ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Ενα Σύστημα για Configuration Management

Γεώργιος Ξουρής

Μεταπτυχιακή Εργασία

Ηράκλειο, Ιούνιος 1995

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

Ενα Σύστημα για Configuration Management

Εργασία που υποβλήθηκε από τον
ΓΕΩΡΓΙΟ ΞΟΥΡΗ
ως μερική εκπλήρωση των απαιτήσεων
για την απόκτηση
ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΕΙΔΙΚΕΥΣΗΣ

Συγγραφέας:

Γεώργιος Ξουρής
Τμήμα Επιστήμης Υπολογιστών

Εισηγητική Επιτροπή:

Πάνος Κωνσταντόπουλος, Αναπληρωτής Καθηγητής, Επόπτης

Απόστολος Τραγανίτης, Αναπληρωτής Καθηγητής, Μέλος

Πάνος Τραχανιάς, Επίκουρος Καθηγητής, Μέλος

Δεκτή:

Πάνος Κωνσταντόπουλος, Αναπληρωτής Καθηγητής
Πρόεδρος Επιτροπής Μεταπτυχιακών Σπουδών

Ηράκλειο, Ιούνιος 1995

Ενα Σύστημα για Configuration Management

Γεώργιος Ξουρής

Μεταπτυχιακή Εργασία

Τμήμα Επιστήμης Υπολογιστών
Πανεπιστήμιο Κρήτης

ΠΕΡΙΛΗΨΗ

Configuration Management (CM) είναι ο τομέας που ασχολείται με τον προσδιορισμό της διαμόρφωσης ενός συστήματος σε διακριτά σημεία στο χρόνο με στόχο το συστηματικό έλεγχο των αλλαγών στη διαμόρφωση, και τη διατήρηση της ακεραιότητας αυτής κατά τη διάρκεια ζωής του συστήματος, δηλαδή είναι ο τομέας που ασχολείται με τον έλεγχο της εξέλιξης πολύπλοκων συστημάτων. Software Configuration Management (SCM) είναι η ειδίκευση του CM για προγράμματα ηλεκτρονικών υπολογιστών και για σχετικά με αυτά έγγραφα.

Τα συστατικά στοιχεία του CM είναι οι έξι παρακάτω λειτουργίες : αναγνώριση, έλεγχος αλλαγών, αναφορά κατάστασης, έλεγχος απαιτήσεων, επιλογή έκδοσης και κατασκευή βάσης, και κατασκευή λογισμικού.

Η επιλογή εκδόσεων είναι μία ενεργή περιοχή έρευνας μέσα στο CM, η οποία ασχολείται με την επιλογή των κατάλληλων εκδόσεων στοιχείων, ώστε να προκύπτουν βιώσιμες διαμορφώσεις συστημάτων. Η γενική προσέγγιση στο θέμα, από τα υπάρχοντα συστήματα για SCM, συσχετίζει περιορισμούς με μοντέλα συστημάτων, όπου οι περιορισμοί είναι συνθήκες πάνω σε γνωρίσματα λογισμικών αντικειμένων, οι οποίες επιλέγουν τις κατάλληλες διασκευές (revisions) και παραλλαγές (variants) εκδόσεων.

Η προσέγγιση που προτείνουμε εμείς, μέσα από την παρούσα εργασία, χρησιμοποιεί μία μερικά διατεταγμένη άλγεβρα πάνω στα ονόματα των εκδόσεων των αντικειμένων. Μία συγκεκριμένη έκδοση ενός συστήματος σχηματίζεται από το συνδυασμό των πλησιέστερων εκδόσεων των στοιχείων από τα οποία αποτελείται το σύστημα. Η επιλογή της πλησιέστερης έκδοσης γίνεται μέσω της άλγεβρας.

Το μοντέλο της άλγεβρας, με τη δεδομένη μορφή του, δε μπορεί να εκφράσει ορισμένες

βασικές έννοιες του CM και ειδικότερα της επιλογής εκδόσεων, για παράδειγμα, δε μας επιτρέπει να καταλάβουμε αν μία έκδοση είναι διασκευή ή παραλλαγή. Στην παρούσα εργασία προτείνουμε ένα μοντέλο για configuration management (περιγράφεται με τη γλώσσα παράστασης γνώσης TELOS), με το οποίο, αντιμετωπίζουμε τα προβλήματα που αναφέρουμε προηγουμένως και εισάγουμε τις έννοιες των περιορισμών ασυμβατότητας και των προαιρετικών (optional) αντικειμένων.

Επιδεικνύουμε τη χρησιμότητα αυτής της προσέγγισης μέσω ενός συστήματος, που υλοποιήσαμε, για CM. Το σύστημά μας διαχειρίζεται εκδόσεις αντικειμένων, εισάγει δεδομένα στη βάση δεδομένων και συγκεκριμένα περιορισμούς, και επιλέγει εκδόσεις για την κατασκευή μίας έκδοσης συστήματος ελέγχοντας την ύπαρξη περιορισμών μεταξύ τους.

Επόπτης : Πάνος Κωνσταντόπουλος, Αναπληρωτής Καθηγητής
Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης

A System for Configuration Management

George Ksouris

Master of Science Thesis

Department of Computer Science
University of Crete

ABSTRACT

Configuration management (CM) is concerned with identifying the configuration of a system, at discrete points in time, for the purpose of systematically controlling changes to the configuration and maintaining the integrity of the configuration throughout the system life cycle, i.e. CM is concerned with controlling the evolution of complex systems ; software configuration management is its specialization for computer programs and associated documents.

The basic elements of CM are the following six functions : identification, control, status accounting, auditing, version selection and baselining, and software manufacture.
Version selection is currently an active research area within CM, which deals with the selection of the appropriate object versions in order to produce viable system configurations. The general approach is to associate constraints with system models. The constraints are conditions on attributes of software objects that select appropriate variants and revisions.

Our approach applies a partially ordered algebra on the version labels of the objects. A particular version of an entire system is formed by combining the most relevant existing versions of each of the various components of the system. The selection of the most relevant version is decided by the algebra.
A weakness of the original algebra model lies in its inability to express certain basic concepts of CM, in particular version selection. For example, it cannot distinguish between variants and revisions. In the present work we have designed a CM model (implemented in the TELOS knowledge representation language), which deals with the problems of the original model and introduces the concepts of incompatibility constraints and optional objects.

We demonstrate the utility of this approach through a system for CM, which we have developed. The system manages object versions, inserts data (incompatibility constraints) into the database, and selects object versions for the construction of a system version, checking the existence of constraints between object versions.

Supervisor : Panos Constantopoulos, Associate Professor
Department of Computer Science, University of Crete

Ευχαριστίες

Θα ήθελα να ευχαριστήσω όλους όσους με βοήθησαν στην ολοκλήρωση αυτής της εργασίας. Πρώτα από όλους τον επιβλέποντα καθηγητή μου κ. Πάνο Κωνσταντόπουλο για την καθοδήγηση και τις συμβουλές του.

Ευχαριστώ τον δρ. Martin Dörr, για τις μαραθώνιες συζητήσεις που είχαμε. Η συνεργασία μαζί του υπήρξε καθοριστική για την εξέλιξη της εργασίας αυτής.

Επίσης θα ήθελα να ευχαριστήσω τους καθηγητές κ.κ. Απόστολο Τραγανίτη και Πάνο Τραχανιά για την συμμετοχή τους στην επιτροπή που έκρινε αυτή την εργασία.

Θα ήθελα επίσης να ευχαριστήσω το Ινστιτούτο Πληροφορικής του Ιδρύματος Τεχνολογίας και Ερευνας (Ι.Τ.Ε.) για την υλικοτεχνική υποδομή που μου παρείχε.

Τέλος, θέλω να ευχαριστήσω τους γονείς μου και την οικογένεια μου για την κατανόηση και την αγάπη τους, και τους φίλους και συναδέλφους Χαρά Ξανθάκη και Τάσο Στογιαννίδη για τη βοήθειά τους στη διόρθωση του τελικού κειμένου και τη συμπαράσταση τους κατά τη διάρκεια αυτής της εργασίας.

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1

Εισαγωγή

Κατά τη διάρκεια των τελευταίων ετών, ο ρόλος του Software Configuration Management (SCM) επεκτάθηκε απότομα, και αυξήθηκε σημαντικά η σπουδαιότητα της χρήσης του για διάφορους λόγους. Η μεγάλη αύξηση του μεγέθους του λογισμικού σε διάφορες εργασίες είχε ως αποτέλεσμα την ανάγκη διαχείρισης περισσοτέρων στοιχείων. Η εισαγωγή των εργαλείων CASE (Computer - Aided Software Engineering) αύξησε τον αριθμό και τους τύπους των αντικειμένων που είναι δυνατό να χρησιμοποιηθούν από κάποια μηχανή και πρέπει να συντηρούνται. Η άφιξη νέων υπολογιστικών τοπολογιών και δομών εφαρμογής επέκτεινε την πρόκληση του SCM σε διαφορετικές εφαρμογές. Το ενδεχόμενο για ουσιώδη βελτίωση της παραγωγικότητας κατά την κατασκευή λογισμικού και για μεγάλη μείωση του κόστους κατασκευής του μέσω της χρήσης βιβλιοθηκών αναχρησιμοποίησης λογισμικού, είναι πολύ μεγαλύτερο. Επιπλέον, όλο και περισσότεροι οργανισμοί αναζητούν τρόπους για να διαχειρισθούν δεδομένα και πληροφορία σχετικά με τεχνολογία σαν στρατηγική επιχείρησης, αφού αυτό μπορεί να τους βοηθήσει να κερδίσουν και να στηρίξουν ανταγωνιστικά πλεονεκτήματα, που κυμαίνονται από αυξητικές βελτιώσεις στην ποιότητα του προϊόντος ή χαμηλότερο κόστος σε σημαντικά ανοίγματα, που δημιουργούν την ευκαιρία μιας νέας αγοράς [47].

Λόγω αυτών των αλλαγών, αποδίδεται μία αυξημένη έμφαση στο SCM. Η περιοχή του SCM δεν ασχολείται πλέον μόνο με τον έλεγχο του κώδικα, αλλά με όλα τα προϊόντα που σχετίζονται με μία εργασία, όπως : συμβόλαια, εκτελέσιμο κώδικα, δεδομένα, μετρήσεις, αναφορές κ.λ.π. Στη συνέχεια του κεφαλαίου θα δώσουμε τον ορισμό του configuration management και του software configuration management, θα δούμε τις διαφορές τους, και θα μιλήσουμε για το σκοπό της χρήσης του. Τέλος θα περιγράψουμε συνοπτικά το περιεχόμενο και τα αποτελέσματα της παρούσας εργασίας.

1.1 Ορισμοί

Configuration management (CM) είναι ο τομέας που ασχολείται με τον προσδιορισμό της διαμόρφωσης ενός συστήματος σε διακριτά σημεία στο χρόνο με στόχο το συστηματικό έλεγχο των αλλαγών στη διαμόρφωση, και τη διατήρηση της ακεραιότητας αυτής κατά τη διάρκεια ζωής του συστήματος [4], δηλαδή είναι ο τομέας που ασχολείται με τον έλεγχο της εξέλιξης πολύπλοκων συστημάτων · software configuration management (SCM) είναι η ειδίκευση του CM για προγράμματα ηλεκτρονικών υπολογιστών και για σχετικά με αυτά έγγραφα [52].

Το CM στη γενική του μορφή είναι ωφέλιμο για κάθε μεγάλο σύστημα το οποίο, λόγω της πολυπλοκότητάς του, δεν μπορεί να έχει τέλεια συμπεριφορά σε όλες τις χρήσεις για τις οποίες προορίζεται. Ενα τέτοιο σύστημα, κατά τη διάρκεια της ζωής του, θα υποστεί έναν αριθμό από τροποποιήσεις, οι οποίες μπορεί να είναι και αλληλοσυγκρουόμενες, δημιουργώντας με αυτόν τον τρόπο, όχι ένα μοναδικό σύστημα, αλλά ένα σύνολο από συσχετιζόμενα συστήματα, το οποίο θα ονομάζουμε *οικογένεια συστημάτων*. Μία οικογένεια συστημάτων αποτελείται από έναν αριθμό εξαρτημάτων, τα οποία μπορούν να συναρμοσθούν κατάλληλα ώστε να σχηματίσουν ξεχωριστό μέλος της οικογένειας. Ενας μεγάλος αριθμός από τα εξαρτήματα πρέπει να είναι κοινός μεταξύ των μελών έτσι ώστε η οικογένεια να είναι οικονομικά βιώσιμη. Στόχος του CM είναι η συντήρηση μεγάλων και επεκτάσιμων οικογενειών συστημάτων.

Η βασική διαφορά του SCM από το γενικό CM είναι η εξής : Οι αλλαγές στο λογισμικό είναι πιο εύκολες απ' ότι στο υλικό, και γι' αυτό το λογισμικό αλλάζει γρηγορότερα. Ακόμη και σε σχετικά μικρά συστήματα λογισμικού, τα οποία κατασκευάζονται από μία ομάδα ατόμων, μπορεί να υπάρχει η εμπειρία ενός σημαντικού ρυθμού αλλαγών, ενώ σε μεγάλα συστήματα, όπως συστήματα τηλεπικοινωνιών, οι ενέργειες που χρειάζονται ώστε να γίνουν αλλαγές και τροποποιήσεις μπορούν να αποτρέψουν διαδικασίες για configuration management.
Θα μπορούσε κάποιος να πει ότι το SCM είναι ενδεχομένως πιο αυτοματοποιημένο σε σχέση με το γενικό CM, γιατί όλα τα εξαρτήματα ενός συστήματος λογισμικού μπορούν εύκολα να αποθηκευτούν, για παράδειγμα στη βοηθητική μνήμη ενός ηλεκτρονικού υπολογιστή. Ομως, όπως το CAD/CAM και η ρομποτική θέτουν όλο και περισσότερες διαδικασίες κάτω από τον έλεγχο ηλεκτρονικού υπολογιστή, έτσι και το φυσικό configuration management θα υιοθετήσει αναμφίβολα, μερικές από τις προσεγγίσεις που χρησιμοποιούνται για το λογισμικό. Αυτό γίνεται ήδη στην περιοχή των κυκλωμάτων VLSI : ο σχεδιασμός και η επεξεργασία κυκλωμάτων μπορούν να αντιμετωπισθούν όπως ο σχεδιασμός και η μετάφραση

λογισμικού.

Από τα παραπάνω παρατηρούμε ότι, δεν υπάρχει ουσιαστική διαφορά μεταξύ CM και SCM, πέραν της ευκολίας κατασκευής εκδόσεων λογισμικού, επομένως τόσο το CM όσο και το SCM αντιμετωπίζουν τα ίδια προβλήματα. Αρα, όπου στη συνέχεια χρησιμοποιείται ο όρος SCM (software configuration management), δεν πρέπει ο αναγνώστης να θεωρεί ότι όσα αναφέρονται στο συγκεκριμένο σημείο ισχύουν αποκλειστικά και μόνο για το λογισμικό. Η χρήση του όρου γίνεται για λόγους συμβατότητας με τη βιβλιογραφία.

1.2 Σκοπός της Χρήσης SCM

Η χρήση SCM έχει σαν στόχο να εξασφαλίσει την ακεραιότητα ενός προϊόντος και να διευκολύνει την διαχείριση του προϊόντος στα διάφορα στάδια της εξέλιξης του. Παρ' όλο που η χρήση configuration management επιβάλλει μία επιβάρυνση, είναι γενικά αποδεκτό ότι οι συνέπειες από τη μη χρήση SCM μπορούν να οδηγήσουν σε πολλά προβλήματα. Η επιβάρυνση που προκαλείται με τη χρήση SCM σχετίζεται με το χρόνο μέσα στον οποίο πρέπει να ολοκληρωθεί η κατασκευή ενός προϊόντος, τους διαθέσιμους πόρους, και τα αποτελέσματα πάνω σε άλλες πλευρές, κατά τη διάρκεια ζωής, του λογισμικού. Για παράδειγμα, το SCM επηρεάζει πολλούς από τους προσανατολισμούς κατά τη λήψη αποφάσεων και την εκτίμηση του ρίσκου που θα έχει ένα προϊόν. Οι αποφάσεις που πρέπει να ληφθούν είναι σχετικές με το πώς θα πρέπει να αναπτυχθεί ο κώδικας και ποια θα είναι η τιμή του · ποιο σύστημα SCM πρέπει να χρησιμοποιηθεί · πότε πρέπει να τεθεί ένα προϊον κάτω από SCM [10]. Ολοι αυτοί οι παράγοντες επηρεάζουν το λογισμικό κατά τη διάρκεια ζωής του. Ετσι, όσο λιγότερη είναι η επιβάρυνση που επιφέρει η χρήση ενός συστήματος SCM, τόσο το καλύτερο, οπότε το σύστημα με λιγότερη επιβάρυνση αποτελεί με μεγάλη πιθανότητα την καλύτερη επιλογή. Γιατί δεν πρέπει να είναι μόνο το κόστος χαμηλό, αλλά επίσης η επίδραση της καθημερινής χρήσης του συστήματος δε θα πρέπει να καταβάλλει τους χρήστες. Το ιδανικό σύστημα SCM πρέπει να είναι όσο γίνεται περισσότερο βοηθητικό και να έχει μια αποδεκτή επιβάρυνση.

1.3 Περιεχόμενο και Αποτελέσματα της Εργασίας

Ενα από τα βασικά στοιχεία του CM είναι η επιλογή εκδόσεων, η οποία ασχολείται με την επιλογή των κατάλληλων εκδόσεων στοιχείων, ώστε να προκύπτουν βιώσιμες διαμορφώσεις συστημάτων. Η γενική προσέγγιση στο θέμα, από τα υπάρχοντα συστήματα για SCM, συσχετίζει περιορισμούς με μοντέλα συστημάτων, όπου οι περιορισμοί είναι συνθήκες πάνω σε

γνωρίσματα λογισμικών αντικειμένων, οι οποίες επιλέγουν τις κατάλληλες διασκευές (*revisions*) και παραλλαγές (*variants*) εκδόσεων.

Η προσέγγιση που προτείνουμε εμείς, μέσα από την παρούσα εργασία, χρησιμοποιεί μία μερικά διατεταγμένη άλγεβρα [42] πάνω στα ονόματα των εκδόσεων των αντικειμένων. Μία συγκεκριμένη έκδοση ενός συστήματος σχηματίζεται από το συνδυασμό των πλησιέστερων εκδόσεων των στοιχείων από τα οποία αποτελείται το σύστημα. Η επιλογή της πλησιέστερης έκδοσης γίνεται μέσω της άλγεβρας.

Το μοντέλο της άλγεβρας, με τη δεδομένη μορφή του, δε μπορεί να εκφράσει ορισμένες βασικές έννοιες της επιλογής εκδόσεων, για παράδειγμα, δε μας επιτρέπει να καταλάβουμε αν μία έκδοση είναι διασκευή ή παραλλαγή. Στην παρούσα εργασία σχεδιάσαμε ένα μοντέλο για configuration management (περιγράφεται με τη γλώσσα παράστασης γνώσης TELOS [41]), με το οποίο, αντιμετωπίζουμε τα προβλήματα που αναφέρουμε προηγουμένως και εισάγουμε τις έννοιες των περιορισμών ασυμβατότητας (ένα παρόμοιο είδος περιορισμών χρησιμοποιείται από το CMA [43]) και των προαιρετικών (*optional*) αντικειμένων.

Επίσης, σχεδιάσαμε και υλοποιήσαμε ένα σύστημα, που βασίζεται στο μοντέλο, για configuration management (η υλοποίηση του συστήματος έγινε με τη γλώσσα προγραμματισμού ANSI C). Το σύστημά μας διαχειρίζεται εκδόσεις αντικειμένων, εισάγει δεδομένα στη βάση δεδομένων και συγκεκριμένα περιορισμούς, και επιλέγει εκδόσεις για την κατασκευή μίας έκδοσης συστήματος ελέγχοντας την ύπαρξη περιορισμών μεταξύ τους.

Στη συνέχεια περιγράφεται η οργάνωση της εργασίας. Στο δεύτερο κεφάλαιο παρουσιάζονται τα βασικά στοιχεία του SCM (λειτουργίες, εκδόσεις αντικειμένων, σχέσεις μεταξύ εκδόσεων, πράξεις που χρησιμοποιούνται στη διαχείριση εκδόσεων) και η περιοχή του CM με την οποία ασχοληθήκαμε, κατά κύριο λόγο. Στο τρίτο κεφάλαιο χωρίζεται η εξέλιξη του SCM σε τρεις τεχνολογικές γενεές, γίνεται μία κατάταξη διαφόρων συστημάτων σε αυτές και περιγράφονται συγκεκριμένα γνωρίσματα συστημάτων για CM, τα οποία μπορούν να ιδωθούν σαν ένα φάσμα λειτουργιών, όπου κάθε γνώρισμα χτίζεται πάνω σε κάποιο άλλο. Στο τέταρτο κεφάλαιο εξετάζεται αναλυτικά ένα αντιπροσωπευτικό σύστημα από κάθε γενεά, και συγκεκριμένα : το RCS, το Gandalf και το shape, αντίστοιχα. Στο πέμπτο κεφάλαιο παρουσιάζεται μία προσέγγιση στο CM που χρησιμοποιεί μία μερικά διατεταγμένη άλγεβρα πάνω στα ονόματα των εκδόσεων αντικειμένων, οι "αδυναμίες" αυτής και ένα μοντέλο με το οποίο επιχειρούμε να αντιμετωπίσουμε αυτές. Στο έκτο κεφάλαιο περιγράφεται η υλοποίηση ενός συστήματος για CM και στο έβδομο διάφορες βελτιώσεις και επεκτάσεις του συστήματος.

Κεφάλαιο 2

Βασικά Στοιχεία Software Configuration Management

Στο παρόν κεφάλαιο θα ασχοληθούμε με τα βασικά στοιχεία του SCM, και πρώτα από όλα θα μιλήσουμε για τις λειτουργίες του (αναγνώριση, έλεγχος αλλαγών, αναφορά κατάστασης, έλεγχος απαιτήσεων, επιλογή έκδοσης και κατασκευή βάσης, κατασκευή λογισμικού) και θα εξετάσουμε κάθε μία αναλυτικά. Στη συνέχεια θα μιλήσουμε για εκδόσεις αντικειμένων, σχέσεις μεταξύ εκδόσεων και πράξεις που χρησιμοποιούνται στη διαχείριση εκδόσεων. Τέλος, θα μιλήσουμε περιληπτικά για την περιοχή του CM με την οποία ασχοληθήκαμε, κατά κύριο λόγο, στην παρούσα εργασία.

2.1 Λειτουργίες του Software Configuration Management

Ο πρωταρχικός, αντικειμενικός στόχος του SCM είναι η αποτελεσματική διαχείριση του κύκλου ζωής ενός συστήματος λογισμικού και των διαμορφώσεων του συστήματος, που λαμβάνουν χώρα κατά την ανάπτυξή του. Μία θεμελιώδης έννοια γι' αυτήν τη διαχείριση είναι η έννοια της *βάσης* (*baseline*) [3]. Ως βάση χαρακτηρίζεται ένα σχέδιο ή ένα προϊόν, το οποίο έχει τυπικά θεωρηθεί ότι, από εδώ και στο εξής, θα εξυπηρετεί σαν βάση για περαιτέρω ανάπτυξη και μπορεί να αλλάξει μόνο μέσω τυπικών διαδικασιών ελέγχου [24]. Δηλαδή, μία βάση αποτελεί ένα σημείο αναφοράς στην κατασκευή ενός συστήματος [1]. Αποτελείται από ένα ή περισσότερα στοιχεία λογισμικού, τα οποία έχουν τυπικά οριοθεί ως βασικά στοι-

[1] Κατά την κατασκευή ενός συστήματος μπορούν να υπάρξουν περισσότερες από μία βάσεις.

χεία μία συγκεκριμένη χρονική στιγμή. Τα βασικά στοιχεία έχουν ολοκληρωθεί, ελεγχθεί, εγκριθεί και είναι αποθηκευμένα με τέτοιο τρόπο ώστε να ελέγχεται κάθε πρόσβαση σε αυτά. Τα στοιχεία αυτά μπορούν να αλλάξουν, όπως αναφέραμε και παραπάνω, μόνο με τη χρήση καθορισμένων διαδικασιών αλλαγής [5].

Η διαχείριση μίας βάσης και των προϊόντων που περιλαμβάνονται σε αυτή απαιτεί την εφαρμογή διαφορετικών λειτουργιών, οι οποίες, αποτελούν τα συστατικά στοιχεία του software configuration management. Οι λειτουργίες αυτές (παρουσιάζονται στο σχήμα 2.1), σύμφωνα με τον Bersoff [3] είναι τέσσερεις, ενώ ο Tichy [52] προσθέτει άλλες δύο, τις οποίες και εμείς θεωρούμε απαραίτητες. Όλες οι λειτουργίες ορίζονται στη συνέχεια.

2.1.1 Αναγνώριση

Ενα σχήμα αναγνώρισης χρειάζεται για να καθορίζει τη δομή του προϊόντος. Αυτό περιλαμβάνει προσδιορισμό του είδους και της δομής των στοιχείων του λογισμικού, κάνοντάς τα μοναδικά και παρέχοντας τη δυνατότητα πρόσβασης σε κάποια μορφή τους με την εκχώρηση σε κάθε στοιχείο ενός ονόματος, ενός αναγνωριστικού έκδοσης και ενός αναγνωριστικού διαμόρφωσης. Πρέπει να δοθεί έμφαση στο ότι η λειτουργία της αναγνώρισης δεν πρέπει να διευκολύνει μόνο την επικοινωνία γύρω από το λογισμικό αλλά και τη σχετική με όλα τα προϊόντα της εργασίας, στο βασικό σύστημα. Η αξιόπιστη αναγνώριση βοηθά να αποφευχθούν τα ακόλουθα προβλήματα [52] :

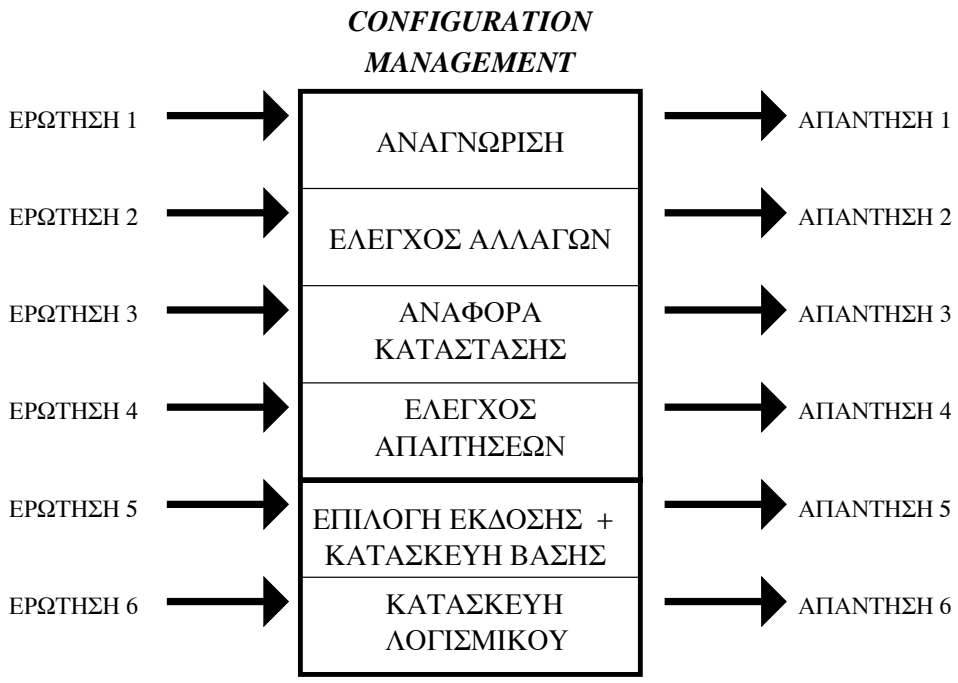
- ***Αυτό το πρόγραμμα δούλευε χθες. Τι συνέβη;***
- ***Δε μπορώ να εντοπίσω το λάθος σε αυτή τη διαμόρφωση.***
- ***Διόρθωσα αυτό το πρόβλημα πριν από πολύ καιρό. Γιατί εμφανίσθηκε πάλι.***
- ***Εχουμε την τελευταία έκδοση;***

2.1.2 Ελεγχος Αλλαγών

Ελέγχει την ολοκληρωμένη έκδοση ενός προϊόντος και τις αλλαγές σε αυτό καθ' όλη τη διάρκεια ζωής του, έχοντας ελέγχους σε σημεία, τα οποία εξασφαλίζουν συνεπές λογισμικό μέσω της δημιουργίας ενός βασικού προϊόντος.

Χωρίς έλεγχο αλλαγών προκύπτουν προβλήματα συντονισμού μεταξύ των μελών της ομάδας εργασίας, που οδηγούν σε προβλήματα όπως [2] :

Διπλή συντήρηση : Οταν υπάρχουν δύο αντίγραφα του ίδιου λογισμικού, πρέπει να συντηρούνται και τα δύο. Ετσι, όταν βρεθεί ένα λάθος, πρέπει κάποιος να θυμηθεί να το διορθώσει και στα δύο αντίγραφα. Αν αυτό αποτύχει, το λάθος αναδύεται ανεξάρτητα σε κάθε αντίγραφο



Σχήμα 2.1: Τα έξι συστατικά στοιχεία του configuration management

- ΕΡΩΤΗΣΗ1 : Ποια είναι η συγκρότηση του συστήματός μου;
  - ΑΠΑΝΤΗΣΗ1 : Το σύστημα αποτελείται από τα παρακάτω στοιχεία : *item1, item2, ... itemN.*
- ΕΡΩΤΗΣΗ2 : Πώς ελέγχω τις αλλαγές στη διαμόρφωσή μου;
  - ΑΠΑΝΤΗΣΗ2: Τα βήματα στην επεξεργασία των αλλαγών που επηρεάζουν, άμεσα ή έμμεσα, τη διαμόρφωση είναι : *step1, step2, ... stepM.*
- ΕΡΩΤΗΣΗ3 : Τι αλλαγές έχω κάνει στο σύστημά μου;
  - ΑΠΑΝΤΗΣΗ3 : Η διαμόρφωση του συστήματος και οι σχετικές αλλαγές αυτή τη στιγμή είναι : *(item1, item2, ... itemN) + (change1, change2, ... changeP, pending-change1, pending-change2, ... pending-changeQ).*
- ΕΡΩΤΗΣΗ4 : Το σύστημα που κατασκευάζω ικανοποιεί τις αρχικές προδιαγραφές;
  - ΑΠΑΝΤΗΣΗ4 : Η παρούσα κατασκευή διαφέρει από τις προσδιορισμένες ανάγκες ως εξής : *difference1, difference2, ... differenceR.*
- ΕΡΩΤΗΣΗ5 : Από ποιές εκδόσεις στοιχείων σχηματίζεται η X έκδοση του συστήματός μου;
  - ΑΠΑΝΤΗΣΗ5 : Αποτελείται από τις εκδόσεις : *version1, version2, ... versionN.*
- ΕΡΩΤΗΣΗ6 : Ποιές δοκιμές ελέγχου απομένει να γίνουν στη διαμόρφωσή μου;
  - ΑΠΑΝΤΗΣΗ6 : Οι έλεγχοι που θα εφαρμοσθούν είναι : *test1, test2, ... testL.*

και στη συνέχεια πρέπει να διαγνωσθεί και να διορθωθεί δύο φορές.

