ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟΥΠΟΛΗ ΙΩΑΝΝΙΝΩΝ, ΙΩΑΝΝΙΝΑ



Π.Ε.-2013-4 25 Ιουνίου 2013

Οπτικοποίηση της Εξέλιξης Σχήματος Βάσεων Δεδομένων

Κολοζώφ Μιχαήλ - Ρωμανός

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: Π. Βασιλειάδης

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Πρόλογος

Μια βάση δεδομένων, από τη στιγμή που θα δημιουργηθεί, αλλάζει εσωτερική δομή με το πέρασμα του χρόνου. Με άλλα λόγια το σχήμα της βάσης δε μένει σταθερό αλλά εξελίσεται συνεχώς. Πίνακες και πεδία προστίθενται, διαγράφονται, μετονομάζονται, κλπ. Σκοπός αυτής της πτυχιακής εργασίας είναι η δημιουργία ενός εργαλείου για την οπτικοποίηση της εξέλιξης του σχήματος βάσεων δεδομένων στο χρόνο, δηλαδή ανά διαφορετικές εκδόσεις. Το εργαλείο αυτό έχει σκοπό να βοηθήσει στην εύκολη μελέτη της της εξέλιξης αυτής αφού είναι πολύ εύκολο να βγάλει κανείς συμπεράσματα για το πως εξελίχθηκε η βάση στον χρόνο και/ή τις σχέσεις τών πινάκων μεταξύ τους, με απλή οπτική παρατήρηση γραφημάτων.

Με το πέρας αυτής της πτυχιακής εργασίας θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή μου κ. Παναγιώτη Βασιλειάδη για την βοήθεια και τον χρόνο που μου αφιέρωσε καθ'όλη την διάρκεια της χρονιάς, καθώς και για το κουράγιο που μου έδινε. Επίσης θα ήθελα να δώσω ένα μεγάλο ευχαριστώ στους μεταπτυχιακούς φοιτητές Ιωάννη Σκουλή και Δημήτρη Γκεσούλη για την στήριξη και καθοδήγηση που μου πρόσφεραν.Τέλος θα ήθελα να ευχαριστήσω την οικογένεια μου και όλους όσους με βοήθησαν νοερά για να ολοκληρώσω αυτην την εργασία.

Ιούνιος 2013

Κολοζώφ Μιχαήλ - Ρωμανός

Περίληψη στα ελληνικά

Το σχήμα μιας βάσης δεδομένων απο την στιγμή της δημιουργίας της μέχρι και το τέλος της αλλάζει συνεχώς. Πίνακες προστίθενται και αφαιρούνται. Ακομα και οι ίδιοι οι πίνακες δεν παραμένουν σταθεροί αλλά τα πεδία τους αφαιρούνται, νέα προστίθενται ή ακόμα αλλάζουν τύπο. Σκοπός αυτής της πτυχιακής εργασίας είναι η ανάπτυξη ενός εργαλείου για την οπτικοποίηση αυτών των αλλαγών ούτως ώστε να γίνονται εύκολα αντιληπτές απο τον χρήστη. Για την υλοποίηση του εργαλείου αυτου ακολουθήσαμε τα εξής βήματα:

- Θα λάβουμε ως είσοδο το βίο της βάσης δεδομένων και τις αλλαγές που συντελούνται σε κάθε μετάβαση
- Θα εντοπίσουμε όλους τους πίνακες που συμμετέχουν στη ζωή μιας βάσης δεδομένων, σε όλες τις εκδόσεις τους. Επιπλέον, θα εντοπίσουμε και τα ξένα κλειδιά που τους συνδέουν
- Θα δημιουργήσουμε τον διαχρονικό γράφο της βάσης, δηλαδή ένα γράφημα με την ένωση των πινάκων και των ξένων-κλειδιών τους,
- Θα τοποθετήσουμε σε μια διδιάσταση επιφάνεια τον διαχρονικό γράφο, δίνοντάς του συντεταγμένες
- Για κάθε μία από τις versions του πίνακα, θα ζωγραφήσουμε την εικόνα της, με βάση τις παραπάνω συντεταγμένες
- Θα τοποθετήσουμε τις εικόνες σε μια παρουσίαση Microsoft PowerPoint
- Τέλος, θα δημιουργήσουμε αρχεία τύπου wmv και mp4 της παραπάνω παρουσίασης

Λέξεις Κλειδιά: Βάση Δεδομένων, Σχήμα Βάσης δεδομένων, Εξέλιξη Σχήματος Βάσεων Δεδομένων, Οπτικοποίηση, Γραφήματα, Jung, Xuggler, Apache-POI

Περίληψη στα αγγλικά

A database's schema is constantly evolving, from its creation until its end. Tables are added and removed. Even these tables are not staying unchangeable since they have their fields added or removed or even their type changed. The ultimate goal of this Diploma thesis is the development of a tool capable of visualizing the aforementioned changes in a way that they can be easily perceived by the user. Our method is outlined as the following sequence of steps:

- We use as input a database's lifetime as well as the changes that contribute in each transitions
- We locate every table that takes part in a database's lifetime in all its versions. Additionally, we locate the foreign keys that link them
- We create the diachronic graph of the database, which is a graph that contains the union of the database's tables and its foreign keys
- We place the diachronic graph in a two dimensional surface, by doing so each table gets fixed coordinates
- For each version of a table we depict the subgraph that corresponds to it, retaining the coordinates of the nodes as previously computed
- We put together those images in a PowerPoint Presentation
- Finally, we convert the PowerPoint Presentation to wmv and mp4 files

Keywords: Database, Database's Schema, Database's Schema Evolution, Visualization, Graphs, Jung, Xuggler, Apache-POI

Πίνακας περιεχομένων

1 Εισαγωγή 1

1.1 Αντικείμενο της πτυχιακής 1

1.2 Οργάνωση του τόμου 2

2 Περιγραφή Θέματος 3

2.1 Υπόβαθρο 3

2.2 Στόχος 4

3 Ανάλυση και Σχεδίαση 7

3.1 Στατική Περιγραφή 8

4 Υλοποίηση 13

4.1 Πλατφόρμες και προγραμματιστικά εργαλεία 13

4.2 Λεπτομέρειες υλοποίησης 13

5 Οδηγός Χρήσης 38

5.1 Αναλυτική παρουσίαση 38

6 Επίλογος 44

6.1 Σύνοψη και συμπεράσματα 44

6.2 Μελλοντικές επεκτάσεις 45

7 Βιβλιογραφία 46

1 Εισαγωγή

Όπως περιγράφεται στην προηγούμενη ενότητα, το σχήμα μιας βάσης δεδομένων εξελίσσεται στο χρόνο: πίνακες και πεδία προστίθενται, διαγράφονται, μετονομάζονται, κλπ. Μπορούμε να φτιάξουμε μια ταινία με την εξέλιξη αυτή?

1.1 Αντικείμενο της πτυχιακής

Σκοπός αυτής της πτυχιακής εργασίας είναι η οπτικοποίηση της εξέλιξης του σχήματος μια βάσης δεδομένων στον χρόνο με τρόπο που να είναι εύκολα αντιληπτός στο χρήστη. Με τον όρο **βάση δεδομένων** εννοείται μία συλλογή από συστηματικά οργανωμένα σχετιζόμενα δεδομένα στα οποία είναι δυνατή η ανάκτηση δεδομένων. Το **σχήμα μιας βάσης δεδομένων** αποτελεί την δομή, που περιγράφεται σε μια επίσιμη γλώσσα που υποστηρίζεται απο το σύστημα διαχείρισης βάσεων δεδομένων, και αναφέρεται στην οργάνωση των δεδομένων στο εσωτερικό της βάσης. Με τον όρο **οπτικοποίηση της εξέλιξης του σχήματος μια βάσης δεδομένων στον χρόνο** εννοούμε την δημιουργία γραφημάτων για κάθε έκδοση της βάσης και την κωδικοποίηση αυτών σε μια "ταινία" δηλαδή την δημιουργία μια παρουσίασης Microsoft PowerPoint και αρχεία τύπου mp4.

1.2 Οργάνωση του τόμου

Η ανάλυση της σχεδίασης του προαναφερθέντος εργαλείου καθώς και τα συμπεράσματα που προκύπτουν περιγράφονται λεπτομερώς στα παρακάτω κεφάλαια:

Κεφάλαιο 2: Στην ενότητα αυτή περιγράφεται το υπόβαθρο από τις διαθέσιμες τεχνολογίες που χρησιμοποιήθηκαν για την ολοκλήρωση της πτυχιακής.

Κεφάλαιο 3: Στην ενότητα αυτή περιγράφονται τα components της αρχιτεκτονικής του εργαλείου μας, και δείχνεται ο τρόπος που συνεργάζονται μεταξύ τους. Για την πλήρη κατανόηση απο τον αναγνώστη παρέχεται το αναλυτικό μοντέλο UML των κλάσεων.

Κεφάλαιο 4: Στην ενότητα αυτή περιγράφονται τα χαρακτηριστικά της συγκεκριμένης υλοποίησης, όπως η πλατφόρμα ανάπτυξης και εκτέλεσης, τα προγραμματιστικά εργαλεία, και οι απαιτήσεις της εφαρμογής σε hardware.

Κεφάλαιο 5: Στην ενότητα αυτή γίνεται η αναλυτική παρουσίαση του εργαλείου. Ουσιαστικά αποτελεί ένα σαφές εγχειρίδιο χρήσης.

Κεφάλαιο 6: Στην ενότητα αυτή συνοψίζουμε τα αποτελέσματα τις πτυχιακής εργασίας και περιγράφουμε τα όποια συμπεράσματα εξάγαμε. Επίσης παρέχουμε μια σειρά ιδεών που θα μπορούσαν να πραγματοποιηθούν στο μέλλον.

2 Περιγραφή Θέματος

2.1 Υπόβαθρο

Σαν βάση για την ολοκλήρωση της πτυχιακής χρησιμοποιήθηκαν τέσσερις βασικές τεχνολογίες.

- Η πτυχιακή εργασία του Ιωάννη Σκουλή, Εκάτη, η οποία αφορά την οπτικοποίηση διαφορών μεταξύ δύο σχημάτων SQL. Το σύστημά μας είναι στενά συνδεδεμένο με την Εκάτη, για την ακρίβεια οι κλάσεις της Εκάτης χρησιμοποιούνται απο το σύστημα κατά την φόρτωση του φακέλου εισόδου για την μεταφορά της πληροφορίας απο τον δίσκο στην κύρια μνήμη καθώς και για την δημιουργία του αρχείου μεταβάσεων transitions.xml. Κατα την φόρτωση του φακέλου εισόδου, οι κλάσεις της Εκάτης είναι υπεύθυνες για την ανάγνωση των αρχείων SQL, τα οποία είναι η ακολουθία των εκδόσεων του σχήματος της βάσης, και την αποθήκευση της πληροφόριας, δηλαδή των πινάκων που υπάρχουν ανα έκδοση και τις συσχετίσεις ξένων κλειδιών μεταξύ τους, σε κατάλληλες δομές δεδομένων στην μνήμη του προγράμματος. Το αρχείο μεταβάσεων είναι ένα αρχείο xml που για δυο εκδόσεις της βάσης περιέχει πληροφορίες σχετικά με ποιοί πίνακες διαγράφηκαν ποιοί δημιουργήθηκαν και ποιοί ανανεώθηκαν, παρακάτω γίνεται αναλυτική επεξήγηση για την δημιουργία και χρήση του αρχείου.
- Το API της Jung το οποίο είναι υπεύθυνο για την οπτικοποίηση και επεξεργασία των γραφημάτων. Η Jung είναι μια βιβλιοθήκη λογισμικού που παρέχει μια κοινή και επεκτάσιμη γλώσσα για τη μοντελοποίηση, ανάλυση και οπτικοποίηση των δεδομένων που μπορεί να παρασταθεί ως ένα γράφημα ή δίκτυο. Είναι γραμμένο σε Java, το οποίο επιτρέπει εφαρμογές βασισμένες σε Jung να κάνουν χρήση των

εκτεταμένων ενσωματωμένων δυνατοτήτων της Java, καθώς και άλλων υφιστάμενων τρίτων βιβλιοθηκών της. Όταν ο χρήστης επιλέξει έναν φάκελο εισόδου το σύστημά μας, αρχικά πρέπει να οπτικοποιήσει τον *Διαχρονικό γράφο* της βάσης στον χρήστη. Όπως είπαμε και πιο πάνω ο *Διαχρονικός γράφος* περιέχει την ένωση των πινάκων που υπάρχουν στην ακολουθία εκδόσεων, διάρκεια ζωής, της βάσης καθώς και την ένωση των συσχετίσεων ξένων κλειδιών μεταξύ τους. Αφού οπτικοποιηθεί ο *Διαχρονικός γράφος*, ο χρήστης μπορεί να τον διαμορφώσει ανάλογα με τις προτιμήσεις του. Η οπτικοποίηση και επεξεργασία του γραφήματος επιτυγχάνεται με την χρήση κατάλληλα διαμορφομένων κλάσεων που περιλαμβάνονται στο API της Jung και αναλύονται εκτενώς παρακάτω.

