Πανεπιστήμιο Πατρών
Πολυτεχνική Σχολή
Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής

Ανάπτυξη Εφαρμογών Καθολικής Μετάφρασης Περιεχομένου στο Περιβάλλον Java Spring

Κοντοτάσιου Ιωάννα
ΑΜ: 3125

Επιβλέπων: Αναπλ. Καθ. κ. Ζαρολιάγκης Χρήστος

Πάτρα, Ιούλιος 2008

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, αναπληρωτή Καθηγητή κ. Ζαρολιάγκη Χρήστο, ο οποίος με τίμησε με τη συνεργασία του και μου έδωσε την ευκαιρία να ασχοληθώ με ένα ενδιαφέρον αντικείμενο. Επίσης, θα ήθελα να ευχαριστήσω το συνεπιβλέποντα Δρ. κ. Χατζηγιαννάκη Ιωάννη για την πολύτιμη βοήθεια και συνδρομή του στην επιτυχή ολοκλήρωση της παρούσας διπλωματικής εργασίας.

Ακόμη θα ήθελα να ευχαριστήσω ιδιαίτερα την οικογένειά μου που με στήριξε και συνεχίζει να με στηρίζει στις επιλογές μου.

Περιεχόμενα

Περίληψη

Στην παρούσα διπλωματική εργασία γίνεται μελέτη και χρήση ενός εναλλακτικού τρόπου υλοποίησης εφαρμογών που χρησιμοποιούν το διαδίκτυο για την επικοινωνία τόσο με τους χρήστες όσο και με άλλα συστήματα. Αρχικά μελετάται το περιβάλλον ανάπτυξης εφαρμογών Spring που βασίζεται στην αρχιτεκτονική Java/JEE και που προσφέρει ιδιαίτερα καινοτόμες διαδικασίες και μεθόδους που επιτρέπουν γρήγορη αναπτυξη εφαρμογών ελαχιστοποιώντας τον παραγόμενο κώδικα. Επίσης μελετάται η βιβλιοθήκη Hibernate της Java που υλοποιεί τη διεπαφή που ορίζεται στα πλαίσια του Java Persistence API.

Στη συνέχεια παρουσιάζεται μία εφαρμογή καθολικής μετάφρασης περιεχομένου, η οποία υλοποιήθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας, και η οποία υλοποιεί γενικευμένες διαδικασίες παραγωγής και διαχείρισης μεταφραστικού υλικού. Κατά τη διαδικασία ανάπτυξης έγινε χρήση των τεχνολογιών που μελετήθηκαν, ενώ παράλληλα έγινε χρήση διαφόρων εργαλείων που βοηθούν στη βελτίωση της ποιότητας του παραγόμενου κώδικα και στην ευκολότερη παραγωγή αυτού.

Κεφάλαιο 1

Εισαγωγή

1.1 Πρόβλημα και Σημασία

Το διαδίκτυο είναι ένας τομέας που γνωρίζει τεράστια άνθιση τα τελευταία χρόνια. Όλο και περισσότερα πράγματα γίνονται πλέον ηλεκτρονικά μέσω του διαδικτύου, από την απλή αναζήτηση πληροφοριών μέχρι την πολύπλοκη πραγματοποίηση διατραπεζικών συναλλαγών. Έτσι, η ανάπτυξη μεγάλων εφαρμογών που χρησιμοποιούν το διαδίκτυο για την επικοινωνία τόσο με τον χρήστη όσο και με άλλα συστήματα είναι ένας συχνός τομέας ενασχόλησης του μηχανικού υπολογιστών.

Για την ανάπτυξη εφαρμογών που κάνουν χρήση του διαδικτύου έχουν αναπτυχθεί πολλές διαφορετικές τεχνικές και τεχνολογίες που κάνουν εφικτή και εύκολη την όλη διαδικασία. Η συνηθέστερη μορφή εφαρμογής είναι οι ιστοσελίδες, οι οποίες, στις περισσότερες των περιπτώσεων, παρουσιάζουν πληροφορίες κάνοντας χρήση δεδομένων που είναι αποθηκευμένα σε μια βάση δεδομένων και χρησιμοποιούν κάποιου είδους γλώσσα προγραμματισμού (π.χ. PHP) για την περάτωση απλών ενεργειών.

Κατά την περίπτωση που οι ενέργειες που πρέπει να εκτελεστούν από τις εφαρμογές είναι πιο πολύπλοκες η προσέγγιση ανάπτυξης αλλάζει. Γίνεται χρήση πλατφορμών εργασίας, οι οποίες διευκολύνουν κατά πολύ την ανάπτυξη εφαρμογών λογισμικού. Οι δύο πιο γνωστές και ευρέως διαδεδομένες τέτοιες πλατφόρμες είναι το *.NET framework* της Microsoft και η *Java Enterprise Edition*

(J2EE) της Sun. Και οι δύο αυτές πλατφόρμες παρέχουν δυνατότητες προγραμματισμού εξυπηρετητών (server side programming) καθιστώντας εύκολη τη χρήση βάσεων δεδομένων, υπηρεσιών διαδικτύου, συναλλαγών, κ.ά.

Ωστόσο, παρά τις πολλές δυνατότητες που παρέχουν οι δύο αυτές πλατφόρμες, έχουν ορισμένα σημαντικά αρνητικά στοιχεία. Είναι πολύ 'βαριές', τόσο από πλευράς όγκου όσο και απόδοσης κατά την εκτέλεση, ενώ ο κώδικας που παράγεται με τη χρήση αυτών είναι 'δεμένος' με την εκάστοτε τεχνολογία. Αυτό έχει αρνητικό αντίκτυπο στη μεταφερσιμότητα των εν λόγω εφαρμογών. Το γεγονός αυτό δημιουργεί την ανάγκη εξεύρεσης εναλλακτικών τρόπων ανάπτυξης, οι οποίες να καθιστούν πιο εύκολη την ανάπτυξη, εξασφαλίζοντας παράλληλα καλύτερα αποτελέσματα.

Ένα άλλο σημαντικό θέμα που ανακύπτει από την τόσο ευρεία χρησιμοποίηση του διαδικτύου είναι η ανάγκη διάθεσης μεγάλων ποσοτήτων πληροφορίας σε πολλές διαφορετικές γλώσσες. Αυτό δημιουργεί την ανάγκη ύπαρξης κάποιου τρόπου οργάνωσης και διαχείρισης των διαφόρων μεταφράσεων των πληροφοριών που παρέχονται σε πολλές γλώσσες. Η λύση δίνεται από τις εφαρμογές *καθολικής μετάφρασης περιεχομένου*. Οι εφαρμογές αυτές έχουν ως χαρακτηριστικό τους την ευκολία ενσωμάτωσης σε γενικής φύσεως εφαρμογές, με μικρό κόστος προσαρμογής τόσο από πλευράς χρόνου όσο και από πλευράς γραμμών κώδικα. Ο σχεδιασμός και υλοποίηση τέτοιων συστημάτων σε κατανεμημένο περιβάλλον εκτέλεσης είναι μια ιδιαίτερα σύνθετη και επίπονη διαδικασία

1.2 Στόχος Διπλωματικής Εργασίας

Στόχο της παρούσας διπλωματικής εργασίας αποτελεί η μελέτη ενός εναλλακτικού τρόπου υλοποίησης εφαρμογών που χρησιμοποιούν το διαδίκτυο για την επικοινωνία τόσο με τους χρήστες όσο και με άλλα συστήματα. Ως βασική γλώσσα προγραμματισμού επιλέχθηκε η *Java*, ενώ ως αντικείμενο μελέτης ορίστηκε το περιβάλλον εργασίας Spring. Επιπλέον, στόχο αποτέλεσε η ανάπτυξη μιας εφαρμογής καθολικής μετάφρασης περιεχομένου η οποία να επιλύει το πρόβλημα της οργάνωσης και διαχείρισης μεταφράσεων.

1.3 Συνεισφορά Διπλωματικής Εργασίας

Στο πλαίσιο της παρούσας διπλωματικής εργασίας έγινε μελέτη μίας εναλλακτικής τεχνολογίας ανάπτυξης μεγάλων Java εφαρμογών, του περιβάλλοντος εργασίας *Spring*. Διερευνήθηκαν οι βασικές αρχές που διέπουν τον τρόπο λειτουργίας του δίνοντας περισσότερη έμφαση στον τρόπο που υλοποιούνται υπηρεσίες που έχουν σχέση με το διαδίκτυο.

Επιπλέον μελετήθηκε η *Hibernate*, μία τεχνολογία μέσω της οποίας διευκολύνονται λειτουργίες που έχουν να κάνουν με αντιστοίχηση Java αντικειμένων σε πίνακες βάσεων δεδομένων.

Επιπρόσθετα αναπτύχθηκε μία εφαρμογή καθολικής μετάφρασης περιεχομένου, η οποία χειρίζεται φράσεις και μεταφράσεις αυτών σε διάφορες γλώσσες. Η εφαρμογή κάνει χρήση της Hibernate για την αποθήκευση των αντικειμένων σε μία βάση δεδομένων και χρησιμοποιεί τη Spring ως επί το πλείστον για την εκτέλεση ενεργειών που έχουν σχέση με το διαδίκτυο. Για την παραγωγή ποιοτικού κώδικα κατά την διαδικασία ανάπτυξης της εφαρμογής χρησιμοποιήθηκε μία σειρά εργαλείων. Τα εργαλεία αυτά διευκολύνουν τη διαδικασία ανάπτυξης ελέγχοντας παράλληλα την ορθότητα του παραγομένου κώδικα.

1.4 Δομή Διπλωματικής Εργασίας

Η δομή της παρούσας διπλωματικής εργασίας χωρίζεται σε δύο ενότητες. Στην πρώτη, που αποτελείται από τα κεφάλαια 2 έως 4, γίνεται περιγραφή και σχολιασμός των τεχνολογιών που μελετήθηκαν, ενώ στη δεύτερη, που αποτελείται από τα κεφάλαια 5 έως 7, γίνεται περιγραφή και ανάλυση της εφαρμογής που αναπτύχθηκε. Το πρώτο κεφάλαιο είναι εισαγωγικό, ενώ στο τελευταίο αποτιμάται η προσπάθεια που έγινε με την παρουσίαση των συμπερασμάτων που προέκυψαν από την παρούσα διπλωματική.

Αρχικά, στο κεφάλαιο 2, περιγράφεται η τεχνολογία *Hibernate*. Γίνεται αναφορά στους λόγους ύπαρξής της, καθώς επίσης και μία γενική ανάλυση των δυνατοτήτων που προσφέρει. Επιπλέον, για να γίνει πιο κατανοητός ο τρόπος λειτουργίας της και τα βασικά της συστατικά, παρουσιάζεται ένα απλό παράδειγμα χρήσης της.

Στη συνέχεια, στο κεφάλαιο 3, γίνεται ανάλυση του περιβάλλοντος εργασίας *Spring*. Αρχικά έγινε γενική περιγραφή των δυνατοτήτων του, καθώς και ανάλυση των διαφόρων τμημάτων που την αποτελούν. Κατόπιν δημιουργήθηκε μία απλή εφαρμογή, αντίστοιχη του γνωστού 'Hello World!', η οποία παρουσιάζει το βασικό τρόπο λειτουργίας της. Έπειτα περιγράφηκαν οι βασικότερες αρχές που διέπουν την φιλοσοφία της Spring και την καθιστούν ξεχωριστή, το *Aspect-Oriented Programming* και το *Dependency Injection*.

Στο κεφάλαιο 4 γίνεται ανάλυση του τι είναι ποιότητα κώδικα και περιγράφονται διάφοροι τρόποι και εργαλεία για την ευκολότερη παραγωγή του. Επιπλέον περιγράφονται τρόποι για την διευκόλυνση του συντονισμού και της παρακολούθησης της διαδικασίας παραγωγής, όταν αυτή γίνεται από μεγάλες ομάδες ατόμων. Η ανάλυση προσανατολίζεται περισσότερο σε εργαλεία που χρησιμοποιήθηκαν κατά τη διαδικασία ανάπτυξης της εφαρμογής που υλοποιήθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας. Τα εργαλεία αυτά είναι το *PMD*, το *JUnit*, το *SVN* και το *IntelliJ IDEA*.

Στο 5ο κεφάλαιο ξεκινάει η δεύτερη ενότητα της διπλωματικής εργασίας. Εκεί γίνεται αναλυτική περιγραφή των προδιαγραφών της εφαρμογής που υλοποιήθηκε. Αρχικά εξηγούνται οι βασικές έννοιες για εξοικείωση του αναγνώστη με την ορολογία που χρησιμοποιείται και στη συνέχεια περιγράφονται οι βασικές λειτουργίες που επιτελεί το σύστημα.

Κατόπιν, στο κεφάλαιο 6, γίνεται περαιτέρω ανάλυση της εφαρμογής με την παρουσίαση του κύκλου ζωής των δεδομένων και της γενικής ροής εκτέλεσης. Επιπρόσθετα παρουσιάζονται οι διεπαφές χρήστη, οι λειτουργίες που αυτές προσφέρουν καθώς και ο τρόπος χρήσης τους.

Στο κεφάλαιο 7 περιγράφεται ο τρόπος με τον οποίο υλοποιήθηκε η εφαρμογή. Αναλύεται η βασική δομή της και περιγράφονται τα σημαντικότερα κομμάτια αυτής. ++++

Τέλος, στο κεφάλαιο 8 γίνεται η αποτίμηση της όλης προσπάθειας ++++

Κεφάλαιο 2

Hibernate

Οι περισσότερες εφαρμογές πρέπει να χρησιμοποιήσουν κάποιου είδους δεδομένα. Παρά το γεγονός ότι το πρόβλημα της διατήρησής τους δεν αποτελεί κάποιο καινούριο ή ασυνήθιστο θέμα για τις εφαρμογές που είναι γραμμένες σε Java ο προγραμματιστής δεν έχει την δυνατότητα απλά να αποφασίσει ποιο δρόμο να ακολουθήσει επιλέγοντας από μια γκάμα παρόμοιων και ευρέως χρησιμοποιούμενων λύσεων.

Για πολλά χρόνια η διατήρηση δεδομένων (persistence) υπήρξε θέμα συζήτησης στην κοινότητα των προγραμματιστών της Java. Πολλοί προγραμματιστές δε συμφωνούν ούτε καν με το θέμα του προβλήματος. Είναι η διατήρηση δεδομένων ένα πρόβλημα που έχει ήδη λυθεί από τη σχεσιακή τεχνολογία (relational technology) και επεκτάσεις (extentions) αυτής, όπως οι αποθηκευμένες διαδικασίες (stored procedures), ή αποτελεί ένα πιο σημαντικό πρόβλημα το οποίο χρειάζεται διαχείριση μέσω ειδικών μοντέλων όπως τα Enterprice Java Beans (EJB); Πρέπει να χρησιμοποιείται SQL κώδικας ακόμη και για τις πιο βασικές ενέργειες όπως create, read, update, delete ή πρέπει αυτές να γίνονται αυτόματα; Πώς επιτυγχάνεται μεταφερσιμότητα, όταν κάθε σύστημα διαχείρισης βάσεων δεδομένων χρησιμοποιεί τη δική του διάλεκτο; Πρέπει η SQL να εγκαταλειφθεί τελείως και να υιοθετηθεί ένα διαφορετικό σύστημα διαχείρισης βάσεων δεδομένων; Η συζήτηση συνεχίζεται. Παρ' όλα αυτά, για την επίλυση του προβλήματος έχει αναπτυχθεί μία προγραμματιστική τεχνική γνωστή ως 'Αντιστοίχηση μεταξύ Δεδομένου - Σχέσης' — object/relational mapping

(ORM) — η οποία έχει αποκτήσει ευρεία αποδοχή.

2.1 Διατήρηση δεδομένων στην Java

Για την διατήρηση των δεδομένων (persistence) στην Java, στα πλαίσια των Enterprice Java Beans 3, έχει οριστεί μία τεχνολογία που ονομάζεται **Java Persistence API**. Η τεχνολογία αυτή ορίζει ένα περιβάλλον εργασίας (framework) για την αντιστοίχηση οντοκεντρικών μοντέλων (object - oriented models) σε σχήματα παραδοσιακών σχεσιακών βάσεων δεδομένων. Κεντρικός στόχος για την ανάπτυξή της υπήρξε ο ορισμός ενός ενιαίου τρόπου για την διατήρηση δεδομένων σε όλες τις Java εφαρμογές.

2.2 Τι είναι η Hibernate

Η **Hibernate** αποτελεί μία βιβλιοθήκη ελεύθερου λογισμικού για τη γλώσσα προγραμματισμού Java, η οποία υλοποιεί την διεπαφή που ορίζεται στα πλαίσια του Java Persistence API. Είναι μία ολοκληρωμένη λύση στο πρόβλημα της διαχείρισης δεδομένων εκτελώντας όλες τις ενέργειες μεταξύ των εφαρμογών και των σχεσιακών βάσεων δεδομένων απαλλάσσοντας τον προγραμματιστή από επιπρόσθετο κόπο και τον κίνδυνο αστοχιών στην αποτύπωση σχέσεων μεταξύ κλάσεων και σχήματος βάσης.

Ο κεντρικός στόχος αυτής είναι η δημιουργία μίας διεπαφής (interface) μεταξύ των διαδεδομένων σχεσιακών βάσεων δεδομένων και του αντικειμενοστραφούς προγραμματισμού. Με άλλα λόγια, επιτρέπει τη χρήση μίας σχεσιακής βάσης δεδομένων ως αντικειμενοστραφή. Για την επίτευξη αυτού, δημιουργούνται αντιστοιχίες μεταξύ των εννοιών του αντικειμενοστραφούς προγραμματισμού, όπως οι συσχετίσεις, η κληρονομικότητα και ο πολυμορφισμός (τα οποία δεν υπάρχουν σε μία σχεσιακή βάση δεδομένων), και των πινάκων και σχέσεων μεταξύ αυτών μίας σχεσιακής βάσης. Με αυτόν τον τρόπο ο προγραμματιστής βλέπει τελικά μία αντικειμενοστραφή βάση δεδομένων, παρ' όλο που στην ουσία χρησιμοποιεί μία σχεσιακή. Έτσι ο προγραμματιστής χρησιμοποιεί τα αντικείμενα της συγκεκριμένης εφαρμογής, τα τροποποιεί σχετικά με τη λογική της

εφαρμογής που αναπτύσσει και τα αποθηκεύει (τροποποιεί, διαγράφει και αναζητά) στη βάση ως αντικείμενα. Σκέπτεται, δηλαδή, με αντικειμενοστραφείς έννοιες και όχι με βάση το σχήμα της σχεσιακής βάσης δεδομένων. Έτσι η Hibernate, γνωρίζοντας την αντιστοιχία μεταξύ βάσης και λογικής της εφαρμογής, αναλαμβάνει να κατασκευάσει τον κατάλληλο κώδικα τον οποίο στέλνει τελικά στη βάση δεδομένων. Έπειτα τα αποτελέσματα που επιστρέφονται από τη βάση στην Hibernate δίνονται στην εφαρμογή ως αντικείμενα. Πρόκειται, δηλαδή, ένα ενδιάμεσο επίπεδο μεταξύ εφαρμογής και βάσης δεδομένων.

2.3 Πλεονεκτήματα χρήσης της Hibernate

Υπάρχουν πάρα πολλοί λόγοι για τους οποίους κάποιος θα επέλεγε να χρησιμοποιήσει τη Hibernate για την ανάπτυξη μίας εφαρμογής που χρειάζεται κάποιου είδους πρόσβαση σε βάση δεδομένων. Οι πιο σημαντικοί από αυτούς συνοψίζονται παρακάτω:

- Η Hibernate ενδείκνυται για ανάπτυξη πολύπλοκων εφαρμογών. Παρέχει μια σειρά από εργαλεία που επιτρέπουν την εύκολη αποτύπωση αντικειμενοστραφών εννοιών σε σχήματα βάσης ασχέτως του είδους της βάσης που χρησιμοποιείται. Επιτρέπει τη χρήση κληρονομικότητας, πολυμορφισμού και σύνθεσης (composition) στις αντιστοιχιζόμενες κλάσεις και παρέχει μια πολύ ισχυρή γλώσσα ερωτημάτων, την HQL, η οποία επιτρέπει τη χρήση όλων των παραπάνω.

- Μπορεί να βοηθήσει να βελτιωθεί η απόδοση της εφαρμογής. Αυτή η βελτίωση πολύ δύσκολα θα επιτυγχανόταν δημιουργώντας απ' ευθείας κώδικα που να εκτελεί τις επιθυμητές λειτουργίες (hand-coding). Η Hibernate δημιουργεί πολύ αποδοτικά ερωτήματα, πράγμα που διασφαλίζει την απόδοση σε πολλές περιπτώσεις. Εκτός αυτού, όμως, υποστηρίζει μία πολύ έξυπνη και αποδοτική πολιτική πρώτου και δεύτερου επιπέδου caching. Με αυτόν τον τρόπο επιτυγχάνεται μεγάλη κλιμακωσιμότητα. Επίσης δίνει τη δυνατότητα στον προγραμματιστή να επιλέξει το επίπεδο caching που επιθυμεί, όντας πολύ έξυπνη στην πολιτική των write-backs.

Επίσης, μπορεί να συνδυαστεί πολύ καλά με κάποια από τα σημαντικότερα λογισμικά caching τόσο ελεύθερου λογισμικού όσο και εμπορικά πακέτα.

- Επιτρέπει σε μεγάλο βαθμό τη μεταφερσιμότητα των εφαρμογών μεταξύ σχεσιακών βάσεων δεδομένων. Η μόνη διαφοροποίηση στις περισσότερες των περιπτώσεων είναι η αλλαγή μίας μόνο παραμέτρου — της διαλέκτου επικοινωνίας με την βάση. Επιπλέον, οι κλάσεις που αντιστοιχίζονται είναι απλές κλάσεις Java (Plain Old Java Objects — POJOs). Έτσι δε χρειάζεται να είναι απόγονοι μιας πολύπλοκης υποχρεωτικής δομής, με αποτέλεσμα να αυξάνεται ακόμη περισσότερο η μεταφερσιμότητα.

- Παρ' όλο που χρειάζεται κάποιο χρόνος για την εκμάθησή της, η Hibernate τελικά αυξάνει σε μεγάλο βαθμό την παραγωγικότητα. Ο προγραμματιστής ενδιαφέρεται μόνο για τα αντικείμενα. Αυτά αποθηκεύονται στη βάση και ανακτώνται από αυτήν με ελάχιστο κόπο.

2.4 Ανάπτυξη εφαρμογών σε Hibernate

Για την ανάπτυξη μίας εφαρμογής σε Hibernate ο προγραμματιστής χρειάζεται να ακολουθήσει τα παρακάτω βήματα:

1. Επιλογή της διαδικασίας ανάπτυξης της εφαρμογής
2. Δημιουργία της 'υποδομής' της εφαρμογής
3. Ανάπτυξη του κώδικα της εφαρμογής και του κώδικα 'αντιστοίχησης' (mapping)
4. Ρύθμιση και εκτέλεση της Hibernate
5. Εκτέλεση της εφαρμογής

2.4.1 Επιλογή της διαδικασίας ανάπτυξης της εφαρμογής

Η ανάπτυξη πολλών εφαρμογών καθοδηγείται κατά κύριο λόγο από την ανάλυση της επιχειρηματικής δομής μιας εταιρίας σε αντικειμενοστραφείς όρους.

Με άλλα λόγια, η ανάπτυξη επηρεάζεται σε ένα μεγάλο βαθμό από ένα υπάρχον σχεσιακό μοντέλο, το οποίο προέρχεται είτε από μια ήδη υπάρχουσα βάση δεδομένων, είτε από ένα καινούριο σχήμα το οποίο έχει σχεδιαστεί από έναν ειδικό σχεδιαστή. Έτσι ο προγραμματιστής πρέπει να αποφασίσει τον τρόπο ανάπτυξης της εφαρμογής. Οι περισσότερες περιπτώσεις ανάπτυξης ανήκουν σε μία από τις τέσσερις παρακάτω περιπτώσεις:

- *Από πάνω προς τα κάτω (Top down)* — Σε αυτόν τον τρόπο ανάπτυξης ο προγραμματιστής ξεκινάει με δεδομένο το μοντέλο της εφαρμογής (δηλαδή γνωρίζει τις διάφορες κλάσεις και τις σχέσεις μεταξύ αυτών), το οποίο είναι ήδη υλοποιημένο και (ιδανικά) έχει πλήρη ελευθερία σε σχέση με τη δημιουργία του σχήματος βάσης. Έτσι αρκεί να δημιουργήσει μεταδεδομένα αντιστοίχησης (metadata) είτε μέσω XML είτε μέσω επιπρόσθετου κώδικα περιγραφής στον ήδη υπάρχοντα πηγαίο κώδικα (annotations) και ύστερα, αν θέλει, να αφήσει τη Hibernate να δημιουργήσει το σχήμα βάσης. Σημειώνεται ότι αυτός ο τρόπος ανάπτυξης είναι ο πιο βολικός για τους περισσότερους προγραμματιστές Java.

- *Από κάτω προς τα πάνω (Bottom up)* — Αντίστροφα, σε αυτή την τεχνική υπάρχει δεδομένο το σχήμα βάσης και έχει αποτυπωθεί σε κάποια βάση δεδομένων. Σε αυτή την περίπτωση ο ευκολότερος τρόπος ανάπτυξης είναι η εξαγωγή μεταδεδομένων από το σχήμα βάσης μέσω εργαλείων αντίστροφης μηχανικής (reverse - engeneering). Με αυτά τα μεταδεδομένα στη συνέχεια, αν χρησιμοποιηθούν κατάλληλα τα εργαλεία της Hibernate, μπορούν να παραχθούν οι αντίστοιχες Java κλάσεις, καθώς και αντικείμενα για το χειρισμό των δεδομένων.

- *Από τη μέση προς τα έξω (Middle out)* — Τα μεταδεδομένα αντιστοίχησης της Hibernate παρέχουν επαρκείς πληροφορίες τόσο για την δημιουργία του σχήματος βάσης όσο και για τη δημιουργία του πηγαίου κώδικα της Java. Έτσι δίνεται η δυνατότητα στους προγραμματιστές να γράψουν μόνο τον XML κώδικα περιγραφής και στη συνέχεια χρησιμοποιώντας τα κατάλληλα εργαλεία της Hibernate να δημιουργήσουν τόσο το σχήμα βάσης όσο και τον αντίστοιχο Java κώδικα.

- *Συνάντηση στη μέση (Meet in the middle)* — Το πιο δύσκολο σενάριο είναι αυτό του συνδυασμού ήδη υπάρχοντος κώδικα Java και σχήματος βάσης. Στην περίπτωση αυτή τα εργαλεία της Hibernate δεν μπορούνε να βοηθήσουν ιδιαίτερα. Ο προγραμματιστής καλείται να γράψει με το χέρι τον XML κώδικα αντίστοιχης και στις περισσότερες περιπτώσεις να αλλάξει ή τον πηγαίο κώδικα ή το σχήμα βάσης ή και τα δύο έτσι, ώστε αυτά να μπορούν να συνδυαστούν.

Για τη συνέχεια θα χρησιμοποιηθεί η διαδικασία ανάπτυξης 'από πάνω προς τα κάτω' και θα περιγραφεί ο τρόπος δημιουργίας μιας εφαρμογής με τη Hibernate.