Κοινά Δεδομένα : Οταν προγραμματιστές τροποποιούν ένα, μοναδικό, αντίγραφο ενός προγράμματος, οι αλλαγές που γίνονται από έναν προγραμματιστή μπορεί να εμποδίζουν την πρόοδο των άλλων. Για παράδειγμα, όταν διορθωθεί ένα λάθος σε κώδικα που είναι κοινός μεταξύ μερικών ατόμων, η τροποποίηση μπορεί να προκαλέσει τη "συντριβή" ολόκληρου του συστήματος, οπότε όλοι σταματούν να δουλεύουν έως ότου διορθωθεί το νέο λάθος.

Ταυτόχρονη Τροποποίηση : Οταν δύο ή περισσότερα άτομα τροποποιούν το ίδιο λογισμικό, οι αλλαγές ενός μπορεί, από απροσεξία, να καταστρέψουν ή να συγκρούονται με τη δουλειά κάποιου άλλου. Λόγω του προβλήματος των κοινών δεδομένων, μία ομάδα κατασκευής λογισμικού δε μπορεί να εργάζεται άνετα με ένα μόνο αντίγραφο πηγαίου κώδικα, το οποίο μοιράζονται όλοι. Αλλά επίσης, δεν είναι εφικτό να υπάρχει ένα αντίγραφο για κάθε προγραμματιστή γιατί θα προκύψει το πρόβλημα της διπλής συντήρησης. Ετσι, πρέπει να υπάρχει συμφωνία και να χρησιμοποιείται μία πραγματοποιήσιμη λύση ώστε να αποφευχθεί το πρόβλημα της ταυτόχρονης υλοποίησης.

Μία πετυχημένη λειτουργία ελέγχου αλλαγών βοηθάει να αποτραπούν προβλήματα όπως τα παρακάτω [52] :

- ***Γιατί η αλλαγή μου σ' αυτό το σημείο εξαφανίσθηκε;***
- ***Τι συνέβη στα τμήματα που δεν είχα τελειώσει, κατά τη διάρκεια της απουσίας μου;***
- ***Πώς συγχωνεύω αυτές τις αλλαγές στην έκδοσή μου;***
- ***Οι αλλαγές μας συγκρούονται;***

2.1.3 Αναφορά Κατάστασης

Είναι ο μηχανισμός με τον οποίο τα αποτελέσματα εφαρμογής της αναγνώρισης, του ελέγχου αλλαγών και του ελέγχου απαιτήσεων συγκεντρώνονται, καταγράφονται και αναφέρονται, συνεχώς για όλα τα βασικά στοιχεία. Η λειτουργία της αναφοράς κατάστασης βοηθά στην απάντηση των παρακάτω ερωτήσεων [52] :

- ***Εχει διορθωθεί αυτό το πρόβλημα;***
- ***Ποιές διορθώσεις λαθών υπάρχουν σ' αυτό το αντίγραφο;***
- ***Αυτή η αλλαγή μοιάζει προφανής. Εχει δοκιμασθεί κάποια προηγούμενη φορά;***
- ***Ποιός είναι υπεύθυνος γι' αυτήν την τροποποίηση;***
- ***Πού συγχωνεύθηκαν αυτές οι ανεξάρτητες αλλαγές;***

2.1.4 Ελεγχος Απαιτήσεων

Κυρώνει την πληρότητα ενός προϊόντος και διατηρεί τη συνέπεια μεταξύ των τμημάτων του, εξασφαλίζοντας ότι τα τμήματα βρίσκονται σε μία κατάλληλη κατάσταση καθ' όλη τη διάρκεια της εργασίας και έτσι το προϊόν είναι μία καλά ορισμένη συλλογή από τμήματα. Η λειτουργία του ελέγχου απαιτήσεων παρέχει απαντήσεις στις ακόλουθες ερωτήσεις [47] :

- ***Προχωράει λογικά η ανάπτυξη του λογισμικού;***
- ***Η ανάπτυξη του λογισμικού προχωράει σύμφωνα με τις απαιτήσεις για το λογισμικό;***
- ***Εχει πραγματοποιηθεί ο σκοπός της προηγούμενης βάσης;***
- ***Ποια είναι η παρούσα κατάσταση του λογισμικού;***

2.1.5 Επιλογή Εκδοσης και Κατασκευή Βάσης

Η επιλογή σωστών εκδόσεων, στοιχείων και διαμορφώσεων, για έλεγχο και κατασκευή βάσης μπορεί να είναι δύσκολη (βλέπε ενότητα 2.3). Η επιλογή έκδοσης με τη χρήση υπολογιστών βοηθά στη σύνθεση σταθερών διαμορφώσεων και στην απάντηση των παρακάτω ερωτήσεων [52] :

- ***Πώς μπορώ να διαμορφώσω ένα σύστημα (για να ελέγξω την ορθότητα της λειτουργίας του) το οποίο θα περιέχει τις προσωρινές αλλαγές μου στην τελευταία βάση, και τις μόνιμες αλλαγές όλων των άλλων στοιχείων;***
- ***Εχοντας ως δεδομένο μία λίστα από τροποποιήσεις και επαυξήσεις, πώς μπορώ να διαμορφώσω ένα σύστημα που τις συγχωνεύει;***
- ***Αυτή η επαύξηση δε θα είναι έτοιμη έως την επόμενη ανακοίνωση έκδοσης συστήματος. Πώς μπορώ να τη διαμορφώσω έξω από την παρούσα βάση;***
- ***Πώς ακριβώς διαφέρει αυτή η έκδοση από τη βάση;***

2.1.6 Κατασκευή Λογισμικού

Η κατασκευή μίας διαμόρφωσης απαιτεί διάφορα βήματα, όπως : προ και μετα-επεξεργασία, compiling, linking, formatting και δοκιμές ελέγχου. Τα συστήματα για SCM πρέπει να αυτοματοποιούν αυτή τη διεργασία και για να ελλατώσουν την περιττή εργασία πρέπει να διαχειρίζονται μία δεξαμενή από πρόσφατα παραγόμενα στοιχεία. Με την αυτοματοποίηση αποφεύγουμε τα παρακάτω προβλήματα [52] :

- ***Μόλις το έφτιαξα αυτό. Μήπως κάτι δεν έχει περάσει από τον compiler;***
- ***Πόση θα είναι η επιβάρυνση αυτής της αλλαγής σε recompilation;***

- ***Παραδώσαμε την τελευταία εκτελέσιμη έκδοση στον πελάτη;***
- ***Αναρωτιέμαι κατά πόσο εφαρμόσαμε με τη σωστή σειρά τα βήματα επεξεργασίας.***
- ***Πώς ακριβώς παράχθηκε αυτή η διαμόρφωση;***
- ***Εφαρμόσθηκαν όλοι οι έλεγχοι σε αυτή την έκδοση;***

2.2 Εκδόσεις Στοιχείων

Τα συστήματα για SCM έχουν να αντιμετωπίσουν συνεχείς αλλαγές. Διορθώσεις, προσαρμογές και τελειοποιήσεις παράγουν μία σταθερή ροή από τροποποιήσεις πάνω σε λογισμικά αντικείμενα [2]. Επειδή οι περισσότερες αλλαγές είναι αυξητικές, μπορούν να εξεταοθούν καλύτερα με την παραγωγή συνδεόμενων εκδόσεων αντικειμένων παρά χωριστών ασυσχέτιστων αντικειμένων. Σε αυτή την ενότητα, αλλά και γενικότερα στην παρούσα εργασία, θα ασχοληθούμε με εκδόσεις πηγαίων αντικειμένων [3] (υπάρχουν και οι εκδόσεις των παραγώγων αντικειμένων [4]).

2.2.1 Ομάδες Εκδόσεων

Μία σημαντική έννοια για τη διαχείριση πολλαπλών εκδόσεων είναι η ομάδα εκδόσεων. Μία ομάδα εκδόσεων είναι ένα σύνολο από πηγαία αντικείμενα, τα οποία συνδέονται μέσω των σχέσεων *διασκευή-του* (*revision-of*) και *παραλλαγή-του* (*variant-of*). Οι σχέσεις αυτές ορίζονται [5] στη συνέχεια.

y διασκευή-του x : Αυτή η σχέση ισχύει αν και μόνο αν τα x και y είναι πηγαία αντικείμενα και το y έχει παραχθεί αλλάζοντας ένα αντίγραφο του x. Ετσι, η *διασκευή-του* καταγράφει το ιστορικό εξέλιξης των πηγαίων αντικειμένων.

Η σχέση *διασκευή-του* είναι μεταβατική, αντισυμμετρική και μη ανακλαστική. Τα αντικείμενα μίας ομάδας εκδόσεων που είναι συσχετισμένα με την εν λόγω σχέση ονομάζονται διασκευασμένες εκδόσεις ή απλώς διασκευές (*revisions*).

[2] Ενα **λογισμικό αντικείμενο** είναι οποιοδήποτε είδος αναγνωρίσιμου, αναγνώσιμου από μηχανή εγγράφου, το οποίο παράγεται κατά τη διάρκεια μιας εργασίας. Παραδείγματα αντικειμένων λογισμικού είναι : έγγραφα απαιτήσεων, προδιαγραφές, περιγραφές συστημάτων διασύνδεσης, κώδικας προγράμματος, προγράμματα ελέγχου, δεδομένα ελέγχου, δυαδικός κώδικας, εγχειρίδια για το χρήστη ή σχέδια VLSI [52].

[3] Ενα **πηγαίο αντικείμενο** είναι ένα αντικείμενο λογισμικού, το οποίο συντίθεται με το χέρι, για παράδειγμα, με ένα διαλογικό κειμενογράφο. Η δημιουργία ενός πηγαίου αντικειμένου απαιτεί ανθρώπινη ενέργεια και δε μπορεί να παραχθεί αυτόματα [52].

[4] Ενα **παράγωγο αντικείμενο** δημιουργείται αυτόματα από ένα πρόγραμμα (παραδείγματα τέτοιων προγραμμάτων είναι : compilers, linkers, formatters), συνήθως από άλλα αντικείμενα λογισμικού [52].

[5] Ο ορισμός της σχέσης *παραλλαγή-του* έχει δοθεί από τον Tichy στο [52]. Αξίζει να αναφέρουμε σε αυτό το σημείο ότι, στη συζήτηση για τις παραλλαγές εκδόσεων στο [53] κανείς δε μπόρεσε να δώσει έναν ορισμό γι' αυτή.

y παραλλαγή-του x : Αυτή η σχέση ισχύει αν και μόνο αν τα x και y είναι πηγαία αντικείμενα και δεν είναι διακρινόμενα κάτω από μία συγκεκριμένη αφαίρεση. Μία αφαίρεση καθορίζει τις ενδιαφέρουσες ιδιότητες ενώ αγνοεί τις λεπτομέρειες. Έτσι, με το να αγνοεί συγκεκριμένες ιδιότητες ή συμπεριφορές επιτρέπει παραλλαγές. Ο σκοπός είναι να οριοθούν αφαιρέσεις με τέτοιο τρόπο ώστε να μπορούν οι παραλλαγές εκδόσεων να αντικαταστήσουν η μία την άλλη σε συστήματα λογισμικού, χωρίς να απαιτούνται αλλαγές στα προγράμματα των πελατών.

Για μία δεδομένη αφαίρεση, η σχέση *παραλλαγή-έκδοσης-του* είναι μία σχέση ισοδυναμίας γιατί είναι μεταβατική, συμμετρική και ανακλαστική. Τα αντικείμενα μίας ομάδας εκδόσεων που είναι συσχετισμένα με αυτή τη σχέση ονομάζονται παραλλαγές (*variants*).

2.2.2 Πράξεις σε Ομάδες Εκδόσεων

Όλα τα συστήματα για SCM χρησιμοποιούν κάποια μέθοδο του κύκλου εξαγωγής / κειμενογράφησης / εισαγωγής για να προσθέσουν διασκευασμένες εκδόσεις (*revisions*) σε ομάδες εκδόσεων. Η πράξη της **εξαγωγής** (*check-out*) δημιουργεί ένα αντίγραφο της έκδοσης που πρόκειται να τροποποιηθεί και το κρατάει για το χρήστη, ο οποίος μπορεί να κάνει τις αλλαγές που θέλει με τη βοήθεια ενός κειμενογράφου. Κατά τη διάρκεια που το αντίγραφο τροποποιείται παραμένει απρόσιτο στους άλλους χρήστες. Η πράξη της **εισαγωγής** (*check-in*) σηματοδοτεί την ολοκλήρωση των αλλαγών και κάνει το (τροποποιημένο) αντίγραφο ορατό και στους υπόλοιπους χρήστες. Πριν γίνει η εισαγωγή μίας έκδοσης, πρέπει να ελέγχεται, ώστε να ικανοποιεί κάποια κριτήρια, για να μπορεί να χρησιμοποιηθεί με σιγουριά από τα άλλα μέλη της ομάδας.

2.3 Επιλογή Εκδόσεων

Ένα μοντέλο συστήματος (βλέπε ενότητα 3.1.7) μπορεί να απεικονίσει ένα μεγάλο αριθμό από εκδόσεις. Για παράδειγμα, ένα μεσαίου μεγέθους σύστημα λογισμικού μπορεί εύκολα να αποτελείται από 100 ομάδες εκδόσεων. Αν υποθέσουμε ότι κάθε ομάδα έχει μόνο δύο εκδόσεις, ένα μοντέλο συστήματος που περιέχει και τις 100 ομάδες απεικονίζει $2^{100} \approx 10^{30}$ διαφορετικές βάσεις - ένα τρομακτικά μεγάλο νούμερο. Στην πράξη, ελάχιστες από αυτές θα δουλέψουν. Το πρόβλημα επιλογής του configuration management είναι η εύρεση βιώσιμων διαμορφώσεων χωρίς εξαντλητική αναζήτηση.

Η επιλογή εκδόσεων είναι μία ενεργή περιοχή έρευνας μέσα στο CM, με την οποία

ασχοληθήκαμε και εμείς στην παρούσα εργασία. Η γενική προσέγγιση στο θέμα, από τα υπάρχοντα συστήματα για SCM, συσχετίζει περιορισμούς με μοντέλα συστημάτων, όπου οι περιορισμοί είναι συνθήκες πάνω σε γνωρίσματα αντικειμένων, οι οποίες επιλέγουν τις κατάλληλες διασκευές (*revisions*) και παραλλαγές (*variants*). Η προσέγγιση που προτείνουμε εμείς απεικονίζει τα μοντέλα συστημάτων με τη βοήθεια της γλώσσας παράστασης γνώσης TELOS [41], επιλέγει εκδόσεις με χρήση μίας άλγεβρας εκδόσεων [42] και αποφασίζει αν είναι αποδεκτή η διαμόρφωση (του συστήματος) που προκύπτει με τη βοήθεια περιορισμών μεταξύ εκδόσεων αντικειμένων.

Κεφάλαιο 3

Η Εξέλιξη του Software Configuration Management

Οπως προαναφέραμε στην Εισαγωγή, το SCM είναι μία από τις περιοχές έρευνας της οποίας η εξέλιξη ήταν ραγδαία από την δεκαετία του '80 και μετά, λόγω των πολλών αποτυχιών της βιομηχανίας παραγωγής λογισμικού κατά τη δεκαετία του '70, όπου στις αρχές και στα μέσα αυτής κατασκευάζονται δύο συστήματα (SCCS [44], Make [20]), το πρώτο μόνο για έλεγχο των εκδόσεων και το δεύτερο μόνο για αυτόματη κατασκευή συστημάτων, τα οποία αποτελούν τους προδρόμους των μετέπειτα συστημάτων για software configuration management.

Τα συστήματα [1] για SCM που έχουν αναπτυχθεί από τις αρχές της δεκαετίας του '80 μέχρι σήμερα, μπορούν να χωρισθούν σε τρεις τεχνολογικές γενεές [1] χρησιμοποιώντας ως κριτήρια τη χρήση διαφορετικών βασικών μεθόδων και τεχνολογιών για τη στήριξη των λειτουργιών του SCM. Ετσι, στην πρώτη γενεά κατατάσουμε το RCS/Make [51], [15] και συστήματα, όπως τα : CCC [55], DMS [11], ADC [54], ISTAR [16], [22], τα οποία βρίσκονται κοντά στο μοντέλο του RCS/Make, με κάποια επιπλέον γνωρίσματα. Αυτό που χαρακτηρίζει τα συστήματα της δεύτερης γενεάς, όπως τα : Cedar [49], [48], Gandalf [23], Rational [19], Mosaix [50], είναι το γνώρισμα του μοντέλου συστήματος [2] και η χρήση γλωσσών

[1] Αξίζει τον κόπο να αποσαφηνίσουμε την έννοια του συστήματος CM και του εργαλείου CM. Ενα **σύστημα** για CM είναι τμήμα ενός περιβάλλοντος, όπου η υποστήριξη για CM είναι αναπόσπαστο μέρος αυτού του περιβάλλοντος και στο εμπόριο διατίθεται ως τμήμα ενός πακέτου. Για παράδειγμα, ένα σύστημα CM είναι συχνά διαθέσιμο είτε σαν τμήμα ενός συστήματος PIM (*Product Information Management*) είτε σαν επέκταση σε λογισμικό CAD/CAM, όπως συμβαίνει στα παρακάτω εμπορικά συστήματα : TDMS, EDICS, EDICS/ECC, Adra Vault Drawing Management System, Auto-trl EIMS, EDL, EDC, CATIA, I/PDM, DMCS, Sherpa και PPDM [45], [46]. Ενα **εργαλείο** για CM είναι ένα αυτόνομο σύστημα. Για παράδειγμα το RCS είναι ένα εργαλείο CM, αφού, είναι σχεδιασμένο έτσι ώστε να τοποθετηθεί σ' ένα ήδη υπάρχον περιβάλλον. Στην παρούσα εργασία ο όρος σύστημα CM θα χρησιμοποιείται για την απόδοση και των δύο εννοιών.

[2] Ενα μοντέλο συστήματος δεν αναγνωρίζει πλήρως μία διαμόρφωση ενός συστήματος, αλλά προσδιορίζει

MIL (Module Interconnection Language) για την περιγραφή των χαρακτηριστικών και των περιεχομένων των στοιχείων ενός συστήματος. Τέλος, τα συστήματα της τρίτης γενεάς, όπως τα DSEE [32], [33], Adele [17], [18], Jasmine [38], CMA [43], shape [34], [35], [36], Gypsy [7], NSE [8], [40], χαρακτηρίζονται από τα γνωρίσματα γραμμής διαμόρφωσης [3] (*configuration thread*) και δεξαμενής αντικειμένων [4] (*object pool*), τη χρήση βάσεων δεδομένων και τον προσανατολισμό προς γλώσσες υψηλού επιπέδου, οι οποίες έχουν τη δυνατότητα περιγραφής σχέσεων μεταξύ των στοιχείων και μπορούν να παράγουν αυτόματα πληροφορία σχετική με τη σύνθεση των στοιχείων.

Στη συνέχεια του κεφαλαίου θα περιγράψουμε συγκεκριμένα γνωρίσματα συστημάτων CM τα οποία μπορούν να ιδωθούν σαν ένα φάσμα λειτουργιών, όπου καθε γνώρισμα χτίζεται πάνω σε κάποιο άλλο.

3.1 Φάσμα Λειτουργιών σε Συστήματα CM

Πριν αρχίσουμε την περιγραφή των γνωρισμάτων πρέπει να σημειώσουμε ότι τα συστήματα και τα γνωρίσματα που θα δούμε στην παρούσα ενότητα είναι αντιπροσωπευτικά του τι υπάρχει και δεν αποτελούν μία ολοκληρωμένη περίληψη ή εκτίμηση όλων των συστημάτων που υπάρχουν. Επειδή δεν υπάρχει κοινή ορολογία γύρω από το θέμα των αυτοματοποιημένων λειτουργιών CM έγινε μία επιλογή γνωρισμάτων, και όχι γενίκευσή τους, από συγκεκριμένα συστήματα για CM. Ενα χαρακτηριστικό παράδειγμα είναι το εξής : στο [52] χρησιμοποιείται ο όρος *cache* και στο [33] ο όρος *pool* για να δηλώσουν το ίδιο με αυτό που εμείς ορίζουμε ως *δεξαμενή αντικειμένων*.

3.1.1 Αποθήκη (Repository)

Το RCS παρέχει την έννοια της αποθήκης για αρχεία κειμένου, όπου στην πραγματικότητα, η αποθήκη είναι μία κεντρική βιβλιοθήκη από αρχεία, η οποία εξασφαλίζει έλεγχο εκδόσεων. Τα περισσότερα συστήματα για CM χρησιμοποιούν κάποιο είδος αποθήκης και διαχειρίζονται τις εκδόσεις των αποθηκευμένων στοιχείων.

τι κομμάτια πληροφορίας χρειάζονται για τη σύγκριση, το συνδυασμό και την αναγνώριση διαφόρων διαμορφώσεων [6].

[3] Γραμμή διαμόρφωσης : ορίζει ποια έκδοση από κάθε στοιχείο πρέπει να χρησιμοποιηθεί στην κατασκευή του συστήματος, και τις παραμέτρους που πρέπει να χρησιμοποιηθούν κατά τη μετάφραση κάθε στοιχείου [33].

[4] Δεξαμενή Αντικειμένων : περιέχει και διατηρεί τα αποτελέσματα της κατασκευής συστημάτων [33].

3.1.2 Κατανεμημένα Στοιχεία (Distributed Components)

Το DMS παρέχει την έννοια μίας κεντρικής αποθήκης, η οποία ελέγχει αρχεία κατανεμημένα σε διαφορετικά μηχανήματα. Το DMS είναι ενήμερο για την κατανομή και πραγματοποιεί το CM χωρίς να γίνεται γνωστή η κατανομή στους χρήστες, οι οποίοι εκτελούν την εργασία τους στην αποθήκη σα να ήταν τοποθετημένα όλα τα αρχεία στο δικό τους σταθμό εργασίας. Οταν είναι έτοιμοι να επιστρέψουν τα αρχεία που έχουν τροποποιήσει στην κύρια αποθήκη, χρησιμοποιούν τις δυνατότητες που παρέχει το DMS. Ετσι, μία ομάδα από χρήστες που είναι σκορπισμένοι γεωγραφικά μπορεί να εργάζεται πάνω στην ίδια διαμόρφωση αρχείων.

3.1.3 Φάση Κύκλου Ζωής (Lifecycle Phase)

Το σύστημα CCC παρέχει μερικές ιδέες για την υποστήριξη ενός συγκεκριμένου μοντέλου, όπου ο κύκλος ζωής ενός συστήματος (προϊόντος) χωρίζεται σε φάσεις. Συγκεκριμένα το CCC διαχωρίζει την ανάπτυξη, τον έλεγχο και την έκδοση ενός προϊόντος. Αυτός ο διαχωρισμός επιτρέπει διαφόρων ειδών χρήστες να εργάζονται, καθώς φαίνεται, στον ίδιο κώδικα την ίδια χρονική στιγμή. Αυτό επιτυγχάνεται περνώντας τον κώδικα δια μέσου των χωριστών directories, που συμβολίζουν την κάθε φάση. Η εργασία μπορεί να συνεχισθεί ανεξάρτητα σε κάθε directory, αλλά στην ουσία, προχωράει από φάση σε φάση, από άτομο σε άτομο. Οταν η ανεξάρτητη εργασία ολοκληρωθεί, όλες οι αλλαγές συγχωνεύονται σ' ένα τελικό προϊόν στην αποθήκη, ελέγχονται και εγκρίνονται. Μία νέα έκδοση μπορεί να δημιουργηθεί και ο κύκλος μπορεί να αρχίσει πάλι. Στην πραγματικότητα, το μοντέλο κύκλου ζωής επιτυγχάνεται μέσω παραλλήλων εργασιών σε εκδόσεις διαμορφώσεων.

3.1.4 Αιτήσεις Αλλαγής (Change Requests)

Στο CCC μία αίτηση αλλαγής μοιάζει με τη συμπλήρωση των κενών σε ένα έγγραφο αίτησης, μόνο που εδώ όλα γίνονται ηλεκτρονικά. Μία αίτηση αφορά αλλαγές που πρέπει να γίνουν σε ένα τμήμα του κώδικα ή σε πολλά συσχετιζόμενα κομμάτια του συστήματος (προϊόντος), όπως στα στοιχεία μιας διαμόρφωσης. Ο σκοπός της αλλαγής ορίζεται από τον χρήστη, αλλά το CCC καταγράφει κάθε αίτηση (στην οποία εκχωρείται ένα μοναδικό όνομα) και τι επηρεάζεται από την αλλαγή. Αφού συμπληρωθεί η φόρμα με όλες τις λεπτομέρειες που χρειάζονται, όπως ποια στοιχεία θα επηρεασθούν, ταχυδρομείται ηλεκτρονικά στο διαχειριστή της εργασίας, ο οποίος απαντά με έγκριση ή με απόρριψη της αίτησης. Αν εγκριθεί, το άτομο που είχε κάνει την αίτηση αναλαμβάνει να κάνει και τις αλλαγές. Οταν αυτές ελεγχθούν

και εγκριθούν μπορούν τα επηρεαζόμενα στοιχεία να περάσουν στην εγκριθείσα διαμόρφωση. Οι αιτήσεις αλλαγής παρέχουν το ιστορικό όλων των αλλαγών στην αποθήκη, μία αναφορά κατάστασης για αλλαγές που βρίσκονται σε εξέλιξη και τις απαιτήσεις που είχαν αλλαγές που έγιναν ή απορρίφθηκαν.

3.1.5 Συμβόλαια (Contracts)

Το περιβάλλον ISTAR παρέχει για μοντελοποίηση ορισμένα τμήματα μίας διεργασίας ανάπτυξης λογισμικού. Μοντελοποιεί τη ροή πληροφορίας και την ολοκλήρωση εργασιών, μέσω συμβολαίων [5]. Τα συμβόλαια ενσωματώνουν ροή πληροφορίας, ρόλους, εργασίες και κομμάτια του προϊόντος και είναι "ανταλλασσόμενα" μεταξύ των εργοληπτών. Ενα συμβόλαιο θεωρείται εκπληρωμένο με την ολοκλήρωση της κατασκευής μίας συγκεκριμένης έκδοσης του προϊόντος (ή τμήματος αυτού). Τότε τα κομμάτια περνούν σε μία νέα φάση κατασκευής, κατά την οποία μεταβιβάζεται, σε άλλον εργολήπτη, τμήμα της αποθήκης του προϊόντος.

3.1.6 Σύνολο Αλλαγών (Change Set)

Το σύστημα ADC προσπαθώντας να διευκολύνει τη δημιουργία μίας συγκεκριμένης έκδοσης ενός συστήματος (προϊόντος) ή τμήματος αυτού, τροποποιεί το μηχανισμό της αποθήκης αρχείων με έλεγχο εκδόσεων επεκτείνοντας την έννοια της δέλτα, στην οποία αποδίδεται ένα όνομα και ο χρήστης ορίζει μία μέθοδο με την οποία υποδεικνύει ποιές δέλτα (για την έννοια της δέλτα βλέπε ενότητα 4.1.1, παράγραφος 4) πρέπει να συνδυασθούν για την κατασκευή μίας συγκεκριμένης έκδοσης ενός αρχείου ή ενός τμήματος του προϊόντος. Ο συνδυασμός των δέλτα, τα αρχεία στα οποία εφαρμόζονται και οι αιτίες για κάθε δέλτα συγκροτούν ένα σύνολο αλλαγών. Δηλαδή, ο χρήστης μπορεί να ορίσει ένα σύνολο το οποίο περιέχει όλες τις αλλαγές σε μία ομάδα αρχείων και τους λόγους για τους οποίους πρέπει να γίνει κάθε μία · το ADC καταγράφει τις αντιστοιχίες δέλτα - αρχείων στην ομάδα. Ετσι, ο χρήστης μπορεί να χειρίζεται την ομάδα σαν μία οντότητα, να εκτελεί πράξεις σ' αυτή, και να δημιουργήσει, οποιαδήποτε στιγμή, μία έκδοση του προϊόντος ορίζοντας μία έκδοση βάσης και ένα συνδυασμό από σύνολα αλλαγών.

[5] Συμβόλαιο είναι ένα καλά ορισμένο πακέτο εργασίας η οποία μπορεί να εκτελεσθεί ανεξάρτητα, από έναν εργολήπτη για έναν πελάτη.

3.1.7 Μοντέλο Συστήματος (System Model)

Ο Cedar System Modeller είναι ένα από τα πρώτα συστήματα στα οποία ορίζεται σαφώς η δομή μιας διαμόρφωσης, δηλαδή ένα σύνολο από συσχετιζόμενα εξαρτήματα τα οποία ορίζουν ένα υπό ανάπτυξη προϊόν. Ο System Modeller καταγράφει τις αλλαγές στα εξαρτήματα και ελέγχει το compliling και το loading των διαμορφώσεων. Το μοντέλο συστήματος ξεκαθαρίζει στο περιβάλλον ανάπτυξης του λογισμικού ποια εξαρτήματα είναι συσχετιζόμενα και τι σχέσεις υπάρχουν μεταξύ τους. Εχει μία αναγνώσιμη αναπαράσταση κειμένου η οποία μπορεί να τροποποιηθεί από ένα χρήστη οποιαδήποτε στιγμή και να χρησιμοποιηθεί από εργαλεία όπως browsers, debuggers και inter-module analyzers. Στην πραγματικότητα, το περιβάλλον μπορεί να χρησιμοποιήσει το μοντέλο για να αναλύσει τα εξαρτήματα και να σιγουρευθεί για το ότι το προϊόν βρίσκεται σε μία λογική κατάσταση. Για παράδειγμα, μπορεί ένας χρήστης αλλάζοντας ένα αρχείο να καταστήσει άκυρα κάποια άλλα σχετιζόμενα αρχεία. Ο System Modeller αναγνωρίζει αυτόματα τέτοιες περιπτώσεις και μπορεί να περάσει πάλι από τον compiler τα σχετιζόμενα αρχεία, δημιουργώντας με αυτόν τον τρόπο μία νέα τελευταία έκδοση του προϊόντος. Επίσης, ορίζοντας τη στατική δομή ενός προϊόντος, ο System Modeller επιτρέπει, να βοηθήσει το περιβάλλον στη διατήρηση της ακεραιότητας του προϊόντος.

