

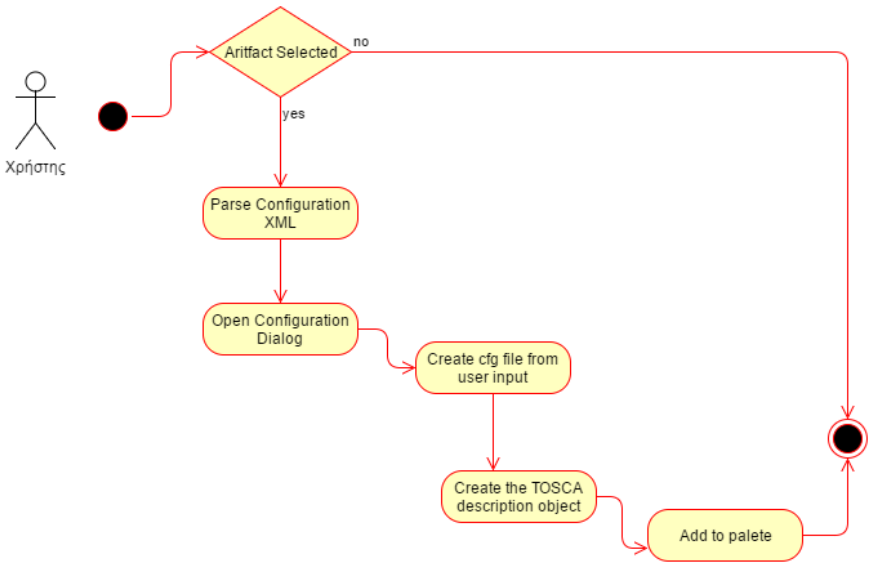


Σχήμα 4. 30 Όταν ο χρήστης επιχειρήσει να πατήσει το κουμπί download artifact χωρίς να έχει επιλέξει κάποιο artifact από τη λίστα, του εμφανίζεται ανάλογο μήνυμα λάθους

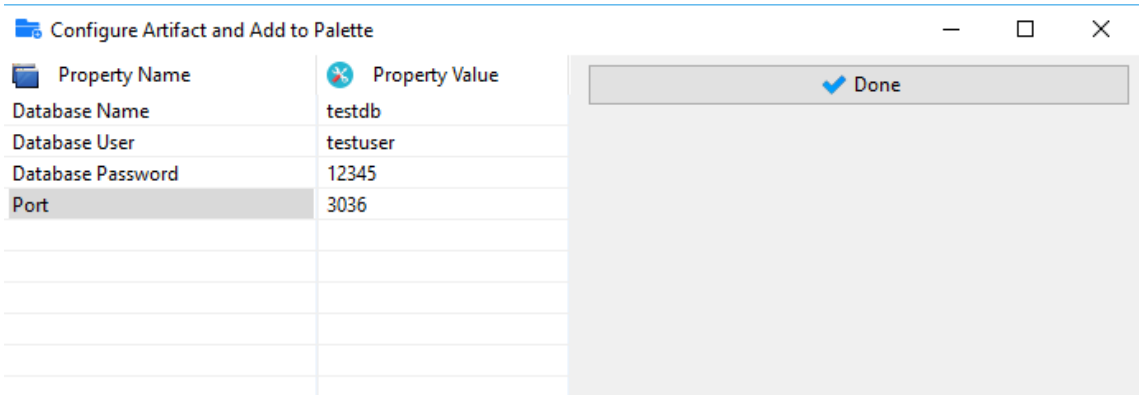
Στην περίπτωση όπου κατά τη διάρκεια του download, συμβεί κάτι και χαθεί η σύνδεση με το repository, το plugin θα εμφανίσει μήνυμα λάθους ανάλογο με εκείνο του Σχήματος 4.24.

4.3.1.3 Σενάριο Χρήσης Configuration

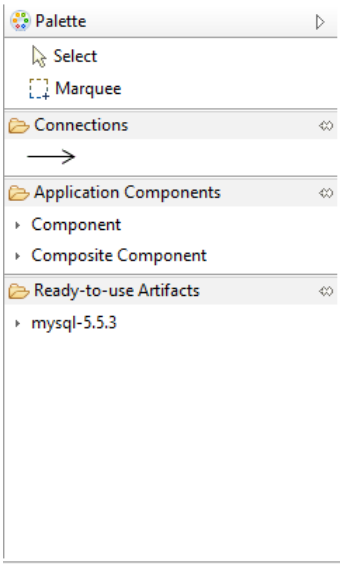
Σε αυτό το σενάριο ο χρήστης έχει ήδη κατεβάσει το artifact και θα δώσει τιμές στις configurable παραμέτρους του artifact, για να δημιουργηθεί το TOSCA TNodeTemplateExtension και τα TDeploymentArtifacts αντικείμενα της περιγραφής και θα εισαχθεί στην παλέτα του CAMF.



Σχήμα 4. 31 Διάγραμμα καταστάσεων σεναρίου χρήσης Configure



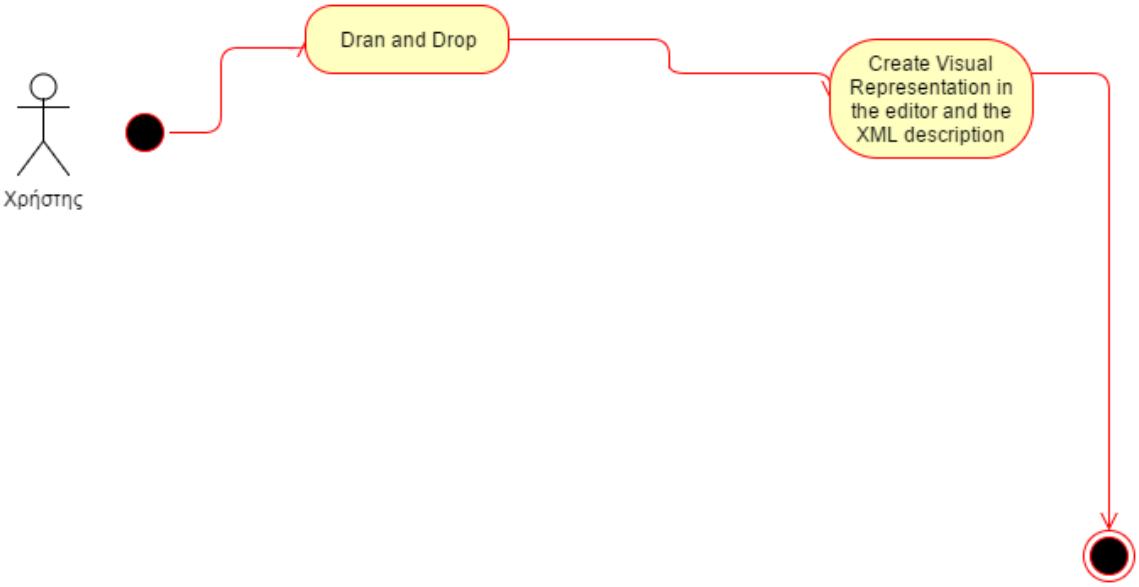
Σχήμα 4. 32 Με επιλεγμένο το artifact που θέλει χρήστης να χρησιμοποιήσει και πατώντας το κουμπί configure από το μενού ενεργειών του view των local artifacts, ανοίγει το παράθυρο configuration, στο οποίο ο χρήστης εισάγει τιμές στις configurable παραμέτρους



Σχήμα 4. 33 Μετά την ολοκλήρωση του configuration δημιουργείται στην παλέτα το TOSCA αντικείμενο που αντιπροσωπεύει τη περιγραφή του συγκεκριμένου artifact

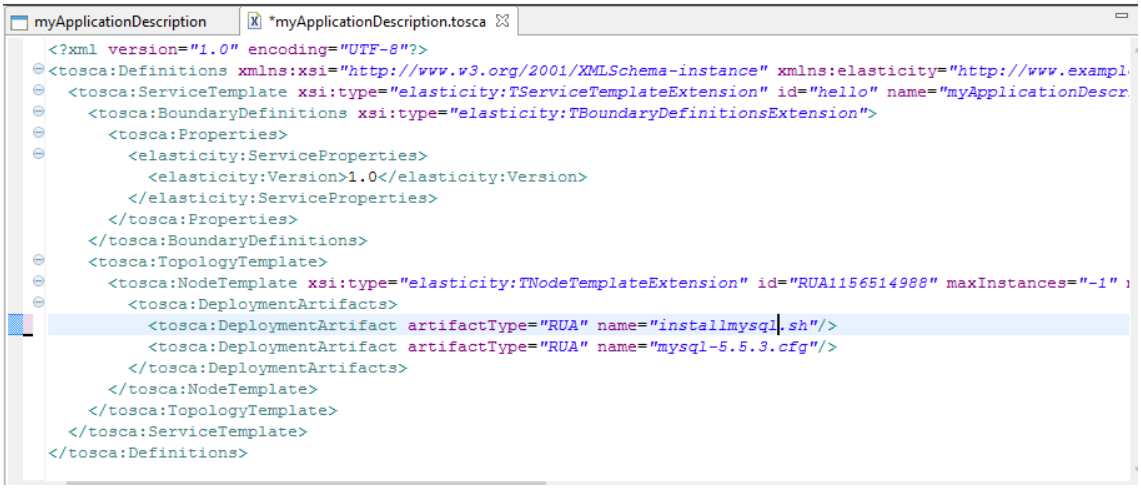
4.3.1.4 Σενάριο Χρήσης εισαγωγής artifact στη περιγραφή

Σε αυτό το σημείο ο χρήστης μπορεί να εισάγει στην περιγραφή της Cloud εφαρμογής του στον Application Modeler του CAMF το artifact που έχει ετοιμάσει.



