Ατομική Διπλωματική Εργασία

ΑΥΤΟΜΑΤΗ ΜΕΤΑΦΡΑΣΗ ΣΥΣΤΗΜΑΤΩΝ Π-ΛΟΓΙΣΜΟΥ ΣΕ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗ ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Ευανθία Τιγγίρη

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2016

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Αυτόματη μετάφραση συστημάτων π-λογισμού σε αντικειμενοστρεφή γλώσσα προγραμματισμού

Ευανθία Τιγγίρη

Επιβλέπουσα Καθηγήτρια
Άννα Φιλίππου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2016

Ευχαριστίες

Θα ήθελα να αφιερώσω αρχικά μία σελίδα σε κάποιους ανθρώπους των οποίων η βοήθεια και η στήριξη για την ολοκλήρωση της εργασίας μου ήταν πολύτιμη. Δεν θα μπορούσα να παραλείψω να τους εκφράσω το πόσο απαραίτητη ήταν η συμβολή τους.

Πρώτα απ' όλα, θέλω να ευχαριστήσω την επιβλέπουσα καθηγήτριά μου, Κα Άννα Φιλίππου, που μου εμπιστεύτηκε ένα τόσο ενδιαφέρον θέμα και με υποστήριξε καθ' όλη τη διάρκεια της εκπόνισης της εργασίας. Μέσω των γνώσεών της που με υπομονή μου μεταλαμπάδευσε, εξοικιώθηκα με το αντικείμενο της εργασίας και απέκτησα τα απαραίτητα εφόδια που χρειαζόμουν. Η βοήθεια και οι συμβουλές της, και όσον αφορά την εργασία αλλά και τις σπουδές μου γενικότερα ήταν καθοριστικές.

Έπειτα, είμαι ευγώμων στους γονείς μου, τους πιο σημαντικούς μου ανθρώπους, που στέκονται πάντα δίπλα μου σε όλες μου τις επιλογές και σε κάθε προσπάθειά μου. Έτσι, και κατά τη διάρκεια εκπόνισης της εργασίας μου, η υποστήριξή τους ήταν απόλυτη και με ενθάρρυνε συνεχώς.

Ευχαριστώ, Ευανθία Τιγγίρη

Περίληψη

Ζούμε στην εποχή που τα συστήματα κινητού υπολογισμού έχουν γίνει αναπόσπαστο κομμάτι στη ζωή μας. Επίσης, η Κοινωνία της Πληροφορίας είναι πλέον πραγματικότητα, λόγω της ραγδαίας εξέλιξης των τεχνολογιών και κοινωνικών δικτύων που ασχολούνται με την συλλογή, διαχείριση και ανταλλαγή πληροφοριών. Ως αποτέλεσμα, δημιουργήθηκε η ανάγκη για προστασία της ιδιωτικότητας των προσωπικών δεδομένων του κάθε χρήστη, την οποία τα συστήματα πληροφορικής δεν πρέπει να παραβιάζουν. Έτσι, συνδυάζοντας τα δύο πιο πάνω δεδομένα, πλέον είναι αδήριτη ανάγκη τα συστημάτα κινητού υπολογισμού, που έχουν καταλάβει μία τόσο σημαντική θέση στην καθημερινότητά μας, να τηρούν την ιδιωτικότητα και να προστατεύουν τα προσωπικά δεδομένα του χρήστη.

Η εργασία αυτή λοιπόν, ασχολείται με μία επέκταση της άλγεβρας διεργασιών π-λογισμός με ομάδες. Η άλγεβρα διεργασιών π-λογισμός είναι ιδανική για την μοντελοποίηση συστημάτων κινητού υπολογισμού και την αναπαράσταση των επικοινωνιών και της ανταλλαγής μηνυμάτων των συστημάτων. Από την άλλη, η επέκταση του π-λογισμού με ομάδες, δίνει την δυνατότητα να τεθούν περιορισμοί των δικαιωμάτων στα ευαίσθητα δεδομένα σε κάθε διεργασία του συστήματος, ανάλογα την ομάδα που ανήκει.

Πιο συγκεκριμένα, στο πλαίσιο της μελέτης, δημιουργήθηκε ένα εργαλείο το οποίο παίρνοντας ως είσοδο ένα μοντελοποιημένο σύστημα στην επέκταση του π-λογισμού που περιγράφηκε πιο πάνω, υλοποιεί το σύστημα αυτό στην γλώσσα αντικειμενοστρεφούς προγραμματισμού Java.

Το εργαλείο που υλοποιήθηκε στην ουσία συμβάλλει στην αυτοματοποίηση της υλοποίησης μοντελοποιημένων συστημάτων, με διεργασίες που επικοινωνούν μεταξύ τους και τηρούν κάποιους κανόνες ιδιωτικότητας. Για να γίνει αυτό, ο χρήστης αλληλεπιδρώντας με τη γραφική διεπιφάνεια, πρέπει να εισάγει τα απαραίτητα δεδομένα και XML αρχεία που αναπαριστούν συστήματα της επέκτασης του π-λογισμού. Έπειτα το σύστημα, μετά τους απαραίτητους ελέγχους, θα παράξει τις κλάσεις Java που αποτελούν την υλοποίηση του συστήματος.

Περιεχόμενα

Κεφάλαιο **1 Εισαγωγή** **1**

1.1 Κίνητρο 1

1.2 Στόχος 3

1.3 Προηγούμενες εργασίες 4

1.4 Μεθοδολογία 6

1.5 Οργάνωση Εργασίας 8

Κεφάλαιο **2 Άλγεβρες Διεργασιών και πολιτικές ιδιωτικότητας στον π-λογισμό** **9**

2.1 Άλγεβρες Διεργασιών 9

2.2 Ιστορία 10

2.3 Άλγεβρα διεργασιών: Λογισμός των επικοινωνούντων συστημάτων (CCS) 12

2.4 Άλγεβρα διεργασιών: π – Λογισμός (π-calculus) 14

2.5 Επέκταση άλγεβρας διεργασιών π-λογισμού με ομάδες 18

2.6 Συστημα Τύπων: 21

2.7 Typing Judgments 22

2.8 Τύπος συστήματος 23

2.9 Πολιτικές 26

Κεφάλαιο **3 Ανάλυση απαιτήσεων και προδιαγραφών** **28**

3.1 Ανάλυση απαιτήσεων και προδιαγραφών - Ορισμός 28

3.2 Σκοπός και προοπτική 29

3.3 Περιγραφή 30

3.4 Διεπιφάνειες 31

3.4.1 Διεπιφάνειες Συστημάτων (System Interfaces) 31

3.4.2 Διεπιφάνειες Χρήστη (User Interfaces) 31

3.4.3 Διεπιφάνεια Υλικού (Hardware Interface) 32

3.4.4 Διεπιφάνεια Λογισμικού (Software Interface) 32

3.5 Απαιτήσεις εγκατάστασης από τον χρήστη 35

3.6 Υποθέσεις και Εξαρτήσεις (Assumptions and Dependencies) 36

3.7 Λειτουργικές Απαιτήσεις (Functional Requirements) 37

3.7.1 Λειτουργία 1 37

3.7.2 Λειτουργία 2 38

3.7.3 Λειτουργία 3 39

3.7.4 Λειτουργία 4 40

3.7.5 Λειτουργία 5 41
3.8 Μη λειτουργικές απαιτήσεις 42
3.9 Σχεδιασμός Συστήματος με UML 46
3.9.1 Διάγραμμα Συστατικών 47
3.9.2 Διαγράμματα κλάσσεων 48
3.9.3 Διαγράμματα περιπτώσεων χρήσης 49
3.9.4 Διαγράμματα καταστάσεων 50
3.9.3 Διαγράμματα Ακολουθίας 52

Κεφάλαιο **4 Υλοποίηση Εργαλείου** **53**
4.1 Εισαγωγή 53
4.2 Κανόνες Μετάφρασης του λογισμού σε Java 54
4.2.1 Επίπεδο Συστήματος 56
4.2.2 Επίπεδο Διεργασίας 68
4.3 Ορθότητα Μεταφρασμένου Προγράμματος: 77
4.4 Εγχειρίδιο Χρήστη 85
4.4.1 Οδηγίες συγγραφής XML αρχείου 86
4.4.2 Οδηγίες χρήσης διεπιφάνειας αλληλεπίδρασης 90

Κεφάλαιο **5 Παραδείγματα εκτέλεσης του εργαλείου** **96**
5.1 Εισαγωγή 96
5.2 Παράδειγμα δειπνούντων φιλοσόφων 97
5.3 Παράδειγμα συνομιλίας σε chat room 101
5.4 Παράδειγμα ηλεκτρονικής τιμολόγησης κυκλοφορίας 109

Κεφάλαιο **6 Συμπεράσματα** **122**
6.1 Γενικά Συμπεράσματα 122
6.2 Προβλήματα και μειονεκτήματα 123
6.3 Μελλοντικές Εργασίες 124

Βιβλιογραφία **126**

Παράρτημα Α API κλάσεων του εργαλείου **128**

Κεφάλαιο 1

Εισαγωγή

1.1 Κίνητρο 1
1.2 Στόχος 3
1.3 Προηγούμενες εργασίες 4
1.4 Μεθοδολογία 6
1.5 Οργάνωση Εργασίας 8

1.1 Κίνητρο

Η τεχνολογική εξέλιξη οδήγησε στην εδραίωση των κινητών συστημάτων στη ζωή μας και στη χρήση τους όσο ποτέ άλλοτε. Επίσης, οδήγησε στην ύπαρξη μίας Εποχή της Πληροφορίας, με τη βοήθεια της τεχνολογίας μπορούμε να συσσωρεύσουμε τόση γνώση και πληροφορία, όση ποτέ άλλοτε. Αυτή η μεγάλη τεχνολογική πρόοδος και η αλλαγή των δεδομένων, οδήγησε στην απαίτηση των χρηστών από κάθε υπόβαθρο για προστασία των δεδομένων που η ελεύθερη ή μη-περιορισμένη πρόσβαση σε αυτά καθιστά παραβίαση του δικαιώματος της ιδιωτικότητάς τους.

Το κίνητρο λοιπόν για αυτήν την εργασία, ήταν η τήρηση αυτού του βασικού δικαιώματος όλων των κατηγοριών χρηστών και η εφαρμογή του σε πραγματικά συστήματα επικοινωνίας. Κύρια αιτία για την επιλογή της υλοποίησης της άλγεβρας διεργασιών π-λογισμός, είναι η ιδιότητά της να μπορεί να εκφράσει συστήματα διεργασιών των οποίων η δομή είναι μεταβαλλόμενη. Όχι μόνο επιτρέπει την αναπαράσταση επικοινωνίας μεταξύ συνδεδεμένων διεργασιών, αλλά μέσα από την ίδια την επικοινωνία, η σύνδεση μεταξύ τους μπορεί να μεταβάλλεται και διεργασίες που προηγουμένως δεν μπορούσαν να επικοινωνήσουν, μετά τις απαραίτητες ενέργειες επικοινωνίας, να έχουν την δυνατότητα να το πράξουν.

Η υλοποίηση όμως, δεν αφορά μόνο τον π-λογισμό, αλλά μία επέκταση του π-λογισμού με ομάδες[10], που υποστηρίζει την ύπαρξη κανόνων ιδιωτικότητας και το χαρακτηριστικό η κάθε διεργασία να έχει συγκεκριμένα δικαιώματα που μπορεί να ασκήσει σε κάθε ευαίσθητο δεδομένο. Τα δικαιώματα της κάθε διεργασίας, επίσης μπορεί να είναι μεταβαλλόμενα κατά τη ροή της επικοινωνίας.

Έναυσμα για την επιλογή της συγκεκριμένης επέκτασης του π-λογισμού είναι μία προσέγγιση της ανάλυσης και σχεδιασμού των συστημάτων που λαμβάνει υπόψην και ενσωματώνει την τήρηση κανόνων ιδιωτικότητας σε όλη τη διάρκεια της διαδικασίας σχεδίασης και υλοποίησης (*Privacy by Design*). Η Ιδιωτικότητας στο επίπεδο του Σχεδιασμού αναπτύχθηκε τη δεκαετία του ’90 από την Ann Cavoukian. Η έννοια αυτή μπορεί να επεκταθεί σε μία τριλογία εφαρμογών: στα συστήματα πληροφορικής, σε επιχειρηματικές πρακτικές και στο φυσικό σχεδιασμό και δικτυωμένη υποδομή [15]. Βασίζεται σε 7 θεμελιώδεις αρχές που πρέπει να ακολουθούνται κατά την υιοθέτησή της:

1. Πρόληψη, όχι αντίδραση
   Το σύστημα πρέπει να είναι σχεδιασμένο όχι για να αντιμετωπίζει τυχόν παραβιάσεις ιδιωτικότητας δεδομένων, αλλά για να τις αποτρέπει πριν γίνουν.
2. Ιδιωτικότητα ως προεπιλογή
   Μπορούμε όλοι να είμαστε σίγουροι για ένα πράγμα, για τους προεπιλεγμένους κανόνες! Το σύστημα πρέπει να εξασφαλίζει ότι τα προσωπικά δεδομένα προστατεύονται αυτόματα. Δεν απαιτείται καμία ενέργεια από την πλευρά του ατόμου για να διαφυλάξει την ιδιωτικότητά του, είναι ενσωματωμένη στο σύστημα, ως προεπιλογή.
3. Η ιδιωτικότητα ενσωματωμένη στο σχεδιασμό
   Η προστασία των ιδιωτικών δεδομένων δεν λαμβάνεται υπόψην μετά τον σχεδιασμό σαν επιπρόσθετη παράμετρος, αλλά είναι βασικό συστατικό στοιχείο του σχεδιασμού που δεν πρέπει να προστίθεται στο τέλος μειώνοντας τη λειτουργικότητα.
4. Πλήρης λειτουργικότητα
   Με την ενσωμάτωση της ιδιωτικότητας στο σχεδιασμό, αποφεύγονται αναγκαίοι συμβιβασμοί που θα έπρεπε να γίνουν αργότερα όσον αφορά τη λειτουργικότητα.

5. Πλήρης προστασία του κύκλου ζωής
   Η καθ' όλη τη διάρκεια τήρησης της ιδιωτικότητας, διασφαλίζει ότι κάθε δεδομένο είναι αρκετά προστατευμένο.
6. Ορατότητα και Διαφάνεια
   Διαβεβαιώνεται σε όλους τους ενδιαφερόμενους ότι όποια και αν είναι η επιχειρηματική πρακτική ή την τεχνολογία που εμπλέκονται, λειτουργούν σύμφωνα με τις δηλωμένες υποσχέσεις και τους στόχους και οι λειτουργίες παραμένουν ορατές και διαφανείς, τόσο στους χρήστες όσο και στους παρόχους.
7. Διατήρηση της ιδιωτικότητας με επίκεντρο το χρήστη

1.2 Στόχος

Η εργασία στοχεύει, λοιπόν, στην ορθή υλοποίηση μοντελοποιημένων συστημάτων που ήδη από τη φάση σχεδιασμού, είναι προσαρμοσμένα στο να τηρούν κάποιους κανόνες ιδιωτικότητας, με τη βοήθεια της επέκτασης του π-λογισμού με ομάδες. Δεν περιέχουν τρόπους αντιμετώπισης παραβίασης ιδιωτικότητας, αλλά είναι σχεδιασμένα με τέτοιο τρόπο ούτως ώστε η λειτουργία του συστήματος να γίνεται στα πλάισια των επιτρεπτών κανόνων και μόνο. Οπότε και τα συστήματα που θα υλοποιούνται αυτόματα μέσω του εργαλείου, πρέπει να υπακούν σε αυτούς τους κανόνες.

Στόχος είναι η παραγωγή κώδικα μοντελοποιημένων συστημάτων της επέκτασης του π-λογισμού με ομάδες [10] σε μία αντικειμενοστρεφή γλώσσα προγραμματισμού, τη Java. Αρχικός στόχος ήταν η μετάφραση της αυστηρής διατύπωσης συστημάτων σε π-λογισμό, σε με μία εξίσου αυστηρή δομή στη Java. Η μετάφραση σε Java πρέπει να ταυτίζεται σε απόλυτο βαθμό με την ακριβή και αυστηρή διατύπωση των μοντελοποιημένων συστημάτων. Δηλαδή, δεν πρέπει μόνο να επιβεβαιώνεται ότι η αντίστοιχη μετάφραση σε Java κάνει όλες τις ενέργειες που διατυπώνονται από την άλγεβρα διεργασιών, αλλά να επιβεβαιώνεται και το ότι δεν επιτρέπει σε καμία περίπτωση περισσότερες ενέργειες από αυτές.

Επιπρόσθετα, περαιτέρω στόχος, μετά την εύρεση μίας ακριβής αντιστοιχίας κώδικα – μοντέλου, ήταν και η αυτοματοποίηση της διαδικασίας μετάφρασης. Στόχος της εργασίας ήταν η δημιουργία ενός εργαλείου που δεδομένου ενός συστήματος π-

λογισμού με ομάδες σε XML, να μπορεί να παράγει τον ακριβή Java κώδικα αυτόματα, ακόμα και αν ο χρήστης δεν έχει προγραμματιστικές γνώσεις στη γλώσσα Java.

1.3 Προηγούμενες εργασίες

Η διπλωματική εργασία αυτή βασίζεται και χρησιμοποιεί δεδομένα που αναπτύχθηκαν σε προηγούμενες εργασίες.

Είναι συνέχεια της εργασίας [14] όπου προτείνεται μία διατύπωση της μοντελοποίησης συστημάτων π-λογισμού σε γλώσσα XML. Η συγκεκριμένη σύνταξη αποτελεί μία ακριβή μεταφορά της σύνταξης του π-λογισμού και δίνονται οδηγίες στον χρήστη για την διαδικασία ανάπτυξης ενός συστήματος π-λογισμού σε XML. Επίσης, στα πλαίσια της εργασίας αυτής αναπτύχθηκε ένα εργαλείο το οποίο δοσμένου ενός συστήματος π-λογισμού σε XML και μία πολιτικής ιδιωτικότητας, αποφασίζει κατά πόσον το σύστημα παραβιάζει ή όχι τη δοσμένη πολιτική. Εφόσον το σύστημα αυτής της εργασίας και της παρούσας, θα χρησιμοποιούνται ενιαία, για το εργαλείο μας θεωρείται δεδομένο ότι το δοσμένο σύστημα έχει ελεχθεί ότι τηρεί μία πολιτική ιδιωτικότητας που έχει ορίσει ο χρήστης προηγουμένως. Επομένως, η Java μετάφραση που θα εξάγει το συγκεκριμένο εργαλείο εφόσον θα είναι ακριβής μετάφραση του λογισμού, ως επακόλουθο θα τηρεί κι αυτή την απαιτούμενη πολιτική.

Προηγούμενες εργασίες που είναι άμεσα σχετιζόμενες με την παρούσα διπλωματική και που ήταν καθοριστικές για την επιτυχή ολοκλήρωσή της, αφορούν την υλοποίηση άλλων άλγεβρων διεργασιών. Πρώτα απ’ όλα πολύ σημαντική δουλειά είναι η βιβλιοθήκη JCSP. Είναι το βασικό συστατικό για την επιτυχή μετάφραση του π-λογισμού σε java. Η βιβλιοθήκη υλοποιεί τις βασικές αρχές της άλγεβρας CSP μαζί με μία ποικιλία πολύ χρήσιμων επεκτάσεων συνδυάζοντάς τις με ιδέες του Milner για τον π-λογισμό. Στόχος της συγκεκριμένης δουλειάς, ήταν να δοθεί ένα εργαλείο εύχρηστο για τους χρήστες με εύκολη και γρήγορη εξοικίωση με την άλγεβρα CSP διατηρώντας την ασφάλεια (αποκλείοντας κινδύνους όπως αδιέξοδα) αλλά και να κάνουν το μοντέλο CSP να προσαρμόζεται σε περισσότερες περιπτώσεις συστημάτων. Ανταλλάζει μηχανισμούς ασφαλείας του μεταγλωττιστή με κανόνες ελέγχου από τον

προγραμματιστή, έχοντας μεν πολύ μικρές απώλειες στην αποτελεσματική διαχείριση των διεργασιών, αλλά κερδίζοντας μία γλώσσα προγραμματισμού γενικού σκοπού.

Παρόμοια βιβλιοθήκη που προσφέρει ένα σύνολο κλάσεων για την υλοποίηση των κανόνων της CSP σε γλώσσα Java είναι η CTJ. Η CTJ περιλαμβάνει ένα πιο απλό και αξιόπιστο μοντέλο ταυτοχρονισμού απ' ό,τι το μοντέλο νημάτων της Java. Περιέχει ένα μικρό σύνολο σχεδιαστικών προτύπων που είναι αρκετό για ταυτόχρονο προγραμματισμό σε Java. Το σημαντικό πλεονέκτημα είναι ότι ο προγραμματιστής μπορεί να εφαρμόσει ένα σύνολο κανόνων για την αποφυγή προβλημάτων όπως αδιέξοδα ή εξάντληση πόρων μνήμης, από το στάδιο του σχεδιασμού και της υλοποίησης. Η χρήση της CTJ δεν απαιτεί μαθηματική γνώση της CSP. Αποτελεί μία γέφυρα ανάμεσα στη θεωρία και την πρακτική της άλγεβρας CSP [17].

Η έννοια των καναλιών CSP και στις δύο αυτές βιβλιοθήκες είναι ένας τρόπος να χρησιμοποιηθεί παραλληλοποίηση χωρίς να έρχεται ο προγραμματιστής αντιμέτωπος με προβλήματα των νημάτων της Java. Η διαχείριση του νήματος αφήνεται εξ' ολοκλήρου στο κανάλι CSP. Η CTJ περιέχει ένα πυρήνα πραγματικού χρόνου που χρονοδρομολογεί τα δικά του νήματα σε ένα καλά καθορισμένο και πραγμτικού χρόνου τρόπο. Δεν χρειάζεται λειτουργικό σύστημα και μπορεί να χρησιμοποιηθεί ακόμα και στο υλικό (hardware). Η JCSP παρέχει ένα μεγαλύτερο σύνολο επεκτάσεων και είναι ιδανικότερη για εκπαιδευτικούς σκοπούς. Και οι δύο βιβλιοθήκες χρησιμοποιούνται εκτεταμένα και αναμένεται περαιτέρω αναβάθμισή τους [18].

Τέλος, η άλγεβρα διεργασιών π-λογισμός, έχει υλοποιηθεί από τον Liwu Li, το 2005, χωρίς δυστυχώς να μπορέσουμε να βρούμε κάπου το εργαλείο-μεταγλωττιστή που υλοποιήσε. Το άρθρο του όμως [19] ήταν πολύ σημαντικό στο να μου δώσει κατευθυντήριες γραμμές στην τεχνική μετάφρασης που θα ήταν πιο καλό να ακολουθήσω. Δεν χρησιμοποιεί κάποια άλλη βιβλιοθήκη σε αντίθεση με την παρούσα εργασία που χρησιμοποιεί την JCSP. Αντίθετα υλοποιεί τις παράλληλες διεργασίες με τη χρήση του μοντέλου νημάτων της Java. Σημαντικό στοιχείο της υλοποίησής του είναι ότι μεταφράζει κάθε κανόνα του π-λογισμού σε μία κλάση της Java που την υλοποιεί. Επομένως, στο τέλος η μετάφρασή του ακολουθεί μία ιεραρχική δομή, με αντικείμενα κλάσεων που στο εσωτερικό τους περιέχουν ορίσματα που είναι

αντικείμενα άλλων κλάσεων. Ένα απλό παράδειγμα, είναι η κλάση της παράλληλης διεργασίας που αποτελείται από αντικείμενα κλάσεων που αφορούν τις επιμέρους ενέργειες της διεργασίας. Αυτή η τεχνική, χρησιμοποιήθηκε αρκετά και στην παρούσα εργασία και ήταν χρήσιμη στο θέμα της τήρησης των κανόνων ιδιωτικότητας. Υπάρχουν βέβαια διαφορές στον τρόπο αντιμετώπισης κάποιων κανόνων του π-λογισμού. Σε αυτή την υλοποίηση υπάρχει ένα ενιαίο πρόγραμμα που τρέχει ενεργοποιώντας παράλληλα νήματα. Στη δική μας υλοποίηση, κάθε διεργασία λειτουργεί ανεξάρτητη, με τη δική της τοπική μνήμη. Αυτό μας βοήθησε στην αποφυγή χρήσης της τεχνικής κλειδώματος μεταβλητής και στην διαδικασία της μετονομασίας που θα εξηγηθεί στο κεφάλαιο 2 όπου αναλύεται ο π-λογισμός.

1.4 Μεθοδολογία

Για την ολοκλήρωση της εργασίας ακολούθησα διάφορα στάδια που αφορούσαν την εξοικίωση τόσο με τις θεωρητικές έννοιες όσο και με τα εργαλεία που θα ήταν καταλληλότερα να χρησιμοποιήσω. Αρχικά προσπάθησα να κατανοήσω τις βασικές έννοιες των άλγεβρων διεργασίας. Μελέτησα το υλικό που μου πρότεινε (άρθρα και βιβλίο) η επιβλέπουσα καθηγήτριά μου για να κατανοήσω τη σημασία των άλγεβρων διεργασιών και ειδικότερα της CCS. Λύνοντας ασκήσεις της CCS και του π-λογισμού εξοικιώθηκα με την επικοινωνία παράλληλων διεργασιών αλλά και με την ιδιαιτερότητα του π-λογισμού να χρησιμοποιεί τα κανάλια και ως μέσο επικοινωνίας αλλά και ως αντικείμενα αποστολής.

Ακολούθως, έπρεπε να αποκτήσω πλήρη κατανόηση της επέκτασης που θα υλοποιούσα σε Java. Διάβασα το άρθρο [10] της καθηγήτριάς μου που εξηγεί εκτεταμένα την εφαρμογή της ιδιωτικότητας και των ομάδων στον π-λογισμό και τους απαραίτητους κανόνας που θα πρέπει ένα σύστημα της επέκτασης αυτής να τηρεί. Σε αυτό το σημείο είχα καταλάβει τι ακριβώς έπρεπε να υλοποιεί το εργαλείο μου.

Τέλος λαμβάνοντας υπόψην όλους τους παράγοντες και όλες τις απαιτήσεις του συστήματος ούτως ώστε να ακολουθεί όλα όσα έμαθα στη θεωρητική μελέτη, έπρεπε να βρω τα κατάλληλα εργαλεία που θα με βοηθούσαν στην υλοποίηση του εργαλείου. Αρχικά ακολουθώντας τη μεθοδολογία της προηγούμενης εργασίας, χρησιμοποίησα

τον Java Dom Parser (εργαλείο για τη γλώσσα Java που δημιουργεί για XML περιεχόμενο, μία αντίστοιχη δενδρική δομή) για την ανάγνωση του XML αρχείου.

Σε επόμενο στάδιο, έπρεπε να αποφασίσω την κατάλληλη αναπαράσταση της κάθε ενέργειας του π-λογισμού σε γλώσσα Java. Μελέτησα διάφορες επιλογές όπως για παράδειγμα η χρήση συγχρονισμένων νημάτων (threads) της Java. Τελικά, μελετώντας προηγούμενες εργασίες κατέληξα στη χρήση της βιβλιοθήκης JCSP κι έπειτα καθόρισα τις βασικές αντιστοιχίες κώδικα Java – μοντέλου π-λογισμός, ούτως ώστε κάθε ενέργεια της άλγεβρας να μεταφράζεται σε μία αντίστοιχα σταθερή και αυστηρή ομάδα εντολών.

Έπρεπε να δημιουργήσω το εργαλείο που αναλύοντας το σύστημα δοσμένο σε XML, δημιουργεί αυτόματα τις απαραίτητες κλάσεις Java με τον αναγκαίο αντίστοιχο κώδικα. Δεν θα μπορούσε βέβαια από κάποιο σύστημα να λείπει το στάδιο του ελέγχου, το οποίο είναι και το τελικό. Με διάφορα παραδείγματα κλασσικών προβλημάτων παράλληλης επικοινωνίας διεργασιών, έπρεπε να ελέγξω την ορθή λειτουργία του εργαλείου μου. Για να γίνει αυτό, τα μοντελοποίησα στην άλγεβρα π-λογισμός και μετά σε αντίστοιχη σύνταξη XML. Εισάγοντάς τα στο σύστημα έβλεπα αν το εργαλείο λειτουργούσε χωρίς σφάλματα και αν η παραγόμενη υλοποίηση αντικατόπτριζε την ακριβή λειτουργία του μοντελοποιημένου συστήματος σε π-λογισμό. Αυτή η διαδικασία με βοήθησε στην εύρεση λαθών αλλά και στο να βελτιώσω και την απόδοση του εργαλείου.

Ως επιπρόσθετο κομμάτι, θέλησα να ασχοληθώ και με τη δημιουργία ενός απλού γραφικού περιβάλλοντος χρήστη, ούτως ώστε η χρήση του εργαλείου να είναι προσιτή και ευχάριστη.

1.5 Οργάνωση Εργασίας

Η εργασία αποτελείται από 6 κεφάλαια και ένα παράρτημα.
Στο Κεφάλαιο 2 παρουσιάζονται οι άλγεβρες διεργασιών γενικότερα αλλά και πιο συγκεκριμένα η CCS, ο π-λογισμός. Εν συνεχεία, παρουσιάζεται ο η επέκταση του π-λογισμού με ομάδες που είναι ο λογισμός στον οποίο βασίζεται η ανάπτυξη του εργαλείου.

Στο κεφάλαιο 3, παρουσιάζεται η τεκμηρίωση της υλοποίησης του εργαλείου, ακολουθώντας τις βασικές αρχές της Τεχνολογίας Λογισμικού. Αναφέρονται οι απαιτήσεις του λογισμικού (λειτουργικές και μη-λειτουργικές) όπως και οι προαπαιτήσεις. Επιχειρείται επίσης να γίνει ένας σχεδιασμός του συστήματος με διαφράματα UML (ενοποιημένη γλώσσα μοντελοποίησης) ούτως ώστε να υπάρχει επαρκής τεκμηρίωση σε περίπτωση επέκτασης του εργαλείου από άλλους προγραματιστές.

Στο κεφάλαιο 4, παρουσιάζεται η τεχνική μετάφρασης και προτείνεται μία αυστηρή διατύπωση του κάθε κανόνα του λογισμού μας σε κώδικα. Τεκμηριώνονται κάθε φορά οι λόγοι που ακολουθήθηκε η κάθε μέθοδος, δίνοντας παραδείγματα. Ακολουθεί μία απόδειξη της ορθότητας της μετάφρασης, δείχνοντας ότι δεν παραβιάζει κανέναν κανόνα ιδιωτικότητας και κανει ακριβώς τις αντίστοιχες ενέργειες με αυτές που αναπαριστά το μοντέλο. Επίσης, δίνεται ο οδηγός χρήσης, τόσο για τη δημιουργία ορθών αρχείων εισόδου, όσο και για τη σωστή χρήση της διεπιφάνειας αλληλεπίδρασης.

Στο κεφάλαιο 5 τίθενται δύο προβλήματα εφαρμογής του λογισμού αλλά και του εργαλείου. Επεξηγούνται και καταλήγουμε στο αποτέλεσμα που θα έχει το εργαλείο γι' αυτά.

Στο κεφάλαιο 6, παρουσιάζονται κάποια βασικά συμπεράσματα που προέκυψαν από την εργασία αλλά και τα προβλήματα που δημιουργήθηκαν κατά τη διάρκεια της πορείας ολοκλήρωσής της. Επιπρόσθετα, προτείνονται σχετικές με την εργασία ιδέες που θα μπορούσαν να είναι μέρος μελλοντικών εργασιών.

Στο παράρτημα Α, δίνεται το API (Application Programming Interface) των κλάσεων και συναρτήσεων που δημιούργησα για την ολοκλήρωση κάθε μέρους και λειτουργίας του συστήματος.

Κεφάλαιο 2

Άλγεβρες διεργασιών και πολιτικές ιδιωτικότητας στον π-λογισμό

2.1 Άλγεβρες Διεργασιών 9
2.2 Ιστορία 10
2.3 Άλγεβρα διεργασιών: Λογισμός των επικοινωνούντων συστημάτων (CCS) 12
2.4 Άλγεβρα διεργασιών: π – Λογισμός (π-calculus) 14
2.5 Επέκταση άλγεβρας διεργασιών π-λογισμού με ομάδες 18
2.6 Συστημα Τύπων: 21
2.7 Typing Judgments 22
2.8 Τύπος συστήματος 23
2.9 Πολιτικές 26

2.1 Άλγεβρες Διεργασιών

Άλγεβρα διεργασιών είναι μία μέθοδος αναπαράστασης και ανάγνωσης κατανεμημένων ή παράλληλων συστημάτων με αλγεβρικό τρόπο. Αρχικά, ας δούμε τι ακριβώς εννοούμε με τη λέξη «διεργασίες». Με τον όρο «διεργασίες» λοιπόν, αναφερόμαστε στην *συμπεριφορά* ενός *συστήματος*. Σύστημα είναι οτιδήποτε παρουσιάζει μία συγκεκριμένη συμπεριφορά, όπως είναι η εκτέλεση ενός λογισμικού συστήματος, οι ενέργειες μίας μηχανής, η εξέλιξη που μπορεί να παρουσιάσει ένα σύστημα, η σειρά που αυτά θα παρουσιαστούν όπως και άλλες παραμέτροι της εκτέλεσης όπως είναι για παράδειγμα ο χρόνος. Βέβαια, κάθε φορά περιγράφουμε μία αφαιρετική προσέγγιση της *πραγματικής* συμπεριφοράς αφού συνήθως εστιάζουμε σε ορισμένες πτυχές της εκτέλεσης μη λαμβάνοντας υπόψην τις υπόλοιπες. Με τον όρο «άλγεβρα» δηλώνεται ότι η περιγραφή της *συμπεριφοράς* του συστήματος γίνεται με αλβεβρική προσέγγιση, χρησιμοποιώντας μεθόδους και τεχνικές της καθολικής άλγεβρας. Μία άλγεβρα διεργασιών μπορεί να οριστεί ως οποιαδήποτε μαθηματική δομή που ικανοποιεί τα αξιώματα που δίνονται για τους *βασικούς τελεστές* (basic operators). Με τη χρήση των

αξιωμάτων μπορούμε να εκτελέσουμε *υπολογισμούς* (calculations) με διεργασιες. Φυσικά, μία άλγεβρα διεργασιών μπορεί να ξεπερνά τα όρια της καθολικής άλγεβρας όπως συμβαίνει για παράδειγμα με τη δέσμευση μεταβλητών. Γενικότερα, στην επιστήμη της πληροφορικής, οι άλγεβρες διεργασιών είναι μία οικογένεια διαφόρων προσεγγίσεων για την τυπική μοντελοποίηση παράλληλων συστημάτων [1]. Αποτελούν ένα εργαλείο για την υψηλού-επιπέδου περιγραφή των αλληλεπιδράεων, επικοινωνιών και συγχρονισμών μέσα σε μία συλλογή ανεξάρτητων αντιπροσώπων (agents) και διεργασιών (process). Κύρια παραδείγματα άλγεβρας διεργασιών είναι η CCS[2], CSP[2] και ACP[2] και με πιο πρόσφατο τον π-λογισμό (π - calculus)[10]. Σήμερα υπάρχουν πολλές διαφορετικές άλγεβρες διεργασιών. Παρόλα αυτά, υπάρχουν κάποια χαρακτηριστικά που είναι κοινά σε όλες:

1. Αναπαράσταση αλληλεπιδράσεων μέσω ανεξάρτητων διεργασιών ως επικοινωνία περάσματος μηνυμάτων (message passing) και όχι ως τροποποίηση κοινών μεταβλητών.
2. Περιγραφή διεργασιών και συστημάτων χρησιμοποιώντας ένα σύνολο θεμελιωδών στοιχείων και τελεστών για το συνδυασμό αυτών των θεμελιωδών στοιχείων.
3. Καθορισμός αλγεβρικών αξιωμάτων για τους τελεστές που επιτρέπει τον χειρισμό των εκφράσεων διεργασιών μέσω αλγεβρικων αντικατασεων.

2.2 Ιστορία

Σε αυτό το σημείο, θα αναφερθεί η ιστορία της άλγεβρας διεργασιών ξεκινώντας από τις αρχές της δεκαετίας του ’70. Τότε το μόνο στοιχείο που υπήρχε σε σχέση με τη θεωρία παράλληλων συστημάτων ήταν η θεωρία των δικτύων Petri (Petri nets), που σχεδιάστηκε από τον Petri το 1962 στα πλαίσια της διατριβής του [3]. Έπειτα, το 1970 μπορούμε να πούμε ότι υπήρχαν 3 κύρια είδη προσεγγίσεων που ξεχώρισαν για την παροχή μίας αυστηρά μαθηματικής κατανόησης της σημασιολογίας:

1. Λειτουργική σημασιολογία (operational): Ένα πρόγραμμα υπολογιστή διαμορφώνεται ως μία εκτέλεση μιας αφηρημένης μηχανής. Μία κατάσταση (state) μίας τέτοιας μηχανής είναι η αποτίμηση μεταβλητών και η μετάβαση μεταξύ των καταστάσεων είναι μία στοιχειώδης εντολή προγράμματος.
2. Δηλωτική σημασιολογία (denotational): Είναι πιο αφαιρετική από τη Λειτουργική σημασιολογία όπου τα υπολογιστικά προγράμματα συνήθως μοντελοποιούνται από μία

συνάρτηση μετατροπής της εισόδου σε έξοδο.
3. Αξιωματική σημασιολογία (Axiomatic): Σε αυτή τη σημασιολογία δίνεται έμφαση στις μεθόδους απόδειξης ορθών προγραμμάτων. Οι κεντρικές έννοιες είναι οι ισχυρισμοί του προγράμματος που αποτελούνται από το τρίπτυχο: προϋποθέσεις/προσυνθήκες (preconditions), δήλωση του προγράμματος και μετασυνθήκες και σταθερές.

Στη συνέχεια δημιουργήθηκε το ερώτημα πώς μπορεί να δοθεί σημασιολογία σε προγράμματα που περιέχουν έναν παράλληλο τελεστή. Διαπιστώθηκε ότι αυτό θα ήταν δύσκολο χρησιμοποιώντας τις μεθόδους της λειτουργικής, δηλωτικής ή αξιωματικής σημασιολογίας, παρόλο που έγιναν κάποιες προσπάθειες. Έτσι στις αρχές της δεκαετίας του 80 μπορούμε να θεωρήσουμε ότι οι άλγεβρες διεργασιών καθιερώθηκαν ως ξεχωριστή ερευνητική περιοχή. Ένας από τους ανθρώπους που ερεύνησε τις σημασιολογίες των παράλληλων προγραμμάτων στις αρχές της δεκαετίας του 70 ήταν ο Hans Bekic. Δούλεψε στο εργαστήριο της IBM στη Βιέννη που ήταν γνωστό για τη δουλειά του στον ορισμό και τη σημασιολογία των γλωσσών προγραμματισμού. Ο ίδιος εργάστηκε στη δηλωτική σημασιολογία της ALGOL και της PL/I και στα πλαίσια της δουλειάς του δημιουργήθηκε ο προβληματισμός στο πώς θα μπορούσε να δοθεί δηλωτική σημασιολογία για τις παράλληλες συνθέσεις. Αντιμετώπισε το πρόβλημα αυτό στο [4] όπου αναφέρει συγκεκριμμένα: «*Το πλάνο μας είναι να αναπτύξουμε μία άλγεβρα διεργασιών που θα μπορούσε να θεωρηθεί μία υψηλού-επιπέδου προσέγγιση: ενδιαφερόμαστε στο πώς μπορούμε να συνθέσουμε πολύπλοκες διεργασίες χρησιμοποιώντας πιο απλές»* [5].

Μία από τις σημαντικότερες προσωπικότητες στην ιστορία της άλγεβρας διεργασιών είναι χωρίς αμφιβολία ο Robert Milner που ανέπτυξε το Λογισμό των Επικοινωνούντων Συστημάτων (CCS). Στην CCS μοντελοποιούνται αδιαίρετες επικοινωνίες μεταξύ ακριβώς 2 συμμετεχόντων. Η τυπική γλώσσα περιλαμβάνει βασικά στοιχεία για την περιγραφή της παράλληλης σύνθεσης (||), της επιλογής μεταξύ ενεργειών (+) και του περιορισμού εμβέλειας(\ ). Η CCS χρησιμοποείται για την εκτίμηση ορθότητας ιδιοτήτων ενός συστήματος (πχ αδιέξοδα) [6]. Η CCS θα αναλυθεί εκτενέστερα σε επόμενο υποκεφάλαιο.

Ο Tony Hoare ήταν ακόμα ένας πολύ σημαντικός άνθρωπος που συνέβαλε στην ανάπτυξη των άλγεβρων διεργασιών. Έδωσε την αρχική περιγραφή της τυπικής γλώσσας CSP (Επικοινωνούσες Ακολουθιακές Διεργασίες). Η αρχική αυτή περιγραφή ήταν πιο πολύ μία ταυτόχρονή γλώσσα προγραμματισμού και όχι λογισμού διεργασιών, με διαφορετική σύνταξη σε σχέση με τις μεταγενέστερες εκδόσεις της CSP, χωρίς μαθηματικά ορισμένη σημασιολογία και χωρίς να μπορεί να αναπαραστήσει μη-ντετερμινισμό χωρίς άνω όριο. Έπειτα, μετά από επιρροές από τη δουλειά του Milner στο CCS, η ανάπτυξη της CSP στράφηκε στις άλγεβρες διεργασιών. Επιτρέπει λοιπόν την περιγραφή συστημάτων που αποτελούνται από διεργασίες που λειτουργούν ανεξάρτητα και αλληλεπιδρούν μόνο μέσω περάσματος μηνυμάτων. Πλέον, οι διεργασίες της CSP δεν είναι αποκλειστικά ακολουθιακές, αλλά μπορεί να είναι και μία παράλληλη σύνθεση από πιο βασικές διεργασίες. Ο τρόπος επικοινωνίας μεταξύ των διεργασιών και το περιβάλλον τους περιγράφεται μέσω των τελεστών[7]. Θεώρησα σημαντική την αφαιρετική περιγραφή της CSP καθώς όπως θα εξηγηθει και παρακάτω εκτενέστερα, για τη μετάφραση του π-λογισμού στη γλώσσα προγραμματισμού Java χρησιμοποιήθηκε η βιβλιοθήκη JCSP η οποία αποτελεί μία υλοποίηση της CSP.