- Το API της Apache-POI το οποίο είναι υπεύθυνο για την δημιουργία και την μορφοποίηση παρουσιάσεων PowerPoint. Όπως είπαμε παραπάνω ο χρήστης μπορεί να επεξεργαστεί τον *διαχρονικό γράφο* της βάσης, όταν τελείωσει μπορεί να αναπαράξει ένα αρχείο εικόνας για κάθε έκδοση της βάσης. Στόχος του συστήματός μας είναι η δημιουργία μίας παρουσίασης PowerPoint με τις εικόνες που παράχτηκαν για την ακολουθία των εκδόσεων της βάσης. Η Apache-POI μας παρέχει ένα πλήρες API για την δημιουργία εγγράφων της Microsoft, κατ' επεκταση και για την δημιουργία παρουσιάσεων PowerPoint, ο τρόπος χρήσης του API καθώς και οι λεπτομέρειες υλοποίησης περιγράφονται αναλυτικά παρακάτω.
- Τέλος, Το API της Xuggler το οποίο είναι υπεύθυνο για την κωδικοποίηση αρχείων εικόνας σε αρχεία mp4. Το σύστημά μας, δίνει την δυνατότητα στον χρήστη της δημιουργίας αρχείων mp4 δοθέντος ώς είσοδο μια παρουσίαση PowerPoint. Αρχικά εξάγουμε απο την παρουσίαση κάθε διαφάνεια της με την χρήση μια εφαρμογής που παρέχεται απο το προαναφερθέν API της Apache-POI και στην συνέχεια κωδικοποιούμε τις εικόνες αυτές σε αρχείο video. Το παρόν API περιέχει κατάλληλα διαμορφωμένες κλασεις για την εύκολη κωδικοποιήση σε αρχεία mp4. Στην συνέχεια ακουλουθεί αναλυτική περιγραφή της χρήσης της Xuggler για την υλοποίηση των άνω.

2.2 Στόχος

Το σχήμα μιας βάσης δεδομένων απο την στιγμή της δημιουργίας της μέχρι και το τέλος της αλλάζει συνεχώς. Πίνακες προστίθενται και αφαιρούνται. Ακομα και οι ίδιοι οι πίνακες δεν παραμένουν σταθεροί αλλά τα πεδία τους αφαιρούνται, νέα προστίθενται ή ακόμα αλλάζουν

τύπο. Σκοπός αυτής της πτυχιακής εργασίας είναι η ανάπτυξη ενός εργαλείου για την οπτικοποίηση αυτών των αλλαγών ούτως ώστε να γίνονται εύκολα αντιληπτές απο τον χρήστη. Ως είσοδο για το σύστημά μας έχουμε:

- Μία ακολουθία αρχείων SQL που περιγράφουν το σχήμα της βάσης ανα έκδοση
- Ένα αρχείο μεταβάσεων σε μορφή xml το οποίο περιέχει τις αλλαγές που συμβαίνουν στην βάση, στους πίνακές της και/ή στις συσχετίσεις ξένων κλειδιών μεταξύ τους, σε κάθε έκδοση.
- Ένα αρχείο αποθήκευσης δεδομένων σχετικά με τις συντεταγμένες των κόμβων στο επίπεδο. Η ύπαρξη του αρχείου αυτού δεν είναι δεσμευτική, σίγουρα δεν υπάρχει την πρώτη φορά που ο χρήστης επιλέγει τον φάκελο εισόδου, και/ή ο χρήστης μπορεί να επιλέξει να μην αποθηκεύσει τις αλλαγές που έχει πραγματοποιήσει.

Το σύστημά μας έχει ώς στόχο την παραγωγή των εξής αρχείων εξόδου:

- Ένα αρχείο εικόνας για κάθε αρχείο της ακολουθίας αρχείων SQL που περιγράφουν το σχήμα της βάσης. Το αρχείο εικόνας περιέχει κατάλληλη επισήμανση για τους πίνακες της εκάστοτε έκδοσης ανάλογα με την πληροφορία που έχει αντληθεί απο το αρχείο μεταβάσεων xml. Η ανάλυση της παραγωγής αυτών των αρχείων περιγράφεται παρακάτω.
- Ένα αρχείο παρουσίασης PowerPoint. Το σύστημά μας, μπορεί να μετατρέψει το παραπάνω σύνολο αρχείων εικόνας σε μια δομημένη παρουσίαση PowerPoint, με την χρήση της εξωτερικής βιβλιοθήκης Apache-POI. Όπου φαίνεται αρχικά ο διαχρονικός γράφος της βάσης, και στην συνέχεια η εξέλιξη της διάρκειας ζωής του σχήματος της βάσης, απο την αρχή μέχρι το τέλος της.
- Ένα αρχείο video. Το σύστημά μας, μπορεί να μετατρέψει την παραπάνω παρουσίαση PowerPoint σε αρχείο mp4 με την βοήθεια της εξωτερικής βιβλιοθήκης Xuggler. Αρχικά διασπά την κάθε διαφάνεια της παρουσίασης σε εικόνες και στην συνέχεια κωδικοποιεί αυτές τις εικόνες σε video.
- Ένα αρχείο graphml. Το σύστημά μας μπορεί να αποθηκέυσει τις αλλαγές που επιφέρει ο χρήστης στο *Διαχρονικό γράφο* που προβάλλεται σε μορφή graphml. Η μορφή graphml είναι ένα πρότυπο αποθήκευσης το οποίο είναι βασισμένο σε XML μορφή αρχείου και χρησιμοποιείται ευρέως για γραφήματα.

- Τέλος, ένα αρχείο μεταβάσεων σε μορφή xml. Το αρχείο μεταβάσεων, που αναφέρθηκε παραπάνω και είναι πολύ χρήσιμο για την οπτικοποίηση της εξέλιξης του σχήματος μια βάσης δεδομένων, μπορεί να παραχθεί απο το σύστημά μας χρησιμοποιώντας την Εκάτη.

3 Ανάλυση και Σχεδίαση

Σε αυτήν την ενότητα περιγράφεται αναλυτικά η σχεδίαση του συστήματος. Γίνεται λεπτομερής περιγραφή των components της αρχιτεκτονικής του εργαλείου και του τρόπου που συνεργάζονται και αλληλοεξαρτώνται. Για την πλήρη κατανόηση αυτής της περιγραφής παρέχεται και το μοντέλο UML του συστήματος. Ακολουθούν οι περιπτώσεις χρήσης του εργαλείου.

- ✓ Ο χρήστης μπορεί να επιλέξει έναν φάκελο εισόδου. Ένας φάκελος εισόδου πρέπει να περιέχει μια ακολουθία αρχείων SQL, τα οποία περιγράφουν το σχήμα της βάσης. Ένα xml αρχείο μεταβάσεων, το οποίο περιγράφει τις διαφορές που έγιναν στην βάση ανάμεσα σε δύο εκδόσεις. Τέλος μπορεί ή όχι να περιέχει ένα αρχείο graphml με τις αλλαγές που έχει επιφέρει ο χρήστης στην επεξεργασία του γραφήματος. Όταν το σύστημα φορτώσει τον φάκελο εισόδου προβάλει το σύνολο των πινάκων που υπάρχουν στην διάρκεια ζωής της βάσης καθώς και τις εξαρτήσεις ξένων κλειδιών μεταξύ τους και δίνει την δυνατότητα επεξεργασίας του γραφήματος στον χρήστη.
- ✓ Αν ο φάκελος εισόδου δεν περιέχει το αρχείο μεταβάσεων ο χρήστης μπορεί να το δημιουργήσει απο το σύστημά μας.
- ✓ Κατα την επεξεργασία του γραφήματος ο χρήστης μπορεί να μετατοπίσει έναν κόμβο, ή και όλο το γράφημα.
- ✓ Μετά το τέλος της επεξεργασίας του γραφήματος ο χρήστης μπορεί να αποθηκεύσει τις αλλαγές που έχει πραγματοποιήσει. Μια τέτοια ενέργεια δημιουργεί το αρχείο αποθήκευσης graphml που αναφέραμε παραπάνω
- ✓ Ο χρήστης μπορεί να παράξει τα αρχεία εικόνας για κάθε έκδοση της βάσης
- ✓ Ο χρήστης μπορεί να δημιουργήσει μια παρουσίαση PowerPoint με την εξέλιξη της του σχήματος της βάσης χρησιμοποιώντας τις εικόνες που δημιούργησε πριν.

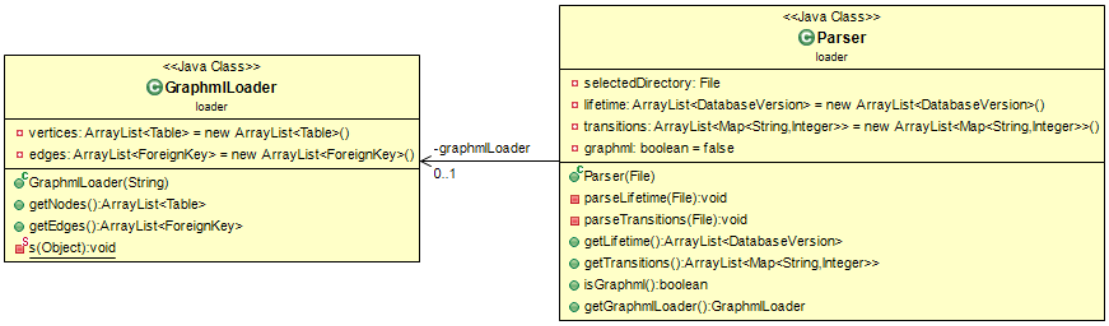
- ✓ Ο χρήστης έχει την δυνατότητα μετατροπής της παρουσίασης PowerPoint σε αρχείο mp4 και wmv

3.1 Στατική Περιγραφή

Για την υλοποίηση συστήματος [Parmenidian Truth] μας ορίσαμε ένα πλήθος κλάσεων τις οποίες οργανώσαμε κατάλληλα σε πακέτα του συστήματος. Στην συνέχεια θα αναλύσουμε κάθε πακέτο, τις κλάσεις που περιέχει, καθώς και τον τρόπο που συνεργάζονται μεταξύ τους για την πραγματοποίηση των προαναφερθέντων λειτουργιών.

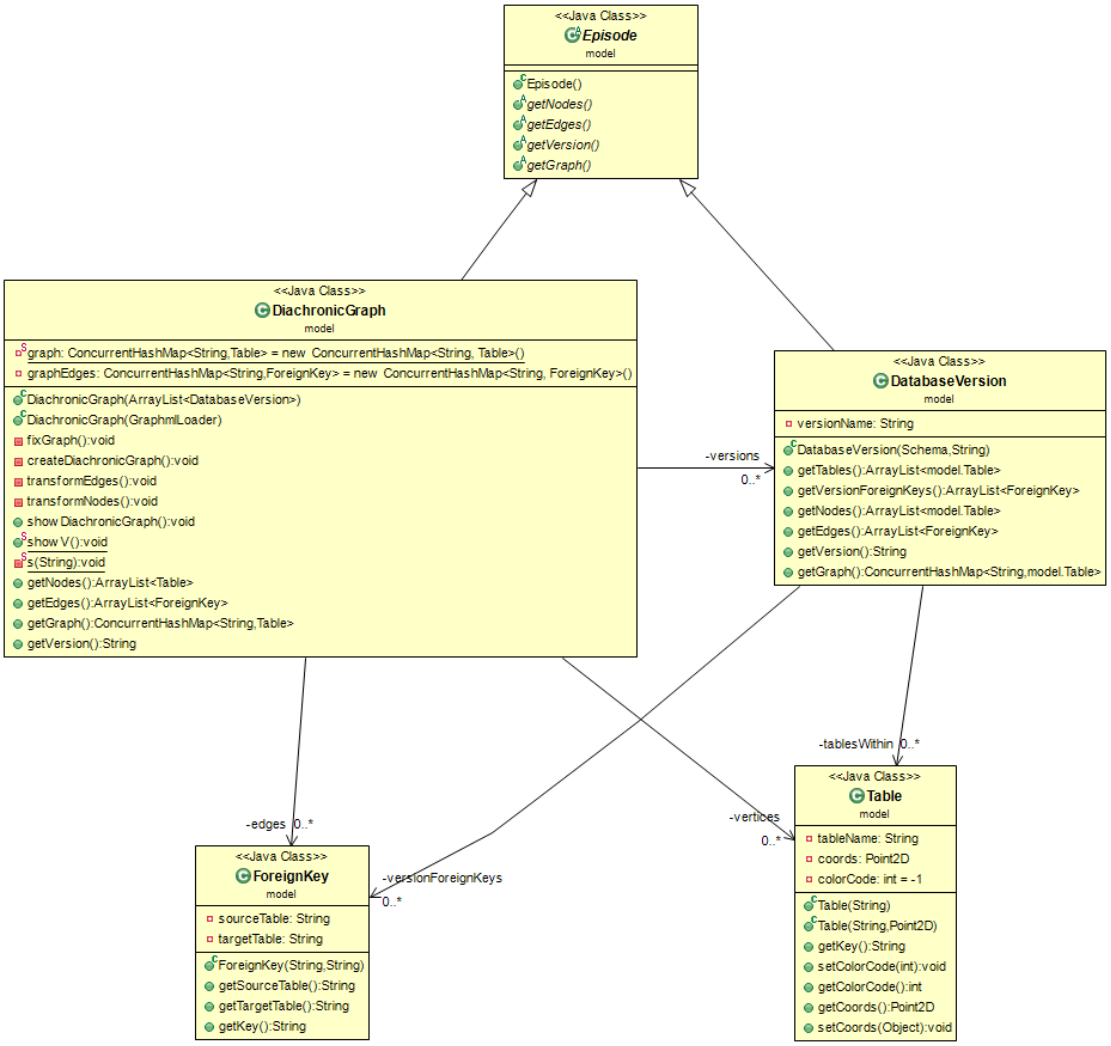
- ◆ Το πακέτο loader περιέχει τις κλάσεις που αφορούν το φόρτωμα αρχείων απο τον δίσκο στην κύρια μνήμη του συστήματός.
  - Η κλάση Parser είναι υπεύθυνη για την οργάνωση και μετατροπή των αρχείων εισόδου απο τον δίσκο σε αντικείμενα στο σύστημά μας.
  - Η κλάση GraphmlLoader είναι υπεύθυνη για το φόρτωμα αρχείων εισόδου τα οποία είναι προεπεξεργασμένα απο τον χρήστη και η πληροφορία τους είναι αποθηκευμένη σε αρχεία τύπου graphml.