Σχήμα 4. 34 Διάγραμμα καταστάσεων σεναρίου χρήσης εισαγωγής artifact

Σχήμα 4. 35 Ο χρήστης με drag and drop εισάγει στην περιγραφή του το artifact.



Σχήμα 4. 36 Περιγραφή TOSCA που δημιουργείται από την εισαγωγή του artifact

4.3.2 Σενάρια Χρήσης διαχειριστή

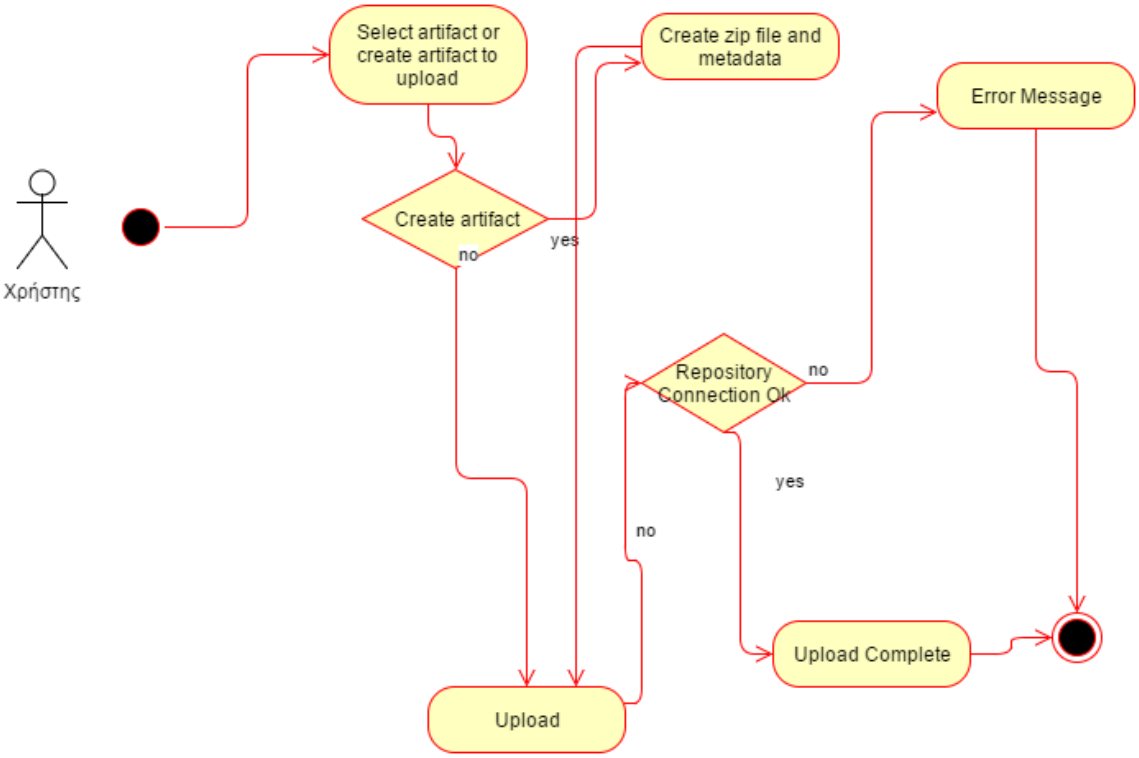
Ο ρόλος του διαχειριστή του Marketplace περιλαμβάνει τις εξής λειτουργίες:

- Δημιουργία και upload artifact στο repository
- Διαγραφή artifact από το repository.

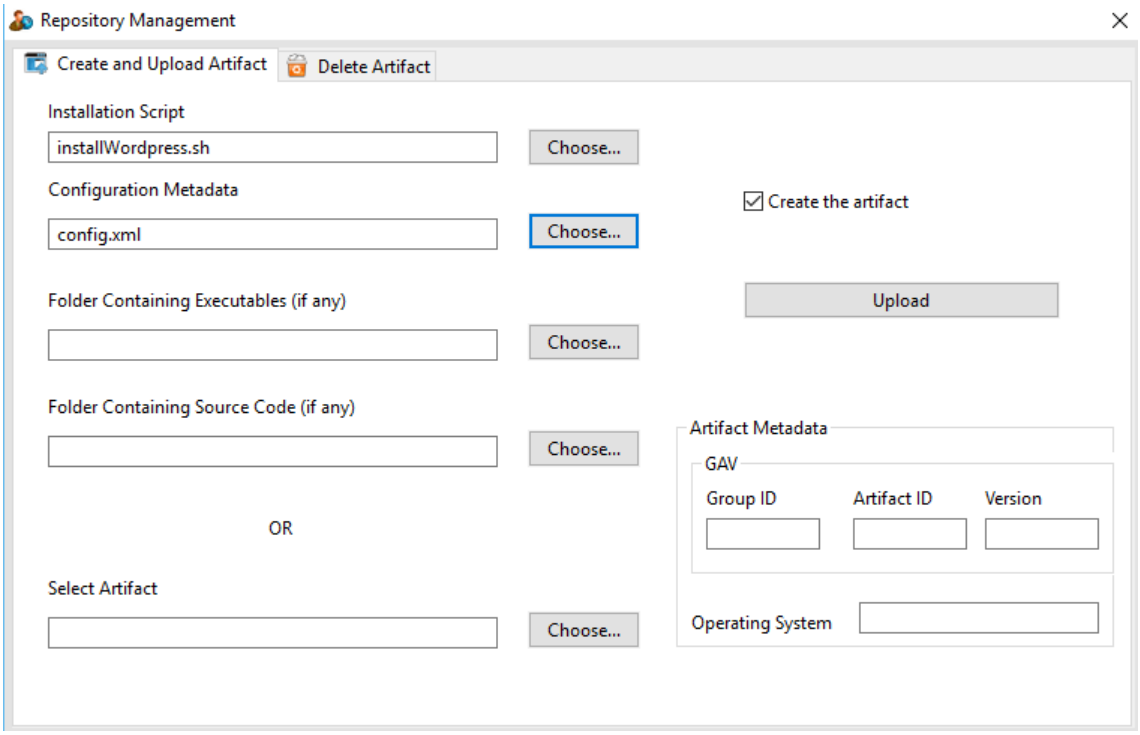
Για να μπορέσει κάποιος να χρησιμοποιήσει τις λειτουργίες της οθόνης του διαχειριστή, πρέπει να δώσει τα διαπιστευτήρια του, που τον αναγνωρίζουν ως διαχειριστή του Nexus repository. Στο σενάριο upload, γίνεται η υπόθεση ότι τα artifacts που γίνονται upload στο repository είναι ορθά και ακολουθούν τις προδιαγραφές του μοντέλου τωνartifacts που ορίστηκε στο Κεφάλαιο 3.

4.3.2.1 Σενάριο Χρήσης δημιουργίας artifact και upload

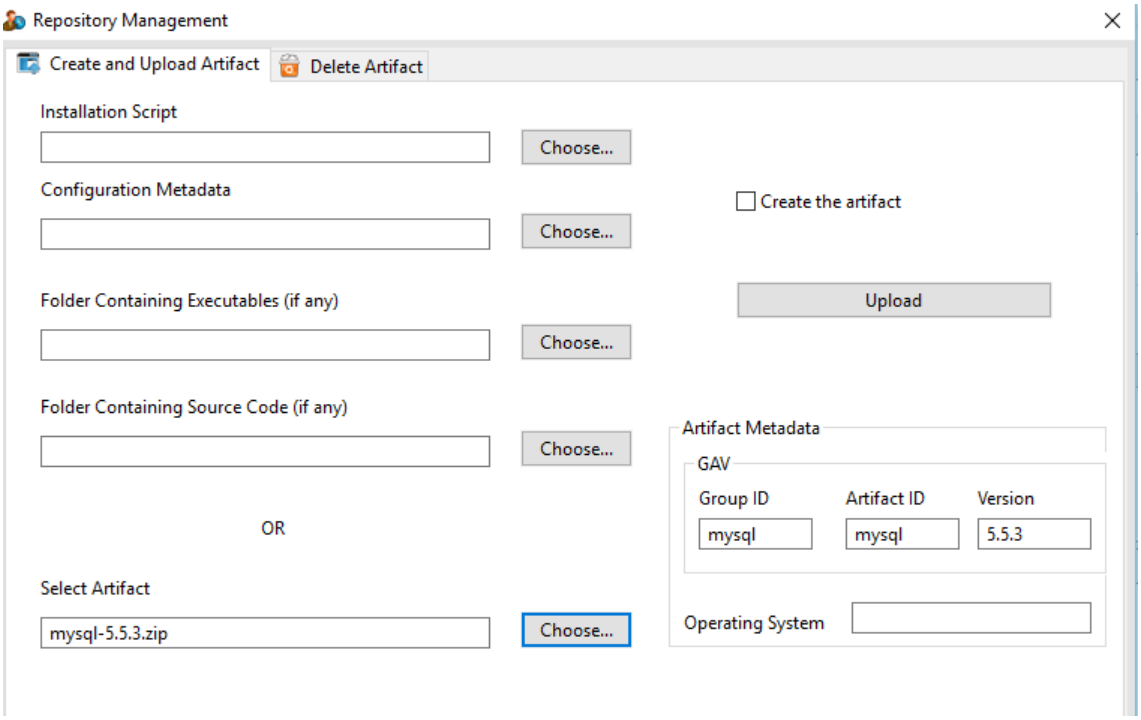
Σε αυτό το σενάριο, ο χρήστης, σαν διαχειριστής του συστήματος πλέον, επιθυμεί να δημιουργήσει και να ανεβάσει στο repository του ένα νέο artifact.