Μία πιο πρόσφατη άλγεβρα διεργασιών είναι ο π-λογισμός που εστιάζει περισσότερο σε δίκτυα διεργασιών όπου οι διεργασίες είναι κινητές και η διαμόρφωση των συνδέσμων επικοινωνίας (κανάλια) είναι δυναμική. Η «καινοτομία» λοιπόν του π-λογισμού είναι ότι λόγω της διπλής ιδιότητας των καναλιών που μπορούν να χρησιμοποιηθούν και ως συνδέσεις μεταφοράς αντικειμένων αλλά και ως αντικείμενα που μεταφέρονται μέσω άλλων καναλιών, μπορεί να μοντελοποιήσει κινητό υπολογισμό (mobility) [5].

2.3 Άλγεβρα διεργασιών: Λογισμός των επικοινωνούντων συστημάτων (CCS)

Όπως αναφέρθηκε, αναπτύχθηκε από τον Robin Milner τέλη του 1970 – αρχές του 1980. Οι διεργασίες κτίζονται από ένα σύνολο *ατομικών ενεργειών* (atomic actions) με τη χρήση *τελεστών* σύνθεσης διεργασιών. Οι ατομικές ενέργειες αυτές μπορεί να είναι ενέργειες εισόδου, εξόδου ή εσωτερικές ενέργειες και μοντελοποιούν αδιαίρετες επικοινωνίες μεταξύ ακριβώς δύο συμμετεχόντων (δυαδικός συγχρονισμός) [6].

«Σε άλγεβρες διεργασιών όπως η CCS συστήματα επικοινωνούν με το περιβάλλον τους (και μεταξύ τους) μέσω συγχρονισμών που συμβαίνουν σε κανάλια. Αν μια διεργασία είναι διαθέσιμη για είσοδο και κάποια άλλη για έξοδο στο ίδιο κανάλι, τότε ο συγχρονισμός λαμβάνει χώρο και οι διεργασίες προχωρούν με την εκτέλεσή τους. Οι εξωτερικές ενέργειες που μπορεί να εκτελέσει ένα σύστημα μπορούν να θεωρηθούν ως η συμπεριφορά/διεπιφάνειά του» [8].

Έστω ότι έχουμε το σύνολο από ετικέτες **Λ** (ονόματα καναλιών) και η ετικέτα **τ**. Η ενέργεια στη CCS μπορεί να είναι ένα από τα ακόλουθα:

- Ενέργεια εισόδου στο κανάλι λ ∈ Λ: λ
- Ενέργεια εξόδου στο κανάλι λ ∈ Λ: $\overline{\lambda}$
- Εσωτερική ενέργεια: τ

Το σύνολο των διεργασιών του CCS ορίζεται από την εξής γραμματική BNF(*)[1]

P **::=** ∅ | α.P1 | P1+P2 | P1|P2 | P1[b/a] | P1\a | C

∅: κενή διεργασία – που δεν ανταποκρίνεται (τερματισμός ή αδιέξοδο)

α.P1: ενέργεια – αυτή η διεργασία μπορεί να εκτελέσει την ενέργεια α και στη συνέχεια συνεχίζει σαν διεργασία P1.

P1+P2: επιλογή – Διεργασία η οποία προσφέρει την επιλογή ανάμεσα στις P1 και P2

P1|P2: παράλληλη σύνθεση – Διεργασία στην οποία τρέχουν παράλληλα οι P1 και P2. Οι υποδιεργασίες αυτές μπορούν να εκτελούν ανεξάρτητα την εργασία τους ή και να επικοινωνούν μεταξύ τους στα κανάλια που τις συνδέουν.

P1[b/a]: μετονομασία – Η διεργασία P1 με όλες τις ενέργειες με το όνομα *a* να μετονομάζονται σε *b*.

C: κλήση

[1] (*)BNF (Κανονική μορφή του Μπάκους Backus Normal Form) είναι μια τεχνική συμβολισμού για γραμματικές χωρίς συμφραζόμενα (context-free grammars),που συχνά χρησιμοποιείται για να περιγράψει τη σύνταξη μιας γλώσσας της πληροφορικής, όπως οι γλώσσες προγραμματισμού υπολογιστών, οι τύποι εγγράφων (document formats), τα σύνολα εντολών (instruction sets) και τα πρωτόκολλα επικοινωνιών. Εφαρμόζεται όπου χρειάζονται ακριβείς περιγραφές γλωσσών, για παράδειγμα σε επίσημους ορισμούς γλωσσών, σε εγχειρίδια, ή σε βιβλία για θεωρία γλωσσών προγραμματισμού [9].

Ο τελεστής '.' έχει μεγαλύτερη προτεραιότητα από τον '+', ενώ η κατάληξη '.0' και όροι όπως '… | 0' συχνά παραλείπονται [8].

2.4 Άλγεβρα διεργασιών: π – Λογισμός (π-calculus)

Η άλγεβρα διεργασιών π-calculus αναπτύχθηκε από τους Robin Milner, Joachim Parrow και David Walker το 1992. Μπορεί να θεωρηθεί μία συνέχεια της δουλειάς του Milner στην άλγεβρα διεργασιών CCS. Ο π-λογισμός είναι ένα μαθηματικό μοντέλο διεργασιών που καθώς αυτές αλληλεπιδρούν (ανταλλάζοντας μηνύματα μέσω καναλιών) οι διασυνδέσεις τους μπορούν να μεταβάλλονται.
Κύριο χαρακτηριστικό του π-λογισμού είναι η δυνατότητα μεταφοράς ενός καναλιού επικοινωνίας ως αντικείμενο μεταξύ δύο διεργασιών, μέσω ενός άλλου καναλιού. Η διεργασία – παραλήπτης μπορεί έπειτα να χρησιμοποιήσει το κανάλι για περαιτέρω αλληλεπίδραση με άλλα μέρη του συστήματος/περιβάλλοντος. Αυτό κάνει τον π-λογισμό κατάλληλο για τη μοντελοποίηση συστημάτων κινητού υπολογισμού όπου οι προσβάσιμοι πόροι μεταβάλλονται. [10]

Γενικότερα, η σύνταξη του π-λογισμού μας επιτρέπει να αναπαραστήσουμε διεργασίες, παράλληλη σύνθεση διεργασιών, συγχρονισμένη επικοινωνία διεργασιών μέσω καναλιών, δημιουργία καινούργιων καναλιών, αντιγραφή διεργασιών και μη-ντετερμινισμό.

<u>Η σύνταξή του λοιπόν ορίζεται από την εξής γραμματική BNF:</u>
Έστω N ένα σύνολο αντικειμένων που ονομάζονται *ονόματα* (names) και *a,b,…, z* ανήκουν στο N.

Προθέματα (prefixes):

α ::= $\overline{\alpha}x$	Έξοδος
$\alpha(x)$	Είσοδος
τ	Εσωτερική ενέργεια

Όροι (agents):

P ::= 0	Κενή διεργασία

α.P	Πρόθεμα
P+P	Επιλογή
P | P	Παράλληλη Σύνθεση
If (x=y) then P	Ισότητα (match)
If (x≠y) then P	Ανισότητα (mismatch)
(νx)P	Περιορισμός (restriction)
!P	Αντιγραφή (replication)
A(y1, …. ,yn)	Αναγνωριστικό (identifier)

A(y1, …. ,yn) $\stackrel{\text{def}}{=}$ P (όπου $i \neq j \rightarrow x_i \neq x_j$)

Βασικοί Ορισμοί [10]:

1. Κενή διεργασία **0**: δεν μπορεί να παρουσιάσει καμία ενέργεια (τερματισμός/αδιέξοδο)

2. Πρόθεμα εξόδου $\overline{\alpha}x.P$: Το όνομα x αποστέλλεται μέσω του καναλιού x κι έπειτα η διεργασία συνεχίζει ως P. Επομένως το α μπορεί να θεωρηθεί το κανάλι εξόδου και το x το δεδομένο που στέλνεται μέσω αυτού του καναλιού.

3. Πρόθεμα εισόδου $\alpha(x).P$: Ένα όνομα λαμβάνεται μέσω ενός καναλιού α και το x είναι ο χωροθέτης (placeholder) για το όνομα που λαμβάνεται. Η διεργασία συνεχίζει ως P με το x να έχει αντικατασταθεί από το όνομα που λήφθηκε. Έπειτα το x δεσμεύεται.

4. Πρόθεμα εσωτερικής ενέργειας $\tau.P$: Μία διεργασία μπορεί να συνεχίζει ως P χωρίς κάποια αλληλεπίδραση με το περιβάλλον/διεπιφάνεια.

5. Επιλογή P + Q : Μία διεργασία μπορεί να συνεχίσει ως P ή ως Q.

6. Παράλληλη σύνθεση P | Q : παρουσιάζει τη συμπεριφορά του συστήματος με τις διεργασίες P και Q να εκτελούνται παράλληλα και είτε ανεξάρτητα είτε επικοινωνώντας η μία με την άλλη.

7. Ισότητα If (x=y) then P : Αυτή η διεργασία θα συνεχίσει ως P μόνο αν το x και το y ισούνται.

8. Ανισότητα If (x≠y) then P : Αυτή η διεργασία θα συνεχίσει ως P μόνο αν το x και το y δεν ισούνται.

9. Περιορισμός $(\boldsymbol{\nu x})$P : Η διεργασία συμπεριφέρεται ως P όμως το όνομα x είναι δεσμευμένο. Δεν είναι άμεσα προσβάσιμο από το περιβάλλον του P, εκτός κι αν το P το αποστείλει μέσω μίας επικοινωνίας. Μπορεί όμως να χρησιμοποιηθεί για την επικοινωνία εντός του P.

10. Αντιγραφή !P : Η διεργασία δημιουργεί αντίγραφο του εαυτού της

Μέσω του περιορισμού και του προθέματος εισόδου κάποια ονόματα δεσμεύονται. Για να γίνει αυτό πιο ξεκάθαρο, ακολουθεί πιο κάτω ένας πίνακας με τους όρους της σύνταξης του π-λογισμού και με τα ονόματα που παραμένουν ελεύθερα και αυτά που δεσμεύονται [11]:

Όροι	**Ελεύθερα κανάλια**	**Δεσμευμένα κανάλια**
0	Κανένα	Κανένα
$\overline{\boldsymbol{\alpha}}\boldsymbol{x}.\boldsymbol{P}$	Όλα τα ελεύθερα κανάλια του P και τα *α* και *x*	Όλα τα δεσμευμένα κανάλια του P
$\boldsymbol{\alpha}(\boldsymbol{x}).\boldsymbol{P}$	Το *α* και όλα τα ελεύθερα κανάλια του P εκτός το *x*	Όλα τα δεσμευμένα κανάλια του P και το *x*
P | Q	Όλα τα ελεύθερα κανάλια του P και Q	Όλα τα δεσμευμένα κανάλια του P και Q
$(\boldsymbol{\nu x})$**P**	Όλα τα ελεύθερα κανάλια του P εκτός το *x*	Όλα τα δεσμευμένα κανάλια του P και το *x*
!P	Όλα τα ελεύθερα κανάλια του P	Όλα τα δεσμευμένα κανάλια του P

Πίνακας 1

Παράδειγμα εφαρμογής του π-λογισμού[12]:

Σε αυτό το σημείο θεωρώ σκόπιμο να δοθεί ένα απλό και κατανοητό παράδειγμα εφαρμογής του π-λογισμού, ούτως ώστε να γίνει πιο αντιληπτή η χρησιμότητά του αλλά και η σημαντικότητα του εργαλείου που αναπτύχθηκε στα πλαίσια αυτής της διπλωματικής.

Υποθέτουμε ότι θέλουμε να μοντελοποιήσουμε μία απομακρυσμένη επικοινωνία μεταξύ ενός εξυπηρετητή και ενός πελάτη. Θεωρούμε ότι η ακόλουθη συνάρτηση *increase* τρέχει στον εξυπηρετητή:

int *increase* (int x) { return x+1; } – Η συνάρτηση δέχεται έναν ακέραιο και επιστρέφει τον αμέσως μεγαλύτερο.

Η αντίστοιχη διεργασία στον π-λογισμό θα είχε ως εξής:

$$!increase(a, x).\bar{a}\langle x+1\rangle$$

Όπως βλέπουμε η διεργασία του εξυπηρετητή δέχεται μέσω του καναλιού increase δύο εισόδους: το κανάλι a μέσω του οποίου θα στείλει το αποτέλεσμα στον πελάτη και το όρισμα x (που θα έχει ορισθεί ως ακέραιος από τον πελάτη). Έπειτα από την κλήση του εξυπηρετητή, αυτός θα στείλει στον πελάτη τον αμέσως μεγαλύτερο ακέραιο του x (γι' αυτό και αυξάνει το x κατά 1), μέσω του a. Χρησιμοποιείται ο τελεστής ! ούτως ώστε ο εξυπηρετητής να δημιουργεί αντίγραφα του εαυτού του και να μπορεί να εξυπηρετήσει περισσότερα από ένα αιτήματα πελατών.

Έπειτα πρέπει να μοντελοποιηθεί στον π-λογισμό η αποστολή αιτήματος από τον πελάτη. Θέλουμε λοιπόν, να καλεστεί η increase με τον αριθμό 15 και το αποτέλεσμα να δεσμευτεί στην μεταβλητή y. Επομένως η αντίστοιχη διεργασία από πλευράς πελάτη θα είχε ως εξής:

$$(\nu a)(\ increase\ \langle a, 15\rangle \mid a(y)\ )$$

Όπως βλέπουμε η διεργασία του πελάτη παράλληλα στέλνει το κανάλι a και τον αριθμό 15 μέσω του καναλιού increase στον εξυπηρετητή και λαμβάνει το αποτέλεσμα μέσω του a και το τοποθετεί το αποτέλεσμα στο y, το οποίο θα δεσμευτεί. Ο περιορισμός (va) εγγυάται ότι για κάθε αίτημα πελάτη δημιουργείται νέο ιδιωτικό κανάλι επικοινωνίας, οπότε ο πελάτης θα λάβει το ορθό αποτέλεσμα για τον αριθμό που έστειλε.
Αυτό που μένει είναι να θέσουμε τις δύο διεργασίες, του εξυπηρετητή και του πελάτη να τρέχουν παράλληλα:
!*increase*(a, x).ā⟨x+1⟩| (va)(*increase* ⟨a, 15⟩ | a(y))

2.5 Επέκταση άλγεβρας διεργασιών π-λογισμού με ομάδες

Η παρούσα εργασία ασχολείται με τη μετάφραση μίας επέκτασης της άλγεβρας διεργασιών π-λογισμού με ομάδες σε μία γλώσσα προγραμματισμού. Στον π-λογισμό με ομάδες, οι ομάδες δεν μπορούν να μεταφερθούν μεταξύ των διεργασιών και επομένως ένα δεσμευμένο όνομα/κανάλι εντός μίας ομάδας, δεν μπορεί να γίνει γνωστό στο περιβάλλον έξω από αυτήν. Ο λογισμός με τον οποίο ασχολείται η εργασία, επεκτείνει τον π-λογισμό με ομάδες και ένα σύστημα τύπων με τρόπο ώστε να ελέγχεται το πώς τα δεδομένα διαδίδονται μέσα στο σύστημα.
Θεωρούμε ότι υπάρχουν οι δύο βασικές οντότητες: το σύνολο ομάδων *G* (G, G1 ..) και το σύνολο ονομάτων/καναλιών *N* (α, b, x, y…). Επιπρόσθετα, έχουμε ένα σύνολο βασικών τύπων (base types) D (t_i) το οποίο αναφέρεται στα δεδομένα του λογισμού στα οποία πρέπει να επιβάλλονται οι πολιτικές ιδιωτικότητας. Σε κάθε στοιχείο του συνόλου *N* θα προσδίδεται ένας τύπος, ο οποίος μπορεί να είναι είτε βασικός (t_i) είτε τύπου G[T] όπου G είναι η ομάδα του ονόματος/καναλιού και T ο τύπος των δεδομένων που μπορεί να μεταφέρονται μέσω του ονόματος/καναλιού.

Με βάση τα πιο πάνω, ένας τύπος ορίζεται με την ακόλουθη γραμματική BNF:

T ::= t | G [T]

Μία ακόμα διαφοροποίηση που έχει η επέκταση του π-λογισμού με ομάδες, είναι ότι η σύνταξη ορίζεται σε δύο επίπεδα. Το επίπεδο *διεργασίας* (*P*) και το επίπεδο *συστήματος* (*S*). Στο επίπεδο διεργασίας έχουμε την τυπική σύνταξη του π-λογισμού. Στο επίπεδο

συστήματος, υπάρχει και η δομή δημιουργίας ομάδας (group) που εφαρμόζεται και στο επίπεδο διεργασίας (**v** G)P και στο επίπεδο συστήματος (**v** G)S. Υπάρχουν επίσης ο τελεστής περιορισμού (restriction) καναλιού όπως και ο τελεστής παράλληλης σύνθεσης για το επίπεδο συστήματος.

P ::= $x(y{:}T).P \mid \overline{x}.\langle z\rangle.P \mid (\nu\, a{:}T)P \mid P1|P2 \mid !P \mid \mathbf{0}$
S ::= $(\nu\, G)P \mid (\nu\, G)S \mid (\nu\, a{:}T)S \mid S1|S2 \mid \mathbf{0}$

Με τους κανόνες $(\nu\, a{:}T)P$ και $(\nu\, a{:}T)S$ το όνομα/κανάλι a δεσμεύεται στο P και στο S, αντίστοιχα. Στον κανόνα $(\nu\, G)P$ και $(\nu\, G)S$, η ομάδα G δεσμεύεται στο P και στο S, αντίστοιχα. Όσον αφορά τις ενέργειες εισόδου, εξοδου και την παράλληλη σύνθεση, τα ελεύθερα και δεσμευμένα κανάλια είναι τα ίδια με αυτά του π-λογισμού που περιγράφονται στον Πίνακα 1. Το σύνολο ελεύθερων καναλιών είναι το fn(P) για το επίπεδο διεργασιών και fn(S) για το επίπεδο συστήματος. Το σύνολο ελεύθερων ομάδων είναι το fg(T) για το επίπεδο συστήματος και fg(T) για έναν τύπο T.

Σε προηγούμενο υποκεφάλαιο, δώσαμε το πώς ορίζεται το συστημα μετάβασης (ενέργειες) στην άλγεβρα διεργασιών CCS με ένα σύνολο ετικετών Λ (σελ 13). Έτσι, και στον παρόν λογισμό, πρέπει να καθορίσουμε μία σημασιολογία μετάβασης με ετικέτες (labeled transition semantic), το οποίο περιέχει διαφοροποιήσεις από αυτήν του π-λογισμού με ομάδες με σκοπό τη μείωση της πολυπλοκότητας [14].

$l ::= \tau \mid x(y) \mid \overline{x}\langle y\rangle \mid (\nu\, y)\overline{x}\langle y\rangle$

Η ετικέτα τ αντιπροσωπεύει την εσωτερική ενέργεια και οι ετικέτες $x(y)$ και $\overline{x}\langle y\rangle$ αντιπροσωπεύουν τις ενέργειες εισόδου και εξόδου αντίστοιχα. Η ετικέτα $(\nu\, y)\overline{x}\langle y\rangle$ αντιπροσωπεύει την ενέργεια εξόδου ενός καναλιού το οποίο είναι δεσμευμένο. Οι συναρτήσεις fn(l) και bn(l) επιστρέφουν το σύνολο των ελεύθερων και δεσμευμένων καναλιών αντίστοιχα. Ορίζουμε τη σχέση dual(l, l') που συσχετίζει δύο ενέργειες:

dual(l, l') αν και μόνο αν $\{l, l'\} = \{ x(y) , \overline{x}\langle y\rangle \}$ ή $\{l, l'\} = \{ x(y), (\nu\, y)\overline{x}\langle y\rangle \}$
Δηλαδή η σχέση dual ορίζει δύο ενέργειες, μία εισόδου και μία εξόδου, ενώ πριν την έξοδο μπορεί να δεσμεύεται το κανάλι που αποστέλλεται.

Έπειτα, ορίζεται η μετασημειογραφία (meta-notation) F ::= P | S για να ορίσουμε το σημασμένο συστημα μεταβάσεων

$x(y{:}T).P \xrightarrow{x(z)} P\{z/y\}$ (In)

$\overline{x}\langle z\rangle.P \xrightarrow{\overline{x}\langle z\rangle} P$ (Out)

$$\frac{F_1 \xrightarrow{l} F'_1 \quad bn(l)\cap fn(F_2)=\emptyset}{F_1|F_2 \xrightarrow{l} F'_1|F_2}$$ (ParL)

$$\frac{F_2 \xrightarrow{l} F'_2 \quad bn(l)\cap fn(F_1)=\emptyset}{F_1|F_2 \xrightarrow{l} F_1|F'_2}$$ (ParR)

$$\frac{F \xrightarrow{l} F' \quad x\notin fn(l)}{(\nu\, x{:}T)F \xrightarrow{l} (\nu\, x{:}T)F'}$$ (ResN)

$$\frac{F \xrightarrow{\overline{x}\langle z\rangle} F'}{(\nu\, x{:}T)F \xrightarrow{(\nu\, x:T)\overline{x}\langle z\rangle} F'}$$ (Scope)

$$\frac{F \xrightarrow{l} F'}{(\nu\, G)F \xrightarrow{l} (\nu\, G)F'}$$ (ResG)

$$\frac{F \equiv_a F'' \quad F'' \xrightarrow{l} F'}{F \xrightarrow{l} F'}$$ (Alpha)

$$\frac{P \xrightarrow{l} P'}{!P \xrightarrow{l} P'|\,!P}$$ (Repl)

$$\frac{F_1 \xrightarrow{l} F'_1 \; F_2 \xrightarrow{l} F'_2 \quad dual(l_1,l_2)}{F_1|F_2 \xrightarrow{\tau} \big(\nu\, bn(l_1)\cup bn(l_2)\big)(F'_1|F'_2)}$$ (Com)

ResG: ο κανόνας αυτός διατυπώνει ότι με τη δημιουργία μίας ομάδας, οι μεταβάσεις γίνονται μέσα στον περιορισμό της ομάδας.

In: Όταν η διεργασία $x(y{:}T).P$ πραγματοποιήσει ενέργεια εισόδου με είσοδο του z στον χωροθέτη (placeholder) y τότε η διεργασία θα συνεχίσει ως P με το y να έχει αντικατασταθεί από το z.

Out: Όταν η διεργασία $\overline{x}\langle z\rangle.P$ πραγματοποιήσει ενέργεια εξόδου, τότε συνεχίζει ως P.

ParL: Αν η διεργασία F1 μετά από μία ενέργεια l συνεχίζει ως F'1 και αν δεν υπάρχουν δεσμευμένα κανάλια στο l που να ανήκουν στα ελεύθερα κανάλια μίας δεύτερης διεργασίας F2, τότε η παράλληλη σύνθεση των δύο διεργασιών F1 και F2, μετά από την ενέργεια l συνεχίζει ως F1' | F2.
(Το αντίστοιχο για τον κανόνα ParR)

ResN: Σε περίπτωση που μία διεργασία F μετά από μία ενέργεια l συνεχίζει ως F' και στην ενέργεια l το κανάλι x δεν είναι ελεύθερο, τότε αν το κανάλι x δεσμεύεται με τον κανόνα περιορισμού (restriction), η διεργασία F μετά την ενέργεια l συνεχίζει ως F' διατηρώντας το όνομα x δεσμευμένο.

Scope: Αν μία διεργασία F μετά από μία ενέργεια εξόδου με έξοδο το y συνεχίζει ως F', τότε σε περίπτωση που το y ήταν δεσμευμένο, η ενέργεια εξόδου είναι δεσμευμένη. Το y παύει να είναι δεσμευμένο στην F', αλλά είναι δεσμευμένο υπό την F' και την διεργασία στην οποία έχει σταλεί.

Com: Αν δύο παράλληλες διεργασίες μπορούν να επικοινωνήσουν με dual ενέργειες (όπως ορίζονται πιο πάνω), μπορεί να γίνει αυτό με εσωτερική ενέργεια και μετά την επικοινωνία, στην παράλληλη σύνθεση δεσμεύονται τα δεσμευμένα κανάλια των dual ενεργειών.

Alpha: Αν δύο διεργασίες, F και F'' συνδέονται με τη σχέση α-equivalence, δηλαδή αντιπροσωπεύουν στην ουσία την ίδια λειτουργία και η F'' μπορεί να ληφθεί από την F μετονομάζοντας δεσμευμένα ονόματα της F, τότε αν η F'' μετά από μία ενέργεια l συνεχίζει ως F', τότε και η F μετά από μία ενέργεια l συνεχίζει ως F'.

2.6 Συστημα Τύπων:

Όπως εξηγήθηκε αναλυτικότερα πιο πάνω, ένας τύπος στο σύστημα μπορεί να οριστεί με δύο τρόπους:

T ::= t | G [T]

2.7 Typing Judgments

Το περιβάλλον στο οποίο γίνεται ο έλεγχος τύπων (type checking) αποτελείται από το Γ. Κατά τη διάρκεια του έλεγχου τύπου συμπεριλαμβάνουμε και δύο επιπρόσθετες δομές, το περιβάλλον Δ και την διεπιφάνεια Θ. Τα τρία αυτά στοιχεία ορίζονται και επεξηγούνται πιο κάτω.

- Γ ::= ∅ | Γ· x :T | Γ· G

Το περιβάλλον Γ ορίζεται από τα ονόματα/κανάλια και τις ομάδες και των τύπο τους.

- $\Delta ::= \emptyset \mid t : \tilde{p}.\Delta$

Το περιβάλλον Δ δίνει δικαιώματα (permissions) σε τύπους ευαίσθητων δεδομένων t. Όταν τα δικαιώματα εγγραφής (read) και ανάγνωσης (write) ανατίθενται σε βασικούς τύπους t, τότε αυτό σημαίνει ότι υπάρχει δυνατότητα εγγραφής και ανάγνωσης ενός δεδομένου τύπου t μέσω καναλιών του τύπου G[T] (δηλ. κανάλια που μεταφέρουν δεδομένα τύπου t) για οποιαδήποτε ομάδα G. Το δικαίωμα πρόσβασης (access), όταν ανατίθεται σε ένα βασικό τύπο t, εκφράζει ότι είναι δυνατόν να *ληφθεί* ένα κανάλι τύπου G[T] για οποιαδήποτε ομάδα G. Τέλος, αν το δικαίωμα disclose G λ ανατεθεί σε ένα βασικό τύπο t τότε υπάρχει η δυνατότητα αποστολής καναλιών τύπου G[T] μέχρι λ φορές. Επομένως, τα δικαιώματα εγγραφής και ανάγνωσης χειρίζονται ευαίσθητα δεδομένα ενώ τα δικαιώματα πρόσβασης και αποκάλυψης (disclose) χειρίζονται κανάλια ευαίσθητων δεδομένων.

- $\Theta ::= t \gg H^{\theta}; \Theta \mid t \gg H^{\theta}$

Το H^{θ} ορίζεται ως εξής: $H^{\theta} ::= G[H^{\theta}] \mid G[\tilde{p}]$. Το H^{θ} εκφράζει μία ειδικού τύπου ιεραρχία όπου το φώλιασμα ομάδων είναι γραμμικό.
Αναφερόμαστε στο H^{θ} ως μία διεπιφάνεια ιεραρχιών. Άρα η διεπιφάνεια Θ συσχετίζει τύπους ευαίσθητων δεδομένων με μία γραμμική ιεραρχία ομάδων και ένα σύνολο δικαιωμάτων, μία οντότητα στη μορφή $G_1[G_2[\dots G_n[\tilde{p}] \dots]]$.

Σε αυτό το σημείο θα διατυπώσουμε τρεις αποφάσεις τύπου (typing judgments):

$\Gamma \vdash x \triangleright T$

Στο περιβάλλον Γ, το όνομα x θα είναι τύπου T

$\Gamma \vdash P \triangleright \Delta$

Ορίζει ότι η διεργασία P είναι ορθά ορισμένη στο περιβάλλον Γ και παράγει ένα περιβάλλον δικαιωμάτων Δ. Το Γ καταγράφει τους τύπους των ονομάτων στην P και το περιβάλλον Δ καταγράφει τα δικαιώματα των ονομάτων της διεργασίας P για κάθε τύπο.

$\Gamma \vdash S \triangleright \Theta$

Το σύστμα S είναι ορθά ορισμένο στο περιβάλλον Γ και παράγει τη διεπιφάνεια Θ, η οποία καταγράφει τις ομάδες του συστήματος S όπως και τα δικαιώματα που έχει κάθε οντότητα του συστήματος.

2.8 Τύπος συστήματος

Σε αυτό το σημείο θα ορίσουμε το τύπο συστήματος.
Πρώτα θα ξεκινήσουμε με μία χρήσιμη σημειωγραφία:

$$\Delta_T^r \begin{cases} t: read & if\ T = t \\ t: access & if\ T = G[t] \\ \emptyset & otherwise \end{cases}$$

$$\Delta_T^w \begin{cases} t: write & if\ T = t \\ t: disclose\ G1 & if\ T = G[t] \\ \emptyset & otherwise \end{cases}$$

Τελεστής $\oplus$:

Όταν ο τελεστής αυτός εφαρμόζεται σε δύο Δ διεπιφάνειες, συνδυάζει τα δικαιώματά τους για κάθε βασικό τύπο. Στην περίπτωση όμως του δικαιώματος αποκάλυψης (disclose), αν τα δικαιώματα disclose των 2 Δ διεπιφανειών αναφέρονται στην ίδια ομάδα, τότε προστίθεται ο αριθμός των δικαιωμάτων disclose (disclose (λ1+λ2)).

Όταν ο τελεστής αυτός εφαρμόζεται σε μία ομάδα G και σε μία διεπιφάνεια Δ (G $\oplus$ Δ), τότε επισυνάπτεται η ομάδα G σε όλα τα σύνολα δικαιωμάτων της διεπιφάνειας Δ, αποδίδοντας έτσι μία διεπιφάνεια Θ.

Όταν εφαρμόζεται σε μία ομάδα G και μία διεπιφάνεια Θ (G $\oplus$ Δ) , τότε η ομάδα G επισυνάπτεται σε όλες τις ιεραρχίες της διεπιφάνειεας Θ.

Αφού εξηγήσαμε κάποια σύμβολα της σημειογραφείας του τύπου συστήματος (typing system), είμαστε σε θέση να το διατυπώσουμε:

$$\frac{fg(T)\subseteq\Gamma}{\Gamma\cdot x{:}T \vdash x \triangleright T}\ \text{(Name)} \qquad \Gamma \vdash \mathbf{0} \triangleright \emptyset\ \text{(Nil)}$$

$$\frac{\Gamma\cdot y{:}T \vdash P \triangleright \Delta \quad \Gamma \vdash x \triangleright G[T]}{\Gamma \vdash x(y{:}T).P \triangleright \Delta \oplus \Delta_T^r}\ \text{(In)} \qquad \frac{\Gamma \vdash P \triangleright \Delta \quad \Gamma \vdash x \triangleright G[T] \ \ \Gamma \vdash y \triangleright T}{\Gamma \vdash \overline{x}\langle y\rangle.P \triangleright \Delta \oplus \Delta_T^w}\ \text{(Out)}$$

$$\frac{\Gamma \vdash P_1 \triangleright \Delta_1 \qquad \Gamma \vdash P_2 \triangleright \Delta_2}{\Gamma \vdash P_1|P_2 \triangleright \Delta_1 \oplus \Delta_2}\ \text{(ParP)} \qquad \frac{\Gamma \vdash S_1 \triangleright \Theta_1 \quad \Gamma \vdash S_2 \triangleright \Theta_2}{\Gamma \vdash S_1|S_2 \triangleright \Theta_1 \cdot \Theta_2}\ \text{(ParS)}$$

$$\frac{\Gamma \cdot x{:}T \vdash P \triangleright \Delta}{\Gamma \vdash (\nu\, x{:}T)P \triangleright \Delta}\ \text{(ResNP)} \qquad \frac{\Gamma \cdot x{:}T \vdash S \triangleright \Theta}{\Gamma \vdash (\nu\, x{:}T)S \triangleright \Theta}\ \text{(ResNS)}$$

$$\frac{\Gamma \cdot G \vdash P \triangleright \Delta}{\Gamma \vdash (\nu\, G)P \triangleright G \oplus \Delta}\ \text{(ResGP)} \qquad \frac{\Gamma \cdot G \vdash S \triangleright \Theta}{\Gamma \vdash (\nu\, G)S \triangleright G \oplus \Theta}\ \text{(ResGS)}$$

$$\frac{\Gamma \vdash P \triangleright \Delta}{\Gamma \vdash !P \triangleright \Delta^!}\ \text{(Rep)}$$

Επεξήγηση των πιο πάνω κανόνων:

1. Name: Όλα τα ονόματα ομάδων παρουσιάζονται στο περιβάλλον Γ.

2. In: ο κανόνας σχετίζεται με τις ενέργειες εισόδου. Αν ένα περιβάλλον Γ επεκταθεί με έναν τύπο y, παράγει το Δ μία διεπιφάνεια της διεργασίας P. Επομένως, μία διεργασία $x(y{:}T).P$ παράγει μία διεπιφάνεια όπου ο τύπος T επεκτείνεται με τα δικαιώματα Δ_T^r, ανάλογα με τις 3 περιπτώσεις που ορίστηκαν και πιο πάνω για το Δ_T^r:
    i. Αν T είναι βασικός τύπος t τότε το Δ επεκτείνεται με το δικαίωμα t:read, αφού αυτό σημαίνει ότι η διεργασία διαβάζει ένα αντικείμενο τύπου t.
    ii. Αν T = G[t] τότε το Δ επεκτείνεται με το δικαίωμα t:access (πρόσβαση), αφού αυτό σημαίνει ότι η διεργασία έχει πάρει πρόσβαση σε ένα κανάλι για έναν βασικό τύπο t.
    iii. Το Δ παραμένει ως έχει.

3. Out: Παρόμοια με τον κανόνα In. Αν το y είναι τύπου T, το x τύπου G[T] και το Δ είναι η διεπιφάνεια δικαιωμάτων του Γ, τότε η διεργασία $\overline{x}\langle y\rangle.P$ παράγει μία διεπιφάνεια που επεκτείνει την Δ με τα δικαιώματα Δ_T^w, ανάλογα με τις 3 περιπτώσεις που ορίστηκαν και πιο πάνω για το Δ_T^w:
    i. Αν T είναι βασικός τύπος t τότε το Δ επεκτείνεται με το δικαίωμα t:write, αφού αυτό σημαίνει ότι η διεργασία κάνει εγγραφή ενός δεδομένου τύπου t.
    ii. Αν T = G[t], τότε το Δ επεκτείνεται με το δικαίωμα t:disclose G1.
    iii. Το Δ παραμένει ως έχει.

4. ParP: Συνδυάζονται οι διεπιφάνειες των διεργασιών P1 και P2 σε μία παράλληλη σύνθεση.

5. ParS: Συνδυάζονται οι διεπιφάνειες των συστημάτων S1 και S2 σε μία παράλληλη σύνθεση.

6. ResNP: Αν η διεργασία P είναι ορθά διατυπωμένη στο ένα περιβάλλον $\Gamma\cdot x{:}T$ (περιβάλλον Γ που περιέχει το x:T), τότε η διεργασία $(\nu\ x)P$ είναι ορθά διατυπωμένη στο περιβάλλον Γ.

7. ResNS: Παρόμοια με ResNP, αλλά για συστήματα.

8. ResGP: Ο κανόνας αυτός αναφέρεται στη δημιουργία ομάδας διεργασιών. Αν η διεργασία P παράγει μία διεπιφάνεια δικαιωμάτων Δ, τότε η διεργασία (ν G)P παράγει μία Θ διεπιφάνεια όπου σε όλα τα δικαιώματα του Δ, προστίθεται και η ομάδα G (τελεστής $\oplus$ όπως εξηγήθηκε για G $\oplus$ Δ).

9. ResGS: Ο κανόνας αυτός αναφέρεται στη δημιουργία ομάδας συστημάτων. Αν το σύστημα S παράγει μία διεπιφάνεια Θ, τότε το σύστημα (ν G)S παράγει μία διεπιφάνεια G $\oplus$ Θ. Στη διεπιφάνεια αυτή δηλαδή, η ομάδα G προστίθεται σε όλες τις ιεραρχίες του Θ.

10. Rep: Ο κανόνας αυτός αναφέρει ότι αν η διεργασία P παράγει μία διεπιφάνεια Δ, τότε η διεργασία !P παράγει τη διεπιφάνεια $\Delta^!$, όπου $\Delta^!$ είναι τέτοιο ώστε: Αν ένας τύπος έχει το δικαίωμα της αποκάλυψης για πάνω από μία φορά (disclose G λ > 1) στο Δ, τότε έχει το δικαίωμα της αποκάλυψης για άπειρες φορές στο $\Delta^!$.

2.9 Πολιτικές

Η επέκταση του π-λογισμού με ομάδες που μελετά η συγκεκριμμένη εργασία, διατυπώνει μία γλώσσα έκφρασης των απαιτήσεων ιδιωτικότητας ενός συστήματος. Η γλώσσα αυτή αναλύεται πλήρως στο [13 σελ 5].

Η παρούσα εργασία αποτελεί συνέχεια μία προηγούμενης, στα πλαίσια της οποίας δημιουργήθηκε ένα εργαλείο το οποίο δεχόταν την διατύπωση ενός συστήματος και της πολιτικής του. Έπειτα, το εργαλείο αυτό επεξεργαζόταν τα δεδομένα και αποφάσιζε κατά πόσον το σύστημα υπακούει στην πολιτική. Το εργαλείο της παρούσας εργασίας, αποτελώντας συνέχεια του προηγούμενου, δέχεται τη διατύπωση του συστήματος, και όχι της πολιτικής, και το υλοποιεί στη γλώσσα Java. Επομένως, στο παρόν εργαλείο θεωρείται δεδομένο ότι η διατύπωση του συστήματος έχει ήδη ελεγχθεί με το

προηγούμενο εργαλείο για το αν τηρεί την πολιτική, οπότε δεν είναι αναγκαία η υποβολή διατύπωσης της πολιτικής.

Θεωρώ όμως σημαντικό για τη συνοχή εργασίας, να αναφερθούν κάποια σημαντικά σημεία όσον αφορά τη γλώσσα πολιτικής.
Αυτή η γλώσσα πολιτικής εκφράζει θετικούς και αρνητικούς κανόνες του συστήματος. Αυτοί οι κανόνες καθορίζουν τι μπορεί να συμβεί σε ένα σύστημα στην περίπτωση των θετικών κανόνων και τι δεν μπορει να συμβεί σε ένα σύστημα στην περίπτωση των αρνητικών κανόνων, σε ορίσματα δεδομένων που είναι τύποι ευαίσθητων δεδομένων στο σύστημα. Ειδικότερα, καθορίζουν πως οι διάφοροι αντιπροσώποι, αναφέρονται ανάλογα με το ρόλο τους, στο αν μπορούν ή δεν μπορούν να διαχειριστούν αυτά τα δεδομένα. Η γλώσσα πολιτικής ορίζεται με τρόπο που να καθορίζει τα *επιτρεπόμενα* και *μή επιτρεπόμενα* δικαιώματα των διαφόρων ομάδων βασικών τύπων. Τα δικαιώματα που μπορεί να έχει μια πολιτική είναι τα εξής: PPer={read, write, access, nondisclose}∪ {disclose G λ| G ∪ G , λ ∪ {1,2, …} ∪ {*}}. Από τα πιο πάνω δικαιώματα τα read , write , access και disclose , αποτελούν θετικούς κανόνες και συγκεκριμένα το read εκφράζει ότι η πληροφορία μπορεί να διαβαστεί, το write ότι μπορεί να γραφτεί, το access ότι η πληροφορία μπορεί να είναι προσβάσιμη και το disclose G λ ότι η πληροφορία μπορεί να αποκαλυφθεί σε μέλη της ομάδας λ φορές ή άπειρες φορές.

Μια πολιτική μπορεί να οριστεί με το ακόλουθο BNF και pop ⊆ PPer:

P ::= t >> H | P;P H::= ε | G : pop [H_i] i

Μια πολιτική έχει τη μορφή t_i >> H_i ;….; t_n >> H_n όπου τα ti αποτελούν τους βασικούς τύπους (base types) ενώ τα H_i αποτελούν τις ιεραρχίες των δικαιωμάτων (permissions hierarchies), δηλαδή τα δικαιώματα για κάθε βασικό τύπο (base type). Μια ιεραρχία δικαιωμάτων (permission hierarchy H) έχει τη μορφή G : pop [H_i,.....H_m], όπου η εξωτερική ομάδα έχει τα επιτρεπόμενα δικαιώματα pop και επίσης τα δικαιώματα της ιεραρχίας. Συνεπώς, μέσα απο αυτό προκύπτει ότι μια οντότητα η οποία συμμετέχει σε μια ομάδα (group) G έχει τα δικαιώματα pop και εαν επίσης ανήκει και σε μια άλλα ομάδα όπου H_i = G_I : pop i[…] τότε επίσης έχει και αυτά τα δικαιώματα pop.

Περισσότερα για τη σύνταξη της συγκεκριμμένης γλώσσας στον λογισμό μας αλλά και τον οδηγό έκφρασής της σε XML για την χρήση του σχετικού εργαλείου, είναι διαθέσιμες στην εργασία [15].

