Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων
και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης
και Υπολογιστων

Αλγεβρική Προδιαγραφή Προτύπων

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Κωνσταντίνος Δ. Μπάρλας

Αθήνα, Μάρτιος 2018

Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων
και Μηχανικων Υπολογιστων

Τομεας Τεχνολογιας Πληροφορικης
και Υπολογιστων

Αλγεβρική Προδιαγραφή Προτύπων

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Κωνσταντίνος Δ. Μπάρλας

Συμβουλευτική Επιτροπή : Γεώργιος Κολέτσος (Επιβλέπων Καθηγητής)
Πέτρος Στεφανέας
Ελένη Μπέρκη

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την 15 Μαρτίου 2018

.....	.....	.....
Α. Σταφυλοπάτης	Π. Φράγκος	Γ. Στάμου
Καθηγητής	Καθηγητής	Αν. Καθηγητής
ΣΗΜΜΥ, Ε.Μ.Π.	ΣΗΜΜΥ, Ε.Μ.Π.	ΣΗΜΜΥ, Ε.Μ.Π.
.....	.....	.....
Π. Καβάσαλης	Π. Στεφανέας	Α. Αρβανιτάκης
Αν. Καθηγητής	Επίκουρος Καθηγητής	Επίκουρος Καθηγητής
Παν. Αιγαίου	ΣΕΜΦΕ, Ε.Μ.Π.	ΣΕΜΦΕ, Ε.Μ.Π.

.....
Ε. Μπέρκη
Επίκουρη Καθηγήτρια
University of Jyväskylä

Αθήνα, Μάρτιος 2018

.....
Κωνσταντίνος Δ. Μπάρλας

Υποψήφιος Διδάκτωρ της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Ε.Μ.Π.

Διπλωματούχος της Σχολής Εφαρμοσμένων Μαθηματικών και Φυσικών Επιστημών, Ε.Μ.Π.

Περίληψη

Στη διδακτορική αυτή διατριβή θα εξετάσουμε την ιδέα της χρήσης Τυπικών Μεθόδων (και συγκεκριμένα Αλγεβρικών Προδιαγραφών) για τη δημιουργία τυπικών προδιαγραφών προτύπων για να συμπληρώσουν ή να αντικαταστήσουν τις προδιαγραφές που συνοδεύουν ένα πρότυπο και είναι γραμμένες σε κάποια φυσική γλώσσα, σε μια προσπάθεια να ελαχιστοποιηθούν τα λάθη κατανόησης τέτοιων κειμένων που οφείλονται κυρίως στην ασάφεια που εμπεριέχουν οι φυσικές γλώσσες. Οι προδιαγραφές αυτές μπορεί να είναι εκτελέσιμες, δίνοντάς μας έτσι τη δυνατότητα να περιγράψουμε και να επαληθεύσουμε την ισχύ κάποιων απαιτούμενων ιδιοτήτων του προτύπου.

Οι Τυπικές Μέθοδοι είναι τεχνικές που χρησιμοποιούνται για την μοντελοποίηση (και επαλήθευση) περίπλοκων συστημάτων σε μορφή μαθηματικών οντοτήτων. Η κατασκευή ενός τέτοιου αυστηρού μαθηματικού μοντέλου επιτρέπει την επαλήθευση ιδιοτήτων του συστήματος με τρόπο πολύ πιο αναλυτικό και ενδελεχή από τον εμπειρικό έλεγχο. Τα πλεονεκτήματα της χρήσης Τυπικών Μεθόδων φαίνεται να αντισταθμίζονται κάπως από το υψηλό κόστος εφαρμογής τους, κάτι που κάνει τον ρυθμό αποδοχής τους από τη βιομηχανία πολύ αργό.

Ισχυριζόμαστε πως η χρήση των τυπικών μεθόδων σε βιομηχανικά πρότυπα βοηθάει τόσο τη βιομηχανία, αφού κάνει τις προδιαγραφές των προτύπων πιο σαφείς και περιεκτικές, όσο και την κοινότητα των τυπικών Μεθόδων.

Λέξεις Κλειδιά

Τυπικές Μέθοδοι, Τυπικές Προδιαγραφές, Αλγεβρικές Προδιαγραφές, (Ανοικτά) Πρότυπα, φυσικές γλώσσες, CafeOBJ, Abstract Syntax Notation One, Open Document Architecture, Rich Site Summary, Κοινωνικά Δίκτυα, Εκπαίδευση Τυπικών Μεθόδων.

Abstract

Open standardization seems to be very popular among software developers as it simplifies the standard's adoption by the software engineering. Formal specification methods, while very promising, are being adopted very slowly as the industry seems to have little motivation to move into this territory. In this thesis we will present the idea of applying formal specification techniques to (open) standards' specifications.

We provide evidence for the advantages of the open standards formal specification over natural language documentations: Formal specifications are more concise, less ambiguous, more complete with respect to the original documentation and, when using certain kinds of specification languages, executable and reusable as they support module inheritance. The merging of formal specification methods and open standards allows i) a more concrete standard design; ii) an improved understanding of the environment under design; iii) an enforced certain level of precision into the specification, and also iv) provides software engineers with extended property checking/verification capabilities, especially if they opt to use any algebraic specification language.

Keywords

Formal Methods, Formal Specifications, Algebraic Specifications, (Open) Standards, Natural Languages, CafeOBJ, Abstract Syntax Notation One, Open Document Architecture, Rich Site Summary, Social Networks, Formal Methods in Education.

Ευχαριστίες

Αφιερώνεται στην οικογένεια μου...

Η παρούσα διατριβή αν και είναι το αποτέλεσμα προσωπικού αγώνα, δε θα μπορούσε να είχε ολοκληρωθεί χωρίς τη συμβολή και καθοδήγηση κάποιων ανθρώπων, τους οποίους θα ήθελα να ευχαριστήσω θερμά.

Κατ' αρχάς θα ήθελα να ευχαριστήσω τον Καθηγητή του Ε.Μ.Π. και επιβλέποντα της διατριβής μου κ. Γεώργιο Κολέτσο για την εμπιστοσύνη που έδειξε στο πρόσωπό μου και την στήριξη που μου παρείχε όλα αυτά τα χρόνια.

Θα ήθελα ακόμα να ευχαριστήσω θερμά τον Επίκουρο Καθηγητή του Ε.Μ.Π. κ. Πέτρο Στεφανέα για την ψύχραιμη καθοδήγησή του, για τις πολύτιμες ιδέες του και κυρίως γιατί ήταν αυτός που με έφερε σε επαφή με το αντικείμενο της διατριβής μου, τις Τυπικές Μεθόδους. Η βοήθειά του υπήρξε ανεκτίμητη και τον ευχαριστώ θερμά για αυτό.

Επίσης, ευχαριστώ την επιβλέπουσα μου κατά τη διάρκεια της παραμονής μου στο Τάμπερε της Φινλανδίας, κα. Ελένη Μπέρκη για την αμέριστη υποστήριξή της και τη βοήθεια που μου παρείχε. Η διαμονή και εργασία μου εκεί ήταν εμπειρία ζωής.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια και τους φίλους μου για τη αδιάκοπη στήριξη και κατανόησή τους.

Κωνσταντίνος Μπάρλας
Αθήνα, Μάρτιος 2018

Περιεχόμενα

Κατάλογος σχημάτων

Κατάλογος πινάκων

Κατάλογος Κωδίκων

1 Εισαγωγή

Η διδακτορική αυτή διατριβή προτείνει τη χρήση Τυπικών Μεθόδων (και συγκεκριμένα Αλγεβρικών Προδιαγραφών) για τη δημιουργία τυπικών προδιαγραφών προτύπων, δηλαδή προδιαγραφών γραμμένων με τέτοιο τρόπο ώστε να ελαχιστοποιηθούν τα λάθη κατανόησης που παρατηρούνται στις προδιαγραφές που συναντάμε συνήθως και είναι γραμμένες σε φυσικές γλώσσες. Οι Τυπικές Μέθοδοι είναι τεχνικές που χρησιμοποιούνται για την μοντελοποίηση (και επαλήθευση) περίπλοκων συστημάτων σε μορφή μαθηματικών οντοτήτων. Η κατασκευή ενός τέτοιου αυστηρού μαθηματικού μοντέλου επιτρέπει την επαλήθευση ιδιοτήτων του συστήματος με τρόπο πολύ πιο αναλυτικό και ενδελεχή από τον εμπειρικό έλεγχο. Τα πλεονεκτήματα της χρήσης Τυπικών Μεθόδων φαίνεται να αντισταθμίζονται κάπως από το υψηλό κόστος εφαρμογής τους, κάτι που κάνει τον ρυθμό αποδοχής τους από τη βιομηχανία πολύ αργό.

Οι τυπικές προδιαγραφές αυτές θα είναι καλό να συνοδεύουν το πρότυπο κατά την κυκλοφορία του στο κοινό και θα είναι σκόπιμο να σχεδιάζονται παράλληλα με το πρότυπο. Οι προδιαγραφές αυτές μπορεί να είναι εκτελέσιμες, δίνοντάς μας έτσι τη δυνατότητα να περιγράψουμε και να επαληθεύσουμε την ισχύ κάποιων απαιτούμενων ιδιοτήτων του προτύπου.

Πριν παρουσιάσουμε την ιδέα, θα χρειαστούμε μερικά θεωρητικά κεφάλαια (Κεφάλαιο 2) για την κατανόηση των αποτελεσμάτων μας. Στο Κεφάλαιο 2.1 περιγράφουμε πώς γίνεται ο σχεδιασμός και η επαλήθευση συστημάτων και τα συνηθισμένα προβλήματα της μεθόδου. Στο Κεφάλαιο 2.2 παρουσιάζουμε τις τυπικές μεθόδους και τις αλγεβρικές προδιαγραφές, ενώ στο Κεφάλαιο 2.3 παρουσιάζουμε το εργαλείο αλγεβρικών προδιαγραφών CafeOBJ. Τέλος, στο Κεφάλαιο 2.4 περιγράφουμε τα (ανοιχτά) πρότυπα.

Έπειτα, στο Κεφάλαιο 3 θα παρουσιάσουμε αναλυτικά την ιδέα της αλγεβρικής προδιαγραφής προτύπων. Στηρίζουμε ότι η ιδέα αυτή έχει τα εξής πλεονεκτήματα:

- Για τη βιομηχανία γιατί:

    1. Από τη σκοπιά του σχεδιαστή που θέλει να χρησιμοποιήσει το εν λόγω πρότυπο, ίσως για να γράψει μία εφαρμογή που το χρησιμοποιεί: Η αλγεβρική προδιαγραφή ενός προτύπου δουλεύει και ως συνοδευτικό κείμενο του προτύπου και υποστηρίζουμε ότι αυτό είναι πιο "βολικό" καθώς είναι πιο μικρό σε έκταση, έχει λιγότερες ασάφειες και δεν είναι πιο δύ-

σκολο στην κατανόηση. Για παράδειγμα, στο Κεφάλαιο 4.1 τυποποιούμε την Abstract Syntax Notation One (ASN.1) και δείχνουμε πως κάποιος που χρησιμοποιεί την ASN.1 ως πρότυπο για να τυποποιήσει την επικοινωνία κάποιου συστήματός του, μπορεί να χρησιμοποιήσει την τυπική εκδοχή του για να μπορέσει να κάνει έλεγχο ιδιοτήτων του συστήματος.

2. Από τη σκοπιά των σχεδιαστών του προτύπου: συμβάλλει σε πιο σαφή και ορθό σχεδιασμό συστημάτων αφού αναγκάζει τους σχεδιαστές να αναλύσουν σε βάθος τις απαιτήσεις του προτύπου πριν τις προδιαγράψουν και προσφέρει εκτεταμένα εργαλεία επαλήθευσης ιδιοτήτων και εντοπισμού σφαλμάτων, ειδικά με τη χρήση Αλγεβρικών Προδιαγραφών.

   Για την στήριξη αυτής της πρότασης θα παρουσιάσουμε τις εξής εργασίες:

   (α΄) Στο Κεφάλαιο 4.2 σχεδιάσαμε από την αρχή ένα σύστημα κοινωνικής δικτύωσης χρησιμοποιώντας τυπικές μεθόδους και χρησιμοποιήσαμε την προδιαγραφή αυτή για να ελέγξουμε ιδιότητές του. Παράλληλα με την ανάλυση της μεθόδου, βρήκαμε με τον τρόπο αυτό λάθη στον αρχικό σχεδιασμό μας, αναγκάζοντάς μας να τροποποιήσουμε το πρότυπό μας πριν το λάθος περάσει σε μετέπειτα στάδια της δημιουργίας, κερδίζοντας έτσι χρόνο.

   (β΄) Στα Κεφάλαια 4.3 και 4.4 δημιουργήσαμε τυπικές προδιαγραφές για τα ήδη υπάρχοντα πρότυπα Open Document Architecture (ODA) και Rich Site Summary (RSS). Εφόσον δεν έχουμε δημιουργήσει εμείς τα πρότυπα, μετατρέψαμε τις προδιαγραφές που συνοδεύουν τα πρότυπα (και είναι γραμμένες σε φυσικές γλώσσες) σε τυπικές προδιαγραφές, τονίζοντας έτσι τα πλεονεκτήματα μιας τυπικής προδιαγραφής έναντι μιας προδιαγραφής σε τυπική γλώσσα: μικρότερο μέγεθος προδιαγραφής, έλλειψη ασαφειών, εύκολο στην κατανόηση. Ειδικά στην περίπτωση του RSS, αυτή η διαδικασία κατέδειξε μερικά προβληματικά μέρη της υπάρχουσας προδιαγραφής του προτύπου, δείχνοντας έτσι πώς η συγγραφή τυπικών προδιαγραφών ευνοεί την βαθύτερη κατανόηση του προτύπου και των εννοιών του.

- Για την κοινότητα των τυπικών μεθόδων καθώς η χρήση μη ακαδημαϊκών παραδειγμάτων (όπως η προδιαγραφή βιομηχανικών προ-

τύπων) δείχνει στον υπόλοιπο κόσμο ότι οι τυπικές μέθοδοι είναι μία εφικτή λύση στο σχεδιασμό και ανάπτυξη λογισμικού.

Στο Κεφάλαιο 4.5 ερευνούμε κατά πόσο είναι εύκολο να διαβαστεί και να γραφεί μία τυπική προδιαγραφή. Παρουσιάζουμε την προσωπική θετική εμπειρία από τη διδασκαλία τυπικών μεθόδων σε μεταπτυχιακούς σπουδαστές για 3 μήνες. Στο τέλος του εξαμήνου οι σπουδαστές είχαν φτάσει σε ικανοποιητικό επίπεδο κατανόησης των τυπικών προδιαγραφών αφού ήταν σε θέση να γράψουν από την αρχή μία τυπική προδιαγραφή ενός προτύπου.

Τέλος, στο Κεφάλαιο 4.6, επιχειρούμε μία επέκταση του Αλγεβρικού Κοινωνικού Δικτύου του Κεφαλαίου 4.2 ώστε να υποστηρίζει νεότερα χαρακτηριστικά των κοινωνικών δικτύων, όπως αυτά των εικονομηνυμάτων με ημερομηνία λήξης. Για να γίνει αυτό επεκτείναμε την υπάρχουσα προδιαγραφή ώστε να υποστηρίζει συστήματα πραγματικού χρόνου, δείχνοντας έτσι πόσο ευέλικτες είναι οι αλγεβρικές προδιαγραφές και πώς αυτές τροποποιούνται με βάση τις εξελίξεις της τεχνολογίας.

2 Θεωρία

2.1 Σχεδιασμός και επαλήθευση συστημάτων

2.1.1 Σχεδιασμός συστημάτων

Ο σχεδιασμός ενός συστήματος που απευθύνεται σε ένα πρόβλημα γίνεται από κάποια ομάδα σε συνεργασία με τους χρήστες που ζητάνε μια λύση στο πρόβλημα αυτό. Συνήθως, ο σχεδιασμός ξεκινάει με το "Software Requirements Specification" (SRS), έγγραφο που αντιπροσωπεύει αυτό που καταλαβαίνει η ομάδα σχεδιασμού σαν απαιτήσεις των χρηστών που ζητάνε λύση σε ένα πρόβλημα. Είναι το έγγραφο που προηγείται της σχεδίασης της λύσης, αφού είναι σημαντικό να έχουν γίνει κατανοητές οι απαιτήσεις των χρηστών πριν το σχεδιασμό λύσης. Το SRS καταγράφει τις απαιτήσεις του συστήματος, τους περιορισμούς που υπάρχουν και περιγράφει τι μπορεί να κάνει η προτεινόμενη λύση. Σε ένα τέτοιο έγγραφο, αλλά και γενικά (στο χώρο της ανάπτυξης λογισμικού) έχουμε τις εξής έννοιες [1]:

- D : Οι ιδιότητες του πεδίου (ο κόσμος).
- R : Οι απαιτήσεις (η δήλωση του προβλήματος).
- S : Η προδιαγραφή. Η γέφυρα ανάμεσα στα D, R και στα P, C και η λύση του προβλήματος.
- P : Το πρόγραμμα (Η υλοποίηση της λύσης).
- C : Ο υπολογιστής (hardware).

Το Παράδειγμα 2.1, εξηγεί τους παραπάνω όρους:

Παράδειγμα 2.1:

Για παράδειγμα, αν ένα online κατάστημα θέλει να δημιουργήσει ένα σύστημα που θα ενημερώνει και θα δείχνει στο χρήστη το διαθέσιμο απόθεμα σε ένα προϊόν, οι παραπάνω έννοιες θα ήταν οι εξής:

- R: Θα θέλαμε ο χρήστης να βλέπει σε πραγματικό χρόνο τις διαθέσιμες ποσότητες στα προϊόντα μας.
- D: Οι μόνοι δύο τρόποι που αλλάζουν το απόθεμά μας σε κάποιο προϊόν είναι η παραγγελία που κάνει ένας χρήστης και η παράδοση προϊόντος από τους προμηθευτές μας.

- S:
    1. Όταν ένας προμηθευτής μας παραδώσει ένα προϊόν, η ποσότητα αυτή θα προστεθεί στην ποσότητα του προϊόντος που υπάρχει στην αποθήκη μας.
    2. Όταν ένας χρήστης ζητήσει ένα προϊόν τότε η ζητούμενη ποσότητα θα αφαιρεθεί από το διαθέσιμο απόθεμα μας.
- P: Το πρόγραμμα που υλοποιεί την προδιαγραφή S.
- C: Ο υπολογιστής που τρέχει το πρόγραμμα P.

Τα SRS έγγραφα σχεδιάζονται στα πρώτα στάδια της διαδικασίας ανάπτυξης λογισμικού. Για να συνταχθεί ένα τέτοιο έγγραφο υλοποιούνται συνεντεύξεις, έρευνες, ερωτηματολόγια, επισκέψεις με (δυνητικούς) χρήστες. Οι απαιτήσεις που συλλέγονται έτσι αξιολογούνται και όσες θεωρηθούν χρήσιμες μπαίνουν στο SRS έγγραφο πριν γίνει η προδιαγραφή του συστήματος. Στις απαιτήσεις δίνεται μοναδικός αριθμός, συνδέονται με τις πηγές τους και αναλύονται με χρήση use case diagrams (συνήθως). Στα τυποποιημένα πρότυπα που υπάρχουν για τέτοια έγγραφα (π.χ. [2]) μπορεί κάποιος να βρει ένα σκελετό για την καταγραφή και ανάλυση των απαιτήσεων του συστήματος. Σε ένα τέτοιο πρότυπο φαίνεται πόσο μακροσκελή μπορεί να είναι τα SRS έγγραφα.

2.1.2 Επαλήθευση συστημάτων

Αν δημιουργήσουμε ένα πρόγραμμα P, θέλουμε να βεβαιωθούμε για την ορθότητά του και το κάνουμε εκτελώντας τις διαδικασίες της επαλήθευσης (verification) και της εγκυροποίησης (validation).

Ο όρος **verification** ελέγχει το κατά πόσο ανταποκρίνεται:

1. Η προδιαγραφή S έναντι στο πρόβλημα R
2. Το πρόγραμμα P έναντι στη προδιαγραφή S

Η έννοια του **validation** είναι πιο χρήσιμη από αυτή της επαλήθευσης και ελέγχει:

1. Το σύστημα των $P + C$ έναντι στον κόσμο $(D + R)$
2. Την πληρότητα του R, ότι δηλαδή περιέχει όλες τις σημαντικές απαιτήσεις

3. Την πληρότητα του D, ότι δηλαδή περιέχει όλες τις σημαντικές πληροφορίες

Αν υποθέσουμε ότι ισχύει το C, τότε το $S + D$ συνεπάγεται το R. Το Παράδειγμα 2.2 δείχνει αυτές τις σχέσεις.

Παράδειγμα 2.2:

Ένα παράδειγμα αυτών των εννοιών φαίνεται στην παρακάτω περιγραφή μιας πολιτικής ασφαλείας μιας βάση δεδομένων μιας εταιρίας.

- R: Μόνο άτομα με εξουσιοδότηση μπορούν να έχουν πρόσβαση στη βάση δεδομένων
- D:
  1. Τα άτομα με εξουσιοδότηση έχουν κατάλληλους κωδικούς.
  2. Οι κωδικοί αυτοί είναι ατομικοί και δεν μοιράζονται με άλλους.
- S: Η πρόσβαση στη βάση δεδομένων θα γίνεται με κωδικό.

Παρατηρούμε ότι $S + D \Rightarrow R$

2.1.3 Συνηθισμένα προβλήματα

Προβλήματα μπορούν να εντοπιστούν σε κάθε μία έννοια που συναντήσαμε στο Κεφάλαιο 2.1.1. Έτσι:

- **C**: Λάθη στο hardware (power, chip, device, network, ..) είναι αρκετά σπάνια.
- **P**: Λάθη στις ιδιότητες του προγράμματος δεν είναι πολύ συχνά και συνήθως οφείλονται είτε σε κάποιο bug, είτε σε κάποια παρανόηση της προδιαγραφής.
- **S**: Λάθη στην προδιαγραφή εμφανίζονται συχνά και οφείλονται κυρίως σε μη κατανοητές, είτε ημιτελείς είτε ασυνεπείς απαιτήσεις.
- **R**: Λάθη στις απαιτήσεις εμφανίζονται συχνά και οφείλονται συνήθως στην κακή επικοινωνία (ή και στην έλλειψή της) μεταξύ

των μελών της ομάδας ανάπτυξης και του πελάτη ή στην ελλιπή ανάλυση τους.

- **D**: Λάθη στο domain εμφανίζονται αρκετά συχνά και συνήθως οφείλονται σε ελλιπή εμπειρία, αστήρικτες υποθέσεις και ελλιπή ανάλυση του κόσμου.

Λάθη που συναντώνται σε οποιοδήποτε από τις έννοιες P, S, R, D μπορεί να οφείλονται στη χρήση των φυσικών γλωσσών.

2.1.4 Φυσικές γλώσσες

Οι απαιτήσεις και οι προδιαγραφές ενός συστήματος γράφονται χρησιμοποιώντας φυσικές γλώσσες μιας και είναι ο πιο φυσικός τρόπος να περιγραφεί το σύστημα. Παρά τη δεδομένη ευκολία που προσφέρει, η χρήση των φυσικών γλωσσών σε μία προδιαγραφή μπορεί να δημιουργήσει προβλήματα σε οποιοδήποτε στάδιο του σχεδιασμού ενός συστήματος (βλ. Παράγραφο 2.1.3). Τα συνηθέστερα προβλήματα είναι τα εξής [1]:

1. **Μέγεθος**: Τέτοια έγγραφα μπορεί να είναι τεράστια σε μέγεθος, όπως π.χ. η προδιαγραφή του ανοικτού προτύπου Digital Imaging and Communications in Medicine (DICOM), ένα πρότυπο για τη διαχείριση, αποθήκευση, εκτύπωση και μεταφορά πληροφοριών ιατρικών απεικονίσεων που περιλαμβάνει περιγραφές ενός τύπου αρχείου και ενός πρωτοκόλλου δικτυακής επικοινωνίας. Η σελίδα του DICOM ([3]) παρέχει το έγγραφο αυτό, μια συλλογή από 18 προδιαγραφές που συνολικά είναι 4900 σελίδες, με το καθένα έγγραφο να πιάνει κατά μέσο όρο 270 σελίδες.

2. **Έλλειψη σαφήνειας**: Δεν είναι εύκολο για τις φυσικές γλώσσες να εκφράσουν όρους με ακρίβεια και χωρίς ασάφειες χωρίς να καταφύγουν σε εκτενή και δυσνόητα κείμενα [4]. Επίσης, η κατανόηση μιας προδιαγραφής σε φυσική γλώσσα στηρίζεται στο ότι τόσο οι συγγραφείς όσο και οι αναγνώστες της προδιαγραφής χρησιμοποιούν τις ίδιες λέξεις για τις ίδιες έννοιες. Όμως αυτό δεν ισχύει πάντα, λόγω της εγγενούς ασάφειας τους. Ένα παράδειγμα του πώς συγγραφείς και αναγνώστες μπορούν να μη συνεννοηθούν σωστά φαίνεται στο [5], όπου οι συγγραφείς τονίζουν την εγγενή ασάφεια των φυσικών γλωσσών σχολιάζοντας δύο επιγραφές που εμφανίζονται δίπλα σε έναν ανελκυστήρα και λένε:

    - "Shoes must be worn"

- "Dogs must be carried"

Ενώ είναι εύκολο να καταλάβει κάποιος τι εννοούν, όταν προσπαθούμε να τις ερμηνεύσουμε **επίσημα** δημιουργούνται ερωτήματα όπως:

- Χρειάζεται να κουβαλάω ένα σκύλο για να μπω στον ανελκυστήρα;
- Πρέπει να φοράω και τα παπούτσια που βρίσκονται στη βαλίτσα που κρατάω;
- Πρέπει να φοράνε παπούτσια και τα σκυλιά;

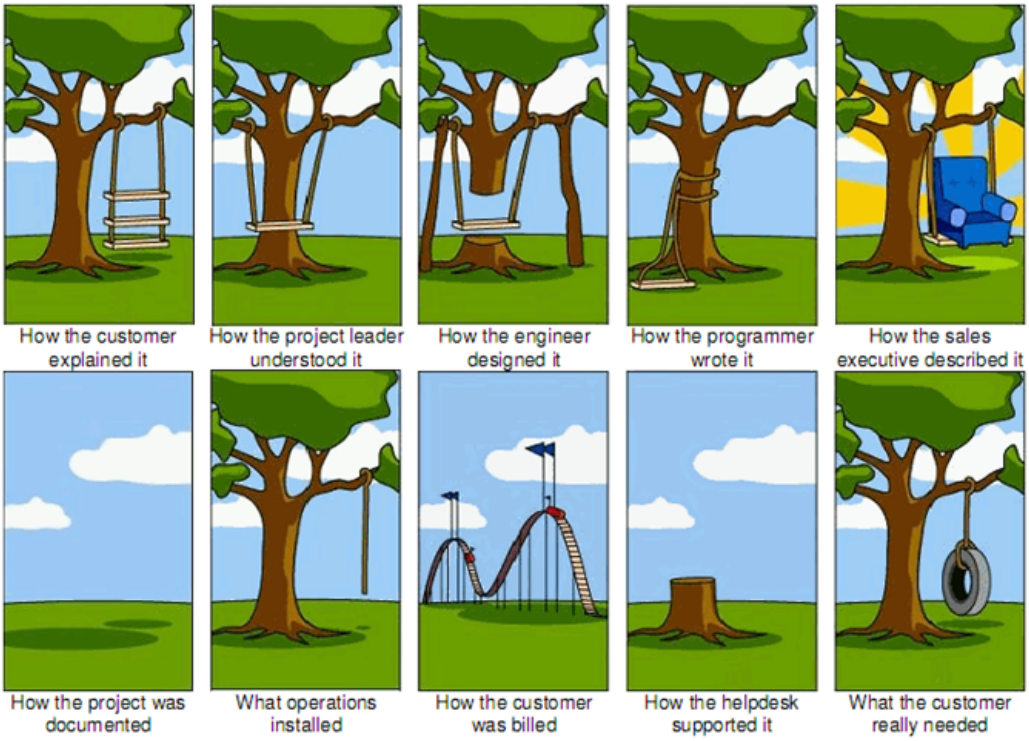
Μιας και όλοι είμαστε εξοικειωμένοι με ένα τέτοιο περιβάλλον, μας είναι εύκολο να ερμηνεύσουμε σωστά τις επιγραφές αυτές, όμως όταν διαβάζουμε μια προδιαγραφή ενός συστήματος με το οποίο δεν είμαστε εξοικειωμένοι, μπορεί να συναντήσουμε συχνά τέτοια προβλήματα. Αυτό συμβαίνει γιατί τέτοιες προδιαγραφές θεωρούν δεδομένη μια εξοικείωση με το domain του συστήματος: για παράδειγμα, η προδιαγραφή για μια ιατρική συσκευή μπορεί να θεωρεί δεδομένο ένα ιατρικό background κάτι που όμως συνήθως δεν είναι κατάλληλα ορισμένο.

3. **Συγχωνεύσεις απαιτήσεων**: Είναι πιθανό, διαφορετικές απαιτήσεις του συστήματος να ενώνονται σε μία· μιας και δεν υπάρχει εύκολος τρόπος να γραφούν ανεξάρτητα τότε είναι δύσκολο να απομονωθεί μία απαίτηση αλλά και να βρεθούν όλες οι σχετικές απαιτήσεις, κάτι που θα θέλαμε να κάνουμε όταν θέλουμε να εξετάσουμε τις επιπτώσεις μιας αλλαγής στο σύστημα [4]. Επίσης, είναι πιθανό να συναντήσουμε μια απαίτηση συστήματος σε διαφορετικά μέρη του εγγράφου και να το δούμε διατυπωμένο με διαφορετικούς τρόπους λόγω της υπερβολικής ευελιξίας των φυσικών γλωσσών.

4. **Συγκεχυμένες απαιτήσεις**: Είναι πιθανό σε μία προδιαγραφή να παρατηρείται μια σύγχυση ανάμεσα σε λειτουργικές απαιτήσεις, μη-λειτουργικές απαιτήσεις, πληροφορίες σχεδιασμού και στόχους του συστήματος [4]. (Οι λειτουργικές απαιτήσεις ενός συστήματος είναι αυτές που προσδιορίζουν κάτι που θα πρέπει να κάνει το σύστημα που σχεδιάζουμε, π.χ. η εκτύπωση μιας παραγγελίας ή η προσθήκη ενός χρήστη στο σύστημα. Οι μη-λειτουργικές απαιτήσεις ενός συστήματος είναι αυτές που προσδιορίζουν πώς θα

έπρεπε να συμπεριφέρεται το σύστημα που σχεδιάζουμε, τα χαρακτηριστικά ποιότητάς του, π.χ. "η αφαίρεση εμπορεύματος από την αποθήκη θα πρέπει να ενημερώνει τις τρέχουσες παραγγελίες που ζητάνε το συγκεκριμένο εμπόρευμα εντός λίγων δευτερολέπτων".)

Τέτοια προβλήματα μπορούν να οδηγήσουν σε παρεξηγήσεις και λάθη στις προδιαγραφές των απαιτήσεων ενός συστήματος (Σχήμα 1) και τις περισσότερες φορές, τέτοια λάθη ανακαλύπτονται στα τελευτάια στάδια της διαδικασίας ανάπτυξης λογισμικού. Η διόρθωση τέτοιων σφαλμάτων τόσο αργά συνήθως κοστίζει πολύ [4].



Σχήμα 1: Η έλλειψη σωστής επικοινωνίας στα διάφορα στάδια της ανάπτυξης λογισμικού

2.2 Τυπικές Μέθοδοι

Οι τυπικές μέθοδοι [6, 7] είναι τεχνικές που βασίζονται στην μαθηματική αυστηρότητα και χρησιμοποιούνται για την περιγραφή, το σχεδιασμό και την ανάλυση της συμπεριφοράς ενός συστήματος επαλη-

θεύοντας ιδιότητες του συστήματος με χρήση αυτόματων και αυστηρών συλλογιστικών εργαλείων (reasoning tools). Οι τυπικές μέθοδοι ξεκίνησαν ως μια λύση στα προβλήματα που εμφανίζονταν στο σχεδιασμό συστημάτων, επιτρέποντας την ενσωμάτωση της διαδικασίας της επαλήθευσης νωρίς κατά τη διαδικασία σχεδιασμού, παρέχοντας πιο αποδοτικές τεχνικές επαλήθευσης και τέλος, μειώνοντας τον απαιτούμενο χρόνο για την επαλήθευση του συστήματος [8]. Οι τυπικές μέθοδοι συνιστώνται από την International Electrotechnical Commission (IEC), την Federal Aviation Administration (FAA), τη National Aeronautics and Space Administration (NASA) και άλλες υπηρεσίες για την ανάπτυξη λογισμικού που είναι απαραίτητο να είναι ασφαλές [8]. Μπορούν να χρησιμοποιηθούν σε οποιοδήποτε τύπο συστήματος όπως συστήματα μεταφοράς, τηλεπικοινωνίες, στο διαδίκτυο και σε online υπηρεσίες, κ.ά. ([9]), αν και θα εστιάσουμε κυρίως στον τομέα της ανάπτυξης λογισμικού. Οι τυπικές μέθοδοι για τον σχεδιασμού λογισμικού περιλαμβάνουν τα εξής βήματα:

1. Τυπική προδιαγραφή του συστήματος: Στο στάδιο αυτό οι απαιτήσεις του συστήματος προσδιορίζονται από τον αναλυτή και χρησιμοποιούμε κάποια τυπική μέθοδο για να δημιουργήσουμε την προδιαγραφή.

2. Ανάλυση και απόδειξη: Στο στάδιο αυτό ελέγχουμε την προδιαγραφή για συνέπεια, κάτι που δεν μπορούμε να κάνουμε σε μια προδιαγραφή που έχει γραφτεί σε κάποια φυσική γλώσσα.

3. Ανάπτυξη του μετασχηματισμού: Για να πάμε από ένα βήμα στο επόμενο πρέπει να διασφαλίσουμε μια "συνέχεια" ώστε το νέο βήμα να ερμηνεύει σωστά αυτά που γίνανε πιο πριν. Το πρόβλημα είναι ότι όταν αυτό που έχει γίνει σε κάθε βήμα έχει καταγραφεί σε μια φυσική γλώσσα, η ερμηνεία που θα δοθεί στις ενέργειες των προηγούμενων βημάτων ενδέχεται να είναι διαφορετική από άτομο σε άτομο.

   Το βήμα αυτό βασίζεται στην ιδέα του να:

   - πάρουμε μια τυπική προδιαγραφή,
   - την ελέγξουμε για συνέπεια,
   - δημιουργήσουμε το design του προγράμματος από αυτήν,
   - επαληθεύσουμε το design,
   - δημιουργήσουμε (αυτόματα) τον κώδικα και

- επαληθεύσουμε ότι ο κώδικας ανταποκρίνεται στην προδιαγραφή.

Τα περισσότερα από αυτά τα βήματα μπορούν να αυτοματοποιηθούν, ελαχιστοποιώντας έτσι την ανθρώπινη παρέμβαση και πιθανές ασάφειες.

4. Επαλήθευση του προγράμματος: Στο βήμα αυτό ελέγχουμε αν αυτό που δημιουργήσαμε ανταποκρίνεται στις απαιτήσεις του βήματος 1. Το βήμα αυτό μπορεί να γίνει και αυτόματα καθώς μπορούμε να δημιουργήσουμε δοκιμαστικά σενάρια από την προδιαγραφή και να ελέγξουμε αν συμπεριφέρονται όπως θα περιμέναμε από το σύστημα.

2.2.1 Τυπικές Προδιαγραφές

Μια υποκατηγορία των τυπικών μεθόδων είναι η *Τυπική Προδιαγραφη (Formal Specification)* που είναι η προδιαγραφή ενός συστήματος που εκφράζεται σε μία γλώσσα με αυστηρά ορισμένο λεξιλόγιο, συντακτικό και σημασιολογία (όπως για παράδειγμα η Z - [10]). Ο τυπικός αυτός ορισμός εξηγεί γιατί οι γλώσσες τυπικών προδιαγραφών πρέπει να στηρίζονται σε μαθηματικές έννοιες των οποίων οι ιδιότητες [11] είναι πλήρως κατανοητές [4]. Οι μαθηματικές αυτές έννοιες είναι συνήθως η θεωρία συνόλων, ο κατηγορηματικός λογισμός και η άλγεβρα Bool.

Υπάρχουν δύο διαφορετικές σχολές στις τυπικές προδιαγραφές. Η σχολή του Model Checking και αυτή της Αλγεβρικής Προδιαγραφής.

2.2.1.1 Model Checking

Το Model Checking είναι μια αυτοματοποιημένη τεχνική που δοσμένου ενός finite-state μοντέλου ενός συστήματος και μιας τυπικής ιδιότητας, ελέγχει συστηματικά αν η ιδιότητα αυτή ισχύει σε κάποια κατάσταση του μοντέλου. Το μοντέλο σε αυτό τον ορισμό αναφέρεται σε ένα σύνολο καταστάσεων του συστήματος τα οποία έχουν ονομαστεί χρησιμοποιώντας βασικές προτάσεις. Οι καταστάσεις αυτές συνδέονται με σχέσεις μετάβασης και τέλος, οι καταστάσεις αυτές χρησιμοποιούνται για τη σύνθεση του συστήματος [8].

Η συνηθισμένη διαδικασία που ακολουθείται στο Model Checking αποτελείται από 3 στάδια ([8]):

1. Σχεδιασμός

   - Σχεδιασμός του συστήματος

- Εκτέλεση κάποιων πρώτων προσομοιώσεων
- Τυποποίηση της ιδιότητας που θέλουμε να ελεγχθεί

2. Εκτέλεση

   - Εκτέλεση του model checker για τον έλεγχο της εγκυρότητας της ιδιότητας στο μοντέλο

3. Ανάλυση

   - Αν η ιδιότητα ισχύει τότε προχωράμε στην επόμενη ιδιότητα (αν υπάρχει)
   - Αν η ιδιότητα δεν ισχύει, τότε:
     (α΄) αναλύουμε το αντιπαράδειγμα που ανέφερε η προσομοίωση
     (β΄) βελτιώνουμε το μοντέλο, το σχεδιασμό ή την ιδιότητα και επαναλαμβάνουμε τη διαδικασία
   - Αν πάλι η προσομοίωση τερματιστεί επειδή το μοντέλο είναι πιο μεγάλο από τη διαθέσιμη μνήμη τότε προσπαθούμε να ελαττώσουμε το μοντέλο και δοκιμάζουμε πάλι

Η μέθοδος του Model Checking ([8]) μπορεί να εφαρμοστεί σε μεγάλο αριθμό συστημάτων διαφορετικού τύπου (hardware, software, πρωτόκολλα, κ.ά.), επιτρέπει μερικό έλεγχο εγκυρότητας (μόνο για τις ιδιότητες που θέλουμε), δίνει αντιπαράδειγμα για τις περιπτώσεις που η ιδιότητα δεν επαληθεύεται, δίνει ασφαλή αποτελέσματα (λόγω του μαθηματικού υπόβαθρου) και τέλος, σε αντίθεση με τις συνηθισμένες μεθόδους ελέγχου, δεν δίνει έξτρα βαρύτητα στα πιο πιθανά σενάρια, αναλύοντας έτσι κάθε πιθανή κατάσταση του συστήματος.

2.2.1.2 Αλγεβρικές Προδιαγραφές

Η μέθοδος της Αλγεβρικής Προδιαγραφής είναι μια τεχνική που χρησιμοποιείται κυρίως στο σχεδιασμό λογισμικού για να τυποποιήσει τη συμπεριφορά του συστήματος. Η αλγεβρική προσέγγιση στην γραφή τυπικών προδιαγραφών εστιάζει στον προσδιορισμό ενός συστήματος βάση των λειτουργιών του και των σχέσεων που σχηματίζουν μεταξύ τους αυτές. Οι τύποι δεδομένων ενός συστήματος περιγράφονται και αυτοί μαζί με τις λειτουργίες που γίνονται πάνω στα δεδομένα αυτά. Οι λειτουργίες αυτές μπορεί να είναι είτε constructors είτε συναρτήσεις που δρουν πάνω στους τύπους δεδομένων που έχουν οριστεί. Έτσι, μια

τέτοια προδιαγραφή περιγράφει όλες τις πιθανές καταστάσεις των δεδομένων και όλες τις πιθανές μεταβάσεις ανάμεσα στις καταστάσεις. Λεπτομέρειες όπως το μέγεθος των αναπαραστάσεων αυτών είναι αρκετά ασαφής, κάτι που δεν είναι πρόβλημα μιας και μία τυπική προδιαγραφή ενός συστήματος περιγράφει τι πρέπει να κάνει ένα σύστημα χωρίς να εξηγεί το πώς υλοποιείται μια τέτοια λειτουργία [12]. Αυτή η ασάφεια μας επιτρέπει να κατανοούμε τη λειτουργικότητα ενός συστήματος χωρίς να χρειάζεται να διαβάζουμε σελίδες κώδικα που περιγράφουν την υλοποίηση [12]. Κάτι τέτοιο δεν υπαινίσσεται ότι όλες οι ιδιότητες ενός συστήματος μπορούν να αποδειχθούν αλγοριθμικά, κάτι που φυσικά θα εναντιωνόταν στα θεωρήματα μη πληρότητας του Gödel. Το αν μια ιδιότητα που μας ενδιαφέρει να εξετάσουμε είναι αποδείξιμη ή όχι είναι κάτι που βαραίνει το συντάκτη του προσδιορισμού [13].

Η μέθοδος εμφανίστηκε στα μέσα της δεκαετίας του '70 και έχει σαν κεντρική ιδέα την προδιαγραφή τύπων δεδομένων με τρόπο παρόμοιο με αυτό που χρησιμοποιείται για τη μελέτη μαθηματικών δομών, όπως οι (υπό)ομάδες, οι δακτύλιοι, τα πεδία, κ.ά. στην άλγεβρα. Η εξισωτική λογική επιλέχθηκε ως ο κατάλληλος φορμαλισμός για την περιγραφή, ενώ η καθολική άλγεβρα και η θεωρία κατηγοριών παρείχαν σημασιολογικές τεχνικές [14]. Έκτοτε η μέθοδος βελτιώθηκε ώστε να γίνει μια τυπική μέθοδος προδιαγραφής που μπορεί να καλύψει ολόκληρη τη διαδικασία της ανάπτυξης συστημάτων: παρέχει ένα σκελετό για την τυπική περιγραφή λογισμικού. Αυτός ο σκελετός επιτρέπει την καλύτερη κατανόηση της διαδικασίας ανάπτυξης λογισμικού αφού παρέχει μεθοδική επίγνωση των θεμάτων που περιγράφονται. Επίσης, οι αλγεβρικές προδιαγραφές παρέχουν μεθόδους και εργαλεία (π.χ. LOTOSPHERE, ACT, PROSPECTRA, κ.ά.) τα οποία χρησιμοποιούνται άμεσα στην ανάπτυξη λογισμικού [14].

Πέρα από τα πλεονεκτήματα των τυπικών μεθόδων, οι αλγεβρικές προδιαγραφές είναι έτσι κατασκευασμένες που μια προδιαγραφή μπορεί να χτιστεί σταδιακά διευκολύνοντας την ανάλυση του συστήματος κάνοντας συστηματική χρήση της αρχής της διάκρισης των υποσυστημάτων [14]. Επίσης, στις αλγεβρικές προδιαγραφές μπορούμε να εκμεταλλευτούμε τη σπονδυλωτή δομή (που έχει συνταχθεί στο στάδιο της προδιαγραφής) στο στάδιο του σχεδιασμού, απλουστεύοντας έτσι την ορθή εκτέλεση του συστήματος [14].

Βασικοί τύποι και διαδικασίες: Σύμφωνα με το [14], μία προδιαγραφή για έναν βασικό τύπο δίνεται από την αλγεβρική προδιαγραφή

$$SPEC = (S, OP, E)$$

που περιλαμβάνει ένα σετ S συνόλων, ένα σετ OP που αντιπροσωπεύει τα σύμβολα των διαδικασιών και τέλος, ένα σετ E εξισώσεων ή αξιωμάτων.

Οι αλγεβρικές προδιαγραφές επιτρέπουν τη σύνθεση μεγαλύτερων προδιαγραφών από μικρότερα κομμάτια χρησιμοποιώντας τους μηχανισμούς της επέκτασης, ένωσης και μετονομασίας. Αν έχουμε μία προδιαγραφή (με όνομα $List_0$) για λίστες που περιέχει τον ορισμό της άδειας λίστας και μια διαδικασία για την προσθήκη στοιχείων σε μία λίστα από μία κατεύθυνση τότε [14]:

- Ο μηχανισμός της επέκτασης μας επιτρέπει να προσθέτουμε σύνολα, διαδικασίες και εξισώσεις σε μία δοσμένη προδιαγραφή. Έτσι, η προδιαγραφή $List$ θα ήταν μια επέκταση της $List_0$ αν χρησιμοποιούσε τον ορισμό της $List_0$ και προσέθετε π.χ. διαδικασία για την ένωση δύο λιστών.

- Ο μηχανισμός της ένωσης δημιουργεί την ένωση δύο ή περισσότερων προδιαγραφών που μοιράζονται υπό-προδιαγραφές. Έτσι, αν χρειαστούμε μια συνάρτηση υπολογισμού μήκους και ένα κατηγόρημα ισότητας τότε μπορούμε να ενώσουμε την προδιαγραφή $List$ με την προδιαγραφή των φυσικών αριθμών (που ονομάζεται Nat) και στη συνέχεια μπορούμε να επεκτείνουμε αυτή την ένωση προσθέτοντας διαδικασίες για τη συνάρτηση υπολογισμού μεγέθους, το κατηγόρημα ισότητας και τέλος, τα κατάλληλα αξιώματα.

- Ο μηχανισμός της μετονομασίας μάς επιτρέπει να έχουμε μια αμφιμονοσήμαντη μετονομασία των συνόλων και των συμβόλων διαδικασιών μιας προδιαγραφής.

Παραμετρικές προδιαγραφές: Ένα πλεονέκτημα της χρήσης των αλγεβρικών προδιαγραφών είναι η ικανότητα να επαναχρησιμοποιούμε μέρη της προδιαγραφής είτε αυτούσια είτε με την αλλαγή κάποιων παραμέτρων [15]. Τα κομμάτια αυτά της προδιαγραφής ονομάζονται έτσι ώστε η χρήση τους αργότερα να γίνεται με αναφορά στο όνομά τους. Μία τέτοια αναφορά σε κάποιο βασικό κομμάτι της προδιαγραφής του δίνει υπόσταση παρέχοντας για κάθε παράμετρο ένα όρισμα προδιαγραφής και έναν κατάλληλο μορφισμό από την παράμετρο στο όρισμα προδιαγραφής [15].

Παράδειγμα 2.3: Σύνθεση λιστών από λίστες ([14])

Ας υποθέσουμε ότι έχουμε μια προδιαγραφή ενός τύπου λίστας, με όνομα **List** που παρέχει το σύνολο $Data$ και που έχει τον ορισμό της άδειας λίστας, τον ορισμό της λίστας με ένα στοιχείο και μια προσεταιριστική διαδικασία που ενώνει λίστες. Η τυπική παράμετρος της προδιαγραφής **List** είναι το σύνολο $Data$ και οι λίστες κατασκευάζονται με το σύνολο **List**$(Data)$.
Αν θέλουμε να δημιουργήσουμε μια νέα προδιαγραφή που να περιγράφει λίστες από λίστες τότε το μόνο που έχουμε να κάνουμε είναι να καλέσουμε την προδιαγραφή List με όρισμα προδιαγραφής το $List(Data)$ και να δημιουργήσουμε έναν μορφισμό (π.χ. $h1$) που θα αντιστοιχεί το σύνολο $Data$ της αρχικής προδιαγραφής στο νέο σύνολο $List(Data)$ δημιουργώντας έτσι προδιαγραφή για λίστες από λίστες.

Η μέθοδος αυτή επιτρέπει τη δημιουργία δομημένων προδιαγραφών: ξεκινώντας από τις βασικές προδιαγραφές, χρησιμοποιούμε τις μεθόδους που αναπτύξαμε για να συνθέσουμε την τελική προδιαγραφή του συστήματος.

Γλώσσες τυπικών προδιαγραφών: Ο Πίνακας 1 δείχνει μερικές από τις πιο δημοφιλείς γλώσσες τυπικών προδιαγραφών χωρισμένες ανάλογα με τη σχολή που αντιπροσωπεύουν (Model Checking ή Αλγεβρικών Προδιαγραφών) και ανάλογα με το αν εστιάζουν σε sequential ή concurrent (πολλές υπο-προδιαγραφές που δουλεύουν ταυτόχρονα) προγράμματα. Στις γλώσσες Αλγεβρικών Προδιαγραφών θα συναντήσουμε την οικογένεια γλωσσών OBJ (CafeOBJ, Eqlog, FOOPS, Kumo, Maude, OBJ3), τη CASL και τη LARCH στην κατηγορία της sequential σχεδίασης και τη γλώσσα LOTOS στην κατηγορία της concurrent σχεδίασης. Στις γλώσσες που στηρίζονται στο Model Checking θα συναντήσει κανείς τη Z και τη VDM στην κατηγορία της sequential σχεδίασης και τις γλώσσες CSP και Petri Nets στην κατηγορία της concurrent σχεδίασης.

Σχολή/Τύπος	**Sequential**	**Concurrent**
Αλγεβρικές	Larch, OBJ	Lotos
Model-based	Z, VDM	CSP, Petri Nets

Πίνακας 1: Είδη γλωσσών τυπικών προδιαγραφών

Παράδειγμα αλγεβρικής προδιαγραφής συστήματος: Στο Παράδειγμα 2.4:

- περιγράφουμε τις προδιαγραφές που έχει ένα σύστημα και τις απαιτήσεις που έχουμε από αυτό
- τις τυποποιούμε μαθηματικά
- χρησιμοποιούμε την προδιαγραφή μας για να αποδείξουμε ιδιότητες.

Παράδειγμα 2.4: Σύστημα Βιβλιοθήκης

Έχουμε μια απλή βιβλιοθήκη που υποστηρίζει τις βασικές διαδικασίες του δανεισμού βιβλίου. Ένα βιβλίο της βιβλιοθήκης μπορεί να βρίσκεται σε μία από τις ακόλουθες τρεις καταστάσεις:

1. Διαθέσιμο: Το βιβλίο βρίσκεται στη βιβλιοθήκη και είναι διαθέσιμο για δανεισμό.
2. Έχει ζητηθεί: Το βιβλίο βρίσκεται στη βιβλιοθήκη αλλά δεν είναι διαθέσιμο για δανεισμό γιατί κάποιος το έχει ήδη ζητήσει.
3. Δανεισμένο: Το βιβλίο έχει δοθεί σε κάποιο μέλος της βιβλιοθήκης.