Σχήμα 4. 37Διάγραμμα καταστάσεων σεναρίου χρήσης δημιουργίας και ανεβάσματος artifact

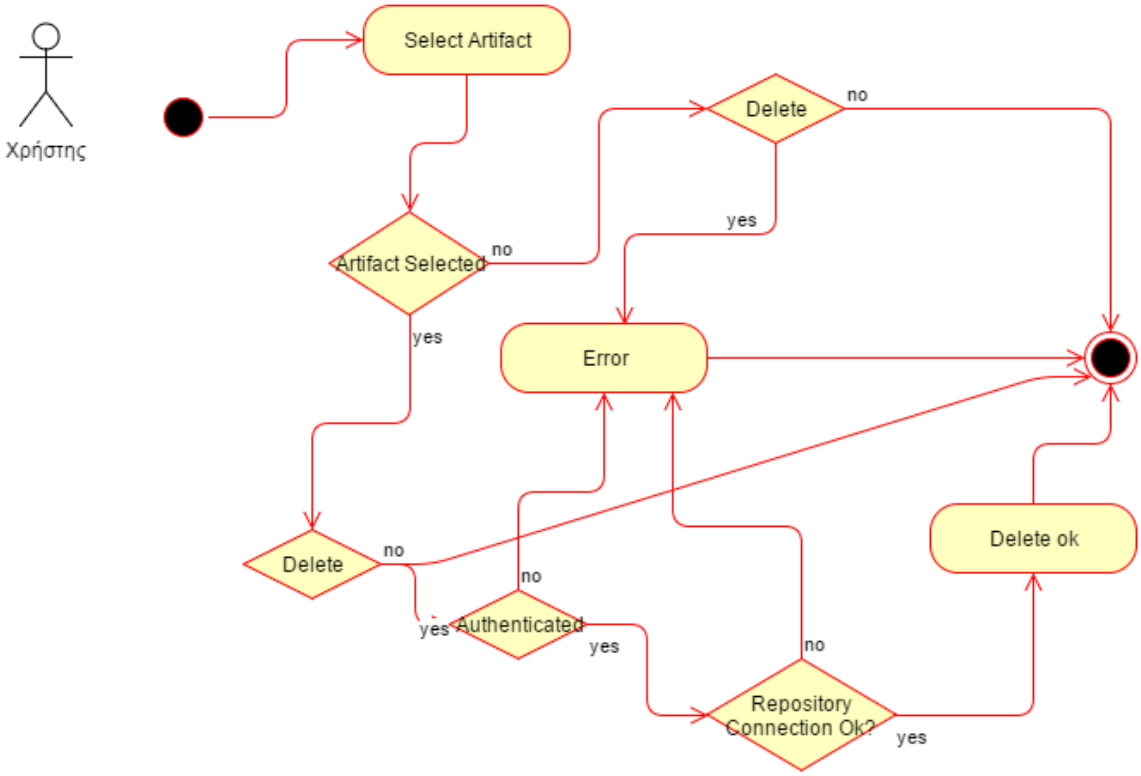


Σχήμα 4. 38 Ο χρήστης εισάγει στα κατάλληλα πεδία, τα αρχεία που αποτελούν το artifact του. Πρέπει να δώσει τιμές και στις παραμέτρους GAV.Με το πάτημα του κουμπιού upload δημιουργείται το zipped artifact και ανεβαίνει το artifact στο repository

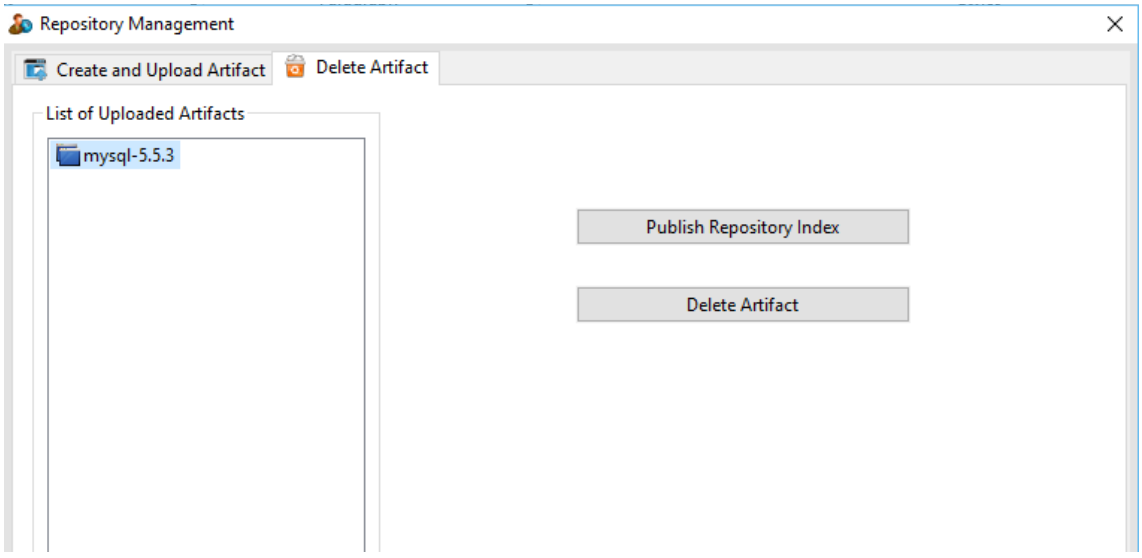


Σχήμα 4. 39 Στην περίπτωση που ο χρήστης έχει ήδη έτοιμο το artifact, το εισάγει στο κατάλληλοπεδίο, αποεπιλέγει την επιλογή create the artifact. Τα στοιχεία GAV μαντέυονται από το όνομα του αρχείου.

4.3.2.2 Σενάριο Χρήσης διαγραφής artifact από το repository



Σχήμα 4. 40 Διάγραμμα καταστάσεων σεναρίου χρήσης διαγραφής artifact



Σχήμα 4. 41 Ο Χρήστης διαχειριστής επιλέγει το artifact που θέλει να σβήσει από το repository του και ακολούθως πατάει delete

4.4 Διαχείριση λαθών

Για λόγους ευχρηστίας, το plugin είναι σε θέση να εντοπίζει τυχόν λάθη που προκύπτουν κατά τη διάρκεια της χρήσης του. Όταν ένα τέτοιο λάθος προκύψει στην ομαλή λειτουργία του plugin, ο χρήστης ενημερώνεται αμέσως με το κατάλληλο μήνυμα λάθους.

Ο τρόπος με τον οποίο γίνεται η διαχείριση λαθών, είναι με τον ορισμό της κλάσης RepoExceptions (Σχήμα 4.42).

```
1  package cy.ac.ucy.linc.CloudSoftwareRepo.Exceptions;
2
3  public class RepoExceptions extends Exception {
4
5      /**
6       *
7       */
8      private static final long serialVersionUID = 1L;
9
10     public RepoExceptions() {
11         super();
12     }
13
14     public RepoExceptions(String message){
15         super(message);
16     }
17 }
18
```

Σχήμα 4. 42 Η κλάση RepoExceptions

Όταν μια λειτουργία του Marketplace plugin εντοπιστεί ότι πετάει κάποιο exception, στο catch κομμάτι του κώδικα, το API, μετασχηματίζει αυτό το Exception σε RepoException με το ανάλογο μήνυμα. Έτσι, το plugin, γνωρίζει ότι , όταν κάποιο exception προέρχεται από την κλάση RepoException οφείλει να ενημερώσει το χρήστη.

```
/**
 * @throws RepoExceptions
 *
 */
public void uploadArtifact(Map<String, String> parameters)
        throws RepoExceptions {
    // TODO Auto-generated method stub

    for (Map.Entry<String, String> entry : parameters.entrySet()) {
        System.out.println(entry.getKey() + " " + entry.getValue());
    }

    try {
        cHttp.addPart(parameters);
        cHttp.CloudHttpPostRequest(CloudSoftwareRepoConstants.NEXUS_URL
                + CloudSoftwareRepoConstants.NEXUS_UPLOAD_ARTIFACT);
    } catch (UnsupportedEncodingException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (RepoExceptions e) {
        // TODO Auto-generated catch block
        throw new RepoExceptions("Could not complete the upload action");
    }

}
```

Σχήμα 4. 43 Παράδειγμα Repoexception στην κλάση uploadArtifact

Ακόμη, ένας άλλος τρόπος, αυτή τη φορά αποτρεπτικός, διαχείρισης λαθών γίνεται μέσα από το ίδιο το user interface των views των plugins.

Στο view του χρήστη, όταν αυτός προσπαθήσει να πατήσει το κουμπί Download Artifact χωρίς προηγουμένως να έχει επιλέξει κάποιο artifact από τη λίστα, το plugin θα το εντοπίσει, θα ενημερώσει το χρήστη με το κατάλληλο μήνυμα και δεν θα προβεί σε καμία άλλη ενέργεια.