2.4.2 Ένα παράδειγμα

Αρχικά για τη δημιουργία μίας εφαρμογής είναι απαραίτητη η ύπαρξη μίας έκδοσης της Hibernate. Αυτή είναι διαθέσιμη μέσω του ιστοτόπου http://www.hibernate.org. Επίσης είναι αναγκαίο στο μηχάνημα στο οποίο γίνεται ανάπτυξη να υπάρχει εγκατεστημένο το Apache Ant καθώς και κάποιο σύστημα διαχείρισης βάσεων δεδομένων. Υποθέτουμε ότι αυτά είναι διαθέσιμα καθώς και ότι το μηχάνημα ανάπτυξης έχει εγκατεστημένη κάποια έκδοση MySQL.

2.4.2.1 Δημιουργία του καταλόγου εργασίας

Αρχικά δημιουργούμε έναν κατάλογο στην τοποθεσία που επιθυμούμε και σε αυτόν τοποθετούμε ένα φάκελο *lib*. Σε αυτόν το φάκελο θα τοποθετήσουμε όλα τα jar αρχεία που είναι απαραίτητα για την εκτέλεση της εφαρμογής. Τα περιεχόμενα αυτού του φακέλου θα πρέπει να είναι τα επόμενα:

```
WORKDIR
  +lib
    antlr.jar
    asm.jar
    asm-attrs.jars
    cglib.jar
    commons-collections.jar
```

```
commons-logging.jar
dom4j.jar
hibernate3.jar
jta.jar
log4j.jar
mysql-connector-java-5.1.6-bin.jar
```

Οι παραπάνω βιβλιοθήκες προέρχονται από τη διανομή της Hibernate και οι περισσότερες από αυτές είναι απαραίτητες για τη δημιουργία μιας τυπικής εφαρμογής. Το *mysql-connector-java-5.1.6-bin.jar* αποτελεί το λογισμικό διασύνδεσης μεταξύ της εφαρμογής και του συστήματος διαχείρισης βάσεων δεδομένων. Σε περίπτωση που γίνεται χρήση άλλου συστήματος διαχείρισης θα πρέπει το αρχείο αυτό να αντικατασταθεί από τον αντίστοιχο οδηγό (driver).

2.4.2.2 Δημιουργία 'αποθηκευόμενης' κλάσης

Οι εφαρμογές της Hibernate ορίζουν αποθηκευόμενες κλάσεις (persistent classes) οι οποίες αντιστοιχίζονται με πίνακες σε κάποια σχεσιακή βάση δεδομένων. Παρακάτω φαίνεται μια απλή κλάση την οποία θα αντιστοιχίσουμε σε έναν πίνακα στη βάση δεδομένων.

Listing 2.1: Language.java

```
1  package language;
2
3  import java.util.Date;
4
5  public class Language {
6      private String name;
7      private Date date;
8
9      public Language() {}
10
11     public String getName() {
12         return name;
13     }
```

```
14      public void setName(final String name) {
15          this.name = name;
16      }
17
18      public Date getDate() {
19          return date;
20      }
21      public void setDate(Date date) {
22          this.date = date;
23      }
24  }
```

Η κλάση *Language* έχει δύο μεταβλητές μέλη: το *name* — το όνομα της γλώσσας, το οποίο θα αποτελέσει το πρωτεύον κλειδί, το *date* — την ημερομηνία καταχώρησης της γλώσσας. Το μέλος name θα επιτρέπει στην εφαρμογή να ξεχωρίζει τις διάφορες εγγραφές. Δηλαδή, αν δύο στιγμιότυπα τύπου *Language* έχουν το ίδιο *name*, θα αντιστοιχούν στην ίδια εγγραφή στη βάση δεδομένων.

Ένα σημαντικό σημείο είναι ότι όλες οι μεταβλητές - μέλη της κλάσης που θα αντιστοιχηθούν σε κάποια στήλη του πίνακα της βάσης πρέπει να έχουν μεθόδους τύπου JavaBeans μέσω των οποίων δίνεται η πρόσβαση σε αυτές από άλλες κλάσεις. Ακόμη πρέπει να υπάρχει και ένας contructor ο οποίος να μην παίρνει κανένα όρισμα.

2.4.2.3 Αντιστοίχηση κλάσης σε σχήμα βάσης

Για να επιτευχθεί η αντιστοίχηση μεταξύ του Java κώδικα και των πινάκων της βάσης δεδομένων, η Hibernate χρειάζεται κάποια επιπλέον πληροφορία. Πρέπει να ξέρει, δηλαδή, πώς να αποθηκεύει και να φορτώνει τα διάφορα στιγμιότυπα της κλάσης αυτής. Αυτού του είδους τα μεταδεδομένα μπορούν να είναι γραμμένα σε ένα XML έγγραφο αντιστοίχησης, το οποίο περιγράφει, εκτός των άλλων, πώς οι μεταβλητές - μέλη της κλάσης *Language* θα αντιστοιχιστούν με τις στήλες του πίνακα *LANGUAGES*. Το έγγραφο αυτό πρέπει να έχει το όνομα της κλάσης με κατάληξη *hbm.xml*, δηλαδή, στην περίπτωση που περιγράφουμε, *Language.hbm.xml*.

Listing 2.2: Language.hbm.xml

```
1  <?xml version="1.0"?>
2  <!DOCTYPE hibernate-mapping PUBLIC
3          "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4          "http://hibernate.sourceforge.net/hibernate-
              mapping-3.0.dtd">
5  <hibernate-mapping>
6      <class name="language.Language" table="LANGUAGES">
7          <id name="name" length="50"/>
8          <property name="date"/>
9      </class>
10 </hibernate-mapping>
```

Ο παραπάνω κώδικας λέει στη Hibernate πώς να αντιστοιχίσει την κλάση *Language* με τον πίνακα *LANGUAGES*. Το tag id ορίζει ποιο θα είναι το πρωτεύον κλειδί, ενώ το property χρησιμοποιείται για τις υπόλοιπες στήλες του πίνακα. Θα μπορούσαμε να ορίσουμε ρητά τι είδους τύπο επιθυμούμε να έχει κάθε στήλη (αυτός ο τύπος θα πρέπει να είναι συμβατός με τον τύπο δεδομένων της κλάσης *Language*) καθώς και το όνομα κάθε στήλης. Παρόλα αυτά, στις περισσότερες των περιπτώσεων η Hibernate μπορεί να επιλέξει αυτόματα τον κατάλληλο τύπο. Εντούτοις, πολλές φορές ο προγραμματιστής θα χρειαστεί να επέμβει, για να ορίσει ρητά αυτό ακριβώς που χρειάζεται.

Μετά τη δημιουργία του αρχείου Java και του XML αρχείου αντιστοίχησης ο φάκελος δημιουργίας πρέπει να μοιάζει ως εξής:

```
WORKDIR
  +lib
    <Hibernate and third-party libraries>
  +src
    +language
      Language.java
      Language.hbm.xml
```

Η Hibernate δίνει τη δυνατότητα ορισμού πολύπλοκων τύπων στηλών, όπως άλλες κλάσεις (είτε της Java είτε δημιουργημένες από τον προγραμματιστή),

καθώς και σύνθετων σχέσεων μεταξύ κλάσεων (σχέση ένα προς ένα, ένα προς πολλά, πολλά προς ένα, κτλ). Η περαιτέρω ανάλυση των δυνατοτήτων της ξεφεύγει από τα πλαίσια του σκοπού αυτής της εργασίας. Περισσότερες και πιο αναλυτικές οδηγίες υπάρχουν στην ιστοσελίδα http://www.hibernate.org/hib_docs/v3/reference/en/html.

2.4.2.4 Αποθήκευση και ανάκτηση αντικειμένων

Στη συνέχεια θα δείξουμε πώς μπορούμε να αποθηκεύσουμε ένα αντικείμενο στη βάση και έπειτα να το ανακτήσουμε.

Listing 2.3: LanguagesManager.java

```
1  package language;
2
3  import org.hibernate.*;
4  import java.util.*;
5
6
7  public class LanguagesManager {
8
9      public static void main(String[] args) {
10         Calendar cal = new GregorianCalendar(Locale.
               getDefault());
11         Language language = new Language();
12         language.setName("english");
13         language.setDate(cal.getTime());
14
15         // Insert the language into the database
16         Session session = HibernateUtil.getSessionFactory
               ().getCurrentSession();
17         session.beginTransaction();
18         session.save(language);
19         session.getTransaction().commit();
20         HibernateUtil.getSessionFactory().close();
21
```

```
22          // Retrieve languages from database
23          Session newsession = HibernateUtil.
               getSessionFactory().getCurrentSession();
24          newsession.beginTransaction();
25          Query query = newsession.createQuery("from
               Language");
26          Collection languages =  query.list();
27          newsession.getTransaction().commit();
28          HibernateUtil.getSessionFactory().close();
29
30      }
31  }
```

Ο παραπάνω κώδικας πρέπει να τοποθετηθεί σε ένα αρχείο με το όνομα *LanguagesManager.java*. Είναι μία απλή κλάση Java που περιέχει μία *main* μέθοδο, η οποία δημιουργεί ένα αντικείμενο τύπου *Language*, το αποθηκεύει στη βάση και ανακτά όλες τις εγγραφές αυτού του τύπου από τη βάση.

Οι διεπαφές (interfaces) της Hibernate που χρησιμοποιούνται για την πρόσβαση στη βάση είναι:

- Session — Το Session της Hibernate είναι πολλά πράγματα σε ένα. Είναι ένα μονονηματικό, μη διαμοιραζόμενο αντικείμενο, το οποίο αντιπροσωπεύει ένα κομμάτι εργασίας με την βάση. Αποτελεί το API διαχείρισης της διασύνδεσης μεταξύ βάσης και Java κώδικα. Είναι αυτό που χρησιμοποιείται για την ανάκτηση και την αποθήκευση αντικειμένων.

- Transaction — Αυτό το API της Hibernate μπορεί να χρησιμοποιηθεί για να θέσει προγραμματιστικά περιορισμούς συνδιαλλαγής (transaction), αλλά αυτό είναι προαιρετικό (οι περιορισμοί συνδιαλλαγής δεν είναι προαιρετικοί). Με άλλα λόγια, δίνεται η δυνατότητα προσθήκης επιπλέον κώδικα στα πλαίσια της πραγματοποιούμενης συνδιαλλαγής.

- Query — Το ερώτημα που θα τεθεί στη βάση για την ανάκτηση των επιθυμητών αντικειμένων. Αυτό το ερώτημα μπορεί να είναι γραμμένο είτε σε απλή SQL γλώσσα είτε στην αντικειμενοστραφή γλώσσα ερωτημάτων που παρέχει η Hibernate.

Τα σχετικά με την κλάση *HibernateUtil* θα εξηγηθούν αργότερα.

2.4.2.5 Ρύθμιση και χρήση της Hibernate

Η αρχικοποίηση της Hibernate γίνεται με τη δημιουργία ενός αντικειμένου τύπου *SessionFactory* μέσω ενός αντικειμένου *Configuration*. Το *SessionFactory* χρειάζεται για τη δημιουργία νέων Sessions, ενώ το αντικείμενο τύπου *Configuration* αποτελεί ένα είδος αναπαράστασης ενός αρχείου ρυθμίσεων (configuration file) για τη Hibernate. Όλα αυτά μπορούν να συνδυαστούν σε μία κλάση, την *HibernateUtil*.

Listing 2.4: HibernateUtil.java

```
1  package util;
2
3  import org.hibernate.*;
4  import org.hibernate.cfg.*;
5
6  public class HibernateUtil {
7
8      private static final SessionFactory sessionFactory;
9
10     static {
11         try {
12             // Create the SessionFactory from hibernate.
                   cfg.xml
13             sessionFactory = new Configuration().
                   configure().buildSessionFactory();
14         } catch (Throwable ex) {
15             // Make sure you log the exception, as it
                   might be swallowed
16             System.err.println("Initial SessionFactory
                   creation failed." + ex);
17             throw new ExceptionInInitializerError(ex);
18         }
19     }
```

```
20
21      public static SessionFactory getSessionFactory() {
22          return sessionFactory;
23      }
24  }
```

Με την παραπάνω κλάση γίνεται η χρήση του *SessionFactory* βολική. Ουσιαστικά δημιουργείται ένας τρόπος χειρισμού ενός μοναδικού *SessionFactory* έτσι, ώστε όλα τα *Sessions* της συγκεκριμένης εφαρμογής να χρησιμοποιούν αυτό. Για την αρχικοποίηση του *sessionFactory* η *Hibernate* θα ψάξει να βρει default αρχείο ρυθμίσεων στον root κατάλογο. Το αρχείο αυτό θα πρέπει να έχει όνομα *hibernate.cfg.xml*. Εντούτοις, θα μπορούσε να έχει και οποιοδήποτε άλλο όνομα, αρκεί να οριζόταν αυτό ρητά γράφοντας:

```
sessionFactory = new Configuration()
                 .configure("/foo/myconfigurationfile.cfg.xml")
                 .buildSessionFactory();
```

Το αρχείο ρυθμίσεων της Hibernate είναι αυτό που ορίζει τον τρόπο σύνδεσης της εφαρμογής με το σύστημα διαχείρισης των βάσεων δεδομένων. Στην περίπτωση που περιγράφουμε το αρχείο αυτό θα μοιάζει κάπως έτσι:

[caption=hibernate.cfg.xml]

```
1  <?xml version='1.0' encoding='utf-8'?>
2  <!DOCTYPE hibernate-configuration PUBLIC
3    "-//Hibernate/Hibernate Configuration DTD//EN"
4    "http://hibernate.sourceforge.net/hibernate-
       configuration-3.0.dtd">
5  <hibernate-configuration>
6      <session-factory>
7          <property name="connection.driver_class">com.
             mysql.jdbc.Driver</property>
8          <property name="connection.url">jdbc:mysql://
             localhost/tmt?characterEncoding=UTF-8</
             property>
9          <property name="connection.username">user</
             property>
```

```
10          <property name="connection.password">userpassword
               </property>
11          <property name="dialect">org.hibernate.dialect.
               MySQL5InnoDBDialect</property>
12
13          <mapping resource="language/Language.hbm.xml"/>
14      </session-factory>
15  </hibernate-configuration>
```

Στο παραπάνω αρχείο ορίζουμε τις διάφορες βάσεις δεδομένων στις οποίες επιθυμούμε η εφαρμογή να έχει πρόσβαση καθώς και τη συμπεριφορά της Hibernate σε κάθε περίπτωση. Για κάθε μία διαφορετική βάση που θέλουμε η Hibernate να επικοινωνίσει, αρκεί να ορίσουμε ένα διαφορετικό *<session-factory>*. Τα διάφορα *<session-factory>* μπορούν να τοποθετηθούν όλα στο ίδιο αρχείο, αλλά θεωρείται ευκολότερο προγραμματιστικά αυτά να είναι διαχωρισμένα σε διαφορετικά αρχεία.

Για κάθε διαφορετική σύνδεση με τη βάση πρέπει να ορίσουμε συγκεκριμένες παραμέτρους έτσι, ώστε να είναι δυνατή η αντιστοίχηση. Τέτοιες παράμετροι είναι το είδος της βάσης δεδομένων (πχ MySQL, Oracle κτλ), ο οδηγός (driver) που θα αναλάβει τις ενδιάμεσες ενέργειες, το όνομα και ο κωδικός του χρήστη που έχει πρόσβαση στη βάση, η διάλεκτος που χρησιμοποιεί η βάση, η τοποθεσία των αρχείων ρυθμίσεων για κάθε μία αντιστοιχισμένη κλάση. Βέβαια, εκτός αυτών των παραμέτρων που είναι απαραίτητες για τη λειτουργία της εφαρμογής υπάρχουν και άλλες προαιρετικές οι οποίες ορίζουν τον τρόπο συμπεριφοράς της Hibernate. Παραδείγματα τέτοιων παραμέτρων είναι το αν επιθυμούμε να χρησιμοποιείται cache, ο μέγιστος αριθμός των αντικειμένων που θέλουμε να ανακτώνται κάθε φορά και αν θέλουμε κάθε φορά να δημιουργείται η βάση από την αρχή. Αναλυτική περιγραφή όλων αυτών υπάρχει στην τεκμηρίωση της Hibernate.

Με αυτόν τον τρόπο, έχοντας κάνει και τις ρυθμίσεις της Hibernate, έχουμε ολοκληρώσει όλες τις απαραίτητες ενέργειες για τη δημιουργία μιας απλής εφαρμογής.

Κεφάλαιο 3

Spring

Τα τελευταία χρόνια ο κόσμος της Java υπήρξε μάρτυρας μίας δραματικής αλλαγής στον τρόπο ανάπτυξης μεγάλων Java εφαρμογών. Η χρήση των λεγόμενων 'βαρέων' αρχιτεκτονικών (Heavyweight Architectures), όπως είναι τα Enterprise Java Beans (EJB), έχει περιοριστεί και τη θέση τους έχουν πάρει πιο 'ελαφριά' περιβάλλοντα εργασία (frameworks). Πολύπλοκες και εξαρτώμενες από την κάθε τεχνολογία υπηρεσίες, όπως το object/relational mapping (ORM) και τα συστήματα διαχείρισης συναλλαγών (transaction management systems), έχουν αντικατασταθεί από πιο απλές εναλλακτικές. Ένα τέτοιο, πολλά υποσχόμενο και γρήγορα διαδιδόμενο περιβάλλον εργασίας είναι η *Spring*. Η Spring αποτελεί ένα υψηλής ποιότητας και ευχρηστίας project το οποίο ικανοποιεί τις ανάγκες και τις απαιτήσεις των πραγματικών Java εφαρμογών.

3.1 Τι είναι η Spring

Η *Spring* είναι ένα ελεύθερο (open source) περιβάλλον εργασίας για εφαρμογές Java που δημιουργήθηκε από τον Rob Johnson και περιγράφηκε στο βιβλίο του "*Expert One-on-One: J2EE Design and Development*". Αναπτύχθηκε κατά κύριο λόγο, για να αντιμετωπίσει την πολυπλοκότητα ανάπτυξης enterprise εφαρμογών. Η Spring κάνει εφικτή την χρήση απλών JavaBeans για την επίτευξη πραγμάτων που προηγουμένως μπορούσαν να γίνουν μόνο μέσω EJBs. Παρ' όλα αυτά, η Spring δεν είναι μόνο χρήσιμη για την ανάπτυξη server-side

εφαρμογών. Η χρήση της μπορεί να βοηθήσει στην απλοποίηση του κώδικα, τον ευκολότερο και πιο αποτελεσματικό έλεγχο και τη *χαλαρή διασύνδεση* (*loose coupling*) κάθε Java εφαρμογής.

Η *Spring* κάνει πολλά πράγματα, αλλά, αν την αναλύσει κάποιος στα βασικά της κομμάτια, θα δει ότι είναι ένας ελαφρύς (lightweight) aspect-oriented container ο οποίος εφαρμόζει dependency injection, ενώ παράλληλα αποτελεί και περιβάλλον εργασίας (framework). Για να γίνουν τα πράγματα πιο κατανοητά θα γίνεται ανάλυση της περιγραφής αυτής:

- *Lightweight* (Ελαφρύς) — Με τον όρο αυτό αναφερόμαστε τόσο στο μέγεθος όσο και στον επιπλέον φόρτο (overhead). Η διανομή του Spring Framework μπορεί να συμπεριληφθεί σε ένα απλό jar αρχείο μεγέθους 2,5 MB, ενώ ο επιπρόσθετος φόρτος εργασίας που απαιτείται είναι αμελητέος. Ο προγραμματιστής χρειάζεται να κάνει ελάχιστες, αν όχι καθόλου, αλλαγές στον κώδικα της εφαρμογής που αναπτύσσει έτσι, ώστε να επωφεληθεί από τον πυρήνα της, ενώ μπορεί οποιαδήποτε στιγμή να σταματήσει να τη χρησιμοποιεί. Βέβαια αυτού του είδους η ευχέρεια παρέχεται μόνο στον πυρήνα της Spring. Πολλά επιπλέον μέρη της, όπως είναι η πρόσβαση δεδομένων, απαιτούν πολύ στενότερη 'διασύνδεση' (coupling) από ότι το Spring framework. Παρ' όλα αυτά, το κέρδος στις περισσότερες από αυτές τις περιπτώσεις είναι πολύ σημαντικό.

- *Dependency Injection* — Η Spring προάγει τη χαλαρή διασύνδεση χρησιμοποιώντας μία τεχνική που είναι γνωστή ως dependency injection (DI). Όταν εφαρμόζεται η DI, τα αντικείμενα λαμβάνουν παθητικά τις εξαρτήσεις τους (dependencies) αντί να τις δημιουργούν ή να τις αναζητούν μόνα τους. Είναι κάτι σαν ένα αντίστροφο Java Naming and Directory Interface (JNDI) — αντί ένα αντικείμενο να ψάχνει μόνο του για τις εξαρτήσεις του σε έναν container, ο container δίνει τις εξαρτήσεις στο αντικείμενο χωρίς να περιμένει πρώτα να ερωτηθεί.

- Aspect - Oriented — Η Spring παρέχει πλούσια υποστήριξη σε Aspect - Oriented Programming (AOP). Αυτό έχει σαν αποτέλεσμα τη δημιουργία εφαρμογών με περισσότερη συνοχή ακόμη και αν είναι απαραίτητη η συνύπαρξη διαφορετικών λογικών λειτουργίας. Έτσι, κάθε αντικείμενο της

εφαρμογής κάνει μόνο ό,τι το αφορά και δεν είναι υπεύθυνο (ενδεχομένως δε γνωρίζει καν) για πράγματα που έχουν να κάνουν με άλλα συστήματα. Τέτοιου είδους πράγματα μπορεί να είναι η υποστήριξη συνδιαλλαγών ή η καταγραφή των διαφόρων κινήσεων (logging).

- *Container* — Η Spring αποτελεί έναν container με την έννοια ότι κρατάει και διαχειρίζεται τον κύκλο ζωής και τη διαμόρφωση των αντικειμένων της εφαρμογής. Ο προγραμματιστής μπορεί να ορίσει πώς θέλει να διαμορφώνονται και να δημιουργούνται τα αντικείμενα, καθώς και ποιες θέλει να είναι οι σχέσεις μεταξύ τους.

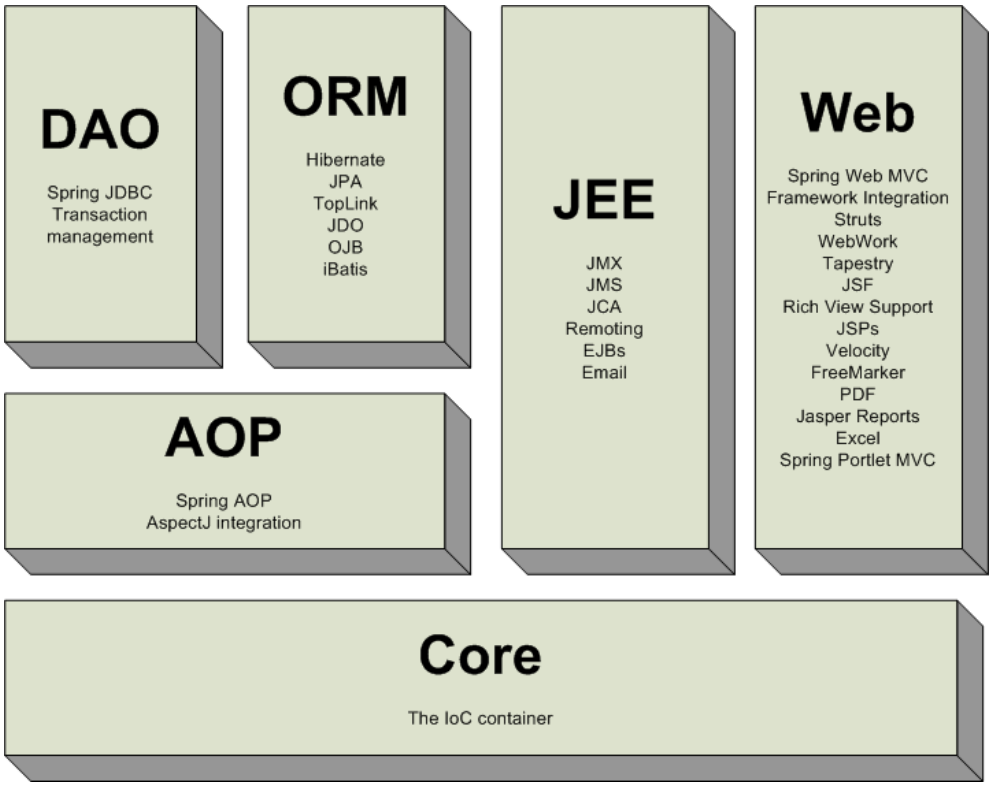
- *Framework* (Περιβάλλον Εργασίας) — Η Spring δίνει τη δυνατότητα να δημιουργηθούν πολύπλοκες εφαρμογές με το συνδυασμό άλλων, λιγότερο πολύπλοκων, κομματιών. Στη Spring τα αντικείμενα δημιουργούνται μέσω δηλώσεων σε απλά XML αρχεία. Επίσης παρέχει πολλές δομικές λειτουργίες, όπως η διαχείριση συναλλαγών κ.ά., επιτρέποντας στον προγραμματιστή να ασχοληθεί περισσότερο με τη λογική της εφαρμογής του.

Έτσι, μπορούμε να πούμε ότι η Spring είναι ένα περιβάλλον εργασίας το οποίο βοηθάει τον προγραμματιστή να δημιουργήσει εφαρμογές με χαλαρά διασυνδεδεμένο κώδικα. Ακόμη και αν αυτό ήταν το μόνο που έκανε, τα οφέλη θα ήταν πάρα πολλά από άποψη συντηρησιμότητας και ελεγξιμότητας και θα άξιζε τον κόπο να γίνει χρήση της. Όμως η Spring είναι πολλά περισσότερα. Περιέχει κομμάτια που χρησιμοποιούν DI και AOP και έτσι συνθέτουν μία αρκετά αξιόλογη πλατφόρμα στην οποία μπορούν να αναπτυχθούν εφαρμογές.

3.2 Τα modules της Spring

Το περιβάλλον εργασίας της Spring αποτελείται από αρκετά, καλά ορισμένα, κομμάτια (modules). Τα κομμάτια αυτά σαν σύνολο δίνουν στον προγραμματιστή ό,τι χρειάζεται, για να αναπτύξει enterprise εφαρμογές. Δεν απαιτείται από τον προγραμματιστή να βασίσει όλη την εφαρμογή του πάνω στη Spring. Αντίθετα, του παρέχεται η δυνατότητα να επιλέξει όποια από τα modules πιστεύει ότι καλύπτουν τις ανάγκες του. Επιπλέον παρέχονται τρόποι σύνδεσης

με άλλα περιβάλλοντα εργασίας και βιβλιοθήκες έτσι, ώστε ο προγραμματιστής να μη χρειαστεί να τα αναπτύξει από την αρχή.



Σχήμα 3.1: Η αρχιτεκτονική του περιβάλλοντος εργασίας Spring.