Μπορούμε να τυποποιήσουμε τις έννοιες της βιβλιοθήκης και τις καταστάσεις ενός βιβλίου και μετά να αποδείξουμε κάποια θεωρήματα ελέγχοντας την εγκυρότητα της προδιαγραφής μας. Έτσι ελέγχουμε αν η προδιαγραφή που έχουμε ταιριάζει με το τι θέλουμε και αντιστρόφως.
Έτσι λοιπόν, ένα βιβλίο:

- S : βρίσκεται σε ράφι
- R : βρίσκεται σε στοίβα
- L : έχει δανεισθεί
- Q : ζητήθηκε από κάποιο μέλος

Θεώρημα 1. *Ένα βιβλίο μπορεί να βρίσκεται σε μία (μόνο) από τις τρεις πιθανές καταστάσεις (S, R, ή L):*
$S \iff \neg(R \vee L)$ $(S \Rightarrow \neg(R \vee L) \wedge \neg(R \vee L) \Rightarrow S)$
$R \iff \neg(S \vee L)$
$L \iff \neg(S \vee R)$

Θεώρημα 2. *Αν ένα βιβλίο έχει ζητηθεί τότε βρίσκεται είτε σε ράφι είτε σε στοίβα.* $Q \Rightarrow (S \vee R)$

Χρησιμοποιώντας τα 2 θεωρήματα της προδιαγραφής μας θα προσπαθήσουμε να αποδείξουμε ιδιότητες του συστήματος που περιγράφουμε ώστε να αυξήσουμε την εμπιστοσύνη μας στην υπάρχουσα προδιαγραφή. Θα προσπαθήσουμε να αποδείξουμε την ακόλουθη πρόταση:

Πρόταση 1. *Δε μπορούμε να ζητήσουμε βιβλίο που έχει δανειστεί.* $L \Rightarrow \neg Q$

Απόδειξη.

$$\begin{aligned} \neg(L \Rightarrow \neg Q) = & \quad \text{(Υποθέτουμε το αντίθετο)} \\ \neg(\neg L \vee \neg Q) = & \quad (A \Rightarrow B) \equiv (\neg A \vee B) \\ L \wedge Q = & \quad \text{De Morgan:} \neg(A \vee B) \Leftrightarrow (\neg A) \wedge (\neg B)) \\ (\neg(S \vee R)) \wedge (S \vee R) & \quad \text{(Άτοπο)} \end{aligned}$$

Άρα $L \Rightarrow \neg Q$ □

2.2.2 Κριτική των τυπικών μεθόδων

Παρά το πλεονεκτήματα των τυπικών μεθόδων (επιτρέπουν μεγάλη ακρίβεια, σαφήνεια και δημιουργούν ένα πλαίσιο πάνω στο οποίο μπορούν να αποδειχθούν ιδιότητες ενός συστήματος), δε χρησιμοποιούνται ιδιαίτερα στη βιομηχανία, παρά μόνο σε κρίσιμα συστήματα στα οποία ένα λάθος θα μπορούσε να είναι καταστροφικό [16] και άρα χρειάζεται να είμαστε όσο το δυνατόν πιο σίγουροι ότι θα δουλεύουν σωστά. Μερικοί λόγοι που συμβαίνει αυτό (παρά τις αρχικές προβλέψεις για τις τυπικές μεθόδους στη δεκαετία του '70) είναι:

- Η βιομηχανία λογισμικού δεν είναι ιδιαίτερα πρόθυμη να αλλάξει τις υπάρχουσες διαδικασίες έρευνας και ανάπτυξης και να υιοθετήσει νέες τεχνικές που θα απαιτούσαν κατάλληλο προσωπικό (θα απαιτούσε είτε νέο προσωπικό είτε την εκπαίδευση του τρέχοντος στις τυπικές μεθόδους [13]) και κάτι τέτοιο κοστίζει σε χρόνο και χρήμα.
- Όμως, ακόμα και με κατάλληλα εκπαιδευμένο προσωπικό, η ανάπτυξη μιας τυπικής προδιαγραφής απαιτεί περισσότερο χρόνο στα

αρχικά της στάδια λόγω της δυσκολίας ανάπτυξης των μαθηματικών μοντέλων. Η τρέχουσα κατάσταση της αγοράς είναι περισσότερο προσανατολισμένη στο να κυκλοφορήσει το λογισμικό γρήγορα παραβλέποντας τυχόν λάθη που θα μπορούν να εντοπιστούν και να διορθωθούν αργότερα με κάποια ενημέρωση. Χρησιμοποιώντας τυπικές μεθόδους πετυχαίνουμε λογισμικό με ελάχιστα λάθη σε βάρος του χρόνου. Όμως, ο συνολικός χρόνος που απαιτείται για τη σχεδίαση λογισμικού έτσι μπορεί να είναι λιγότερος από όσο χρειάζεται με τις τρέχουσες μεθόδους αφού οι τυπικές μέθοδοι απαιτούν μια λεπτομερή ανάλυση των απαιτήσεων και αυτό ελαττώνει τα λάθη λόγω της ελλιπούς κατανόησής τους ή λόγω κακού σχεδιασμού [4]. Έτσι, οι ημιτελείς ή ασυνεπείς απαιτήσεις μπορούν να εντοπιστούν νωρίς έτσι ώστε οι σχεδιαστές να μη χρειάζεται να επιστρέψουν σε προηγούμενα στάδια και να σχεδιάσουν ξανά το σύστημα πριν συνεχίσουν με τις δοκιμές. Ο συνολικός χρόνος που απαιτείται λοιπόν είναι μάλλον ο ίδιος με τις τρέχουσες μεθόδους, απλά κατανέμεται διαφορετικά στα διάφορα στάδια του σχεδιασμού [4]: με τις τυπικές μεθόδους απαιτείται περισσότερος χρόνος στα αρχικά στάδια, ενώ με τις συμβατικές μεθόδους απαιτείται περισσότερος χρόνος στο τελικό στάδιο των ελέγχων αφού αν κάτι δε δουλέψει όπως πρέπει πρέπει να γίνουν αλλαγές στη σχεδίαση του συστήματος που έχει γίνει στα πρώτα στάδια.

- Η τρέχουσα κατάσταση απλά δουλεύει. Η ποιότητα κατασκευής βελτιώνεται όσο βελτιώνονται τα εργαλεία δημιουργίας (αυξημένη αποδοτικότητα και καλύτερη ποιότητα προϊόντος με λιγότερα λάθη και αυτό ελαττώνει την ανάγκη χρήσης τυπικών μεθόδων.

- Τέλος, το μαθηματικό υπόβαθρο από μόνο του είναι αρκετό για να αποθαρρύνει κάποια εταιρία να ασχοληθεί με τις τυπικές μεθόδους.

Στον αντίποδα, σύμφωνα με το [12], μία τυπική προδιαγραφή μπορεί να χρησιμοποιηθεί ως ένα μοναδικό και αξιόπιστο σημείο αναφοράς για όσους ερευνούν τις απαιτήσεις ενός πελάτη, όσους αναπτύσσουν προγράμματα για να καλύψουν αυτές τις ανάγκες, όσους δοκιμάζουν τα προγράμματα αυτά και, τέλος, όσους γράφουν τα εγχειρίδια αυτών των συστημάτων.

2.3 CafeOBJ

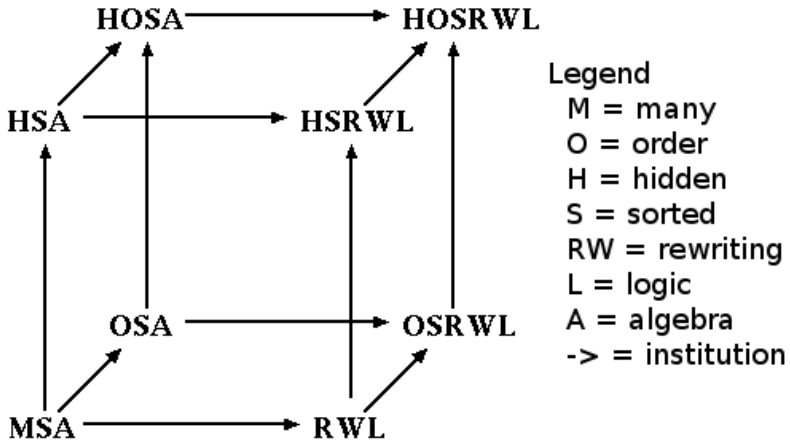
Στη διατριβή αυτή χρησιμοποιούμε τη γλώσσα τυπικών προδιαγραφών (Κεφάλαιο 2.2)) CafeOBJ, μέλος της οικογένειας αλγεβρικών προδιαγραφών OBJ και πιο συγκεκριμένα, της OBJ3 [17]. Οι γλώσσες αλγεβρικών προδιαγραφών OBJ υποστηρίζουν παραμετρικό προγραμματισμό και στηρίζονται κυρίως στην εξισωτική λογική με διατεταγμένους τύπους (order sorted equational logic), συνδυασμένη με άλλες λογικές όπως τη λογική της αναγραφής (rewritting logic), την εξισωτική λογική με κρυμμένους τύπους (hidden sorted equational logic) ή τη λογική πρώτου βαθμού (first order logic).

Η οικογένεια γλωσσών OBJ έχει χρησιμοποιηθεί σε πολλές εφαρμογές [18] όπως: αλγεβρικές προδιαγραφές, ταχεία προτυποποίηση, ορισμό άλλων γλωσσών προγραμματισμού με τρόπο που αποδίδει άμεσα διερμηνέα, προδιαγραφές συστημάτων λογισμικού (GKS graphics kernel system, Ada configuration manager, MacIntosh QuickDraw, κ.λπ.), προδιαγραφές hardware, προδιαγραφή και επαλήθευση imperative προγραμμάτων, σχεδιασμός user interface, αποδείξεις θεωρημάτων. Η OBJ3 έχει χρησιμοποιηθεί για τη δημιουργία του FOOPS (ένα αντικειμενοστραφές σύστημα προδιαγραφών και προγραμματισμού), το σύστημα Eqlog (για εξισωτική λογική ή σχεσιακό προγραμματισμό, την αντικειμενοστραφή γλώσσα προδιαγραφών (με επιρροές από τη Z) γλώσσα OOZE, κ.ά. Η CafeOBJ ([19, 20]) είναι σχετικά πρόσφατη προσθήκη στην κοινότητα των γλωσσών OBJ. Τα κύρια λογικά συστήματα που χρησιμοποιεί είναι:

- Άλγεβρα με διατεταγμένους τύπους (order-sorted algebra) [19, 21]: Χρησιμοποιείται για να προσδιορίσει αφηρημένους τύπους δεδομένων.

- Κρυφή άλγεβρα [19, 22]: Χρησιμοποιείται για να προσδιορίσει αφηρημένες καταστάσεις συστημάτων, παρέχοντας έτσι υποστήριξη για αντικειμενοστραφείς προδιαγραφές.

Όλα τα συστήματα λογικής που χρησιμοποιεί η CafeOBJ και οι τρόποι που αυτά συνδυάζονται φαίνονται στον κύβο του Σχήματος 2.3. Τα βέλη του κύβου δείχνουν τις ενσωματώσεις ανάμεσα στις λογικές και ο προσανατολισμός των βελών δηλώνει την ενσωμάτωση μιας "απλούστερης" λογικής σε μία πιο σύνθετη.



Σχήμα 2: Ο Κύβος της *CafeOBJ*

2.3.1 Σύνταξη της CafeOBJ

Στο Παράδειγμα 2.5 φαίνεται μια απλή προδιαγραφή των φυσικών αριθμών με διάδοχο και πρόσθεση [23].

Παράδειγμα 2.5:

```
mod! SIMPLE−NAT {
  signature {
    [ Zero NzNat < Nat ]
    op 0 : −> Zero
    op s : Nat −> NzNat
    op _+_ : Nat Nat −> Nat
  }
  axioms {
    vars N 'N : Nat
    eq 0 + N = N .
    eq s(N) + 'N = s(N + 'N) .
  }
}
```

Κώδικας 1: Φυσικοί αριθμοί με διάδοχο και πρόσθεση στη CafeOBJ

Η λέξη *mod* (από το module) δηλώνει την αρχή της προδιαγραφής, η οποία βρίσκεται ανάμεσα στα {}. Ο χαρακτήρας ! (ή ∗) μετά τη λέξη *mod*

υποδηλώνει ότι η προδιαγραφή θα έχει πρωταρχική (ή χαλαρή για το $*$) σημασιολογία, δηλαδή θα δηλώνει όλα τα μοντέλα που ικανοποιούν την προδιαγραφή και όχι μόνο τα ισομορφικά με το πρωταρχικό μοντέλο. Μετά ακολουθεί το όνομα που δίνουμε στο module ($SIMPLE-NAT$).

Στην επόμενη σειρά, με τη λέξη κλειδί $signature$, ξεκινάει το κομμάτι των δηλώσεων του module: Ορίζουμε τα ονόματα των τύπων που ορίζει αυτή η προδιαγραφή μέσα σε αγκύλες. Οι τύποι μπορούν να ιεραρχηθούν με χρήση συγκριτικών τελεστών και εδώ βλέπουμε ότι οι τύποι $Zero$ και $NzNat$ είναι υποτύποι του τύπου Nat, χωρίς όμως να φαίνεται η σχέση μεταξύ των δύο πρώτων τύπων. Η λέξη κλειδί op ξεκινά τη δήλωση τελεστή και ακολουθείται από το όνομά του, μία άνω και κάτω τελεία, και τους τύπους των πεδίων ορισμού και τιμών, χωριζόμενους από το βέλος ($->$). Στο παράδειγμά μας, ο τελεστής 0 δεν έχει πεδίο ορισμού ενώ το πεδίο τιμών του είναι τύπου $Zero$. Ο τελεστής s δέχεται σαν όρισμα έναν τύπο Nat και επιστρέφει έναν τύπο Nat. Ο τελεστής $+$ δέχεται σαν όρισμα δύο τύπους Nat (έναν στα αριστερά του και έναν στα δεξιά του, όπως δείχνουν οι κάτω παύλες) και επιστρέφει τύπο Nat.

Η λέξη κλειδί $axioms$ ξεκινάει τα αξιώματα της προδιαγραφής. Η λέξη κλειδί eq σηματοδοτεί την έναρξη του ορισμού μίας εξίσωσης το τέλος της οποίας σηματοδοτείται από μία τελεία "." Οι μεταβλητές που χρησιμοποιούνται στον ορισμό ενός τελεστή δηλώνονται χρησιμοποιώντας τη λέξη κλειδί var ακολουθούμενη από το όνομα της μεταβλητής, μία άνω και κάτω τελεία και το τύπο της μεταβλητής. Μπορούμε να δηλώσουμε πολλές μεταβλητές του ίδιου τύπου σε μία γραμμή χρησιμοποιώντας τη λέξη κλειδί $vars$. Τα δύο αξιώματα στο παράδειγμα δίνουν τον επαγωγικό ορισμό της πρόσθεσης χρησιμοποιώντας τη συνάρτηση του διάδοχου αριθμού (s). Οι λέξεις κλειδιά $signature$ και $axioms$ δεν είναι υποχρεωτικές.

Το παράδειγμα 2.6 δείχνει την προδιαγραφή του παραγοντικού, γραμμένο στην CafeOBJ. Οι γραμμές που ξεκινάνε με $--$ είναι σχόλια και η CafeOBJ δεν τα λαμβάνει υπόψη. Το module αυτό "προστατεύει" τον ενσωματωμένο τύπο της CafeOBJ Int, κληρονομώντας έτσι τους τύπους, τους τελεστές και τα αξιώματά του. Έτσι λοιπόν, δημιουργούμε τον τελεστή $fact$ που δέχεται έναν ακέραιο και επιστρέφει έναν ακέραιο και δημιουργούμε την ακέραια μεταβλητή X. Οι δύο εξισώσεις που ορίζουμε αποτελούν τον αναδρομικό ορισμό του παραγοντικού, με την λέξη κλειδί ceq να ορίζει μία υπό-συνθήκη εξίσωση. Η λέξη κλειδί ceq ακολουθείται από μία εξίσωση στο τέλος της οποίας βρίσκεται μία έκφραση της μορφής $if\ P$, όπου P είναι κάποιο κατηγόρημα. Έτσι, ο ορισμός του πρώτου αξιώματος λέει ότι το $fact(X)$ θα είναι ίσο με 1 αν

το X δεν είναι θετικό. Ομοίως, ο ορισμός του δεύτερου αξιώματος λέει πως αν το $X > 0$ τότε το παραγοντικό του X θα είναι ίσο με το επί το παραγοντικό του $X - 1$.

Παράδειγμα 2.6:

```
mod! FACT{
  -- ορισμόςπαραγοντικού
  protecting(INT)
  op fact : Int -> Int
  var X : Int
  ceq fact(X) = 1 if not (X > 0) .
  ceq fact(X) = X * fact(X - 1) if X > 0 .
}

select FACT .
red fact(5) .
```

Κώδικας 2: Ένα module στην CafeOBJ για το παραγοντικό

Κάτω από τον ορισμό της προδιαγραφής του παραγοντικού έχουμε δύο σειρές που ανοίγουν τη προδιαγραφή αυτή (εντολή *select*) και ζητάνε το παραγοντικό του 5 (με την εντολή *reduce* ή *red*). Το σύστημα της CafeOBJ θα χρησιμοποιήσει τις εξισώσεις της προδιαγραφής του FACT (και όσων άλλων modules καλούνται από αυτό) σαν κανόνες αναγραφής (από αριστερά στα δεξιά) για να απαντήσει στο ερώτημά μας.

Στη γλώσσα ορίζονται δύο ειδών ισότητες μέσω δύο κατηγορημάτων, ο πρώτος τελεστής ισότητας ορίζεται ως $=$ ενώ ο δεύτερος ως $==$. Και οι δύο τελεστές αυτοί επιστρέφουν $true$ αν τα δύο τους ορίσματα μπορούν να αναγραφούν (με βάση το σύστημα αναγραφής της γλώσσας) στο ίδιο όρο. Αν έχουμε δύο μεταβλητές (L, L') τύπου S τότε ο όρος $L == L'$ επιστρέφει $false$ αν ο L αναγράφεται σε $t1$ και ο L' σε $t2$ ενώ ο ορός $L = L'$ θα επιστρέψει $t1 = t2$.

2.3.2 Συστήματα Παρατηρήσεων Μεταβάσεων (Observational Transition Systems)

Ένα *Observational Transition System*, ή *OTS* [24, 25], είναι ένα σύστημα μεταβάσεων που μπορεί να περιγραφεί με εξισώσεις. Υποθέτοντας ότι υπάρχει ένας χώρος καταστάσεων Y και ότι κάθε τύπος δεδομένων που χρειάζεται να χρησιμοποιήσουμε στο OTS (συμπεριλαμβανομένων και των σχέσεων ισοδυναμίας τους) έχει δηλωθεί ήδη, τότε ένα OTS S

ορίζεται σαν τη τριάδα $< O, I, T >$, όπου:

1. D είναι ένα πεπερασμένο σύνολο παρατηρητών. Κάθε $o \in O$ είναι μια συνάρτηση $o : Y \to D$, όπου D είναι ένας τύπος δεδομένων που μπορεί να είναι διαφορετικός για κάθε παρατηρητή. Δοθέντος ενός OTS S και δύο καταστάσεων $u_1, u_2 \in Y$, η ισότητα $u_1 =_s u_2$ μεταξύ τους σε σχέση με το S ορίζεται ως: $\forall o \in O, o(u_1) = o(u_2)$.

2. I είναι ένα σύνολο των καταστάσεων του συστήματος έτσι ώστε $I \subseteq Y$. Το I λέγεται και σύνολο αρχικών καταστάσεων.

3. T είναι ένα σύνολο από υπό-συνθήκη κανόνων μετάβασης. Κάθε τέτοιος κανόνας $\tau \in T$ είναι μία συνάρτηση $\tau : Y \to Y$, έτσι ώστε $\tau(u_1) =_s \tau(u_2)$ για κάθε $u_1, u_2 \in Y/ =_s$ (το $(Y/ =_s$ υποδηλώνει τις κλάσεις ισοδυναμίας για το χώρο καταστάσεων Y). Λέμε ότι δύο καταστάσεις θα είναι ισοδύναμες αν όλοι οι παρατηρητές επιστρέφουν τις ίδιες τιμές. Για κάθε $u \in Y, \tau(u)$ θα λέγεται η διάδοχη κατάσταση του u μετά την εφαρμογή του κανόνα μετάβασης τ. Με κάθε κανόνα μετάβασης ορίζουμε και την αναγκαία συνθήκη c_τ για να πραγματοποιηθεί ο κανόνας μετάβασης τ. Τέλος, αν ένας κανόνας μετάβασης τ εφαρμοστεί σε μία κατάσταση συστήματος u η οποία δεν ικανοποιεί τη αναγκαία συνθήκη του (c_τ), τότε η κατάσταση παραμένει η ίδια. (Για κάθε $u \in Y, \tau(u) = u$ αν $\neg c_\tau(u)$).

Τόσο οι παρατηρητές όσο και οι μεταβάσεις μπορούν να παραμετροποιηθούν. Οι παρατηρητές συμβολίζονται με $o_{i1,\ldots,im}$ και οι μεταβάσεις με $\tau_{j1,\ldots,jn}$, εφόσον $m, n \geq 0$ και εφόσον υπάρχουν τύποι δεδομένων D_k, D με $k = i_1, ..., i_m, j_1, ..., j_n$.

Ένα *Συμπεριφοριακό Αντικείμενο* (Behavioral Object) ορίζεται [26], μη τυπικά, ως μία συμπεριφοριακή προδιαγραφή που δηλώνει επίσημα το χώρο καταστάσεων του αντικειμένου μαζί με ένα σύνολο μεθόδων (μεταβάσεις) που μεταβάλλουν την κατάσταση και ένα σύνολο παρατηρητών που επιστρέφουν τιμές τύπων δεδομένων που μας ενδιαφέρουν. Ο ορισμός του OTS συμμορφώνεται με αυτό τον άτυπο ορισμό του συμπεριφοριακού αντικειμένου.

Θα αποκαλούμε εκτέλεση, μια άπειρη ακολουθία από καταστάσεις $u_0, \ldots, u_n, \ldots$ οι οποίες ικανοποιούν τις ακόλουθες συνθήκες:

- (Αρχική συνθήκη) $u_0 \in \mathcal{I}$
- (Διαδοχή) για κάθε $i \in \{0, 1, \ldots\}$ $u_{i+1} =_S \tau(u_i)$, για κάποιο $\tau \in \mathcal{T}$, δηλαδή όλες οι καταστάσεις προκύπτουν από την προηγούμενη

κατάσταση της ακολουθίας εφαρμόζοντας κάποιο κανόνα μετάβασης.

Παράδειγμα 2.7: Παράδειγμα ενός OTS

Ως σύστημα, έχουμε μια λίστα ακεραίων αριθμών ($list$). Έστω:

- ένας παρατηρητής $head$ που επιστρέφει το πρώτο στοιχείο της λίστας
- ένας παρατηρητής $tail$ που επιστρέφει το τελευταίο στοιχείο της λίστας
- ένας κανόνας μετάβασης $add(list, x)$ ο οποίος προσθέτει τον αριθμό x στη λίστα $list$
- ένας κανόνας μετάβασης $pop(list)$ ο οποίος αφαιρεί το πρώτο στοιχείο της λίστας, αν αυτή δεν είναι κενή.

Η αρχική κατάσταση u_0 του συστήματος είναι να έχουμε κενή λίστα. Εφαρμόζοντας διαδοχικά τον κανόνα μετάβασης add, κάθε φορά η τρέχουσα κατάσταση του συστήματος

- (Αρχική συνθήκη) $u_0 \in \mathcal{I}$: Η λίστα είναι κενή. Έτσι το $head$ της λίστας θα είναι κενό.
- (Διαδοχή) Κάθε εφαρμογή κάποιου κανόνα μετάβασης στην τρέχουσα κατάσταση αλλάζει τη λίστα καταλλήλως. Η λίστα είναι παρατηρήσιμη μέσω των δύο παρατηρητών της, $head$ και $tail$. Έτσι η πρώτη διάδοχη κατάσταση u_1 της αρχικής μπορεί π.χ. να προκύπτει από την εφαρμογή του $add(list, 3)$ και ο τρόπος που έχουμε να παρατηρήσουμε αυτή την κατάσταση είναι μέσω του $head(add(list, 3))$ και $tail(add(list, 3))$. Για να προκύψει κάποια κατάσταση από την εφαρμογή του κανόνα μετάβασης pop, θα πρέπει να επαληθεύεται η αναγκαία συνθήκη του, δηλαδή η λίστα να μην είναι κενή. Η κατάσταση (λίστα) που προκύπτει σε αυτή την περίπτωση παρατηρείται πάλι με χρήση των δύο παρατηρητών.

Μία κατάσταση του συστήματος λέγεται *προσβάσιμη* (reachable) ανν εμφανίζεται ως αποτέλεσμα ενεργειών που περιγράφει το S. Με R_S θα συμβολίζουμε το σύνολο όλων των προσβάσιμων καταστάσεων του S.

Μια ιδιότητα ονομάζεται *αναλλοίωτη* (invariant) στο S, ανν ισχύει για όλες τις προσβάσιμες καταστάσεις του S. Δηλαδή:
invariant(p) ανν $(\forall u \in \mathcal{I}, p(u))$ και $(\forall u \in R_S \wedge \tau \in \mathcal{T}.p(u) \rightarrow p(\tau(u)))$

Για να προδιαγράψουμε ένα OTS στην CafeOBJ:

- Το σύνολο των καταστάσεων του συστήματος περιγράφεται από ένα κρυφό τύπο (hidden sort), H.
- Ορίζουμε τους κατάλληλους παρατηρητές $o_{i_1,\ldots,i_m} \in \mathcal{O}$.
- Προδιαγράφουμε τους τύπους δεδομένων που θα χρειαστούμε (D_{i_1}, $\ldots D_{i_m}$ και D) ώστε να δώσουν τους αντίστοιχους ορατούς τύπους $V_{i_1}, \ldots V_{i_m}$ και V.

Τότε:

- Ο τελεστής της CafeOBJ που αντιστοιχεί στις παρατηρήσεις $o_{i_1,\ldots,i_m}$ ορίζεται ως, bop o : H $V_{i_1} \ldots V_{i_m}$ -> V.
- Κάθε αρχική κατάσταση $u \in \mathcal{I}$ αναπαριστάται από μία κρυφή σταθερά $op\, u_{init} : -> H$.
  Αν η τιμή του $o_{i_1,\ldots,i_m}$ στην αρχική κατάσταση του συστήματος u ορίζεται ως $f(i_1, \ldots, i_m)$ τότε αυτό ορίζεται ως

  $$o(u_{init}, X_{i_1}, \ldots, X_{i_m}) = f(X_{i_1}, \ldots X_{i_m})$$

  όπου $X_{i_1}, \ldots, X_{i_m}$ είναι μεταβλητές τύπων $V_{i_1}, \ldots V_{i_m}$ αντίστοιχα ορισμένες.
- Ένας κανόνας μετάβασης $\tau_{j_1,\ldots,j_n}$ αναπαρίσταται από μία μέθοδο: $bop\, a : H\, V_{j_1} \ldots, V_{j_n} -> H$.
- Αν ένας υπό-συνθήκη κανόνας μετάβασης εφαρμοστεί σε μία κατάσταση όπου η συνθήκη του ισχύει, τότε η τιμή ενός παρατηρητή, έστω $o_{i_1,\ldots,i_m}$, μπορεί να αλλάξει. Αυτό περιγράφεται με μία εξίσωση:
  ceq o(a(u,$X_{j_1}, \ldots, X_{j_n}$), $X_{i_1}, \ldots, X_{i_m}$) = c-a(u,$X_{j_1}, \ldots, X_{j_n}$, X_{i_1}, X_{i_m}) if c-a(u,$X_{j_1,\ldots,X_{j_n}}$).
- Τέλος αν ένας υπό-συνθήκη κανόνας μετάβασης a εφαρμοστεί σε μία κατάσταση στην οποία δεν ισχύει η συνθήκη του, τότε η νέα κατάσταση πρέπει να είναι ισοδύναμη με την προηγούμενη όσον αφορά στην $=_S$. Αυτό δηλώνεται με μία εξίσωση:
  $ceq\ a(u, X_{j_1}, \ldots, X_{j_n}) = u\ if\ not\ c - a(u, X_{j_1}, \ldots, X_{j_n})$.

2.3.3 Απόδειξη ιδιοτήτων συστήματος με τη μεθοδολογία των Proof Scores

Ένα πλεονέκτημα της χρήσης αλγεβρικών προδιαγραφών είναι η δυνατότητα που έχουμε να επαληθεύσουμε ότι το σύστημά μας διατηρεί κάποιες επιθυμητές ιδιότητες σε κάθε προσβάσιμη κατάστασή του. Μια μεθοδολογία για αυτό είναι η μέθοδος *OTS/Proof Score*[24, 27]. Η μέθοδος είναι διαδραστική ανάμεσα στο χρήστη και στον υπολογιστή. Το Proof Score είναι ένα σχέδιο επαλήθευσης ότι μια ιδιότητα ισχύει στην προδιαγραφή μας. Υλοποιείται ως ένα σύνολο οδηγιών, γραμμένες από άνθρωπο σε μία μηχανή αποδείξεων έτσι ώστε όταν αυτές εκτελεστούν, αν όλα αποτιμηθούν όπως αναμένεται, το επιθυμητό θεώρημα αποδεικνύεται. Η μέθοδος αυτή αποκρύπτει τους ενδιάμεσους υπολογισμούς που κάνει ο υπολογιστής και φανερώνει μόνο το σχέδιο απόδειξης που έχει καταστρώσει ο άνθρωπος [27]. Η κύρια διαφορά από το Model Checking είναι η δυνατότητα της μεθοδολογίας των Proof Scores να χειριστούν συστήματα με άπειρο αριθμό καταστάσεων και η εγγενής τους τάση προς την ασάφεια και την έμφαση στην επαναχρησιμοποίηση [27].

Αν και υπάρχουν πλήρως αυτοματοποιημένοι τρόποι αποδείξεων ιδιοτήτων οι μέθοδοι αυτές συχνά δε δίνουν στο χρήστη να καταλάβει τον τρόπο απόδειξης. Οι αυτοματοποιημένοι theorem provers μπορούν να δείξουν αν μία ιδιότητα ισχύει ή όχι αλλά αν η απόδειξη αποτύχει δεν παρέχουν αρκετά στοιχεία στο χρήστη. Έτσι ο χρήστης δεν είναι σίγουρος αν η αποτυχία στην απόδειξη οφείλεται σε κάποιο λάθος στη σχεδίαση, ή χρειάζεται περισσότερα στοιχεία στην είσοδο ή κάποιο λήμμα. Αντίθετα, στη μέθοδο OTS/Proof Score, αν η απόδειξη μιας ιδιότητας αποτύχει, η CafeOBJ θα επιστρέψει είτε "false" είτε κάποια έκφραση. Στην πρώτη περίπτωση ο χρήστης απέδειξε ότι στην κατάσταση αυτή δεν ισχύει η ιδιότητα που θέλαμε να ελέγξουμε, στη δεύτερη περίπτωση ο χρήστης μπορεί να παρέμβει και να δώσει περισσότερα στοιχεία στη προδιαγραφή ελέγχοντας έτσι κατά πόσο η κατάσταση που ελέγχουμε είναι προσβάσιμη από το σύστημα. Τα proof scores εκτελούνται πάνω στη προδιαγραφή άρα έχουν τον ίδιο βαθμό ασάφειας με την ίδια την προδιαγραφή. Η μεθοδολογία αυτή συνήθως σκοπεύει να επαληθεύσει τη σχεδίαση και όχι την υλοποίηση (αν και μπορεί να γίνει και αυτό).

Μία ιδιότητα ενός συστήματος θα ονομάζεται *αμετάβλητη (invariant)* αν ισχύει σε κάθε προσβάσιμη κατάσταση του συστήματος. Ακολουθώντας το συμβολισμό από την προηγούμενη ενότητα, μία invariant ορίζεται ως εξής [24]:

$$\text{invariant } p \stackrel{\text{def}}{=} (\forall u \in I.p(u)) \wedge (\forall u \in R_s.\forall \tau \in T.(p(u) \Rightarrow p(\tau(u)))),$$

που σημαίνει ότι το κατηγόρημα p είναι αληθές σε κάθε προσβάσιμη κατάσταση του S. Για να αποδείξουμε ότι μία ιδιότητα είναι αμετάβλητη με τη μέθοδο των Proof Scores, χρησιμοποιούμε τέσσερα βήματα:

1. Πρώτα ορίζουμε την ιδιότητα που θέλουμε να αποδείξουμε σαν κατηγόρημα σε ένα module της CafeOBJ.

2. Γράφουμε το επαγωγικό βήμα σαν κατηγόρημα που περιέχει δύο καταστάσεις του συστήματος, την s και την s', που συμβολίζουν μία τυχαία κατάσταση του συστήματος και τη διάδοχή της. Αν υποθέσουμε ότι η ιδιότητα ισχύει για την κατάσταση s, θα πρέπει να δειχτεί ότι ισχύει και για την s'.

3. Χρησιμοποιώντας την εντολή `reduce`, η CafeOBJ κάνει αναγραφές όρων και εξετάζει αν η ιδιότητα ισχύει για μία τυχαία αρχική κατάσταση.

4. Τέλος, χρησιμοποιώντας όλους τους κανόνες μετάβασης, ορίζουμε την κατάσταση s' και ζητάμε από την CafeOBJ να αποδείξει το επαγωγικό βήμα γιά κάθε περίπτωση. Το βήμα αυτό μπορεί να μας δώσει τρία πιθανά αποτελέσματα:

   (α΄) Αν μας επιστρέψει *true* τότε η απόδειξη μας ήταν επιτυχημένη.

   (β΄) Αν μας επιστρέψει κάποια έκφραση αντί για *true* ή *false* τότε αυτό σημαίνει πως υπάρχουν κάποιες εκφράσεις που δεν μπόρεσαν να αποτιμηθούν. Στην περίπτωση αυτή ο χρήστης σπάει την περίπτωση σε δύο υποπεριπτώσεις δηλώνοντας πως η προβληματική έκφραση είναι πρώτα αληθής και μετά ψευδής. Έτσι δημιουργούνται δύο νέα κομμάτια που πρέπει να αποδειχτούν, με την ίδια μέθοδο. Αυτή η διαδικασία ονομάζεται *case splitting*.

   (γ΄) Αν μας επιστρέψει *false* τότε ή η ιδιότητα δεν ισχύει ή η κατάσταση του συστήματος που επέστρεψε το false είναι μη προσβάσιμη. Αν ισχύει το πρώτο τότε έχουμε βρει μία περίπτωση που η ιδιότητα αποτυγχάνει και θα πρέπει να βελτιώσουμε τη σχεδίαση του συστήματος. Στη δεύτερη περίπτωση ορίζουμε ένα λήμμα, μια νέα αμετάβλητη ιδιότητα που δηλώνει ότι η περίπτωση που επέστρεψε το false είναι μη-προσβάσιμη.

Ένα παράδειγμα χρήσης της μεθοδολογίας των Proof Scores βρίσκεται στο [28] και φαίνεται στο Παράδειγμα 2.8.

Παράδειγμα 2.8: Παράδειγμα ενός proof score [28]

Ξεκινάμε με την προδιαγραφή (Κώδικας 3) που έχουμε για τους ακέραιους αριθμούς και στην οποία έχουμε ορίσει το μηδέν, το διάδοχο ενός ακεραίου και την πρόσθεση.

```
mod! SIMPLE-NAT {
  [ Nat ]
  op 0 : -> Nat
  op s : Nat -> Nat
}

mod! NAT+ {
  pr(SIMPLE-NAT)
  op _+_ : Nat Nat -> Nat
  eq 0 + M:Nat = M .
  eq s (N:Nat) + M:Nat = s (N + M) .
}
```

Κώδικας 3: Ένα απλό module στην CafeOBJ για τους ακέραιους αριθμούς

Η ιδιότητα που θέλουμε να αποδείξουμε είναι η προσεταιριστικότητα της πρόσθεσης, ότι δηλαδή $\forall i, j, k \in NAT : (i + j) + k = i + J + k)$. Θα γράψουμε το proof score που φαίνεται στον Κώδικα 4, θα το αποθηκεύσουμε και θα το εκτελέσουμε στην CafeOBJ.

```
1 open NAT+ + EQL
2
3 ops i j k : -> Nat .
4
5 red ((0 + j) + k) = (0 + (j + k)) .
6 eq (i + j) + k = i + (j + k) .
7 red ((s(i) + j) + k) = (s(i) + (j + k)) .
8
9 close
```

Κώδικας 4: Το Proof Score για την απόδειξη της προσεταιριστικής ιδιότητας της πρόσθεσης

Στη γραμμή 3 δηλώνουμε τρεις τυχαίες μεταβλητές τύπου Nat. Η γραμμή 5 αποτελεί το βασικό βήμα που λογικά θα βγει αληθές. Η γραμμή 6 περιέχει την επαγωγική υπόθεση και η γραμμή 7 περιέχει το επαγωγικό βήμα που θέλουμε να βγει αληθές.
Η εκτέλεση του proof score φαίνεται στον Κώδικα 5.

```
CafeOBJ> in simple−nat−proof−assoc.mod
processing input : /Users/konstantinos/cafeob/simple−nat.mod_*
−− opening module NAT+ + EQL.. done._*
−− reduce in %NAT+ + EQL : (0 + j) + k = 0 + (j + k)
true : Bool
(0.003 sec for parse, 3 rewrites(0.002 sec), 11 matches)
−−> should be true.*
−− reduce in %NAT+ + EQL : (s(i) + j) + k = s(i) + (j + k)
true : Bool
(0.004 sec for parse, 5 rewrites(0.003 sec), 26 matches)
−−> should be true.
CafeOBJ>
```

Κώδικας 5: Η απάντηση της CafeOBJ στο proof score

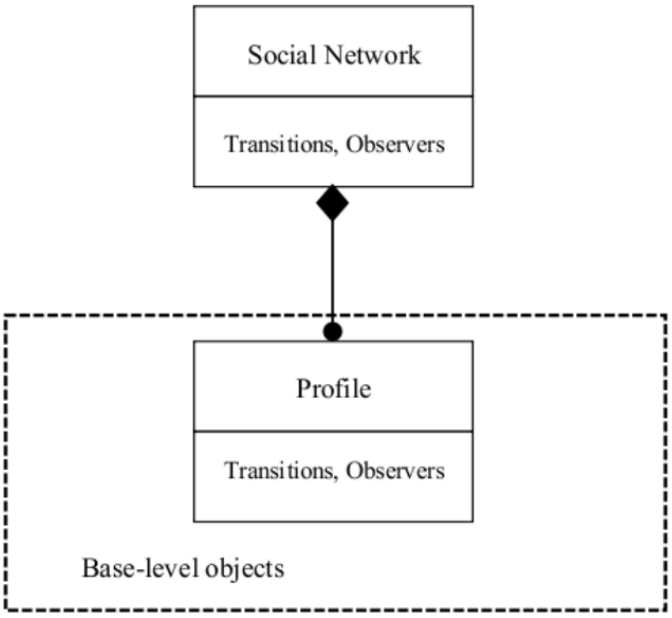
Παρατηρούμε ότι πήραμε true και στα δύο reductions (βασικό και επαγωγικό βήμα) άρα η ιδιότητα που θέλαμε αποδείχτηκε.

2.3.4 Σύνθεση Συμπεριφοριακών Αντικειμένων

Η σημασιολογία της συμπεριφοριακής προδιαγραφής είναι βασισμένη στη Κρυφή Άλγεβρα [22, 29, 30], η οποία είναι μια εκλέπτυνση της γενικής άλγεβρας με πολλούς τύπους [26]. Η Κρυφή Άλγεβρα επεκτείνει τη γενική άλγεβρα με επιπλέον τύπους που αναπαριστούν "καταστάσεις" ενός αντικειμένου ή ενός αφηρημένου συστήματος.

Η μεθοδολογία της σύνθεσης συμπεριφοριακών αντικειμένων έχει οριστεί επίσημα στο [26] και μπορεί να εξαχθεί σε κάθε προδιαγραφή και γλώσσα επαλήθευσης που εφαρμόζει συμπεριφοριακή λογική. Η μεθοδολογία αυτή είναι ιεραρχική αφού η σύνθεση συμπεριφοριακών αντικειμένων δίνει και αυτή ένα συμπεριφοριακό αντικείμενο που με τη σειρά του μπορεί να χρησιμοποιηθεί σε άλλη σύνθεση.

Τα μη σύνθετα αντικείμενα (δηλαδή τα αντικείμενα χωρίς δομικά στοιχεία) ονομάζονται *βασικά αντικείμενα*, *base-level objects*. Μια σύνθεση εκφράζεται με βέλη (Σχήμα 3) που έχουν ρόμβους στην άκρη τους και, αν είναι απαραίτητο, ένα σύμβολο που υποδηλώνει τον αριθμό των δομικών στοιχείων του· 1 για ένα στοιχείο και * για πολλά. Επίσης, ο κύκλος στην ουρά του βέλους υποδηλώνει ότι το σύνθετο αντικείμενο περιέχει έναν αόριστο αριθμό δομικών στοιχείων. Μπορούμε να πάρουμε την κατάσταση των δομικών αντικειμένων μέσω τελεστών προβολής [31, 32], οι οποίοι είναι ειδικοί παρατηρητές ενός σύνθετου αντικειμένου που δοσμένης μιας κατάστασής του, επιστρέφουν την κατάσταση ενός



Σχήμα 3: Σύνθεση συμπεριφοριακών αντικειμένων

βασικού αντικειμένου του. Υπάρχουν διάφοροι τρόποι να συνθέσουμε ένα αντικείμενο από δομικά στοιχεία. Για παράδειγμα, η Παράλληλη Σύνθεση χωρίς Συγχρονισμό εφαρμόζεται όταν οι αλλαγές στην κατάσταση ενός αντικειμένου δεν επηρεάζουν την κατάσταση αντικειμένων του ίδιου επιπέδου. Η Παράλληλη Σύνθεση με Συγχρονισμό εφαρμόζεται όταν οι αλλαγές στην κατάσταση ενός αντικειμένου μπορούν να αλλάξουν την κατάσταση κάποιου αντικειμένου που βρίσκεται στο ίδιο επίπεδο. Ανάλογα με τον αριθμό των αντικειμένων που συνθέτουν ένα σύνθετο αντικείμενο, έχουμε τη Δυναμική σύνθεση, αν ο αριθμός αυτό δεν είναι σταθερός, αλλιώς έχουμε τη Στατική Σύνθεση.

2.3.5 Χρονικά Συστήματα Παρατηρήσεων Μεταβάσεων

Αν σε ένα OTS (βλ. Κεφάλαιο 2.3.2) εισάγουμε ειδικούς παρατηρητές (που ονομάζονται "ωρολογιακοί παρατηρητές" ή "ρολόγια") που επιστρέφουν πραγματικούς αριθμούς, τότε το OTS αυτό μπορεί να μοντελοποιήσει συστήματα πραγματικού χρόνου και να χρησιμοποιηθεί στη μελέτη ιδιοτήτων συστήματος με χρονικούς περιορισμούς. Ένα τέτοιο OTS αποκαλείται *Χρονικό Σύστημα Παρατηρήσεων Μεταβάσεων*

ή *Timed Observational Transition System (TOTS)* [33]. Ένα TOTS είναι μία επέκταση του OTS με την προσθήκη παρατηρητών που χειρίζονται το χρόνο.

Υποθέτουμε ότι υπάρχει ένας χώρος καταστάσεων Y και ότι έχουμε ήδη ορίσει τους τύπους δεδομένων που χρησιμοποιούμε, συμπεριλαμβανόμενου της ισοδυναμίας μεταξύ δύο τιμών u_1, u_2 του τύπου δεδομένων που ορίζεται με το $u_1 = u_2$. Έστω ότι B, N, R^+ ένα σύνολο αληθοτιμών, ένα σύνολο φυσικών αριθμών και ένα σύνολο μη-αρνητικών πραγματικών αριθμών αντίστοιχα.

Ορισμός 1. TOTS: Ένα TOTS S αποτελείται από $\langle O, I, T \cup \{tick_r | r \in R^+\}\rangle$, όπου:

- O : Ένα σύνολο παρατηρητών. Το σύνολο $O = D \cup C$ ανήκει στο σύνολο D των διακριτών παρατηρητών και το σύνολο C ανήκει στο σύνολο των ωρολογιακών παρατηρητών (ή ρολογιών). Κάθε $o \in O$ είναι μία συνάρτηση $o : Y \to D$, όπου D είναι κάποιος τύπος δεδομένων και μπορεί να διαφέρει από παρατηρητή σε παρατηρητή. Δοθέντος δύο καταστάσεων $u_1, u_2 \in Y$, η ισοδυναμία μεταξύ δύο καταστάσεων, όπως αυτή ορίζεται με το $u_1 =_s u_2$ κατά το S ορίζεται ως $\forall o \in O.o(u_1) = o(u_2)$.

- I: Το σύνολο των αρχικών καταστάσεων του συστήματος έτσι ώστε $I \subseteq Y$.

- $T \cup \{tick_r | r \in R^+\}$: Ένα σύνολο υπό-συνθήκη μεταβάσεων. Κάθε $\tau \in T \cup \{tick_r | r \in R^+\}$ είναι μία συνάρτηση $\tau : Y \to Y$, τέτοια ώστε $\tau(u_1) =_s \tau(u_2)$ για κάθε $[u] \in Y/ =_s$ και κάθε $u_1, u_2 \in [u]$. Το $\tau(u)$ ονομάζεται διάδοχη κατάσταση του $u \in Y$ κατά το τ. Η συνθήκη του τ ονομάζεται αναγκαία συνθήκη. Κάθε τ πρέπει να ικανοποιεί την απαίτηση $u =_s \tau(u)$ για κάθε $u \in Y$ έτσι ώστε η αναγκαία συνθήκη του τ να είναι ψευδής στο u.

Για κάθε ρολόι, το D είναι ένα υποσύνολο (υπότυπος) του $R^+ \cup \{\infty\}$. Για κάθε $\tau \in T$, υπάρχουν δύο ρολόγια $l_\tau : Y \to R^+$ και $u_\tau : Y \to (R^+ \setminus \{0\} \cup \{\infty\})$, που επιστρέφουν το κάτω και άνω άκρο του τ αντίστοιχα. Χρησιμοποιούνται είτε για να αναγκάσουν το τ να εκτελεστεί ή ανάμεσα στο κάτω άκρο που επιστρέφει το l_τ και το άνω άκρο που επιστρέφει το u_τ. Επίσης, υπάρχει ένα ειδικό ρολόι, το $now : Y \to R^+$, που χρησιμοποιείται ως το κυρίως ρολόι και επιστρέφει το χρόνο που πέρασε μετά την εκτέλεση του S. Το now αρχικά επιστρέφει 0. Το C περιέχει τα δύο ρολόγια l_τ και u_τ για κάθε $\tau \in T$ και το κυρίως ρολόι now. Η αναγκαία συνθήκη για το κάθε $\tau \in T$ αποτελείται από

το μη-χρονικό κομμάτι (συμβολίζεται ως c_τ) και από το χρονικό κομμάτι. Δοθείσης μιας κατάστασης $u \in Y$, η χρονική αναγκαία συνθήκη είναι $l_\tau \leq now(u)$. Κάθε $tick_r$ είναι μια χρονική μετάβαση. Δοθείσης μιας κατάστασης $u \in Y$, για κάθε $tick_r$, η αναγκαία συνθήκη του είναι $now(u) + r \leq u_\tau(u)$ για κάθε $\tau \in T$, και $now(tick_r(u))$ είναι $now(u) + r$ αν η αναγκαία συνθήκη του $tick_r$ ισχύει στο u. Το $tick_r$ δεν επηρεάζει την τιμή που επιστρέφει κάποιος παρατηρητής εκτός του now, και η τιμή που επιστρέφεται από το now επηρεάζεται μόνο από από τις χρονικές μεταβάσεις.

Για κάθε $\tau \in T$, πέρα από τα δύο ρολόγια l_τ και u_τ, έχουμε δύο συναρτήσεις $d_\tau^{min}, d_\tau^{max}$ με τύπους ίδιους με τα l_τ και u_τ αντίστοιχα. Οι συναρτήσεις αυτές επιστρέφουν τις ελάχιστες και μέγιστες καθυστερήσεις του τ αντίστοιχα, οι οποίες χρησιμοποιούνται όπως και οι τιμές που επιστρέφουν τα l_τ και u_τ. Θα περιγράψουμε πώς ορίζονται οι αρχικές τιμές των l_τ και u_τ και πώς τροποποιούνται οι τιμές που επιστρέφουν τα l_τ και u_τ όταν κάποιο $\tau' \in T$ εφαρμόζεται σε κάποια κατάσταση στην οποία ισχύει η αναγκαία συνθήκη του.

- Έστω $init$ η αρχική κατάσταση του S:

$$l_\tau(init) = \begin{cases} d_\tau^{min}(init), & \text{αν } c_\tau(init) \\ 0, & \text{αλλιώς} \end{cases}$$

$$u_\tau(init) = \begin{cases} d_\tau^{max}(init), & \text{αν } c_\tau(init) \\ \infty, & \text{αλλιώς} \end{cases}$$

- Υποθέτουμε ότι κάποιο τ' εφαρμόζεται σε μια κατάσταση $u \in Y$ έτσι ώστε $c_{\tau'}(u)$ και $l_{\tau'}(u) \leq now(u)$ και έστω ότι u' είναι $\tau'(u)$.
  - Αν το τ' είναι ίδιο με το τ:

$$l_\tau(u') = \begin{cases} now(u) + d_\tau^{min}(u), & \text{αν } c_\tau(u') \\ 0, & \text{αλλιώς} \end{cases}$$

$$u_\tau(u') = \begin{cases} now(u) + d_\tau^{max}(u), & \text{αν } c_\tau(u') \\ 0, & \text{αλλιώς} \end{cases}$$

  - Αν το τ' δεν είναι ίδιο με το τ:

$$l_\tau(u') = \begin{cases} now(u) + d_\tau^{min}(u), & \text{αν } \neg c_\tau(u) \wedge c_\tau(u') \\ 0, & \text{αν } c_\tau(u) \wedge \neg c_\tau(u') \\ l_\tau(u), & \text{αλλιώς} \end{cases}$$

$$l_\tau(u') = \begin{cases} now(u) + d_\tau^{max}(u), & \text{αν } \neg c_\tau(u) \wedge c_\tau(u') \\ \infty, & \text{αν } c_\tau(u) \wedge \neg c_\tau(u') \\ u_\tau(u), & \text{αλλιώς} \end{cases}$$

Αν το d_τ^x επιστρέφει σταθερή τιμή σε κάθε κατάσταση u τότε το d_τ^x μπορεί να χρησιμοποιηθεί για να εκφράσει τη σταθερή τιμή αντί για το $d_\tau^x(u)$, όπου $x = min, max$. □

Ορισμός 2. Εκτέλεση: Μία εκτέλεση του S είναι μία άπειρη διαδοχή καταστάσεων $u_1, u_2, \cdots, u_i, \cdots$ που ικανοποιούν:

- **Αρχικοποίηση**: $u_0 \in I$
- **Διαδοχή**: Για κάθε $i \in N$, υπάρχει $\tau \in T \cup \{tick_r | r \in R^+\}$ έτσι ώστε $u_{i+1} =_s \tau(u_i)$
- **Χρονική απόκλιση**: Καθώς το i αυξάνεται, το $now(u_i)$ αυξάνεται χωρίς κάποιο όριο

Έστω ϵ_S το σύνολο όλων των εκτελέσεων που παίρνουμε από το S. □

Από τον Ορισμό 2 μπορούμε να πάρουμε μία τέτοια εκτέλεση $u_0, .., u_i, u_{i+1}, ...$ έτσι ώστε το u_{i+1} να είναι $\tau(u_i)$, το $c_\tau(u_i)$ να μην ισχύει και έτσι το u_{i+1} να είναι u_i. Για να αποκλείσουμε μία τέτοια εκτέλεση μπορούμε να τροποποιήσουμε τη Διαδοχή έτσι ώστε $\forall i \in N, \exists\ \tau \in T \cup \{tick_r | r \in R^+\} : u_{i+1} =_S \tau(u_i)$ και το $c_\tau(u_i)$ ισχύει. Όμως, προτιμούμε τον Ορισμό 2 γιατί είναι καταλληλότερος για την τυποποίηση των TOTSs στην CafeOBJ. Όπως θα πούμε παρακάτω, μία μετάβαση τ ορίζεται από έναν τελεστή μετάβασης α. Δοθείσης μιας κατάστασης u που ορίζεται από έναν όρο s, ο όρος $\alpha(s)$ υποδηλώνει την διάδοχη κατάσταση του u κατά το τ ανεξάρτητα από την αληθοτιμή του $c_\tau(u)$. Το ότι ο όρος $\alpha(s)$ είναι καλά-ορισμένος μόνο αν ισχύει το $c_\tau(u)$ δεν είναι ιδιαίτερα βολικό.