Αντίστοιχα, στο view του διαχειριστή, κατά την ενέργεια upload, ο χρήστης έχει δύο επιλογές. Η πρώτη είναι να δημιουργήσει το artifact επιλέγοντας τα συστατικά του στοιχεία στα κατάλληλα πεδία και η δεύτερη είναι με το ανέβασμα του έτοιμου artifact. Το κλειδί για να γνωρίζει το plugin ποια μέθοδος upload χρησιμοποιείται είναι μέσω του check box create artifact. Στην περίπτωση που είναι επιλεγμένο, το user interface περιμένει από το χρήστη να συμπληρώσει τα κατάλληλα πεδία που είναι υποχρεωτικά για τη δημιουργία του artifact, ενώ στην περίπτωση που το check box δεν είναι επιλεγμένο, το user interface αναμένει ο χρήστης να του δώσει ένα zip αρχείο στο κατάλληλο πεδίο. Και στις δύο περιπτώσεις, όταν ο χρήστης δεν κάνει σωστές ενέργειες, λαμβάνει τα κατάλληλα μηνύματα.

```
} catch (IndexOutOfBoundsException ex){
    wShell = new Shell();
    MessageBox msBox = new MessageBox(wShell.getShell(),
            SWT.ICON_ERROR | SWT.OK);
    msBox.setText("Error");
    msBox.setMessage("No artifact selected for download");
    msBox.open();
```

Σχήμα 4. 44 Παράδειγμα διαχείρισης λαθών στο επίπεδο του GUI του plugin.

Κεφάλαιο 5

Αξιολόγηση του plugin

5.1 Μεθοδολογία Αξιολόγησης

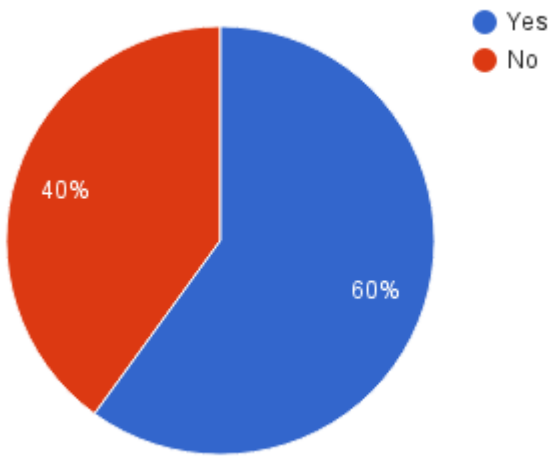
Μετά την ολοκλήρωση της υλοποίησης του plugin, ακολούθησε η αξιολόγηση του από πιθανούς μελλοντικούς χρήστες. Ζητήθηκε λοιπόν, από δείγμα 20 ατόμων, με αρκετά καλές γνώσεις τον τομέα της Πληροφορικής να δοκιμάσουν τις λειτουργίες του plugin και απαντώντας στο ερωτηματολόγιο του Παραρτήματος Α να δώσουν το feedback και τα σχόλιά τους.
Μετά από συναντήσεις με τα άτομα αυτά και αφού τους εξηγήθηκε αρχικά τι είναι το CAMF και ποιος είναι ο σκοπός του plugin που δημιουργήθηκε, τους ανατέθηκε να δημιουργήσουν μια απλή περιγραφή εφαρμογής ενός Apache Web Server και μιας βάσης δεδομένων MySQL στο CAMF χρησιμοποιώντας το Marketplace κάνοντας αναζήτηση, download και configuration και στη συνέχεια να εισάγουν στον editor του CAMF τα εκθέματα λογισμικού για τη δημιουργία της περιγραφής. Τέλος τους ζητήθηκε να δημιουργήσουν και να ανεβάσουν ένα artifact στο repository και ακολούθως να το διαγράψουν.
Στην ενότητα 5.2 παρουσιάζονται αναλυτικά τα αποτελέσματα της αξιολόγησης.

5.2 Παρουσίαση αποτελεσμάτων αξιολόγησης

Στις πρώτες δύο ερωτήσεις του ερωτηματολογίου έγινε προσπάθεια να καθοριστεί το προφίλ των ατόμων του δείγματος όσων αφορά τις γνώσεις τους σε θέματα που αφορούν την Πληροφορική και την εξοικείωση τους με την πλατφόρμα Eclipse. Σύμφωνα με τις απαντήσεις τους, όλοι τους, βαθμολογούν τις γνώσεις τους σε θέματα

Πληροφορικής σαν “Expert”, ενώ μόνο το 10% δεν είναι πολύ εξοικειωμένο με την πλατφόρμα Eclipse, μιας και δήλωσε ότι το χρησιμοποιεί “Slightly Often”.

Στην ερώτηση σχετικά με το αν τα άτομα του δείγματος έχουν χρησιμοποιήσει οποιονδήποτε από τους διαθέσιμους παρόχους υπηρεσιών Cloud για την ανάπτυξη εφαρμογών τους οι απαντήσεις τους παρουσιάζονται στο Σχήμα 5.1.



Σχήμα 5. 1 Αποτελέσματα Ερώτησης: Έχετε χρησιμοποιήσει ποτέ οποιονδήποτε πάροχο υπηρεσιών Cloud για την ανάπτυξη εφαρμογών σας?

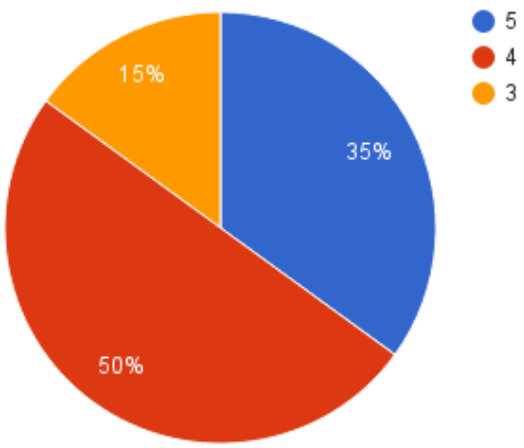
Ιδιαίτερη εντύπωση προκαλούν τα αποτελέσματα των ερωτήσεων 4,5 και 6. Οι ερωτήσεις αυτές προσπαθούν να ανακαλύψουν το κατά πόσο οι χρήστες είναι εξοικειωμένοι με τα Cloud Application Management Frameworks, ποια από αυτά γνωρίζουν και αν έχουν χρησιμοποιήσει κάποια. Παρόλο που το 60% των ερωτηθέντων απάντησε ότι έχει χρησιμοποιήσει κάποιον πάροχο υπηρεσιών Cloud για την ανάπτυξη εφαρμογών, μόνο το 16% αυτών γνωρίζει κάποιο Cloud Application Management Framework. Το Application Management Framework που δήλωσαν ότι γνωρίζουν είναι το Juju Charms, ενώ κανένας δεν έχει χρησιμοποιήσει κάποιο AMF για την ανάπτυξη εφαρμογών.

Στη συνέχεια ερωτηθήκαν αν πριν από τη δοκιμή γνώριζαν ή είχαν χρησιμοποιήσει το CAMF. Η ερώτηση αυτή προσπαθεί να ανακαλύψει χρήστες του CAMF οι οποίοι να το είχαν χρησιμοποιήσει πριν την εισαγωγή του Marketplace. Η άποψη αυτών των χρηστών είναι ιδιαίτερα σημαντική καθώς είναι και αυτοί που θα έχουν μια πιο πλήρη εικόνα των συνεπειών της εισαγωγής του Marketplace. Από το δείγμα μόνο ένας στους είκοσι το CAMF από πριν.

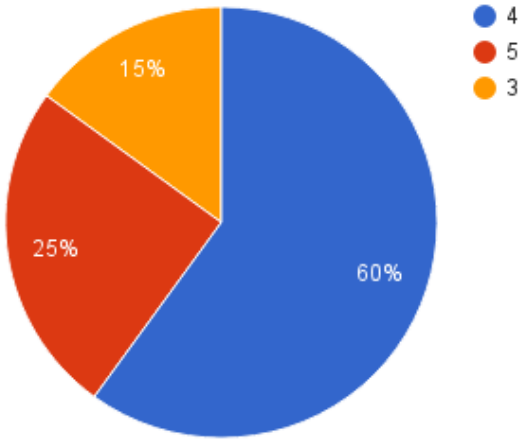
Τέλος, η τελευταία ερώτηση έχει ως στόχο να λάβει βαθμολογίες για μια ομάδα ποιοτικών χαρακτηριστικών του plugin. Τα ποιοτικά αυτά χαρακτηριστικά είναι:

- Ευκολία εκμάθησης
- Ευκολία χρήσης
- Χρησιμότητα του plugin
- Επίδοση του plugin
- Σχεδιασμός (γραφικό περιβάλλον)
- Διαδραστικότητα με το χρήστη
- Κατά πόσο το plugin εκπλήρωσε τις απαιτήσεις του χρήστη.

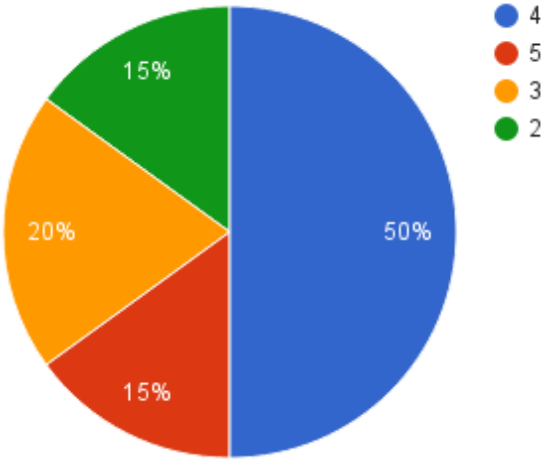
Η μέγιστη βαθμολογία που μπορούσε να βαθμολογήσει κάποιος μια συγκεκριμένη ομάδα ήταν το 5 ενώ η ελάχιστη το 1. Στα Σχήματα 5.2 έως το 5.8 φαίνονται οι βαθμολογίες των χαρακτηριστικών. Ιδιαίτερη αναφορά πρέπει να γίνει στο χρήστη που γνώριζε από πριν το CAMF και βαθμολόγησε όλες τις ομάδες χαρακτηριστικών με 5.