Παρακάτω φαίνεται η UML αναπαράστασή του:



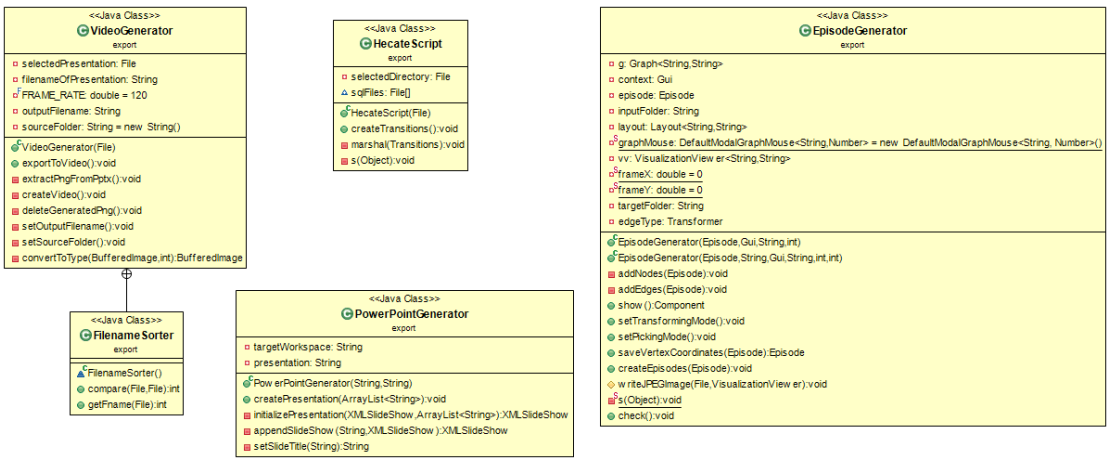
- ◆ Το πακέτο model περιέχει την βασική οργάνωση της πληροφορίας που αφορά τα περιεχόμενα κάθε έκδοσης της βάσης στην μνήμη του συστήματος.
  - Η κλάση Episode είναι μια αφηρημένη κλάση η οποία έχει δημιουργηθεί για την διευκόλυνση της δημιουργίας αντικειμένων της κλάσης EpisodeGenerator. Ουσιαστικά ομαδοποιεί τις κλάσεις DiachronicGraph και DatabaseVersion για χάρη της αφαίρεσης.

- Η κλάση DatabaseVersion αφορά την αναπαράσταση κάθε SQL αρχείου απο το dataset της εισόδου. Για κάθε έκδοση της βάσης υπάρχει ένα αντικείμενο DatabaseVersion που περιέχει το σύνολο των πινάκων που περιέχει η βάση και τις εξαρτήσεις **ξένων-κλειδιών** μεταξύ τους.
- Η κλάση DiachronicGraph περιέχει την ένωση των πινάκων απο το σύνολο των εκδόσεων της βάσης, καθώς και τις εξαρτήσεις **ξένων-κλειδιών** μεταξύ τους.
- Η κλάση Table αντιπροσωπεύει τον εκάστοτε πίνακα που έχει διαβαστεί απο την είσοδο και περιέχει πληροφορίες για το πώς θα γίνει η γραφική του αναπαράσταση.
- Η κλάση ForeignKey αντιπροσωπεύει την εξάρτηση **ξένου-κλειδιού** μεταξύ δύο πινάκων.



- Το πακέτο export περιέχει το σύνολο των κλάσεων που αφορούν την δημιουργία διάφορων αρχείων εξόδου.
  - Η κλάση EpisodeGenerator είναι υπεύθυνη για την οπτικοποίηση του DiachronicGraph καθώς και για την παραγωγή αρχείου εικόνας για κάθε ξεχωριστή έκδοση της βάσης.
  - Η κλάση HecateScript αφορά την δημιουργία του αρχείου transitions.xml που περιέχει τις διαφορές μεταξύ δύο διαδοχικών εκδόσεων.
  - Η κλάση PowerPointGenerator αφορά την δημιουργία μιας παρουσίασης PowerPoint δοθέντος ένος σύνολου εικόνων.
  - Η κλάση VideoGenerator αφορά την μετατροπή μιας παρουσίασης PowerPoint σε αρχείο mp4.

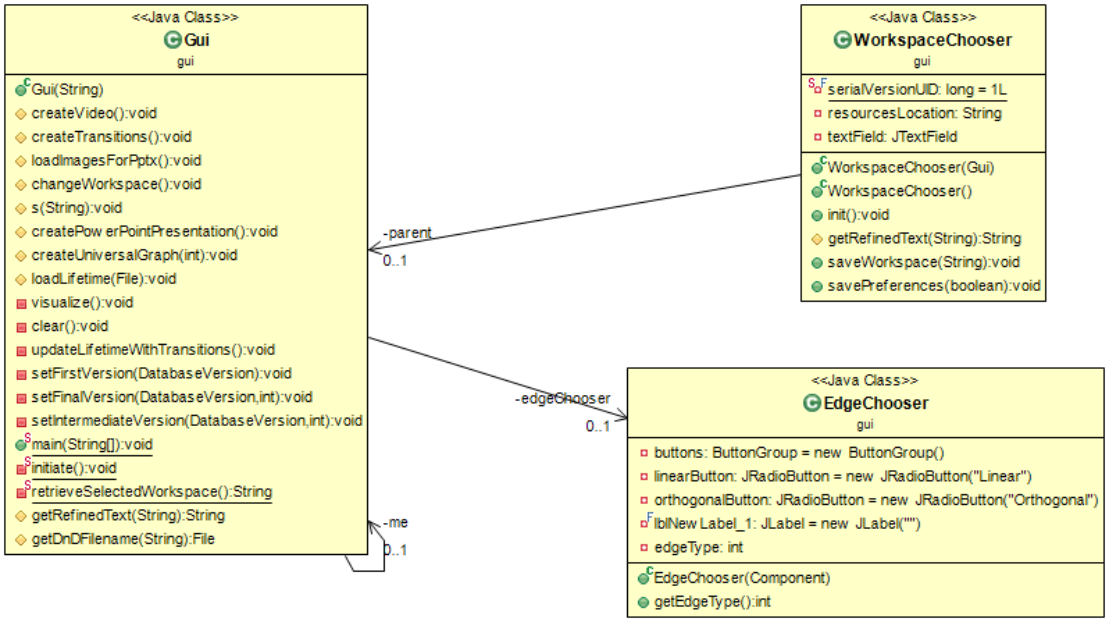
Με την UML αναπαράσταση:



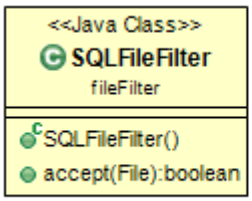
- Το πακέτο gui περιέχει όλες τις κλάσεις που αφορούν την γραφική διεπαφή με τον χρήστη.
  - Η κλάση GUI είναι η βασική διεπαφή με τον χρήστη. Είναι η κύρια κλάση που περιέχει όλες τις λειτουργικότητες που είναι διαθέσιμες απο την εφαρμογή μας.
  - Η κλάση WorkspaceChooser είναι μια βοηθητική γραφική διεπαφή με την οποία ο χρήστης μπορεί να επιλέξει την επιφάνεια εργασίας του. Αυτό προσφέρει πληροφορίες (το πλήρες όνομα αρχείου) στο σύστημά μας σχετικά με τον χώρο όπου πρέπει να γίνει η δημιουργία των αρχείων εξόδου.

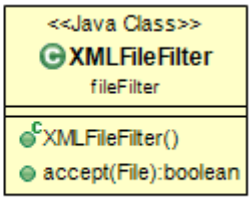
- Η κλάση EdgeChooser είναι επίσης μια βοηθητική γραφική διεπαφή η οποία δίνει την δυνατότητα στον χρήστη να προσδιορίσει το είδος των ακμών του γραφήματος.

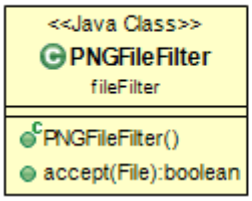
Με την UML αναπαράσταση:

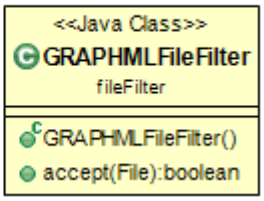


- Το πακέτο fileFilter περιέχει ένα σύνολο κλάσεων οι οποίες είναι υπεύθυνες για το "φιλτράρισμα" συγκεκριμένων τύπων αρχείων.
  - Η κλάση GRAPHMLFileFilter επιστρέφει μόνο το σύνολο των αρχείων τύπου graphml.
  - Η κλάση PNGFileFilter επιστρέφει μόνο το σύνολο των αρχείων τύπου png.
  - Η κλάση SQLFileFilter επιστρέφει μόνο το σύνολο των αρχείων τύπου SQL.
  - Η κλάση XMLFileFilter επιστρέφει μόνο το σύνολο των αρχείων τύπου xml.









- Το πακέτο hecateImports περιέχει το σύνολο των κλάσεων της Εκάτης, τις οποίες κάνει χρήση το σύστημά μας.

- Το πακέτο icons περιέχει το σύνολο των εικονιδίων που κάνει χρήση το σύστημά μας.

- Το πακέτο resources περιέχει την εφαρμογή PPTtoALL.exe η οποία είναι γραμμένη σε C# και μετατρέπει ένα οποιοδήποτε αρχείο pptx σε αρχείο wmv. Η χρήση αυτής της εφαρμογής γίνεται απο το σύστημά μας με κλήση συστήματος απο την γραμμή εντολών.

4 Υλοποίηση

4.1 Πλατφόρμες και προγραμματιστικά εργαλεία

Το σύστημα που δημιουργήθηκε στα πλαίσια αυτής της πτυχιακής εργασίας αναπτύχθηκε σε Java στο ολοκληρωμένο περιβάλλον ανάπτυξης Eclipse [http://www.eclipse.org/]. Ένας απο του βασικούς λόγους που χρησιμοποιήθηκε η Java για την ανάπτυξη του εργαλείου είναι ικανότητα της να έχει όσο το δυνατόν λιγότερες εξαρτήσεις απο το hardware της συσκευής που εκτελείται. Δηλαδή αφου μεταφραστεί ο κώδικας της εφαρμογής είναι ικανός να τρέξει σε οποιοδήποτε μηχάνη με εικονική μηχανή Java (Java virtual machine – JVM) ανεξαρτήτως απο την αρχιτεκτονική του υπολογιστή, πράγμα που μας προσφέρει φορητότητα. Οι απαιτήσεις του συστήματος σε hardware δεν είναι υψηλές ούτε ώς προς την κύρια μνήμη, ούτε ώς προς την υπολογιστική ισχύ. Παρόλαυτά η είσοδος πολυ μεγάλων αρχείων είναι λογικό να επιφέρει πρόσθετη καθυστέρηση για τις λειτουργίες της εμφάνισης του γραφήματος, της δημιουργίας παρουσίασης PowerPoint και μετατροπης σε mp4.

4.2 Λεπτομέρειες υλοποίησης

Σε αυτήν την ενότητα αυτό θα αναλύσουμε τον τρόπο υλοποιήσης των βασικών λειτουργιών του εργαλείου.

Όπως αναφέραμε πριν ο φάκελος εισόδου αποτελείται απο:

- Ένα σύνολο αρχείων SQL, τα οποία περιγράφουν το σχήμα της βάσης ανα έκδοση.

- Ένα αρχείο xml, το οποίο περιγράφει τις μετατροπές του σχήματος της βάσης μεταξύ δύο διαδοχικών εκδόσεων.

- Τέλος μπορεί να υπάρχει και ένα αρχείο graphml, σε περίπτωση που ο χρήστης έχει αποθηκεύσει προγενέστερα τις αλλάγες που έχει επιφέρει στο συνολικό γράφο του σχήματος της βάσης.

