ΣΥΝΔΕΣΗ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΔΙΑΔΙΚΑΣΙΩΝ BPMN ΓΙΑ ΤΗΝ ΕΠΙΤΥΞΗ ΕΚΤΕΛΕΣΙΜΩΝ ΜΟΝΤΕΛΩΝ

ΠΑΠΑΙΩΑΝΝΟΥ ΣΩΤΗΡΙΟΣ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ, ΒΟΛΟΣ, ΕΛΛΑΔΑ
sopapaio@uth.gr

Περίληψη:Περιγράφουμε μια *επίσημα καλά θεμελιωμένη* προσέγγιση για τη σύνδεση των δεδομένων και των διαδικασιών εννοιολογικά, βασιζόμενοι στην υιοθέτηση των διαγραμμάτων UML για την αναπαράσταση δεδομένων, και των BPMN για να την αναπαράσταση των διαδικασιών. Τα διαγράμματα κλάσης UML μαζί με ένα σύνολο πρόσθετων μεταβλητών της διαδικασίας, που ονομάζεται Artifact, αποτελούν το μοντέλο πληροφοριών της διαδικασίας. Όλες οι δραστηριότητες των BPMN διαδικασιών αναφέρονται σε ένα τέτοιο μοντέλο πληροφοριών μέσω της λειτουργίας OCL. Δείχνουμε ότι η προκύπτουσα σημασιολογία ενώ είναι αφηρημένη είναι πλήρως εκτελέσιμη. Παρέχουμε επίσης μια εφαρμογή της εκτέλεσης.

Λέξεις-κλειδιά: *BPMN · UML · Διαδικασίες Artifact-centric · Διαδικασίες Data-aware (δεδομένων-ενημερώσεων)*

1. Εισαγωγή

Τα δύο κύρια προτερήματα οποιουδήποτε οργανισμού είναι (i)οι πληροφορίες, δηλ. Τα δεδομένα, τα οποία είναι τα πράγματα που γνωρίζει ο οργανισμός και (ii) οι διαδικασίες, που είναι συλλογές

δραστηριοτήτων που περιγράφουν τον τρόπο με τον οποίο εκτελείται η εργασία μέσα σε μια οργάνωση.

Προφανώς υπάρχει η ανάγκη να εκπροσωπούνται και να καθίσταται σαφείς και ακριβείς το περιεχόμενο αυτών των δύο πολύτιμων στοιχείων. Αυτό έχει οδηγήσει σε εννοιολογικά μοντέλα για τα δεδομένα,όπως είναι τα διαγράμματα της κατηγορίας UML [1], και σε εννοιολογικά μοντέλα για τις διαδικασίες, όπως ειναι τα BPMN [2,3].Δυστυχώς αυτά τα εννοιολογικά μοντέλα είναι σπάνια τυπικά [4,5]. Στην πραγματικότητα, αναπτύσσονται συνήθως από διαφορετικές ομάδες, την ομάδα διαχείρισης δεδομένων και την ομάδα διαχείρισης των διαδικασίας, αντίστοιχα,όπου χρησιμοποιούν τα δικά τους μοντέλα και μεθοδολογίες. Αυτό οδηγεί στην ανάπτυξη δύο ανεξάρτητων και άσχετων σχεδίων και μορφοποιήσεων,ένα που αφορά τα δεδομένα και ένα τις διαδικασίες,ενώ η αλληλεπίδραση μεταξύ των δύο παραμελείται [6,7].Επιπλέον, όταν φτάνουμε σε εργαλεία προσομοίωσης διαδικασιών, παρακολούθησης και εκτέλεσης, οι δύο πτυχές πρέπει να ενωθούν, και όντως όλα τα εργαλεία, όπως το Bizagi Studio ή Signavio, παρέχουν έναν τυπικό ιδιόκτητο τρόπο για να πραγματοποιηθεί η σύνδεση.

Ωστόσο, μια τέτοια σύνδεση γίνεται ουσιαστικά προγραμματικά, καθορίζοντας ένα εσωτερικό μοντέλο δεδομένων και συνδέοντάς το με τις κατασκευές BPMN στην διαδικασία μέσω κατάλληλων επιχειρηματικών κανόνων που εκφράζονται ως πραγματικός κώδικας (π.χ.στην Java) για να διευκρινίσει τι συμβαίνει με τα δεδομένα και πώς αυτά ανταλλάσσονται μετους χρήστες και άλλες διαδικασίες. Δυστυχώς, αυτός ο τρόπος σύνδεσης δεδομένων και διαδικασιών γίνεται προγραμματικά, αλλά όχι εννοιολογικά.

Πρόσφατη έρευνα παρουσιάζει την ανάγκη να ληφθούν υπόψη και τα δύο,δεδομένα και διαδικασίες ως πολίτες πρώτης κατηγορίας στον σχεδιασμό διαδικασιών και υπηρεσιών [7-9]. Συγκεκριμένα,οι αποκαλούμενες artifact-centric προσεγγίσεις, οι οποίες υποστηρίζουν ένα είδος μέσης μεταξύ μιας εννοιολογικής μορφοποίησης των δυναμικών συστημάτων και της πραγματικής τους εφαρμογή, υπόσχονται να είναι αρκετά αποτελεσματικοί στην πράξη [6,10,11].

Σε αυτή την εργασία, εμπνευσμένη από τις artifact-centric προσεγγίσεις, θεωρούμε την περίπτωση στην οποία τα δεδομένα του τομέα ενδιαφέροντος μιας δοσμένης διαδικασίας αντιπροσωπεύονται εννοιολογικά χρησιμοποιώντας ένα διάγραμμα κλάσης UML, ενώ η ίδια η διαδικασία περιγράφεται στο BPMN. Εγκρίνουμε τις κατηγορίες διαγραμμάτων UML και BPMN, καθώς είναι οι πρότυπες και οι ποίο συνηθισμένες διατυπώσεις για την εννοιολογική αναπαράσταση των δεδομένων στην μηχανική λογισμικού και στις διαδικασίες στο BPM, αντίστοιχα. Με αυτόν τον τρόπο, δεν προτείνουμε ακόμα ανομοιογένεια, αλλά συνδυάζουμε τα κλασικά με ένα νέο ολοκληρωμένο τρόπο σύνδεσης των δεδομένων και διαδικασιών.Άλλες γλώσσες μπορούν να επιλεγούν καθώς και εφόσον έχουν αναμφισβήτητη σημασιολογία, π.χ. ORM / ER-διαγράμματα για τον ορισμό των δεδομένων, ή διαγράμματα δραστηριότητας UML, όπως χρησιμοποιείται για παράδειγμα στο [12], για τον καθορισμό της διαδικασίας.

Η βασική ιδέα στην οποία βασίζεται η πρότασή μας είναι ότι, προκειμένου να συνδεθούν οι δύο διατυπώσεις,προτείνουμε επίσης: (1) την έννοια του Artifact,η οποία λειτουργεί ως συλλογή μεταβλητών διεργασίας που πρέπει να συσχετιστεί με μια παρουσία διεργασίας, και (2) τον προσδιορισμό του τρόπου με τον οποίο οι δραστηριότητες της διαδικασίας αναφέρονται και ενημερώνουν τις μεταβλητές του Artifacth η τα δεδομένα του τομέα. Και οι δύο έννοιες μπορούν να προσδιοριστούν μέσω τυποποιημένων γλωσσών που προσαρμόζονται κατάλληλα στα διαγράμματα

UML και BPMN. Πράγματι, το Artifact μπορεί να εκπροσωπείται ως μια νέα τάξη του διαγράμματος κλάσης UML με τα βολικά χαρακτηριστικά του και τους συσχετισμούς του με τις υπόλοιπες τάξεις του UML, και οι δραστηριότητες των διαδικασίων μπορούν να προσδιοριστούν μέσω της λειτουργίας OCL. Και πάλι, άλλες γλώσσες μπορεί να επιλεγούν για τη σύνδεση,αλλά το κρίσιμο σημείο εδώ είναι να επιλέξουμε μια γλώσσα του οποίου η εκφραστικότητα είναι ουσιαστικά λογική πρώτης τάξης (δηλαδή, σχεσιακή άλγεβρα), όπως συμβαίνει με τις εκφράσεις OCL που χρησιμοποιούνται ως επί το πλείστον [13].

Με αυτό τον τρόπο, η εκτελεστότητα του συνολικού πλαισίου μπορεί να βασιστεί στη σχεσιακή τεχνολογία SQL, δεδομένου ότι τα δεδομένα για εισαγωγή / διαγραφή / επιστροφή από κάθε δραστηριότητα μπορούν να χαρακτηριστούν μέσω μιας ερώτησης σχεσιακής άλγεβρας και, συνεπώς, μιας δήλωσης SQL. Συγκεκριμένα, το διάγραμμα κλάσης UML κωδικοποιείται ως σχεσιακή βάση δεδομένων, το διάγραμμα BPMN ως δίκτυο Petri και οι συμβάσεις OCL ως λογικοί κανόνες που εξάγουν τις δηλώσεις SQL που πρέπει να εφαρμοστούν στη βάση δεδομένων όταν εκτελείται μια δραστηριότητα. Ως απόδειξη της έννοιας, έχουμε αναπτύξει ένα πρωτότυπο, γραμμένο σε Java, η οποία επιτρέπει τη φόρτωση κατά τη μεταγλώττιση όλων των μοντέλων στο πλαίσιο μας και στη συνέχεια εκτελεί τις λειτουργίες τους σε χρόνο εκτέλεσης σε μια σχεσιακή βάση δεδομένων.

2. Προκαταρκτικά

Διαγράμματα κλάσης UML και τα στιγμιότυπα τους. Ένα διάγραμμα κατηγορίας UML [1] αποτελείται από μια ιεραρχία των τάξεων, τις n-ary ενώσεις μεταξύ αυτών των τάξεων (όπου μερικές από αυτές μπορούν να γίνουν πειστικές, δηλαδή, κλάσεις συσχέτισης) και τα χαρακτηριστικά μέσα σε αυτές τις τάξεις.Επιπλέον, ένα σχήμα UML μπορεί να σχολιαστεί με ελάχιστους/μέγιστους περιορισμούς πολλαπλότητας σε σχέση με τα άκρα / χαρακτηριστικά σύνδεσης, και περιορισμούς ιεραρχίας (δηλ., διασπασμένοι / πλήρεις περιορισμοί). Σε αυτή την εργασία, χρησιμοποιούμε τη σημείωση C <_C' για να παραπέμψουμε ότι το C είναι μια υποκλάση του C'. Υιοθετούμε μια εννοιολογική προοπτική (σε αντίθεση με την προοπτική του λογισμικού) των διαγραμμάτων τάξης UML, ως χαρακτηριστικό της φάσης ανάλυσης της διαδικασίας ανάπτυξης [14].