Όπως φαίνεται στο σχήμα 3.1, όλα τα modules της Spring είναι χτισμένα πάνω από το βασικό της container — πυρήνα. Ο πυρήνας καθορίζει πώς δημιουργούνται τα beans, πώς αρχικοποιούνται και πώς γίνεται ο χειρισμός τους — κάτι που αποτελεί τις κυριότερες λειτουργίες της Spring. Όμως ο προγραμματιστής ενδιαφέρεται περισσότερο για άλλα modules, αυτά που χρησιμοποιούν τον πυρήνα και τις υπηρεσίες που αυτός παρέχει.

Στη συνέχεια θα περιγράψουμε κάθε ένα από τα διάφορα modules της Spring.

Ο πυρήνας — core container

Ο πυρήνας της Spring παρέχει όλη τη βασική λειτουργικότητα του Spring Framework. Το module αυτό περιέχει το BeanFactory, το οποίο είναι ο θεμε-

λιώδης container και η βάση με την οποία η Spring υλοποιεί το DI.

Το DAO module

Η δουλεία με JDBC συχνά περιλαμβάνει τη δημιουργία αρκετού κώδικα, ο οποίος κάνει βασικές και τετριμμένες ενέργειες όπως σύνδεση με τη βάση, δημιουργία κάποιας εντολής (statement), επεξεργασία του αποτελέσματος και κλείσιμο της σύνδεσης. Μέσω του Data Access Object (DAO) module η Spring δίνει τη δυνατότητα απλοποίησης όλης αυτή της διαδικασίας περιορίζοντας τα προβλήματα που προκύπτουν. Επιπλέον, χτίζει ένα layer από exceptions πάνω από τα μηνύματα σφάλματος που παράγουν αρκετοί εξυπηρετητές βάσεων δεδομένων.

Επιπρόσθετα, αυτό το module (χρησιμοποιώντας το AOP module) παρέχει υπηρεσίες διαχείρισης συναλλαγών για τα αντικείμενα των εφαρμογών που χρησιμοποιούν τη Spring.

Το ORM module

Για αυτούς που προτιμούν να χρησιμοποιούν το εργαλεία για object - relational mapping (ORM) (αντιστοίχηση σχέσης - αντικειμένου) αντί για την απευθείας χρήση του JDBC η Spring παρέχει το ORM module. Αυτό χτίζει πάνω στο DAO παρέχοντας έναν αρκετά βολικό τρόπο για ανάπτυξη DAO πάνω από αρκετές λύσεις που προσφέρουν object - relational mapping. Παρ΄ όλο που η Spring δεν παρέχει το δικό της ORM, υποστηρίζει αρκετά δημοφιλή περιβάλλοντα εργασίας, συμπεριλαμβανομένων των Hibernate, Java Persistence API, Java Data Objects και iBatis SQL Maps. Το σύστημα διαχείρισης συναλλαγών της Spring υποστηρίζει εκτός από αυτά τα περιβάλλοντα εργασίας και το JDBC.

Το AOP module

Η Spring προσφέρει υποστήριξη για aspect - oriented programming στο AOP module της. Όπως και το DI, το AOP υποστηρίζει τη χαλαρή διασύνδεση μεταξύ των αντικειμένων κάθε εφαρμογής, όμως με το AOP σημαντικά θέματα

που αφορούν το σύνολο των εφαρμογών (όπως οι συναλλαγές και η ασφάλεια) χωρίζονται από τα αντικείμενα στα οποία αυτά εφαρμόζονται.

Το Web module

Η Spring παρέχει μία πλούσια συλλογή κλάσεων για τη δημιουργία εφαρμογών που στηρίζονται στο διαδίκτυο (web - based applications) μέσω του Web module της. Υποστηρίζεται η χρήση του προτύπου σχεδίασης εφαρμογών Model/View/Controller (MVC), ο χειρισμός ηλεκτρονικής αλληλογραφίας (mail support) καθώς και η χρήση απομακρυσμένων υπηρεσιών (remoting support).

Όσον αφορά τη χρήση του προτύπου σχεδίασης εφαρμογών Model/View-/Controller (MVC), η Spring υποστηρίζει πλήρως το πιο γνωστό MVC περιβάλλον εργασίας, το Apache Struts. Με αυτόν τον τρόπο δίνεται η δυνατότητα στον προγραμματιστή να χρησιμοποιήσει στις ήδη σχεδιασμένες κλάσεις του τις αρχές του dependency injection που προσφέρει η Spring. Βέβαια, πέραν της υποστήριξης του Apache Struts, η Spring προσφέρει και τη δική της υλοποίηση του MVC. Μέσω αυτού μπορεί να γίνει χρήση μίας ευρείας γκάμας τεχνολογιών παρουσίασης, από σελίδες JSP και Apache Jakarta Velocity μέχρι Microsoft Excel και Adobe PDF.

Σε σχέση με την αποστολή μηνυμάτων ηλεκτρονικής αλληλογραφίας, η Spring παρέχει ένα αρκετά απλοποιημένο API, το οποίο ταιριάζει με την όλη φιλοσοφία του DI που αυτή προσφέρει. Για το σκοπό αυτό παρέχονται δύο υλοποιήσεις, μία του JavaMail και μία της MailMessage κλάσης από το πακέτο *com.oreilly.servlet* του Jason Hunter. Μέσω αυτών δίνεται η δυνατότητα δημιουργίας ενός πρότυπου μηνύματος το οποίο στη συνέχεια μπορεί να χρησιμοποιηθεί σαν βάση για την αποστολή αλληλογραφίας μέσω της εφαρμογής.

Τέλος, σε σχέση με χρήση απομακρυσμένων υπηρεσιών, παρέχεται εκτεταμένη υποστήριξη μίας μεγάλης συλλογής τεχνικών απομακρυσμένης πρόσβασης για τη γρήγορη δημιουργία και πρόσβαση σε απομακρυσμένες υπηρεσίες. Τέτοιες τεχνικές είναι το Java RMI, το JAXRPC, το Caucho Hessian και το Caucho Burlap. Εκτός αυτών, η Spring παρέχει και το δικό της πρωτόκολλο, που χρησιμοποιεί το HTTP πρωτόκολλο επικοινωνίας και το οποίο βασίζεται στο απλό Java serialization.

Java EE Connector API (JCA)

Αν και τα τελευταία χρόνια όλο και περισσότεροι προγραμματιστές τείνουν να χρησιμοποιούν τις λεγόμενες ελαφριές πλατφόρμες με τεχνολογίες όπως η Spring και ο Tomcat, υπάρχουν πολλές περιπτώσεις όπου η χρήση αρκετών παραδοσιακών J2EE APIs κρίνεται απαραίτητη. Η σύνδεση και επικοινωνία μεταξύ τέτοιου είδους εφαρμογών και εφαρμογών που χρησιμοποιούν άλλες τεχνολογίες μπορεί να είναι δύσκολη. Το JCA της Spring παρέχει ένα σταθερό (stable) και αξιόπιστο τρόπο για την επικοινωνία με μία σειρά από τέτοιου είδους enterprise συστήματα. Τα κυριότερα APIs που υποστηρίζονται είναι το Java Naming and Directory Interface (JNDI), τα EJB καθώς και η Java Message Service (JMS).

3.3 Ένα παράδειγμα

Το dependency injection είναι το πιο βασικό πράγμα που κάνει η Spring. Στη συνέχεια θα παρουσιαστεί μία απλή εφαρμογή, μία διαφοροποίηση του συνηθισμένου "Hello World", η οποία θα παρουσιάσει τα βασικά σημεία της Spring.

Η πρώτη κλάση που χρειάζεται η Hello World εφαρμογή είναι μία κλάση εξυπηρετητής (service class) ο σκοπός της οποίας είναι να τυπώσει το γνωστό χαιρετισμό. Παρακάτω φαίνεται το interface *GreetingService* το οποίο ορίζει τις συναρτήσεις της κλάσης εξυπηρετητή.

Listing 3.1: GreetingService.java

```
1 public interface GreetingService {
2     void sayGreeting();
3 }
```

Ακολούθως παρουσιάζεται η συνάρτηση *GreetingServiceImpl* η οποία υλοποιεί (implements) το παραπάνω interface. Η χρήση interface για τον ορισμό των ενεργειών που μπορούν να γίνουν με τα διάφορα αντικείμενα δεν είναι απαραίτητη, εντούτοις η πολιτική αυτή συνιστάται.

Listing 3.2: GreetingServiceImpl.java

```
1 public class GreetingServiceImpl implements
```

```
    GreetingService {
2     private String greeting;
3
4     public GreetingServiceImpl () {}
5     public GreetingServiceImpl (String greeting) {
6         this.greeting = greeting;
7     }
8
9     void sayGreeting() {
10        System.out.println(greeting);
11    }
12    public void setGreeting(String greeting) {
13        this.greeting = greeting;
14    }
15 }
```

Η κλάση GreetingServiceImpl έχει μία μόνο μεταβλητή μέλος, την greeting. Αυτή είναι ένα απλό String το οποίο θα κρατάει τον χαιρετισμό που θα τυπωθεί, όταν θα καλεστεί η μέθοδος *sayGreeting()*. Επιπλέον έχει και δύο μεθόδους δημιουργούς, έναν κενό και έναν που παίρνει σαν όρισμα τον χαιρετισμό. Οποιοσδήποτε από τους δύο αυτούς δημιουργούς μπορεί να καλεστεί, ανάλογα με τις ρυθμίσεις που θα οριστούν. Παρακάτω φαίνεται το αρχείο ρυθμίσεων (configuration file) (hello.xml) το οποίο ορίζει στον πυρήνα της Spring πώς ακριβώς πρέπει να εκτελέσει την υπηρεσία που δημιουργήθηκε.

Listing 3.3: hello.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns=
3          "http://www.springframework.org/schema/beans"
4        xmlns:xsi="http://w3.org/2001/XMLSchema-instance"
5        xsi:schemaLocation="http://www.springframework.org
            /shema/beans/spring-beans-2.0.xsd">
6   <bean id="greetingService" class="GreetingServiceImpl">
7       <property name="greeting" value="Hello World!">
8   </bean>
9 </beans>
```

Στο παραπάνω αρχείο δηλώνεται ένα στιγμιότυπο της GreetingServiceImpl στον container της String στο οποίο ορίζεται ότι η μεταβλητή μέλος (property) θα έχει την τιμή "Hello World!".

Αναλύοντας λίγο τον παραπάνω XML κώδικα βλέπουμε ότι το root στοιχείο είναι το <*beans*>, κάτι που ισχύει για οποιοδήποτε αρχείο ρυθμίσεων της Spring. Το στοιχείο <*bean*> λέει στον container για μία κλάση και πώς αυτή πρέπει να αρχικοποιηθεί. Το χαρακτηριστικό (attribute) id του στοιχείου <bean> χρησιμοποιείται, για να ορίσει το όνομα του bean, ενώ το class χρησιμοποιείται για να οριστεί το ακριβές όνομα της κλάσης (class path).

Μέσα στο στοιχείο <bean> ορίζεται ένα στοιχείο <property> το οποίο χρησιμοποιείται, για να οριστεί μία μεταβλητή μέλος, σε αυτή την περίπτωση η greeting. Το στοιχείο property λέει στον πυρήνα να καλέσει την μέθοδο sayGreeting δίνοντάς της την τιμή "Hello World" κατά την αρχικοποίηση του bean.

Παρακάτω φαίνεται το τι είναι αυτό που κάνει ακριβώς ο πυρήνας κατά την αρχικοποίηση της υπηρεσίας σύμφωνα με το XML αρχείο ρυθμίσεων.

```
1 GreetingServiceImpl greetingService =
2     new GreetingServiceImpl();
3 greetingService.setGreeting("Hello World!");
```

Αυτό που μένει είναι η δημιουργία μίας κλάσης η οποία να φορτώνει τον πυρήνα της Spring και να τον χρησιμοποιεί για την εκτέλεση της υπηρεσίας.

Listing 3.4: HelloApp.java

```
1  import org.springframework.beans.factory.BeanFactory;
2  import org.springframework.beans.factory.xml.
       XmlBeanFactory;
3  import org.springframework.core.io.FileSystemResource;
4
5  public class HelloApp {
6    public static void main(String[] args)
7         throws Exception
8    {
9      BeanFactory factory = new XmlBeanFactory
10         (new FileSystemResource("hello.xml"));
```

```
11      GreetingService service = (GreetingService)
12          factory.getBean("greetingService");
13      service.sayGreeting();
14    }
15  }
```

Η κλάση *BeanFactory* που χρησιμοποιείται παραπάνω αποτελεί τον πυρήνα της Spring. Μετά τη φόρτωση του hello.xml αρχείου από τον πυρήνα η main(·) καλεί τη μέθοδο getBean(·), ώστε να ανακτήσει τις εξαρτήσεις της υπηρεσίας. Έχοντας αυτές τις εξαρτήσεις καλείται τελικά η μέθοδος *sayGreeting(·)*. Όταν εκτελεστεί η εφαρμογή, όπως αναμενόταν, τυπώνεται το:

```
Hello World!
```

Η παραπάνω εφαρμογή αποτελεί το πιο απλό πιθανό παράδειγμα χρήσης της Spring. Παρά την απλότητά του παρουσιάζει το βασικό τρόπο ρύθμισης και χρήσης κλάσεων μέσω της Spring. Όμως, λόγω της απλότητάς του, δε δείχνει πώς μπορεί να ρυθμιστεί κάποιο bean, ώστε να περαστεί η τιμή ενός String στο πεδίο (injection). Η πραγματική δύναμη της Spring έγκειται στο πώς μπορούν τα διάφορα beans να συνδεθούν με άλλα beans κάνοντας χρήση του DI.

3.4 Κατανοώντας το dependency injection

Αν και η Spring έχει πολλές δυνατότητες, το DI αποτελεί την καρδιά του περιβάλλοντος εργασίας. Παρά το γεγονός ότι ο όρος αυτός ακούγεται σαν μια περίπλοκη προγραμματιστική τεχνική ή τρόπο σχεδιασμού (design pattern) στην πραγματικότητα δεν είναι τόσο πολύπλοκο όσο ακούγεται. Αντίθετα, η χρήση του κάνει τον κώδικα πολύ πιο απλό, ευκολονόητο και εύκολα συντηρήσιμο.

Κάθε μη τετριμμένη εφαρμογή (δηλαδή οτιδήποτε πιο πολύπλοκο από το κλασικό παράδειγμα HelloWorld.java) αποτελείται από δύο ή περισσότερες κλάσεις οι οποίες συνεργάζονται, για να εκτελέσουν τις απαραίτητες ενέργειες. Παραδοσιακά κάθε αντικείμενο είναι υπεύθυνο να διατηρεί τις αναφορές του (reference) στα αντικείμενα με τα οποία 'συνεργάζεται' (δηλαδή τις εξαρτήσεις του — de-

pendencies). Κάτι τέτοιο οδηγεί σε έναν ισχυρά δεμένο και δύσκολο στον έλεγχο κώδικα.

Όταν γίνεται χρήση του DI, τα αντικείμενα λαμβάνουν τις εξαρτήσεις τους κατά την ώρα της δημιουργίας τους από μία εξωτερική οντότητα η οποία συντονίζει κάθε αντικείμενο στο όλο σύστημα. Με άλλα λόγια, οι εξαρτήσεις (dependencies) δίνονται (injected) στα αντικείμενα. Έτσι, το DI αποτελεί ενός είδους αντιστροφής στην ευθύνη που αφορά τη διατήρηση των αναφορών των αντικειμένων. Η λειτουργικότητα αυτή παρέχεται από τον πυρήνα της Spring, ο οποίος είναι υπεύθυνος να δίνει στα αντικείμενα τις εξαρτήσεις τους. Ο πυρήνας γνωρίζει πώς πρέπει να δώσει αυτές τις εξαρτήσεις στα κατάλληλα αντικείμενα διαβάζοντας το XML αρχείο ρυθμίσεων της Spring.

Το κλειδί της παραπάνω τεχνικής είναι η χαλαρή διασύνδεση. Αν ένα αντικείμενο γνωρίζει για τις εξαρτήσεις του μόνο μέσω μίας διεπαφής (όχι μέσω της υλοποίησής τους ή μέσω του τρόπου που δημιουργούνται), η εξάρτηση μπορεί να αλλαχτεί πολύ εύκολα από μια διαφορετική υλοποίηση χωρίς το αντικείμενο να γνωρίζει για τη διαφορά.

3.5 Aspect-Oriented Programming

Όπως περιγράφηκε στην προηγούμενη παράγραφο, το DI καθιστά εφικτή τη χαλαρή διασύνδεση τμημάτων λογισμικού σε εφαρμογές. Αντίθετα, το aspect-oriented programming επιτρέπει τη συγκέντρωση λειτουργιών οι οποίες χρησιμοποιούνται σε όλη την εφαρμογή σε επαναχρησιμοποιήσιμα κομμάτια.

Το aspect-oriented programming συχνά ορίζεται ως μια τεχνική η οποία προάγει το διαχωρισμό ευθυνών μέσα στο σύστημα λογισμικού. Τα συστήματα αποτελούνται από πολλά ξεχωριστά κομμάτια το καθένα από τα οποία είναι υπεύθυνο για συγκεκριμένο κομμάτι λειτουργικότητας. Συχνά όμως αυτά τα κομμάτια έχουν επιπλέον ευθύνες πέρα από αυτές τις βασικές λειτουργίες για τις οποίες προορίζονται. Υπηρεσίες όπως είναι η διατήρηση αρχείου, η διαχείριση συναλλαγών και η ασφάλεια συχνά υλοποιούνται μέσα σε κομμάτια λογισμικού των οποίων η βασική ευθύνη είναι τελείως διαφορετική.

Μοιράζοντας τέτοιου είδους ευθύνες κατά μήκος πολλαπλών κομματιών κώδικα δημιουργούνται δύο επίπεδα πολυπλοκότητας στον κώδικα που αναπτύσ-

σεται:

- Ο κώδικας ο οποίος υλοποιεί λειτουργίες που αφορούν όλη την εφαρμογή είναι διασκορπισμένος σε πολλά διαφορετικά τμήματα κώδικα αντί να είναι συγκεντρωμένος. Αυτό σημαίνει ότι, αν ο προγραμματιστής αποφασίσει να αλλάξει τον τρόπο που εκτελούνται τέτοιου είδους λειτουργίες, πρέπει να ανατρέξει σε πολλά διαφορετικά κομμάτια. Ακόμα και αν η διαφοροποίηση έγκειται μόνο στην αλλαγή του τρόπου κλήσης μίας και μόνο μεθόδου, η διόρθωση θα πρέπει να γίνει σε πολλά διαφορετικά κομμάτια κώδικα.

- Τα διάφορα κομμάτια κώδικα είναι επιφορτισμένα με λειτουργίες οι οποίες δεν τα αφορούν προσθέτοντάς τους επιπλέον κώδικα. Για παράδειγμα, αν η λειτουργία μίας μεθόδου είναι να προσθέσει μία εγγραφή σε έναν τηλεφωνικό κατάλογο, δεν πρέπει να την αφορά το αν αυτή η διαδικασία γίνει με ασφάλεια.

Το AOP κάνει εφικτό το διαχωρισμό των διαφόρων υπηρεσιών επιτρέποντας να γίνεται χρήση τους, όποτε κάτι τέτοιο κρίνεται επιθυμητό, από τα κομμάτια κώδικα (components) που τις χρειάζονται. Αυτό οδηγεί στην ανάπτυξη πιο συνεκτικών κομματιών τα οποία είναι περισσότερο συγκεντρωμένα στο βασικό τους στόχο, αγνοώντας τελείως τις διάφορες άλλες υπηρεσίες του συστήματος. Με λίγα λόγια η χρήση των aspects διασφαλίζει ότι τα POJOs θα παραμείνουν απλά (plain).

Κεφάλαιο 4

Οργάνωση και Ανάπτυξη Κώδικα

Με τους όρους ανάπτυξη κώδικα ή προγραμματισμός στην επιστήμη των υπολογιστών συχνά αναφερόμαστε στη διαδικασία ανάπτυξης, ελέγχου και συντήρησης πηγαίου κώδικα προγραμμάτων για ηλεκτρονικούς υπολογιστές. Η διαδικασία αυτή μπορεί να αποδειχτεί πολύ δύσκολη, απαιτώντας από τον προγραμματιστή πολλές διαφορετικές δεξιότητες και γνώσεις. Εκτός αυτού, πλέον η πολυπλοκότητα και το μέγεθος του μεγαλύτερου μέρους των σύγχρονων προγραμμάτων έχει μεγαλώσει σε τέτοιο βαθμό που κάνει την ανάπτυξή τους ακόμη πιο δύσκολη. Για το λόγο αυτό έχουν αναπτυχθεί πολλά εργαλεία τα οποία διευκολύνουν αρκετά τη δουλειά του προγραμματιστή προσπαθώντας ταυτόχρονα να διασφαλίσουν την παραγωγή ποιοτικού κώδικα.

4.1 Ποιότητα Κώδικα

Η ερμηνεία του όρου ποιότητα όσον αφορά τον κώδικα μιας εφαρμογής μπορεί να ποικίλει ανάλογα με τη φύση αυτής. Για παράδειγμα σε μία εφαρμογή που χειρίζεται δεδομένα σε πραγματικό χρόνο ίσως δοθεί μεγαλύτερη έμφαση στην ταχύτητα εκτέλεσης και όχι στις υπόλοιπες συνηστώσες ποιότητας. Αντίθετα, σε μία εφαρμογή που πραγματοποιεί διατραπεζικές συναλλαγές είναι απαραίτη μεγαλύτερη ασφάλεια. Παρ΄ όλα αυτά, είναι γενικά αποδεκτό ότι, οποιαδήπο-

τε και αν είναι η προσέγγισή της ανάπτυξης λογισμικού, το πρόγραμμα τελικά θα πρέπει να πληρεί κάποια βασικά κριτήρια. Τα ποιο σημαντικά από αυτά τα χαρακτηριστικά περιγράφονται παρακάτω:

- Αποδοτικότητα (Efficiency) — Η αποδοτικότητα αφορά την όσο το δυνατόν πιο αποδοτική χρήση των πόρων του συστήματος. Με τον όρο πόροι ενός συστήματος συνήθως αναφερόμαστε στη μνήμη, τόσο την μόνιμη (σκληρός δίσκος) όσο και την προσωρινή (RAM - cache), τον επεξεργαστή, τις περιφερειακές συσκευές, το δίκτυο, κ.ά.

- Αξιοπιστία (Reliability) — Ο παραγόμενος κώδικας πρέπει να είναι αξιόπιστος. Δηλαδή, θα πρέπει να υλοποιεί με ακρίβεια αυτά που έχουν οριστεί από τις προδιαγραφές προβλέποντας τα διάφορα προβλήματα που μπορεί να προκύψουν κατά τη διάρκεια της εκτέλεσης (όπως η υπερχείλιση ή η έλλειψη μνήμης).

- Ευρωστία (Robustness) — Ο προγραμματιστής θα πρέπει να προβλέψει όσο το δυνατόν περισσότερες περιπτώσεις κατά τις οποίες, λόγω της ασυμβατότητας των διαφόρων δεδομένων ή στοιχείων προερχόμενων από το χρήστη, υπάρχει πιθανότητα να προκύψουν σφάλματα κατά την ώρα εκτέλεσης. Τέτοιου είδους σφάλματα θα πρέπει να συμβαίνουν όσο γίνεται πιο σπάνια και στην περίπτωση που συμβούν να περιγράφονται κατάλληλα με μηνύματα σφάλματος.

- Μεταφερσιμότητα (Portability) — Η τεχνολογία πλέον εξελίσσεται ταχύτατα τόσο σε επίπεδο υλικού όσο και σε επίπεδο λογισμικού. Έτσι είναι πολύ χρήσιμο ο παραγόμενος κώδικας να μπορεί να μεταφέρεται σε οποιοδήποτε περιβάλλον και να μπορεί να εκτελεστεί χωρίς κάποια επιπλέον προσπάθεια επαναπρογραμματισμού.

- Αναγνωσιμότητα (Readability) — Η ανάπτυξη και η συντήρηση του κώδικα δεν είναι πλέον θέμα ενός μόνο ατόμου. Η ανάπτυξη γίνεται στα πλαίσια ομάδων όπου κάθε άτομο αναλαμβάνει ένα μικρό κομμάτι της όλης εφαρμογής. Έτσι είναι πολύ χρήσιμο ο κώδικας που παράγουν τα διάφορα μέλη των ομάδων να είναι ευανάγνωστος και κατανοητός έτσι, ώστε να

μπορεί οποιοδήποτε μέλος της ομάδας να διαβάσει οποιοδήποτε μέρος του κώδικα εφαρμογής.

Στη συνέχεια θα περιγραφούν διάφορα εργαλεία μέσω των οποίων μειώνεται ο κόπος του προγραμματιστή κατά την διάρκεια της ανάπτυξης λογισμικού, ενώ παράλληλα ελέγχεται η ποιότητα του λογισμικού που παράγεται. Η ανάλυση θα επικεντρωθεί σε εργαλεία που αφορούν τη γλώσσα προγραμματισμού Java. Παρόμοια εργαλεία, όμως, έχουν αναπτυχθεί και είναι διαθέσιμα για τις περισσότερες δημοφιλείς γλώσσες προγραμματισμού.

4.2 Έλεγχος Ποιότητας Κώδικα

Ο έλεγχος του κατά πόσο ο κώδικας μίας εφαρμογής είναι ποιοτικός ή όχι δεν είναι κάτι το τετριμμένο. Αντίθετα, αποτελεί μία πολύπλοκη διαδικασία η οποία απαιτεί αρκετό χρόνο και προσπάθεια. Όσο αργότερα στην διαδικασία ανάπτυξης αρχίσει να γίνεται ο έλεγχος τόσο πιο δύσκολο είναι να επιτευχθεί το επιθυμητό αποτέλεσμα. Επιπρόσθετα, δεν είναι εύκολο για τον προγραμματιστή να ελέγχει την ποιότητα του κώδικα που παράγει, αφού, αν μπορούσε εύκολα να εντοπίσει τα σημεία στα οποία υστερούσε, θα τα είχε διορθώσει από την αρχή.

Για την, κατά κάποιο τρόπο, πιστοποίηση της ποιότητας του παραγόμενου κώδικα έχουν αναπτυχθεί πολλά εργαλεία. Τα πιο διαδεδομένα κάνουν ***στατική ανάλυση κώδικα*** (***static code analysis***), δηλαδή διαβάζουν και αναλύουν τον κώδικα ελέγχοντας για τυχόν λάθη, παραλήψεις ή μη αποδοτικές τεχνικές προγραμματισμού. Στα πλαίσια της παρούσας διπλωματικής εργασίας χρησιμοποιήθηκε το ***PMD***, ένας δυναμικός αναλυτής για κώδικα Java. Ο αναλυτής αυτός διαβάζει τον κώδικα και παράγει μία αναφορά στην κατάλληλη μορφή (πχ html, xml, κτλ) ανάλογα με το τι του έχει ορίσει ο χρήστης. Η αναφορά αυτή περιέχει παρατηρήσεις για διάφορα θέματα (ανάλογα με τους κανόνες που έχουν οριστεί) χωρισμένες σε πέντε επίπεδα σοβαρότητας. Ο προγραμματιστής μπορεί, διαβάζοντας την αναφορά, να τροποποιήσει τον κώδικά του κάνοντάς τον πιο ποιοτικό.

Βέβαια η ποιότητα του κώδικα δεν μπορεί να διασφαλιστεί μόνο κάνοντας χρήση τέτοιου είδους εργαλείων. Υπάρχουν πολλές περιπτώσεις κατά τις ο-

ποίες γίνεται σπατάλη πόρων ή υπάρχουν λογικά λάθη οι οποίες δεν μπορούν να εντοπιστούν. Εντούτοις, η χρήση τους παρέχει ενός είδους μέτρο για την ποιότητα του παραγομένου κώδικα.

4.3 Έλεγχος Ορθότητας Κώδικα

Ο έλεγχος της ορθότητας του κώδικα μιας εφαρμογής, δηλαδή η πιστοποίηση ότι ο κώδικας κάνει ακριβώς αυτό για το οποίο σχεδιάστηκε, αποτελεί ένα πολύ δύσκολο και χρονοβόρο κομμάτι της ανάπτυξης λογισμικού. Το μεγαλύτερο και πιο δύσκολα αντιμετωπίσιμο πρόβλημα είναι ο εντοπισμός του κομματιού της όλης εφαρμογής το οποίο δε συμπεριφέρεται όπως θα έπρεπε ύστερα από κάποια αλλαγή σε ένα άλλο κομμάτι αυτής. Στις περισσότερες των περιπτώσεων, αν δε γίνουν κάποιες ενέργειες πρόληψης, η αντιμετώπιση τέτοιου είδους προβλημάτων παίρνει τόσο πολύ χρόνο που βγάζει όλο το χρονοδιάγραμμα ανάπτυξης εκτός ορίων.

Μία λύση για την πρόωρη αντιμετώπιση τέτοιου είδους προβλημάτων είναι η χρήση ανεξάρτητων κομματιών κώδικα ελέγχου, γνωστά ως ***test units***. Τα κομμάτια κώδικα αυτά, αναπτύσσονται για κάθε λειτουργία της εφαρμογής σε όλα τα επίπεδα της υλοποίησης και ελέγχουν κάθε φορά, αν το αποτέλεσμα είναι το αναμενόμενο. Το γεγονός αυτό επιτρέπει κάθε φορά που γίνεται μία αλλαγή σε κάποιο σημείο της εφαρμογής να ελέγχεται, αν όλα τα υπόλοιπα κομμάτια εξακολουθούν να εκτελούνται όπως και πριν. Στην περίπτωση που κάτι πάει λάθος ο προγραμματιστής μπορεί να βρει πολύ εύκολα ποια κομμάτια δεν ανταποκρίνονται όπως θα έπρεπε.

Υπάρχουν πολλά περιβάλλοντα εργασίας που υλοποιούν τέτοιου είδους test units και είναι διαθέσιμα για μια πληθώρα γλωσσών προγραμματισμού. Κατά τη διάρκεια ανάπτυξης και ελέγχου της εφαρμογής που αναπτύχθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας χρησιμοποιήθηκε το ***JUnit***, ένα πολύ σημαντικό τέτοιου είδους λογισμικό για Java εφαρμογές που χρησιμοποιείται ευρύτατα. Αυτό παρέχει όλες τις απαιτούμενες υπηρεσίες για τον έλεγχο της ορθότητας του παραγομένου κώδικα.

4.4 Συστήματα Διαχείρισης Εκδόσεων (VCS)

Όπως προειπώθηκε, πλέον το μέγεθος των εφαρμογών δεν επιτρέπει την ανάπτυξή τους από ένα μόνο άνθρωπο, αλλά κάνει αναγκαία τη συνεργασία πολλών. Αυτό δημιουργεί ένα πρόβλημα που δεν υπήρχε πριν: την έγκαιρη και χωρίς προβλήματα παρακολούθηση των αλλαγών που γίνονται από κάθε μέλος της ομάδας. Έτσι υπάρχει ανάγκη για την ύπαρξη κάποιου είδους μηχανισμού ο οποίος θα παρακολουθεί τις αλλαγές, θα εμποδίζει τις συγκρούσεις μεταξύ διαφορετικών εκδόσεων ίδιων αρχείων και θα δίνει τη δυνατότητα να έχουν όλοι πρόσβαση σε οτιδήποτε νέο προστεθεί. Τέτοιου είδους μηχανισμοί ονομάζονται ***Συστήματα Διαχείρισης Εκδόσεων*** (***Version Control Systems*** – ***VCS***).

Τα συστήματα διαχείρισης εκδόσεων είναι συνήθως αυτόνομα προγράμματα τα οποία διαχειρίζονται πολλαπλές εκδόσεις ίδιων μονάδων πληροφορίας. Οι διαφορετικές εκδόσεις διαχωρίζονται μέσω κάποιου είδους αύξοντα αριθμό και συνοδεύονται από σχόλια που προσθέτει ο χρήστης και συνδέονται με το χρήστη που πραγματοποίησε τις αλλαγές.

Κατά τη διάρκεια υλοποίησης της παρούσας διπλωματικής εργασίας χρησιμοποιήθηκε ένα σύστημα διαχείρισης εκδόσεων, το ***subversion*** — ***SVN***. Παρά το γεγονός ότι δεν χρειαζόταν συντονισμός αρχείων, κρίθηκε αρκετά χρήσιμη η χρήση αυτού έτσι, ώστε να κρατείται αρχείο των σημαντικότερων εκδόσεων και να υπάρχει δυνατότητα επαναφοράς μίας παλιότερης έκδοσης.

4.5 Ολοκληρωμένα Περιβάλλοντα Ανάπτυξης Λογισμικού (IDEs)

Για τη δημιουργία κάποιου προγράμματός σήμερα αυτό που είναι απαραίτητο είναι η ύπαρξη κάποιου είδους κειμενογράφου και ενός μεταγλωττιστή (compiler) ή διερμηνέα(interpreter) μέσω του οποίου θα είναι δυνατή η μετατροπή του κώδικα από υψηλού επιπέδου γλώσσα σε γλώσσα μηχανής. Έτσι, για παράδειγμα, για την ανάπτυξη μίας Java εφαρμογής σε ένα windows σύστημα αρκεί η ύπαρξη ενός σημειωματαρίου (notepad) και η εγκατάσταση

της επιθυμητής έκδοσης της Java Virtual Machine. Σε αυτήν την περίπτωση ο προγραμματιστής κατά την διάρκεια της ανάπτυξης της εφαρμογής του θα πρέπει να μεταγλωττίζει και να τρέχει την εφαρμογή μέσω της γραμμής εντολών ξανά και ξανά, έχοντας ως μόνο του βοηθό στην περίπτωση ύπαρξης κάποιου σφάλματος τα μηνύματα σφάλματος του μεταγλωττιστή. Αυτό σημαίνει την κατανάλωση πολύ χρόνου και κόπου ακόμη και για τις πιο απλές εφαρμογές. Όσο το μέγεθος και η πολυπλοκότητα των εφαρμογών (άρα και του κώδικα) αυξάνεται, τα μεγέθη αυτά αυξάνονται σχεδόν εκθετικά, πράγμα που καθιστά αυτή τη μέθοδο ανάπτυξης μη πρακτική.

Τη λύση σε αυτό το πρόβλημα δίνουν τα ολοκληρωμένα περιβάλλοντα ανάπτυξης λογισμικού (Integrated Development Environment) — ευρέως γνωστά ως IDEs. Τα IDEs είναι ολοκληρωμένες εφαρμογές μέσα στις οποίες μπορεί να πραγματοποιηθεί κάθε απαραίτητη διαδικασία για την ανάπτυξης μίας εφαρμογής. Οι συνηθέστερες λειτουργίες που παρέχονται είναι η συγγραφή του κώδικα, η τροποποίησή του, η μεταγλώττισή του, η εκτέλεσή του καθώς και κάποια διαδικασία εντοπισμού σφαλμάτων. Ο βασικός σκοπός τους είναι να περιορίσουν και κατά κάποιο τρόπο να κρύψουν όλες τις ρυθμίσεις που χρειάζονται για το συντονισμό των διαφορετικών κομματιών, ο συνδυασμός των οποίων είναι απαραίτητος για την επιτυχημένη εκτέλεση μίας εφαρμογής. Αυτό έχει σαν συνέπεια την ευκολότερη εκμάθηση μιας γλώσσας προγραμματισμού καθώς και την αύξηση της παραγωγικότητας

4.5.1 Γνωστά IDEs

Τα τελευταία χρόνια έχουν αναπτυχθεί πολλά ολοκληρωμένα περιβάλλοντα ανάπτυξης λογισμικού. Μερικά από τα διασημότερα και ευρύτερα χρησιμοποιούμενα είναι το Visual Studio της Microsoft, το Eclipse, το NetBeans της Sun, και το IntelliJ IDEA της JetBrains. Η υλοποίηση της εφαρμογής που αναπτύχθηκε στα πλαίσια της παρούσας διπλωματικής εργασίας έγινε στο περιβάλλον ανάπτυξης IntelliJ. Τα βασικά σημεία και δυνατότητες αυτού περιγράφονται στην ακόλουθη ενότητα.

IntelliJ IDEA

Το IntelliJ είναι ένα αρκετά δημοφιλές εμπορικό ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού, κυρίως για Java εφαρμογές. Εκτός της Java παρέχει υποστήριξη και για άλλες γλώσσες προγραμματισμού όπως JavaScript, HTML/XHTML/CSS, XML/XSL, Ruby. Επιπλέον της βασικής έκδοση της Java υποστηρίζεται η χρήση πολλών τεχνολογιών και περιβάλλοντων εργασίας όπως Spring, Hibernate, Web Services, Struts, JSP, JSF, EJB, AJAX.

Στο πακέτο παρέχεται ένας αναπτυγμένος κειμενογράφος ο οποίος προσφέρει κάποιες πολύ χρήσιμες υπηρεσίες. Τέτοιες είναι η αυτόματη παραγωγή κώδικα, η ανάλυση κώδικα κατά την ώρα πληκτρολόγησης, ο αυτόματος σχολιασμός σύμφωνα με το πρότυπο της Java και ο έλεγχος λαθών και πιθανών παραλήψεων κατά την ώρα πληκτρολόγησης

Επιπλέον δίνεται η δυνατότητα χρήσης εργαλείων που βοηθούν την παρακολούθηση της πορείας ανάπτυξης του κώδικα καθ' όλη τη διαδικασία παραγωγής. Υποστηρίζεται η χρήση αρκετών γνωστών συστημάτων διαχείρισης εκδόσεων (Version Control Systems), ενώ παράλληλα παρέχεται ένα ισχυρό εργαλείο παρακολούθησης της τοπικής ιστορίας. Διευκολύνεται η χρήση δημοφιλών Java εργαλείων όπως το JUnit και το Ant, ενώ τέλος προσφέρεται ένα πολύ εύχρηστο περιβάλλον μεταγλώττισης, εκτέλεσης και αποσφαλμάτωσης (debugging).

Ο λόγος για τον οποίο επιλέχτηκε το συγκεκριμένο εργαλείο ανάπτυξης έναντι των υπολοίπων, πέρα από το γεγονός ότι είναι πολύ εύχρηστο και εύκολο στην εκμάθηση, είναι ότι υποστηρίζει όλες τις τεχνολογίες που μελετήθηκαν και χρησιμοποιήθηκαν στα πλαίσια της παρούσας διπλωματικής εργασίας.

Κεφάλαιο 5

Προδιαγραφές Εφαρμογής

Υπάρχουν πάρα πολλές εφαρμογές των οποίων το περιεχόμενο χρειάζεται να είναι διαθέσιμο σε πάνω από μία γλώσσες. Οι πιο συνηθισμένες τέτοιες περιπτώσεις είναι οι διάφορες ιστοσελίδες μεγάλων οργανισμών οι οποίες απευθύνονται σε ανθρώπους διαφορετικών εθνοτήτων. Το γεγονός αυτό δημιουργεί την ανάγκη ύπαρξης ενός εργαλείου μέσω του οποίου θα γίνεται η διαχείριση των διαφόρων φράσεων ή κομματιών κειμένου και οι αντίστοιχες μεταφράσεις αυτών σε διάφορες άλλες γλώσσες. Στα πλαίσια της παρούσας διπλωματικής εργασίας αναπτύχθηκε μία εφαρμογή η οποία επιχειρεί να υλοποιήσει ένα τέτοιου είδους εργαλείο. Το όνομά της είναι ***Σύστημα Διαχείρισης Μεταφράσεων*** ή στα αγγλικά *Transalation Management Tool (TMT)*. Η περιγραφή των απαιτήσεών της περιγράφεται στη συνέχεια σε αυτό το κεφάλαιο.

5.1 Βασικές Έννοιες

Σε αυτή την ενότητα γίνεται ανάλυση των βασικών εννοιών που χρησιμοποιούνται στην περιγραφή της εφαρμογής.

5.1.1 Περιγραφή Χρησιμοποιούμενων Όρων

- *Κύρια Γλώσσα* (*Master Language – ML*) — Είναι η βασική γλώσσα στην οποία είναι γραμμένες όλες οι φράσεις (strings). Συνήθως αυτή είναι η

αρχική γλώσσα της εφαρμογής (Εφαρμογή Χρήστης – Target Application) που χρησιμοποιεί το εργαλείο για το συντονισμό των μεταφράσεων. Μπορεί να υπάρξει μόνο μία ML τόσο στο Σύστημα Διαχείρισης Μεταφράσεων όσο και στην Εφαρμογή Χρήστη του εργαλείου.

- *Άλλες Γλώσσες* (*Other Languages – OL*) — Άλλες γλώσσες στις οποίες οι φράσεις που υπάρχουν για τη ML πρέπει να μεταφραστούν. Είναι επιθυμητό τελικά να υπάρχουν μεταφράσεις κάθε φράσης που είναι γραμμένη στη ML σε κάθε μία OL.

- *Κατηγορία* (*Category*) — Αποτελεί μία λογική σύνδεση των φράσεων που έχουν να κάνουν με ένα συγκεκριμένο σκοπό, ενώ παράλληλα περιγράφει τη χρήση της κάθε φράσης. Για παράδειγμα, μπορεί να υπάρχουν κατηγορίες “login form”, “adminitrator menu”, “system” κ.ά.

- *Εφαρμογή Χρήστης* (*Target Application – TA*) — Είναι η εφαρμογή η οποία θα κάνει χρήση των υπηρεσιών που θα παρέχει το Σύστημα Διαχείρισης Μεταφράσεων. Σκοπός είναι η διαχείριση και αναβάθμιση των στοιχείων για τις φράσεις που έχει η εφαρμογή χρήστης.

- *Σύστημα Διαχείρισης Μεταφράσεων* (*Translation Managment Tool – TMT*) — Είναι ένα αυτόνομο σύστημα το οποίο επικοινωνεί με την Εφαρμογή Χρήστη και παρέχει τις ακόλουθες υπηρεσίες:

  - Δίνει τη δυνατότητα συγχρονισμού των περιεχομένων των φράσεων που έχει το Σύστημα Διαχείρισης Μεταφράσεων με αυτά που έχει η Εφαρμογή Χρήστης.
  - Επιτρέπει στον χρήστη να εξάγει τα περιεχόμενα των διαφόρων φράσεων τόσο στη ML όσο και στις OLs σε ένα αρχείο μορφής CSV[1], το οποίο θα είναι συμβατό μέσω απλών κειμενογράφων.
  - Επιτρέπει στο χρήστη να εισάγει τα περιεχόμενα διαφόρων φράσεων τόσο στη ML όσο και στις OLs μέσω ενός αρχείου CSV μορφής, το οποίο θα είναι συμβατό μέσω απλών κειμενογράφων.

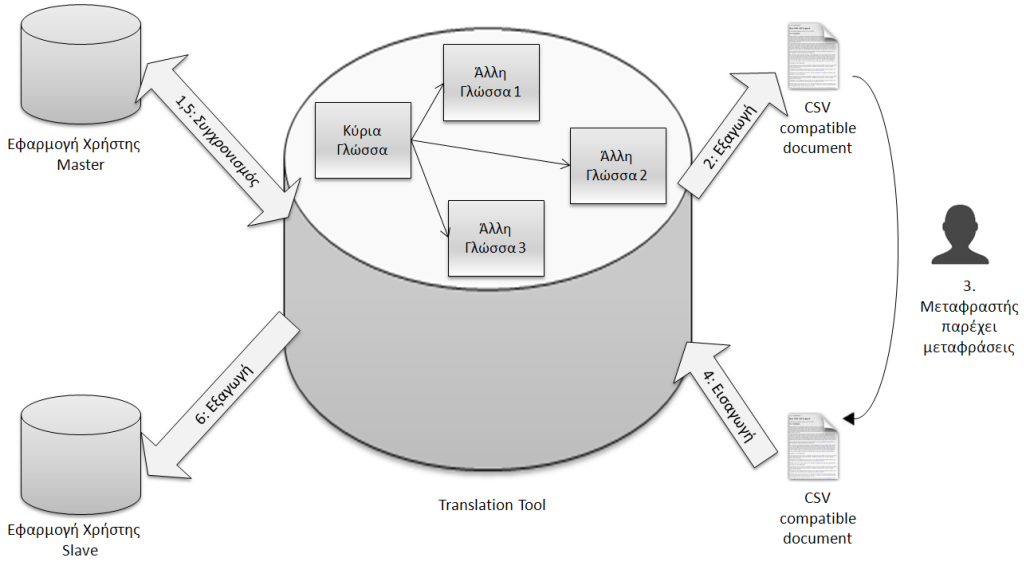
[1] Ο τύπος του αρχείου είναι CSV (Comma Seperated Values) σε UTF-8 κωδικοποίηση.

– Παράγει διάφορες αναφορές σε σχέση με την κατάσταση των διαφόρων φράσεων στο Σύστημα Διαχείρισης Μεταφράσεων.

– Παρέχει πρόσβαση στον χρήστη ανάλογα με τις πιστοποιήσεις που δέχεται από την Εφαρμογή Χρήστη.

5.1.2 Κύκλος Ζωής των Δεδομένων

Το σχήμα 5.1 δείχνει τον κύκλο ζωής των δεδομένων μέσα στο Σύστημα Διαχείρισης Μεταφράσεων. Έτσι φαίνεται η διαδικασία που ακολουθείται για την διαχείριση μεταφράσεων μέσω του συγκεκριμένου εργαλείου.



Σχήμα 5.1: Ο κύκλος ζωής των δεδομένων στο Σύστημα Διαχείρισης Μεταφράσεων

Αρχικά τα δεδομένα που υπάρχουν στην Εφαρμογή Χρήστη ανακτώνται από τη βάση αυτού και μεταφέρονται στη βάση δεδομένων του Συστήματος Διαχείρισης Μεταφράσεων. Η ενέργεια αυτή εκτελείται μέσω μίας συγκεκριμένου είδους διαδικασίας συγχρονισμού, του *'Καθαρού Συγχρονισμού' (Clean Synchronisation)*.

Στη συνέχεια ο χρήστης του Συστήματος Διαχείρισης Μεταφράσεων εξάγει τα δεδομένα που θέλει από το Συστήματος Διαχείρισης Μεταφράσεων μέσω

ενός αρχείου CSV μορφής.

Το αρχείο αυτό ο χρήστης το επεξεργάζεται μεταφράζοντας και τροποποιώντας όποιες από τις εγγραφές θέλει. Η διαδικασία αυτή πραγματοποιείται έξω από τα πλαίσια της εφαρμογής και έχει να κάνει καθαρά με την κρίση του χρήστη - μεταφραστή.

Έπειτα το τροποποιημένο αρχείο εισάγεται πίσω στο Συστήμα Διαχείρισης Μεταφράσεων και τα νέα δεδομένα εγγράφονται στη βάση δεδομένων αυτού.

Τέλος, δίνεται η δυνατότητα τόσο για συγχρονισμό των δεδομένων μεταξύ του Συστήματος Διαχείρισης Μεταφράσεων και της Εφαρμογής Χρήστη (*Δέλτα Συγχρονισμός — Delta Sychronisation*) όσο και για απευθείας αποθήκευση όλων των δεδομένων που περιέχει η βάση του Συστήματος Διαχείρισης Μεταφράσεων στην Εφαρμογή Χρήστη (*Συγχρονισμός Εξαγωγής — Data Export Sychronisation*).

5.1.3 Κύρια Γλώσσα και Άλλες Γλώσσες

Ο ορισμός τόσο της Κύριας Γλώσσας όσο και των Άλλων Γλωσσών για το Σύστημα Διαχείρισης Μεταφράσεων γίνεται μέσω ενός αρχείου ρυθμίσεων το οποίο περιέχει τις διάφορες γλώσσες και πληροφορίες για το ποια από αυτές είναι κύρια. Η ενέργεια αυτή γίνεται μόνο μία φορά κατά την αρχικοποίηση του Συστήματος Διαχείρισης Μεταφράσεων και τα στοιχεία αυτά δεν αλλάζουν καθ' όλη τη διάρκεια εκτέλεσης της εφαρμογής.

Το ακόλουθο παράδειγμα περιγράφει τον τρόπο με τον οποίο χρησιμοποιούνται η ML και οι OLs.

Αν ως Κύρια Γλώσσα έχει οριστεί η γαλλική, τότε αρχικά όλες οι εγγραφές που περιέχονται τόσο στην Εφαρμογή Χρήστη όσο και στο Συστήμα Διαχείρισης Μεταφράσεων (μέσω του Καθαρού Συγχρονισμού) είναι γραμμένες στα γαλλικά. Αν η γερμανική και η αγγλική γλώσσα είναι ορισμένες ως άλλες γλώσσες, τότε ο χρήστης μπορεί να εισάγει μεταφράσεις στις γλώσσες αυτές για τις εγγραφές που υπάρχουν για την κύρια γλώσσα.

Σημειώνεται ότι η διατήρηση ιστορίας και διαφορετικών εκδόσεων για τις αλλαγές στις μεταφράσεις δεν είναι μέσα στις προδιαγραφές του συστήματος και, εάν χρειάζονται, μπορούν να υλοποιηθούν στα πλαίσια της Εφαρμογής Χρήστη.

5.1.4 Εκτέλεση του Συστήματος Διαχείρισης Μεταφράσεων μέσω της Εφαρμογής Χρήστη

Το Σύστημα Διαχείρισης Μεταφράσεων είναι διαθέσιμο στο χρήστη της Εφαρμογής Χρήστη μέσω ενός υπερσυνδέσμου. Ο υπερσύδεσμος αυτός θα δείχνει σε μια ιστοσελίδα σύνδεσης μέσω της οποίας ο χρήστης θα μπορεί (πληκτρολογώντας το ψευδώνυμο και τον κωδικό του) να κάνει χρήση των υπηρεσιών που παρέχονται από το Σύστημα Διαχείρισης Μεταφράσεων. Η Εφαρμογή Χρήστης θα ξέρει για το url στο οποίο ακούει το Σύστημα Διαχείρισης Μεταφράσεων μέσω ενός αρχείου ρυθμίσεων το οποία θα του παρέχεται.

5.2 Κανόνες Εφαρμογής και Περιορισμοί

5.2.1 Γενικοί Κανόνες

Στη συνέχεια ακολουθούν οι κανόνες οι οποίοι ορίζουν τη λειτουργία του Συστήματος Διαχείρισης Μεταφράσεων.

- Στο Σύστημα Διαχείρισης Μεταφράσεων υπάρχει μία μόνο Κύρια Γλώσσα (ML).

- Η κάθε εγγραφή χαρακτηρίζεται από ένα ID το οποίο ορίζεται για την κάθε φράση μέσω της Εφαρμογής Χρήστη και ακολουθεί την παρακάτω μορφή:

  ID = "ID1-CAT-INTERFACE-ID2"

  ID1: Είναι ένας υποχρεωτικός μοναδικός προσδιοριστής (identifier) ο οποίος δημιουργείται από την Εφαρμογή Χρήστη. Αποτελείται από 200 αλφαριθμητικούς χαρακτήρες μέσα στους οποίος δεν περιέχεται ο χαρακτήρας '-'.

  '-': Υποχρεωτικός διαχωριστικός χαρακτήρας.

  CAT: Υποχρεωτικός τετραψήφιος αριθμός. Κάθε διαφορετικός τετραψήφιος αριθμός αντιστοιχίζεται σε μία κατηγορία φράσεων της Εφαρμογής Χρήστη.

'-': Υποχρεωτικός διαχωριστικός χαρακτήρας.

INTERFACE: Καθορίζει σε ποια διεπαφή ανήκει η φράση. Υπάρχουν δύο δυνατές τιμές, οι PUBLIC και INTERNAL. Η πρώτη επιλογή ορίζει ότι η φράση αυτή εμφανίζεται στη δημόσια διεπαφή της Εφαρμογής Χρήστη, ενώ η δεύτερη ότι ανήκει στην εσωτερική διεπαφή.

'-': Μη υποχρεωτικός διαχωριστικός χαρακτήρας.

ID2: Μη υποχρεωτικός δεύτερος προσδιοριστής.

- Όλες οι εγγραφές (είτε είναι γραμμένες στην Κύρια Γλώσσα είτε σε κάποια από τις Άλλες Γλώσσες) έχουν τα ακόλουθα στοιχεία:

  String: Είναι ένα υποχρεωτικό αλφαριθμητικό πεδίο που περιέχει την ίδια τη φράση.

  ID: Ένας προσδιοριστής η περιγραφή του οποίου έγινε παραπάνω.

  Language: Η γλώσσα στην οποία είναι γραμμένη η φράση.

  Status: Η κατάσταση στην οποία βρίσκεται η φράση. Η περιγραφή του πεδίου αυτού γίνεται παρακάτω.

  Username: Το ψευδώνυμο του χρήστη που άλλαξε τελευταία φορά τη συγκεκριμένη φράση.

  Date: Η ημερομηνία και ώρα κατά την οποία πραγματοποιήθηκε η τελευταία αλλαγή σε αυτήν τη φράση.

- Υπάρχουν τρεις διαφορετικές τιμές που μπορεί να πάρει το πεδίο Status κάθε εγγραφής:

  EMPTY (Άδεια): Είναι η αρχική κατάσταση μίας εγγραφής, όταν δεν παρέχεται για αυτήν το αντίστοιχο string. Είναι επίσης η τιμή που παίρνει η κατάσταση της εγγραφής η οποία είναι γραμμένη σε μία Άλλη Γλώσσα, όταν το περιεχόμενο της αντίστοιχης εγγραφής στην Κύρια Γλώσσα ανανεώνεται.

  CHANGED (Αλλαγμένη): Αυτήν την τιμή παίρνει μία εγγραφή, όταν, ενώ παρέχεται το string για αυτή, η μεταφρασμένη τιμή αυτού

δεν έχει ακόμη μεταφερθεί στην Εφαρμογή Χρήστη μέσω διαδικασίας συγχρονισμού που περιγράφεται στην ενότητα 5.2.4.

SYNCHRONISED (Συγχρονισμένη): υτήν την τιμή παίρνει μία εγγραφή, όταν παρέχεται το string για αυτήν, και αυτό έχει μεταφερθεί στην βάση του Εφαρμογής Χρήστη μέσω μεθόδου συγχρονισμού που περιγράφεται στην ενότητα 5.2.4.

- Οι μεταφράσεις των εγγραφών στις Άλλες Γλώσσες αναμένεται να εισαχθούν από τους χρήστες του συστήματος.

- Όταν κάποια εγγραφή που είναι γραμμένη στην Κύρια Γλώσσα τροποποιηθεί, όλες οι αντίστοιχες εγγραφές στις Άλλες Γλώσσες αλλάζουν την τιμή του status τους σε EMPTY, ενώ παράλληλα διατηρούν το περιεχόμενου του πεδίου string.

5.2.2 Εξαγωγή Δεδομένων σε Αρχείο

Με τη διαδικασία της εξαγωγής δεδομένων (export) δίνεται η δυνατότητα στο χρήστη να εξάγει όλα ή συγκεκριμένα δεδομένα κάποιας γλώσσας στο τοπικό του σύστημα μέσω ενός αρχείου. Το αρχείο αυτό στη συνέχεια μπορεί να δοθεί σε κάποιο μεταφραστή έτσι, ώστε αυτός να παρέχει τις επιθυμητές μεταφράσεις.

Η γλώσσα εγγραφές της οποίας επιλέγονται για εξαγωγή κλειδώνεται κατάλληλα έτσι, ώστε να αποτρέψει οποιοδήποτε χρήστη (συμπεριλαμβανομένου και αυτού που ζήτησε την εξαγωγή) από το να κάνει άλλη εξαγωγή. Επιπλέον, το κλείδωμα αυτό αποτρέπει οποιοδήποτε άλλο χρήστη εκτός από αυτόν που εξήγαγε τα δεδομένα να κάνει εισαγωγή δεδομένων για τη συγκεκριμένη γλώσσα.

Στη συνέχεια περιγράφονται οι κανόνες σύμφωνα με τους οποίους πραγματοποιείται η λειτουργία εξαγωγής δεδομένων σε αρχείο.

- Η εξαγωγή γίνεται με βάση την επιλογή της επιθυμητής γλώσσας (πεδίο Language) και της επιθυμητής κατηγορίας (μέρος CAT από το πεδίο ID). Δηλαδή εξάγονται οι κατάλληλες εγγραφές ανάλογα με τις επιλογές του χρήστη. Οι δυνατές επιλογές για τη γλώσσα είναι ή μία από τις Άλλες

Γλώσσες ή η Κύρια Γλώσσα ή όλες οι γλώσσες, ενώ οι επιλογές για την κατηγορία είναι είτε μία συγκεκριμένη κατηγορία είτε όλες οι κατηγορίες.

- Οι εγγραφές που εξάγονται φιλτράρονται ανάλογα με την τιμή του πεδίου status. Οι επιλογές που δίνονται είναι η εξαγωγή όλων των εγγραφών ανεξάρτητα από την τιμή του πεδίου (Complete List), η εξαγωγή μόνο αυτών που έχουν την τιμή EMPTY, η εξαγωγή μόνο αυτών που έχουν την τιμή CHANGED και η εξαγωγή αυτών που έχουν τιμή είτε EMPTY είτε CHANGED.

- Η εξαγωγή γίνεται σε ένα αρχείο το οποίο θα είναι αναγνώσιμο από κάποια CSV εφαρμογή. Το αρχείο θα περιέχει κείμενο στο οποίο οι τιμές χωρίζονται μέσω κομμάτων (Comma Separated Values), ενώ θα έχει κατάληξη ".csv".

- Τα στοιχεία που εξάγονται στο αρχείο είναι τα ακόλουθα:

  - Το ID της εγγραφής.
  - Το string της αντίστοιχης εγγραφής στην Κύρια Γλώσσα.
  - Αν η εγγραφή είναι στην Κύρια Γλώσσα τότε η τιμή του string. Διαφορετικά, στην περίπτωση που το πεδίο status έχει την τιμή:
    * EMPTY και υπάρχει αποθηκευμένο το string για την εγγραφή αυτή, τότε αυτό το string εξάγεται.
    * EMPTY και δεν υπάρχει αποθηκευμένο το string για την εγγραφή αυτή, τότε το αντίστοιχο string στην Κύρια Γλώσσα εξάγεται.
    * SYNCHRONISED ή CHANGED, τότε εξάγεται το string της εγγραφής.
  - Ένα βοηθητικό πεδίο το οποίο πληροφορεί το μεταφραστή για την κατάσταση του string. Οι διαφορετικές τιμές του πεδίου αυτού είναι:

    EMPTY-OL: Όταν η κατάσταση της εγγραφής είναι EMPTY και η εμφανιζόμενη τιμή του string είναι Άλλης Γλώσσας.

    EMPTY-ML: Όταν η κατάσταση της εγγραφής είναι EMPTY και η εμφανιζόμενη τιμή του string είναι Κύριας Γλώσσας.

SYNCH: Όταν η κατάσταση της εγγραφής είναι SYNCHRONISED.

CHANGED: Όταν η κατάσταση της εγγραφής είναι CHANGED.

- Όταν κάποιος χρήστης διαλέξει κάποια γλώσσα (ή όλες τις γλώσσες) για εξαγωγή, αυτή κλειδώνεται έτσι, ώστε μόνο ο ίδιος χρήστης να μπορεί να κάνει εισαγωγή δεδομένων για τη γλώσσα αυτή. Από τον κανόνα αυτό εξαιρούνται οι χρήστες που έχουν δικαιώματα διαχειριστή, οι οποίοι μπορούν να κλειδώνουν και να ξεκλειδώνουν τις γλώσσες κατά βούληση (η περίπτωση αυτή θα εξηγηθεί στη συνέχεια στην ενότητα 5.2.4). Βέβαια, ακόμη και αν ο χρήστης είναι διαχειριστής, πάλι δεν μπορεί να κάνει εισαγωγή δεδομένων, αν δεν ακολουθήσει τη διαδικασία εξαγωγής και στη συνέχεια εισαγωγής δεδομένων.

- Από τις εγγραφές που περιέχονται στο αρχείο υπολογίζεται το άθροισμα ελέγχου (checksum) μέσω του αλγορίθμου MD5. Το άθροισμα αυτό τοποθετείται στην κορυφή του αρχείου. Τα πεδία που χρησιμοποιούνται για την παραγωγή του αθροίσματος είναι το όνομα της γλώσσας οι εγγραφές της οποίας εξάγονται, το όνομα της Κύριας Γλώσσας καθώς και η ακολουθία των IDs των εγγραφών που εξάγονται.

5.2.3 Εισαγωγή Δεδομένων από Αρχείο

Αφού εξαχθούν σε ένα CSV αρχείο και αφού τροποποιηθούν κατάλληλα από ένα μεταφραστή, τα δεδομένα εισάγονται και πάλι στο Σύστημα Διαχείρισης Μεταφράσεων μέσω της διαδικασίας εισαγωγής δεδομένων (import). Με τη διαδικασία αυτή τροποποιούνται κατάλληλα οι τιμές των εγγραφών σύμφωνα με τα περιεχόμενα του αρχείου που εισάγεται. Επίσης η γλώσσα (ή οι γλώσσες) που είχε προηγουμένως κλειδωθεί (κατά τη διαδικασία εξαγωγής) ξεκλειδώνεται και είναι έτοιμη για τον επόμενο κύκλο εξαγωγής - εισαγωγής δεδομένων.