Παρόλαυτά η βασική απαίτηση του εργαλείου είναι η ύπαρξη μόνο του συνόλου των αρχείων SQL καθώς το ίδιο το εργαλείο είναι υπεύθυνο για την δημιουργία του αρχείου transitions.xml και graphml. Στο σημείο αυτό θα αναλύσουμε την κλάση που είναι υπεύθυνη για την παραγωγή του αρχείου μεταβάσεων απο έκδοση σε έκδοση – transitions.xml.

```
public class HecateScript {
	private File selectedDirectory;
	File[] sqlFiles;

	public HecateScript(File folder) throws Exception{
		Exception wrong = new Exception();

		selectedDirectory=folder;
		sqlFiles = selectedDirectory.listFiles(new SQLFileFilter());

		if(sqlFiles.length==0)
			throw wrong;
	}

	public void createTransitions(){

		HecateParser parser= new HecateParser();
		Schema currentSchema;
		ArrayList<Schema> schemata= new ArrayList<Schema>();

		//create schema per sql and store them
		for(int i=0;i<sqlFiles.length;++i){
			currentSchema=parser.parse(sqlFiles[i].getAbsolutePath());
			currentSchema.setTitle(String.valueOf(i));
			schemata.add(currentSchema);
		}
```

```
                Delta delta = new Delta();
                TransitionList tl;
                Transitions trs = new Transitions();

                for(int i=0;i<schemata.size()-1;++i){
                        tl=delta.minus(schemata.get(i),schemata.get(i+1));
                        trs.add(tl);
                }
                marshal(trs);
        }
        private void marshal(Transitions trs) {
                try {
                        JAXBContext jaxbContext = JAXBContext
                        .newInstance(Update.class,
                         Deletion.class, Insersion.class, TransitionList.class,
                        Transitions.class);

                        Marshaller jaxbMarshaller = jaxbContext.createMarshaller();
                        jaxbMarshaller
                        .setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

                        jaxbMarshaller.marshal(trs, new File(selectedDirectory
                        +"\\transitions.xml"));

                } catch (JAXBException e) {
                        e.printStackTrace();
                }
        }
}
```

Η κλάση **HecateScript** έχει πεδία selectedDirectory και sqlFiles που αναφέρονται στον φάκελο που περιέχει το σύνολο των αρχείων SQL και την λιστα απο αυτά τα αρχεία αντίστοιχα. Τα πεδία αυτά αρχικοποιούνται στον constructor με όρισμα τον φάκελο εισόδου που έχει επιλέξει ο χρήστης. Η κύρια μέθοδος για την δημιουργία του αρχείου μεταβάσεων είναι η *createTransitions()*, η οποία δημιουργεί πρώτα έναν **HecateParser** και τον χρησιμοποιεί για να παράξει ένα ***Schema*** για κάθε αρχείο SQL το οποίο αποθηκέυει σε μια λίστα. Στο τέλος αυτή η λίστα θα περιέχει το σχήμα της βάσης για κάθε έκδοση – αρχείο SQL. Στην συνέχεια για κάθε στοιχείο αυτής της λίστα θα δημιουργήσουμε αντικείμενα τύπου **TransitionList**, τα οποία περιέχουν τις αλλαγές ανάμεσα σε δύο διαδοχικές εκδόσεις.

Η δημιουργία των αντικειμένων **TransitionList** γίνεται με την βοήθεια της μεθόδου *minus(schema1, schema2)* της κλάσης **Delta.** Τέλος αποθηκεύουμε το σύνολο των **TransitionList** σ'ένα αντικείμενο **Transitions** το οποίο στην συνέχεια γράφουμε σε μορφή xml με την μέθοδο *marshal(transitions).*

Η μέθοδος *marshal(transitions)* μεταφέρει το όρισμά της, αντικείμενο τύπου **Transitions** απο την μνήμη του συστήματος στον δίσκο, στην τοποθεσία selectedDirectory σε μορφή xml με την βοήθεια της βιβλιοθήκης της Java JAXB. Η JAXB [https://jaxb.java.net/], Java Architecture for XML Binding, είναι μια βιβλιοθήκη που παρέχει έναν εύκολο τρόπο για mapping Java αντικειμένων σε xml αναπαραστάσεις και το αντίθετο. Ένα δείγμα του transitions.xml προβάλεται ενδεικτικά:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<transitions>
   <transition oldVersion="0" newVersion="1">
      <insersion type="UpdateTable">
         <table>
            <name>reference</name>
            <pKey>
               <mode>0</mode>
            </pKey>
         </table>
         <attribute>
            <name>reference_identifier</name>
            <type>VARCHAR(32)</type>
            <isKey>false</isKey>
         </attribute>
      </insersion>
      <deletion type="UpdateTable">
         <table>
            <name>taxon</name>
            <pKey>
               <mode>0</mode>
            </pKey>
         </table>
         <attribute>
            <name>right_id</name>
            <type>INT(10)</type>
```

```
            <isKey>false</isKey>
        </attribute>
    </deletion>
  </transition>
</transitions>
```

Με την ύπαρξη συμβατού αρχείου με το εργαλείο μας μπορεί να αρχίσει η διαδικασία της φόρτωσης και οπτικοποίησης του *Διαχρονικού γράφου*. Παρακάτω παρουσιάζεται ο κώδικας αυτών των λειτουργιών καθώς και λεπτομερής επεξήγησή του.

```
protected void loadLifetime(File selectedFile) throws Exception {
        edgeChooser = new EdgeChooser(me);
        input=selectedFile.toString();
        String [] array= selectedFile.toString().split("\\\\");
        targetFolder=workspace+"\\"+array[array.length-1];

        boolean done =new File(targetFolder).mkdirs();
        int episodeGeneratorMode;
        panel.setVisible(false);

        Parser myParser;

        myParser = new Parser(selectedFile);
        lifetime=myParser.getLifetime();
        transitions=myParser.getTransitions();
        updateLifetimeWithTransitions();

        if(myParser.isGraphml()){
                savedChanges=myParser.getGraphmlLoader();
                createDiachronicGraph(1);
                episodeGeneratorMode=1;
        }else{
                createDiachronicGraph(0);
                episodeGeneratorMode=0;
        }
```

```
        eg = new EpisodeGenerator(wholeGraph,input,me,targetFolder,
                edgeChooser.getEdgeType(),
                episodeGeneratorMode);
        canvas=eg.show();
        }
```

Αρχικά δημιουργούμε ένα αντικείμενο **EdgeChooser** για να δώσουμε στον χρήστη την επιλογή του είδους ακμών που θέλει να χρησιμοποιηθεί. Στην συνέχεια παίρνουμε πληροφορίες απο το αρχείο εισόδου και ενημερώνουμε τα πεδία input και targetFolder της κλάσης μας GUI. Με την βοήθεια αυτών των πεδίων το σύστημα έχει αρκετές πληροφορίες για το που θα δημιουργήσει τα αρχεία εξόδου. Δημιουργούμε ένα αντικείμενο της κλάσης **Parser** το οποίο διαβάζει τα αρχεία SQL xml, και graphml αν υπάρχει, και ενημερώνει τα πεδια lifetime και transitions τα οποία είναι μια λίστα απο **DatabaseVersion** και μια λίστα απο hashmap, της μορφής <όνομα_πίνακα, κωδικός_χρώματος>, αντίστοιχα. Ο τρόπος λειτουργίας της κλάσης **Parser** θα περιγραφεί παρακάτω, ενώ έχουμε ορίσει πιο πάνω την λειτουργία της κλάσης **DatabaseVersion**. Στην συνέχεια ελέγχουμε για το αν υπάρχει ή όχι το αρχείο graphml, δηλαδή προγενέστερες μεταβολές του χρήστη τις οποίες έχει αποθηκεύσει, και δημιουργούμε ένα αντικείμενο της κλάσης **DiachronicGraph** με τον κατάλληλο τρόπο. Τέλος δημιουργούμε ένα αντικείμενο **EpisodeGenerator** για την προβολή του συνόλου των κόμβων στον χρήστη. Ο τρόπος δημιουργίας των **DiachronicGraph** και **EpisodeGenerator** παραλείπεται για τώρα και θα αναφερθει αναλυτικά παρακάτω.

Στον παραπάνω κώδικα πολύ σημαντική είναι η συμβολή της συνάρτησης *updateLifetimeWithTransitions()*:

```
private void updateLifetimeWithTransitions(){
        for(int i=0;i<lifetime.size();++i)
                if(i==0)
                        setFirstVersion(lifetime.get(i));
                else if(i==lifetime.size()-1)
                        setFinalVersion(lifetime.get(i),i);
                else
                        setIntermediateVersion(lifetime.get(i),i);
}
```

η οποία διατρέχει το σύνολο των εκδόσεων της βάσης και ενημερώνει την χρωματική τιμή του συνόλου των αντικειμένων **Table**, που έχει αναλυθεί στην ενότητα 3. Η χρωματική τιμή των αντικειμένων **Table** είναι ουσιαστικά ο τρόπος για να επισημάνουμε, χρωματικά, τους πίνακες που θα διαγραφούν, που μεταβλήθηκαν ή μόλις δημιουργήθηκαν. Για κάθε ενδιάμεση έκδοση πρέπει να δούμε την επόμενη και την προηγούμενή της. Ως ειδικές περιπτώσεις αντιμετωπίζουμε την πρώτη και την τελευταία, όπου πρέπει να ελέγξουμε μόνο το επόμενο και το προηγούμενο αντίστοιχα. Παρακάτω περιγράφεται αναλυτικά η διαδικασία που ακολουθείται.

```
private void setFirstVersion(DatabaseVersion fversion){
   for(int i=0;i<fversion.getTables().size();++i)
        if(transitions.get(0).containsKey(fversion.getTables().get(i).getKey())
        && transitions.get(0).get(fversion.getTables().get(i).getKey())==1)
                fversion.getTables().get(i).setColorCode(1);
}
```

Για την πρώτη έκδοση κοιτάζουμε το πρώτο hashmap. Ελέγχουμε για κάθε πίνακα της έκδοσης αυτής αν υπάρχει καταχώριση με ένδειξη διαγραφής και ενημερώνουμε το εκάστοτε αντικείμενο της κλάσης **Table** με τον κωδικό χρώματος για την διαγραφή.

```
private void setFinalVersion(DatabaseVersion fversion,int k){
   for(int i=0;i<fversion.getTables().size();++i)
        if(transitions.get(k-1).containsKey(fversion.getTables().get(i).getKey())
        && transitions.get(k-1).get(fversion.getTables().get(i).getKey())!=1)
                fversion.getTables().get(i).setColorCode(transitions.get(k-1)
                .get(fversion.getTables().get(i).getKey()));

}
```

Για την τελευταία έκδοση κοιτάζουμε το τελευταίο hashmap. Ελέγχουμε για κάθε πίνακα της έκδοσης αυτής αν υπάρχει καταχώριση με ένδειξη διάφορη της διαγραφής και ενημερώνω το εκάστοτε αντικείμενο της κλάσης **Table** με τον κωδικό χρώματος που του αντιστοιχεί.

```
private void setIntermediateVersion(DatabaseVersion version,int k){
   for(int i=0;i<version.getTables().size();++i){

        if(transitions.get(k).containsKey(version.getTables().get(i).getKey())
```

```
        && transitions.get(k).get(version.getTables().get(i).getKey())==1)
                version.getTables().get(i).setColorCode(1);

        if(transitions.get(k-1).containsKey(version.getTables().get(i).getKey())
        && transitions.get(k-1).get(version.getTables().get(i).getKey())!=1)
                version.getTables().get(i).setColorCode(transitions.get(k-1)
                .get(version.getTables().get(i).getKey()));
    }
}
```

Τέλος για κάθε ενδιάμεση έκδοση κοιτάζουμε 2 hashmaps. Ελέγχουμε για κάθε πίνακα της τρέχουσας έκδοσης άν υπάρχει καταχώριση διαγραφής στο hashmap του οποίου το index είναι ίδιο με το index της τρέχουσας έκδοσης. Στην συνέχεια ελέγχουμε αν υπάρχει καταχώριση διάφορη της διαγραφής στο hashmap του οποίου το index προηγείται κατα ένα απο την τρέχουσα έκδοση και ενημερώνουμε το εκάστοτε αντικείμενο της κλάσης **Table** με τον κωδικό χρώματος που του αντιστοιχεί, είτε για δημιουργία καινουργιου πίνακα είτε για ανανέωση πίνακα.

Σε αυτό το σημείο θα ασχοληθούμε με την κλάση **Parser** και τον τρόπο με τον οποίο δουλεύει. Η κλαση περιέχει πολλές μεθόδους get έτσι θα αναλύσουμε μόνο τις μεθόδους και τα σημεία που έχουν ενδιαφέρον, αρχίζοντας απο τον constructor της.