3.1.8 Υποσυστήματα (Subsystems)

Το περιβάλλον Rational επαυξάνει την έννοια του μοντέλου συστήματος και αποθήκης με την ενσωμάτωση γνωρισμάτων που έχουν σχέση με πολύ μεγάλα συστήματα (προϊόντα). Το Rational παρέχει τη δυνατότητα διαίρεσης ενός μεγάλου προϊόντος Ada σε τμήματα, επιτυγχάνοντας έτσι τον περιορισμό του πεδίου επίδρασης των αλλαγών. Τα τμήματα αυτά ονομάζονται υποσυστήματα, έχουν περιγραφές συστήματος διασύνδεσης και συμβολίζουν ένα σημείο διαμόρφωσης (γι' αυτό θεωρούνται ως μία οντότητα). Οι χρήστες μπορούν να εργάζονται σε ένα υποσύστημα κάνοντας τροποποιήσεις και εφόσον δεν αλλάζει κάποια περιγραφή συστήματος διασύνδεσης κάθε recompilation γίνεται μόνο στα στοιχεία που περιλαμβάνονται στο υποσύστημα. Αλλαγές στο σύστημα διασύνδεσης απαιτούν, πιθανώς, να γίνει recompiled ολόκληρο το προϊόν. Τα υποσυστήματα έχουν έλεγχο εκδόσεων στα στοιχεία τους, αλλά και τα ίδια μπορεί να είναι μια συγκεκριμένη έκδοση, ελαχιστοποιώντας με αυτόν τον τρόπο την ανάγκη για recompiling ολόκληρου του προϊόντος. Επίσης, λόγω της περιγραφής συστήματος διασύνδεσης, ο χρήστης μπορεί να αναμίξει και να ταιριάξει υποσυστήματα, κατασκευάζοντας μία συγκεκριμένη έκδοση του προϊόντος. Το Rational ελέγχει και επιτρέπει το συνδυασμό μόνο συμβατών εκδόσεων υποσυστημάτων. Η έννοια του υποσυστήματος

παρουσιάζει έναν τρόπο περιορισμού του πεδίου αλλαγών που έχουν οι χρήστες και δίνει τη δυνατότητα στο περιβάλλον να ελέγχει την εγκυρότητα των συνδυαζομένων διαμορφώσεων.

3.1.9 Γραμμή Διαμόρφωσης (Configuration Thread)

Το Domain Software Engineering Environment (DSEE) επαυξάνει την έννοια του μοντέλου συστήματος ενσωματώνοντας παραγόμενα αντικείμενα (με άλλα λόγια object αρχεία) και εργαλεία που κάνουν την παραγωγή στο μοντέλο. Αυτό επιτρέπει στο χρήστη να συλλάβει με περισσότερη ακρίβεια τα στατικά εξαρτήματα που σχετίζονται με την κατασκευή ενός συστήματος (προϊόντος). Αυτά τα εξαρτήματα περιλαμβάνουν : τις εκδόσεις των εργαλείων, τις παραμέτρους που χρησιμοποιούνται για την κλήση των εργαλείων και τις εκδόσεις των εξαρτημάτων, μαζί με άλλες, συγκεκριμένες, παραμέτρους που αξίζει να καταγραφούν. Καθώς το προϊόν περνάει απο τον compiler, το DSEE καταγράφει, αυτόματα, τις εκδόσεις των εξαρτημάτων που χρησιμοποιούνται και κρατάει αυτή την πληροφορία στην αποθήκη σ' ένα αντικείμενο που είναι γνωστό ως γραμμή διαμόρφωσης, στο οποίο αποδίδεται ένα μοναδικό αναγνωριστικό. Το αναγνωριστικό μπορεί να χρησιμοποιηθεί στη συνέχεια από το DSEE σαν το μόνιμο, ολοκληρωμένο ιστορικό των εξαρτημάτων, απαραίτητο στην κατασκευή του προϊόντος. Οποιαδήποτε χρονική στιγμή, ο χρήστης μπορεί να ζητήσει τη δημιουργία μίας συγκεκριμένης έκδοσης του προϊόντος. Επίσης, οι χρήστες δε χρειάζεται, πλέον, να διατηρούν πολλαπλά αντίγραφα των ίδιων εκδόσεων, αρχείων και εργαλείων, και ενδιάμεσα αρχεία σε κάποια κεντρική αποθήκη · το DSEE διαθέτει ένα πλήρη κατάλογο όλων των εξαρτημάτων, τις εκδόσεις τους και το χρόνο δημιουργίας τους, περιορίζοντας έτσι την ανάγκη για πολλαπλά αντίγραφα και συνεπώς τη σπατάλη χώρου. Σχεδόν όλη η διαχείριση των παραγόμενων αντικειμένων, όπως και των πηγαίων, βρίσκεται κάτω από τον έλεγχο του συστήματος CM.

3.1.10 Δεξαμενή Στοιχείων (Object Pool)

Χρησιμοποιώντας τις έννοιες του μοντέλου συστήματος και γραμμής διαμόρφωσης, το DSEE έχει όλη την αναγκαία πληροφορία για να αναγνωρίσει τι απαιτείται για τη δημιουργία μίας συγκεκριμένης έκδοσης ενός παραγόμενου αντικειμένου. Συσχετίζοντας αυτή την πληροφορία με κάθε παραγόμενο αντικείμενο, όλα τα αντικείμενα μπορούν να τοποθετηθούν μέσα σε μία δεξαμενή. Ετσι, οποιαδήποτε στιγμή ο χρήστης χρειάζεται ένα συγκεκριμένο (ή ένα συμβατό με αυτό) παραγόμενο αντικείμενο, το DSEE μπορεί να ελέγξει αυτόματα την ύπαρξη του στη δεξαμενή, οπότε, αν υπάρχει αποτρέπεται η εκ νέου δημιουργία του. Ο χρήστης δε

χρειάζεται να γνωρίζει ότι το συγκεκριμένο αντικείμενο υπάρχει · το DSEE κάνει όλο τον έλεγχο βασιζόμενο στην πληροφορία κατασκευής που παίρνει από αυτόν και στις υπάρχουσες γραμμές διαμόρφωσης. Με αυτόν τον τρόπο ελλατώνεται ο χρόνος που απαιτείται για compiling και αναχρησιμοποιείται εργασία που έχει ήδη γίνει. Ουσιαστικά, το CM σύστημα γνωρίζει πώς να αναδημιουργήσει, με ακρίβεια και βέλτιστο τρόπο, μία συγκεκριμένη έκδοση του προϊόντος.

3.1.11 Γνωρίσματα (Attributes)

Το σύστημα Adele γενικεύει τις έννοιες της αποθήκης, μοντέλου συστήματος και γραμμής διαμόρφωσης χρησιμοποιώντας μία βάση δεδομένων και δυνατότητες μοντελοποίησης δεδομένων. Ετσι, η αποθήκη γίνεται περισσότερο οντοκεντρική και ο χρήστης μπορεί να περιγράψει ένα σύστημα (προϊόν) με όρους ενός μοντέλου δεδομένων. Στην ουσία, τα εξαρτήματα ενός προϊόντος ορίζονται ως αντικείμενα της βάσης δεδομένων με γνωρίσματα και σχέσεις. Τα γνωρίσματα είναι πληροφορία που ορίζεται από τον χρήστη σχετική με ένα αντικείμενο, για παράδειγμα, το αντικείμενο A είναι ένα παραγόμενο αντικείμενο κατάλληλο για VMS σύστημα. Οι σχέσεις ορίζουν εξαρτήσεις μεταξύ των αντικειμένων, όπως, το αντικείμενο A εξαρτάται από το B. Ετσι, ο χρήστης μπορεί να περιγράψει μία διαμόρφωση από την άποψη των χαρακτηριστικών των αντικειμένων και όχι από την άποψη της σύνθεσης συγκεκριμένων εκδόσεων των αντικειμένων. Δηλαδή, με τη χρήση κανόνων επιλογής και περιορισμών γύρω από τα γνωρίσματα, το σύστημα μπορεί να συνθέσει μία κατάλληλη διαμόρφωση. Με αυτόν τον τρόπο, ο χρήστης μπορεί να ορίσει ένα προϊόν σε ένα υψηλότερο επίπεδο αφαίρεσης (και όχι μόνο με τη μορφή μίας λίστας αρχείων).

3.1.12 Ελεγχος Συνέπειας (Consistency Checking)

Το Configuration Management Assistant (CMA) επαυξάνει τη μοντελοποίηση δεδομένων για configuration management με την προσθήκη ορισμένων κλάσεων γνωρισμάτων και σχέσεων, στις οποίες βασίζεται το CMA για να ελέγξει και να αποφασίσει κατά πόσο μία διαμόρφωση (η οποία είναι ένα σύνολο αντικειμένων) είναι έγκυρη. Η εγκυρότητα έγκειται στο ότι όλα τα αντικείμενα στη διαμόρφωση μπορούν να συνδυασθούν για την κατασκευή ενός ορθού συνόλου, δηλαδή υπάρχει συνέπεια μεταξύ τους. Οι κλάσεις των γνωρισμάτων αντιπροσωπεύουν χαρακτηριστικά που ορίζονται από το χρήστη συμπεριλαμβάνοντας περιορισμούς, καθορισμούς τύπων και εκδόσεις διαμορφώσεων. Οι κλάσεις των σχέσεων αντιπροσωπεύουν σχέσεις : στοιχείων, μελών, λογικής και κληρονομήσιμης εξάρτησης, συνέπειας

και συμβατότητας. Αυτές επιτρέπουν στο CMA να εξασφαλίσει ότι όλα τα μέλη των αντικειμένων έχουν συμβατά χαρακτηριστικά και σχέσεις, στο σχηματισμό μίας διαμόρφωσης. Συγκεκριμένα, το CMA ελέγχει και αποφασίζει το κατά πόσο διάφορες διαμορφώσεις είναι ολοκληρωμένες, συνεπείς, σαφείς και περιλαμβάνουν τις σωστές εκδόσεις. Με το σχηματισμό μίας διαμόρφωσης, το CMA μπορεί να προσθέσει τα χαρακτηριστικά της στη βάση δεδομένων του, οπότε διαθέτει (το CMA) αρκετή πληροφορία για να ελέγξει τη συνέπεια της, σε μία μεταγενέστερη χρήση αυτής ή τμημάτων της. Ετσι, ο χρήστης μπορεί να βασιστεί στο σύστημα για την αναγνώριση ασυνεπειών και τη συντήρηση τής συνέπειας κατά τη δημιουργία και την αναχρησιμοποίηση διαμορφώσεων.

3.1.13 Χώρος Εργασίας (Workspace)

Το σύστημα shape συνδυάζει τις έννοιες της αποθήκης, μοντέλου συστήματος, γραμμής διαμόρφωσης και δεξαμενής αντικειμένων και προσθέτει σ' αυτές την έννοια του χώρου εργασίας. Χώροι εργασίας ανήκουν σε κάθε χρήστη και είναι σχεδιασμένοι ώστε να εμποδίζουν τους χρήστες να ανακατεύονται στη δουλειά άλλων. Ο χρήστης ορίζει ένα χώρο εργασίας, ο οποίος είναι σχεδιασμένος να αντιπροσωπεύει μία συγκεκριμένη κατάσταση μίας διαμόρφωσης, και εισάγει αρχεία από την κεντρική αποθήκη. Ολη η εργασία, όπως το γράψιμο κώδικα και το compiling, μπορεί να γίνει μέσα στο χώρο εργασίας χωρίς να επηρεάζει οτιδήποτε από όσα έχουν κάνει οι άλλοι χρήστες στους δικούς τους χώρους. Οταν ο χρήστης τελειώσει τις αλλαγές στη διαμόρφωση, και αφού εγκριθούν, εξάγεται (η διαμόρφωση) πίσω στην κεντρική αποθήκη.

Ο χώρος εργασίας δεν είναι ένα απλό directory αφού παρέχονται δυνατότητες ελέγχου εκδόσεων μέσα σε αυτόν και όχι μόνο στο γενικό επίπεδο της κοινής αποθήκης. Επιπλέον, μόνο ο χρήστης ή καθορισμένοι χρήστες, συσχετισμένοι με το χώρο εργασίας μπορούν να έχουν πρόσβαση σε αυτόν, και η πρόσβαση σε μία διαμόρφωση γίνεται μόνο μέσα από ένα χώρο εργασίας και όχι από οποιοδήποτε directory.

3.1.14 Διάφανη Οψη (Transparent View)

Το σύστημα Gypsy επαυξάνει την έννοια του χώρου εργασίας με μία αποθήκη διάφανης όψης. Αυτό σημαίνει ότι ο χρήστης θα μπορεί να δει μέσα στο χώρο εργασίας μόνο τις εκδόσεις των αρχείων για τις οποίες ενδιαφέρεται (δηλαδή, η επιλογή των εκδόσεων γίνεται διάφανα), ενώ όλες οι άλλες εκδόσεις είναι κρυμμένες, αν και υπάρχουν. Ετσι, ο χώρος εργασίας μοιάζει με μία ειδικευμένη αποθήκη για το χρήστη. Τα αρχεία που ανακαλούνται από την κύρια

αποθήκη τοποθετούνται στο χώρο εργασίας και εκχωρείται στους χρήστες πρόσβαση σε αυτόν, οπότε στην πραγματικότητα, τα αρχεία ανήκουν στο χώρο εργασίας που βρίσκονται και όχι σε ένα χρήστη, ο οποίος μπορεί να πάρει ένα στιγμιότυπο εκδόσεων, αρχείων τοπικών στο χώρο εργασίας. Η ονομασία των αρχείων είναι τοπική και όχι μοναδική καθολικά. Η διάφανη όψη δρα σαν μία τοπική αποθήκη και σαν μηχανισμός προστασίας απέναντι στον υπερβολικό φόρτο πληροφορίας και σε προσβάσεις σε μη τοπικές εκδόσεις αρχείων, που δεν έχουν εγκριθεί.

3.1.15 Transaction

Η έννοια της transaction στο NSE είναι μία επαύξηση των εννοιών του χώρου εργασίας και της διάφανης όψης. Εκφράζει μία μονάδα εργασίας και υποστηρίζει : απομόνωση της εργασίας των χρηστών, αλληλεπιδράσεις μεταξύ τους, συγχώνευση των αλλαγών που βασίζονται στη δομή του συστήματος (προϊόντος). Η transaction αποτελείται από ένα *περιβάλλον*, το οποίο παρέχει τα γνωρίσματα του χώρου εργασίας και της διάφανης όψης, και ένα σύνολο εντολών, οι οποίες αναπαριστούν ένα πρωτόκολλο που χρησιμοποιείται για το συντονισμό των ενεργειών μεταξύ χρηστών και για να εκφράσει την επικοινωνία λόγω των πραγματικών αλλαγών. Οι χρήστες εργάζονται ανεξάρτητα, καθένας στο περιβάλλον του, αλλάζοντας τα ίδια ή διαφορετικά κομμάτια του προϊόντος, και μπορούν να τροποποιήσουν την αποθήκη, προσθέτοντας μία νέα έκδοση ενός αρχείου που άλλαξε στο περιβάλλον που δουλεύουν. Το NSE βοηθά στη συγχώνευση των αλλαγών στην αποθήκη, αλλά ελέγχει αν αυτό που υπάρχει, την παρούσα στιγμή, συγκρούεται με τις νέες αλλαγές. Αν υπάρχει σύγκρουση, το NSE ειδοποιεί το χρήστη σχετικά με τα προβλήματα της συγχώνευσης και παρέχει βοήθεια για την εξάλειψη των συγκρούσεων.

Οπως οι χρήστες μπορούν να τροποποιήσουν την αποθήκη με τη συγχώνευση αλλαγών από το χώρο εργασίας, μπορούν να ζητήσουν να συγχωνευτούν στους τοπικούς τους χώρους εργασίας όλες οι αλλαγές που έγιναν από έναν άλλο χρήστη (ο οποίος μπορεί να έχει ήδη τροποποιήσει την αποθήκη). Με αυτόν τον τρόπο, η transaction δίνει τη δυνατότητα σε ομάδες να αλλάζουν τα ίδια ή συσχετιζόμενα κομμάτια του προϊόντος.

Κεφάλαιο 4

Συστήματα για Software Configuration Management

Στο προηγούμενο κεφάλαιο μιλήσαμε για την εξέλιξη του SCM και τη χωρίσαμε σε τρεις τεχνολογικές γενεές. Στο παρόν κεφάλαιο θα εξετάσουμε αναλυτικά ένα αντιπροσωπευτικό σύστημα από κάθε γενεά. Ετσι, από την πρώτη γενεά θα δούμε το RCS/Make, γιατί οι εντολές και γενικότερα η διαχείριση εκδόσεων του RCS αποτελεί τη βάση για όλα σχεδόν τα επόμενα συστήματα. Από τη δεύτερη το Gandalf, γιατί παρ' όλο που έχει τους ίδιους περίπου προσανατολισμούς με το Cedar, που όπως είδαμε είναι ένα από τα πρώτα συστήματα που εισάγουν την έννοια του Μοντέλου Συστήματος (ενότητα 3.1.7), αποτελεί υπερσύνολο του. Τέλος, από την τρίτη το shape γιατί, πρώτον ο σχεδιασμός του έχει επηρεασθεί από τα DSEE και Adele που είναι δύο συστήματα με πολύ σημαντικό ρόλο στην εξέλιξη του SCM, και δεύτερον, γιατί είχαμε στη διάθεση μας έναν ευρύ αριθμό από δημοσιεύσεις (σε σχέση με όλα τα άλλα συστήματα που αναφερόμαστε) σχετικές με το σύστημα shape ([31], [34], [35], [36], [37]).

4.1 Πρώτη Γενεά (RCS)

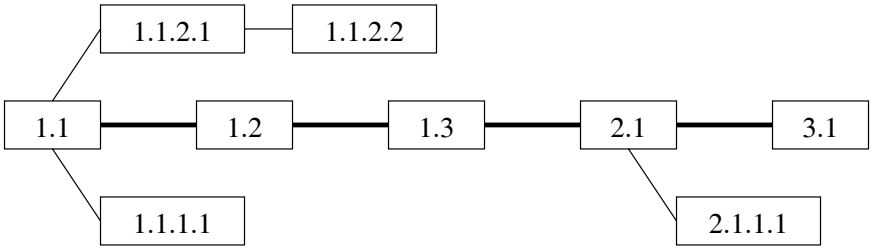
Βασική λειτουργία του *RCS (Revision Control System)* συστήματος είναι η διαχείριση ομάδων εκδόσεων (ο όρος που χρησιμοποιείται από τον Tichy είναι *revision groups*). Μία ομάδα εκδόσεων είναι ένα σύνολο από αρχεία κειμένου (ο όρος που χρησιμοποιείται είναι *revisions*) τα οποία εξελίσσονται το ένα από το άλλο, δηλαδή, ένα νέο αρχείο, θα περιέχει τις αλλαγές που έχουν γίνει σε κάποιο άλλο, παλιότερο, αρχείο της ομάδας. Το RCS οργανώνει τις εκδόσεις σε γενεολογικά δένδρα, όπου μία καινούργια έκδοση μπορεί να δημιουργηθεί

με την τροποποίηση μιας ήδη υπάρχουσας. Εκτός από τη διαχείριση ομάδων εκδόσεων, το RCS παρέχει λειτουργίες επιλογής για τη σύνθεση διαμορφώσεων. Αυτό το επιτυγχάνει σε συνδυασμό με το Make.

4.1.1 Δένδρο Εκδόσεων

Οπως είπαμε στην προηγούμενη παράγραφο το RCS οργανώνει τις εκδόσεις ενός στοιχείου σε ένα δένδρο προγόνων, το οποίο κατασκευάζεται με χρήση του μηχανισμού εισαγωγής (*check-in*). Ενα δένδρο εκδόσεων αποτελείται από τον κορμό και τα κλαδιά. Η πρώτη έκδοση είναι η ρίζα του και φέρει, συνήθως, τον αριθμό 1.1 · οι εκδόσεις (κόμβοι) που ακολουθούν και βρίσκονται πάνω στον κορμό παίρνουν τους αριθμούς 1.2, 1.3 κ.λπ. Το πρώτο πεδίο σε αυτή την αρίθμηση ονομάζεται αριθμός έκδοσης (*release number*) και το δεύτερο αριθμός επιπέδου (*level number*). Το σύστημα (και συγκεκριμένα ο μηχανισμός εισαγωγής) αυξάνει αυτόματα μόνο τον αριθμό επιπέδου. Ο αριθμός έκδοσης αλλάζει μόνο με παρέμβαση του χρήστη.

Ενα RCS δένδρο, όπως είπαμε, εκτός από τον κορμό μπορεί να έχει και κλαδιά, τα οποία αναπτύσσονται από τους κόμβους του κορμού. Τα κλαδιά χρησιμεύουν σε διάφορες περιπτώσεις, όπως για παράδειγμα, όταν θέλουμε να τροποποιήσουμε, όχι την τελευταία, αλλά κάποια παλιότερη έκδοση ενός στοιχείου ή όταν θέλουμε να διερευνήσουμε μία διαφορετική υλοποίηση, παράλληλα με την κύρια γραμμή ανάπτυξης. Ενα παράδειγμα δένδρου εκδόσεων του RCS παρουσιάζεται στο σχήμα 4.1. Για την αρίθμηση των εκδόσεων που βρίσκονται σε



Σχήμα 4.1: Ενα RCS μοντέλο

κλαδιά προστίθενται άλλοι δύο αριθμοί, από τους οποίους ο πρώτος δηλώνει τον αριθμό του κλαδιού και ο δεύτερος τη σειρά των εκδόσεων, που βρίσκονται στο συγκεκριμένο κλαδί.

Κάθε κόμβος σ' ένα δένδρο εκδόσεων αποτελείται από τα ακόλουθα γνωρίσματα : έκδοση, ημερομηνία και ώρα εισαγωγής στην αποθήκη, όνομα συγγραφέα, μία μικρή περιγραφή του περιεχομένου του, κατάσταση και το πραγματικό κείμενο. Ολα τα γνωρίσματα καθορίζονται

τη στιγμή που η έκδοση εισάγεται στην αποθήκη (με τον check-in μηχανισμό). Το γνώρισμα κατάσταση υποδηλώνει την κατάσταση της έκδοσης, και τίθεται αυτόματα στην κατάσταση Exp (*experimental*). Ο χρήστης μπορεί αργότερα να την αλλάξει επιλέγοντας ανάμεσα στις : Test (*tested*), Rel (*Released*) και Fail (*Failed*).

Για να έχει μικρότερη κατανάλωση χώρου, το RCS αποθηκεύει τις εκδόσεις σαν *δέλτα* [1] , δηλαδή κρατάει μόνο τις διαφορές μεταξύ εκδόσεων του ίδιου στοιχείου. Το σύστημα διασύνδεσης με το χρήστη κρύβει ολοκληρωτικά το γεγονός ότι το RCS είναι υλοποιημένο με δέλτα. Μία δέλτα είναι μία ακολουθία από εντολές που μετατρέπουν έναν ορμαθό χαρακτήρων σε έναν άλλο. Οι δέλτα που χρησιμοποιούνται από το RCS διαχειρίζονται γραμμές χαρακτήρων, οπότε οι μόνες εντολές που επιτρέπονται είναι εισαγωγή και διαγραφή γραμμών.

Το κλασικό πρόβλημα που προκύπτει με τη χρήση δέλτα είναι ότι ενώ ελλατώνεται ο χώρος που χρησιμοποιείται, αυξάνεται ο χρόνος πρόσβασης. Το RCS αντιμετωπίζει τις δέλτα με τον ακόλουθο τρόπο : η τελευταία έκδοση στον κορμό αποθηκεύεται ανέπαφη. Όλες οι άλλες εκδόσεις του κορμού αποθηκεύονται ως αντίστροφες δέλτα. Μία αντίστροφη δέλτα περιγράφει πώς μπορούμε να πάμε προς τα πίσω στην ιστορία ανάπτυξης · παράγει την επιθυμητή έκδοση εάν εφαρμοσθεί στο διάδοχο αυτής. Αντίθετα για τα κλαδιά, χρησιμοποιεί προς τα εμπρός δέλτα. Η δημιουργία μίας έκδοσης σε κλαδί γίνεται ως εξής : επιλέγει την τελευταία έκδοση στον κορμό, εφαρμόζει αντίστροφες δέλτα μέχρι να φτάσει στην έκδοση του κορμού που ανήκει το κλαδί και στη συνέχεια εφαρμόζει προς τα εμπρός δέλτα για να πάρει την επιθυμητή έκδοση.

Όλες οι εκδόσεις ενός στοιχείου αποθηκεύονται στο ίδιο αρχείο (RCS αρχείο) και για να περιορίσει το σωρό των αρχείων στο directory που εργάζεται ένας χρήστης, όλα τα RCS αρχεία μπορούν να τοποθετηθούν σε ένα directory, που ονομάζεται RCS, και οι εντολές του RCS ψάχνουν πρώτα σε αυτό για RCS αρχεία.

Το RCS έχει επίσης τη δυνατότητα να "κλειδώνει" ένα αρχείο, δηλαδή, αν ένας χρήστης τροποποιεί μία έκδοση ενός στοιχείου δε μπορεί και ένας άλλος χρήστης να κάνει αλλαγές στην ίδια έκδοση, παρά μόνο αν ο πρώτος "ξεκλειδώσει" το αρχείο. Τέλος, μπορεί να συγχωνεύσει δύο εκδόσεις ενός στοιχείου, σε σχέση με έναν κοινό πρόγονο.

[1] Μία άλλη τεχνική αποθήκευσης εκδόσεων (προτείνεται από τους Katz και Lehman στο [25]) χρησιμοποιεί δομές αποθήκευσης που βασίζονται σε B-δένδρα, για την κωδικοποίηση εκδόσεων ως "αρνητικών" αρχείων διαφορών.

4.1.2 Configuration Management

Για το χειρισμό του SCM, το RCS παρέχει ένα σύστημα διασύνδεσης με το Make. Μία περιγραφή του συστήματος που θέλουμε να κατασκευάσουμε είναι ένα σύνολο από εκδόσεις, που ανήκουν σε διαφορετικές ομάδες εκδόσεων. Κάθε έκδοση επιλέγεται με βάση ένα σύνολο από κριτήρια. Οι μηχανισμοί επιλογής βασίζονται στα χαρακτηριστικά : έκδοση που ορίζεται από το χρήστη, κατάσταση, όνομα συγγραφέα, ημερομηνία, ώρα και προκαθορισμένη έκδοση.

Για να επιτευχθεί η συνεργασία RCS και Make, το δεύτερο έχει επαυξηθεί με ένα μηχανισμό με τον οποίο, όταν ένα αρχείο που πρέπει να επεξεργασθεί το Make δεν είναι στο παρόν directory, το επιλέγει από το RCS directory και στη συνέχεια εκτελεί τη ζητούμενη επεξεργασία. Οι παράμετροι επιλογής που χρησιμοποιούνται για τον παραπάνω μηχανισμό μπορούν να περάσουν σαν παράμετροι στο Make ή να τοποθετηθούν απ' ευθείας στο *Makefile*. Αν όμως το αρχείο είναι παρόν, το Make αγνοεί εντελώς το αντίστοιχο RCS αρχείο.

4.2 Δεύτερη Γενεά (Gandalf)

Η εργασία Gandalf (*Gandalf project*) παρέχει ένα σύνολο μηχανισμών που χρησιμοποιούνται για να παράγουν περιβάλλοντα ανάπτυξης λογισμικού [2], τα οποία συνδυάζουν και τις δύο έννοιες : του περιβάλλοντος προγραμματισμού [3] και του περιβάλλοντος ανάπτυξης συστημάτων [4]. Η δημιουργία ενός συστήματος στο Gandalf αποτελείται από δύο κομμάτια · περιγραφή των λειτουργιών του συστήματος και έλεγχο των εκδόσεων του συστήματος. Το περιβάλλον, μέσα στο οποίο παράγεται το σύστημα, ελέγχει τη συνέπεια των περιγραφών του συστήματος, παρέχει αυτόματη παραγωγή των εκδόσεων του, διατηρεί τη συνέπεια της κατάστασης της εργασίας και εγγυάται κατάλληλη πρόσβαση στη βάση δεδομένων αυτής.

Εχουν σχεδιασθεί και υλοποιηθεί αρκετά περιβάλλοντα Gandalf. Τρία από τα πιο σημαντικά είναι : το Gandalf Prototype (GP), ένα ολοκληρωμένο Gandalf περιβάλλον ανάπτυξης λογισμικού, το GNOME, ένα σύνολο από περιβάλλοντα προγραμματισμού που χρησιμοποιήθηκε για εκπαιδευτικούς λόγους και το SMILE, ένα εσωτερικό περιβάλλον ανάπτυξης συστημάτων που βοηθά στον έλεγχο πηγαίου κώδικα και στη διαχείριση εργασιών. Στη συνέχεια θα αναλύσουμε το περιβάλλον του *Gandalf Prototype (GP)*.