Επιπλέον, για λόγους ευκολίας, υποθέτουμε ότι το διάγραμμα κατηγορίας UML περιέχει μόνο εκείνα τα χαρακτηριστικά που μπορούν να αντιστοιχιστούν σε πίνακες SQL με περιορισμούς πρωτεύοντος / ξένου κλειδιού.Για παράδειγμα, εκφράζουμε στο διάγραμμα προαιρετικό / υποχρεωτικό (ελάχιστη πολλαπλότητα 0 ή 1), μεμονωμένες / πολλαπλές ιδιότητες (μέγιστη πολλαπλότητα 1 ή *), αλλά όχι, π.χ., πολλαπλότητα min / max 3. Όλες οι άλλες εκφράσεις υποτίθεται ότι γράφονται και αντιμετωπίζονται ως περιορισμοί OCL (βλ. Παρακάτω). Μια παράσταση διαγράμματος κλάσης UML είναι ένα σύνολο αντικειμένων και σχέσεων μεταξύ τέτοιων αντικειμένων. Κάθε αντικείμενο ταξινομείται ως παράδειγμα μίας ή περισσότερων κατηγοριών UML και κάθε σχέση ως παράδειγμα μιας σύνδεσης UML. Υποθέτουμε ότι, κάθε φορά που ένα αντικείμενο λ ταξινομείται ως ένα παράδειγμα του C, και το C<_C', τότε, το λ ταξινομείται επίσης ως παράδειγμα του C'. Σημειώστε ότι αυτή η διαδικασία ολοκλήρωσης των ταξινομήσεων ενός αντικειμένου μπορεί να υπολογιστεί αυτόματα μέσω μιας κυνηγημένης ιεραρχίας της κλάσης UML. Αυτός ο αυτόματος μηχανισμός ονομάζεται ISA closure.

OCL. Το OCL [15] είναι μια γλώσσα κειμένου για τον ορισμό ερωτημάτων μέσω ενός σχήματος UML, το αποτέλεσμα του οποίου εξαρτάται από το περιεχόμενο της παρουσίας του UML. Συγκεκριμένα, οι εκφράσεις boolean OCL χρησιμοποιούνται ευρέως για να ορίσουν: (1) περιορισμούς ακεραιότητας κειμένου που θα πρέπει να ικανοποιούνται από τις περιπτώσεις UML του σχήματος, (2) τις συμβάσεις λειτουργίας pre / postconditions, δηλαδή εκφράσεις που πρέπει να πληρούνται από τις περιπτώσεις UML ορισμένων σχημάτων πριν / μετά την εκτέλεση κάποιας λειτουργίας και (3) ερωτημάτων, καθορίζοντας την τιμή επιστροφής ορισμένων λειτουργιών. Οι εκφράσεις OCL συνδέονται συνήθως με μια συγκεκριμένη κλάση UML. Για παράδειγμα, η σύμβαση λειτουργίας OCL για μια συγκεκριμένη πράξη συνδέεται με την κλάση στην οποία ορίζεται η πράξη. Σε αυτή την περίπτωση, η έκφραση OCL self, αναφέρεται στο αντικείμενο στο οποίο γίνεται η κλήση (με παρόμοιο τρόπο με τη λέξη-κλειδί Java). Ομοίως, ένας περιορισμός OCL
συνδεδεμένο με κάποια κατηγορία C χρησιμοποιεί τον εαυτό του για να αναφερθεί σε οποιαδήποτε περίπτωση του C.

Η βασική ιδέα της OCL είναι η έννοια της πλοήγησης. Δεδομένου ότι μια έκφραση OCL αναφέρεται σε ένα αντικείμενο, όπως ο ίδιος, μπορούμε να πλοηγήσουμε σε αντικείμενα / τιμές που σχετίζονται με ένα τέτοιο αντικείμενο μέσω κάποιου συσχετισμού / χαρακτηριστικού χρησιμοποιώντας το όνομα του συσχετισμού-τέλος / χαρακτηριστικού που θέλουμε να διασχίσουμε. Για παράδειγμα, η έκφραση OCL self.album που συνδέεται με κάποια κλάση περιβάλλοντος καλλιτέχνη επιστρέφει τα άλμπουμ που σχετίζονται με τον συγκεκριμένο καλλιτέχνη που αναφέρεται από τον εαυτό. Μια πλοήγηση μπορεί επίσης να οριστεί ξεκινώντας από μια έκφραση OCL που αναφέρεται σε μια συλλογή αντικειμένων. Για παράδειγμα, η έκφραση OCL Artist.allInstances () αναφέρεται στο σύνολο όλων των αντικειμένων του Artist, έτσι η Artist.allInstances ().album επιστρέφει όλα τα άλμπουμ που μπορούν να ληφθούν από όλους τους καλλιτέχνες. Επιπλέον, λόγω αυτής της δυνατότητας πλοήγησης από τις συλλογές, η OCL επιτρέπει την αλυσίδα μιας πλοήγησης μετά την άλλη. Για παράδειγμα, το auto.album.track αναφέρεται σε όλα τα κομμάτια όλων των άλμπουμ ενός συγκεκριμένου καλλιτέχνη. Με αυτές τις πλοήγησης, η OCL προσφέρει αρκετούς φορείς εκμετάλλευσης OCL για να αποκτήσουν βασικές τιμές τύπου (όπως τιμές boolean ή ακέραιες) ή άλλες συλλογές από αυτές. Για παράδειγμα, το auto.album.track-> forAll (λ|λ.duration> 0) επιστρέφει αληθές αν όλες οι διάρκειες όλων των κομματιών λ κάποιου εαυτού καλλιτέχνη είναι μεγαλύτερες από 0.\

Υποθέτουμε σε αυτό το έγγραφο ότι όλες οι εκφράσεις του OCL γράφονται στο πρώτο τμήμα του OCL [13], δηλαδή το κομμάτι της OCL που μπορεί να θεωρηθεί ως πλήρως δηλωτικό και να κωδικοποιηθεί σε σχεσιακή άλγεβρα. Ουσιαστικά, εξαιρούνται οι λειτουργίες OCL που περιλαμβάνουν επανάληψη, κλείσιμο, βασικές λειτουργίες τύπου δεδομένων (όπως String concat) και τύπους δεδομένων OrderedSet και Bag.

BPMN. Το BPMN (Business Process Model and Notation) [3] είναι μια ευρέως χρησιμοποιούμενη και γνωστή γλώσσα ISO και OMG για τη μοντελοποίηση επιχειρηματικών διαδικασιών. Παρέχει μια γραφική και διαισθητική σημείωση που μπορεί εύκολα να γίνει κατανοητή από τους επιχειρηματίες, τους αναλυτές και τους προγραμματιστές. Με λίγα λόγια, η γλώσσα χρησιμοποιεί κόμβους για να αντιπροσωπεύει τις δραστηριότητες ή τα καθήκοντα της διαδικασίας, των οποίων η σειρά εκτέλεσης είναι καθορίζεται από ένα σύνολο κατευθυνόμενων άκρων. Διαφορετικοί κόμβοι πύλης είναι διαθέσιμοι για τον έλεγχο της ροής, για παράδειγμα για παράλληλες ή εναλλακτικές διαδρομές εκτέλεσης. Επιπλέον, με τη χρήση του BPMN είναι επίσης δυνατή η εκπροσώπηση της αλληλεπίδρασης μεταξύ διαφορετικών εμπλεκομένων στη διαδικασία, η ροή μηνυμάτων μεταξύ τους ή τα αντικείμενα που εμπλέκονται στη διαδικασία, για να αναφέρουμε μόνο μερικά παραδείγματα. Το διάγραμμα έχει σημασιολογική συμβολική μορφή. Καθώς πραγματοποιούνται οι διάφορες δραστηριότητες, το συμβολικό (ή οι μάρκες) ρέει μέσω του διαγράμματος επιτρέποντας την εκτέλεση των ακόλουθων δραστηριοτήτων. Λόγω αυτού, είναι δυνατό να επισημοποιηθεί ένα υποσύνολο της γλώσσας σε ένα δίχτυ Petri [16]. Αυτό έχει ως αποτέλεσμα την ακριβή εκτέλεση σημασιολογίας για το διάγραμμα BPMN.

3. Σύνδεση δεδομένων και μοντέλων BPMN

Παρουσιάζουμε την πρότασή μας για τη σύνδεση διαδικασιών και δεδομένων μέσω του ακόλουθου παραδείγματος. Όπως θα δούμε, το κύριο πλεονέκτημα της πρότασής μας είναι ότι, πέραν των πλεονεκτημάτων μιας τεκμηριωμένης προσέγγισης που μας επιτρέπει να αντιπροσωπεύουμε τόσο τα δομικά (δηλαδή τα δεδομένα) όσο και τις δυναμικές (δηλαδή τις δραστηριότητες ή τα καθήκοντα) διαστάσεις της διαδικασίας, τα μοντέλα μας παρέχουν αρκετές πληροφορίες για την επίτευξη την αυτόματη εκτελεστότητά τους.