Στη συνέχεια περιγράφονται οι κανόνες σύμφωνα με τους οποίους πραγματοποιείται η λειτουργία εισαγωγής δεδομένων από αρχείο.

- Η διαδικασία εισαγωγής δεδομένων μπορεί να πραγματοποιηθεί μόνο από τον ίδιο χρήστη που έκανε την εξαγωγή δεδομένων.

- Οι εγγραφές εισάγονται, αφού ο χρήστης επιλέξει τα κριτήρια που επιθυμεί. Τα κριτήρια αυτά είναι η γλώσσα (πεδίο Language), όπου μπορεί να επιλέξει είτε μία συγκεκριμένη γλώσσα είτε όλες τις γλώσσες, και η κατηγορία (μέρος CAT από το πεδίο ID), όπου έχει την επιλογή να επιλέξει κάποια συγκεκριμένη κατηγόρια ή όλες τις κατηγορίες.

  Αφού ο χρήστης επιλέξει τα κριτήρια που θέλει, τότε το σύστημα ελέγχει, αν αυτά συμφωνούν με τα στοιχεία που περιέχονται στο αρχείο. Αν ο έλεγχος δεν είναι επιτυχής, τότε ο χρήστης λαμβάνει ένα μήνυμα σφάλματος. Διαφορετικά, η διαδικασία συνεχίζεται κανονικά.

- Το αρχείο που εισάγεται θα πρέπει να είναι της ίδιας μορφής με το αρχείο που εξήχθη. Δηλαδή, είναι αναγκαίο να είναι τύπου CSV και να έχει την κατάληξη ".csv". Αν αυτό δεν ισχύει, τότε η εισαγωγή ματαιώνεται.

- Το άθροισμα ελέγχου που είχε τοποθετηθεί στην κορυφή του αρχείου ελέγχεται, ώστε να διασφαλιστεί η αξιοπιστία αυτού. Ο έλεγχος γίνεται σε σχέση με τις γλώσσες και τα IDs των εγγραφών που εξήχθησαν. Στην ουσία γίνεται ο επαναϋπολογισμός του αθροίσματος και αυτό συγκρίνεται με εκείνο που υπάρχει στο αρχείο. Αν τα δύο δε συμφωνούν, η εισαγωγή ματαιώνεται και ο χρήστης ενημερώνεται με κατάλληλο μήνυμα σφάλματος.

- Για κάθε εγγραφή που υπάρχει στο αρχείο που εισάγεται το Σύστημα Διαχείρισης Μεταφράσεων ελέγχει, αν υπάρχει εγγραφή σε αυτή τη γλώσσα για αυτό το ID. Αν δεν υπάρχει, τότε δημιουργεί μία **καινούργια εγγραφή** με τα στοιχεία της εγγραφής που διάβασε από το αρχείο και θέτει την τιμή του πεδίου status σε CHANGED. Διαφορετικά, **αλλάζει** τα πεδία της εγγραφής ανάλογα με τα δεδομένα του αρχείου και θέτει την τιμή του πεδίου status σε CHANGED.

- Μετά την ολοκλήρωση της διαδικασίας εισαγωγής δεδομένων από το αρχείο η γλώσσα (ή οι γλώσσες) που πήρε μέρος στην διαδικασία ξεκλειδώνεται.

- Αν επιχειρηθεί εισαγωγή δεδομένων για μία γλώσσα η οποία δεν είναι

κλειδωμένη, τότε η διαδικασία διακόπτεται και εμφανίζεται ένα κατάλληλο μήνυμα σφάλματος.

- Αν κάποια αλλαγή αφορά μία εγγραφή η γλώσσα της οποίας είναι Κύρια όλες οι αντίστοιχες εγγραφές (δηλαδή όλες οι εγγραφές με το ίδιο ID) που αφορούν Άλλες Γλώσσες αλλάζουν την κατάστασή τους σε EMPTY, ενώ κρατούν την τιμή του string.

5.2.4 Συγχρονισμός Δεδομένων με την Εφαρμογή Χρήστη

Ο συγχρονισμός δεδομένων είναι μία διαδικασία κατά την οποία το Σύστημα Διαχείρισης Μεταφράσεων και η Εφαρμογή Χρήστης ανταλλάσσουν δεδομένα με στόχο να αναβαθμίσουν την ποιότητα των δεδομένων που έχουν στις επιμέρους βάσεις δεδομένων τους.

Υπάρχουν τρία είδη συγχρονισμού:

- Καθαρός Συγχρονισμός (Clean Synchronisation) — Ανακτά όλα τα δεδομένα από τη βάση δεδομένων της Εφαρμογής Χρήστη και τα αποθηκεύει κατάλληλα στη βάση δεδομένων του Συστήματος Διαχείρισης Μεταφράσεων. Η διαδικασία αυτή χρησιμοποιείται για την αρχικοποίηση του Συστήματος Διαχείρισης Μεταφράσεων.

- Δέλτα Συγχρονισμός (Delta Sychronisation) — Είναι η διαδικασία συγχρονισμού μέσω της οποίας γίνεται η αναβάθμιση των δεδομένων των βάσεων των δύο συστημάτων. Η χρήση της γίνεται, αφού πρώτα αρχικοποιηθεί το Σύστημα Διαχείρισης Μεταφράσεων.

- Συγχρονισμός Εξαγωγής (Data Export Sychronisation) — Στέλνει όλα τα δεδομένα που έχει το Σύστημα Διαχείρισης Μεταφράσεων στην Εφαρμογή Χρήστη. Η Εφαρμογή Χρήστης διαγράφει όλα τα δεδομένα που έχει στη βάση της και τα αντικαθιστά με αυτά που λαμβάνει από το Σύστημα Διαχείρισης Μεταφράσεων.

Στη συνέχεια περιγράφονται οι βασικοί κανόνες σύμφωνα με τους οποίους πραγματοποιούνται όλες οι διαδικασίες συγχρονισμού.

- Για να πραγματοποιηθεί κάποια διαδικασία συγχρονισμού που τροποποιεί δεδομένα της Εφαρμογής Χρήστη πρέπει πρώτα να ελεγχθεί, αν ο χρήστης έχει το δικαίωμα να πραγματοποιήσει μία τέτοια ενέργεια. Για να γίνει αυτό, ο χρήστης δίνει το ψευδώνυμό και τον κωδικό του και αυτά στέλνονται μέσω του Συστήματος Διαχείρισης Μεταφράσεων στο σύστημα διαχείρισης χρηστών της Εφαρμογής Χρήστη. Αν αυτό απαντήσει ότι ο χρήστης έχει τα κατάλληλα δικαιώματα, η διαδικασία συγχρονισμού συνεχίζει κανονικά. Διαφορετικά διακόπτεται και ο χρήστης λαμβάνει το κατάλληλο μήνυμα σφάλματος.

- Συγχρονισμός μπορεί να γίνει για μία συγκεκριμένη γλώσσα ή για όλες τις γλώσσες συνολικά.

- Το Σύστημα Διαχείρισης Μεταφράσεων γνωρίζει μέσω ενός αρχείου ρυθμίσεων για μία κύρια Εφαρμογή Χρήστη μέσω της οποίας πραγματοποιούνται ο δέλτα και ο καθαρός συγχρονισμός καθώς και για τουλάχιστον μία δευτερεύουσα Εφαρμογή Χρήστη στην οποία γίνεται ο συγχρονισμός εξαγωγής.

  Για κάθε μία από τις Εφαρμογές Χρήστη το Σύστημα Διαχείρισης Μεταφράσεων γνωρίζει το url μέσω του οποίου αυτό είναι διαθέσιμο και ένα συμβολικό όνομα υψηλού επιπέδου, ώστε να μπορεί ο χρήστης να ξεχωρίζει τις διάφορες Εφαρμογές Χρήστες.

5.2.5 Παραγωγή Αναφορών

Το Σύστημα Διαχείρισης Μεταφράσεων παρέχει τη δυνατότητα παραγωγής δύο ειδών αναφορών για την παρακολούθηση της κατάστασης στην οποία αυτό βρίσκεται σε σχέση με την κατάσταση των γλωσσών και των εγγραφών που υπάρχουν σε αυτό. Τα δύο αυτά είδη είναι αναφορά Status και αναφορά Changed. Οι αναφορές αυτές παρέχονται στο χρήστη μέσω ενός απλού αρχείου κειμένου. Ακολουθεί η περιγραφή των περιεχομένων των αρχείων αυτών.

- Η αναφορά Status περιέχει:

  – Τον αριθμό των εγγραφών στην Κύρια Γλώσσα.

  - Για κάθε Άλλη Γλώσσα:
    * Το όνομα της Άλλης Γλώσσας.
    * Το συνολικό αριθμό των εγγραφών στην γλώσσα αυτή.
    * Τον αριθμό των εγγραφών που έχουν status EMPTY στη γλώσσα αυτή.
    * Το ποσοστό των εγγραφών στη γλώσσα αυτή που έχουν status EMPTY έναντι του συνολικού αριθμού των εγγραφών στη γλώσσα αυτή.
    * Τον αριθμό των εγγραφών που έχουν status διαφορετικό από SYNCHRONISED στη γλώσσα αυτή.
    * Το ποσοστό των εγγραφών στην γλώσσα αυτή που έχουν status διαφορετικό από SYNCHRONISED έναντι του συνολικού αριθμού των εγγραφών στη γλώσσα αυτή.

- Η αναφορά Changed περιέχει για κάθε εγγραφή που έχει κατάσταση CHANGED:
  - Το ID της εγγραφής.
  - Το όνομα της γλώσσας στην οποία είναι η εγγραφή.
  - Το πεδίο string της εγγραφής.
  - Το ψευδώνυμο του χρήστη που τελευταίος τροποποίησε την εγγραφή.
  - Την ημερομηνία και ώρα της τελευταίας εγγραφής.

5.2.6 Έλεγχος Πρόσβασης και Δικαιώματα

Στο Σύστημα Διαχείρισης Μεταφράσεων υποστηρίζονται δύο διαφορετικοί τύποι χρηστών, οι ***μεταφραστές*** και οι ***διαχειριστές***. Το είδος του χρήστη καθώς και τα προσωπικά του στοιχεία (όνομα, επώνυμο, κτλ) καθορίζεται από την Εφαρμογή Χρήστη κατά την είσοδό του στο Σύστημα Διαχείρισης Μεταφράσεων. Αν ο χρήστης έχει το ρόλο μεταφραστή, τότε οι ενέργειες που μπορεί να πραγματοποιήσει είναι Εξαγωγή Δεδομένων, Εισαγωγή Δεδομένων

και παραγωγή αναφοράς (οποιαδήποτε από τις δύο). Στην περίπτωση που ο χρήστης είναι διαχειριστής, μπορεί να κάνει οποιαδήποτε από τις ενέργειες που μπορεί να πραγματοποιήσει και ο απλός χρήστης καθώς και οποιαδήποτε από τις διαδικασίες συγχρονισμού. Επιπλέον αυτών, ο διαχειριστής μπορεί να κλειδώνει και να ξεκλειδώνει οποιεσδήποτε από τις γλώσσες επιθυμεί οποιαδήποτε στιγμή. Παρ΄ όλα αυτά, αν κλειδώσει μία γλώσσα μέσω του μενού διαχειριστή δεν μπορεί, να κάνει εισαγωγή δεδομένων από αρχείο. Η ενέργεια αυτή μπορεί να πραγματοποιηθεί μόνο μέσω του κύκλου εξαγωγής - εισαγωγής δεδομένων.

Κεφάλαιο 6

Ανάλυση Εφαρμογής

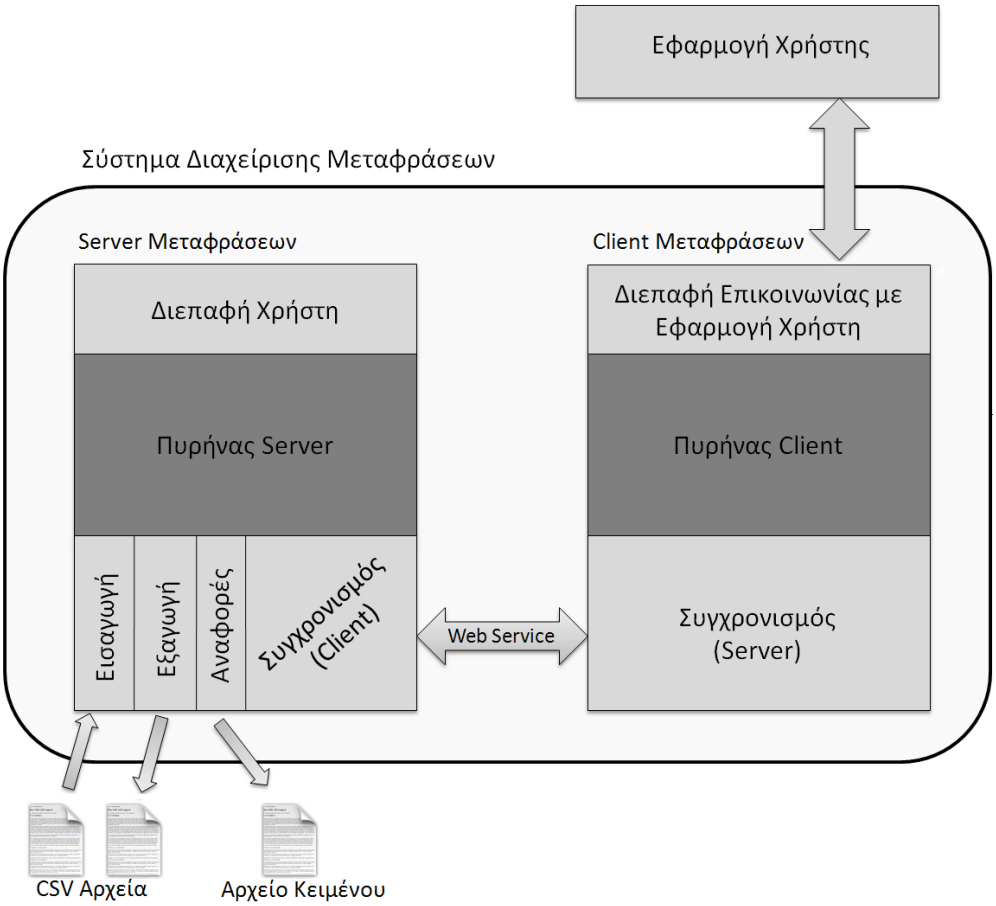
Στο κεφάλαιο αυτό θα γίνει περιγραφή του τρόπου με τον οποίο σχεδιάστηκε η εφαρμογή, ώστε να ικανοποιήσει τις απαιτήσεις που τέθηκαν στο προηγούμενο κεφάλαιο.

6.1 Γενική Περιγραφή

Όπως φαίνεται στο σχήμα 6.1 (σελίδα 56), το Σύστημα Διαχείρισης Μεταφράσεων χωρίζεται σε δύο ανεξάρτητα μέρη, στο ***Server Μεταφράσεων*** και στον ***Client Μεταφράσεων***.

Ο server μεταφράσεων είναι μία εφαρμογή η οποία μπορεί να εκτελείται αυτόνομα. Είναι αυτή που διατηρεί τη βάση δεδομένων του Συστήματος Διαχείρισης Μεταφράσεων και εκτελεί τις βασικές λειτουργίες που αυτό παρέχει. Επίσης παρέχει μία απλή διεπαφή μέσω της οποίας ο χρήστης μπορεί να συνδεθεί στο σύστημα και να εκτελέσει όποια ενέργεια επιθυμεί.

Ο client μεταφράσεων είναι μία εφαρμογή που εκτελείται μαζί με κάθε Εφαρμογή Χρήστη που επιθυμεί να λαμβάνει μέρος στη διαδικασία διαχείρισης μεταφράσεων. Παρέχει την απαραίτητη διεπαφή για την επικοινωνία του Συστήματος Διαχείρισης Μεταφράσεων και της Εφαρμογής Χρήστη, ενώ παράλληλα εκτελεί όλες τις ενέργειες που σχετίζονται με τη βάση δεδομένων της Εφαρμογής Χρήστη. Ο χειρισμός αυτός της βάσης δεδομένων γίνεται μέσω ενός interface το οποίο ορίζεται από το Σύστημα Διαχείρισης Μεταφράσεων



Σχήμα 6.1: Διάγραμμα Συστήματος Διαχείρισης Μεταφράσεων

και το οποίο οφείλει να υλοποιήσει η Εφαρμογή Χρήστης, για να είναι δυνατή η διαδικασία.

Η επικοινωνία μεταξύ των ανεξάρτητων μερών του Συστήματος επιτυγχάνεται μέσω web services. Ο πάροχος αυτών των web services είναι ο client μεταφράσεων, ενώ ο χρήστης είναι ο server. Η επικοινωνία είναι απαραίτητη μόνο στις περιπτώσεις συγχρονισμού και στη διαδικασία εξουσιοδότησης του χρήστη.

6.2 Ροή Εκτέλεσης

Ο τρόπος με τον οποίο γίνεται η παροχή των υπηρεσιών που παρέχονται από το Σύστημα Διαχείρισης Μεταφράσεων περιγράφεται παρακάτω.

Αρχικά ο χρήστης, μέσω ενός υπερσυνδέσμου ο οποίος παρέχεται από την Εφαρμογή Χρήστη ή απευθείας πληκτρολογώντας το κατάλληλο url σε έναν browser, ζητάει από το server μεταφράσεων τη σελίδα εισόδου στο σύστημα. Ο server στέλνει τη σελίδα, η οποία ζητάει από το χρήστη να εισάγει το όνομα χρήστη και τον κωδικό του. Όταν ο χρήστης εισάγει τα δεδομένα, ο server επικοινωνεί με τον client μέσω ενός web service που παρέχει ο δεύτερος και ζητάει την εξουσιοδότηση του χρήστη.

Όταν ο client δεχτεί μία αίτηση για εξουσιοδότηση, επικοινωνεί με το σύστημα διαχείρισης χρηστών που έχει η Εφαρμογή Χρήστης και μαθαίνει, αν υπάρχει αντίστοιχος χρήστης σε αυτό (και αν ναι, ποια είναι τα δικαιώματά του). Την απάντηση που παίρνει την προωθεί στον server.

Όταν φτάσει η απάντηση για τον χρήστη και τα δικαιώματά του, ο server προχωρά στις κατάλληλες ενέργειες. Αν η απάντηση είναι αρνητική (ότι δηλαδή δεν υπάρχει τέτοιος χρήστης στη βάση) δεν επιτρέπεται καμμία ενέργεια στο χρήστη. Αν η απάντηση είναι θετική, επιτρέπονται στο χρήστη οι κατάλληλες ενέργειες ανάλογα με τα δικαιώματά του μέσω της αντίστοιχης διεπαφής.

Αν ο χρήστης είναι απλός μεταφραστής, όπως περιγράφηκε και στο προηγούμενο κεφάλαιο, ο χρήστης μπορεί να κάνει μόνο Εξαγωγή Δεδομένων, Εισαγωγή Δεδομένων και παραγωγή Αναφοράς. Σε κάθε μία από αυτές τις περιπτώσεις οι ενέργειες εκτελούνται τοπικά στο server μεταφράσεων. Δηλαδή, δε λαμβάνει χώρα κανενός είδους επικοινωνία με τον client μεταφράσεων.

Στην περίπτωση που ο χρήστης είναι διαχειριστής, για τις ενέργειες συγχρονισμού τα δύο τμήματα του Συστήματος Διαχείρισης Μεταφράσεων επικοινωνούν για την εκτέλεση τους. Σε κάθε περίπτωση, αφού ο χρήστης ζητήσει την εκτέλεση της επιθυμητής ενέργειας, ο server μεταφράσεων ζητάει την εξυπηρέτηση από τον client μέσω των web services. Σημειώνεται ότι σε όλες τις περιπτώσεις συγχρονισμού οι οποίες τροποποιούν δεδομένα της Εφαρμογής Χρήστη (συγχρονισμός εξαγωγής και δέλτα συγχρονισμός) πριν την εκτέλεση του συγχρονισμού εκτελείται η διαδικασία εξουσιοδότησης που περιγράφηκε

παραπάνω.

6.3 Ροή Εκτέλεσης Συγχρονισμών

Όπως προαναφέρθηκε, υπάρχουν τρία είδη συχρονισμών (Καθαρός, Δέλτα και Εξαγωγής) μέσω τον οποίων ο διαχειριστής συγχρονίζει τις βάσεις δεδομένων των δύο επιμέρους συστημάτων, της Εφαρμογής Χρήστη και του Συστήματος Διαχείρισης Μεταφράσεων.

Για να εκτελεστεί οποιαδήποτε από αυτές τις ενέργειες, πρέπει να πληρούνται οι εξής προϋποθέσεις:

1. Ο χρήστης πρέπει να είναι εξουσιοδοτημένος διαχειριστής και να έχει ακολουθήσει τη διαδικασία εισόδου του Συστήματος Διαχείρισης Μεταφράσεων.

2. Το Σύστημα Διαχείρισης Μεταφράσεων πρέπει να μπορεί να επικοινωνήσει με την Εφαρμογή Χρήστη μέσω διαδικτύου.

Στη συνέχεια ακολουθεί η περιγραφή της ροής εκτέλεσης για κάθε έναν από τους συγχρονισμούς.

Καθαρός Συγχρονισμός

1. Ο χρήστης εκκινεί τη διαδικασία Καθαρού Συγχρονισμού.
2. Το Σύστημα Διαχείρισης Μεταφράσεων (Server Μεταφράσεων) διαγράφει όλες τις εγγραφές που συμφωνούν με τις επιλογές του χρήστη από τη βάση του.
3. Τα προς μετάφραση δεδομένα μαζί με τις κατάλληλες πληροφορίες που τα συνοδεύουν (γλώσσα, κατηγορία κτλ) ανακτώνται από τη βάση δεδομένων της Εφαρμογής Χρήστη από τον Client Μεταφράσεων και στέλνονται στο Server Μεταφράσεων μέσω των ενδιάμεσων web services. Όταν ο Server Μεταφράσεων λάβει τα δεδομένα, τα εγγράφει στη βάση δεδομένων του.

Δέλτα Συγχρονισμός

1. Ο χρήστης εκκινεί τη διαδικασία Δέλτα Συγχρονισμού.
2. Ο Server Μεταφράσεων παράγει μία λίστα με όλα τα διαφορετικά IDs που υπάρχουν στη βάση του Συστήματος Διαχείρισης Μεταφράσεων.
3. Ο Server Μεταφράσεων στέλνει τη λίστα που δημιούργησε στο βήμα 2 στον Client Μεταφράσεων ζητώντας του μία λίστα που να έχει τα IDs που περιέχονται στη βάση δεδομένων της Εφαρμογής Χρήστη και όχι στη λίστα που του έστειλε.
4. Ο Server Μεταφράσεων λαμβάνει τη λίστα των IDs που δεν περιέχονται στη βάση του.
5. Ο Server Μεταφράσεων στέλνει τη λίστα με τα IDs που δεν περιέχονται στη βάση του στον Client Μεταφράσεων ζητώντας τις εγγραφές που αντιστοιχούν σε αυτά τα IDs.
6. Ο Server Μεταφράσεων λαμβάνει τις εγγραφές που ζήτησε στο βήμα 5 και τις αποθηκεύει στη βάση του.
7. Ο Server Μεταφράσεων στέλνει μία λίστα με όλες τις εγγραφές που έχουν κατάσταση CHANGED στον Client Μεταφράσεων.
8. Ο Client Μεταφράσεων, αφού λάβει τη λίστα, διαγράφει για κάθε εγγραφή που έλαβε την αντίστοιχη τιμή του string που είναι αποθηκευμένη στην βάση δεδομένων της Εφαρμογής Χρήστη και την αντικαθιστά με την τιμή της εγγραφής που έλαβε.

Υπάρχουν τρεις διαφορετικές ροές εκτέλεσης του Δέλτα Συγχρονισμού.

Complete: Εκτελείται ολόκληρη η διαδικασία του Δέλτα Συγχρονισμού. Αποτελείται από τα βήματα 1 έως 8.

Read Only: Εκτελείται μόνο η διαδικασία εγγραφής από την Εφαρμογή Χρήστη στο Σύστημα Διαχείρισης Μεταφράσεων. Αποτελείται από τα βήματα 1 έως 6.

Write Only: Εκτελείται μόνο η διαδικασία εγγραφής από το Σύστημα Διαχείρισης Μεταφράσεων στην Εφαρμογή Χρήστη. Αποτελείται από τα βήματα 1, 6, 7 και 8.

Συγχρονισμός Εξαγωγής

1. Ο χρήστης εκκινεί τη διαδικασία Συγχρονισμού Εξαγωγής.
2. Ο Server Μεταφράσεων στέλνει όλες τις εγγραφές στον Client Μεταφράσεων.
3. Ο Client Μεταφράσεων διαγράφει όλες τις εγγραφές από τη βάση δεδομένων της Δευτερεύουσας Εφαρμογής Χρήστη και εγγράφει τις εγγραφές που έλαβε.

Σημειώνεται ότι κατά τη διαδικασία αυτή εγγράφονται μόνο εγγραφές των οποίων τα IDs προϋπήρχαν στην Εφαρμογή Χρήστη. Δηλαδή, η Δευτερεύουσα Εφαρμογή Χρήστης δε μαθαίνει για τυχόν νέες προς μετάφραση εγγραφές.

6.4 Διεπαφή Χρήστη και Λειτουργίες της

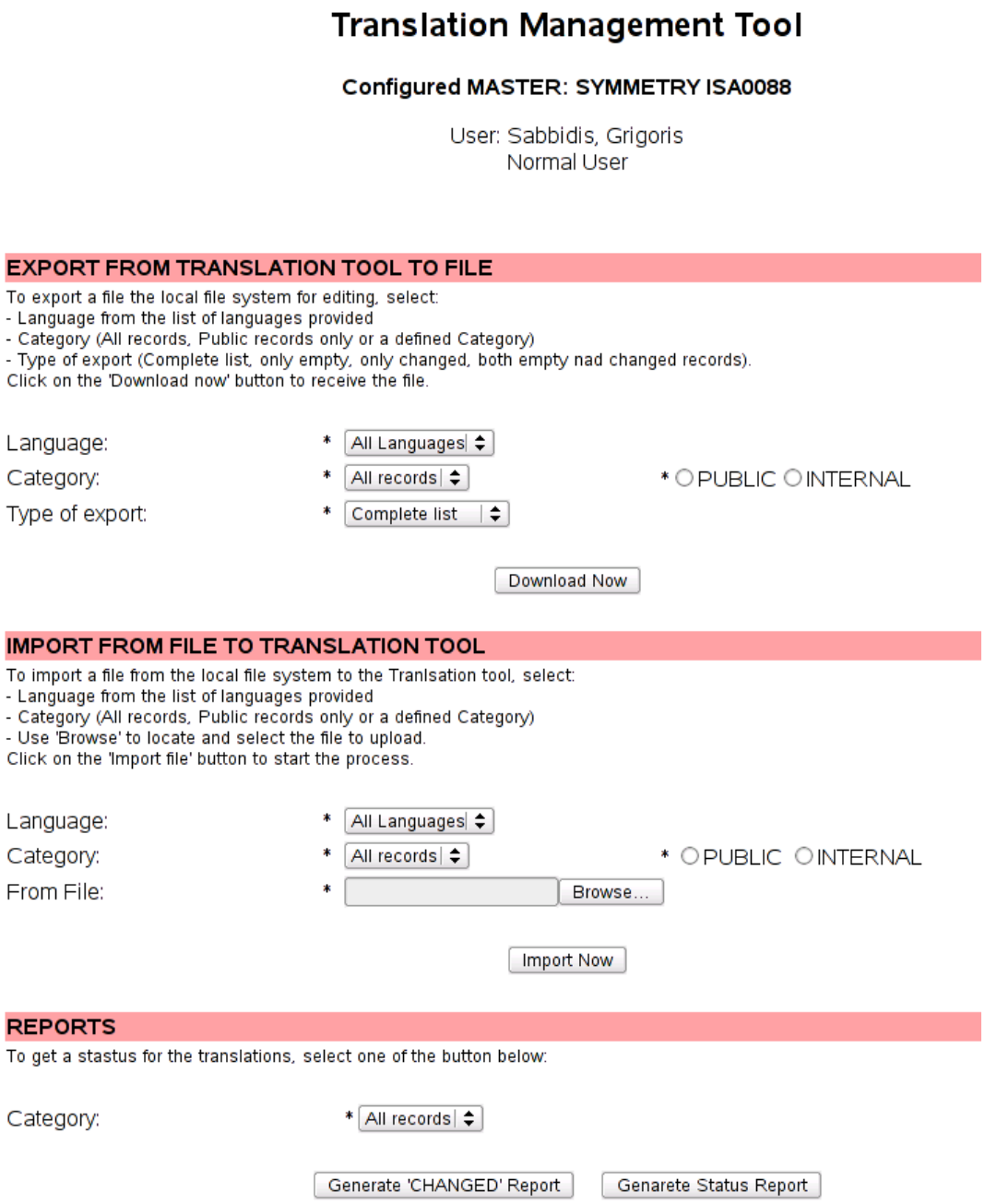
Σε αυτήν την ενότητα παρουσιάζεται η διεπαφή χρήστη που παρέχεται από το Σύστημα Διαχείρισης Μεταφράσεων, ενώ παράλληλα αναλύονται οι λειτουργίες που παρέχονται μέσω αυτής.

6.4.1 Διεπαφή Μεταφραστή

Στο σχήμα 6.2, που βρίσκεται στη σελίδα 61, παρουσιάζεται η διεπαφή χειρισμού του Συστήματος Διαχείρισης Μεταφράσεων που παρέχεται για τον απλό χρήστη - μεταφραστή.

Η διεπαφή χρήστη παρέχει τις ακόλουθες πληροφορίες:

- '*Configured Master*' — Παρουσιάζει το όνομα της Εφαρμογής Χρήστη που έχει ρυθμιστεί ως Κύρια (Master).
- '*User*' — Ο χρήστης που έχει συνδεθεί στο σύστημα. Τα στοιχεία που φαίνονται είναι το επώνυμο και το όνομα του χρήστη με τη μορφή 'Επώνυμο, Όνομα'.



Σχήμα 6.2: Διεπαφή μεταφραστή

- '*Type*' — Το είδος του χρήστη, δηλαδή, αν είναι απλός μεταφραστής ή διαχειριστής Αν ο χρήστης είναι μεταφραστής, τότε φαίνεται η φράση '*Normal User*' (απλός χρήστης), ενώ, αν είναι διαχειριστής, η φράση '*Administrator* (διαχειριστής).

Η διεπαφή χρήστη παρέχει τις ακόλουθες λειτουργίες:

- '**Export from Translation Tool to File**' — Είναι η διαδικασία εξαγωγής στο τοπικό σύστημα του χρήστη των επιθυμητών δεδομένων που περιέχονται στο Σύστημα Διαχείρισης Μεταφράσεων μέσω ενός αρχείου τύπου CSV. Η διαδικασία αυτή έχει περιγραφεί στην ενότητα 5.2.2. Ακολουθούν οι δυνατές επιλογές για την εξαγωγή των επιθυμητών δεδομένων:

  Language drop-down: Εγγραφές από την επιλεγμένη γλώσσα ή όλες τις γλώσσες (στην περίπτωση επιλογής του 'All Languages') που θα εξαχθούν στο αρχείο.

  Category drop-down: Εγγραφές από την επιλεγμένη κατηγορία ή όλες τις κατηγορίες (στην περίπτωση επιλογής του 'All Records') που θα εξαχθούν στο αρχείο.

  Public-Internal radio box: Επιλογή ανάμεσα στο ποιας διεπαφής της Εφαρμογής Χρήστη (της δημόσιας ή της εσωτερικής) οι εγγραφές θα εξαχθούν. Στην περίπτωση που δεν επιλεχθεί κανένα από τα δύο εξάγονται όλες οι εγγραφές.

  Type drop-down: Επιλογή εγγραφών που έχουν την επιθυμητή τιμή στο πεδίο status. Οι επιλογές που υποστηρίζονται είναι 'Complete list' για εξαγωγή ανεξαρτήτως της τιμής αυτού του πεδίου, 'EMPTY' για εξαγωγή αυτών που έχουν status EMPTY, 'CHANGED' για εξαγωγή αυτών που έχουν status CHANGED και 'EMPTY/CHANGED' για εξαγωγή αυτών που έχουν status είτε EMPTY είτε CHANGED.

  Για να ξεκινήσει η εκτέλεση της λειτουργίας, αρκεί να επιλεχθούν τα επιθυμητά κριτήρια εξαγωγής και να πατηθεί το κουμπί που γράφει

'Download Now'. Όταν τελειώσει η διαδικασία, εμφανίζεται ίδια διεπαφή χρήσης.

Σημειώνεται ότι, ακόμη και αν επιλεχθεί συγκεκριμένο υποσύνολο εγγραφών μέσω της υποβολής κριτηρίων που αναλύθηκε παραπάνω, η γλώσσα (ή οι γλώσσες) που συμμετέχει στη διαδικασία κλειδώνεται εξολοκλήρου (και όχι μόνο οι συγκεκριμένες εγγραφές που επιλέχθηκαν).

- '**Import from file to Translation Tool**' — Είναι η διαδικασία εισαγωγής μεταφράσεων στο Σύστημα Διαχείρισης Μεταφράσεων από το τοπικό σύστημα του χρήστη μέσω του CSV αρχείου. Η περιγραφή της διαδικασίας έχει γίνει στην ενότητα 5.2.3. Ακολουθούν οι δυνατές επιλογές για την εισαγωγή δεδομένων:

  Language drop-down: Εγγραφές από την επιλεγμένη γλώσσα ή όλες τις γλώσσες (στην περίπτωση επιλογής του 'All Languages') που θα εξαχθούν στο αρχείο.

  Category drop-down: Εγγραφές από την επιλεγμένη κατηγορία ή όλες τις κατηγορίες (στην περίπτωση επιλογής του 'All Records') που θα εξαχθούν στο αρχείο.

  Public-Internal radio box: Επιλογή ανάμεσα στο ποιας διεπαφής της Εφαρμογής Χρήστη (της δημόσιας ή της εσωτερικής) οι εγγραφές θα εξαχθούν. Στην περίπτωση που δεν επιλεχθεί κανένα από τα δύο εξάγονται όλες οι εγγραφές.

  From File: Μέσω αυτού γίνεται αναζήτηση από τον χρήστη για την επιλογή που αρχείου που θέλει να στείλει στο σύστημα.

  Για να ξεκινήσει η εκτέλεση της λειτουργίας, αρκεί να επιλεχθούν τα επιθυμητά κριτήρια εισαγωγής και να πατηθεί το κουμπί που γράφει 'Import File'. Όταν τελειώσει η διαδικασία, η ροή εκτέλεσης επανέρχεται στην ίδια διεπαφή.

  Σημειώνεται ότι, ακόμη και αν επιλεχθεί συγκεκριμένο υποσύνολο εγγραφών μέσω της υποβολής κριτηρίων που αναλύθηκε παραπάνω, η γλώσ-

σα (ή οι γλώσσες) που συμμετέχει στη διαδικασία ξεκλειδώνεται εξολοκλήρου (και όχι μόνο οι συγκεκριμένες εγγραφές που επιλέχθηκαν).

- '**Reports**' — Είναι η διαδικασία μέσω της οποίας παράγονται τα δύο είδη αναφορών. Η περιγραφή του τρόπου με τον οποίο πραγματοποιούνται περιγράφονται στην ενότητα 5.2.5. Ακολουθούν οι δυνατές επιλογές για την παραγωγή αναφορών.

  Category drop-down: Εγγραφές από την επιλεγμένη κατηγορία ή όλες τις κατηγορίες (στην περίπτωση επιλογής του 'All Records') θα χρησιμοποιηθούν για την παραγωγή της επιθυμητής αναφοράς.

  Για την παραγωγή της changed αναφοράς αρκεί να πατηθεί το 'Generate CHANGED report', ενώ για την παραγωγή της status αναφοράς αρκεί να πατηθεί το 'Generate Status report'.

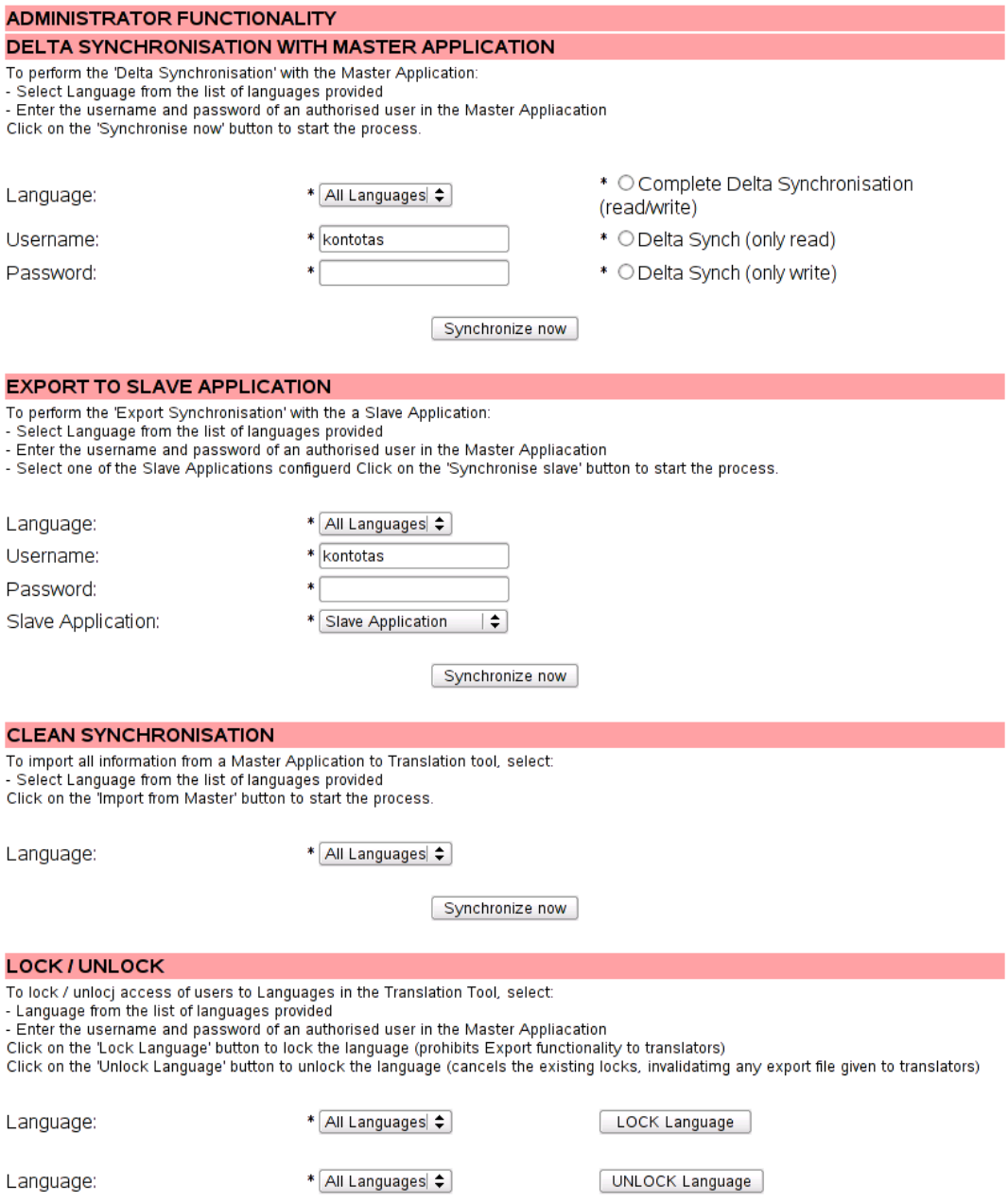
6.4.2 Διεπαφή Διαχειριστή

Η διεπαφή που παρέχεται για το χρήστη - διαχειριστή είναι ίδια με αυτή του χρήστη - μεταφραστή με τη διαφορά ότι παρέχει και κάποιες περισσότερες επιλογές. Στο σχήμα 6.3 της σελίδας 65 φαίνεται το επιπρόσθετο κομμάτι αυτής, το οποίο παρουσιάζεται στης οθόνη του χρήστη αμέσως κάτω από αυτό που φαίνεται στο σχήμα 6.2.

Η διεπαφή του διαχειριστή παρέχει τις παρακάτω επιπρόσθετες λειτουργίες:

- '**Delta Synchronisation with Master Application**' — Αφορά το Δέλτα Συγχρονισμό, τη διαδικασία μέσω της οποίας συγχρονίζονται η βάση δεδομένων του Συστήματος Διαχείρισης Μεταφράσεων και της Εφαρμογής Χρήστη. Η γενική περιγραφή αυτής έγινε στην ενότητα 5.2.4. Ο χρήστης έχει τις ακόλουθες επιλογές:

  Language drop-down: Εγγραφές από την επιλεγμένη γλώσσα ή όλες τις γλώσσες (στην περίπτωση επιλογής του 'All Languages') θα λάβουν μέρος στο Δέλτα Συγχρονισμό



Σχήμα 6.3: Επιπρόσθετη διεπαφή διαχειριστή

Complete Delta Synchronisation (read/write): Η διαδικασία κατά την οποία δεδομένα μεταφέρονται τόσο από το Σύστημα Διαχείρισης Μεταφράσεων προς την Εφαρμογή Χρήστη όσο και αντίστροφα. Η περιγραφή της διαδικασίας αυτής έγινε στην ενότητα 6.2.

Delta Synchronisation (read only): Η διαδικασία κατά την οποία δεδομένα μεταφέρονται μόνο από την Εφαρμογή Χρήστη στο Σύστημα Διαχείρισης Μεταφράσεων. Η περιγραφή της διαδικασίας αυτής έγινε στην ενότητα 6.2.

Delta Synchronisation (write only): Η διαδικασία κατά την οποία δεδομένα μεταφέρονται μόνο από το Σύστημα Διαχείρισης Μεταφράσεων προς την Εφαρμογή Χρήστη. Η περιγραφή της διαδικασίας αυτής έγινε στην ενότητα 6.2.

Αφού ο χρήστης επιλέξει το είδος του Δέλτα Συγχρονισμού που επιθυμεί καθώς και τη γλώσσα, οι εγγραφές της οποίας επιθυμεί να συγχρονιστούν, πρέπει να γράψει τον κωδικό με τον οποίο έχει πρόσβαση στην Κύρια Εφαρμογή Χρήστη (Master Target Application). Σημειώνεται ότι το όνομα χρήστη του συμπληρώνεται αυτόματα.

Όταν ο χρήστης κάνει αυτές τις ενέργειες αρκεί να πατήσει το κουμπί 'Synchronise Now' μέσω του οποίου ξεκινάει η διαδικασία. Όταν η διαδικασία ολοκληρωθεί, η ροή επιστρέφει στην ίδια διεπαφή.

- '**Export to Slave Application**' — Αφορά το Συγχρονισμό Εξαγωγής, τη διαδικασία μέσω της οποίας ο διαχειριστής μπορεί να εξάγει τα δεδομένα του Συστήματος Διαχείρισης Μεταφράσεων σε μία από τις Δευτερεύουσες Εφαρμογές Χρήστες. Η γενική περιγραφή αυτής έγινε στην ενότητα 5.2.4. Ο χρήστης έχει τις ακόλουθες επιλογές:

  Language drop-down: Εγγραφές από την επιλεγμένη γλώσσα ή όλες τις γλώσσες (στην περίπτωση επιλογής του 'All Languages') θα λάβουν μέρος στον Συγχρονισμό Εξαγωγής.

  Slave Application drop-down: Επιλέγεται η Δευτερεύουσα Εφαρμογή Χρήστης στην οποία θα εξαχθούν τα δεδομένα.

Αφού ο χρήστης επιλέξει τη γλώσσα οι εγγραφές της οποίας επιθυμεί να εξαχθούν και την Εφαρμογή Χρήστη στην οποία θα σταλούν τα δεδομένα, θα πρέπει να γράψει τον κωδικό με τον οποίο έχει πρόσβαση στην Κύρια Εφαρμογή Χρήστη (Master Target Application). Σημειώνεται ότι το όνομα χρήστη του συμπληρώνεται αυτόματα.

Όταν ο χρήστης κάνει αυτές τις ενέργειες, αρκεί να πατήσει το κουμπί 'Synchronise Slave' μέσω του οποίου ξεκινάει η διαδικασία. Όταν η διαδικασία ολοκληρωθεί, η ροή επιστρέφει στην ίδια διεπαφή.

- '**Clean Sychronisation**' — Αφορά τον Καθαρό Συγχρονισμό, τη διαδικασία μέσω της οποίας ο διαχειριστής μπορεί να αρχικοποιήσει το Συστήμα Διαχείρισης Μεταφράσεων με τα δεδομένα της Κύριας Εφαρμογής Χρήστη (master). Η γενική περιγραφή αυτής έγινε στην ενότητα 5.2.4. Ο χρήστης έχει τις ακόλουθες επιλογές:

  Language drop-down: Εγγραφές από την επιλεγμένη γλώσσα ή όλες τις γλώσσες (στην περίπτωση επιλογής του 'All Languages') θα εισαχθούν στο σύστημα.

  Η ενέργεια ξεκινάει με το πάτημα του κουμπιού που γράφει 'Import from Master'. Όταν η διαδικασία ολοκληρωθεί, η ροή επανέρχεται σε αυτή την διεπαφή.

- '**Lock / Unlock**' — Είναι η υπηρεσία μέσω της οποίας δίνεται η δυνατότητα στο διαχειριστή του συστήματος να επιτρέπει ή να απαγορεύει την πρόσβαση στους μεταφραστές για όποιες από τις γλώσσες επιθυμεί (κλείδωμα - ξεκλείδωμα γλώσσας). Η διαδικασία αυτή έχει περιγραφεί στην ενότητα 5.2.6.

  Για την πραγματοποίηση της υπηρεσίας παρέχονται δύο drop-down μενού και δύο κουμπιά, ένα για κάθε περίπτωση (lock και unlock). Και τα δύο drop-down μενού αφορούν τη γλώσσα η οποία θα υποστεί την αλλαγή κατάστασης. Στην περίπτωση που επιλεγεί το 'All Languages' η ενέργεια αφορά όλες τις γλώσσες. Για να εκτελεστούν οι λειτουργίες, στην περίπτωση του κλειδώματος αρκεί να πατηθεί το 'Lock Language', ενώ στην περίπτωση του ξεκλειδώματος το 'Unlock Language'.

Κεφάλαιο 7

Περιγραφή Υλοποίησης Εφαρμογής

Στο κεφάλαιο αυτό περιγράφεται ο τρόπος με τον οποίο υλοποιήθηκε το Σύστημα Διαχείρισης Μεταφράσεων. Γίνεται περιγραφή της γενικής δομής του κώδικα ενώ παράλληλα παρουσιάζονται κάποια βασικά κομμάτια αυτού για την καλύτερη κατανόησή του.

Όπως προειπώθηκε, το Σύστημα Διαχείρισης Μεταφράσεων χωρίζεται σε δύο επιμέρους αυτόνομα κομμάτια, τον server μεταφράσεων και τον client μεταφράσεων. Το λογισμικό που τρέχει καθένα από τα κομμάτια αυτά είναι αυτόνομο και ξεχωριστό. Έτσι η ανάλυση που ακολουθεί θα χωριστεί στα δύο αντίστοιχα μέρη.

7.1 Server Μεταφράσεων

Ο server μεταφράσεων είναι αυτός που πραγματοποιεί τις βασικές λειτουργίες που προσφέρει το Σύστημα Διαχείρισης Μεταφράσεων. Διατηρεί και χειρίζεται τη βάση δεδομένων με τις μεταφρασμένες εγγραφές, παρέχει τις διαθέσιμες διεπαφές χρήστη, εκτελεί τις επιθυμητές ενέργειες (όπως εισαγωγή και εξαγωγή δεδομένων) και επικοινωνεί με τον client μεταφράσεων για την πραγματοποίηση των διαφόρων ενεργειών συγχρονισμού.

Στη συνέχεια ακολουθεί η αναλυτική περιγραφή των επιμέρους κομματιών

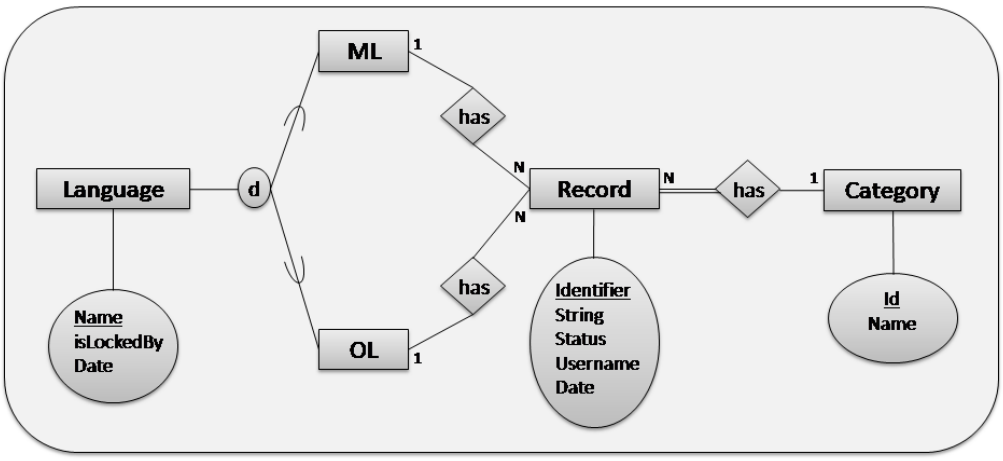
που αποτελούν τον server μεταφράσεων.

7.1.1 Η Βάση Δεδομένων

Το κομμάτι της βάσης δεδομένων του server μεταφράσεων, καθώς η γενικότερη διαχείρισή της, αποτελεί ένα από τα κυριότερα σημεία της εφαρμογής. Είναι αυτό που διατηρεί όλες τις μεταφράσεις και εκτελεί όλες ενέργειες που έχουν να κάνουν με προσθήκη νέων ή ανανέωση υπαρχουσών εγγραφών. Επιπλέον κρατάει στοιχεία για τις υποστηριζόμενες γλώσσες και κατηγορίες εγγραφών, καθώς και για τις διάφορες Εφαρμογές Χρήστες που συμμετέχουν στην διαδικασία μετάφρασης.

7.1.1.1 Περιγραφή του Σχήματος Βάσης

Στα σχήματα 7.1 και 7.2 που βρίσκονται στις σελίδες 70 και 71 αντίστοιχα, παρουσιάζεται το σχήμα της βάσης δεδομένων που δημιουργείται από τον server. Όπως φαίνεται και στα σχήματα ισχύουν οι εξής κανόνες:

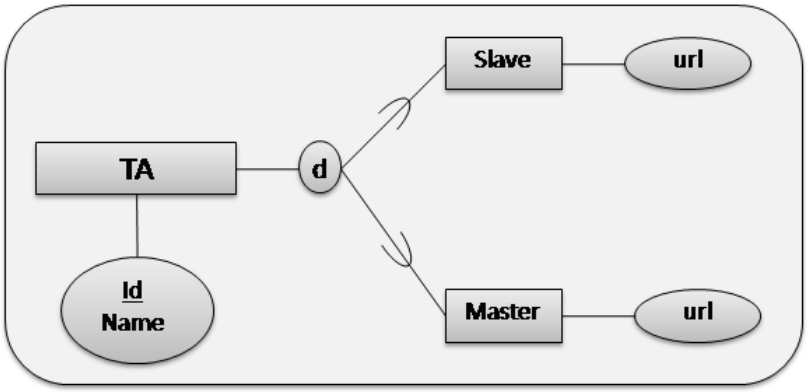


Σχήμα 7.1: Διάγραμμα οντοτήτων συσχετίσεων περιγραφής του σχήματος βάσης του server μεταφράσεων

- Υπάρχει μία οντότητα με το όνομα *Laguage*, η οποία αντιπροσωπεύει τις διάφορες γλώσσες του συστήματος. Κάθε γλώσσα έχει ένα μοναδικό

χαρακτηριστικό *Name* (όνομα), το οποίο αποτελεί και το κλειδί της. Επιπλέον, η οντότητα αυτή κρατάει πληροφορία για το ποιος χρήστης έχει κλειδώσει την γλώσσα, μέσω του πεδίου *isLockedBy*, καθώς και για το πότε έλαβε χώρα το κλείδωμα, μέσω του πεδίου *Date* (ημερομηνία).

- Υπάρχει μία οντότητα με το όνομα *Category*, η οποία αντιπροσωπεύει τις διάφορες κατηγορίες που γνωρίζει το σύστημα. Κάθε κατηγορία έχει ένα μοναδικό id και ένα όνομα (η περιγραφή των πεδίων αυτών έγινε στην ενότητα 5.2.1).



Σχήμα 7.2: Διάγραμμα οντοτήτων συσχετίσεων περιγραφής του σχήματος βάσης του server μεταφράσεων

- Υπάρχει μία οντότητα με το όνομα *Record*, η οποία αντιπροσωπεύει τις διάφορες εγγραφές φράσεων που υπάρχουν στο σύστημα. Κάθε εγγραφή διαθέτει έναν μοναδικό διαχωριστή το *dbId*, ο οποίος υπάρχει μόνο για την βάση δεδομένων (για διαχωρισμό των εγγραφών) και δεν παίζει κανέναν ουσιαστικό ρόλο στην λογική της εφαρμογής. Ακόμη, τα Records περιέχουν όλα τα στοιχεία που περιγράφηκαν στην ενότητα 5.2.1 σε σχέση με τις εγγραφές.

- Κάθε Language μπορεί να είναι είτε Κύρια Γλώσσα (Master Language – ML), είτε Άλλη Γλώσσα (Other Language – OL).

- Κάθε Record πρέπει να ανήκει μία γλώσσα, είτε αυτή είναι Κύρια, είτε Άλλη.

- Κάθε Record πρέπει να ανήκει μία κατηγορία.

- Υπάρχει μία οντότητα με το όνομα *TA* (Target Application), η οποία αντιπροσωπεύει τις διάφορες Εφαρμογές Χρήστες που είναι γνωστές στο σύστημα. Κάθε τέτοια οντότητα μπορεί να είναι είτε *Master* (Κύρια), είτε *Slave* (Δευτερεύουσα). Για κάθε ΤΑ κρατείται ένας μοναδικός διαχωριστής – *Id*, ένα όνομα για τον διαχωρισμό αυτών από τους χρήστες – *Name*, καθώς και το *url* μέσω του οποίου είναι θα γίνεται η πρόσβαση στις Εφαρμογές Χρήστες.

7.1.1.2 Χειρισμός της Βάσης Δεδομένων

Για τη δημιουργία και διαχείριση των αναγκαίων πινάκων που χρειάζονται για την υλοποίηση του σχήματος βάσης που περιγράφηκε στην προηγούμενη ενότητα δημιουργήθεκε το πακέτο ***db***. Μέσα σε αυτό το πακέτο αναπτύχθηκαν οι κατάλληλες κλάσεις για την διαχείριση των αναγκαίων αντικειμένων. Για κάθε οντότητα δημιουργήθηκε η αντίστοιχη κλάση, η οποία έχει ως μεταβλητές μέλη τα πεδία του αντίστοιχου πίνακα. Οι κλάσεις αυτές αντιστοιχίστηκαν κατάλληλα μέσω XML αρχείων στους επιθυμητούς πίνακες, μέσω των εργαλείων που παρέχει η Hibernate.

Για τον χειρισμό κάθε μίας κλάσης δημιουργήθηκε ένα interface το οποίο ορίζει τον τρόπο με τον οποίο γίνεται ο χειρισμός κάθε αντικειμένου. Κάθε τέτοιο interface έχει την κατάλληξη *Dao*. Παράλληλα, δημιουργήθηκε και μία αντίστοιχη κλάση διαχειριστής που υλοποιεί κάθε ένα τέτοιο interface. Αυτές στην ουσία αποτελούν τις κλάσεις διαχειριστές των αντίστοιχων αντικείμενου. Τα ονόματα όλων των κλάσεων διαχειριστών τελειώνουν με τη λέξη *Manager*.

Στη συνέχεια θα αναλυθεί η υλοποίηση της κάθε οντότητας.

7.1.1.3 Language

Για την αντιστοίχιση της οντότητας Language στη βάση δεδομένων δημιουργήθηκε μία κλάση με το ίδιο όνομα – ***Language***. Αυτή, είναι μέλος του πακέτου *language* και έχει ως μεταβλητές μέλη τα:

***name*:** Ένα string που περιέχει το όνομα της γλώσσας.

***isMasterLanguage*:** Μία boolean μεταβλητή που ορίζει αν η γλώσσα είναι Κύρια ή όχι.

***isLockedBy*:** Μία μεταβλητή τύπου TmtUser, που περιέχει τον χρήστη που κλείδωσε τη γλώσσα. Στην περίπτωση που η γλώσσα είναι κλειδωμένη το πεδίο περιέχει τον χρήστη (τα στοιχεία του) που πραγματοποίησε το κλείδωμα, ενώ στην αντίθετη περίπτωση η τιμή του πεδίου είναι NULL. Η περιγραφή της κλάσης TmtUser βρίσκεται στην ενότητα +++++.

***date*:** Ένα αντικείμενο τύπου date που περιέχει την στιγμή του τελευταίου κλειδώματος.

Εκτός από τις μεταβλητές μέλη, η κλάση αυτή περιέχει και κάποιες μεθόδους. Οι περισσότερες από αυτές είναι οι γνωστές μέθοδοι getters και setters, μέσω των οποίων προσπελαύνονται οι μεταβλητές μέλη από μεθόδους εκτός της κλάσης. Οι μέθοδοι αυτές είναι απαραίτητες για την αντιστοίχιση της κλάσης στη βάση δεδομένων μέσω της Hibernate. Επιπλέον, έχει γίνει υπερφόρτωση των γνωστών μεθόδων *toString(), equals()* και *hashCode()*, ενώ υπάρχει και μία μεθοδος, η *isValide*, η οποία ελέγχει αν τα στοιχεία του αντικειμένου θεωρούνται έγκυρα για να αντιστοιχιστούν σε μία γλώσσα (έτσι ώστε στη συνέχεια, για παράδειγμα, αυτό να αποθηκευτεί στη βάση).

Η αντιστοίχιση της κλάσης Language στην βάση δεδομένων έγινε μέσω ενός αρχείου ρυθμίσεων, του *Language.hbm.xml*. Το περιεχόμενο του αρχείου αυτού φαίνεται παρακάτω.

Listing 7.1: Language.hbm.xml