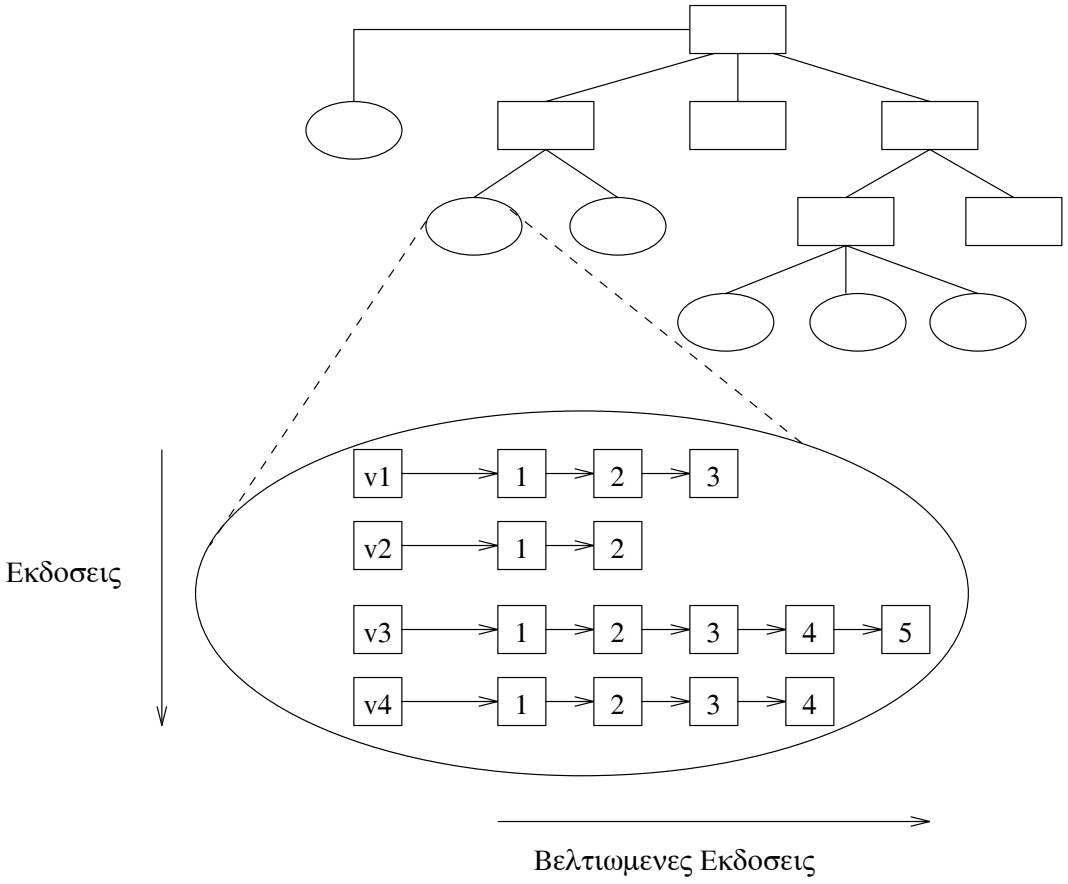
[2] Βασικός στόχος ενός περιβάλλοντος ανάπτυξης λογισμικού (*software development environment*) είναι να παρέχει κατάλληλη υποστήριξη, ώστε να απλοποιηθεί η διαδικασία ανάπτυξής του.

[3] Περιβάλλον προγραμματισμού : διευκολύνει την εργασία του προγραμματισμού.

[4] Περιβάλλον ανάπτυξης συστημάτων : περιορίζει το βαθμό που μία εργασία παραγωγής λογισμικού εξαρτάται από την καλή θέληση των μελών της ομάδας εργασίας.

4.2.1 Σύστημα Ελέγχου Εκδόσεων

Το σύστημα ελέγχου εκδόσεων που χρησιμοποιείται στο GP υποστηρίζει δύο στόχους : περιγραφή συστημάτων και αυτόματη δημιουργία εκδόσεων συστημάτων.



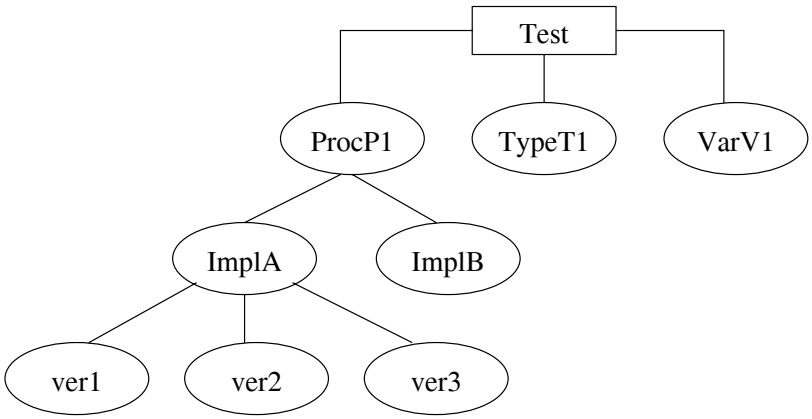
Σχήμα 4.2: Δομές ελέγχου εκδόσεων στο GP

Τα κουτιά, που συμβολίζονται με τετράγωνα, επιβάλλουν μία δενδροειδή δομή στην περιγραφή συστημάτων στο GP. Τα στοιχεία, που συμβολίζονται με κύκλους, αντιστοιχούν στους ορισμούς στοιχείων προγραμμάτων. Κάθε ένα από τα στοιχεία μπορεί να υλοποιηθεί σε διαφορετικές εκδόσεις. Κάθε έκδοση μπορεί να παράγει, μέσω διαδοχικών αλλαγών, χρονικά, ένα σύνολο από διασκευασμένες εκδόσεις.

Η περιγραφή ενός συστήματος λογισμικού στο GP αποτελείται από ένα στατικό και ένα δυναμικό τμήμα. Το στατικό τμήμα είναι η περιγραφή των συστημάτων διασύνδεσης των στοιχείων και καθορίζει πώς διάφορα κομμάτια - όπως δεδομένα, ορισμοί τύπων, διεργασίες - μπορούν να χρησιμοποιηθούν σε άλλα στοιχεία. Το δυναμικό τμήμα είναι η αφηρημένη περιγραφή της σύνθεσης ενός υποσυστήματος, δηλαδή, δεν γίνεται αναφορά σε συγκεκριμένες εκδόσεις στοιχείων.

Μία περιγραφή συστήματος βασίζεται σε διάφορες έννοιες, από τις οποίες κάθε μία

ορίζεται σαν ένα κομμάτι της αφηρημένης σύνταξης του GP. Τα *κουτιά*, όμοια με τα directories στα παραδοσιακά συστήματα αρχείων, περιέχουν μία συλλογή από άλλα κουτιά και *στοιχεία*. Τα στοιχεία παρέχουν ένα σύστημα διασύνδεσης, που αποτελείται από ένα σύνολο προδιαγραφών δυνατοτήτων και ένα σύνολο εκδόσεων (*versions*) που υλοποιούν αυτό το σύστημα διασύνδεσης με διαφορετικούς τρόπους. Κάθε έκδοση αποτελείται από ένα σύνολο διασκευασμένων εκδόσεων (*revisions*), οι οποίες με τη σειρά τους αντιστοιχούν σε μία πραγματική υλοποίηση του στοιχείου και είναι αμετάβλητες (δηλαδή, αν θέλουμε να τροποποιήσουμε μία διασκευασμένη έκδοση - π.χ. διόρθωση ενός λογικού λάθους - πρέπει να δημιουργήσουμε μία νέα διασκευασμένη έκδοση, που θα περιέχει την αλλαγή. Για να μην γίνεται σπατάλη χώρου ο κοινός κώδικας περιέχεται μόνο σε μία από διασκευασμένες εκδόσεις). Η δενδροειδής δομή των κουτιών και των στοιχείων, μαζί με τη διδιάστατη δομή των εκδόσεων και των διασκευών μέσα σε ένα στοιχείο, φαίνεται στο σχήμα 4.2.



Σχήμα 4.3: Ο γράφος του συστήματος Test

Στο σχήμα 4.3 παρουσιάζεται το σύστημα Test με την παραπάνω δομή. Αποτελείται από τον τύπο TypeT1, τη διαδικασία ProcP1 και τη μεταβλητή VarV1. Μέσα στο στοιχείο ProcP1 μπορεί να γίνει επιλογή μεταξύ δύο δυνατών εκδόσεων, των ImplA και ImplB, και για το ImplA μπορούμε να επιλέξουμε μεταξύ τριών διασκευών.

Κάθε έκδοση ενός στοιχείου πρέπει να υλοποιεί ακριβώς το ίδιο σύστημα διασύνδεσης. Δοθέντος αυτού του περιορισμού, οι εκδόσεις ενός στοιχείου αντικαθίστανται ίσως αυθαίρετα και πρέπει να επηρεάζουν μόνο την απόδοση του συστήματος. Μία άλλη συνέπεια του περιορισμού είναι ότι, αν τροποποιηθεί το σύστημα διασύνδεσης ενός στοιχείου, πρέπει να δημιουργηθεί ένα νέο στοιχείο, που θα έχει ένα νέο όνομα και πρέπει να δοθούν γι' αυτό, μία ή περισσότερες νέες υλοποιήσεις.

Το GP, επιπλέον, διευκολύνει την επέκταση και τροποποίηση των συστημάτων διασύνδεσης επιτρέποντας σε υλοποιήσεις στοιχείων να χρησιμοποιούν δυνατότητες που έχουν οριοθεί σε άλλα στοιχεία. Αυτό επιτυγχάνεται με το διαχωρισμό μεταξύ *υλοποιήσεων* (αντιστοιχούν στις παραλλαγές εκδόσεων (*variants*) των άλλων συστημάτων) και *συνθέσεων* (αντιστοιχούν στις εκδόσεις διαμορφώσεων). Μία υλοποίηση είναι, ουσιαστικά, ένα πηγαίο πρόγραμμα που ορίζει τμήματα κώδικα για όλες τις περιγραφές δυνατοτήτων, στο σύστημα διασύνδεσης του στοιχείου. Μία σύνθεση περιγράφεται από τη λίστα των στοιχείων ή εκδόσεων στοιχείων από τα οποία συντίθεται. Για παράδειγμα, αν υποθέσουμε ότι το στοιχείο M παρέχει το σύνολο δυνατοτήτων (f, g) και το στοιχείο N το σύνολο (h, k), η σύνθεση C = (M, N) θα είναι μία νόμιμη έκδοση ενός στοιχείου, που μπορεί να παρέχει το σύνολο δυνατοτήτων (f, g, h, k)

Οπως αναφέραμε προηγουμένως, ο δεύτερος στόχος του συστήματος ελέγχου εκδόσεων είναι η αυτόματη δημιουργία εκτελέσιμων εκδόσεων συστημάτων.

Ο χρήστης μπορεί να δημιουργήσει ένα υποσύστημα εφαρμόζοντας μία εντολή σε ένα στοιχείο, μία έκδοση ή μία διασκευασμένη έκδοση. Για να εκτελεσθεί η εντολή στην περίπτωση που δηλώνεται μόνο το στοιχείο, και όχι μία συγκεκριμένη έκδοση αυτού, κάθε στοιχείο ορίζει μία από τις εκδόσεις του ως την *καθιερωμένη* (*standard*) έκδοση. Ο χρήστης μπορεί να αλλάξει αυτόν τον ορισμό, αλλά πάντα πρέπει να υπάρχει μία. Ετσι, κάθε φορά που η διεργασία δημιουργίας ενός συστήματος συναντά το όνομα ενός στοιχείου, που συμμετέχει στην κατασκευή του συστήματος, θα επιλέγει την καθιερωμένη έκδοση αυτού του στοιχείου. Ομοια, κάθε υλοποίηση πρέπει να έχει μία καθιερωμένη διασκευασμένη έκδοση. Αν τώρα, ένας χρήστης θέλει να χρησιμοποιήσει εκδόσεις ή διασκευασμένες εκδόσεις, διαφορετικές από τις καθιερωμένες, μπορεί να τις ορίσει μέσα σε λίστες και συνθέσεις.

4.2.2 Κατασκευή Προγραμμάτων

Η κατασκευή προγραμμάτων στο GP στηρίζεται σε τρεις βελτιώσεις του κύκλου κατασκευής. Πρώτον, η χρήση κειμενογράφων ALOE [39] μειώνει το χρόνο και την προσπάθεια που χρειάζεται για να περάσει ένα πρόγραμμα από τον compiler. Δεύτερον, περιορίζεται η διεργασία σύνδεσης στις τροποποιημένες διαδικασίες. Τρίτον, παρέχεται υποστήριξη για debugging σε επίπεδο πηγής.

Το βασικό πρόβλημα των απλών κειμενογράφων είναι η γενικότητά τους, που δεν τους επιτρέπει να καταλαβαίνουν τα αντικείμενα που χειρίζονται. Για παράδειγμα, αν κάποιος γράψει τα γράμματα "b", "e", "g", "i", "n" στο πληκτρολόγιο, ο κειμενογράφος δεν καταλαβαίνει

ότι πρόκειται για μία λέξη κλειδί, η οποία πρέπει να έχει μία αντίστοιχη λέξη κλειδί που θα δηλώνει το τέλος. Από την άλλη πλευρά, ένας κειμενογράφος ALOE βοηθά το χρήστη να κατασκευάζει προγράμματα σύμφωνα με τη συντακτική δομή της γλώσσας που χρησιμοποιεί. Έτσι, είναι αδύνατο να παραχθούν προγράμματα που δεν είναι σωστά συντακτικά. Οι εντολές των ALOE κειμενογράφων βασίζονται στις κατασκευές ανάπτυξης λογισμικού (όπως : δηλώσεις γλωσσών προγραμματισμού, λίστες ελέγχου πρόσβασης και τύποι στοιχείων) παρά σε γραμμές και χαρακτήρες.

Το πρόβλημα που εμφανίζεται με την ανάγκη επανάληψης της διαδικασίας σύνδεσης και δημιουργίας μιας νέας εκτελέσιμης έκδοσης ενός συστήματος, αν ένα από τα στοιχεία του τροποποιηθεί, αντιμετωπίζεται στο GP με τη χρήση αμετάβλητων διευθύνσεων για διαδικασίες μέσα σε στοιχεία. Κάθε στοιχείο έχει ένα διάνυσμα εισαγωγής, που περιέχει μία θέση για κάθε διαδικασία που αυτό το στοιχείο παρέχει σε άλλα. Εξω από ένα στοιχείο αυτές οι διαδικασίες είναι γνωστές από μία καθορισμένη θέση στο διάνυσμα εισαγωγής. Αυτό επιτρέπει μεταφορά των διαδικασιών, χωρίς να επηρεάζονται οι αναφορές σε αυτές από άλλα στοιχεία. Το περιεχόμενο μιας θέσης στο διάνυσμα εισαγωγής είναι η απόλυτη διεύθυνση μιας διαδικασίας. Έτσι, το κόστος για αμετάβλητη σύνδεση είναι μία έμμεση αναφορά στη μνήμη σε κάθε κλήση διαδικασίας.

Τέλος, παρέχονται εντολές για debugging, με τις οποίες ο χρήστης μπορεί να σώσει καταστάσεις ενός προγράμματος, να αρχίσει την εκτέλεση από μία αρχική κατάσταση ή να ξαναρχίσει από μία κατάσταση που έχει ήδη κρατηθεί.

4.2.3 Διαχείριση Εργασιών

Η λειτουργία της διαχείρισης εργασιών στο GP είναι να εγγυηθεί την ακεραιότητα της πληροφορίας που σχετίζεται με τις καταστάσεις ανάπτυξης ενός συστήματος και να εξασφαλίσει την πρόσβαση σε αυτή.

Πρωταρχικά, η διαχείριση εργασιών συνεπάγεται την περιγραφή ενός συστήματος σε υψηλό επίπεδο, η οποία εκφράζεται με όρους κουτιών, στοιχείων και εκδόσεων. Η περιγραφή συστήματος επεκτείνεται με ορισμένα αντικείμενα που εξυπηρετούν ειδικά το σκοπό της διαχείρισης εργασιών. Τα πιο σημαντικά αντικείμενα αυτού του είδους είναι η *λίστα πρόσβασης* (*access list*), μία προσκολλημένη σε κάθε κουτί, και το *ιστορικό διασκευασμένων εκδόσεων* (*revision history*), ένα συνδεδεμένο με κάθε κουτί, ένα με κάθε στοιχείο και ένα με κάθε έκδοση στοιχείου. Η λίστα πρόσβασης ενός κουτιού περιγράφει τα δικαιώματα των προγραμματιστών

που χειρίζονται το κουτί. Για παράδειγμα, ένας απλός χρήστης, που περιλαμβάνεται στη λίστα πρόσβασης ενός κουτιού, μπορεί να διαβάζει πληροφορία και να εκτελεί προγράμματα. Από την άλλη πλευρά, ένας υπερ-χρήστης μπορεί, να τροποποιεί κουτιά και στοιχεία που περιέχονται σε ένα κουτί, να διαγράφει πληροφορία και να αλλάζει τη λίστα πρόσβασης. Ενα ιστορικό διασκευασμένης έκδοσης είναι μία συλλογή μηνυμάτων, ταξινομημένη χρονολογικά, όπου σε κάθε μήνυμα περιγράφονται εν συντομία οι αλλαγές που έγιναν. Κάθε μήνυμα είναι χρονολογημένο και "σημαδεμένο" με το όνομα του προγραμματιστή που προκάλεσε τη δημιουργία του.

Οταν μία ομάδα από προγραμματιστές εργάζεται στην σύνδεση των επιμέρους τμημάτων μίας εργασίας, υπάρχει πιθανότητα ορισμένοι από αυτούς να προσπαθήσουν να τροποποιήσουν ταυτόχρονα κάποιο στοιχείο του συστήματος. Για την αποφυγή συγκρούσεων αυτής της μορφής, η διαχείριση εργασιών του GP παρέχει ένα σύνολο εντολών, όμοιες με τις εντολές εισαγωγής και εξαγωγής εκδόσεων (check-in, check-out) που συναντήσαμε στο RCS. Αν ένας χρήστης θέλει να τροποποιήσει ένα στοιχείο πρέπει να κάνει μία "κράτηση" και να το "κλειδώσει". Αν έχει προλάβει κάποιος άλλος, του επιστρέφεται ένα μήνυμα λάθους, διαφορετικά το στοιχείο κλειδώνεται και αντιγράφεται στην προσωπική του βάση δεδομένων. Οταν τελειώσει με τις τροποποιήσεις, μπορεί αν θέλει, να κάνει τις αλλαγές μόνιμες, τοποθετώντας το τροποποιημένο στοιχείο πίσω στην κοινή βάση δεδομένων. Αν δε θέλει, διαγράφει την κράτηση και αφήνει το στοιχείο στην αρχική του κατάσταση.

4.3 Τρίτη Γενεά (shape)

Το σύστημα ShapeTools είναι μία συλλογή από εργαλεία, τα οποία βοηθούν στην άσκηση αποτελεσματικού ελέγχου πάνω στις περιπλοκές της διεργασίας αλλαγής, κατά την παραγωγή εργασιών λογισμικού. Αποτελείται από τρία μέρη : ένα σύστημα ελέγχου εκδόσεων που υποστηρίζει σειριακές (*revisions*) και παράλληλες (*variants*) γραμμές ανάπτυξης, το πρόγραμμα shape που είναι μία σημαντικά επαυξημένη μορφή του προγράμματος Make και μία βάση αντικειμένων.

Το σύστημα ShapeTools έχει χρησιμοποιηθεί και στα πλαίσια της εργασίας STONE [37], η οποία είχε ως στόχο την ενσωμάτωση των λειτουργιών ενός εργαλείου για software configuration management σε ένα γενικό περιβάλλον, το οποίο έχει ως πρόθεση την αύξηση της αποδοτικότητας στην κατασκευή λογισμικού και την βελτίωση της ποιότητας του παραγόμενου λογισμικού.

4.3.1 Σύστημα Ελέγχου Εκδόσεων

Οι εντολές ελέγχου εκδόσεων του συστήματος ShapeTools - μοιάζουν πολύ με αυτές του RCS - χρησιμεύουν για το χειρισμό πολλαπλών εκδόσεων πηγαίων αντικειμένων (*source objects*) (συνήθως αρχεία κειμένου) καθώς επίσης και πολλαπλών στοιχείων παραγόμενων αντικειμένων (*derived objects*) (συνήθως παράγονται από το compiling ενός ή περισσοτέρων πηγαίων αντικειμένων). Ολα τα αντικείμενα αποθηκεύονται στη βάση αντικειμένων και υπάρχουν εντολές, που αποθηκεύουν ή ανακαλούν μία έκδοση ενός αντικειμένου από τη βάση, χειρίζονται το ιστορικό και τα γνωρίσματα εκδόσεων αντικειμένων και εξασφαλίζουν πληροφορία σχετική με τα περιεχόμενα της βάσης.

Τα αντικείμενα είναι κάτι παραπάνω από αρχεία, αφού είναι δυνατό να τους εκχωρηθούν διάφορα γνωρίσματα (με τη βοήθεια του συστήματος διαχείρισης αντικειμένων) όπως : αριθμός έκδοσης, συμβολικό όνομα, κατάσταση (*busy, saved, proposed, published, accessed, frozen*), όνομα συγγραφέα, ημερομηνία και ώρα κατασκευής (ή τροποποίησης) και τέλος χαρακτηριστικά που ορίζονται από το χρήστη. Εννοιολογικά, δεν υπάρχει καμία διαφορά μεταξύ των ήδη υπαρχόντων χαρακτηριστικών και αυτών που ορίζονται από το χρήστη, οπότε, μπορούν να χρησιμοποιηθούν και τα δύο με τον ίδιο ακριβώς τρόπο.

Το σχήμα οργάνωσης των εργασιών που υποστηρίζεται από το σύστημα ShapeTools κάνει ένα διαχωρισμό μεταξύ προσωπικών χώρων εργασίας (*workspaces*), που αντιστοιχεί ένας σε κάθε χρήστη, και μίας κεντρικής αποθήκης (*repository*), την οποία μοιράζονται τα μέλη μιας ομάδας εργασίας και διαχειρίζεται ο αρχηγός της ομάδας. Οταν ένας χρήστης θέλει να τροποποιήσει μία έκδοση ενός αντικειμένου ακολουθεί το σενάριο που προαναφέραμε στο σύστημα Gandalf. Προσπαθεί να κάνει μία "κράτηση" και να "κλειδώσει" το αντικείμενο. Αν έχει προλάβει και το έχει κλειδώσει κάποιος άλλος, επιστρέφεται ένα μήνυμα λάθους. Διαφορετικά, το σύστημα ζητά μία περιγραφή των αλλαγών που σκοπεύει ο χρήστης να κάνει και θέτει στη βάση την έκδοση του αντικειμένου, που πήρε για τροποποίηση ο χρήστης, στην κατάσταση busy. Οταν τελειώσει με τις αλλαγές, υποβάλλει την εργασία που έκανε στον αρχηγό της ομάδας και είναι πλέον θέμα αυτού, αν θα τη δεχτεί οπότε θα δημιουργηθεί μία νέα έκδοση στη βάση, ή θα την απορρίψει.

Τέλος, παρ' όλο που το βασικό εργαλείο για τη δημιουργία, συντήρηση και το χειρισμό διαμορφώσεων είναι το πρόγραμμα shape, το σύστημα ελέγχου εκδόσεων επιτρέπει στο χρήστη, να συνδέσει διαφορετικά αντικείμενα σε μία διαμόρφωση, και μπορεί να ανακατασκευάσει μία συγκεκριμένη διαμόρφωση, από τη βάση αντικειμένων, με μία εντολή.

4.3.2 Το Πρόγραμμα shape

Το πρόγραμμα shape έχει σαν κύριο στόχο να εξασφαλίσει και να συντηρήσει τον έλεγχο πάνω στην εξέλιξη της γενικής δομής των εξαρτημάτων ενός πολύπλοκου συστήματος λογισμικού. Παρ' όλο το ευρύ φάσμα δυνατοτήτων που παρέχει, είναι εύκολο στη χρήση του. Η βασική απλότητα του εργαλείου έχει κληρονομηθεί από το Make και έτσι τα αρχεία ελέγχου του shape (Shapefile) έχουν τα ίδια συντακτικά στοιχεία με τα Makefile. Ενα Shapefile είναι το κεντρικό σημείο, όπου αποθηκεύεται η δομημένη πληροφορία σχετικά με το σύστημα που διαχειρίζεται ένας χρήστης, δηλαδή, έχει το ρόλο του μοντέλου συστήματος. Εκτός όμως από αυτό, ένα Shapefile μπορεί να περιέχει κανόνες επιλογής, ορισμούς παραλλαγών (*variant definitions*) - αποτελούν τις σημαντικότερες βελτιώσεις σε σχέση με το Make - και κανόνες μετατροπής.

Μεταξύ των Make και shape υπάρχει μία βασική εννοιολογική διαφορά, παρ' όλο που η σύνταξη των αρχείων ελέγχου είναι αρκετά παρεμφερής. Τα ονόματα των αντικειμένων που υπάρχουν σε ένα Makefile αναφέρονται, άμεσα, σε αρχεία που βρίσκονται στο παρόν directory. Αντίθετα, το shape αναφέρεται σε αφηρημένα αντικείμενα, τα οποία πρέπει να συνδεθούν στο Shapefile με μία συγκεκριμένη έκδοση πριν εφαρμοσθεί σε αυτά οποιοδήποτε εργαλείο. Η σύνδεση αυτή καθορίζεται από τους κανόνες επιλογής εκδόσεων του shape.

Κανόνες Επιλογής Διαμορφώσεων

Οι κανόνες επιλογής χρησιμοποιούνται για να συνδέσουν συγκεκριμένα στοιχεία αντικειμένων στη βάση αντικειμένων με ονόματα στοιχείων που χρησιμοποιούνται στο Shapefile, κατά την κατασκευή ενός συστήματος από τα επιμέρους τμήματά του. Ενας κανόνας επιλογής (στον οποίο έχει εκχωρηθεί ένα όνομα) είναι μία ακολουθία από εναλλακτικές λύσεις, οι οποίες διαχωρίζονται με ερωτηματικό και αποτελούν μία έκφραση λογικής διάζευξης. Κάθε εναλλακτική λύση αποτελείται από μία ακολουθία από κατηγορήματα (*predicates*), τα οποία διαχωρίζονται με κόμμα και συνιστούν μία έκφραση λογικής σύζευξης. Ενας κανόνας επιλογής είναι επιτυχής αν μία από τις εναλλακτικές λύσεις είναι επιτυχής (δηλαδή, οδηγεί σε μοναδική αναγνώριση ενός στοιχείου κάποιου αντικειμένου).

Για παράδειγμα, αν για ένα σύστημα ο χρήστης θέλει τη διαμόρφωση που αποτελείται : ***από την τελευταία έκδοση όλων των εξαρτημάτων*** `(*.c)` ***τα οποία έχουν συγγραφέα τον andy*** `(author = andy and state = busy)` ***και από την τελευταία δημοσιευμένη έκδοση*** `(state = published and max version)`, ***για όλα τα άλλα εξαρτήματα***, μπορεί να το πετύχει με τον παρακάτω κανόνα επιλογής, στον οποίο έχει εκχωρηθεί το όνομα `exprule`.

```
exprule :
          *.c, attr(author, andy), attr(state, busy);
          *.c, attrge(state, published), attrmax(version);
```

Κανόνες Μετατροπής

Οι κανόνες μετατροπής του shape περιγράφουν : τι είδος πηγαία αντικείμενα μετατρέπονται σε παραγόμενα αντικείμενα, πώς εκτελείται αυτή η μετατροπή και ποιες παράμετροι είναι σημαντικές γι' αυτή. Η σύνταξή τους είναι μία επέκταση των προκαθορισμένων κανόνων περιγραφής τύπων του Make. Για παράδειγμα, ο κανόνας με τον οποίο, ένα πηγαίο αντικείμενο C (".c") μετατρέπεται σε ".o" παραγόμενο αντικείμενο, ορίζεται ως εξής :

```
%.o :  %.c +(CC) +(CFLAGS)
    $(CC) -c $(CFLAGS) %.c
```

(Με την μακροεντολή `CC` ορίζεται ο compiler και με τη `CFLAGS` τα compile switches.)

Οταν υπολογίζεται ένας κανόνας μετατροπής, το όνομα του πηγαίου αντικειμένου περνάει στον παρόντα κανόνα επιλογής, και έτσι, συνδέεται με μία συγκεκριμένη έκδοση πηγαίου αντικειμένου στη βάση αντικειμένων. Αν ο κανόνας μετατροπής εκτελεσθεί επιτυχώς, το shape θεωρεί ότι το συγκεκριμένο παραγόμενο αντικείμενο έχει δημιουργηθεί, το αποθηκεύει στη δεξαμενή στοιχείων, η οποία αποτελεί ένα κομμάτι της βάσης αντικειμένων και του εκχωρεί ένα γνώρισμα, το οποίο περιγράφει τη μετατροπή που έγινε (δηλαδή, περιέχει τα πηγαία αντικείμενα και τις μακροεντολές που χρησιμοποιήθηκαν για την κατασκευή του συγκεκριμένου παραγόμενου αντικειμένου). Με αυτό τον τρόπο αποφεύγεται η ανακατασκευή παραγόμενων αντικειμένων, κατά τη δημιουργία μίας διαμόρφωσης ενός συστήματος.

