ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη συστήματος διακίνησης και επεξεργασίας μηνυμάτων μεταξύ εφαρμογών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Νικόλαος Π. Αποστολάκος

Επιβλέπων : Στέφανος Κόλλιας
Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2009

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη συστήματος διακίνησης και επεξεργασίας μηνυμάτων μεταξύ εφαρμογών

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Νικόλαος Π. Αποστολάκος

Επιβλέπων : Στέφανος Κόλλιας
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την

.....	.....	.....
Στέφανος Κόλλιας	Ανδρέας-Γεώργιος Σταφυλοπάτης	Παναγιώτης Τσανάκας
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2009

.....

Νικόλαος Π. Αποστολάκος

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Στόχος της παρούσας διπλωματικής εργασίας είναι η σχεδίαση και ανάπτυξη ενός συστήματος διακίνησης και επεξεργασίας μηνυμάτων ανάμεσα σε ανεξάρτητες μεταξύ τους εφαρμογές, με τη χρήση της προγραμματιστικής πλατφόρμας Java, το οποίο προορίζεται για τη σύνδεση διάφορων εφαρμογών του Image, Video and Multimedia Systems Lab (IVML) του Εθνικού Μετσόβιου Πολυτεχνείου.

Το εν λόγω σύστημα προσθέτει τις εξής δυνατότητες στην επικοινωνία:

- Επικοινωνία εφαρμογών οι οποίες χρησιμοποιούν διαφορετικά πρωτόκολλα επικοινωνίας. Το εν λόγο σύστημα παρέχει τη δυνατότητα σύνδεσης εφαρμογών οι οποίες δεν σχεδιάστηκαν για να επικοινωνούν μεταξύ τους, επιτρέποντάς τον ορισμό του τρόπου με τον οποίο γίνεται η μετάφραση μεταξύ των πρωτοκόλλων, χωρίς να απαιτείται η μετατροπή των εφαρμογών. Για παράδειγμα, είναι δυνατή η σύνδεση μιας εφαρμογής η οποία αποθηκεύει το αποτέλεσμα των υπολογισμών της σε ένα αρχείο με μία εφαρμογή που δέχεται το αποτέλεσμα για περαιτέρω ανάλυση μέσω δικτύου, με μια TCP σύνδεση.
- Προώθηση των μηνυμάτων μιας εφαρμογής σε περισσότερες από μία εφαρμογές. Με τη χρήση του εν λόγω συστήματος ο χρήστης μπορεί να προωθήσει την έξοδο μίας εφαρμογής σε περισσότερες από μία, ανεξάρτητες μεταξύ τους, εφαρμογές. Για παράδειγμα τα μηνύματα μίας εφαρμογής η οποία καταγράφει φωνή μπορούν να προωθηθούν ταυτόχρονα σε εφαρμογές για αποθήκευση ήχου, για υπαγόρευση κειμένου και για αναγνώριση προσώπου.
- Στοιχειώδης επεξεργασία των μηνυμάτων. Για παράδειγμα το σύστημα μπορεί να υπολογίζει και να προωθεί τον μέσο όρο των εισερχόμενων μηνυμάτων, να προωθεί σε διαφορετική εφαρμογή τα μηνύματα ανάλογα με την τιμή τους ή να καταγράφει στατιστικές τιμές για τα μηνύματα.

Τέλος, στο παρόν έγγραφο παρουσιάζεται η εφαρμογή του συστήματος για τη σύνδεση τριών εφαρμογών του Image, Video and Multimedia Systems Lab (IVML), οι οποίες επικοινωνούν μέσω δικτύου με TCP μηνύματα.

Λέξεις Κλειδιά

σύνδεση εφαρμογών, Java, δίκτυα υπολογιστών, TCP/IP επικοινωνία, πρωτόκολλα επικοινωνίας, XML

Abstract

The purpose of this diploma thesis is to present the design and implementation of a system for transferring and processing messages between independent applications, with the use of the Java platform, to be used for connecting different applications of the Image, Video and Multimedia Systems Lab (IVML) of the National Technical University of Athens.