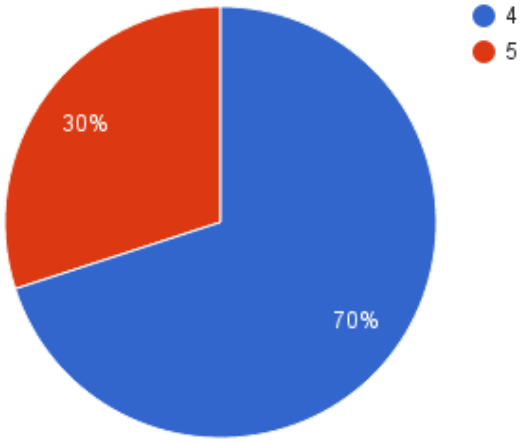
Σχήμα 5. 2 Βαθμολογία Ευκολία Εκμάθησης



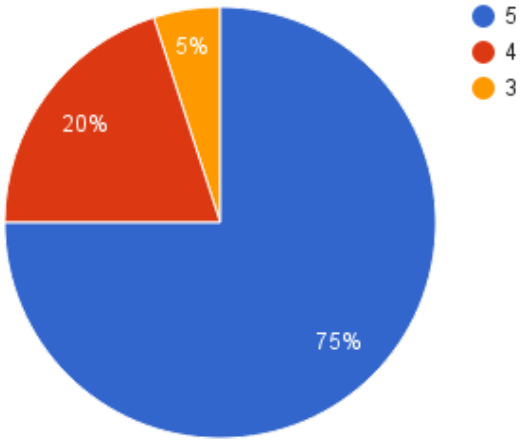
Σχήμα 5. 3 Βαθμολογία Ευκολίας Χρήσης



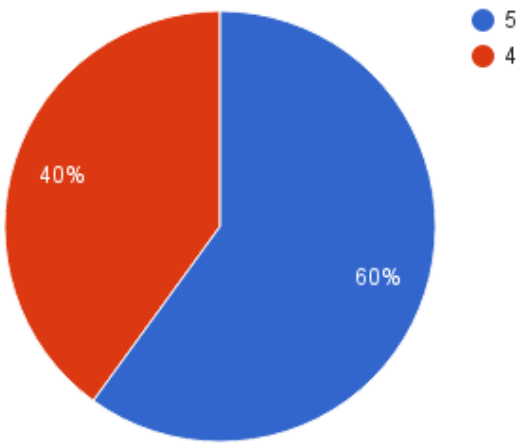
Σχήμα 5. 4 Βαθμολογία Χρησιμότητας



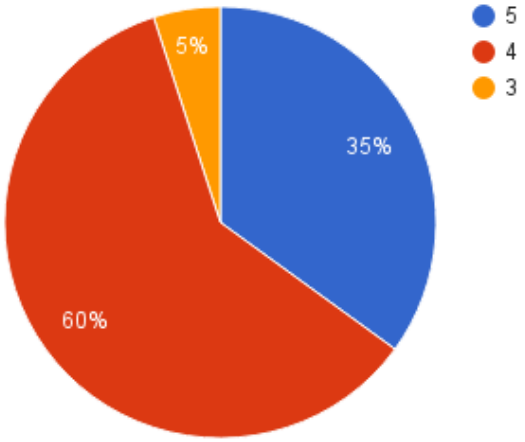
Σχήμα 5. 5 Βαθμολογία Επίδοσης



Σχήμα 5. 6 Βαθμολογία Σχεδιασμού



Σχήμα 5. 7 Βαθμολογία Διαδραστικότητας με το χρήστη



Σχήμα 5. 8 Βαθμολογίας Εκπλήρωσης Απαιτήσεων Χρήστη

Κεφάλαιο 6

Συμπεράσματα και Μελλοντική Εργασία

6.1 Συμπεράσματα

Διανύουμε την εποχή όπου οι τεχνολογίες cloud κερδίζουν ολοένα και περισσότερο έδαφος. Οι χρήστες των υπηρεσιών cloud στις μέρες μας θέλουν να χρησιμοποιούν καλύτερες και ποιοτικότερες υπηρεσίες, οι προγραμματιστές εφαρμογών θέλουν να παραδίδουν καλύτερο λογισμικό πιο γρήγορα. Αυτό συνεπάγεται ότι οι χρήστες πρέπει να έχουν στη διάθεσή τους εργαλεία τα οποία να αυτοματοποιούν διαδικασίες οι οποίες είναι επαναληπτικές και για τις οποίες ο χρήστης καταναλώνει αρκετό από τον πολύτιμο χρόνο του, κάνοντας πράγματα τα οποία αρκετές φορές δεν κατανοεί πλήρως.
Όσον αφορά το CAMF με την προσθήκη του Software Marketplace, μία διαδικασία η οποία ήταν χρονοβόρα και όχι ιδιαίτερα φιλική προς τον χρήστη, αφού απαιτούσε από αυτόν με αρκετό κόπο να εισάξει τα software artifacts που θα ήθελε να κάνει deploy στο cloud, έγινε πολύ πιο απλή και γρήγορη. Ακόμη με τη δυνατότητα που παρέχει το plugin για pre-deployment configuration των artifacts, ο χρήστης πλέον μέσα σε ελάχιστα λεπτά μπορεί να τρέχει στην υποδομή του παρόχου του την εφαρμογή του, όσο περίπλοκη και αν είναι η περιγραφή της.
Ακόμη, η εισαγωγή του Software Marketplace στο CAMF, ενδυναμώνει τα χαρακτηριστικά και τη λειτουργικότητα του σε σχέση με τα αντίστοιχα λογισμικά και εξαλείφει μία προς μία τις αδυναμίες του.

6.2 Μελλοντική Εργασία

Γενικά, ο τομέας των τεχνολογιών cloud, όπως κάθε τομέας της Πληροφορικής και των νέων τεχνολογιών, εξελίσσεται πολύ γρήγορα. Καθώς οι εξελίξεις προχωρούν πολύ γρήγορα, το ίδιο γρήγορα αυξάνονται και οι απαιτήσεις των υπαρχόντων και μελλοντικών χρηστών. Μία από αυτές τις απαιτήσεις, είναι η απαίτηση για τη χρήση ποιοτικά καλύτερου λογισμικού.

Όπως αναφέρθηκε και στο Κεφάλαιο 3, στο παρών στάδιο στο plugin δεν γίνεται κάποιος ποιοτικός έλεγχος των εκθεμάτων λογισμικού τα οποία ανεβαίνουν στο repository. Απλά υποθέσαμε ότι οι χρήστες ανεβάζουν εκθέματα λογισμικού τα οποία είναι σωστά και συμβατά με την όλη λειτουργία του plugin. Σε αυτόν τον τομέα θα μπορούσε να υπάρξει μία συνεισφορά στο plugin. Θα μπορούσε να αναπτυχθεί ένα σύστημα ελέγχου και αξιολόγησης των artifacts που ανεβαίνουν στο repository. Ένα τέτοιο σύστημα θα βασιζόταν κυρίως στο feedback και βαθμολογίες των χρηστών. Από αυτό θα μπορούσε να εξαχθεί το συμπέρασμα ότι ένα artifact είναι ποιοτικό και καλό.

Ένα άλλο σημείο το οποίο θα μπορούσε να δοθεί ιδιαίτερη βαρύτητα στη μελλοντική ανάπτυξη και βελτίωση του plugin, είναι η υλοποίηση του API του plugin και για κάποιο άλλο λογισμικό. Στο Κεφάλαιο 2 έγινε αναφορά σε άλλα Software Repositories όπως το Artifactory και το Archiva. Η μέχρι τώρα δομή του κώδικα του plugin επιτρέπει σε μελλοντικούς developers να υλοποιήσουν το API που έχει οριστεί συν όποιες άλλες βοηθητικές μεθόδους για να κάνουν το plugin να επικοινωνεί και με άλλα Software repositories πέραν του Nexus.

Βιβλιογραφία

[1] Oliver Kopp, Tobias Binz, Uwe Breitenbücher, and Frank Leymann, Winery – A Modeling Tool for TOSCA-based Cloud Applications

[2] N. Loulloudes, C. Sofokleous, D. Trihinas, M. D. Dikaiakos, and G. Pallis, "Enabling Interoperable Cloud Application Management through an Open Source Ecosystem.", IEEE Internet Computing. Vol. 19, Issue 3, May/June 2015, pp. 54-59.

[3] Manfred Moser, Tim O'Brien, Jason Van Zyl, Damian Bradicich, John Casey, Tamás Cservenák, Brian Demers, Brian Fox, Marvin Froeder, Anders Hammar, Rich Seddon, Peter Lynch, Juven Xu, "Repository Management with Nexus", https://books.sonatype.com/nexus-book/reference/preface.html, May 3, 2016

[4] C. Sofokleous, N. Loulloudes, D. Trihinas, G. Pallis and M. Dikaiakos, "c-Eclipse: An Open-Source Management Framework for Cloud Applications", EuroPar 2014 Parallel Procession 20th International Conference, Porto, Portugal, August 25-29, 2014.