Παράδειγμα. Στόχος μας είναι η πραγματοποίηση μιας διαδικασίας δημιουργίας λιστών αναπαραγωγής από κομμάτια μουσικών άλμπουμ. Ειδικότερα, η διαδικασία θα πρέπει να ασχολείται με τα ακόλουθα δεδομένα και ροή διεργασιών:

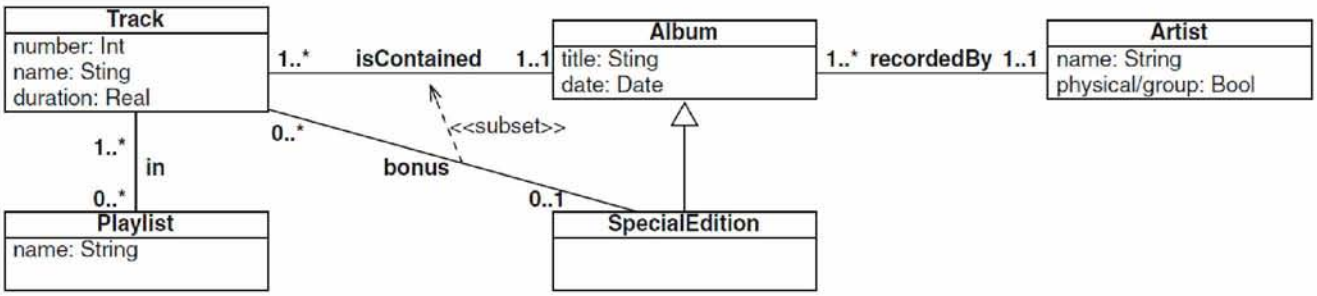
–**Δεδομένα**: Κάθε άλμπουμ έχει τίτλο, ημερομηνία πρώτης προβολής και έναν συγκεκριμένο καλλιτέχνη. Ένας καλλιτέχνης έχει ένα όνομα και είναι είτε ένα φυσικό πρόσωπο ή μια ομάδα. Κάθε καλλιτέχνης έχει τουλάχιστον ένα λεύκωμα. Τα άλμπουμ περιέχουν ένα ή περισσότερα κομμάτια. Κάθε κομμάτι έχει έναν αριθμό, ένα όνομα και μια διάρκεια και ανήκει σε ένα ακριβώς άλμπουμ. Ορισμένα άλμπουμ είναι ειδικές εκδόσεις και, σε αυτή την περίπτωση, μπορεί να περιέχουν κομμάτια bonus. Οι λίστες αναπαραγωγής έχουν ένα όνομα και περιέχουν ένα μη εξαντλημένο σύνολο κομματιών (για απλότητα η σειρά δεν παρουσιάζει ενδιαφέρον).

–**Ροή διεργασιών**: Διαδοχικά, η διαδικασία ζητά από το χρήστη το όνομα ενός καλλιτέχνη και συνεχίζει με δύο παράλληλους κλάδους. Ο πρώτος υπολογίζει και επιστρέφει στον χρήστη το σύνολο των κομματιών που αποτελούν μέρος μιας ειδικής έκδοσης που έχει καταγραφεί από τον καλλιτέχνη. στη συνέχεια, ζητά από τον χρήστη να επιλέξει ένα υποσύνολο αυτών των κομματιών και να δημιουργήσει μια λίστα αναπαραγωγής μαζί τους. Στη δεύτερη, η διαδικασία αποκτά το σύνολο των λιστών αναπαραγωγής που περιέχουν ένα κομμάτι από τον επιλεγμένο καλλιτέχνη. Στο τέλος των δύο κλάδων, το σύνολο των κομματιών στη νέα λίστα αναπαραγωγής επιστρέφεται στον χρήστη. Μετά από αυτό, ο χρήστης αποφασίζει εάν επιθυμεί να συνεχίσει να προσθέτει λίστες αναπαραγωγής στο σύστημα ή τερματισμό της διαδικασίας.

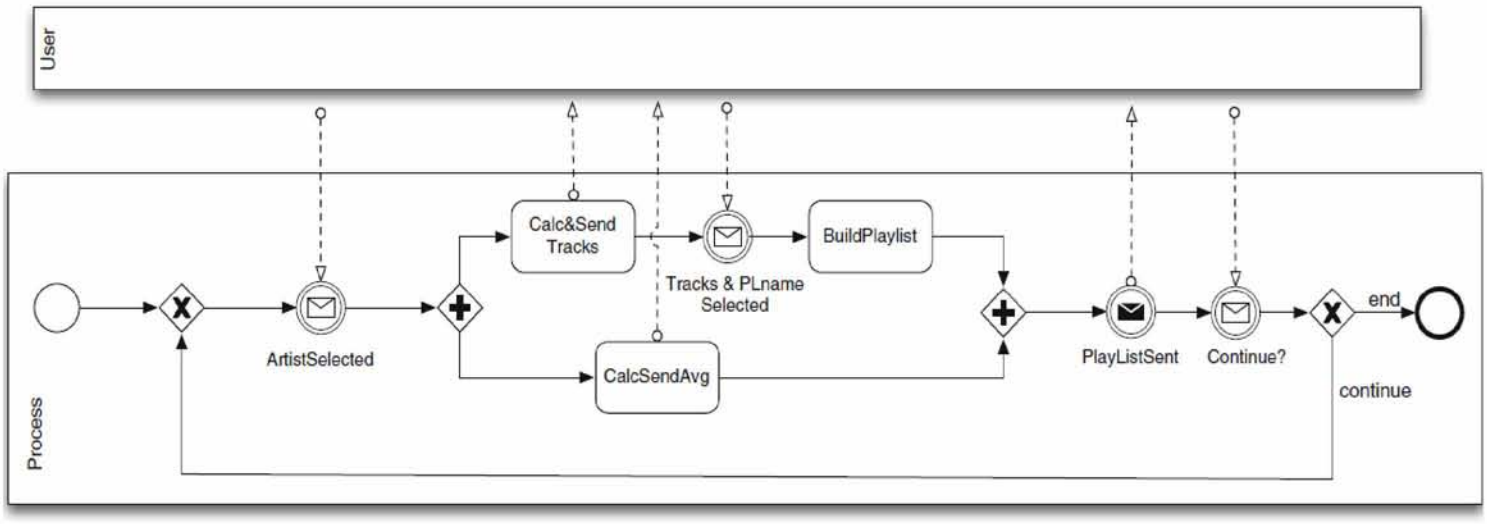
Στην πρότασή μας, εκφράζουμε τις απαιτήσεις δεδομένων ως διάγραμμα κλάσης UML (βλέπε σχήμα 1), ενώ η ροή της διαδικασίας εκφράζεται στο BPMN (όπως φαίνεται στο σχήμα 2). Παρατηρήστε ότι, όπως συνηθίζεται στο BPMN, έχουμε υιοθετήσει συμβάντα μηνυμάτων για απλές δραστηριότητες που συλλαμβάνουν μόνο δεδομένα από τον χρήστη ή ρίχνουν δεδομένα στον χρήστη. Αυτά περιλαμβάνουν ArtistSelected, TracksPLnameSelected, PlaylistSent και Continue.

Τώρα, ο στόχος μας είναι να συνδέσουμε τα γεγονότα της διαδικασίας με τα δεδομένα. Για να γίνει αυτό, πρέπει να διασφαλίσουμε ότι το διάγραμμα κλάσης UML εξετάζει όλα τα δεδομένα που έχουν τροποποιηθεί / πρόσβαση σε κάθε ατομική δραστηριότητα, απόφαση και μήνυμα που ελήφθη ή απεστάλη στο BPMN. Εφόσον, συνήθως, η εκτέλεση μιας διαδικασίας χρειάζεται να αποθηκεύσει κάποιες επιπλέον πληροφορίες σε μεταβλητές διαδικασίας (π.χ., πρέπει να θυμηθούμε τον καλλιτέχνη

που έχει επιλέξει ο χρήστης στην αρχή της διαδικασίας, δεδομένου ότι χρησιμοποιείται σε μεταγενέστερα συμβάντα BPMN), πρέπει να επεκτείνουμε το διάγραμμα κλάσης για να τα καταγράψουμε. Συγκεκριμένα, θεωρούμε μια νέα τάξη που καλούμε Artifact που περιέχει τέτοιες μεταβλητές διαδικασίας. Για να διαφοροποιήσουμε αυτή την κλάση από τα υπόλοιπα, την χαρακτηρίζουμε με το στερεότυπο Artifact.

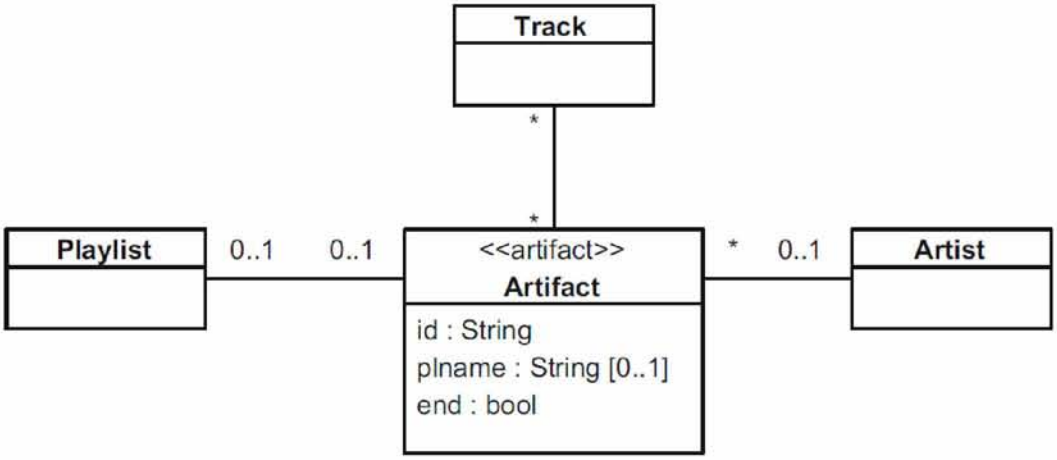


Εικ. 1. Διάγραμμα κλάσης για το παράδειγμα λίστας αναπαραγωγής



Εικ. 2. Διάγραμμα BPMN που αντιπροσωπεύει μια διαδικασία για τη δημιουργία λιστών αναπαραγωγής..

Για παράδειγμα, η εικόνα 3 δείχνει το τεχνητό στοιχείο για το τρέχον παράδειγμα μας. Αυτό το Artifact μπορεί να αποθηκεύσει τον καλλιτέχνη που επιλέχθηκε στην αρχή της διαδικασίας (μέσω σύνδεσης με τον καλλιτέχνη), το όνομα της λίστας αναπαραγωγής για να δημιουργήσει (μέσω του plname χαρακτηριστικού) τα κομμάτια για να προσθέσει αυτή τη νέα λίστα αναπαραγωγής , τον δημιουργό της ίδιας της λίστας αναπαραγωγής (συσχετισμός με τη λίστα αναπαραγωγής) και αν ο χρήστης επιλέγει να τερματίσει τη διαδικασία ή να συνεχίσει (τέλος).