The system adds the following functionality to the applications communication:

- Communication between applications which use different communication protocols. The described system provides the user the capability to connect applications which are not designed to communicate with each other, by allowing him to specify the way the translation between protocols is happening, without the need of modification of the original applications. An example of such applications is one that is storing the result of its calculations in a file and another one which receives this output for further processing through the network with a TCP connection.
- Forwarding of the messages from one application to more than one applications. With the use of the described system the user can forward the output of an application to more than one independent applications. For example the messages of an application which records voice can be forwarded the same time to an application for audio recording, and application for dictation and an application for person recognition.
- Basic process of the messages. For example the system can calculate and forward the average of the incoming messages, to forward to different applications the messages depended of their value or to record statistics about the messages.

Finally, on this document is presented the use of the system for connecting three applications of the Image, Video and Multimedia Systems lab (IVML), which communicate through network with TCP messages.

Key Words

applications communication, Java, computer networks, TCP/IP communication, communication protocols, XML

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον κ. Στέφανο Κόλλια για την ευκαιρία που μου έδωσε να συνεργαστώ με το εργαστήριο Image, Video and Multimedia Systems Lab (IVML) και να υλοποιήσω την ιδέα του συστήματος που περιγράφεται στην παρούσα διπλωματική. Επίσης θα ήθελα να ευχαριστήσω τον κ. Κώστα Καρπούζη για την άψογη συνεργασία που υπήρξε μεταξύ μας κατά τη διεκπεραίωση της διπλωματικής μου εργασίας, καθώς και τους δημιουργούς των προγραμμάτων Components, Torque και Callas AMS τα οποία χρησιμοποιήθηκαν, και την κα Λώρη Μαλατέστα για την πολύτιμη βοήθειά της κατά τη χρήση των προγραμμάτων αυτών.

Τέλος θέλω να ευχαριστήσω την οικογένειά μου καθώς και όσους μου στάθηκαν με οποιονδήποτε τρόπο και βοήθησαν στην ολοκλήρωση της εργασίας αυτής κατά τη διάρκεια της στρατιωτικής μου θητείας.

Η εργασία αυτή αφιερώνεται στην Danuta Paraficz,
πηγή έμπνευσης της δουλειάς μου

Πίνακας Περιεχομένων

Ευρετήριο Σχημάτων

Ευρετήριο Πινάκων

1. Εισαγωγή

Το παρόν έγγραφο αποτελεί την τεκμηρίωση του σχεδιασμού και της ανάπτυξης ενός συστήματος διακίνησης και επεξεργασίας μηνυμάτων μεταξύ ανεξάρτητων εφαρμογών. Στόχος των κεφαλαίων που ακολουθούν είναι ο καθορισμός των προδιαγραφών ενός τέτοιου συστήματος, η πλήρης περιγραφή της διαδικασίας σχεδιασμού του, η περιγραφή της ανάπτυξης και υλοποίησης αυτού καθώς και η εφαρμογή του για τη σύνδεση κάποιων ήδη υλοποιημένων εφαρμογών.

1.1. Αντικείμενο της διπλωματικής

Η δημιουργία των υπολογιστών αναμφισβήτητα αποτέλεσε ένα από τα μεγαλύτερα βήματα προς τη διευκόλυνση και την εξυπηρέτηση του ανθρώπου. Αναρίθμητοι τομείς της ζωής απλοποιήθηκαν σε σημαντικό βαθμό, επιτρέποντας την εστίαση της νοητικής δύναμης και την κατανάλωση χρόνου σε πιο ουσιώδεις διεργασίες, επιτρέποντας την υλοποίηση στόχων οι οποίοι ήταν θα ήταν αδύνατον να υλοποιηθούν. Τα παραδείγματα προς απόδειξη του ανωτέρω συναντώνται καθημερινά, από την εκτέλεση μέσα σε δευτερόλεπτα επιστημονικών αλγορίθμων οι οποίοι απαιτούν εκατομμύρια υπολογισμούς, την εύρεση στοιχείων μηχανογράφησης ανάμεσα σε χιλιάδες εγγραφές με το πάτημα ενός κουμπιού, μέχρι τον αυτόματο έλεγχο πτήσης αεροσκαφών με διορθώσεις ακριβείας σε χρόνους όπου ο άνθρωπος δεν θα προλάβαινε να αντιδράσει.