[5] "Topology and Orchestration Specification for Cloud Applications Version 1.0", http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html ,November 25, 2013

[6] Juju Charms, https://jujucharms.com/

[7] AWS CloudFormation, https://aws.amazon.com/cloudformation/

[8] CSC Agility Platform, http://www.csc.com/cloud/offerings/53410/104965-csc_agility_platform_cloud_management

[9] Artifactory, https://www.jfrog.com/artifactory/

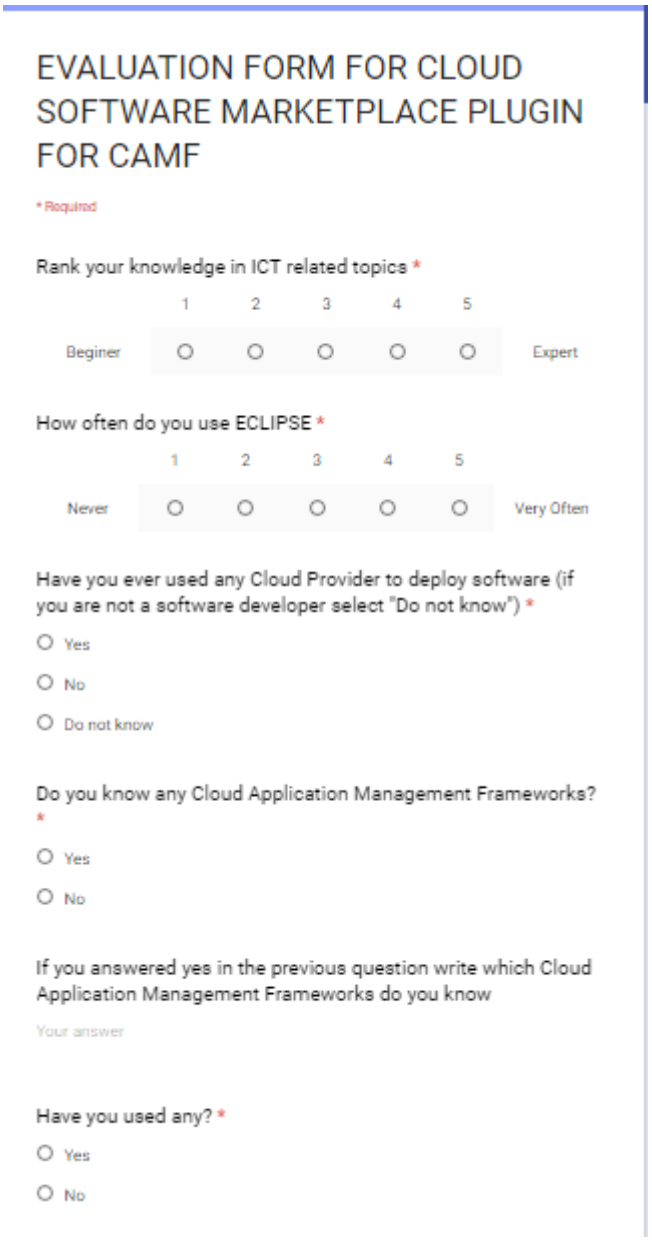
[10] Apache Archiva, http://archiva.apache.org/docs/2.2.0/tour/index.html#

[11] Eclipse Documentation, http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjface%2Futil%2FDelegatingDragAdapter.html

[12] Ort, Mehta, Java Architecture for XML Binding, http://www.oracle.com/technetwork/articles/javase/index-140168.html

Παράρτημα Α

Ερωτηματολόγιο που χρησιμοποιήθηκε για την αξιολόγηση του Plugin. Δημιουργήθηκε με το Google Forms.



EVALUATION FORM FOR CLOUD SOFTWARE MARKETPLACE PLUGIN FOR CAMF

* Required

Rank your knowledge in ICT related topics *

	1	2	3	4	5	
Beginer	○	○	○	○	○	Expert

How often do you use ECLIPSE *

	1	2	3	4	5	
Never	○	○	○	○	○	Very Often

Have you ever used any Cloud Provider to deploy software (if you are not a software developer select "Do not know") *

- ○ Yes
- ○ No
- ○ Do not know

Do you know any Cloud Application Management Frameworks? *

- ○ Yes
- ○ No

If you answered yes in the previous question write which Cloud Application Management Frameworks do you know

Your answer

Have you used any? *

- ○ Yes
- ○ No

If you answered yes in the previous question write which Cloud Application Management Frameworks have you used

Your answer

Have you used CAMF before *

- Yes
- No

Evaluate the Cloud Software Marketplace Plugin in terms of the following characteristics

	5	4	3	2	1
Easy to Learn					
Easy To Use					
Usefull					
Performace					
Design					
Interactive					
Achieved its requirements					

SUBMIT

Παράρτημα Β

Υλοποίηση API

```
public class CloudSoftwareRepo implements ICloudSoftwareRepo {

    public CloudHttp cHttp;
    private static String NEXUS_URL = "http://52.31.76.210:8081";
    private static String username;// = "admin";
    private static String password;// ="admin123";
    private static String NEXUS_DOWNLOAD_FOLDER;
    private static String ARTIFACTS_FOLDER;

    public static String getARTIFACTS_FOLDER() {
        return ARTIFACTS_FOLDER;
    }

    public static void setARTIFACTS_FOLDER(String aRTIFACTS_FOLDER) {
        ARTIFACTS_FOLDER = aRTIFACTS_FOLDER;
    }

    public static String getNEXUS_DOWNLOAD_FOLDER() {
        return NEXUS_DOWNLOAD_FOLDER;
    }

    public static void setNEXUS_DOWNLOAD_FOLDER(String nEXUS_DOWNLOAD_FOLDER) {
        NEXUS_DOWNLOAD_FOLDER = nEXUS_DOWNLOAD_FOLDER;
    }

    public static String getNEXUS_URL() {
        return NEXUS_URL;
    }

    public static void setNEXUS_URL(String nEXUS_URL) {
        NEXUS_URL = nEXUS_URL;
    }

    public static String getUsername() {
        return username;
    }
```

```
public static void setUsername(String username) {
    CloudSoftwareRepo.username = username;
}

public static String getPassword() {
    return password;
}

public static void setPassword(String password) {
    CloudSoftwareRepo.password = password;
}

public CloudSoftwareRepo() {
    super();
    // TODO Auto-generated constructor stub
    cHttp = new CloudHttp();

}
```

```
 /**
  *
  */
 public ArrayList<Artifacts> keywordSearch(String keyword)
         throws RepoExceptions {
     // TODO Auto-generated method stub
     try {
         CloudXMLParser parser = new CloudXMLParser();
         String srchResult = cHttp.CloudHttpGetRequest(keyword);
         ArrayList<Artifacts> lstArtifacts;

         lstArtifacts = parser.xmlParseKeywordSearch(srchResult);
         System.out.println(srchResult);
         return lstArtifacts;
     } catch (ParserConfigurationException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
     } catch (SAXException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
     } catch (IOException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
     } catch (RepoExceptions e) {
         // TODO Auto-generated catch block
         throw e;
         // System.out.println(e.getMessage());
     }
     return null;
 }
```

```
public void downloadArtifact(String url, String localArtifact)
        throws IOException, RepoExceptions {
    // TODO Auto-generated method stub

    URL download = new URL(url);

    InputStream is = null;
    FileOutputStream fos = null;

    try {
        URLConnection urlConn = download.openConnection();

        is = urlConn.getInputStream(); // get connection inputstream
        fos = new FileOutputStream(localArtifact); // open outputstream to
                                                   // local file

        int fileSize = urlConn.getContentLength();
        System.out.println("[*] File Size: "+fileSize);


        byte[] buffer = new byte[4096]; // declare 4KB buffer
        int len;
        int total=0;
        // while we have availble data, continue downloading and storing to
        // local file
        while ((len = is.read(buffer)) > 0) {
            System.out.println("[*] Downloaded: "+len+" Bytes");
            fos.write(buffer, 0, len);
            total+=len;
        }
        if(total<fileSize){
            throw new RepoExceptions("Something went wrong with connection and the download did not finish");
        }
    } catch (IOException e) {
        // TODO Auto-generated catch block
        throw new RepoExceptions("Something went wrong with connection and the download did not finish");
    } finally {
```

```
            try {
                if (is != null) {
                    is.close();
                }
            } finally {
                if (fos != null) {
                    fos.close();
                }
            }
        }

    }
```

```java
/**
 * @throws RepoExceptions
 *
 */
public void uploadArtifact(Map<String, String> parameters)
        throws RepoExceptions {
    // TODO Auto-generated method stub

    for (Map.Entry<String, String> entry : parameters.entrySet()) {
        System.out.println(entry.getKey() + " + " + entry.getValue());
    }

    try {
        cHttp.addPart(parameters);
        cHttp.CloudHttpPostRequest(CloudSoftwareRepoConstants.NEXUS_URL
                + CloudSoftwareRepoConstants.NEXUS_UPLOAD_ARTIFACT);
    } catch (UnsupportedEncodingException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (RepoExceptions e) {
        // TODO Auto-generated catch block
        throw new RepoExceptions("Could not complete the upload action");
    }

}

 public ArrayList<Artifacts> indexRepository() throws RepoExceptions {
     CloudXMLParser parser = new CloudXMLParser();
     try {
         String XMLIndex;

         /*
          * XMLIndex = cHttp
          * .CloudHttpGetRequest(CloudSoftwareRepoConstants.NEXUS_URL +
          * CloudSoftwareRepoConstants.NEXUS_INDEX_BROWSER);
          */

         XMLIndex = cHttp.CloudHttpGetRequest(NEXUS_URL + "/"
                 + CloudSoftwareRepoConstants.NEXUS
                 + CloudSoftwareRepoConstants.NEXUS_INDEX_BROWSER);

         System.out.println("INDEX: " + XMLIndex);
         ArrayList<Artifacts> index = new ArrayList<Artifacts>();
         ArrayList<String> arts = parser.xmlParseIndex(XMLIndex);
         for (int i = 0; i < arts.size(); i++) {
             // System.out.println("Art: "+arts.get(i));
             /*
              * ArrayList<Artifacts> tmp =
              * keywordSearch(CloudSoftwareRepoConstants.NEXUS_URL +
              * CloudSoftwareRepoConstants.NEXUS_KEYWORD_SEARCH +
              * arts.get(i));
              */
             ArrayList<Artifacts> tmp = keywordSearch(NEXUS_URL + "/"
                     + CloudSoftwareRepoConstants.NEXUS
                     + CloudSoftwareRepoConstants.NEXUS_KEYWORD_SEARCH
                     + arts.get(i));
             index.addAll(tmp);
             // System.out.println("Index Size: "+index.size());
         }
```

```
            for (int j = 0; j < index.size(); j++) {
                System.out.println("Artifact " + j + " : "
                        + index.get(j).artifactId);
            }
            return index;
        } catch (RepoExceptions e1) {
            // TODO Auto-generated catch block
            throw e1;
        } catch (ParserConfigurationException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (SAXException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return null;

    }
    public void deleteArtifact(Artifacts artifact) {
        // TODO Auto-generated method stub
        /*
         * String url = CloudSoftwareRepoConstants.NEXUS_URL +
         * CloudSoftwareRepoConstants.NEXUS_REPOSITORIES + artifact.repositoryId
         * + "/" + CloudSoftwareRepoConstants.NEXUS_CONTENT + artifact.groupId +
         * "/" + artifact.artifactId + "/" + artifact.version;
         */
        String url = NEXUS_URL + "/" + CloudSoftwareRepoConstants.NEXUS
                + CloudSoftwareRepoConstants.NEXUS_REPOSITORIES
                + artifact.repositoryId + "/"
                + CloudSoftwareRepoConstants.NEXUS_CONTENT + artifact.groupId
                + "/" + artifact.artifactId + "/" + artifact.version;
        String deleteReq = cHttp.CloudHttpDeleteRequest(url);

        System.out.println(deleteReq);

    }

    public String pingRepository(String url) throws RepoExceptions {
        // TODO Auto-generated method stub
        String response = null;
        String rUrl = url + "/" + CloudSoftwareRepoConstants.NEXUS
                + CloudSoftwareRepoConstants.NEXUS_STATUS;
        String pingResponse = cHttp.CloudHttpGetRequest(rUrl);
        System.out.println(pingResponse);
        try {
            JAXBContext jc = JAXBContext.newInstance(StatusFactory.class);

            Unmarshaller unmarshaller = jc.createUnmarshaller();

            Status status = (Status) unmarshaller.unmarshal(new InputSource(
                    new StringReader(pingResponse)));
            response = status.getData().getState();
            System.out.println(status.getData().getState());
            return response;
        } catch (JAXBException e) {
            // TODO Auto-generated catch block
            throw new RepoExceptions(
                    "The url you provided does not contain a valid Software Repository. Please Try again.");
        }

    }
```