Μία κατάσταση $u \in Y$ εμφανίζεται στην εκτέλεση $u_0, u_1, \ldots$ του S ($u \in u_0, u_1, \ldots$) ανν υπάρχει $i \in N : u =_S u_i$.

Ορισμός 3. Προσβάσιμη Κατάσταση: Μία κατάσταση $u \in Y$ ονομάζεται προσβάσιμη ως προς το S ανν υπάρχει μία εκτέλεση $e \in \epsilon_S : u \in e$. Έστω R_S το σύνολο όλων των προσβάσιμων καταστάσεων ως προς το S. □

Ορισμός 4. Αμετάβλητο: Ένα κατηγόρημα $p : Y \to B$ ονομάζεται αμετάβλητο ως προς το $S(invariant_S p)$, ανν $\exists u \in R_{S.p(u)}$. Το S μπορεί να παραληφθεί από το $invariant_S p$ αν είναι σαφές από το περιεχόμενο. □

Έστω $x_1, x_2, \ldots$ (με τύπους $D_1, D_2, \ldots$ αντίστοιχα) ελεύθερες μεταβλητές στο $S(invariant_Sp)$. Υποθέτουμε ότι το $invariant_Sp$ μεταφράζεται ως $\forall x_1 \in D_1.\forall x_2 \in D_2 \ldots (invariant_Sp)$. Όταν γράφουμε ένα proof score για τον τύπο αυτόν, οι ελεύθερες μεταβλητές αντικαθιστώνται με σταθερές που υποδηλώνουν αυθαίρετες τιμές και οι καθολικοί ποσοτικοποιητές απαλείφονται.

Ένα TOTS ονομάζεται μη-μηδενικό αν οποιαδήποτε πεπερασμένη ακολουθία καταστάσεων που παράγει το TOTS μπορεί να επεκταθεί σε εκτέλεση. Μία ικανοποιητική προϋπόθεση για ένα μη-μηδενικό TOTS είναι για κάθε $\tau \in T$ και κάθε κατάσταση $u \in Y$, $l_\tau(u) \leq u_\tau(u)$ και συγκεκριμένα $d_\tau^{min}(u) \leq d_\tau^{max}(u)$. Για κάθε $\tau \in T$ αν το d_τ^{min} επιστρέφει πάντα 0, το l_τ μπορεί να παραληφθεί από το O και αν το d_τ^{max} επιστρέφει πάντα ∞, το u_τ μπορεί να παραληφθεί από το O.

Οι παρατηρητές και οι μεταβάσεις μπορούν να παραμετροποιηθούν, όπως το $tick_r$. Οι παρατηρητές και οι μεταβάσεις εκφράζονται ως $o_{i_1,\ldots i_m}$ και $\tau_{j_1,\ldots j_n}$ αντίστοιχα, εφόσον $m, n \geq 0$ και εφόσον υπάρχει ένας τύπος D_k, τέτοιος ώστε $k \in D_k$, όπου $k = i_1, \ldots, i_m, j_1, \ldots, j_n$.

2.3.5.1 Προδιαγραφή των TOTSs στην CafeOBJ

Ένα TOTS είναι ένα OTS [33] το οποίο περιέχει ένα επιπρόσθετο module (TIMEVAL), το οποίο προδιαγράφει μη-αρνητικούς πραγματικούς αριθμούς. Η υπογραφή του module αυτού φαίνεται στον Κώδικα 6.

```
[Zero NzReal+ < Real+]
[NzReal+ Inf < NzTimeval]
[Real+ NzTimeval < Timeval]
op 0 : -> Zero
op oo : -> Inf
op _+_ : Real+ Real+ -> Real+ {assoc comm}
op _+_ : Timeval Timeval -> Timeval {assoc comm}
op _<_ : Timeval Timeval -> Bool
op _<=_ : Timeval Timeval -> Bool
op _=_ : Timeval Timeval -> Bool {comm}
```

Κώδικας 6: Η υπογραφή του TOTS στη CafeOBJ

Οι ορατοί τύποι *Zero*, *NzReal*, *Real+*, *Inf*, *NzTimeval* και *Timeval* συμβολίζουν αντίστοιχα τα σύνολα $\{0\}, R^+ \setminus \{0\}, R^+, \{\infty\}, (R^+ \setminus \{0\}) \cup \{\infty\}$ και $R^+ \cup \{\infty\}$. Οι σταθερές *0* και *oo* συμβολίζουν το 0 και το ∞ αντίστοιχα. Ο τελεστής $+$ προσθέτει δύο θετικούς πραγματικούς αριθμούς, ο τελεστής $<$ ελέγχει εάν ένας θετικός πραγματικός αριθμός είναι μικρότερος από κάποιον άλλο, ο τελεστής $<=$ ελέγχει εάν ένας θετικός

πραγματικός αριθμός είναι μικρότερος ή ίσος με κάποιον άλλο και, τέλος, ο τελεστής $=$ ελέγχει εάν δύο θετικοί πραγματικοί αριθμοί είναι ίσοι.

Οι ιδιότητες των τελεστών προδιαγράφονται με εξισώσεις και το κατά πόσο θα χρησιμοποιήσουμε κάποιον τελεστή ή όχι εξαρτάται από το σύστημα που θέλουμε να προδιαγράψουμε. Για παράδειγμα, στον Κώδικα 7 φαίνονται οι ορισμοί μερικών τελεστών, όπου οι X και Y είναι μεταβλητές τύπου $Real+$ και $T, T1, T2$ μεταβλητές τύπου *Timeval*.

```
eq (X < 0) = false .
eq (X < oo) = true .
eq (oo < X) = false .
eq (X < X) = false .
ceq (T1 < T2) = false if T2 <= T1 .
ceq (X+T1 < X + T2) = true if T1 < T2 .
ceq (T < T1 + T2) = true if T < T2 .
eq (0 <= X) = true .
eq (X <= oo) = true .
eq (oo <= X) = false .
eq (X <= X) = true .
eq (X <= X + T) = true .
ceq (T1 + T2 <= T1) = false if 0 < T2 .
ceq (T1 <= T2) = false if T2 < T1 .
```

Κώδικας 7: Ορισμοί τελεστών του TOTS

Με αυτές τις εξισώσεις, το module `TIMEVAL` περιέχει αρκετές πληροφορίες για να χρησιμοποιηθεί τόσο για την προδιαγραφή συστημάτων πραγματικού χρόνου όσο και για την διερεύνηση ιδιοτήτων τέτοιων συστημάτων.

Σε αντιστοιχία με την Παράγραφο 2.3.2, δοθέντος ενός τύπου δεδομένων D_k (ή D), έστω V_k (ή V) ένας ορατός τύπος που αντιστοιχεί στον τύπο δεδομένων και έστω X_k (ή X) μία μεταβλητή της CafeOBJ με τύπο V_k (ή V). Ο χώρος καταστάσεων Y συμβολίζεται με έναν κρυφό τύπο, H. Ένας παρατηρητής $o_{i_1,\ldots,i_m} \in O$ δηλώνεται με έναν παρατηρητικό τελεστή της CafeOBJ, η δήλωση του οποίου είναι η εξής:

$$bop\ o\ :\ H\ V_{i_1} \ldots V_{i_m} \rightarrow V$$

Η αρχική κατάσταση (στο) συμβολίζεται με μία σταθερά, `init`, η οποία δηλώνεται ως εξής:

$$op\ init :\rightarrow H$$

Υποθέτουμε ότι η αρχική τιμή του $o_{i_1,\ldots,i_m}$ είναι $f(i_1, \ldots, i_m)$. Αυτό δηλώνεται ως εξής:

$eq\ o(init, X_{i_1}, \ldots, X_{i_m}) = f(X_{i_1}, \ldots, X_{i_m})$., όπου $f(X_{i_1}, \ldots, X_{i_m})$ είναι όρος της CafeOBJ που υποδηλώνει το f.

Μία μετάβαση $\tau_{j_1,\ldots,j_n} \in T \cup \{tick_r | r \in R^+\}$ δηλώνεται με έναν μεταβατικό τελεστή της CafeOBJ, ο οποίος δηλώνεται ως εξής:

$$bop\ \alpha\ :\ H\ V_{j_1} \ldots V_{j_n} \to H$$

Η μετάβαση $\tau_{j_1,\ldots,j_n}$ μπορεί να αλλάξει την τιμή που επιστρέφει ο $o_{i_1,\ldots,i_m}$ αν εφαρμοστεί σε μία κατάσταση στην οποία ισχύει η αναγκαία συνθήκη του. Αυτό γράφεται ως εξής:

$$ceq\ o(\alpha(S, X_{j_1}, \ldots, X_{j_n}), X_{i_1}, \ldots, X_{i_m}) = e{-}\alpha(S, X_{j_1}, \ldots, X_{j_n}, X_{i_1}, \ldots, X_{i_m})$$

$$if\ c - \alpha(S, X_{j_1}, \ldots, X_{j_n}).$$

Το S είναι μια μεταβλητή της CafeOBJ τύπου H. Το $\alpha(S, X_{j_1}, \ldots, X_{j_n})$ υποδηλώνει τη διάδοχη κατάσταση του S μετά την εφαρμογή του $\tau_{j_1,\ldots,j_n}$. Το $e - \alpha(S, X_{j_1}, \ldots, X_{j_n}, X_{i_1}, \ldots, X_{i_m})$ υποδηλώνει την τιμή που επιστρέφει ο $o_{i_1,\ldots,i_m}$ στη διάδοχη κατάσταση. Το $c - \alpha(S, X_{j_1}, \ldots, X_{j_n})$ υποδηλώνει την αναγκαία συνθήκη του $\tau_{j_1,\ldots,j_n}$.

Το $\tau_{j_1,\ldots,j_n}$ δεν αλλάζει κάτι αν εφαρμοσθεί σε μία κατάσταση που δεν ισχύει η αναγκαία συνθήκη του, κάτι που γράφεται ως εξής:

$$ceq\ \alpha(S, X_{j_1}, \ldots, X_{j_n}) = S \text{ if not } c - \alpha(S, X_{j_1}, \ldots, X_{j_n}).$$

Αν η τιμή που επιστρέφει ο $o_{i_1,\ldots,i_m}$ δεν επηρεάζεται από την εφαρμογή του $\tau_{j_1,\ldots,j_n}$ σε οποιαδήποτε κατάσταση (ανεξάρτητα από το αν ισχύει ή όχι η αναγκαία συνθήκη του) δηλώνεται ως εξής:

$$ceq\ o(\alpha(S, X_{j_1}, \ldots, X_{j_n}), X_{i_1}, \ldots, X_{i_m}) =\ o(S, X_{i_1}, \ldots, X_{i_m}).$$

Η απόδειξη ιδιοτήτων συστημάτων που μπορούν να περιγραφούν ως TOTS γίνεται με τη μέθοδο των proof scores, όπως αυτή περιγράφηκε στην Παράγραφο 2.3.3.

2.4 Πρότυπα

Ένα πρότυπο είναι ένα καθιερωμένο σύνολο κανόνων ή απαιτήσεων για ένα σύστημα. Συνήθως είναι ένα επίσημο έγγραφο που θεσπίζει τεχνικά κριτήρια, μεθόδους, διαδικασίες και πρακτικές. Ο όρος "πρότυπο" έχει και πιο συγκεκριμένες ερμηνείες όταν αναφέρεται σε ένα συγκεκριμένο σύστημα [34]: Η μορφή αρχείου που ένα πρόγραμμα εισάγει ή εξάγει, το μέγεθος και ο τύπος των βιδών που μπαίνουν σε μία

πλακέτα, οι ηλεκτρικές προδιαγραφές και ο τύπος βύσματος του φορτιστή ενός κινητού, κ.ά. Θα μπορούσε να θεωρηθεί ως μια συμφωνία που γίνεται ανάμεσα στον κατασκευαστή και στον καταναλωτή και αφορά ένα προϊόν [34, 35].

2.4.1 Χρήσεις των προτύπων

Τα πρότυπα μπορούν να χωριστούν σε τρεις κατηγορίες ανάλογα με τη χρήση που προορίζονται [34]:

1. Επιδόσεις: Τα πρότυπα αυτά προσδιορίζουν τους τρόπους εκτέλεσης διεργασιών έχοντας σα στόχο να εξασφαλίσουν ένα ελάχιστο επίπεδο ποιότητας προσδιορίζοντας είτε μια διαδικασία (π.χ. τα πρότυπα του ISO 9000 για την Ποιότητα της Διαχείρισης) είτε ένα επιθυμητό αποτέλεσμα, όπως για παράδειγμα ένα περιβάλλον εργασίας ασφαλές από τον κίνδυνο της πυρκαϊάς.

2. Μετρήσεις: Τα πρότυπα αυτά προσδιορίζουν μία αντικειμενική και μετρήσιμη μονάδα μέτρησης (π.χ. μέτρο, λίτρο, κ.τ.λ.), επιτρέποντας έτσι τη σύγκριση φυσικών ιδιοτήτων όπως το μήκος ή τον όγκο.

3. Συμβατότητα: Τα πρότυπα αυτά καθορίζουν τον τρόπο επικοινωνίας (και τη διαλειτουργικότητα) μεταξύ διαφορετικών και διακριτών αντικειμένων. Τα πρωτόκολλα Universal Serial Bus (USB) και Ethernet είναι τέτοια παραδείγματα αφού επιτρέπουν την επικοινωνία διαφορετικών εξαρτημάτων του υπολογιστή.

Η χρησιμότητα των προτύπων της τρίτης κατηγορίας είναι ότι επιτρέπουν τη σύνθεση διαφορετικών εξαρτημάτων για τη δημιουργία ενός συνεκτικού και ευέλικτου συστήματος [34].

2.4.2 Θέσπιση προτύπων

Τα πρότυπα συνήθως θεσπίζονται από κάποιον οργανισμό (Standards Developing Organisation - SDO ή standards setting Organisation - SSO) που δραστηριοποιείται στην ανάπτυξη, το συντονισμό, την αναθεώρηση, τροποποίηση, επανέκδοση και φυσικά στο σχεδιασμό τεχνικών προτύπων που απαντούν σε μία ανάγκη της αγοράς. Οι οργανισμοί αυτοί δε δημιουργούν απαραίτητα την τεχνολογία αλλά θεσπίζουν "συμφωνίες" για τη τεχνολογία. Τέτοιοι οργανισμοί είναι οι: AT&T (American Telephone and Telegraph Company), PTT (Postal, telegraph

and telephone service), British Engineering Standards Committee, American Standards Association (ASA), International Standards Organization, International Federation for Information Processing, και το Institute of Electrical and Electrotechnical Engineers.

Τα περισσότερα πρότυπα κατά βάση δεν επιβάλλονται βάσει νόμου αλλά προσφέρονται για χρήση από ανθρώπους ή βιομηχανίες. Κάποια πρότυπα καθίστανται υποχρεωτικά για κάποιον τομέα όταν υιοθετηθούν από νομικές ρυθμιστικές αρχές. Η υιοθέτηση ενός προτύπου γίνεται συνήθως με δύο τρόπους [34]:

1. *De jure*: Τα πρότυπα αυτά ορίζονται από τους κατάλληλους οργανισμούς και εφαρμόζονται από τις τοπικές / κρατικές / διεθνείς αρχές μέσω νομοθεσίας. Παραδείγματα προτύπων *de jure* είναι οι κανόνες προστασίας του περιβάλλοντος ή το πώς λειτουργούν τα δίκτυα τηλεφωνίας.

2. *De facto*: Τα πρότυπα αυτά ξεκινάνε ως πρωτόκολλα (ή τεχνολογίες) και μέσα από τη συχνή χρήση τους ή την αποδοχή της αγοράς γίνονται πρότυπα χωρίς απαραίτητα να έχουν εγκριθεί από κάποιον κατάλληλο οργανισμό - τουλάχιστον αρχικά. Δημιουργούνται είτε από εταιρίες ή ακόμα και μεμονωμένα άτομα και συνήθως διαδίδονται μέσω κάποιου σπόνσορα. Ένα παράδειγμα *de facto* προτύπου είναι το format αρχείων του Microsoft Word (.doc).

Η θέσπιση προτύπων στον κόσμο των υπολογιστών (software / ανάπτυξη πρωτοκόλλων) αναφέρεται στη διαδικασία της ανάπτυξης προτύπων για την ανταλλαγή δεδομένων μεταξύ συγκεκριμένων εφαρμογών χρησιμοποιώντας συγκεκριμένη σύνταξη. Ένα τέτοιο πρότυπο μπορεί να είναι μια προδιαγραφή, μια test method, ένας ορισμός, μία διαδικασία, μία πρακτική κ.ά. Τέτοια πρότυπα περιγράφονται συνήθως με χρήση φυσικών γλωσσών (Κεφάλαιο 2.1.4).

2.4.3 Ανοιχτά Πρότυπα

Μια υποκατηγορία των προτύπων είναι τα "Ανοιχτά Πρότυπα", πρότυπα που είναι διαθέσιμα στο κοινό και που μπορούν να έχουν διάφορα δικαιώματα χρήσης και διάφορες ιδιότητες σχεδιασμού. Ο ορισμός των ανοιχτών προτύπων μπορεί να είναι λίγο προβληματικός αφού ο όρος "ανοιχτό" ορίζεται δύσκολα: υπάρχουν διαφορετικοί ορισμοί που δίνονται από λεξικά, εθνικές υπηρεσίες IT, την Interoperable Delivery of European eGovernment Services to public Administrations, το Businesses

and Citizens (IDABC), τον World Trade Organization (WTO), τον Organization for the Advancement of Structured Information Standards (OASIS), το American National Standards Institute (ANSI), και άλλα. Από όλους αυτούς τους διαφορετικούς ορισμούς μπορούμε να εντοπίσουμε μερικά κοινά σημεία, κυρίως σε ό,τι έχει να κάνει με τα κίνητρα, την ανάπτυξη και τη χρήση τους. Ένα κίνητρο που συναντάμε στους περισσότερους ορισμούς είναι η στήριξη της διαλειτουργικότητας και προσπάθεια διάδοσης νέων τεχνολογιών.

Όσον αφορά στην ανάπτυξη, ένα ανοιχτό πρότυπο:

- αναπτύσσεται συνήθως από μια ανοιχτή διαδικασία στην οποία μπορεί να συμμετάσχει ο οποιοσδήποτε
- δέχεται προτάσεις και ιδέες [36, 37]
- δεν ελέγχεται ή δεσμεύεται από οποιοδήποτε group ή προμηθευτή

Αν και δεν υπάρχει βραχυπρόθεσμο όφελος από την ανάπτυξη και τήρηση ανοιχτών προτύπων, υπάρχουν μακροπρόθεσμα οφέλη (όπως για παράδειγμα αειφόρος ανάπτυξη, ανοικτοί δίαυλοι επικοινωνίας και διαλειτουργικότητα) που κάνουν τη χρήση τους συμφέρουσα.

2.4.4 Πλεονεκτήματα των Ανοιχτών Προτύπων

Τα πλεονεκτήματα της χρήσης των ανοιχτών προτύπων μπορούν να συνοψιστούν στα ακόλουθα [38, 39]:

- Η χρήση των ανοιχτών προτύπων ελευθερώνει τους χρήστες από την εξάρτηση σε μία συγκεκριμένη τεχνολογία ή σε μία εταιρία. Εφόσον οι προδιαγραφές του προτύπου είναι ανοιχτές και γνωστές είναι εύκολο για κάποια άλλη εταιρία ή οργανισμό να δημιουργήσουν μία παρόμοια εφαρμογή που να ακολουθεί το συγκεκριμένο πρότυπο.
- Η χρήση ανοιχτών προτύπων κάνει πιο εύκολη την επικοινωνία μεταξύ διαφορετικών συστημάτων παρά το ότι το κάθε σύστημα μπορεί να χρησιμοποιεί διαφορετική υλοποίηση του προτύπου. Για παράδειγμα, αν έχουμε μια μεγάλη εταιρία που απαιτεί τα αρχεία της να είναι γραμμένα στη μορφή του Open Document, τότε κάθε γραφείο μπορεί να διαλέξει όποια (από τις πολλές διαθέσιμες) εφαρμογή προτιμάει χωρίς να εμποδίζει την επικοινωνία του με τα υπόλοιπα γραφεία της εταιρίας.

- Η χρήση τους παρέχει μια προστασία ενάντια στις εφαρμογές που σταματάνε να αναπτύσσονται ή καταργούνται. Αν η μορφή του αρχείου δεδομένων είναι σε μη-ανοιχτό πρότυπο και καταργηθεί τότε οι χρήστες μπορεί να δυσκολευτούν να μετατρέψουν τα δεδομένα τους σε κάποια διαφορετική μορφή αρχείου ώστε να ανοίξουν σε νέα εφαρμογή. Αν όμως η μορφή του αρχείου δεδομένων είναι σε ανοιχτό πρότυπο τότε ακόμα και αν η εφαρμογή που τα ανοίγει καταργηθεί θα είναι εύκολο είτε να ανοιχτούν από άλλη εφαρμογή συμβατή με τη μορφή αυτή είτε να μετατραπούν σε διαφορετική μορφή.

  Το θέμα με τη μορφή αποθήκευσης αρχείων δεδομένων είναι ακόμα πιο σημαντικό για μεγάλες εταιρίες/οργανισμούς. Ένα παράδειγμα είναι οι ανάγκες των κυβερνήσεων ή των αστυνομικών τμημάτων μιας χώρας [40] ή οι κρατικοί ηλεκτρονικοί φάκελοι υγείας [41].

- Τα ανοιχτά πρότυπα συνοδεύονται από την προδιαγραφή τους, γραμμένη σε φυσική γλώσσα (Κεφάλαιο 2.1.4). Η προδιαγραφή αυτή είναι το σύνολο των απαιτήσεων, όπως αυτές έχουν οριστεί από τον οργανισμό που σχεδίασε το πρότυπο, που πρέπει να ικανοποιεί κάθε υλοποίηση του προτύπου [42]. Η προδιαγραφή αυτή είναι διαθέσιμη στο κοινό, επιτρέποντας στον κάθε ενδιαφερόμενο να το εξετάσει.

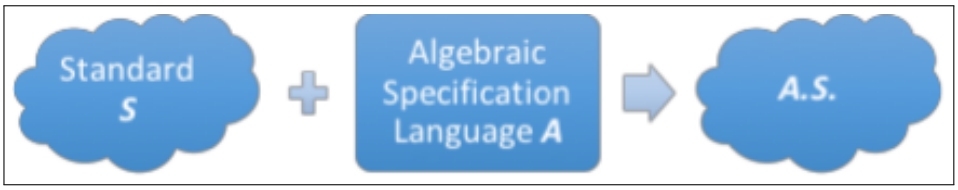
2.4.5 Προβλήματα των Ανοιχτών Προτύπων

Στον αντίποδα των πλεονεκτημάτων της χρήσης ανοιχτών προτύπων που είδαμε, υπάρχουν τα εξής προβλήματα: Η όλη διαδικασία της αναθεώρησης, αξιολόγησης και υλοποίησης ενός ανοιχτού προτύπου είναι δαπανηρή σε χρόνο, χρήματα, εξειδικευμένες γνώσεις, τεχνογνωσία, κ.λπ. Η διαδικασία αυτή (ανάλογα και με τον οργανισμό θέσπισης προτύπων που ασχολείται με ένα πρότυπο) μπορεί να πάρει από λίγους μήνες μέχρι αρκετά χρόνια. Αυτό είναι σε αντιδιαστολή με την ιδέα της καινοτομίας: συνήθως εμφανίζεται σε επαγγελματικούς στόχους όταν υπάρχει περιορισμός χρόνου. Επίσης, η διαλειτουργικότητα δεν είναι προσόν μόνο των ανοιχτών προτύπων. Υπάρχουν παραδείγματα μη-ανοιχτών προτύπων που έχουν κυκλοφορήσει με όρους Reasonable And Non-Discriminatory (RAND) και δεν έχουν εμποδίσει την διαλειτουργικότητα (π.χ. GSM, CD, DVD, MPEG, Wi-Fi, κ.ά.) [38, 39].

3 Τυποποίηση Προτύπων

Η ιδέα που παρουσιάζεται σε αυτή τη διατριβή είναι η χρήση τυπικών μεθόδων (Κεφάλαιο 2.2) για να δημιουργηθεί μία τυπική προδιαγραφή ενός (ανοιχτού) προτύπου (Κεφάλαιο 2.4)· ένα Τυπικό (Ανοιχτό) Πρότυπο. Η τυπική προδιαγραφή που προκύπτει είναι αρκετά διαφορετική από μια προδιαγραφή που γράφεται σε κάποια φυσική γλώσσα (Κεφάλαιο 2.1.4) και υποστηρίζουμε ότι τα πλεονεκτήματα αυτής της μεθόδου υπερτερούν των μειονεκτημάτων της. Στηρίζουμε τη θέση ότι τα (ανοιχτά) πρότυπα θα πρέπει να συνοδεύονται από μία τέτοια προδιαγραφή κατά την κυκλοφορία του προτύπου στο κοινό (ή έστω με κάποια μικρή καθυστέρηση, όπως αναφέρεται στο [16]) και ότι η ανάπτυξη της τυπικής προδιαγραφής του προτύπου A πρέπει να γίνεται παράλληλα με την ανάπτυξη του ίδιου του προτύπου.

Η προτεινόμενη μέθοδος φαίνεται στο Σχήμα 4: Το βιομηχανικό πρότυπο S περιγράφεται με χρήση μιας αλγεβρικής γλώσσας προδιαγραφών (A) δημιουργώντας το $A.S$. Το $A.S$ είναι η τυπική προδιαγραφή του S. Στο επόμενο κεφάλαιο θα επιχειρηματολογήσουμε για το κατά πόσο η προδιαγραφή $A.S.$ υπερτερεί της προδιαγραφής που γράφεται σε φυσική γλώσσα.



Σχήμα 4: Προτεινόμενη μέθοδος μετασχηματισμού

3.1 Πλεονεκτήματα της μεθόδου

Μια τυπική περιγραφή ενός (ανοιχτού) προτύπου έχει τα ακόλουθα πλεονεκτήματα:

1. **Λιγότερα θέματα ασάφειας**: Όπως είπαμε στο Κεφάλαιο 2.1.4, οι προδιαγραφές γραμμένες σε τυπικές γλώσσες είναι ανεπίσημες και, σαν αποτέλεσμα της χρήσης φυσικής γλώσσας, συνήθως περιέχουν ασάφειες. Όσο προσεκτικοί και αν είμαστε κατά τη γραφή μιας προδιαγραφής σε τυπική γλώσσα, δε μπορούμε να είμαστε σίγουροι πως κάποιος που θα το διαβάσει για να καταλάβει το πρότυπο, θα καταλάβει αυτό ακριβώς που είχαμε στο μυαλό μας.

Η κατανόηση κειμένου εξαρτάται και από τα συμφραζόμενα και το γενικό πλαίσιο του προτύπου. Συνήθως το γενικό πλαίσιο εννοείται. Για παράδειγμα, η προδιαγραφή μιας ιατρικής συσκευής θα θεωρεί δεδομένο το ιατρικό υπόβαθρο για τον αναγνώστη. Το υπόβαθρο αυτό συνήθως δεν είναι καλά ορισμένο. Η χρήση φυσικής γλώσσας για την έκφραση απαιτήσεων κατά τη διάρκεια των διαφορετικών σταδίων κατά τον σχεδιασμού ενός προτύπου μπορεί να οδηγήσει σε παρερμηνείες [43, 5]. Η εφαρμογή (κάποιου βαθμού) φορμαλισμού ελαχιστοποιεί το πρόβλημα ([16]) καθώς αναγκάζει τους σχεδιαστές να κάνουν τις σωστές "ερωτήσεις" και αυξάνει το επίπεδο κατανόησης του προτύπου, κάτι επιθυμητό αν θέλουμε να φτάσουμε κάποιο επίπεδο ποιότητας (του λογισμικού).

Αν και είναι δύσκολο να αντικασταθεί η προδιαγραφή σε φυσική γλώσσα από μια προδιαγραφή σε τυπική γλώσσα, καθώς είναι πιο φυσικό για τους ανθρώπους να ξεκινάνε τη προδιαγραφή έτσι, η συμμετοχή της μπορεί να ελαχιστοποιηθεί. Έτσι, αντί να χρησιμοποιείται η φυσική γλώσσα σε κάθε στάδιο της ανάπτυξης ενός προτύπου (συγκέντρωση απαιτήσεων, ανάλυση, ανάπτυξη, υλοποίηση και τέλος, εγκατάσταση), μπορεί να χρησιμοποιηθεί για τη συλλογή των απαιτήσεων και μετά μόνο ως συμπλήρωμα της τυπικής προδιαγραφής, σε μορφή σχολίων. Η τυποποίηση μιας προδιαγραφής μειώνει τις ασάφειες καθώς προδιαγραφές που βασίζονται στα μαθηματικά μπορούν να ερμηνευθούν μόνο με έναν τρόπο ([11, 10]), το σωστό.

2. **Μέγεθος της προδιαγραφής**: Οι τυπικές προδιαγραφές των προτύπων (ή οποιουδήποτε πρωτοκόλλου) είναι σημαντικά πιο μικρές σε μέγεθος από αυτές που γράφονται σε φυσικές γλώσσες. Η ιδιότητα αυτή μπορεί να αποδοθεί στη φύση των μαθηματικών και στο επίπεδο ασάφειας που υποστηρίζουν οι τυπικές γλώσσες. Ένας άλλος παράγοντας που εξηγεί το μικρότερο μέγεθος είναι η ικανότητα επαναχρησιμοποίησης των modules· μία καλογραμμένη προδιαγραφή ενός module μπορεί να χρησιμοποιηθεί σε άλλα, μεγαλύτερα συστήματα.

3. **Πιο συμπαγής σχεδιασμός συστημάτων**: Το πόσο διαλειτουργικό είναι ένα πρότυπο εξαρτάται από την ακρίβεια με την οποία έχουν οριστεί οι απαιτήσεις του. Όσο καλύτερα έχουν αυτές προσδιοριστεί, τόσο ευκολότερο είναι να πάρουμε ένα πρότυπο ορθά ορισμένο. Η ανάγκη για ορθά ορισμένα πρότυπα είναι ακόμα μεγαλύτερη όταν τα πρότυπα αυτά είναι μέρη διαλειτουργικών συ-

στημάτων. Πρότυπα όπως τα συστήματα αρχείων, τα πρωτόκολλα δικτύου ή τα λειτουργικά συστήματα είναι πρότυπα τα οποία θα μπορούσαν να ωφεληθούν από την προτεινόμενη μέθοδο καθώς δεν υπάρχει αρκετή έρευνα για την παροχή ενός ελαχίστου συνόλου απαιτήσεων μεταξύ των προτύπων αυτών.

4. **Έλεγχος και επαλήθευση ιδιοτήτων**: Ανάλογα με το είδος της γλώσσας τυπικών προδιαγραφών που χρησιμοποιούμε, η τυπική προδιαγραφή που προκύπτει μπορεί να ελεγχθεί για την εγκυρότητά της (για το αν δηλαδή το πρότυπο κάνει αυτό που υποτίθεται ότι κάνει) και επίσης, μία υλοποίηση του προτύπου μπορεί να επαληθευτεί. Με τον όρο επαλήθευση εννοούμε το να βεβαιωνόμαστε ότι η υλοποίηση του προτύπου συμμορφώνεται πλήρως με τις προδιαγραφές του. Για παράδειγμα, θα μπορούσαμε να επαληθεύσουμε αν η υλοποίηση ενός προτύπου πληροί τις απαιτήσεις που είχαν εκφραστεί κατά το πρώτο στάδιο της ανάπτυξης του προτύπου. Οι μεθοδολογίες αλγεβρικών προδιαγραφών επιτρέπουν τέτοιου είδους διερεύνηση αφού τέτοιες προδιαγραφές είναι εκτελέσιμες. Στην περίπτωση που η προδιαγραφή δεν είναι εκτελέσιμη θα πρέπει να γίνει κάποιος μετασχηματισμός με κατάλληλη γλώσσα, εφόσον φυσικά αυτός γίνει αυτόματα και ορθά, αλλιώς δεν μπορούμε να μιλήσουμε για επαλήθευση [44].

Από την άλλη, το κόστος της εφαρμογής των Τυπικών Μεθόδων σε συστήματα (ή έστω, σε κομμάτια αυτών) είναι ακόμα υπό έρευνα. Ο Sommerville [4] ισχυρίζεται πως το κόστος δεν είναι υψηλότερο από ότι όταν χρησιμοποιούμε συμβατικές μεθόδους, είναι απλά κατανεμημένο διαφορετικά· περισσότερος χρόνος, χρήματα και ενέργεια δαπανώνται στα αρχικά στάδια της ανάπτυξης αντί για τα τελικά στάδια.

Το κόστος της εισαγωγής μεθοδολογιών Τυπικών Μεθόδων στον τομέα έρευνας μιας εταιρίας θεωρείται υψηλό κυρίως λόγω της ανάγκης για εξειδικευμένο προσωπικό που μπορεί να καλυφθεί είτε με νέες προσλήψεις είτε με εκπαίδευση του υπάρχοντος προσωπικού. Όμως, οι Bowen και Hinchey [16] ισχυρίζονται πως μία εταιρία μπορεί να δει το κόστος παραγωγής να αυξάνεται όταν ξεκινά να χρησιμοποιεί τυπικές μεθόδους, όμως σε βάθος χρόνου η μέθοδος έχει το ίδιο κόστος (αν όχι μικρότερο) με τις συμβατικές μεθόδους. Οι συγγραφείς έχουν θετικές εμπειρίες εκπαιδεύοντας φοιτητές μεταπτυχιακού επιπέδου στις Τυπικές Προδιαγραφές. Η εκπαιδευτική προσέγγισε συνδύαζε Μάθηση βασισμένη σε Πρόβλημα (Problem Based Learning) και Μάθηση με έμφαση σε Πρόβλημα (Problem Focused Education) και μηχανισμούς ανατροφοδότησης για την παρακολούθηση της διαδικασίας της εκπαίδευ-

σης και την εφαρμογή των εννοιών των τυπικών μεθόδων και τυπικών προδιαγραφών [45]. Μέσα σε τρεις μήνες οι φοιτητές είχανε φτάσει σε τέτοιο επίπεδο εξοικείωσης με αυτές τις έννοιες που ήταν ικανοί να κατανοήσουν ένα πρωτόκολλο βασισμένοι στην τυπική προδιαγραφή του και να προδιαγράψουν οι ίδιοι κομμάτια κάποιων συστημάτων, ως μέρος των απαιτήσεων του μαθήματος (βλ. Κεφάλαιο 4.5). Μία άλλη ενδιαφέρουσα προσέγγιση για την εκπαίδευση φοιτητών στις τυπικές μεθόδους είναι η [46]. Ο Newcombe και οι υπόλοιποι συγγραφείς ([47]) ισχυρίζονται ότι οι μηχανικοί της Amazon κατάφεραν να φτάσουν σε ένα χρήσιμο επίπεδο στην κατανόηση των τυπικών μεθόδων μέσα σε τρεις εβδομάδες και πως τα πλεονεκτήματα της χρήσης των τυπικών μεθόδων είναι πολύ σημαντικά.

3.2 Σχετικές δουλειές

Η ιδέα της τυποποίησης ανοιχτών προτύπων δεν είναι καινούργια· εταιρίες που θεσπίζουν πρότυπα όπως ο International Organization for Standardization (ISO), το ITU Telecommunication Standardization Sector (ITU-T) και η International Electrotechnical Commision (IEC) έχουν σχέδια (μερικά συνεχίζονται μέχρι σήμερα) για να εκφράσουν μέρη των προτύπων τους με ένα πιο τυπικό τρόπο. Για παράδειγμα, το Reference Model of Open Distributed Processing (RM-ODP) τυποποίησε την αρχιτεκτονική σημασιολογία του σε τέσσερις γλώσσες τυπικών προδιαγραφών (LOTOS, ESTELLE, SDL and Z) [48] και αργότερα στη UML 2 [49].

Υπάρχει επίσης ένα πρότυπο κατευθυντήριων γραμμών που εξηγεί τις τυπικές περιγραφικές τεχνικές (Formal Description Techniques [50, 51]). Το βιβλίο του Turner [51] είναι ενδιαφέρον γιατί επίσης δείχνει πως το ίδιο πρότυπο μπορεί να περιγραφεί σε διαφορετικές τυπικές γλώσσες προδιαγραφών. Ένα ακόμα σχετικό έργο είναι το [52] και είναι μια προσπάθεια των ISO/ITU-T για να προτυποποιήσουν έναν ορισμό για τη συμμόρφωση στο πλαίσιο των τεχνικών τυπικών προδιαγραφών. Σε αντιδιαστολή με αυτά τα πρότυπα, η διατριβή αυτή επικεντρώνεται σε λιγότερο ασαφή πρότυπα όπως πρότυπα λογισμικού που χρησιμοποιούνται ως υπηρεσίες web.

4 Μελέτη περιπτώσεων

Στο κεφάλαιο αυτό θα μελετήσουμε περιπτώσεις για να στηρίξουμε την ιδέα που παρουσιάστηκε στο Κεφάλαιο 3. Κάθε υπο-ενότητα αυτού του κεφαλαίου αντιστοιχεί και σε μία δημοσιευμένη εργασία, σε συνέδρια ή περιοδικά.

4.1 Abstract Syntax Notation One (ASN.1)

Η ASN.1 ([53]) είναι ένα πρότυπο που δημιουργήθηκε από τον International Organization for Standardization (ISO), την International Electrotechnical Commission (IEC) σε συνεργασία με τον ITU Telecommunication Standardization Sector (ITU-T). Είναι μια πολύ γνωστή σημειογραφία που χρησιμοποιείται για την περιγραφή μηνυμάτων που ανταλλάσσονται μεταξύ προγραμμάτων επικοινωνίας ([53, 54]). Είναι ένα πλαίσιο για την αναπαράσταση δομών δεδομένων δενδροειδούς μορφής που παρέχει ένα σύνολο τυπικών κανόνων για την περιγραφή της δομής των δεδομένων που είναι ανεξάρτητα από το πώς τα κωδικοποιεί το κάθε μηχάνημα. Αυτό ελευθερώνει τους σχεδιαστές πρωτοκόλλων από το να μεριμνούν για τα bits και τα bytes της δομής των μηνυμάτων. Η ASN.1 είναι μια ακριβής, τυπική σημειογραφεία που έχει σα στόχο να αφαιρεί ασάφειες. Η ASN.1 παρέχει την αφηρημένη σύνταξη (abstract syntax) των δεδομένων.

Μια προδιαγραφή ASN.1 δεν προσδιορίζει από μόνη της τη συγκεκριμένη αναπαράσταση των δεδομένων κατά τη μετάδοση ή την αποθήκευση. Η συγκεκριμένη αναπαράσταση εκφράζεται ως σύνταξη μεταφοράς (transfer syntax) με ορολογία της ASN.1 και καθορίζεται από την εφαρμογή ενός συνόλου κανόνων κωδικοποίησης στην αφηρημένη σύνταξη.Ένα προφανές πλεονέκτημα της ASN.1 είναι ότι τα ίδια δεδομένα (σημασιολογικά) μπορούν να διαμορφωθούν με διάφορους τρόπους (συντακτικά) με την εφαρμογή διαφορετικών κανόνων κωδικοποίησης. Για παράδειγμα, μπορούμε να πάρουμε μία πολύ συμπαγή δυαδική αναπαράσταση χρησιμοποιώντας τους κανόνες PER (Packed Encoding Rules, [54]).

Η ASN.1 ξεκίνησε για την περιγραφή μηνυμάτων ηλεκτρονικής αλληλογραφίας του πρωτοκόλλου Open Systems Interconnection και από τότε, έχει χρησιμοποιηθεί για μία ευρεία γκάμα εφαρμογών όπως διαχείριση δικτύων, ασφαλή email, κινητή τηλεφωνία, έλεγχος εναέριας κυκλοφορίας, VOIP, τραπεζικά συστήματα, ασύρματα δίκτυα, μεταφορές, ασφάλεια, έλεγχο ταυτότητας και κρυπτογράφηση, ενέργεια, ηλεκτρο-

νικές κάρτες, υγεία, γενετική, γραφικά υπολογιστών και μεταφορά αρχείων.

Η ASN.1 έχει σχεδιαστεί με στόχο την απόδοση· έτσι, τα δεδομένα αποστέλλονται με μόλις μερικά bytes να ξεχωρίζουν τα διαφορετικά δεδομένα μεταξύ τους οπότε δεν είναι εύκολα αναγνώσιμα ή διαχειρίσιμα. Αφού τα δεδομένα της ASN.1 είναι δομημένα, αυτά μπορούν να αναπαρασταθούν σε μορφή Extensible Markup Language (XML) σε κόστος βέβαια της απόδοσης· τα δεδομένα XML είναι πολύ πιο αναγνώσιμα (σε κόστος του μεγέθους των δεδομένων) και πιο εύκολα προσβάσιμα αφού υπάρχουν πολλά εργαλεία που διαβάζουν και επεξεργάζονται την XML. Η XML μπορεί να λειτουργήσει ως σύνταξη μεταφοράς για την ASN.1 κάνοντας έτσι ακόμα πιο ισχυρή την ASN.1 από την άποψη ότι επιτρέπει σε πλήθος κόσμου προσκείμενου στην XML να χρησιμοποιήσουν την ASN.1. Ανάμεσα στις εταιρίες που έχουν δημιουργήσει εργαλεία μετατροπής από ASN.1 σε XML είναι και η IBM, ως μέρος του πακέτου XML Security Suite .

Η ASN.1 συνδέεται και με γλώσσες προγραμματισμού, όπως τη C (ASN.1 to C/C++ Compiler [55]), ή τη Java (ASN.1 to Java/C# Compiler [56]). Τα προγράμματα αυτά παράγουν ένα πλήρως εκτελούμενο πρόγραμμα σε C (ή Java) από τα modules της ASN.1. Επίσης, υπάρχει ένας cross compiler από την ASN.1 στην Erlang [57] που παράγει συναρτήσεις κωδικοποίησης και αποκωδικοποίησης για χρήση σε προγράμματα Erlang που στέλνουν ή δέχονται δεδομένα ASN.1. Τέλος, υπάρχει ένα σύνολο εργαλείων για τον Python (ASN.1 for Python [58]) που φέρνει τους τύπους δεδομένων και τη σύνταξη μεταφοράς της ASN.1 στον Python.

4.1.1 Σύνταξη της ASN.1

Εδώ θα παρουσιάσουμε τους βασικούς συντακτικούς κανόνες της ASN.1 που θα βοηθήσουν τον αναγνώστη να καταλάβει τα παραδείγματα που ακολουθούν. Θα χρησιμοποιήσουμε το παράδειγμα ενός τραπεζικού λογαριασμού· πώς αυτό περιγράφεται στην ASN.1 και πώς αυτό αργότερα θα μετατραπεί στην CafeOBJ. Το module με όνομα *Account* (κώδικας 8) χρησιμοποιείται για να ορίσει τα δεδομένα που ανταλλάσσονται μεταξύ διαφορετικών οντοτήτων της τράπεζας. Όλες οι λέξεις κλειδιά της ASN.1 γράφονται πάντα με κεφαλαία γράμματα εκτός μερικών τύπων χαρακτήρων όπως το *NumericString*.

```
Account ::= SEQUENCE {
 iban NumericString (SIZE (27)),
 client Client,
```

```
 balance Balance
}
```

Κώδικας 8: Παράδειγμα του SEQUENCE

Ο κώδικας 8 μπορεί να διαβαστεί ως: Ένα Account είναι μια δομή ακολουθίας που απαρτίζεται από τρία συστατικά: το πρώτο, με όνομα *iban* (ξεκινάει με πεζό χαρακτήρα) χρησιμοποιείται για να υποδηλώσει αριθμητικά δεδομένα σταθερού μήκους (27 χαρακτήρες). Το δεύτερο ονομάζεται *client* και δηλώνει δεδομένα τύπου *Client* (ξεκινάει με κεφαλαίο χαρακτήρα). Το τρίτο στοιχείο λέγεται *balance* και δηλώνει δεδομένα τύπου *Balance*. Το *Account* είναι τύπος της ASN.1, γράφεται με κεφαλαίο πρώτο γράμμα και ακολουθείται από το σύμβολο ::=. Μέσα στα σύμβολα { και } τοποθετείται ο ορισμός του τύπου.

Ο τύπος *SEQUENCE* είναι ο πιο συχνός τύπος στην ASN.1 και χρησιμοποιείται για εμφωλευμένα δεδομένα. Ένας παρόμοιος τύπος δεδομένων (και εξίσου συχνά χρησιμοποιούμενος) είναι ο *CHOICE* που δηλώνει ότι μόνο ένας από τους εμφωλευμένους τύπους που ορίζει θα μεταδοθεί. Η σύνταξη του *CHOICE* είναι ίδια με του *SEQUENCE*. Οι τύποι δεδομένων *Client* και *Balance* δεν είναι γνωστοί στην ASN.1 και δηλώνονται αργότερα στην προδιαγραφή του συστήματος. Κάθε συστατικό ενός ορισμένου τύπου χωρίζεται από το επόμενο του με το κόμμα.

```
Payment−method ::= CHOICE {
 check Check−number,
 credit−card SEQUENCE {
        number Card−number,
        expiry−date Date }
}
```

Κώδικας 9: Παράδειγμα του CHOICE

Ο κώδικας 9 μπορεί να διαβαστεί ως: Το *Payment-method* είναι μια δομή επιλογής που απαρτίζεται από δύο συστατικά: το πρώτο, με όνομα *check* τύπου *Check-number* (τύπος δεδομένων που ορίζεται αργότερα) και το δεύτερο, με όνομα *credit-card* είναι μια δομή ακολουθίας με δύο στοιχεία· το *number*, τύπου *Card-number* και το *expiry-date* τύπου ημερομηνίας. Επειδή μιλάμε για τύπο *CHOICE*, μόνο ένα από τα δύο συστατικά (είτε το check είτε το credit-card) θα μεταδοθούν.

Ο κώδικας 9 δείχνει και πώς μπορούμε να εμφωλεύσουμε δομές δεδομένων (ένα *SEQUENCE* μέσα σε ένα *CHOICE*). Οι τύποι δεδομένων που δεν είναι γνωστοί στην ASN.1 (*Client* και *Balance* στον κώδικα 8 και *Check-number*, *Card-number* στον κώδικα 9) μπορούν να δηλωθούν οπουδήποτε στην προδιαγραφή.

```
Client ::= SEQUENCE {
 clientid Integer,
 firstname PrintableString (SIZE (1..30)),
 lastname PrintableString (SIZE (1..50)),
 street PrintableString (SIZE (1..50)),
 postcode NumericString (SIZE (5)),
 city PrintableString (SIZE (1..30)),
 country PrintableString (SIZE (1..20))
}
```

Κώδικας 10: Τα δεδομένα ενός πελάτη

Ο κώδικας 10 διαβάζεται ως εξής: Ο *Client* (που εμφανίζεται στον κώδικα 8) είναι μια δομή ακολουθίας με επτά συστατικά:

1. Το *clientid* που είναι ακέραιος αριθμός.
2. Το *firstname* που είναι ακολουθία μήκους το πολύ 30 χαρακτήρων.
3. Το *lastname* που είναι ακολουθία μήκους το πολύ 50 χαρακτήρων.
4. Το *street* που είναι ακολουθία μήκους το πολύ 50 χαρακτήρων.
5. Το *postcode* που είναι αριθμητικά δεδομένα μήκους 5 χαρακτήρων.
6. Το *city* που είναι ακολουθία μήκους το πολύ 30 χαρακτήρων.
7. Το *country* που είναι ακολουθία μήκους το πολύ 20 χαρακτήρων.

Το module για το Balance του κώδικα 8 ορίζεται αντίστοιχα.

4.1.2 Τυπική προδιαγραφή της ASN.1

Η ASN.1 και η CafeOBJ χρησιμοποιούνται για προδιαγραφές συστημάτων, όμως χρησιμοποιούν τελείως διαφορετική προσέγγιση και μπορούν να προδιαγράψουν διαφορετικές πτυχές ενός συστήματος. Η ASN.1 χρησιμοποιείται κυρίως για να περιγράψει τη στατική δομή των δεδομένων που ανταλλάσσονται κατά τη λειτουργία του συστήματος και τα ίχνη των διαδικασιών που ορίζουν ένα σύστημα. Δε μπορεί να περιγράψει κάποια πτυχή της δυναμικής συμπεριφοράς του συστήματος αφού δεν έχει κάποιον τρόπο να κατανοήσει τις καταστάσεις του συστήματος. Δε μπορούμε να χρησιμοποιήσουμε την ASN.1 για να περιγράψουμε πώς μια διαδικασία μπορεί να αλλάξει τα δεδομένα του συστήματος ή τις αλλαγές που μπορούν αν συμβούν σε αυτά. Η CafeOBJ μπορεί να

χειριστεί και τις δυναμικές συμπεριφορές ενός συστήματος αλλά και τις αλλαγές που γίνονται στα δεδομένα· οι δράσεις ("actions") στην CafeOBJ περιγράφουν τον τύπο των αλλαγών που γίνονται σε ένα σύστημα και με τους τελεστές παρατήρησης μπορούμε να δούμε τις αλλαγές που έχουν γίνει στα δεδομένα ως αποτέλεσμα μιας δράσης. Η "ασάφεια" της CafeOBJ επιτρέπει στις δομές των δεδομένων να συνεπάγονται από τη προδιαγραφή (σε μορφή αφηρημένων συνόλων) αντί να περιγράφονται ρητά (όπως κάνει η ASN.1). Αυτό σημαίνει ότι η μετατροπή μιας προδιαγραφής από την ASN.1 στη CafeOBJ δε μπορεί να είναι πλήρης αφού η προδιαγραφή που θα προκύπτει θα είναι ένα μέρος μόνο από την προδιαγραφή που θα γράφαμε στην CafeOBJ. Τα κομμάτια που λείπουν θα πρέπει να γραφούν από το σχεδιαστή ώστε να μπορέσουμε να χρησιμοποιήσουμε την προδιαγραφή για επαλήθευση ιδιοτήτων. Έτσι, προσθέτοντας μόνο τις ενέργειες που μπορούν να γίνουν στο σύστημα, έχουμε όχι απλά μια πλήρη προδιαγραφή του συστήματος, αλλά έχουμε και μια προδιαγραφή που μπορεί να χρησιμοποιηθεί για την επαλήθευση του συστήματος. Η μετατροπή από ASN.1 στην CafeOBJ γίνεται αυτόματα και πάνω σε αυτό που προκύπτει προδιαγράφουμε τις δράσεις του συστήματος.