Εικ. 3. Διάγραμμα κλάσης με την παράσταση του τεχνητού

Η παρουσίαση των μεταβλητών διαδικασίας ως κλάσης Artifact που σχετίζεται με τα υπόλοιπα στοιχεία του διαγράμματος κλάσης παρέχει τα πλεονεκτήματα του παραδειγματικού αντικειμένου. Δηλαδή, μπορούμε να καθορίσουμε τροποποιήσεις πάνω στα δεδομένα διεργασίας καθορίζοντας δημιουργίες, διαγραφές και ενημερώσεις αντικειμένων / σχέσεων αυτού του τεχνητού. Σημειώστε ότι με αυτόν τον τρόπο:

– Αποφεύγουμε σφάλματα κατά την εκτέλεση του μοντέλου, καθώς διασφαλίζουμε ότι το τεχνούργημα συνδέεται με μια συγκεκριμένη παρουσία μιας κλάσης και όχι με ένα id μιας στιγμής που πιθανόν να μην υπάρχει, λόγω του γεγονότος ότι το id είναι λάθος.

– Απλοποιούμε τον ορισμό των συμβάσεων λειτουργίας με χειρισμό αντικειμένων (δηλαδή στιγμιότυπα κλάσεων) αντί για αναγνωριστικά.

Στη συνέχεια, η ιδέα είναι ότι, όταν ξεκινά μια νέα διεργασία διεργασίας, δημιουργείται ένα νέο αντικείμενο Artifact για να αποθηκεύει όλες αυτές τις μεταβλητές διαδικασίας. Παρατηρήστε ότι αυτή η συμπεριφορά είναι παρόμοια με τον ελεγκτή περίπτωσης χρήσης στο [17], καθώς μια κατηγορία διατηρεί τις απαιτούμενες πληροφορίες για την εκτέλεση αρκετών σχετικών λειτουργιών ή εργασιών.

Το διάγραμμα κλάσης UML και η παράστασή του, συμπεριλαμβανομένου τουArtifact, μπορούν να θεωρηθούν ως το πληροφοριακό μοντέλο της διαδικασίας. Σημειώστε ότι αυτή η παράσταση μπορεί να θεωρηθεί (και στην πραγματικότητα αποθηκευμένη) ως σχεσιακή βάση δεδομένων (δηλ. Ένα μοντέλο πρώτης τάξης).

Τώρα, για οποιαδήποτε στιγμιαία στιγμή, ορίζουμε την κατάσταση της διαδικασίας ως εξής: (α) Η παράσταση του διαγράμματος κλάσης UML που περιλαμβάνει το Artifact. (β) Οι θέσεις των μαρκών σε διαγράμματα BPMN. Χρησιμοποιώντας αυτήν την έννοια της κατάστασης, μπορούμε να περιγράψουμε με ακρίβεια τη διαδικασία από την άποψη της εξέλιξης της καταστασεις. Για παράδειγμα, η προηγούμενη διαδικασία μας μπορεί να περιγραφεί με ακρίβεια ως εξής:

1. Στην αρχή μιας επανάληψης εμφανίζεται μήνυμα με τον επιλεγμένο καλλιτέχνη ως ωφέλιμο φορτίο. Αυτός ο καλλιτέχνης αποθηκεύεται στο Artifact μέσω της αντίστοιχης σύνδεσης.
2. Στη συνέχεια, ταυτόχρονα η διαδικασία ακολουθεί δύο κλάδους.
   - Πρώτο σκέλος:
     (α) Η δραστηριότητα CalcSendTracks υπολογίζει όλα τα κομμάτια που αποτελούν μέρος του κάποια ειδική έκδοση που καταγράφεται από τον artist στον artist και τα στέλνει στο χρήστη, τα κομμάτια που προκύπτουν από τον υπολογισμό δεν αποθηκεύονται στο Artifact, καθώς δεν χρησιμοποιούνται περαιτέρω στη διαδικασία, αλλά αποστέλλονται απευθείας στον χρήστη
     (β) Στη συνέχεια, ο χρήστης στέλνει τα επιλεγμένα κομμάτια και το όνομα της νέοας playlist. Και τα δύο αυτά στοιχεία αποθηκεύονται στο Artifact
     (γ) Χρησιμοποιώντας τα αποθηκευμένα τραγούδια Artifact και plname, η BuildPlayList δημιουργεί μια νέα λίστα αναπαραγωγής. Αυτή η playlist αποθηκεύεται στη συνέχεια στο Artifact

   - Δεύτερος κλάδος:
     (α) Η δραστηριότητα CalcSendPlaylists, ξεκινώντας από την αποθήκευση καλλιτέχνη του Artifact συλλέγει όλα τα κομμάτια του, υπολογίζει το σύνολο των λιστών αναπαραγωγής που ήδη υπάρχουν οπου περιέχουν κομμάτια από τον επιλεγμένο καλλιτέχνη και τα στέλνουν στο χρήστη. Παρατηρήστε ότι, δεδομένου ότι το αποτέλεσμα αυτό δεν χρησιμοποιείται πλέον στη διαδικασία,Δεν αποθηκεύεται στο Artifact.

3. Μετά την ολοκλήρωση των υπολογισμών τους και την ένταξή τους, αποστέλλεται στο χρήστη ένα μήνυμα με το νεο δημιουργημένο playlist.

4.Τέλος, με το Continue? η δραστηριότητα λαμβάνει πληροφορίες σχετικά με το αν ο χρήστης θέλει να συνεχίσει ή όχι και το αποθηκεύει Artifact boolean variable end. Στη συνέχεια, ανάλογα με αυτές τις πληροφορίες, η πύλη XOR τερματίζει τη διαδικασία ή εκτελεί μια άλλη επανάληψη

Αυτή η περιγραφή της εξέλιξης της κατάστασης μπορεί να γίνει πλήρως εκτελέσιμη με (1) τον προσδιορισμό των προηγούμενων δραστηριοτήτων και την εκδήλωση έναρξης / λήξης / μηνύματος μέσω μιας επίσημης γλώσσας. και (2) υιοθετόντας την σημασιολογία Petri Net για τον έλεγχο ροής BPMN.

Έτσι, για το σκοπό αυτό, προσδιορίζουμε κάθε δραστηριότητα στο διάγραμμα BPMN μέσω μιας σύμβασης εκτέλεσης OCL. Κάθε λειτουργία OCL θα έχει μια προϋπόθεση, δηλώνοντας τις συνθήκες που πρέπει να είναι αληθινές πριν από την εκτέλεση της εργασίας και μια μεταβατική προϋπόθεση, υποδεικνύοντας την προκύπτουσα κατάσταση του συστήματος μετά την εκτέλεση της λειτουργίας. Μερικές από τις εργασίες επιστρέφουν μόνο πληροφορίες στον χρήστη χωρίς να κάνουν αλλαγές (θα τους ονομάσουμε ερωτήματα): αυτές οι εργασίες θα περιλαμβάνουν το αποτέλεσμα της λέξης-κλειδιού ως μέρος της μεταβατικής προϋποθέσεως. Οι συμβάσεις λειτουργίας OCL πρέπει να αναφερθούν στις περιπτώσεις του Artifact για να απαλλαγούμε ρητά από τις πληροφορίες που χειραγωγούνται από τη διαδικασία.

Στον Πίνακα 1 παρουσιάζουμε τις συμβάσεις λειτουργίας OCL για το διάγραμμα BPMN στο Σχήμα 2. Σημειώστε ότι έχουμε επίσης καθορίσει συμβόλαιο για το συμβάν έναρξης και τέλους σε αυτό το διάγραμμα. Ο πρώτος (Initialize) είναι υπεύθυνος για την δημιουργία του Artifact αντικειμένου που θα κρατήσει τις πληροφορίες για την εκτέλεση της τρέχουσας διαδικασίας. Ο τελευταίος (end) είναι υπεύθυνος για τη διαγραφή του Artifact και των σχέσεών του. Εκτός από την εντολή Initialize, η οποία είναι μια κλάση, οι υπόλοιπες εργασίες είναι λειτουργίες στιγμιότυπων που προβάλλονται πάνω στο Artifact που χειρίζεται η διαδικασία (αυτή που δημιουργήθηκε από την Initialize).

Πίνακας 1. Συμβάσεις OCL για τα γεγονότα και τις δραστηριότητες του διαγράμματος BPMN

 Start Event	**context** artifact::Initialize() **post**: Artifact.allInstances()->exists(af | af.oclIsNew() **and** af.end=false **and** result=af) Initialize creates a new artifact with its end attribute set to false.
 Artist Selected	**context** artifact::ArtistSelected(artist:Artist) **post**: self.artist=artist ArtistSelected assigns the artist given as input to the process's artifact.
 Calc Send Tracks	**context** artifact::CalcAndSendTracks(): **Set**(Track) **post**: result = Track.allInstances()->select(t | t.album.artist = self.artist **and** t.album.oclIsTypeOf(SpecialEdition)) CalcSendTracks obtains all the tracks and selects those belonging to an album whose artist is equal to the artist linked to the artifact and which are part of an special edition. It returns this list as a result.
 Tracks Plname Selected	**context** artifact::TracksPlnameSelected(trackL:**Set**(Track), plName:**String**) **post**: self.track=trackL **and** self.plname=plName TracksPlnameSelected assigns the set of tracks provided as input to the artifact, and stores the playlist name given as input in the corresponding attribute of the artifact.
 Build Playlist	**context** artifact::BuildPlaylist() **post**: Playlist.allInstances()->exists(pl | pl.oclIsNew() **and** pl.name=self.plname **and** pl.track->includesAll(self.track)) BuildPlaylist creates a new instance of Playlist (oclIsNew). Its name is the name stored in the artifact and its tracks will correspond to the tracks linked to the artifact.
 Calc Send Playlists	**context** artifact::CalcSendPlaylists(): **Set**(Playlist) **post**: result = self.artist.album.track.playlist->asSet() CalcSendPlaylists obtains the playlists that already exist which contain tracks by the selected artist and sends this information to the user.
 Playlist Sent	**context** artifact::PlaylistSent(): Playlist **post**: result = self.playlist PlaylistSent returns the playlist that has been created (the one assigned to the artifact) as a result.
 Continue	**context** artifact::Continue(e:bool) **post**: self.end=e Continue updates the value of attribute end in the artifact with the given input.
 End Event	**context** artifact::End() **post**: Artifact.allInstances()->excludes(self) End deletes the artifact linked to this instance of the process and all the relationships it takes part in.