Υλοποίηση Communication Module

```
public class CloudHttp {

    public HttpClient httpClient;
    public HttpGet httpGet;
    public HttpPost httpPost;
    public HttpEntity httpEntity;
    public HttpResponse httpResponse;
    public HttpDelete httpDelete;

    public MultipartEntityBuilder meBuilder;

    public String response;

    public CloudHttp() {
        super();
        // TODO Auto-generated constructor stub
        httpClient = HttpClientBuilder.create().build();
        meBuilder = MultipartEntityBuilder.create();
    }

   public String CloudHttpGetRequest(String url) throws RepoExceptions {
       httpGet = new HttpGet(url);
       try {
           httpResponse = httpClient.execute(httpGet);
           httpEntity = httpResponse.getEntity();
           response = EntityUtils.toString(httpEntity);

           System.out.println(response);
       } catch (ClientProtocolException e) {
           // TODO Auto-generated catch block
           throw new RepoExceptions(
                   "Malformed repository url. Repository url must have the following format: http(s)://host:port");
       } catch (HttpHostConnectException e) {
           throw new RepoExceptions("Repository unreachable at the moment");
           // System.out.println("Repository unreachable at the moment");

       } catch (IllegalArgumentException e) {
           throw new RepoExceptions(
                   "Malformed repository url. Repository url must have the following format: http(s)://host:port");
       } catch (IOException e) {
           // TODO Auto-generated catch block
           e.printStackTrace();
       }
       // httpEntity = httpResponse.getEntity();
       catch (ParseException e) {
           // TODO Auto-generated catch block
           e.printStackTrace();
       } finally {
           httpGet.releaseConnection();
       }
       return response;
   }
```

```
public String CloudHttpPostRequest(String url) throws RepoExceptions {
    httpPost = new HttpPost(url);
    HttpEntity httpEn = meBuilder.build();
    // System.out.println(meBuilder.)
    httpPost.addHeader("Authorization", "Basic " + authenticate());
    httpPost.setEntity(httpEn);
    try {
        httpResponse = httpClient.execute(httpPost);
    } catch (ClientProtocolException e) {
        // TODO Auto-generated catch block
        throw new RepoExceptions(
                "Malformed repository url. Repository url must have the following format: http(s)://host:port");
    } catch (HttpHostConnectException e) {
        throw new RepoExceptions("Repository unreachable at the moment");
        // System.out.println("Repository unreachable at the moment");

    } catch (FileNotFoundException e) {
        throw new RepoExceptions("No such file");
        // System.out.println("Repository unreachable at the moment");

    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    httpEntity = httpResponse.getEntity();
    try {
        response = EntityUtils.toString(httpEntity);
    } catch (ParseException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } finally {
        httpPost.releaseConnection();
    }
    System.out.println(response);

    return response;
}

 public String CloudHttpDeleteRequest(String url) {

     httpDelete = new HttpDelete(url);
     httpDelete.addHeader("Authorization", "Basic " + authenticate());

     try {
         httpResponse = httpClient.execute(httpDelete);
     } catch (ClientProtocolException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
     } catch (IOException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
     }
     response = httpResponse.getStatusLine().toString();

     System.out.println(response);
     return response;

 }

 private String authenticate() {
     // TODO Auto-generated method stub
     String encoding = new String(Base64.encodeBase64(("admin:admin123")
             .getBytes()));
     return encoding;
 }
```

```
    public void addPart(Map<String, String> parameters)
            throws UnsupportedEncodingException {
        meBuilder.setMode(HttpMultipartMode.BROWSER_COMPATIBLE);
        System.out.println("Parameters: ");
        for (Map.Entry<String, String> entry : parameters.entrySet()) {
            System.out.println("Parameter: " + entry.getKey() + " Value: "
                    + entry.getValue());
            if (entry.getKey().equals("file")) {

                meBuilder.addPart("file",
                        new FileBody(new File(entry.getValue()),
                                ContentType.DEFAULT_BINARY));
            } else {

                meBuilder.addPart(entry.getKey(),
                        new StringBody(entry.getValue(),
                                ContentType.DEFAULT_TEXT));

            }
        }

    }

}
```