Ορισμοί Παραλλαγών

Μία παραλλαγή (*variant*) είναι μία εναλλακτική υλοποίηση της ίδιας έννοιας. Το shape παρέχει ένα μηχανισμό για το χειρισμό παραλλαγών, ο οποίος υποστηρίζει και βασικές έννοιες διαχείρισης και τον συνδυασμό τους. Σε κάθε παραλλαγή αντιστοιχίζεται ένα όνομα (*variant name*) το οποίο συνδέεται με ένα σύνολο μακροεντολών. Το παράδειγμα που ακολουθεί αναφέρεται στη χρήση των ορισμών παραλλαγών :

```
#%VARIANT-SECTION
vclass system ::= (vaxbsd, munix)
vclass customer ::= (smith, miller)
```

```
vaxbsd :
        vflags = "-DUNIX -DBSD43 -DSTDCC -DVAX -DPSDEBUG"
        vpath = "sys/vaxbsd"
munix :
        vflags = "-DUNIX -DSYS5 -DPCSCC -DNOVAX -DPSDEBUG"
        vpath = "sys/sys5"
smith :
        vflags = "-DSMITH"
        vpath = "smith"
        VARSOURCE = smithboot.c storeif.c creport.c
        VARINCLUDE = cformats.h
        DEPENDENCIES = smith/Dependencies
        COMPILER = prop
miller :
        vpath = "miller"
#%END-VARIANT-SECTION
```

Τα ονόματα παραλλαγών χρησιμοποιούνται στους κανόνες επιλογής με χρήση του προκαθορισμένου κατηγορήματος *attrvar* (γνώρισμα παραλλαγής), μέσω του οποίου επιτυγχάνεται η διαχείριση των παραλλαγών μέσα σε ένα σύστημα ελέγχου εκδόσεων. Για παράδειγμα, ένας κανόνας επιλογής διαμόρφωσης που χρησιμοποιεί το γνώρισμα παραλλαγής είναι ο παρακάτω, στον οποίο έχει εκχωρηθεί το όνομα `fsexp` :

```
fsexp :
    *.[ch], attrvar(vaxbsd), attrvar(smith), attr(state, busy)
```

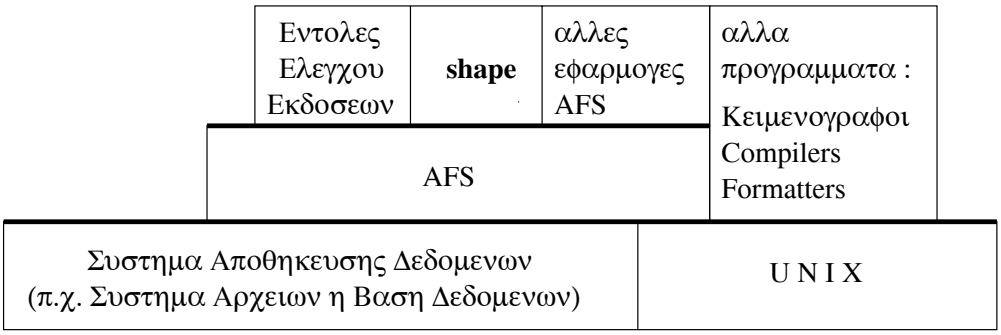
(Η μακροεντολή vpath ορίζει το χώρο αναζήτησης των αντικειμένων και η vflags τις επιπλέον παραμέτρους που πρέπει να χρησιμοποιηθούν. Έτσι, για τον παραπάνω κανόνα επιλογής, με όνομα `fsexp`, έχουμε :

```
        vflags = "-DUNIX -DBSD43 -DSTDCC -DVAX -DPSDEBUG -DSMITH"
        vpath = "sys/sys5:smith" )
```

Με τη δομή vclass το shape ορίζει κλάσεις παραλλαγών και τη χρησιμοποιεί για τον έλεγχο των συνδυασμών παραλλαγών, γιατί δε μπορεί να καταλάβει από μόνο του ποιοι συνδυασμοί έχουν νόημα. Δηλαδή, μία κλάση παραλλαγών είναι μία ομάδα ορισμών παραλλαγών, οι οποίοι είναι αμοιβαία αποκλειόμενοι.

4.3.3 Βάση Αντικειμένων

Μία βάση αντικειμένων είναι υβρίδιο ενός συστήματος αρχείων και μίας βάσης δεδομένων, το οποίο προσπαθεί να συνδυάσει τα πλεονεκτήματα των δύο τύπων συστημάτων αποθήκευσης δεδομένων. Η βάση αντικειμένων που χρησιμοποιεί το σύστημα ShapeTools ονομάζεται AFS (*Attributed FileSystem*) [31] και είναι ένα σύστημα αποθήκευσης αντικειμένων λογισμικού σαν μία σύνθεση από δεδομένα και έναν αυθαίρετο αριθμό από, συσχετισμένα με αυτά, γνωρίσματα. Το AFS έχει σχεδιασθεί έτσι ώστε να αποτελεί τη βάση για την ενοποίηση ενός ευρέου φάσματος διαφορετικών εργαλείων ανάπτυξης λογισμικού, όπως : κειμενογράφοι, compilers, ένα σύστημα software configuration management κ.λπ. (σχήμα 4.4). Τα γνωρίσματα δίνουν τη δυνατότητα σε κάθε πρόγραμμα εφαρμογής του AFS να αποθηκεύει συνεχώς πληροφορία σχετική με ένα αντικείμενο λογισμικού είτε για μετέπειτα χρήση ή αφήνοντάς την για ερμηνεία από άλλες εφαρμογές (ή και τα δύο).



Σχήμα 4.4: Το AFS και οι εφαρμογές του

Μέχρι σήμερα το AFS υπάρχει σε δύο διαφορετικές υλοποιήσεις με παρόμοιο σύστημα διασύνδεσης. Μία πάνω από το σύστημα αρχείων UNIX και μία πάνω από τη βάση δεδομένων DAMOKLES [12].

Κεφάλαιο 5

Μία Αλλη Προσέγγιση στο Configuration Management

Οπως προαναφέραμε στο δεύτερο κεφάλαιο (ενότητα 2.3) η περιοχή του configuration management με την οποία ασχοληθήκαμε στην παρούσα εργασία είναι η επιλογή εκδόσεων.(*"Η διεργασία της επιλογής σωστών εξαρτημάτων για μία διαμόρφωση είναι πολύ επιρρεπής σε λάθη και αποτελεί το βασικό πρόβλημα του configuration management"* [18].) Τα υπάρχοντα συστήματα, όπως είδαμε και στο προηγούμενο κεφάλαιο, αντιμετωπίζουν το θέμα της επιλογής εκδόσεων με τη σύνδεση γνωρισμάτων (προκαθορισμένων ή ορισμένων από το χρήστη) με τα αντικείμενα λογισμικού και την εφαρμογή περιορισμών πάνω στα γνωρίσματα.

Σ' αυτό το κεφάλαιο θα παρουσιάσουμε μία νέα προσέγγιση στο θέμα, η οποία χρησιμοποιεί μία μερικά διατεταγμένη άλγεβρα πάνω στα ονόματα των εκδόσεων των αντικειμένων, όπως αυτή έχει οριοθεί από τους Plaice και Wadge στο [42]. Στη συνέχεια θα δούμε τις "αδυναμίες" που παρουσιάζει η άλγεβρα των εκδόσεων, για παράδειγμα, δε μας επιτρέπεται να καταλάβουμε αν μία έκδοση είναι διασκευή (*revision*) ή παραλλαγή (*variant*). Τέλος, θα παρουσιάσουμε ένα μοντέλο για configuration management (περιγράφεται με τη γλώσσα παράστασης γνώσης TELOS), με το οποίο, αντιμετωπίζουμε τις αδυναμίες που αναφέρουμε προηγουμένως και εισάγουμε τις έννοιες των περιορισμών ασυμβατότητας και των προαιρετικών (*optional*) αντικειμένων.

5.1 Αλγεβρα Εκδόσεων

Η βασική αδυναμία των υπαρχόντων εργαλείων είναι ότι οι διαφορετικές εκδόσεις ενός αντικειμένου έχουν μόνο τοπική σημασία. Για παράδειγμα, έστω ότι υπάρχει η τρίτη έκδοση

ενός αντικειμένου `A` και η τρίτη έκδοση ενός αντικειμένου `B`. Δεν υπάρχει κανένας λόγος να περιμένουμε οποιαδήποτε σχέση μεταξύ αυτών των εκδόσεων.

Στην παρούσα προσέγγιση, τα ονόματα των εκδόσεων προορίζονται να έχουν μία καθολική, ομοιόμορφη σημασία. Ετσι η `fast` έκδοση ενός στοιχείου `A` έχει νόημα να συνδυασθεί με τη `fast` έκδοση του `B`. Από τους προγραμματιστές αναμένεται να εξασφαλίσουν το ότι αυτές οι εκδόσεις είναι συμβατές.

Ενα πλεονέκτημα αυτής της προσέγγισης είναι ότι μπορούμε πλέον να μιλάμε όχι μόνο για εκδόσεις αντικειμένων (στοιχείων) αλλά και για εκδόσεις ολόκληρου του συστήματος. Για παράδειγμα, υποθέτουμε ότι έχει δημιουργηθεί η `fast` έκδοση κάθε εξαρτήματος ενός compiler. Τότε μπορεί να κατασκευασθεί η `fast` έκδοση του compiler συνδυάζοντας τις `fast` εκδόσεις όλων των εξαρτημάτων. Γενικά, είναι μη πραγματική η απαίτηση να υπάρχει μία `fast` έκδοση για κάθε εξάρτημα. Είναι όμως εφικτό να αυξάνεται η ταχύτητα ενός compiler με την τροποποίηση μερικών εξαρτημάτων του, οπότε μόνο αυτά θα έχουν `fast` εκδόσεις. Ετσι, επεκτείνεται ο κανόνας διαμόρφωσης ως εξής : για την κατασκευή της `fast` έκδοσης του compiler επιλέγεται, αν υπάρχει, η `fast` έκδοση κάθε εξαρτήματος, διαφορετικά επιλέγεται η έκδοση βάσης.

Η προσέγγιση αυτή γενικεύεται με τον ορισμό μίας μερικά διατεταγμένης άλγεβρας από ονόματα εκδόσεων. Η μερική διάταξη είναι η σχέση εκλέπτυνσης : $V \sqsubseteq W$, διαβάζεται "η V προηγείται της W", που πρακτικά σημαίνει ότι η έκδοση W είναι το αποτέλεσμα μίας επιπλέον ανάπτυξης της V. Η βασική αρχή είναι ότι στη διαμόρφωση της έκδοσης W ενός συστήματος, μπορεί να χρησιμοποιηθεί η έκδοση V ενός συγκεκριμένου εξαρτήματος εάν αυτό δεν υπάρχει σε μία πλησιέστερη έκδοση. Δηλαδή, χρησιμοποιείται η έκδοση V του εξαρτήματος εφόσον αυτό δεν έχει μία έκδοση V', τέτοια ώστε $V \sqsubseteq V' \sqsubseteq W$.

5.1.1 Χώρος Εκδόσεων

Σε αυτή την ενότητα θα εξετάσουμε τους τελεστές της άλγεβρας και πρακτικές εφαρμογές για κάθε έναν. Η πιο απλή, πιθανή, άλγεβρα θα επιτρέπει μόνο μία έκδοση. Αυτή την έκδοση θα την ονομάζουμε *έκδοση βάσης* (*base version*) και θα τη συμβολίζουμε με το ϵ.

Αριθμητικές Εκδόσεις

Οι πιο απλές εκδόσεις είναι αυτές που αντιστοιχούν στα διαδοχικά στάδια της διεργασίας ανάπτυξης : έκδοση `1`, έκδοση `2`, έκδοση `3`, κ.ο.κ . Μία προφανής επέκταση είναι να επιτραπεί η ύπαρξη υποσειρών : `1.1`, `1.2`, ή `2.3.1`.

Το αρχικό σύνολο των δυνατών εκδόσεων μπορεί να περιγραφεί με την παρακάτω γραμματική :

$$V ::= \epsilon \mid N$$
$$N ::= n \mid N.n$$

όπου, n ένας θετικός ακέραιος.

Η διάταξη εκλέπτυνσης όπως περιγράφηκε προηγουμένως δείχνει πώς μία έκδοση προκύπτει από άλλες. Αυτή η διάταξη, που συμβολίζεται $\sqsubseteq$, πρέπει να είναι καλά ορισμένη και μεταβατική :

$$\epsilon \sqsubseteq V$$
$$\frac{V \sqsubseteq V' \ , \ V' \sqsubseteq V''}{V \sqsubseteq V''}$$

Για το παρόν σύνολο εκδόσεων, χρησιμοποιείται η διαισθητική, λεξικογραφική διάταξη :

$$\frac{n \leq m}{n \sqsubseteq m}$$
$$N \sqsubseteq N.n$$
$$\frac{N \sqsubseteq M \ , \ N \ is \ not \ prefix \ of \ M}{N.n \sqsubseteq M}$$

Ετσι, για παράδειγμα, `1.2.3.4` $\sqsubseteq$ `1.3` $\sqsubseteq$ `2.4.5`.

Υποεκδόσεις

Η έννοια της υποέκδοσης (*subversion*) αντιστοιχεί σε μία έκδοση σε κλαδί στο δένδρο εκδόσεων του RCS. Δυστυχώς, η χρήση αριθμητικών ορμαθών για την αναγνώριση των κλαδιών είναι δύσχρηστη. Γι' αυτό στην άλγεβρα χρησιμοποιούνται ονόματα. Επιπλέον, εκτός από την παραπάνω αντιστοιχία, μία υποέκδοση μπορεί να είναι η διασκευή μίας έκδοσης ενός αντικειμένου. Ο νέος χώρος των δυνατών εκδόσεων γίνεται :

$$V ::= \epsilon \mid N \mid x \mid V \ \% \ V$$
$$N ::= n \mid N.n$$

όπου x είναι οποιοοδήποτε αλφαριθμητικός ορμαθός χαρακτήρων. Για παράδειγμα, η `graphics%mouse` έκδοση ενός συστήματος διασύνδεσης θα είναι η υποέκδοση της γραφικής έκδοσης του συστήματος διασύνδεσης στην οποία χρησιμοποιείται "ποντίκι".

Αντίθετα με το RCS, τα ονόματα δεν είναι μεταβλητές, αλλά σταθερές. Εξω από τη σχέση εκλέπτυνσης, είναι όλα μη συγκρίσιμα.

Για τις υποεκδόσεις χρειάζεται άλλο ένα αξίωμα :

$$V \sqsubseteq V \,\%\, V'$$

Για τον τελεστή % η ϵ θεωρείται ουδέτερο στοιχείο και ισχύει η προσεταιριστική ιδιότητα :

$$\epsilon \,\%\, V \equiv V$$
$$V \,\%\, \epsilon \equiv V$$
$$(V \,\%\, V') \,\%\, V'' \equiv V \,\%\, (V' \,\%\, V'')$$

Παράδειγμα : υποθέτουμε ότι υπάρχουν δύο εν χρήσει εκδόσεις, η `2.3.4` και η `3.5.6`. Αν θελήσουμε να διορθώσουμε κάποια λάθη στην `2.3.4`, θα χρειασθεί μία υποέκδοση. Αν αυτή ονομασθεί `2.3.4.1` θα θεωρείται ότι είναι προηγούμενη της `3.5.6`, το οποίο δεν ανταποκρίνεται στην πραγματικότητα. Αντ' αυτού όμως, μπορεί να χρησιμοποιηθεί το όνομα `2.3.4%bugfix`, χωρίς το παραπάνω πρόβλημα.

Ενώσεις Εκδόσεων

Δεν είναι ασυνήθιστο το γεγονός να θέλουμε να συνδυάσουμε διαφορετικές, συμβατές, υποεκδόσεις . Για να επιτευχθεί κάτι τέτοιο πρέπει να υπάρχει η δυνατότητα της ένωσης εκδόσεων. Ετσι, ο τελικός χώρος εκδόσεων γίνεται :

$$V ::= \epsilon \mid N \mid x \mid V \,\%\, V \mid V \,+\, V$$
$$N ::= n \mid N.n$$

Για να ολοκληρωθεί η διάταξη, προστίθενται άλλα δύο αξιώματα :

$$V \sqsubseteq V \,+\, V'$$
$$\frac{V_1 \sqsubseteq V_1' \quad V_2 \sqsubseteq V_2'}{V_1 \,+\, V_2 \sqsubseteq V_1' \,+\, V_2'}$$

Ο τελεστής + είναι ταυτοδύναμος και ισχύει γι' αυτόν η αντιμεταθετική, η προσεταιριστική και η αριστερά επιμεριστική ως προς τον % ιδιότητα :

$$V \,+\, V \equiv V$$
$$V \,+\, V' \equiv V' \,+\, V$$
$$(V \,+\, V') \,+\, V'' \equiv V \,+\, (V' \,+\, V'')$$
$$V\%V' \,+\, V\%V'' \equiv V \,\%\, (V' \,+\, V'')$$

Ο τελεστής + ορίζεται έτσι, ώστε να παράγει το ελάχιστο άνω φράγμα για τη σχέση $\sqsubseteq$. Εστω δύο εκδόσεις, V_1 και V_2. Η έκδοση $V_1 \,+\, V_2$ είναι το ελάχιστο άνω φράγμα των V_1 και V_2, δηλαδή, για όλα τα V, τέτοια ώστε $V_1 \sqsubseteq V$ και $V_2 \sqsubseteq V$, ισχύει η σχέση $V_1 + V_2 \sqsubseteq V$. Από το 2ο αξίωμα της παρούσας ενότητας έχουμε ότι : $(V_1 \,+\, V_2) \sqsubseteq (V \,+\, V)$. Αλλά από

την ισοδυναμία $V + V \equiv V$ προκύπτει ότι $(V_1 + V_2) \sqsubseteq V$. Επομένως, η $V_1 + V_2$ είναι το ελάχιστο άνω φράγμα.

Το γεγονός ότι η άλγεβρα επιτρέπει την ένωση ανεξαρτήτων εκδόσεων (δε μπορεί να γίνει κάτι διαφορετικό, γιατί ο μόνος έλεγχος που γίνεται είναι συντακτικός, στο επίπεδο των ονομάτων των εκδόσεων) δε σημαίνει απαραίτητα ότι έχει νόημα και κάθε αυθαίρετη ένωση. Είναι πλέον θέμα του διαχειριστή των διαμορφώσεων να εξασφαλίσει ότι έχει νόημα η συγχώνευση των εκδόσεων των εξαρτημάτων που επιλέγονται.

5.1.2 Κατασκευή Εκδόσεων Συστημάτων

Οπως αναφέραμε και προηγουμένως, η άλγεβρα δεν απαιτεί κάθε εξάρτημα (ενός συστήματος) να υπάρχει σε όλες τις εκδόσεις. Αντίθετα, η απουσία μίας συγκεκριμένης έκδοσης έχει την έννοια ότι μία πιο "γενική" έκδοση είναι κατάλληλη · στην απλούστερη περίπτωση η έκδοση βάσης.

Γενικά όμως, η έκδοση βάσης δεν είναι πάντα η καλύτερη επιλογή. Για παράδειγμα, έστω ότι χρειαζόμαστε την έκδοση `K%apple%fast`. Αν ένα συγκεκριμένο στοιχείο δεν υπάρχει σε αυτή ακριβώς την έκδοση, δύσκολα θα δικαιωθούμε αν θεωρήσουμε την έκδοση βάσης ως την καταλληλότερη επιλογή. Εαν υπάρχει η έκδοση `K%apple`, πρέπει αυτή να χρησιμοποιηθεί, διαφορετικά η έκδοση `K`, αν υπάρχει. Αν τέλος, δεν υπάρχει καμία από τις παραπάνω συγκεκριμένες εκδόσεις, μόνο τότε ενδείκνυται η χρήση της έκδοσης βάσης.

Ο γενικός κανόνας είναι ότι όταν κατασκευάζεται η έκδοση V ενός συστήματος S, επιλέγεται από κάθε εξάρτημα η έκδοση που προσεγγίζει περισσότερο από κάθε άλλη τη V (σύμφωνα με την άλγεβρα των εκδόσεων), και ονομάζεται *πλησιέστερη έκδοση* [1]. Συγκεκριμένα, για να επιλεγεί η κατάλληλη έκδοση ενός εξαρτήματος C, έστω $\mathcal{V}$ το σύνολο των εκδόσεων στις οποίες το C είναι διαθέσιμο. Το σύνολο των σχετικών εκδόσεων είναι το $R = \{V' \in \mathcal{V} \mid V' \sqsubseteq V\}$. Η πλησιέστερη έκδοση είναι το μέγιστο στοιχείο αυτού του συνόλου - αν υπάρχει. Αν δεν υπάρχει μέγιστο στοιχείο τότε δεν υπάρχει και η έκδοση V του δοθέντος συστήματος.

Η παραπάνω αρχή γενικεύεται ως εξής : έστω ότι ένα αντικείμενο αποτελείται από τα εξαρτήματα C_1, C_2, ... C_n. Τότε η πλησιέστερη στη V, έκδοση του S, σχηματίζεται με την ένωση των εκδόσεων V_1 του C_1, V_2 του C_2 ... , όπου σε κάθε περίπτωση η V_i είναι η πλησιέστερη στην V, έκδοση του C_i. Ετσι, η έκδοση που κατασκευάζεται είναι η $V_1 + V_2 +$

[1] Αν ο τελεστής + δεν ήταν ορισμένος ώστε να εξασφαλίζει το ελάχιστο άνω φράγμα, ο όρος "πλησιέστερη" δε θα είχε κανένα νόημα.

$... + V_n$.

Αυτή η αρχή καθορίζει ότι για την κατασκευή, για παράδειγμα, της έκδοσης `3.2` ενός αντικειμένου επιλέγεται η έκδοση `2.8.2`, όταν αυτό υπάρχει στις εκδόσεις `3.4, 2.8.2, 2.7.9, 1.8, 1.5.6` και ϵ, αφού $R = \{$`2.8.2, 2.7.9, 1.8, 1.5.6`, $\epsilon\}$ και το μέγιστο στοιχείο αυτού είναι η `2.8.2`. Καθορίζει ότι στην κατασκευή της έκδοσης `K%apple%fast` επιλέγεται η `K%apple` έκδοση ενός εξαρτήματος που υπάρχει στις εκδόσεις `K, K%apple, K%fast, apple%fast, M%apple, fast` και ϵ, αφού $R = \{$`K, K%apple`, $\epsilon\}$ και το μέγιστο στοιχείο αυτού είναι η `K%apple`.

Επίσης, εξηγεί πώς ο τελεστής + επιλύει το πρόβλημα του συνδυασμού εκδόσεων, για παράδειγμα, του συνδυασμού των εκδόσεων `orange` του M και `apple` του K. Η ζητούμενη έκδοση του συστήματος θα είναι η `M%orange + K%apple`. Σύμφωνα με τον κανόνα, για κάθε εξάρτημα επιλέγεται η πλησιέστερη στη `M%orange` έκδοση όταν δεν υπάρχει `K` έκδοση διαθέσιμη, και η πλησιέστερη στην `K%apple` όταν δεν υπάρχει `M` έκδοση διαθέσιμη. Ετσι, αν οι διαθέσιμες εκδόσεις του εξαρτήματος C είναι οι `K%pear, F%apple, K, apple` και ϵ, επιλέγεται η `K`, αφού $R = \{$`K`, $\epsilon\}$ και το μέγιστο στοιχείο αυτού είναι η `K`. Από την άλλη, αν είναι οι `K%apple, M` και ϵ, τότε δεν υπάρχει βέλτιστη επιλογή και το σύστημα δεν υπάρχει στη ζητούμενη έκδοση, αφού $R = \{$`K%apple, M`, $\epsilon\}$ και δεν υπάρχει μέγιστο στοιχείο. Η λύση σε αυτή την περίπτωση είναι να κατασκευάσουν ο K και ο M, από κοινού, μία αμοιβαία αποδεκτή έκδοση του εξαρτήματος, που θα είναι συμβατή με τις K%apple και `M`. Αξίζει εδώ να σημειώσουμε ότι είναι δυνατό να κατασκευασθεί η έκδοση `M%orange + K%apple` ακόμα και αν δεν υπάρχει κανένα εξάρτημα σε αυτή την έκδοση.

5.2 "Αδυναμίες" του Μοντέλου Αλγεβρας Εκδόσεων

Στην προηγούμενη ενότητα παρουσιάσαμε τους κανόνες και τα αξιώματα του μοντέλου μίας άλγεβρας εκδόσεων, η οποία μπορεί να χρησιμοποιηθεί για την αυτόματη κατασκευή ενός ολοκληρωμένου συστήματος. Στην παρούσα ενότητα θα δούμε ορισμένες "αδυναμίες" της άλγεβρας. Η έκφραση "αδυναμίες της άλγεβρας" έχει το νόημα ότι με τη δεδομένη μορφή του, το μοντέλο της άλγεβρας δε μπορεί να εκφράσει ορισμένες βασικές έννοιες του configuration management και ειδικότερα της επιλογής εκδόσεων. Αυτές είναι οι παρακάτω:

1. Αδυναμία ορισμού μοντέλου συστήματος, δηλαδή, με το μοντέλο της άλγεβρας εκδόσεων δε μπορούμε να ορίσουμε σαφώς τη δομή μίας διαμόρφωσης, με άλλα λόγια, το σύνολο των συσχετιζομένων εξαρτημάτων τα οποία ορίζουν ένα υπό ανάπτυξη σύστημα (προϊόν).

Ετσι, ο γενικός κανόνας της άλγεβρας, ο οποίος ορίζει πώς μπορεί να κατασκευαοθεί μία έκδοση ενός συστήματος S που αποτελείται από τα εξαρτήματα $C_1,\ C_2,\ ...\ C_n$, μπορεί να εφαρμοοθεί μόνο θεωρητικά και θεωρώντας δεδομένες τις σχέσεις ότι το $C_i,\ i = 1...n$, είναι εξάρτημα του S και η V είναι μία έκδοση του C_i.

2. Δεν είναι δυνατός ο διαχωρισμός μεταξύ διασκευασμένων εκδόσεων (*revisions*) και παραλλαγών (*variants*), κάτι που οι Plaice και Wadge παραδόξως θεωρούν επίτευγμά τους [42]. Ο τελεστής % που χρησιμοποιείται για να δηλώσει τις υποεκδόσεις δεν είναι αρκετός. Αυτό φαίνεται στα δύο παρακάτω παραδείγματα. Εστω, οι εκδόσεις `graphics%mouse` και `graphics%keyboard`, όπου η πρώτη αποτελεί τη γραφική έκδοση ενός συστήματος διασύνδεσης, στην οποία χρησιμοποιείται "ποντίκι" και η δεύτερη τη γραφική έκδοση στην οποία χρησιμοποιείται το πληκτρολόγιο. Είναι φανερό ότι μεταξύ των δύο, υπάρχει διαφορά στη λειτουργικότητά τους και η μία αποτελεί παραλλαγή της άλλης. Ετσι μπορεί η μία να αντικαταστήσει την άλλη ανάλογα με τις απαιτήσεις των χρηστών. Εστω τώρα ότι υπάρχουν δύο εν χρήσει εκδόσεις ενός συστήματος, η `2.3.4` και η `3.5.6`, και δύο άτομα ελέγχουν την `2.3.4`, ανεξάρτητα ο ένας από τον άλλο, για λάθη. Υποθέτουμε ότι και οι δύο ανακαλύπτουν λάθη, διαφορετικά μεταξύ τους, και δημιουργούν δύο νέες εκδόσεις, ο ένας την `2.3.4%bugfixX` και ο άλλος την `2.3.4%bugfixY` (Δε μπορούν να χρησιμοποιηθούν αριθμητικές εκδόσεις για τους λόγους που αναφέρονται στο αντίστοιχο παράδειγμα της ενότητας "Υποεκδόσεις".). Οι δύο αυτές εκδόσεις είναι φανερό, από όσα είπαμε και όχι από τον τελεστή % , ότι δεν αποτελούν παραλλαγή η μία της άλλης αλλά στην ουσία είναι δύο διασκευές της `2.3.4` με την οποία (καθώς και μεταξύ τους) έχουν την ίδια λειτουργικότητα.

Αλλο ένα πρόβλημα του μοντέλου της άλγεβρας, όχι τόσο σημαντικό όσο τα προηγούμενα αλλά που δεν παύει όμως να υπάρχει, είναι ότι, κάθε φορά που γίνονται αλλαγές σε μία έκδοση ενός στοιχείου και δημιουργείται μία διασκευή αυτής, πρέπει να δίνεται ένα νέο όνομα. Για παράδειγμα, έστω οι εκδόσεις `2.3.4` και `3.5.6` που αναφέραμε προηγουμένως, και έστω ότι ένας προγραμματιστής ελέγχει την `2.3.4` για λάθη. Βρίσκει κάποια, κάνει αλλαγές και δημιουργεί την `2.3.4%bugfix`. Σε αυτή εμφανίζονται κάποια άλλα προβλήματα, κάνει νέες αλλαγές και δημιουργεί την `2.3.4%bugfix%bugfix`. Αν η κατάσταση των διορθώσεων και αλλαγών συνεχισθεί, γεγονός διόλου απίθανο, θα καταλήξει σε ένα όνομα με αρκετά `bugfix` το οποίο θα είναι εντελώς δύσχρηστο.

3. Μία άλλη αδυναμία του μοντέλου της άλγεβρας εκδόσεων είναι ότι δεν υπάρχει η δυνατότητα έκφρασης περιορισμών μεταξύ εκδόσεων διαφορετικών αντικειμένων, οπότε δεν είναι δυνατό να κατασκευασθεί αυτόματα μία συνεπής έκδοση ενός συστήματος, αφού ο μόνος έλεγχος που γίνεται (όπως αναφέραμε και στην ενότητα "Ενώσεις Εκδόσεων") είναι συντακτικός στο επίπεδο των ονομάτων των εκδόσεων, μέσω των αξιωμάτων της άλγεβρας. Ετσι, απαιτείται κάθε φορά ο έλεγχος της διαμόρφωσης που προκύπτει, από κάποιον (διαχειριστής εκδόσεων) που θα εγγυηθεί ότι έχει νόημα ο συνδυασμός των εξαρτημάτων που επιλέχθηκαν.

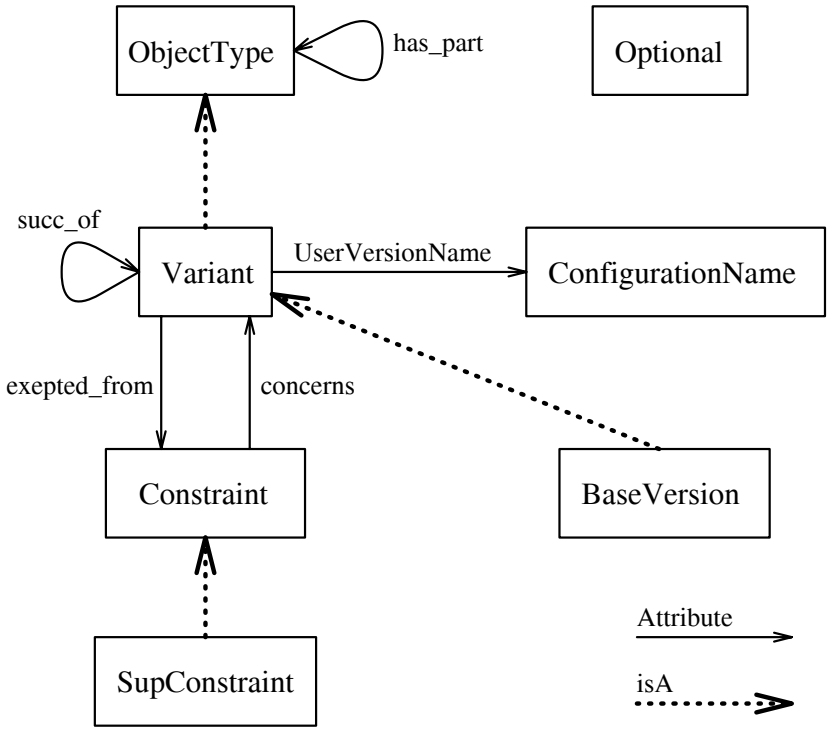
Για παράδειγμα, έστω ένα σύστημα S, το οποίο αποτελείται από τα στοιχεία A και B. Το A υπάρχει στις εκδόσεις `x, x%y, x%y%z` και ϵ ενώ το B υπάρχει στις εκδόσεις `w, x, x%y` και ϵ. Εστω ότι λόγω κάποιων αλλαγών η έκδοση `x%y%z` του A δε μπορεί, προς το παρόν, να συνδυασθεί με την έκδοση `x%y` του B. Αυτός ο περιορισμός δε μπορεί να εκφρασθεί με την άλγεβρα των εκδόσεων, οπότε αν ο χρήστης ζητήσει την έκδοση `x%y%z` του S θα επιλεγούν : η έκδοση `x%y%z` για το στοιχείο A και η `x%y` για το B. Ως γνωστόν όμως, ο συνδυασμός των συγκεκριμένων εκδόσεων είναι προβληματικός αφού οδηγεί στην κατασκευή μίας μη συνεπής έκδοσης του συστήματος.