```
1 <?xml version="1.0"?>
2 <!DOCTYPE hibernate-mapping PUBLIC
3     "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4     "http://hibernate.sourceforge.net/hibernate-mapping
          -3.0.dtd">
5 <hibernate-mapping>
6   <class name="db.language.Language" >
7     <id name="name" length="50"/>
8     <component name="isLockedBy"
```

```
9               class="communObjects.TmtUser">
10        <property name="username"/>
11        <property name="firstname"/>
12        <property name="lastname"/>
13        <property name="isAdmin" type="true_false"/>
14      </component>
15      <property name="isMasterLanguage"
16              type="true_false" not-null="true" />
17      <property name="date"/>
18    </class>
19 </hibernate-mapping>
```

Όπως φαίνεται στον παραπάνω κώδικα, στο XML αρχείο ρυθμίσεων ορίζεται ο τρόπος αντιστοίχισης της κλάσης Language σε έναν πίνακα στη βάση δεδομένων (ο πίνακας θα έχει το ίδιο όνομα με την κλάση αφού δεν ορίζεται κάτι άλλο) και περιγράφεται ο τρόπος αντιστοίχισης των διάφορων μεταβλητών μελών της κλάσης. Το μεγαλύτερο κομμάτι του κώδικα είναι αυτοεξηγούμενο και έτσι δεν θα αναλυθεί. Τα μόνα κομμάτια που χαίρουν σχολιασμού είναι αυτά που αφορούν τα στοιχεία id και component. Μέσω του πρώτου, ορίζεται ότι το πεδίο name αποτελεί κλειδί του πίνακα που θα δημιουργηθεί στην βάση. Με το στοιχείο property περιγράφεται το σύνθετο πεδίο isLockedBy. Ορίζεται ότι το πεδίο αυτό αντιστοιχεί στην κλάση *communObjects.TmtUser* ενώ περιγράφονται κατάλληλα τα πεδιά (μεταβλητές μέλη) της κλάσης αυτής που θα αποθηκευτούν στη βάση.

Για την αποθήκευση και την γενικότερη διαχείριση της κλάσης δημιουργήθηκε το interface ***LanguageDao***. Αυτό ορίζει τον τρόπο με τον οποίο γίνεται οποιαδήποτε ενέργεια έχει να κάνει με την κλάση Language. Ο κυριότερος λόγος ύπαρξής του είναι η απαίτηση της Spring για διαχωρισμό του interface και της υλοποίησης κατά την επικοινωνία της με την Hibarnate.

Το interface αυτό υλοποιείται από την κλάση ***LanguagesManager***. Η κλάση αυτή αποτελεί το μέσο για τη διαχείριση της κλάσης Language. Ακολουθεί το γνωστό πρότυπο σχεδίασης (design pattern) *Singleton* — έτσι δημιουργείται μόνο ένα στιγμιότυπο της κλάσης. Η κλάση αυτή για λόγους απόδοσης κρατάει ένα αντικείμενο τύπου Map για την τοπική αποθήκευση των

διαφόρων γλωσσών. Αυτό έχει ως κλειδί το όνομα της γλώσσας (name) και δείχνει σε μία γλώσσα (αντικείμενο τύπου Language). Η κλάση διαθέτει μεθόδους για την αποθήκευση και ανάκτηση γλωσσών (με διάφορα κριτήρια επιλογής), καθώς και για το κλείδωμα και ξεκλείδωμα τόσο από μεταφραστή (ενέργειες εισαγωγής - εξαγωγής) όσο και από διαχειριστή.

Επιπλέον, στο πακέτο language περιέχεται η κλάση ***InvalidLanguageException***, η οποία καλείται όποτε υπάρξει οποιοδήποτε πρόβλημα με το χειρισμό κάποιας γλώσσας.

7.1.1.4 Record

Για την αντιστοίχιση της οντότητας Record στη βάση δεδομένων δημιουργήθηκε μία κλάση με το ίδιο όνομα και τοποθετήθηκε σε ένα αρχείο με όνομα ***Record.java***. Αυτή, είναι μέλος του πακέτου *record*, διαθέτει μεθόδους αντίστοιχες αυτών της κλάσης Language, ενώ έχει ως μεταβλητές μέλη τα:

***dbId*:** Ένας ακέραιος αριθμός που χρησιμοποιείται ως κλειδί στη βάση.

***recordID*:** Ένα αντικείμενο τύπου *Identifier* το οποίο αντιστοιχεί στον διαχωριστή του Συστήματος Διαχείρισης Μεταφράσεων που περιγράφικε στην ενότητα 5.2.1.

***string*:** Ένα String με το κείμενο της εγγραφής.

***language*:** Ένα αντικείμενο τύπου Language που αντιστοιχεί την γλώσσα της εγγραφής.

***status*:** Ένα String με την κατάσταση στης εγγραφής.

***lastUpdater*:** Ένα String με το ψευδώνυμο που χρήστη που έκανε τελευταίος ανανέωση της εγγραφής.

***date*:** Ένα αντικείμενο τύπου Date που περιέχει την στιγμή της τελευταίας αλλαγής.

Η αντιστοίχιση της κλάσης Record στην βάση δεδομένων έγινε μέσω ενός αρχείου ρυθμίσεων, του *Record.hbm.xml*. Το περιεχόμενο του αρχείου αυτού

φαίνεται παρακάτω (δεν περιλαμβάνοται ο ορισμός του είδους του κειμένου – header – αφού και έχει φανεί παραπάνω).

Listing 7.2: Record.hbm.xml