```
public Parser(File folder) throws Exception{
        selectedDirectory = new File(folder.getPath());

        File[] versions = selectedDirectory.listFiles(new SQLFileFilter());
        for(int i=0;i<versions.length;++i)
                parseLifetime(versions[i]);

        File transitions = selectedDirectory.listFiles(new XMLFileFilter())[0];
        parseTransitions(transitions);

        if(selectedDirectory.listFiles(new GRAPHMLFileFilter()).length==1){
                graphml=true;
```

```
                try {
                        graphmlLoader = new GraphmlLoader(selectedDirectory
                        .listFiles(new GRAPHMLFileFilter())[0].toString());
                } catch (FileNotFoundException e1) {
                        e1.printStackTrace();
                }
        }
}
```

Η διάρθρωση του κώδικα είναι απλή και έχει τρία στάδια:

- ✓ Επεξεργάζεται τα αρχεία SQL, που το καθένα περιέχει μια έκδοση της βάσης.
- ✓ Επεξεργάζεται το αρχείο xml, το οποίο περιέχει πληροφορίες μεταβολής της βάσης κατα την μετάβασή της απο την μια έκδοση στην επόμενη.
- ✓ Επεξεργάζεται, αν υπάρχει, το αρχείο graphml, το οποίο περιέχει τις συντεταγμένες του κάθε κόμβου όπως έχουν δοθεί προγενέστερα απο τον χρήστη.

Για την επεξεργασία των αρχείων SQL:

```
private void parseLifetime(File version){
        try {
                HecateParser parser= new HecateParser();
                Schema schema1 = parser.parse(version.getAbsolutePath());
                DatabaseVersion current = new DatabaseVersion(
                schema1,version.getName());
                lifetime.add(current);
        } catch (RecognitionException e) {
                e.printStackTrace();
        }
}
```

Όπως φαίνεται δημιουργούμε ένα αντικείμενο της κλάσης **HecateParser**, το οποίο διαβάζει κάθε αρχείο SQL το μετατρέπει σε αντικείμενο της κλάσης **Schema**, απο το αντικείμενο αυτό δημιουργείται ένα νέο αντικείμενο της κλάσης **DatabaseVersion** και προστίθεται στην λίστα lifetime.

Για την επεξεργασία του αρχείου xml:

```
private void parseTransitions(File transition){
        InputStream inputStream;
        Transitions t = null;
        try {
                inputStream = new FileInputStream(transition.getAbsolutePath());
                JAXBContext jaxbContext = JAXBContext.newInstance(Update.class,
                Deletion.class,Insersion.class, TransitionList.class,
                Transitions.class);
                Unmarshaller u = jaxbContext.createUnmarshaller();
                t = (Transitions)u.unmarshal( inputStream );
        } catch (Exception e) {
                e.printStackTrace();
        }
        for(TransitionList tl :t.getList()){
           Map<String,Integer>  temp = new HashMap<String,Integer>();
          if(tl.getTransitionList()!=null){
           for(int i=0;i<tl.getTransitionList().size();++i){
              if(tl.getTransitionList().get(i).type.equals("NewTable"))
                  temp.put(tl.getTransitionList().get(i).affectedTable.getName(),0);
              else if(tl.getTransitionList().get(i).type.equals("DeleteTable"))
                  temp.put(tl.getTransitionList().get(i).affectedTable.getName(),1);


              else if(tl.getTransitionList().get(i).type.equals("UpdateTable"))
                 temp.put(tl.getTransitionList().get(i).affectedTable.getName(),2);
           }
        }
        transitions.add(temp);
    }
}
```

Αρχικά κάνουμε *unmarshal* το αρχείο xml, με αυτόν τον τρόπο μετατρέπουμε το περιεχόμενο του αρχείου σε αντικείμενα του συστήματος μας. Η διαδικασία είναι η αντίστροφη απο αυτήν που περιγράφηκε παραπάνω για την **δημιουργία** του αρχείου xml απο το σύστημά μας. Έτσι μετά το πέρας της διαδικασίας αυτής όλη η πληροφορία για τις μεταβάσεις του σχήματος της βάσης απο έκδοση σε έκδοση υπάρχει συγκεντρωμένη σε ένα αντικείμενο της κλάσης **Transitions**, το οποίο έχει μια συλλογή απο αντικείμενα τύπου **TransitionList**. Διατρέχουμε την σύλλογη απο **TransitionList** και για κάθε στοιχείο της παίρνουμε το όνομα του affectedTable και το type του και φτιάχνουμε ένα hashmap<String,Integer>, με το όνομα του τρέχοντος πίνακα και τον κωδικό χρωματισμού του, ο οποίος έχει ως εξής:

- 0: Ο πίνακας δημιουργήθηκε πρώτη φορά σε αυτήν την έκδοση.
- 1: Ο πίνακας θα διαγραφεί στην επόμενη έκδοση.
- 2: Ο πίνακας είχε την εσωτερική του δομη ανανεωμένη σε σχέση με την προηγούμενη έκδοση.

Για την επεξεργασία του αρχείου graphml:

Η ύπαρξη του αρχείου graphml δεν είναι υποχρεωτική, αν ένα dataset περιέχει το σύνολο των αρχείων SQL και το transitions.xml μπορεί να φορωθεί χωρίς κανένα πρόβλημα στο σύστημα. Σε περίπτωση που το αρχείο graphml υπάρχει ο **Parser** ενημερώνει μια μεταβλητή οτι υπάρχει είσοδος δεδομένων απο το συγκεκριμένο αρχειο, δηλαδή η παρουσίαση του **DiachronicGraph** στον χρήστη θα γίνει σύμφωνα με τις προϋπάρχουσες συντεταγμένες και οχι απο τον default αλγόριθμο. Στην συνέχεια δημιουργεί να αντικείμενο της κλάσης **GraphmlLoader** για την εξαγωγή των συντεταγμένων απο το αρχείο graphml και την οργάνωσή τους στην μνήμη.

Η επόμενη κλάση που πρέπει να αναλύσουμε είναι η **DiachronicGraph**. Αντικείμενα αυτής της κλάσης αντιπροσωπεύουν την ένωση όλων των πινάκων που υπάρχουν κατα την διάρκεια ζωής της βάσης, δηλαδή στο σύνολο των εκδόσεων της. Η χρησιμότητα αυτής της κλάσης είναι αφενός να δώσει στον χρήστη την δυνατότητα να επεξεργαστεί το γράφημα ο ίδιος και αφετέρου να "κλειδώσει" τις συντεταγμένες των κόμβων κατά την παραγωγή των επιμέρους στιγμιοτύπων της διάρκειας ζωής της βάσης.

Γι'αυτήν την κλάση έχουμε ορίσει δύο constructors:

```
public DiachronicGraph(ArrayList<DatabaseVersion> vrs) {
        versions = vrs;
        createDiachronicGraph();
}

public DiachronicGraph(GraphmlLoader gml){
        vertices=gml.getNodes();
        edges=gml.getEdges();
        fixGraph();
}
```

Ο πρώτος είναι η default περίπτωση όπου δεν υπάρχει πληροφορία για προϋπάρχουσες συντεταγμένες κόμβων και έτσι παίρνουμε το σύνολο των κόμβων και τον ακμών απο το

πεδίο lifetime το οποίο έχουμε αναφέρει πιο πάνω και είναι μια λίστα απο **DatabaseVersion** αντικείμενα. Ενώ ο δεύτερος constructor αφορά την περίπτωση όπου το σύνολο των κόμβων και ακμών έχουν διαβαστεί απο το αρχείο graphml.

Επόμενο βήμα είναι η οπτικοποίηση του DiachronicGraph στον χρήστη ούτως ώστε να μπορέσει να το επεξεργαστεί. Υπεύθυνη κλάση για τον σκοπό αυτόν είναι η **EpisodeGenerator**, η οποία ενσωματώνει στο σύστημα το API της Jung, το οποίο είναι υπεύθυνο για την οπτικοποίηση γραφημάτων. Όπως και πριν και εδώ έχουμε ορίσει 2 constructors:

```
public EpisodeGenerator(Episode ep, Gui gui, String tf, int et) {

        edgeType = et == 0 ? new EdgeShape.Line<String, String>():
        new EdgeShape.Orthogonal<String, String>();

        g = new DirectedSparseGraph<String, String>();
        context = gui;
        episode = ep;

        targetFolder = tf;
        addNodes(episode);
        addEdges(episode);

        layout = new FRLayout<String, String>(g);
        layout.setSize(new Dimension(700, 700));
        vv = new VisualizationViewer<String, String>(layout);
        vv.setPreferredSize(new Dimension(1000, 1000));

        Transformer<String, Paint> vertexPaint = new Transformer<String, Paint>() {
                public Paint transform(String i) {
                        return new Color(207, 247, 137, 200);
                }
        };

        vv.getRenderContext().setVertexFillPaintTransformer(vertexPaint);
        vv.getRenderContext().setVertexLabelTransformer(new ToStringLabeller());
        vv.getRenderer().getVertexLabelRenderer().setPosition(Position.NE);
```

```
        vv.setBackground(Color.WHITE);
        vv.getRenderContext().setEdgeShapeTransformer(edgeType);

        graphMouse.setMode(ModalGraphMouse.Mode.PICKING);
        vv.setGraphMouse(graphMouse);
        }

public EpisodeGenerator(Episode ep,String in,Gui gui, String tf, int et,int mode) {

        edgeType = et == 0 ? new EdgeShape.Line<String, String>():
        new EdgeShape.Orthogonal<String, String>();

        g = new DirectedSparseGraph<String, String>();
        context = gui;
        inputFolder=in;
        episode = ep;
        targetFolder = tf;
        addNodes(episode);
        addEdges(episode);

        layout = new FRLayout<String, String>(g);
        layout.setSize(new Dimension(700, 700));
        vv = new VisualizationViewer<String, String>(layout);
        vv.setPreferredSize(new Dimension(1000, 1000));

        Transformer<String, Paint> vertexPaint = new Transformer<String, Paint>() {
                public Paint transform(String i) {
                        return new Color(207, 247, 137, 200);
                }
        };
        vv.getRenderContext().setVertexFillPaintTransformer(vertexPaint);
        vv.getRenderContext().setVertexLabelTransformer(new ToStringLabeller());
        vv.getRenderer().getVertexLabelRenderer().setPosition(Position.NE);
        vv.setBackground(Color.WHITE);
        vv.getRenderContext().setEdgeShapeTransformer(edgeType);

        graphMouse.setMode(ModalGraphMouse.Mode.PICKING);
        vv.setGraphMouse(graphMouse);
        if(mode==1)
```

```
                for (int i = 0; i < episode.getNodes().size(); ++i) {
                        layout.setLocation(episode.getNodes().get(i).getKey(),
                        episode.getNodes().get(i).getCoords());
                        layout.lock(episode.getNodes().get(i).getKey(), true);
                }
}
```

Ο πρώτος constructor χρησιμοποιείται για την παραγωγή του αρχείου απεικόνισης για κάθε έκδοση του σχήματος της βάσης, ενώ ο δεύτερος για την δημιουργία και προβολή στον χρήστη του *Διαχρονικού Γράφου* (DiachronicGraph). Ο δεύτερος constructor μάλιστα καλύπτει τις δύο περιπτώσεις δημιουργίας του DiachronicGraph, αν κατασκεύαστηκε σύμφωνα με το graphml (mode == 1) ή όχι (mode==0). Σε αυτό το σημείο βλέπουμε την χρησιμότητα της

Η κλάση **EpisodeGenerator** προσφέρει επιπλέον λειτουργικότητες:

- Ο χρήστης μπορεί να μετακινήσει οποιονδήποτε κόμβο επιθυμεί, με την επιλογή 'Pick Vertex'. Αυτό επιτυγχάνεται με την μέθοδο

```
public void setPickingMode() {
                graphMouse.setMode(ModalGraphMouse.Mode.PICKING);
                vv.setGraphMouse(graphMouse);
            }
```

- Ο χρήστης μπορεί να μετακινήσει όλο το γράφημα στο παράθυρο προβολής με την επιλογή 'Translate Graph'. Αυτό επιτυγχάνεται με την μέθοδο

```
public void setTransformingMode() {
                graphMouse.setMode(ModalGraphMouse.Mode.TRANSFORMING);
                vv.setGraphMouse(graphMouse);
            }
```

- Ο χρήστης μπορεί να αποθηκεύσει τις αλλαγές που έχει επιφέρει στον *Διαχρονικό Γράφο* σύμφωνα με το πρότυπο graphml, πράγμα που προσφέρει φορητότητα σε εφαρμογές που είναι συμβατές με αυτό το πρότυπο με την επιλογή 'Save Vertex Coordinates'. Αυτό επιτυγχάνεται με την μέθοδο

```
public Episode saveVertexCoordinates(Episode episode) throws IOException {

                for (int i = 0; i < episode.getNodes().size(); ++i)
                    episode.getNodes()
                    .get(i)
                    .setCoords(layout.transform(episode.getNodes().get(i).getKey()));
```

```
GraphMLWriter<String,String> graphWriter = new GraphMLWriter<String
,String> ();

PrintWriter out = new PrintWriter(new BufferedWriter(new
FileWriter(inputFolder+"\\uber.graphml")));



graphWriter.addVertexData("x", null, "0",new Transformer<String,
String>() { public String transform(String v) {
        return Double.toString(((AbstractLayout<String, String>)
        layout).getX(v));
        }
    }
);

graphWriter.addVertexData("y", null, "0",
new Transformer<String, String>() {
public String transform(String v) {
         return Double.toString(((AbstractLayout<String, String>)
         layout).getY(v));
         }
      }
);

graphWriter.save(g, out);
return episode;
}
```

- Τέλος ο χρήστης μπορεί να δημιουργήσει αρχεία εικόνας για κάθε μια απο τις εκδόσεις του σχήματος της βάσης με την επιλογή 'Produce Episodes'. Αυτό επιτυγχάνεται με την μέθοδο

```
public void createEpisodes(final Episode Diachronic) {

layout = new FRLayout<String, String>(g);
layout.setSize(new Dimension(1000, 1000));
VisualizationViewer<String, String> vv = new
VisualizationViewer<String, String>(layout);
vv.setPreferredSize(new Dimension(1100, 1100));

Transformer<String, Paint> vertexPaint = new Transformer<String,
Paint>() {
```

```
    public Paint transform(String v) {
        for (int i = 0; i < episode.getNodes().size(); ++i)
            if (episode.getNodes().get(i).getKey().equals(v))
               if (episode.getNodes().get(i).getColorCode() == -1)
                   return new Color(129, 215, 244, 200);
               else if (episode.getNodes().get(i).getColorCode() == 0)
                   return new Color(0, 255, 0, 200);
               else if (episode.getNodes().get(i).getColorCode() == 1)
                   return new Color(255, 0, 0, 200);
               else
                   return new Color(255, 255, 0, 200);

        return Color.BLACK;