4. Τέλος, στο μοντέλο της άλγεβρας δε μπορούμε να δηλώσουμε την ύπαρξη προαιρετικών εξαρτημάτων για ένα σύστημα. Για παράδειγμα, έστω ένα σύστημα S, το οποίο αποτελείται από τα στοιχεία C_1, C_2, ... C_n. Για να κατασκευασθεί μία έκδοση του S πρέπει, σύμφωνα με το γενικό κανόνα, να επιλεγεί μία έκδοση από κάθε εξάρτημα C_i, $i = 1...n$. Μπορεί όμως αυτό που πραγματικά θέλουμε να είναι το εξής : για μερικές εκδόσεις του S να μην επιλέγεται έκδοση από το C_1, για κάποιες άλλες να μην επιλέγεται έκδοση του C_n και για άλλες να μην επιλέγεται ούτε του C_1 ούτε του C_n. Κάτι τέτοιο δεν είναι δυνατό να εκφρασθεί με την άλγεβρα των εκδόσεων. Η "αδυναμία" αυτή υπάρχει όχι μόνο στο μοντέλο που παρουσιάσαμε στην προηγούμενη ενότητα, αλλά γενικότερα δεν αντιμετωπίζεται σε κανένα από τα συστήματα για configuration management που εξετάσαμε στην βιβλιογραφία.

Στη συνέχεια προτείνουμε ένα μοντέλο για configuration management με το οποίο μπορούμε να αντιμετωπίσουμε τις "αδυναμίες" που αναφέραμε παραπάνω.

5.3 Μοντέλο για Configuration Management

Με το μοντέλο για configuration management που θα προτείνουμε σε αυτή την ενότητα - του οποίου μία ολική σχηματική παράσταση δίνεται με το σχήμα 5.1 - επιχειρούμε να αντιμετωπίσουμε τα προβλήματα που αναφέραμε στην προηγούμενη ενότητα αλλά και να κρατήσουμε πληροφορία (σχετική με εκδόσεις αντικειμένων), η οποία θα μας επιτρέψει να κατασκευάσουμε, αυτόματα, ένα σύστημα από τα εξαρτήματά του με χρήση των αξιωμάτων και των κανόνων της άλγεβρας εκδόσεων που περιγράψαμε στην αρχή του κεφαλαίου.



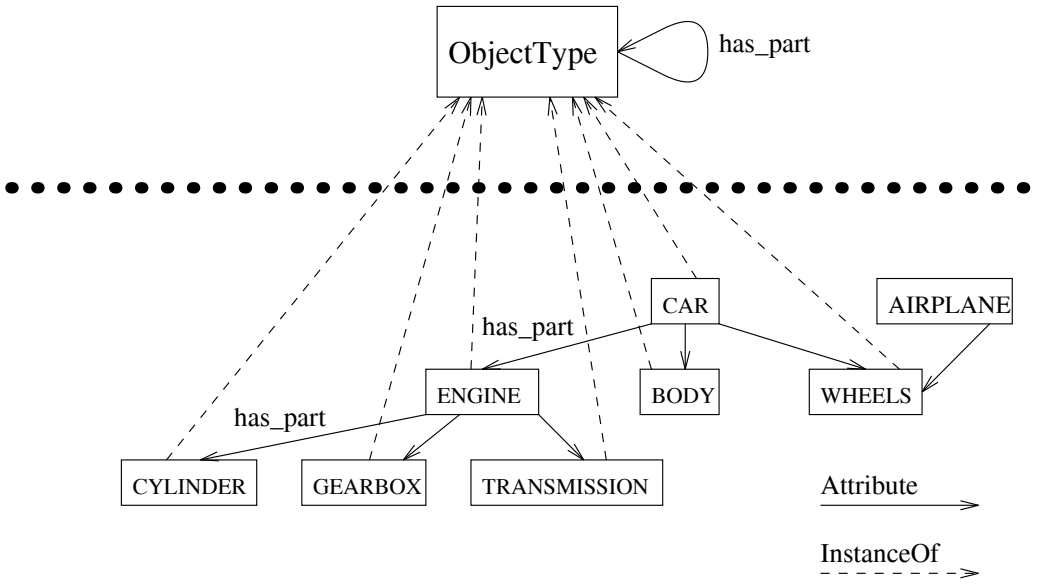
Σχήμα 5.1: Σχηματική παράσταση του μοντέλου για configuration management

Το μοντέλο έχει παρασταθεί στη γλώσσα TELOS (μπορούμε να δούμε την περιγραφή στο Παράρτημα Α) με ένα σύνολο οντοτήτων και σχέσεων μεταξύ αυτών. Ακολουθεί αναλυτική περιγραφή των κλάσεων του μοντέλου (θα γράφονται με **τονισμένα** γράμματα) και των γνωρισμάτων τους (θα γράφονται με ***πλάγια τονισμένα*** γράμματα).

5.3.1 Μοντέλο Συστήματος

Η κλάση **Object** (αντικείμενο), με τη χρήση του γνωρίσματος ***has_part*** ορίζει τη δομή μίας διαμόρφωσης, δηλαδή το σύνολο των συσχετιζομένων εξαρτημάτων που ορίζουν ένα υπό

ανάπτυξη σύστημα. Ετσι, ένα σύστημα (στιγμιότυπο της **Object**) αποτελείται από ένα σύνολο απλών, σύνθετων ή συνδυασμό και των δύο, αντικειμένων (επίσης στιγμιοτύπων της **Object**). Ενα σύνθετο αντικείμενο συντίθεται από άλλα αντικείμενα, σύνθετα ή απλά, ενώ αντίθετα ένα απλό δε συντίθεται από κανένα. Επομένως, ένα σύστημα είναι ένα σύνθετο αντικείμενο και το αντίστροφο. Για παράδειγμα, μπορούμε να δούμε στο σχήμα 5.2 το σύνθετο αντικείμενο



Σχήμα 5.2: Παράδειγμα μοντέλου συστήματος, με απλά και σύνθετα αντικείμενα, στιγμιότυπα της κλάσης **Object**

`CAR` (σύστημα), το οποίο αποτελείται από το σύνθετο αντικείμενο `ENGINE` και τα απλά αντικείμενα `BODY` και `WHEELS`. Το σύνθετο αντικείμενο `ENGINE` (σύστημα) συντίθεται από τα απλά αντικείμενα `CYLINDER, GEARBOX` και `TRANSMISSION`.

Επιπλέον της δομής διαμόρφωσης, η σχέση ***has_part*** ορίζει μία ιεραρχία μεταξύ των αντικειμένων που συγκροτούν ένα σύστημα (*component hierarchies* [27]). Στο σχήμα 5.2 παρουσιάζεται μία ιεράρχηση των αντικειμένων σε δομή δένδρου, όπου μπορούμε να παρατηρήσουμε ότι τα απλά αντικείμενα είναι "φύλλα" του δένδρου. Γενικότερα όμως, επιτρέπονται ιεραρχήσεις με δομή διευθυνόμενων, άκυκλων γράφων (DAG). Ετσι, ένα συγκεκριμένο αντικείμενο (εξάρτημα) μπορεί να χρησιμοποιείται ταυτόχρονα από περισσότερα από ένα σύνθετα αντικείμενα (συστήματα), αποδίδοντας μία δομή διευθυνόμενου, άκυκλου γράφου. Για παράδειγμα, το αντικείμενο `WHEELS` μπορεί να είναι στοιχείο τόσο του αντικειμένου `CAR`, όσο και ενός νέου αντικειμένου `AIRPLANE`.

5.3.2 Εκδόσεις Αντικειμένων

Η κλάση **Variant** δηλώνει το σύνολο των εκδόσεων όλων των αντικειμένων, δηλαδή, κάθε φορά που δημιουργείται μία νέα έκδοση ενός αντικειμένου (στιγμιοτύπου της **Object**), αυτή ορίζεται να είναι στιγμιότυπο της **Variant**. Τα γνωρίσματα της είναι τα παρακάτω :

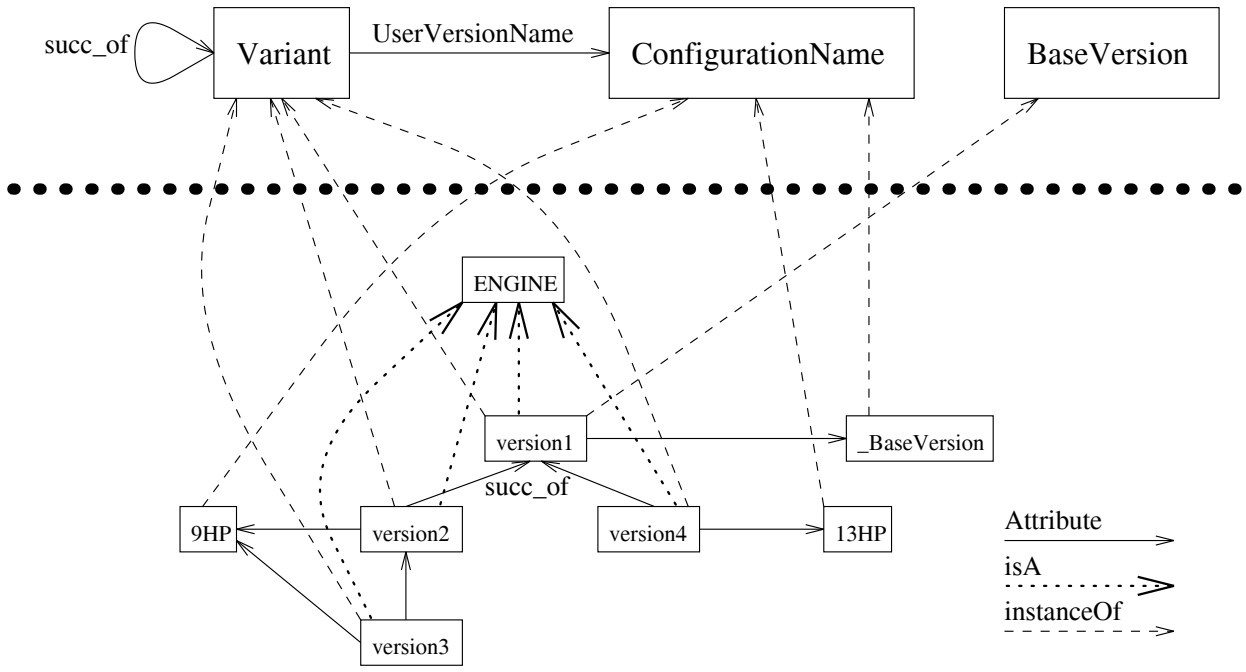
- ***succ_of*** : Καταγράφει τη σχέση προγόνου / απογόνου μεταξύ διαφορετικών εκδόσεων (στιγμιοτύπων της **Variant**), που ανήκουν στο ίδιο αντικείμενο (στιγμιότυπο της **Object**). Οι εκδόσεις ενός συγκεκριμένου αντικειμένου συνδέονται με αυτό μέσω της σχέσης isA (όπως θα δούμε και στο επόμενο κεφάλαιο), αφού στην πραγματικότητα, κάθε έκδοση είναι μία διαφορετική υλοποίηση ενός αντικειμένου.

  Το γνώρισμα ***succ_of*** αντιστοιχεί στη σχέση *διασκευή-του*, όπως την ορίσαμε στην παράγραφο 2.2.1 . Μέσω αυτού, μπορούμε πλέον να διαχωρίσουμε πότε δύο η περισσότερες εκδόσεις είναι παραλλαγές (*variants*) και πότε διασκευές (*revisions*). Εάν δημιουργηθεί μία νέα έκδοση ενός αντικειμένου, έστω V', από την τροποποίηση μιας ήδη υπάρχουσας έκδοσης αυτού, έστω V, τότε αυτές οι δύο συνδέονται με τη σχέση ***succ_of***, δηλαδή, V' ***succ_of*** V. Εστω V_k και V_l δύο διαφορετικές εκδόσεις ενός αντικειμένου O. Θα λέμε ότι η V_l είναι διασκευασμένη έκδοση (ή διασκευή) της V_k αν και μόνο αν υπάρχουν n εκδόσεις του O, $n \geq 0$, οι οποίες συνδέονται με τη ***succ_of*** ως εξής : V_l ***succ_of*** v_n ***succ_of*** v_{n-1} ***succ_of*** ... ***succ_of*** v_1 ***succ_of*** V_k. Αν η V_l δεν είναι διασκευασμένη έκδοση της V_k τότε είναι παραλλαγή της.

- ***UserVersionName*** : Αναφέρεται στο όνομα που δίνει ο χρήστης σε μία έκδοση ενός αντικειμένου, το οποίο είναι διαφορετικό από το όνομα που παίρνει αυτόματα η έκδοση από το σύστημα. Παίρνει τιμές στην κλάση **ConfigurationName**, η οποία δηλώνει το σύνολο των ονομάτων που έχουν εκχωρηθεί από χρήστες σε εκδόσεις αντικειμένων. Πάνω στα ονόματα των εκδόσεων εφαρμόζεται η άλγεβρα που περιγράψαμε στην ενότητα 5.1, για να επιλέξει την πλησιέστερη στη ζητούμενη έκδοση ενός αντικειμένου, κατά την κατασκευή μιας έκδοσης συστήματος. Το όνομα κάθε έκδοσης δεν είναι κατ' ανάγκη μοναδικό. Ετσι, για παράδειγμα, δύο εκδόσεις δύο διαφορετικών αντικειμένων μπορούν να έχουν το ίδιο όνομα. Επίσης, εκδόσεις που συνδέονται με τη ***succ_of***, μπορούν να διατηρούν το όνομα του προγόνου τους, γεγονός που διευκολύνει το χρήστη, γιατί δε χρειάζεται να δίνει ένα νέο όνομα κάθε φορά που, π.χ. ανακάλυψε ένα λάθος και δημιουργεί μία διασκευασμένη έκδοση (βλέπε δεύτερη παράγραφο του τμήματος 2 της ενότητας 5.2). Με αυτό το ζήτημα θα ασχοληθούμε και στο επόμενο

κεφάλαιο.

- ***excepted_from*** : Παίρνει τιμές στην κλάση **Constraint** και θα το δούμε πιο αναλυτικά στη συνέχεια, όταν μιλήσουμε για περιορισμούς.



Σχήμα 5.3: Παράδειγμα εκδόσεων αντικειμένων

Παρατηρούμε ότι η έκδοση `version1`, που είναι έκδοση βάσης του αντικειμένου `ENGINE`, είναι στιγμιότυπο δύο κλάσεων : της **Variant** και της **BaseVersion**.
Η οντότητα **_BaseVersion** αντιστοιχεί στο συμβολισμό ϵ της άλγεβρας και είναι το όνομα έκδοσης που εκχωρείται αυτόματα από το σύστημα σε κάθε έκδοση βάσης.
Ο σύνδεσμος instanceOf μεταξύ του αντικειμένου `ENGINE` και της κλάσης **Object** έχει παραλειφθεί για λόγους απλότητας του σχήματος.

Στην ενότητα 5.1 είχαμε πει ότι η πιο απλή άλγεβρα θα επιτρέπει μόνο μία έκδοση, την οποία ονομάσαμε *έκδοση βάσης*. Επίσης, για ένα εξάρτημα, αυτή είναι η έκδοση που επιλέγεται κατά την κατασκευή μίας έκδοσης συστήματος, αν δεν υπάρχει άλλη πλησιέστερη στη ζητούμενη έκδοση. Η κλάση **BaseVersion** δηλώνει το σύνολο των εκδόσεων που έχουν χαρακτηριοθεί εκδόσεις βάσης.

Στο σχήμα 5.3 παρουσιάζεται ένα παράδειγμα εκδόσεων αντικειμένων, όπου φαίνονται οι παραπάνω κλάσεις και στοιχεία τους.

5.3.3 Περιορισμοί

Οι κλάσεις **Constraint** και **SupConstraint** εκφράζουν μαζί, το σύνολο των περιορισμών που έχουν οριοθεί από το χρήστη. Ενας περιορισμός υπάρχει μόνο μεταξύ εκδόσεων διαφορετικών αντικειμένων, που ανήκουν στο ίδιο σύστημα. Οι εκδόσεις που ανήκουν σε έναν περιορισμό καθορίζονται με το γνώρισμα ***concerns*** της **Constraint**, το οποίο παίρνει τιμές στην κλάση **Variant**, και είναι γνώρισμα και της **SupConstraint**, αφού, όπως μπορούμε να δούμε και στο σχήμα 5.1, οι δύο κλάσεις συνδέονται με μία σχέση isA, μέσω της οποίας τα γνωρίσματα της **Constraint** κληρονομούνται από τη **SupConstraint**. Αν παρατηρήσουμε πάλι το σχήμα 5.1 θα δούμε ότι η κλάση **SupConstraint** δεν έχει, σε σχέση με την **Constraint**, επιπλέον γνωρίσματα και στην ουσία είναι ίδια με αυτή. Η διαφορά μεταξύ των δύο έγκειται στο ότι η **SupConstraint** δηλώνει το σύνολο των περιορισμών, των οποίων τμήματα τους υπάρχουν ήδη σαν περιορισμοί (στιγμιότυπα της **Constraint**). Για παράδειγμα, έστω ότι υπάρχει ένας περιορισμός `constraint1` (στιγμιότυπο της **Constraint**) με στοιχεία τις εκδόσεις : V_1 του C_1 και V_2 του C_2. Αν στη συνέχεια χρειασθεί ένας νέος περιορισμός `constraint2` με στοιχεία τις : V_1 του C_1, V_2 του C_2 και V_3 του C_3, θα πρέπει να οριοθεί στιγμιότυπο της **SupConstraint**, και να συνδεθεί μέσω της σχέσης isA με τον `constraint1` (από τον οποίο θα κληρονομήσει τα στοιχεία V_1 και V_2) και μέσω της ***concerns*** με το V_3.

Η κλάση **SupConstraint** εκφράζει τους περιορισμούς, για τους οποίους, περιορίζεται το σύνολο των στοιχείων που αφορούν, σε σχέση με το σύνολο των στοιχείων που ισχύουν οι περιορισμοί που συνδέονται μαζί τους μέσω της σχέσης isA. Για παράδειγμα, έστω ένας περιορισμός (`constraintX`) (μέλος της **Constraint**), ο οποίος ισχύει για όλες τις εκδόσεις του συστήματος `CAR` με κίτρινο χρώμα, και ένας άλλος (`constraintY`) (μέλος της **Constraint**) που ισχύει για όλες τις εκδόσεις του `CAR` με πετρελαιοκινητήρα. Ενας τρίτος περιορισμός,

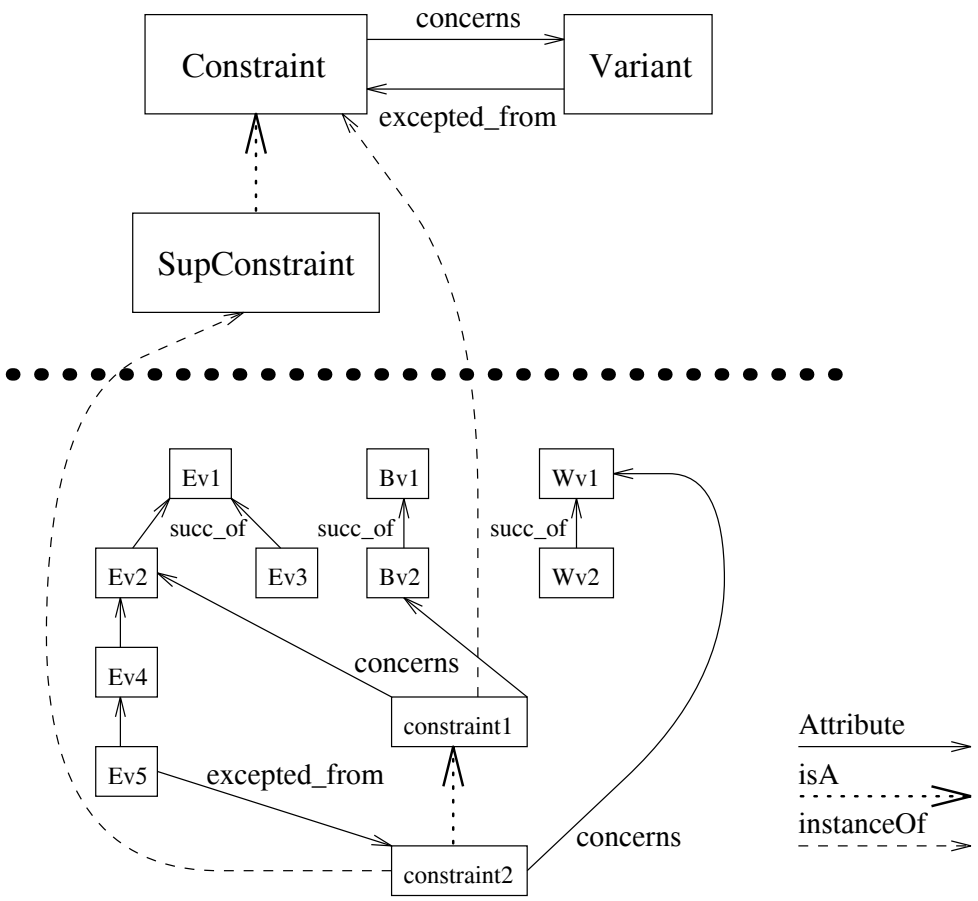
$$\texttt{constraintZ}\ \text{isA} \begin{cases} \texttt{constraintX} \\ \texttt{constraintY} \end{cases}$$

(μέλος της **SupConstraint**) περιορίζει το σύνολο εκδόσεων του `CAR`, και συγκεκριμένα ισχύει για τις εκδόσεις με : κίτρινο χρώμα **και** πετρελαιοκινητήρα. Επίσης, η διαφοροποίηση σε **Constraint** και **SupConstraint** χρησιμεύει και στην εισαγωγή των περιορισμών. Το πώς θα το δούμε στο επόμενο κεφάλαιο.

Η έννοια ενός περιορισμού [2] είναι αυτή του αμοιβαίου αποκλεισμού, δηλαδή οι

[2]Ενα παρόμοιο είδος περιορισμών χρησιμοποιείται από το CMA [43] . Υπάρχουν, φυσικά, και άλλα είδη, όπως για παράδειγμα στο [26], όπου προτείνεται ένα σύνολο περιορισμών - ονομάζονται περιορισμοί διαμόρφωσης (*configuration constraints*) - που χρησιμεύουν σα μηχανισμός για τον περιορισμό της διάδοσης των αλλαγών.

συγκεκριμένες εκδόσεις, που αποτελούν στοιχεία του, δε λειτουργούν σωστά μαζί και επομένως, αν κατά την κατασκευή μίας έκδοσης συστήματος επιλεγούν αυτές (μέσω της άλγεβρας εκδόσεων), τότε το σύστημα δεν πρέπει να γίνεται αποδεκτό γιατί δε θα λειτουργεί ορθά. Για παράδειγμα, έστω ένα σύστημα S που αποτελείται από τα εξαρτήματα C_1, C_2, C_3 και έχει οριοθεί ο περιορισμός `constraint1` (προηγούμενο παράδειγμα). Αν για την κατασκευή μίας έκδοσης του S επιλεγούν οι V_1, V_2 και V_3', αντίστοιχα για κάθε ένα από τα εξαρτήματα, το σύστημα δεν είναι αποδεκτό γιατί ισχύει ο `constraint1`.



Σχήμα 5.4: Παράδειγμα περιορισμών

Οι οντότητες `Ev1, Ev2, Ev3, Ev4, Ev5` είναι εκδόσεις του αντικειμένου `ENGINE`, οι `Bv1, Bv2` του `BODY` και οι `Wv1, Wv2` του `WHEELS` (βλέπε σχήμα 5.2). Για λόγους απλότητας του σχήματος παραλείπονται τα προαναφερθέντα αντικείμενα και οι σύνδεσμοι isA μεταξύ αυτών και των εκδόσεών τους, οι σύνδεσμοι instanceOf μεταξύ των εκδόσεων και της κλάσης **Variant** και τα ονόματα εκδόσεων (στιγμιότυπα της **ConfigurationName**).

Ενας περιορισμός συνδέει τα στοιχεία του με λογικό ΚΑΙ. Ετσι, αν ο περιορισμός `X` αφορά τις εκδόσεις V_1, V_2, ... V_n, των αντίστοιχων εξαρτημάτων, θα ισχύει μόνο αν υπάρχουν και

οι n παραπάνω εκδόσεις και όχι κάποιος συνδυασμός τους. Δηλαδή, αν για τη δημιουργία μίας έκδοσης συστήματος έχουν επιλεγεί οι V_1', V_2, ... V_n, ο X δεν ισχύει.

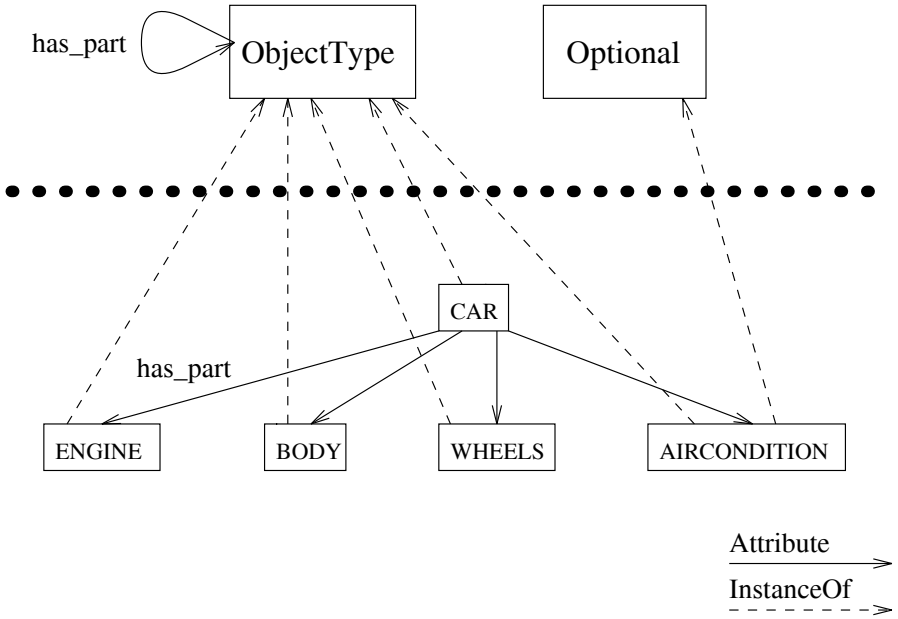
Τέλος, αν ένας περιορισμός X αφορά μία έκδοση V, θα αφορά και όλες τις εκδόσεις, V^1, V^2, ... V^n, που είναι απόγονοι (διασκευασμένες εκδόσεις) της V. Ο συσχετισμός μεταξύ του X και των V^1, V^2, ... V^n δεν είναι άμεσος (όπως με τη V, μέσω της σχέσης X ***concerns*** V) αλλά έμμεσος και "κληρονομείται" (στο επόμενο κεφάλαιο θα δούμε το πώς) μέσω των σχέσεων ***succ_of*** - όχι αυτόματα, όπως συμβαίνει στην ιεραρχία isA - που υπάρχουν μεταξύ των V, V^1, V^2, ... V^n. Ομως, κατά την εξέλιξη μίας έκδοσης είναι πιθανό, από κάποιο σημείο και μετά, να μη θέλουμε να ισχύει ένας περιορισμός. Το σκοπό αυτό εξυπηρετεί το γνώρισμα ***excepted_from*** της κλάσης **Variant**, με το οποίο, ακυρώνεται η ισχύς ενός περιορισμού για μία έκδοση και για όλες τις διαδόχους της. Για παράδειγμα, έστω ένας περιορισμός Y και οι εκδόσεις V^1, V^2, ... V^i, V^{i+1}, ... V^n τέτοιες ώστε η μία είναι διασκευασμένη έκδοση της άλλης, δηλαδή, V^n ***succ_of*** V^{n-1} .... Επίσης, υποθέτουμε ότι ένα από τα στοιχεία του περιορισμού είναι η έκδοση V^2 (Y ***concerns*** V^2) και η V^{i+1} εξαιρείται από τον περιορισμό Y (V^{i+1} ***excepted_from*** Y). Από τα παραπάνω προκύπτει ότι ο Y αφορά τις εκδόσεις V^2, ... V^i ενώ εξαιρούνται οι V^{i+1}, ... V^n.

Στο σχήμα 5.4 παρουσιάζεται ένα παράδειγμα περιορισμών, όπου φαίνονται οι παραπάνω κλάσεις και στοιχεία τους.

5.3.4 Προαιρετικά Αντικείμενα

Η κλάση **Optional** δηλώνει το σύνολο των αντικειμένων που χαρακτηρίζονται προαιρετικά. Ενα αντικείμενο (στιγμιότυπο της κλάσης **Object**) είναι προαιρετικό εάν δεν είναι υποχρεωτική η συμμετοχή του στην κατασκευή ενός συστήματος [3]. Για παράδειγμα, ο κλιματισμός AIRCONDITION δεν είναι υποχρεωτικός για να υπάρξει ένα σύστημα αυτοκινήτου CAR, αλλά ένα αυτοκίνητο μπορεί να έχει κλιματισμό. Ετσι, όπως φαίνεται και στο σχήμα 5.5, η οντότητα AIRCONDITION, που είναι εξάρτημα του αντικειμένου CAR, θα είναι στιγμιότυπο της κλάσης **Optional**. Με αυτό τον τρόπο μπορούμε να κατασκευάσουμε εκδόσεις του CAR με ή χωρίς εκδόσεις του AIRCONDITION.