Υλοποίηση XML Parser

```
public class CloudXMLParser {

    public ArrayList<Artifacts> xmlParseKeywordSearch(String sres)
            throws ParserConfigurationException, SAXException, IOException {
        try {

            ArrayList<Artifacts> nxArtifactList = new ArrayList<Artifacts>();

            DocumentBuilderFactory factory = DocumentBuilderFactory
                    .newInstance();
            DocumentBuilder builder;
            InputSource is;

            builder = factory.newDocumentBuilder();
            is = new InputSource(new StringReader(sres));
            Document doc = builder.parse(is);
            NodeList list = doc.getChildNodes();
            Element root = (Element) list.item(0);

            // Search for tags named artifact (inside are stored information
            // about
            // the artifacts)
            NodeList artifactsNl = root.getElementsByTagName("artifact");

            // Count how many artifacts we get based on the search term
            int numArtifacts = artifactsNl.getLength();

            // Loop through all the results and create artifacts objects.
            for (int i = 0; i < numArtifacts; i++) {
                Node nodeArtifact = artifactsNl.item(i);

                NodeList nlArtifacts = nodeArtifact.getChildNodes();
                Element elementArtifact = (Element) nlArtifacts;
```

```
        /*************************************************************************
         * Get all the basic data needed to create an artifact object
         ************************************************************************/
        String repositoryId = elementArtifact
                .getElementsByTagName("repositoryId").item(0)
                .getTextContent();
        String artifactId = elementArtifact
                .getElementsByTagName("artifactId").item(0)
                .getTextContent();
        String groupId = elementArtifact
                .getElementsByTagName("groupId").item(0)
                .getTextContent();
        String tkGroupId = splitStringGroup(groupId);
        String version = elementArtifact
                .getElementsByTagName("version").item(0)
                .getTextContent();
        String latestRelease = elementArtifact
                .getElementsByTagName("latestRelease").item(0)
                .getTextContent();
        int extensionCount = elementArtifact.getElementsByTagName(
                "extension").getLength();

        /*************************************************************************************************
         * Get only the actual extension of the artifact not the.pom
         * extension usually NEXUS gives as extension
         ************************************************************************************************/
        String extension = null;
        for (int j = 0; j < extensionCount; j++) {
            if (!elementArtifact.getElementsByTagName("extension")
                    .item(j).getTextContent().equals("pom"))
                extension = elementArtifact
                        .getElementsByTagName("extension").item(j)
                        .getTextContent();

        }

String repositoryURL = CloudSoftwareRepo.getNEXUS_URL()+"/"+CloudSoftwareRepoConstants.NEXUS/*
        + CloudSoftwareRepoConstants.NEXUS_ARTIFACT_CONTENT
        + repositoryId + "/" + tkGroupId + "/" + artifactId
        + "/" + version + "/" + artifactId + "-" + version
        + "." + extension;

String zipURL = CloudSoftwareRepo.getNEXUS_URL()+"/"+CloudSoftwareRepoConstants.NEXUS/*CloudSo
        + CloudSoftwareRepoConstants.NEXUS_ARTIFACT_CONTENT
        + CloudSoftwareRepoConstants.NEXUS_SHADOW_ZIP + "/"
        + tkGroupId + "/" + artifactId + "/" + version + "/"
        + artifactId + "-" + version + "."
        + "zip-unzip/metadata.xml";

CloudHttp cHttp = new CloudHttp();

String retrieveMeta = cHttp.CloudHttpGetRequest(zipURL);

/**
 * DEBUG: View the information that go into the artifact object
 */

System.out.println("Repository id: " + repositoryId);
System.out.println("Artifact id: " + artifactId);
System.out.println("Group id: " + groupId);
System.out.println("Version: " + version);
System.out.println("Latest Release: " + latestRelease);
System.out.println("Extension Count: " + extensionCount);
System.out.println("Extension: " + extension);
System.out.println("Artifcat Download URL: " + repositoryURL);

System.out.println("-----------------------------");

Artifacts nxArtifact = new Artifacts(repositoryId,
        repositoryId, repositoryURL, tkGroupId, artifactId,
        version, latestRelease, extension, "", "", "", "", "");

Artifacts finalArtifact = readInternalXML(nxArtifact,
        retrieveMeta);
```

```
            // //Artifacts nxArtifact = new Artifacts(repositoryId,
            // repositoryId, repositoryURL, tkGroupId, artifactId,
            // version, latestRelease, extension);
            nxArtifactList.add(finalArtifact);

        }

        return nxArtifactList;
    } catch (RepoExceptions e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        return null;
    }

}

public Artifacts readInternalXML(Artifacts artifact, String meta)
        throws ParserConfigurationException, SAXException, IOException {
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder;
    InputSource is;

    builder = factory.newDocumentBuilder();
    is = new InputSource(new StringReader(meta));
    Document doc = builder.parse(is);
    NodeList list = doc.getChildNodes();
    Element root = (Element) list.item(0);

    // Search for tags named artifact (inside are stored information
    // about
    // the artifacts)
    /*
     * NodeList artifactsNl = root.getElementsByTagName("artifact");
     *
     * Node nodeArtifact = artifactsNl.item(0); NodeList nlArtifacts =
     * nodeArtifact.getChildNodes(); Element elementArtifact = (Element)
     * nlArtifacts;
     */

    Element elementArtifact = root;

    String os = elementArtifact.getElementsByTagName("os").item(0)
            .getTextContent();
    String config = elementArtifact.getElementsByTagName("config").item(0)
            .getTextContent();
    String installer = elementArtifact.getElementsByTagName("installer")
            .item(0).getTextContent();
    String hasBin = elementArtifact.getElementsByTagName("binaries")
            .item(0).getTextContent();
    String hasSource = elementArtifact.getElementsByTagName("sources")
            .item(0).getTextContent();

    artifact.setOs(os);
    artifact.setConfig(config);
    artifact.setInstallScript(installer);
    artifact.setHasBin(hasBin);
    artifact.setHasSrc(hasSource);
    return artifact;
}

/**
 * Function that converts the groupId string from a.b into a/b (path)
 *
 * @param groupId
 *            the group id tou be converted
 * @return a string that is in the form of a/b
 */
public String splitStringGroup(String groupId) {

    String[] token = groupId.split("\\.");
    String newString = token[0];
    int tkLength = token.length;
    for (int i = 1; i < tkLength; i++) {
        newString = newString + "/" + token[i];
    }

    return newString;
}
```

```
public ArrayList<String> xmlParseIndex(String sres)
        throws ParserConfigurationException, SAXException, IOException {

    ArrayList<String> index = new ArrayList<String>();
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder;
    InputSource is;

    builder = factory.newDocumentBuilder();
    is = new InputSource(new StringReader(sres));
    Document doc = builder.parse(is);
    NodeList list = doc.getChildNodes();
    Element root = (Element) list.item(0);

    // Search for tags named children
    NodeList childrenNl = root.getElementsByTagName("indexBrowserTreeNode");
    // Count how many children we get from the index
    int numChildren = childrenNl.getLength();
    //System.out.println("Number of Indexed: "+numChildren);
    for (int i = 0; i < numChildren; i++) {
        Node nodeChildren = childrenNl.item(i);
        NodeList nlChildren = nodeChildren.getChildNodes();
        Element elementArtifact = (Element) nlChildren;

        String artName = elementArtifact
                .getElementsByTagName("nodeName").item(0)
                .getTextContent();

        //System.out.println("Node: "+artName);
        index.add(artName);
    }

    return index;

}
```

```java
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "groupId",
    "artifactId",
    "version",
    "uploadDate",
    "os",
    "installer",
    "config",
    "binaries",
    "sources"
})
@XmlRootElement(name = "Artifact")
public class Artifact {

    @XmlElement(required = true)
    protected String groupId;
    @XmlElement(required = true)
    protected String artifactId;
    @XmlElement(required = true)
    protected String version;
    @XmlElement(required = true)
    @XmlSchemaType(name = "date")
    protected XMLGregorianCalendar uploadDate;
    @XmlElement(required = true)
    protected String os;
    @XmlElement(required = true)
    protected String installer;
    @XmlElement(required = true)
    protected String config;
    protected int binaries;
    protected int sources;
```

```
     * Gets the value of the groupId property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getGroupId() {
        return groupId;
    }

    /**
     * Sets the value of the groupId property.
     *
     * @param value
     *     allowed object is
     *     {@link String }
     *
     */
    public void setGroupId(String value) {
        this.groupId = value;
    }

    /**
     * Gets the value of the artifactId property.
     *
     * @return
     *     possible object is
     *     {@link String }
     *
     */
    public String getArtifactId() {
        return artifactId;
    }
```

```
public void setArtifactId(String value) {
    this.artifactId = value;
}

/**
 * Gets the value of the version property.
 *
 * @return
 *     possible object is
 *     {@link String }
 *
 */
public String getVersion() {
    return version;
}

/**
 * Sets the value of the version property.
 *
 * @param value
 *     allowed object is
 *     {@link String }
 *
 */
public void setVersion(String value) {
    this.version = value;
}

/**
 * Gets the value of the uploadDate property.
 *
 * @return
 *     possible object is
 *     {@link XMLGregorianCalendar }
 *
 */
public XMLGregorianCalendar getUploadDate() {
    return uploadDate;
}
```

```
public void setUploadDate(XMLGregorianCalendar value) {
    this.uploadDate = value;
}

/**
 * Gets the value of the os property.
 *
 * @return
 *     possible object is
 *     {@link String }
 *
 */
public String getOs() {
    return os;
}

/**
 * Sets the value of the os property.
 *
 * @param value
 *     allowed object is
 *     {@link String }
 *
 */
public void setOs(String value) {
    this.os = value;
}

/**
 * Gets the value of the installer property.
 *
 * @return
 *     possible object is
 *     {@link String }
 *
 */
public String getInstaller() {
    return installer;
}
```

```
public void setInstaller(String value) {
    this.installer = value;
}

/**
 * Gets the value of the config property.
 *
 * @return
 *     possible object is
 *     {@link String }
 *
 */
public String getConfig() {
    return config;
}

/**
 * Sets the value of the config property.
 *
 * @param value
 *     allowed object is
 *     {@link String }
 *
 */
public void setConfig(String value) {
    this.config = value;
}

/**
 * Gets the value of the binaries property.
 *
 */
public int getBinaries() {
    return binaries;
}

public void setBinaries(int value) {
    this.binaries = value;
}

/**
 * Gets the value of the sources property.
 *
 */
public int getSources() {
    return sources;
}

/**
 * Sets the value of the sources property.
 *
 */
public void setSources(int value) {
    this.sources = value;
}
```

```
    @XmlAccessorType(XmlAccessType.FIELD)
    @XmlType(name = "", propOrder = {
        "name",
        "value"
    })
    public static class Property {

        @XmlElement(required = true)
        protected String name;
        @XmlElement(required = true)
        protected String value;


        public String getName() {
            return name;
        }


        public void setName(String value) {
            this.name = value;
        }


        public String getValue() {
            return value;
        }


        public void setValue(String value) {
            this.value = value;
        }

    }


}
```