     }
 };

for (int i = 0; i < episode.getNodes().size(); ++i) {
        layout.setLocation(episode.getNodes().get(i).getKey(),
        (Diachronic.getGraph().get(episode.getNodes().get(i)
        .getKey()).getCoords()));
        layout.lock(episode.getNodes().get(i).getKey(), true);
}

MutableTransformer modelTransformer = vv.getRenderContext()
.getMultiLayerTransformer().getTransformer(Layer.LAYOUT);
modelTransformer.setTranslate(frameX, frameY);

vv.getRenderContext().setVertexFillPaintTransformer(vertexPaint);
vv.getRenderContext().setVertexLabelTransformer(new
ToStringLabeller());
vv.getRenderer().getVertexLabelRenderer().setPosition(Position.NE);
vv.getRenderContext().setEdgeShapeTransformer(edgeType);
vv.setBackground(Color.WHITE);

JFrame frame = new JFrame((episode.getVersion()));
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.getContentPane().add(vv);
frame.pack();
frame.setVisible(false);
```

```
            writeJPEGImage(new File(targetFolder + "/"
                                + episode.getVersion() + ".jpg"), vv);

        }
```

Αρχικά φτιάχνουμε ένα αντικείμενο τύπου **Layout**, η κλάσης **Layout** είναι αφηρημένη οπότε πρέπει να φτιάξουμε ένα αντικείμενο μιας κλάσης που έχει ως *γονική* κλάση την **Layout**, επιλέξαμε να δημιουργήσουμε ένα αντικείμενο τύπου **FRLayout**, το οποίο όμως δεν έχει μεγάλη σημασία αφου οι συντεταγμένες θα δωθούν ρητά απο το αντικείμενο DiachronicGraph το οποίο περιέχει τις συντεταγμένες που έχει δώσει ο χρήστης προγενέστερα. Επίσης φτιάχνουμε ένα αντικείμενο τύπου **VisualizationViewer** το οποίο μας βοηθάει να προβάλουμε πάνω του τα γραφήματα. Στην συνέχεια ορίζουμε έναν αντικείμενο τύπου **Transformer** το οποίο είναι υπεύθυνο για το χρώμα των κόμβων που θα δημιουργηθούν στο γράφημα. Το μόνο που κάνουμε είναι να αντιστοιχίσουμε τον χρωματικό κωδικό που αναφέραμε πριν στα ανάλογο χρώμα

- Διαγραφή, κόκκινο
- Δημιουργία, πράσινο
- Ανανέωση, κίτρινο

Έπειτα διατρέχουμε όλους τους κόμβους της έκδοσης της βάσης, και για κάθε πίνακα, ορίζουμε τις συντεταγμένες του σύμφωνα με την αντίστοιχη εγγραφή που υπάρχει στο πεδίο *graph* του αντικειμένου τύπου **DiachronicGraph**.

Ο χρήστης έχει την δυνατότητα να να διαμορφώσει τον γράφο όπως θέλει αυτός, μετακινώντας τους κόμβους ή ακόμα και ολόκληρο τον γράφο. Παραπάνω δείξαμε πως οι πίνακες της κάθε έκδοσης παίρνουν τις συντεταγέμενες που έχει δώσει ο χρήστης τώρα θα δείξουμε πως η κάθε έδοση έχει μετακινημένο ολόκληρο το γράφημά της σύμφωνα με αυτό που έχει ορίσει ο χρήστης. Δημιουργούμε ένα αντικείμενο τύπου **MutableTransformer** το οποίο περιέχει μια αναφορά στο layout του του αντικειμένου **VisualizationViewer** και με την εντολή *setTranslate(frameX,frameY)* ορίζουμε το πόσο θέλουμε να μετακινηθεί το γράφημα μας. Τις ποσότητες *frameX, frameY* τις οριστικοποιήσαμε μετά το τέλος της επεξεργασίας του γραφήματος απο τον χρήστη.

Στην συνέχεια βάζουμε τους transformers που ορίσαμε πριν στον **VisualizationViewer**, για το χρώμα του κόμβου,για το πως θα προβάλονται τα ονόματα του κάθε πίνακα, και για το είδος των ακμών, θέτουμε το background του σε άσπρο και τον προβάλουμε σε ένα JFrame

το οποίο όμως δεν προβάλουμε στον χρήστη. Τέλος μετατρέπουμε το περιεχόμενο του JFrame σε αρχείο εικόνας με την συνάρτηση *writeJPEGImage* η οποία είναι ώς εξής:

```
protected void writeJPEGImage(File file, VisualizationViewer vv) {
        int width = vv.getWidth();
        int height = vv.getHeight();

        BufferedImage bi = new BufferedImage(width, height,
        BufferedImage.TYPE_INT_RGB);
        Graphics2D graphics = bi.createGraphics();
        vv.paint(graphics);
        graphics.dispose();

        try {
                ImageIO.write(bi, "jpeg", file);
        } catch (Exception e) {
                e.printStackTrace();
        }
}
```

Όσον αφορά τις κλάσεις που είναι υπεύθυνες για την παραγωγή αρχείων εξόδου, έχουμε τις **PowerPointGenerator** και **VideoGenerator**.

Για την δημιουργία μίας παρουσίασης PowerPoint χρησιμοποιούμε την εξωτερική βιβλιοθήκη Apache-POI που μας παρέχει ένα πλήρες API γι'αυτον τον σκοπό.

```
public class PowerPointGenerator {
        private String targetWorkspace;
        private String presentation;

        public PowerPointGenerator(String workspace,String pres){
                targetWorkspace=workspace;
                presentation=pres;
        }

        public void createPresentation(ArrayList<String> args)
        throws  FileNotFoundException, IOException{

                XMLSlideShow ppt = new XMLSlideShow();

                ppt=initializePresentation(ppt,args);
```

```
        for(int i=0;i<args.size();++i)
                ppt=appendSlideShow(args.get(i),ppt);

        FileOutputStr FileOutputStream(targetWorkspace
        +"/"+presentation+".pptx");

        ppt.write(out);
        out.close();
}

private XMLSlideShow initializePresentation(XMLSlideShow ppt,ArrayList<String>
files) throws IOException {

        XSLFSlideMaster defaultMaster = ppt.getSlideMasters()[0];

        for(int i=0;i<files.size();++i){
           if(files.get(i).contains("Universal Graph")){
              XSLFSlide slide0 = ppt.createSlide();

              BufferedImage bimg = ImageIO.read(new File(files.get(i)));
              int width          = bimg.getWidth();
              int height         = bimg.getHeight();

              byte[] data = IOUtils.toByteArray(
              new FileInputStream(files.get(i)));

              int pictureIndex = ppt.addPicture(data,
              XSLFPictureData.PICTURE_TYPE_JPEG);

             XSLFPictureShape shape = slide0.createPicture(pictureIndex);
            ppt.setPageSize(new java.awt.Dimension(1100,height+100));

        }
}
        XSLFSlideLayout title =defaultMaster.getLayout(SlideLayout.TITLE_ONLY);
        XSLFSlide slide = ppt.createSlide(title);
        XSLFTextShape title1 = slide.getPlaceholder(0);
        title1.setAnchor(new Rectangle(0,1200/4-100/2,1100,100));
        title1.setText(setSlideTitle("Evolution Story"));
```

```
            return ppt;
        }

        private XMLSlideShow appendSlideShow(String imgPath,XMLSlideShow ppt)
        throws FileNotFoundException, IOException{

            if(!imgPath.contains(".jpg")|| imgPath.contains("Universal Graph"))
               return ppt;


            XSLFSlideMaster defaultMaster = ppt.getSlideMasters()[0];
            XSLFSlideLayout title = defaultMaster.getLayout(SlideLayout.TITLE);
            XSLFSlide slide = ppt.createSlide(title);

            XSLFTextShape title1 = slide.getPlaceholder(0);
            title1.setAnchor(new Rectangle(0,0,1100,100));
            title1.setText(setSlideTitle(imgPath));

            BufferedImage bimg = ImageIO.read(new File(imgPath));
            int width= bimg.getWidth();
            int = bimg.getHeight();


            byte[] data = IOUtils.toByteArray(new FileInputStream(imgPath));
            int pictureIndex = ppt.addPicture(data, XSLFPictureData.PICTURE_TYPE_JPEG);
            XSLFPictureShape shape = slide.createPicture(pictureIndex);
            shape.setAnchor(new Rectangle(0,100,width,height));

            ppt.setPageSize(new java.awt.Dimension(1100,height+100));

            return ppt;
        }

        private String setSlideTitle(String imgPath) {
                String[] leftArray = imgPath.split(".jpg",2);
                String[] rightArray = leftArray[0].split("\\\\");
                return rightArray[rightArray.length-1];
        }
}
```

Ο constructor της κλάσης είναι απλός, χρειάζεται μόνο το όνομα της επιφάνειας εργασίας της session του εργαλείου και το όνομα του αρχείου εισόδου. Η κύρια μέθοδος της κλάσης, και η μόνη public είναι η createPresentation. Αρχικά δημιουργεί ένα αντικείμενο παρουσίασης PowerPoint τύπου XMLSlideShow το οποίο αρχικοποιεί η initializePresentation, η οποία ουσιαστικά προσθέτει στην αρχή τον DiachronicGraph και άλλη μια διαφάνεια με το λεκτικό "Evolution Story". Στην συνέχεια για κάθε στιγμιότυπο του σχήματος της βάσης η appendSlideShow προσθέτει στην παρουσίαση μας μία νέα διαφάνεια με το όνομα της έκδοσης της βάσης και το στιγμιότυπο του σχήματός της.

Τέλος η κλάση **VideoGenerator** είναι υπεύθυνη για την δημιουργία αρχείων mp4. Για την υλοποίησή της χρησιμοποιήσαμε το πλήρες API της Xuggler.

```
public class VideoGenerator {
    private File selectedPresentation;
    private String filenameOfPresentation;
    private  final double FRAME_RATE = 120;
    private  String outputFilename ;
    private String sourceFolder=new String();

        public VideoGenerator(File sp){
            selectedPresentation=sp;
             filenameOfPresentation= selectedPresentation.getAbsolutePath();
            setOutputFilename();
            setSourceFolder();
        }


        public void exportToVideo() throws IOException{
                extractPngFromPptx();
                createVideo();
                deleteGeneratedPng();
        }


        private void extractPngFromPptx(){
                try {
```

```
                PPTX2PNG.main(new String[]{filenameOfPresentation});
        } catch (Exception e) {
                e.printStackTrace();
        }
}


private void createVideo() throws IOException{

   IMediaWriter writer = ToolFactory.makeWriter(outputFilename);
   Dimension screenBounds = Toolkit.getDefaultToolkit().getScreenSize();

    writer.addVideoStream(0, 0, Icodec.ID.CODEC_ID_MPEG4,
    screenBounds.width,screenBounds.height);
    long startTime = System.nanoTime();

    File directory = new File(sourceFolder);
    ArrayList<File> filenames = new ArrayList<File> ();

    File[] list= directory.listFiles(new PNGFileFilter());
        for(int i=0;i<list.length;i++){
                filenames.add(list[i]);

    Collections.sort(filenames,new FilenameSorter());

    for (int index = 0; index < filenames.size(); index++) {
        BufferedImage screen = ImageIO.read(filenames.get(index))

        BufferedImage bgrScreen = convertToType
        (screen,BufferedImage.TYPE_3BYTE_BGR);

        int i=0;
        while(i<FRAME_RATE){
            writer.encodeVideo(0, bgrScreen, System.nanoTime() - startTime,
            TimeUnit.NANOSECONDS);
            ++i;
        }
    }
    writer.close();
```

```
}

private void deleteGeneratedPng(){
        File directory = new File(sourceFolder);