Κεφάλαιο 3
Ανάλυση απαιτήσεων και προδιαγραφών

3.1 Ανάλυση απαιτήσεων και προδιαγραφών - Ορισμός 28
3.2 Σκοπός και προοπτική 29
3.3 Περιγραφή 30
3.4 Διεπιφάνειες 31
3.4.1 Διεπιφάνειες Συστημάτων (System Interfaces) 31
3.4.2 Διεπιφάνειες Χρήστη (User Interfaces) 31
3.4.3 Διεπιφάνεια Υλικού (Hardware Interface) 32
3.4.4 Διεπιφάνεια Λογισμικού (Software Interface) 32
3.5 Απαιτήσεις εγκατάστασης από τον χρήστη 35
3.6 Υποθέσεις και Εξαρτήσεις (Assumptions and Dependencies) 36
3.7 Λειτουργικές Απαιτήσεις (Functional Requirements) 37
3.7.1 Λειτουργία 1 37
3.7.2 Λειτουργία 2 38
3.7.3 Λειτουργία 3 39
3.7.4 Λειτουργία 4 40
3.7.5 Λειτουργία 5 41
3.8 Μη λειτουργικές απαιτήσεις 42
3.9 Σχεδιασμός Συστήματος με UML 46

3.1 Ανάλυση απαιτήσεων και προδιαγραφών - Ορισμός

Είναι καλύτερο να προλαμβάνεις, παρά να θεραπεύεις. Είναι αποτελεσματικότερο να φτιάχνεις κάτι προσεκτικά σχεδιασμένο από την αρχή, παρά να το διορθώνεις εκ των υστέρων. Καλύτερα να προηγείσαι παρά να έπεσαι, να προλαμβάνεις και να προβλέπεις παρά να περιμένεις την έκβαση και κατόπιν να δράσεις. Καλύτερα να έχεις κατασταλαγμένες επιθυμίες και να ξέρεις τι ζητάς. Αυτές είναι εν πολλοίς οι βασικές αρχές που διέπουν την ανάλυση απαιτήσεων (requirements analysis). Η ανάλυση

απαιτήσεων εν ολίγοις είναι μια διαδικασία κατάρτησης μιας λίστας, όπου αναφέρονται οι προδιαγραφές που πρέπει να πληροί το έργο. Η λίστα που καταρτίζεται χρησιμεύει τόσο σε αυτούς που θα αναπτύξουν τη λύση λογισμικού όσο και σ' εκείνους που θα τη χρησιμοποιήσουν (χρήστες, πελάτες) και εφαρμόζεται σε μεγάλα αλλά και μικρά έργα.

Συχνά προγραμματιστές, αναλυτές, πελάτες και δυνητικοί χρήστες υποτιμούν την ανάλυση απαιτήσεων και δεν της αποδίδουν τη δέουσα σημασία. Αυτό έχει ως αποτέλεσμα τη δημιουργία έργων που χρήζουν βελτιώσεων ή αλλαγών, γιατί δεν ανταποκρίνονται στους στόχους και τις επιδιώξεις που είχαν αρχικά τεθεί. Ωστόσο, οι βελτιώσεις και αλλαγές στα έργα είναι ιδιαίτερα δαπανηρές, δύσκολες και χρονοβόρες, ενώ η πραγματοποίησή τους μπορεί να απαιτήσει εκ θεμελίων αναδημιουργία.

Για το λόγο αυτό, η ανάλυση απαιτήσεων, στην περιοχή της Τεχνολογίας Λογισμικού, θεωρείται απαραίτητο συστατικό για επιτυχημένη υλοποίηση εφαρμογών.

Οι σημαντικότερες ωφέλειες που απορρέουν από τη χρήση της ανάλυσης των απαιτήσεων είναι οργανωτικές, λειτουργικές και οικονομικές, με αυτήν ακριβώς τη σειρά, όχι βάσει σπουδαιότητας αλλά χρονικής ακολουθίας. Συμπιέζει σημαντικά το κόστος υλοποίησης του έργου, καθώς εξασφαλίζει ότι η ολοκλήρωσή του θα γίνει βάσει χρονοδιαγράμματος, πλαισίου και συγκεκριμένων προδιαγραφών. Γι’ αυτό στο παρόν κεφάλαιο, ακολουθώντας τις αρχές τις Τεχνολογίας Λογισμικού, γίνεται μία σημαντική αναφορά στο σκόπό, στην περιγραφή, στις λειτουργικές και μη λειτουργικές απαιτήσεις του συστήματος και στη σχεδίαση του συστήματος.

3.2 Σκοπός και προοπτική

Σκοπός του λογισμικού της παρούσας εργασίας είναι η υλοποιήση ενός συστήματος που δίνεται μοντελοποιημένο στον λογισμό που περιγράφηκε στα προηγούμενα κεφάλαια, σε μία αντικειμενοστρεφή γλώσσα προγραμματισμού. Οι δυνατοί χρήστες του συστήματος, είναι οποιαδήποτε άτομα γνωρίζουν και κατανοούν τούς κανόνες του πιο πάνω λογισμού.

Στόχος λοιπόν του εργαλείου αυτού, είναι η υλοποίηση του κώδικα ενός παράλληλου συστήματος σε γλώσσα προγραμματισμού Java. Βασικό κίνητρο είναι ότι δεν υπάρχουν αρκετά αντίστοιχα έργα μετάφρασης συστημάτων μοντελοποιημένων σε άλγεβρες

διεργασιών τα οποία ταυτόχρονα να εξυπηρετούν και σκοπούς τήρησης πολιτικών ιδιωτικότητας. Οπότε υπάρχουν περιθώρια ελαχιστοποίησης του κενού μεταξύ μοντέλου και κώδικα.

Η προοπτική μας για το εργαλείο, είναι να είναι απλό, εύχρηστο και προσιτό σε οποιοδήποτε άτομο έχει μελετήσει την επέκταση του π-λογισμού με ομάδες που περιγράφεται πιο πάνω, με το πλεονέκτημα ότι δεν χρειάζεται να έχει προγραμματιστικές γνώσεις για την υλοιποίηση ενός παράλληλου συστήματος.

Ο χρήστης αφού θα έχει έτοιμο τον βασικό κώδικα ανταλλαγής μηνυμάτων και επικοινωνίας των διεργασιών σε μία ιεραρχία ομάδων, ο οποίος σε καμία περίπτωση δεν θα παραβιάζει τις πολιτικές ιδιωτικότητας (όπως θα εξηγηθεί παρακάτω μέσω της απόδειξης ορθότητας), θα μπορεί εφόσον το επιθυμεί να προσθέσει κώδικα υπολογισμών στο σύστημα και γενικώς να προσθέσει υπολογιστικές εντολές οι οποίες δεν μπορούν να εκφραστούν στο λογισμό που αναλύθηκε πιο πάνω, αφού αυτός ο λογισμός αναφέρεται στην επικοινωνία μεταξύ των διεργασιών.

3.3 Περιγραφή

Ο χρήστης αρχικά μέσω μίας διεπιφάνειας θα καθορίζει τους βασικούς τύπους (base types) του συστήματός του. Έπειτα, θα καταχωρεί μέσω της διεπιφάνειας, το XML αρχείο όπου υπάρχει η διατύπωση του συστήματος. Ακολούθως, ο χρήστης εισάγει και το αρχείο με τα κανάλια του Γ περιβάλλοντος. Το συγκεκριμμένο αρχείο είναι σημαντικό για την παραγωγή του κώδικα μετάφρασης.

Οδηγίες για το πώς η σύνταξη του λογισμού που αναφέρεται στο Κεφάλαιο 2 μπορεί να εκφραστεί σε XML, θα δοθεί στο εγχειρίδιο χρήστη. Αρχικά το σύστημα θα ελέγχει τη σύνταξη του XML αρχείου. Σε περίπτωση που η σύνταξη παρουσιάζει κάποιο σφάλμα, θα ενημερώνεται ο χρήστης. Ειδάλλως, θα προχωράει στην επόμενη λειτουργία.

Μετά, για σκοπούς επιβεβαίωσης, θα εμφανίζεται στον χρήστη η αντίστοιχη διατύπωση του συστήματος στο μοντέλο της επέκτασής μας π-λογισμού με ομάδες. Αυτό θα γίνεται καθώς ακόμα και αν το XML αρχείο δεν περιέχει κάποιο συντακτικό ή δομικό λάθος, ίσως ο χρήστης να μην μετέφρασε ορθά την διατύπωση του λογισμού σε XML και το λάθος να μην είναι συντακτικά ανιχνεύσιμο. Με την παρουσίαση του

συστήματος σε διατύπωση π-λογισμού, θα έχει την ευκαιρία ο χρήστης να εντοπίσει αν υπάρχει κάποιο λάθος στο XML αρχείο του.

Τέλος, θα παράγεται ο κώδικας του συστήματος σε Java, ακολουθώντας διάφορες τεχνικές που θα εξηγηθούν αργότερα. Τα αρχεία του παράλληλου συστήματος, θα αποθηκεύονται στον τρέχον κατάλογο. Στο τέλος, θα εμφανίζεται μήνυμα στο χρήστη για το πιο αρχείο πρέπει να τρέξει για να εκτελεστεί το σύστημα.

Το αρχείο είναι για προσωπική χρήση και δεν θα δημοσιοποιεί ούτε θα αποθηκεύει τα XML αρχεία. Επίσης, εφόσον δεν χρησιμοποιεί σε καμία περίπτωση προσωπικά δεδομένα του χρήστη, δεν κρίνεται απαραίτητο η αναγνώριση/αυθεντικοποιήση χρήστη.

3.4 Διεπιφάνειες

3.4.1 Διεπιφάνειες Συστημάτων (System Interfaces)

Το εργαλείο αυτό δεν έχει τέτοιαν απαίτηση καθώς δεν υπάρχει η αναγκαιότητα σύνδεσής του με άλλο σύστημα/υπηρεσία, για την διεκπαιρέωση οποιασδήποτε λειτουργίας του. Ναι μεν θεωρείται δεδομένο ότι ο χρήστης έχει χρησιμοποιήσει προηγουμένως το εργαλείο προηγούμενης διπλωματικής εργασίας που ελέγχει αν το σύστημα ταυτίζεται με μία πολιτική ιδιωτικότητας, αλλά δεν απαιτείται πραγματική σύνδεση των δύο συστημάτων, αφού το παρόν εργαλείο μπορεί να εκτελέσει τις λειτουργίες του ανεξάρτητα χωρίς την ανάγκη άλλου συστήματος για να εξάγει την υλοποιήση.

3.4.2 Διεπιφάνειες Χρήστη (User Interfaces)

Αρχικά, μέσω μίας διεπιφάνειας, ο χρήστης καθορίζει τους βασικούς τύπους για τον καθένα από τους οποίους θα δημιουργηθεί μία κλάση. Για τον κάθε τύπο μπορεί να ορίσει διάφορα χαρακτηριστικά που μπορεί να είναι τύπου λέξη (string), ακέραιος (integer) ή ακόμα και πίνακας ή λίστα (array list). Έπειτα μέσω μίας διεπιφάνειας εισαγωγής αρχείου, θα εισάγει το xml αρχείο με την διατύπωση του συστήματος και το xml αρχείο όπου περιγράφονται τα κανάλια του Γ περιβάλλοντος. Σε επόμενη διεπιφάνεια, θα εμφανίζονται τα ανάλογα παράθυρα μηνυμάτων εξόδου.

Οι παραπάνω λειτουργίες θα διεκπεραιώνονται βάσει του μενού της διεπαφής του εργαλείου το οποίο περιλαμβάνει μεταξύ άλλων:

- Παράθυρα εισαγωγής κειμένου
- Λίστες επιλογής
- Κουμπιά (buttons)
- Παράθυρα προβολής μηνύματος
- Προβολή μηνυμάτων λάθους

3.4.3 Διεπιφάνεια Υλικού (Hardware Interface)

Το σύστημά μας δεν έχει τέτοια απαίτηση καθώς δεν αποκτά πρόσβαση σε κάποιο εξοπλισμό όπως για παράδειγμα μία κάμερα ή μικρόφωνο, ούτε επικοινωνεί με κάποιον εξυπηρετητή ή βάση δεδομένων.

3.4.4 Διεπιφάνεια Λογισμικού (Software Interface)

Γλώσσα Υλοποίησης:

Όνομα: Java

Έκδοση: Version 8

Πηγή: Oracle

Η Java είναι μία γενικού σκοπού γλώσσα προγραμματισμού που είναι ταυτόχρονη (concurrent), βασισμένη σε κλάσεις και αντικειμενοστρεφής. Είναι σχεδιασμένη ούτως ώστε να έχει όσο το δυνατόν λιγότερες εξαρτήσεις υλοποίησης. Προοπτική της είναι να δίνει την ευκαιρία στους δημιουργούς εφαρμογών «να γράφουν μία φορά και να τις τρέχουν παντού» (write once, run everywhere), κάτι που σημαίνει ότι ένας μετταγλωτισμένος κώδικας σε Java μπορεί να τρέξει σε κάθε πλατφόρμα που υποστηρίζει Java χωρίς να χρειάζεται ξανά μεταγλώττιση. Οι Java εφαρμογές μεταγλωττίζονται σε bytecode(*) που μπορεί να τρέξει σε οποιοδήποτε Java Virtual Machine, ανεξάρτητα της αρχιτεκτονικής του υπολογιστή.

Είναι μία από της πιο δημοφιλείς γλώσσες προγραμματισμού που χρησιμοποιείται, ιδιαίτερα σε διαδικτυακές εφαρμογές μοντέλου πελάτη-εξυπηρετητή, με περίπου 9 εκατομμύρια προγραμματιστές.
Στις αρχές του 1991, η Sun αναζητούσε το κατάλληλο εργαλείο για να αποτελέσει την πλατφόρμα ανάπτυξης λογισμικού σε μικρο-συσκευές (έξυπνες οικιακές συσκευές έως πολύπλοκα συστήματα παραγωγής γραφικών). Τα εργαλεία της εποχής ήταν γλώσσες όπως η C++ και η C. Μετά από διάφορους πειραματισμούς προέκυψε το συμπέρασμα ότι οι υπάρχουσες γλώσσες δεν μπορούσαν να καλύψουν τις ανάγκες τους. Ο "πατέρας" της Java, James Gosling, που εργαζόταν εκείνη την εποχή για την Sun, έκανε ήδη πειραματισμούς πάνω στη C++ και είχε παρουσιάσει κατά καιρούς κάποιες πειραματικές γλώσσες (C++ ++,που μετέπειτα ονομαστικέ C#) ως πρότυπα για το νέο εργαλείο που αναζητούσαν στην Sun. Τελικά μετά από λίγο καιρό κατέληξαν με μια πρόταση για το επιτελείο της εταιρίας, η οποία ήταν η γλωσσα Oak. Το όνομά της το πήρε από το ομώνυμο δένδρο (βελανιδιά) το οποίο ο Gosling είχε έξω από το γραφείο του και έβλεπε κάθε μέρα.
Στις 27 Απριλίου 2010 η εταιρία λογισμικού Oracle Corporation ανακοίνωσε ότι μετά από πολύμηνες συζητήσεις ήρθε σε συμφωνία για την εξαγορά της Sun Microsystems και των τεχνολογιών (πνευματικά δικαιώματα/ πατέντες) που η δεύτερη είχε στην κατοχή της ή δημιουργήσει. Η συγκεκριμένη συμφωνία θεωρείται σημαντική για το μέλλον της Java και του γενικότερου οικοσυστήματος τεχνολογιών γύρω από αυτή μιας και ο έμμεσος έλεγχος της τεχνολογίας και η εξέλιξη της περνάει σε άλλα χέρια.

Η σύνταξη της Java είναι εμπνευσμένη από την C++, αλλά σε αντίθεση με την C++ αναπτύχθηκε σχεδόν αποκλειστικά ως αντικειμενοστρεφής γλώσσα. Ο κώδικας γράφεται σε κλάσεις και κάθε δεδομένο είναι αντικείμενο, με εξαίρεση τους πιο βασικούς τύπους δεδομένων (πχ ακέραιοι – int) που δεν είναι αντικείμενα για σκοπούς απόδοσης.

Οι λόγοι για τους οποίους επιλέγηκε η Java για την ανάπτυξη του εργαλείου μας είναι αρχικά ότι είναι ανεξάρτητη λειτουργικού συστήματος και πλατφόρμας και η αντικειμενοστρέφειά της. Τα προγράμματα που είναι γραμμένα σε Java τρέχουμε με ακριβώς ίδια εκτέλεση σε Windows, Linux, Unix κλπ χωρίς να χρειαστεί να ξαναγίνει μεταγλώττιση ή να αλλάξει ο πηγαίος κώδικας. Αυτό επιτυγχάνεται με την Εικονική Μηχανή (Java Virtual Machine).

Ο βασικότερος παράγοντας όμως ήταν η αντικειμενοστρέφεια της γλώσσας που διευκολύνει πολύ την αντιστοιχία της υλοποίησης με την επέκταση του π-λογισμού με ομάδες. Οι διεργασίες μεταφράζονται σε κλάσεις και για κάθε διεργασία δημιουργείται αντικείμενο της κλάσης στην κατάλληλη ιεραρχία ομάδων (διεπιφάνεια Θ) και μπορούν να τρέξουν παράλληλ. Επίσης, τα κανάλια και οι βασικοί τύποι μπορούν να μεταφραστούν ορθά σε αντικείμενα.

Πλατφόρμα Υλοποίησης του Εργαλείου Μετάφρασης:
Όνομα: Eclipse Jee Mars
Έκδοση: 4.5.0
Πηγή: The Eclipse Foundation
Το eclipse είναι μία ανοικτού πηγαίου κώδικα πλατφόρμα βασισμένη στην Java, η οποία επιτρέπει στους προγραμματιστές να δημιουργήσουν ένα προσαρμοσμένο περιβάλλον ανάπτυξης (IDE) από επιπρόσθετα συστατικά (plug – ins) της οικογένειας Eclipse. Προσφέρει εργαλεία για ανάπτυξη κώδικα (coding), εκτέλεση (running) και αποσφαλμάτωση (debugging).

Ξεκίνησε ως εγχείρημα της IBM, σχεδιασμένο κυρίως για ανάπτυξη έργων σε Java, αλλά πλέον υποστηρίζει πολλές άλλες γλώσσες προγραμματισμού με τη χρήση plug-ins. Προσφέρει πολύ καλή υποστήριξη για τη γλώσσα C και C++ αλλά έχει τη δυνατότητα να υποστηρίξει και διάφορες άλλες γλώσσες Perl, PHP, Prolog, Python, Ruby, Scala, JavaScript. Προσφέρει περιβάλλοντα ανάπτυξης όπως το Eclipse Java Development (JDK) για τις γλώσσες Java και Scala, το Eclipse CDT για C και C++, το Eclipse PDT για PHP και άλλα.

Βασικό πλεονέκτημά του είναι ότι αποτελεί μία γενικού σκοπού ανοικτού κώδικα πλατφόρμα που διευκολύνει και ενθαρρύνει την ανάπτυξη plug-ins από τρίτους.
Ο λόγος που επιλέχθηκε η συγκεκριμμένη πλατφόρμα για την ανάπτυξη του εργαλείου που κάνει τη μετάφραση είναι αρχικά ότι είναι δωρεάν, με χρήσιμα εργαλεία που διευκολύνουν την ανάπτυξη γραφικού περιβάλλοντος. Επίσης, είναι μία πλατφόρμα που χρησιμοποιείται από μία ευρεία κοινότητα, οπότε υπάρχει αρκετό υλικό τεκμηρίωσης στον παγκόσμιο ιστό.

Για να τρέξει το Eclipse χρειάζεται η εγκατάσταση JRE ή JDK, αφού είναι γραμμένο σε Java.

Εκδίδεται σύμφωνα με τους όρους της Eclipse Public License, Eclipse SDK είναι ελεύθερο και ανοικτού κώδικα λογισμικό (αν και είναι ασυμβίβαστη με την GNU General Public License).

3.5 Απαιτήσεις εγκατάστασης από τον χρήστη

JRE:
Ο χρήστης πρέπει να έχει εγκατεστημένο Java Runtime Environment καθώς το εργαλείο μετάφρασης είναι γραμμένο σε Java, και είναι απαραίτητο για την εκτέλεση Java προγραμμάτων. Το Java Runtime Environment δεν χρειάζεται μόνο για την εκτέλεση του εργαλείου μετάφρασης, αλλά και για την εκτέλεση του παραγόμενου κώδικα, ο οποίος θα είναι κι αυτός σε Java.

JCSP:

Η JCSP είναι μία βιβλιοθήκη που αναπτύχθηκε αρχικά από τους Peter Welch και Paul Austin στο Πανεπιστήμιο του Κεντ. Η πρώτη έκδοση έγινε τον Ιούνιο του 1998. Είναι μία Java βιβλιοθήκη που παρέχει το βασικό εύρος των αρχών και κανόνων της άλγεβρας διεργασιών CSP αλλά και ένα επιπρόσθετο σύνολο από επεκτάσεις. Παρόλο που η CSP είναι ένα μαθηματικό μοντέλο, η βιβλιοθήκη JCSP δεν απαιτεί εις βάθους μαθηματικές δεξιότητες, απλώς επιτρέπει στους προγραμματιστές να αναπτύξουν λογισμικά που ελαχιστοποιούν κακές συμπεριφορές, για παράδειγμα με αποφυγή των αδιεξόδων.

Η JCSP επιτρέπει τη δυναμική και την ιεραρχική δομή των δικτύων διεργασίας, συνδέδεμένων μέσω συγχρονισμού πάνω σε ένα μικρό αρχών/κανόνων, όπως είναι το πέρασμα μηνυμάτων μέσω καναλιών και τα αυθυκατευθυνόμενα γεγονότα. Κάθε διαδικασία διαχειρίζεται τη δική της κατάσταση και ενσωματώνεται στα πρότυπα της επικοινωνίας του περιβάλλοντός της (που αντιπροσωπεύεται από τα κανάλια, κανόνες,

κλπ) που μπορεί να είναι αυστηρά διατυπωμένα. Κάθε διεργασία έχει κατασκευαστεί και ελεχθεί ανεξάρτητα, χωρίς την ανησυχία για προβλήματα που συναντώνται συχά στην παράλληλη επεξεργασία, ούτε υπάρχει ανάγκη για μηχανισμούς κλειδώματος μεταβλητων. Η συμπεριφορα ενός συστήματος που χρησιμοποιεί την JCSP δεν περιέχει εκπλήξεις ως προς τη λειτουργία των καναλιών και μπορεί να αναλυθεί με αυστηρή διατύπωση για την ύπαρξη ανεπιθύμητης συμπεριφοράς (πχ. αδιέξοδα). Η JCSP είναι ένα εναλλακτικό μοντέλο ταυτοχρονισμού έναντι των νημάτων και των μηχανισμών παρακολούθησης (monitot mechanisms) της Java. Είναι, επίσης, συμβατή με αυτά και η υλοποίησή της τα χρησιμοποεί. Επομένως, με προσοχή τα δύο μοντέλα ταυτοχρονισμού μπορούν να συνδυαστούν με καλό αποτέλεσμα [16].

Η συγκεκριμμένη βιβλιοθήκη είναι απαραίτητη για τον χρήστη. Δεν χρησιμοποιείται από το εργαλείο που παράγει τη μετάφραση, αλλά από τον παραγόμενο κώδικα. Κλάσεις και μεθόδοι της JCSP χρησιμοποιούνται για την ορθή αντιστοιχία των κανόνων του λογισμού μας στην γλώσσα προγραμματισμού Java. Ο χρήστης μπορεί να κατεβάσει την συγκεκριμμένη βιβλιοθήκη από το σύνδεσμο: https://www.cs.kent.ac.uk/projects/ofa/jcsp/

3.6 Υποθέσεις και Εξαρτήσεις (Assumptions and Dependencies)

Το εργαλείο που θα δημιουργηθεί θεωρείται συνέχιση προηγούμενης διπλωματικής εργασίας, στην οποία με βάση τη διατύπωση του συστήματος σε XML και τη διατύπωση μία πολιτικής, αποφασίζεται αν το σύστημα τηρεί την πολιτική. Γι’ αυτό το σκοπό, η σύνταξη του XML για την διατύπωση του συστήματος, ακολουθεί τους κανόνες που ορίστηκαν στην προηγούμενη εργασία, για να μπορούν τα εργαλεία να δέχονται το ίδιο XML αρχείο. Επίσης, θα χρησιμοποιηθεί το ίδιο εργαλείο ανάγνωσης και ανάλυσης του XML αρχείου (Parser) το οποίο ονομάζεται Java Dom Parser.

Επίσης, ο παραγόμενος κώδικας που θα αποτελεί την υλοποιήση του μοντελοποιημένου παράλληλου συστήματος, πρέπει να ακολουθεί και να μην παραβιάζει τους κανόνες του λογισμού που διατυπώθηκαν στο προηγούμενο κεφάλαιο. Ο χρήστης όμως, εφόσον λάβει τον παραγόμενο κώδικα, θα μπορεί εφόσον το επιθυμεί να προσθέσει λειτουργίες και υπολογισμούς οι οποίοι δεν μπορούν να εκφραστούν στο λογισμό μας και δεν

αφορούν την επικοινωνία μεταξύ των διεργασιών. Ο χρήστης έχει αυτή τη δυνατότητα καθώς σκοπός μας είναι ο παραγόμενος κώδικας να μπορεί να αποτελέσει σκελετό ενός πραγματικού συστήματος με πραγματική εφαρμογή, και συνήθως τέτοιου τύπου συστήματα, δεν περιορίζονται απλά στην ανταλλαγή μηνυμάτων μεταξύ των διεργασιών, αλλά και στην πραγματοποίηση υπολογισμών εντός των διεργασιών. Οι υπολογισμοί αυτοί, δεν θα επηρεάζουν την ορθότητα του συστήματος αφού δεν θα πρέπει να επεμβαίνουν στα σημεία επικοινωνίας διεργασιών, ούτως ώστε να αποφεύγεται η δημιουργία προβλημάτων όπως είναι τα αδιέξοδα.

3.7 Λειτουργικές Απαιτήσεις (Functional Requirements)

Λειτουργία 1: Καθορισμός Βασικών Τύπων Συστήματος
Λειτουργία 2: Εισαγωγή της διατύπωσης του συστήματος και των καναλιών του περιβάλλοντος Γ
Λειτουργία 3: Έλεγχος ορθότητας σύνταξης του XML αρχείου
Λειτουργία 4: Εμφάνιση της διατύπωσης του συστήματος σε διατύπωση του μαθηματικού λογισμού
Λειτουργία 5: Πραγματοποίηση μετάφρασης και παραγωγή των κλάσεων του συστήματος

3.7.1 Λειτουργία 1

Καθορισμός Βασικών Τύπων Συστήματος

Περιγραφή:
Το μοντελοποιημένο σύστημα, πέρα από κανάλια, θα έχει αντικείμενα ευαίσθητων δεδομένων των οποίων οι τύποι ονομάζονται Βασικοί Τύποι. Για κάθε τύπο αντικειμένου θα δημιουργείται μία κλάση.

Εισόδοι:
Ο χρήστης μέσα από τη διεπιφάνεια, για κάθε βασικό τύπο αντικειμένου του συστήματός του θα επιλέγει πατώντας το ανάλογο κουμπί, τη δημιουργία νέου αντικειμένου. Έπειτα, για το αντικείμενο, θα πρέπει να εισάγει πόσα ορίσματα θα

έχει η κλάση του αντικειμένου. Εισάγει κάθε φορά ένα νέο όρισμα και επιλέγει μέσω μίας λίστας επιλογών τον τύπο του ορίσματος από τους διαφορετικές δυνατές επιλογές που δίνονται. Πρέπει κάθε φορά να εισάγει, μέσω παραθύρων εισόδου (Message Input Dialogue) το όνομα του κάθε αντικειμένου και το όνομα του κάθε ορίσματος των κλάσεων.

Επεξεργασία:
Αφού το εργαλείο μας, έχει δεχτεί τα απαραίτητα δεδομένα για τους βασικούς τύπους από τον χρήστη, τότε ξεκινά η παραγωγή του κώδικα της κάθε κλάσης αντικειμένου. Τα ορίσματα θα ορίζονται ως public για διευκόλυνση στη μετέπειτα χρήση των βασικών τύπων, αλλά παρόλα αυτά για λόγους επεκτασιμότητας, για κάθε όρισμα θα υπάρχουν οι αντίστοιχες συναρτήσεις get και set.

Έξοδος:
Στον τρέχον κατάλογο στον οποίο τρέχει το σύστημα, θα παράγονται οι κλάσεις των διαφορετικών βασικών τύπων με τις βασικές συναρτήσεις.

3.7.2 Λειτουργία 2

Εισαγωγή της διατύπωσης του συστήματος και των καναλιών του περιβάλλοντος Γ

Περιγραφή:
Ο χρήστης θα πρέπει να δίνει τα δύο αρχεία που θα πρέπει να ανταποκρίνονται στις οδηγίες που θα δοθούν παρακάτω.

Εισόδοι:
Τα δύο αρχεία XML τα οποία αποτελούν την διατύπωση του συστήματος αλλά και τα ονόματα (names) του περιβάλλοντος Γ (όπως περιγράφηκε στο προηγούμενο κεφάλαιο), θα εισάγονται στο σύστημα από τον χρήστη μέσω ενός παραθύρου εισαγωγής αρχείων.

Επεξεργασία:
Εφόσον δίνονται τα δύο αρχεία, το περιεχόμενό τους θα αναλύεται και θα

διαβάζεται με τη χρήση του Java Dom Parser, ο οποίος μετατρέπει το XML περιεχόμενο σε ένα δέντρο ιεραρχημένων κόμβων.

Έξοδος:
Θα εμφανίζονται μηνύματα λάθους σε περίπτωση που δεν υπάρχει το αρχείο. Ειδάλλως το σύστημα θα προχωράει στην επόμενη λειτουργία.

3.7.3 Λειτουργία 3

Έλεγχος ορθότητας σύνταξης των XML αρχείων

Περιγραφή:
Είναι σημαντικό πριν προχωρήσει το εργαλείο στην πραγματοποίηση της μετάφρασης, να ελέγξει ότι η σύνταξη των αρχείων είναι ορθή. Ναι μεν δίνεται ο οδηγός συγγραφής τους στο εγχειρίδιο χρήστη, αλλά παρόλα αυτά πρέπει ο χρήστης αλλά και το σύστημα να επιβεβαιωθούν για την ορθότητά του ούτως ώστε να μην παρουσιαστούν τυχόν προβλήματα κατά την μετάφραση που μπορεί να έχουν ως αποτέλεσμα ακόμα και τη παραγωγή μη-μεταγλωττίσιμου και λανθασμένου συντακτικά κώδικα.

Είσοδος:
Η είσοδος της συγκεκριμμένης λειτουργίας είναι η ρίζα των δέντρων που έχει δημιουργήσει ο Java Dom Parser από την Λειτουργία 2. Με τη ρίζα του δέντρου μπορεί το πρόγραμμα να προχωρήσει στο πέρασμα όλου του δέντρου που αντιπροσωπεύει το κάθε XML αρχείο. Δηλαδή, γι’ αυτή τη λειτουργία δεν χρειάζεται κάποια επιλέον είσοδος από τον χρήστη μέσω της διεπιφάνειας.

Επεξεργασία:
Το πρόγραμμα χρησιμοποιώντας αναδρομική κατά βάθος ανάγνωση, θα διαπερνά το δέντρο. Κάθε φορά, στο επίπεδο ελέγχου, ανάλογα με τον κόμβο που διαβάζει το πρόγραμμα, θα καθορίζει ποιες είναι οι επιτρεπόμενες επιλογές που μπορεί να είναι ο κόμβος στο κατώτερο επίπεδο. Εφόσον διαβάσει τον κόμβο του κατώτερου

επιπέδου θα αποφασίζει αν αυτός είναι στα πλαίσια των επιτρεπόμενων επιλογών. Σε περίπτωση που δεν είναι, θα εμφανίζεται μήνυμα λάθους και θα πρέπει ο χρήστης να εισάγει νέο αρχείο.

Έξοδος:
Στη λειτουργία αυτή, σε περίπτωση που ανιχνευτεί λάθος σε κάποιο από τα XML αρχεία, θα ζητείται από το χρήστη να ξαναεισάγει μέσω διεπιφάνειας εισαγωγής αρχείου. Ειδάλλως, θα εμφανίζει μήνυμα στον χρήστη, ότι τα αρχεία είναι ορθά οπότε το σύστημα προχωράει στην μετάφραση.

3.7.4 Λειτουργία 4

Εμφάνιση της διατύπωσης του συστήματος σε διατύπωση του μαθηματικού λογισμού

Περιγραφή:
Είναι σημαντικό για πλήρη επιβεβαίωση του χρήστη ότι το XML αρχείο του συστηματος διατυπώνει αυτό ακριβώς που επιθυμεί, να εμφανίζεται το περιεχόμενο του αρχείου στη σύνταξη του λογισμού που περιγράφηκε στο προηγούμενο κεφάλαιο.

Είσοδος:
Η είσοδος της συγκεκριμμένης λειτουργίας είναι η ρίζα των δέντρων που έχει δημιουργήσει ο Java Dom Parser από τη Λειτουργία 2. Με τη ρίζα του δέντρου μπορεί το πρόγραμμα να προχωρήσει στο πέρασμα όλου του δέντρου που αντιπροσωπεύει το κάθε XML αρχείο. Δηλαδή, γι’ αυτή τη λειτουργία δεν χρειάζεται κάποια επιλέον είσοδος από τον χρήστη μέσω της διεπιφάνειας.

Επεξεργασία:
Το πρόγραμμα χρησιμοποιώντας αναδρομική κατά βάθος ανάγνωση, θα διαπερνά το δέντρο. Κάθε φορά, στο επίπεδο ελέγχου, ανάλογα με τον κόμβο που διαβάζει το πρόγραμμα, θα καθορίζει ποια είναι η αντιστοιχία σύνταξης στο λογισμό μας και θα το εμφανίζει. Ιδιαίτερο ενδιαφέρον παρουσιάζει η διαδικασία ισοζυγισμού των παρενθέσεων. Κάθε φορά που ο κατεβαίνουμε επίπεδο, αν εντοπίζουμε κάτι

εκτός από ενέργειες εισόδου ή εξόδου, ανοίγουμε παρενθέσεις, έπειτα ανεβαίνουμε επίπεδα μέχρι τη ρίζα για να κλείσουμε τις παρενθέσεις.

Έξοδος:
Σε ένα παράθυρο εμφάνισης μηνύματος, εμφανίζεται η διατύπωση του συστήματος στη μορφή του λογισμού μας.

3.7.5 Λειτουργία 5

Πραγματοποίηση μετάφρασης και παραγωγή των κλάσεων του συστήματος

Περιγραφή:
Αυτό το σημείο, είναι το κύριο μέρος λειτουργίας του εργαλείου μας. Εδώ με βάση τα XML αρχεία, θα δημιουργηθεί η μετάφραση του μοντελοποιημένου συστήματος σε γλώσσα Java. Η μετάφραση, πρέπει να είναι πολύ προσεκτική, ελαχιστοποιώντας τις πιθανότητες σφάλματος αφού ο παραγόμενος κώδικας πρέπει να είναι συντακτικά ορθός.

Είσοδος:
Η είσοδος της συγκεκριμμένης λειτουργίας είναι η ρίζα των δέντρων που έχει δημιουργήσει ο Java Dom Parser από την Λειτουργία 2. Με τη ρίζα του δέντρου μπορεί το πρόγραμμα να προχωρήσει στο πέρασμα όλου του δέντρου που αντιπροσωπεύει το κάθε XML αρχείο. Δηλαδή, γι’ αυτή τη λειτουργία δεν χρειάζεται κάποια επιπλέον είσοδος από τον χρήστη μέσω της διεπιφάνειας.

Επεξεργασία:
Ακολουθείται και σε αυτό το σημείο κατα βάθος προσπέλαση του δέντρου του συστήματος. Το εργαλείο με προσοχή αποφασίζει σε ποια σημεία πρέπει να δημιουργηθούν νέες κλάσεις, σε ποια σημεία δημιουργείται νέο όνομα και σε ποια σημεία έχουμε ενέργειες εισόδου ή εξόδου. Εν τω μεταξύ, ακόμα και πριν το στάδιο της μετάφρασης, γίνεται μία προσπέλαση του δέντρου ούτως ώστε να καθοριστούν τα ελεύθερα κανάλια, τα οποία πρέπει να δημιουργηθούν στην κλάση που αντιστοιχεί στην υψηλότερη ιεραρχία ομάδων.

Έξοδος:

Η έξοδος του προγράμματος σε αυτό το σημείο είναι οι κλάσεις του μεταφρασμένου μοντέλου συστήματος. Στην οθόνη, σε ένα παράθυρο εμφάνισης μηνύματος, εμφανίζεται το όνομα της κλάσης με την συνάρτηση main την οποία πρέπει ο χρήστης να τρέξει ούτως ώστε να εκτελέσει το πρόγραμμα.

3.8 Μη λειτουργικές απαιτήσεις

Οι μη λειτουργικές απαιτήσεις περιγράφουν πώς (ή το πόσο καλά) το σύστημα θα υποστηρίξει τις λειτουργικές απαιτήσεις. Μπορούμε να διαχωρίσουμε τις μη λειτουργικές απαιτήσεις σε Απαιτήσεις Χρηστών, Απαιτήσεις Ανάπτυξης και Εξωτερικές απαιτήσεις.

Απαιτήσεις χρηστών:

- Διαθεσιμότητα (*availability*). Μέσος χρόνος για αποτυχία (*mean time to failure, MTTF*).
- Επεκτασιμότητα (*extensibility*). Δυνατότητα προσθήκης συγκεκριμένου χαρακτηριστικού.
- Αποδοτικότητα (*performance*). Ταχύτητα απόκρισης, αιτήσεις ανά δευτερόλεπτο, αριθμός ταυτόχρονων χρηστών.
- Αξιοπιστία (*reliability*). Ποσοστό διαδικασιών που εκτελούνται σωστά.
- Στιβαρότητα (*robustness*). Ανεκτικότητα σε λάθος δεδομένα εισόδου ή σε ελαττώματα του υλικού ή άλλου λογισμικού.
- Ευχρηστία (*usability*). Ευκολία με την οποία τον χρήστης ολοκληρώνει μια διαδικασία.

Απαιτήσεις Ανάπτυξης:

- Συντηρησιμότητα (*maintainability*). Δυνατότητα διόρθωσης ενός λάθους ή αναβάθμισης.
- Μεταφερσιμότητα (*portability*). Ποσοστό μεταφέρσιμου κώδικα, αριθμός συμβατών συστημάτων.
- Δυνατότητα επαναχρησιμοποίησης (*reusability*). Προσδιορισμός μονάδων που θα πρέπει να υλοποιηθούν ως ανεξάρτητες βιβλιοθήκες.

Εξωτερικές Απαιτήσεις - Νομικό πλαίσιο (*legislative*):

- Προστασία προσωπικών δεδομένων (*privacy*).
- Ασφάλεια (*safety*).
- Έλεγχος (*auditing*).

Διαθεσιμότητα

Το εργαλείο είναι διαθέσιμο από όλους τους χρήστες για όλο το 24ωρο, αφού είναι εγκατεστημένο και τρέχει τοπικά στη μηχανή του χρήστη. Δεν επικοινωνεί με κάποια βάση δεδομένων, ούτε με κάποιον εξυπηρετητή ούτως ώστε η διαθεσιμότητά του να εξαρτάται από κάποιο άλλο σύστημα.

Επεκτασιμότητα

Το εργαλείο πρέπει να μπορεί να επεκταθεί ούτως ώστε στο μέλλον, με επόμενες εργασίες να μπορεί να υποστηρίξει περισσότερες λειτουργίες ή ακόμα και να αναβαθμίσει κάποιες ήδη υπάρχουσες. Το εργαλείο μπορεί με τρόπο απλό να επεκταθεί, αφού για κάθε λειτουργία, υπάρχει μία ανεξάρτητη κλάση η οποία αναλαμβάνει να τη διεκπαιρεώσει. Οπότε, η προσθήκη λειτουργιών, δεν μπορεί να επηρεάσει αρνητικά τον κώδικα των άλλων λειτουργιών.

Αποδοτικότητα

Οι πληροφορίες που θα εισάγονται στο σύστημα θα είναι τύπου κειμένου, αριθμών ή αρχεία XML. Οπότε η αποδοτικότητα του εργαλείου δεν θα επηρεάζεται σε σημαντικό βαθμό από τις εισόδους του χρήστη. Η πολυπλοκότητα και το μέγεθος του συστήματος που μοντελοποιείται στο αρχείο XML είναι αυτά που θα μπορούν να επηρεάσουν την αποδοτικότητα. Παρόλα αυτά, ακόμα και για πολύ μεγάλα μοντελοποιημένα συστήματα, δεν αναμένεται ο χρόνος ανταπόκρισης να ξεπεράσει το μέγιστο όριο των 10 δευτερολέπτων.

Αξιοπιστία

Η αξιοπιστία του συστήματος, δηλαδή η ορθότητά του, είναι άμεσα συνδεδεμένη με την τήρηση των κανόνων της επέκτασης του π-λογισμού με ομάδες που περιγράψαμε

στο προηγούμενο κεφάλαιο. Γι' αυτό θεώρησα σκόπιμο να αναλυθεί λεπτομερώς ανεξάρτητα σε επόμενο κεφάλαιο.

Στιβαρότητα

Σαφέστατα, είναι πολύ σημαντικό, το σύστημα να είναι ανεκτικό σε λανθασμένα δεδομένα δοσμένα από τον χρήστη. Για τον σκοπό αυτό, υπάρχουν δύο τρόποι ελέγχου των XML αρχείων που περιγράφηκαν στις λειτουργικές απαιτήσεις, ούτως ώστε να ελαχιστοποιηθεί η πιθανότητα το πρόγραμμα να προχωρήσει με την ύπαρξη λανθασμένων δεδομένων και να οδηγηθεί σε μία απρόβλεπτη συμπεριφορά παράγοντας λάνθασμένη υλοποιήση του συστήματος.

Ευχρηστία

Κυριότερη επιθυμία κάθε προγραμματιστή, είναι ο χρήστης να μην δυσκολεύεται να κατανοήσει την διεπιφάνεια αλληλεπίδρασης και να μπορεί να αντιληφθεί την λειτουργία κάθε επιλογής στη διεπιφάνεια εργασίας. Γι' αυτό, βασική αρχή των παραθύρων γραφικού περιβάλλοντος είναι η απλότητα και η τήρηση των αρχών της HCI (Αλληλεπίδραση Ανθρώπου – Υπολογιστή). Χρησιμοποιώντας αυτοπροσδιορισμένα κουμπιά, λίστες επιλογής και παράθυρα μηνυμάτων, ο χρήστης μπορεί να αλληλεπιδράσει σε πραγματικό χρόνο με το σύστημα.