Η επανάσταση αυτή ξεκίνησε με απλά συστήματα, αλλά εξελίχθηκε με ραγδαίους ρυθμούς σε

ιδιαίτερα πολύπλοκα συστήματα, για την εξυπηρέτηση όλο και πολυπλοκότερων αναγκών. Ιδιαίτερα στον τομέα του λογισμικού αυτή η εξέλιξη είναι ιδιαίτερα έντονη. Τα πρώτα απλά και κατανοητά προγράμματα, κυρίως για υλοποίηση μαθηματικών αλγορίθμων, τα οποία αποτελούνταν από μερικές δεκάδες ή εκατοντάδες γραμμές κώδικα, έγιναν πολύπλοκα συστήματα πολλών χιλιάδων γραμμών κώδικα, των οποίων η κατανόηση της λειτουργίας τους απαιτεί έναν αρκετά υπολογίσιμο χρόνο. Η εξέλιξη αυτή κατέστησε την επαναχρησιμοποίηση των διάφορων προγραμμάτων υποχρεωτική και δημιούργησε έναν καινούργιο τρόπο σκέψης στον προγραμματισμό που ευνοεί την επαναχρησιμοποίηση γραμμένου κώδικα, ο οποίος υλοποιήθηκε μέσα από τις αντικειμενοστραφείς γλώσσες προγραμματισμού.

Το σκεπτικό αυτό έχει εφαρμοστεί και στο επίπεδο εφαρμογών, όπου οι χρήστες των υπολογιστών χρησιμοποιούν τα είδη υπάρχοντα προγράμματα για τη δημιουργία της εισόδου των δικών τους προγραμμάτων. Το γεγονός αυτό όμως δεν έχει μόνο θετικές συνέπειες. Ο χρόνος ο οποίος καταναλώνεται για τη σύνδεση των διαφορετικών εφαρμογών, αν και σημαντικά μικρότερος από τον χρόνο που θα χρειαζόταν για την υλοποίηση των εφαρμογών από την αρχή, αυξάνεται εκθετικά με τον αριθμό των εμπλεκόμενων εφαρμογών και την πολυπλοκότητά τους. Αυτό έχει ως αποτέλεσμα ο ανθρώπινος χρόνος να αφιερώνεται περισσότερο σε τέτοια προβλήματα και όχι στην υλοποίηση νέων ιδεών, εκεί δηλαδή όπου χρειάζεται.

Το αντικείμενο της παρούσας διπλωματικής είναι η υλοποίηση ενός συστήματος το οποίο θα βοηθήσει στη λύση αυτού του προβλήματος. Το εν λόγο σύστημα θα καθιστά δυνατή και με εύκολο τρόπο τη σύνδεση εφαρμογών οι οποίες δεν σχεδιάστηκαν να επικοινωνούν μεταξύ τους, καθώς επίσης θα παρέχει και τη δυνατότητα συνδυασμού των εξόδων διαφορετικών εφαρμογών για τη δημιουργία μηνυμάτων προς άλλες εφαρμογές, την προώθηση των μηνυμάτων σε περισσοτέρων της μίας εφαρμογές, καθώς και την στοιχειώδη επεξεργασία των μηνυμάτων.

1.2. Οργάνωση του τόμου

Το παρόν κείμενο χωρίζεται σε οκτώ κεφάλαια. Στο πρώτο κεφάλαιο γίνεται μια εισαγωγή στο αντικείμενο της διπλωματικής, καθώς και μια συζήτηση και ομαδοποίηση της βιβλιογραφίας και η επεξήγηση της επιλογής της.

Στο δεύτερο κεφάλαιο αναφέρεται η ανάλυση του συστήματος. Η ανάλυση αυτή περιλαμβάνει τον εντοπισμό των λειτουργικών προδιαγραφών, δηλαδή της αναμενόμενης συμπεριφοράς του συστήματος, τον εντοπισμό των περιπτώσεων χρήσεως για την ικανοποίηση των λειτουργικών προδιαγραφών και την εξαγωγή πιθανών σεναρίων από τις περιπτώσεις χρήσεως. Στο τέλος του κεφαλαίου γίνεται και μια αναφορά στις πλατφόρμες και τα προγραμματιστικά εργαλεία που

επιλέχθηκαν για την υλοποίηση του συστήματος.