Η μέθοδος αυτή αξιοποιεί μια υπάρχουσα προδιαγραφή σε ASN.1, γλυτώνοντας χρόνο στον αναλυτή του συστήματος που θέλει να επαληθεύσει το σύστημα. Αν ήθελε να χρησιμοποιήσει τυπικές μεθόδους για επαλήθευση, θα έπρεπε να έγραφε την προδιαγραφή του συστήματος από την αρχή.

4.1.3 Οι κανόνες της μετατροπής

Η μετατροπή από ASN.1 σε CafeOBJ γίνεται με κάποιους κανόνες που εξασφαλίζουν την αντιστοιχία της προδιαγραφής που προκύπτει με την αρχική προδιαγραφή:

Ο πιο συχνός τύπος δεδομένων στην ASN.1 είναι το *Sequence*. Ο τύπος αυτός δίνει οδηγίες για το πώς μεταδίδονται τα δεδομένα που είναι εμφωλευμένα σε κάθε sequence· τι τύποι δεδομένων περιέχονται και με ποια σειρά αυτά στέλνονται. Κάθε sequence θα τυποποιηθεί ως *record* με σύνολα και υποσύνολα. Κάθε όνομα sequence δηλώνεται ως σύνολο και κάθε τύπος δεδομένων μέσα σε αυτό το sequence δηλώνεται ως υποσύνολο, διατηρώντας έτσι την αρχική σημασιολογία. Για κάθε σύνολο δημιουργούμε ένα κατασκευαστικό τελεστή που συνθέτει το σύνολο από τα υποσύνολα που το απαρτίζουν. Αντίστοιχα, για κάθε υποσύνολο δημιουργούμε τελεστές που δοθέντος του κυρίως συνόλου, επιστρέφουν το συγκεκριμένο υποσύνολο. Αν και αυτοί οι τελεστές δεν

υπάρχουν στην προδιαγραφή της ASN.1, μπορούν να μας βοηθήσουν αργότερα στην επαλήθευση, αφού έτσι έχουμε πρόσβαση σε κάθε πληροφορία που προδιαγράφεται.

Ο τύπος *choice* τυποποιείται με αντίστοιχο τρόπο· η μόνη διαφορά είναι ότι θα προσθέσουμε τελεστές που επιτρέπουν μόνο ένα από τα υποσύνολα να αποσταλούν· μόνο μία από τις διαθέσιμες παραμέτρους επιτρέπεται να μην είναι κενή, αλλιώς η κατάσταση του συστήματος που περιγράφεται δε θα είναι έγκυρη.

4.1.4 Μετασχηματίζοντας προδιαγραφές της ASN.1 στην CafeOBJ

Το πρόγραμμα asn2obj (γραμμένο σε python) μετασχηματίζει μία προδιαγραφή γραμμένη σε ASN.1 σε μια προδιαγραφή σε CafeOBJ ακολουθώντας τους παραπάνω κανόνες. Το αποτέλεσμα μπορεί να χρησιμοποιηθεί ως σκελετός για περιγραφή και επαλήθευση ιδιοτήτων του συστήματος, όμως, πριν κάτι τέτοιο θα πρέπει να είμαστε σίγουροι ότι η προδιαγραφή που προκύπτει είναι μια υλοποίηση της αρχικής περιγραφής. Στη μετατροπή αυτή θα δώσουμε έμφαση στις προδιαγραφές δεδομένων τύπου Boolean και αριθμητικών δεδομένων.

Η ιδιότητα που μας ενδιαφέρει εδώ είναι η εξής:

Ιδιότητα 4.1. *Δοθείσης μιας προδιαγραφής σε ASN.1 και ενός συνόλου κανόνων για τη μετατροπή της σε προδιαγραφή CafeOBJ, η προδιαγραφή που προκύπτει είναι μια υλοποίηση της αρχικής προδιαγραφής.*

Η απόδειξη της ιδιότητας αυτής είναι πολύ σημαντική για την ιδέα αυτή. Για να αποδείξουμε την ιδιότητα θα τυποποιήσουμε τους κανόνες μετατροπής, τις προδιαγραφές σε ASN.1 και τις προδιαγραφές σε CafeOBJ, όπως φαίνεται στο [59]. Η διαδικασία έχει ως εξής:

Θα προδιαγράψουμε τους κανόνες μετάφρασης στην CafeOBJ. Έχουμε ένα σύνολο από κανόνες μετάφρασης για τα sequences (και για τους υπόλοιπους τύπους της ASN.1) και τους βασικούς τύπους δεδομένων (bool, int, κ.λπ.) που μοντελοποιούνται σαν τελεστές. Για παράδειγμα, δοθείσης μιας sequence, έχουμε το σύνολο κανόνων *sequence2module* που:

- δημιουργεί το module της CafeOBJ με όνομα ίδιο με αυτό του sequence
- ορίζει το κυρίως σύνολο με όνομα ίδιο με αυτό του sequence
- ορίζει τον κατασκευαστικό τελεστή με όνομα ίδιο με αυτό του sequence

- δημιουργεί μεταβλητή με όνομα aXXX (όπου XXX το όνομα του sequence)

Επίσης έχουμε το σύνολο κανόνων *subseq2subsort* που για κάθε στοιχείο που ορίζεται σε ένα sequence:

- το δηλώνει ως υποσύνολο του συνόλου που όρισε το sequence
- δημιουργεί έναν τελεστή που δοθέντος του συνόλου του sequence επιστρέφει το υπο-στοιχείο του sequence
- δημιουργεί την κατάλληλη μεταβλητή

Ορίζουμε έναν τελεστή με όνομα *translate* που δέχεται σαν όρισμα μία προδιαγραφή της ASN.1, *AnASN* (τύπου ASN) και επιστρέφει ένα "πρόγραμμα" της CafeOBJ, *ACafeOBJ*, τύπου CAFEOBJ. Η Ιδιότητα 4.1 μπορεί τώρα να γραφεί ως:

Ιδιότητα 4.2. *Δοθείσης μιας προδιαγραφής σε ASN.1 (B), το ASN.1-Cafe πρόγραμμα A = translate(B) είναι πάντα μια υλοποίηση του B.*

Για να δείξουμε ότι το A είναι μια υλοποίηση του B πρέπει να δείξουμε ότι υπάρχει μια refinement σχέση R ([60]) ανάμεσα στις καταστάσεις του A και του B έτσι ώστε:

- για μία αυθαίρετη αρχική κατάσταση s του A υπάρχει μια αρχική κατάσταση u του B έτσι ώστε $(s, u) \in R$
- για μία αυθαίρετη κατάσταση s του A και μία αυθαίρετη μέθοδο m του A υπάρχει μία κατάσταση u του B και μία ακολουθία μεταβάσεων (η εκτέλεση μιας δράσης α) του B έτσι ώστε $(s, u) \in R$ και $(m(s), a(u)) \in R$, όπου m είναι η μετάφραση μιας δράσης α.

Η απόδειξη αυτή δεν έχει ολοκληρωθεί ακόμα.

4.1.5 Επαλήθευση ιδιοτήτων

Έχοντας το σκελετό της προδιαγραφής του συστήματος από ASN.1 σε CafeOBJ, μπορούμε τώρα να προδιαγράψουμε τις επιθυμητές ιδιότητες του συστήματος και χρησιμοποιώντας τη μηχανή αποδείξεων της CafeOBJ μπορούμε να επαληθεύσουμε την ισχύ τους. Για να γίνει αυτό θα φέρουμε την προδιαγραφή στη μορφή του OTS (Observational Transitional Systems) ώστε να χρησιμοποιήσουμε τη μεθοδολογία με όνομα "proof score" ([24, 25, 61]):

Υποθέτοντας ότι Y είναι ο χώρος καταστάσεων (όπως ορίζεται από το κρυφό σύνολο που χειροκίνητα εισάγουμε στην προδιαγραφή) και $D*$ οι τύποι δεδομένων που χρησιμοποιούνται στα συστήματα OTS ([25]), ένα σύστημα OTS S είναι ένα σύνολο O, I, T έτσι ώστε:

- Το O είναι ένα πεπερασμένο σύνολο παρατηρητών από το Y στο $D*$. Κάθε τελεστής $o \in O$ είναι μια συνάρτηση από το Y στο D.
- Το I είναι ένα σύνολο από αρχικές καταστάσεις έτσι ώστε $I \subset Y$
- Το T είναι ένα πεπερασμένο σύνολο από μεταβάσεις από το Y στο Y. (Οι μεταβάσεις απλά "μετακινούν" το σύστημα από μία κατάσταση σε μία άλλη)

Η ASN.1 δε γνωρίζει για καταστάσεις συστήματος, οπότε πρέπει κάπως να συμπεριλάβουμε αυτή την ιδέα στην προδιαγραφή που προκύπτει. Ο χώρος καταστάσεων που ορίζει το κρυφό σύνολο μεταβάλλεται κάθε φορά που κάποια δράση λαμβάνει χώρα. Ορίζουμε λοιπόν μεταβατικούς τελεστές που δέχονται τον τρέχοντα χώρο καταστάσεων και εφαρμόζοντας κάποια δράση δημιουργούν ένα καινούργιο χώρο καταστάσεων. Για να ολοκληρωθεί η υλοποίηση του OTS χρειαζόμαστε και τελεστές παρατήρησης που δέχονται σαν είσοδο την κατάσταση του συστήματος και επιστρέφουν τιμές ιδιοτήτων. Είναι οι τελεστές που μας ενημερώνουν για τις αλλαγές που επιφέρει μια δράση. Οι τελεστές αυτοί έχουν δημιουργηθεί αυτόματα, όπως εξηγήσαμε στο Κεφάλαιο 4.1.4.

Τέλος, αυτό που μένει να κάνουμε είναι να προδιαγράψουμε μία ιδιότητα που θέλουμε να ισχύει σε κάθε προσβάσιμη κατάσταση του συστήματος και για κάθε τέτοια κατάσταση να γράψουμε ένα proof passage ([25]) που στέλνει το σύστημα σε αυτή την κατάσταση και εκεί ελέγχουμε αν στέκει. Το Σχήμα 5 περιγράφει συνοπτικά τη διαδικασία απόδειξης.

4.1.6 Παράδειγμα δομής τραπεζικού λογαριασμού

Στο παράδειγμα αυτό θα θεωρήσουμε μια δομή που περιγράφει ένα λογαριασμό τραπέζης στην ASN.1. Στη συνέχεια, θα χρησιμοποιήσουμε τους κανόνες που περιγράψαμε για να πάρουμε ένα σύνολο από modules της CafeOBJ στα οποία θα προσθέσουμε και θα επιχειρήσουμε να επαληθεύσουμε ιδιότητες. Το παράδειγμα είναι αρκετά μινιμαλιστικό, χωρίς να εξετάζει σε πολύ βάθος ένα τραπεζικό σύστημα, όμως είναι αρκετό για να δείξει τη μεθοδολογία που χρησιμοποιούμε.

Θα περιγράψουμε τα δεδομένα που χρειάζεται ένας πελάτης της τράπεζας για να ανοίξει ένα τραπεζικό λογαριασμό, το λογαριασμό,

Σχήμα 5: Τα προτεινόμενα βήματα για την επαλήθευση ιδιοτήτων συστήματος

δύο πιθανές ενέργειες (κατάθεση και ανάληψη) και μία ερώτηση για το υπόλοιπο του λογαριασμού.

4.1.6.1 ASN.1 module

Τα στοιχεία του πελάτη είναι ένα sequence στην ASN.1 που κρατάει στοιχεία όπως το όνομα του πελάτη, τη διεύθυνσή του και ένα μοναδικό ClientID. Το module της ASN.1 για τον πελάτη φαίνεται στο 10 (Σελίδα 63). Κάθε λογαριασμός περιγράφεται από τον IBAN του (International Bank Account System) και πρέπει να έχει έναν ιδιοκτήτη - μέλος της τράπεζας ώστε να έχει ένα ClientID. Μία ιδιότητα του λογαριασμού είναι το υπόλοιπό του. Το module της ASN.1 για το υπόλοιπο ενός λογαριασμού φαίνεται στον κώδικα 11. Το module της ASN.1 για το λογαριασμό φαίνεται στο 8 (Σελίδα 61).

```
Balance ::= SEQUENCE {
        iban  Iban,
        amount  REAL
}
```

Κώδικας 11: Υπόλοιπο λογαριασμού

Τέλος, προδιαγράφουμε τη μορφή των δεδομένων που ανταλλάσσονται στα τραπεζικά συστήματα όταν συμβεί κάποια από τις δύο ενέργειες (κατάθεση σε λογαριασμό, ανάληψη από λογαριασμό). Κάθε μία από τις δύο ενέργειες αναφέρεται σε έναν συγκεκριμένο λογαριασμό, σε έναν συγκεκριμένο πελάτη της τράπεζας, στην ημερομηνία που έγινε η ενέργεια και στο ποσό. Ο κώδικας 12 δείχνει τα modules αυτά.

```
Date ::= NumericString (SIZE (8)) -- DDMMYYYY

Deposit ::= SEQUENCE {
        account Account
        clientid ClientID,
        date  Date,
        amount  Real }

Withdraw ::= SEQUENCE {
        account Account
        clientid ClientID,
        date  Date,
        amount  Real }
```

Κώδικας 12: Κατάθεση/ανάληψη

4.1.6.2 Κώδικας CafeOBJ

Ο κώδικας 13 δείχνει το module της CafeOBJ που προκύπτει από το μετασχηματισμό της ASN.1 προδιαγραφής. Το όνομα του module είναι *CLIENT*, όπως και του sequence δηλαδή. Ομοίως και το όνομα του κυρίως συνόλου. Όλα τα υπόλοιπα δεδομένα που βρίσκονται στο sequence ορίζονται ως υποσύνολα του Client. Ο τελεστής *client* δημιουργεί τα στοιχεία του πελάτη από όλα τα υπο-δεδομένα. Επίσης, κάθε δεδομένο στο sequence έχει τον δικό του τελεστή που επιστρέφει την τιμή αυτή, δοθέντος του πελάτη. Οι τελεστές αυτοί έχουν όνομα της μορφής *returnXXX*, όπου XXX το όνομα του κάθε υπο-τύπου. Τέλος, παρατηρούμε ότι δημιουργήθηκαν μεταβλητές της μορφής *aXXX*, όπου XXX το όνομα του κάθε υπο-τύπου.

```
mod CLIENT{

[Client > ClientID FirstName LastName Address PostCode City Country]

op client : ClientID FirstName LastName Address PostCode City Country
        -> Client
```

```
op returnclientid : Client -> ClientID
op returnfirstname : Client -> FirstName
op returnlastname : Client -> LastName
op returnaddress : Client -> Address
op returnpostcode : Client -> PostCode
op returncity : Client -> City
op returncountry : Client -> Country

var aclient : Client
var aclientid : ClientID
var afirstname : FirstName
var alastname : LastName
var aaddress : Address
var apostcode : PostCode
var acity : City
var acountry : Country

eq client(aclientid, afirstname, alastname, aaddress, apostcode, acity, acountry)
        = aclient .
eq returnclientid(client(aclientid, afirstname, alastname, aaddress, apostcode,
        acity, acountry)) = aclientid .
eq returnfirstname(client(aclientid, afirstname, alastname, aaddress, apostcode,
        acity, acountry)) = afirstname .
eq returnlastname(client(aclientid, afirstname, alastname, aaddress, apostcode,
        acity, acountry)) = alastname .
eq returnaddress(client(aclientid, afirstname, alastname, aaddress, apostcode,
        acity, acountry)) = aaddress .
eq returnpostcode(client(aclientid, afirstname, alastname, aaddress, apostcode,
        acity, acountry)) = apostcode .
eq returncity(client(aclientid, afirstname, alastname, aaddress, apostcode,
        acity, acountry)) = acity .
eq returncountry(client(aclientid, afirstname, alastname, aaddress, apostcode,
        acity, acountry)) = acountry .
}
```

Κώδικας 13: Το module του πελάτη

Το κύριο module στην προδιαγραφή είναι αυτό του λογαριασμού. Θα θέσουμε το σύνολο που δημιουργήθηκε (με όνομα *Account*) ως το κρυφό σύνολο, βάζοντας το μέσα σε $*[\ ]*$. Το module του λογαριασμού θα χρησιμοποιεί τα υπόλοιπα modules που δημιουργήσαμε (η εντολή *pr*). Εδώ, τα modules αυτά είναι αυτό του πελάτη και το INT που είναι το ενσωματωμένο module της CafeOBJ για τους ακέραιους αριθμούς. Μία αλλαγή που θα κάνουμε στο module είναι ότι αφού το Account θα

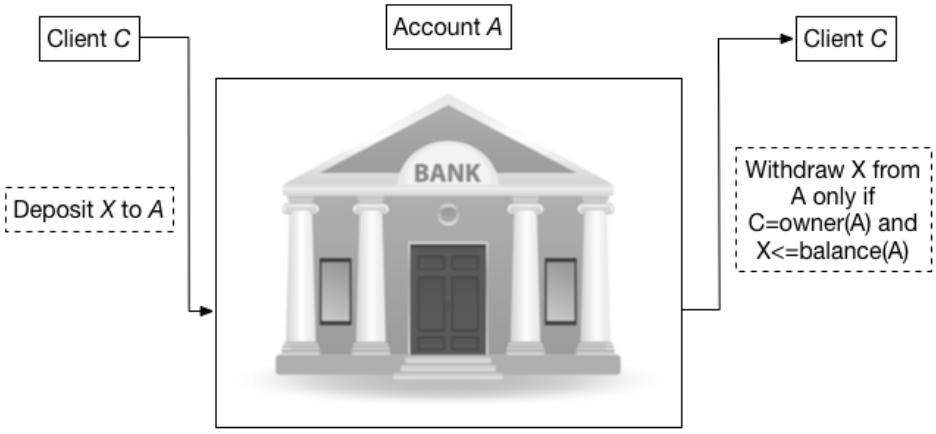
είναι κρυφό σύνολο, όλοι οι τύποι δεδομένων που το χρησιμοποιούν θα εμφωλευθούν από κάτω. Το module που φαίνεται στον κώδικα 14 είναι αυτό που παίρνουμε τελικά. Ένα σύνολο από εκτελέσιμα modules που πέρα από την προδιαγραφή του συστήματος, μπορούν να χρησιμοποιηθούν για την επαλήθευση ιδιοτήτων όπως θα δούμε παρακάτω.

```
mod ACCOUNT {
pr(INT + CLIENT )

*[Account]*
[Iban ClientID Balance]
[Deposit > ClientID Iban Date Int]
[Withdraw > ClientID Iban Date Int]

op account : Iban ClientID Balance -> Account
bop returniban : Account -> Iban
bop returnclientid : Account -> ClientID
bop returnbalance : Account -> Balance

op balance : Iban Int -> Balance
op returniban2 : Balance -> Iban
op returnamount : Balance -> Int

op deposit : Account ClientID Date Int -> Deposit
op returnaccount : Deposit -> Account
op returnclientid2 : Deposit -> ClientID
op returndate : Deposit -> Date
op returnamount : Deposit -> Int

op withdraw : Account ClientID Date Int -> Withdraw
op returnaccount2 : Withdraw -> Account
op returnclientid3 : Withdraw -> ClientID
op returndate2 : Withdraw -> Date
op returnamount2 : Withdraw -> Int

var adeposit : Deposit
var awithdraw : Withdraw
var aaccount : Account
var aclientid : ClientID
var aint : Int
var adate : Date
var aiban : Iban
var abalance : Balance
```

```
eq account(aiban, aclientid, abalance) = aaccount .
eq returniban(account(aiban, aclientid, abalance)) = aiban .
eq returnclientid(account(aiban, aclientid, abalance)) = aclientid .
eq returnbalance(account(aiban, aclientid, abalance)) = abalance .

eq balance(aiban, aint) = abalance .
eq returniban2(balance(aiban, aint)) = aiban .
eq returnamount(balance(aiban, aint)) = aint .

eq deposit(aaccount, aclientid, adate, aint) = adeposit .
eq returnaccount(deposit(aaccount, aclientid, adate, aint)) = aaccount .
eq returnclientid2(deposit(aaccount, aclientid, adate, aint)) = aclientid .
eq returndate(deposit(aaccount, aclientid, adate, aint)) = adate .
eq returnamount(deposit(aaccount, aclientid, adate, aint)) = aint .

eq withdraw(aaccount, aclientid, adate, aint) = awithdraw .
eq returnaccount2(withdraw (aaccount, aclientid, adate, aint)) = aaccount .
eq returnclientid3(withdraw (aaccount, aclientid, adate, aint)) = aclientid .
eq returndate2(withdraw (aaccount, aclientid, adate, aint)) = adate .
eq returnamount2(withdraw (aaccount, aclientid, adate, aint)) = aint . }
```

Κώδικας 14: Το module για το λογαριασμό

4.1.6.3 Προσθήκες στον κώδικα



Σχήμα 6: ASN.1: Το τραπεζικό σύστημα που προδιαγράφουμε

Θα επιχειρήσουμε τώρα να φέρουμε τον κώδικα που έχουμε στη μορφή του OTS. Πρέπει να μοντελοποιήσουμε τις αλλαγές που μπορούν

να γίνουν στις καταστάσεις του συστήματος. Μετά, θα περιγράψουμε τις ιδιότητες που θέλουμε να επαληθεύσουμε (Σχήμα 6):

Ιδιότητα 4.3. *Η πράξη της κατάθεσης προσθέτει το ποσό της κατάθεσης στο τρέχον υπόλοιπο του λογαριασμού.*

Ιδιότητα 4.4. *Η πράξη της ανάληψης μπορεί να γίνει μόνο από τον ιδιοκτήτη του λογαριασμού και μόνο αν το ποσό που ζητείται είναι μικρότερο ή ίσο του τρέχοντος υπολοίπου.*

Ιδιότητα 4.5. *Αν μια ανάληψη μπορεί να πραγματοποιηθεί (βλ. Ιδιότητα 4.4) τότε το ποσό που ζητείται θα αφαιρεθεί από το τρέχον υπόλοιπο του λογαριασμού.*

Τέτοιες ιδιότητες δε βρίσκονται στην προδιαγραφή της ASN.1 οπότε θα τις προσθέσουμε εμείς, προσθέτοντας μία αρχική κατάσταση για το σύστημα· ένα λογαριασμό με μηδενικό υπόλοιπο και για λόγους αναγνωσιμότητας θα απλουστεύσουμε λίγο τον τελεστή του υπολοίπου (Κώδικας 15).

```
op init : –> Account
op balance : Account –> Int

eq balance(init) = 0 .
eq balance(deposit(aaccount, aclientid, adate, aint)) = balance(aaccount) + aint .
```

Κώδικας 15: Αρχική κατάσταση και η εξίσωση του υπολοίπου

Η πράξη της κατάθεσης δε δεσμεύεται από κάποιο περιορισμό αφού οποιοσδήποτε μπορεί να καταθέσει χρήματα σε κάποιο λογαριασμό. Η πράξη της κατάθεσης δημιουργεί μια νέα έκδοση του *account* που θα είναι η προηγούμενη κατάστασή του με μόνη διαφορά το διαφορετικό υπόλοιπο λογαριασμού. Οι τελεστές *withdraw* και *deposit* είναι οι πράξεις που μπορούν να συμβούν στο σύστημα και ο τελεστής *balance* "παρατηρεί" το λογαριασμό μετά την ενέργεια μίας πράξης, επιστρέφοντας το υπόλοιπο.

Η πράξη της ανάληψης όμως έχει δεσμεύσεις (Ιδιότητα 4.4, οπότε θα μοντελοποιήσουμε την πράξη της ανάληψης σα μια εξίσωση υπό όρους (Κώδικας 16).

```
ceq balance(withdraw(aaccount,aclientid,adate,aint)) =
        balance(aaccount) − aint
        if (balance(aaccount) >= aint and returnclient(aaccount) == aclientid) .
```

Κώδικας 16: Έλεγχος ιδιοκτήτη και υπολοίπου πριν την ανάληψη

Η προδιαγραφή που δημιουργήσαμε από τον κώδικα της ASN.1 και με τις προσθήκες μας είναι εκτελέσιμος και μπορούμε να τον χρησιμοποιήσουμε για να προδιαγράψουμε ιδιότητες που θα θέλαμε να έχει το σύστημα.

4.1.7 Σχόλια

Μία εφαρμογή που χρησιμοποιεί την ASN.1 για να περιγράψει τα μηνύματα που ανταλλάσσονται από διαφορετικά μέρη του συστήματος μπορεί να προδιαγραφεί τυπικά, σχεδόν αυτόματα, με τον τρόπο που περιγράψαμε παραπάνω. Η προδιαγραφή που παίρνουμε ως αποτέλεσμα μπορεί στη συνέχεια να χρησιμοποιηθεί για να ελέγξει ο δημιουργός της εφαρμογής κατά πόσο τα μηνύματα που ανταλλάσσονται δεν παραβιάζουν τις απαιτήσεις του συστήματος που θέλει ο δημιουργός να ισχύουν. Για να γίνει αυτό αρκεί να εκφραστούν οι ιδιότητες του συστήματος στην ίδια γλώσσα με την τυπική προδιαγραφή των ASN.1 μηνυμάτων. Στην περίπτωση αυτή η προδιαγραφή έγινε χρησιμοποιώντας τη γλώσσα CafeOBJ (Κεφάλαιο 2.3) αλλά οποιαδήποτε άλλη γλώσσα τυπικών προδιαγραφών μπορεί να χρησιμοποιηθεί, καθεμία με τα δικά της πλεονεκτήματα και μειονεκτήματα αλλά τελικά, είναι θέμα προσωπικής προτίμησης.

4.2 Μία Αλγεβρική Προδιαγραφή Κοινωνικού Δικτύου

Στο κεφάλαιο αυτό:

1. δημιουργούμε μία τυπική προδιαγραφή μιας αφηρημένης έκδοσης ενός κοινωνικού δικτύου σα δυναμική σύνθεση από συμπεριφοριακά αντικείμενα (behavioural objects)

2. χρησιμοποιούμε αυτή την προδιαγραφή για να αποδείξουμε ιδιότητες ασφάλειας του συστήματος

Το σύστημα που θα προδιαγράψουμε δεν είναι πλήρες καθώς στερείται πολλών χαρακτηριστικών που έχει ένα τυχαίο κοινωνικό δίκτυο που είναι σε κυκλοφορία, αλλά έχει κάποια βασικά χαρακτηριστικά για να μας επιτρέψει να δείξουμε τη μεθοδολογία της αλγεβρικής προδιαγραφής ενός συστήματος και της επαλήθευσης ιδιοτήτων του. Μέσα από το κεφάλαιο αυτό θα φανεί πώς η εφαρμογή τυπικών μεθόδων σε ένα σύστημα οδηγεί σε μια πιο βαθιά κατανόηση του συστήματος και πώς μπορούμε να βρούμε λάθη στη σχεδίαση του συστήματος νωρίς και να τα διορθώσουμε έγκαιρα.

4.2.1 Εισαγωγικά

Ένα *Κοινωνικό Δίκτυο (Social Network)* είναι μία κοινωνική δομή που απαρτίζεται από άτομα ή οργανισμούς και τους δυαδικούς δεσμούς μεταξύ τους. Η έννοια των κοινωνικών δικτύων εμφανίζεται πρώτη φορά τον $19^{ο}$ αιώνα. Ένα *Κοινωνικό Δίκτυο* μπορεί να οριστεί με οποιονδήποτε από τους κάτωθι ορισμούς:

1. *ένα σύνολο σχέσεων*: Μια επίσημη δήλωση που περιέχει ένα σύνολο αντικειμένων (κόμβοι - αναπαριστούν άτομα) και μία χαρτογράφηση ή περιγραφή των σχέσεων (που συνήθως εκφράζουν τύπους αλληλεξάρτησης, όπως η φιλία ή τα κοινά ενδιαφέροντα) μεταξύ των κόμβων [62].

2. *ως ιστοσελίδες που παρέχουν διαδικτυακές υπηρεσίες*: Επιτρέπουν σε άτομα να δημιουργήσουν ένα δημόσιο ή ένα ημι-δημόσιο προφίλ μέσα σε ένα κλειστό σύστημα, να στοιχειοθετήσουν μία λίστα από άλλους χρήστες με τους οποίους μοιράζονται κάποια σύνδεση και τέλος, να δουν και να διασχίσουν τη λίστα αυτή αλλά και τις λίστες άλλων χρηστών του συστήματος. Η φύση αλλά και η ονομασία αυτών των συνδέσεων ποικίλει ανάλογα το σύστημα ([63]).

3. *ως μια online αναπαράσταση*: Ο τελευταίος αυτός ορισμός, που είναι και ο πιο δημοφιλής σήμερα, απαρτίζεται από την αναπαράσταση ενός χρήστη (μέσω του προφίλ του), τις συνδέσεις του με άλλα προφίλ, κοινωνικούς δεσμούς και από μια ποικιλία πρόσθετων υπηρεσιών.

Με την διάδοση του Internet τα κοινωνικά δίκτυα χρησιμοποίησαν online υποδομές και γιγαντώθηκαν τόσο που πλέον ο $3^{ος}$ άνωθεν ορισμός είναι και ο ορισμός τους. Η έκρηξη στη δημοτικότητα τους είναι τέτοια που τα εκατομμύρια των χρηστών τους βαραίνουν τις υποδομές των υπηρεσιών αυτών σε μεγάλο βαθμό. Λόγω αυτών των προβλημάτων, τα τελευταία χρόνια έχουν γίνει προσπάθειες να περιγραφούν και να αναλυθούν οι δομές και οι ιδιότητες τους. Η εφαρμογή των Τυπικών Μεθόδων μπορεί να προσφέρει μια βαθύτερη και καθαρότερη κατανόηση του συστήματος αποσαφηνίζοντας τις έννοιες, ενώ επαληθεύοντας τις ιδιότητές του, αυξάνει την εμπιστοσύνη των χρηστών στο σύστημα.

4.2.2 Σχετικές εργασίες

Δύο εργασίες σχετικές με την τυποποίηση κοινωνικών προτύπων είναι οι [63, 64]. Στην [63], οι συγγραφείς επιχειρούν να οπτικοποιήσουν

ένα κοινωνικό δίκτυο αλλά και να ελαττώσουν το μέγεθός του χρησιμοποιώντας Formal Concept Analysis. Γι αυτούς, ένα κοινωνικό δίκτυο είναι μια στατική δομή και εξετάζουν στιγμιότυπα, που έχουν ληφθεί κάποια συγκεκριμένη στιγμή, του συστήματος και παίρνουν αποτελέσματα που αφορούν τη στιγμή αυτή. Στην [64], οι συγγραφείς προτείνουν ότι οι τυπικές μέθοδοι μπορούν να παρέχουν μια λογική υποδομή για την έκφραση και την επιβολή πολιτικών ασφαλείας στα κοινωνικά δίκτυα. Παρέχουν μια αφηρημένη προδιαγραφή και υποστηρίζουν πως μία υλοποίηση ενός δικτύου μπορεί να δημιουργηθεί ακολουθώντας πιστά και αποδεδειγμένα τις οδηγίες της προδιαγραφής. Η εργασία αυτή παρουσιάζει πολλά πλεονεκτήματα, αλλά δεν διευκρινίζει τι συμβαίνει όταν νέοι χρήστες γράφονται στο σύστημα ή όταν ένα προφίλ διαγράφεται. Επίσης, δεν υπάρχει κανένας διαχωρισμός μεταξύ των προδιαγραφών της αρχιτεκτονικής του συστήματος και του χρήστη. Έτσι, η αλλαγή οποιασδήποτε προδιαγραφής μπορεί να σημαίνει ότι θα πρέπει να αλλάξει όλη η προδιαγραφή.

Χρησιμοποιούμε μια διαφορετική προσέγγιση στην τυποποίηση ενός κοινωνικού δικτύου: Ενώ μας ενδιαφέρει να διατηρήσουμε τις ιδιότητες διασφάλισης του ιδιωτικού απορρήτου, θέλουμε η τυποποίησή μας να εκφράζει το ότι ένα κοινωνικό δίκτυο είναι ένα σύστημα που αποτελείται από υποσυστήματα τα οποία αλληλεπιδρούν. Υποστηρίζουμε ότι κάθε ένας από τους χρήστες σε ένα τέτοιο δίκτυο θα πρέπει να μοντελοποιηθεί ως ένα μεμονωμένο σύστημα, που μπορεί να αλλάξει την κατάσταση του συστήματος ανεξάρτητα και σε συνδυασμό με τους άλλους χρήστες. Επιπλέον, το ίδιο το δίκτυο είναι ένα σύστημα του οποίου οι καταστάσεις μπορούν να αλλάξουν όχι μόνο από τις αλλαγές των καταστάσεων των χρηστών, αλλά και από ενέργειες που συμβαίνουν στο συνολικό δίκτυο, όπως η εισαγωγή νέων χρηστών, ή αλλαγές στις προεπιλεγμένες ρυθμίσεις προστασίας του ιδιωτικού απορρήτου από τους διαχειριστές του δικτύου και ούτω καθεξής. Η προσέγγισή μας μοντελοποιεί ένα κοινωνικό δίκτυο ως ένα σύστημα πολλών συστημάτων, δίνοντας έτσι μια νέα προοπτική για την τυποποίηση των δικτύων αυτών. Αυτή η προδιαγραφή είναι γραμμένη σε modules. Στη συνέχεια, διερευνούμε τις ιδιότητες που μπορούν να επαληθευτούν με αυτή την προσέγγιση.

Η προδιαγραφή δεν έχει στόχο να αποτελέσει προδιαγραφή κάποιου συγκεκριμένου κοινωνικού δικτύου αλλά στοχεύει στο να δείξει πώς μοντελοποιείται ένα σύστημα από το μηδέν και κυρίως να δείξει μερικά από τα οφέλη αυτής της προσέγγισης.

4.2.3 Το Κοινωνικό Δίκτυο σαν OTS

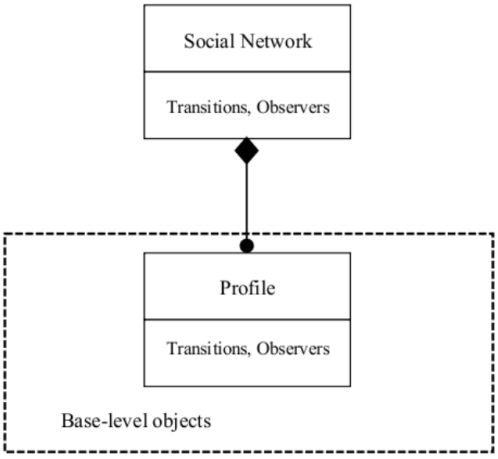
Στο κεφάλαιο αυτό θα περιγράψουμε ένα κοινωνικό δίκτυο ως ένα OTS (Κεφάλαιο 2.3.2) στην CafeOBJ, με σύνθεση συμπεριφοριακών αντικειμένων (Κεφάλαιο 2.3.4). Συγκεκριμένα, για την προδιαγραφή του συστήματος αυτού χρησιμοποιούμε τη Δυναμική Παράλληλη Σύνθεση με Συγχρονισμό. Φυσικά, αντί για την CafeOBJ θα μπορούσαμε να χρησιμοποιήσουμε οποιαδήποτε άλλη γλώσσα Αλγεβρικών Προδιαγραφών, όπως π.χ. την Maude [65] ή την CASL [66].

Στόχος είναι να προδιαγράψουμε το δίκτυο σαν ένα σύστημα υποσυστημάτων (σύνθετων αντικειμένων). Αυτό μας επιτρέπει να εξετάσουμε τόσο τη συμπεριφορά μεμονωμένων αντικειμένων ανεξάρτητα αλλά και τη συμπεριφορά ολόκληρου του συστήματος, επιτρέποντας μας να απομονώσουμε πιθανά σχεδιαστικά λάθη.

Ως *base-level object* χρησιμοποιούμε OTSs που αντιπροσωπεύουν τα προφίλ των χρηστών· κάθε $Profile =< O, I, T >$ είναι ένας χρήστης του δικτύου. Το δίκτυο εκφράζεται σαν ένα σύνθετο OTS, $SN =< O' \cup \{p\}, I', T' \cup \{a, d\} >$, όπου $I' \subseteq Y','$ ένα σύνολο από μεταβάσεων και a, d δύο ειδικά transitions που προσθέτουν και αφαιρούν αντίστοιχα ένα προφίλ στο Social Network OTS. Θα μπορούσαμε να δημιουργήσουμε και άλλα τέτοια transitions του συνολικού δικτύου που θα αντιστοιχούσαν σε ενέργειες όπως η αλλαγή της προεπιλεγμένης πολιτικής ιδιωτικότητας από τους διαχειριστές του δικτύου, ή στην εισαγωγή ενός νέου τύπου φιλίας μεταξύ χρηστών, κ.τ.λ. Εστιάσαμε μόνο στην εισαγωγή/διαγραφή χρήστη καθώς είναι δύο ενέργειες απαραίτητες σε κάθε κοινωνικό δίκτυο.

Τέλος, το O' είναι το σύνολο των παρατηρητών και p είναι ο παραμετρικός τελεστής προβολής που μας μεταφέρει από μια κατάσταση του συνολικού δικτύου στην αντίστοιχη κατάσταση ενός προφίλ. Αυτό ορίζεται τυπικά ως: $p : Y' D_{i1}, \ldots D_{in} \rightarrow Y$, με $p(X_{i1}, ..., X_{in}, H') = H$. Στην δεύτερη εξίσωση, τα X_{ik} είναι οι απαραίτητες μεταβλητές των ορατών συνόλων, H' είναι η κατάσταση του σύνθετου αντικειμένου και H είναι η κατάσταση των base-level objects. Το Σχήμα 7 περιγράφει όλα αυτά.

Ο χώρος καταστάσεων του συστήματος Y ορίζεται στην CafeOBJ σα κρυφό σύνολο ενώ κάθε παρατηρητής ορίζεται με κάποιον τελεστή παρατήρησης. Αν υποθέσουμε ότι τα ορατά σύνολα V_{ij} και V αντιστοιχούν στους τύπους δεδομένων D_k και D, όπου $k = i_1, \ldots, i_m$, τότε ο τελεστής παρατήρησης που δηλώνει τα $o_{i1,\ldots,im}$ ορίζεται ως $op\ o : V_{i1} \ldots V_{im}\ H \rightarrow V$. Κάθε κατάσταση στο I δηλώνεται με μία σταθερά κρυφού τύπου (π.χ. $init$) και ορίζεται ως $op\ init : \rightarrow H$. Μία μετάβαση $\tau_{j1,\ldots,jn} \in T$ δηλώνεται με τελεστή μετάβασης: $bop\ \tau : V_{j1} \ldots V_{jm}\ H \rightarrow H$, με το V_k



Σχήμα 7: Ιεραρχική σύνθεση συμπεριφοριακών αντικειμένων για το OTS του Κοινωνικού Δικτύου σε UML

να είναι οι ορατοί τύποι που αντιστοιχούν στους τύπους δεδομένων D_k και $k = j_1, \ldots, j_n$.

Κάθε μετάβαση ορίζεται περιγράφοντας τι συμβαίνει στις τιμές που επιστρέφουν οι παρατηρητές όταν μία κατάσταση μεταβεί στη διάδοχη της u με την εφαρμογή της $\tau_{j1,\ldots,jn}$, εφόσον η μετάβαση $\tau_{j1,\ldots,jn}$ μπορεί να συμβεί, δηλαδή εφόσον ισχύει το $c - \tau_{j1,\ldots,jn}(u)$. Αυτό εκφράζεται στην CafeOBJ με τη λέξη-κλειδί *ceq*. Η τιμή που επιστρέφουν οι $o_{i1,\ldots,im}$ δεν αλλάζει αν η $tau_{j1,\ldots,jn}$ εφαρμοστεί σε μία κατάσταση u στην οποία $\neg c - \tau_{j1,\ldots,jn}(u)$.

Στον πίνακα 2 παρουσιάζουμε τον ορισμό του Κοινωνικού OTS σε σημειολογία της CafeOBJ. Τα H και H' είναι οι κρυφοί τύποι των βασικών αντικειμένων και του σύνθετου αντικειμένου αντίστοιχα. Το V_{ij} υποδηλώνει τους ορατούς τύπους D_{ij} και τέλος, το V_1 είναι ένας ορατός τύπος που αντιπροσωπεύει τη μοναδική ταυτότητα του βασικού αντικειμένου (π.χ. κάποιο ID).

4.2.4 Αλγεβρική Προδιαγραφή ενός Κοινωνικού Δικτύου

Το κύριο δομικό κομμάτι ενός συστήματος κοινωνικού δικτύου είναι το προφίλ του χρήστη (*User Profile*). Ένα προφίλ ορίζει το χρήστη πίσω από αυτό μέσω ενός συνόλου δεδομένων όπως τις *συλλογές φωτογραφιών* τους, τον *τοίχο* τους, το *γραμματοκιβώτιο* τους, κ.ά. και σχετικές ενέργειες. Τα προφίλ αυτά συνδέονται μεταξύ τους με μία (προσεταιρι-

Παρατηρητές του OTS	**CafeOBJ Notation**
απλοί παρατηρητές: $o : Y' \rightarrow D$	*bop o :* $V_{i1} \ldots V_{im}$ $H' \rightarrow V$.
παρατηρητές προβολής: $p : Y' \rightarrow Y$	*bop p :* $V_{i1} \ldots V_{im}$ $H' \rightarrow H$.
απλές μεταβάσεις: $\tau : Y' \rightarrow Y'$	*bop* τ : $V_{i1} \ldots V_{im}$ $H' \rightarrow H'$.
ειδικές μεταβάσεις: α *:* $Y' \rightarrow Y'$, d : $Y' \rightarrow Y'$	*bop a :* V_1 $H' \rightarrow H'$. *bop d :* V_1 $H' \rightarrow H'$.
συνθήκες κανόνων μετάβασης: $c_{\tau j1,\ldots,hn}$	*op c-*$\tau_{j1,\ldots,jn}$: $V_{i1} \ldots V_{im}$ $H' \rightarrow Bool$.

Πίνακας 2: Το Κοινωνικό OTS σε όρους της CafeOBJ

στική) σχέση που συνήθως αποκαλείται *φιλία* και που επιτρέπει σε ένα χρήστη να δει ή να τροποποιήσει τα δεδομένα κάποιου άλλου χρήστη. Οι αλλαγές αυτές στα παρατηρούμενα δεδομένα αντιστοιχούν σε αλλαγές στις καταστάσεις του OTS σε αυτό το κρυφό αλγεβρικό πλαίσιο.

Το πρώτο βήμα για να μοντελοποιήσουμε ένα κοινωνικό δίκτυο σαν OTS είναι να προδιαγράψουμε αυτούς τους τύπους δεδομένων σαν ορατούς τύπους στα modules της CafeOBJ. Μετά, όταν προδιαγράψουμε τον κρυφό τύπο (που αντιπροσωπεύει το σύστημα ενός προφίλ χρήστη), τα προηγούμενα modules εισάγονται. Για παράδειγμα, ο αφηρημένος τύπος δεδομένων του *τοίχου* αντιπροσωπεύει ένα ανοιχτό χώρο για τους χρήστες να δημοσιεύουν περιεχόμενο στο προφίλ κάποιου χρήστη. Το περιεχόμενο (Content) είναι μια τριάδα· η ταυτότητα του χρήστη που το δημοσίευσε, ένα μοναδικό αναγνωριστικό για το περιεχόμενο και τέλος, το ίδιο το περιεχόμενο. Το module της CafeOBJ που ορίζει τον τοίχο κάποιου χρήστη φαίνεται στον Κώδικα 17. Οι υπόλοιποι τύποι δεδομένων ορίζονται με παρόμοιο τρόπο.

```
mod! WALLCONTENT
 { pr(CONTENT + ACCOUNTID+Nat + LIST3)
 [Wallcontent]

 op _&_&_ : Accountid Nat Content -> Wallcontent
 op Id? : Wallcontent -> Nat
 op Accountid? : Wallcontent -> Accountid
 op Like? : Wallcontent -> ListofAccountid }
```

Κώδικας 17: Ο τοίχος του χρήστη

4.2.4.1 Προδιαγραφή του χρήστη σα σύστημα OTS

Στη συνέχεια, προδιαγράφουμε τον ίδιο το χρήστη σε module που ονομάζουμε *Profile*. Ορίζει ένα OTS που ο χώρος καταστάσεών του ορίζεται από τον κρυφό τύπο *ProfileSys*. Οι παρατηρητές που ορίζουν τις καταστάσεις αυτού του OTS φαίνονται στον Πίνακα 3. Στο προφίλ έχουμε μεταβάσεις που αλλάζουν την κατάσταση του OTS. Οι μεταβάσεις:

- `receivefriendrequest`,
- `acceptfriendrequest` και
- `ignorefriendrequest`

αντιπροσωπεύουν την λήψη, αποδοχή ή άρνηση ενός αιτήματος φιλίας αντίστοιχα. Οι μεταβάσεις:

- `receivelike`
- `receive` και
- `changetype`

αντιπροσωπεύουν: τη λήψη "έγκρισης" από κάποιο χρήστη για το περιεχόμενο ενός τοίχου, τη λήψη περιεχομένου από κάποιο προφίλ (είτε κάποιο μήνυμα, είτε κάποιο περιεχόμενο στον τοίχο, κ.τ.λ.) και ότι ο ιδιοκτήτης του προφίλ άλλαξε τις ρυθμίσεις ορατότητας. Οι μεταβάσεις που αλλάζουν τις καταστάσεις αυτού του OTS φαίνονται στον Πίνακα 4.

Περιγραφή	**Παρατηρητές OTS**	**Κώδικας CafeOBJ**
Ο τοίχος ενός χρήστη	wall: $Y \rightarrow D_1$	bop wall: ProfileSys → Walllist
Το γραμματοκιβώτιο ενός χρήστη	inbox: $Y \rightarrow D_1$	bop inbox: ProfileSys → Inboxlist
Η συλλογή φωτογραφιών	photoalbum: $Y \rightarrow D_1$	bop photoalbum: ProfileSys → Photolist
Ένα σύνολο από ID χρηστών στους οποίους άρεσε μία ανάρτηση/φωτογραφία κ.τ.λ.	likeset: $YD_1D_2D_3 \rightarrow D_4$	bop likeset: ProfileSys Nat Placeholder → SetofAccountid
Μία λίστα από αιτήματα φιλίας	requests: $Y \rightarrow D_1$	bop requests: ProfileSys → ListofAccountid
Μια λίστα από ID χρηστών που συνδέονται με σχέση φιλίας με ένα χρήστη	friends: $Y \rightarrow D_1$	bop friends: ProfileSys → ListofAccountid
Το μοναδικό αναγνωριστικό (ID) ενός προφίλ/χρήστη	myid: $Y \rightarrow D_1$	bop myid: ProfileSys → Accountid
Η ρύθμιση ιδιωτικότητας του προφίλ	type: $Y \rightarrow D_1$	bop type: ProfileSys → Policy
Μια λίστα από ID χρηστών στους οποίους το προφίλ είναι προσβάσιμο	speciallist: $Y \rightarrow D_1$	bop speciallist: ProfileSys → ListofAccountid
Μία λίστα που περιέχει τους φίλους των φίλων ενός χρήστη	friendsoffriends: $Y \rightarrow D_1$	bop friendsoffriends: ProfileSys → ListofAccountid
Επιστρέφει true αν ένας χρήστης μπορεί να δει μια φωτογραφία	view3: $YD_1D_2 \rightarrow D_3$	bop view3: ProfileSys Accountid Picture → Bool
Μία λίστα χρηστών που έχουν περιορισμένη πρόσβαση στο προφίλ ενός χρήστη	speciallist2: $Y \rightarrow D_1$	bop speciallist2: ProfileSys → ListofAccountid

Πίνακας 3: Παρατηρητές που ορίζουν το OTS του προφίλ

Περιγραφή	**Μεταβάσεις OTS**	**Κώδικας CafeOBJ**
Λήψη αιτήματος φιλίας από χρήστη	$receivefriend : Y\ D_1 \rightarrow Y$	*bop receivefriendrequest: ProfileSys Accountid* $\rightarrow ProfileSys$
Αποδοχή αιτήματος φιλίας από χρήστη	$acceptfriend : Y\ D_1 \rightarrow Y$	*bop acceptfriendrequest: ProfileSysAccountid* $\rightarrow ProfileSys$
Αγνόηση αιτήματος φιλίας από χρήστη	$ignorefriend : Y\ D_1 \rightarrow Y$	*bop ignorefriendrequest: ProfileSysAccountid* $\rightarrow ProfileSys$
Ένας χρήστης εγκρίνει περιεχόμενο	$reclike : Y\ D_1\ D_2\ D_3 \rightarrow Y$	*bop receivelike: ProfileSys PlaceholderNat Accountid* $\rightarrow ProfileSys$
Ένας χρήστης λαμβάνει περιεχόμενο από άλλο χρήστη	$receive : Y\ D_1\ D_2\ D_3 \rightarrow Y$	*bop receive: Content Accountid Placeholder ProfileSys* $\rightarrow ProfileSys$
Ένας χρήστης βλέπει τη λίστα φίλων άλλου χρήστη	$viewfriends : Y\ D_1 \rightarrow Y$	*bop viewfriends: ProfileSys Accountid* $\rightarrow ProfileSys$
Ένας χρήστης βλέπει τις φωτογραφίες άλλου χρήστη	$viewphoto : Y\ D_1 \rightarrow Y$	*bop viewphoto: ProfileSys Accountid* $\rightarrow ProfileSys$
Ένας χρήστης αλλάζει τις ρυθμίσεις ιδιωτικότητας του προφίλ του	$changetype : Y\ D_1\ D_2 \rightarrow Y$	*bop changetype: ProfileSys Accountid Case* $\rightarrow ProfileSys$

Πίνακας 4: Μεταβάσεις που ορίζουν το Profile OTS

Το αποτέλεσμα της εφαρμογής της μετάβασης `acceptfriendrequest` σε μία κατάσταση S δίνεται σε σημειογραφία της CafeOBJ στον Κώδικα 18. Οι αριθμοί στην αρχή των γραμμών δεν είναι μέρος του κώδικα. Οι γραμμές 1 και 2 δείχνουν την υπογραφή και τον ορισμό της αναγκαίας συνθήκης της μετάβασης. Η γραμμή 3 λέει ότι στην περίπτωση που η αναγκαία συνθήκη ισχύει για κάποιο τυχαίο χρήστη ($A1$), τότε αυτός ο χρήστης προστίθεται στη λίστα των φίλων του προφίλ. Η γραμμή 4 λέει ότι ο χρήστης $A1$ δε θα ανήκει πια στη λίστα με τα εκκρεμή ζητήματα φιλίας. Στις γραμμές 5 και 6 ορίζουμε τι θα συμβεί αν ένας χρήστης ($A3$) επιχειρήσει να δει μια φωτογραφία (Pi) του χρήστη $A1$ αφού ο χρήστης $A1$ δεχτεί αίτημα φιλίας από τον $A3$. Η γραμμή 6

ορίζει ότι ο $A3$ θα είναι πλέον σε θέση να δει την Pi, αν η φωτογραφία αυτή υπάρχει στη συλλογή φωτογραφιών του $A1$ και είτε ο $A1$ είναι ο ίδιος χρήστης με τον $A3$ είτε ο $A3$ ανήκει στη λίστα των φίλων του $A1$ στην κατάσταση S και το προφίλ του $A1$ είναι ανοιχτό μόνο σε φίλους. Διαφορετικά, αν το προφίλ είναι ανοιχτό και στους φίλους των χρηστών που περιλαμβάνονται στη λίστα φίλων του προφίλ και ο $A3$ είναι φίλος φίλου του $A1$ στην κατάσταση S ή ο $A3$ είναι φίλος του $A1$. Σε κάθε άλλη περίπτωση ο $A3$ μπορεί να δει τη Pi μόνο αν μπορούσε ήδη να τη δει στην κατάσταση S.

```
1. op c−acceptfriendrequest : Accountid ProfileSys −> Bool
2. eq c−acceptfriendrequest(A1,S)= A1 //in requests(S).
3. ceq friends(acceptfriendrequest(A1,S))= A1|friends (S) if
        c−acceptfriendrequest(A1,S) .
4. ceq requests(acceptfriendrequest(A1,S)) = (requests (S) \ A1) if
        c−acceptfriendrequest(A1,S) .
5. ceq view3(acceptfriendrequest(A1,S),A3,Pi) = view3(S, A3,Pi) if
        c−acceptfriendrequest(A1,S) and (Pi/in photoalbum(S))
                and not(((type(S)=friend) and (A3=A1))
                                or((type(S) = friend) and (A3 //in friends(S)))
                                or ((type(S) = ffriend) and ((A3 = A1)
                                or (A3 //in friends(S))
                                or (A3 //in flist(A1))
                                or (A3 //in friendsoffriends(S))))
                        ).
6. ceq view3(acceptfriendrequest(A1,S),A3,Pi) = true if
        c− acceptfriendrequest(A1,S) and (Pi /in photoalbum (S))
                and (((type(S) = friend) and (A3 = A1))
                        or ((type(S) = friend) and (A3 //in friends(S)))
                        or ((type(S) = friend) and ((A3 = A1)
                        or (A3 //in friends(S)) or (A3 //in flist(A1))
                        or (A3 //in friendsoffriends(S))))
                ).
```

Κώδικας 18: Μέρος του ορισμού της `acceptfriendrequest`

Περιγραφή	**Παρατηρητές του OTS**	**Κώδικας CafeOBJ**
Σύνολο των προφίλ του δικτύου	accounts: Y $\rightarrow D_1$	bop accounts: Sys $\rightarrow$ Setofaccountid
Λίστα των αναγνωριστικών (IDs) λογαριασμών που έχουν αναφερθεί ως ύποπτα	susplist: Y $\rightarrow D_1$	bop suspList: Sys $\rightarrow$ ListofAccountid
Λίστα με τα μπλοκαρισμένα προφίλ	banned: Y $\rightarrow D_1$	banned: Sys $\rightarrow$ ListofAccountid
Φορές που ένα προφίλ έχει δεχθεί αναφορά	numberofreports: Y $D_1 \rightarrow D_2$	bop numberofreports: Sys Accountid $\rightarrow$ Nat
Προβολή από την κατάσταση του δικτύου σε αυτή του προφίλ	profile: D_1 Y $\rightarrow$ Y'	bop profile: Accountid Sys $\rightarrow$ ProfileSys

Πίνακας 5: Παρατηρητές του OTS ενός Κοινωνικού Δικτύου

4.2.4.2 Προδιαγραφή για το OTS ενός Κοινωνικού Δικτύου

Στη συνέχεια μπορούμε να παρουσιάσουμε τη προδιαγραφή ολόκληρου του Κοινωνικού Δικτύου, ως ένα δυναμικό συμπεριφοριακό αντικείμενο, όπως αυτό περιγράφηκε στην ενότητα 2.3.4. Θα είναι ένα σύστημα που δυναμικά δημιουργεί και διαγράφει άλλα υποσυστήματα, όπως τους χρήστες. Το σύνθετο αντικείμενο θα αναπαριστά το Κοινωνικό Προφίλ που δυναμικά περιλαμβάνει πολλά συστήματα OTS για τα προφίλ, ως base-level αντικείμενα. Οι παρατηρητές του OTS ενός Κοινωνικού Δικτύου φαίνονται στον Πίνακα 5. Οι αντίστοιχες μεταβάσεις φαίνονται στον Πίνακα 6. Οι μεταβάσεις περιλαμβάνουν τις `receiveFB`, `receivefriendFB`, `acceptfriendFB`, `ignorefriendFB`, `receivelikeFB` και `changetypeFB`. Η σημασιολογία τους είναι ότι δοσμένου ενός προφίλ (που αντιπροσωπεύεται από τον τύπο `Accountid`, όπως αυτός εμφανίζεται στα ορίσματα), αντιστοιχούν σε ενέργειες στο OTS των προφίλ. Αυτό επιτυγχάνεται με χρήση προβολών.