Συντηρησιμότητα

Η τεκμηρίωση των φάσεων ανάπτυξης του συστήματος όπως επίσης και η καταγραφή δυσκολιών ή και λαθών που αντιμετωπίστηκαν θα διευκολύνουν την συντήρηση.

Ακόμα, ο κώδικας θα συνοδεύεται με σχόλια και επεξηγήσεις και θα είναι όσο το δυνατόν πιο εύληπτος από κάποιον τρίτο, ούτως ώστε να είναι το σύστημα συντηρήσιμο στο μέλλον ακόμα και από μελλοντικούς φοιτητές που μπορεί να ασχοληθούν με το αντικείμενο αυτό.

Μεταφερσιμότητα

Το εργαλείο είναι γραμμένο σε Java γι' αυτό μπορεί να τρέξει σε οποιοδήποτε υπολογιστή, με οποιοδήποτε λειτουργικό σύστημα, αφού όπως αναφέρθηκε πιο πάνω για την εκτέλεση του είναι υπεύθυνη η εικονική μηχανή, το μόνο που χρειάζεται είναι η εγκατάσταση του JRE.

Δυνατότητα επαναχρησιμοποίησης

Για διαφορετικές λειτουργίες του συστήματος, υπάρχει διαφορετική κλάση που είναι υπεύθυνη γι' αυτές και δεν επηρεάζεται από τις υπόλοιπες. Γι' αυτό, οι διαφορετικές κλάσεις, όπως για παράδειγμα η κλάση που χρησιμοποιείται για την εισαγωγή των βασικών τύπων και την παραγωγή της κλάσης τους, μπορεί να χρησιμοποιηθεί και σε επόμενα εργαλεία που σχετίζονται με άλγεβρες διεργασιών. Γενικότερα, οι διαφορετικές κλάσεις του συστήματος μπορούν να χρησιμοποιηθούν σε επόμενες εργασίες που ασχολούνται με τον π-λογισμό με ομάδες.

Προστασία προσωπικών δεδομένων

Ο χρήστης δεν εισάγει προσωπικά δεδομένα στο σύστημα. Τα μόνα προσωπικά του αρχεία είναι τα XML διατύπωσης του συστήματος. Αυτά λοιπόν, με το τέλος της εκτέλεσης του προγράμματος, δεν αποθηκεύονται κάπου από το εργαλείου ούτε βέβαια δημοσιεύονται. Οπότε ο χρήστης δεν χρειάζεται να έχει κάποια ανησυχία όσον αφορά το κατά πόσον τα προσωπικά του δεδομένα είναι προστατευμένα, αφού ούτως ή άλλως το εργαλείο τρέχει στον προσωπικό του υπολογιστή.

Ασφάλεια

Το σύστημα, παρέχει ασφάλεια τόσο στην αρχική λειτουργία στην εισαγωγή των βασικών τύπων όπου οι επιλογές είναι περιορισμένες μέσω της διεπιφάνειας χρήστη, όσο και στις επόμενες αφού ελέγχονται τα αρχεία εισόδου.
Επίσης, το πρόγραμμα τρέχει τοπικά και δεν χρησιμοποιεί, ούτε συνδέεται στο Διαδίκτυο ή με άλλο σύστημα, οπότε η πιθανότητα κακόβουλης πρόσβασης είναι αμελητέα.

Έλεγχος

Το σύστημα τρέχει στην προσωπική μηχανή του κάθε χρήστη τοπικά, οπότε δεν κρίνεται απαραίτητη η ύπαρξη αυθεντικοποίησης χρήστη. Είναι κάτι που θα ήταν κουραστικό για το χρήστη και που δεν προσφέρει κάποια ουσιαστικότερη και απαραίτητη ασφάλεια.

3.9 Σχεδιασμός Συστήματος με UML

Ορισμός UML:

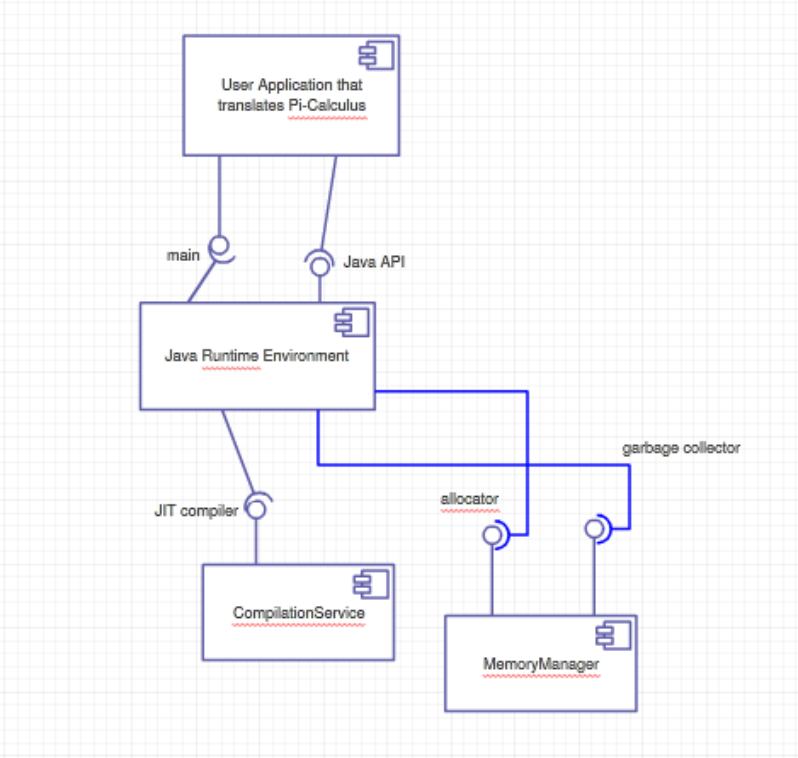
Η **Unified Modeling Language** (**UML**, *Ενοποιημένη Γλώσσα Μοντελοποίησης*) πλέον είναι η πρότυπη γλώσσα μοντελοποίησης στη μηχανική λογισμικού. Χρησιμοποιείται για τη γραφική απεικόνιση, προσδιορισμό, κατασκευή και τεκμηρίωση των στοιχείων ενός συστήματος λογισμικού. Μπορεί να χρησιμοποιηθεί σε διάφορες φάσεις ανάπτυξης, από την ανάλυση απαιτήσεων ως τον έλεγχο ενός ολοκληρωμένου συστήματος. Αποτελείται από ένα σύνολο προσυμφωνημένων όρων, συμβόλων και διαγραμμάτων που επιτρέπουν:

- την εμφάνιση των ορίων ενός συστήματος και των βασικών λειτουργιών του, χρησιμοποιώντας «περιπτώσεις χρήσης» (use-cases) και «actors».
- την επεξήγηση της πραγματοποίησης των περιπτώσεων χρήσης με «διαγράμματα αλληλεπίδρασης».
- την αναπαράσταση μιας στατικής δομής ενός συστήματος χρησιμοποιώντας «διαγράμματα κλάσεων».
- τη μοντελοποίηση της συμπεριφοράς των αντικειμένων με «διαγράμματα καταστάσεων».
- τη μοντελοποίηση της εργασιακής ροής με «διαγράμματα δραστηριοτήτων»
- την αποκάλυψη της υλοποίησης της αρχιτεκτονικής με «διαγράμματα συστατικών» και «ανάπτυξης».
- την επέκταση της λειτουργικότητας με «στερεότυπα».

Σκοπός του παρόντος υποκεφαλαίου είναι να δοθεί το αυστηρό σχεδιαστικό πλαίσιο που ακολούθησε η υλοποίηση του εργαλείου της εργασίας που παράγει τον κώδικα για τα συστήματα π-λογισμού. Ο σχεδιασμός του συστήματος με διαγράμματα UML είναι

ένα πολύ σημαντικό κομμάτι στον κύκλο ζωής ενός έργου καθώς αποτελεί πολύτιμο κομμάτι τεκμηρίωσης για τυχόν μελλοντική επέκταση του συστήματος ή αναβάθμισή του. Μπορεί να εξοικονομήσει χρόνο στους προγραμματιστές που ίσως ασχοληθούν στο μέλλον με τη συντήρηση του εργαλείου, αφού προσφέρει μία πιο αφαιρετική αλλά ακριβή περιγραφή, χωρίς να μπαίνει στις λεπτομέρειες του κώδικα.

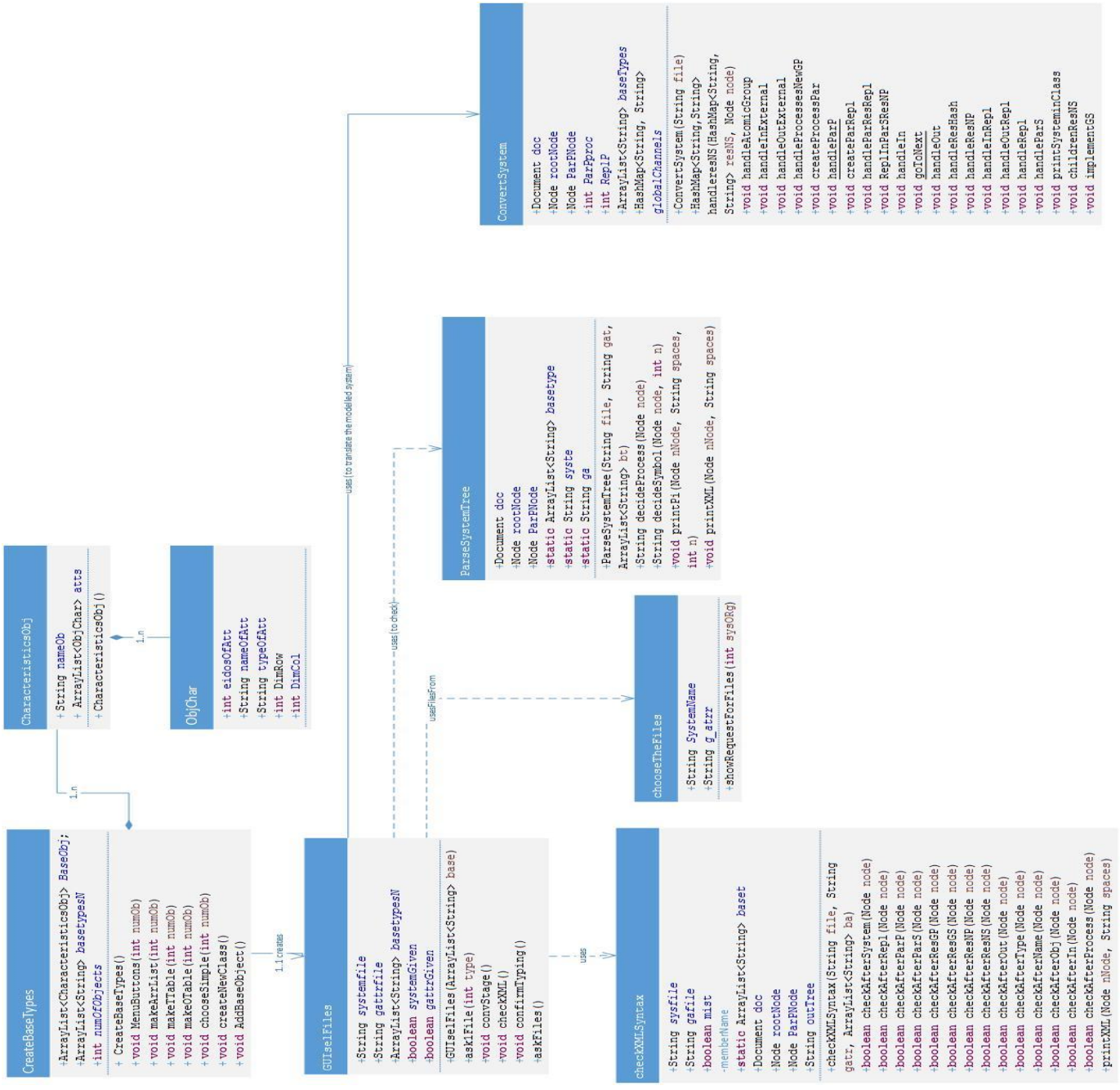
3.9.1 Διάγραμμα Συστατικών



Το πιο πάνω διάγραμμα απεικονίζει τα απαιτούμενα συστατικά στοιχεία για τη λειτουργία του εργαλείου της παρούσας εργασίας. Όπως αναφέρθηκε και σε προηγούμενο στάδιο το σύστημα που υλοποιήθηκε δεν επικοινωνεί ούτε με κάποιον εξυπηρετητή ούτε με κάποια βάση δεδομένων, αλλά τρέχει τοπικά. Τα συστατικά λοιπόν που απαιτούνται είναι πρώτα απ' όλα η εφαρμογή και το Java Runtime Environment που είναι αναγκαίο για να τρέχει ο χρήστης στον υπολογιστή του το πρόγραμμα Java. Συστατικά μέρη του JRE είναι αυτά που συνθέτουν τον Java Virtual Machine, το οποίο είναι υπεύθυνο για την εκτέλεση του bytecode που παράγει το JRE. Η εικονική μνήμη Java, διαχειρίζεται τη μνήμη (garbage collector και allocator). Μέρος

του JVM είναι και ο μεταγλωττιστής JIT (just-in-time) ο οποίος ασχολείται με μεταγλώττιση που γίνεται κατά τη φάση εκτέλεσης του προγράμματος.

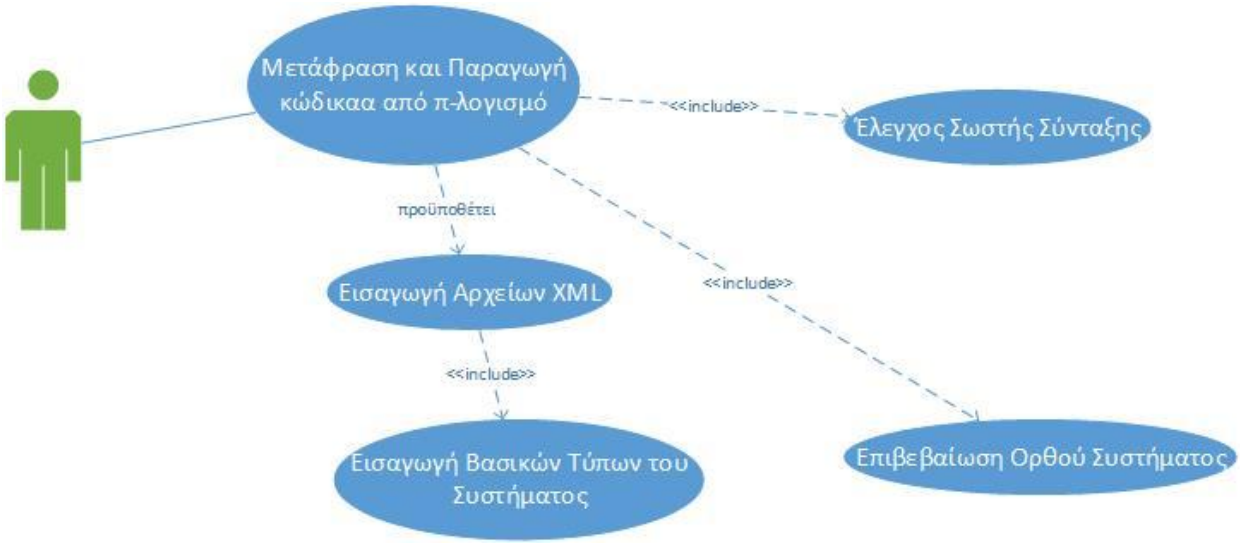
3.9.2 Διαγράμματα κλάσσεων



Στο πιο πάνω διάγραμμα βλέπουμε τις αλληλοεξαρτήσεις μεταξύ των διαφορετικών κλάσεων. Η κλαση ObjChar δημιουργεί αντικείμενα που αφορούν το κάθε ένα διαφορετικό όρισμα ενός Βασικού Τύπου. Η κλάση CharacteristicsObj δημιουργεί αντικείμενα για τον κάθε ένα διαφορετικό βασικό τύπο και περιέχει μία λίστα με

ObjChar αντικείμενα, δηλαδή μία λίστα με τα χαρακτηριστικά των ορισμάτων του βασικού τύπου. Η κλάση CreateBaseTypes παράγει τη διεπιφάνεια αλληλεπίδρασης για την εισαγωγή Βασικών Τύπων. Για κάθε βασικό τύπο που εισάγει ο χρήστης, δημιουργείται νέο αντικείμενο CharacteristicsObj που προστίθεται μέσα στη λίστα BaseObj η οποία χρησιμεύει στην καταχώρηση των βασικών τύπων. Η κλάση αυτή επίσης δημιουργεί ένα αντικείμενο της κλάσης GUIselFiles. Η GUIselFiles είναι υπεύθυνη για την εισαγωγή των απαραίτητων αρχείων από τον χρήστη. Μετά την εισαγωγή αρχείων, η GUIselFiles δημιουργεί αντικείμενα CheckXMLSyntax και ParseSystemTree που είναι υπεύθυνα για τον έλεγχο της σύνταξης των αρχείων XML και την παραγωγή διατύπωσης σε π-λογισμό για επιβεβαίωση από τον χρήστη. Ανάλογα με την κατάσταση των αντικειμένων των δύο αυτών κλάσσεων, καθορίζεται και η κατάσταση της GUIselFiles. Η GUIselFiles βέβαια, δημιουργεί και το κατάλληλο αντικείμενο για την υλοποίηση της μετάφρασης.

3.9.3 Διαγράμματα περιπτώσεων χρήσης

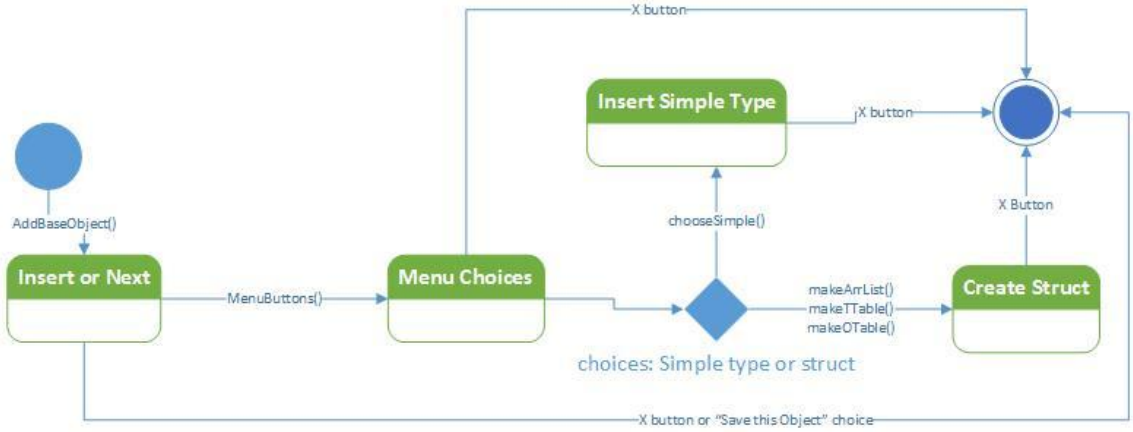


Σημαντικό επίσης διάγραμμα στην τεκμηρίωση απαιτήσεων και λειτουργιών του συστήματος είναι το διάγραμμα περιπτώσεων χρήσης. Η βασική λειτουργία που επιθυμεί ο χρήστης να εκτελέσει με τη χρήση του συστήματος είναι φυσικά η μετάφραση ενός μοντέλου συστήματος π-λογισμού σε κώδικα γλώσσας Java. Για να πραγματοποιηθεί όμως η λειτουργία αυτή υπάρχουν άλλες λειτουργίες που πρέπει πρώτα να ολοκληρωθούν. Πρέπει να Εισάγει τα αρχεία XML για να ελεχθεί η σύνταξή τους αλλά και να επιβεβαιώσει ο ίδιος ο χρήστης ότι επιθυμεί να μεταφράσει το μοντελοποιημένο σύστημα. Ακόμα όμως και πριν την εισαγωγή των αρχείων XML ο

χρήστης αλληλεπιδρώντας με μία διεπιφάνεια, πρέπει να εισάγει και να καθορίσει τους βασικούς τύπους του συστήματος.

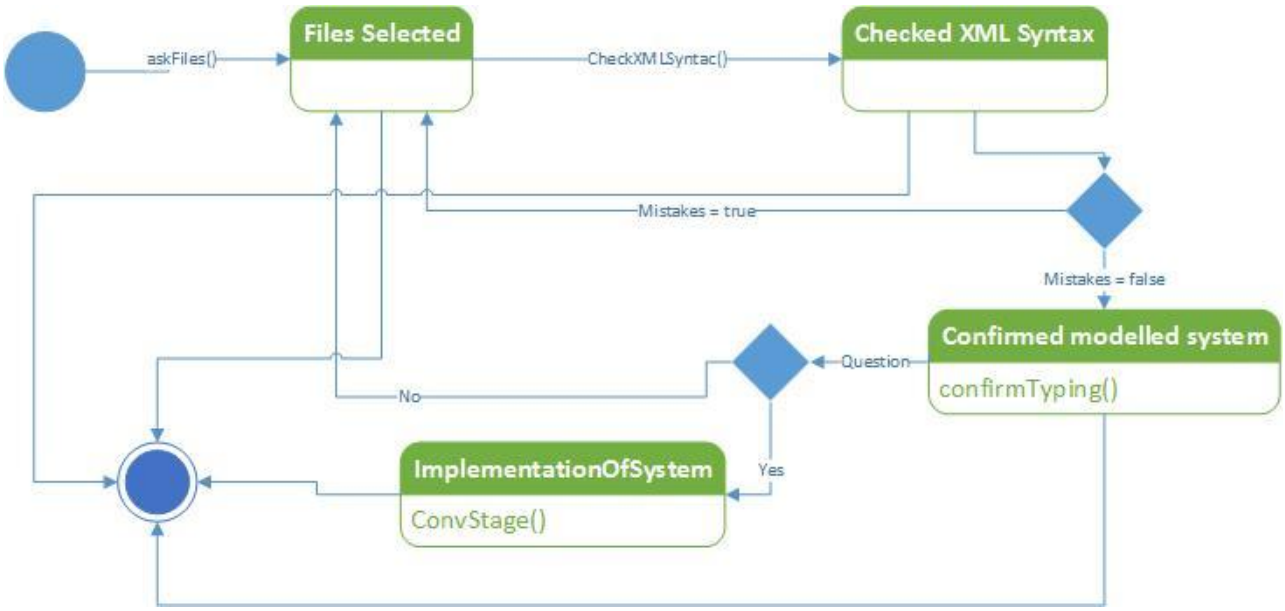
3.9.4 Διαγράμματα καταστάσεων

Διάγραμμα καταστάσεων για την κλάση δημιουργίας Βασικών Τύπων:



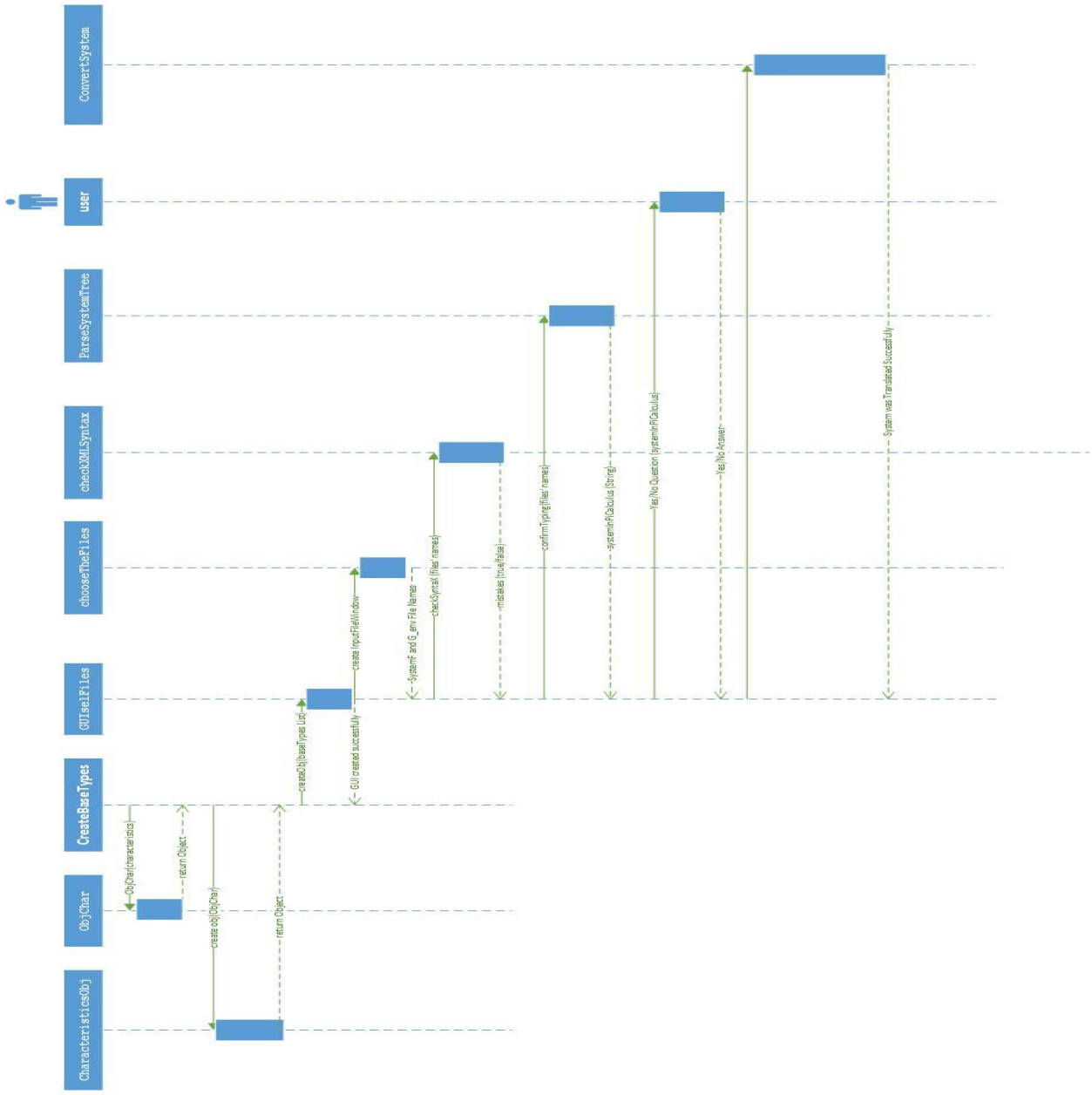
Η κλάση CreateBaseTypes είναι υπεύθυνη για την εισαγωγή βασικών τύπων και ένα αντικείμενο της κλάσης αυτής μπορεί να περάσει από διάφορες καταστάσεις. Αρχικά βρίσκεται στην κατάσταση όπου ζητείται από το χρήστη να επιλέξει αν θα εισάγει νέο βασικό τύπο ή αν θα προχωρήσει στο επόμενο βήμα. Αν προχωρήσει στο επόμενο βήμα ή επιλέξει την έξοδο, τότε το αντικείμενο της κλάσης αυτής φτάνει στην τερματική του κατάσταση. Ειδάλλως, προχωράει στην επόμενη κατάσταση όπου δίνει επιλογές στον χρήστη όσον αφορά τα ορίσματα του τρέχοντος βασικού τύπου. Η επόμενη κατάσταση ορίζεται ανάλογα την επιλογή που έχει δόσει ο χρήστης, είτε να εισάγει όρισμα απλού τύπου είται να εισάγει κάποια δομή (πίνακα ή λίστα). Κάθε κατάσταση μπορεί να οδηγηθεί σε τερματισμό με το κουμπί εξόδου.

Διάγραμμα καταστάσεων για την κλάσση Εισαγωγής και μετάφρασης αρχείων XML



Σε αυτό το σημείο έχουμε το διάγραμμα καταστάσεων αντικειμένου της κλάσης GUIselFiles η οποία είναι υπεύθυνη για την εισαγωγή, έλεγχο και υλοποίηση των XML αρχείων, μέσω της χρήσης άλλων κλάσεων. Η πρώτη κατάσταση είναι αυτή στην οποία ο χρήστης επιλέγει τα αρχεία που θα εισάγει. Έπειτα προχωράμε στην κατάσταση όπου γίνεται ο έλεγχος της σύνταξης του XML. Αν ο έλεγχος εντοπίσει λάθη, τότε επιστρέφουμε στην κατάσταση εισαγωγής νέων αρχείων. Διαφορετικά, προχωράμε στην επόμενη κατάσταση που αφορά την επιβεβαίωση της διατύπωσης π-λογισμού από τον χρήστη. Αν ο χρήστης επιβεβαιώσει, τότε προχωράμε στην υλοποίηση του συστήματος. Διαφορετικά, μπορεί να τερματίσει την κλάση ή να επιστρέψει στην κατάσταση εισαγωγής αρχείων. Κάθε κατάσταση μπορεί να οδηγηθεί σε τερματισμό με το κουμπί εξόδου.

3.9.3 Διαγράμματα Ακολουθίας



Πιο πάνω βλέπουμε τις διαδικασίες που ακολουθούνται για την ολοκλήρωση όλων των λειτουργιών οι οποίες περιγράφηκαν και στα προηγούμενα διαγράμματα. Στο τρέχον διάγραμα βλέπουμε την αλληλεπίδραση μεταξύ των κλάσεων ακολουθιακά, σε σχέση με το χρόνο.

Κεφάλαιο 4

Υλοποιήση Εργαλείου

4.1 Εισαγωγή 53
4.2 Κανόνες Μετάφρασης του λογισμού σε Java 54
4.2.1 Επίπεδο Συστήματος 56
4.2.2 Επίπεδο Διεργασίας 68
4.3 Ορθότητα Μεταφρασμένου Προγράμματος: 77
4.4 Εγχειρίδιο Χρήστη 85
4.4.1 Οδηγίες συγγραφής XML αρχείου 86
4.4.2 Οδηγίες χρήσης διεπιφάνειας αλληλεπίδρασης 90

4.1 Εισαγωγή

Σε αυτό σημείο είμαστε στη φάση υλοποίησης του συστήματος, όπου οι λειτουργικές απαιτήσεις θα εφαρμοστούν σε πρόγραμμα. Δίνεται έμφαση αρχικά στην περιγραφή της μεθόδου μετάφρασης ενός μοντελοποιημένου συστήματος σε πρόγραμμα Java. Είναι σημαντικό, να δοθεί μία τυπική σύνταξη του κάθε κανόνα του λογισμού σε Java, ούτως ώστε να μπορεί μετέπειτα να αποδειχτεί η ορθότητα της μετάφρασης που είναι το πιο σημαντικό κομμάτι.

Έπειτα γίνεται μία αφαιρετική παρουσίαση της ροής του προγράμματος που αποτελείται από διάφορα συστατικά στοιχεία για την κάθε λειτουργία. Ακολούθως, είναι σημαντικό να δοθεί ένα εγχειρίδιο οδηγιών για το χρήστη, όσον αφορά τη συγγραφή των XML αρχείων όσο και για τη γενικότερη χρήση του εργαλείου.

Δένδρο XML – Java Dom Parser

Ο λόγος για τον οποίο ζητείται από το χρήστη να εισάγει το μοντελοποιημένο σύστημα σε μία τυπική διατύπωση μορφής, είναι γιατί με το εργαλείο Java Dom Parser μπορεί

εύκολα να γίνει ανάγνωση του συστήματος και ο καθορισμός των ιεραρχιών ομάδων. Όταν ο χρήστης εισάγει το σύστημά του σε XML μορφή, με τη χρήση του JDP θα καταγραφεί στη μνήμη ως ένα δέντρο ιεραρχίας, ανάλογα με τους σημαντήρες της XML. Επομένως, για τον έλεγχο της σύνταξης, την μεταφορά του σε διατύπωση π-λογισμού αλλά και η μετάφραση, θα γίνεται με κατά βάθος αναζήτηση στο δέντρο (DFS), χρησιμοποιώντας αναδρομή.

4.2 Κανόνες Μετάφρασης του λογισμού σε Java

Το πρόγραμμα δέχεται την διατύπωση του Επιπέδου Συστήματος και Διεργασίας σε xml μορφή και έπειτα παράγει τις απαραίτητες κλάσεις οι οποίες αναλαμβάνουν να κάνουν αρχικοποιήσουν τις διεργασίες που θα τρέχουν παράλληλα. Το εργαλείο σε αυτό το σημείο αναλαμβάνει να τηρήσει την πολιτική ιδιωτικότητας που περιγράφεται από τη διατύπωση του συστήματος. Δημιουργεί τα νέα ονόματα και στέλνει στην κάθε διεργασία μόνο αυτά στα οποία μπορεί να έχει πρόσβαση.

Στα επόμενα υποκεφάλαια θα περιγραφεί το πως το σύστημα διαβάζει το XML αρχείο και αποφασίζει πώς να διαχειριστεί τον κάθε κανόνα. Επίσης, δίνεται η μετάφραση του κάθε όρου της σύνταξης των δύο επιπέδων και εξηγώντας τους λόγους. Όμως, για πιο ξεκάθαρη κατανόηση, θεωρώ σκόπιμο να δώσω ένα σκελετό της μετάφρασης του λογισμού σε Java, πριν εξηγηθούν λεπτομερέστερα.

Αρχικά θα αναφερθούν οι κλάσεις της βιβλιοθήκης JCSP που χρησιμοποιούνται στη μετάφραση και σε τι χρησιμεύει η κάθε μία:

- CSProcess: ένα αντικείμενο αυτής της κλάσης αντιπροσωπεύει μία διεργασία P του λογισμού και εκκινούμε μία διεργασία P με τη συνάρτηση run() της κλάσης
- Parallel: ένα αντικείμενο αυτής της κλάσης περιέχει ένα πίνακα με αντικείμενα CSProcess και μπορεί να τα εκτελέσει παράλληλα και εκκινούμε μία παράλληλη σύνθεση με τη συνάρτηση run() της κλάσης
- One2OneChannel: ένα αντικείμενο αυτής της κλάσης αποτελεί ένα κανάλι το οποίο μπορεί να αλληλεπιδρά μόνο με ένα άλλο κανάλι και όχι περισσότερα. Μπορούμε να πραγματοποιήσουμε ενέργεια εξόδου σε ένα κανάλι με τη συνάρτηση out().write(x) της κλάσης και ενέργεια εισόδου με την in().read().

- ProcessManager: ένα αντικείμενο αυτής της κλάσης αντιπροσωπεύει μία διεργασία που εκτελείται πατάλληλα με την τρέχουσα. Χρησιμοποιείται για την εκκίνηση μίας διεργασίας ενός τελεστή αντιγραφής. Εκκινεί παράλληλα με τη συνάρτηση start().

Πιο κάτω δίνεται εν συντομία η αντίστοιχη μετάφραση κάθε κανόνα σε Java, ούτως ώστε να επεξηγηθεί σε βάθος στα επόμενα υποκεφάλαια. Για τον κάθε κανόνα, δίνεται, στο πρώτο κελί από τα δυο, ο κώδικας που θα παραχθεί στην τρέχουσα κλάση της ιεραρχίας, δηλαδή στην κλάση που βρίσκεται το πρόγραμμα όταν συναντά τον κανόνα στο XML αρχείο. Αναφέρεται επίσης ποιες κλάσεις δημιουργούνται όταν συναντούμε κάθε κανόνα.

Επίπεδο Διεργασίας

x(y:T).P	y = (ChannelValue) x.channel.in().read(); Δεν παράγονται νέες κλάσεις
x<z>.P	x.channel.out().write(z); Δεν παράγονται νέες κλάσεις
(v a:T)P	ChannelValue a = **new** ChannelValue(); a.type = "[a_Type]"; Δεν παράγονται νέες κλάσεις
P1|P2	ParallelPrroc1 P1 = **new** ParallelProc1 ([channel1],..[channeln]); ParallelProc1 P2 = **new** ParallelProc2 ([channel1],..[channeln]); CSProcess[] parParts = **new** CSProcess[]{P1, P2}; Parallel par = **new** Parallel(parParts); par.run(); **Παράγονται και οι κλάσεις:** ParallelPrroc1 για την υλοποίηση της P1 και η ParallelPrroc2 για την υλοποίσης P2

!P	**while**(**true**){
	[in action ή out_action ανάλογα με το πρώτο prefix l του process P]
	Repl1 rep = **new** Repl1([channel1],..[channeln]);
	ProcessManager manager = **new** ProcessManager(rep);
	manager.start();

	}
	Παράγεται και η κλάση: Repl[counterRepl] που υλοποιεί τη διεργασία P’

Επίπεδο Συστήματος

(v G) S	SystemG G = **new** SystemG ([channel1],..[channeln]);
	Δεν παράγονται νέες κλάσεις
(v a:T) S	ChannelValue a = **new** ChannelValue(); a.type = “T” ;
	Δεν παράγονται νέες κλάσεις
S1|S2	SystemS1 S1 = **new** SystemS1([channel1],..[channeln]); SystemS1 S1 = **new** SystemS1 ([channel1],..[channeln]); CSProcess[] parParts = **new** CSProcess[]{S1,S2}; Parallel par = **new** Parallel(parParts); par.run();
	Παράγονται και οι κλάσεις: SystemS1 και SystemS2 για το σύστημα S1 και S2 αντίστοιχα
(v G) P	ProcessG G = **new** ProcessG([channel1],…,[channeln]); CSProcess[] Gprocesses = **new** CSProcess[]{G}; atomicGroup [G: groupName]= **new** atomicGroup(Gprocesses , "G");
	Παράγεται και η κλάση: ProcessG για την υλοποίηση του P

4.2.1 Επίπεδο Συστήματος

Η σύνταξη στο επίπεδο συστήματος, περιλαμβάνει τη δημιουργεία ομάδων (είτε ομάδες διεργασιών είτε ομάδες συστήματος), τη δημιουργία νέων ονομάτων σε επίπεδο συστήματος και τη σύνθεση παράλληλων συστημάτων:

S::= (v G) P | (v G) S | (v a:T) S | S1|S2 | 0

Το εργαλείο, βασίζει τη διαδικασία μετάφρασης σε μελέτη περιπτώσεων όσον αφορά το χαμηλότερο επίπεδο του συστήματος. Αν η διατύπωση του συστήματος ξεκινά με (v G) S τότε η κλάση που θα εκκινεί όλο το σύστημα θα είναι η SystemG. Αν όμως η διατύπωση του συστήματος ξεκινά με S1|S2 ή (v a:T) S, τότε το χαμηλότερο επίπεδο του Pi Calculus προγράμματος, δεν περιλαμβάνει κάποια συγκεκριμμένη ομάδα συστήματος (system group) η οποία θα μπορούσε να εκκινήσει τα υπόλοιπα υποσυστήματα παράλληλα. Οπότε σε αυτή την περίπτωση, δημιουργείται η κλάση SystemBoot.java. Αν για παράδειγμα έχουμε στο χαμηλότερο επίπεδο την διατύπωση S1 | S2 τότε η κλάση SystemBoot.java θα εκκινήσει τα συστήματα S1 και S2 παράλληλα. Αντίθετα, στην περίπτωση που το χαμηλότερο επίπεδο του συστήματος, δημιουργεί ομάδες συστήματος, για παράδειγμα (v ETP)(S1|S2), τότε δεν θα χρειαστεί η δημιουργία της κλάσης SystemBoot.java για να ξεκινήσουν το S1 και S2, καθώς αυτόν τον ρόλο θα τον αναλάβει η κλάση SystemETP.java.

Αυτή μελέτη περιπτώσεων θα μπορούσε να αποφευχθεί με το να διασχίζαμε το δέντρο xml αναδρομικά και σε οποιοδήποτε κανόνα της σύνταξης να δημιουργούνταν νέα κλάση, η οποία θα εκκινούσε από τον πρόγονό της στην ιεραρχία. Δηλαδή, στο πιο πάνω παράδειγμα (v ETP)(S1|S2), η SystemETP.java κλάση δεν θα εκκινούσε απευθείας τα S1 και S2 παράλληλα, αλλά λόγω της αναδρομής που σε κάθε κανόνα/ενέργεια της σύνταξης θα δημιουργούσε νέα κλάση, η SystemETP.java θα εκκινούσε μία ενδιάμεση της παράλληλης σύνθεσης και αυτή η ενδιάμεση κλάση θα εκκινούσε τα S1 και S2 παράλληλα. Με αυτόν τον τρόπο, θα μπορούσε να γίνει η μετάφραση διασχίζοντας αναδρομικά το δέντρο που θα παραχθεί από το xml, χωρίς να χρειαστεί η μελέτη περιπτώσεων. Ταυτόχρονα όμως, θα δημιουργούνταν ενδιάμεσες κλάσεις, οι οποίες σε μεγάλης έκτασης προγράμματα, θα διόγκωναν πολύ τον συνολικό αριθμό κλάσεων, δυσκολεύοντας τον χρήστη να μελετήσει το πρόγραμμα που θα παράγεται.

Ροή προγράμματος:

Το πρόγραμμα δέχεται το xml αρχείο. Έπειτα ξεκινώντας από τη ρίζα, διασχίζει το παραγόμενο δέντρο του xml. Στις περιπτώσεις που στη ρίζα του δέντρου έχουμε τα (v G) P ή (v a:T) S ή S1|S2, τότε η βασική κλάση που θα δημιουργηθεί θα είναι η

SystemBoot.java. Στην περίπτωση που στη ρίζα έχουμε (v G) S τότε η βασική κλάση που θα δημιουργηθεί θα είναι η SystemG.java.
Έπειτα, δημιουργείται νέα κλάση για κάθε υποσύστημα που βρίσκεται στο δέντρο. Το κάθε υποσύστημα ορίζει τα απαραίτητα στοιχεία του, όπως είναι τα νέα ονόματα/κανάλια και είναι υπεύθυνο για τη διάδοση των νέων ονομάτων στις διεργασίες που έχουν πρόσβαση σε αυτά. Αυτό επιτυγχάνεται αναζητώντας αναδρομικά στους προγόνους των υποσυστημάτων/διεργασιών. Αν τα υποσυστήματα/διεργασίες έχουν τον πρόγονο resNS στο δέντρο, τότε σημαίνει ότι έχουν πρόσβαση στα ονόματα/κανάλια που ορίζει το συγκεκριμμένο resNS. Το κάθε υποσύστημα/διεργασία τίθεται σε εκτέλεση από το σύστημα που είναι πιο άμεσος πρόγονός του.