        FileFilter filter =new PNGFileFilter();
        File[] filesToBeDeleted =directory.listFiles(filter);

        for(int i=0;i<filesToBeDeleted.length;++i)
                filesToBeDeleted[i].delete();
}

private void setOutputFilename(){
        String[] leftArray = filenameOfPresentation.split("\\.",2);
        outputFilename=leftArray[0]+".mp4";
}

private void setSourceFolder() {
        String[] leftArray = filenameOfPresentation.split("\\.",2);
        String[] leftovers = leftArray[0].split("\\\\");

        for(int i=0;i<leftovers.length-1;++i)
                sourceFolder+=leftovers[i]+"\\";
}

private BufferedImage convertToType(BufferedImage sourceImage,int targetType) {
        BufferedImage image;

         if (sourceImage.getType() == targetType)
             image = sourceImage;
         else {
            image = new BufferedImage(sourceImage.getWidth(),
            sourceImage.getHeight(), targetType);
            image.getGraphics().drawImage(sourceImage, 0, 0, null);
        }

        return image;
}
```

```
class FilenameSorter implements java.util.Comparator<File> {
	public int compare(File c1,File c2) {
		if(getFname(c1) > getFname(c2))
			return 1;
		else if(getFname(c1) < getFname(c2))
			return -1;
		else
			return 0;
	}