4. Επίτευξη των εκτελέσιμων μοντέλων επιχειρησιακών διαδικασιών

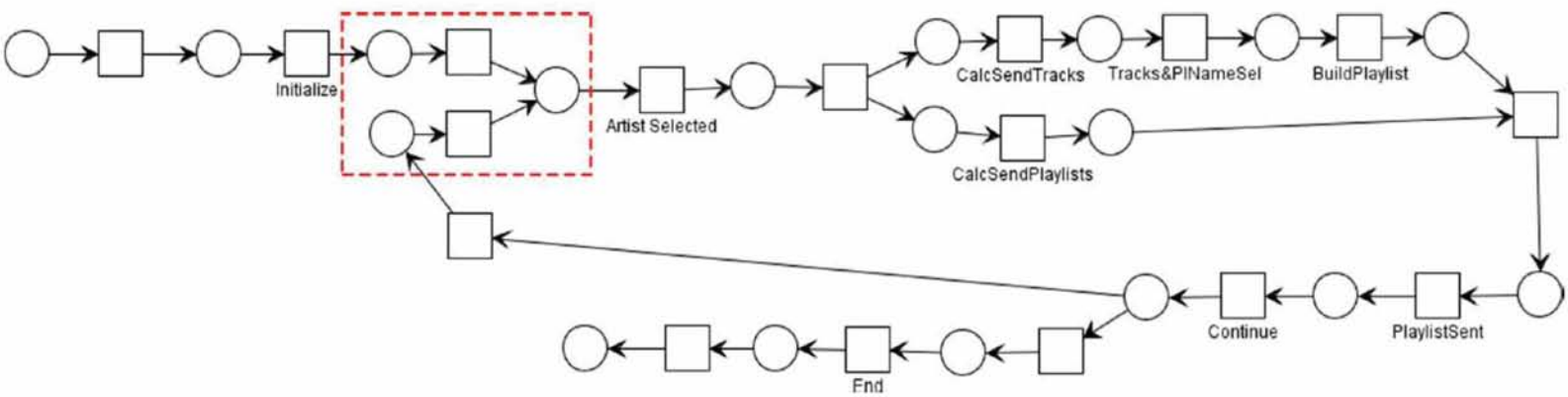
Για να καταστεί εκτελέσιμο αυτό το πλαίσιο, κωδικοποιούμε το διάγραμμα κλάσης UML ως σχεσιακή βάση δεδομένων που μπορεί να διαχειριστεί μέσω SQL, το διάγραμμα BPMN ως δίκτυο Petri και τις OCL συμβασεις ως λογικους κανόνες που εξάγουν τις δηλώσεις SQL που πρέπει να εφαρμοστούν στη βάση δεδομένων όταν η αντίστοιχη δραστηριότητα εκτελείται. Με αυτό τον τρόπο, αποκτάται η δυνατότητα εκτέλεσης του πλαισίου που επωφελείται από την τυποποιημένη τεχνολογία σχεσιακών βάσεων δεδομένων.

Από το διάγραμμα κλάσης σε ένα σχήμα βάσης δεδομένων. Από το διάγραμμα κλάσης σε ένα σχήμα βάσης δεδομένων. Κωδικοποιούμε το διάγραμμα κλάσης UML σε σχεσιακή βάση δεδομένων ακολουθώντας πολύ γνωστές τεχνικές σχεδίασης βάσεων δεδομένων [18]. Σημειώστε ότι σε αυτό το βήμα αποθηκεύουμε επίσης το Artifact (δηλ. Τις μεταβλητές διαδικασίας) στη βάση δεδομένων, καθώς το Artifact εμφανίζεται στο σχήμα UML.

Από το διάγραμμα BPMN σε ένα δίκτυο Petri. Το διάγραμμα BPMN μπορεί να τυποποιηθεί σε ένα δίχτυ Petri ακολουθώντας [16]. Η παρούσα πρόταση επικεντρώνεται στην επισημοποίηση της ροής ελέγχου (δηλ. Της σειράς εκτέλεσης των καθηκόντων και των γεγονότων) των μοντέλων BPMN, πράγμα που ακριβώς χρειαζόμαστε στην περίπτωση αυτή. Κατά προσέγγιση, κάθε εργασία θα χαρτογραφηθεί σε μια μετάβαση με μία είσοδο και μία θέση εξόδου. Οι κόμβοι πύλης θα αντιστοιχούν γενικά σε ένα συνδυασμό θέσεων και σιωπηλών μεταβάσεων, που αντιπροσωπεύουν τη συμπεριφορά δρομολόγησης της πύλης. Αυτή η μετάφραση σε ένα δίκτυο Petri είναι απαραίτητη για να βεβαιωθείτε ότι είναι τυπικά η σειρά εκτέλεσης των διαδικασιών
ακριβώς αυτή που ορίζεται από το BPMN.

Τα δίκτυα Petri απαιτούν επίσης μια αρχική σήμανση, η οποία αντιπροσωπεύει την αρχική κατάσταση του μοντέλου BPMN. Γενικά, αυτό σημαίνει να τοποθετήσετε ένα ενιαίο κουπόνι στη θέση που αντιστοιχεί στον κόμβο εκκίνησης του μοντέλου BPMN. Ακολουθώντας τη σημασιολογική σημασιολογία του προκύπτοντος Petri net, είναι δυνατόν να γνωρίζουμε ακριβώς ποιες εργασίες ή γεγονότα είναι έτοιμα να πραγματοποιηθούν.

Το δίκτυ Petri που λαμβάνουμε στο παράδειγμα μας παρουσιάζεται στο Σχήμα 4. Κάθε εργασία αντιστοιχεί σε μία ετικετοποιημένη μετάβαση, η οποία έχει μία είσοδο και μία θέση εξόδου. Κάθε κόμβος πύλης αντιστοιχεί σε ένα σύνολο θέσεων και μεταβάσεων. Για παράδειγμα, η πύλη συγχώνευσης XOR που τοποθετείται πριν από την εργασία ArtistSelected αντιστοιχεί στις μεταβάσεις και τις θέσεις μέσα στο διακεκομμένο ορθογώνιο στο Σχήμα 4. Η αρχική σήμανση συνίσταται στην τοποθέτηση ενός συμβόλου στην πιο αριστερή πλευρά (εκείνη χωρίς τόξα εισόδου) .



Σχήμα 4. Πλέγμα Petri που προκύπτει από τη μετάβαση του BPMN. Το διάστικτο ορθογώνιο δείχνει τις μεταβάσεις και τις θέσεις που αντιστοιχούν στη μετάφραση της πύλης συγχώνευσης XOR που τοποθετήθηκε πριν από την επιλογή ArtistSelected.

Από τις Συμβάσεις Λειτουργίας OCL έως τους Λογικους Κανόνες Αποκωδικοποίησης. Κάθε σύμβαση λειτουργίας OCL κωδικοποιείται σε ένα σύνολο λογικών κανόνων που διαισθητικά εισάγουν τις παρεμβολές / διαγραφές / ενημερώσεις SQL που πρέπει να εκτελέσουμε στη βάση δεδομένων SQL κατά την εφαρμογή της λειτουργίας. Με αυτό τον τρόπο, προχωρούμε από τις δηλωτικές προδιαγραφές OCL σε έναν επιτακτικό φορμαλισμό που μπορεί να εκτελεστεί.

Οι λογικοί κανόνες που λαμβάνουμε από κάθε λειτουργία έχουν την ακόλουθη μορφή:

—

$$ins\_P(\overline{x}) : -opName(a), arg0\_opName(\overline{x_0}), ..., argN\_opName(\overline{x_n}), pre(\overline{x_{pre}}), query(\overline{x_q})$$
$$del\_P(\overline{x}) : -opName(a), arg0\_opName(\overline{x_0}), ..., argN\_opName(\overline{x_n}), pre(\overline{x_{pre}}), query(\overline{x_q})$$
$$result(\overline{x}) : -opName(a), arg0\_opName(\overline{x_0}), ..., argN\_opName(\overline{x_n}), pre(\overline{x_{pre}}), query(\overline{x_q})$$

Ο επικεφαλής κάθε κανόνα καθορίζει το είδος της δήλωσης SQL που εφαρμόζεται (εισαγωγή, διαγραφή ή ερώτημα), ενώ το σώμα καθορίζει για ποιες τιμές. Αυτό είναι, διαισθητικά, ένας κανόνας της πρώτης μορφής δηλώνει ότι όταν ένας χρήστης επικαλείται τη λειτουργία opName στο Artifact a με τα n επιχειρήματα που ορίζονται στο arg0 opName, ..., argN opName, τότε κάποια γεγονότα P (x) πρέπει να εισαχθούν στη βαση δεδομενων εάν η προϋπόθεση που κωδικοποιείται από το pre (xpre) ικανοποιητε.

Οι μεταβλητές x εμφανίζονται με παράσταση χρησιμοποιώντας τα επιχειρήματα που δίνουν ο χρήστης $\overline{x0}$, ..., $\overline{xn}$ ή ακόμα και το αποτέλεσμα ερωτήματος πρώτης σειρας ($\overline{xq}$) που ανακτά τιμές από την τρέχουσα κατάσταση βάσης δεδομένων (ή δεδομένα διεργασίας αποθηκευμένα στο Artifact a). Εάν το ερώτημα επιστρέφει ένα σύνολο πλειάδων, ή ένα ίδιο το όρισμα είναι ένα σύνολο πλειάδων, ο κανόνας αποδίδει τόσες παρεμβολές ως στοιχεία στο σετ.

Ομοίως, οι κανόνες της δεύτερης και της τρίτης μορφής δηλώνουν διαγραφές γεγονότων και καθορίζουν τις πλειάδες για να επιστρέψουν στον χρήστη ως αποτέλεσμα. Οι τροποποιήσεις χαρακτηριστικών κωδικοποιούνται χρησιμοποιώντας τη γνωστή στρατηγική συνδυασμού μιας διαγραφής και ενός κανόνα εισαγωγής για το ίδιο γεγονός.