Το τρίτο κεφάλαιο ξεκινάει με μια περιγραφή και επεξήγηση των ιδιοτήτων της Java των οποίων η γνώση είναι απαραίτητη για την κατανόηση της σχεδίασης του συστήματος, και συνεχίζει χωρίζοντας το σύστημα σε μικρότερα τμήματα και εντοπίζοντας τις απαραίτητες κλάσεις για την υλοποίησή τους, χρησιμοποιώντας τις λειτουργικές προδιαγραφές και τις περιπτώσεις χρήσεως που εντοπίστηκαν στο δεύτερο κεφάλαιο.

Το τέταρτο κεφάλαιο παρουσιάζει την υλοποίηση των κλάσεων που εντοπίστηκαν στο τρίτο, καθώς και τις προγραμματιστικές τεχνικές που χρησιμοποιήθηκαν.

Το πέμπτο κεφάλαιο περιγράφει την εφαρμογή του συστήματος για τη μεταφορά μηνυμάτων ανάμεσα σε τρεις εφαρμογές του Image, Video and Multimedia Systems Lab (IVML) του Εθνικού Μετσόβιου Πολυτεχνείου. Η μία εφαρμογή από αυτές ανιχνεύει την ταχύτητα με την οποία κινούνται τα χέρια ενός χρήστη μέσω μιας κάμερας, και η τιμή αυτή χρησιμοποιείται για τον έλεγχο της ταχύτητας ενός τραγουδιού και της κίνησης μιας 3D φιγούρας στις άλλες δύο εφαρμογές.

Το έκτο κεφάλαιο παρουσιάζει μια σύνοψη των προηγουμένων κεφαλαίων και τα συμπεράσματα που προέκυψαν κατά τη διάρκεια διεκπεραίωσης της διπλωματικής, καθώς και πιθανές μελλοντικές επεκτάσεις του συστήματος.

Τα δύο τελευταία κεφάλαια (έβδομο και όγδοο) περιέχουν τα παραρτήματα και τη βιβλιογραφία αντίστοιχα.

1.3. Αναφορά στη βιβλιογραφία

Σε αυτήν την ενότητα υπάρχει μια σύντομη περιγραφή της βιβλιογραφίας η οποία βρίσκεται στο τελευταίο κεφάλαιο. Ο σκοπός αυτής της ενότητας είναι να καθοδηγήσει τον αναγνώστη στην αναζήτηση περαιτέρω πηγών πληροφοριών. Επίσης εδώ η βιβλιογραφία ομαδοποιείται ανάλογα με το πεδίο στο οποίο αναφέρεται.

1.3.1. Βιβλιογραφία σχετική με Java

Ένα πολύ καλό βιβλίο για εισαγωγή στη Java και γενικότερα τον αντικειμενοστραφή προγραμματισμό είναι το “Thinking in Java, 4th Edition” ([Eck05]). Στα πρώτα κεφάλαια ο αναγνώστης θα βρει μια αρκετά εκτενή εισαγωγή στον αντικειμενοστραφή προγραμματισμό και στο υπόλοιπο βιβλίο μια καλή εισαγωγή στη Java, η οποία καλύπτει και τα καινούργια χαρακτηριστικά της Java 5. Ένα μειονέκτημα του βιβλίου είναι ότι ο συγγραφέας είναι υπερβολικά

αναλυτικός με αποτέλεσμα το βιβλίο να αποτελείται από περίπου 1500 σελίδες. Η προηγούμενη έκδοση του βιβλίου είναι διαθέσιμη δωρεάν στο Internet από το site http://www.bruceeckel.com αλλά δεν καλύπτει τα χαρακτηριστικά της Java 5.

Για μια πολύ καλή και πλήρη εισαγωγή στη Java ο αναγνώστης μπορεί επίσης να αναφερθεί στο πολύ καλό online tutorial από την Sun Microsystems. Το tutorial μπορεί επίσης να βρεθεί σε τυπωμένη μορφή στα τρία βιβλία [CWH+01], [CWH+98] και [WCHZ04] (χωρίς όμως τα χαρακτηριστικά των νεότερων εκδόσεων). Η online έκδοση ενημερώνεται τακτικά και περιέχει μια πολύ καλή εισαγωγή σε κάθε χαρακτηριστικό της Java, νέο ή παλιό. Το θετικό με αυτό το tutorial είναι ότι προέρχεται από την Sun Microsystems με αποτέλεσμα να περιέχει (συνήθως) τις πιο ορθές προγραμματιστικές τεχνικές. Αποτελεί σίγουρα την καλύτερη πηγή για μια γρήγορη εισαγωγή σε μια καινούργια Java τεχνολογία.