Πιο κάτω δίνεται το πρωτότυπο του κώδικα κλάσεων σε αντιστοιχία με τα δομικά στοιχεία της Pi Calculus (Επίπεδο Συστήματος) και την αντίστοιχη διατύπωσή τους σε XML

1. (v G) S : Δημιουργία νέου γκρουπ συστήματος. Ο χρήστης θα πρέπει να το διατυπώσει σε xml στην πιο κάτω μορφή:

```
(vG)S : <resGS>
          <group>G</group>
          < system >S</ system>
        </resGS>
```

Πρωτότυπο κώδικα στην κλάση προγόνου απ' όπου θα εκκινείται το σύστημα:

System[G: groupName] [G: groupName] = **new** System[G: groupName] ([channel1],..[channeln]);
[G: groupName]: είναι το όνομα του γκρουπ του συστήματος.
[channel1],..[channeln]: τα κανάλια που ορίστηκαν σε υψηλότερο επίπεδο αλλά το σύστημα μπορεί να αποκτήσει πρόσβαση

Πρωτότυπο κώδικα της κλάσης του νέου συστήματος:

```
import org.jcsp.lang.*;
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
```

Αναγκαίες βιβλιοθήκες για να τρέχει ο κώδικας.

```
public class System[G: groupName]  {
      public String nameSystem = "System[G: groupName]";

      public ChannelValue channel1 ;
            .
            .              .
      public ChannelValue channeln ;
```

Δηλώνονται τα κανάλια που έχουν οριστεί πιο πάνω στην ιεραρχία και το σύστημα μπορεί να έχει πρόσβαση σε αυτά.

```
 public System[G: groupName]( ChannelValue [channel1],…, ChannelValue
[channeln]){
            [channel1] = new ChannelValue();
            [channel1].type = [channel1Type];
                  .
                  .
            [channeln] = new ChannelValue();
            [channeln].type = [channelnType];

      }

      public void run() {
                  .
                  .
                  .
            }
      }
```

Constructor της κλάσης. Δέχεται σαν είσοδο τις αναφορές των καναλιών που έχει πρόσβαση από την κλάση που εκκινεί το σύστημα αυτό και τα συνδέει με τις εσωτερικές αναφορές των καναλιών, ορίζοντας ακολούθως και τον τύπο τους.

Στη συνάρτηση εγγράφονται οι διεργασίες που θα περιέχονται στη συγκεκριμμένη ομάδα.

2. (v a:T) S: Δημιουργία νέου καναλιού/ονόματος στο οποίο θα έχει πρόσβαση το μετέπειτα σύστημα S. Ο χρήστης θα πρέπει να το διατυπώσει σε xml στην πιο κάτω μορφή:

```
(va:T)S : <resNS >
          <name>a</name>
          <type>T</type>
          < system >S</ system>
        </resNS >
```

Πρωτότυπο κώδικα δημιουργίας του καναλιού:

Περίπτωση στην οποία ο τύπος του καναλιού δεν ανήκει στους βασικούς τύπους:

```
ChannelValue [a: channelname] = new ChannelValue();
[a: channelname].type = [T: type of channel] ;
```

Περίπτωση στην οποία ο τύπος του καναλιού ανήκει στους βασικούς τύπους:

```
[T: type of channel] [a: channelname] = new [T: type of channel] ();
```

3. S1|S2: Δύο συστήματα που τρέχουν παράλληλα. Ο χρήστης θα πρέπει να το διατυπώσει σε xml στην πιο κάτω μορφή:

```
S1|S2: <parS>
          <system>S1</ system>
          < system>S2</ system>
        </parS>
```

Πρωτότυπο κώδικα δημιουργίας της δομής για παράλληλη εκκίνηση συστημάτων:

```
System[G1: groupName] [G1: groupName] = new System[G1: groupName]
([channel1],..[channeln]);
        .
        .
System[Gn: groupName] [Gn: groupName] = new System[Gn: groupName]
([channel1],..[channeln]);
```

} Δηλωση παράλληλων συστημάτων

Τα συστήματα μπαίνουν σε ένα κοινό πίνακα:

```
CSProcess[] parParts = new CSProcess[]{[G1: groupName],…, [Gn: groupName]};
```

Ορίζουμε ότι τα συστήματα στον πίνακα θα εκτελεστούν παράλληλα:

```
Parallel par = new Parallel(parParts);
```

Τρέχουμε τα παράλληλα συστήματα:

```
par.run();
```

4. (v G) P: Δημιουργία ενός νέας ομάδας διεργασίας. Ο χρήστης θα πρέπει να το διατυπώσει σε xml στην πιο κάτω μορφή:

```
(vG)P : <resGP>
          <group>G</group>
          < process >P</ process>
        </resGP>
```

Πρωτότυπο κώδικα στην κλάση προγόνου απ' όπου θα εκκινείται η διεργασία:

```
Process[G: groupName]  [G: groupName]  = new Process[G: groupName]
([channel1],…,[channeln]);
CSProcess[][G: groupName]processes = new CSProcess[]{[G: groupName]};
atomicGroup [G: groupName]= new atomicGroup(OBEprocesses, "[G: groupName]");
```

Δηλώνεται η νέα διεργασία λαμβάνοντας ταυτόχρονα τις αναφορές των καναλιών στα οποία δικαιούται να έχει πρόσβαση (channel1-n). Η κλάση atomicGroup είναι ήδη υλοποιημένη και χρησιμοποείται αυτούσια χωρίς χρειάζεται οποιαδήποτε αλλαγή για κάθε πρόγραμμα PiCalculus. Η atomicGroup θέτει εν ενεργεία την ομάδα διεργασίας.

Πρωτότυπο κώδικα της κλάσης του νέου συστήματος:

```
import org.jcsp.lang.CSProcess;
import org.jcsp.lang.Parallel;
import org.jcsp.lang.ProcessManager;
```

} Αναγκαίες βιβλιοθήκες για να τρέχει ο κώδικας.

```
public class Process[G: groupName] implements CSProcess  {

        public ChannelValue channel1 ;
                .
                .                .
        public ChannelValue channeln ;
```

} Δηλώνονται τα κανάλια που έχουν οριστεί πιο πάνω στην ιεραρχία και το σύστημα μπορεί να έχει πρόσβαση σε αυτά

```
public Process[G: groupName] (ChannelValue [channel1],…, ChannelValue
[channeln]){
        [channel1] = new ChannelValue();
        [channel1].type = [channel1Type];
            .
            .
        [channeln] = new ChannelValue();
        [channeln].type = [channelnType];
    }
```

Constructor της κλάσης. Δέχεται σαν είσοδο τα instances των καναλιών που έχει πρόσβαση από την κλάση που εκκινεί τη διεργασία αυτή και τα συνδέει με τις εσωτερικές αναφορές των καναλιών, ορίζοντας ακολούθως και τον τύπο τους.

```
@Override
    public void run() {
        .
        .
        .
    }

}
```

Στη συνάρτηση εγγράφονται οι διεργασίες που θα περιέχονται στο συγκεκριμμένο

Παραδείγματα διατύπωσης συστήματος και μετάφρασης:

1ο Παράδειγμα:

(v ETP)(v spotcheck:Tsc)(v topa:Tpa)[(v PA) A | (v Car)((v read:Tr)((v OBE) O | (v GPS) L))]

Όπως φαίνεται, στο πιο πάνω παράδειγμα, το ETP είναι η ομάδα συστήματος πιο ψηλά στην ιεραρχία. Τα υποσυστήματα του ETP έχουν πρόσβαση στα κανάλια spotcheck και topa, τα οποία θα οριστούν από την SystemETP.java. Η κλάση αυτή επίσης θα θέτει σε εκτέλεση παράλληλα το την ομάδα διεργασίας PA και το υποσύστημα Car, στέλνοντάς

τους και τα 2 κανάλια spotcheck και topa. Η κλάση SystemCar.java, ορίζει το κανάλι read το οποίο θα το στείλει στις ομάδες διεργασίας OBE και GPS, που περιλαμβάνουν τις διεργασίες O και L, αντίστοιχα.

Επομένως, ο κώδικας που παράγεται είναι:

```
import org.jcsp.lang.*;
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
public class SystemETP {
	public String nameSystem = "SystemETP";
	public ChannelValue spotcheck ;
	public ChannelValue topa ;
		public SystemETP(){
			spotcheck = new ChannelValue();
			spotcheck.type = "Tsc" ;
			topa = new ChannelValue();
			topa.type = "Tpa" ;
		}
		public void run() {
			ProcessA A = new ProcessA(spotcheck, topa);
			CSProcess[] PAprocesses = new CSProcess[]{A};
			atomicGroup PA = new atomicGroup(PAprocesses,
"PA");
			SystemCar Car = new SystemCar(spotcheck, topa);
			CSProcess[] parParts = new CSProcess[]{PA,Car};
			Parallel par = new Parallel(parParts);
			par.run();
		}
	}


import org.jcsp.lang.*;
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
public class SystemCar {
	public String nameSystem = "SystemCar";
	public ChannelValue read ;
		public ChannelValue spotcheck ;
```

```
            public ChannelValue topa ;
            public SystemCar( ChannelValue spotcheck,
ChannelValue topa){
                  read = new ChannelValue();
                  read.type = "Tr" ;
                  this.spotcheck = spotcheck;
                  this.topa = topa;
            }
            public void run() {
                  ProcessO O = new ProcessO(spotcheck, topa,
read);
                  CSProcess[] OBEprocesses = new CSProcess[]{O};
                  atomicGroup OBE = new atomicGroup(OBEprocesses,
"OBE");
                  ProcessL L = new ProcessL(spotcheck, topa,
read);
                  CSProcess[] GPSprocesses = new CSProcess[]{L};
                  atomicGroup GPS = new atomicGroup(GPSprocesses,
"GPS");
                  CSProcess[] parParts = new
CSProcess[]{OBE,GPS};
                  Parallel par = new Parallel(parParts);
                  par.run();
            }
      }
```

2ο Παράδειγμα:

(v read: Tr)[(v st: Tsc)((v pl: Tx)((vOBE)(O|L))|(v MA) M)|(v channel: Tsc)(v ETP)(v spotcheck:Tpa)(v topa:Tpa)((v GPS) G | (v PA) A)]

Στην περίπτωση αυτή δημιουργείται ένα κανάλι read στο οποίο οποίο έχουν πρόσβαση δύο παράλληλα συστήματα. Το (v st: Tsc)((v pl: Tx)((vOBE)(O|L))|(v MA) M) και το)|(v channel: Tsc)(v ETP)(v spotcheck:Tpa)(v topa:Tpa)((v GPS) G | (v PA) A). Σε αυτήν την περίπτωση, θα δημιουργηθεί η κλάση SystemBoot.java, η οποία θα ορίσει τα ονόματα read, st, pl και channel. Το read θα το στείλει σε όλα τα υποσυστήματα. Το st θα το στείλει στις ομάδες διεργασίας OBE και MA ενώ το pl μόνο στο OBE. Το κανάλι channel θα το στείλει στο υποσύστημα ETP. Η SystemBoot.java εκκινεί παράλληλα λοιπόν τις ομάδες OBE, MA και ETP. Για την ομάδα ETP δημιουργείται η κλάση

SystemETP.java. Η κλάση αυτή ορίζει τα κανάλια spotcheck και topa στα οποία έχουν πρόσβαση οι ομάδρε διεργασιών GPS και PA τα οποία εκκινεί.

Επομένως, ο κώδικας που παράγεται είναι:

```
import org.jcsp.lang.*;
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
public class SystemBoot {
	public String nameSystem = "SystemBoot" ;
	public ChannelValue channel ;
	public ChannelValue st ;
	public ChannelValue read ;
	public ChannelValue pl ;
		public SystemBoot (){
			channel = new ChannelValue();
			channel.type = "Tr" ;
			st = new ChannelValue();
			st.type = "Tsc" ;
			read = new ChannelValue();
			read.type = "Tr" ;
			pl = new ChannelValue();
			pl.type = "Tx" ;
		}
		public void run() {
			ProcessO O = new ProcessO(pl, st, read);
			ProcessL L = new ProcessL(pl, st, read);
			CSProcess[] OBEprocesses = new CSProcess[]{O,
L};
			atomicGroup OBE = new atomicGroup(OBEprocesses,
"OBE");
			ProcessM M = new ProcessM(st, read);
			CSProcess[] MAprocesses = new CSProcess[]{M};
			atomicGroup MA = new atomicGroup(MAprocesses,
"MA");
			SystemETP ETP = new SystemETP(channel, read);
			CSProcess[] parParts = new
CSProcess[]{OBE,MA,ETP};
			Parallel par = new Parallel(parParts);
			par.run();
```

```
            }
}
import org.jcsp.lang.*;
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
public class SystemETP {
      public String nameSystem = "SystemETP";
      public ChannelValue spotcheck ;
      public ChannelValue topa ;
            public ChannelValue channel ;
            public ChannelValue read ;
            public SystemETP( ChannelValue channel, ChannelValue
read){
                  spotcheck = new ChannelValue();
                  spotcheck.type = "Tpa" ;
                  topa = new ChannelValue();
                  topa.type = "Tpa" ;
                  this.channel = channel;
                  this.read = read;
            }
            public void run() {
                  ProcessG G = new ProcessG(channel, topa, read,
spotcheck);
                  CSProcess[] GPSprocesses = new CSProcess[]{G};
                  atomicGroup GPS = new atomicGroup(GPSprocesses,
"GPS");
                  ProcessA A = new ProcessA(channel, topa, read,
spotcheck);
                  CSProcess[] PAprocesses = new CSProcess[]{A};
                  atomicGroup PA = new atomicGroup(PAprocesses,
"PA");
                  CSProcess[] parParts = new CSProcess[]{GPS,PA};
                  Parallel par = new Parallel(parParts);
                  par.run();
            }
      }
```

3ο Παράδειγμα:

(v st: Tsc)((v pl: Tx)((vOBE)(O|L))|(v MA) M)|(v channel: Tsc)(v ETP)(v spotcheck:Tpa)(v topa:Tpa)((v GPS) G | (v PA) A)

Το συγκεκριμμένο παράδειγμα είναι το ίδιο με το παράδειγμα 2, μόνο που δεν υπάρχει το κανάλι read που υπήρχε πιο πάνω. Δημιουργείται λοιπόν, η SystemBoot.java με τον ίδιο τρόπο, θέτοντας εν ενεργεία τα 2 παράλληλα υποσυστήματα χωρίς να τους στέλνει κάποιο κοινό κανάλι.

4ο Παράδειγμα:

(v ETP)(v spotcheck:Tsc)[(v read:Tr)((v OBE) O | (v LA) L) | (v topa:Tpa)(v LAV) P]

Στο συγκεκριμμένο παράδειγμα, έχουμε το σύστημα ETP στο οποίο εκτελούνται παράλληλα δύο υποσυστήματα, που μοιράζονται το κανάλι spotcheck. Στο πρώτο υποσύστημα εκτελούνται παράλληλα οι ομάδες διεργασιών OBE και LA με κοινό κανάλι το read και στο δεύτερο υποσύστημα η ομάδα διεργασίας LAV το οποίο είναι το μόνο που έχει πρόσβαση στο κανάλι topa.
Επομένως, η βασική κλάση που θέτει εν ενεργεία το σύστημα είναι η SystemETP.java η οποία δημιουργεί την αναφορά του καναλιού spotcheck και το διαδίδει και στις ομάδες διεργασιών που θα εκκινήσει. Στην κλάση αυτή επίσης, ορίζονται τα κανάλια read και topa, αλλά το read διαδίδεται μόνο στις ομάδες διεργασιών OBE και LA, ενώ το topa μόνο στην ομάδα LAV. Έπειτα τα οι ομάδες διεργασιών ξεκινούν παράλληλη εκτέλεση.

Επομένως, ο κώδικας που παράγεται είναι:

```
import org.jcsp.lang.*;
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
public class SystemETP {
	public String nameSystem = "SystemETP";
	public ChannelValue spotcheck ;
	public ChannelValue topa ;
	public ChannelValue read ;
		public SystemETP(){
```

```
                spotcheck = new ChannelValue();
                spotcheck.type = "Tsc" ;
                topa = new ChannelValue();
                topa.type = "Tpa" ;
                read = new ChannelValue();
                read.type = "Tr" ;
            }
            public void run() {
                ProcessO O = new ProcessO(spotcheck, read);
                CSProcess[] OBEprocesses = new CSProcess[]{O};
                atomicGroup OBE = new atomicGroup(OBEprocesses,
"OBE");
                ProcessL L = new ProcessL(spotcheck, read);
                CSProcess[] LAprocesses = new CSProcess[]{L};
                atomicGroup LA = new atomicGroup(LAprocesses,
"LA");
                ProcessP P = new ProcessP(spotcheck, topa);
                CSProcess[] LAVprocesses = new CSProcess[]{P};
                atomicGroup LAV = new atomicGroup(LAVprocesses,
"LAV");
                CSProcess[] parParts = new
CSProcess[]{OBE,LA,LAV};
                Parallel par = new Parallel(parParts);
                par.run();
            }
        }
```

4.2.2 Επίπεδο Διεργασίας

Μέχρι τώρα δόθηκε η τεχνική μετάφρασης για τους τελεστές που αφορούν το επίπεδο συστήματος. Τα παραδείγματα που αναφέρθηκαν δεν περιείχαν τη δημιουργία διεργασιών με ενέργειες επικοινωνίας, παρά μόνο τη δημιουργία και κατάλληλη σύνθεση συστημάτων και υποσυστημάτων. Η σύνταξη στο επίπεδο διεργασίας, περιλαμβάνει τη σήμανση μία πράξης εισόδου (*input prefix*), τη σήμανση μία πράξης εξόδου (*output prefix*), τη δημιουργία νέου δεσμευμένου name, τη σύνθεση παράλληλων διεργασιών και την αντιγραφή διεργασίας (replication):

P::= x(y:T).P | x<z>.P | (v a:T)P | P1|P2 | !P | 0

Όταν το εργαλείο μετάφρασης εντοπίσει στο δέντρο που παράγει ο Dom Parser από XML αρχείο, κόμβο με το όνομα <resGP>, αναγνωρίζει ότι πλέον το πρόγραμμα Pi Calculus έχει εισέλθει στο επίπεδο διεργασίας και οποιαδήποτε δομή είναι ιεραρχικά πιο κάτω από το resGP (το οποίο υποδηλώνει νέο process group), πρέπει να αναφέρεται στη σύνταξη του επιπέδου διεργασίας που περιγράφεται. Για κάθε παράλληλη διεργασία που ορίζεται στο pi calculus σύστημα, δημιουργείται μία νέα κλάση. Για κάθε περιορισμό (*restriction*) που ορίζεται στο σύστημα, δημιουργείται στην κλάση της αντίστοιχης ιεραρχίας με αυτήν στην οποία υπάγεται ο περιορισμός (restriction), νέο αντικείμενο. Ανάλογα με τον τύπο του νέου ονόματος που ορίζεται στον περιορισμό περιορισμός, το αντικείμενο που δημιουργείται στην κλάση μπορεί να είναι είτε κανάλι (ChannelValue) είτε βασικός τύπος αντικειμένου (base type).

Για κάθε κόμβο ενέργειας εισόδου (*input prefix*), στην κλάση της αντίστοιχης ιεραρχίας που ορίζει το pi calculus σύστημα, στον κώδικα ορίζεται ότι από το κανάλι μέσω του οποίου λαμβάνεται μήνυμα, θα κάνει πράξη read() και το αντικείμενο που αποτελεί τον χωροθέτη (placeholder) θα πάρει το αποτέλεσμα της λήψης. Αυτό σημαίνει, ότι το αντικείμενο που αποτελεί τον χωροθέτη, μετά την πραγματοποίηση του in action, δεσμεύεται για την διεργασία.

Για κάθε κόμβο ενέργειας εξόδου (*output prefix*), στην κλάση της αντίστοιχης ιεραρχίας που ορίζει το pi calculus σύστημα, στον κώδικα ορίζεται ότι στο κανάλι μέσω του οποίου θα αποσταλεί το μήνυμα, θα γίνει πράξη write() (εγγραφή), με όρισμα στη συνάρτηση write() το αντικείμενο/κανάλι που αποστέλλεται.

Για κάθε αντιγραφή (*replication*) κόμβο που εντοπίζει το εργαλείο στο δέντρο του xml αρχείου, θα δημιουργείται μία νέα κλάση που θα εκκινείται και θα τρέχει παράλληλα με την τρέχουσα διεργασία, μόλις εκτελεστεί η πρώτη ενέργεια εισόδου ή εξόδου της διεργασίας που αντιγράφεται με τον κανόνα (replicate). Για το σκοπό του κανόνα της αντιγραφής έχει χρησιμοποιηθεί η κλάση Process Manager της βιβλιοθήκης JCSP και ο τρόπος μετάφρασης της αντιγραφής και της χρήσης της κλάσης Process Manager αναλύεται σε επόμενο υποκεφάλαιο.

Πιο κάτω δίνεται το πρωτότυπο του κώδικα κλάσεων σε αντιστοιχία με τα δομικά στοιχεία της Pi Calculus (Επίπεδο Διεργασιών) και την αντίστοιχη διατύπωσή τους σε XML

1. x(y:T).P : Ενέργεια Εισόδου (Input Prefix) με το x να είναι το κανάλι μέσω του οποίου θα γίνει η ενέργεια εισόδου και το y να είναι το αντικείμενο στο οποίο θα τοποθετηθεί το αντικείμενο που λαμβάνεται. Το y πρέπει να είναι τύπου T.

Σε αυτό το σημείο πρέπει να αναφερθεί ότι η ενέργεια εισόδου μπορεί να αναφέρεται σε μία επικοινωνία με μία διεργασία εντός του ορισμένου συστήματος στο xml αρχείο αλλά μπορεί να αναφέρεται ακόμα και σε μία επικοινωνία με μία διεργασία εξωτερικά του συστήματος, που δεν γνωστοποιείται στο xml αρχείο. Γι' αυτές τις δύο περιπτώσεις, ο χρήστης πρέπει να διαφοροποιήσει την xml σύνταξή του για το input prefix

Επικοινωνία εσωτερικά του συστήματος:

$x(y:T)P$:
```
<in>
    <subj>x</subj>
    <obj>y</obj>
    <type>T</type>
    < process>P</ process>
</in>
```

Επικοινωνία με διεργασία του συστήματος:

x(*y:T*).P:
```
<inExternal>
    <subj>x</subj>
    <obj>z</obj>
    <type>T</type>
    <process>P</process>
</inExternal>
```

Πρωτότυπο κώδικα στην κλάση της διεργασίας όπου υπάρχει η ενέργεια εισόδου:

Περίπτωση 1: Το αντικείμενο y, ανήκε προηγουμένως στα ελεύθερα κανάλια και πλέον θα δεσμευτεί και ο τύπος του δεν είναι βασικός τύπος:

```
[name_y] = (ChannelValue) x.channel.in().read();
```

Περίπτωση 2: Το αντικείμενο y, είναι νέο όνομα που δεν υπάρχει στα ελεύθερα κανάλια του συστήματος και ο τύπος του δεν είναι βασικός τύπος:

```
ChannelValue [name_y] = new ChannelValue();
[name_y].type = "[channelnType]";
[name_y]= (ChannelValue) x.channel.in().read();
```

Περίπτωση 3: Ο τύπος του αντικειμένου y, ανήκει στα base types:

```
[base_type_y] [name_y] = new base_type_y();
[name_y].type = "[channelnType]";
[name_y]= (base_type_y) x.channel.in().read();
```

Περίπτωση 4 – inExternal: Την πληροφορία που πρέπει να δεχτεί το αντικείμενο y μέσω του καναλιού x, θα τη δώσει ο χρήστης:

```
String nameY = JOptionPane.showInputDialog(null,"Communication with an external system – Give the value of y:");
[name_y] = (Object)nameY;
```

2. x<z>.P : Ενέργεια Εξόδου (Output Prefix) με το x να είναι το κανάλι μέσω του οποίου θα γίνει η ενέργεια και το y να είναι το αντικείμενο που θα σταλεί.

Σε αυτό το σημείο πρέπει να αναφερθεί ότι η ενέργεια εξόδου μπορεί να αναφέρεται σε μία επικοινωνία με μία διεργασία εντός του ορισμένου συστήματος στο xml αρχείο αλλά μπορεί να αναφέρεται ακόμα και σε μία επικοινωνία με μία διεργασία εξωτερικά του συστήματος, που δεν γνωστοποιείται στο xml αρχείο. Γι' αυτές τις δύο περιπτώσεις, ο χρήστης πρέπει να διαφοροποιήσει την xml σύνταξή του για την ενέργεια εξόδου.

Επικοινωνία εσωτερικά του συστήματος:

$\bar{x}(z).P$:
```
<out>
  <subj>x</subj>
  <obj>z</obj>
  < process >P</ process >
</out>
```

Επικοινωνία με διεργασίες εξωτερικά του συστήματος:

x<z>.P :
```
<outExternal>
    <subj>x</subj>
    <obj>z</obj>
    <process>P</process>
</outExternal>
```

Πρωτότυπο κώδικα στην κλάση της διεργασίας όπου υπάρχει ενέργεια εξόδου:

Περίπτωση 1 – επικοινωνία εσωτερικά του συστήματος:

[channel x].channel.out().write([name_z]);

Περίπτωση 2 – επικοινωνία εξωτερικά του συστήματος:

```
JOptionPane.showMessageDialog(null,"Communication with an external system –The value of y is: " + [name_y]);
```

3. (v a:T)P: Δημιουργία νέου ονόματος a που μπορεί να είναι είτε κανάλι είτε βασικός τύπος (base type). Το νέο όνομα δημιουργείται στην κλάση της αντίστοιχης διεργασίας στην οποία ορίζεται στο xml αρχείο και ορατό και προσβάσιμο μόνο από την ιεραρχικά κατωτερη δομή διεργασίας. Ο χρήστης θα πρέπει να το διατυπώσει σε xml στην πιο κάτω μορφή:

```
(va : T)P : <resNP>
              <name>a</name>
              <type>T</type>
              < process >P</ process >
            </ resNP >
```

Πρωτότυπο κώδικα στην κλάση της διεργασίας όπου δημιουργείται το νεό όνομα:

Περίπτωση 1 – ο τύπος του νέου ονόματος δεν είναι βασικός τύπος (base type):

```
ChannelValue [name_a] = new ChannelValue();
[name_a].type = "[channel_a_Type]";
```

Περίπτωση 2 – ο τύπος του νέου ονόματος είναι βασικός τύπος (base type):

```
[base_type_a] [name_a] = new [base_type_a]();
```

4. P1|P2: Δύο διεργασίες που τρέχουν παράλληλα. Ο χρήστης θα πρέπει να το διατυπώσει σε xml στην πιο κάτω μορφή:

```
P | Q: <parP>
        < process>P</ process>
        < process>Q</ process>
     </parP>
```

Οι δύο διεργασίες θα εκκινηθούν παράλληλα από τη διεργασία η οποία είναι ο άμεσος απόγονος στο δέντρο του xml αρχείου. Το εργαλείο μετάφρασης για κάθε μία από τις παράλληλες διεργασίες, δημιουργεί μία καινούργια κλάση η οποία υλοποιεί τη διεπιφάνεια CSProcess. Η κλάση της κάθε διεργασίας, έχει ως ορίσματα όλα τα ονόματα στα οποία μπορεί να έχει πρόσβαση και αποκτά την αναφορά τους ούτως ώστε να μπορεί να έχει πρόσβαση σε αυτά, στον constructor από την διεργασία που την εκκινεί.

Πρωτότυπο κώδικα στην κλάση προγόνου απ' όπου θα εκκινούνται οι παράλληλες διεργασίες:

```
ParallelProc[processCounter] [ParP_1] = new ParallelProc[processCounter]
([channel1],..[channeln]);
        .
        .
ParallelProc[processCounter+n] [ParP_n] = new ParallelProc[processCounter+n]
([channel1],..[channeln]);
```

Οι παράλληλες διεργασίες μπαίνουν σε ένα κοινό πίνακα:

```
CSProcess[] parParts = new CSProcess[]{[ParP_1],…, [ParP_n]};
```

Ορίζουμε ότι τα συστήματα στον πίνακα θα εκτελεστούν παράλληλα:

```
Parallel par = new Parallel(parParts);
```

Τρέχουμε τα παράλληλα συστήματα:

```
par.run();
```

Πρωτότυπο κώδικα της κλάσης της κάθε παράλληλης διεργασίας:

Αναγκαίες βιβλιοθήκες για να τρέχει ο κώδικας.

```
import org.jcsp.lang.CSProcess;
import org.jcsp.lang.Parallel;
import org.jcsp.lang.ProcessManager;

public class ParallelProc[processCounter] implements CSProcess  {

        public ChannelValue channel1 ;
                .
                .                .
        public ChannelValue channeln ;

        public ParallelProc[processCounter] (ChannelValue [channel1],…,
ChannelValue [channeln]){
                [channel1] = new ChannelValue();
                [channel1].type = [channel1Type];
                        .
                        .
                [channeln] = new ChannelValue();
                [channeln].type = [channelnType];
        }

@Override
        public void run() {
                .
                .
                .
        }

}
```

Δηλώνονται τα κανάλια που έχουν οριστεί πιο πάνω στην ιεραρχία και η διεργασία μπορεί να έχει πρόσβαση σε αυτά.

Constructor της κλάσης. Δέχεται σαν είσοδο τις αναφορές των καναλιών που έχει πρόσβαση από την κλάση που εκκινεί τη διεργασία αυτή και τα συνδέει με τις εσωτερικές αναφορές των καναλιών, ορίζοντας ακολούθως και τον τύπο τους.

Στη συνάρτηση εγγράφονται τα οι ενέργειες διεργασιών που θα περιέχονται στη συγκεκριμμένη διεργασία.

5. !P : Πιο κάτω τεκμηριώνεται το πώς μεταφράζεται ο κανόνας αντιγραφής (rep) της pi-calculus σε java με τη χρήση της κλάσης ProcessManager της JCSP.

Το replication $!P$ είναι μία διεργασία που πάντα δημιουργεί ένα αντίγραφο της P. Ξέρουμε ότι με τον κανόνα αντιγραφής ισχύει ο πιο κάτω σημασιολογία μετάβασης:

$$\frac{P \xrightarrow{\ell} P'}{!P \xrightarrow{\ell} P' \mid !P} \text{(Repl)}$$

Για παράδειγμα, αν !P = ! in a(b) . out b<p> . 0

Τότε με μία μετάβαση a:

! in a(b) . out b<p> . 0

↓ a

! in a(b) . out b<p> . 0 | out b<p> . 0

Σε αυτό το σημείο θα χρησιμοποιηθεί η κλάση ProcessManager της βιβλιοθήκης JCSP. Η κλάση Process Manager επιτρέπει την εκτέλεση ενός νέου process ταυτόχρονα με το process που το δημιουργεί. Αυτό δεν επιτρέπεται από την κλάση CSProcess που χρησιμοποιούμε για τις άλλου είδους διεργασίες. Για παράδειγμα, έστω ότι έχουμε το CSProcess a, το οποίο εκκινείται μέσω ενός άλλου CSProcess b, από τη στιγμή που το CSProcess b εκτελέσει την εντολή a.run(), δεν θα προχωρήσει στις επόμενες εντολές μέχρι να ολοκληρωθεί η εκτέλεση του CSProcess a. Αυτό είναι κάτι μη επιθυμητό στην περίπτωση του Replication Process. Στην περίπτωση αυτή, η διεργασία που θα δημιουργεί το νέο αντίγραφο δεν πρέπει να διακόπτεται αλλά να συνεχίζει την εκτέλεσή της παράλληλα με αυτό. Η λειτουργία αυτή επιτυγχάνεται με τη δημιουργία ενός αντικειμένου τύπου ProcessManager με είσοδο την διεργασία που θέλουμε να εκτελεστεί παράλληλα με την τρέχουσα. Έπειτα αυτό το αντικείμενο θα γίνει start. Στο πιο πάνω παράδειγμα, ο κώδικας του CSProcess b τη στιγμή που θα πρέπει να γίνει το αντίγραφο θα περιλαμβάνει:

```
ProcessManager manager = new ProcessManager(a);
```

Η προηγούμενη Διεργασία Αντιγραφής (Replication Process) !P = ! in a(b) . out b<p> . 0 σε κώδικα java θα είχε την πιο κάτω μορφή:

```
while(true){
```

```
this.b = this.a.channel.in().read();
processContinue m = new processContinue(b,p);
ProcessManager managerP = new ProcessManager(m);
 managerP.start ();
}
```

Το αντικείμενο ProcessContinue m, είναι ένα CSProcess στο οποίο περιλαμβάνονται ενέργειες του P εκτός του πρώτου που είναι το in a(b). Το m γίνεται είσοδος του αντικειμένου ProcessManager managerP (που επιτρέπει την παράλληλη εκτέλεση του αντιγράφου) και θα ξεκινήσει την εκτέλεσή του με την εντολή managerP.start();.

Επεξήγηση λειτουργίας του πιο πάνω κώδικα:

Κάθε φορά που γίνεται read από το κανάλι a, το ProcessManager αντικείμενο, θα αναλάβει τη συνέχεια της εκτέλεσης του P και θα εκτελέσει τις επόμενες ενέργειες που περιλαμβάνει το P, την ίδια στιγμή που στην τρέχουσα διεργασία, το πρόγραμμα θα βρίσκεται και πάλι στην $1^{η}$ εντολή της επανάληψης για να περιμένει να διαβάσει ξανά από το a. Όταν ξαναδιαβάσει από το a, θα δημιουργήσει ακόμα ένα αντίγραφο της υπόλοιπης διεργασίας P.

Ο χρήστης θα πρέπει να το διατυπώσει τον κανόνα αντιγραφής σε xml στην πιο κάτω μορφή:

```
!P: <repl>!
      < process >P</ process >
    </repl>
```

Πρωτότυπο κώδικα στην κλάση του process που περιέχει τον κανόνα αντιγραφής:

```
while(true){
        [πρότυπο κώδικα in action ή out_action ανάλογα με το πρώτο
        prefix του process P]
        Repl[counterRepl] rep = new Repl[counterRepl]
        ([channel1],..[channeln]);
        ProcessManager manager = new ProcessManager(rep);
        manager.start();
}
```

Πρωτότυπο κώδικα της κλάσης της κάθε αντιγραμμένης διεργασίας: το ίδιο με τη σελ71, αλλά στην συνάρτηση run(), δεν μπαίνει η πρώτη ενέργεια της διεργασίας p, εφόσον έχει εκτελεστεί στο while.

Διαχείριση ελεύθερων καναλιών από το εργαλείο μετάφρασης:

Το εργαλείο μετάφρασης, αφότου δεχτεί από τη διεπιφάνεια τους βασικούς τύπους που θα δώσει ο χρήστης και δημιουργήσει τις κατάλληλες κλάσεις, διαχειρίζεται τα ελεύθερα κανάλια στο σύστημα. Το πρόγραμμα εντοπίζει τα ελεύθερα κανάλια (όσα δεν δημιουργούνται μέσω ενός περιορισμού και όσα δεν δεσμεύονται εξαρχής μέσω ενός input prefix) και τα κρατά σε ένα Hash Map. Για τα ελεύθερα κανάλια, αν δεν ορίζονται στα ορίσματα του περιβάλλοντος Γ, δεν μπορούμε να ξέρουμε τον τύπο τους. Οπότε, στο αντικείμενο ChannelValue των ελεύθερων καναλιών, το πεδίο type ορίζεται ως unknown. Έπειτα, στην κλάση του συστήματος που είναι πιο υψηλά στην ιεραρχία του δοσμένου pi calculus συστήματος, τα ελεύθερα κανάλια, ορίζονται ως **public** πεδία της κλάσης, ούτως ώστε αν στο μέλλον κάποιο εξωτερικό σύστημα επιχειρήσει να επικοινωνήσει με το τρέχον μέσω αυτών των καναλιών, να μπορεί να έχει πρόσβαση σε αυτά. Από τη στιγμή που τα κανάλια είναι ελεύθερα, δεν παραβιάζεται με αυτόν τον τρόπο καμία πολιτική ιδιωτικότητας.

4.3 Ορθότητα Μεταφρασμένου Προγράμματος:

Για να επιβεβαιώσουμε την ορθότητα του συστήματος, θέλουμε να επιβεβαιώσουμε ότι κάθε ενέργεια που αναπαριστάται στο μοντελοποιημένο σύστημα με τη σύνταξη της επέκτασης του π-λογισμού με ομάδες, εκτελείται και στο παραγόμενο κώδικα του συστήματος σε Java και αντίστροφα. Δηλαδή, μία ενέργεια γίνεται στο δοσμένο μοντελοποιημένο σύστημα σε π-λογισμό αν και μόνο αν, γίνεται στο αντίστοιχο σημείο του υλοποιημένου συστήματος σε Java.

Επομένως, επιθυμούμε για κάθε κανόνα της επέκτασης να ισχύει το πιο κάτω θεώρημα.

Θεώρημα:

Έστω ένα σύστημα S και [[S]] η μετάφρασή του σε Java που παράγεται από το εργαλείο. Τότε:

$$S \xrightarrow{a} S' \text{ αν και μόνο αν } [[S]] \mapsto \text{Pr}, \text{όπου Pr} = [[S']]$$

και όπου $\mapsto$ συμβολίζει την εκτέλεση ενός βημάτων σε κώδικα Java

Κανόνας Αντιγραφής:

$$\frac{P \xrightarrow{l} P'}{!P \xrightarrow{l} P' \mid !P} \text{ (Repl)}$$

Μετάφραση σε java:

```
1 while(true){
        2       this.b = this.a.channel.in().read();  → παράδειγμα μίας ενέργειας l
        3       processContinue m = new processContinue(b,p);
        4       ProcessManager P = new ProcessManager(m);
        5        P.start ();
                }
```

Αυτό που επιθυμούμε είναι κάθε φορά που εκτελείται η πρώτη ενέργεια της διεργασίας του κανόνα αντιγραφής, να δημιουργείται ένα αντίγραφό του που θα εκτελείται παράλληλα. Όπως βλέπουμε στον κώδικα, κάθε φορά που θα εκτελείται η ενέργεια l, δημιουργείται ένα αντίγραφο της διεργασίας P το οποίο θα τρέχει παράλληλα με την επαναληπτική δομή της τρέχουσας διεργασίας. Αν επιχειρήσουμε να δώσουμε μία αυστηρή αντιστοιχία του κώδικα με τον κανόνα, τότε αμέσως πριν το σημείο του while (στη γραμμή 1), έχουμε την διεργασία [[!P]], δηλαδή η μετάφραση του !P σε Java. Η πρώτη γραμμή του εσωτερικού της επανάληψης (γραμμή 2), αντιστοιχεί στην ενέργεια l που εμφανίζεται πιο πάνω στον κανόνα. Το αμέσως επόμενο βήμα στον κώδικα (γραμμή 3) στην ουσία αντιπροσωπεύει τη δημιουργία του [[P]] που είναι η μετάφραση του P σε Java. Οι τελευταίες δύο γραμμές (γραμμές 4+5) αντιπροσωπεύουν το ότι το [[P]] τίθεται σε παράλληλη σύνθεση με την τρέχουσα διεργασία (λόγω της ιδιότητας της κλάσης ProcessManager). Εφόσον έχουμε επανάληψη με συνθήκη πάντα αληθή, τότε η τρέχουσα διεργασία, μετά την ενέργεια, θα συνεχίσει από το σημείο της γραμμής 1, που όπως είπαμε αντιπροσωπεύει το [[!P]]. Άρα με βάση το τι έχουμε πει, το [[P]] τίθεται σε παράλλλη σύνθεση με το [[!P]]: [[P]] | [[!P]]. Άρα έχοντας τη διεργασία [[!P]] με ένα βήμα l τριών σειριακών εντολών, η διεργασία συνεχίζει ως [[P]] | [[!P]]

Επομένως αν υποθέσουμε ότι η υλοποίηση του P είναι ορθή, τότε και η υλοποίηση του $!P$ είναι ορθή, αφού όντως γίνεται αντιγραφή μετά την ενέργεια l.

1 **while**(**true**){	[[!P]] που αντιστοιχεί στη μετάφραση του P σε Java
2 **this**.b = **this**.a.channel.in().read();	Βήμα σε κώδικα Java που αντιπροσωπεύει μία ενέργια l
3 processContinue m = **new** processContinue(b,p);	Δημιουργία διεργασίας [[P']] (μετάφραση του P' σε Java)
4 ProcessManager P = **new** ProcessManager(m); 5 P.start (); }	Θέτουμε την τρέχουσα διεργασία να τρέχει παράλληλα με το [[P']], δηλαδή [[P']] | [[!P]], λόγω του while

Κανόνας Ενέργειας Εισόδου:

$x(y:T).P \xrightarrow{x(z)} P\{z/y\}$ (In)

Μετάφραση σε java:

1 [name_y] = (ChannelValue) x.channel.in().read(); → ενέργεια εισόδου $\xrightarrow{x(z)}$

2 [P].run(); → συμβολισμός για [[P]].