Η μετάφραση από τις συμβασεις της OCL σε αυτόν τον λογικό φορμαλισμό είναι μια επέκταση του [19]. Συγκεκριμένα, η επέκταση που προτείνουμε στο παρόν έγγραφο αποσκοπεί: (1) να επιτρέψει τη χρήση του ερωτήματος ($\overline{xq}$) για την εμφάνιση των μεταβλητών που χρησιμοποιούνται στις εισαγωγές / διαγραφές για εφαρμογή,(2) ασχολούνται με τα OCL Set δακτυλογραφημένα επιχειρήματα και (3) την ανάκτηση αποτελεσμάτων για το χρήστη.

Με βάση μια σύμβαση OCL, η μετάφρασή της σε λογική συνίσταται σε δύο βήματα. Το πρώτο αναλύει τη μεταβατική προϋπόθεση του OCL για να προσδιορίσει τους διαφορετικούς κανόνες που πρέπει να δημιουργήσουμε (δηλ. Προσδιορίζει τις επικεφαλίδες των διαφορετικών κανόνων που πρέπει να δημιουργηθούν). Ο δεύτερος είναι υπεύθυνος για τη δημιουργία των σωμάτων αυτών των κανόνων, η οποία γίνεται με την ανάλυση του ονόματος της λειτουργίας, των επιχειρημάτων και της προ / postcondition για τον προσδιορισμό του τρόπου εκδήλωσης
οι μεταβλητές από το κεφάλαιο κανόνα.

Προσδιορισμός της κεφαλής. Αναλύουμε την μεταβατική προϋπόθεση OCL για να καθορίσουμε το είδος των ενημερώσεων που εκτελούνται από τη λειτουργία. Ουσιαστικά, αυτές οι ενημερώσεις είναι: δημιουργία / διαγραφή αντικειμένων, εξειδίκευση / γενίκευση αντικειμένου, εισαγωγή / διαγραφή σχέσεων, εισαγωγή / διαγραφή / τροποποίηση χαρακτηριστικών και ερωτήματα. Κάθε μία από αυτές

τις ενημερώσεις θα οδηγήσει σε έναν ή περισσότερους κανόνες αποδόσεων. Για παράδειγμα, μια δημιουργία αντικειμένων της κλάσης C, όπου C είναι μια υποκλάση του C', οδηγεί σε έναν κανόνα εξαγωγής της μορφής ins C (o), μαζί με έναν άλλο κανόνα παραγω- γής της μορφής ins C'(o). Διαισθητικά, το σύνολο των κανόνων παραγωγής που δημιουργούνται για κάθε εισαγωγή / διαγραφή αντικειμένου πραγματοποιεί το κλείσιμο ISA όπως αναφέρεται στα Προκαταρκτικά.

Στον Πίνακα 2 παρουσιάζουμε τον τρόπο με τον οποίο εντοπίζουμε τέτοιες ενημερώσεις και τους κανόνες απόκτησης που προέρχονται. Αυτός ο πίνακας αποτελεί επέκταση των κανόνων μετάφρασης που ορίζονται στο [19] σε σύνολα και δεδομένα που εξάγονται από τη βάση δεδομένων. Διαισθητικά, διασχίζουμε την μεταβατική προϋπόθεση OCL για να βρούμε τα πρότυπα OCL που αναφέρονται στην αριστερή στήλη του πίνακα και, για κάθε αγώνα, δημιουργούμε έναν νέο κανόνα εξαγωγής όπως αναφέρεται στη δεξιά στήλη. Στον πίνακα αυτό, χρησιμοποιούμε o και u για να αναφερθούμε σε εκφράσεις αντικειμένων OCL τύπου C και a και b για να αναφερθούμε σε εκφράσεις τιμών OCL (όπως σταθερές). Επιπλέον, χρησιμοποιούμε τον ρόλο για να αναφερθούμε στις πλοηγήσεις κλήσεων ιδιοτήτων μέσω των συνδέσεων R, όπως στις πλοηγήσεις κλήσεων ιδιοτήτων στα χαρακτηριστικά A, και το ερώτημα για αναφορά σε μια έκφραση ερωτήματος OCL. Τέλος, υποθέτουμε ότι t είναι μια πλειάδα μεταβλητών n, όπου n είναι η arity του TupleType που επιστρέφεται από το ερώτημα OCL ή 1 εάν το ερώτημα OCL επιστρέφει ένα αντικείμενο / τιμή.

OCL pattern	Update kind	Derivation rules to create
o.oclIsNew()	Object creation	ins_C(o) ins_C'(o), for each C $\sqsubseteq$ C'
C.allInstances()->excludes(o)	Object deletion	del_C(o) del_C'(o) for each C' $\sqsubseteq$ C del_C"(o) for each C $\sqsubseteq$ C"
o.oclIsKindOf(C')	Object specialization	ins_C'(o) ins_C"(o) for each C' $\sqsubseteq$ C" $\sqsubseteq$ C
not o.oclIsKindOf(C')	Object generalization	del_C'(o) del_C"(o) for each C" $\sqsubseteq$ C'
o.role->includes(u) o.role->includesAll(u)	Relationship insertion	ins_R(o, u)
o.role->excludes(u) o.role->excludesAll(u)	Relationship deletion	del_R(o, u)
o.oclINew() and o.attr = a	Attribute insertion	ins_A(o, a)
o.attr = null	Attribute deletion	del_A(o, a)
o.attr = b	Attribute update	ins_A(o, b) del_A(o, a)
result = query	Query	result($\bar{t}$)

Πίνακας 2. Σχέδια OCL στους κανόνες αποδόσεων

Παρέχοντας το Σώμα. Αφού γνωρίζουμε το είδος των ενημερώσεων που εφαρμόζονται σε κάθε λειτουργία, πρέπει να καθορίσουμε τις τιμές για τις οποίες πρέπει να εφαρμοστούν. Αυτό επιτυγχάνεται μέσω της έκφρασης στο σώμα του κανόνα, ο οποίος αποτελείται από δύο διαφορετικά μέρη: ένα που είναι κοινό σε όλους τους κανόνες αποδόσεων κάθε πράξης, προσδιορίζοντας το όνομα της λειτουργίας, τα επιχειρήματα και την προϋπόθεση. και ένα συγκεκριμένο τμήμα για το καθένα η οποία δηλώνει τα συγκεκριμένα ερωτήματα (δηλ. μια συσχέτιση κυριολεκτικών που αναφέρεται στην κατάσταση βάσης δεδομένων) που χρησιμοποιείται για την παράσταση των μεταβλητών στον κανόνα. Εξηγούμε κάθε μέρος στα παρακάτω.

–Κοινό μέρος του σώματος. Το κοινό μέρος του σώματος αποτελείται από ένα γράμμα που αντιπροσωπεύει τη λειτουργία που μεταφράζουμε opName (a), της οποίας η μοναδική μεταβλητή αντιπροσωπεύει το Artifact στο οποίο εφαρμόζουμε τη λειτουργία, τα arg0 opName ($\overline{X0}$), ..., argN opName ($\overline{Xn}$) που αντιπροσωπεύουν τις τιμές που δίδει ο χρήστης για να εκτελέσει μια τέτοια λειτουργία και ένα ερώτημα λογικής προ ($\overline{Xpre}$) που κωδικοποιεί την προϋπόθεση της λειτουργίας. Τέτοιο λογικό ερώτημα προκύπτει από τη μετάφραση της προϋπόθεσης OCL σε λογικές σύμφωνα με την πρόταση στο [20].
–Ειδικό μέρος του σώματος. Τα ερωτήματα σε αυτό το μέρος λαμβάνονται μέσω της λογικής μετάφρασης των εκφράσεων αντικειμένων o, u, a, b και της έκφρασης ερωτήματος που εμφανίζεται στον Πίνακα 2, η οποία εκτελείται μόνο εάν οι εκφράσεις δεν αναφέρονται ρητά σε επιχειρήματα (έχουν ήδη κωδικοποιηθεί ήδη στο προηγούμενο βήμα). Χρησιμοποιούμε επίσης το [20] για την εκτέλεση αυτής της κωδικοποίησης. Ουσιαστικά, αυτό συνίσταται στη μετάφραση κάθε πλοήγησης OCL ως μια ακολουθία λογικών ατόμων που αντιπροσωπεύουν τις διάφορες ενώσεις στις οποίες διασχίζει. Για παράδειγμα, το t.album.artist μεταφράζεται σε truck (T, Al) ∧ recordedBy (Al, Ar). Η ιδέα της μετάφρασης είναι ότι κάθε λογική μεταβλητή που χρησιμοποιείται στις πλοήγησης αντιπροσωπεύει ένα διαφορετικό αντικείμενο UML και έτσι μπορεί να χρησιμοποιηθεί περαιτέρω για να δηλώσει τις συνθήκες πάνω σε ένα τέτοιο αντικείμενο. Για παράδειγμα, το specialEdition (Al) δηλώνει ότι το αντικείμενο UML που αντιπροσωπεύεται από τη μεταβλητή Al είναι ένα specialEdition.