```
1  <hibernate-mapping>
2    <class name="db.record.Record" >
3      <id name="dbId">
4        <generator class="native"/>
5      </id>
6      <property name="string" not-null="false"/>
7      <component name="recordID"
8              class="db.record.Identifier">
9        <property name="id1" not-null="true"/>
10       <property name="cat" length="4"
11             not-null="true"/>
12       <property name="iface" length="10"
13             not-null="true"/>
14       <property name="id2" />
15     </component>
16     <component name="language"
17             class="db.language.Language">
18       <property name="name"/>
19     </component>
20     <property name="status" length="15"/>
21     <property name="lastUpdater"/>
22     <property name="date"/>
23   </class>
24 </hibernate-mapping>
```

Το σημείο που αξίζει να σχολιαστεί στον παραπάνω κώδικα είναι το στοιχείο id, το οποίο έχει ως παιδί το στοιχείο generator. Με το στοιχείο generator ορίζεται ο τρόπος με τον οποίο δημιουργείται ο διαχωριστής των εγγραφών που αποθηκέυονται στη βάση. Με τον ορσιμό της κλάσης (class) ως native δηλώνεται στην Hibernate ότι θα δημιουργεί την τιμή αυτή δίνοντάς της έναν αύξοντα ακέραιο αριθμό.

Η διαχείρηση των αντικειμένων τύπου Record γίνεται με παρόμοιο τρόπο με αυτόν των αντικειμένων τύπου Language. Υπάρχει ένα interface που ορίζει τον τρόπο χειρισμού αυτών, το ***RecordDao***, καθώς και μία κλάση που υλοποιεί το interface αυτό, η ***RecordsManager***. Η κλάση RecordsManager ακολουθεί το πρότυπο σχεδίασης *Singleton*. Η κλάση RecordsManager παρέχει μεθόδους για την δημιουργία, τροποποίηση και ανάκτηση των επιθυμητών εγγραφών.

Επιπλέον, στο πακέτο record περιέχεται η κλάση ***Identidier*** η οποία υλοποιεί τον διαχωριστή των εγγραφών που περιγράφηκε στην ενότητα 5.2.1, καθώς και μία κλάση που χρησιμοποιείται για την σύγκριση αντικειμένων τύπου Identifier, η ***IdentifierComparator***. Τέλος, υπάρχει μία κλάση, η ***InvalidRecordException***, η οποία καλείται όποτε υπάρξει πρόβλημα με κάποια εγγραφή.

7.1.1.5 Category

Για την αντιστοίχιση της οντότητας Category στη βάση δεδομένων δημιουργήθηκε μία κλάση με το ίδιο όνομα και τοποθετήθηκε σε ένα αρχείο με όνομα ***Category.java***. Αυτή, είναι μέλος του πακέτου *category*, διαθέτει μεθόδους αντίστοιχες αυτών της κλάσης Language, ενώ έχει ως μεταβλητές μέλη τα:

***id*:** Ένας ακέραιος αριθμός που αποτελεί τον μοναδικό τετραψήφιο διαχωριστή που έχει περιγραφεί την ενότητα 5.2.1.

***name*:** Ένα string που περιέχει το όνομα της κατηγορίας.

Η αντιστοίχιση της κλάσης Category στην βάση δεδομένων έγινε μέσω ενός αρχείου ρυθμίσεων, του *Category.hbm.xml*.

Η διαχείρηση των αντικειμένων τύπου Category γίνεται με παρόμοιο τρόπο με αυτόν των αντικειμένων που περιγράφησαν παραπάνω. Υπάρχει ένα interface που ορίζει τον τρόπο χειρισμού αυτών, το ***CategoryDao***, καθώς και μία κλάση που υλοποιεί το interface αυτό, η ***CategoriesManager***. Η κλάση RecordsManager ακολουθεί το πρότυπο σχεδίασης *Singleton*. Η κλάση CategoriesManager παρέχει μεθόδους για την δημιουργία και ανάκτηση των επιθυμητών εγγραφών, ενώ παράλληλα διαθέτει ένα τοπικό αντικείμενο τύπου Map που λειτουργεί ως cache για την αποθήκευση των διαθέσιμων κατηγοριών.

Επιπλέον, στο πακέτο category υπάρχει μία κλάση, η ***InvalidCategoryException***, η οποία καλείται όποτε υπάρξει πρόβλημα με κάποια κατηγορία.

7.1.1.6 TA

Για την αντιστοίχιση της οντότητας TA στη βάση δεδομένων δημιουργήθηκε μία κλάση με το ίδιο όνομα και τοποθετήθηκε σε ένα αρχείο με όνομα ***TA.java***. Αυτή, είναι μέλος του πακέτου *ta*, διαθέτει μεθόδους αντίστοιχες αυτών της κλάσης Language, ενώ έχει ως μεταβλητές μέλη τα:

***id*:** Ένας ακέραιος αριθμός που αποτελεί τον μοναδικό διαχωριστή που χρησιμοποιείται για το διαχωρισμό των διάφορων αποθηκευμένων Εφαρμογών Χρήστων.

***name*:** Ένα string που περιέχει το όνομα της Εφαρμογής Χρήστη.

***url*:** Ένα string που περιέχει το url μέσω του οποίου είναι προσβάσιμη η Εφαρμογή Χρήστης.

***isMaster*:** Μία boolean μεταβλητή που καθορίζει αν η Εφαρμογή Χρήστης είναι Κύρια ή όχι.

Η αντιστοίχιση της κλάσης TA στη βάση δεδομένων έγινε μέσω ενός αρχείου ρυθμίσεων, του *TA.hbm.xml*.

Η διαχείρηση των αντικειμένων τύπου TA γίνεται με παρόμοιο τρόπο με αυτόν των αντικειμένων που περιγράφησαν παραπάνω. Υπάρχει ένα interface που ορίζει τον τρόπο χειρισμού αυτών, το ***TADao***, καθώς και μία κλάση που υλοποιεί το interface αυτό, η ***TAManager***. Η κλάση TAManager ακολουθεί το πρότυπο σχεδίασης *Singleton*. Η κλάση TAManager παρέχει μεθόδους για τη δημιουργία και ανάκτηση των επιθυμητών εγγραφών, ενώ παράλληλα διαθέτει ένα τοπικό αντικείμενο τύπου List που λειτουργεί ως cache για την αποθήκευση των διαθέσιμων κατηγοριών.

Επιπλέον, στο πακέτο ta υπάρχει μία κλάση, η ***InvalidTAException***, η οποία καλείται όποτε υπάρξει πρόβλημα με κάποια κατηγορία.

7.1.1.7 Ρύθμιση Hibernate

Για να χρησιμοποιηθεί και να λειτουργήσει σωστά η Hibernate χρειάζεται τις κατάλληλες ρυθμίσης. Αυτές παρέχονται μέσω ενός XML αρχείου ρυθμίσεων, του ***hibernate.cfg.xml***. Ο κώδικας αυτού φαίνεται πιο κάτω:

Listing 7.3: hibernate.cfg.xml