Ο κανόνας In αναφέρει ότι σε περίπτωση που μία διεργασία $x(y:T)$.P πραγματοποιήσει ενέργεια εισόδου μέσω του x, τότε συνεχίζει ως P. Αυτό όντως τηρείται. Όταν η ροή του προγράμματος φτάσει στην πιο πάνω γραμμή κώδικα, η διεργασία δεν θα συνεχίσει ως P (δηλαδή δεν θα συνεχίσει στις επόμενες ενέργειες του P) μέχρι να πραγματοποιηθεί ενέργεια εισόσου μέσω του x, αφού η συγκεκριμμένη συνάρτηση της βιβλιοθήκης JCSP δεν επιτρέπει στο πρόγραμμα να προχωρήσει αν δεν λάβει αντικείμενο στο κανάλι επικοινωνίας. Μετά από αυτήν την εντολή, όντως η διεργασία συνεχίζει ως P, αφού όντως θα προχωρήσει στην εκτέλεση των επόμενων ενεργειών της διεργασίας. Αν επιχειρήσουμε να δώσουμε μία αυστηρή αντιστοιχία του κώδικα με τον κανόνα, τότε ακριβώς πριν εκτελεστεί η γραμμή 1, η τρέχουσα διεργασία είναι η [[$x(y:T)$.P]] που αντιπροσωπεύει τη μετάφραση του $x(y:T)$.P σε κώδικα. Το αμέσως

επόμενο βήμα (γραμμή 1) που θα εκτελεστεί, αντιπροσωπεύει την ενέργεια εισόδου x(z) σε κώδικα (↦). Η γραμμή 3, συμβολίζει ότι μπορούν να ακολουθήσουν άλλες ενέργειες αμέσως μετά την εκτέλεση της ενέργειας εισόδου, που συνθέτουν τη διεργασία [[P]]. Η μετονομασία του y σε z, εγγυόμαστε ότι συμβαίνει αφού στην εντολή της ενέργειας εισόδου (γραμμή 1) γίνεται ανάθεση σε αντικείμενο, δηλαδή το αντικείμενο χωροθέτης αλλάζει αναφορά και «δείχνει» στο αντικείμενο που λαμβάνει. Επομένως αν υποθέσουμε ότι η υλοποίηση του P είναι ορθή, τότε και η υλοποίηση του $x(y{:}T).P$ είναι ορθή.

Πριν την εκτέλεση γραμμής 1	[[$x(y{:}T).P$]], που αντιπροσωπεύει τη μετάφραση του $x(y{:}T).P$ σε Java
1 [name_y] = (ChannelValue) x.channel.in().read();	(↦): ενέργεια εισόδου σε Java κώδικα
2 [P].run();	[[P]], που αντιπροσωπεύει τη μετάφραση του P σε Java

Κανόνας Ενέργειας Εξόδου:

$\overline{x}\langle z\rangle.P \xrightarrow{\overline{x}\langle z\rangle} P$ (Out)

Μετάφραση σε java:

1 [channel x].channel.out().write([name_z]); → ενέργεια $\xrightarrow{\overline{x}\langle z\rangle}$

2 [P].run(); → συμβολισμός για την συνέχεια της διεργασίας P.

Ο κανόνας Out αναφέρει ότι όταν μία διεργασία $\overline{x}\langle z\rangle.P$ εκτελέσει ενέργεια εξόδου μέσω του καναλιού x, τότε συνεχίζει ως P. Παρόμοια λοιπόν με την υλοποίηση της ενέργειας εξόδου, όταν η ροή του προγράμματος φτάσει στην πιο πάνω γραμμή εντολής, δεν θα προχωρήσει στις υπόλοιπες ενέργειες (άρα δεν θα συνεχίσει ως P) μέχρι να εκτελεστεί η ενέργεια εξόδου. Όταν όμως εκτελεστεί η ενέργεια εξόδου μέσω του καναλιού x, τότε η ροή του προγράμματος θα προχωρήσει στις υπόλοιπες ενέργειες της διεργασίας, οπότε θα συνεχίσει ως P. Αν επιχειρήσουμε να δώσουμε μία αυστηρή αντιστοιχία του κώδικα με τον κανόνα, τότε ακριβώς πριν την εκτέλεση της γραμμής 1, η τρέχουσα διεργασία είναι η [[$\overline{x}\langle z\rangle.P$]], που αντιπροσωπεύει τη μετάφραση του $\overline{x}\langle z\rangle.P$ σε κώδικα Java. Η γραμμή 3, συμβολίζει ότι μπορούν να ακολουθήσουν άλλες

ενέργειες αμέσως μετά την εκτέλεση της ενέργειας εξόδου, που συνθέτουν τη διεργασία [[P]]. Το αμέσως επόμενο βήμα που θα εκτελεστεί (↦) είναι η μετάφραση της ενέργειας εξόδου σε κώδικα. Επομένως αν υποθέσουμε ότι η υλοποίηση του P είναι ορθή, τότε και η υλοποίηση του $\overline{x}\langle y\rangle.P$ είναι ορθή.

Πριν την εκτέλεση γραμμής 1	[[$\overline{x}\langle z\rangle.P$]], που αντιπροσωπεύει τη μετάφραση του $\overline{x}\langle z\rangle.P$ σε Java
1 [name_y] = (ChannelValue) x.channel.in().read();	(↦): ενέργεια εξόδου σε Java κώδικα
2 [P].run();	[[P]], που αντιπροσωπεύει τη μετάφραση του P σε Java

Κανόνας δημιουργίας νέου καναλιού:

$$\frac{F \xrightarrow{l} F' \quad x \notin fn(l)}{(\nu\, x{:}T)F \xrightarrow{l} (\nu\, x{:}T)F'} \quad \text{(ResN)}$$

Γενική μορφή κώδικα σε java:

```
1 ChannelValue [name_x] = new ChannelValue();
2 [name_x].type = ""[channel_a_Type]";
3 [F].run(name_x); → συμβολισμός εκτέλεσης διεργασίας F η οποία έχει πρόσβαση στο όνομα α.
4 [action l].run();
5 [F'].run(name_x);
```

Με βάση τον κανόνα ResN, αν η διεργασία F έχει δεσμευμένο το κανάλι x, συνεχίζει να το έχει δεσμευμένο κατά την επικοινωνία της με άλλες διεργασίες, εκτός και αν το γνωστοποιεί με ενέργεια εξόδου σε άλλη διεργασία. Ο συγκεκριμένος κανόνας λοιπόν, με βάση τον πιο πάνω κώδικα, τηρείται, αφού το αντικείμενο δημιουργείται τοπικά εντός της κλάσσης της διεργασίας. Άρα, αν η ίδια η διεργασία δεν το γνωστοποιήσει με ενέργεια εξόδου σε κάποια άλλη, τότε δεν μπορεί καμία άλλη διεργασία να αποκτήσει πρόσβαση στο x και παραμένει δεσμευμένο και εντός της εκτέλεσης της F'. Αν επιχειρήσουμε να δώσουμε μία αυστηρή αντιστοιχία του κώδικα με τον κανόνα, τότε ακριβώς πριν την εκτέλεση της γραμμής 1, είμαστε στη διεργασία [[$(\nu\, x : T)F$]] που είναι η υλοποίηση του $(\nu\, x : T)F$ σε κώδικα Java. Οι γραμμές 1 και 2, αντιστοιχούν στην δημιουργία νέου καναλιού x, δεσμευμένο στην τρέχουσα διεργασία. Η γραμμή 3,

συμβολίζει ότι μπορούν να ακολουθήσουν άλλες ενέργειες που αντιπροσωπεύουν τη διεργασία [[F]], στην οποία το νέο κανάλι x είναι δεσμευμένο αφού δημιουργήθηκε στην τρέχουσα διεργασία. Έπειτα μπορεί να ακολουθήσει μία ενέργεια l και η γραμμή 4 συμβολίζει την εκτέλεση ενός βήματος για αυτήν την ενέργεια l. Η γραμμή 5 συμβολίζει ότι μπορούν να ακολουθήσουν εντολές που συνθέτουν την μετάφραση [[F']] που αντιστοιχεί στην διεργασία F. Επομένως αν υποθέσουμε ότι η διεργασία F μεταφράζεται ορθά, τότε και η διεργασία $(v\ x)F$ μεταφράζεται. Τα ίδια ισχύουν και για τον κανόνα δημιουργίας νέου καναλιού σε σύστημα **(ResNS)**.

1 ChannelValue [name_x] = **new** ChannelValue(); 2 [name_x].type = ""[channel_a_Type]";	Μετάφραση της δημιουργίας νέου δεσμευμένου καναλιού $(v\ x : T)$ σε Java κώδικα. Άρα σε αυτό το σημείο έχουμε τη διεργασία $[[(v\ x : T)F]]$
3 [F].run(name_x);	[[F]] που είναι η μετάφραση του F σε Java κώδικα
4 [action l].run();	(↣) ενέργεια l σε Java κώδικα
5 [F'].run(name_x);	[[F']] που είναι η μετάφραση του F σε Java κώδικα

Κανόνας παράλληλης σύνθεσης:

$$\frac{F_1 \xrightarrow{l} F'_1 \; F_2 \xrightarrow{l} F'_2 \;\; dual(l_1,l_2)}{F_1|F_2 \xrightarrow{\tau} (v\ bn(l_1) \cup bn(l_2))(F'_1|F'_2)} \quad \text{(Com)}$$

Μετάφραση σε Java:

```
1 ParallelProc[processCounter] [ParP_1] = new ParallelProc[processCounter]
([channel1],..[channeln]);
2 ParallelProc[processCounter+n] [ParP_2] = new ParallelProc[processCounter+n]
([channel1],..[channeln]);
3 CSProcess[] parParts = new CSProcess[]{[ParP_1], [ParP_2]};
4 Parallel par = new Parallel(parParts);
5 par.run();
```

Ο κανόνας Com αναφέρει ότι αν δύο παράλληλες διεργασίες μπορούν να επικοινωνήσουν με dual ενέργειες (όπως ορίζονται πιο πάνω), μπορεί να γίνει αυτό με εσωτερική ενέργεια και μετά την επικοινωνία, στην παράλληλη σύνθεση δεσμεύονται τα δεσμευμένα κανάλια των dual ενεργειών.

Αυτός ο κανόνας όπως βλέπουμε και από τον κώδικα, μπορεί να τηρηθεί, αφού οι δύο διεργασίες τρέχουν παράλληλα στην σύνθεση: Parallel par = **new** Parallel(parParts); Επομένως, η ενέργεια που θα εκτελεστεί για την επικοινωνία των δύο διεργασιών, θα είναι εσωτερική της παράλληλης σύνθεσης. Επομένως, τα δεσμευμένα κανάλια της επικοινωνίας, πλέον δεν είναι δεσμευμένα εντός της διεργασίας, αλλά ένα επίπεδο πιο πάνω, εντός της παράλληλης σύνθεσης. Αν επιχειρήσουμε να δώσουμε μία αυστηρή αντιστοιχία του κώδικα με τον κανόνα, τότε οι γραμμές 1 και 2 δημιουργούν τις διεργασίες [[F1]] και [[F2]]. Η γραμμή 3 και 4, αντιπροσωπεύει τη διάταξή τους σε μία παράλληλη σύνθεση [[F1]] | [[F2]] που είναι η Java υλοποίηση του όρου F1 | F2. Έπειτα, στη γραμμή 5, βλέπουμε ότι τρέχει η παράλληλη σύνθεση των δύο διεργασιών και οι ενέργειες που μπορεί να εκτελέσουν είναι εσωτερικές και γι' αυτό δεν ακολουθεί περαιτέρω κώδικας σε αυτό το σημείο.

Επομένως αν υποθέσουμε ότι η υλοποίηση των διεργασιών F1 ([ParP_1]) και F2 ([ParP_2]) είναι ορθή, τότε και η υλοποίηση της παράλληλης σύνθεσης F1 | F2 (Parallel(parParts)) είναι ορθή. Τα ίδια ισχύουν και για τον κανόνα παράλληλης σύνθεσης συστημάτων **(ParS)**.

ParallelProc[processCounter] [ParP_1] = **new** ParallelProc[processCounter] ([channel1],..[channeln]); 2 ParallelProc[processCounter+n] [ParP_2] = **new** ParallelProc[processCounter+n] ([channel1],..[channeln]);	Δημιουργία αντικειμένων των κλάσεων που υλοποιούν τις δύο διεργασίες. Δημιουργία αντικειμένου για τις διεργασίες [[F1]] και [[F2]]
3 CSProcess[] parParts = **new** CSProcess[]{[ParP_1], [ParP_2]}; 4 Parallel par = **new** Parallel(parParts);	Δημιουργία παράλληλης σύνθεσης των δύο διεργασιών [[F1]] | [[F2]]
5 par.run();	Οι δύο διεργασίες τρέχουν σε μία παράλληλη σύνθεση αλληλεπιδρώντας με εσωτερικές ενέργειες (τ) : $[[(v\ bn(l_1)\ \cup\ bn(l_2))(F'_1 \mid F'_2)]]$

Κανόνας δημιουργίας νέας ομάδας:

$$\frac{F \xrightarrow{l} F'}{(\nu\ G)F \xrightarrow{l} (\nu\ G)F'} \quad \text{(ResG)}$$

Βασική μορφή κώδικα σε Java:

1 Process[G: groupName] [G: groupName] = **new** Process[G: groupName] ([channel1],…,[channeln]);
2 CSProcess[][G: groupName]processes = **new** CSProcess[]{[G: groupName]};
3 atomicGroup [G: groupName]= **new** atomicGroup(OBEprocesses, "[G: groupName]");
4 [G: groupName].run();

Ο κανόνας δημιουργίας νέας ομάδας αναφέρει ότι με τη δημιουργία μίας ομάδας, οι μεταβάσεις της διεργασίας εντός της ομάδας γίνονται μέσα στον περιορισμό της ομάδας. Αυτός ο κανόνας όντως τηρείται, αφού στο αντικείμενο που αρχικοποιείται της διεργασίας, στέλνονται μόνο τα κανάλια στα οποία μπορεί να έχει πρόσβαση η ομάδα, επομένως οι διάφορες μεταβάσεις γίνονται εντός περιορισμών της ομάδας. Αν επιχειρήσουμε να δώσουμε μία αυστηρή αντιστοιχία του κώδικα με τον κανόνα, στις γραμμές 1-3 υλοποιείται η δημιουργία αντικειμένου της κλάσης που αντιστοιχεί στη διεργασία F και η δημιουργία της νέας ομάδας, δηλαδή αντιστοιχεί στο [[$(\nu\ G)F$]], που συμβολίζει τη μετάφραση Java για τη διεργασία $(\nu\ G)F$. Στην γραμμή 4, η διεργασία [[F]] τίθεται υπό εκτέλεση αλλά υπό την ιεραρχία της ομάδας G. Και άρα, ακόμα και αν στο εσωτερικό της συνάρτησης run() της γραμμής 4, υπάρχει η υλοποίηση μίας ενέργειας l (↦), πάλι η διεργασία θα τρέχει ανήκωντας στην ομάδα G. Επομένως, αν υποθέσουμε ότι η μετάδραση και υλοποίηση της διεργασίας P είναι ορθή, τότε και η υλοποίηση της διεργασία (ν G)P είναι ορθή. Με αντίστοιχο τρόπο επιβεβαιώνεται και η ορθότητα της μετάφρασης της δημιουργίας ομάδας στο επίπεδο συστήματος **(ResGS)**.

1 Process[G: groupName] [G: groupName] = **new** Process[G: groupName]	Αντιστοιχεί στη δημιουργία της διεργασίας [[F]], που είναι η Java μετάφραση της F.

([channel1],…,[channeln]); 2 CSProcess[][G: groupName]processes = **new** CSProcess[]{[G: groupName]};	
3 atomicGroup [G: groupName]= **new** atomicGroup(OBEprocesses, "[G: groupName]");	Αντιστοιχεί στη δημιουργία της ομάδας G και στην ένταξη της [[F]] υπό τον περιορισμό της ομάδας G. Δηλαδή, αντιστοιχεί στο $[[(\nu\ G)F]]$
4 [G: groupName].run();	Η ομάδα με τη διεργασία τρέχει πραγματοποιώντας ενέργεια l, χωρίς να φεύγει από τον περιορισμό της ομάδας. Δηλαδή, αντιστοιχεί στο: $[[(\nu\ G)F']]$

Πόρισμα:

Με βάση τα πιο πάνω, συμπεραίνουμε ότι το θεώρημα που ορίσαμε πιο πάνω της αντιστοιχίας υλοποίησης και μοντέλου, ισχύει. Επίσης, η υλοποίηση δεν εξάγει δικαιώματα ούτε επιτρέπει ενέργειες που δεν αναφέρονται στο επίπεδο του π-λογισμού.

4.4 Εγχειρίδιο Χρήστη

Η τεκμηρίωση (documentation) του εργαλείου, για να είναι πλήρης πρέπει να περιέχει και τις απαραίτητες οδηγίες για το χρήστη, όχι μόνο για τους μελλοντικούς προγραμματιστές που ίσως χρειαστεί να το χρησιμοποιήσουν ή επεκτείνουν. Το εγχειρίδιο αυτό αποσκοπεί στην κατανόηση του τρόπου λειτουργίας του συστήματος από τον χρήστη, στην ορθή αλλά και αποδοτικότερη χρησιμοποίησή του, ούτως ώστε να μπορεί να δημιουργήσει συστήματα διεργασιών παράλληλης επικοινωνίας, χωρίς την ανάγκη να γράψει κώδικα, αλλά μόνο με την αλγεβρική τους μοντελοποίηση σε π-λογισμό με ομάδες. Βέβαια, αυτό δε σημαίνει ότι ο χρήστης δεν μπορεί να κάνει όσες προσθέσεις επιθυμεί στον παραγώμενο κώδικα που θα χρησιμεύουν για υπολογισμούς, πέρα από το κομμάτι της επικοινωνίας διεργασιών που θα είναι υλοποιημένο από το εργαλείο. Πιο κάτω δίνονται οδηγίες στον χρήστη τόσο για τη συγγραφή του XML αρχείου για το σύστημα αλλά και για τη χρήση του γραφικού περιβάλλοντος του εργαλείου.

4.4.1 Οδηγίες συγγραφής XML αρχείου

Όπως αναφέρθηκε και πιο πριν, ακολουθείται η ίδια σύνταξη που προτείνεται στο [14] μαζί με κάποιες προσθέσεις.

Παράλληλη Σύνθεση Διεργασιών:

```
P | Q:   <parP>
                <process>P</process>
                <process>Q</process>
        </parP>
```

Εκτελείται η διεργασία P παράλληλα με την Q. Ο αρχικός σημαντήρας <parP> δηλώνει το όνομα του κανόνα, ενώ τα παιδιά του που είναι οι σημαντήρες <proccess> δηλώνουν τις 2 διεργασίες που πρόκειται να εκτελεστούν παράλληλα. Στο εσωτερικό των <proccess> μπορούν να ακολουθήσουν κανόνες του επιπέδου διεργασίας.

Για περισσότερες από δύο παράλληλες διεργασίες, υπάρχουν δύο δυνατές διατυπώσεις:

```
<parP>
        <process>P</process>
                .
                .
                .
        < process>Q</process>
</parP>
```

και αναδρομικά:

```
 <parP>
        <process>
                <parP>
                        <process>
                                .
                                .

                </parP>
        <process>Q</process>
```

</parP>

Ενέργεια Εισόδου:

$x(y:T)P$: <in>
<subj>x</subj>
<obj>y</obj>
<type>T</type>
<process>P</process>
</in>

Ο αρχικός σημαντήρας <in> δηλώνει τον κανόνα ενέργειας εισόδου. Το <subj> δηλώνει το κανάλι μέσω του οποίου θα επικοινωνήσει η διεργασία για να δεχτεί το αντικείμενο-κανάλι. Το <obj> αποτελεί τον χωροθέτη, το αντικείμενο δηλαδή στο οποίο θα τοποθετηθεί το αντικείμενο που ληθφεί. Ο σημαντήρας <process> υποδηλώνει ότι έπειτα μπορεί να ακολουθήσει οποιοσδήποτε κανόνας του επιπέδου διεργασίας.

Ενέργεια Εισόδου από εξωτερικό σύστημα:

$x(y:T)P$: <inExternal>
<subj>x</subj>
<obj>y</obj>
<type>T</type>
< process>P</process>
</inExternal>

Ισχύουν τα ίδια με την αμέσως προηγούμενη διατύπωση. Όμως, κάποιες φορές τα μοντελοποιημένα συστήματα σε π-λογισμό, πρέπει να εκφράζουν το γεγονός ότι δέχονται δεδομένα από εξωτερικά συστήματα τα οποία τα χρησιμοποιούν αναλόγως στη συνέχεια. Επειδή όμως, το εργαλείο που αναπτύχθηκε δεν έχει πρόσβαση σε αυτά τα εξωτερικά συστήματα, υπήρχε ο κίνδυνος δημιουργίας αδιεξόδου, αφού σε αυτό το σημείο ο το παραγόμενο πρόγραμμα δεν θα μπορούσε να προχωρήσει μέχρι να πραγματοποιήσει την ενέργεια εισόδου. Γι' αυτό και επιβάλλεται να δίνεται ξεχωριστός σημαντήρας σε αυτές τις περιπτώσεις, ούτως ώστε το εργαλείο να μπορεί να τις διαχειριστεί διαφορετικά και να παράγει κώδικα ο οποίος θα ζητά από τον ίδιο τον χρήστη να εισάγει την είσοδο.

Ενέργεια Εξόδου:

$\overline{x}(z)P$: <out>
<subj>x</subj>
<obj>z</obj>
<process>P</process>

</out>

Ο αρχικός σημαντήρας <out> δηλώνει τον κανόνα ενέργειας εξόδου. Το <subj> δηλώνει το κανάλι το οποίο μέσω αυτού θα σταλεί ένα άλλο αντικείμενο (<obj>). Αφού εκτελεστεί ο κανόνας έχουμε τον σημαντήρα <proccess> ο οποίος δηλώνει ότι μπορεί να ακολουθήσει στη συνέχεια οποιοσδήποτε άλλος κανόνας από τους κανόνες του επιπέδου διεργασιών.

Ενέργεια Εξόδου σε εξωτερικό σύστημα:

$\overline{x}(z)P$: <outExternal>
<subj>x</subj>
<obj>z</obj>
< process>P</ process>
</outExternal>

Ισχύουν τα ίδια με τον κανόνα <inExternal> αλλά για τον κανόνα ενέργειας εξόδου.

Δημιουργία νέου ονόματος/καναλιού στο επίπεδο διεργασιών:

$(\nu\ a{:}T)P$: <resNP>
<name>a</name>
<type>T</type>
< process >P</process>
</ resNP >

Ο αρχικός σημαντήρας <resNP> δηλώνει τον κανόνα δημιουργίας νέου ονόματος. Το <name> δηλώνει το όνομα του δεσμευμένου καναλιού ενώ το <type> τον τύπο του. Αφού εκτελεστεί ο κανόνας έχουμε τον σημαντήρα <proccess> ο οποίος δηλώνει ότι μπορεί να ακολουθήσει στη συνέχεια οποιοσδήποτε άλλος κανόνας από τους υπάρχοντες του επιπέδου διεργασίας. Δηλαδή, το νέο κανάλι, είναι δεσμευμένο στη διεργασία που θα ακολουθήσει στο εσωτερικό του <process>

Κανόνας Δημιουργίας Αντιγράφου (replication):

!P : <repl>!
< process>P</process> // το P απαγορεύεται να είναι <parP>
</repl>

Ο αρχικός σημαντήρας <repl> δηλώνει το όνομα του κανόνα δημιουργίας αντιγράφου της διεργασίας. Αφού εκτελεστεί ο κανόνας έχουμε τον σημαντήρα <proccess> ο οποίος δηλώνει ότι μπορεί να ακολουθήσει στη συνέχεια κανόνας που σχετίζεται με το επίπεδο της διεργασίας, **εκτός από παράλληλη σύνθεση**. Η παράλληλη σύνθεση μπορεί να είναι μεν πιο κάτω στην ιεραρχία, αλλά όχι ο πρώτος κανόνας στο εσωτερικό του replication.

Κανόνας Κενής Διεργασίας – Τερματισμού:

0:<Nil>0</Nil>

Ο αρχικός σημαντήρας <Nil> δηλώνει τον κανόνα τερματισμού. Εδώ δεν έχουμε τον σημαντήρα <proccess> ο οποίος δηλώνει ότι μπορεί να ακολουθήσει στη συνέχεια οποιοσδήποτε άλλος κανόνας από τους υπάρχοντες όπως είχαμε και στους προηγούμενους κανόνες αφού με το Nil έχουμε τερματισμό.

Κανόνας δημιουργίας ομάδας διεργασιών:

(v G)*P*: <resGP>
<group>G</group>
<process>P</process>
</resGP>

Ο αρχικός σημαντήρας <resGP> δηλώνει τον κανόνα δημιουργίας νέας ομάδας διεργασίας. Το <group> δηλώνει το όνομα της ομάδας. Αφού εκτελεστεί ο κανόνας έχουμε τον σημαντήρα <proccess> ο οποίος δηλώνει ότι μπορεί να ακολουθήσει στη συνέχεια οποιοσδήποτε άλλος κανόνας από τους υπάρχοντες που σχετίζονται με το επίπεδο διεργασιών. Στην ουσία ο συγκεκριμένος κανόνας αποτελεί την αλλαγή από επίπεδο συστήματος σε επίπεδο διεργασίας.

Κανόνας Δημιουργίας νέας ομάδας συστημάτων:

(v G)S: <resGS>
<group>G</group>
<system>S</system>
</resGS>

Ο αρχικός σημαντήρας <resGS> δηλώνει τον κανόνα. Το <group> δηλώνει το όνομα της ομάδας. Αφού εκτελεστεί ο κανόνας έχουμε τον σημαντήρα <system> ο οποίος δηλώνει ότι μπορεί να ακολουθήσει στη συνέχεια οποιοσδήποτε άλλος κανόνας από τους υπάρχοντες που σχετίζονται με το επίπεδο συστημάτων.

Δημιουργία νέου ονόματος/καναλιού στο επίπεδο συστημάτων:

$(v\ a{:}T)S$: <resNS>
<name>a</name>
<type>T</type>

```
        <system>S</system>
    </resNS>
```

Ο αρχικός σημαντήρας <resNS> δηλώνει τον κανόνα δημιουργίας νέου ονόματος στο επίπεδο συστήματος. Το <name> δηλώνει το όνομα του δεσμευμένου καναλιού ενώ το <type> τον τύπο του. Αφού εκτελεστεί ο κανόνας έχουμε τον σημαντήρα <system> ο οποίος δηλώνει ότι μπορεί να ακολουθήσει στη συνέχεια οποιοσδήποτε άλλος κανόνας από τους υπάρχοντες του επιπέδου συστήματος. Δηλαδή, το νέο κανάλι, είναι δεσμευμένο στο σύστημα που θα ακολουθήσει στο εσωτερικό του <system>.

Κανόνας Παράλληλης Σύνθεσης Συστημάτων:

```
S1 | S2 : <parS>
            <system>S1</system>
            <system>S2</system>
    </parS>
```

Ο αρχικός σημαντήρας <parS> δηλώνει τον κανόνα παράλληλης σύνθεσης συστημάτων, ενώ τα παιδιά του που είναι οι σημαντήρες <system> δηλώνουν τα 2 συστήματα που πρόκειται να εκτελεστούν παράλληλα. Στην περίπτωση που έχουμε περισσότερα από δύο συστήματα που θα εκτελεστούν παράλληλα, ακολουθείται το ίδιο μοτίβο σύνταξης όπως παρουσιάστηκε στην παράλληλη σύνθεση διεργασιών.

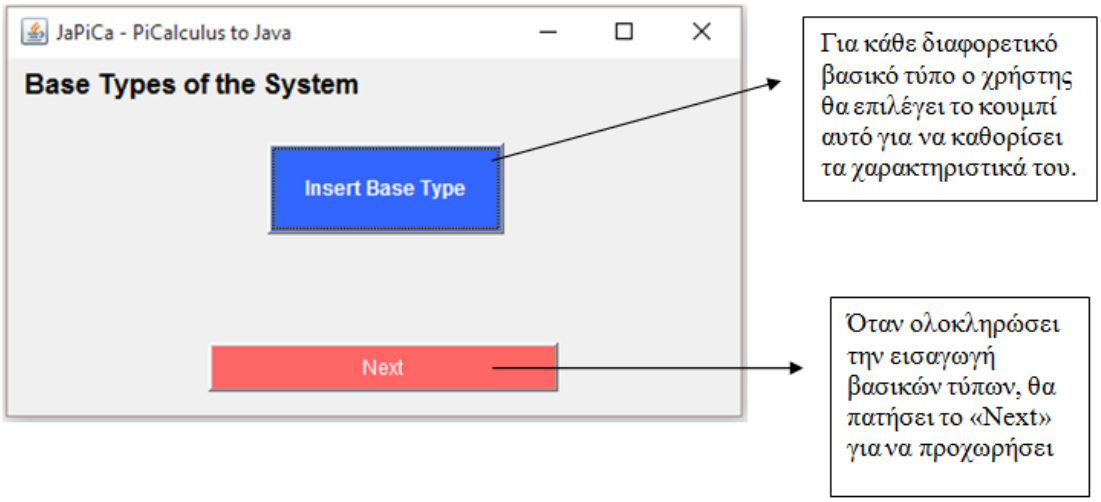
Σύνταξη αρχείου XML για τον καθορισμό των καναλιών του Γ περιβάλλοντος στο οποίο εκτελείται το σύστημα:

Αρχικά ο χρήστης θα πρέπει να γράψει σε ένα σημαντήρα που θα πρέπει να ονομάζεται <g_attributes> και τα κανάλια επικοινωνίας σε σειρά διαχωρίζοντάς τα με κόμμα «,». Για το κάθε κανάλι θα αναγράφει το όνομά του και μετά από άνω και κάτω τελεία «:» τον τύπο του καναλιού/αντικειμένου, όπως για παράδειγμα : <g_attributes>fee:ETP[Fee],send:ETP[ETP[Fee]]</g_attributes>.

4.4.2 Οδηγίες χρήσης διεπιφάνειας αλληλεπίδρασης

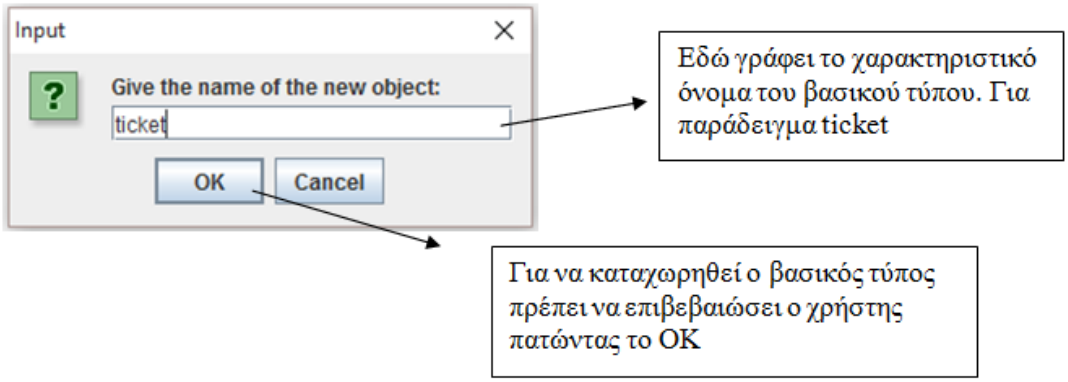
1. Αρχικά ο χρήστης πρέπει να τρέξει το πρόγραμμα. Μόλις το εκκινήσει, θα του εμφανιστεί το παράθυρο μέσω του οποίου θα του ζητείται να καταχωρήσει τους

βασικούς τύπους του μοντελοποιημένου συστήματος σε π-λογισμό.

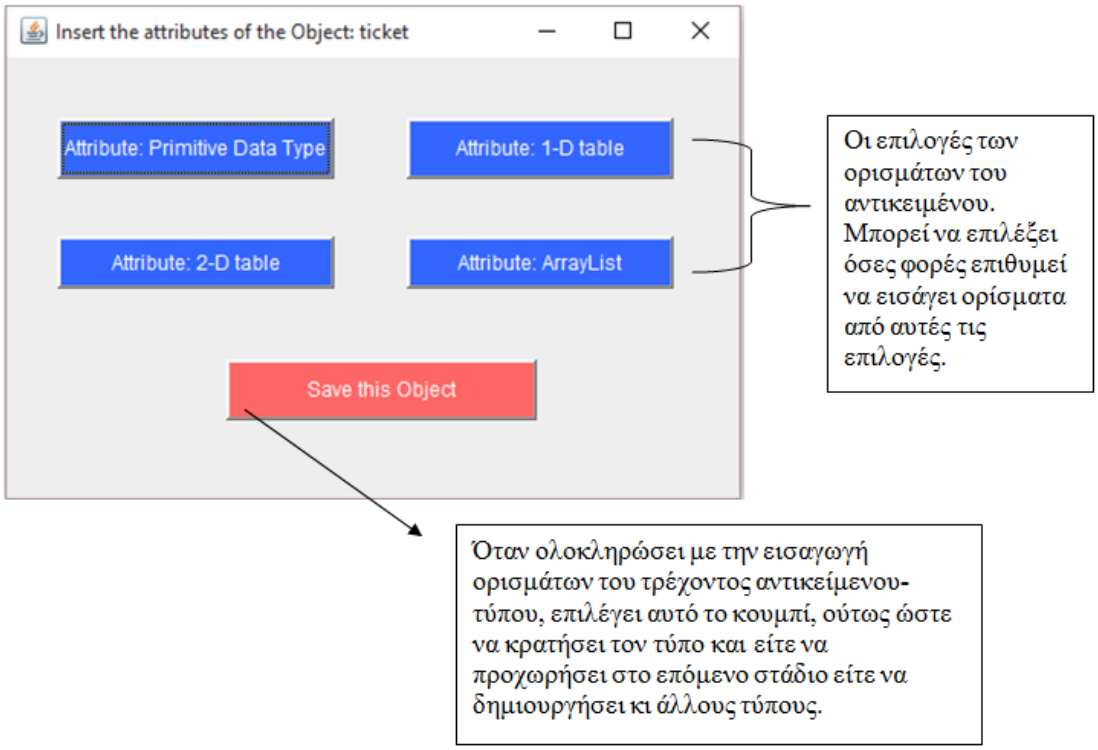


1.1: Επιλογή Insert Base Type

Όταν ο χρήστης επιλέξει να εισάγει αντικείμενο, τότε θα του ζητηθεί το όνομα του αντικειμένου. Το παράθυρο δεν μπορεί να κλείσει αν ο χρήστης δώσει κενό όνομα.

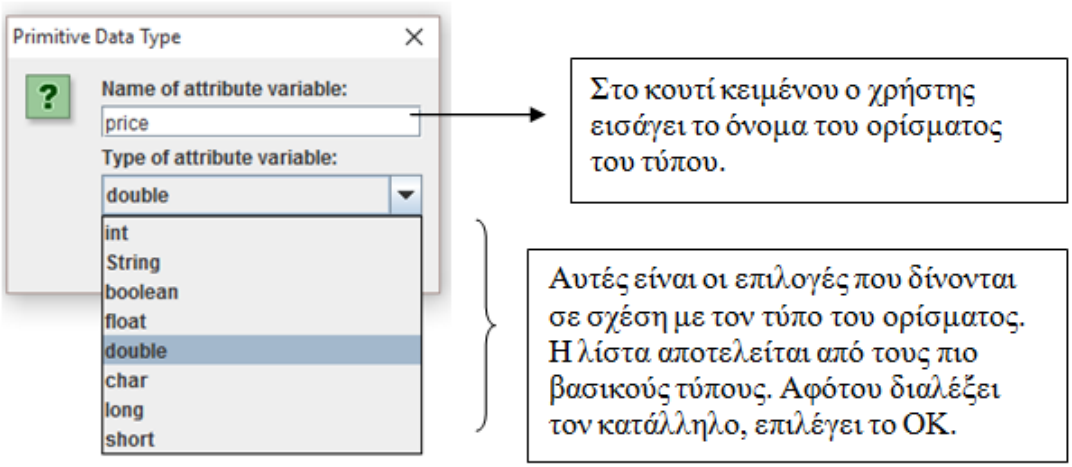


Όταν ο χρήστης εισάγει το όνομα του αντικειμένου τότε θα εμφανισθεί στην οθόνη ένα παράθυρο για την επιλογή των χαρακτηριστικών του βασικού αυτού τύπου. Μπορεί να επιλέξει να είναι βασικός τύπος (πχ ακέραιος, λέξη, δυαδικός), να καθορίσει έναν μονοδιάστατο ή δισδιάστατο πίνακα ή ακόμα και να δημιουργήσει μία δομή τύπο Array List.



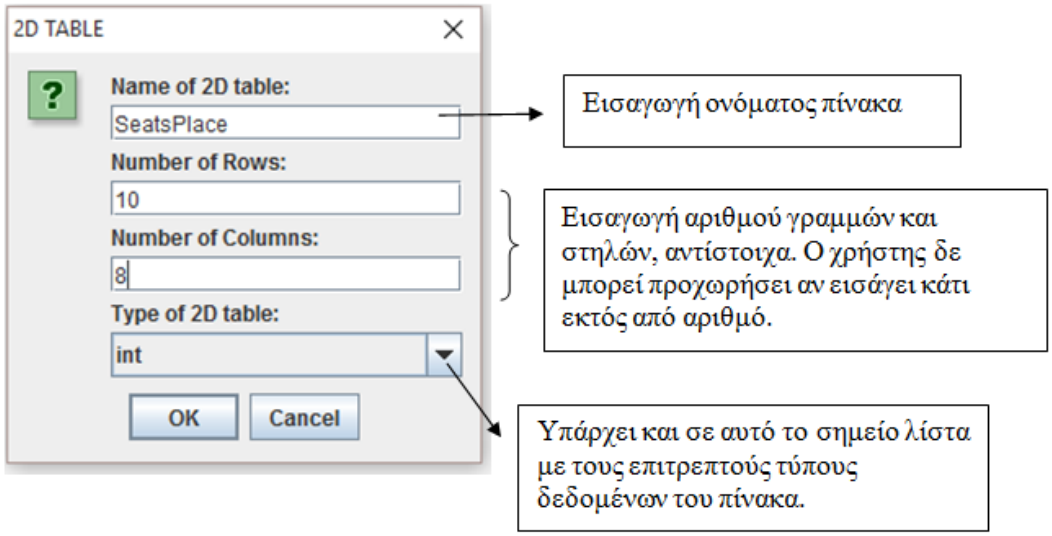
Εισαγωγή ορίσματος απλού τύπου

Όταν ο χρήστης επιλέξει “Attribute: Primitive Data Type”, τότε εμφανίζεται στην οθόνη το κατάλληλο παράθυρο που του ζητά να δώσει τον τύπο του ορίσματος, μέσω μίας απλής λίστας αλλά και να εισάγει το όνομα του ορίσματος.



Εισαγωγή ορίσματος ως δυσδιάστατος πίνακας

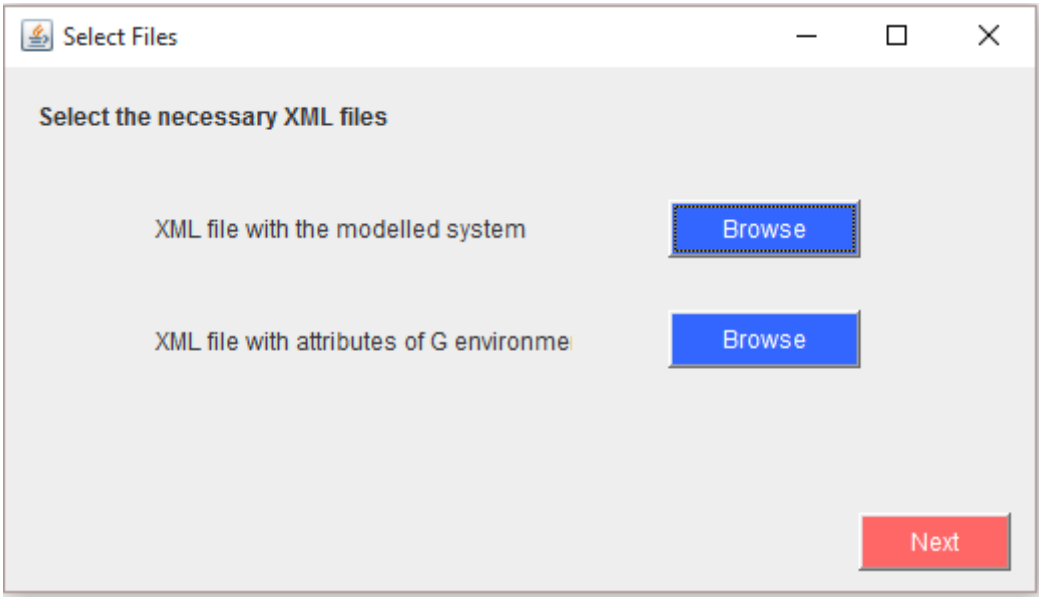
Όταν ο χρήστης επιλέξει “Attribute: 2D table”, τότε θα έχει τη δυνατότητα να εισάγει ένα δυσδιάστατο πίνακα. Θα καθορίσει όπως και προηγουμένως το όνομα του ορίσματος κι έπειτα τον αριθμό γραμμών και στηλών του πίνακα.



(αντίστοιχα ισχύουν και στο μονοδιάστατο πίνακα και στην εισαγωγή Array List)

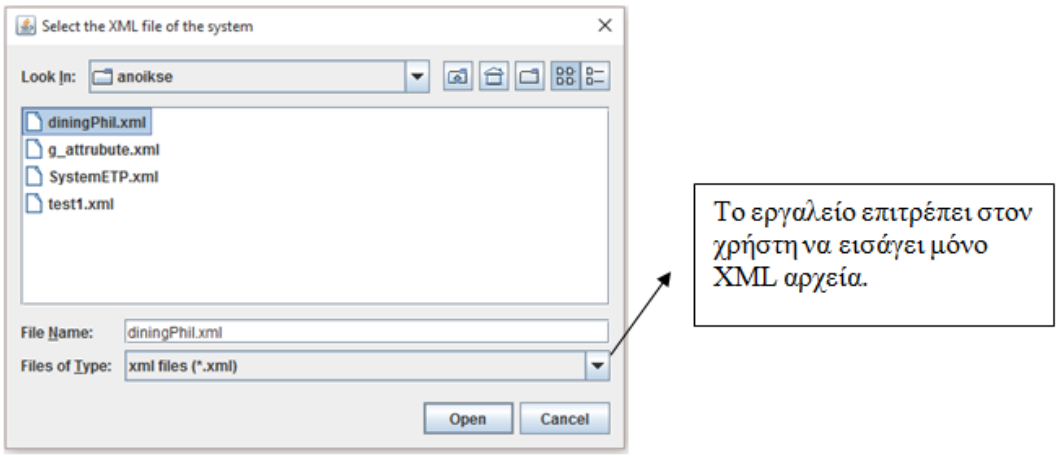
2. Εισαγωγή αναγκαίων αρχείων XML

Όταν ο χρήστης επιλέξει το “Next” στο παράθυρο του σημείου 1, τότε θα προχωρήσει στο επόμενο στάδιο, όπου θα πρέπει να εισάγει τα απαραίτητα αρχεία XML. Αν δεν τα εισάγει, το εργαλείο δεν του επιτρέπει να προχωρήσει με το “Next”, εκτός και αν επιλέξει την έξοδο (X)που τότε το πρόγραμμα τερματίζεται.