Περιγραφή	**Μεταβάσεις του OTS**	**Κώδικας CafeOBJ**
Δημιουργία νέου προφίλ	add: Y $D_1 \rightarrow Y$	bop add: Sys Accountid $\rightarrow Sys$
Διαγραφή ενός προφίλ	delete : Y $D_1 \rightarrow Y'$	bop del : Sys Accountid $\rightarrow Sys$
Ένα προφίλ (π.χ. id1) λαμβάνει περιεχόμενο από άλλο προφίλ (π.χ. id2)	receiveFB: D_1 D_2 D_3 D_4 $Y \rightarrow Y$	bop receiveFB: Accountid Content Accountid Placeholder Sys $\rightarrow Sys$
Ένα προφίλ id1 δέχεται αίτημα φιλίας από ένα προφίλ id2	receivefriendFB: D_1 D_2 $Y \rightarrow Y$	bop receivefriendFB: Accountid Accountid Sys $\rightarrow Sys$
Το προφίλ id1, δέχεται το αίτημα φιλίας του id2	acceptfriendFB: D_1 D_2 $Y \rightarrow Y$	bop acceptfriendFB: Accountid Accountid Sys $\rightarrow Sys$
Το προφίλ id2 εγκρίνει κάποιο περιεχόμενο του id1	receivelikeFB: D_1 D_2 D_3 D_4 $Y \rightarrow Y$	bop receivelikeFB: Accountid Placeholder Nat Accountid Sys $\rightarrow Sys$

Το προφίλ id2, βλέπει τις φωτογραφίες του id1	*viewphotoFB:* D_1 D_2 Y $D_3 \rightarrow Y$	bop viewphotoFB: Accountid Accountid Sys Picture $\rightarrow Sys$
Το προφίλ id2, "σημειώνει" το χρήστη id1 στη φωτογραφία C1	*tagFB :* D_1 D_2 D_3 D_4 D_5 $Y \rightarrow Y$	bop tagFB : Accountid Accountid Content Placeholder Nat Sys $\rightarrow Sys$
Ο χρήστης αλλάζει τον τύπο του λογαριασμού του	*changetypeFB:* D_1 $D_2 \rightarrow Y$	bop changetypeFB: Accountid Accountid Sys $\rightarrow Sys$
Ο χρήστης 1 αναφέρει το χρήστη 2	*reportuserFB:* D_1 $D_2 \rightarrow Y$	bop reportuserFBn: Accountid Accountid Sys $\rightarrow Sys$
Ο διαχειριστής του συστήματος επιβεβαιώνει ότι κάποιο προφίλ που έχει αναφερθεί ως ύποπτο, είναι όντως	*confirm:* D_1 $Y \rightarrow Y$	bop confirm: Accountid Sys $\rightarrow Sys$
Ο διαχειριστής του συστήματος βγάζει κάποιο προφίλ που έχει αναφερθεί από τη λίστα με τα ύποπτα	*reject:* D_1 $Y \rightarrow Y$	bop reject: Accountid Sys $\rightarrow Sys$

Πίνακας 6: Μεταβάσεις του OTS για το κοινωνικό δίκτυο

Για να έχουμε ένα πλήρως λειτουργικό κοινωνικό δίκτυο χρειαζόμαστε επιπλέον μεταβάσεις, οι οποίες δεν έχουν αντίστοιχες στα base-level αντικείμενα. Μερικές από αυτές είναι οι:

- `add`: προσθήκη ενός προφίλ στο δίκτυο
- `del`: διαγραφή ενός προφίλ από το δίκτυο
- `tagFB`: μαρκάρισμα ενός χρήστη σε μία φωτογραφία άλλου χρήστη
- `reportuserFB`: αναφορά χρήστη ως ύποπτου

- `reportphotoFB`: αναφορά μίας φωτογραφίας ως ακατάλληλης
- `confirm`: επιβεβαίωση από το διαχειριστή του συστήματος ότι μία φωτογραφία είναι ακατάλληλη και απόρριψή της

Οι μεταβάσεις `add` και `del` είναι ιδιαίτερα σημαντικές καθώς ευθύνονται για την αλλαγή της διάταξης του συστήματος.

1. **eq** c−add(A1,S) = **not** (A1 /in accounts(S)) .
2. **ceq** profile(A2,add(A1,S))= init **if** (A1 = A2) **and** c−add(A1,S) .
3. **ceq** profile(A2,add(A1,S))= profile(A2,S) **if** (**not** (A1 = A2))
   and c−add(A1,S) .
4. **ceq** accounts(add(A1,S))= (A1 U accounts(S)) **if** c−add(A1,S) .

Κώδικας 19: Η μετάβαση `add` του OTS του κοινωνικού δικτύου

Όταν ένας νέος χρήστης προστίθεται στο σύστημα, αναπαρίσταται ως νέο ProfileOTS στην αρχική του κατάσταση. Τα αποτελέσματα της ενέργειας `add` σε μία τυχαία κατάσταση S φαίνονται στον Κώδικα 19. Η γραμμή 1 είναι η αναγκαία συνθήκη της μετάβασης `add` και λέει πως για να προστεθεί ένα προφίλ στο κοινωνικό δίκτυο, το προφίλ αυτό δε θα πρέπει να ανήκει ήδη στο δίκτυο. Οι γραμμές 2 και 3 περιγράφουν πώς η κατάσταση του προφίλ A_2 επηρεάζεται από την προσθήκη του προφίλ A_1 στο δίκτυο, εφόσον η αναγκαία συνθήκη του `add` ισχύει: Στη γραμμή 2, αν το A_1 είναι το ίδιο με το A_2 τότε η κατάσταση του A_2 θα είναι η αρχική κατάσταση του OTS του προφίλ. Αυτό είναι μια προβολή, καθώς από μία κατάσταση του σύνθετου OTS (την κατάσταση που πήραμε από την εφαρμογή του `add(A1,S)`) πήραμε μία κατάσταση για το δομικό αντικείμενο, την αρχική του κατάσταση. Στη γραμμή 3 περιγράφουμε τι συμβαίνει αν τα προφίλ A_1 και A_2 είναι διαφορετικά· στην περίπτωση αυτή το A_2 δεν επηρεάζεται. Τέλος, στη γραμμή 4 περιγράφουμε την προσθήκη του προφίλ A_1 στη λίστα με τα προφίλ που περιέχει το δίκτυο. Οι υπόλοιποι παρατηρητές δεν επηρεάζονται από την προσθήκη κάποιου προφίλ στο δίκτυο.

Είναι επίσης ενδιαφέρον να δούμε πώς αλλαγές στην κατάσταση ενός αντικειμένου μπορούν να αλλάξουν την κατάσταση άλλου αντικειμένου. Αν για παράδειγμα ο χρήστης A_1 δεχτεί αίτημα φιλίας από το προφίλ A_2 είναι εύκολο να δούμε πώς η κατάσταση του A_1 αλλάζει αφού ο A_2 προστίθεται στη λίστα των φίλων του A_1. Όμως η σχέση φιλίας στην προδιαγραφή μας είναι αμφίδρομη, άρα η άλλη αλλαγή που θα συμβεί είναι ότι στη λίστα φίλων του A_2 θα προστεθεί ο A_1. Έτσι, μία αλλαγή στην κατάσταση ενός OTS μπορεί να αλλάξει την κατάσταση και άλλου, δηλαδή οι παρατηρητές των καταστάσεων των δύο Profile OTS θα πρέπει να αλλάξουν, όπως φαίνεται στον Κώδικα 20

```
1. ceq profile(A3,acceptfriendFB(A1,A2,S))= acceptfriendrequest(A2,
   profile(A1,S)) if c−acceptfriendFB (A1,A2,S) and (A1 = A3) .
2. ceq profile(A3,acceptfriendFB(A1,A2,S))= acceptfriendrequest(A1,
   profile(A2,S)) if c−acceptfriendFB (A1,A2,S) and (A1 = A3) .
```

Κώδικας 20: Η αποδοχή αιτήματος φιλίας

Έστω ότι ο χρήστης A_1 μαρκάρει το χρήστη A_2 σε μία φωτογραφία. Τότε, αυτόματα, η φωτογραφία αυτή θεωρείται ότι ανήκει και στον A_2. Αυτό φαίνεται σε κώδικα CafeOBJ στον Κώδικα 21. Οι γραμμές 1 και 2 ορίζουν την αναγκαία συνθήκη για τη μετάβαση αυτή· ένας χρήστης μπορεί να μαρκάρει άλλο χρήστη σε κάποιο περιεχόμενο αν το περιεχόμενο αυτό είναι φωτογραφία και βρίσκεται στη συλλογή φωτογραφιών του. Η γραμμή 3 λέει πως αν ο χρήστης A_2 μαρκαριστεί από το χρήστη A_1 τότε η κατάσταση του A_2 αλλάζει και γίνεται ίδια με την κατάσταση στην οποία ο A_2 λαμβάνει τη φωτογραφία αυτή ως περιεχόμενο και την αποθηκεύει στη συλλογή του. Αυτό επιτυγχάνεται δηλώνοντας ότι η προβαλλόμενη κατάσταση του A_2 αλλάζει σε `receive(C1,A2,PH,profile(A2,S))`. Η γραμμή 4 λέει ότι το προφίλ δεν αλλάζει κατάσταση αν η αναγκαία συνθήκη δεν ισχύει.

```
1. op c−tagFB : Accountid Accountid Placeholder Nat Content Sys −> Bool
2. eq c−tagFB(A1,A2,PH,N,C1,S) = (A1 /in accounts(S))
        and (A2 /in accounts(S)) and ( placetype?(PH) = phototype )
        and ( type?(C1) = picture) and (A1 & N & C1
                //in photoalbum(profile(A1,S)) ) .
3. ceq profile(A3 ,tagFB(A1 ,A2 ,C1,PH, N,S)) =
        receive(C1,A2,PH, profile(A2 , S))
                if (c−tagFB(A1,A2,PH,N,C1,S) and (A3 = A2)) .
4. ceq profile(A3 ,tagFB(A1 ,A2 ,C1,PH,N, S)) = profile(A3 , S)
        if (not(A3 = A1) and c−tagFB(A1,A2,PH,N,C1,S) ) .
```

Κώδικας 21: Μέρος της προδιαγραφής της μετάβασης tagFB του OTS του κοινωνικού δικτύου

4.2.5 Επαλήθευση

Αφού προδιαγράψαμε το σύστημα μας στην CafeOBJ, μπορούμε τώρα να χρησιμοποιήσουμε τη μεθοδολογία των Proof Scores (Κεφάλαιο 2.3.3) για να ελέγξουμε κατά πόσο το σύστημά μας ικανοποιεί κάποιες επιθυμητές ιδιότητες. Θα επιχειρήσουμε να αποδείξουμε την ισχύ των Ιδιοτήτων 4.6 και 4.7.

Ιδιότητα 4.6. *invariant 1: αν ένας χρήστης έχει αναφερθεί πάνω από N φορές, το προφίλ του μαρκάρεται ως ύποπτο*

Ιδιότητα 4.7. *invariant 2: ένας χρήστης μπορεί να δει τις φωτογραφίες άλλου χρήστη μόνο αν ικανοποιεί την πολιτική ιδιωτικότητας του $2^{ου}$ χρήστη*

4.2.5.1 Απόδειξη της Ιδιότητας 4.6

Η απόδειξη της ιδιότητας γίνεται σε 4 βήματα:

1. Γράφουμε την Ιδιότητα 4.6 σαν κατηγόρημα, (*invariant pred(p,x)*), όπου p είναι μια ελεύθερη μεταβλητή για καταστάσεις και η x συμβολίζει άλλες ελεύθερες μεταβλητές του $pred$. Μετά γράφουμε το $pred(p, x)$ σε ορολογία CafeOBJ σε ένα module (Κώδικας 22).

```
op inv1 : Sys Accountid -> Bool
inv1(S,A) = not((numberofreports(S,A) <= n1) and (A //in suspList(S))) .
```

Κώδικας 22: Η ιδιότητα 4.6 σε όρους CafeOBJ

2. Αποδεικνύουμε ότι το κατηγόρημα ισχύει για μία αρχική κατάσταση (`init`) εκτελώντας το `pred(init,x)` και περιμένοντας την CafeOBJ να μας απαντήσει true (Κώδικας 23).

```
open INV
red inv1(initFB,a) .
close
```

Κώδικας 23: Η αρχική κατάσταση

3. Γράφουμε το επαγωγικό βήμα, χρησιμοποιώντας δύο καταστάσεις: την τυχαία κατάσταση s και τη διάδοχή της s'. Αν η `inv1` ισχύει στην κατάσταση s θα πρέπει να ισχύει και στην s'. Ο Κώδικας 22 δείχνει το επαγωγικό βήμα. Η κατάσταση s' είναι η κατάσταση που προκύπτει μετά την εφαρμογή κάποιου κανόνα μετάβασης (Κώδικας 24).

```
op istep1 : -> Bool
eq istep1 = inv1(s,a) implies inv1(s',a) .
```

Κώδικας 24: Το επαγωγικό βήμα

4. Τέλος, γράφουμε ένα proof score για κάθε κατάσταση. Για παράδειγμα, για την κατάσταση s' του συστήματος που προκύπτει από την εφαρμογή του κανόνα μετάβασης `receivelikeFB`, έχουμε τον Κώδικα 25. Η CafeOBJ απαντάει με true που σημαίνει πως για την κατάσταση αυτή του συστήματος, η ιδιότητα `inv1` ισχύει.

```
open ISTEP
eq s' = receivelikeFB(a1,ph,n,a2,s) .
red istep1 .
close
```

Κώδικας 25: Ο τελεστής `receivelikeFB`

Όμως, για την κατάσταση του συστήματος που προκύπτει από την εφαρμογή της μετάβασης `reportuserFB`, η CafeOBJ δεν επιστρέφει true ή false, αλλά κάποια έκφραση που δεν μπόρεσε να αναγάγει.

```
open ISTEP
eq s' = reportuserFB(a1,a2,s) .
red istep1 .
close
```

Κώδικας 26: Ο τελεστής `reportuserFB`

Η εκτέλεση του Κώδικα 26 κολλάει γιατί η CafeOBJ δεν μπορεί να ανάγει την αναγκαία συνθήκη της μετάβασης `reportuserFB` σε true/false οπότε θα χρειαστεί να κάνουμε case splitting, δηλαδή να πάρουμε δύο περιπτώσεις ανάλογα με το αν η αναγκαία συνθήκη της `reportuserFB` δίνει true ή false. Το case splitting φαίνεται στον Πίνακα 7.

Η περίπτωση α του Πίνακα 7 δίνει true. Η περίπτωση β όμως δε δίνει ούτε true ούτε false. Αυτό συμβαίνει γιατί η αναγκαία συνθήκη είναι περίπλοκη. Στην περίπτωση αυτή θα πάρουμε τις δύο

α) αναγκαία συνθήκη ψευδής	**β) αναγκαία συνθήκη αληθής**
`open ISTEP`	`open ISTEP`
`eq c-reportuserFB(a1,a2,s) = false .`	`eq c-reportuserFB(a1,a2,s) = true .`
`eq s' = reportuserFB(a1,a2,s) .`	`eq s' = reportuserFB(a1,a2,s) .`
`red istep1 .`	`red istep1 .`
`close`	`close`

Πίνακας 7: Case splitting 1

υπο-συνθήκες που απαρτίζουν την αναγκαία συνθήκη (δηλαδή τις: $a1\ /in\ accounts(s)$) και την $a2\ /in\ accounts(s)$) και θα τις ορίσουμε ως true. Αν δε δουλέψει ούτε τώρα (όπως και συμβαίνει) τότε θα χρειαστεί να κάνουμε και άλλο διαχωρισμό περιπτώσεων.

(α′) Πρώτα εξετάζουμε την περίπτωση που $a = a2$. Σε αυτή την περίπτωση, χωρίζουμε σε υποπεριπτώσεις ανάλογα με το αν το $numberofreports(s, a) <= n1$ ισχύει ή όχι:

i. Αν δεν ισχύει η σχέση, η CafeOBJ απαντάει με true, άρα η ιδιότητα ισχύει για την περίπτωση αυτή.

ii. Αν ισχύει η σχέση θα χρειαστούμε άλλες δύο υποπεριπτώσεις ανάλογα με το ποιο είναι το αποτέλεσμα του $a\ //in\ suspList(s)$:

A′. Αν δεν ισχύει η σχέση, η CafeOBJ απαντάει με true.

B′. Αν ισχύει η σχέση θα χρειαστούμε άλλες δύο υποπεριπτώσεις ανάλογα με το αποτέλεσμα του `numberofreports (s,a2) <= n1`. Αν η CafeOBJ απαντήσει με true τότε η ιδιότητα ισχύει. Αν απαντήσει με false τότε είτε βρήκαμε μια περίπτωση που η ιδιότητα μας δεν ισχύει και θα πρέπει να βελτιώσουμε το σχεδιασμό του συστήματος είτε η περίπτωση αυτή αντιστοιχεί σε μία μη-προσβάσιμη κατάσταση, οπότε και θα την απορρίψουμε με κατάλληλο λήμμα (το 4.1).

Λήμμα 4.1. *eq inv3(K,L) = not(K <= L) implies not((K + 1) <= L) .*

Χρησιμοποιώντας το `inv3`, η περίπτωση αυτή δίνει true. Όλος ο κώδικας για την περίπτωση αυτή φαίνεται στον Κώδικα 27. Το Λήμμα 4.1 χρειάζεται να αποδειχθεί πριν χρησιμοποιηθεί και η απόδειξη του γίνεται με παρόμοιο τρόπο (σε δικό του module, με κατάλληλο case splitting).

(β′) Στην περίπτωση που $a \neq a2$ κάνουμε την ίδια διαδικασία, με case splitting και λήμματα όπου χρειάζεται. Η CafeOBJ απαντάει (τελικά) με true.

Τελικά δείξαμε ότι η `inv1` ισχύει για κάθε προσβάσιμη κατάσταση του συστήματος, άρα και η Ιδιότητα 4.6 θα ισχύει στο σύστημα που σχεδιάσαμε.

```
open ISTEP
 eq (a1 /in accounts(s)) = true .
```

```
 eq (a2 /in accounts(s)) = true .
 eq a = a2 .
 eq (numberofreports(s,a2) + 1 <= n1) = true .
 eq (a2 //in suspList(s)) = true .
 eq (numberofreports(s,a2) <= n1) = false .
 eq s' = reportuserFB(a1,a2,s) .
 red inv3(numberofreports(s,a2),n1) implies istep1 .
close
```

Κώδικας 27: Χρήση λήμματος για να απορρίψουμε μια περίπτωση που επιστρέφει false

4.2.5.2 Απόδειξη της Ιδιότητας 4.7

Η ιδιότητα 4.7 πιο αναλυτικά: Έστω μια φωτογραφία P στη συλλογή του χρήστη B και ένας χρήστης A. Ο A δεν μπορεί να δει την P εκτός αν ισχύει ένα από τα ακόλουθα:

- Ο τύπος του προφίλ του B είναι "δημόσιο"
- Το προφίλ του B είναι ανοιχτό στους φίλους του και ο A είναι φίλος του B
- Το προφίλ του B είναι ανοιχτό στους φίλους του και στους φίλους των φίλων του και ο A είναι είτε φίλος του B είτε φίλος κάποιου φίλου του B
- Το προφίλ του B είναι ανοιχτό σε μία ειδική λίστα ατόμων και ο A είναι μέλος αυτής της λίστας
- Ο B είναι ο A

Σε ορολογία CafeOBJ, η ιδιότητα 4.7 φαίνεται στον Κώδικα 28

```
inv2(S,A,P) = ((P /in photoalbum(S)) and not
        ( (type(S) = public)
        or ( (type(S) = friend) and (A //in friends(S)))
        or ((type(S) = ffriend) and ((A //in friendsoffriends(S)
        or (A //in friends(S)))))
        or (myid(S) = A)
        or ( (type(S) = splist) and (A //in speciallist(S)))
    ))
    implies not view3(S,A,P)
```

Κώδικας 28: Η ιδιότητα 4.7

Ακολουθώντας μεθοδολογία παρόμοια με αυτή που χρησιμοποιήσαμε στο Κεφάλαιο 4.2.5.1, βρίσκουμε ότι η Ιδιότητα 4.7 ισχύει για το OTS του προφίλ αλλά όχι για το OTS του δικτύου. Βρέθηκε μία περίπτωση κατά την εφαρμογή της μετάβασης `tagFB` όπου η CafeOBJ απάντησε με false ενώ η κατάσταση αυτή είναι προσβάσιμη από το σύστημα άρα δε μπορούμε να την απορρίψουμε με κάποιο λήμμα. Από εκεί μπορούμε να βγάλουμε το συμπέρασμα ότι το πρόβλημα είναι στο σχεδιασμό του σύνθετου OTS και όχι του profile OTS. Συγκεκριμένα η περίπτωση αυτή είναι η εξής:

- Ο χρήστης A έχει το προφίλ του ανοιχτό μόνο στους φίλους του
- Ο χρήστης B είναι φίλος του A
- Ο χρήστης C δεν είναι φίλος του A

Αν ο B μαρκάρει τον A σε μία φωτογραφία P, τότε μετά την εφαρμογή της μετάβασης tagFB, η φωτογραφία P ανήκει στη συλλογή του χρήστη A και ο C μπορεί να τη δει. Από αυτά τα δεδομένα που μας δίνει η CafeOBJ μπορούμε να τροποποιήσουμε κατάλληλα την προδιαγραφή του συστήματός μας ώστε το πρόβλημα αυτό να σταματήσει να υπάρχει.

4.2.6 Συμπεράσματα

Στο κεφάλαιο αυτό παρουσιάσαμε την αλγεβρική προδιαγραφή ενός κοινωνικού δικτύου σαν ένα σύστημα παρατηρήσεων μεταβάσεων (OTS) που αποτελείται από μικρότερα OTS που αλληλεπιδρούν και αλλάζουν τις καταστάσεις των άλλων και κατ' επέκταση, την κατάσταση ολόκληρου του συστήματος. Δείξαμε πώς μπορούμε να χρησιμοποιήσουμε τη προδιαγραφή μας για να διερευνήσουμε την ισχύ ιδιοτήτων ασφάλειας στο σύστημα με χρήση της μεθοδολογίας των proof scores.

Είδαμε επίσης πως η προδιαγραφή δουλεύει ανεξάρτητα από τον τρόπο που υλοποιείται το σύστημα (για παράδειγμα, προσδιορίσαμε λίστες με αιτήματα φιλίας που βρίσκονται σε αναμονή και περιγράψαμε τις συνθήκες που κάποιος χρήστης μπαίνει και βγαίνει από αυτή τη λίστα αλλά δε βάλαμε κάποιον περιορισμό στο πώς θα υλοποιηθεί αυτή η λίστα: λίστα, στοίβα, ουρά, κ.τ.λ.). Αυτό μας επιτρέπει να αποδεικνύουμε ιδιότητες που θα ισχύουν για κάθε τρόπο υλοποίησης του συστήματος, εφόσον βέβαια ικανοποιεί την προδιαγραφή.

Τέλος, είδαμε πώς μπορούμε να εντοπίσουμε λάθη στο σχεδιασμό του συστήματος με τη μέθοδο OTS/Proof Scores και πώς με το ημιαυτόματο σύστημα αποδείξεων της CafeOBJ μπορούμε να εξάγουμε συμπεράσματα για το λόγο που μια ιδιότητα δεν ισχύει. Η χρήση Τυπικών

Μεθόδων μπορεί να βρει τέτοια λάθη σχετικά νωρίς, πριν το σύστημα περάσει στη φάση της υλοποίησής του. Πράγματι, μιας και το πρωτόκολλο είναι σχεδιασμένο από το μηδέν από εμάς, είδαμε από πρώτο χέρι πως η προδιαγραφή του πρωτοκόλλου που γίνεται παράλληλα με το σχεδιασμό του βοηθάει στην σωστότερη (και γρηγορότερη) ανάπτυξή του.

4.3 Open Document Architecture (ODA)

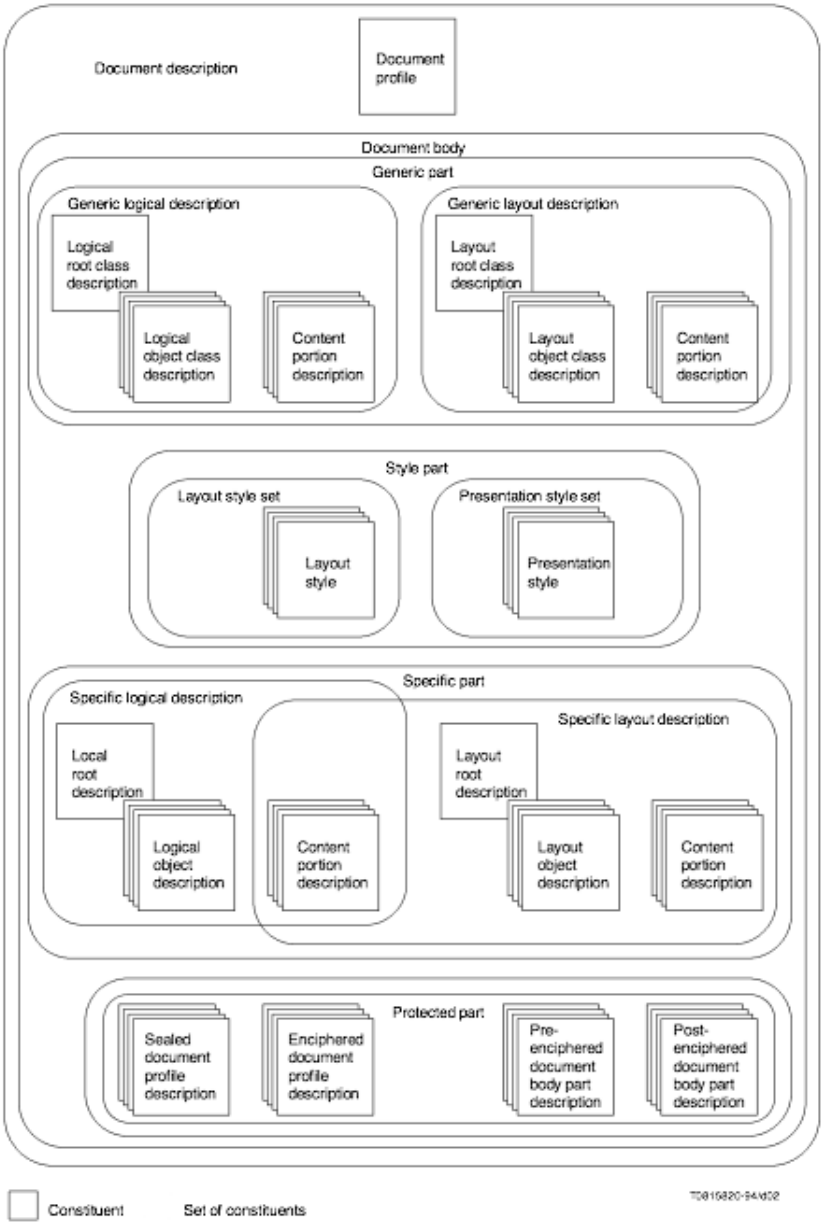
Το Open Document Architecture [67] είναι ένα ανοικτό και ελεύθερο διεθνές πρότυπο αρχείου εγγράφων που ξεκίνησε από την ITU-T και είχε σα στόχο να αντικαταστήσει τα ιδιόκτητα format αρχείων. Αν και η ανάπτυξη του ODA έχει πλέον παγώσει, λειτούργησε ως μεγάλη επιρροή για πολλά από τα πιο πρόσφατα και επιτυχημένα format εγγράφων, όπως το HTML, CSS, XML, XSL και τα OpenDocument και Office Open XML.

Η μορφή αρχείου ODA υποστήριζε τρεις διαφορετικές μορφές αποθήκευσης: τη μορφοποιημένη, τη μορφοποιημένη με δυνατότητα επεξεργασίας και την επεξεργάσιμη. Η πρώτη μορφή δίνει ένα αρχείο που δεν μπορεί να αλλαχθεί και είναι παρόμοια με αυτή της μορφής PDF της Adobe Systems. Το κύριο χαρακτηριστικό στη μορφή αρχείου ODA είναι η δομή. Η δομή ενός εγγράφου είναι η διαίρεση και η επαναλαμβανόμενη υποδιαίρεση του περιεχόμενου του εγγράφου σε όλο και μικρότερα τμήματα που ονομάζονται αντικείμενα [67]. Δύο δομές μπορούν να εφαρμοστούν στο έγγραφο: η λογική δομή και η δομή διάταξης. Στη λογική δομή, το έγγραφο διαιρείται και υποδιαιρείται με βάση το νόημά του (κεφάλαια, ενότητες, σχήματα, παράγραφοι) όπως κάνει δηλαδή και η HTML. Στη δομή διάταξης, το έγγραφο διαιρείται και υποδιαρείται με βάση τη διάταξη (σελίδες και μπλοκ), σε αντιστοιχία με το Cascading Style Sheets (CSS).

Η τυποποίηση ενός προτύπου αρχιτεκτονικής εγγράφων ενισχύει τις προσπάθειες ηλεκτρονικής διακυβέρνησης· πολλοί τύποι δημόσιων εγγράφων παράγονται καθημερινά όπως έγγραφα πιστοποίησης, εξουσιοδότησης, κ.ά. Η χρήση τυπικών μεθόδων σε τέτοια πρότυπα συνεισφέρει στη διαφάνεια και ακρίβεια τέτοιων διαδικασιών.

4.3.1 Περιγραφική Αναπαράσταση εγγράφου

Ένα έγγραφο αντιπροσωπεύεται από συστατικά που ομαδοποιούνται σε ομάδες συστατικών και τα οποία έχουν μεταξύ τους σχέση, όπως ορίζεται από το ITU-T Rec. T.410-Series | ISO/IEC 8613 [68]. Οι δυνατοί



Σχήμα 8: ODA: Περιγραφική αναπαράσταση ενός εγγράφου

A/A	Ιδιότητα
1	*Το έγγραφο αποτελείται από ένα προφίλ εγγράφου και (προαιρετικά) έναν αριθμό συστατικών που σχηματίζουν το σώμα του εγγράφου*
2	*Τα συστατικά που αντιπροσωπεύουν την γενική λογική δομή αποτελούνται από τις λογικές περιγραφές της τάξης των αντικειμένων και τις σχετικές περιγραφές κομματιών γενικού περιεχομένου*
3	*Αν ένα έγγραφο περιέχει και κάποια συγκεκριμένη λογική δομή αλλά και κάποια συγκεκριμένη δομή διάταξης τότε τα κομμάτια περιεχομένου που σχετίζονται με αυτές τις δομές είναι κοινά*

Πίνακας 8: Ιδιότητες του ODA

τύποι των συστατικών στην περιγραφική αναπαράσταση του εγγράφου φαίνονται στο Σχήμα 8. Κάποιες ιδιότητες που ισχύουν εδώ (από το [68]) φαίνονται στον Πίνακα 8.

4.3.2 Τυποποίηση του ODA

Για την πρώτη ιδιότητα του Πίνακα 8: Προδιαγράφουμε το module με όνομα *DOCUMENT-DESCRIPTION* στην CafeOBJ. Το αποτέλεσμα φαίνεται στον Κώδικα 29. Το έγγραφο (με όνομα συνόλου *Document-Description*) πρέπει να περιέχει ένα προφίλ εγγράφου (όνομα συνόλου *Document-Profile*) και, προαιρετικά, το σώμα του εγγράφου (με όνομα συνόλου *Document-Body*). Και τα δύο αυτά σύνολα ορίζονται κατάλληλα στα δικά τους modules και εδώ εισάγουμε τους ορισμούς τους με τις δύο εντολές *pr()*.

Στη συνέχεια, ορίζουμε τον τελεστή *Document-Description* δύο φορές καθώς μπορεί να οριστεί επαρκώς είτε με το σώμα του εγγράφου (δεύτερη δήλωση) είτε χωρίς (πρώτη δήλωση). Η ιδιότητα ότι το σώμα του εγγράφου σχηματίζεται από ένα πλήθος συστατικών απεικονίζεται στον ορισμό του module *Document-Body*.

```
mod! DOCUMENT−DESCRIPTION {
 pr(DOCUMENT−PROFILE)
 pr(DOCUMENT−BODY)

 [ Document−Description ]

 op Document−Description : Document−Profile −> Document−Description
```

```
 op Document−Description : Document−Profile Document−Body
          −> Document−Description
}
```

Κώδικας 29: Το module της περιγραφής του εγγράφου

Για τη δεύτερη ιδιότητα του Πίνακα 8: Όπως και πριν, δηλώνουμε ότι το *Generic-Logical-Description* πρέπει να περιέχει ένα Logical Root Class Description, ένα Logical Object Class Description List και προαιρετικά, ένα Content Portion Description List (Κώδικας 30). Η λέξη κλειδί *List* στο Content Portion Description List (ή στο Logical Object Class Description List) δηλώνει ότι το Content Portion Description List είναι μια "συλλογή" από Content Portion Descriptions. Αυτό το ορίζουμε στον Κώδικα 31, όπου *CONTENT-PORTION-DESCRIPTION* είναι το βασικό module και το *C-CONTENT-PORTION-DESCRIPTION* δημιουργεί μια λίστα από Content Portion Descriptions.

```
mod! GENERIC−LOGICAL−DESCRIPTION {
 pr(LOGICAL−ROOT−CLASS−DESCRIPTION)
 pr(C−LOGICAL−OBJECT−CLASS−DESCRIPTION)
 pr(C−CONTENT−PORTION−DESCRIPTION)
 [ Generic−Logical−Description ]
 op Generic−Logical−Description : Logical−Root−Class−Description
  Logical−Object−Class−DescriptionList Content−Portion−DescriptionList
  −> Generic−Logical−Description
 op Generic−Logical−Description : Logical−Root−Class−Description
  Logical−Object−Class−DescriptionList −> Generic−Logical−Description
}
```

Κώδικας 30: Το module της γενικής λογικής περιγραφής

Η λίστα δημιουργείται εισάγοντας το module με όνομα *List* (Κώδικας 32) μετονομάζοντας το σύνολο *Elt* σε *Content-Portion-Description*, το σύνολο *List* σε *Content-Portion-DescriptionList* και τέλος, το σύνολο *nil* σε *nilContent-Portion-Description*. Η μέθοδος αυτή εφαρμόζεται στα περισσότερα στοιχεία του Σχήματος 8.

```
mod CONTENT−PORTION−DESCRIPTION {
 [ Content−Portion−Description ]
}

mod* C−CONTENT−PORTION−DESCRIPTION {
 pr(LIST(CONTENT−PORTION−DESCRIPTION {sort Elt −>
        Content−Portion−Description})ιατ
```

```
*{sort List -> Content-Portion-DescriptionList,
      op nil -> nilContent-Portion-Description}) }
```

Κώδικας 31: Το module για τη λίστα περιγραφής κομματιού περιεχομένου

```
mod! LIST (X :: TRIV) {
 pr(NAT)
 [Elt < List]
 op nil : -> List .
 op null : -> Elt .
 op _|_ : Elt List -> List .
 op _//in_ : Elt List -> Bool
 op _\_ : List Elt -> List
 op # : List -> Nat
 op element : List Nat -> Elt .
 vars L L1 L2 : List .
 vars E1 E2 : Elt .
 op _=_ : List List -> Bool {comm} .
 eq (L = L) = true .
 eq (nil = (E2 | L2)) = false .
 eq ((E1 | L1) = (E2 | L2)) = (E1 = E2) and (L1 = L2) .
 ceq (E1 //in E2) = true if (E1 = E2) .
 ceq (E1 //in (E2 | L)) = true if (E1 = E2) or (E1 //in L) .
 eq (E1 //in nil) = false .
 eq (nil \ E1) = nil .
 ceq (E1 \ E2) = nil if (E1 = E2) .
 ceq ((E1 | L) \ E2) = L if (E1 = E2) .
 ceq ((E1 | L) \ E2) = E1 | (L \ E2) if not(E1 = E2) . }
```

Κώδικας 32: Το module για την περιγραφή του εγγράφου

Η τρίτη ιδιότητα του Πίνακα 8 λέει πως αν έχουμε κάποιο κομμάτι εγγράφου που περιέχει μια λογική περιγραφή και μια περιγραφή διάταξης και οι δύο αυτές δομές έχουν κάποιο περιεχόμενο (cont1 και cont2 αντίστοιχα) τότε το περιεχόμενο αυτό είναι κοινό. Ο τελεστής *Specific-Logical-Description-Exists?* ελέγχει αν το δοσμένο Specific Part περιέχει μια λογική περιγραφή. Ο τελεστής *Specific-Layout-Description-Exists?* κάνει το ίδιο για τη περιγραφή διάταξης. Ο τελεστής *Content-from-Specific-Logical-Description-Exists?* ελέγχει αν μια λογική περιγραφή έχει περιεχόμενο και ο τελεστής *Content-from-Specific-Layout-Description-Exists?* ελέγχει για περιεχόμενο μια περιγραφή διάταξης. Αν και οι τέσσερις αυτές συνθήκες ισχύουν τότε τα δύο αυτά περιεχόμενα θα είναι τα ίδια.

Η ιδιότητα αυτή μπορεί να γραφεί με χρήση μαθηματικών ως:

$$\exists\ specific\_logical\_structure(cont1) \wedge \exists\ specific\_ \\ \_layout\_structure(cont2) \implies cont1 = cont2. \tag{1}$$

Η προδιαγραφή αυτή της ιδιότητας καθώς και ολόκληρο το module *Specific-Part* φαίνεται στον Κώδικα 33.

```
mod! SPECIFIC−PART {
 pr(SPECIFIC−LOGICAL−DESCRIPTION)
 pr(SPECIFIC−LAYOUT−DESCRIPTION)
 [ Specific−Part ]
 op Specific−Part : Specific−Logical−Description Specific−
        Layout−Description  −> Specific−Part
 op Specific−Part : Specific−Logical−Description −> Specific−Part
 op Specific−Part : Specific−Layout−Description −> Specific−Part
 op Specific−Logical−Description−Exists? :  Specific−Part −> Bool
 op Specific−Layout−Description−Exists? : Specific−Part −> Bool
 var SP : Specific−Part
 var SLOD : Specific−Logical−Description
 var SLAD : Specific−Layout−Description
 eq Specific−Logical−Description−Exists?(Specific−Part(SLOD, SLAD)) = true.
 eq Specific−Logical−Description−Exists?(Specific−Part(SLOD)) = true .
 eq Specific−Logical−Description−Exists?(Specific−Part(SLAD)) = false .
 eq Specific−Layout−Description−Exists? (Specific−Part(SLOD, SLAD))
        = true .
 eq Specific−Layout−Description−Exists?(Specific−Part(SLOD)) = false .
 eq Specific−Layout−Description−Exists?(Specific−Part(SLAD)) = true .
 vars CONT, CONT1, CONT2 : Content−Portion−DescriptionList
 var LARD : Layout−Root−Description
 var LAOD : Layout−Object−DescriptionList
 var LORD : Local−Root−Description
 var LOOD : Logical−Object−DescriptionList
 op Content−from−Specific−Logical−Description−Exists? : Specific−Logical
     −Description −> Bool
 eq Content−from−Specific−Logical−Description−Exists?(
        Specific−Logical−Description(LORD, LOOD, CONT)) = true .
 eq Content−from−Specific−Logical−Description−Exists?(
        Specific−Logical−Description(LORD, LOOD)) = false .
 op Content−from−Specific−Layout−Description−Exists? : Specific−
    Layout−Description −> Bool
 eq Content−from−Specific−Layout−Description−Exists?(
    Specific−Layout−Description(LARD, LAOD, CONT)) = true .
 eq Content−from−Specific−Layout−Description−Exists?(
```

```
    Specific−Layout−Description(LARD, LAOD)) = false .
ceq CONT1 = CONT2 if Specific−Logical−Description−Exists?(Specific−
    Part(SLOD, SLAD)) and Specific−Layout−Description−Exists?(Specific
    −Part(SLOD, SLAD)) and Content−from−Specific−Logical−Description−
    Exists?(Specific−Logical−Description(LORD, LOOD, CONT1)) and
    Content−from−Specific−Layout−Description−Exists?(Specific−
    Layout−Description(LARD, LAOD, CONT2)) . }
```

Κώδικας 33: Το module για ένα συγκεκριμένο κομμάτι

4.3.3 Σχόλια

Στο κεφάλαιο αυτό είδαμε μια πρώτη προδιαγραφή του Open Document Architecture χρησιμοποιώντας τυπικές μεθόδους, πώς μπορούμε να μοντελοποιήσουμε ένα πρότυπο χρησιμοποιώντας την προδιαγραφή του που κυκλοφορεί γραμμένη σε φυσική γλώσσα και πώς μπορούμε να χρησιμοποιήσουμε την προδιαγραφή αυτή για να μοντελοποιήσουμε τις απαιτήσεις του συστήματος (σε μορφή ιδιοτήτων). Εφόσον δε δημιουργήσαμε εμείς το πρότυπο, το προδιαγράψαμε τυπικά σύμφωνα με αυτό που εμείς πιστεύουμε ότι αντικατοπτρίζει τις προθέσεις των σχεδιαστών. Είδαμε πως η τυπική προδιαγραφή του ODA δεν επιτρέπει ασάφειες στον καθορισμό απαιτήσεων, έχει μικρότερο μέγεθος από την προδιαγραφή σε φυσική γλώσσα λόγω του μαθηματικού υπόβαθρου και τέλος, πως η προδιαγραφή που γράφεται σε μορφή modules είναι επαναχρησιμοποιούμενη, ως κομμάτι μιας μεγαλύτερης προδιαγραφής. Παρόμοιες δουλειές στην ανάλυση εγγράφων είναι οι [69, 70, 71]. Όπως και στην προηγούμενη περίπτωση, έτσι και εδώ, η προδιαγραφή έγινε χρησιμοποιώντας τη γλώσσα CafeOBJ (Κεφάλαιο 2.3) αλλά οποιαδήποτε άλλη γλώσσα τυπικών προδιαγραφών μπορεί να χρησιμοποιηθεί.

4.4 Rich Site Summary (RSS)

Στο κεφάλαιο αυτό θα συνεχίσουμε να δείχνουμε πως η αλγεβρική προδιαγραφή ενός προτύπου μπορεί να αντιμετωπίσει τα προβλήματα που παρουσιάστηκαν στο Κεφάλαιο 2.1.4, προδιαγράφοντας το πρότυπο RSS v2.0 χρησιμοποιώντας την αλγεβρική γλώσσα προδιαγραφών CafeOBJ (Κεφάλαιο 2.3). Θα εξετάσουμε τα πλεονεκτήματα της τυπικής προδιαγραφής έναντι της προδιαγραφής που παρέχεται από τους κατασκευαστές του προτύπου. Όπως και στο Open Document Architecture (Κεφάλαιο 4.3), τυποποιούμε αυτό που εμείς καταλαβαίνουμε ότι εννοούν οι δημιουργοί του προτύπου [72]. Θα εξετάσουμε πώς η τυπική

προδιαγραφή του μας κάνει να εξετάζουμε σε βάθος έννοιες που πιθανόν να μην έχουν υλοποιηθεί ή περιγραφεί σωστά και στη συνέχεια να τις περιγράψουμε αμφιμονοσήμαντα και με σαφήνεια.

Το Rich Site Summary (RSS) είναι ένας τύπος αρχείου βασισμένος στο Extensible Markup Language (XML) που χρησιμοποιείται είτε για την αναμετάδοση περιεχομένου του Web ώστε να μπορεί να αναδημοσιευτεί σε άλλους δικτυακούς τόπους ή για να παρουσιάζει στους χρήστες του web το πιο πρόσφατο περιεχόμενο ενός ιστότοπου. Έτσι, οι χρήστες μπορούν να 'γράφονται' στα websites που επιθυμούν και να λαμβάνουν ενημερώσεις για διαθέσιμο νέο περιεχόμενο μέσω των browsers τους χωρίς να χρειάζεται να επισκέπτονται τον κάθε τέτοιο ιστότοπο. Οι μεταδόσεις μέσω RSS μπορούν να διαβαστούν από κατάλληλο λογισμικό που συνήθως ονομάζεται "RSS reader", "feed reader", ή "aggregator" και το οποίο μπορεί να είναι υπηρεσία web, desktop software ή εφαρμογή κινητού. Αν και το RSS δεν είναι τόσο δημοφιλές όσο ήταν πριν το 2013 (όσο δηλαδή πριν η Google κλείσει το Google Reader), παραμένει ένα ευρέως χρησιμοποιούμενο πρότυπο το οποίο μετατοπίζεται προς παρασκηνιακή υποστήριξη διαδικτυακών υπηρεσιών (π.χ. η εφαρμογή ειδήσεων του iOS 9 χρησιμοποιεί το RSS για να μεταδώσει τις ιστορίες που συλλέγει στο χρήστη [73]).

Η προδιαγραφή του RSS v2.0 [72] περιγράφει τη μορφή που πρέπει να ακολουθεί ένα αρχείο RSS. Το RSS είναι ένας τύπος αρχείου XML οπότε κάθε αρχείο RSS πρέπει να ακολουθεί την προδιαγραφή του XML (συγκεκριμένα την έκδοση 1), όπως αυτή έχει δημοσιευτεί στην ιστοσελίδα του World Wide Web Consortium (W3C). Ο κώδικας 34 δείχνει ένα απλό RSS-XML αρχείο [74].

Στο πιο πάνω επίπεδο, ένα έγγραφο RSS είναι ένα στοιχείο <rss> με μία απαραίτητη ιδιότητα, τη version, που προσδιορίζει την έκδοση του RSS στην οποία αντιστοιχεί το έγγραφο. Κάτω από το στοιχείο <rss> υπάρχει ένα μόνο στοιχείο (<channel>) που περιέχει πληροφορίες για το κανάλι (μετα-δεδομένα) και τα στοιχεία του [72]. Ένα <channel> μπορεί να περιέχει έναν οποιοδήποτε αριθμό δομών <item>, που αναπαριστούν μία "ιστορία" και περιέχουν στοιχεία όπως *title*, *description* και *link* [72]. Στο Κεφάλαιο 4.4.1 δίνουμε μια πιο εκτεταμένη εξήγηση του Κώδικα 34.

```
<rss version=''2.0''>
 <channel>
  <title>Liftoff News</title>
  <link>http://liftoff.msfc.nasa.gov/</link>
  <description>Liftoff to Space Exploration.</description>
  <language>en-us</language>
```

```
  <pubDate>Tue, 10 Jun 2003 04:00:00 GMT</pubDate>
  <docs>http://blogs.law.harvard.edu/tech/rss</docs>
  <generator>Weblog Editor 2.0</generator>
  <managingEditor>editor@example.com</managingEditor>
  <webMaster>webmaster@example.com</webMaster>
  <item>
   <title>Star City</title>
   <link>http://liftoff.msfc.nasa.gov/news/2003/news−starcity.asp</link>
    <description>How do Americans get ready to work with Russians
      aboard the International Space Station? They take a crash
      course in culture, language and protocol at Russia's
      <a href=''http://howe.iki.rssi.ru/GCTC/gctc_e.htm''>Star City</a>.
    </description>
    <pubDate>Tue, 03 Jun 2003 09:39:21 GMT</pubDate>
    <guid>http://liftoff.msfc.nasa.gov/2003/06/03.html#item573</guid>
    <enclosure url=''http://www.scripting.com/mp3s/weatherReportSuite.mp3''
      length=''12216320'' type=''audio/mpeg''>
    </enclosure>
  </item>
 </channel>
</rss>
```

Κώδικας 34: Υπόδειγμα αρχείου RSS

4.4.1 Δομές XML

Ένα αρχείο RSS ακολουθεί τη μορφή αρχείου XML, μία γλώσσα σήμανσης που ορίζει ένα σύνολο κανόνων για την κωδικοποίηση εγγράφων σε μια μορφή που είναι αναγνώσιμη τόσο από τον άνθρωπο όσο και από ένα μηχάνημα. Είναι ένα ανοιχτό πρότυπο, που ορίζεται στην προδιαγραφή του W3C, XML 1.0. Αν και η XML είναι σχεδιασμένη να εστιάζει σε έγγραφα, χρησιμοποιείται ευρέως για την αναπαράσταση αυθαίρετων δομών δεδομένων για παράδειγμα στον τομέα των υπηρεσιών Ιστού. Στην πραγματικότητα, η XML είναι το πιο χρησιμοποιούμενο εργαλείο για τη μετάδοση δεδομένων μεταξύ κάθε είδους εφαρμογής και είναι πλέον τόσο σημαντική για το Web, όσο ήταν η HTML στη δημιουργία του Web [75].