```
1  <hibernate-configuration>
2    <session-factory>
3      <property name="connection.driver_class">
4          com.mysql.jdbc.Driver
5      </property>
6      <property name="connection.url">jdbc:mysql://
          localhost/tmt?characterEncoding=UTF-8</property>
7      <property name="connection.username">symmetry</
          property>
8      <property name="connection.password">pl@t1hma</
          property>
9      <property name="show_sql">false</property>
10     <property name="dialect">
11         org.hibernate.dialect.MySQL5InnoDBDialect
12     </property>
13     <property name="hbm2ddl.auto">create</property>
14     <property name="current_session_context_class">
15        thread</property>
16     <mapping resource="db/language/Language.hbm.xml"/>
17     <mapping resource="db/category/Category.hbm.xml"/>
18     <mapping resource="db/ta/TA.hbm.xml"/>
19     <mapping resource="db/record/Record.hbm.xml"/>
20   </session-factory>
21 </hibernate-configuration>
```

Μέσω του παραπάνω κώδικα ορίζεται η βάση δεδομένων στην οποία θα αποθηκευτούν τα δεδομένα. Το είδος της βάσης είναι MySql, το όνομα της είναι ***tmt***, ενώ είναι εγκατεστημένη στο ίδιο μηχάνιμα με αυτό που είναι εγκατεστημένο το Σύστημα Διαχείρισης Μεταφράσεων (localhost). Ακόμη, ορίζεται ο

χρήστης (όνομα και κωδικός) μέσω του οποίου η εφαρμογή έχει πρόσβαση στη βάση, καθώς και η διάλεκτος μέσω της οποίας γίνεται η επικοινωνία. Επιπλέον, ορίζεται το σχετικό μονοπάτι (path) στο οποίο είναι τοποθετημένα τα υπόλοιπα αρχεία ρυθμίσεων που ορίζουν τον τρόπο αντιστοίχισης των αντικειμένων στη βάση.

Τέλος, δημιουργήθηκε η κατάλληλη κλάση ***HibernateUtil*** (παρόμοια με αυτή που δημιουργήθηκε στην ενότητα 2.4.2.5). Μέσω αυτής γίνεται η δημιουργία και διαχείρηση ενός αντικειμένου τύπου *SessionFactory* της Hibernate. Την πρώτη φορά που αυτή καλλείται, διαβάζει ένα αρχείο, το *test/test_ sql.txt*, στο οποίο περιέχεται ο κατάλληλος SQL κώδικας για την αρχικοποίηση της βάσης, τον οποίο εκτελεί. Με αυτόν τον τρόπο αποθυκεύονται στη βάση στοιχεία για τις διάφορες γλώσσες, κατηγορίες και Εφαρμογές Χρήστες.

7.1.2 Λειτουργίες Μεταφραστή

Όλες οι λειτουργίες που μπορεί να εκτελέσει ο μεταφραστής υλοποιούνται μέσω ενός πακέτου, του ***operation*** (λειτουργία). Στο πακέτο αυτό ορίζεται η κλάση ***Operation***, η οποία συμβολίζει μία λειτουργία εισαγωγής ή εξαγωγής δεδομένων. Κρατάει πληροφορίες για το ποιος χρήστης κάνει την λειτουργία (αντικείμενο τύπου TmtUser), ποια γλώσσα, κατηγορία ή interface αφορά η λειτουργία., καθώς και το όνομα του αρχείου που εισάγεται ή εξάγεται. Επιπλέον διαθέτει ένα Set με τα Identifiers που παίρνουν μέρος στη λειτουργία.

7.1.2.1 Λειτουργία Εξαγωγής

Η λειτρουργία της εισαγωγής δημιουργήθηκε στο υποπακέτο *exportoperation*. Αυτό περιέχει μία κλάση, την ***ExportOperation***, η οποία είναι παιδί της κλάσης Operation και υλοποιεί το interface ***View*** (org.springframework.web.servlet.View) της Spring.

Ο δημιουργός της κλάσης ExportOperation παίρνει ως ορίσματα όλες τις απαραίτητες πληροφορίες σχετικά την επιθυμητή εξαγωγή. Αρχικά ελέγχει αν η επιθυμητή γλώσσα (ή γλώσσες αν τον αντικείμενο language είναι null) είναι ξεκλείδωτη, και αν ναι την κλειδώνει. Στην περίπτωση που κάποια γλώσσα είναι κλειδωμένη τότε η διαδικασία ματαιώνεται και γίνεται throw μίας ***Lan-***

guageLockedException. Αν η γλώσσα κλειδώση η Operation αρχικοποιήται κανονικά και η διαδικασία είναι έτοιμη να εκτελεστεί.

Για να εκτελεστεί η λειτουργία εξαγωγής πρέπει να καλεστεί η μέθοδος ***createFile(·)***, η οποία εκτελεί την μέθοδο ***createHeader(·)*** για την δημιουργία της κεφαλίδας ελέγχου (ενότητα 5.2.2) και στη συνέχεια, για κάθε προς εξαγωγή γλώσσα, τη μέθοδο ***writeLanguage(·)***. Για την εγγραφή των δεδομένων στην επιθυμητή CSV μορφή έγινε χρήση του ελεύθερου πακέτου ***opencsv***.

Η μέθοδος createFile(·) καλείται μέσω της μεθόδου ***render(·)***, την οποία υλοποιεί η κλάση λόγο του interface View. Η κλάση αυτή εκτελείται στην συνέχεια όταν μέσω της διεπαφής χρήστη θα δημιουργηθεί ένα αντικείμενο τύπου ExportOperation. Τότε θα σταλεί το δημιουργημένο αρχείο μέσω του *HttpServletResponse* που επιστρέφεται στον browser του χρήστη. Τα όσα αφορούν την διεπαφή χρήστη και την λειτουργεία της θα εξηγηθούν στη συνέχεια στην ενότητα +++++.

7.1.2.2 Λειτουργία Εισαγωγής

Η λειτρουργία της εισαγωγής δημιουργήθηκε στο υποπακέτο *importoperation*. Αυτό περιέχει μία κλάση, την ***ImportOperation***, η οποία είναι παιδί της κλάσης Operation.

Ο δημιουργός της κλάσης ImportOperation παίρνει ως ορίσματα όλες τις απαραίτητες πληροφορίες σχετικά την επιθυμητή εισαγωγή. Αρχικά ελέγχει το κλείδωμα της γλώσσας (ή όλων των γλωσσών, αν η μεταβλητή language έχει τιμή null). Αν η γλώσσα δεν είναι κλειδωμένη ή είναι κλειδωμένη από άλλο χρήστη γίνεται throw μίας εξαίρεσης τύπου ***LanguageLockedException*** με το κατάλληλο μήνυμα σφάλματος και η διαδικασία ματαιώνεται. Διαφορετικά, αρχικοποιείται κατάλληλα το στιγμιότυπο της κλάσης, και έτσι είναι έτοιμο να πραγματοποιήσει την διαδικασία εισαγωγής.

Για να πραγματοποιηθεί η εισαγωγή των δεδομένων μέσω του αρχείου πρέπει να καλεστεί η μέθοδος ***doImport(·)***. Αυτή, ελέγχει αρχικά την αξιοπιστία του αρχείου, και αν ο έλεγχος είναι επιτυχής καλείται η μέθοδος ***saveRecords(·)***. Μέσω αυτής διαβάζονται τα περιεχόμενά του αρχείου και οι

αλλαγές στις εγγραφές αποθηκεύονται στη βάση μέσω των αντίστοιχων διαχειριστών (RecordsManager). Αντίθετα, αν εντοπιστεί κάποιο πρόβλαμα στο εισαχθέν αρχείο η λειτουργία ματαιώνεται και γίνεται throw μια εξαίρεση τύπου ***ImportedFileIntegrityException*** με το κατάλληλο μύνημα σφάλματος.

Σημειώνεται ότι η διαχείριση του αρχείου που εισήχθηκε γίνεται μέσω της κλάσης ***FileHandler***. Αυτή διαθέτει μεθόδους για εξαγωγή της κεφαλίδας του αρχείου, των διαφορετικών Identifiers και Languages που περιέχονται σε αυτό, καθώς και οι διαφόρων εγγραφών μίας μίας κάθε φορά (*getNextRecord(·)*).

Ο τρόπος που αρχίζει η διαδικασία εισαγωγής των δεδομένων μέσω του αρχείου περιγράφεται στην ενότητα +++++.

7.1.2.3 Δημιουργία Αναφορών

Η λειτουργία της δημιουργάς των επιθυμητών αναφορών δημιουργήθηκε στο υποπακέτο *report*. Αυτό περιέχει μία abstract κλάση, την ***Report***, η οποία περιέχει ως μεταβλητές μέλη της στιγμιότυπα των κλάσεων διαχειριστών των αποθηκευόμενων οντοτήτων Language, Category και Record. Επίσης, αυτή διαθέτει μία abstract κλάση, την ***getReport(·)***, η οποία υλοποιείται στις υποκλάσεις, και είναι αυτή που δημιουργεί και επιστρέφει το κείμενο της αναφοράς.

Για τη δημιουργία της Changed αναφοράς δημιουργήθηκε η κλάση ***ChangedReport***, η οποία είναι υποκλάση της Report και παράλληλα υλοποιεί (implements) την κλάση *View* της Spring. Η ChangedReport υλοποιεί τη μέθοδο ***getReport(·)***, μέσω της οποίας δημιουργείται το κατάλληλο κείμενο που περιέχει η αναφορά. Η μέθοδος getReport(·) καλείται μέσω της μεθόδου ***render(·)***, την οποία υλοποιεί η κλάση λόγο του interface View. Η κλάση αυτή εκτελείται στην συνέχεια όταν μέσω της διεπαφής χρήστη θα δημιουργηθεί ένα αντικείμενο τύπου ChangedReport. Τότε θα σταλεί το δημιουργημένο κείμενο μέσω του *HttpServletResponse* που επιστρέφεται στον browser του χρήστη. Τα όσα αφορούν την διεπαφή χρήστη και την λειτουργεία της θα εξηγηθούν στη συνέχεια στην ενότητα +++++.

Η δημιουργία των Status αναφορών εκτελούνται μέσω της κλάσης ***StatusReport***, ακριβώς με τον ίδιο τρόπο που εκτελούνται οι Changed αναφορές.

7.1.3 Λειτουργίες Διαχειριστή

Όλες οι λειτουργίες που μπορεί να εκτελέσει ο διαχειριστής υλοποιούνται μέσω ενός πακέτου, του ***web.synchronization***. Στο πακέτο υπάρχουν υποπακέτα μέσω των οποίων υλοποιούνται όλες οι λειτουργίες συχρονισμού. Βέβαια, η ολοκληρωμένη παροχή των υπηρεσιών συγχρονισμού γίνεται μέσω των λειτουργιών που αφορούν την διεπαφή του χρήστη που αναλύεται στην ενότητα +++++.

7.1.3.1 Clean Συγχρονισμός

Ο clean συγχρονισμός (εξηγήθηκε στις ενότητας 5.2.4 και 6.2) υλοποιήθηκε στο υποπακέτο ***clean***. Στο πακέτο αυτό δημιουργήθηκε το interface ***CleanSynchronization*** μέσω του οποίου ορίζεται ο τρόπος επικοινωνίας με τον client μεταφράσεων. Η μέθοδος που ορίζεται από αυτό, η ***doCleanSynch(.)***, υλοποιείται στον client μεταφράσεων, ο οποίος παρέχει τις επιθυμητές web services. Μέσω αυτής στέλνεται στον client μεταφράσεων η γλώσσα την οποία αφορά ο συγκεκριμένος συχρονισμός και στη συνέχεια, αν όλα πάνε καλά, επιστρέφεται μία λίστα με τις επιθυμητές εγγραφές.

Για την πραγματοποίηση του συγχρονισμού δημιουργήθηκε η κλάση ***CleanSynchronize***, η οποία παρέχει όλες τις απαραίτητες μεθόδους για την πραγματοποίηση του συγχρονισμου (εκτός από την υπηρεσία εξουσιοδότησης, ο τρόπος υλοποίησης της οποίας περιγράφεται στην ενότητα +++++). Παρέχονται δύο μέθοδοι, η ***getRecordsFromMasterTA(·)*** και η ***saveCommonRecords(·)***. Η πρώτη, παίρνει ως όρισμα την γλώσσα που αφορά η εξαγωγή (είναι null αν ο συγχρονισμός αφορά όλες τις γλώσσες) και επιτρέφει τις εγγραφές της συγκεκριμένης γλώσσας που περιέχονται στην Εφαρμογή Χρήστη. Σημειώνεται ότι η μορφή των εγγραφών που επιτρέφονται δεν είναι η Record που περιγράφηκε στην ενότητα 7.1.1.4. Είναι τύπου ***communObjects.CommonRecord***, που είναι μία κοινή κλάση για τους server και client μεταφράσεων, έτσι ώστε να είναι εφηκτή η μεταξή τους επικοινωνία. Η μέθοδος ***saveCommonRecords(·)*** παίρνει ως όρισμα τη λίστα με τις εγγραφές που επιστράφηκαν από τον client μεταφράσεων, τα μετατρέπει κατάλληλα, και στη συνέχεια τα αποθηκεύει στη βάση δεδομένων.

Ο τρόπος με τον οποίο γίνεται η πρόσβαση και χρήση των υπηρεσιών που προσφέρονται μέσω των web services που παρέχονται από τον client μεταφράσεων περιγράφονται στην ενότητα +++++.

7.1.3.2 Delta Συγχρονισμός

Ο delta συγχρονισμός (εξηγήθηκε στις ενότητας 5.2.4 και 6.2) υλοποιήθηκε στο υποπακέτο ***delta***. Στο πακέτο αυτό δημιουργήθηκε το interface ***DeltaSynchronization*** μέσω του οποίου ορίζεται ο τρόπος επικοινωνίας με τον client μεταφράσεων. Οι μέθοδοι που υπάρχουν είναι οι ***getIdsNotInTmt(·)***, ***getRecordsWithIds(·)*** και ***updateChangedRecords(·)***.

Με τη μέθοδο getIdsNotInTmt(·), ο server μεταφράσεων στέλνει στον client μεταφράσεων μία λίστα με τα διαφορετικά Ids που περιέχονται στη βάση του και λαμβάνει από αυτόν μία λίστα με τα Ids που περιέχονται στην Εφαρμογή Χρήστη και όχι στην λίστα που εστάλει (βήματα 3 και 4 από την περιγραφή του Δέλτα συγχρονισμού που περιγράφηκε στην ενότητα 6.2).

Με τη μέθοδο ***getRecordsWithIds(·)*** στέλνεται η λίστα με τα Ids που έλαβε ο server μεταφράσεων από την προηγούμενη συνάρτηση και λαμβάνεται μία λίστα με τις εγγραφές (CommonRecord) που αντιστοιχούν σε αυτά τα Ids (βήματα 5 και 6 από την περιγραφή του Δέλτα συγχρονισμού).

Με τη μέδοδο ***updateChangedRecords(·)***, ο server μεταφράσεων στέλνει στον client μεταφράσεων μία λίστα με όλες τις εγγραφές που περιόχονται στη βάση του και έχουν την κατάσταση (status) CHANGED. Ο δεύτερος λαμβάνει τη λίστα και φροντίζει έτσι ώστε να ανανεωθεί κατάλληλα η βάση δεδομένων της Εφαρμογής Χρήστη (βήματα 7 και 8 από την περιγραφή του Δέλτα συγχρονισμού).

Για την πραγματοποίηση του συγχρονισμού από την πλευρά του server μεραφράσεων[1] δημιουργήθηκε η κλάση ***DeltaSynchronize***, η οποία παρέχει όλες τις απαραίτητες μεθόδους για την πραγματοποίηση του συγχρονισμου (εκτός από την υπηρεσία εξουσιοδότησης, ο τρόπος υλοποίησης της οποίας περιγράφεται στην ενότητα +++++). Παρέχονται δύο μέθοδοι, η ***deltaRead(·)*** και η ***deltaWrite(·)***. Η πρώτη κάνει τις απαραίτητες ενέργειες μόνο για την ενη-

[1] Η υλοποίηση της αντίστοιχης ενέργειας από την πλευρά του client μεταφράσεων υπάρχει στην ενότητα +++++

μέρωση της βάσης δεδομένων του Συστήματος Διαχείρισης Μεταφράσεων για τις αλλαγές στην βάση της Εφαρμογής Χρήστη, ενώ η δεύτερη για το αντίθετο (ενημέρωση της Εφαρμογής Χρήστη για τις αλλαγμένες εγγραφές).

Ο τρόπος με τον οποίο γίνεται η πρόσβαση και χρήση των υπηρεσιών που προσφέρονται μέσω των web services που παρέχονται από τον client μεταφράσεων περιγράφονται στην ενότητα +++++.

7.1.3.3 Export Συγχρονισμός

Ο export συγχρονισμός (εξηγήθηκε στις ενότητας 5.2.4 και 6.2) υλοποιήθηκε στο υποπακέτο ***export***. Στο πακέτο αυτό δημιουργήθηκε το interface ***ExportSynchronization*** μέσω του οποίου ορίζεται ο τρόπος επικοινωνίας με τον client μεταφράσεων. Η μέθοδος που ορίζει το interface είναι ***doExportSynch(·)***, η οποία ειδοποιεί τον client μεταφράσεων για τον προκείμενο συγχρονισμό.

Για την πραγματοποίηση του συγχρονισμού δημιουργήθηκε η κλάση *ExportSynchronize*. Αυτή έχει τη μέθοδο ***doExportSynch(·)*** η οποία ξεκινάει την διαδικασία χρησιμοποιώντας την web service που ορίστηκε μέσω του interface.

Στα πλαίσια της υλοποίησης του export συγχρονισμού ο server μεταφράσεων παρέχει μία web service. Έτσι ορίζεται ένα interface, το ***ExportSynchRecordManager***, το οποίο γνωρίζει και ο client μεταφράσεων για να καλεί τις κατάλληλες συναρτήσεις όποτε υπάρχει ανάγκη. Το interface αυτό υλοποιείται από την κλάση ***ExportSynchRecordManagerImp***. Η μέθοδος που υλοποιείται είναι η ***getRecords(·)*** μέσω της οποίας ο client μεταφράσεων ζητάει από τον server μεταφράσεων να του δόσει όλες τις εγγραφές μίας γλώσσας που περιέχει στην βάση του.

Εκτός αυτών, για να υλοποιηθεί η web service είναι απαραίτητες και κάποιες ρυθμίσεις μέσω της Spring. Αυτές παρέχονται στο αρχείο *WEB-INF/communicate-servlet.xml*. Ο κώδικας που αφορά την web service αυτή φαίνεται πιο κάτω.

Listing 7.4: WEB-INF/communicate-servlet.xml

```
1 <bean id="defaultHandlerMapping" class="org.
    springframework.web.servlet.handler.
```

```
    BeanNameUrlHandlerMapping"/>
2
3 <bean name="/getRecordsCleanSynch" class="org.
    springframework.remoting.httpinvoker.
    HttpInvokerServiceExporter">
4   <property name="service" ref="
      exportSynchRecordManagerImpl"/>
5   <property name="serviceInterface" value="web.
      synchronization.export.ExportSynchRecordManager"/>
6 </bean>
7
8 <bean id="exportSynchRecordManagerImpl" class="web.
    synchronization.export.ExportSynchRecordManagerImpl"/>
```

Με τον παραπάνω κώδικα ορίζεται το url στο οποίο θα ακούει η web service (/getRecordsCleanSynch), ο τρόπος επικοινωνίας (Spring HttpInvoker), το interface σύμφωνα με το οποίο θα γίνεται η επικοινωνία, καθώς και το ποια θα είναι η κλάση που θα υλοποιεί το interface αυτό.

Επιπλέον, για να είναι προσβάσιμο το web service θα πρέπει να οριστεί και ένα κατάλληλο servlet στο ***web.xml*** αρχείο ρυθμίσεων. Ο κώδικας αυτός παρουσιάζεται παρακάτω.

Listing 7.5: web.xml

```
1 <servlet>
2   <servlet-name>communicate</servlet-name>
3   <servlet-class>org.springframework.web.servlet.
      DispatcherServlet</servlet-class>
4   <load-on-startup>3</load-on-startup>
5 </servlet>
6 <servlet-mapping>
7   <servlet-name>communicate</servlet-name>
8   <url-pattern>/http/*</url-pattern>
9 </servlet-mapping>
```

Με τον κώδικα αυτό, ο HTTP web server γνωρίζει όταν έρθει μία αίτηση που να αρχίζει με το http, ότι πρέπει να την προωθήσει στο αντίστοιχο

servlet. Επιπλέον, λόγο του bean με defaultHandlerMapping που ορίστικε στο communicate-servlet.xml, η αίτηση προωθείται στην κλάση που αντιστοιχεί στο bean, το όνομα του οποίου έχει την ίδια κατάλληξη με την αίτηση που ελήφθει.

Ο τρόπος με τον οποίο γίνεται η πρόσβαση και χρήση των υπηρεσιών που προσφέρονται μέσω των web services που παρέχονται από τον client μεταφράσεων, καθώς και το πως χρησιποιείται η web service που παρέχει ο server μεταφράσεων, περιγράφονται στην ενότητα +++++.

7.1.4 Λειτουργία Διεπαφής Χρήστη

Για την εμφάνιση και λειτουργία της διεπαφής χρήστη, έγινε χρήση των εργαλείων που προσφέρει η Spring. Αρχικά δημιουργήθηκε το web.xml το οποίο ενημερώνει τον HTTP web server για το που θα πρέπει να προωθεί τα διάφορα http requests που θα λαμβάνει. Τα περιεχόμενα του αρχείου αυτού φαίνονται παρακάτω στο κώδικα 7.6.

Listing 7.6: web.xml

```
1  <web-app version="2.4"
2           xmlns="http://java.sun.com/xml/ns/j2ee"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-
               instance"
4           xsi:schemaLocation="http://java.sun.com/xml/ns/
               j2ee
5           http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
                >
6  <context-param>
7    <param-name>contextConfigLocation</param-name>
8    <param-value>
9      /WEB-INF/tmtWeb-servlet.xml
10   </param-value>
11 </context-param>
12 <servlet>
13   <servlet-name>tmtWeb</servlet-name>
```

```
14    <servlet-class>org.springframework.web.servlet.
          DispatcherServlet</servlet-class>
15    <load-on-startup>1</load-on-startup>
16  </servlet>
17  <servlet-mapping>
18    <servlet-name>tmtWeb</servlet-name>
19    <url-pattern>*.html</url-pattern>
20  </servlet-mapping>
21  <servlet>
22    <servlet-name>communicate</servlet-name>
23    <servlet-class>org.springframework.web.servlet.
          DispatcherServlet</servlet-class>
24    <load-on-startup>3</load-on-startup>
25  </servlet>
26  <servlet-mapping>
27    <servlet-name>communicate</servlet-name>
28    <url-pattern>/http/*</url-pattern>
29  </servlet-mapping>
30  </web-app>
```

Το αρχείο αυτό ενημερώνει τον HTTP web server για την ύπαρξη δύο servlets. Το ένα, το ***tmtWeb***, είναι αυτό το οποίο αφορούν οι αιτήσεις που έχουν να κάνουν με τη διεπαφή χρήστη. Για αυτό το servlet ορίζεται το αρχείο ρυθμίσεων που το αφορά, η κατάλληξη των αιτήσεων που το αφορούν, καθώς και ποια είναι η κλάση που το διαχειρίζεται (ορίζεται η DispatcherServlet της Spring). Το δεύτερο servlet, το ***communicate*** αφορά την παροχή μίας web service από τον server μεταφράσεων και εξηγήθηκε στην ενότητα 7.1.3.3.

Στη συνέχεια, στον κώδικα 7.7, παρουσιάζεται ο κώδικας του αρχείου *tmtWeb-servlet.xml*, μέσω του οποίου γίνονται όλες οι ρυθμίσεις που αφορούν την διεπαφή χρήστη.

Listing 7.7: web.xml

```
1  <beans xmlns="http://www.springframework.org/schema/beans
       "
```

```
2         xmlns:xsi="http://www.w3.org/2001/XMLSchema-
              instance"
3         xsi:schemaLocation="http://www.springframework.org
              /schema/beans
4         http://www.springframework.org/schema/beans/spring
              -beans-2.5.xsd">
5     <bean id="sessionFactory" class="org.springframework.
          orm.hibernate3.LocalSessionFactoryBean">
6       <property name="mappingResources">
7         <list>
8           <value>hibernate.cfg.xml</value>
9           <value>db/category/Category.hbm.xml</value>
10          <value>db/language/Language.hbm.xml</value>
11          <value>db/record/Record.hbm.xml</value>
12          <value>db/ta/TA.hbm.xml</value>
13        </list>
14      </property>
15      <property name="hibernateProperties">
16        <props>
17         <prop key="hibernate.dialect">
18          org.hibernate.dialect.MySQL5InnoDBDialect
19          </prop>
20        </props>
21      </property>
22    </bean>
23    <bean id="languageDao" class="db.language.
          LanguagesManager">
24      <property name="sessionFactory"><ref local="
            sessionFactory"/></property>
25    </bean>
26    <bean id="categoryDao" class="db.category.
          CategoriesManager">
27      <property name="sessionFactory"><ref local="
            sessionFactory"/> </property>
28    </bean>
```

```
29   <bean id="recordDao" class="db.record.RecordsManager">
30     <property name="sessionFactory"><ref local="
          sessionFactory"/> </property>
31   </bean>
32   <bean id="taDao" class="db.ta.TAManager">
33     <property name="sessionFactory"><ref local="
          sessionFactory"/></property>
34   </bean>
35   <bean id="publicUrlMapping" class="org.springframework.
        web.servlet.handler.SimpleUrlHandlerMapping">
36     <property name="mappings">
37       <props>
38         <prop key="/tmt.html">tmtFormController</prop>
39         <prop key="/index.html">
             authenticationFormController</prop>
40       </props>
41     </property>
42   </bean>
43   <bean id="viewResolver" class="org.springframework.web.
        servlet.view.InternalResourceViewResolver">
44     <property name="viewClass" value="org.springframework
          .web.servlet.view.JstlView"/>
45     <property name="prefix" value="/WEB-INF/jsp/"/>
46     <property name="suffix" value=".jsp"/>
47   </bean>
48   <bean id="tmtFormController" class="web.
        TmtFormController">
49     <property name="languageDao" ref="languageDao"/>
50     <property name="categoryDao" ref="categoryDao"/>
51     <property name="TADao" ref="taDao"/>
52     <property name="sessionForm" value="true"/>
53     <property name="successView" value="tmt.html"/>
54     <property name="commandName" value="tmtForm"/>
55     <property name="commandClass" value="web.
          TmtFormCommand"/>
```

```
56    </bean>
57    <bean id="authenticationFormController" class="web.
         AuthenticationFormController">
58      <property name="sessionForm" value="true"/>
59      <property name="commandName" value="authenticate"/>
60      <property name="successView" value="tmt.html"/>
61      <property name="commandClass" value="web.authenticate
           .AuthenticateCommand"/>
62    </bean>
63    <bean id="multipartResolver" class="org.springframework
         .web.multipart.commons.CommonsMultipartResolver">
64      <property name="maxUploadSize" value="1000000"/>
65      <property name="defaultEncoding" value="UTF-8"/>
66    </bean>
67    <bean id="operationStatus" class="web.OperationStatus"
         factory-method="getInstance"/>
68    <bean id="synchStatus" class="web.SynchStatus" factory-
         method="getInstance"/>
69  </beans>
```

Όπως προειπώθηκε ο παραπάνω κώδικας ορίζει όλες τις απαραίτητες ρυθμίσεις για την εμφάνιση και χρήση της διεπαφής χρήστη. Στη συνέχεια περιγράφονται τα σημαντικότερα στοιχεία αυτού.

Αρχικά, με το bean ***sessionFactory*** (γραμμή 5) γίνονται όλες οι απαραίτητες ρυθμίσεις που έχουν να κάνουν με τη βάση δεδομένων. Ορίζεται στη Spring ότι θα υπάρξει σύνδεση με βάση δεδομένων και ότι αυτήν θα αναλλάβει η Hibernate. Ταυτόχρονα, ορίζονται όλα τα αρχεία στα οποία υπάρχουν ρυθμίσεις που αφορούν την σύνδεδη με τη βάση.

Τα beans που έχουν την κατάλληξη ***Dao*** υπάρχουν για να ορίσουν στη Spring ότι υπάρχουν οι κλάσεις διαχειριστές των αποθηκευόμενων κλάσεων που αντιστοιχίζονται μέσω της Hibernate σε πίνακες στη βάση δεδομένων, έτσι ώστε να είναι εφικτή η διαχείρισή τους.

Με το bean ***publicUrlMapping*** (γραμμή 35) ορίζεται στη Spring σε ποιο bean θα προωθήσει την αίτηση όταν θα υπάρξει κάποια αίτηση για ένα συγκεκριμένο url. Έτσι αντιστοιχίζεται το /tmt.html στο tmtFormController και

το /index.html στο authenticationFormController.

Με το bean ***viewResolver*** (γραμμή 43) ορίζεται στη Spring ότι για οποιαδήποτε αίτηση της έρχεται θα πρέπει να ψάχνει για το αντίστοιχο αρχείο στον φάκελο *WEB-INF/jsp/*. Το αρχείο όμως που θα βρίσκει στον φάκελο θα έχει στην πραγματικότητα την κατάληξη ***.jsp*** αντί για αυτή που έχει στο request που ελήφθει. Έτσι για παράδειγμα αν έρθει μία αίτηση ζητώντας το αρχείο tmt.html, η Spring θα ψάξει για το αρχείο WEB-INF/jsp/tmt.jsp.

Το bean ***tmtFormController*** (γραμμή 48) είναι αυτό που περιέχει τις ρυθμίσεις που αφορούν την αίτηση για το url tmt.html. Αυτό ορίζει την κλάση που διαχειρίζεται τις αιτήσεις για αυτό το url, ενώ επίσης δίνει στην κλάση της εξαρτήσεις της (languageDao, recordDao, taDao). Μέσω του url tmt.html είναι προσβάσιμη η διεπαφή του χρήστη (είτε του μεταφραστή, είτε του διαχειριστή). Έτσι, μιας και η διεπαφή περιέχει φόρμα (τα διάφορα στοιχεία που επιλέγει ο χρήστης εξάγονται μέσω html φόρμας), το bean αυτό επίσης ορίζει ποια θα είναι η κλάση που θα αντιστοιχιστεί με τη φόρμα αυτή καθως και ποιο είναι το όνομά της.

Το bean ***multipartResolver*** (γραμμή 57) είναι αυτό που περιέχει τις ρυθμίσεις που αφορούν την αίτηση για το url index.html. Το url αυτό χρησιμοποιήται για την εισαγωγή από τον χρήστη των στοιχείων του (ψευδώνυμο, κωδικό) έτσι ώστε να γίνει η διαδικασία αναγνώρισης και εξουσιοδότησης, και στην συνέχεια να επιτραπεί στον χρήστη η είσοδός του στο σύστημα. Το bean αυτό ορίζει την κλάση που αντιστοιχεί στην φόρμα που περιέχει η διεπαφή καθώς και το όνομας της φόρμας αυτής.

7.2 Client Μεταφράσεων

Κεφάλαιο 8

Συμπεράσματα

8.1 Προοπτικές