Για παράδειγμα, οι συμβάσεις OCL των λειτουργιών BuildPlaylist και Calc-SendTracks μεταφράζονται ως οι κανόνες που εμφανίζονται στις Καταχωρήσεις 1.1 και 1.2. Σημειώστε ότι οι μεταβλητές στο κεφάλι των κανόνων εμφανίζονται με παραστατικά χρησιμοποιώντας ερωτήματα στο σώμα των κανόνων.

```
ins_Playlist(Pl) :- buildplaylist(A), artifactPlname(A,Pl)
ins_TrackIn(Tr, Al, Pl) :- buildplaylist(A), artifactPlname(A,Pl),
    artifactTrack(A,Tr,Al)
ins_ArtifactPlaylist(A,Pl) :- buildplaylist(A), artifactPlname(A,Pl)
```

Λίστα 1.1. Κωδικοποίηση λογικής για το έργο BuildPlaylist

```
result_CalcSendTracks(T,Al) :- calcSendTracks(A), track(T,Al), recordedBy(Al,
    Ar), artifactArtist(A,Ar), specialEdition(Al)
```

Λίστα 1.2. Κωδικοποίηση λογικής για την εργασία CalcSendTracks

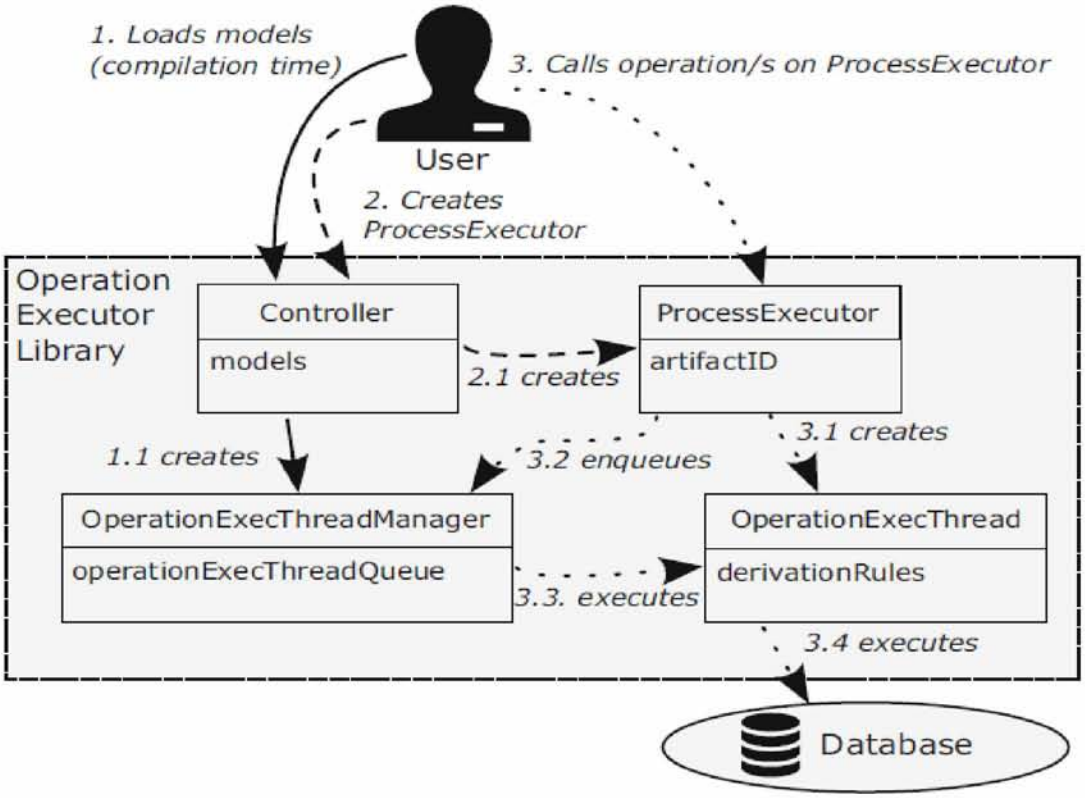
5. Εκτέλεση του πλαισίου

Το προτεινόμενο πλαίσιο μας επιτρέπει να εκτελούμε αυτόματα και χωρίς αμφιβολία διαδικασίες που καθορίζονται σύμφωνα με τα μοντέλα προδιαγραφών μας. Έχουμε δημιουργήσει μια βιβλιοθήκη Java για αυτό το σκοπό1. Αυτή η βιβλιοθήκη επιτρέπει τη φόρτωση στο χρόνο σύνταξης των υποκείμενων σημασιολογικών μοντέλων του πλαισίου και την εκτέλεση των λειτουργιών του κατά το χρόνο εκτέλεσης. Αυτό είναι:
- Δεδομένου (κατά το χρόνο σύνταξης): (1) μια σύνδεση βάσης δεδομένων SQL που κωδικοποιεί ένα σχήμα UML, (2) ένα σύνολο κανόνων εξάρτησης που ορίζουν τη σημασιολογία των λειτουργιών, (3) ένας χάρτης από τη λογική εξαρτάται από τους πίνακες.

- Δεδομένα (κατά την εκτέλεση): (4) όνομα λειτουργίας, (5) τιμές για τα επιχειρήματά τους.
- Εκτελεί (κατά το χρόνο εκτέλεσης): (6) τις ενημερώσεις που καθορίζονται στους κανόνες παραγωγής της λειτουργίας στη βάση δεδομένων και (7) επιστρέφει στον χρήστη τις πληροφορίες που καθορίζονται στο τμήμα αποτελεσμάτων της λειτουργίας.

Η τρέχουσα έκδοση της βιβλιοθήκης Java δεν ελέγχει ακόμα αν οι πράξεις που εκτελεί ο χρήστης ταιριάζουν με την εντολή που επέβαλαν τα δίκτυα Petri. Ωστόσο, κατανοούμε ότι αυτή η κρίσιμη (και απαραίτητη) λειτουργικότητα μπορεί να επιτευχθεί με την ενσωμάτωση οποιουδήποτε προσομοιωτή Petri Net στη βιβλιοθήκη μας και αυτός είναι ο λόγος για τον οποίο αφήσαμε την εφαρμογή αυτού του μέρους για μελλοντική δουλειά. Αντίθετα, το εργαλείο λειτουργεί σε οποιοδήποτε σχεσιακό σύστημα διαχείρισης βάσεων δεδομένων και είναι σε θέση να ελέγξει εάν οι εκτελεσθείσες λειτουργίες προκαλούν παραβίαση κάποιου περιορισμού ακεραιότητας (όπως οι περιορισμοί πολλαπλότητας min / max του διαγράμματος κλάσης UML, άλλοι σχολιασμοί διαγράμματος κλάσης UML, όπως ως υποσύνολο), μέσω της εφαρμογής της προσέγγισης ελέγχου αυξητικής ακεραιότητας [21].

Αρχιτεκτονική Βιβλιοθήκης Εκτελεσης Λειτουργίας.Η αρχιτεκτονική της βιβλιοθήκης μας φαίνεται στο σχήμα 5. Συνοπτικά, ένας χρήστης φορτώνει (κατά τη διάρκεια της σύνταξης) τα προηγούμενα μοντέλα στο στοιχείο του ελεγκτή, το οποίο τα αποθηκεύει. Όταν ο χρήστης θέλει να ξεκινήσει την εκτέλεση της διαδικασίας, καλεί τον ελεγκτή να παράσχει ένα νέο ProcessExecutor. Αυτή η κλάση εκτελεί όλες τις επικλήσεις λειτουργίας μιας τέτοιας στιγμής διεργασίας. Έτσι, κάθε περίπτωση αυτής της τάξης έχει το δικό της (μοναδικό) ID Artifact,που χρησιμοποιείται για την αποθήκευση στη βάση δεδομένων όλων των δεδομένων διεργασίας που σχετίζονται με την εν λόγω διεργασία διεργασίας.



Εικόνα 5. Αρχιτεκτονική βιβλιοθήκης JavaExecutor

Όταν ένας χρήστης κάνει χρήση μιας διαδικασίας στον ProcessExecutor, ο ProcessExecutor δημιουργεί ένα OperationExecutorThread, στον οποίο αποθηκεύουμε τους κανόνες αποδόσεων που σχετίζονται με αυτή τη λειτουργία. Στη συνέχεια, ο ProcessExecutor το προσθέτει στο OperationsExecutionThreadManager. Αυτό το στοιχείο είναι υπεύθυνο για την εκτέλεση της λειτουργίας αμέσως μόλις μπορεί να εκτελεστεί. Όταν εκτελείται το OperationExecutionThread, εκτελεί τα ακόλουθα βήματα:

1. Ερμηνεύει τις ενημερώσεις (εισαγωγές / διαγραφές) που πρέπει να εφαρμόσει σύμφωνα με τους κανόνες αποδόσεων και την κατάσταση της βάσης δεδομένων.
2. Ελέγχει ότι αυτές οι ενημερώσεις δεν προκαλούν παραβίαση περιορισμών σύμφωνα με τη μέθοδο incremental checking που ορίζεται στο [21].
3. Εάν δεν εντοπιστεί παραβίαση, οι ενημερώσεις μεταφράζονται ως δηλώσεις SQL εισαγωγής / διαγραφής / ενημέρωσης και εκτελούνται και εκτελείται το ερώτημα για την ανάκτηση του αποτελέσματος της εκτέλεσης της λειτουργίας (αν η ενέργεια επιστρέφει κάποιο αποτέλεσμα).
4. Διαφορετικά, ρίχνεται μια εξαίρεση.

6. Σχετική εργασία

Στη συνέχεια, συζητούμε πρώτα τα σχετικά πλαίσια για τη σύνδεση των δεδομένων και των μοντέλων διεργασιών και, στη συνέχεια, για αρκετές από τις επισημοποιήσεις τους για να επιτύχουμαι την εκτελεστότητά τους.

Όσον αφορά το πλαίσιο για τη μοντελοποίηση δεδομένων και επιχειρηματικών διαδικασιών, πολλά από τα υπάρχοντα έργα [9-11] χρησιμοποιούν γλωσσικές γνώσεις, οι οποίες είναι τυπικές και σαφείς, αλλά δυσκολότερο να γίνουν κατανοητές από τις BPMN και UML. Υπάρχουν και άλλες προσεγγίσεις που χρησιμοποιούν γραφικές παραστάσεις οι οποίες είναι πιο διαισθητικές και ελκυστικές για τους επιχειρηματικούς αναλυτές και προγραμματιστές, όπως [12,22,23]. [23]

βασίζεται στην προσέγγιση Guard-Stage-Milestone, η οποία αντιπροσωπεύει την εξέλιξη κάθε σχετικού αντικειμένου σε έναν κύκλο ζωής ακολουθώντας μια πιο δηλωτική προσέγγιση από τη δική μας. [22] χρησιμοποιεί γραφήματα συλλογής τεχνών, που είναι παρόμοια με τα δίκτυα Petri, για να αντιπροσωπεύσουν τη διαδικασία. [12] είναι η πιο παρόμοια προσέγγιση με τη δική μας και βασίζεται σε διάφορα διαγράμματα UML (διαφορετικά από αυτά που θεωρούμε) και συμβάσεις OCL για την αναπαράσταση των δεδομένων και της διαδικασίας. Ωστόσο, κανένα από αυτά τα έργα δεν ασχολείται με την εκτελεστότητα της διαδικασίας. οι περισσότεροι επικεντρώνονται στη μελέτη της ορθότητας του μοντέλου.