Ένα βιβλίο άξιο αναφοράς είναι το "Just Java 2" ([Lin04]). Το βιβλίο αυτό περιέχει μια σύντομη αλλά ουσιαστική περιγραφή της Java 2 και των βασικότερων βιβλιοθηκών της. Σίγουρα δεν προτείνεται για εκμάθηση της γλώσσας, αλλά είναι η πρώτη αναφορά του Java προγραμματιστή για να λύσει τα προβλήματα που συναντάει. Το βιβλίο περιγράφει πλήρως όλα τα νέα χαρακτηριστικά της Java 5 και παρουσιάζει χρήσιμες προγραμματιστικές τεχνικές για τη δημιουργία ενός πιο αποδοτικού κώδικα.

Για τον ήδη έμπειρο προγραμματιστή με την Java, το βιβλίο "Java 1.5 Tiger: A Developer's Notebook" ([LF04]) είναι μια πολύ καλή και χρήσιμη αναφορά σε όλες τις αλλαγές από την έκδοση 1.4 στην έκδοση 1.5. Επίσης το "Effective Java" ([Blo01]) περιέχει προγραμματιστικές τεχνικές για τη σωστή χρήση της Java και την παραγωγή πιο αποδοτικού κώδικα.

Τέλος το "Java 2 API Specification" ([Sun04]) είναι το μέρος όπου ο Java προγραμματιστής μπορεί να λύσει κάθε πρόβλημα που είναι δυνατόν να συναντήσει, όπως επίσης να μελετήσει ακόμα και τη μικρότερη λεπτομέρεια της Java και των βιβλιοθηκών της. Ο συγγραφέας του παρών κειμένου προτείνει στον κάθε Java προγραμματιστή να ανατρέχει σε αυτό για κάθε Java κλάση που χρησιμοποιεί στα προγράμματά του.

1.3.2. Βιβλιογραφία σχετική με δίκτυα

Δύο πολύ καλά βιβλία για προγραμματισμό δικτύων με Java είναι τα συνώνυμα "Java Network Programming" από τις εκδόσεις Manning ([HSH99]) και O'Reilly ([Har04]) . Από αυτά το πρώτο εστιάζεται περισσότερο στην ασφάλεια των επικοινωνιών ενώ το δεύτερο εστιάζεται περισσότερο στην περιγραφή του προγραμματισμού δικτύων με τη Java, χωρίς εκτενείς λεπτομέρειες των δικτύων (οι οποίες δεν χρειάζονται, αφού η Java τις αποκρύπτει από τον χρήστη).

Ο αναγνώστης που θέλει ένα καλό βιβλίο δικτύων (όχι σχετικό με τη Java) μπορεί να αναφερθεί στο "Unix Network Programming" ([Ste90]), το οποίο περιέχει όλες τις πληροφορίες που χρειάζεται σε μόνο 700 σελίδες.

1.3.3. Βιβλιογραφία σχετική με XML

Ένα βιβλίο για την εκμάθηση της τεχνολογίας XML το οποίο απευθύνεται στον αναγνώστη χωρίς προηγούμενη εμπειρία, είναι το "Beginning XML" από τις εκδόσεις John Wiley & Sons ([HRF+07]). Ο αναγνώστης ο οποίος έχει κάποια εμπειρία σε παρόμοιες τεχνολογίες, όπως HTML, μπορεί να αναφερθεί στο "Learning XML" από τις εκδόσεις O'Reilly ([Ray03]).

Για το σκοπό ανάγνωσης του παρών κειμένου δεν είναι απαραίτητη η εκτενής γνώση της τεχνολογίας XML. Ο αναγνώστης ο οποίος δεν έχει καμία γνώση σχετικά με XML αρκεί να αναφερθεί σε κάποιο tutorial στο ίντερνετ. Ένα καλό παράδειγμα τέτοιου tutorial μπορεί να βρεθεί στη σελίδα <http://www.w3schools.com/xml/default.asp>.