Τα έγγραφα XML σχηματίζουν μια δενδροειδή δομή που ξεκινάει από τη "ρίζα" και εξαπλώνεται στα "φύλλα". Ο κώδικας 34 είναι σχεδόν ένα σωστό αρχείο XML. Θα ήταν σωστό αν αντικαθιστούσαμε στη πρώτη γραμμή, το στοιχείο "rss" με το $<?xmlversion = ''1.0''encoding = ''UTF-8''?>$. Η πρώτη αυτή γραμμή είναι τώρα η δήλωση του XML

αρχείου, ορίζοντας την έκδοση της XL (1.0). Τα έγγραφα XML πρέπει να περιέχουν ένα στοιχείο ρίζας έτσι ώστε η επόμενη γραμμή να περιγράφει το γονέα του εγγράφου. Στην περίπτωση του RSS, αυτός είναι ο *channel*. Το *channel* περιέχει 10 στοιχεία-παιδιά (*Title*, *Link*, *Description*, *Language*, *pubDate*, *κ.λπ.*). Τα στοιχεία σε ένα έγγραφο XML σχηματίζουν ένα δέντρο που ξεκινάει από τη ρίζα και επεκτείνεται μέχρι τα χαμηλότερα επίπεδα του δέντρου. Οι όροι γονέας, παιδί και αδερφός χρησιμοποιούνται για να περιγράψουν τις σχέσεις μεταξύ των στοιχείων. Τα γονεϊκά στοιχεία έχουν παιδιά. Τα παιδιά που βρίσκονται στο ίδιο επίπεδο λέγονται αδέρφια [75]. Για παράδειγμα τα στοιχεία *Title*, *link*, *description*, *language*, *κ.λπ.* είναι αδέρφια. Το στοιχείο *Item* είναι γονέας 5 παιδιών (*Title*, *Link*, *Description*, *PubDate* και *guid*).

Μία ετικέτα XML είναι ένα κατασκεύασμα με ειδική σήμανση που ξεκινάει με το "<" και τελειώνει με το ">". Υπάρχουν τρία είδη ετικετών XML. Οι ετικέτες εκκίνησης (π.χ. <channel>), οι ετικέτες τερματισμού (π.χ. </channel>) και οι ετικέτες χωρίς στοιχείο (π.χ. < />).

Οι ετικέτες κάνουν διάκριση πεζών-κεφαλαίων. Ένα στοιχείο XML απαρτίζεται από την ετικέτα εκκίνησής του μέχρι και την ετικέτα τερματισμού του. Όλα τα στοιχεία XML πρέπει να έχουν ετικέτα εκκίνησης και ετικέτα τερματισμού. Ένα στοιχείο μπορεί να περιέχει κείμενο, ιδιότητες, άλλο στοιχείο ή στοιχεία ή κάποιο συνδυασμό όλων. Οι τιμές των ιδιοτήτων μπαίνουν πάντα μέσα σε εισαγωγικά, είτε μονά είτε διπλά. Μία ιδιότητα μπορεί να έχει μόνο μία τιμή και κάθε ιδιότητα μπορεί να εμφανιστεί μόνο μία φορά μέσα σε κάθε στοιχείο.

```
<?xml version=''1.0'' encoding=''UTF−8''?>
```

Κώδικας 35: Δήλωση αρχείου XML

Ένα έγγραφο XML που ακολουθεί τους παραπάνω συντακτικούς κανόνες ονομάζεται "καλά ορισμένο". Για να βεβαιωθούμε ότι ένα έγγραφο XML είναι καλά ορισμένο συνήθως χρησιμοποιούμε κάποιον XML validator. Αυτό βέβαια είναι διαφορετικό από αυτό που αποκαλούμε "έγκυρο" αρχείο XML. Ένα έγκυρο XML αρχείο πρέπει να είναι καλά ορισμένο αλλά και να συμμορφώνεται σε κάποιο συγκεκριμένο τύπο εγγράφου. Ένας τύπος εγγράφου είναι ένα σύνολο κανόνων που ορίζουν τα επιτρεπτά στοιχεία και τις πιθανές ιδιότητές τους· δηλαδή τι μπορεί να εμφανιστεί μέσα σε ένα έγγραφο XML. Υπάρχουν διαφορετικοί τύποι ορισμών εγγράφων που μπορούν να χρησιμοποιηθούν στην XML: Το "Document Type Definition" (DTD) και το βασισμένο στην XML "XML Schema". Ένα έγγραφο XML που συμμορφώνεται είτε με ένα DTD είτε με ένα XML Schema είναι και καλά ορισμένο και έγκυρο.

4.4.2 XML2OBJ

Το XML2OBJ είναι ένα εργαλείο με στόχο τη δημιουργία ενός πλαισίου στην CafeOBJ για τη περιγραφή δομών XML που παρέχει μεθόδους για την ανάλυση ολόκληρης της δενδροειδής δομής ενός αρχείο XML και που μπορεί να:

- βρίσκει αν υπάρχει ένα συγκεκριμένο στοιχείο (όπως αυτό ορίζεται από την ετικέτα του) και, αν χρειαστεί, το γονέα του
- να επιστρέφει αυτό το στοιχείο ή το περιεχόμενό του
- να ελέγχει αν ένα στοιχείο έχει ιδιότητες και να επιστρέφει τόσο τα ονόματα των ιδιοτήτων του αλλά και τις τιμές τους.

Το XML2OBJ είναι ένα module της CafeOBJ που περιέχει όλους τους τύπους και τελεστές που χρειάζονται για να επιτύχουμε αυτό το είδος λειτουργικότητας. Το module αυτό μπορεί να εισαχθεί σε οποιοδήποτε προδιαγραφή της CafeOBJ χρειάζεται λειτουργικότητα XML. Το module αυτό δεν είναι Document Type Definition (DTD) [76] αφού παρέχει μόνο τις μεθόδους που χρειάζεται η CafeOBJ για να συλλέξει πληροφορίες από ένα αρχείο XML. Ο ορισμός των επιτρεπόμενων δομικών στοιχείων ενός αρχείου XML γίνεται αργότερα (βλ. Κεφάλαιο 4.4.3). Για να χρησιμοποιήσουμε ένα αρχείο XML σαν είσοδο αυτού του module χρειαζόμαστε μερικές συντακτικές αλλαγές, για λόγους αναγνωσιμότητας αλλά και συμβατότητας με την CafeOBJ.

4.4.2.1 Περιγραφή και συντακτικές διαφορές

$$
\begin{array}{lll}
<\text{“}TagName\text{”} & Attributes & [ElementValue] > \\
<ElemName & AttribList & [ContentList] > \quad : Element
\end{array}
$$

Σχήμα 9: Σύνταξη ενός στοιχείου στο XML2OBJ και οι αντίστοιχοι τύποι

Στο XML2OBJ, ένα στοιχείο ακολουθεί τη σύνταξη που φαίνεται στο Σχήμα 9. Το κομμάτι που ξεκινάει με το “<” και τελειώνει με το “>” είναι τύπου *Element*. Το “Tag Name” είναι το όνομα του XML στοιχείου και είναι τύπου *ElemName*. Τα “Attributes” είναι τύπου *AttribList* και μπορεί να είναι είτε $noAttrib$ αν το στοιχείο δεν έχει ιδιότητες ή

$$
\begin{array}{l}
((\text{“}Attribute_1Name\text{”} \ @ = \text{“}Attribute_1Value\text{”}) \\
@(\text{“}Attribute_2Name\text{”} \ @ = \text{“}Attribute_2Value\text{”}) \ @ \ ...)
\end{array}
$$

```
< "Parent" Attributes [
    (< "Nested Tag_1 Name" Attributes
        [ Element Value ]>) @
    (< "Nested Tag_2 Name"  Attributes
        [ Element Value ]>)
]>
```

Σχήμα 10: XML2OBJ: Σύνταξη εμφωλευμένων στοιχείων

στην περίπτωση που υπάρχει μία ή περισσότερες ιδιότητες. Ο τελεστής $noAttrib$ ορίζει μία άδεια λίστα από ιδιότητες. Ο τελεστής @ = δίνει μία τιμή σε μία ιδιότητε ενώ ο τελεστής @ ενώνει ιδιότητες σε παρένθεση. Το "Element value" είναι τύπου *ContentList* και στην περίπτωση του RSS μπορεί να είναι κείμενο, αριθμός, ημερομηνία ή κάποιο άλλο στοιχείο. Έτσι, το ElementValue μπορεί να είναι κάτι από τα ακόλουθα:

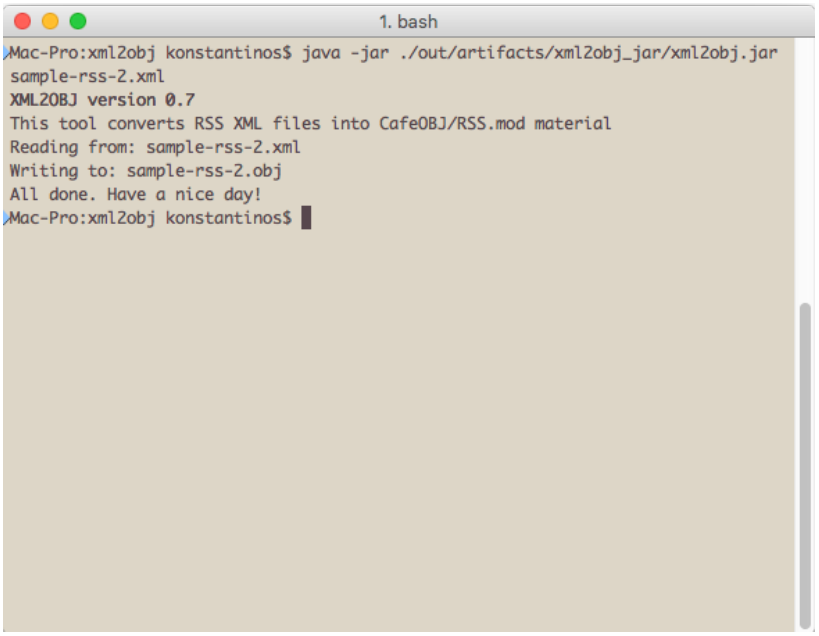
$$txt(\text{“}TextValue\text{”})|nat(IntegerValue)|dat(DateValue)$$

ή κάποιο άλλο στοιχείο, στην περίπτωση που έχουμε εμφωλευμένα στοιχεία XML. Αυτό μας λέει ότι ο τύπος *Element* είναι υποσύνολο του *Content*, και έτσι μπορούμε να δηλώσουμε την ακόλουθη σχέση τύπων: $[Element < Content]$, κάτι που φυσικά μπορεί να εφαρμοσθεί και στις εκδόσεις των τύπων σε λίστες ($[ElementList < ContentList]$).

Δεν υπάρχουν ετικέτες τερματισμού αφού η σύνταξη "$< ... ... [...] >$" τις κάνει περιττές. Η εμφώλευση στοιχείων XML ακολουθεί τη σύνταξη του Σχήματος 10. Τα στοιχεία Nested Tag_1 και Nested Tag_2 είναι παιδιά του στοιχείου "Parent". Ο τελεστής @ συνδέει στοιχεία.

Παρατηρούμε ότι το αρχείο RSS/XML με το αντίστοιχο OBJ αρχείο έχουν διαφορές στη σύνταξη και στη δομή, όμως η μετατροπή είναι απαραίτητη για να μπορέσει η CafeOBJ να διαβάσει ένα τέτοιο αρχείο πριν το επεξεργαστεί κατάλληλα. Η μετατροπή από RSS/XML στο αντίστοιχο OBJ αρχείο γίνεται αυτόματα με μία απλή εφαρμογή που αναπτύξαμε, με όνομα XML2OBJ. Η εφαρμογή, γραμμένη σε JAVA, τρέχει στη γραμμή εντολών και δέχεται (Σχήμα 11) ένα υποχρεωτικό όρισμα, το όνομα του αρχείου που περιέχει το RSS/XML αρχείο και (προαιρετικά) το αρχείο εξόδου. Αν δε δοθεί αρχείο εξόδου, το πρόγραμμα κρατάει το όνομα αρχείου εισόδου και βάζει την επέκταση .obj.

Σχήμα 11: Εκτέλεση του προγράμματος XML2OBJ χωρίς arguments



Σχήμα 12: Επιτυχής εκτέλεση του προγράμματος XML2OBJ

Αν θέλαμε να μετατρέψουμε το αρχείο RSS/XML του Κώδικα 34 (με την αλλαγή της πρώτης σειράς, όπως φαίνεται στον Κώδικα 35) σε αρχείο OBJ, τότε η επιτυχής μετατροπή του (Σχήμα 12) θα παίρναμε αυτό που φαίνεται μερικώς στον Κώδικα 36.

```
< ''Channel'' noAttrib [
 (< ''Title'' noAttrib [ txt(''Liftoff News'') ]>)@
 (< ''Link'' noAttrib [ txt(''http://liftoff.msfc.nasa.gov/'') ]>)@
 (< ''Description'' noAttrib [ txt(''Liftoff to Space Exploration.'') ]>)@
 (< ''language'' noAttrib [ txt(''en−us'') ]>)@

          ...

 (< ''Item'' noAttrib [
    ( < ''Title'' noAttrib [ txt(''Star City<'') ]>) @
    ( < ''Link'' noAttrib [ txt(''>http://liftoff.msfc.nasa.gov/news/2003/
      news−starcity.asp<'') ]>) @

                  ...

    ( < ''PubDate'' noAttrib [ dat(date(''Thu'', 2003, 6, 3,
        9, 39, 21,''GMT''')) ]>) @
    ( < ''Enclosure'' ((''url'' @= ''http://www. scripting.com/mp3s/
        weatherReportSuite.mp3'')
        @ (''length'' @= ''12216320'')
        @ (''type'' @= ''audio/mpeg'')) [ txt('''') ]>)
 ]>)
]>
```

Κώδικας 36: Το XML2OBJ αρχείο που προκύπτει από την αυτόματη μετατροπή

4.4.3 Αλγεβρική Προδιαγραφή του Πρωτοκόλλου RSS

Μπορούμε να εισάγουμε το XML2OBJ module ώστε να δημιουργήσουμε ένα DTD ισοδύναμο του RSS στην CafeOBJ. Αυτό μας παρέχει πλεονεκτήματα που τα απλά DTD δεν υποστηρίζουν όπως επικύρωση των δεδομένων του αρχείου RSS (κάτι που μπορεί να κάνει και το XML Schema). Για παράδειγμα, ένα DTD δεν ορίζει περιορισμούς στα δεδομένα τύπου χαρακτήρα. Αν επιτρέπονται δεδομένα τύπου χαρακτήρα τότε οποιοιδήποτε χαρακτήρες είναι επιτρεπτοί. Αυτό που δε μπορεί να κάνει ένα XML Schema ή ένα DTD είναι πιο σύνθετες διεργασίες, όπως για παράδειγμα το να συγκρίνει τα δεδομένα ενός XML στοιχείου

με τα δεδομένα κάποιου άλλου, ή το να οριστεί ένας περιορισμός που θα έχει ισχύ σε στοιχεία του ίδιου τύπου χωρίς να χρειάζεται να δηλωθεί ξανά. Η προσέγγιση που ακολουθούμε μπορεί να χειριστεί τέτοια θέματα εύκολα.

4.4.3.1 Προσέγγιση Προδιαγραφής

Η προδιαγραφή του RSS για τα αρχεία αυτού του τύπου είναι αρκετά διαφορετική από το είδος των εγγράφων που συνηθίζεται να συνοδεύουν μία περιγραφή, όπως π.χ. ενός προτύπου της ISO. Αυτό συμβαίνει επειδή παρά το ότι κάθε παράμετρος του RSS εξηγείται, αρκετές φορές η περιγραφή είναι μπερδεμένη και ασαφής. Η προδιαγραφή του RSS, που βρίσκεται στο [72], δείχνει αρκετά βερμπαλιστική, όπως όλες οι προδιαγραφές γραμμένες σε φυσική γλώσσα, αλλά συγκρινόμενη με το μέγεθος μίας συνηθισμένης προδιαγραφής, θεωρείται μικρού μεγέθους.

Κάθε στοιχείο του RSS v2.0 θα περιγραφεί σε ξεχωριστό module. Ο στόχος είναι να κρατάμε την περιγραφή σε ανεξάρτητα και επαναχρησιμοποιούμενα modules ώστε να μπορούν να χρησιμοποιηθούν και σε άλλα μέρη της προδιαγραφής. Κάθε τέτοιο module ονομάζεται όπως το στοιχείο του RSS που περιγράφει. Περιέχει επίσης και μέρος της πρωτότυπης προδιαγραφής (σε φυσική γλώσσα), όπως αυτή υπάρχει στο [72], σε μορφή σχολίου, που έχει σα στόχο να συμπληρώνει την τυπική προδιαγραφή. Τέλος, κάθε τέτοιο module περιέχει τις τυχόν απαιτήσεις του.

Η προδιαγραφή αρχίζει με τη δήλωση όλων των βοηθητικών modules. Τα πιο σημαντικά από αυτά είναι τα:

- Το module που προδιαγράφει δομές λίστας (LIST) και τις διαδικασίες που μπορούν να συμβούν σε αυτές. Μια λίστα εδώ μπορεί να είναι μια συλλογή αντικειμένων που ενώνονται με τον τελεστή @. Τα αντικείμενα εδώ είναι τα στοιχεία XML.

- Το module που προδιαγράφει δομές ημερομηνίας και τις διαδικασίες που μπορούν να συμβούν σε αυτές, όπως η εξαγωγή του έτους, μήνα, ημέρας κ.λπ. από μία δομή ημερομηνίας, ή ο έλεγχος για το αν μια ημερομηνία είναι σωστά ορισμένη (ο μήνας να είναι ανάμεσα στο 1 και στο 12 κ.τ.λ.).

- Το module που εισάγει τις δομές της XML, όπως αυτό αναφέρθηκε στο Κεφάλαιο 4.4.2.

Αυτά τα modules δηλώνονται πριν την περιγραφή του RSS μέσα από το "channel", καθώς η CafeOBJ ακολουθεί προσέγγιση bottom-up· συστή-

ματα που ενώνονται μαζί για να δημιουργήσουν πιο σύνθετα συστήματα κάνοντας έτσι τα αρχικά συστήματα υπο-συστήματα των συστημάτων που προκύπτουν.

Τα περισσότερα από τα στοιχεία του RSS μπορούν να χωριστούν σε τρεις κατηγορίες, ανάλογα με τον τύπο των δεδομένων που περιγράφουν· κείμενο, αριθμοί και ημερομηνίες. Εφόσον τα περισσότερα στοιχεία της ίδιας κατηγορίας είναι παρόμοια, αντί να γράφουμε ένα module για το καθένα, δημιουργούμε ένα πρότυπο module για κάθε κατηγορία και, χρησιμοποιώντας τις μεθόδους της CafeOBJ για εισαγωγή και μετανομασία, το επεκτείνουμε για τη δημιουργία του module που χρειαζόμαστε, μειώνοντας έτσι σημαντικά το μέγεθος της προδιαγραφής [31]. Ένα τέτοιο module μπορεί να χρησιμοποιηθεί ως βάση και περιέχει έναν τελεστή που εξάγει το περιεχόμενο του στοιχείου, έναν τελεστή που εξάγει τη λίστα με τις ιδιότητες του στοιχείου (εφόσον αυτό έχει ιδιότητες) και μία σταθερά που δηλώνει το όνομα του στοιχείου.

Ένα τέτοιο γενικευμένο module-constructor (δηλώνεται με το σύμβολο $*$) για στοιχεία που περιγράφουν δεδομένα κειμένου φαίνεται στον Κώδικα 37. Ο τύπος *ElementList* περιγράφει λίστες στοιχείων, ο τύπος *ElemName* περιγράφει το αναγνωριστικό κάθε τέτοιου στοιχείου (το tag του στοιχείου στο αρχικό αρχείο XML) και ο τύπος *AttribList* περιγράφει λίστες ιδιοτήτων. Ο τελεστής *get-xmlcontent* επιστρέφει την τιμή ενός στοιχείου.

```
mod* BUILDINGBLOCK {
 pr(XML)

 op get-xmlcontent : ElementList -> String .
 op get-attributes : ElementList -> String .
 op XML-TitlePrefix : -> ElemName .
 eq XML-TitlePrefix = ''' .

 vars X X1 : ElemName .
 vars A A1 : AttribList .
 var S : String .
 vars EL EL1 : ElementList .

 ceq get-xmlcontent( < X A [ txt(S) ]>) = S
    if (X = XMLTitlePrefix) and (A = noAttrib ) .

 ceq get-xmlcontent( ( < X A [ txt(S) ]>) @ EL1) = S
    if (X = XMLTitlePrefix) and (A = noAttrib ) .
```

```
ceq get−xmlcontent( ( < X A [ txt(S) ]>) @ EL) =
    get−xmlcontent(EL) if not((X = XMLTitlePrefix)
    and (A = noAttrib)) .

ceq get−xmlcontent(< X A [EL]>) = get−xmlcontent(EL)
    if not((X == XMLTitlePrefix) and (A = noAttrib)) .

ceq get−xmlcontent( < X A [ ( < X1 A1 [ EL ]>) ]>)
    = get−xmlcontent( < X1 A1 [ EL ]> )
    if not((X = XMLTitlePrefix) and (A = noAttrib )) . }
```

Κώδικας 37: Κατασκευαστής για στοιχεία με κείμενο

Υπάρχουν δύο ακόμα παρόμοιοι constructors, για τα στοιχεία που περιέχουν αριθμούς και ημερομηνίες. Αυτά τα στοιχεία προστατεύουν τα αντίστοιχα constructor modules και μετονομάζουν τους τελεστές κατάλληλα.

Για να ορίσουμε ένα στοιχείο με δεδομένα κειμένου (όπως για παράδειγμα τα Title, Enclosure, Language, κ.τ.λ.), το μόνο που πρέπει να κάνουμε είναι να προστατεύσουμε το δομικό module και να αλλάξουμε:

- το όνομα του τελεστή *get-xmlcontent* σε *get-ElementName*
- το όνομα του τελεστή *XML-TitlePrefix* σε *ElementName-XMLPrefix* και να θέσουμε την τιμή του.

Αν το στοιχείο που θέλουμε να προδιαγράψουμε έχει ιδιότητες, προσθέτουμε ένα τελεστή για κάθε ιδιότητα και ορίζουμε την τιμή του. Στον Κώδικα 38 βλέπουμε πως το module για το στοιχείο *Enclosure* έχει τρεις ιδιότητες: URL, Length και Type.

```
mod! ENCLOSURE {
 −− Describes a media object that is attached to the item. It has three required
 −− attributes. url says where the enclosure is located, length says how big it
 −− is in bytes, and type says what its type is, a standard MIME type.
 −− The url must be an http url. example: <enclosure url=''
 −− http://www.scripting.com/mp3s/weatherReportSuite.mp3''
 −− length=''12216320'' type=''audio/mpeg'' />

 pr(BUILDINGBLOCK * {op get−xmlcontent −> get−enclosure,
        op XML−TitlePrefix −> Enclosure−XMLPrefix})
 eq Enclosure−XMLPrefix = ''Enclosure'' .
 op Enclosure−URLXMLPrefix : −> ElemName .
 eq Enclosure−URLXMLPrefix = ''url'' .
 op Enclosure−LengthXMLPrefix : −> ElemName .
```

```
 eq Enclosure−LengthXMLPrefix = ‘‘length’’ .
 op Enclosure−TypeXMLPrefix : −> ElemName .
 eq Enclosure−TypeXMLPrefix = ‘‘type’’ .
}
```

Κώδικας 38: Στοιχείο με ιδιότητες: Enclosure

Ορισμένα στοιχεία μπορούν να ωφεληθούν από τελεστές ελέγχου που μπορούν να μας βεβαιώσουν ότι το στοιχείο που προδιαγράψαμε δουλεύει όπως πρέπει. Για παράδειγμα, το στοιχείο για τη γλώσσα (Language) έχει μία λίστα από επιτρεπόμενες τιμές· ένα στοιχείο γλώσσας θεωρείται αποδεκτό μόνο αν η τιμή του στοιχείου ταιριάζει με μία από αυτές που έχουν δηλωθεί στη εξίσωση του *properlanguage?* (Κώδικας 39).

```
mod! LANGUAGE {
 −− The language the channel is written in. This allows aggregators to group
 −− all Italian language sites, for example, on a single page. A list of
 −− allowable values for this element, as provided by Netscape, is here. You
 −− may also use values defined by the W3C.
 −− example: en−us

 pr(BUILDINGBLOCK * {op get−xmlcontent −> get−language,
        op XML−TitlePrefix −> Language−XMLPrefix})
 eq Language−XMLPrefix = ‘‘Language’’ .
 op properlanguage? : String −> Bool
 var L : String .
 eq properlanguage?(L) = if
        L = ‘‘en−us’’
        or L = ‘‘el−gr’’
 −− or L = ‘‘...’’
        then true
        else false
 fi .
}
```

Κώδικας 39: Στοιχείο με επαλήθευση δεδομένων: Language

Κρατήσαμε μέρος της αρχικής προδιαγραφής του RSS σε φυσική γλώσσα, ως σχόλια στα αντίστοιχα modules, όπως φαίνεται στους Κώδικες 38 και 39. Τα σχόλια αυτά μπορούν να βοηθήσουν κάποιον που θέλει να μάθει τι κάνει το κάθε module χωρίς να χρειαστεί να ανατρέξει στο documentation.

```
mod CHANNEL {
 pr(TITLE + LINK + DESCRIPTION + CHANNEL−OPTIONAL−ELEMENTS)
 op Channel−XMLPrefix : −> ElemName .
 eq Channel−XMLPrefix = ''Channel'' .
 op properchannel? : ElementList −> Bool .

 vars X X1 : ElemName .
 vars A A1 : AttribList .
 var S : String .
 vars EL EL1 : ElementList .
 vars PUBDATE TODAY : Date .
 var TTL1 : Nat .

 eq properchannel?(< X A [ EL ]>) =
  if ( (X = ChannelXMLPrefix) and (A = noAttrib) )
  and ( ElemNameexists?(Title−XMLPrefix, EL) and (getxmlatt(Title−XMLPrefix, EL) = noAttrib) and
     (getparent(Title−XMLPrefix, < X A [ EL ]>) = Channel−XMLPrefix) )
  and ( ElemNameexists?(Link−XMLPrefix, EL) and (getparent(Link−XMLPrefix, < X A [ EL ]>)= Channel−XMLPrefix)
     and (getxmlatt(Link−XMLPrefix, EL) = noAttrib) )
  and ( ElemNameexists?(Description−XMLPrefix, EL)
     and (getparent(Description−XMLPrefix, < X A [ EL ]>) = Channel−XMLPrefix)
     and (getxmlatt(Description−XMLPrefix, EL) = noAttrib) )
  and (ElemNameexists?(Language−XMLPrefix, EL) implies ((getxmlatt(Language−XMLPrefix, EL) = noAttrib)
     and properlanguage?(get−language(EL))) )
  and (ElemNameexists?(WebMaster−XMLPrefix, EL) implies getxmlatt(WebMaster−XMLPrefix, EL) = noAttrib)
  and (ElemNameexists?(Managing−EditorXMLPrefix, EL) implies getxmlatt(ManagingEditor−XMLPrefix, EL) = noAttrib )
  and (ElemNameexists?(Copyright−XMLPrefix, EL) implies getxmlatt(Copyright−XMLPrefix, EL) = noAttrib)
  and (ElemNameexists?(Docs−XMLPrefix, EL) implies getxmlatt(Docs−XMLPrefix, EL) = noAttrib )
  and (ElemNameexists?(Rating−XMLPrefix, EL)implies getxmlatt(Rating−XMLPrefix, EL) = noAttrib )
  and (ElemNameexists?(Generator−XMLPrefix, EL) implies getxmlatt(Generator−XMLPrefix, EL) = noAttrib)
  and ((ElemNameexists?(TTL−XMLPrefix, EL)) implies (getxmlatt(TTL−XMLPrefix, EL) = noAttrib))
  and ((ElemNameexists?(PubDate−XMLPrefix, EL) and (getparent(PubDate−XMLPrefix, < X A [ EL ]>) =
     Channel−XMLPrefix)) implies (getxmlatt(PubDate−XMLPrefix, EL) = noAttrib and properdate?( get−pubdate(EL))) )
  and (ElemNameexists?(SkipHours−XMLPrefix, EL) implies ((getxmlatt(SkipHours−XMLPrefix, EL) == noAttrib)
     and validhour(returnxmlnode(SkipHours−XMLPrefix, EL), hour(today))))
  and (ElemNameexists?(SkipDays−XMLPrefix, EL) implies ( (getxmlatt(SkipDays−XMLPrefix, EL) == noAttrib)
     and validday(returnxmlnode(SkipDays−XMLPrefix, EL), dayT(today))))
  and (ElemNameexists?(Cloud−XMLPrefix, EL) implies propercloud?(returnxmlnode(Cloud−XMLPrefix, EL)) )
  and (ElemNameexists?(TextInput−XMLPrefix, EL) implies propertextinput?(returnxmlnode(TextInput−XMLPrefix, EL)))
  and (ElemNameexists?(Item−XMLPrefix, EL) implies properitem?(returnxmlnode(Item−XMLPrefix, EL)))
  and (ElemNameexists?(Image−XMLPrefix, EL) implies properimage?(returnxmlnode(Image−XMLPrefix, EL)))
  and (ElemNameexists?(LastBuildDate−XMLPrefix, EL) implies ( (getxmlatt(LastBuildDate−XMLPrefix,EL) == noAttrib)
     and properdate?(get−lastbuilddate(EL))) )

  then true
  else false
 fi .
}
```

Κώδικας 40: Το module του καναλιού (Channel)

4.4.3.2 Το κανάλι του RSS - *Channel*

Το κανάλι (*channel* - Κώδικας 40) είναι το πιο σημαντικό κομμάτι της προδιαγραφής του RSS. Είναι αυτό που εισάγει τα τρία απαραίτητα στοιχεία (modules) (*Title*, *Link* και *Description*) και τα υπόλοιπα, προαιρετικά στοιχεία που εισάγονται από το meta-module *CHANNEL-OPTIONAL-ELEMENTS* που δουλειά του είναι να ομαδοποιεί και να εισάγει τα προαιρετικά στοιχεία, βελτιώνοντας έτσι το παρουσιαστικό της προδιαγραφής και την αναγνωσιμότητά της. Αφού κάθε εισαγόμενο module εισάγει με τη σειρά του άλλα, το module *channel* πρακτικά εισάγει όλα τα υπόλοιπα modules της προδιαγραφής. Στη συνέχεια ορίζουμε το XML prefix του καναλιού και τέλος, δημιουργούμε έναν τελεστή με όνομα *properchannel?* που εξετάζει ένα στοιχείο channel και το ελέγχει για εγκυρότητα.

Ο τελεστής *Properchannel?* ορίζει τις απαιτήσεις του καναλιού, όπως αυτές δηλώνονται στο [72]:

1. Το στοιχείο ρίζας του αρχείου είναι το "Channel" και αυτό δηλώνεται χωρίς κάποια ιδιότητα.

2. Υπάρχει ένα στοιχείο με όνομα *Title* που δηλώνεται χωρίς ιδιότητες και είναι παιδί του *Channel*. Ένα *Title* μπορεί να είναι και παιδί του *Item* οπότε είναι σημαντικό να δηλώσουμε και αυτή την απαίτηση. Τα ίδια ισχύουν και για τα στοιχεία *Link* και *Description*.

3. Τα προαιρετικά στοιχεία: Ο τελεστής*Properchannel?* ελέγχει πως αν υπάρχει στοιχείο *Language*, θα πρέπει να έχει δηλωθεί χωρίς ιδιότητες και ότι η τιμή του στοιχείου ανήκει στη λίστα με τις έγκυρες τιμές (κωδικοί γλωσσών, π.χ. "en-us"). Το στοιχείο αυτό δηλώνει τη γλώσσα στην οποία το κανάλι είναι γραμμένο, επιτρέποντας στους RSS aggregators να μπορούν να ομαδοποιούν site με κοινή γλώσσα, π.χ. όλα τα site γραμμένα στα ιταλικά, σε μία σελίδα [72].

4. Αν υπάρχει στοιχείο *WebMaster* (το email του υπεύθυνου για τεχνικά θέματα του καναλιού), θα πρέπει να έχει δηλωθεί χωρίς ιδιότητες. Η ίδια απαίτηση ισχύει και για τα στοιχεία:

   (α΄) *ManagingEditor* - η διεύθυνση email του συντάκτη του καναλιού.

   (β΄) *Copyright* - οι πληροφορίες για την πνευματική ιδιοκτησία του περιεχομένου του καναλιού.

(γ΄) *Docs* - ένα URL που κατευθύνει το χρήστη στο έγγραφο της μορφοποίησης που χρησιμοποιήθηκε στο αρχείο RSS, κάτι που μάλλον θα δείχνει στο [72].

(δ΄) *Rating* - η αξιολόγηση κατά PICS ([77]) για το κανάλι.

(ε΄) *Generator* - ένα κείμενο που αναφέρει το πρόγραμμα που χρησιμοποιήθηκε για την παραγωγή του καναλιού.

(στ΄) *TTL* - "time to live"; ένας αριθμός που δείχνει για πόσα λεπτά μπορεί το κανάλι να μείνει στην cache πριν χρειαστεί ανανέωση.

5. Το στοιχείο TTL δεν εξηγείται καθαρά στην προδιαγραφή του RSS αφού κάποιες πηγές ([78]) αναφέρουν ότι το TTL δείχνει πόση ώρα μπορεί ένα feed να μείνει "ζωντανό" μετά τη δημοσίευσή του, κάτι που είναι διαφορετικό από αυτό που αναφέρει η προδιαγραφή του RSS ([72]). Αν ισχύει αυτό, τότε θα μπορούσαμε να προσθέσουμε έναν τελεστή με όνομα *validfeed?* που δέχεται την ημερομηνία δημοσίευσης (*PUBDATE*), ένα TTL (*TTL1*) και τη σημερινή ημερομηνία (*TODAY*) και ελέγχει αν ο χρόνος στο μέλλον που θα πάρουμε αν προσθέσουμε το TTL1 στην ημερομηνία δημοσίευσης είναι πριν ή μετά την τρέχουσα ημερομηνία (Κώδικας 41). Αν αυτό είναι απαίτηση του πρωτοκόλλου RSS τότε μπορούμε να προσθέσουμε μία απαίτηση αυτή στον Κώδικα 40 που να ελέγχει την απαίτηση: Αν υπάρχει στοιχείο TTL τότε το περιεχόμενό του πρέπει να επαληθεύει τον τελεστή *validfeed?* (Κώδικας 41).

```
op validfeed? : Date Nat Date -> Bool .
eq validfeed?(PUBDATE, TTL1, TODAY) = if
 (timestamp(year(PUBDATE), month(PUBDATE), day(PUBDATE),
    hour(PUBDATE), minutes(PUBDATE), seconds(PUBDATE))
    + (TTL1 * 60) >=
 timestamp(year(TODAY), month(TODAY), day(TODAY), hour(TODAY),
    minutes(TODAY), seconds(TODAY)))
then true
else false
fi .
```

Κώδικας 41: Ο τελεστής validfeed?

6. Αν υπάρχει το στοιχείο *PubDate* (η ημερομηνία δημοσίευσης περιεχομένου στο κανάλι) και είναι παιδί του *Channel* (αφού μπορεί να είναι και παιδί του *item*) πρέπει να είναι δηλωμένο χωρίς ιδιότητες και πρέπει το περιεχόμενο του στοιχείου (η ημερομηνία) να

είναι ορθή· δηλαδή ο μήνας να είναι αριθμός από το 1 έως το 12, η ημέρα από 1-31, η ώρα από 00-23 κ.τ.λ. Ο τελεστής *properdate?* δηλώνεται στο βοηθητικό module *DATE* και επιστρέφει true αν η ημερομηνία ικανοποιεί αυτές τις απαιτήσεις και false σε κάθε άλλη περίπτωση. Το στοιχείο *LastBuildDate* είναι παρόμοιο (αντιπροσωπεύει την τελευταία φορά που άλλαξε το περιεχόμενο του καναλιού), αλλά δεν απαιτείται να είναι παιδί του *Channel*.

7. Το στοιχείο *Cloud* επιτρέπει σε διαδικασίες να εγγραφούν σε ένα "σύννεφο" για να μαθαίνουν για τυχόν ενημερώσεις του καναλιού, υλοποιώντας έτσι ένα ελαφρύ πρωτόκολλο δημοσίευσης-εγγραφής για RSS feeds. Ένα τέτοιο σύννεφο έχει πέντε απαιτούμενα χαρακτηριστικά. Το *domain* που είναι το όνομα του τομέα ή η διεύθυνση IP του σύννεφου, το *port* που είναι η θύρα TCP στην οποία το σύννεφο "ακούει", το *path* που είναι η τοποθεσία της υπηρεσίας που απαντάει, το *registerProcedure* που είναι το όνομα της διαδικασίας που καλεί για να ζητήσει ειδοποίηση, και τέλος, το *protocol* (που είναι είτε xml-rpc είτε soap ή http-post με διάκριση πεζών-κεφαλαίων) που δείχνει ποιο πρωτόκολλο θα χρησιμοποιηθεί [72]. Ένα παράδειγμα στοιχείου cloud είναι το εξής:

$$<clouddomain = ``rpc.sys.com"port = ``80"$$
$$path = ``/RPC2"registerProcedure = ``pingMe"$$
$$protocol = ``soap"/>$$

Ο τελεστής *properchannel?*, που φαίνεται στον Κώδικα 40, ελέγχει και για το αν κάποιο δοσμένο στοιχείο element είναι ορθά ορισμένο κάνοντας χρήση του τελεστή *propercloud?* (Κώδικας 42) που ελέγχει αν κάθε μία από τις απαιτούμενες ιδιότητες έχει οριστεί.

8. Το στοιχείο *SkipHours* περιέχει έως και 24 υπο-στοιχεία με νούμερα από το 0 έως και το 23 που αντιπροσωπεύουν τις ώρες (στο σύστημα GMT) που οι RSS aggregators δεν επιτρέπεται να διαβάσουν το κανάλι ([72]), εφόσον υποστηρίζουν αυτή τη δυνατότητα. Ο τελεστής *properchannel?* εξασφαλίζει ότι κάθε παιδί του στοιχείου SkipHours περιέχει μια έγκυρη ώρα (μέσω του τελεστή *validhour?* - Κώδικας 43): ένα νούμερο που αντιπροσωπεύει ώρα, από 0 έως και 23, που δεν μπορεί να είναι ίσο με τη τρέχουσα ώρα (ελέγχεται ενάντια στο *hour(today)*). Ο τελεστής *Hour* επιστρέφει την ώρα της σταθεράς "today" που δηλώνεται στην αρχή της προδιαγραφής. Το στοιχείο *SkipDays* είναι παρόμοιο· είναι ένα XML στοιχείο

```
eq propercloud?(< X:ElemName A:AttribList [ txt(S:String) ]>) =
 if (X == Cloud−XMLPrefix)
    and (xmlattexists?(CloudDomainXMLPrefix,
           getxmlatt(Cloud−XMLPrefix, < X A [ txt(S)]>)))
    and (xmlattexists?(CloudPortXMLPrefix,
           getxmlatt(Cloud−XMLPrefix, < X A [ txt(S)]>)))
    and (xmlattexists?(CloudPathXMLPrefix,
           getxmlatt(Cloud−XMLPrefix, < X A [ txt(S)]>)))
    and (xmlattexists?(CloudRegisterProcedureXMLPrefix,
          getxmlatt(Cloud−MLPrefix, < X A [ txt(S)]>)))
    and (xmlattexists?(CloudDomainXMLPrefix,
          getxmlatt(Cloud−XMLPrefix, < X A [ txt(S)]>)))

 then true
 else false
 fi .
```

Κώδικας 42: Ο τελεστής propercloud?

που μπορεί να περιέχει έως και επτά υπο-στοιχεία <day> με επιτρεπτές τιμές τις Monday, Tuesday, κ.τ.λ. Οι RSS aggregators που υποστηρίζουν αυτή τη λειτουργία δεν επιτρέπεται να διαβάσουν το κανάλι αν η τρέχουσα μέρα περιλαμβάνεται στα υπο-στοιχεία του *SkipDays* ([72]). Ο τελεστής *properchannel?* εξασφαλίζει ότι το στοιχείο αυτό δεν έχει ιδιότητες και ότι κάθε ένα από τα παιδιά του είναι διαφορετικό από τη τρέχουσα μέρα.

```
eq validhour((< X A [ ( < X1 A1 [ nat(N) ]>) ]>),
 CURRENTHOUR) = if
  ( (X == SkipHours−XMLPrefix)
  and (A == noAttrib)
  and (X1 == Hour−XMLPrefix)
  and (A1 == noAttrib)
  and ( (N >= 0) and (N <= 23) )
  and not(N == CURRENTHOUR) )

 then true
 else false
fi .
```

Κώδικας 43: Ο τελεστής validhour

9. Το στοιχείο *TextInput* είναι ένα στοιχείο που σύμφωνα με την ίδια

την προδιαγραφή του RSS ([72]) είναι "something of a mystery" και είναι ένα στοιχείο που οι περισσότεροι RSS aggregators αγνοούν. Το στοιχείο αυτό υποτίθεται ότι ορίζει μία φόρμα για την υποβολή ερωτημάτων στον εκδότη του feet μέσω του Common Gateway Interface (CGI). Αν ένα κανάλι περιέχει το στοιχείο αυτό, τότε αυτό θα πρέπει να έχει δηλωθεί με τέσσερα παιδιά: το *title* που είναι το κείμενο που φέρει το κουμπί της υποβολής που βρίσκεται κάτω από το χώρο που γράφονται τα σχόλια, το *description* που εξηγεί το σκοπό της φόρμας, το *name* που είναι το όνομα του στοιχείου της φόρμας που περιέχει τα σχόλια και τέλος, το *link* που είναι το URL του CGI script που χειρίζεται τα δεδομένα που υποβάλλονται στη φόρμα. Ο τελεστής *Properchannel?* ελέγχει ότι αν υπάρχει ένα TextInput τότε αυτό είναι ορθά ορισμένο με χρήση του τελεστή *propertextinput?* που ορίζεται στο module με όνομα TEXTINPUT. Ο τελεστής αυτός ελέγχει ότι τόσο το στοιχείο αυτό, όσο και τα 4 παιδιά του ορίζονται χωρίς ιδιότητες.

10. Το στοιχείο *Image* ορίζει μία εικόνα μορφής GIF, JPEG ή PNG που συνοδεύει το κανάλι. Περιέχει τρία υποχρεωτικά και τρία προαιρετικά υπο-στοιχεία. Τα υποχρεωτικά στοιχεία είναι τα:

    (α′) Το *url* που είναι το URL της εικόνας

    (β′) Το *title* που είναι η περιγραφή της εικόνας που χρησιμοποιείται στην ιδιότητα ALT του HTML στοιχείου <img>

    (γ′) Το *link* που είναι το URL του site· όταν το channel σχηματίζεται, η εικόνα είναι ένας δεσμός που δείχνει αυτό το site.

    Το [72] αναφέρει ότι πρακτικά, τα στοιχεία *title* και *link* του στοιχείου image θα πρέπει να έχουν τις ίδιες τιμές με τα στοιχεία *title* και *link* του channel. Αν και δεν είναι σαφές αν αυτό είναι υποχρεωτικό και άρα αν πρέπει να το χρησιμοποιήσουμε ως απαίτηση, αν πρέπει να το κάνουμε απαιτούμενο τότε απλά βάζουμε τον τελεστή *properchannel?* να διαβάσει τις τιμές αυτών των τριών στοιχείων και να τις συγκρίνει με τις αντίστοιχες του καναλιού. Ο τελεστής *properchannel?* μας εξασφαλίζει πως αν υπάρχει στοιχείο image τότε αυτό είναι καλά ορισμένο μέσω του τελεστή *properimage?* (Κώδικας 44). Ο τελεστής αυτός ελέγχει κατά πόσο τα τρία αυτά στοιχεία υπάρχουν και είναι δηλωμένα χωρίς κάποια ιδιότητα και αν υπάρχει κάποιο από τα τρία προαιρετικά στοιχεία τότε αυτό είναι ορθά δηλωμένο. Συγκεκριμένα, πέρα από το *description* που αποτελεί μια περιγραφή του site που οδηγεί το link

της εικόνας (και πρέπει να είναι δηλωμένο χωρίς ιδιότητες), τα υπο-στοιχεία *width* και *height* έχουν περιορισμούς: το πλάτος δεν μπορεί να είναι μεγαλύτερο των 144 pixels και το ύψος μεγαλύτερο των 400. Ο Κώδικας 45 δείχνει τον τελεστή *properwidth?* που ελέγχει την απαίτηση για το πλάτος. Ο τελεστής *Properheight?* ελέγχει την απαίτηση για το ύψος και ορίζεται με παρόμοιο τρόπο.

```
eq properimage?(< X A [ EL @ EL1 ]>) =
 if (X == Image−XMLPrefix)
  and (A = noAttrib)
  and (ElemNameexists?(Title−XMLPrefix, (EL @ EL1))
    and (getxmlatt(Title−XMLPrefix, (EL @ EL1)) == noAttrib))
  and (ElemNameexists?(Link−XMLPrefix, (EL @ EL1))
    and (getxmlatt(Link−XMLPrefix, (EL @ EL1)) == noAttrib))
  and (ElemNameexists?(URL−XMLPrefix, (EL @ EL1))
    and (getxmlatt(URL−XMLPrefix, (EL @ EL1)) == noAttrib))
  and (ElemNameexists?(Description−XMLPrefix, (EL @ EL1))
    implies (getxmlatt(Description−XMLPrefix,(EL @ EL1))==noAttrib))
  and (ElemNameexists?(Width−XMLPrefix, (EL @ EL1))
    implies properwidth?(getwidth((EL @ EL1))))
  and (ElemNameexists?(Height−XMLPrefix, (EL @ EL1))
    implies properheight?(getheight((EL @ EL1))))
 then true
 else false
fi .
```

Κώδικας 44: Ο τελεστής properimage?

```
eq properwidth?(W:Nat) = if ((W >= 0 and (W <= 144))
        then true
        else false
fi .
```

Κώδικας 45: Ο τελεστής properwidth?

11. Τέλος, έχουμε το στοιχείο *Item*. Ένα item αναπαριστά μία "ιστορία" όπως είναι μία ιστορία σε μία εφημερίδα ή περιοδικό· η περιγραφή της είναι μια σύνοψη της ιστορίας και ο δεσμός δείχνει το πλήρες άρθρο. Ένα item μπορεί να είναι πλήρες· σε αυτή την περίπτωση η περιγραφή περιέχει το κείμενο και ο δεσμός και ο τίτλος μπορούν να απουσιάζουν. Αυτό συμφωνεί με την προδιαγραφή του RSS που λέει πως ενώ όλα τα υπο-στοιχεία του item είναι προαιρετικά, ο τίτλος ή η περιγραφή πρέπει να υπάρχουν [72]. Ένα <channel> μπορεί να περιέχει οποιοδήποτε πλήθος από στοιχεία

<item>. Ο τελεστής *Properchannel?* μας εξασφαλίζει πως αν υπάρχει στοιχείο <item>, τότε αυτό θα πρέπει να είναι ορθά ορισμένο, δηλαδή να ικανοποιεί τις απαιτήσεις του τελεστή *properitem?* (Κώδικας 46). Ο τελεστής *properitem?* ελέγχει αν υπάρχει τίτλος ή περιγραφή και αν υπάρχουν άλλα προαιρετικά υπο-στοιχεία, ελέγχει αν αυτά είναι δηλωμένα σωστά:

(α΄) Το στοιχείο *enclosure* περιγράφει ένα αντικείμενο που επισυνάπτεται στο item. Έχει τρεις υποχρεωτικές ιδιότητες:

i. Το *URL* που είναι το URL με την τοποθεσία του αντικειμένου.

ii. Το *length* που δηλώνει το μέγεθός του σε bytes.

iii. Το *type* που δηλώνει τον MIME τύπο του αντικειμένου.

Ο τελεστής *Properitem?* ελέγχει για το αν υπάρχει enclosure και αν υπάρχει, ελέγχει ότι υπάρχουν και οι τρεις ιδιότητές του.

(β΄) Τα στοιχεία *link*, *author*, *category* και *comments* δηλώνονται χωρίς ιδιότητες. Το στοιχείο *PubDate* μοιάζει με το αντίστοιχο στοιχείο του καναλιού (Στοιχείο 6).

(γ΄) Το στοιχείο *Source* είναι το όνομα του καναλιού RSS από το οποίο το item ήρθε, όπως αυτό προκύπτει από το στοιχείο του <title>. Συντάσσεται με μία υποχρεωτική ιδιότητα, το *url* αποτελεί το σύνδεσμο για το XML αρχείο της πηγής, π.χ.: <source url="http://www.tom.org">Tom's site</source> ([72]). Το στοιχείο source έχει σα σκοπό να "δείχνει" τους δημιουργούς του άρθρου δημοσιεύοντας τις κατάλληλες πηγές. Μπορεί να χρησιμοποιηθεί στην εντολή `Post` του RSS aggregator. Συνήθως παράγεται αυτόματα κάθε φορά που ένα item προωθείται από έναν aggregator σε κάποιο εργαλείο σύνταξης online άρθρων [72]. Ο τελεστης *properitem?* ελέγχει την ύπαρξη της ιδιότητας URL.

Στον Κώδικα 40, ο τελεστής *getxmlatt* παίρνει ένα στοιχείο XML και επιστρέφει τις ιδιότητές του, ο τελεστής *getparent* επιστρέφει το γονέα ενός δοσμένου στοιχείου και τέλος, ο τελεστής *returnxmlnode* επιστρέφει ένα κόμβο, αφού απομακρύνει το γονέα του και τα αδέρφια του.