Όσον αφορά τη δυνατότητα εκτέλεσης διεργασιών, το BPEL (ή το WS-BPEL) επιτρέπει τον καθορισμό εκτελέσιμων επιχειρηματικών διαδικασιών χρησιμοποιώντας ένα σχήμα XML το οποίο καθιστά δύσκολη την ανάγνωση. Αν και υπάρχει χαρτογράφηση μεταξύ του BPMN 2.0 και του BPEL, είναι ελλιπής και πάσχει από πολλά ζητήματα [24]. Το έργο του [25] χρησιμοποιεί τα δίκτυα XML, μια μέθοδο μοντελοποίησης διαδικασιών με βάση το net-based Petri, που προορίζεται να εκτελέσει. Χρησιμοποιεί μια γραφική γλώσσα, η οποία αντιστοιχεί σε μια DTD (Ορισμός XML Τύπου Εγγράφου) που αντιπροσωπεύει τα δεδομένα που απαιτούνται από τη διαδικασία και οι χειρισμοί των δεδομένων παρουσιάζονται γραφικά στο δίχτυ XML. Σε αντίθεση με την προσέγγισή μας, αυτή η λύση βασίζεται στην τεχνολογία, καθώς οι προδιαγραφές των μοντέλων βασίζονται σε XML και δεν εξηγούνται λεπτομέρειες σχετικά με τον τρόπο επίτευξης της δυνατότητας εκτέλεσης.

Το YAWL [26] είναι μια γραφική γλώσσα ροής εργασίας, της οποίας η σημασιολογία ορίζεται τυπικά και βασίζεται σε δίχτυα Petri με την αντίστοιχη μηχανή εκτέλεσης. Η γλώσσα προσφέρει μια άποψη ροής ελέγχου και ροής δεδομένων της διαδικασίας, όπου τα δεδομένα ορίζονται σύμφωνα με μια μορφή XML. Δυστυχώς, οι εργασίες στη συνέχεια σχετιζονται με τις εισόδους και τις εξόδους τους, αλλά δεν επιτρέπουν τον καθορισμό των αλλαγών που γίνονται από καθένα από αυτά. Επομένως, ο μηχανισμός εκτέλεσης εντοπίζει μόνο λείπεις πληροφορίες και δεν είναι σε θέση να εκτελέσει πλήρως τη λειτουργία.

Στο [27] ορίζονται ένα υβριδικό μοντέλο που χρησιμοποιεί μια δηλωτική προδιαγραφή προσανατολισμένη στις πληροφορίες και μια επιτακτική προδιαγραφή μιας επιχειρησιακής διαδικασίας προσανατολισμένη προς τη ροή του ελέγχου. Χρησιμοποιώντας αυτή την προσέγγιση είναι δυνατό να αποκτηθεί αυτόματα ένα επιτακτικό μοντέλο που είναι εκτελέσιμο σε ένα τυπικό Σύστημα Διαχείρισης Επιχειρησιακών Διεργασιών. Ωστόσο, τα δεδομένα ορίζονται ως ένα σύνολο αδόκιμων μεταβλητών και οι προ και μετασχηματισμοί απλώς δηλώστε τις συνθήκες πάνω στα δεδομένα, αντί να υποδείξετε ακριβώς τι γίνεται από τα διαφορετικά καθήκοντα.

Νωρίτερα, παρόμοιες προσπάθειες με τις δικές μας είναι [28,29]. Και οι δύο προσεγγίσεις εστιάζουν στον ορισμό ενός εννοιολογικού μοντέλου το οποίο στη συνέχεια μπορεί να μεταφραστεί αυτόματα για να επιτευχθεί η εκτέλεση. Ωστόσο, ο σκοπός του [28] είναι διαφορετικός από τον δικό μας: ο κύριος στόχος τους είναι να μπορούν να επικυρώσουν το μοντέλο μέσω εκτέλεσης, ενώ ο στόχος μας είναι να επιτύχουμε τη δυνατότητα εκτέλεσης χρησιμοποιώντας τις τρέχουσες (de facto) τυπικές γλώσσες για την αντιπροσώπευση δεδομένων και διαδικασιών. Ομοίως, η προσέγγιση στο [29] - που μεταφράζει τα μοντέλα σε Pascal - είναι ξεπερασμένη από τις τρέχουσες, αντικειμενοστραφείς γλώσσες προγραμματισμού.

Επιπλέον, αξίζει να σημειωθεί ότι οι περισσότερες από αυτές τις προτάσεις δεν χρησιμοποιούν τυπικούς φορμαλισμούς για την εννοιολογική εκπροσώπηση, όπως συμβαίνει εδω.

7. Συμπεράσματα

Έχουμε προτείνει ένα πλαίσιο για τη σύνδεση δεδομένων και επιχειρηματικών διαδικασιών, οι οποίες μπορούν να εκτελεστούν αυτόματα. Χρησιμοποιεί τη γλώσσα BPMN για να αναπαριστά τις διαδικασίες, το διάγραμμα κλάσης UML για τα δεδομένα και τις συμβάσεις λειτουργίας OCL για να καθορίσει ποια είναι τα καθήκοντα της διαδικασίας. Χρησιμοποιώντας αυτές τις γλώσσες, δεν προτείνουμε ηδη-άλλο-φορμαλισμό, αλλά χρησιμοποιώντας ένα συνηθισμενο σε ενα νέο ολοκληρωμένο τρόπο σύνδεσης των δεδομένων και των διαδικασιών.

Έχουμε δείξει τη σκοπιμότητα της προσέγγισής μας δημιουργώντας μια βιβλιοθήκη Java, η οποία, δεδομένου ενός μοντέλου, είναι σε θέση να εκτελέσει τα καθήκοντα και να ενημερώσει την βάση πληροφοριών αναλόγως. Πριν από την εφαρμογή των αλλαγών, το εργαλείο εκτελεί έναν επαυξητικό έλεγχο των περιορισμών ακεραιότητας για να καθορίσει εάν υπάρχουν παραβιάσεις. Εάν συμβαίνει αυτό, θα κάνει μια εξαίρεση. Διαφορετικά, εφαρμόζει τις αλλαγές στην υποκείμενη βάση δεδομένων που αποθηκεύει τα δεδομένα. Όλα αυτά εκτελούνται χωρίς να απαιτούν παρέμβαση του χρήστη.

Με την προσέγγιση που παρουσιάζουμε εδώ,¨θολώνουμε¨ τη διάκριση μεταξύ προδιαγραφής και υλοποίησης, καθώς η ίδια η προδιαγραφή είναι εκτελέσιμη.

Ευχαριστίες:Το έργο αυτό υποστηρίχθηκε επισης απο την ομάδα του πανεπιστημίου της καταλονίας (upc) όπου και εργάστηκα για 6 μήνες στον ερευνητικό τομέα

Αναφορές

1. OMG: Unified Modeling Language (UML) superstructure, version 2.0. (2005). **http://www.uml.org/**
2. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, Heidelberg (2007)
3. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.: Fundamentals of Business Process Management. Springer, Heidelberg (2013)

4. Reichert, M.: Process and data: two sides of the same coin? In: Meersman, R., Panetto, H., Dillon, T., Rinderle-Ma, S., Dadam, P., Zhou, X., Pearson, S., Ferscha, A., Bergamaschi, S., Cruz, I.F. (eds.) OTM 2012. LNCS, vol. 7565, pp. 2–19. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33606-5 2
5. van der Aalst, W.M.P.: A decade of business process management conferences: personal reflections on a developing discipline. In: Proceedings of BPM 2012 (2012)
6. Cohn, D., Hull, R.: Business artifacts: a data-centric approach to modeling business operations and processes. IEEE-BDE 32(3), 3–9 (2009)
7. Bhattacharya, K., Caswell, N.S., Kumaran, S., Nigam, A., Wu, F.Y.: Artifactcentered operational modeling: lessons from customer engagements. IBM J. 46(4), 703–721 (2007)
8. Hull, R.: Artifact-centric business process models: brief survey of research results and challenges. In: OTM Confederated International Conference (2008)
9. Deutsch, A., Hull, R., Patrizi, F., Vianu, V.: Automatic verification of data-centric business processes. In: Proceedings of ICDT, pp. 252–267(2009)
10. Bagheri Hariri, B., Calvanese, D., De Giacomo, G., Deutsch, A., Montali, M.: Verification of relational data-centric dynamic systems with external services. In: Proceedings of PODS, pp. 163–174 (2013)
11. Belardinelli, F., Lomuscio, A., Patrizi, F.: Verification of agent-based artifact systems. J. Artif. Intell. Res. 51, 333–376 (2014)
12. Esta˜nol, M., Sancho, M.-R., Teniente, E.: Verification and validation of UML artifact-centric business process models. In: Zdravkovic, J., Kirikova, M., Johannesson, P. (eds.) CAiSE 2015. LNCS, vol. 9097, pp. 434–449. Springer, Cham (2015). doi:10.1007/978-3-319-19069-3 27
13. Franconi, E., Mosca, A., Oriol, X., Rull, G., Teniente, E.: Logic foundations of the OCL modelling language. In: Proceedings of Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, 24–26 September 2014, pp. 657–664 (2014)

14. Fowler, M., Scott, K.: UML Distilled - Applying the Standard Object Modeling Laguage. Addison-Wesley, Boston (1997)
15. OMG: Object Constraint Language (UML), version 2.4. Object Management Group (OMG) (2014). http://www.omg.org/spec/OCL/
16. Dijkman, R.M., Dumas,M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Inf. Softw. Technol. 50(12), 1281–1294 (2008)
17. Larman, C.: Applying UML and Patterns, 2nd edn. Prentice Hall, Upper Saddle River (2002)
18. Teorey, T., Lightstone, S., Nadeau, T.: Database Modeling and Design, 4th edn. Morgan Kaufmann, San Francisco (2006)
19. Queralt, A., Teniente, E.: Reasoning on UML conceptual schemas with operations. In: Eck, P., Gordijn, J., Wieringa, R. (eds.) CAiSE 2009. LNCS, vol. 5565, pp. 47–62. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02144-2 9
20. Queralt, A., Teniente, E.: Verification and validation of UML conceptual schemas with OCL constraints. ACM Trans. Softw. Eng. Methodol. 21(2), 13 (2012)
21. Oriol, X., Teniente, E.: Incremental checking of OCL constraints with aggregates through SQL. In: 34th International Conference on Conceptual Modeling, ER 2015, pp. 199–213 (2015)