[3] Ολα τα αντικείμενα, ανεξάρτητα από το αν είναι προαιρετική ή υποχρεωτική η συμμετοχή τους στη συγκρότηση ενός συστήματος, μπορούν να έχουν εκδόσεις.



Σχήμα 5.5: Παράδειγμα προαιρετικών αντικειμένων

Παρατηρούμε ότι το αντικείμενο `AIRCONDITION` που θέλουμε να μην είναι υποχρεωτική η συμμετοχή του κατά την κατασκευή του αντικειμένου `CAR` είναι στιγμιότυπο δύο κλάσεων, της **Object** και της **Optional**.

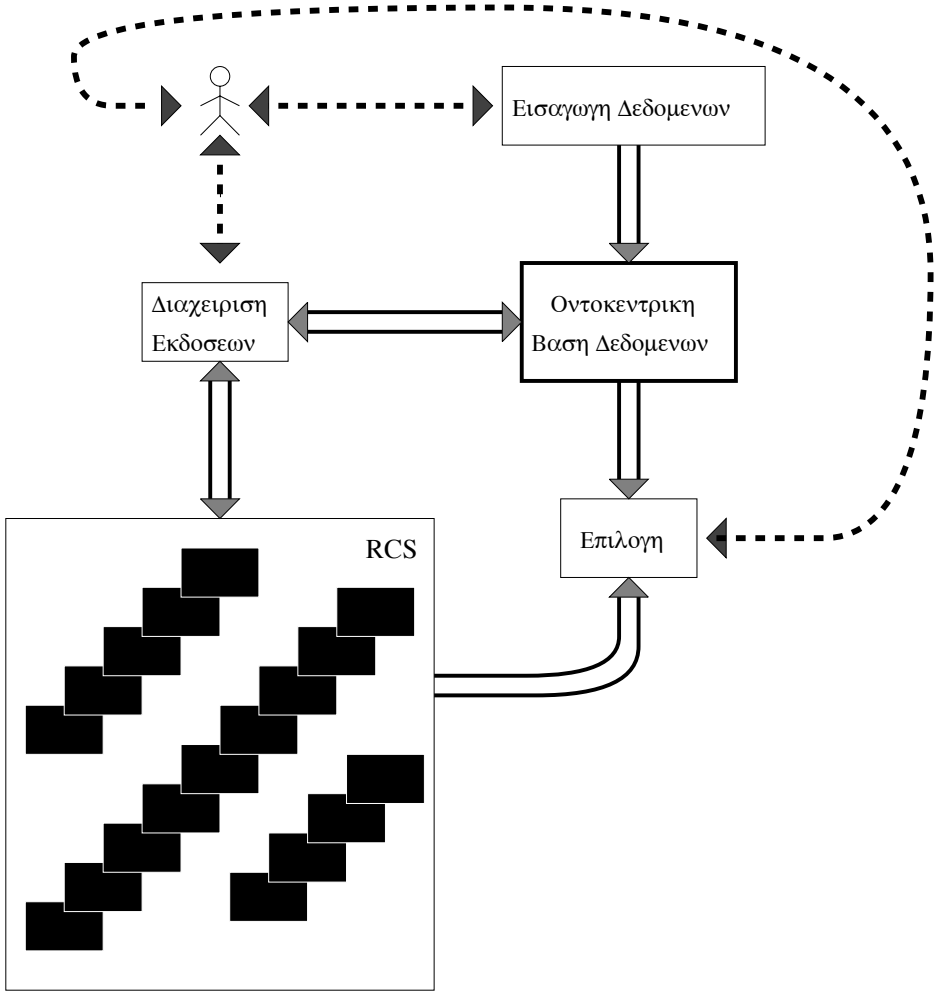
Κεφάλαιο 6

Υλοποίηση Συστήματος για Configuration Management

Στο προηγούμενο κεφάλαιο προτείναμε ένα μοντέλο για configuration management, που αποτελεί κατά κάποιο τρόπο, ένα συμπλήρωμα του μοντέλου της άλγεβρας εκδόσεων, που είδαμε στο ίδιο κεφάλαιο. Στο παρόν κεφάλαιο θα παρουσιάσουμε την υλοποίηση ενός συστήματος για CM, το οποίο βασίζεται στο παραπάνω μοντέλο. Το σύστημά μας, που μπορούμε να το δούμε γραφικά στο σχήμα 6.1, αποτελείται από τρία επιμέρους τμήματα, από τα οποία, ένα χρησιμεύει στη διαχείριση των εκδόσεων των αντικειμένων, το άλλο στην εισαγωγή δεδομένων και συγκεκριμένα των περιορισμών στη βάση δεδομένων και το τελευταίο επιλέγει εκδόσεις για την κατασκευή μίας έκδοσης συστήματος και ελέγχει την ύπαρξη περιορισμών μεταξύ τους. Στη συνέχεια του κεφαλαίου θα δούμε αναλυτικά κάθε ένα από τα προαναφερθέντα τμήματα και τέλος θα μιλήσουμε για τις δοκιμές που κάναμε προκειμένου να ελέγξουμε την ορθή λειτουργία του συστήματος.

6.1 Διαχείριση Εκδόσεων

Για τη διαχείριση εκδόσεων αντικειμένων υλοποιήσαμε τις δύο πιο σημαντικές εντολές (μπορούμε να δούμε τη σύνταξη τους, αναλυτικά, στο Παράρτημα Β), της εισαγωγής (*check-in*) και της εξαγωγής εκδόσεων (*check-out*). Επειδή δεν αποτελούσε στόχο της παρούσας εργασίας η κατασκευή ενός εργαλείου για έλεγχο εκδόσεων (*version control*) αποφασίσαμε να χρησιμοποιήσουμε για το σκοπό αυτό το RCS, και συγκεκριμένα την έκδοση 5.6.0.1, ένα ευρύτατα διαδεδομένο σύστημα ελέγχου εκδόσεων. Ο ρόλος του RCS περιορίζεται στην αποθήκευση των εκδόσεων των αντικειμένων (στη συγκεκριμένη περίπτωση τα αντικείμενα



Σχήμα 6.1: Σχηματική παράσταση του συστήματος για configuration management

είναι αρχεία κειμένου) σε δέλτα μορφή, μέσα σε αρχεία κειμένου του συστήματος UNIX, και στην αναδημιουργία τους σε κανονική μορφή. Για να είναι δυνατή η ορθή λειτουργία του υποσυστήματος διαχείρισης εκδόσεων χρειάστηκε ένα σύστημα διασύνδεσης των δικών μας εντολών εισαγωγής και εξαγωγής εκδόσεων και των αντίστοιχων του RCS (`ci` και `co`) για να επιτευχθεί η μεταξύ τους επικοινωνία (γεγονός που μας οδήγησε σε προσθήκες στον κώδικα των εντολών `ci` και `co`). Στις δύο επόμενες ενότητες θα μιλήσουμε, αντίστοιχα, για την εισαγωγή και την εξαγωγή εκδόσεων.

6.1.1 Εισαγωγή Εκδόσεων

Με την εντολή της εισαγωγής αποθηκεύουμε μία έκδοση ενός αντικειμένου σε δέλτα μορφή, με χρήση της τροποποιημένης εντολής `ci` του RCS, από την οποία παίρνουμε και πληροφορίες,

μέσω των οποίων είμαστε σε θέση να γνωρίζουμε : τον αριθμό έκδοσης (που δίνει αυτόματα το RCS) της νέας έκδοσης που δημιούργησε ο χρήστης και της προγόνου της (π.χ. `1.2` και `1.1`, αντίστοιχα), αν υπάρχει κάποιο πρόβλημα και δε μπορεί να αποθηκευθεί η νέα έκδοση, αν πρόκειται για την πρώτη έκδοση και αν υπάρχουν αλλαγές μεταξύ της νέας έκδοσης και της προγόνου της.

Εάν δεν υπάρχει πρόβλημα και έχουν γίνει τροποποιήσεις μεταξύ της νέας έκδοσης και της προγόνου της, το σύστημα ζητάει από το χρήστη να δώσει ένα όνομα στη νέα έκδοση, δίνοντας του τη δυνατότητα να διαλέξει το όνομα της προγόνου της. Έτσι "κληρονομείται" το όνομα στην απόγονο έκδοση. Αν τώρα, είναι η πρώτη φορά που γίνεται εισαγωγή μίας έκδοσης του συγκεκριμένου αντικειμένου, της εκχωρείται αυτόματα το όνομα `_BaseVersion`, με το οποίο δηλώνεται ότι αποτελεί την έκδοση βάσης.

(Στο σημείο αυτό θα ανοίξουμε μία παρένθεση για να διαχωρίσουμε τις έννοιες του ονόματος έκδοσης και του ονόματος συστήματος. Από τώρα και στο εξής για το όνομα έκδοσης (χρησιμοποιείται από την άλγεβρα εκδόσεων) θα χρησιμοποιείται ο όρος *περιγραφικό όνομα έκδοσης*. Το *όνομα συστήματος* αποτελεί το αναγνωριστικό όνομα που εκχωρείται αυτόματα σε κάθε έκδοση από την εντολή εισαγωγής εκδόσεων και είναι μοναδικό. Συντίθεται από έναν ορμαθό χαρακτήρων, του οποίου, το πρώτο κομμάτι είναι το όνομα του αντικειμένου που διαχειριζόμαστε μία έκδοση του, το δεύτερο είναι ο χαρακτήρας "*" και το τελευταίο ο αριθμός έκδοσης που δίνει το RCS. Για παράδειγμα το όνομα συστήματος της έκδοσης `1.2` του αντικειμένου `ENGINE` θα είναι το `ENGINE*1.2`.)

Τέλος, αποθηκεύονται οι σχετικές με την εισαγόμενη έκδοση πληροφορίες, με χρήση δηλώσεων `TELL` της γλώσσας TELOS [14]. Έτσι, το όνομα συστήματος της έκδοσης συνδέεται μέσω της σχέσης ***succ_of*** με το όνομα συστήματος της προγόνου της και μέσω της σχέσης ***UserVersionName*** με το περιγραφικό όνομα της έκδοσης, το οποίο ορίζεται να είναι στιγμιότυπο της κλάσης **ConfigurationName**, ενώ το όνομα συστήματος τίθεται στιγμιότυπο της κλάσης **Variant** (και της **BaseVersion** αν είναι η πρώτη έκδοση) και συνδέεται με έναν σύνδεσμο isA με το αντικείμενο του οποίου αποτελεί έκδοση. Η αποθήκευση τους γίνεται σε μία οντοκεντρική βάση δεδομένων, που δημιουργείται και εξελίσσεται με το πέρασμα δηλώσεων της TELOS από τον compiler της.

Στην περίπτωση που το αντικείμενο είναι στιγμιότυπο της κλάσης **Optional**, ζητείται από το χρήστη το περιγραφικό όνομα που θα δώσει να είναι είτε μοναδικό είτε το όνομα της προγόνου έκδοσης (το οποίο αφού έχει ήδη γίνει δεκτό είναι και μοναδικό). Αυτό συμβαίνει γιατί με την άλγεβρα εκδόσεων δε μπορούμε να καθορίσουμε πότε θέλουμε να συμμετέχει στην κατασκευή μίας έκδοσης συστήματος ένα εξάρτημα (αντικείμενο) που έχει χαρακτηριοθεί προαιρετικό.

Αν όμως τα περιγραφικά ονόματα των εκδόσεων των μη υποχρεωτικών αντικειμένων είναι μοναδικά, η απουσία κάποιου από αυτά όταν ζητηθεί η κατασκευή μίας έκδοσης συστήματος θα δηλώνει ότι δεν πρέπει να επιλεγεί καμία έκδοση από το συγκεκριμένο αντικείμενο.

Η ανάκτηση πληροφορίας από τη βάση (π.χ. περιγραφικό όνομα της προγόνου έκδοσης, προαιρετικά αντικείμενα), τόσο στο υποσύστημα της διαχείρισης εκδόσεων, όσο και στα άλλα δύο που θα δούμε στις επόμενες ενότητες, γίνεται με χρήση των εντολών του PQI [1] (*Programmatic Query Interface*) [9].

6.1.2 Εξαγωγή Εκδόσεων

Με την εντολή της εξαγωγής δημιουργούμε ένα αντίγραφο μίας έκδοσης (ενός αντικειμένου) που θέλει να τροποποιήσει ο χρήστης. Ο καθορισμός συγκεκριμένης έκδοσης μπορεί να γίνει από το χρήστη με δύο τρόπους : είτε δίνοντας το περιγραφικό της όνομα είτε με τον αριθμό έκδοσης του RCS.

Στην πρώτη περίπτωση, είναι δυνατό να υπάρχουν περισσότερες από μία εκδόσεις, του ίδιου αντικειμένου, οι οποίες είναι διαδοχικές (συνδέονται με τη σχέση ***succ_of***) και έχουν το ίδιο περιγραφικό όνομα. Τότε, επιλέγεται από το σύστημα η πιο πρόσφατα δημιουργημένη έκδοση , γιατί η πιθανότητα να χρειάζεται ο χρήστης την τελευταία απόγονο έκδοση είναι πολύ μεγαλύτερη από την πιθανότητα να χρειάζεται οποιαδήποτε άλλη. Αν όμως, πραγματικά συμβεί αυτό, μπορεί να χρησιμοποιήσει το δεύτερο τρόπο καθορισμού έκδοσης.

Αφού καθοριοθεί η έκδοση, δίνονται τα δεδομένα στην τροποποιημένη εντολή `co` του RCS, όπου με τη χρήση της αναδημιουργείται η συγκεκριμένη έκδοση από δέλτα σε κανονική μορφή κειμένου και ο χρήστης μπορεί πλέον να κάνει τις αλλαγές που θέλει.

6.2 Εισαγωγή Δεδομένων (Περιορισμοί)

Με το υποσύστημα της εισαγωγής δεδομένων εισάγουμε ένα νέο περιορισμό στη βάση δεδομένων. Η εισαγωγή γίνεται έτσι ώστε να δημιουργείται μία ιεραρχία isA μεταξύ των περιορισμών, μέσω της οποίας κληρονομούνται και τα κοινά στοιχεία (εκδόσεις) μεταξύ τους, οπότε υπάρχει και οικονομία στον αριθμό των συνδέσμων της σχέσης ***concerns***. Στη συνέχεια περιγράφουμε αναλυτικά τον αλγόριθμο της εισαγωγής.

[1] Το PQI σχεδιάσθηκε και υλοποιήθηκε στο Ινστιτούτο Πληροφορικής του Ιδρύματος Τεχνολογίας και Ερευνας (I.T.E.) στα πλαίσια των έργων του ESPRIT ITHACA. Παρέχει τη δυνατότητα διασύνδεσης με την ITHACA Software Information Base (SIB) [13].

Εναν περιορισμό μπορούμε να τον θεωρήσουμε σαν ένα σύνολο, που τα στοιχεία του είναι εκδόσεις διαφορετικών αντικειμένων. Ετσι, έστω $S_1,\ S_2,\ ...\ S_m$ και $S_{m+1},\ ...\ S_n$ οι περιορισμοί που είναι στιγμιότυπα των κλάσεων **Constraint** και **SupConstraint**, αντίστοιχα, και έστω S ο νέος περιορισμός που θέλουμε να εισαγάγουμε.

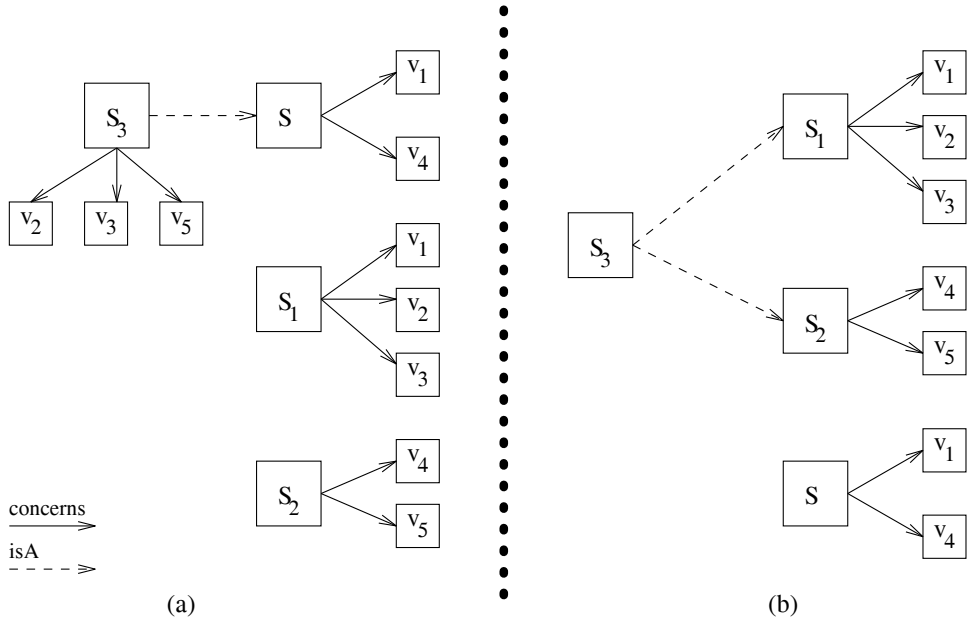
Περίπτωση Α : Ψάχνουμε μεταξύ των $S_i,\ i = 1...n$ για να δούμε αν υπάρχει ένα (ή περισσότερα), τέτοιο ώστε, το S_i να είναι υποσύνολο του S ($S_i \subseteq S$). Αν δεν υπάρχει πηγαίνουμε στην **Περίπτωση Β**.

- [**1**] Εστω λοιπόν, $\mathcal{S} = \{S_j : S_j$ περιορισμός και $S_j \subseteq S,\ j = 1...n\}$, δηλαδή, το $\mathcal{S}$ είναι το σύνολο των περιορισμών που κάθε στοιχείο του είναι υποσύνολο του S. Από αυτά, επιλέγουμε τον περιορισμό που έχει τα περισσότερα κοινά στοιχεία με το S, έστω το S_k.
- [**2**] Θέτουμε το S στιγμιότυπο της **SupConstraint**.
- [**3**] Ορίζουμε $S' = S - S_k$, δηλαδή, το S' είναι η διαφορά των δύο συνόλων (αποτελείται από τα στοιχεία του S που δεν ανήκουν στο S_k).
- [**4**] Αν ο πληθάριθμος του S' είναι μεγαλύτερος από 1 ψάχνουμε, πάλι, μεταξύ των $S_i,\ i = 1...n$ για να δούμε αν υπάρχει ένα (ή περισσότερα), τέτοιο ώστε $S_i \subseteq S'$. Αν βρεθεί, πηγαίνουμε πίσω στο βήμα [**1**], όπου τη θέση του S παίρνει το S'. Αν δεν βρεθεί, ή αν ο πληθάριθμος του S' είναι ίσος με 1, συνδέονται τα εναπομείναντα στοιχεία (εκδόσεις) με το S, μέσω της ***concerns***, και τέλος.

Περίπτωση Β : Ψάχνουμε μεταξύ των $S_i,\ i = 1...m$ (εξετάζουμε μόνο τα στιγμιότυπα της **Constraint**) για να δούμε αν υπάρχει ένα (ή περισσότερα), τέτοιο ώστε, το S να είναι υποσύνολο του S_i ($S \subseteq S_i$). Αν δεν υπάρχει, θέτουμε το S στιγμιότυπο της **Constraint**, το συνδέουμε με τα στοιχεία του με τη ***concerns*** και τέλος.

- [**1**] Εστω λοιπόν, $\mathcal{S} = \{S_j : S_j$ περιορισμός και $S \subseteq S_j,\ j = 1...m\}$, δηλαδή, το $\mathcal{S}$ είναι το σύνολο των περιορισμών που κάθε του στοιχείο είναι υπερσύνολο του S. Από τα στοιχεία του $\mathcal{S}$ επιλέγουμε αυτό με το μικρότερο πληθάριθμο, έστω το S_k.
- [**2**] Θέτουμε το S στιγμιότυπο της **Constraint** και το S_k της **SupConstraint**.
- [**3**] Ορίζουμε $S' = S_k - S$ και πηγαίνουμε στο βήμα [**4**] της **Περίπτωσης Α**, όπου τώρα το S_k αντιστοιχεί στο S.

Παρατηρούμε ότι στην **Περίπτωση Β** ψάχνουμε μόνο ανάμεσα στα στιγμιότυπα της **Constraint** και όχι και της **SupConstraint**. Αυτό συμβαίνει για τον εξής λόγο : Είναι πολύ πιθανό να υπάρχει κάποιο $S_l,\ l = m+1...n$, στιγμιότυπο της **SupConstraint**, για το οποίο θα ισχύει $S \subseteq S_l$. Αυτό όμως που δε μπορούμε να εξασφαλίσουμε είναι ότι καταστρέφοντας την ιεραρχία isA που υπάρχει μεταξύ του S_l και των περιορισμών με τους οποίους ήδη συνδέεται, για να συνδέσουμε το S_l με το S, θα έχουμε καλύτερα αποτελέσματα (που στην πραγματικότητα δεν έχουμε).



Σχήμα 6.2: Παράδειγμα εισαγωγής περιορισμών (χρησιμότητα διαχωρισμού σε **Constraint** και **SupConstraint**)

Για παράδειγμα, έστω οι περιορισμοί $S_1 = \{v_1,\ v_2\ v_3\}$, $S_2 = \{v_4,\ v_5\}$ και $S_3 = \{v_1,\ v_2\ v_3,\ v_4,\ v_5\}$. Οι S_1 και S_2 είναι στιγμιότυπα της **Constraint** και ο S_3 της **SupConstraint** και συνδέεται μέσω isA με τους S_1 και S_2. Θέλουμε να εισαγάγουμε ένα νέο περιορισμό τον $S = \{v_1,\ v_4\}$, που όπως βλέπουμε είναι υποσύνολο του S_3. Αν συνδέσουμε τον S_3, μέσω isA, με τον S και χαλάσουμε την υπάρχουσα ιεραρχία isA καταλήγουμε στη μορφή του σχήματος 6.2.a, ενώ αν εισαγάγουμε απλά τον S στη μορφή του σχήματος 6.2.b. Μεταξύ των δύο παρατηρούμε ότι στο (a) υπάρχουν 11 σύνδεσμοι (10 ***concerns*** και 1 isA) ενώ στο (b) μόνο 9 (7 ***concerns*** και 2 isA). Αρα, η επιλογή εισαγωγής του σχήματος (b) είναι προτιμότερη.

6.3 Αυτόματη Επιλογή

Με το υποσύστημα της αυτόματης επιλογής επιλέγονται, αυτόματα, εκδόσεις αντικειμένων, τα οποία είναι εξαρτήματα ενός συστήματος που θέλουμε να κατασκευάσουμε μία έκδοσή του. Επίσης, ελέγχεται η συνέπεια του συστήματος που προκύπτει, με έλεγχο της ύπαρξης ή όχι, περιορισμών μεταξύ των επιλεγμένων εκδόσεων.

Πρώτα απ' όλα όμως, πρέπει να ορίσουμε ποιοι περιορισμοί ισχύουν για μία έκδοση ενός αντικειμένου (στο προηγούμενο κεφάλαιο είχαμε μιλήσει για έμμεση "κληρονόμηση").
Εστω, V_i το σύνολο όλων των εκδόσεων του αντικειμένου O_i και έστω ότι η έκδοση v ανήκει στο V_i ($v \in V_i$). Τότε, ορίζεται το σύνολο $A_v \subseteq V_i$ ως εξής : $A_v = \{w \in V_i : v$ διασκευή της $w\}$, δηλαδή, είναι το σύνολο των εκδόσεων που είναι πρόγονοι της v. (Ο ορισμός της διασκευής (ή διασκευασμένης έκδοσης) έχει δοθεί στην παράγραφο 5.3.2). Αν τώρα θεωρήσουμε C το σύνολο όλων των περιορισμών που υπάρχουν, τότε, μπορούμε να ορίσουμε τα τέσσερα επόμενα σύνολα, υποσύνολα του C :

$$C_{A_v} = \{c \in C \ : \ \exists\ w \in A_v \text{ τέτοιο ώστε } c\ concerns\ w\}$$
$$C_v = \{c \in C \ : \ c\ concerns\ v\}$$
$$E_{A_v} = \{c \in C \ : \ \exists\ w \in A_v \text{ τέτοιο ώστε } w\ excepted\_from\ c\}$$
$$E_v = \{c \in C \ : \ v\ excepted\_from\ c\}$$

δηλαδή, το C_{A_v} είναι το σύνολο των περιορισμών που ισχύουν για όλες τις προγόνους της v, το C_v το σύνολο των περιορισμών που συνδέονται άμεσα με τη v μέσω της σχέσης ***concerns***, το E_{A_v} το σύνολο των περιορισμών που η ισχύς τους έχει ακυρωθεί από τις προγόνους της v και το E_v το σύνολο των περιορισμών που συνδέονται άμεσα με τη v μέσω της σχέσης ***excepted_from***. Ετσι, το σύνολο των περιορισμών που θα ισχύουν για τη v είναι το : $(C_{A_v} \cup C_v) - (E_{A_v} \cup E_v)$.

Τα βήματα που ακολουθούμε για την επιλογή των εκδόσεων που συμμετέχουν στην κατασκευή ενός συστήματος είναι τα παρακάτω :
[1] Το υποσύστημα ζητά από το χρήστη να καθορίσει το προϊόν (σύστημα) - του οποίου μία έκδοση θέλει ο χρήστης να κατασκευάσει - δίνοντας το όνομά του, π.χ. `CAR`. Ετσι, μέσω του μοντέλου συστήματος βρίσκουμε τα εξαρτήματα (αντικείμενα) C_i από τα οποία αποτελείται.
[2] Ζητείται από το χρήστη να καθορίσει, με χρήση των περιγραφικών ονομάτων και των τελεστών της άλγεβρας εκδόσεων, την έκδοση του προϊόντος που θέλει να κατασκευασθεί, έστω V.
[3] Με χρήση του γενικού κανόνα της άλγεβρας εκδόσεων (ορίσθηκε στην παράγραφο 5.1.2)

το υποσύστημα αναζητεί και επιλέγει την πλησιέστερη στη V έκδοση κάθε εξαρτήματος C_i, οπότε, για κάθε έκδοση επιλέγεται ένα (ή περισσότερα) περιγραφικά ονόματα.

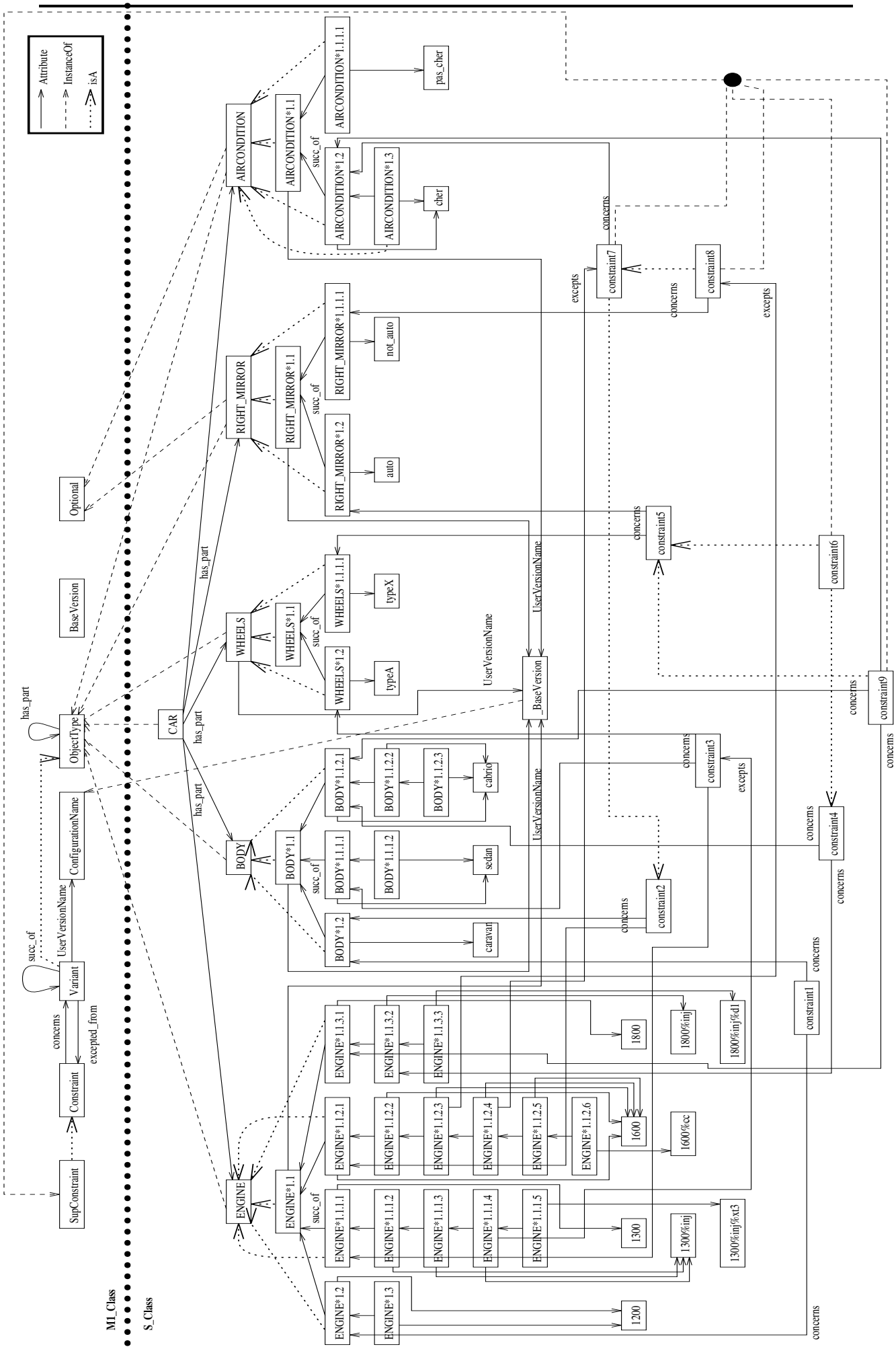
[4] Αξιολόγηση της έκδοσης του προϊόντος που προκύπτει.