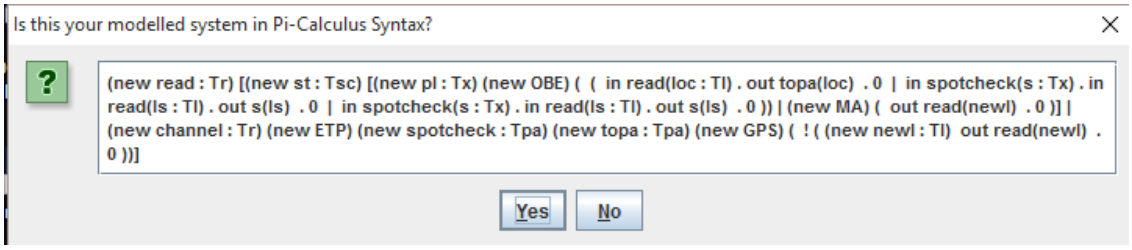
Όταν επιλέξει το πρώτο “Browse”, τότε θα εισάγει το XML αρχείο που αφορά το μοντελοποιημένο σύστημα διεργασιών το οποίο επιθυμεί να υλοποιηθεί από το εργαλείο. Όταν επιλέξει το δεύτερο, τότε πρέπει να εισάγει το XML που αφορά το αρχείο που περιέχει τα κανάλια του περιβάλλοντος Γ, στο οποίο θεωρείται ότι τρέχει και αλληλεπιδρά το μοντελοποιημένο σύστημα σε π-λογισμό.

Πιο κάτω εμφανίζεται το παράθυρο εισαγωγής αρχείου που θα εμφανιστεί στον χρήστη.

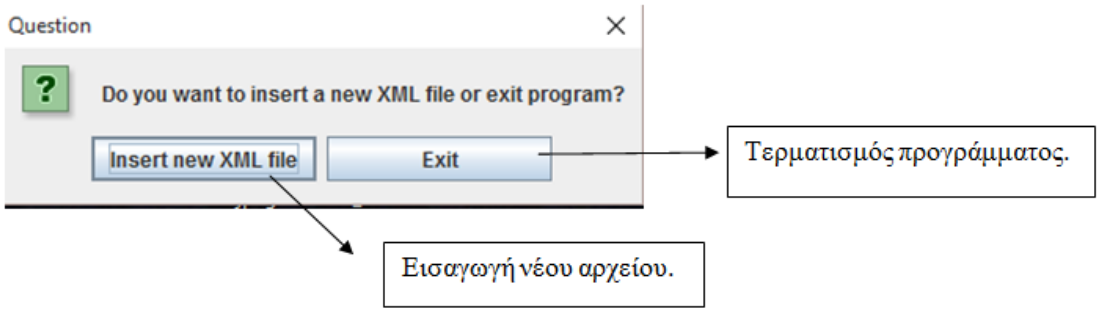


3. Όταν ο χρήστης εισάγει και τα δύο αναγκαία XML αρχεία, τότε το εργαλείο ελέγχει τη σύνταξη του συστήματος. Αν υπάρχει λάθος, ενημερώνει το χρήστη για το σημείο στο οποίο υπάρχει το λάθος στη σύνταξη του XML και του επιτρέπει είτε να τερματίσει, είτε να δώσει νέα αρχεία. Αν όμως η σύνταξη είναι εντάξει, τότε το μοντελοποιημένο σύστημα εμφανίζεται σε μορφή σύνταξης π-λογισμού ούτως ώστε να επιβεβαιώσει ο χρήστης ότι αυτό είναι το σύστημα που θέλει να υλοποιήσει.

Για παράδειγμα θα του εμφανιστεί κάτι ανάλογο:



Αν επιλέξει “No”, δηλαδή δεν επιβεβαιώσει ότι αυτό είναι το συστημα π-λογισμού που επιθυμεί να υλοποιήσει, τότε σημαίνει ότι ο χρήστης έχει κάνει κάποιο λογικό λάθος στη διατύπωση του συστήματος. Οπότε του ζητείται είτε να δώσει νέο αρχείο είτε να τερματίσει το πρόγραμμα.



Ολοκλήρωση Εκτέλεσης:

Όταν ο χρήστης εισάγει ένα σωστό συντακτικά αρχείο και επιβεβαιώση και την ορθότητα του μοντέλου βασιζόμενος στην εμφανιζόμενη διατύπωσή του σε π-λογισμό, τότε το πρόγραμμα υλοποιεί τη μετάφραση του μοντελοποιημένου συστήματος σε κώδικα και αποθηκεύει τις απαραίτητες παραγόμενες κλάσσεις με τον κώδικα στον τρέχον κατάλογο.

Ο κώδικας του εργαλείου μπορεί να βρεθεί στην πιο κάτω ηλεκτρονική διεύθυνση:

https://github.com/etingi01/PiCalculusJava

Κεφάλαιο 5

Παραδείγματα εκτέλεσης του εργαλείου

5.1 Εισαγωγή 96
5.2 Παράδειγμα δειπνούντων φιλοσόφων 97
5.3 Παράδειγμα συνομιλίας σε chat room 101
5.4 Παράδειγμα ηλεκτρονικής τιμολόγησης κυκλοφορίας 109

5.1 Εισαγωγή

Σε αυτό το κεφάλαιο, θα μελετηθούν παραδείγματα πραγματικής εφαρμογής και χρήσης του εργαλείου που αναπτύχθηκε στα πλάισια της εργασίας. Θα δοθεί μία πλήρης περιγραφή και επεξήγηση του κάθε προβλήματος. Έπειτα θα επιχειρηθεί, το σύστημα που προορίζεται για τη λύση του προβλήματος, να μοντελοποιηθεί στη σύνταξη της επέκτασης του π-λογισμού με ομάδες με την οποία ασχολείται η διπλωματική εργασία.
Σε επόμενο στάδιο, θα δοθεί η μοντελοποίηση του συστήματος και σε γλώσσα XML που θα υπακούει στον οδηγό συγγραφής που δόθηκε στο προηγούμενο κεφάλαιο.

Ακολούθως θα εμφανίζονται και θα σχολιάζονται οι παραγόμενες κλάσεις με τον κώδικά τους μετά το στάδιο της μετάφρασης, όπως επίσης και το αποτέλεσμα της εκτέλεσης του παραγόμενο κώδικα.

Το κεφάλαιο αυτό είναι σημαντικό τόσο για να συμπληρώσει την πλήρη κατανόηση του εργαλείου και του τρόπου χειρισμού του αλλά και στο επισημάνει τη σημαντικότητα του εργαλείου που μπορεί να χρησιμοποιηθεί για την αυτόματη υλοποίηση μοντελοποιημένων παράλληλων συστημάτων, ακόμα και χωρίς προγραμματιστικές γνώσεις του χρήστη.

5.1 Παράδειγμα δειπνούντων φιλοσόφων

Περιγραφή

Το πρόβλημα αυτό αναφέρεται στην προσπάθεια ενός αριθμού φιλοσόφων να γευματίσουν χρησιμοποιώντας ο καθένας δύο πιρούνια. Αυτό δε θα αποτελούσε πρόβλημα αν στο τραπέζι υπήρχε ο απαραίτητος αριθμός από πιρούνια. Ανάμεσα όμως σε ζεύγη γειτονικών πιάτων βρίσκεται ένα πιρούνι. Ο κάθε φιλόσοφος περνάει από εναλλασσόμενες περιόδους σκέψης και φαγητού. Αν ο φιλόσοφος αποκτήσει και τα δύο πιρούνια τρωει και στη συνέχεια τα αφήνει στο τραπέζι για να σκεφτεί. Δυστυχώς όμως η προσπάθεια του φιλοσόφου να φαει δεν αποδίδει πάντα, καθώς ακόμα και αν πιάσει το ένα πιρούνι το άλλο μπορεί να είναι κατειλημμένο από το διπλανό του. Το πρόβλημα αυτό είναι από τα βασικότερα στην χρονομοδρολόγηση διεργασιών από ένα Λειτουργικό σύστημα και στην σωστή διαχείριση των πόρων του συστήματος.

Μοντελοποίηση προβλήματος στο π-λογισμό

Θα μοντελοποιήσουμε το πρόβλημα για 4 φιλόσοφους και 4 πιρούνια, όπου το κάθε πιρούνι διαμοιράζεται μεταξύ δύο γειτονικών φιλοσόφων. Οι φιλόσοφοι μπορούν να σκεφτούν και να φάνε. Για να φάει κάποιος φιλόσοφος πρέπει να αποκτήσει 2 πιρούνια, ξεκινώντας από αυτό στα αριστερά κι έπειτα αυτό στα δεξιά. Όλοι οι φιλόσοφοι συμπεριφέρονται το ίδιο, οπότε το πρόβλημα είναι συμμετρικό. Για να μην δημιουργηθεί κίνδυνος αδιεξόδου, μία απλή λύση είναι να εξαναγκάσουμε την απόκτηση 2 πιρουνιών σε ατομικότητα.

Η διεργασία που αφορά τα πιρούνια θα είχε ως εξής:

$$fork_i \;=\; \overline{up}_i\langle f_i\rangle.\overline{dn}_i\langle f_i\rangle.fork_i \quad \text{με i=1-4}$$

Η διεργασία που αφορά τον κάθε φιλόσοφο θα ήταν η εξής:

$Phil_i = think.phil_i +$

$up_i(fp_i{:}\,fork).up_{i+1}(fp_i{:}\,fork).eat.dn_i(fp_i{:}\,fork).dn_{i+1}(fp_i{:}\,fork)$

Η παράλληλη σύνθεση του συστήματος με τις απαραίτητες δημιουργίες ομάδων στον λογισμό είναι:

$(v\ diningPhils)(v\ up_1 : fup)(v\ up_2 : fup)(v\ up_3 : fup)(v\ up_4 : fup)$
$(v\ dn_1 : fdown)(v\ dn_2 : fdown)(v\ dn_3 : fdown)(v\ dn_4 : fdown)((((((Phil_1|Phil_2)|$
$Phil_3)|Phil_4)|\ fork_1)\ |fork_2)\ |fork_3)|\ fork_4)$

Όπως βλέπουμε πιο πάνω, ο για να πάρει ένας φιλόσοφος το πιρούνι πρέπει να γίνει ενέργεια εισόδου up, ενώ για να το αφήσει πρέπει να γίνει ενέργεια εισόδου dn στη διεργασία του φιλόσοφου. Αυτό μας βοηθά να επιβαιβεώσουμε ότι δεν θα υπάρξουν λάθη του να πάρουν δύο φιλόσοφοι ταυτόχρονα το ίδιο πιρούνι, αφού η διεργασία του πιρουνιού, δεν προχωράει σε επόμενη ενέργεια εξόδου μέσω του up, αν δεν πραγματοποιηθεί ενέργεια εξόδου dn, δηλαδή αν δεν φάνει αυτός που έχει πάρει το πιρούνι.

Το αρχείο XML δημιουργείται με την τήρηση των κανόνων διατύπωσης που αναφέρθηκαν στον οδηγό χρήσης. Δημιουργούνται κάποια επιπρόσθετα κανάλια που βοηθούν στην επικοινωνία, χωρίς να επηρεάζουν την ορθότητα.

Για να μπορεί η XML διατύπωση να είναι ευανάγνωστη, θα δείξουμε ξεχωριστά τη διεργασία για τον 1ο φιλόσοφο και ξεχωριστά τη διεργασία για το 1ο πιρούνι.

Διεργασία 1ου πιρουνιού:

```
<out>
	<subj>u1</subj>
	<obj>f1</obj>
	<process>
		<out>
			<subj>d1</subj>
			<obj>f1</obj>
			<process>
				<Nil>0</Nil>
			</process>
		</out>
	</process>
</out>
```

Διεργασία 1ου φιλοσόφου:

```
<inExternal>
	<subj>think</subj>
	<obj>value</obj>
	<type>Integer</type>
	<process>
		<in>
		<subj>up1</subj>
		<obj>fork</obj>
```

```
                  <type>Integer</type>
                  <process>
                        <in>
                        <subj>up2</subj>
                        <obj>fork2</obj>
                        <type>Integer</type>
                        <process>
                              <inExternal>
                              <subj>eat</subj>
                              <obj>value2</obj>
                              <type>Integer</type>
                              <process>
                                    <in>
                                    <subj>d1</subj>
                                    <obj>fork1</obj>
                                    <type>Integer</type>
                                    <process>
                                          <in>
                                          <subj>d2</subj>
                                          <obj>fork3</obj>
                                          <type>Integer</type>
                                    <process>
                                          <Nil>0</Nil>
                                    </process>
                              </in>
                        </process>
                  </in>
            </process>
</inExternal>
</process>
</in>
</process>
</in>
</process>
</inExternal>
```

Λόγω του ότι στην επέκταση π-λογισμού που υλοποιήσαμε, δεν υπάρχει ο τελεστής ‘+’ που αντιπροσωπεύει την επιλογή μεταξύ δύο διεργασιών, χρειάστηκε να γίνουν προσθέσεις στον κώδικά μας ούτως ώστε να λειτουργήσει με την επιθυμητή συμπεριφορά. Φυσικά, αυτό δεν μεταβάλλει τον τρόπο με τον οποίο γίνονται οι επικοινωνίες στο παράλληλο σύστημα των φιλοσόφων. Ο κώδικας που προστέθηκε έχει να κάνει με την υλοποιήση της φάσης όπου ο φιλόσοφος σκέφτεται βάζοντας μία χρονοκαθυστέρηση πριν την συνεχόμενη πρόσβαση σε πιρούνι, με την υλοποίηση του τελεστή ‘+’ χρησιμοποιώντας if-statement, με τη χρήση ατέρμονης επαναληπτικής δομής για την ακριβή αναπαράσταση του προβλήματος και με τον τρόπο παρουσίασης των αποτελεσμάτων ούτως ώστε να είναι πιο ευπαρουσίαστα τα αποτελέσματα.

Παρουσίαση κώδικα

Καθώς λόγω των τεχνικών που χρησιμοποιήθηκαν για την υλοποίηση, δημιουργούνται πολλές, αναγκαίες μεν αλλά πανομοιότυπες, κλάσεις, και σε αυτό το σημείο θα παρουσιαστεί ο κώδικας που παράγεται εντός της συνάρτησης run() που τρέχει τον 1ο φιλόσοφο και ο κώδικας της συνάρτησης run() που τρέχει το 1ο πιρούνι. Επιλέγονται αυτές οι συναρτήσεις καθώς αυτές είναι που στηνουσία υλοποιούν το επικοινωνιακό κομμάτι του συστήματος.

Συνάρτηση run() στην κλάση του 1ου φιλοσόφου:

```
public void run() {
             while(true){

if(!(up1.channel.in().pending()&&up2.channel.in().pending())){
       ChannelValue value = new ChannelValue();
       value.type = "Integer";
       System.out.println("Phil 1 is thinking");
}else{
      1 ChannelValue fork = new ChannelValue();
       fork.type = "Integer";
       System.out.println("Phil 1 will take the up1");
      2 fork = (ChannelValue) up1.channel.in().read();
       System.out.println("Phil 1 took the up1");
      1 ChannelValue fork2 = new ChannelValue();
       fork2.type = "Integer";
      2 fork2 = (ChannelValue) up2.channel.in().read();
      1 ChannelValue value2 = new ChannelValue();
       value2.type = "Integer";
       String namevalue2 =  JOptionPane.showMessageDialog(null, "Phil
1 is eating ");
      1 ChannelValue fork1 = new ChannelValue();
       fork1.type = "Integer";
      3 fork1 = (ChannelValue) d1.channel.in().read();
      1 ChannelValue fork3 = new ChannelValue();
       fork3.type = "Integer";
      3 fork3 = (ChannelValue) d2.channel.in().read();
       try {
            Thread.sleep(5000);
            } catch (InterruptedException e1) {
            e1.printStackTrace();
            }
       }
 }
}
```

Ο προστιθέμενος κώδικας που έχει μπει για διευκόλυνση στην κατανόηση των αποτελεσμάτων είναι μορφοποιημένος σε πλάγια γράμματα. Στις γραμμές με τον αριθμό 1 δημιουργούνται τα δεσμευμένα κανάλια. Στις γραμμές με τον αριθμό 2, σηκώνει το αντίστοιχο πιρούνι ο φιλόσοφος, ενώ σε αυτές με τον αριθμό 3, το

κατεβάζει. Στο τέλος, η συνάρτηση `Thread.sleep(5000);` καλείται για την υλοποιίηση του think. Όταν ο φιλόσοφος τρώει, εμφανίζεται ένα παράθυρο στο χρήστη.

Συνάρτηση run() στην κλάση του 1ου πιρουνιού:

```
public void run() {
            while(true){
                  System.out.println("up1 will be sent from fork");
                1 up1.channel.out().write( f1 );
                  System.out.println("up1 sent from fork");
                2 d1.channel.out().write( f1 );
            }
      }
```

Ο προστιθέμενος κώδικας που έχει μπει για διευκόλυνση στην κατανόηση των αποτελεσμάτων είναι μορφοποιημένος σε πλάγια γράμματα. Όπως βλέπουμε στη γραμμή 1, το πιρούνι «επιτρέπει» με την ενέργεια εξόδου στον έτοιμο φιλόσοφο να το πάρει και γίνεται ξανά διαθέσιμο αφότου ο φιλόσοφος το αφήσει (γραμμή 2).

5.2 Παράδειγμα συνομιλίας σε chat room

Περιγραφή

Πέρα όμως από την εφαρμογή του συστήματος σε κάποιο κλασσικό πρόβλημα της πληροφορικής, είναι σημαντικό να παρουσιαστεί κάποιο παράδειγμα που περιέχει την αναγκαιότητα για τήρηση της ιδιωτικότητας. Υποθέτουμε την ύπαρξη ενός διαδικτυακού χώρου συνομιλίας (chat room) όπου σε κάθε συνομιλία μπορούν να συμμετέχουν από 2 ή περισσότερα άτομα. Στο παρόν παράδειγμα, οι συνδεδεμένοι χρήστες στο chat room είναι τρεις: η Ελένη, ο Μιχάλης και ο Γιώργος. Αρχικά η Ελένη διατηρεί μία προσωπική συνομιλία ανταλλαγής μηνυμάτων με το Μιχάλη με τη χρήση δεσμευμένου καναλιού, αλλά έπειτα χρησιμοποιώντας το κανάλι που είναι δεσμευμένο σε ολόκληρο το σύστημα θα εντάξει στη συνομιλία της με το Μιχάλη και το Γιώργο. Κι έτσι, οτιδήποτε στέλνεται σε αυτή τη συνομιλία έχουν δικαίωμα πρόσβασης και ανάγνωσης και οι τρεις. Το συγκεκριμένο παράδειγμα μπορεί να θεωρηθεί μία μοντελοποίηση ενός γεγονότος που είναι καθημερινά στη ζωή μας, λόγω της χρήσης των κοινωνικών δικτύων αλλά και εφαρμογών που επιτρέπουν την ύπαρξη συνομιλιών με περισσότερα από δύο άτομα, εφόσον κάποιος από τα άτομα στη συνομιλία εντάξει και κάποιον άλλον.

Μοντελοποίηση προβλήματος στο π-λογισμό

Θα μοντελοποιήσουμε το σύστημα χρησιμοποιώντας τη δημιουργία ομάδων συστήματος αλλά και ομάδων διεργασιών. Έχουμε αρχικά τρεις ομάδες διεργασιών, που αντιστοιχούν μία σε κάθε συνδεδεμένο χρήστη στο chat room. Η ομάδα διεργασιών της Ελένης και του Μιχάλη, υπάγονται στην ομάδα επιπέδου συστήματος ΜΕ που αντιπροσωπεύει την προσωπική τους συνομιλία των δύο ατόμων. Η ομάδα ΜΕ έχει ένα δεσμευμένο κανάλι, το priv (private) ούτως ώστε η συνομιλία τους μην είναι ορατή από τρίτους. Η ομάδα ΜΕ και η ομάδα διεργασίας του Γιώργου είναι σε παράλληλη σύνθεση και ο μόνος τρόπος να επικοινωνήσουν είναι μέσω του καναλιού x που είναι δεσμευμένο σε ολόκληρο το σύστημα. Μέσω αυτού του καναλιού λοιπόν, η Ελένη θα γνωστοποιήσει στο Γιώργο το ιδιωτικό κανάλι που διατηρεί με το Μιχάλη κι έτσι κι ο Γιώργος πλέον θα μπορεί να είναι μέλος της αλληλοεπικοινωνίας τους και να τους στείλει ή να λάβει τα μηνύματά τους.

Τύποι: Οι βασικός τύπος του παραδείγματος είναι το **message** που μπορούμε να θεωρήσουμε ότι είναι τύπου String. Έπειτα έχουμε και μη βασικούς τύπους όπως το **cm** = ChatRoom[message] και το **channel** = ChatRoom[cm]

$(\nu\ ChatRoom)(\nu\ x{:}\ channel)[[(\nu\ ME)(\nu\ priv{:}\ cm)((\nu\ E)\ ((\nu\ e{:}\ message)\overline{priv}\langle e\rangle.\ priv(z{:}\ message).\overline{x}\langle priv\rangle.\ priv(l{:}\ message))$
$|\ (\nu\ M)(priv(b{:}\ message).\ (\nu\ p{:}\ message)\overline{priv}\langle p\rangle.\ priv(k{:}\ message)))]$
$|\ (\nu\ G)\ (x(m{:}\ cm).\ (\nu\ d{:}\ message).\ \overline{m}\langle d\rangle.\ \overline{m}\langle d\rangle)]$

Το αρχείο XML δημιουργείται με την τήρηση των κανόνων διατύπωσης που αναφέρθηκαν στον οδηγό χρήσης:

```
<resGS>
        <group>ChatRoom</group>
        <system>
                <resNS>
                <name>x</name>
                <type>channel</type>
                <system>
                        <parS>
                        <system>
                        <resGS>
                        <group>ME</group>
                        <system>
                        <resNS>
                        <name>priv</name>
                        <type>cm</type>
```

<system>
<parS>
<system>
<resGP>
<group>Michael</group>
<process>
<in>
subj>priv</subj>
<obj>b</obj>
<type>message</type>
<process>
<resNP>
<name>p</name>
<type>message</type>
<process>
<out>
<subj>priv</subj>
<obj>p</obj>
<process>
<in>
<subj>priv</subj>
<obj>k</obj>
<type>message</type>
<process>
<Nil>0</Nil>
</process>
</in>
</process>
</out>
</process>
</resNP>
</process>
</in>
</process>
</resGP>
</system>
<system>
<resGP>
<group>Eleni</group>
<process>
<resNP>
<name>e</name>
<type>message</type>
<process>
<out>
<subj>priv</subj>
<obj>e</obj>
<process>
<in>
<subj>priv</subj>
<obj>z</obj>
<type>message</type>
<process>
<out>

<subj>x</subj>
<obj>priv</obj>
<process>
<in>
<subj>priv</subj>
<obj>l</obj>
<type>message</type>
<process>
<Nil>0</Nil>
</process>
</in>
</process>
</out>
</process>
</in>
</process>
</out>
</process>
</resNP>
</process>
</resGP>
</system>
</parS>
</system>
</resNS>
</system>
</resGS>
</system>
<system>
<resGP>
<group>George</group>
<process>
<in>
<subj>x</subj>
<obj>m</obj>
<type>channel</type>
<process>
<resNP>
<name>d</name>
<type>message</type>
<process>
<out>
<subj>m</subj>
<obj>d</obj>
<process>
<out>
<subj>m</subj>
<obj>d</obj>
<process>
<Nil>0</Nil>
</process>
</out>
</process>
</out>

```
        </process>
        </resNP>
        </process>
        </in>
        </process>
        </resGP>
        </system>
        </parS>
        </system>
        </resNS>
        </system>
</resGS>
```

Παρουσίαση κώδικα

Για την υλοποίηση του πιο πάνω παραδείγματος, αναπτύχθηκε ο κώδικας 6 κλάσεων. Πρώτα απ' όλα, ανατπύχθηκε η κλάση του βασικού τύπου (message) με το όρισμα String και τις βασικές συναρτήσεις get και set. Έπειτα δημιουργήθηκαν οι κλάσεις των ομάδων διεργασιών, του Γιώργου, της Ελένης και του Μιχάλη. Δημιουργήθηκε επίσης η κλάση που δημιουργεί την ομάδα σύνθεσης συνομιλίας του Μιχάλη και της Ελένης και βέβαια και η κλάση για τη σύνθεση του συστήματος Chat room.

Κλάσση Βασικού Τύπου message

```
import java.util.ArrayList;
public class message {
private String content;
public message(){
}
public String getcontent() {
return content;
}
public void setcontent(String parameter ) {
content = parameter;
}
}
```

Κλάση διεργασίας Ελένης:

```
import org.jcsp.lang.*;
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
public class ProcessEleni implements CSProcess {
        public String nameProcess = "ProcessEleni";
        public ChannelValue x ;
        public ChannelValue priv ;
        public ProcessEleni( ChannelValue x, ChannelValue priv){
```

```
                this.x = x;
                this.priv = priv;
        }
        public void run() {
                message e = new message ();
                e.setcontent("Geia sou Mixali!");
                priv.channel.out().write( e );
                System.out.println("Eleni sent: " +
e.getcontent());
                message  z = new message();
                z = (message) priv.channel.in().read();
                System.out.println("Eleni received: " +
z.getcontent());
                x.channel.out().write( priv );
                System.out.println("Eleni added George in the
conversation");
                message  l = new message();
                l = (message) priv.channel.in().read();
                System.out.println("Eleni received: " +
l.getcontent());
        }
}
```

Κλάση διεργασίας Μιχάλη:

```
import org.jcsp.lang.*;
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
public class ProcessMichael implements CSProcess {
        public String nameProcess = "ProcessMichael";
        public ChannelValue x ;
        public ChannelValue priv ;
        public ProcessMichael( ChannelValue x, ChannelValue priv){
                this.x = x;
                this.priv = priv;
        }
        public void run() {
                message  b = new message();
                b = (message) priv.channel.in().read();
                System.out.println("Mixalis received: " +
b.getcontent());
                message p = new message ();
                p.setcontent("Geia sou Eleni! Vale k to Giwrgo sti
sinomilia mas");
                priv.channel.out().write( p );
                System.out.println("Mixalis sent: " +
p.getcontent());
                message  k = new message();
                k = (message) priv.channel.in().read();
                System.out.println("Mixalis received: "+
k.getcontent());
        }
}
```

Κλάση διεργασίας Γιώργου:

```
import org.jcsp.lang.*;
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
public class ProcessGeorge implements CSProcess {
	public String nameProcess = "ProcessGeorge";
	public ChannelValue x ;
	public ProcessGeorge( ChannelValue x){
			this.x = x;
	}
	public void run() {
			ChannelValue m = new ChannelValue();
			m.type = "channel";
			m = (ChannelValue) x.channel.in().read();
			System.out.println("Giwrgos: mpainei sti
sinomilia");
			message d = new message ();
			d.setcontent("Geia sas paidia!");
			m.channel.out().write( d );
			m.channel.out().write( d );
			System.out.println("Giwrgos sent: " +
d.getcontent());
	}
}
```

Κλάση ομάδας συνομιλίας του Μιχάλη και της Ελένης (ΜΕ):

```
import org.jcsp.lang.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.HashMap;
public class SystemME implements CSProcess {
	public String nameSystem = "SystemME";
		public ChannelValue x ;
		public SystemME( ChannelValue x){
			this.x = x;
		}
		public void run() {
			ChannelValue priv = new ChannelValue();
			priv.type = "cm";
			ProcessMichael MichaelGrPr = new ProcessMichael(x,
priv);
			CSProcess[] Michaelprocesses = new
CSProcess[]{MichaelGrPr};
			atomicGroup Michael = new
atomicGroup(Michaelprocesses, "Michael");
			ProcessEleni EleniGrPr = new ProcessEleni(x, priv);
			CSProcess[] Eleniprocesses = new
CSProcess[]{EleniGrPr};
```

```
                    atomicGroup Eleni = new atomicGroup(Eleniprocesses,
"Eleni");
                    CSProcess[] parParts = new
CSProcess[]{Michael,Eleni};
                    Parallel par = new Parallel(parParts);
                    par.run();
             }
      }
```

Κλάση σύνθεσης του συστήματος (ChatRoom):

```
import org.jcsp.lang.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.HashMap;
public class SystemChatRoom implements CSProcess {
      public String nameSystem = "SystemChatRoom";
            public SystemChatRoom(){
            }
            public void run() {
                  ChannelValue x = new ChannelValue();
                  x.type = "channel";
                  SystemME ME = new SystemME(x);
                  ProcessGeorge GeorgeGrPr = new ProcessGeorge(x);
                  CSProcess[] Georgeprocesses = new
CSProcess[]{GeorgeGrPr};
                  atomicGroup George = new
atomicGroup(Georgeprocesses, "George");
                  CSProcess[] parParts = new CSProcess[]{ME,George};
                  Parallel par = new Parallel(parParts);
                  par.run();
            }
            public static void main(String[] args){
                  SystemChatRoom system = new SystemChatRoom();
                  system.run();
            }
      }
```

Η συγκεκριμένη κλάση περιέχει τη μέθοδο main, γιατί αυτή είναι η κλάση από την οποία ο χρήστης θα τρέξει το υλοποιημένο σύστημα.

Στην κονσόλα αποτελεσμάτων θα πάρουμε κάτι αντίστοιχο:

```
Mixalis received: Geia sou Mixali!
Eleni sent: Geia sou Mixali!
Eleni received: Geia sou Eleni! Vale k to Giwrgo sti sinomilia mas
Mixalis sent: Geia sou Eleni! Vale k to Giwrgo sti sinomilia mas
Giwrgos: mpainei sti sinomilia
Eleni added George in the conversation
Mixalis received: Geia sas paidia!
Giwrgos sent: Geia sas paidia!
Eleni received: Geia sas paidia!
```

Όπως βλέπουμε στα αποτελέσματα, κάποιες φορές τυπώνεται πρώτα το μήνυμα από την πλευρά της λήψης του μηνύματος και μετά από την πλευρά αποστολής. Αυτό δεν είναι κάτι που θα πρέπει να μας ανησυχεί. Η επικοινωνία και η ανταλλαγή μηνυμάτων γίνεται ακριβώς ταυτόχρονα μέσω των καναλιών. Το φαινόμενο αυτό προκαλείται από το γεγονός ότι μετά την ταυτόχρονη επικοινωνία, κάθε διεργασία λειτουργεί ξανά ατομικά και τυπώνει στην πρώτη οθόνη όποια πάρει προτεραιότητα από το λειτουργικό σύστημα.

5.3 Παράδειγμα ηλεκτρονικής τιμολόγησης κυκλοφορίας [13]

Περιγραφή

Ένα σύστημα ηλεκτρονικής τιμολόγησης κυκλοφορίας υπολογίζει τη χρέωση που πρέπει να πληρώσει ένας οδηγός στις Αρχές ανάλογα με τη διαδρομή των οχημάτων, όπου παράφοντες όπως ο δρόμος και η ώρα χρήσης καθορίζουν τη χρέωση. Για να γίνει αυτό, για κάθε όχημα πρέπει να καταγραφεί η ώρα και η τοποθεσία ούτως ώστε μετά από επεξεργασία από τις αρχές που καθορίζουν τη χρέωση. Υπάρχουν αρκετές δυνατές υλοποιήσεις του συστήματος στο [20]. Εμείς θα μελετήσουμε μία τροποποιημένη εκδοχή της κεντρικής προσέγγισης του προβλήματος, όπου οι πληροφορίες της τοποθεσίας στέλνονται στην Αρχή Χρέωσης που υπολογίζει το ποσό που πρέπει να πληρωθεί βασιζόμενη στις πληροφορίες που δέχεται.

Γίνεται χρήση ενός μηχανισμού (OBE) που υπολογίζει ανά περιόδους τη γεωγραφική θέση του αυτοκινήτου και την προωθεί στην Αρχή Χρέωσης (PA). Η αρχή θα κάνει ελέγχους στο σημείο που λαμβάνει, ούτως ώστε να επιβεβαιώσει ότι έχει λάβει την ορθή τοποθεσία. Γι’ αυτό το σύστημα θεωρούμε ότι έχουμε πέντε ομάδες: την ETP που αντιστοιχεί σε ολόκληρο το σύστημα, την Car που αντιστοιχεί στο αυτοκίνητο διαιρεμένο στις ομάδες OBE και GPS, και την ομάδα PA που αντιστοιχεί στην αρχή χρέωσης.

Μοντελοποίηση προβλήματος στο π-λογισμό

Η ομάδα OBE θα αποτελείται από δύο παράλληλες διεργασίες και θα διαβάζει την τοποθεσία μέσω του καναλιού read από τη διεργασία της ομάδας GPS. Το κανάλι read είναι δεσμευμένο εντός της ομάδας Car οπότε μόνο αυτές οι δύο ομάδες έχουν πρόσβαση σε αυτό. Η ομάδα OBE μπορεί ταυτόχρονα είτε να διαβάζει την τοποθεσία

αλληλεπιδρώντας με την GPS μέσω του read είτε να να επικοινωνεί με την αρχή χρέωσης για επιβεβαίωση και έλεγχο της τοποθεσίας. Αυτός ο έλεγχος γίνεται από την αρχή χρέωσης που δέχεται αρχικά ένα κανάλι z από το μηχανισμό OBE του αυτοκινήτου (Car όμάδα), μέσω του καναλιού topa. Επίσης, μπορεί να υπολογίζει περιοδικά τη χρέωση και να την στέλνει μέσω ενός καναλιού σε ένα εξωτερικό σύστημα. Κάτι τέτοιο όμως θα απαιτούσε κανόνα Αντιγραφής (replication) και επειδή στη δική μας μετάφραση, η επικοινωνία με εξωτερικό σύστημα εμφανίζεται με παράθυρο στην οθόνη, θα εμφανίζονταν συνεχόμενα παράθυρα στην οθόνη, χωρίς να έχουμε τη δυνατότητα να δούμε τη ροή των υπόλοιπων διεργασιών. Οπότε στη σύνταξη, η συγκεκριμένη δραστηριότητα θα μπει στα πλαίσια ενός κανόνα αντιγραφής, αλλά δεν θα γίνει το ίδιο στο αρχείο XML. Επίσης η αρχή χρέωσης κάνει έλεγχο της τοποθεσίας του αυτοκινήτου, δημιουργεί ένα κανάλι και το στέλνει σε διεργασία της OBE και έπειτα περιμένει να λάβει πίσω την τοποθεσία για επιβεβαίωση.

Οι βασικοί τύποι του συστήματος είναι:

Loc: αντιστοιχεί στην τοποθεσία
Fee: αντιστοιχεί στην χρέωση

Έχουμε επίσης τα κανάλια: Tl = ETP[Loc], Tr=Car[Tl], Tpa = ETP[Tl], Tx = ETP[Tl], Tsc=ETP[Tx]

$(v\ ETP)(v\ spotcheck{:}\ Tsc)(v\ topa{:}\ Tpa)[(v\ PA)(!\ (topa(z{:}\ Tl).\ z(l{:}\ Loc).\ 0)$

$|\ !\ \overline{send}\langle fee\rangle.\ 0\ |!\ \left((v\ x{:}\ Tx)\overline{spotcheck}\langle x\rangle.\ x(y{:}\ Tl).\ 0\right)$

$|\ (v\ Car)((v\ read{:}\ Tr)\ ((v\ OBE)(!\ (read(loc{:}\ Tl).\ \overline{topa}\langle loc\rangle.\ 0)$

$|\ !\ spotcheck(s{:}\ Tx).\ read(ls{:}\ Tl).\ \overline{s}\langle ls\rangle.\ 0)$

$|\ (v\ GPS)\big(!\ (v\ newl\ :\ Tl)\overline{read}\langle newl\rangle.\ 0\big)))]$

Το αρχείο XML που δημιουργούμε για τα κανάλια του Γ-περιβάλλοντος του συστήματος είναι:

<g_attributes>fee:ETP[Fee],send:ETP[ETP[Fee]]</g_attributes>

Κι αυτό γιατί το κανάλι send στο σύστημα επικοινωνεί με εξωτερικό σύστημα.

Το αρχείο XML που δημιουργούμε για τη σύνταξη του συστήματος είναι:

```
 <resGS>
<group>ETP</group>
<system>
<resNS>
        <name>spotcheck</name>
        <type>Tsc</type>
        <name>topa</name>
        <type>Tpa</type>
        <system>
                <parS>
                <system>
                <resGP>
                <group>PA</group>
                <process>
                <parP>
                        <process>
                                <repl>
                                <process>
                                <in>
                                <subj>topa</subj>
                                <obj>z</obj>
                                <type>Tl</type>
                                <process>
                                <Nil>0</Nil>
                                </process>
                                </in>
                                </process>
                                </repl>
                        </process>
                        <process>
                                <outExternal>
                                <subj>send</subj>
                                <obj>fee</obj>
                                <process>
                                <Nil>0</Nil>
                                </process>
                                </outExternal>
                        </process>
                        <process>
                                <repl>
                                <process>
                                <out>
                                <subj>spotcheck</subj>
                                <obj>x</obj>
                                <process>
                                <in>
                                <subj>x</subj>
                                <obj>y</obj>
                                <type>Tl</type>
                                <process>
```

Είσοδος τοποθεσίας από το μηχανισμό OBE

Αποστολή χρέωσης σε εξωτερικό σύστημα

Έλεγχος τοποθεσίας

```
                        <Nil>0</Nil>
                        </process>
                        </in>
                        </process>
                        </out>
                        </process>
                        </repl>
                    </process>
            </parP>
            </process>
            </resGP>
            </system>
            <system>
            <resGS>
            <group>Car</group>
            <system>
            <resNS>
            <name>read</name>
            <type>Tr</type>
            <system>
            <parS>
            <system>
            <resGP>
            <group>OBE</group>
<process>
                    <parP>
                    <process>
                        <repl>
                        <process>
                        <in>
                        <subj>read</subj>
                        <obj>loc</obj>
                        <type>Tl</type>
                        <process>
                        <out>
                        <subj>topa</subj>
                        <obj>loc</obj>
                        <process>
                        <Nil>0</Nil>
                        </process>
                        </out>
                        </process>
                        </in>
                        </process>
                        </repl>
                    </process>
                    <process>
                        <repl>
                        <process>
                        <in>
                        <subj>spotcheck</subj>
                        <obj>s</obj>
                        <type>Tx</type>
                        <process>
```

Είσοδος θέσης από GPS και αποστολή στην PA

```
                    <in>
                    <subj>read</subj>
                    <obj>ls</obj>
                    <type>Tl</type>
                    <process>
                    <out>
                    <subj>s</subj>
                    <obj>ls</obj>
                    <process>
                    <Nil>0</Nil>
                    </process>
                    </out>
                    </process>
                    </in>
                    </process>
                    </in>
                    </process>
                    </repl>
                </process>
                </parP>
</process>
        </resGP>
        </system>
        <system>
        <resGP>
        <group>GPS</group>
                <process>
                <repl>
                    <process>
                    <resNP>
                    <name>newl</name>
                    <type>Tl</type>
                    <process>
                    <out>
                    <subj>read</subj>
                    <obj>newl</obj>
                    <process>
                    <Nil>0</Nil>
                    </process>
                    </out>
                    </process>
                    </resNP>
                    </process>
                    </repl>
                </process>
        </resGP>
        </system>
        </parS>
        </system>
        </resNS>
        </system>
        </resGS>
        </system>
        </parS>
```

Επιβεβαίωση τοποθεσίας

Υπολογισμός θέσης από το GPS

</system>
</resNS>
</system>
</resGS>

Παρουσίαση κώδικα

Για την υλοποίηση του μοντελοποιημένου συστήματος, παράχθηκαν 15 κλάσεις που αφορούν το σύσημα. Δημιουργήθηκε η κλάση SystemETP που συνθέτει τις δύο ομάδες, PA και Car του συστήματος. Δημιουργήθηκε επίσης η κλάση SystemCar που συνθέτει τις δύο υπο-ομάδες: OBE και GPS σε παράλληλη σύνθεση. Για την κάθε ομάδα από αυτές τις δύο, δημιουργήθηκαν τόσες κλάσεις ParallelProc όσες είναι οι παράλληλες διεργασίες της. Οπότε οι κλάσεις ProcessOBE και ProcessGPS υλοποίησαν την παράλληλη σύνθεση των δικών τους ParallelProc. Επίσης, δημιουργήθηκαν τόσες κλάσεις όσοι είναι οι κανόνες αντιγραφής στο σύστημα, ούτως ώστε να μπορεί να γίνεται η αντιγραφή στην ατέρμονη επαναληπτική δομή που εξηγήθηκε σε προηγούμενο κεφάλαιο.