Τα υπόλοιπα στοιχεία (modules) του RSS έχουν παραληφθεί αφού δεν έχουν να δείξουν κάτι το ιδιαίτερο. Αυτή είναι και η Τυπική Προδιαγραφή του πρωτοκόλλου RSS v2.0. Το συνολικό μέγεθός της δε ξεπερνάει τις 820 σειρές, σε 44 συνολικά modules. Στις 820 αυτές σειρές

```
eq properitem?(< X A [ EL @ EL1 ]>) = if
  (X == Item−XMLPrefix)
  and (A == noAttrib)
  and ((ElemNameexists?(Title−XMLPrefix, (EL @ EL1))
    and getxmlatt(Title−XMLPrefix, (EL @ EL1)) == noAttrib)
    or (ElemNameexists?(Description−XMLPrefix, (EL @ EL1))
    and getxmlatt(Description−XMLPrefix, (EL @ EL1)) == noAttrib))
  and (ElemNameexists?(Enclosure−XMLPrefix, (EL @ EL1)) implies
    (xmlattexists?(EnclosureURLXMLPrefix, getxmlatt(Enclosure−XMLPrefix,
    returnxmlnode(Enclosure−XMLPrefix, (EL @ EL1) )) )
    and xmlattexists?(EnclosureLengthXMLPrefix,
    getxmlatt(Enclosure−XMLPrefix, returnxmlnode(
    Enclosure−XMLPrefix, (EL @ EL1) )) )
    and xmlattexists?(EnclosureTypeXMLPrefix,
    getxmlatt(Enclosure−XMLPrefix, returnxmlnode(
    Enclosure−XMLPrefix, (EL @ EL1) )) ) ) )
  and (ElemNameexists?(Source−XMLPrefix, (EL @ EL1)) implies
    xmlattexists?(SourceURLXMLPrefix, getxmlatt(Source−XMLPrefix,
    returnxmlnode(Source−XMLPrefix,(EL @ EL1)))))
  and (ElemNameexists?(Link−XMLPrefix, (EL @ EL1)) implies
   getxmlatt(Link−XMLPrefix, (EL @ EL1)) == noAttrib)
  and (ElemNameexists?(Author−XMLPrefix, (EL @ EL1)) implies
        getxmlatt(Author−XMLPrefix, (EL @ EL1)) == noAttrib)
  and (ElemNameexists?(Category−XMLPrefix, (EL @ EL1)) implies
        getxmlatt(Category−XMLPrefix, (EL @ EL1)) == noAttrib)
  and (ElemNameexists?(Comments−XMLPrefix, (EL @ EL1)) implies
   getxmlatt( Comments−XMLPrefix, (EL @ EL1)) == noAttrib)
  and (ElemNameexists?(PubDate−XMLPrefix, (EL @ EL1)) implies
    ((getxmlatt(PubDate−XMLPrefix, (EL @ EL1)) == noAttrib)
        and properdate?(getpubdate(EL @ EL1)) ))
 then true
 else false
fi .
```

Κώδικας 46: Ο τελεστής properitem?

συμπεριλαμβάνεται και το XML2OBJ (Κεφάλαιο 4.4.2) module και επίσης συμπεριλαμβάνονται κενές σειρές για να είναι πιο ευπαρουσίαστο αλλά και σχόλια για κάθε module, όπως αυτά εμφανίζονται στην προδιαγραφή του RSS ([72]). Το μέγεθος της προδιαγραφής χωρίς σχόλια είναι περίπου 650 σειρές.

4.4.3.3 Εκτέλεση προδιαγραφής

Μπορούμε τώρα να ελέγξουμε ένα δοκιμαστικό κανάλι RSS (με όνομα *samplechannel*) ενάντια στη προδιαγραφή που δημιουργήσαμε εφαρμόζοντας τον τελεστή *properchannel?* πάνω του. Το δοκιμαστικό κανάλι ακολουθεί τη σύνταξη της XML2OBJ. Αν η μηχανή αναγραφής της CafeOBJ απαντήσει με true στο ερώτημα μας, τότε μπορούμε να συμπεράνουμε ότι το κανάλι αυτό είναι ορθό. Η διαδικασία του ελέγχου του καναλιού *samplechannel* ενάντια στη προδιαγραφή μας φαίνεται στον Κώδικα 47. Ο Κώδικας 48 δείχνει τα αποτελέσματα. Μετά από 3909 αναγραφές όρων, η CafeOBJ απάντησε πως το *samplechannel* ικανοποιεί τον τελεστή *properchannel?* άρα και την προδιαγραφή.

Μπορούμε επίσης να δοκιμάσουμε να πειράξουμε το *samplechannel* με τέτοιο τρόπο ώστε να μην ανταποκρίνεται πια στη προδιαγραφή και να δούμε τι θα απαντήσει η CafeOBJ. Για παράδειγμα, αν αντικαταστήσουμε τις απαραίτητες ιδιότητες του *Cloud* με τον ειδικό τελεστή *noAttrib* (που σημαίνει ότι το *Cloud* δηλώνεται χωρίς ιδιότητες), τότε ο Κώδικας 49 δείχνει πως η CafeOBJ απαντάει πως το κανάλι αυτό δεν είναι ορθά ορισμένο. Μπορούμε να ενεργοποιήσουμε μία πιο λεπτομερή έξοδο αν θέλουμε να δούμε ποια απαίτηση του τελεστή απέτυχε να αποτιμηθεί σωστά. Οποιαδήποτε άλλη απόκλιση από τις απαιτήσεις της προδιαγραφής μας θα έχει ίδιο αποτέλεσμα.

```
-- opening module CHANNEL.. done.
%CHANNEL> %CHANNEL> %CHANNEL> _
%CHANNEL> *
-- reduce in %CHANNEL :
        (properchannel?(samplechannel)):Bool
(true):Bool
(0.000 sec for parse, 3909 rewrites(4.150 sec),
 30349 matches)
```

Κώδικας 48: Το αποτέλεσμα της αναγραφής του Κώδικα 47

```
open CHANNEL .
 op samplechannel : -> ElementList .
 eq samplechannel = < ''Channel'' noAttrib [
  ( < ''Title'' noAttrib [ txt(''Title goes here'') ]>) @
  ( < ''Link'' noAttrib [ txt(''URL'') ]>) @
  ( < ''Description'' noAttrib [ txt(''This is the description'') ]>) @
  ( < ''Category'' (''Domain'' @= ''Syndic8'') [ txt(''1765'') ]>) @
  ( < ''Generator'' noAttrib [ txt(''generator'') ]>) @
  ( < ''Language'' noAttrib [ txt(''el-gr'') ]>) @
  ( < ''Copyright'' noAttrib [ txt(''Copyright 2002, Spartanburg Herald-Journal'') ]>) @
  ( < ''ManagingEditor'' noAttrib [ txt(''Managing Editor'') ]>) @
  ( < ''WebMaster'' noAttrib [ txt(''email@webmaster.com'') ]>) @
  ( < ''PubDate'' noAttrib [ dat(date(''Thu'', 2013, 4, 24, 17, 22, 0,''GMT'')) ]>) @
  ( < ''Cloud'' ((''domain'' @= ''rpc.sys.com'') @ (''port'' @= ''80'') @ (''path'' @= ''/RPC2'')
     @ (''registerprocedure'' @= ''pingMe'')
     @ (''protocol'' @= ''soap'')) [ txt('''') ]> ) @
  ( < ''Image'' noAttrib [
   ( < ''Title'' noAttrib [ txt(''Title of the image here'') ]>) @
   ( < ''Link'' noAttrib [ txt(''Image redirects here'') ]>) @
   ( < ''URL'' noAttrib [ txt(''URL of the image'') ]>) @
   ( < ''Width'' noAttrib [ nat(55) ]>) @
   ( < ''Height'' noAttrib [ nat(400) ]>)
  ]> ) @
  ( < ''TTL'' noAttrib [nat(60) ]>) @
  ( < ''Item'' noAttrib [
   ( < ''Title'' noAttrib [ txt(''Title of the 1st item here'') ]>) @
   ( < ''Link'' noAttrib [ txt(''Link of the 1st item here'') ]>) @
   ( < ''PubDate'' noAttrib [ dat(date(''Thu'', 2013, 4, 24, 17, 22, 0,''GMT'')) ]>) @
   ( < ''Enclosure'' ((''url'' @= ''http://www.scripting.com/mp3s/weatherReportSuite.mp3'')
     @ (''length'' @= ''12216320'') @ (''type'' @= ''audio/mpeg'')) [ txt('''') ]> )
  ]> ) @
  ( < ''SkipHours'' noAttrib [
   ( < ''Hour'' noAttrib [ nat(11) ]>) @
   ( < ''Hour'' noAttrib [ nat(12) ]>)
  ]> ) @
  ( < ''SkipDays'' noAttrib [
   ( < ''Day'' noAttrib [ txt(''Sunday'') ]>) @
   ( < ''Day'' noAttrib [ txt(''Tuesday'') ]>)
   ]> )
 ]> .
close

red properchannel?(samplechannel) .
```

Κώδικας 47: Ελέγχοντας αν ένα κανάλι είναι ορθά ορισμένο

```
-- opening module CHANNEL.. done.
%CHANNEL> %CHANNEL> %CHANNEL> _
%CHANNEL> *
-- reduce in %CHANNEL :
        (properchannel?(samplechannel)):Bool
(false):Bool
(0.000 sec for parse, 3854 rewrites(4.140 sec),
 30074 matches)
```

Κώδικας 49: Τι συμβαίνει όταν το αρχείο RSS δεν είναι ορθό

4.4.4 Σχόλια

Στο κεφάλαιο αυτό δημιουργήσαμε μία τυπική, αλγεβρική, εκτελέσιμη προδιαγραφή για το ανοικτό πρότυπο RSS v2.0 στην CafeOBJ, που περιγράφει τυπικά τις απαιτήσεις του RSS, όπως αυτές εξηγούνται στο [72] και έγιναν κατανοητές από εμάς. Η προδιαγραφή αυτή λειτουργεί και σαν RSS/XML DTD αφού μπορούμε να ελέγξουμε κατά πόσο ένα οποιοδήποτε αρχείο RSS (αφού αυτό μετασχηματιστεί από το XML2BOJ) συμμορφώνεται στην προδιαγραφή. Τυποποιώντας το πρότυπο ανακαλύψαμε σημεία που δεν είχαν οριστεί σαφώς και δείξαμε πως θα μπορούσε να βελτιωθεί με την αλγεβρική προδιαγραφή. Μία αλγεβρική προδιαγραφή ενός προτύπου μπορεί i) να βελτιώσει την επικοινωνία και την κατανόηση μεταξύ των ενδιαφερόμενων, ii) να χρησιμοποιηθεί ως ένα εργαλείο διαχείρισης για διάφορες ομάδες διαχείρισης και iii) να τυποποιήσει γραμμές παραγωγής.

Όπως και στις προηγούμενες περιπτώσεις, έτσι και εδώ, η προδιαγραφή έγινε χρησιμοποιώντας τη γλώσσα CafeOBJ (Κεφάλαιο 2.3) αλλά οποιαδήποτε άλλη γλώσσα τυπικών προδιαγραφών μπορεί να χρησιμοποιηθεί. Η συγκεκριμένη γλώσσα βολεύει αρκετά στη συγκεκριμένη περίπτωση αφού το RSS εστιάζει στις δομές δεδομένων και η οικογένεια γλωσσών OBJ θεωρείται πολύ κατάλληλη για την τυποποίησή τους. Αλλιώς θα μπορούσαμε να χρησιμοποιήσουμε γλώσσες όπως η Z [10], η VDM (Vienna Development Method), η Estelle, η Lotos, τα Petri Nets, η SDL, η TLA, ή η Raise.

Έχει ενδιαφέρον να ελέγξουμε κατά πόσο η προδιαγραφή που δημιουργήσαμε ανταποκρίνεται στα πλεονεκτήματα της μεθόδου της αλγεβρικής προδιαγραφής προτύπων που εξηγούμε στο Κεφάλαιο 3.1:

- **Σαφήνεια προδιαγραφής**: Μπορούμε να υποθέσουμε με ασφάλεια πως ένα πρότυπο που κυκλοφορεί επίσημα είναι σχεδόν πά-

ντα χωρίς λάθη, παραλήψεις ή ασάφειες αφού θα έχει δοκιμαστεί και ελεγχθεί εξαντλητικά. Αυτό σημαίνει πως δεν περιμέναμε να βρούμε κρίσιμα θέματα ασάφειας στην περίπτωση αυτή, όμως μελετώντας το RSS v2.0 από το [72] βρήκαμε κάποιες ασάφειες σε μερικά στοιχεία, από τις 11 ομάδες απαιτήσεων, όπως αυτές εμφανίζονται στο Κεφάλαιο 4.4.3.2, όπως για παράδειγμα στο στοιχείο *TTL* ή στα υπο-στοιχεία *title* και *link* του στοιχείου *image*. Προδιαγράψαμε αυτό που κατά την κρίση μας ήταν εκείνο που είχαν στο μυαλό τους οι δημιουργοί του πρωτοκόλλου, χωρίς φυσικά να έχουμε τη δυνατότητα να χρησιμοποιήσουμε τη μέθοδο κατά τη δημιουργία του πρωτοκόλλου. Το αν η κατανόηση του πρωτοκόλλου είναι σωστή έχει λιγότερη σημασία μπροστά στο γεγονός ότι για να γράψουμε μια τυπική προδιαγραφή του αναγκαστήκαμε να διερευνήσουμε σε βάθος τις έννοιες του αφού η τυπική προδιαγραφή προϋποθέτει το να κάνει κάποιος τις "σωστές ερωτήσεις". Αυτή η μέθοδος μας δίνει μια ξεκάθαρη εικόνα του προτύπου και πώς αυτό πρέπει να δουλεύει.

Από την άλλη, αν ήμασταν στη θέση κάποιου που διαβάζει την τυπική αυτή προδιαγραφή του πρωτοκόλλου RSS v2.0 θα ήταν πιο εύκολο να συμπεράνουμε και να καταλάβουμε τις απαιτήσεις του σωστά και θα μπορούσαμε εύκολα να επαληθεύσουμε κατά πόσο μία υλοποίηση του πρωτοκόλλου δουλεύει όπως πρέπει.

- **Μέγεθος της προδιαγραφής**: Η προδιαγραφή του RSS v2.0, όπως αυτή εμφανίζεται στο [72] είναι πολύ μεγαλύτερη από τις 820 γραμμές κώδικα που πιάνει η αλγεβρική προδιαγραφή. Υπάρχουν στοιχεία στο [72] που παραπέμπουν σε σελίδες άλλων site που παρέχουν ολόκληρη την προδιαγραφή του στοιχείου. Μόνο οι προδιαγραφές της ημερομηνίας και ώρας, όπως αυτές εμφανίζονται στο RFC 822 είναι 40 σελίδες. Οπότε όχι μόνο έχουμε μια πολύ μικρότερη προδιαγραφή, έχουμε και έναν τύπο εγγράφου.

- **Συγχωνεύσεις απαιτήσεων**: Ο τελεστής *properchannel?* δείχνει κάθε ξεχωριστή απαίτηση του προτύπου. Ίσως είναι το μόνο κομμάτι της τυπικής προδιαγραφής που είναι πιο βερμπαλιστικό από την προδιαγραφή σε τυπική γλώσσα, αφού οι απαιτήσεις κάθε στοιχείου εμφανίζονται ξεχωριστά, σε αντίθεση με τη προδιαγραφή στο [72], όπου πολλές φορές πολλές απαιτήσεις διαφορετικών στοιχείων είναι συγχωνευμένες. Με αυτό τον τρόπο μπορούμε να απομονώσουμε τις απαιτήσεις που θέλουμε στον τελεστή και να ελέγξουμε πώς κάποια συγκεκριμένη αλλαγή σε αυτές κατά το

σχεδιασμό του RSS επηρεάζει όλο το πρωτόκολλο.

Η συγγραφή μιας τυπικής προδιαγραφής ενός προτύπου δεν είναι όσο απαιτητική είναι η γραφή μιας τυπικής προδιαγραφής ενός συγκεκριμένου συστήματος· δε χρειάζεται να γράψουμε ένα πλήρες σύνολο αξιωμάτων που θα μπορούσαν να χρησιμοποιηθούν σε μία αλγεβρική μηχανή αναγραφών και σε αυτή την περίπτωση επιτρέπεται κάποια "ατέλεια" της προδιαγραφής για να επιτρέψουμε μια πληθώρα διαφορετικών συστημάτων με διαφορετικές χρήσεις και λειτουργίες να συμμορφώνονται με το πρότυπο. Στην περίπτωση του RSS, η προδιαγραφή μας όχι μόνο παρέχει τις απαιτήσεις του προτύπου αλλά μπορεί και να ελέγξει την εγκυρότητα κάποιας υπηρεσίας που θέλει να επικοινωνήσει με το πρωτόκολλο RSS μέσω της λειτουργίας DTD.

4.5 Problem-focused education

Στο κεφάλαιο αυτό εξετάζουμε πόσο εύκολη είναι η κατανόηση και η σύνταξη τυπικών προδιαγραφών μέσα από την προσωπική εμπειρία διδασκαλίας τους σε μεταπτυχιακούς σπουδαστές. Τα αποτελέσματα ήταν ιδιαίτερα θετικά και έδειξαν ότι δεν είναι ιδιαίτερα δύσκολο τόσο το να διαβάσει κάποιος και να καταλάβει μία τυπική προδιαγραφή όσο και να γράψει μία.

4.5.1 Εισαγωγή και κίνητρα

Η εργασία αυτή εκπονήθηκε κατά τη διάρκεια της επίσκεψής μου στο Πανεπιστήμιο του Τάμπερε της Φινλανδίας, ως μέρος συμφωνίας ανταλλαγής φοιτητών μεταξύ του Εθνικού Μετσόβιου Πολυτεχνείου και του Tampereen Yliopisto, όπου και βοήθησα στη διδασκαλία του μεταπτυχιακού μαθήματος με τίτλο "Open Source and Software Quality" δίνοντας ομιλίες σχετικά με τις Τυπικές Μεθόδους και την Τυπική Προτυποποίηση.

Αυτό το παιδαγωγικό πείραμα [79] ξεκίνησε με την εξής ερώτηση: Πώς θα μπορούσαμε να κινητοποιήσουμε φοιτητές και προσωπικό ώστε να ενδιαφερθούν πραγματικά στο μάθημα "Open Source and Software Quality" [80, 81, 82, 83, 84] και στις διαδικασίες εκμάθησης ώστε να:

1. μάθουν σε βάθος τις έννοιες του λογισμικού Ανοιχτού Κώδικα [80, 36, 13]

2. αποκτήσουν ικανότητες εννοιολογικής μοντελοποίησης και να τις εφαρμόσουν

3. μάθουν για τις έννοιες της Τυπικής Προτυποποίησης

Επιπλέον (μερικές ερωτήσεις με μετα-γνωστική θεματολογία), πώς θα μπορούσαμε να:

1. αναπτύξουμε τις δεξιότητες και τις ικανότητες μάθησης;

2. σχεδιάσουμε ένα υποστηρικτικό περιβάλλον κατάλληλο για μάθηση;

3. ενισχύσουμε τη θέληση για μάθηση;

Αν θέλουμε να πετύχουμε ένα καλό (ή και βέλτιστο) αποτέλεσμα θα πρέπει να βελτιώσουμε όλα αυτά ταυτόχρονα. Αυτή είναι μία πρόκληση που αντιμετωπίζει (και μάλλον θα αντιμετωπίζει και στο μέλλον) η Επιστήμη Υπολογιστών και Μηχανικής Λογισμικού [36, 13]. Η πρόκληση αυτή δεν αφορά μόνο τα μέσα, δηλαδή τους τρόπους που προσφέρουμε κάποιο περιεχόμενο, γνώση ή πληροφορίες. Αν και αυτό είναι σημαντικό κομμάτι της πρόκλησης, δεν είναι μόνο αυτό. Η πιο σημαντική πτυχή είναι ο **σκοπός**, δηλαδή το να ρωτάμε γιατί κάνουμε κάτι με κάποιον τρόπο, για ποιο σκοπό προτιμούμε να σχεδιάσουμε ένα μάθημα με αυτό, και όχι με κάποιον άλλο, τρόπο.

Υπάρχουν πολλές παιδαγωγικές καινοτομίες [85], με κάθε μία από αυτές να προσπαθεί να αποδείξει γιατί είναι καλύτερη από τις υπόλοιπες. Οι συγγραφείς αυτής της εργασίας πιστεύουν ότι η Εκπαίδευση που εστιάζει σε επίλυση προβλήματος (Problem-Focused Education - **PFE**) είναι η πιο σημαντική παιδαγωγική καινοτομία αυτή τη στιγμή. Ο στόχος της PFE είναι να αμφισβητήσει και να αντικαταστήσει τις παραδοσιακές διδακτικές μεθοδολογίες που εστιάζουν σε κάποιο αντικείμενο και περιστρέφονται γύρω από τον καθηγητή.

Η Problem-Focused Education έχει τα εξής βήματα:

1. Ξεκινάει με ένα πρόβλημα

2. Το πρόβλημα παρουσιάζεται στους μαθητές ως κάτι που εμφανίζεται στην πραγματική ζωή

3. Στηρίζει τόσο τους διαφορετικούς τρόπους σκέψεις των μαθητών όσο και την εργασία σε ομάδες

4. Ενθαρρύνει τους μαθητές να αναγνωρίσουν τις δικές τους μαθησιακές ανάγκες και να αναλάβουν την ευθύνη της ανάπτυξης των δικών τους μαθησιακών διαδικασιών

5. Ενθαρρύνει την εποικοδομητική κριτική, την εκτίμηση και την αξιολόγηση της μαθησιακής διαδικασίας και των αποτελεσμάτων της

Στην PFE:

1. Οι ρόλοι των καθηγητών και των μαθητών επαναπροσδιορίζονται.

2. Η ύλη ενός μαθήματος ανασκευάζεται ώστε να περιστρέφεται γύρω από προσεκτικά διαλεγμένα, καλοσχεδιασμένα αλλά και μη επαρκώς ορισμένα προβλήματα.

3. Οι διαδικασίες επίλυσης προβλημάτων βασίζονται στην ιδέα της σταδιακής αποκάλυψης γνώσης, ενεργοποιώντας τη γνώση που οι μαθητές έχουν ήδη αποκομίσει [86]. Αυτό επιτυγχάνεται με το να:

   (α΄) Βοηθήσουμε τους μαθητές να αναγνωρίσουν ότι οι γνώσεις που ήδη κατέχουν δεν είναι αρκετές για τη λύση του προβλήματος.

   (β΄) Παρέχουμε έγκαιρη και κατάλληλη συμβουλευτική.

Με αυτή τη μέθοδο αποφεύγουμε το να έχουμε μαθητές παθητικούς, χωρίς κίνητρο που να μην είναι σε θέση να χρησιμοποιήσουν γνώσεις που έχουν ήδη αποκομίσει. Τα επόμενα κεφάλαια περιγράφουν την πειραματική εκπαιδευτική διαδικασία που χρησιμοποιήσαμε στο μάθημα "Open Source and Software Quality" αλλά και τις διαδικασίες εκτίμησης και ανάδρασης στις οποίες είναι βασισμένη η μέθοδος PFE [86].

4.5.2 Η ανάγκη για μια νέα πορεία και τα αποτελέσματά της

4.5.2.1 Σκεπτικό και καταλληλότητα του μαθήματος

Η διαδικασία ανάπτυξης λογισμικού που χρησιμοποιείται στη βιομηχανία έχει πρόσφατα αλλάξει κατεύθυνση ώστε να ευνοεί διαφορετικές μεθόδους και κύκλους ζωής σε σχέση με τα παραδοσιακά μεθοδολογικά πρότυπα και τους τρόπους εργασίας. Οπότε, ήταν προφανής η ανάγκη για μια νέα σειρά μαθημάτων που να εστιάζει σε θέματα Ελεύθερου Λογισμικού/Λογισμικού Ανοικτού Κώδικα - Free/Libre/Open Source Software (ΕΛ/ΛΑΚ ή FLOSS). Οι νέες τάσεις της ανάπτυξης ΕΛ/ΛΑΚ λογισμικού, ο αυξανόμενος αριθμός χρηστών που το χρησιμοποιούν αλλά και των ενδιαφερόμενων σχεδιαστών ανέδειξαν θέματα σχετικά με τεχνολογίες βασισμένες στο λογισμικό και σχετικά με την ποιότητα λογισμικού [36, 13, 41, 87, 88] που ζητάνε προσοχή. Η διαχείριση της ποιότητας λογισμικού απαιτεί όλο και περισσότερη εξέταση

και διασφάλιση της ποιότητας (ασφάλεια, αξιοπιστία, χρησιμότητα) του ΕΛ/ΛΑΚ, που, ως νέο κίνημα, συχνά παρερμηνεύεται και η ύπαρξή του συχνά συνοδεύεται από σκεπτικισμό.

4.5.2.2 Τα μαθησιακά αποτελέσματα

Μετά την ολοκλήρωση του μαθήματος "Open Source and Software Quality", περιμένουμε από τους μαθητές να είναι σε θέση να επιδείξουν ότι μπορούν να στηρίξουν τα Μαθησιακά Αποτελέσματα (στήλη ΜΟ) που συνοψίζονται στον Πίνακα 9.

ΜΟ-1	Να μπορούν να υποστηρίξουν και να συζητήσουν για διάφορες κοινότητες και πρακτικές των ΕΛ/ΛΑΚ
ΜΟ-2	Να γνωρίσουν διαφορετικές μεθόδους, τεχνικές και εργαλεία ανάπτυξης ΕΛ/ΛΑΚ (και) μέσω της μεθόδου PFE
ΜΟ-3	Να μπορούν να συγκρίνουν και να αντιπαραβάλλουν ιδιότητες ποιότητας λογισμικού του ΕΛ/ΛΑΚ αλλά και του "κλειστού" λογισμικού
ΜΟ-4	Να μπορούν να εξετάζουν και να αποφασίζουν για την καταλληλότητα του ανοικτού / κλειστού / ιδιόκτητου / μη-ιδιόκτητου λογισμικού με βάση τις ανάγκες
ΜΟ-5	Να σχεδιάζουν και να παρέχουν προγράμματα και τμήματα προγραμμάτων υψηλής ποιότητας που να ενσωματώνουν τις τελευταίες λέξεις της τεχνολογίας του ΕΛ/ΛΑΚ
ΜΟ-6	Να γίνουν μέλη και να υποστηρίζουν τις κοινότητες του ΕΛ/ΛΑΚ αναπτύσσοντας κατάλληλους πόρους
ΜΟ-7	Να μπορούν να προτείνουν (με κριτική επίγνωση) την προώθηση του ΕΛ/ΛΑΚ ως μία εφικτή εναλλακτική λύση στο ιδιόκτητο λογισμικό αλλά και ως στρατηγικό μέσο για την προώθηση της κοινωνίας στο σύνολό της
ΜΟ-8	Να χρησιμοποιηθούν ως σύμβουλοι / μηχανικοί ποιότητας λογισμικού σε ένα εύρος επιχειρήσεων και καινοτομιών
ΜΟ-9	Να κατανοούν και να αποδέχονται διαφορετικές κουλτούρες λογισμικού και στρατηγικών και να μπορούν να δίνουν ευκαιρίες για την μίξη διαφορετικών αλλά δημιουργικών ιδεών

Πίνακας 9: Αναμενόμενα Μαθησιακά Αποτελέσματα

Η κύρια σκέψη για την πρόταση του νέου μαθήματος είναι ότι μία επαρκής (τουλάχιστον στα όρια της διδασκαλίας ενός εξαμήνου) εξέταση σε θέματα ποιότητας λογισμικού θα φέρει χρήσιμες και πρακτικές

γνώσεις σε έναν σχεδιαστή λογισμικού, ο οποίος θα πρέπει να εστιάσει στην επαναχρησιμοποίηση κώδικα. Επίσης, μία πολύπλευρη προσέγγιση στην ανάλυση λογισμικού ΕΛ/ΛΑΚ θα μπορούσε:

- να δώσει στους σχεδιαστές λογισμικού νέες πληροφορίες και γνώσεις για τις δραστηριότητες που σχετίζονται με την ποιότητα λογισμικού στην ανάπτυξη ΕΛ/ΛΑΚ, καθώς και
- να αυξήσει την αξιοπιστία του ΕΛ/ΛΑΚ τόσο για τους σχεδιαστές όσο και για τους τελικούς χρήστες.

Με όλα αυτά υπόψη, η αιτιολόγηση του νέου σχεδιασμού του μαθήματος έγινε δεκτή από το Τμήμα Επιστήμης Υπολογιστών και το πλάνο μαθήματος οργανώθηκε.

4.5.3 Το ιστορικού του σχεδιασμού του μαθήματος

4.5.3.1 Στοιχεία Μαθήματος

Οι φοιτητές που θέλουν να παρακολουθήσουν το μάθημα χρειάζονται σχετικές σπουδές επιπέδου BSc. Θα πρέπει να έχουν γνώσεις προγραμματισμού αλλά και ανάπτυξης λογισμικού· κύκλους ζωής και μεθόδους. Άλλοι πιθανοί συμμετέχοντες που έχουν παρακολουθήσει αντίστοιχα μαθήματα σε σχετικές σχολές (όπως Επιστήμης Πληροφοριών) γίνονται δεκτοί (ως τώρα). Ωστόσο, γνώσεις σχετικές (ή συμπλήρωμα) με τις γνώσεις που αναφέρονται στον Πίνακα 9 είναι προαπαιτούμενες.

Μερικά αριθμητικά στοιχεία σχετικά με το μάθημα του "Open Source and Software Quality" φαίνονται στον Πίνακα 10.

Η διδασκαλία του μαθήματος αλλά και οι έξτρα συνεδρίες (φροντιστήρια, εξετάσεις, συναντήσεις με το εκπαιδευτικό προσωπικό κ.τ.λ.) γίνονται στα αγγλικά. Αυτό το μάθημα είναι επίσης διαθέσιμο για μεταδιδακτορικούς ερευνητές αλλά και φοιτητές που έρχονται με κάποιο πρόγραμμα ανταλλαγής. Συνήθως δεν υπάρχουν γραπτές εξετάσεις στο μάθημα, εκτός αν είναι απαραίτητο σε ειδικές περιπτώσεις (π.χ. μαθησιακών δυσκολιών). Η παρακολούθηση στο μάθημα είναι υποχρεωτική με εξαίρεση τις ομιλίες των επισκεπτών καθηγητών. Υποχρεωτική είναι επίσης η συμμετοχή στις εργασίες. Το μάθημα δίνει πέντε (5) μονάδες του European Credit Transfer Scheme (ECTS) και η τελική βαθμολογία δίνεται σε κλίμακα 1-5. Το υλικό μελέτης είναι επιλεγμένα σχετικά ερευνητικά άρθρα μαζί με σημειώσεις. Τέλος, η Σχολή Επιστήμης Πληροφοριών (School of Information Sciences - SIS), τμήμα της οποίας αποτελεί το Τμήμα Επιστήμης Υπολογιστών, πρόσφατα αποφάσισε ότι

Επίπεδο Μαθήματος	MSc και Ph.D.
Έτος φοιτητών	2ο έτος του Μεταπτυχιακού
Διάρκεια του μαθήματος	2+ μήνες (9 εβδομάδες)
Πλήθος προσωπικού που εμπλέκεται για τις διαλέξεις και τα σεμινάρια	2 μέλη προσωπικού και 2 επισκέπτες καθηγητές
Πλήθος φοιτητών	32 (42 είχαν γραφτεί μέσω email)
Πλήθος φοιτητών που διάλεξε να εργαστεί σε ομάδες	19
Πλήθος φοιτητών που διάλεξε να εργαστεί μόνο του	6
Πλήθος διαλέξεων	12 ώρες
Πλήθος ωρών κατ' ιδίαν συζήτησης με τους φοιτητές	55 ώρες και έξτρα βοηθητικές συνεδρίες
Πλήθος συμβουλευτικών ωρών μέσω email	45 ώρες και παραπάνω όταν ζητήθηκε από τους φοιτητές

Πίνακας 10: Πληροφορίες μαθήματος

το μάθημα θα ανήκει στις παρακάτω ομάδες μαθημάτων (που ανήκουν σε άλλα μεταπτυχιακά προγράμματα - αλλά ανήκουν στη SIS) με την ένδειξη "προαιρετικό" ή "συνιστώμενο":

1. Course Unit in Software Development (προαιρετικό)
2. Advanced Studies in Information Systems in Organizations (προαιρετικό)
3. Courses in Interaction Design and Research (συνιστώμενο)
4. Courses in Development of Interactive Software (συνιστώμενο)
5. Studies in the Specialization in Development of Interactive Software (2013-2015) (προαιρετικό).

Η σχεδίαση του μαθήματος ακολούθησε ορισμένες βασικές αρχές της PFE [13, 86]. Το μάθημα χρησιμοποίησε μια προσέγγιση βασισμένη σε ερωτήματα σε όλες τις παραδόσεις και πέρα από τις γενικές ερωτήσεις που ακολουθούν μία διάλεξη, οι περισσότερες ερωτήσεις και προβλήματα που αντιμετώπισαν οι φοιτητές παρουσιάστηκαν είτε κατ' ιδίαν με τους καθηγητές είτε στην υπόλοιπη τάξη. Πρόσθετα προβλήματα και ερωτήσεις που προέκυψαν κατά τη διάρκεια των διαλέξεων συζητήθηκαν από τις ομάδες εργασίας. Κάποιες από τις κύριες ερωτήσεις που αντιμετωπίσαμε κατά τη διάρκεια του εξαμήνου ήταν:

- Τι είναι το ΕΛ/ΛΑΚ και η ανάπτυξη ΕΛ/ΛΑΚ;
- Πώς διαφέρουν αυτά από παρόμοιες ή και διαφορετικές προσεγγίσεις;
- Τι μπορεί να πιστοποιηθεί και τι όχι στην ανάπτυξη ΕΛ/ΛΑΚ;
- Πώς θα μπορούσαν να χρησιμοποιηθούν τεχνικής διασφάλισης ποιότητας λογισμικού στο πλαίσιο του ΕΛ/ΛΑΚ;
- Πόσο αξιόπιστο, ασφαλές και "εγγυημένο" είναι το ΕΛ/ΛΑΚ και πώς αυτά μπορούν να εξασφαλιστούν και να πιστοποιηθούν;
- Είναι σημαντικό να χρησιμοποιούμε και να στηρίζουμε το ΕΛ/ΛΑΚ; Γιατί ναι / γιατί όχι;
- Πόσο ασφαλές και ηθικό είναι να χρησιμοποιούμε ΕΛ/ΛΑΚ;
- Ποιος χρειάζεται πραγματικά το ΕΛ/ΛΑΚ;
- Για ποιον είναι κατάλληλο το ΕΛ/ΛΑΚ;
- Πού απαιτείται / υποστηρίζεται το ΕΛ/ΛΑΚ; Πού όχι και γιατί;
- Υπάρχει μέλλον στο ΕΛ/ΛΑΚ;

Επίσης, ο μεγάλος στόχος για τους υπεύθυνους της επανασχεδίασης του μαθήματος αλλά και για τους εκπαιδευτές ήταν η κατασκευή ενός μαθήματος που υποστηρίζει μία επαγγελματική βάση γνώσεων, δεξιοτήτων και ικανοτήτων όπως απαιτούν τα σύγχρονα προγράμματα σπουδών [89], προσωπικές και επαγγελματικές αναπτυξιακές δράσεις και αναμενόμενα επαγγελματικά σενάρια στην ποιότητα του ανοικτού λογισμικού.

4.5.4 Το μάθημα, τα τελικά αποτελέσματα και τα σχόλια των φοιτητών

4.5.4.1 Το κύριο coursework

Το κείμενο που ακολουθεί είναι το email που στάλθηκε στους φοιτητές με το οποίο τους ανατέθηκε ως εργασία (ατομική ή σε ομάδες) η τυπική προδιαγραφή ενός τμήματος κάποιου προτύπου. Η εργασία συζητήθηκε σε δύο μαθήματα, ένα πριν το email και ένα μετά, κατά τη διάρκεια της τρίτης εβδομάδας του εξαμήνου. Το κείμενο που δίνει τις λεπτομέρειες του έργου, τη δομή που θα πρέπει να ακολουθηθεί και τις ημερομηνίες παράδοσης και εξέτασης παρατίθεται αυτούσιο:

> Hello all, here are some things you should know about the course's project.
>
> **Deadlines**:
>
> 1. Presentations: 14.5 (Tue) & 16.5 (Thu)
> 2. Final date for project submission: 21.5 (Tue)
>
> **Theme**: Specification of an open standard / creative commons.
>
> Pick an open standard, find a smaller subset that you are comfortable working with and provide a specification of that. You can find a how-to on the pdf i've sent you a month ago, ('formal specification') and feel free to take a look at the slides i've sent you about that. While dealing with creative commons is not necessary, feel free to give that a try too. In fact, if your specification of creative commons is better than your specification of open standards, then you'll get grade for the better work, so it might be useful to have an alternative. You can work alone or in groups, up to 3 people maximum. There is no 'line' minimum when it comes to the specification. Length is not that important. What matters more is the unambiguity of your specification and the correctness. However, a group of 3 people should obviously do more than someone working alone. So, if you want to increase that, just add more of the standard into the subset that you chose to specify.
>
> Please email me soon and let me know:
>
> 1. who are you working with?
> 2. what standard did you choose?

3. are you going to work with Creative Commons, too?

If you have questions, you can see me on Thursday 25.4 (10.00 - 12.00) at B3111. I'll let you know soon of other hours that you can find me as well. Reaching me via email is always an option too :)

You will have to give a small presentation of your work and a report.

Report structure:

1. Describe the standard you chose; especially the part that you chose to specify. Furthermore, if you had any good reasons for picking that. (max. 1 page).
2. Describe the specification language that you chose to use. (max. 1 page).
3. Provide the specification that you wrote and make sure you explain what you're doing so that it could be understood by someone not in your group.
4. Finally, provide some commentary about this procedure. Is that any different from what you've done so far? And if so, in what ways? What problems did you face? What do you think of the method? Try to compare your experiences with the points raised in the Formal Specification chapter I've sent you. Remember, that those are personal opinions so you do not get graded for them. Just as long as you provide arguments for any opinion you express.

Presentation Structure:

1. It should be around 20 minutes.
2. The structure is similar to the report, with less emphasis on the specification language you've used and more on the actual specification.

Best regards - the course instructors.

4.5.4.2 Αποτελέσματα - Σχόλια φοιτητών

Ο Πίνακας 11 περιέχει τα ανοιχτά πρότυπα που επιλέξανε οι φοιτητές για προτυποποίηση, τους βαθμούς τους (σε κλίμακα 1-5) και τον αριθμό των φοιτητών που είχε κάθε ομάδα.

Επιλογή προτύπου: Τυπική Προδιαγραφή Ανοικτών Προτύπων	**Τελικός Βαθμός (κλίμακα: 1-5)**	**Αριθμός φοιτητών**
RSS v2.0 specification modeling	5	3
Case-based Formal Specification of OAuth security	4	3
Formal Specification for OpenID standard	3	3
Modeling of a part of DICOM standard	3	2
Github - Formal Specification	3	2
OpenID User Authentication & DisCo Specification Language	2	1
TCP/IP Specification	2	1
Standard Entity-Relationship Modelling	1	1

Πίνακας 11: Αποτελέσματα φοιτητών

4.5.4.3 Σχόλια φοιτητών

Μαζί με τις εργασίες των φοιτητών πήραμε και τα σχόλιά τους για τις εργασίες αλλά και μια αυτοκριτική τους για το τι μάθανε. Πήραμε σχόλια και από τους δεκαέξι (16) φοιτητές (από τους 32 που ξεκίνησαν) που ολοκλήρωσαν την εργασία και πέρασαν το μάθημα, ένα ποσοστό 50% επιτυχίας. Παρακάτω φαίνονται οι γνώμες των φοιτητών αυτών:

1. στη δομή του μαθήματος, στο περιεχόμενο και στις δραστηριότητες

2. στην προσωπική τους διαδικασία εκμάθησης

3. στο αντικείμενο της εργασίας

Το υπογραμμισμένο κείμενο (μια πρωτοβουλία των συγγραφέων του άρθρου) δείχνει τα σχόλια των φοιτητών πάνω στα θέματα που θίγονται στο άρθρο.

Modelling of a part of DICOM standard

In this report, some SOPs are modelled in a very high abstraction level. FSP language are not very suitable for a detailed and object-oriented modelling. Unfortunately, because of time and resource limitation, FSP as the only formal language we know at this moment, it is more feasible for us to model the part of DICOM standard in this language. It could be better if we have more time to learn a more suitable language.

Formal Specification of OpenID

In the course, we use LTSA to model the openID and implement the simple OpenID procedure in java.

Some Good:

1. Get the chance to know some open standard and understand how open source development works.
2. Be given a lot of time for self-study and self-research, we improved ourselves in the data searching and data collection.
3. Professor and professor assistance are very helpful with offering related information.

Some problems:

1. Group members are not together and lack of discussion.
2. We start the course work very late and we didn't finish the task we want to do. We should do better.
3. At first, we would like to model in DisCo specification language, however, we didn't succeed installing the environment and then we change to LTSA in a hurry.

Case-based Formal Specification of OAuth Security

Experience:

We read documents about OAuth 2.0, including official description (RFC 6749), questions and answers on StackOverFlow, popular website API document(Facebook, Weibo) and many blogs on the Internet etc. During the research, we learned a lot about OAuth and some security issues while using OAuth. The most vital problem is CSRF with OAuth, if Client or Authorization Server are not well designed. In order to simulate

it, we did some experiment on some existing websites. But it is not enough to know OAuth deep, we need to program and realize the problem. Thus, we analyse the mechanism step by step, draw flow diagram and finally use specification language to try the problem. Though we cannot generate executable application in the end, but building the model of OAuth with specification language are very meaningful and efficient. It helps us to learn OAuth or other new knowledge in short time. And we don't need to program it from zero, just focusing on modeling. Problem: During the modeling, some problems were very tricky. First, we are not very familiar with DisCo specification languages. Second, DisCo seems not very perfect in some ways, such as multi references and deadlock. In some cases, we should have easier ways to realize, but only action and class are not enough to show all parts of OAuth transactions. As a famous saying goes: "It is not the destination so much as the journey". Therefore, we combine NLP and DisCo together to build the model. Most of the course work we use DisCo, then using NLP to fill some gaps. In conclusion, before we did the coursework, we thought the usage of Formal Specification were overemphasized. After experiencing the whole journey, Formal Specification helps us to test not just open standard but also ideas generated by our own. Most importantly, using it with protocols like OAuth can increase the study efficiency well.

RSS v2.0 Specification Modelling

Comments about modelling approach

Some of our positive and negative views about the modelling are listed below: Positive comments:

- It provides opportunity to have in depth knowledge about the technology.
- There is possibility to extend, change, customize, create something new in the RSS.
- we are much more specific as in why ? how? each specification is needed.

Negative comments:

- RSS is by no means a perfect format, although it is very popular and widely supported.

- There are not any standard specifications.
- There is no formal specification in order to validate our work.
- There is no formal specification for adding a new feature to the format.

Difficulties

When we started modelling, we came across the following difficulties:

- Version compatibility is an obstacle between different versions. For example, most users are around v.2.0.1 which can soon become incompatible with v.0.91.
- Too many unclear specifications exist which are vague because there is no official standard for RSS.
- The lack of information about current specifications and needs makes it hard to analyze the format.

Summary

In this coursework we modelled the some required channel's and item's elements and some optional elements as well. When we first started to model the elements of RSS, we started doing some research on the RSS specifications and realized there weren't any website which had formally specified the specifications this made it both hard and interesting. The reasons being described above in a positive and negative sense, therein we decided to model the specifications in an algebraic form, as described in Ian Sommerville's book "Software Engineering" (2009). This approach was chosen because RSS is a system that is described in terms of operations and their relationships. Some of the difficulties which were encountered by us upon the completion of the modelling were version compatibility, unclear specifications, and the lack of information about the current specifications.

4.5.5 Σχόλια

Τα παραπάνω σχόλια ήταν ιδιαίτερα αποκαλυπτικά, τόσο για τους εκπαιδευτικούς, όσο και για τους σχεδιαστές του μαθήματος [89, 90]:

- Στα σχόλια των φοιτητών υπάρχουν ικανοποιητικές (αλλά όχι αρκετές) λεπτομέρειες σχετικά με:

    1. το open source και τα ανοιχτά πρότυπα
    2. κατανόηση εννοιών και ιδιοτήτων της ποιότητας λογισμικού
    3. το κατά πόσο οι ίδιοι οι φοιτητές αντιλαμβάνονταν τις μαθησιακές τους ανάγκες αλλά και την πρόοδο που σημείωσαν κατά τη διάρκεια του εξαμήνου.

- Οι εκπαιδευτές έκαναν πολλές ενδιαφέρουσες παρατηρήσεις κατά τη διάρκεια του εξαμήνου:

    1. για την αλληλεπίδραση των φοιτητών
    2. για όλους τους δυνατούς τρόπους εκμάθησης
    3. για τους τρόπους εκμάθησης που προτίμησαν οι φοιτητές
    4. για τη γνώση και πείρα που αποκόμισαν οι φοιτητές
    5. για τους τρόπους σκέψης και επιχειρηματολογίας των φοιτητών
    6. για τους τρόπους μετάδοσης γνώσης και πληροφοριών σε τρίτους
    7. για την αυτο-αξιολόγηση των φοιτητών και των ικανοτήτων τους

Οι παρατηρήσεις πάνω στις προσωπικές διαδικασίες μάθησης ([89, 90]) είτε σε ομαδική εργασία είτε σε εργασία που εκπόνησαν μόνοι θεωρούνται οι πιο σημαντικές. Ο Πίνακας 11 δείχνει καθαρά πως οι φοιτητές που εργάστηκαν σε ομάδες πήραν καλύτερους βαθμούς από αυτούς που εργάστηκαν μόνοι τους. Είναι ενδιαφέρον ότι οι φοιτητές που πήραν 3 ήταν αυτοί που χρησιμοποίησαν ελάχιστα τις κατ' ιδίαν συναντήσεις με τους καθηγητές, αντικαθιστώντας τες με λιγοστές συζητήσεις μέσω email. Ένας σημαντικός αριθμός φοιτητών που πέτυχαν την υψηλότερη (4 ή 5) και τη χαμηλότερη (1 ή 2) βαθμολογία χρησιμοποίησαν πολύ περισσότερο τις κατ' ιδίαν συναντήσεις με τους καθηγητές. Αυτό όμως δε σημαίνει απαραίτητα ότι οι φοιτητές κατανόησαν αυτά που συζητήθηκαν στις συναντήσεις. Η ανάλυση αυτού του θέματος είναι ένας από τους μελλοντικούς στόχους, μαζί με μία συγκριτική έρευνα για την εύρεση των κακών μαθησιακών μεθόδων [91].

Πιο πριν κάναμε λόγο για "πειραματική" επανασχεδίαση του μαθήματος "Open Source and Software Quality" γιατί η διδασκαλία και η

εκμάθηση πραγματοποιήθηκαν για πρώτη φορά με τους τρόπους που περιγράφει το άρθρο. Η προσέγγιση της Εκπαίδευσης που εστιάζει σε επίλυση προβλήματος φάνηκε να είναι πολύ κατάλληλη για τις μαθησιακές ανάγκες των φοιτητών (μετα-γνωστική κατανόηση, αυτοαξιολόγηση, αξιολόγηση των συνεργατών τους) αλλά και για τα επιθυμητά μαθησιακά αποτελέσματα του μαθήματος (π.χ. ικανότητες επίλυσης προβλημάτων, σκέψη και επιχειρηματολογία, κ.ά.), όπως αυτά περιγράφονται στα προηγούμενα κεφάλαια. Πράγματι, σύμφωνα με τα σχόλια των φοιτητών, οι στόχοι της απόκτησης απαραίτητης και εξειδικευμένης γνώσης επιτεύχθηκαν. Επιπλέον, οι φοιτητές κέρδισαν ικανότητες μοντελοποίησης και αφηρημένης προδιαγραφής συστημάτων. Οι φοιτητές λειτούργησαν ως επαγγελματίες στον έλεγχο καλώς-ορισμένων προτύπων, εξέτασαν λεπτομερώς τις δοσμένες πληροφορίες, και τέλος, παρακολούθησαν και έλεγξαν διεξοδικά τη διαδικασία εκπαίδευσής τους.

Τέλος, οι εκπαιδευτές είχαν τις ακόλουθες σκέψεις σχετικά με τις βοηθητικές συνεδρίες: Η (αμφίδρομη) παροχή σχολίων μπορεί να αποτελέσει ένα πολύ ισχυρό εργαλείο όταν παρέχεται καλά. Ερευνητές της εκπαίδευσης [92] υποστηρίζουν την ιδέα πως λιγότερη διδασκαλία, λιγότερες διαλέξεις και περισσότερη παροχή σχολίων δίνει καλύτερα εκπαιδευτικά αποτελέσματα. Η μη παροχή σχολίων, λόγω π.χ. έλλειψης χρόνου, σημαίνει πως δε θα υπάρξει χρόνος για να γίνει σωστή μάθηση [93]. Γενικά, τα σχόλια εξυπηρετούν πολλούς σκοπούς, από διόρθωση μέχρι συνεχή ανάπτυξη. Ωστόσο, η παροχή σχολίων μπορεί να είναι αποτελεσματική μόνο όταν οι φοιτητές είναι σε θέση να τα κατανοήσουν και να δράσουν βάσει αυτών [94]. Δηλαδή τα σχόλια αυτά πρέπει να παρέχονται:

- έγκαιρα, ώστε να μπορούν να χρησιμοποιηθούν άμεσα
- σχετικά, ώστε ο πάροχος να εμπνέει εμπιστοσύνη
- εμπειρικά, ώστε να είναι κατανοητά
- συναισθηματικά, ώστε να μπορούν να εμπνεύσουν
- κατάλληλα.

Για να είναι αποτελεσματική, η παροχή σχολίων πρέπει να είναι μια συνεχής διαδικασία βελτίωσης. Για το λόγο αυτό, μία κεντρική ιδέα της επανασχεδίασης του μαθήματος ήταν να χτιστεί ένα σύστημα συνεχής και λειτουργικής παροχής σχολίων. Αυτό σημαίνει ότι τα σχόλια αυτά θα πρέπει να είναι δοσμένα έγκαιρα, σε κατάλληλη ποσότητα και κατάλληλα παρεχόμενα ώστε να εξυπηρετήσουν τους κύριους σκοπούς της σωστής εκμάθησης των εννοιών της ποιότητας λογισμικού.

Σε γενικές γραμμές, η Problem-Focused Education βοήθησε τους φοιτητές να γίνουν πιο ολοκληρωμένοι στο ρόλο του εκπαιδευόμενου και τους βοήθησε να αναλάβουν την ευθύνη της ανάπτυξης του εαυτού τους (επαγγελματικά και ηθικά) αλλά και της κοινότητας του ανοικτού λογισμικού. Αυτά ήταν μερικά από τα επιθυμητά μαθησιακά αποτελέσματα (βλέπε Πίνακα 9). Επιπρόσθετα, έχουμε και τα προφορικά σχόλια/αξιολογήσεις των μαθητών από το τελευταίο μάθημα (την παρουσίαση των εργασιών), που μας λένε ότι οι ίδιοι οι φοιτητές βρήκαν αυτό το είδος μάθησης μερικές φορές δύσκολο αλλά και πιο ευχάριστο από τις παραδοσιακές εκπαιδευτικές μεθόδους.

4.5.6 Σύνοψη και μελλοντικές ιδέες

Το κεφάλαιο αυτό αντιμετώπισε θέματα σχετικά με τα αποτελέσματα της προσέγγισης της Εκπαίδευσης που εστιάζει σε επίλυση προβλήματος αλλά και τη σημασία των σχολίων και αξιολογήσεων. Τα θέματα αυτά αναδείχτηκαν μέσα από ένα "νέο" μάθημα πάνω στο ΕΛ/ΛΑΚ και στην ποιότητα λογισμικού. Σε αυτό το πλαίσιο, η έννοια της μάθησης και της επιμόρφωσης πάνω στη διαχείρισης της ποιότητας λογισμικού γίνεται πολύ απαιτητική.