1.3.4. Βιβλιογραφία σχετική με UML

Στα κεφάλαια που ακολουθούν γίνεται εκτενής χρήση της UML (Unified Modeling Language). Για την κατανόηση των διαγραμμάτων ο αναγνώστης πρέπει να έχει κάποια γνώση της. Για μια σύντομη εισαγωγή υπάρχουν αρκετά tutorials στο Internet. Ένα παράδειγμα αυτών είναι στη σελίδα <http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/index.htm>. Αυτό το tutorial είναι πολύ σύντομο και δεν περιγράφει την UML σε βάθος, αλλά θα δώσει στον αναγνώστη τη δυνατότητα να κατανοήσει τα διαγράμματα που θα ακολουθήσουν.

Για μια πιο εκτενή περιγραφή ο αναγνώστης μπορεί να ανατρέξει στο "The Unified Modeling Language User Guide" ([Boo05]) το οποίο αποτελεί ένα πολύ καλό εγχειρίδιο της γλώσσας. Επίσης το "UML 2.0 Pocket Reference" ([Pil06]) είναι χρήσιμο ως αναφορά για τον αναγνώστη ο οποίος ήδη έχει κάποια γνώση της UML.

2. Ανάλυση

Στο κεφάλαιο αυτό περιγράφεται η ανάλυση του συστήματος. Στην ενότητα 2.1 περιγράφονται οι λειτουργικές προδιαγραφές του συστήματος, δηλαδή οι απαιτήσεις τις οποίες πρέπει να ικανοποιεί το σύστημα σύστημα. Στην ενότητα 2.2 εντοπίζονται οι χρήσεις τις οποίες μπορεί να έχει το σύστημα κατά την εκτέλεσή του. Από αυτές εξάγονται και αναλύονται τα διάφορα πιθανά σενάρια που θα ακολουθήσει το σύστημα, τα οποία περιγράφονται στην ενότητα 2.3, παρουσιάζοντας τα μηνύματα εσωτερικά στο σύστημα όπως επίσης και τις αλληλεπιδράσεις με τους χρήστες. Τέλος, στην ενότητα 2.4 περιγράφονται τα προγραμματιστικά εργαλεία και οι πλατφόρμες υλοποίησης που θα χρησιμοποιηθούν, συνοδευόμενες με μια επεξήγηση της επιλογής τους.

2.1. Λειτουργικές προδιαγραφές

Με τον όρο λειτουργικές προδιαγραφές εννοούμε τις απαιτήσεις των χρηστών από το σύστημα, δηλαδή την αναμενόμενη συμπεριφορά του συστήματος καθώς και την παρουσίασή του στους χρήστες. Στο υπόλοιπο αυτής της ενότητας εντοπίζονται όλες οι λειτουργικές προδιαγραφές του συστήματος και ανατίθεται στην κάθε μία ένας κωδικός για ευκολότερη αναφορά σε αυτές από τις επόμενες ενότητες.

Για την ευκολότερη ανίχνευση και καταγραφή τους, οι λειτουργικές προδιαγραφές μπορούν να ομαδοποιηθούν στις ακόλουθες κατηγορίες.

2.1.1. Προδιαγραφές γενικού χαρακτήρα

Σε αυτήν την κατηγορία περιλαμβάνονται οι προδιαγραφές του συστήματος που έχουν ένα γενικότερο χαρακτήρα. Ο σκοπός αυτών των προδιαγραφών είναι να προσδιορίσουν με έναν γενικό τρόπο τη λειτουργία του συστήματος, χωρίς να περιγράφουν την αναμενόμενη συμπεριφορά με λεπτομέρειες.

Οι προδιαγραφές γενικού χαρακτήρα προκύπτουν από το σκοπό δημιουργίας του συστήματος, ο οποίος είναι η αποδοτική διασύνδεση ανεξάρτητων μεταξύ τους εφαρμογών. Με άλλα λόγια το σύστημα πρέπει να λειτουργεί ως μια γέφυρα επικοινωνίας μεταξύ μίας ομάδας εφαρμογών οι οποίες δημιουργούν κάποια δεδομένα, και μίας ομάδας εφαρμογών οι οποίες δέχονται τα δεδομένα για περαιτέρω επεξεργασία. Στο υπόλοιπο του παρόν κειμένου οι εφαρμογές που δημιουργούν τα δεδομένα θα αναφέρονται ως εφαρμογές εισόδου (επειδή τα δεδομένα που δημιουργούν είναι η είσοδος του συστήματος), και οι εφαρμογές που δέχονται τα δεδομένα για περαιτέρω επεξεργασία θα αναφέρονται ως εφαρμογές εξόδου (επειδή δέχονται την έξοδο του συστήματος). Ανάλογα, τα μηνύματα από τις εφαρμογές εισόδου προς το σύστημα θα αναφέρονται ως εισερχόμενα μηνύματα και τα μηνύματα από το σύστημα προς τις εφαρμογές εξόδου θα αναφέρονται ως εξερχόμενα μηνύματα.