Κλάσεις βασικών τύπων:

```
import java.util.ArrayList;
public class Loc {
private String Street;
private int Number;
public Loc(){
}
public String getStreet() {
return Street;
}
public void setStreet(String
parameter ) {
Street = parameter;
}
public int getNumber() {
return Number;
}
public void setNumber(int
parameter ) {
Number = parameter;
}
}
```

```
import java.util.ArrayList;
public class Fee {
private int Price;
public Fee(){
}
public int getPrice() {
return Price;
}
public void setPrice(int parameter
) {
Price = parameter;
}
}
```

Κλάση σύνθεσης συστήματος ETP:

```
import org.jcsp.lang.*;
```

```
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.HashMap;
public class SystemETP implements CSProcess {
	public String nameSystem = "SystemETP";
	public ChannelValue x ;
		public SystemETP(){
			x = new ChannelValue();
			x.type = "unknown" ;
		}
		public void run() {
			ChannelValue topa = new ChannelValue();
			topa.type = "Tpa";
			ChannelValue spotcheck = new ChannelValue();
			spotcheck.type = "Tsc";
			ProcessPA PAGrPr = new ProcessPA(spotcheck, topa,
x);
			CSProcess[] PAprocesses = new CSProcess[]{PAGrPr};
			atomicGroup PA = new atomicGroup(PAprocesses,
"PA");
			SystemCar Car = new SystemCar(spotcheck, topa, x);
			CSProcess[] parParts = new CSProcess[]{PA,Car};
			Parallel par = new Parallel(parParts);
			par.run();
		}
		public static void main(String[] args){
			SystemETP system = new SystemETP();
			system.run();
		}
	}
```

Κλάσεις ομάδων PA (αρχή χρέωσης) και Car (οχήματος) αντίστοιχα:

```
import org.jcsp.lang.*;
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
public class ProcessPA implements
CSProcess {
	public String nameProcess =
"ProcessPA";
	public ChannelValue
spotcheck ;
	public ChannelValue topa ;
	public ChannelValue x ;
	public ProcessPA(
ChannelValue spotcheck,
ChannelValue topa, ChannelValue
x){
			this.spotcheck
= spotcheck;
			this.topa =
topa;
```

```
import org.jcsp.lang.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.HashMap;
public class SystemCar implements
CSProcess {
	public String nameSystem =
"SystemCar";
		public ChannelValue
spotcheck ;
		public ChannelValue
topa ;
	public ChannelValue x ;
		public SystemCar(
ChannelValue spotcheck,
ChannelValue topa, ChannelValue
x){
			this.x = x;
			this.spotcheck
= spotcheck;
```

```
                    this.x = x;
        }
        public void run() {
                    ParallelProc1
ParP0 = new
ParallelProc1(spotcheck, topa, x);
                    ParallelProc2
ParP1 = new
ParallelProc2(spotcheck, topa, x);
                    ParallelProc3
ParP2 = new
ParallelProc3(spotcheck, topa, x);
                    CSProcess[]
parParts = new
CSProcess[]{ParP0,ParP1,ParP2};
                    Parallel par =
new Parallel(parParts);
                    par.run();
   }
}
```

```
                    this.topa =
topa;
             }
             public void run() {
                    ChannelValue
read = new ChannelValue();
                    read.type =
"Tr";
                    ProcessOBE
OBEGrPr = new
ProcessOBE(spotcheck, topa, read,
x);
                    CSProcess[]
OBEprocesses = new
CSProcess[]{OBEGrPr};
                    atomicGroup OBE
= new atomicGroup(OBEprocesses,
"OBE");
                    ProcessGPS
GPSGrPr = new
ProcessGPS(spotcheck, topa, read,
x);
                    CSProcess[]
GPSprocesses = new
CSProcess[]{GPSGrPr};
                    atomicGroup GPS
= new atomicGroup(GPSprocesses,
"GPS");
                    CSProcess[]
parParts = new
CSProcess[]{OBE,GPS};
                    Parallel par =
new Parallel(parParts);
                    par.run();
             }
       }
```

Κλάσεις σύνθεσης ομάδας OBE και GPS αντίστοιχα:

```
import org.jcsp.lang.*;
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
public class ProcessOBE implements
CSProcess {
       public String nameProcess =
"ProcessOBE";
       public ChannelValue
spotcheck ;
       public ChannelValue topa ;
       public ChannelValue read ;
       public ChannelValue x ;
       public ProcessOBE(
ChannelValue spotcheck,
ChannelValue topa, ChannelValue
read, ChannelValue x){
```

```
import org.jcsp.lang.*;
import javax.swing.*;
import java.awt.*;
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
public class ProcessGPS implements
CSProcess {
       public String nameProcess =
"ProcessGPS";
       public ChannelValue
spotcheck ;
       public ChannelValue topa ;
       public ChannelValue read ;
       public ChannelValue x ;
       public ProcessGPS(
ChannelValue spotcheck,
ChannelValue topa, ChannelValue
read, ChannelValue x){
```

```
        this.spotcheck = spotcheck;
        this.topa = topa;
        this.read = read;
        this.x = x;
        }
        public void run() {
                        ParallelProc4
ParP0 = new
ParallelProc4(spotcheck, topa,
read, x);
                        ParallelProc5
ParP1 = new
ParallelProc5(spotcheck, topa,
read, x);
                        CSProcess[]
parParts = new
CSProcess[]{ParP0,ParP1};
                        Parallel par =
new Parallel(parParts);
                        par.run();
        }
}
```

```
        this.spotcheck = spotcheck;
        this.topa = topa;
        this.read = read;
        this.x = x;
        }
public void run() {
 while(true){
 ChannelValue newl = new
ChannelValue();
 newl.type = "Tl";
read.channel.out().write( newl );
System.out.println("Output action
through: read Placeholder: newl
");
Repl5 rep = new Repl5(topa, read,
spotcheck, newl, x);
ProcessManager manager = new
ProcessManager(rep);
manager.start();
                        }
        }
}
```

Κλάσεις παράλληλων διεργασιών της ομάδας OBE:

Στη δεύτερη γραμμή του πίνακα, εμφανίζονται οι αντίστοιχες κλάσεις που έχουν δημιουργηθεί για τον κανόνα αντιγραφής της κάθε διεργασίας.

```
import org.jcsp.lang.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.HashMap;
public class ParallelProc4
implements CSProcess {
        public String nameProcess =
"ParallelProc4";
        public ChannelValue
spotcheck ;
        public ChannelValue topa ;
        public ChannelValue read ;
        public ChannelValue x ;
        public ParallelProc4(
ChannelValue spotcheck,
ChannelValue topa, ChannelValue
read, ChannelValue x){
 this.spotcheck = spotcheck;
 this.topa = topa;
 this.read = read;
 this.x = x;
 }
 public void run() {
 while(true){
      ChannelValue loc = new
ChannelValue();
      loc.type = "Tl";
```

```
import org.jcsp.lang.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.HashMap;
public class ParallelProc5
implements CSProcess {
        public String nameProcess =
"ParallelProc5";
        public ChannelValue
spotcheck ;
        public ChannelValue topa ;
        public ChannelValue read ;
        public ChannelValue x ;
        public ParallelProc5(
ChannelValue spotcheck,
ChannelValue topa, ChannelValue
read, ChannelValue x){
this.spotcheck = spotcheck;
this.topa = topa;
this.read = read;
this.x = x;
}
public void run() {
 while(true){
 ChannelValue s = new
ChannelValue();
 s.type = "Tx";
```

```
        loc = (ChannelValue)
read.channel.in().read();
      System.out.println("Input
action through: read Placeholder:
loc ");
 Repl3 rep = new Repl3(loc, read,
topa, spotcheck, x);
 ProcessManager manager = new
ProcessManager(rep);

manager.start();
                    }
        }
}
```

```
 s = (ChannelValue)
spotcheck.channel.in().read();
System.out.println("Input action
through: spotcheck Placeholder: s
");
 Repl4 rep = new Repl4(s, read,
topa, spotcheck, x);
 ProcessManager manager = new
ProcessManager(rep);
manager.start();
                    }
        }
}
```

```
import org.jcsp.lang.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.HashMap;
public class Repl3 implements
CSProcess {
        public String nameProcess =
"Repl3";
        public ChannelValue loc ;
        public ChannelValue read ;
        public ChannelValue topa ;
        public ChannelValue
spotcheck ;
        public ChannelValue x ;
        public Repl3( ChannelValue
loc, ChannelValue read,
ChannelValue topa, ChannelValue
spotcheck, ChannelValue x){
 this.loc = loc;
 this.read = read;
 this.topa = topa;
 this.spotcheck = spotcheck;
 this.x = x;
 }
 public void run() {
 topa.channel.out().write( loc );
 System.out.println("Output action
through: topa Placeholder: loc ");
        }
}
```

```
import org.jcsp.lang.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.HashMap;
public class Repl4 implements
CSProcess {
        public String nameProcess =
"Repl4";
        public ChannelValue s ;
        public ChannelValue read ;
        public ChannelValue topa ;
        public ChannelValue
spotcheck ;
        public ChannelValue x ;
        public Repl4( ChannelValue
s, ChannelValue read, ChannelValue
topa, ChannelValue spotcheck,
ChannelValue x){
 this.s = s;
 this.read = read;
 this.topa = topa;
 this.spotcheck = spotcheck;
 this.x = x;
 }
 public void run() {
 ChannelValue ls = new
ChannelValue();
 ls.type = "Tl";
 ls = (ChannelValue)
read.channel.in().read();
 System.out.println("Input action
through: read Placeholder: ls ");
 s.channel.out().write( ls );
 System.out.println("Output action
through: s Placeholder: ls ");
        }
}
```

Κλάση της ομάδας GPS:

```
import org.jcsp.lang.*;
import javax.swing.*;
import java.awt.*;
```

```
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
public class ProcessGPS implements CSProcess {
	public String nameProcess = "ProcessGPS";
	public ChannelValue spotcheck ;
	public ChannelValue topa ;
	public ChannelValue read ;
	public ChannelValue x ;
	public ProcessGPS( ChannelValue spotcheck, ChannelValue topa,
ChannelValue read, ChannelValue x){
			this.spotcheck = spotcheck;
			this.topa = topa;
			this.read = read;
			this.x = x;
	}
	public void run() {
		while(true){
		ChannelValue newl = new ChannelValue();
		newl.type = "Tl";
		read.channel.out().write( newl );
		System.out.println("Output action through: read
Placeholder: newl ");
		Repl5 rep = new Repl5(topa, read, spotcheck, newl, x);
		ProcessManager manager = new ProcessManager(rep);
		manager.start();
		}
	}
}
```

Κλάση της ομάδας PA (αρχή χρέωσης) που δέχεται την τοποθεσία από την OBE:

```
import org.jcsp.lang.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.HashMap;
public class ParallelProc1 implements CSProcess {
	public String nameProcess = "ParallelProc1";
	public ChannelValue spotcheck ;
	public ChannelValue topa ;
	public ChannelValue x ;
	public ParallelProc1( ChannelValue spotcheck, ChannelValue
topa, ChannelValue x){
			this.spotcheck = spotcheck;
			this.topa = topa;
			this.x = x;
	}
	public void run() {
			while(true){
			ChannelValue z = new ChannelValue();
			z.type = "Tl";
			z = (ChannelValue) topa.channel.in().read();
			System.out.println("Input action through: topa
Placeholder: z ");
			Repl1 rep = new Repl1(z, topa, spotcheck, x);
			ProcessManager manager = new ProcessManager(rep);
			manager.start();
```

```
                }
        }
}
```

Κλάση της ομάδας PA (αρχή χρέωσης) που ελέγχει και επιβεβαιώνει την τοποθεσία του οχήματος (μαζί με την αναγκαία κλάση για τον κανόνα αντιγραφής):

```
import org.jcsp.lang.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.HashMap;
public class ParallelProc3
implements CSProcess {
 public String nameProcess =
"ParallelProc3";
 public ChannelValue spotcheck ;
 public ChannelValue topa ;
 public ChannelValue x ;
 public ParallelProc3(
ChannelValue spotcheck,
ChannelValue topa, ChannelValue
x){
 this.spotcheck = spotcheck;
 this.topa = topa;
 this.x = x;
}
public void run() {
 while(true){
 spotcheck.channel.out().write( x
);
 System.out.println("Output action
through: spotcheck Placeholder: x
");
 Repl2 rep = new Repl2(spotcheck,
topa, x);
 ProcessManager manager = new
ProcessManager(rep);
 manager.start();
                }
        }
}
```

```
import org.jcsp.lang.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.HashMap;
public class Repl2 implements
CSProcess {
 public String nameProcess =
"Repl2";
 public ChannelValue spotcheck ;
public ChannelValue topa ;
 public ChannelValue x ;
 public Repl2( ChannelValue
spotcheck, ChannelValue topa,
ChannelValue x){
 this.spotcheck = spotcheck;
 this.topa = topa;
 this.x = x;
 }
public void run() {
 ChannelValue y = new
ChannelValue();
 y.type = "Tl";
 y = (ChannelValue)
x.channel.in().read();
 System.out.println("Input action
through: x Placeholder: y ");
        }
}
```

Κλάση της ομάδας PA (αρχή χρέωσης) που στέλνει τη χρέωση σε εξωτερικό σύστημα (εμφάνιση μηνύματος στην οθόνη):

```
import org.jcsp.lang.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.HashMap;
public class ParallelProc2 implements CSProcess {
        public String nameProcess = "ParallelProc2";
        public ChannelValue spotcheck ;
        public ChannelValue topa ;
```

```
        public ChannelValue x ;
        public ParallelProc2( ChannelValue spotcheck, ChannelValue
topa, ChannelValue x){
                    this.spotcheck = spotcheck;
                    this.topa = topa;
                    this.x = x;
        }
        public void run() {
JOptionPane.showMessageDialog(null, "Communication with an external
system, through channel send object sent - fee : ");
        }
}
```

Κεφάλαιο 6

Συμπεράσματα

6.1 Γενικά Συμπεράσματα 122
6.2 Προβλήματα και μειονεκτήματα 123
6.3 Μελλοντικές Εργασίες 124

6.1 Γενικά Συμπεράσματα

Ο κύριος στόχος της εργασίας ήταν η υλοποίηση ενός εργαλείου που θα μπορεί να μεταφράζει ένα μοντελοποιημένο σύστημα π-λογισμού σε πρόγραμμα γλώσσα προγραμματισμού Java. Πέρα από τον βασικό αυτό στόχο, επιθυμούσαμε και την υλοποίση ενός εργαλείου που είναι προσιτό προς τον χρήστη και που ελαχιστοποιεί την πιθανότητα σφάλματος πραγματοποιώντας ελέγχους σε κάθε είσοδο του χρήστη, με σκοπό να αποκλειστούν όσες περισσότερες συνθήκες λάθους είναι εφικτό.

Βασικό μας συμπερασμα είναι ότι είναι όντως εφικτό ένα μοντέλο άλγεβρα διεργασιών, να έχει μία αντίστοιχα αυστηρή διατύπωση σε προγραμματιστικό επίπεδο, χωρίς να προκαλούνται ανεπιθύμητες συμπεριφορές στο πρόγραμμα. Εφόσον λοιπόν, βρήκαμε μία αντίστοιχα τυπική σύνταξη σε επίπεδο κώδικα, αυτό σημαίνει ότι ήταν εφικτή και η αυτόματή υλοποίησή του με δεδομένο ένα σύστημα σε π-λογισμό. Κύριος λόγος γι' αυτό είναι ότι η διατύπωση του κάθε όρου στο λογισμό, έχει μία συγκεκριμένη και προκαθορισμένη μετάφραση στον κώδικα.

Ένα επιπρόσθετο συμπέρασμα είναι ότι η τεχνική μετάφρασης για την τήρηση της ιδιωτικότητας έχει ως αποτέλεσμα τη δημιουργία κλάσεων που μπορεί να εκκινούν άλλες διεργασίες, αλλά οι ίδιες δεν περιλαμβάνουν διαδικασίες αλληλοεπικοινωνίας. Είναι μεν μία χρήσιμη τεχνική για την ιδιωτικότητα και την αυτοματοποίηση της διαδικασίας μετάφρασης, αλλά δημιουργεί πολλές κλάσεις που θα μπορούσαν να αποφευχθούν για να μη δισχαιρένεται η απόδοση από άποψη χώρου και πολυπλοκότητας κώδικα.

6.2 Προβλήματα και μειονεκτήματα

Ένα πρόβλημα που είχαμε κατά την υλοποίηση ήταν η μετονομασία καναλιών, καθώς σε μία πραγματική γλώσσα προγραματισμού μία μεταβλητή ή ένα αντικείμενο, δεν μπορεί να αλλάξει δυναμικά το όνομα με το οποίο παρουσιάζεται στον κώδικα. Γι' αυτό, έπρεπε να κινηθούμε σε άλλη κατεύθυνση για την μετατροπή της μετονομασίας σε κώδικα. Μετατρέποντας λοιπόν, όλα τα κανάλια και τις βασικούς τύπους σε αντικείμενα και όχι σε δεδομένα απλού τύπου, με την ανάθεση μπορούν να μετονομαστούν, εφόσον αυτό που αλλάζει είναι η αναφορά που δείχνουν στη μνήμη, επομένως με την ανάθεση, αναφέρονται σε διαφορετκό αντικείμενο (μετονομασία).

Ένα άλλο πρόβλημα που αντιμετωπίσαμε είναι η ύπαρξη ενέργειας εισόδου ή εξόδου στη μοντελοποιημένη διεργασία που όμως επικοινωνεί με το εξωτερικό περιβάλλον του συστήματος. Αν αυτές τις ενέργειες δεν τις διαχειριζόμασταν διαφορετικά, τότε θα υπήρχαν κίνδυνοι αδιεξόδων σε περίπτωση εκτέλεσης του παραγώμενου κώδικα. Οπότε, στην περίπτωση που έχουμε είσοδο από εξωτερικό σύστημα, αυτή την είσοδο θα πρέπει να την εισάγει ο χρήστης στον παραγώμενο κώδικα και όταν συνενωθεί και εκτελείται παράλληλα με το αναγκαίο εξωτερικό σύστημα, το σημείο αυτό μπορεί να μορφοποιηθεί. Βέβαια, και αυτή η αντιμετώπιση έχει τα προβλήματά της, αφού δεν είναι αξιόπιστη για τις περιπτώσεις που ο βασικός τύπος εισόδου δεν είναι String και δεν είναι καν εφικτή στην περίπτωση που το αντικείμενο εισόδου δεν είναι βασικός τύπος, αλλά κανάλι.

Ένα άλλο πρόβλημα που αντιμετωπίσαμε ήταν η περίπτωση που ο πρώτος όρος σε έναν κανόνα αντιγραφής (replication), δεν είναι ενέργεια εισόδου ή εξόδου αλλά παράλληλη σύνθεση. Σε μία παράλληλη σύνθεση, δεν είναι μία η πιθανή ενέργεια που μπορεί να προκαλέσει αντιγραφή, αλλά μία για την κάθε παράλληλη διεργασία. Και για κάθε πιθανή ενέργεια, η αντιγραμένη διεργασία θα έπρεπε να έχει διαφορετική δομή. Ως αποτέλεσμα αυτού, σε αυτήν την περίπτωση δε μπορέσαμε να εφαρμόσουμε την τεχνική μετάφρασης που χρησιμοποιήσαμε για τον κανόνα της αντιγραφής.

6.3 Μελλοντικές Εργασίες

Η παρούσα εργασία δημιούργησε ένα χρήσιμο εργαλείο για την αυτόματη υλοποίηση μοντέλων, όμως ήταν και το έναυσμα για την σκέψη και άλλων επεκτάσεων ή μελλοντικών εργασιών που θα μπορούσαν να αναπτυχθούν στο ίδιο πλαίσιο με αυτήν.

Καταρχήν, μία μελλοντική εργασία θα μπορούσε να λύσει πιο αποδοτικά το πρόβλημα με την ενέργεια εισόσου ή εξόδου (αλλά κυρίως εξόδου καθώς μπορεί να δημιουργήσει μεγαλύτερο πρόβλημα) με το εξωτερικό σύστημα. Η είσοδος μπορεί να μην είναι μόνο μία λέξη, αλλά ένας πίνακας, ένα ArrayList ή κι ένα κανάλι. Οπότε θα μπορούσε να αναπτυχθεί μία βελτιστοποίηση που η ενέργεια εισόδου να τυγχάνει διαφορετικής διαχείρισης, προσαρμοσμένης στον τύπο του δεδομένου εισόδου.

Επιπλέον, θα μπορούσε να δημιουργηθεί μία επέκταση του εργαλείου που θα υλοποιεί και θα μεταφράζει και την περίπτωση ύπαρξης παράλληλης σύνθεσης ως πρώτο όρο σε ένα κανόνα αντιγραφής. Το πρόβλημα αυτό θα μπορούσε να λυθεί αν υπήρχε κάποιος τρόπος, μέσω της JCSP, να επιβεβαιώσουμε πριν τη πραγματοποίηση ενέργειας εξόδου ότι υπάρχει ήδη διεργασία έτοιμη για αλληλοεπικοινωνία και εκτέλεση της ενέργειας. Επίσης, γι’ αυτό το σκοπό,θα ήταν χρήσιμο να μπορούσαμε να ξέρουμε μεταξύ δύο παράλληλων ενεργειών, ποια εκτελέστηκε πρώτη και μετά να ακυρώνει τις παράλληλές της.

Η πιο σημαντική σκέψη όμως για μελλοντική εργασία που είχε ως ερέθισμα την παρούσα εργασία, είναι η ανάπτυξη ενός ενδιάμεσου μοντέλου άλγεβρας και η ανάπτυξη της αντίστοιχης μετατροπής του σε κώδικα. Αυτό το ενδιάμεσο μοντέλο θα μπορούσε να εκφράζει τόσο τις επικοινωνίες μεταξύ των διεργασιών, όσο και τους υπολογισμούς που θα γίνονταν στο εσωτερικό τους. Ο π-λογισμός είναι απαραίτητος για την έκφραση των επικοινωνιών σε ένα σύστημα διεργασιών. Δεν προσφέρει όμως τη δυνατότητα έκφρασης υπολογισμών, που είναι σχεδόν πάντα απαραίτητοι σε ένα σύστημα, αφού σε ένα σύστημα δεν έχουμε μόνο ανταλλαγές μηνυμάτων, αλλά και υπολογισμούς, συνθήκες αποφάσεων κα. Εφόσον υπήρχει αυτή η τυπική άλγεβρα για την έκφραση προγραμματιστικών υπολογισμών, θα μπορούσε να γίνει και ένα αντίστοιχο εργαλείο για την υλοποίησή του, όπως αυτό της εργασίας. Επίσης, μία προτεινόμενη τακτική για μία τέτοια εργασία, είναι το εργαλείο να μη δέχεται τη

διατύπωση του συστήματος π-λογισμού σε μορφή XML αρχείο, αλλά σε μία διαγραμματική μορφή. Για παράδειγμα, θα μπορούσε να γίνεται με μία είσοδο που θα συνδύαζε Λογικό Διάγραμμα που διευκολύνει την αναπαράσταση υπολογισμών και Ακολουθιακό Διάγραμμα που θα διευκολύνει την αναπαράσταση επικοινωνιών μεταξύ διεργασιών. Αυτό θα έδινε την ευκαιρία στο χρήστη να εισάγει μία πιο αφαιρετική μορφή του συστήματος, μειώνοντας τον κίνδυνο λάθους διατύπωσης από το χρήστη.

Βιβλιογραφία

[1] J.C.M. Baeten, D.A. van Beek, J.E. Rooda. Process Algebra. Handbook of Dynamic System Modeling, 2007.

[2] https://en.wikipedia.org/wiki/Process_calculus

[3] C.A. Petri, Kommunikation mit automaten. Ph.D. Thesis, Institut fuer Instrumentelle Mathematik, Bonn, 1962.

[4] H. Bekic. Towards a mathematical theory of processes. Tech. Report TR 25.125, IBM Laboratory Vienna, 1971.

[5] J.C.M. Baeten. A brief history of process algebra. Theoretical Computer Science, 2005

[6] https://el.wikipedia.org/wiki/Λογισμός_των_Επικοινωνούντων_Συστημάτων

[7] https://el.wikipedia.org/wiki/Επικοινωνούσες_Ακολουθιακές_Διεργασίες

[8] Α.Φιλίππου, "Άλγεβρες Διεργασιών", Σημειώσεις μαθήματος ΕΠΛ 664-Ανάλυση και Επαλήθευση Συστημάτων, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου, 2014.

[9] https://el.wikipedia.org/wiki/Μορφή_Μπάκους-Νάουρ

[10] J.Parrow. An Introduction to the Π-Calculus. Handbook of Process Algebra, pages 1-8. Elsevier, 2001.

[11] https://en.wikipedia.org/wiki/%CE%A0-calculus

[12] Jeannette M. Wing. FAQ on π-Calculus. Document of Tutorial, Carnegie Mellon School of Computer Science, 2002.

[13] D. Kouzapas, A. Philippou, Type checking privacy policies in the π-calculus, In Proceedings of Formal Techniques for Distributed Objects, 2015, pages 3-12.

[14] Φωτεινή Νικολαΐδου, «Τυπικές Μεθόδοι Ιδιωτικότητας», Διπλωματική Εργασία, Πανεπιστήμιο Κύπρου, 2015.

[15] A. Cavoukian. Privacy by Design: The 7 Foundational principles, Implementation and Mapping of Fair Information Practises. Information and Privacy Commissioner of Ontario, 2010.

[16] P.H. Welch, N.C.C. Brown, J. Moores, K.V. Chalmers, B. Sputh. Integrating and extending JCSP. In Proceedings of Communicating Process Architectures, 2007.

[17] G. H. Hilderink. Communicating Threads for Java. Architectures, Languages and Techniques, 1999.

[18] N. Schaller, G. Hilderink, P. Welch. Using Java for parallel computing: JCSP versus CTJ, a comparison. Communicating Process Architectures, 2000.

[19] L. Li. Implementing the π-Calculus in Java. Journal of Object Technology, 2005.

[20] W. de Jonge, B. Jacobs. Privacy-friendly electronic traffic pricing via commits. In Proceedings of FAST'08, 2009.

Παράρτημα Α

API κλάσεων του εργαλείου

Κλάση: CreateBaseTypes

Αυτή η κλάση είναι υπεύθυνη για την δημιουργία βασικών τύπων μέσω μιας διεπιφάνειας αλληλεπίδρασης με το χρήστη. Αποτελείται από μια λίστα αντικειμένων με τα απαραίτητα στοιχεία των βασικών τύπων, μια λίστα καταγραφής των ονομάτων των διαφορετικών βασικών τύπων και ένα ακέραιο που αντιπροσωπεύει το πλήθος τους.

Ορίσματα: static ArrayList<CharacteristicsObj> BaseObj

static ArrayList<String> basetypesN

static int numOfObjects

Μέθοδοι:

CreateBaseTypes() - Είναι ο κατασκευαστής της κλάσης και δεν δέχεται κανένα όρισμα.

static void MenuButtons(int numOb) - Συνάρτηση που δίνει επιλογές στον χρήστη για τον τύπο δεδομένων του τρέχοντος ορίσματος. Οι επιλογές που δίνει είναι: απλός τύπος(πχ int), πίνακας και λίστα.

static void makeArrList(int numOb) - Συνάρτηση που δίνει στον χρήστη τις απαραίτητες επιλογές για τον καθορισμό των χαρακτηριστικών ενός arrayList.

static void makeTTable(int numOb) - Συνάρτηση που δίνει στον χρήστη τις απαραίτητες επιλογές για τον καθορισμό των χαρακτηριστικών ενός δυσδιάστατου πίνακα.

static void makeOTable(int numOb) - Συνάρτηση που δίνει στον χρήστη τις απαραίτητες επιλογές για τον καθορισμό των χαρακτηριστικών ενός μονοδιάστατου πίνακα.

static void chooseSimple(int numOb) - Συνάρτηση που δίνει στον χρήστη τις απαραίτητες επιλογές για τον καθορισμό των χαρακτηριστικών ενός απλού τύπου.

static void createNewClass() - Συνάρτηση που δημιουργεί τις κλάσεις με τον απαραίτητο κώδικα για τον κάθε ένα από τους βασικούς τύπου που έχει ορισθεί.

Κλάση: CharacteristicsObj

Κλάση που κρατάει τα στοιχεία ενός βασικού τύπου που έχει ορίσει ο χρήστης. Αποτελείται από το όνομα του βασικού τύπου και μια λίστα ορισμάτων.

Ορίσματα: String nameOb

ArrayList<ObjChar> atts

Μέθοδοι:

CharacteristicsObj() - είναι ο κατασκευαστής της κλάσης και δεν δέχεται κανένα όρισμα.

Κλάση: ObjChar

Κλάση που δημιουργεί αντικείμενο με τα χαρακτηριστικά του κάθε ορίσματος βασικού τύπου.

Ορίσματα: int eidosOfAtt

String nameOfAtt

String typeOfAtt

int DimRow

int DimCol

Μέθοδοι:

CharacteristicsObj() - είναι ο κατασκευαστής της κλάσης και δεν δέχεται κανένα όρισμα.

Κλάση: ConvertSystem

Η κλάση αυτή δεδομένου ενός αρχείου XML ορθής σύνταξης, υλοποιεί το σύστημα δημιουργώντας τις απαραίτητες κλάσεις με τον κώδικα του.

Ορίσματα: Document doc

Node rootNode

Node ParPNode

static int ParPproc

static int ReplP

static ArrayList<String> baseTypes

static HashMap<String, String> globalChannels

Μέθοδοι:

ConvertSystem(String file) - είναι ο κατασκευαστής της κλάσης και δέχεται το όνομα του αρχείου που θα μεταφράσει.

static HashMap<String,String> handleresNS(HashMap<String, String> resNS, Node node) - Η συνάρτηση αυτή όταν βρεθεί στο αρχείο κανόνας <resNS> τότε παράγει τον αντίστοιχο κώδικα υλοποίησης.

static void handleAtomicGroup(Node resGP, String groupName, PrintWriter writer) - Η συνάρτηση αυτή όταν βρεθεί στο αρχείο κανόνας <resGP> τότε παράγει τον αντίστοιχο κώδικα υλοποίησης.

static void handleInExternal(Node in, PrintWriter writer) - Η συνάρτηση αυτή όταν βρεθεί στο αρχείο κανόνας <inExternal> τότε παράγει τον αντίστοιχο κώδικα υλοποίησης.

static void handleOutExternal(Node out, PrintWriter writer) - Η συνάρτηση αυτή όταν βρεθεί στο αρχείο κανόνας <outExternal> τότε παράγει τον αντίστοιχο κώδικα υλοποίησης.

static void handleProcessesNewGP(Node process, ArrayList<String> prname, HashMap<String, String> resNS) - Υπεύθυνη για δημιουργία κλάσης μιας διεργασίας που βρίσκεται στο εσωτερικό ενός <resGP>.

static void createReplClass(Node node, String replName) - Υπεύθυνη για δημιουργία κλάσης μιας διεργασίας που παράγεται από τον κανόνα αναγραφής <replication>.

static void createProcessPar(Node node, String proName) - Υπεύθυνη για δημιουργία κλάσης μιας διεργασίας που βρίσκεται στο εσωτερικό μιας παράλληλης σύνθεσης.

static void handleParP(Node parP, PrintWriter writer) - Η συνάρτηση αυτή όταν βρεθεί στο αρχείο κανόνας <parP> τότε παράγει τον αντίστοιχο κώδικα υλοποίησης στην κλάση της τρέχουσας ιεραρχίας.

static void handleIn(Node in, PrintWriter writer) - Η συνάρτηση αυτή όταν βρεθεί στο αρχείο κανόνας <in> τότε παράγει τον αντίστοιχο κώδικα υλοποίησης.

static void goToNext(Node nextPro, PrintWriter writer) - Αποφασίζει σε ποια συνάρτηση παραγωγής κώδικα πρέπει να πάει ανάλογα με το επόμενο εντικείμενο.

static void handleOut(Node out, PrintWriter writer) - Η συνάρτηση αυτή όταν βρεθεί στο αρχείο κανόνας <out> τότε παράγει τον αντίστοιχο κώδικα υλοποίησης.

static void handleResHash(HashMap<String, String> resNS, PrintWriter writer) - Η συνάρτηση αυτή όταν βρεθεί στο αρχείο κανόνας <resNS> τότε παράγει τον αντίστοιχο κώδικα υλοποίησης.

static void handleResNP(Node resNP, PrintWriter writer) - Η συνάρτηση αυτή όταν βρεθεί στο αρχείο κανόνας <resNP> τότε παράγει τον αντίστοιχο κώδικα υλοποίησης.

static void handleInRepl(Node firstInstr, PrintWriter writer) - Η συνάρτηση αυτή όταν βρεθεί στο αρχείο κανόνας <in> στο εσωτερικό ενός κανόνα <repl>τότε παράγει τον αντίστοιχο κώδικα υλοποίησης.

static void handleOutRepl(Node firstInstr, PrintWriter writer) - Η συνάρτηση αυτή όταν βρεθεί στο αρχείο κανόνας <out> στο εσωτερικό ενός κανόνα <repl>τότε παράγει τον αντίστοιχο κώδικα υλοποίησης.

static void handleRepl(Node repl, PrintWriter writer) - Η συνάρτηση αυτή όταν βρεθεί στο αρχείο κανόνας <repl> τότε παράγει τον αντίστοιχο κώδικα υλοποίησης.

static void handleParS(Node node, PrintWriter writer, HashMap<String, String> resNS) - Η συνάρτηση αυτή όταν βρεθεί στο αρχείο κανόνας <parS > τότε παράγει τον αντίστοιχο κώδικα υλοποίησης.

static void printSysteminClass(Node node, String groupName, PrintWriter writer) - Γράφει τον κώδικα δημιουργίας νέου συστήματος στην κλάση της τρέχουσας ιεραρχίας.

void implementGS(Node node) - Δημιουργεί την κλάση μιας ομάδας συστήματος με τον απαραίτητο κώδικα.

static ArrayList<String> resNSinParS(Node nodeS, PrintWriter writer) - Η συνάρτηση αυτή όταν βρεθεί στο αρχείο κανόνας <resNS> στο εσωτερικό ενός κανόνα <parS>τότε παράγει τον αντίστοιχο κώδικα υλοποίησης.

static ArrayList<String> ParSnotinresGS(Node node, PrintWriter writer) - Η συνάρτηση αυτή όταν βρεθεί στο αρχείο κανόνας <parS> που δεν βρίσκεται στο εσωτερικό ενός κανόνα <resGS>τότε παράγει τον αντίστοιχο κώδικα υλοποίησης.

static void printGlobalinConstr(PrintWriter writer) - Τυπώνει τον κατάλληλο κώδικα για την ανάθεση των ελεύθερων καναλιών στους κατασκευαστές των κλάσεων διεργασιών.

static void GlobalSentInitiateConstr(PrintWriter writer) - Τυπώνει τον κατάλληλο κώδικα για την αρχικοποίηση των ελεύθερων καναλιών.

static void printNewGlobal(PrintWriter writer) - Δημιουργία αντικειμένου για ελεύθερα κανάλια.

static void ReceiveGlobal(PrintWriter writer) - Κώδικας για την λήψη αναφοράς των ελεύθερων καναλιών.

static void SendGlobal(PrintWriter writer) - Κώδικας για την αποστολή των ελεύθερων καναλιών.

HashMap<String, String> takeGattr(Node r) - Δέχεται τον κόμβο της ρίζας του XML δέντρου με τα κανάλια/ονόματα του Γ περιβάλλοντος και καθορίζει τα κανάλια αυτά για την μετέπειτα χρήση τους στο πρόγραμμα.

Κλάση: GUIselFiles

Η συγκεκριμένη κλάση είναι υπεύθυνη για την δημιουργία της διεπιφάνειας όπου ο χρήστης θα εισάγει τα δύο αναγκαία αρχεία XML και θα ελέγξει αν ο χρήστης εισήγαγε όλα τα απαιτούμε να αρχεία. Επίσης, ακολούθως, θα μεριμνήσει ούτως ώστε να στείλει στην αναγκαία κλάση το αρχείο για να ελεγχθεί η σύνταξη και έπειτα στην αναγκαία κλάση για να διατυπωθεί σε π-λογισμό. Τέλος, θα καλέσει την κλάση που είναι υπεύθυνη για την μετάφραση του συστήματος σε java.

Ορίσματα: static String systemfile

static String gattrfile

static ArrayList<String> basetypesN

static boolean systemGiven

static boolean gattrGiven

Μέθοδοι:

GUIselFiles(ArrayList<String> base) - είναι ο κατασκευαστής της κλάσης και δέχεται μια λίστα από strings που αντιπροσωπεύει τα ονόματα των βασικών τύπων που έχει εισάγει ο χρήστης.

void askFiles() - Ζητά από τον χρήστη να δώσει τα δύο αναγκαία αρχεία.

void ask1File(int type) - Αν κάποιο από τα δύο αρχεία δεν έχει δοθεί θα εμφανίσει το κατάλληλο παράθυρο ανάλογα με το αρχείο που δεν εισήγαγε. Ο χρήστης δεν μπορεί να συνεχίσει αν δεν δώσει και τα δύο αρχεία.

void convStage() - Στέλνει το αρχείο του συστήματος για υλοποίηση σε java, εφόσον βέβαια έχουν γίνει οι απαραίτητοι έλεγχοι.

void checkXML() - Στέλνει το αρχείο του συστήματος στην κατάλληλη κλάση για να γίνει ο έλεγχος της XML σύνταξης.

void confirmTyping() - Στέλνει το αρχείο του συστήματος στην κατάλληλη κλάση για να γίνει η διατύπωση του XML αρχείου σε π-λογισμό.

Κλάση: ParseSystemTree

Κλάση που μεταφράζει το XML αρχείο σε διατύπωση π-λογισμού για επιβεβαίωση από τον χρήστη.

Ορίσματα: Document doc

Node rootNode

Node ParPNode

static ArrayList<String> basetype

static String syste

static String ga

Μέθοδοι:

String decideProcess(Node node) - Αποφασίζει την διατύπωση π-λογισμού στο σημείο π συναντά <Process>.

String decideSymbol(Node node, int n) - Ανάλογα με τον κόμβο του δέντρου XML στον οποίον βρίσκεται, αποφασίζει πως θα προχωρήσει στην διατύπωση π-λογισμού.

void printPi(Node nNode, String spaces, int n) - Δημιουργεί αναδρομικά το string το οποίο θα περιέχει την διατύπωση του συστήματος σε π-λογισμό.

void printXML(Node nNode, String spaces) - Τυπώνει την ιεραρχική δομή του XML αρχείου του συστήματος.

ParseSystemTree(String file, String gat, ArrayList<String> bt) - είναι ο κατασκευαστής της κλάσης και δέχεται το όνομα του αρχείου που θα μεταφέρει σε π-λογισμού διατύπωση, το όνομα του αρχείου με τα ορίσματα του Γ περιβάλλοντος και μια λίστα με τα ονόματα των βασικών τύπων.

Κλάση: checkXMLSyntax

Ορίσματα: static String sysfile

static String gafile

boolean mist

static ArrayList<String> baset

Document doc

Node rootNode

Node ParPNode

String outTree

Μέθοδοι:

checkXMLSyntax(String file, String gatr, ArrayList<String> ba) - είναι ο κατασκευαστής της κλάσης και δέχεται το όνομα του αρχείου που θα ελέγξει την σύνταξη του, το όνομα του αρχείου με τα ορίσματα του Γ περιβάλλοντος και μια λίστα με τα ονόματα των βασικών τύπων.

boolean checkAfterSystem(Node node) - Ελέγχει αν οι εσωτερικοί κόμβοι του σημαντήρα <system> στην ιεραρχία του XML είναι οι ορθοί.

boolean checkAfterRepl(Node node) - Ελέγχει αν οι εσωτερικοί κόμβοι του σημαντήρα <repl> στην ιεραρχία του XML είναι οι ορθοί.

boolean checkAfterParP(Node node) - Ελέγχει αν οι εσωτερικοί κόμβοι του σημαντήρα <parP> στην ιεραρχία του XML είναι οι ορθοί.

boolean checkAfterParS(Node node) - Ελέγχει αν οι εσωτερικοί κόμβοι του σημαντήρα <parS> στην ιεραρχία του XML είναι οι ορθοί.

boolean checkAfterResGP(Node node) - Ελέγχει αν οι εσωτερικοί κόμβοι του σημαντήρα <resGP> στην ιεραρχία του XML είναι οι ορθοί.

boolean checkAfterResGS(Node node) - Ελέγχει αν οι εσωτερικοί κόμβοι του σημαντήρα <resGS> στην ιεραρχία του XML είναι οι ορθοί.

boolean checkAfterResNP(Node node) - Ελέγχει αν οι εσωτερικοί κόμβοι του σημαντήρα <resNP> στην ιεραρχία του XML είναι οι ορθοί.

boolean checkAfterResNS(Node node) - Ελέγχει αν οι εσωτερικοί κόμβοι του σημαντήρα <resNS> στην ιεραρχία του XML είναι οι ορθοί.

boolean checkAfterOut(Node node) - Ελέγχει αν οι εσωτερικοί κόμβοι του σημαντήρα <out> στην ιεραρχία του XML είναι οι ορθοί.

boolean checkAfterType(Node node) - Ελέγχει αν οι εσωτερικοί κόμβοι του σημαντήρα <type> στην ιεραρχία του XML είναι οι ορθοί.

boolean checkAfterName(Node node) - Ελέγχει αν οι εσωτερικοί κόμβοι του σημαντήρα <name> στην ιεραρχία του XML είναι οι ορθοί.

boolean checkAfterObj(Node node) - Ελέγχει αν οι εσωτερικοί κόμβοι του σημαντήρα <obj> στην ιεραρχία του XML είναι οι ορθοί.

boolean checkAfterSubj(Node node) - Ελέγχει αν οι εσωτερικοί κόμβοι του σημαντήρα <subj> στην ιεραρχία του XML είναι οι ορθοί.

boolean checkAfterNil(Node node) - Ελέγχει αν οι εσωτερικοί κόμβοι του σημαντήρα <nil> στην ιεραρχία του XML είναι οι ορθοί.

boolean checkAfterIn(Node node) - Ελέγχει αν οι εσωτερικοί κόμβοι του σημαντήρα <in> στην ιεραρχία του XML είναι οι ορθοί.

boolean checkAfterProcess(Node node) - Ελέγχει αν οι εσωτερικοί κόμβοι του σημαντήρα <process> στην ιεραρχία του XML είναι οι ορθοί.

boolean checkAfterGroup(Node node) - Ελέγχει αν οι εσωτερικοί κόμβοι του σημαντήρα <group> στην ιεραρχία του XML είναι οι ορθοί.

void printXML(Node nNode, String spaces) - Είναι υπεύθυνη για να τυπώνει την ιεραρχία του XML αρχείου στην περίπτωση ύπαρξης λάθους στην σύνταξη, διευκρινίζοντας το σημείο στην ιεραρχία όπου υπάρχει το λάθος.

Κλάση: chooseTheFiles

Κλάση για την εμφάνιση και διαχείριση του παραθύρου εισαγωγής αρχείου.

Ορίσματα: static String SystemName

static String g_atrr

Μέθοδοι:

chooseTheFiles() - είναι ο κατασκευαστής της κλάσης και δέχεται το όνομα του αρχείου που θα μεταφράσει.

boolean showRequestForFiles(int sysORg) - εμφανίζει το παράθυρο εισαγωγής αρχείου. Επιστρέφει true αν έχει δοθεί αρχείο αλλιώς false.