- Αν για ένα εξάρτημα έχουν επιλεγεί περισσότερα από ένα περιγραφικά ονόματα, το σύστημα επιλογής δε μπορεί να αποφασίσει πιο από αυτά αποτελεί την καλύτερη εκλογή. Σε μία τέτοια περίπτωση, η παρούσα έκδοση του συστήματος, αποφασίζει ότι δε μπορεί να υπάρξει η έκδοση V του προϊόντος. Σε μία επόμενη έκδοση θα μπορούσε το σύστημα να αναπέμπει το ζήτημα στο χρήστη.

- Αντιστοιχίζεται το περιγραφικό όνομα κάθε έκδοσης με το όνομα συστήματος και ελέγχεται η ύπαρξη περιορισμών μεταξύ των επιλεγμένων εκδόσεων. Ο έλεγχος γίνεται ως εξής : έστω S το σύνολο των εκδόσεων που έχουν επιλεγεί για την κατασκευή της έκδοσης V του προϊόντος και $S_i,\ i = 1...n$, οι περιορισμοί που ισχύουν. Για κάθε i ελέγχουμε αν $S_i \subseteq S$. Αν ισχύει, τότε, δε μπορεί να υπάρξει η έκδοση V του προϊόντος.

6.4 Δοκιμές του Συστήματος

Για να ελέγξουμε την ορθή λειτουργία του συστήματος κάναμε τα εξής : ορίσαμε το σύστημα `CAR`, το οποίο αποτελείται από τα εξαρτήματα `ENGINE, BODY, WHEELS, RIGHT_MIRROR` και `AIRCONDITION`. Για κάθε ένα από αυτά δημιουργήσαμε παραλλαγές (*variants*) και διασκευές (*revisions*) χρησιμοποιώντας το υποσύστημα Διαχείρισης Εκδόσεων. Με τη βοήθεια του υποσυστήματος Εισαγωγής Δεδομένων εισαγάγαμε στη βάση δεδομένων περιορισμούς μεταξύ εκδόσεων διαφορετικών αντικειμένων. Τέλος, με χρήση του υποσυστήματος Επιλογής Εκδόσεων, ζητήσαμε την κατασκευή διαφόρων εκδόσεων του συστήματος `CAR`, μερικών με προαιρετικά αντικείμενα και άλλων χωρίς, επιλέγοντας από τις εκδόσεις των εξαρτημάτων (π.χ. έκδοση συστήματος : `1300%inj + sedan + typeX + auto`, έκδοση συστήματος : `1600%cc + caravan + typeA`, κ.λπ. , βλέπε σχήμα 6.3). Η επιλογή ολοκληρώθηκε με επιτυχία για ένα σύνολο εκδόσεων και ανεπιτυχώς για τις υπόλοιπες, λόγω ύπαρξης περιορισμών. Όλες οι δοκιμές που κάναμε, και για τα τρία υποσυστήματα, ολοκληρώθηκαν επιτυχώς. Στο σχήμα 6.3 παρουσιάζεται, γραφικά, το παράδειγμα που υλοποιήσαμε για τον έλεγχο του συστήματος.


Attribute
InstanceOf
isA
M1_Class
S_Class
SupConstraint
Constraint
concerns
excepted_from
Variant
succ_of
UserVersionName
ConfigurationName
has_part
ObjectType
Optional
BaseVersion
CAR
has_part
has_part
has_part
ENGINE
ENGINE*1.1
succ_of
ENGINE*1.1.1.1
ENGINE*1.1.1.2
ENGINE*1.1.1.3
ENGINE*1.1.1.4
ENGINE*1.1.1.5
ENGINE*1.2
ENGINE*1.3
ENGINE*1.1.2.1
ENGINE*1.1.2.2
ENGINE*1.1.2.3
ENGINE*1.1.2.4
ENGINE*1.1.2.5
ENGINE*1.1.2.6
ENGINE*1.1.3.1
ENGINE*1.1.3.2
ENGINE*1.1.3.3
1200
1300
1600
1800
1300%inj
1600%cc
1800%inj
1300%inj%xt3
1800%inj%dl
BODY
BODY*1.1
succ_of
BODY*1.1.1.1
BODY*1.1.1.2
sedan
BODY*1.2
caravan
BODY*1.1.2.1
BODY*1.1.2.2
BODY*1.1.2.3
cabrio
WHEELS
WHEELS*1.1
succ_of
WHEELS*1.1.1.1
typeX
WHEELS*1.2
typeA
RIGHT_MIRROR
RIGHT_MIRROR*1.1
succ_of
RIGHT_MIRROR*1.1.1.1
not_auto
RIGHT_MIRROR*1.2
auto
AIRCONDITION
AIRCONDITION*1.1
succ_of
AIRCONDITION*1.1.1.1
pas_cher
AIRCONDITION*1.2
AIRCONDITION*1.3
cher
UserVersionName
UserVersionName
UserVersionName
_BaseVersion
constraint1
constraint2
constraint3
constraint4
constraint5
constraint6
constraint7
constraint8
constraint9
concerns
excepts

Σχήμα 6.3: Σχηματική παράσταση του μοντέλου για configuration management και των δεδομένων που εισαγάγαμε στη βάση δεδομένων προκειμένου να χρησιμοποιηθούν στις δοκιμές του συστήματος

Για λόγους απλότητας του σχήματος παραλείψαμε τους συνδέσμους instanceOf μεταξύ όλων των εκδόσεων (`ENGINE*...`, `BODY*...`, `WHEELS*...`, `RIGHT_MIRROR*...`, `AIRCONDITION*...`) και της κλάσης **Variant**, μεταξύ των εκδόσεων βάσης (`ENGINE*1.1`, `BODY*1.1`, `WHEELS*1.1`, `RIGHT_MIRROR*1.1`, `AIRCONDITION*1.1`) και της **BaseVersion**, μεταξύ όλων των περιγραφικών ονομάτων (εξαιρουμένου του `_BaseVersion`) και της **ConfigurationName** και μεταξύ των περιορισμών `constraint1`, `constraint2`, `constraint3`, `constraint4`, `constraint5` και της κλάσης **Constraint**. Επίσης παραλείψαμε τους περισσότερους από τους συνδέσμους isA μεταξύ των εκδόσεων και των αντικειμένων που αντιστοιχούν (π.χ. μεταξύ της `ENGINE*1.3` και του `ENGINE`, της `BODY*1.1.2.3` και του `BODY`, κ.λπ.).

Κεφάλαιο 7

Συμπεράσματα

Το αντικείμενο αυτής της εργασίας ήταν η σχεδίαση και η υλοποίηση ενός συστήματος για configuration management. Το σύστημά μας στηρίζεται σε μία άλγεβρα εκδόσεων (όπως έχει οριοθεί στο [42]) και στο μοντέλο για CM που σχεδιάσαμε και περιγράψαμε στη γλώσσα παράστασης γνώσης TELOS, με το οποίο εισάγουμε τις έννοιες των περιορισμών ασυμβατότητας και των προαιρετικών αντικειμένων. Αποτελείται (το σύστημα) από τρία τμήματα - Διαχείριση Εκδόσεων, Εισαγωγή Δεδομένων, Αυτόματη Επιλογή - από τα οποία κάθε ένα επιτελεί τις αντίστοιχες λειτουργίες. Στη συνέχεια προτείνουμε διάφορες βελτιώσεις και επεκτάσεις του συστήματος.

Μία σημαντική βελτίωση του συστήματος, πιστεύουμε ότι θα αποτελούσε η μετατροπή του συστήματος διασύνδεσης με το χρήστη από το επίπεδο των εντολών γραμμής σε γραφικό επίπεδο. Ετσι ο χρήστης, χρησιμοποιώντας το "ποντίκι", θα μπορούσε να επιλέξει για εξαγωγή μία έκδοση ενός εξαρτήματος από τη γραφική παράσταση του δένδρου εκδόσεων του συγκεκριμένου εξαρτήματος (τμήμα Διαχείρισης Εκδόσεων), θα μπορούσε να καθορίσει τις εκδόσεις που αφορούν έναν περιορισμό και τις εκδόσεις που εξαιρούνται από έναν περιορισμό μέσω μίας γραφικής αναπαράστασης των εκδόσεων των αντικειμένων και της isA ιεράρχησης των περιορισμών (τμήμα Εισαγωγής Δεδομένων) και τέλος θα βοηθούσε, κατά την επιλογή μίας έκδοσης προϊόντος, η γραφική παράσταση των παραλλαγών (*variants*), κάθε εξαρτήματος που μπορεί να συμμετάσχει στην κατασκευή του προϊόντος (τμήμα Αυτόματης Επιλογής).

Μία πιθανή επέκταση του συστήματος αφορά στην υποστήριξη διαμορφώσεων για κατανεμημένα συστήματα, για παράδειγμα, ύπαρξη των εξαρτημάτων ενός προϊόντος σε κατανεμημένο σύστημα και δυνατότητα αυτόματης επιλογής έκδοσης εξαρτήματος για την

κατασκευή μιας έκδοσης προϊόντος. Επίσης, για τη διαχείριση εκδόσεων, μία εντολή συγχώνευσης (*merge*) είναι σημαντική για το συνδυασμό παράλληλων γραμμών ανάπτυξης. Κατά τη συγχώνευση, πρώτα αναγνωρίζονται τα κοινά σημεία μεταξύ μίας έκδοσης βάσης και δύο παράλληλων διασκευασμένων εκδόσεων, και στη συνέχεια ενοποιούνται οι αλλαγές. Επιπλέον, ανακαλύπτονται οι συγκρουόμενες αλλαγές. Τέλος, στο παρόν σύστημα με την επαναλαμβανόμενη χρήση των τελεστών + και % είναι πιθανό να προκύψουν υπερβολικά μεγάλα ονόματα εκδόσεων, γεγονός που δυσχεραίνει το χρήστη. Μία λύση είναι να επιτραπούν μεταβλητές έκδοσης, δηλαδή, να εισαχθούν νέες εκδόσεις, οι οποίες περιέχουν εκφράσεις των υπαρχόντων ονομάτων εκδόσεων.

Παράρτημα Α

Περιγραφή σε TELOS του Μοντέλου CM

Στο κεφάλαιο αυτό του παραρτήματος δίνεται η περιγραφή του μοντέλου για configuration management με τη γλώσσα παράστασης γνώσης TELOS. Το μοντέλο παρουσιάστηκε αναλυτικά στο 5ο κεφάλαιο της παρούσας εργασίας.

```
BEGINTRANSACTION

TELL Individual ConfigurationName in M1_Class
end ConfigurationName

TELL Individual Optional in M1_Class
end Optional

TELL Individual Object in M1_Class with
     attribute
          has_part :  Object
end Component

TELL Individual Variant in M1_Class isA Object with
     attribute
          userVersionName :  ConfigurationName;
          succ_of :  Variant;
          excepted_from :  Constraint
end Variant
```

```
TELL Individual BaseVersion in M1_Class isA Variant
end BaseVersion

TELL Individual Constraint in M1_Class with
     attribute
          concerns :  Variant
end Constraint

TELL Individual SupConstraint in M1_Class isA Constraint
end SupConstraint

ENDTRANSACTION
```

Παράρτημα Β

Εντολές Διαχείρισης Εκδόσεων

Εισαγωγή Εκδόσεων

Με την εντολή εισαγωγής εκδόσεων (`imp`) αποθηκεύεται μία έκδοση ενός αρχείου στην αποθήκη, σε δέλτα μορφή και το αρχείο διαγράφεται από το directory στο οποίο βρίσκεται. Η σύνταξη της εντολής είναι η παρακάτω :

```
imp -{lu} -mmsg -t[textfile] -w[login] filename
```

- `imp filename` : Εισαγάγει μία έκδοση του αρχείου, που επεξεργάζεται ο χρήστης, στην αποθήκη.

- `imp -l filename` : Εισαγάγει μία έκδοση χωρίς να αφήσει το "κλείδωμα" ή να οβήσει το αρχείο που επεξεργάζεται ο χρήστης.

- `imp -u filename` : Εισαγάγει μία έκδοση και αφήνει το "κλείδωμα" χωρίς να οβήσει το αρχείο που επεξεργάζεται ο χρήστης.

- `imp -m"msg" filename` : Εισαγάγει μία νέα έκδοση του αρχείου με όνομα `filename`, αποθηκεύοντας ως μήνυμα περιγραφής των αλλαγών που έγιναν στην παρούσα έκδοση το "`msg`". Η ύπαρξη του μηνύματος είναι υποχρεωτική για κάθε νέα έκδοση, πλην της πρώτης.

- `imp -ttextfile filename` : Εισαγάγει μία νέα έκδοση και αντιγράφει το κείμενο που περιέχεται στο αρχείο `textfile` στην αποθήκη, στο αρχείο RCS/filename,v. Η ύπαρξη του είναι υποχρεωτική μόνο κατά την εισαγωγή της πρώτης έκδοσης του συγκεκριμένου αρχείου.

- `imp -wuser filename` : Εισαγάγει το αρχείο και επιτρέπει την εκχώρηση δικαιοδοσίας στην νέα έκδοση και σε κάποιον άλλο χρήστη.

Εξαγωγή Εκδόσεων

Με την εντολή εξαγωγής εκδόσεων (`exp`) δημιουργείται ένα αντίγραφο μίας έκδοσης, του αρχείου που θέλει να τροποποιήσει ο χρήστης. Η σύνταξη της εντολής είναι η παρακάτω :

```
exp [-vconfname || -rrev] -{lp}[rev] -ddate -w[login] filename
```

- `exp -vconfname filename` : Εξάγει την έκδοση, με περιγραφικό όνομα `confname`, του αρχείου με όνομα `filename`.
- `exp -rrev filename` : Ξαναδημιουργεί το αρχείο με όνομα `filename` επιλέγοντας την `rev` έκδοση αυτού. (Η παρούσα μορφή της εντολής είναι εναλλακτική της προηγούμενης.)
- `exp -l filename` : "Κλειδώνει" το RCS αρχείο και εξάγει μία έκδοσή του, έτσι ώστε να μπορεί να τροποποιηθεί από το χρήστη (*writable*).
- `exp -p filename` : Δείχνει τα περιεχόμενα μίας έκδοσης του αρχείου με όνομα `filename`.
- `exp -ddate filename` : Εξάγει την έκδοση του αρχείου με όνομα `filename` που έχει την πιο κοντινή ημερομηνία δημιουργίας με την ημερομηνία `date`, αλλά είναι πριν από αυτή. (π.χ. "1-sept-90").
- `exp -wuser filename` : Επιλέγει την τελευταία έκδοση του αρχείου `filename` που έχει δημιουργηθεί από το χρήστη `user`.

Βιβλιογραφία - Παραπομπές

[1] **Vincenzo Ambriola, Lars Bendix and Paolo Ciancarini**, The Evolution of Configuration Management and Version Control, *Software Engineering Journal, 5 (6), pp. 303-310, Nov. 1990*

[2] **Wayne A. Babich**, Software Configuration Management, *Reading, MA : Addison-Wesley Publishing Co., 1986*

[3] **Edward H. Bersoff, Vilas D. Henderson, Stan G. Siegel**, Software Configuration Management : A Tutorial, *IEEE Computer, 12 (1), pp. 6-14, Jan. 1979*

[4] **Edward H. Bersoff**, Elements of Software Configuration Management, *IEEE Transactions on Software Engineering, SE-10 (1), pp. ,Jan. 1984*

[5] **Marian Bliss**, Software Configuration Management : Delivering Quality Software Products, *Information Systems Management, 10 (1), pp. 35-46, Summer 1993*

[6] **Robert P. Chase Jr.**, System Modelling, Summary of Plenary Discussion, *Proceedings of the International Workshop on Software Version and Configuration Control, Teubner Verlag, pp. 172-174, Stuttgart, Jan. 1988 (also in ACM SIGSOFT Software Engineering Notes, 13 (4), pp. 61-73, Oct. 1988)*

[7] **Ellis S. Cohen, Dilip A. Soni, Raimund Gluecker, William M. Hasling, Robert W. Schwanke, Michael E. Wagner**, Version Management in Gypsy, *ACM SIGPLAN Notices, 24 (2) pp. 201-215, Feb. 1989*

[8] **W. Courington**, The Network Software Environment, *Sun Technical Report FE197-0, Sun Microsystems Inc., Feb. 1989*

[9] **Costas Dadouris, Martin Dőrr**, The Programmatic Query Interface for the SIS, Programmer's Manual, *Institute of Computer Science, Foundation of Research and Technology-Hellas, Dec. 1993*

[10] **Susan Dart**, Spectrum of Functionality in Configuration Management Systems, *Technical Report, CMU/SEI-90-TR-11, ESD-90-TR-212, Software Engineering Institute, Carnegie Mellon University, Dec. 1990*

[11] **Daniel Deitz**, Pulling the Data Together, *Mechanical Engineering, Feb. 1990*

[12] **Klaus Dittrich, Willi Gotthard and Peter C. Lockemann**, DAMOKLES. A Database System for Software Engineering Environments, *International Workshop on Advanced Programming Environments, pp. 351-371, Springer Verlag, Trondheim Norway, Jun. 1986*

[13] **M. Dőrr, M. Theodoridou P. Klimathianakis**, SIB Data Entry Language Users's Manual, *ITHACA.FORTH.92.E2.10, Institute of Computer Science, Foundation of Research and Technology-Hellas, Dec. 1992*

[14] **Martin Dőrr, Polyvios Klimathianakis, Manos Theodorakis**, SIS DataEntry Language User's Manual, *SIS Static Analyser, Runtime System, Version 1.2, Institute of Computer Science, Foundation of Research and Technology-Hellas, Oct. 1994*

[15] **Brian O' Donovan**, RCS Handbook, *1992*

[16] **Marc Downson**, Integrated Project Support with ISTAR, *IEEE Software, 4 (6), pp. 6-15, Nov. 1987*

[17] **J. Estublier**, A Configuration Manager : The Adele Data Base of Programs, *Proceedings of the Workshop on Software Engineering Environments for Programming in the Large, pp. 140-147, Harwhichport Massuchussets, Jun. 1985*

[18] **Jacky Estublier**, Configuration Management. The Notion and the Tools, *Proceedings of the International Workshop on Software Version and Configuration Control, Teubner Verlag, pp. 38-61, Stuttgart, Jan. 1988*

[19] **Peter H. Feiler, Susan A. Dart, Grace Downey**, Evaluation of the Rational Environment, *Technical Report, CMU/SEI-88-TR-15, ADA 198934, Software Engineering Institute, Carnegie Mellon University, Jul. 1988*

[20] **Stuart I. Feldman**, Make - A Program for Maintaining Computer Programs, *Software - Practice and Experience, 9 (3), pp. 255-265, Mar. 1979*

[21] **Hassan Gomaa**, A reuse-oriented approach for structuring and configuring distributed applications, *Software Engineering Journal, 8 (2), pp. 61-71, Mar. 1993*

[22] **Marc Graham and Dan Miller**, ISTAR Evaluation, *Technical Report, CMU/SEI-88-TR-3, ADA201345, Software Engineering Institute, Carnegie Mellon University, Jul. 1988*

[23] **A. Nico Habermann and David Notkin**, Gandalf : Software Development Environments, *IEEE Transactions on Software Emgineering, SE-12 (12), pp. 1117-1127, Dec.1986*

[24] **IEEE Std. 729 - 1983**, Standard Glossary for Software Engineering Terminology, *New York : IEEE, 1983*

[25] **Randy H. Katz and Tobin J. Lehman**, Database Support for Versions and Alternatives of Large Design Files, *IEEE Transactions on Software Emgineering, SE-10 (2), pp. 191-200, Mar. 1984*

[26] **R. H. Katz and E. Chang**, Managing Change in a Computer-Aided Design Database, *Computer Science Division, Electrical Engineering and Computer Science Department, University of California, Berkeley*

[27] **Randy H. Katz**, Toward a Unified Framework for Version Modeling in Engineering Databases, *ACM Computing Surveys, 22 (4), pp. 375-408, Dec. 1990*

[28] **Jeff Kramer and Jeff Magee**, Dynamic Configuration for Distributed Systems, *IEEE Transactions on Software Engineering, SE-11 (4), pp. 424-436, Apr. 1985*

[29] **J. Kramer, J. Magee, M. Sloman**, Configuration Support for System Description, Construction and Evolution *ACM SIGSOFT Software Engineering Notes, 14 (3), pp. 28-33, Mar. 1989*

[30] **Jeff Kramer, Jeff Magee, Morris Sloman and Naranker Dulay**, Configuring object-based distributed programs in REX, *Software Engineering Journal, 7 (2), pp. 139-149, Mar. 1992*

[31] **Andreas Lampen and Axel Mahler**, An Object Base for Attributed Software Objects, *Proceedings of the Fall 1988 EUUG Conference, European Unix systems User Group, Lisbon Portugal, Oct. 1988*

[32] **David B. Leblang and Robert P. Chase Jr.**, Configuration Management for Large Scale Software Development Efforts, *Proceedings of the Workshop on Software Engineering Environments for Programming in the Large, pp. 122-127, Harwhichport Massuchussets, Jun. 1985*

[33] **David B. Leblang, Robert P. Chase Jr. and Howard Spilke**, Increasing Productivity with a Parallel Configuration Manager, *Proceedings of the International Workshop on Software Version and Configuration Control, Teubner Verlag, pp. 21-37, Stuttgart, Jan. 1988*

[34] **Axel Mahler and Andreas Lampen**, shape - A Software Configuration Management Tool, *Proceedings of the International Workshop on Software Version and Configuration Control, Teubner Verlag, pp. 228-243, Stuttgart, Jan. 1988*

[35] **Axel Mahler and Andreas Lampen**, An Integrated Toolset for Engineering Software Configurations, *ACM SIGPLAN Notices, 24 (2), pp. 191-200, Feb. 1989*

[36] **Axel Mahler**, Using the Shape Toolkit for Cooperative Software Development - A Tutorial, *shapeTools_TUT_1.3, Techische Universitat Berlin, Jul. 1993*

[37] **Axel Mahler and Andreas Lampen**, Integrating Configuration Management into a Generic Environment, *ACM SIGSOFT Software Engineering Notes, 15 (6), pp. 229-237, 1990*

[38] **K. Marzullo and D. Wiebe**, Jasmine : A Software System Modelling Facility, *ACM SIGPLAN Notices, 22 (1), pp. 121-130, Jan. 1987*

[39] **R. Medina-Mora, D. Notkin and R. Ellison**, Aloe users' and implementors' guide, *Second Compendium of Gandalf Documentation, Dep. Comp. Sc., Carnegie Mellon University, May 1982*

[40] **Terence C. Miller**, A Schema for Configuration Management, *Proceedings of the 2nd International Workshop on Software Configuration Management, pp. 26-29, Princeton New Jersey, Oct. 1989 (also in ACM SIGSOFT Software Engineering Notes, 14 (7), Nov. 1989)*

[41] **John Mylopoulos, Alex Borgida, Matthias Jarke and Manolis Koubarakis**, Telos : Representing Knowledge About Information Systems, *ACM Transactions on Information Systems, 8 (4), pp. 325-362, Oct. 1990*

[42] **John Plaice and William W. Wadge**, A New Approach to Version Control, *IEEE Transactions on Software Engineering, 19 (3), pp. 268-275, Mar. 1993*

[43] **Erhard Ploederder, Adel Fergany**, The Data Model of the Configuration Management Assistant, *Proceedings of the 2nd International Workshop on Software Configuration Management, pp. 5-14, Princeton New Jersey, Oct. 1989 (also in ACM SIGSOFT Software Engineering Notes, 14 (7), Nov. 1989)*

[44] **Mark J. Rochkind**, The Source Code Control System, *IEEE Transactions on Softaware Engineering, SE-1 (4), pp. 364-370, Dec. 1975*

[45] **Takumei So**, Configuration Management Systems, *CERC Technical Report, CERC-TR-RN-92-001, West Virginia University, Apr. 1992*

[46] **Takumei So, V. Jagannathan, Ravi S. Raman**, The Role of Configuration Management Systems in a Concurrent Engineering Environment, *CERC Technical Report, CERC-TR-TM-92-15, West Virginia University, Nov. 1992*

[47] **Kristopher G. Sprague**, The Role of Software Configuration Management in a Measurment-Based Software Engineering Program, *ACM SIGSOFT, 16 (2), pp. 62-66, Apr. 1991*

[48] **Daniel C. Swinehart, Polle T. Zellweger, Richard J. Beach and Robert B. Hagman**, A Structural View of the Cedar Programming Environment, *ACM Transactions on Programming Languages and Systems, 8 (4), pp. 419-490, Oct. 1986*

[49] **Warren Teitelman**, A Tour Through Cedar, *IEEE Software, 1 (4), pp. 44-73, Apr. 1984*

[50] **I. Thomas**, Mosaix, *Proceedings of the Workshop on Software Engineering Environments for Programming in the Large, Harwhichport Massuchussets, Jun. 1985*

[51] **Walter F. Tichy**, RCS - A System for Version Control, *Software - Practice and Experience, 15 (7), pp. 637-654, Jul. 1985*

[52] **Walter F. Tichy**, Tools for Software Configuration Management, *Proceedings of the International Workshop on Software Version and Configuration Control, Teubner Verlag, pp. 1-20, Stuttgart, Jan. 1988*

[53] **Walter F. Tichy**, What is a Configuration, Summary of Plenary Discusion, *Proceedings of the International Workshop on Software Version and Configuration Control, Teubner Verlag, pp. 169-171, Stuttgart, Jan. 1988 (also in ACM SIGSOFT Software Engineering Notes, 13 (4), pp. 61-73, Oct. 1988)*

[54] Aide - De - Camp Software Management System, Product Overview, *Concord MA, 1989*

[55] CCC : Change and Configuration Control Environment. A Functional Overview, *Softool 1987*

Αλλη Σχετική Βιβλιογραφία

[*] **Vincenzo Ambriola, Lars Bendix**, Object-Oriented Configuration Control, *Proceedings of the 2nd International Workshop on Software Configuration Management, pp. 133-136, Princeton New Jersey, Oct. 1989 (also in ACM SIGSOFT Software Engineering Notes, 14 (7), Nov. 1989)*

[*] **Fletcher J. Buckley**, Implementing a Software Configuration Management Environment, *IEEE Computer, 27 (2), pp. 56-61, Feb. 1994*

[*] **Eduardo Casais**, Managing Class Evolution in Object-Oriented Systems, *Object Management (Gestion d' Objets), edit. D. Tsichritzis, Universite de Geneve, Jul. 1990*

[*] **Klaus R. Dittrich and Raymond A. Lorie**, Version Support for Engineering Database Systems, *IEEE Transactions on Software Engineering, 14 (4), pp. 429-437, Apr. 1988*

[*] **Peter H. Feiler, Roger Smeaton**, Managing Development of Very Large Systems : Implications on Integrated Environments, *Proceedings of the International Workshop on Software Version and Configuration Control, Teubner Verlag, pp. 62-82, Stuttgart, Jan. 1988*

[*] **S. Gibbs, Dennis Tsichritzis, Eduardo Casais, Oscar Nierstrasz and Xavier Pintado**, Class Management for Software Communities, *Communications of the ACM, 33 (9), pp. 90-103, Sep. 1990*

[*] **Bjorn Gulla, Even-Andre Karlsson and Dashing Yeh**, Change Oriented Version Descriptions in EPOS, *Software Engineering Journal, 6 (6), pp. 378-386, Nov. 1991*

[*] **Dennis Heimbigner, Steven Krane**, A Graph Transform Model for Configuration Management Environments, *ACM SIGPLAN Notices, 24 (2), pp. 216-225, Feb. 1989*

[*] **Susan Horwitz, Jan Prins and Thomas Reps**, Integrating Noninterfering Versions of Programs, *ACM Transactions on Programming Languages and Systems, 11 (3), pp. 345-387, Jul. 1989*

[*] **Susan Horwitz**, Identifying the Semantic and Textual Differences Between Two Versions of a Program, *Proceedings of the ACM SIGPLAN '90 Conference on Programming Language Design and Implementation, White Plains, New York, pp. 234-245, June 20-22, 1990*

[*] **Hans-Ulrich Kobialka** Configuration Editing, Generation and Test within Working Contexts, *ACM SIGSOFT Software Engineering Notes, 15 (6), pp. 173-182, 1990*

[*] **Anund Lie, Reidar Conradi, Tor M. Didriksen, Even-Andre Karlsson, Svein O. Hallsteinsen, Per Holager**, Change Oriented Versioning in a Software Engineering Database, *Proceedings of the 2nd International Workshop on Software Configuration Management, pp. 56-65, Princeton New Jersey, Oct. 1989 (also in ACM SIGSOFT Software Engineering Notes, 14 (7), Nov. 1989)*

[*] **Boris Magnusson, Ulf Asklund, Sten Minor** Fine-Grained Revision Control for Collaborative Software Development, *ACM SIGSOFT Software Engineering Notes, 18 (5), pp. 33-41, 1993*

[*] **David B. Miller, Robert G. Stockton, Charles W. Krueger**, An Inverted Approach to Configuration Management, *Proceedings of the 2nd International Workshop on Software Configuration Management, pp. 1-4, Princeton New Jersey, Oct. 1989 (also in ACM SIGSOFT Software Engineering Notes, 14 (7), Nov. 1989)*

[*] **Mark Moriconi**, A Practical Approach to Semantic Configuration Management, *ACM SIGSOFT Software Engineering Notes, 14 (7), pp. 103-113, 1989*

[*] **K. Narayanaswamy**, A Text-Based Representation for Program Variants, *Proceedings of the 2nd International Workshop on Software Configuration Management, pp. 30-33, Princeton New Jersey, Oct. 1989 (also in ACM SIGSOFT Software Engineering Notes, 14 (7), Nov. 1989)*

[*] **Christoph Reichberger**, Orthogonal Version Management, *Proceedings of the 2nd International Workshop on Software Configuration Management, pp. 137-140, Princeton*

New Jersey, Oct. 1989 (also in ACM SIGSOFT Software Engineering Notes, 14 (7), Nov. 1989)

[*] **Thomas Rose, Matthias Jarke, Michael Gocek, Carlos Maltzahn and Hans Nissen** , A Decision-Based Configuration Process Environment, *Software Engineering Journal, 6 (5), pp. 332-346, Sep. 1991*

[*] **Dave Thomas and Kent Johnson**, Orwell : A Configuration Management System for Team Programming, *ACM SIGPLAN Notices, 23 (11), pp. 135-141, Nov. 1988 (Proceedings of OOPSLA '88, Sept 1988)*