	public int getFname(File file){
		String[] leftArray = file.getAbsolutePath().split("\\.",2);
		String[] leftovers = leftArray[0].split("\\-",2);
		return Integer.parseInt(leftovers[1]);
	}
}
```

Για την μετατροπή μίας παρουσίασης PowerPoint σε mp4 χρησιμοποιούμε αρχικά το API της Apache-POI, για την διάσπαση του αρχείου παρουσίασης σε αρχεία png και στην συνέχεια χρησιμοποιούμε το API της Xuggler για την κωδικοποίηση των αρχείων εικόνας σε Frames ενός αρχείου mp4.

Για να αρχικοποιηθεί ένα αντικείμενο της κλάσης **VideoGenerator** αρκεί ένα αρχείο PowerPoint απο το οποίο εξάγουμε πληροφορία για το όνομα του αρχείου εξόδου και το μονοπάτι κάτω απο το οποίο πρέπει να δημιουργηθεί.

Η μέθοδος για την δημιουργία video είναι η *exportToVideo()* η οποία χάρη της απλότητας καλεί τρείς βοηθητικές μεθόδους.

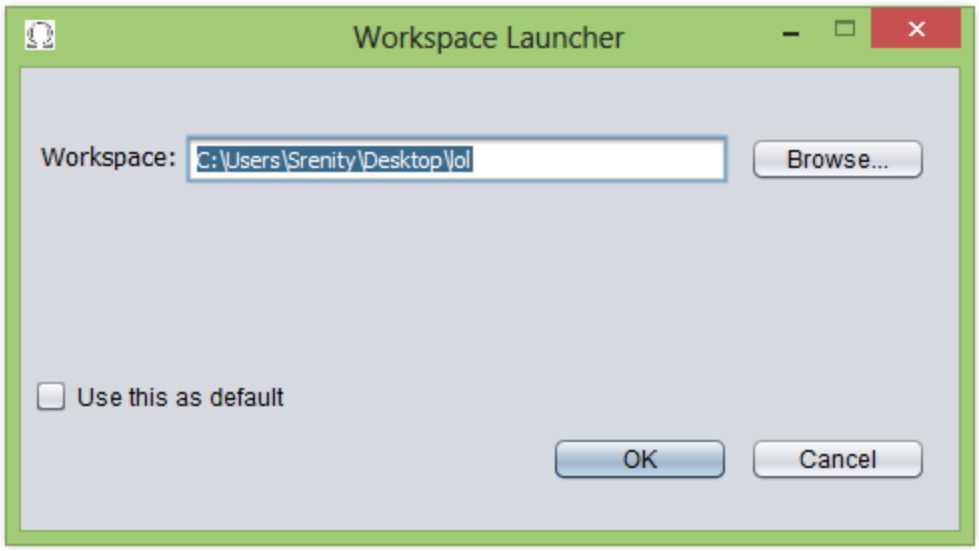
- *extractPngFromPptx()*, η οποία καλεί την εφαρμογή `PPTX2PNG` της εξωτερική βιβλιοθήκης Apache-POI και μετατρέπει κάθε slide της παρουσίασης σε αρχείο εικόνας png.
- *createVideo()*, η οποία είναι η βασική μέθοδος που κωδικοποιεί τα αρχεία εικόνας στα Frames του αρχείου video. Φτιάχνουμε ένα αντικείμενο τύπου **IMEdiaWriter** αρχικοποιώντας το με το το όνομα αρχείου εξόδου. Στην συνέχεια συγκεντρώνουμε το σύνολο των αρχείων εικόνας, τα ταξινομούμε και τα φορτώνουμε στην κύρια μνήμη του συστήματος. Για κάθε εικόνα κωδικοποιούμε τα αντίστοιχο Frame τόσες φορές όσες και το Frame_Rate, εδώ έχουμε επιλέξει ενδεικτικά της τιμή 120. Τέλος βάζουμε το επίμετρο στο αρχείο mp4.

- *deleteGeneratedPng()*, η οποία διαγράφει τα αρχεία png τα οποία δημιουργήθηκαν κατα την διάσπαση του αρχείου παρουσίασης PowerPoint.

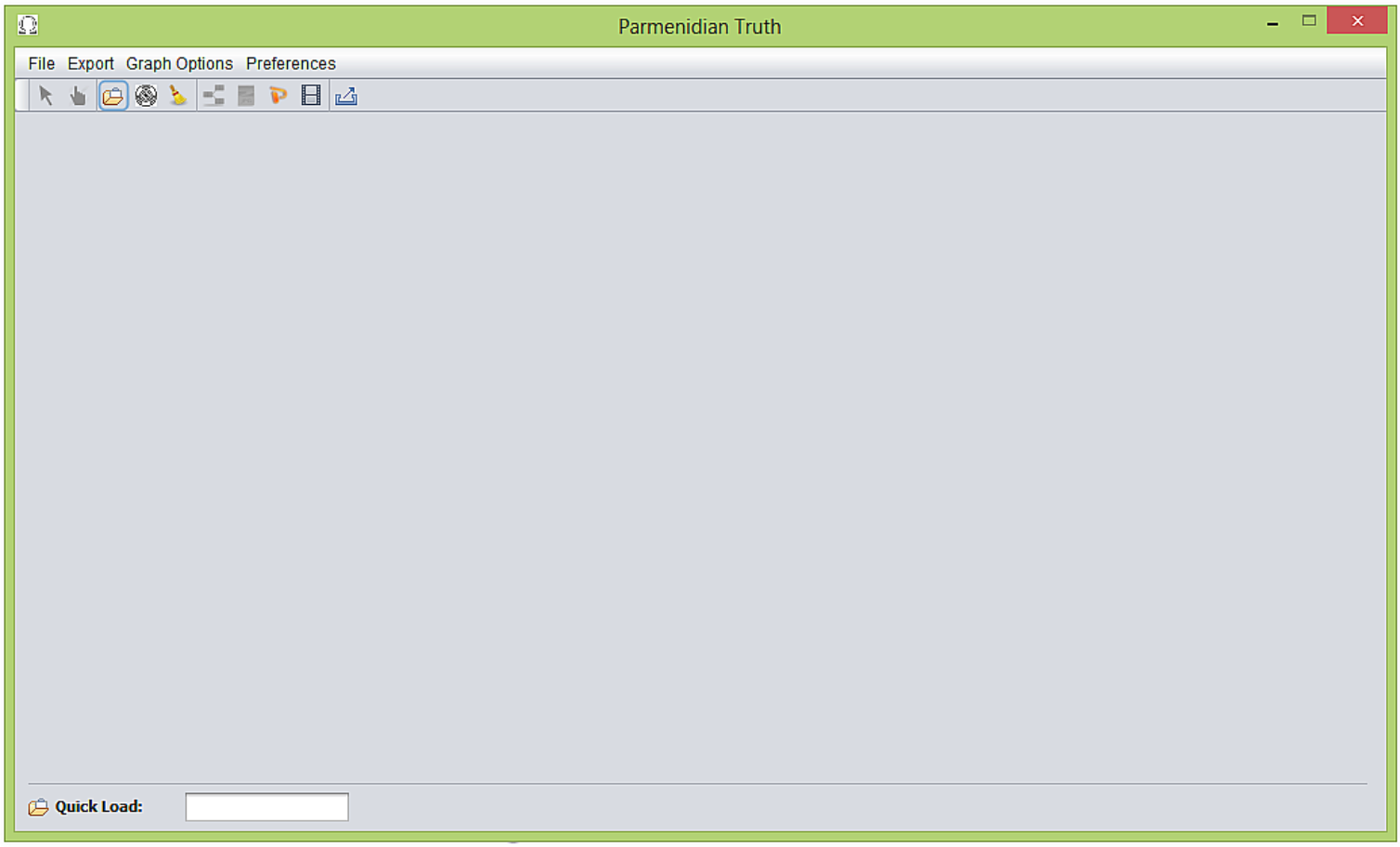
5 Οδηγός Χρήσης

5.1 Αναλυτική παρουσίαση

Σ'αυτό το κεφάλαιο θα παρουσιάσουμε τον τρόπο χρήσης του συστήματος. Με αναλυτικό τρόπο θα δείξουμε τα αποτελέσματα που παράγει το εργαλείο για κάθε επιλογή διαθέσιμη στον χρήστη. Στο τέλος της ενότητας θα παρουσιάσουμε ένα πίνακα με τους χρόνους του συστήματος για διάφορα σύνολα δεδομένων εισόδου.

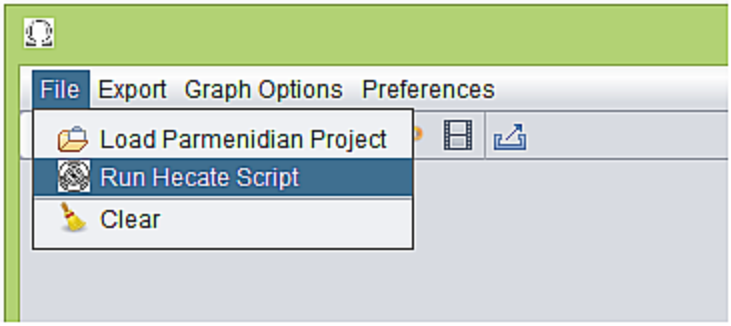


Σε αυτό το σημείο ζητείται απο τον χρήστη να δεσμέυσει χώρο στον δίσκο για τον χώρο εργασίας του συστήματος. Αν και τετριμμένη, αυτή η διαδικασία είναι πολύ σημαντική διότι στο εργαλείο εισάγεται σημαντική πληροφορία όσον αφορά το πλήρες όνομα των αρχείων εξόδου. Η επιλογή μπορεί να γίνει με την επιλογή 'Browse...' ή, για μεγαλύτερη ευκολία ως προς τον χρήστη, με drag-and-drop του επιθυμητου φακέλου στο textfield.

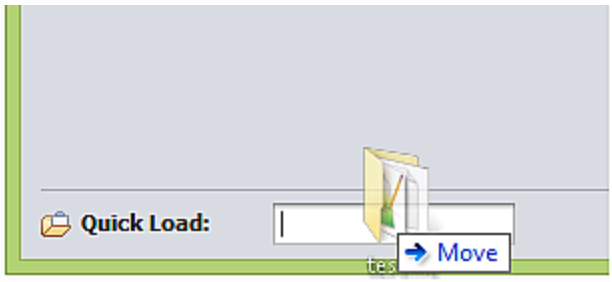


Αυτό είναι το κύριο μενού του εργαλείου, προσφέρεται μια toolbar για την πιο γρήγορη χρήση των επιλογών του. Έχουμε "προστατεύσει" τον χρήστη απενεργοποιώντας τα εικονίδια τα οποία δεν γίνεται να χρησιμοποιηθούν την συγκεκριμένη στιγμή.

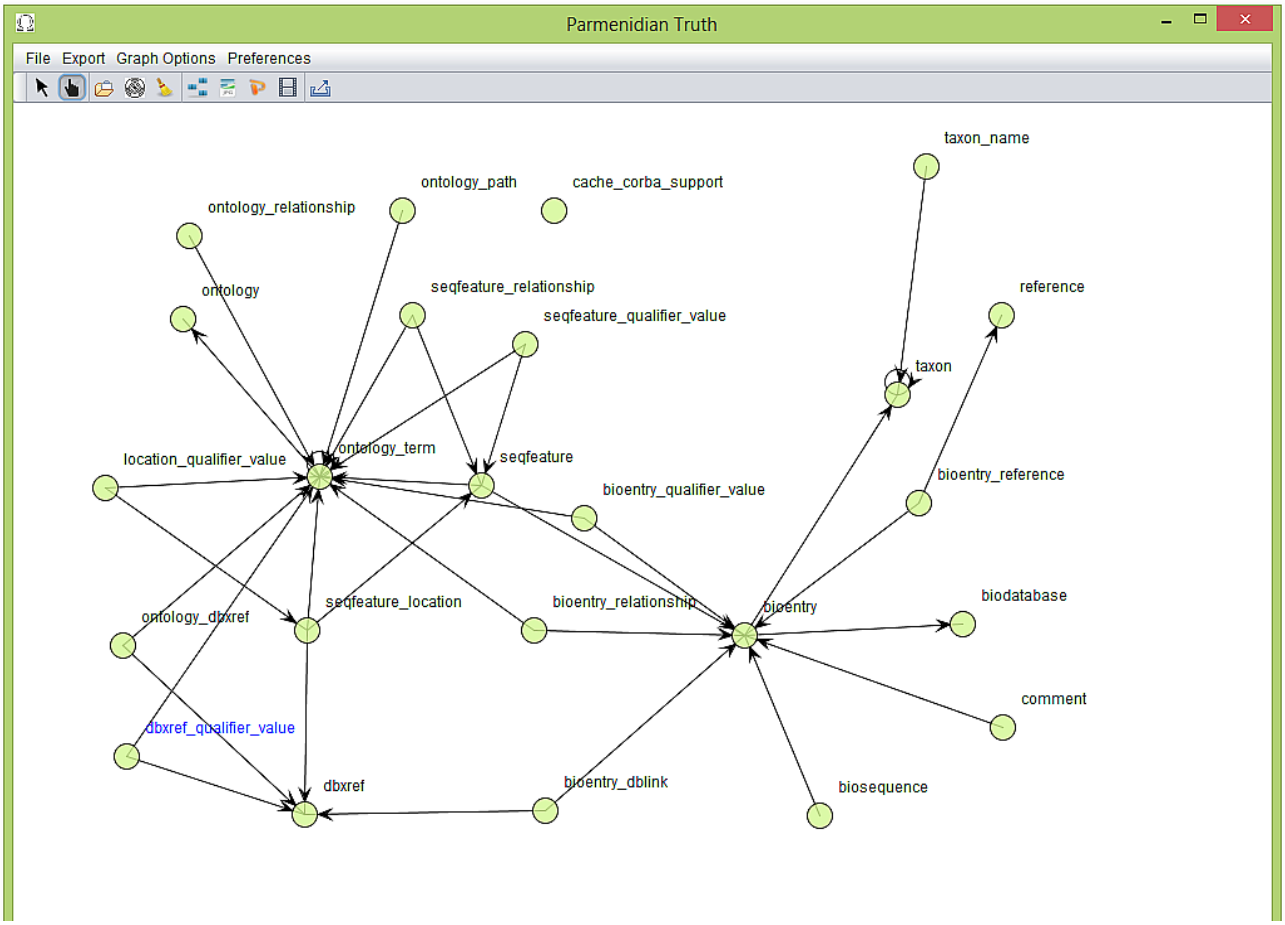
Για την δημιουργία του αρχείου transitions.xml, του οποίου την σπουδαιότητα έχουμε αναλύσει πιο πάνω, αρκεί να επιλέξουμε **File** – **Run Hecate Script** και στην συνέχεια να επιλέξουμε το dataset με τα σχήματα της βάσης.



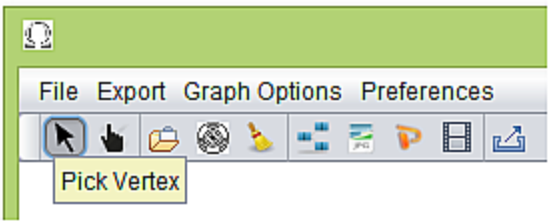
Έχοντας ένα συμβατό αρχείο με το σύστημά μας μπορουμε να το φορτώσουμε στην κύρια μνήμη για την προβολή του *Diachronic Graph.* Αυτό μπορεί να γίνει με πολλους, είτε απο το μενού **File** όπως φαίνεται παραπάνω, είτε απο την toolbar ή, όπως και στην επιλογή χώρου εργασίας, μπορούμε να φορτώσουμε το αρχείο εισόδου κάνοντας drag-and-drop στο textfield "**Quick Load**"



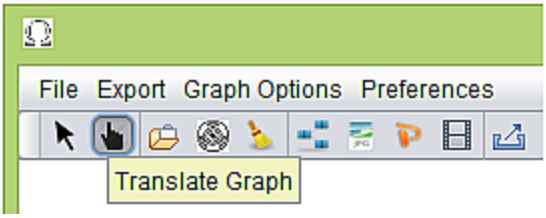
Μετά απο την επιτυχή φόρτωση των αρχείων εισόδου το κύριο μενού του συστήματος παίρνει την μορφή



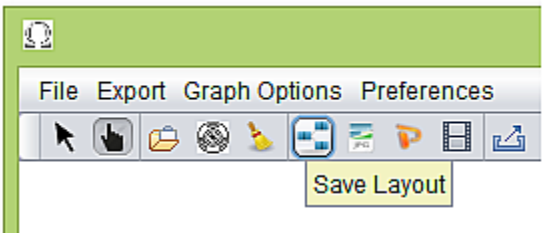
Σε αυτήν την οθόνη ο χρήστης μπορεί να διαμορφώσει τον γράφο όπως αυτός θέλει μετατοπίζοντας τους κόμβους



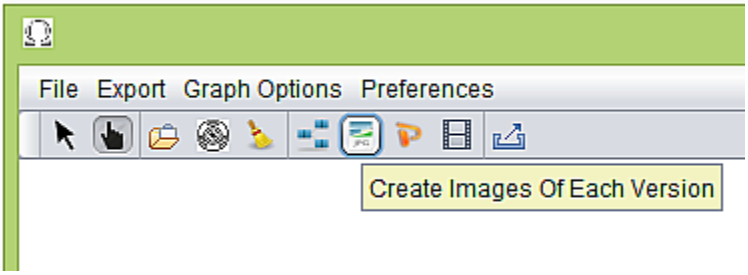
ή και το ίδιο το γράφημα



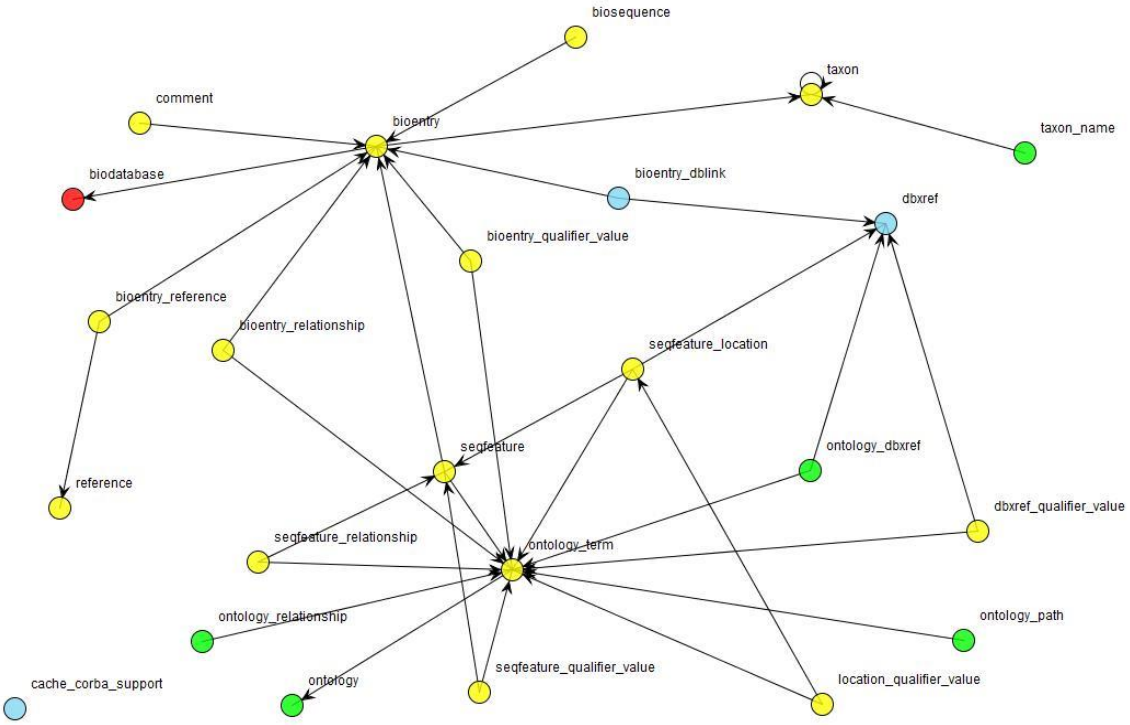
Αφου ο χρήστης τελειώσει με την μορφοποίηση του γραφήματος μπορεί να αποθηκεύσει τις συντεταγμένες του κάθε κόμβου



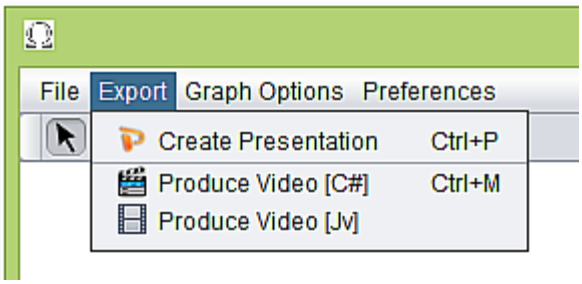
ή να παράξει τα αρχεία εικόνας για κάθε έκδοση της βάσης



Το αρχείο εξόδου θα είναι έχει default χρώμα για κάθε κόμβο το μπλε, πράσινο άν ο κόμβος δημιουργήθηκε στην τρέχουσα έκδοση, κίτρινο άν ο κόμβος ενημερώθηκε στην τρέχουσα έκδοση και τέλος κόκκινο αν ο κόμβος θα διαγραφεί στην επόμενη έκδοση.



Τέλος η παραγωγή της παρουσίασης PowerPoint βρίσκεται στο μενού **Export – Create Presentation**, ή διαφορετικά απο την toolbar,όπου ο χρήστης πρέπει να επιλέξει το φάκελο με τα παραγμένα αρχεία εικόνας της βάσης ανα έκδοση. Όσο για την παραγωγή αρχείων video οι επιλογές βρίσκονται στο ίδιο μενού, αρκεί ο χρήστης να επιλέξει την παρουσίαση PowerPoint που θέλει να μετατρέψει σε αρχείο video



Παρακάτω προβάλλεται ένας ενδεικτικός πίνακας με τις επιδώσεις του συστήματος σε διαφορετικά αρχεία εισόδου.

Datasets	Number Of SQL files per dataset	Duration of dataset loading	Duration of image Production	Duration of pptx Production	Duration of mp4 Production
demo	5	4.4 sec	3.4 sec	6.5 sec	62.5 sec
Biosql	46	8.6 sec	24.7 sec	14.5 sec	367.2 sec
Coppermine	69	6.9 sec	32.8 sec	17 sec	505.6 sec
Atlas	86	21.8 sec	93.2 sec	23.5 sec	684.5 sec
Ensembl	529	85.8 sec	469.3 sec	81.3 sec	3301 sec

Ενδιαφέρον παρουσιάζει η περίπτωση του σχήματος της βάσης Coppermine, η οποία αν και έχει μεγαλύτερο αριθμό αρχείων SQL χρειάζεται λιγότερο χρόνο για την φόρτωση της στο σύστημα, ο λόγος είναι η πλήρης απουσία ξένων-κλειδίων στα σχήματα των εκδόσεων της βάσης.

Ο χρόνος δημιουργίας των αρχείων mp4 εξαρτάται άμεσα απο το frame rate που επιλέχθηκε, στις παραπάνω περιπτώσεις είναι ίσο με 120

6 Επίλογος

6.1 Σύνοψη και συμπεράσματα

Το σχήμα μιας βάσης δεδομένων απο την στιγμή της δημιουργίας της μέχρι και το τέλος της αλλάζει συνεχώς. Πίνακες προστίθενται και αφαιρούνται. Ακομα και οι ίδιοι οι πίνακες δεν παραμένουν σταθεροί αλλά τα πεδία τους αφαιρούνται, νέα προστίθενται ή ακόμα αλλάζουν τύπο. Σκοπός αυτής της πτυχιακής εργασίας ήταν η ανάπτυξη ενός εργαλείου για την οπτικοποίηση αυτών των αλλαγών ούτως ώστε να γίνονται εύκολα αντιληπτές απο τον χρήστη. Υλοποιήσαμε ένα εργαλείο που

- Λαμβάνει ως είσοδο το βίο της βάσης δεδομένων και τις αλλαγές που συντελούνται σε κάθε μετάβαση
- Εντοπίζει όλους τους πίνακες που συμμετέχουν στη ζωή μιας βάσης δεδομένων, σε όλες τις εκδόσεις τους καθώς και τα ξένα κλειδιά που τους συνδέουν
- Δημιουργεί τον διαχρονικό γράφο της βάσης, δηλαδή ένα γράφημα με την ένωση των πινάκων και των ξένων-κλειδιών τους,
- Τοποθετεί σε μια διδιάσταση επιφάνεια το εν λόγω σχήμα, δίνοντάς του συντεταγμένες
- Για κάθε μία από τις versions του πίνακα, ζωγραφίζει την εικόνα της, με βάση τις παραπάνω συντεταγμένες
- Τοποθετεί τις εικόνες σε μια παρουσίαση Microsoft PowerPoint
- Τέλος, δημιουργεί αρχεία τύπου wmv και mp4 της παραπάνω παρουσίασης

6.2 Μελλοντικές επεκτάσεις

Πληθώρα επεκτάσεων μπορεί να γίνει στο κομμάτι της οπτικοποίησης του γράφου στον χρήστη. Θα μπορούσαμε να προβάλλουμε σε κάθε slide της παρουσίασης το *Diachronic Graph* , με τους κόμβους που δεν συμμετέχουν στην παρούσα έκδοση να είναι faded out και αυτά που έχουν διαγραφεί σε βαθύτερο fade out. Επίσης θα μπορούσαμε να αφαιρέσουμε τα ονόματα των πινάκων οι οποίοι έχουν μείνει αμετάβλητοι για έναν αποδεκτό αριθμό εκδόσεων, και να εμφανίζονται μόνο τα ονόματα των κόμβων που υπόκεινται σε κάποια αλλαγή στην τρέχουσα έκδοση. Με αυτόν τον τρόπο αφαιρούμε άχρηστη πληροφορία απο τον χρήστη, επικεντρώνοντάς τον στις αλλαγές της βάσης. Τέλος θα μπορούσαμε να διαμορφώσουμε τον τρόπο που οπτικοποιείται ο *Diachronic Graph* αποδίδοντας διαφορετικά τους κόμβους οι οποίοι έχουν το μεγαλύτερο fan-in ή fan-out ή και πίνακες που χαρακτηρίζονται από κάποιες στατιστικές ιδιότητες, όπως παραδείγματος χάρη τον πιο σταθερό κόμβο, τον πιό ευμετάβλητο κ.α.

7 Βιβλιογραφία

I. Σκουλής. Hecate: SQL Schema diff Viewer. Πτυχιακή Εργασία, Τμήμα Πληροφορικής, Π. Ιωαννίνων, 2010

[POI]. A Java API for Microsoft Documents. Apache. http://poi.apache.org/

[JUNG]. A software library that provides a common and extendible language for the modeling, analysis, and visualization of data that can be represented as a graph or network. JUNG. http://jung.sourceforge.net/

[Xuggler]. An easy way to uncompress, modify, and re-compress any media file (or stream) from Java. Xuggle. http://www.xuggle.com/