Στη γενική περίπτωση, οι εφαρμογές εισόδου και εξόδου θα είναι εφαρμογές οι οποίες δεν σχεδιάστηκαν για να συνεργάζονται μεταξύ τους. Αυτή άλλωστε είναι και μία από τις χρήσεις του συστήματος, η διασύνδεση ασύμβατων εφαρμογών. Για την επίτευξη της ομαλής επικοινωνίας μεταξύ τέτοιων εφαρμογών, το σύστημα πρέπει να παρέχει τη δυνατότητα μιας βασικής επεξεργασίας των εισερχόμενων μηνυμάτων, ώστε να επιτρέπει τη μετατροπή τους σε μηνύματα συμβατά με τις εφαρμογές εξόδου. Ένα απλό παράδειγμα είναι δύο εφαρμογές οι οποίες λειτουργούν με διαφορετικές μονάδες μέτρησης (πχ μέτρα και πόδια), οπότε το σύστημα πρέπει να μετατρέπει τη μία μονάδα στην άλλη, ενώ ένα πιο σύνθετο παράδειγμα είναι μία εφαρμογή η έξοδος της οποίας γίνεται με μεγάλη συχνότητα και είναι επιθυμητός ο υπολογισμός του μέσου όρου των τιμών που στέλνει, για την αποφυγή περιττών υπολογισμών από την εφαρμογή που δέχεται τα μηνύματα (η οποία μπορεί να απαιτεί υπερβολικά μεγάλη υπολογιστική ισχύ).

Για να μην είναι περιορισμένος ο αριθμός των εφαρμογών τις οποίες θα υποστηρίζει το σύστημα, ο τρόπος επεξεργασίας των εισερχόμενων μηνυμάτων, καθώς και οι εφαρμογές εξόδου στις οποίες θα αποσταλούν, πρέπει να προσδιορίζεται από τον χρήστη του συστήματος. Κατά τη διάρκεια εκτέλεσης δεν είναι απαραίτητη η δυνατότητα μεταβολής αυτής της πληροφορίας, αλλά κατά την αρχικοποίηση του συστήματος ο χρήστης πρέπει να μπορεί να προγραμματίσει το σύστημα επιλέγοντας τις εφαρμογές εισόδου, την επεξεργασία των δεδομένων και τις εφαρμογές εξόδου.

Μετά το τέλος της αρχικοποίησης, το σύστημα πρέπει να συνδέει τις εφαρμογές εισόδου και εξόδου με τον τρόπο που επέλεξε ο χρήστης. Για την επιτήρηση της εκτέλεσης του συστήματος, το σύστημα πρέπει να παρέχει στο χρήστη διεπαφές στις οποίες θα παρουσιάζει όλες τις πληροφορίες για την τρέχουσα κατάστασή του. Ο χρήστης πρέπει να έχει τη δυνατότητα ενεργοποίησης και απενεργοποίησης των διεπαφών αυτών για τη βελτιστοποίηση της επίδοσης του συστήματος, καθώς και για τη χρήση του συστήματος σε υπολογιστές χωρίς υποδομή για γραφική έξοδο.

Τέλος, το σύστημα πρέπει να παρέχει τη δυνατότητα ενσωμάτωσης σε άλλες εφαρμογές. Η δυνατότητα αυτή είναι πολύ σημαντική, καθώς είναι μια μεγάλη διευκόλυνση για τους χρήστες που απλώς θέλουν να το χρησιμοποιήσουν για την κατασκευή ενός μεγαλύτερου συστήματος.

Με βάση τα ανωτέρω μπορούν να εξαχθούν οι προδιαγραφές γενικού χαρακτήρα όπως καταγράφονται στον πίνακα 2.1.