Στην αρχή αυτής της προσπάθειας (Μάρτιος 2013), οι συγγραφείς ήταν περίεργοι και σε κάποιο βαθμό διστακτικοί και απρόθυμοι να δοκιμάσουν. Στο τέλος του εξαμήνου (Ιούνιος 2013), συνειδητοποίησαν ότι τελικά ήταν μία αποκαλυπτική εκπαιδευτική εμπειρία. Σύμφωνα με τα σχόλια των φοιτητών και τις παρατηρήσεις των συγγραφέων, η διαδικασία της μάθησης ήταν ενδιαφέρουσα, δυνατή, αυτο-οργανούμενη, αυτοκατευθυνόμενη και μη-αναμενόμενη όσον αφορά τους βαθμούς και τα τελικά αποτελέσματα. Η μάθηση έγινε με έναν ανοικτό, μη-ιδιόκτητο τρόπο και σε συνθήκες που ευνοούν μη συμβατικές μεθόδους διδασκαλίας.

Η διδακτική ομάδα των συγγραφέων του άρθρου αυτού διεξάγει συνεχή έρευνα για την καταλληλότητα της προσέγγισης της Problem-Focused Education για την αντιμετώπιση άλλων ζητημάτων μάθησης (όπως δημιουργικές / κριτικές ικανότητες σκέψης και κοινωνικοτεχνολογικές καινοτομίες) στο συγκεκριμένο μάθημα [41, 87, 88, 95]. Επιπλέον ιδέες έρευνας περιλαμβάνουν άλλα είδη εργασιών για το μάθημα αλλά και προσπάθεια για περισσότερη αυτο-αξιολόγηση αλλά και αξιολόγηση των συνεργατών. Οι ιδέες αυτές πιθανόν να αλλάξουν τη μορφή του προβλήματος που χρησιμοποιήθηκε, δηλαδή την Τυπική Προδιαγραφή Ανοικτών Προτύπων. Εναλλακτικά το μέλλον του μαθήματος μπορεί να τονίσει περισσότερο τη σημασία των ερευνητικών δεξιοτήτων αφού τα

ανοικτά δεδομένα, το ΕΛ/ΛΑΚ αλλά και οι κοινότητες του έχουν γίνει το επίκεντρο αυξημένου ερευνητικού ενδιαφέροντος.

Τέλος, ήταν εξαιρετικά ενδιαφέρον να σταθούμε στο πώς οι σπουδαστές ανταποκρίθηκαν στη διδασκαλία των τυπικών προδιαγραφών· τα αποτελέσματα ήταν πολύ ενθαρρυντικά. Παρά το ότι οι τυπικές μέθοδοι θεωρούνται δύσκολες και πολυέξοδες στην εκμάθηση (από θέμα χρόνου και πόρων), η δικιά μας εμπειρία δείχνει κάτι πιο κοντά στα συμπεράσματα του Sommerville [4], όπως αυτά παρουσιάστηκαν στο Κεφάλαιο 2.2.2. Μέσα σε λιγότερο από 3 μήνες, μεταπτυχιακοί σπουδαστές κατάφεραν να φτάσουν σε τέτοιο επίπεδο εξοικείωσης με τις τυπικές προδιαγραφές που κατάφεραν να γράψουν δικές τους προδιαγραφές. Εύκολα μπορεί κάποιος να φανταστεί πως σε κάποιο εργασιακό περιβάλλον που πιθανόν η επιμόρφωση θα ήταν πιο επικεντρωμένη στις ανάγκες της εργασίας, το χρονικό αυτό πλαίσιο μπορεί να γίνει και αρκετά μικρότερο.

4.6 Επέκταση της Αλγεβρικής Προδιαγραφής κοινωνικού δικτύου

4.6.1 Εισαγωγή

Η προδιαγραφή του κοινωνικού δικτύου του Κεφαλαίου 4.2, εκτός από το να δείχνει πώς είναι δυνατός ο σχεδιασμός και η προδιαγραφή ενός πρωτοκόλλου με χρήση Τυπικών Μεθόδων, δείχνει και μία άλλη δυνατότητα των Αλγεβρικών Προδιαγραφών: δείχνει πώς ένα σύστημα μπορεί να χτιστεί πάνω σε ένα άλλο σύστημα κάνοντας χρήση του modularization που είναι εγγενής ιδιότητα των Τυπικών Μεθόδων. Πράγματι, στο Κεφάλαιο 4.2 χτίσαμε πρώτα μία απλή έκδοση ενός κοινωνικού δικτύου μοντελοποιώντας το προφίλ ενός χρήστη και ό,τι το συνοδεύει. Όταν αργότερα θελήσαμε να δημιουργήσουμε την έννοια του "δικτύου" ως μιας συλλογής προφίλ, κάτι δηλαδή που δείχνει πώς ένα πλήθος προφίλ χρηστών "ενώνεται" και σχηματίζει μία ευρύτερη έννοια, αυτής του δικτύου, τότε απλά επεκτείναμε την αρχική μας προδιαγραφή, εισάγοντας την έννοια της σύνθεσης συμπεριφοριακών αντικειμένων και την εφαρμόσαμε στην προδιαγραφή μας, δημιουργώντας τελικά την προδιαγραφή του Κεφαλαίου 4.2.

Στο κεφάλαιο αυτό θα χτίσουμε πάνω στην προδιαγραφή του Κεφαλαίου 4.2, επεκτείνοντας την προδιαγραφή ώστε να επιτρέπει χαρακτηριστικά που συναντάμε σε "νεότερες" εκδόσεις των κοινωνικών δικτύων και πιο συγκεκριμένα, το χαρακτηριστικό των multimedia μηνυμάτων με "ημερομηνία λήξης". Για να επεκταθεί έτσι η προδιαγραφή θα χρεια-

στεί να εισάγουμε την έννοια του πραγματικού χρόνου στο σύστημά μας και για να γίνει αυτό θα χρησιμοποιήσουμε τη θεωρία των *Χρονικών Συστημάτων Παρατηρήσεων-Μεταβάσεων*, όπως αυτή αναπτύχθηκε στο Κεφάλαιο 2.3.5. Το TOTS έχει χρησιμοποιηθεί για την προδιαγραφή και επαλήθευση ιδιοτήτων για πρωτόκολλα όπως: Asynchronous Data Sending ([33]), Fischer's protocol ([33]), Timed Efficient Stream Loss Tolerant Authentication (*TESLA*, [96]).

4.6.2 Περιγραφή της επέκτασης

Το 2011, η ομάδα των Spiegel, Murphy και Brown δημιούργησε το Snapchat ([97]), μια πλατφόρμα για iOS και Android ανταλλαγής (κυρίως) εικόνων με σύντομη ημερομηνία λήξης μεταξύ χρηστών της υπηρεσίας. Τα εικονομηνύματα αυτά μπορούσαν να σταλούν μόνο μεταξύ αμοιβαίων "φίλων" και είχαν το μοναδικό χαρακτηριστικό ότι ο παραλήπτης μπορούσε να τα δει μόνο μία φορά· τα εικονομηνύματα αυτά διαγράφονταν μετά από την υπηρεσία. Αυτή η ιδέα των "προσωρινών" εικονομηνυμάτων έγινε γρήγορα τόσο δημοφιλής που άλλες υπηρεσίες κοινωνικών δικτύων, όπως π.χ. το Facebook ή το Instagram, την υιοθέτησαν με μικρές αλλαγές ([98, 99]).

Στο κεφάλαιο αυτό θα επιχειρήσουμε να επεκτείνουμε την προδιαγραφή του Κεφαλαίου 4.2 ώστε να υποστηρίζει τέτοια προσωρινά μηνύματα. Η δυνατότητα που θα προδιαγράψουμε είναι η εξής:

Ιδιότητα 4.8. *Αν υποθέσουμε δύο χρήστες του κοινωνικού δικτύου, τους* A *και* B *και τον* A *να αποστέλλει στον* B *ένα μήνυμα* M *με ημερομηνία λήξης T_{max}. Θέλουμε τον* B *να μπορεί να δει το μήνυμα* M *μόνο αν τηρούνται οι παρακάτω προϋποθέσεις:*

1. *Ο* A *είναι φίλος του* B*.*
2. *Το μήνυμα είναι "διαθέσιμο".*
3. *Ο* B *ανοίγει το μήνυμα εντός χρόνου T, με $T \leq T_{max}$.*

Αν ο B *δει τελικά το μήνυμα* M *τότε αυτό διαγράφεται και σταματάει πια να είναι διαθέσιμο, ακυρώνοντας έτσι την ισχύ της Ιδιότητας 4.8.2.* □

Πέρα από τους τελεστές του Κεφαλαίου 4.2.3, θα προσθέσουμε τους εξής (υποθέτοντας ότι με B, SM, P, SL συμβολίζουμε τύπους δεδομένων που αντιστοιχούν σε αληθοτιμές, προσωρινά εικονομηνύματα, προφίλ χρήστη και λίστες προσωρινών μηνυμάτων):

- Παρατηρητές
  - $emptySnapList : P, Y \rightarrow B$, ελέγχει αν η λίστα με τα εισερχόμενα προσωρινά μηνύματα ενός χρήστη είναι άδεια. Αρχικά επιστρέφει true.
  - $snapContent : P, Y \rightarrow SM$, επιστρέφει το προσωρινό εικονομήνυμα, εφόσον ο χρήστης έχει κάποιο και μπορεί να το δει. Αρχικά επιστρέφει `nilphoto`.
  - $snapList : P, Y \rightarrow SL$, επιστρέφει τη λίστα με τα διαθέσιμα προσωρινά εικονομηνύματα ενός χρήστη. Αρχικά επιστρέφει την άδεια λίστα.
- Μεταβάσεις
  - $sendSnap : P, P, SM, Y \rightarrow Y$. Ένας χρήστης στέλνει ένα προσωρινό εικονομήνυμα σε έναν άλλο χρήστη. Η μη-χρονική αναγκαία συνθήκη είναι οι δύο χρήστες να είναι φίλοι. Χρονική συνθήκη δεν υπάρχει
  - $recSnap : P, SM, SL, Y \rightarrow Y$. Ένας χρήστης διαβάζει ένα προσωρινό εικονομήνυμα. Η μη χρονική αναγκαία συνθήκη είναι το μήνυμα να βρίσκεται στη λίστα με τα εισερχόμενα μηνύματά του. Η χρονική αναγκαία συνθήκη είναι το μήνυμα να διαβαστεί πριν την ημερομηνία λήξης του.

4.6.3 Προδιαγραφή της επέκτασης

Οι δύο πρώτοι όροι της Ιδιότητας 4.8 είναι εύκολο να προδιαγραφούν:

- Η ιδιότητα της "φιλίας" έχει ήδη περιγραφεί στο Κεφάλαιο 4.2.4. Το αν οι `A, B` είναι φίλοι μας το δίνει ο παρατηρητικός τελεστής `friends`, αν δηλαδή ικανοποιείται η συνθήκη:

  ```
  A //in friends(B)
  ```

- Το αν το μήνυμα `M` είναι διαθέσιμο εξαρτάται από το αν θα βρίσκεται στην αντίστοιχη λίστα με τα διαθέσιμα μηνύματα του `B`, (`SnapList`) κάτι που απαντάει η συνθήκη:

  ```
  M //in SnapList(B)
  ```

Ο τρίτος όρος της Ιδιότητας 4.8 (και συγκεκριμένα η μετάβαση `recSnap`) χρησιμοποιεί χρόνο άρα θα χρειαστεί να μετατρέψουμε την

υπάρχουσα προδιαγραφή από *OTS* (Κεφάλαιο 2.3.2) σε *TOTS* (Κεφάλαιο 2.3.5). Θα περιγράψουμε τους χρονικούς περιορισμούς του συστήματος που καθορίζουν αν ένα μήνυμα μπορεί να προβληθεί από τον παραλήπτη του ή έχει "λήξει". Για να γίνει αυτό, θα προσθέσουμε τρία ρολόγια και ένα σύνολο χρονικών μεταβάσεων [33]. Ένα από τα ρολόγια αυτά θα είναι το κύριο ρολόι και τα υπόλοιπα θα είναι τα $l_{recSnap}, u_{recSnap}$. Θα έχουμε επίσης δύο συναρτήσεις $d^{min}_{recSnap}, d^{max}_{recSnap}$.

Η αναγκαία μη-χρονική συνθήκη $c_{recSnap}$ της μετάβασης `recSnap` θα είναι η ύπαρξη του προσωρινού εικονομηνύματος στη λίστα με τα διαθέσιμα εικονομηνύματα του χρήστη. Στην αρχική κατάσταση (`init`) του, το σύστημα ξεκινάει με τη λίστα των διαθέσιμων προσωρινών εικονομηνυμάτων κάθε χρήστη να είναι κενή, άρα $c_{recSnap}(init) = false$. Σύμφωνα με τη θεωρία του Κεφαλαίου 2.3.5, οι αρχικές τιμές των $l_{recSnap}(init)$ και $u_{recSnap}(init)$ θα είναι:

$$l_{recSnap}(init) = 0$$
$$u_{recSnap}(init) = \infty$$

Εφαρμόζοντας τη μετάβαση `recSnap` σε μία κατάσταση $u \in Y$ έτσι ώστε $c_{recSnap}(u), l_{recSnap}(u) \leq now(u)$, με $u' = recSnap(u)$:

$$l_{recSnap}(u') = \begin{cases} now(u) + d^{min}_{recSnap} & \text{αν } c_{recSnap}(u') \\ 0, & \text{αλλιώς} \end{cases}$$
$$u_{recSnap}(u') = \begin{cases} now(u) + d^{max}_{recSnap} & \text{αν } c_{recSnap}(u') \\ \infty, & \text{αλλιώς} \end{cases}$$

Η ελάχιστη καθυστέρηση του `recSnap` είναι 0 και η μέγιστη T_{max}, ή d_1. Δηλαδή:

$$d^{min}_{recSnap}(u) = 0$$
$$d^{max}_{recSnap}(u) = T_{max} = d_1$$

Ο Κώδικας 50 δείχνει τις υπογραφές των τελεστών που προσθέσαμε στο κύριο module του κοινωνικού δικτύου, όπως αυτό αναπτύχθηκε στο Κεφάλαιο 4.2.4. Οι τελεστές l και u αντιστοιχούν στα ρολόγια $l_{recSnap}$ και $u_{recSnap}$ αντίστοιχα.

```
* [ Sys * ]
[ SnapMsg, SnapList ]
-- an arbitrary initial state
op init      : -> Sys
```

```
-- observational operators
bop emptySnapList  : ProfileSys Sys -> Bool
bop snapContent  : ProfileSys Sys -> SnapMsg
bop snapList     : ProfileSys Sys -> SnapList
bop now       : Sys     -> Real+
bop l         : ProfileSys SnapMsg SnapList Sys -> Real+
bop u         : ProfileSys SnapMsg SnapList Sys -> NzTimeval
-- action operators
bop sendSnap     : ProfileSys ProfileSys SnapMsg Sys -> Sys
bop recSnap      : ProfileSys SnapMsg SnapList Sys -> Sys
bop tick       : Sys Real+ -> Sys
```

Κώδικας 50: Οι υπογραφές των νέων τελεστών στο main module του κοινωνικού δικτύου

Αν οι μεταβλητές P, SM, SL εκφράζουν ορατούς τύπους `ProfileSys`, `SnapMsg` και `SnapList` αντίστοιχα τότε ο Κώδικας 51 δείχνει τις εξισώσεις που περιγράφουν την αρχική κατάσταση του συστήματος. Ο όρος `emptySL` υποδηλώνει άδεια λίστα προσωρινών εικονομηνυμάτων, ενώ ο `noMessage` συμβολίζει ένα `null` μήνυμα.

```
-- initial state
eq  emptySnapList(P, init) = true .
eq  snapContent(P, init)  = noMessage .
eq  snapList(P, init)   = emptySL .
eq  now(init)      = 0 .
eq  l(P, SM, SL, init)   = 0 .
eq  u(P, SM, SL, init)   = oo .
```

Κώδικας 51: Οι εξισώσεις της αρχικής κατάστασης

Αν η μεταβλητή S εκφράζει έναν κρυφό τύπο `Sys` τότε ο Κώδικας 52 δείχνει τον ορισμό του τελεστή μετάβασης `recSnap`.

```
op c-recSnap : ProfileSys SnapMsg SnapList Sys -> Bool
eq  c-recSnap(P, SM, SL S) = ( SM /in SL and l(P, SM, SL, S) <= now(S)) .
--
ceq emptySnapList(P, recSnap(P, SM, SL, S)) =
        if  c-recSnap(P, SM, SL, S) then false
                else emptySnapList(P,S)
        fi.
eq  snapContent(P, recSnap(P, SM, SL, S)) =
        if  c-recSnap(P, SM, SL, S) then SM
                else noMessage
        fi.
eq  snapList(P, recSnap(P, SM, SL, S)) = snapList(P, S) .
```

```
eq  now(recSnap(P, SM, SL, S)) = now(S) .
eq  l(P, SM, SL, recSnap(P, SM, SL, S)) = l(P, SM, SL, S) .
ceq u(P, SM, SL, recSnap(P, SM, SL, S)) = d1 − now(S) if l(S) <= now(S) .
ceq recSnap(P, SM, SL, S) = S if not c−recSnap(P, SM, SL, S) .
```

Κώδικας 52: Ο ορισμός του `recSnap`

Μπορούμε έτσι να χρησιμοποιήσουμε την επέκταση της προδιαγραφής του κοινωνικού δικτύου με τη χρήση ιδιοτήτων πραγματικού χρόνου για να αποδείξουμε ιδιότητες ασφαλείας του συστήματος χρησιμοποιώντας τη μέθοδο αποδείξεων Proof Scores (Κεφάλαιο 2.3.3).

5 Ανακεφαλαίωση, συνεισφορά και προτάσεις για μελλοντική έρευνα

5.1 Ανακεφαλαίωση

Η παρούσα διατριβή εστίασε στη σημαντικότητα των Τυπικών Μεθόδων (και ειδικότερα των Αλγεβρικών Προδιαγραφών) για την προδιαγραφή προτύπων αλλά και στη χρήση τους για την επαλήθευση και την εγκυροποίηση ενός συστήματος. Για το σκοπό αυτό μελετήθηκε αρχικά η υπάρχουσα κατάσταση στο πεδίο του σχεδιασμού και επαλήθευσης συστημάτων και ιδιαίτερη αναφορά έγινε στις φυσικές γλώσσες και πώς η χρήση τους στις προδιαγραφές των απαιτήσεων ενός συστήματος αλλά και του ίδιου του συστήματος μπορεί να δημιουργήσει προβλήματα.

Στη συνέχεια παρουσιάστηκαν οι Τυπικές Μέθοδοι ως μια διαφορετική προσέγγιση στο σχεδιασμό και την ανάλυση ενός συστήματος με μαθηματικό υπόβαθρο και η τυπική μεθοδολογία στο σχεδιασμό λογισμικού κάνοντας χρήση τυπικών μεθόδων και συγκεκριμένα, Αλγεβρικών Προδιαγραφών. Οι Αλγεβρικές Προδιαγραφές είναι τεχνικές προσδιορισμού συστημάτων βάσει των λειτουργιών τους και των σχέσεων που σχηματίζουν αυτές μεταξύ τους. Χρησιμοποιούν διάφορες άλγεβρες και η χρήση τους έχει πλεονεκτήματα όπως μικρότερο μέγεθος και μεγαλύτερη σαφήνεια προδιαγραφών αλλά και δυνατότητα επαλήθευσης ιδιοτήτων του συστήματος. Εξετάσαμε τις διαφορετικές γλώσσες αλγεβρικών προδιαγραφών και δώσαμε ένα παράδειγμα προδιαγραφής συστήματος και χρησιμοποιήσαμε την προδιαγραφή αυτή για να εξετάσουμε την ισχύ μιας επιθυμητής ιδιότητας. Η γλώσσα αλγεβρικών προδιαγραφών που χρησιμοποιήσαμε στην διατριβή είναι η CafeOBJ χωρίς όμως αυτό να εξειδικεύει πολύ τη διατριβή αφού θα μπορούσαμε να χρησιμοποιήσουμε οποιαδήποτε άλλη αντίστοιχη γλώσσα. Παρουσιάσαμε συνοπτικά λοιπόν την CafeOBJ και δείξαμε πώς προδιαγράφουμε με αυτή ένα σύστημα φέρνοντάς το στη μορφή ενός συστήματος Παρατηρήσεων Μεταβάσεων (*OTS*) και πώς το σύστημα OTS με τη χρήση της μεθοδολογίας των Proof Scores χρησιμοποιείται για την επαλήθευση ιδιοτήτων του. Επίσης, δείξαμε δύο σημαντικές επεκτάσεις της μεθοδολογίας OTS/Proof Scores:

1. τη σύνθεση συμπεριφοριακών αντικειμένων, μια τεχνική που ενώνει πλήθος βασικών αντικειμένων σε ένα πιο σύνθετο, εισάγοντας έτσι σχέσεις μεταξύ των βασικών αντικειμένων αλλά και σχέσεις του σύνθετου αντικειμένου με αυτά.

2. τα Χρονικά Συστήματα Παρατηρήσεων Μεταβάσεων (*TOTS*) που δημιουργούν συστήματα που χρησιμοποιούν την έννοια του χρόνου, μία έννοια απαραίτητη στην προδιαγραφή συστημάτων πραγματικού χρόνου.

Τέλος, παρουσιάσαμε τα Πρότυπα· τι είναι, πώς θεσπίζονται, τα πλεονεκτήματά τους και αναλύσαμε ιδιαίτερα τα Ανοικτά Πρότυπα και αιτιολογήσαμε γιατί αυτά είναι σημαντικά. Η κεντρική ιδέα της διατριβής αυτής είναι η χρήση Τυπικών Μεθόδων για την δημιουργία προδιαγραφών για ένα (ανοικτό) πρότυπο. Στηρίξαμε τη θέση ότι μία τέτοια προδιαγραφή (και ιδιαίτερα μία αλγεβρική προδιαγραφή) έχει σημαντικά πλεονεκτήματα έναντι μίας προδιαγραφής ενός προτύπου που γράφεται σε φυσική γλώσσα. Επιχειρηματολογήσαμε για τα πλεονεκτήματα μίας τέτοιας προσέγγισης και εξετάσαμε σχετικές εργασίες.

Για τη στήριξη της παραπάνω θέσης μελετήσαμε κάποιες περιπτώσεις:

1. Εξετάσαμε την Abstract Syntax Notation One (ASN.1) για να εξετάσουμε πώς κάποιος που τη χρησιμοποιεί ως πρότυπο για την τυποποίηση της επικοινωνίας κάποιου συστήματός του θα μπορούσε να χρησιμοποιήσει την τυπική έκδοση της ASN.1 που προτείνουμε για να μπορέσει να ελέγξει την ισχύ ιδιοτήτων του συστήματος. Το κεφάλαιο αυτό έδειξε πώς κάποιος θα μπορούσε να χρησιμοποιήσει μία προδιαγραφή που ήδη είχε γράψει (σε ASN.1) για να αποκομίσει περισσότερα από αυτήν.

2. Εξετάσαμε δύο υπάρχοντα ανοικτά πρότυπα: Το Open Document Architecture (ODA) και το Rich Site Summary (RSS). Πήραμε τις (ογκώδεις) προδιαγραφές που είναι διαθέσιμες για τα πρότυπα αυτά online και από την κατανόηση μας πάνω σε αυτές, επιχειρήσαμε να παρουσιάσουμε μία τυπική τους έκδοση, τονίζοντας έτσι τα πλεονεκτήματα της τυπικής τους έκδοσης. Ειδικά στην περίπτωση του RSS, δείξαμε πως η συγγραφή μιας τυπικής προδιαγραφής έδειξε αντιφάσεις και προβληματικά σημεία στην πρωτότυπη προδιαγραφή, κάτι που δε θα συνέβαινε αν είχε γραφεί και μία τυπική προδιαγραφή παράλληλα. Αν και τα πρότυπα αυτά (ή οι προδιαγραφές τους) δε σχεδιάστηκαν από εμάς, καταφέραμε να εντοπίσουμε προβληματικά σημεία, κάτι που σίγουρα θα αποτελούσε πρόβλημα για κάποιον σχεδιαστή που θέλει να κάνει το σύστημα του 100% συμβατό με το συγκεκριμένο πρότυπο. Αν το πρότυπο ήταν δικό μας τότε η σύνταξη τυπικής προδιαγραφής θα εντόπιζε το πρόβλημα πολύ νωρίς στο στάδιο του σχεδιασμού.

3. Δημιουργήσαμε από την αρχή ένα δικό μας πρότυπο· ένα σύστημα κοινωνικής δικτύωσης χρησιμοποιώντας αλγεβρικές προδιαγραφές και συγκεκριμένα τη μεθοδολογία OTS με την επέκταση της σύνθεσης συμπεριφοριακών αντικειμένων. Εξετάσαμε την προδιαγραφή αυτή για να ελέγξουμε την ισχύ κάποιων ιδιοτήτων του με τη μέθοδο των Proof Scores. Δείξαμε πως η σχεδίαση προτύπων με αυτό τον τρόπο μάς ανάγκασε να βρούμε γρήγορα κάποια λάθη και να τα διορθώσουμε πριν το σύστημα αυτό περάσει σε μετέπειτα στάδια της ανάπτυξής του, κερδίζοντας έτσι χρόνο και πόρους.

4. Επειδή ένα σύνηθες επιχείρημα κατά των Τυπικών Μεθόδων είναι η δυσκολία κατανόησης και χρήσης τους, παρουσιάσαμε μία προσωπική εμπειρία από τη διδασκαλία Τυπικών Μεθόδων σε μεταπτυχιακούς σπουδαστές που δεν είχαν κάποια προηγούμενη επαφή με το χώρο. Τα πολύ ενθαρρυντικά αποτελέσματα που πήραμε στο τέλος του εξαμήνου συνηγορούν στο ότι η διδασκαλία και εκμάθηση των Τυπικών Μεθόδων ίσως να είναι ευκολότερες απ' όσο πιστεύεται.

5.2 Συνεισφορά

Η επιστημονική συνεισφορά της διατριβής αυτής συνοψίζεται στα ακόλουθα:

- Εισάγουμε μία νέα έννοια· το "Τυπικό (Ανοικτό) Πρότυπο", ένα (ανοικτό) πρότυπο που κυκλοφορεί συνοδευόμενο από μία τυπική προδιαγραφή μαζί με την προδιαγραφή που είναι γραμμένη σε φυσική γλώσσα, αν όχι σε αντικατάσταση αυτής.

- Επιχειρηματολογούμε για τα σημαντικά πλεονεκτήματα της χρήσης Αλγεβρικών Προδιαγραφών για την προδιαγραφή προτύπων αλλά και επιχειρούμε να αντικρούσουμε την επικρατούσα άποψη ότι η συγγραφή τυπικών προδιαγραφών είναι μία πιο δαπανηρή διαδικασία από την τρέχουσα.

- Παρουσιάζουμε την αλγεβρική προδιαγραφή προτύπων που κυκλοφορούν ήδη και εξετάζουμε αν η τυπική έκδοση έχει τα πλεονεκτήματα που περιμένουμε.

- Δείχνουμε πώς οι αλγεβρικές προδιαγραφές μπορούν να χρησιμοποιηθούν κατά το σχεδιασμό ενός προτύπου/πρωτοκόλλου και πώς έτσι μπορούμε να βρούμε λάθη πριν το σύστημα περάσει σε

επόμενο στάδιο και κάνει τη διόρθωσή τους πιο ακριβή διαδικασία.

- Τέλος, παρουσιάζουμε την modular δομή των αλγεβρικών προδιαγραφών και πώς αυτή μπορεί να χρησιμοποιηθεί ώστε να εκμεταλλεύεται υπάρχοντα modules για να χτίζει μεγαλύτερα και πιο σύνθετα.

5.3 Προτάσεις για μελλοντική έρευνα

Όσον αφορά στην αλγεβρική προδιαγραφή προτύπων, θα είχε ενδιαφέρον η χρήση αλγεβρικών μεθόδων για την προδιαγραφή περισσότερων και διαφορετικών προτύπων χρησιμοποιώντας διαφορετικά εργαλεία και όχι μόνο την CafeOBJ. Κάθε εργαλείο έχει τα δικά του πλεονεκτήματα και μειονεκτήματα και ειδικεύεται σε διαφορετικούς τομείς. Η χρήση διαφορετικών εργαλείων σε πρότυπα διαφορετικής φύσης θα έδειχνε ακόμα καλύτερα τα πλεονεκτήματα των αλγεβρικών προδιαγραφών καθώς θα έδειχνε πως μπορεί να καλύψει μεγαλύτερη γκάμα προτύπων.

Επίσης, η προδιαγραφή του κοινωνικού δικτύου που περιγράφεται στο Κεφάλαιο 4.2 μπορεί να επεκταθεί περαιτέρω, καθώς υπάρχουν πολλές ακόμα ιδιότητες ασφαλείας που θα μπορούσαμε να περιγράψουμε και να ελέγξουμε. Και η ίδια η προδιαγραφή του προτύπου θα μπορούσε να επεκταθεί· το κεφάλαιο 4.6 εισάγει την έννοια του χρόνου και η νέα αυτή προδιαγραφή θα μπορούσε να χρησιμοποιηθεί για την περιγραφή και τον έλεγχο ακόμα περισσότερων ιδιοτήτων του συστήματος. Τα κοινωνικά δίκτυα είναι μία περιοχή που εξελίσσεται συνεχώς με νέα χαρακτηριστικά και δυνατότητες και αυτό προσφέρει πολλές ευκαιρίες για μελλοντική έρευνα.

Αναφορές

[1] Pankaj Jalote. *An Integrated Approach to Software Engineering*. Springer Compass International. Springer New York, 2013.

[2] IEEE. IEEE recommended practice for software requirements specifications. *IEEE Std 830-1993*, 1994.

[3] Digital imaging and communications in medicine (DICOM). `http://dicom.nema.org/`, 2014. Accessed: 2014-08-11.

[4] Ian Sommerville. *Software Engineering*. Addison-Wesley, Harlow, England, 9 edition, 2010.

[5] Michael Jackson. *Software Requirements & Specifications: A Lexicon of Practice, Principles and Prejudices*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995.

[6] Jonathan P. Bowen, Peter T. Breuer, and Kevin C. Lano. A compendium of formal techniques for software maintenance. *BCS/IEE Software Engineering Journal*, 8:253–262, 1993.

[7] Robert M. Hierons, Kirill Bogdanov, Jonathan P. Bowen, Rance Cleaveland, John Derrick, Jeremy Dick, Marian Gheorghe, Mark Harman, Kalpesh Kapoor, Paul Krause, Gerald Lüttgen, Anthony J. H. Simons, Sergiy Vilkomir, Martin R. Woodward, and Hussein Zedan. Using formal specifications to support testing. *ACM Comput. Surv.*, 41(2):9:1–9:76, February 2009.

[8] Christer Baier and Joost-Pieter Katoen. *Principles of Model Checkin*. MIT Press, Cambridge, Massachusetts, London, England, 2008.

[9] Stefania Gnesi and Tiziana Margaria. *Formal Methods for Industrial Critical Systems: A Survey of Applications*. John Wiley & Sons, Inc., Hoboken, New Jersey., 2013.

[10] Antoni Diller. *Z: An introduction to formal methods*. Wiley & Sons, Chichester, West Sussex, England; New York, 2nd edition, 1994.

[11] D. Lightfoot. *Formal Specification using Z*. Macmillan Press., 1991.

[12] J.M. Spivey. An introduction to Z and formal specifications. *Software Engineering Journal*, 4(1):40–50, Jan 1989.

[13] Eleni Berki and Juri Valtanen. Critical and creative mathematical thinking with practical problem solving skills - a new old challenge. In Dimitris Dranidis and Ilias Sakellariou, editors, *Proceedings of the 3rd SouthEast European Workshop on Formal Methods (SEEFM07).*, pages 154–170. South-East European Research Centre (SEERC), November 2007.

[14] H. Ehrig, B. Mahr, I. Classen, and F. Orejas. Introduction to algebraic specification. part 1: Formal methods for software development. *j-COMP-J*, 35(5):460–467, October 1992.

[15] Peter D. Mosses, editor. *CASL Reference Manual: The Complete Documentation of the Common Algebraic Specification Language*, volume 2960 of *Lecture Notes in Computer Science*. Springer-Verlag Berlin Heidelberg, 2004.

[16] Jonathan P. Bowen and Michael G. Hinchey. Ten commandments of formal methods. *IEEE COMPUTER*, 28:56–63, 1994.

[17] Joseph Goguen. The OBJ family. `http://cseweb.ucsd.edu/~goguen/sys/obj.html`. Accessed: 2014-10-30.

[18] Joseph A. Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouannaud. *Introducing OBJ*, pages 3–167. Springer US, Boston, MA, 2000.

[19] Răzvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*, volume 6 of *AMAST Series in Computing*. World Scientific, 1998.

[20] Răzvan Diaconescu, Kokichi Futatsugi, and Kazuhiro Ogata. CafeOBJ: Logical foundations and methodologies. *Computing and Informatics*, 22, 2003.

[21] Joseph A. Goguen and José Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105:217–273, 1992.

[22] Ruazvan Diaconescu and Kokichi Futatsugi. Behavioural coherence in object-oriented algebraic specification. *Journal of Universal Computer Science*, 6(1):74–96, jan 2000. `http://www.jucs.org/jucs_6_1/behavioural_coherence_in_object`.

[23] CafeOBJ official site. `https://cafeobj.org/`. Accessed: 30-09-2016.

[24] Kazuhiro Ogata, Kokichi Futatsugi, and Nec Software Hokuriku. Proof scores in the OTS/CafeOBJ method. In *In Proc. of The 6th IFIP WG6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS 2003), volume 2884 of LNCS*, pages 170–184. Springer, 2003.

[25] Kazuhiro Ogata and Kokichi Futatsugi. Some tips on writing proof scores in the OTS/CafeOBJ method. In Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer, editors, *Algebra, Meaning, and Computation*, volume 4060 of *Lecture Notes in Computer Science*, pages 596–615. Springer Berlin / Heidelberg, 2006.

[26] Răzvan Diaconescu. Behavioural specification for hierarchical object composition. *Theor. Comput. Sci.*, 343(3):305–331, October 2005.

[27] Kokichi Futatsugi, Joseph A. Goguen, and Kazuhiro Ogata. *Verifying Design with Proof Scores*, pages 277–290. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[28] Takahiro Seino. Proof scores. `https://cafeobj.org/intro/en/proofscores.html`. Accessed: 2017-01-18.

[29] Joseph A. Goguen and Grigore Rosu. Hiding more of hidden algebra. In *Proceedings of the Wold Congress on Formal Methods in the Development of Computing Systems-Volume II*, FM '99, pages 1704–1719, London, UK, UK, 1999. Springer-Verlag.

[30] Rolf Hennicker and Michel Bidoit. Observational logic. In Armando Haeberer, editor, *Algebraic Methodology and Software Technology*, volume 1548 of *Lecture Notes in Computer Science*, pages 263–277. Springer Berlin / Heidelberg, 1999. 10.1007/3-540-49253-420.

[31] Razvan Diaconescu, Kokichi Futatsugi, and Shusaku Iida. Component-based algebraic specification and verification in CafeOBJ. In *Proceedings of the Wold Congress on Formal Methods in the Development of Computing Systems-Volume II*, FM '99, pages 1644–1663, London, UK, UK, 1999. Springer-Verlag.

[32] Shusaku Iida, Kokichi Futatsugi, and Razvan Diaconescu. Component based algebraic specifications - behavioural specification for component based software engineering. In *In Seventh OOPSLA*

Workshop on Behavioral Semantics of OO Business and System Specification, 1999.

[33] Kazuhiro Ogata and Kokichi Futatsugi. Modeling and verification of real-time systems based on equations. *Science of Computer Programming*, 66(2):162 – 180, 2007.

[34] Andrew L. Russell. *Open Standards and the Digital Age: History, Ideology, and Networks (Cambridge Studies in the Emergence of Global Enterprise)*. Cambridge University Press, April 2014.

[35] Henk J. de Vries. *Standardization: A Business Approach to the Role of National Standardization Organizations*. Springer US, Nov 1999.

[36] M. Muhonen and E. Berki. An open process for quality assurance in systems development. In *In the Conference Proceedings of Dawson, R., Ross, M., and Staples, G. (Eds): Software Quality Management XIX. Global Quality Issues, Loughborough 18-19 Apr.*, pages 231–241, 2011.

[37] Mirjan Merruko. Utilising open source software development for effective electronic health records development. Master's thesis, School of Information Sciences, University of Tampere, 2013.

[38] Nah Soo Hoe. *Free/Open Source Software, Open Standards*. Elsevier, 2006.

[39] Rajiv Shah, Jay Kesan, and Andrew Kennis. Lessons for open standard policies: a case study of the massachusetts experience. In *Proceedings of the 1st international conference on Theory and practice of electronic governance*, ICEGOV '07, pages 141–150, New York, NY, USA, 2007. ACM.

[40] M. Karjalainen. *Large-scale Migration to an Open Source Office Suite: An Innovation Adoption Study in Finland*. Julkaisusarja A. Department of Computer Sciences, University of Tampere, 2010.

[41] M. Merruko, E. Berki, and P. Nykänen. Open source software process: A potential catalyst for major changes in electronic health record systems. In S. Shaikh, I. Stamelos, and A. Cerone, editors, *In OpenCert 2012 + SEFM 2012 Proceedings (2013)*, 2012.

[42] G. Blake and R.W. Bly. *The elements of technical writing*. Elements of Series. Longman, 1993.

[43] Daniel M. Berry, Erik Kamsties, and Michael M. Krieger. From contract drafting to software specification: Linguistic sources of ambiguity, a handbook. `http://se.uwaterloo.ca/~dberry/handbook/ambiguityHandbook.pdf`, 2003.

[44] Robert S. Boyer and J. Strother Moore. *The Correctness Problem in Computer Science*. International Lecture Series in Computer Science. Academic Press, Orlando, FL, USA, 1981.

[45] J. Valtanen, E. Berki, K Barlas, L. Li, and M. Merruko. Problem-focused education and feedback mechanisms for re-designing a course on open source and software quality. In Uhomoibhi J., Barikzai S., Ross M., and Staples G., editors, *The 18th INSPIRE - INternational conference on Software Process Improvement in Research, Education and Training.*, pages 23–36, London, UK, Southampton Solent University Press., September 2013.

[46] Grant Malcolm and Joseph A. Goguen. An executable course in the algebraic semantics of imperative programs. In *Teaching and Learning Formal Methods*, pages 161–179. Academic, 1996.

[47] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. How amazon web services uses formal methods. *Commun. ACM*, 58(4):66–73, March 2015.

[48] ISO/IEC. *Open Distributed Processing – Basic Reference Model – Part 4: Architectural Semantics*. ISO/IEC 10746-4. International Organization for Standardization, Geneva, Switzerland, 1996.

[49] ISO/IEC. Information technology – Open Distributed Processing – use of UML for ODP system specifications. ISO 19793:2015, International Organization for Standardization, Geneva, Switzerland, 2015.

[50] ISO/IEC. *Information Processing Systems – Open Systems Interconnection – Guidelines for the Application of* Estelle, Lotos *and SDL*. ISO/IEC TR 10167. International Organization for Standardization, Geneva, Switzerland, 1990.

[51] Kenneth J. Turner. *Using Formal Description Techniques: An Introduction to ESTELLE, LOTOS and SDL*. Wiley, 1993.

[52] ISO/IEC. *Information Technology – Framework: Formal Methods in Conformance Testing*. ISO/IEC 13245-1. International Organization for Standardization, Geneva, Switzerland, 1997.

[53] John Larmouth. *ASN.1 complete*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[54] Olivier Dubuisson and Philippe Fouquart. *Communication between heterogeneous systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[55] ASN.1 to C compiler. `https://github.com/vlm/asn1c`. Accessed: 30-09-2016.

[56] JAC (Java ASN.1 compiler). `https://sourceforge.net/projects/jac-asn1/`. Accessed: 30-09-2016.

[57] Ericsson AB. ASN.1 compiler and compile-time support functions. `http://erlang.org/documentation/doc-6.3/lib/asn1-3.0.3/doc/html/asn1ct.html`. Accessed: 30-09-2016.

[58] ASN.1 for Python. `http://pyasn1.sourceforge.net/`. Accessed: 30-09-2016.

[59] Jittisak Senachak, Takahiro Seino, Kazuhiro Ogata, and Kokichi Futatsugi. Provably correct translation from CafeOBJ into java. In *SEKE*, pages 614–619, 2005.

[60] Nancy Ann Lynch. *Distributed algorithms*. The Morgan Kaufmann series in data management systems. Morgan Kaufmann Publishers, 1996.

[61] Kokichi Futatsugi. Verifying specifications with proof scores in CafeOBJ. In *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, pages 3–10, Washington, DC, USA, 2006. IEEE Computer Society.

[62] Charles Kadushin. *Introduction to Social Network Theory: Some Basic Network Concepts and Propositions*. 2004.

[63] Danah Boyd and Ellison Nicole. Social network sites: Definition, history, and scholarship. `http://jcmc.indiana.edu/vol13/issue1/boyd.ellison.html`, 2007. Accessed: 31-08-2011.

[64] N. Cataño, V. Kostakos, and I. Oakley. Poporo: A Formal Framework for Social Networking. In M. Harrison and M. Massink, editors, *Proceedings of Formal Methods for Interactive Systems (FMIS)*, Eindhoven, the Netherlands, November 2009.

[65] José Meseguer. A logical theory of concurrent objects and its realization in the maude language. pages 314–390. MIT Press, Cambridge, MA, USA, 1993.

[66] Egidio Astesiano, Michel Bidoit, Hélène Kirchner, Bernd Krieg-Brückner, Peter D. Mosses, Donald Sannella, and Andrzej Tarlecki. CASL: The common algebraic specification language. *Theoretical Computer Science*, 286(2):153–196, 2002. Article dans revue scientifique avec comité de lecture. A02-R-498 || astesiano02a A02-R-498 || astesiano02a.

[67] Open document architecture and interchange format. `http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=15926`. Accessed: 25-07-2011.

[68] Open document architecture and interchange format: Document structures (T.412). `http://www.itu.int/rec/T-REC-T.412/en`. Accessed: 25-07-2011.

[69] Y. Y. Tang, C. D. Yan, M. Cheriet, and C. Y. Suen. Document analysis and understanding: A brief survey. In *In Proc. 1st Int. Conf. Document Analysis and Recogn.*, pages 17–31, Saint-Malo, France, September 1991.

[70] C. H. Chen, L. F. Pau, P. S. P. Wang, Yuan Y. Tang, Ching Y. Suen, M. Cheriet, Jiming Liu, and J. N. Said. Document analysis and recognition by computers, 1999.

[71] Anoop M Namboodiri and Anil K Jain. Document structure and layout analysis. *East*, pages 1–17, 2007.

[72] RSS 2.0 specification. `http://www.rssboard.org/rss-specification`, March 2009. Accessed: 2014-01-30.

[73] Oscar Raymundo. Hands-on with news in ios 9: Apple's response to facebook and snapchat's content platforms. `http://www.macworld.com/article/2947012/software-news/hands-on-with-news-in-ios-9-apples-response-to\`

hyp{}facebook-and-snapchats-content-platforms.html, July 2015. Accessed: 10-08-2015.

[74] Sample file for RSS v2.0. http://www.rssboard.org/files/sample-rss-2.xml, March 2009. Accessed: 30-06-2013.

[75] W3Schools. XML tutorial. http://www.w3schools.com/xml/, 2014. Accessed: 2014-08-01.

[76] W3Schools. DTD tutorial. http://www.w3schools.com/dtd/, 2014. Accessed: 30-09-2016.

[77] W3Schools. PICS platform for internet content selection. https://www.w3.org/PICS/. Accessed: 2017-02-13.

[78] Dave Winer. The RSS "ttl" element and P2P networks. http://scripting.com/2006/09/07.html\#theRssTtlElementAndP2pNetworks, 2006. Accessed: 2014-08-12.

[79] L. Cohen, L. Manion, and K. Morrison. *Research Methods in Education*. Routledge, 6th edition, 2007.

[80] Martha Lagace. Open source science: A new model for innovation. http://hbswk.hbs.edu/item/5544.html. Accessed: 2016-11-20.

[81] Social learning theory. http://en.wikipedia.org/wiki/Social_learning_theory. Accessed: 2013-08-30.

[82] Socio-economic factors related most recent documents for Europe 2020 programme. http://ec.europa.eu/europe2020/index_en.htm. Accessed: 2013-08-30.

[83] European Commission. Rethinking education: Investing in skills for better socio-economic outcomes. http://www.cedefop.europa.eu/files/com669_en.pdf. Accessed: 2012-11-20.

[84] D.A. Schon. *Educating the Reflective Practitioner: Toward a New Design for Teaching and Learning in the Professions*. Wiley, 1987.

[85] K. Illeris. *Contemporary Theories of Learning: Learning Theorists ... In Their Own Words*. Taylor & Francis, 2009.

[86] J. Valtanen, E. Berki, E. Georgiadou, S. Hatzipanagos, M. Ross, I. Stamelos, and G. Staples. Features for suitable problems: IT professionals' and IT students' opinions. *International Journal of Human Capital and IT Professionals (IJHCITP)*, 3(3):27–41, 2012.

[87] K. Isitan, T. Nummenmaa, and E. Berki. Openness as a method for game evolution. In Katherine Blashki, editor, *In Proceedings of the IADIS International Conference of Game and Entertainment Technologies GET*, pages 100–104, 2011.

[88] K. Isitan. An approach to establish a software reliability model for different free and open source software development paradigms. Master's thesis, School of Information Sciences, University of Tampere, Tampere, Finland, 2011.

[89] E. Berki. Mentoring as a learning process - relationships and communication. *Investigations in University Teaching and Learning*, 3(1):42–49, Autumn 2005/Winter 2006 2006.

[90] M. Lohman. Factors influencing teachers' engagement in informal learning activities. *Journal of Workplace Learning*, 18(3):141–156, 2006.

[91] C Raptopoulou, T. Poranen, E. Berki, and I. Stamelos. Software project management antipatterns in students' projects. In J. Valtanen, E. Berki, M. Ruohonen, M. Ross, and G. Staples, editors, *In Seventeenth International Conference Proceedings on Software Process Improvement Research, Education and Training. INSPIRE XVII Education Matters*, pages 63–75, Tampere, Finland, 2012. University of Tampere Press.

[92] John Hattie and Helen Timperley. The power of feedback. *Review of Educational Research*, 77(1):81–112, 2007.

[93] Grant Wiggins. Seven keys to effective feedback. *Educational Leadership*, 70(1):10–16, September 2012.

[94] Margaret Price, Karen Handley, Jill Millar, and Berry O'Donovan. Feedback : all that effort, but what is the effect? *Assessment & Evaluation in Higher Education*, 35(3):277–289, 2010.

[95] J. Valtanen, E. Berki, P. Kampylis, and M. Theodorakopoulou. Manifold thinking and distributed problem based learning. is there

potential for ICT support? In P. Commers, P. Isaias, M Baptista-Nunes, and M. McPherson, editors, *In Proceedings of the IADIS International Multi Conference on Computer Science and Information Systems (MCCSIS'08)*, volume E-Learning Vol. 1, pages 145–152, Amsterdam, Netherlands, July 22-26 2008.

[96] Iakovos Ouranos, Kazuhiro Ogata, and Petros Stefaneas. *Formal Analysis of TESLA Protocol in the Timed OTS/CafeOBJ Method*, pages 126–142. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[97] Snapchat. `https://www.snapchat.com/`, Jul 2017. Accessed: 2017-07-02.

[98] Facebook messenger adds snapchat features. `http://www.bbc.com/news/av/technology-39222358/facebook-messenger-adds-snapchat-features`, Jul 2017. Accessed: 2017-07-02.

[99] Instagram adds snapchat-style stories feature. `https://www.cnet.com/news/instagram-adds-snapchat-style-stories-feature/`, Jul 2017. Accessed: 2017-07-02.

Κατάλογος δημοσιεύσεων του συγγραφέα

Κατά τη διάρκεια της εκπόνησης της διδακτορικής αυτής διατριβής πραγματοποιήθηκαν οι ακόλουθες δημοσιεύσεις σε διεθνή επιστημονικά περιοδικά:

1. Konstantinos Barlas, Eleni Berki, Petros Stefaneas, George Koletsos, "Towards Formal Open Standards: formalizing a standard's requirements", Innovations in Systems and Software Engineering, vol. 13, nr. 1 pp. 51-66, ISSN: 1614-5054, doi: 10.1007/s11334-016-0283-9. (2017)

2. Konstantinos Barlas, George Koletsos, Petros Stefaneas, Iakovos Ouranos, "Towards a correct Translation from ASN.1 into CafeOBJ", Special Issue on Innovations in Intelligent Systems and Applications of Reasoning-based Intelligent Systems (IJRIS, ISSN (Online): 1755-0564 - ISSN (Print): 1755-0556) (2010)

Ακόμα πραγματοποιήθηκαν οι εξής δημοσιεύσεις στα πρακτικά διεθνών επιστημονικών συνεδρίων:

1. Konstantinos Barlas, Eleni Berki, Iulia Adomnita, Thrushna Nalam, Golnaz S. Nejad, Jari Veijalainen, "Formal Specification of Open Standards and the Case of RSS v2.0", in Proceedings of the 18th Panhellenic Conference on Informatics (PCI '14), Athens, Greece, October 2-3, 2014. (2014)

2. Juri Valtanen, Eleni Berki, Konstantinos Barlas, Linfeng Li, Mirjan Merruko, "Problem-Focused Education and Feedback Mechanisms for Re-designing a Course on Open Source and Software Quality", in Proceedings of the INSPIRE (INternational conference in Software Process Improvement in Research Education and training) Conference 2013, London, U.K., September 5-6. (2013)

3. Katerina Ksystra, Nikolaos Triantafylloy, Konstantinos Barlas and Petros Stefaneas, "An Algebraic Specification of Social Networks", in Proceedings of the International Software Quality Management (SQM) and INSPIRE conferences. E. Berki, J. Valtanen, P. Nykänen, M. Ross, G. Staples and K. Systä, Eds, pp. 135-146, Tampere, Finland, August 21-23, 2012. (2012)

4. Konstantinos Barlas, G. Koletsos, P. Stefaneas, "Extending standards with formal methods: Open Document Architecture", International Symposium on Innovations in Intelligent Systems and

Applications (INISTA 2012), Trabzon, Turkey, July 2-4, 2012, IEEE Xplore (2009)

5. K. Barlas, G. Koletsos, P. Stefaneas and I. Ouranos, "Towards a correct translation from ASN.1 into CafeOBJ", International Symposium on Innovations in Intelligent Systems and Applications (INISTA 2009), Trabzon, Turkey, June 29-July 1, 2009, pp. 141-145 (2009)

6. Konstantinos Barlas, George Koletsos, Petros Stefaneas, Iakovos Ouranos, "From ASN.1 into CafeOBJ: Some first steps", 2009 Fourth South-European Workshop on formal methods (SEEFM 09), Thessaloniki, Greece, December 5 2009, pp. 66-72 (2009)

Ακόμα πραγματοποιήθηκαν οι εξής παρουσιάσεις σε μορφή poster:

1. Konstantinos Barlas, Eleni Berki, Petros Stefaneas and George Koletsos, "Towards Formal Open Standards: The Case of RSS v2.0", poster presentation at the Thales Workshop: Algebraic modeling of topological and computational structures and applications, National Technical University of Athens, July 1-3, 2015 (2015)