Πίνακας 2.1: Προδιαγραφές γενικού χαρακτήρα

Κωδικός	**Προδιαγραφή**
ΠΓΧ-1	Το σύστημα πρέπει να δέχεται την έξοδο διαφόρων ανεξάρτητων εφαρμογών ως είσοδο. Οι εφαρμογές αυτές θα αναφέρονται στο υπόλοιπο του παρόν κειμένου ως εφαρμογές εισόδου και τα μηνύματα από αυτές ως εισερχόμενα μηνύματα.
ΠΓΧ-2	Το σύστημα πρέπει να παρέχει τη δυνατότητα επεξεργασίας των εισερχόμενων μηνυμάτων της προδιαγραφής ΠΓΧ-1.
ΠΓΧ-3	Το σύστημα πρέπει να παρέχει τη δυνατότητα προώθησης των αποτελεσμάτων της επεξεργασίας της προδιαγραφής ΠΓΧ-2 καθώς και την απευθείας προώθηση των εισερχόμενων μηνυμάτων της προδιαγραφής ΠΓΧ-1, σε διάφορες ανεξάρτητες εφαρμογές για περαιτέρω επεξεργασία. Οι εφαρμογές αυτές θα αναφέρονται στο υπόλοιπο του παρόν κειμένου ως εφαρμογές εξόδου και τα μηνύματα προς αυτές ως εξερχόμενα μηνύματα.
ΠΓΧ-4	Ο χρήστης πρέπει να προσδιορίζει κατά την αρχικοποίηση του συστήματος τις εφαρμογές εισόδου, την επεξεργασία των εισερχόμενων μηνυμάτων, καθώς και τις εφαρμογές εξόδου.
ΠΓΧ-5	Το σύστημα πρέπει να παρέχει διεπαφές επικοινωνίας με το χρήστη, στις οποίες ο χρήστης θα μπορεί να βλέπει πληροφορίες σχετικά με την τρέχουσα κατάσταση του συστήματος.
ΠΓΧ-6	Το σύστημα πρέπει να παρέχει τη δυνατότητα ενσωμάτωσης σε μελλοντικές εφαρμογές.

Κωδικός	Προδιαγραφή
ΠΓΧ-7	Οι διεπαφές του χρήστη (προδιαγραφή ΠΓΧ-5) πρέπει να είναι προαιρετικές για την ευκολότερη εφαρμογή της προδιαγραφής ΠΓΧ-6, για μεγιστοποίηση της επίδοσης του συστήματος, καθώς και για εκτέλεση του συστήματος σε υπολογιστές χωρίς υποδομή για γραφική έξοδο στο χρήστη.

2.1.2. Προδιαγραφές εισερχόμενων μηνυμάτων

Εδώ καταγράφονται οι προδιαγραφές του συστήματος οι οποίες είναι σχετικές με τα εισερχόμενα μηνύματα από τις εφαρμογές εισόδου (προδιαγραφή ΠΓΧ-1).

Στη γενική περίπτωση, οι διαφορετικές εφαρμογές εισόδου θα χρησιμοποιούν και διαφορετικά πρωτόκολλα επικοινωνίας. Το σύστημα φυσικά δεν πρέπει να απαιτεί μεταβολές στον κώδικα των εφαρμογών (οι οποίες τις περισσότερες φορές δεν θα είναι δυνατόν να πραγματοποιηθούν), αλλά πρέπει να επικοινωνεί με την κάθε εφαρμογή εισόδου με το πρωτόκολλο επικοινωνίας το οποίο σχεδιάστηκε να χρησιμοποιεί.

Λόγω της μεγάλης πληθώρας διαφορετικών πρωτοκόλλων επικοινωνίας, καθώς και για να είναι το σύστημα επεκτάσιμο, ο χρήστης πρέπει να έχει τη δυνατότητα να ορίζει νέα πρωτόκολλα επικοινωνίας με τις εφαρμογές εισόδου. Αυτό πρέπει να γίνεται με ένα δυναμικό τρόπο, χωρίς την απαίτηση μετατροπών στον κώδικα του συστήματος. Επίσης, λόγω των διαφορετικών παραμέτρων που τα διάφορα πρωτόκολλα επικοινωνίας μπορεί να δέχονται, το σύστημα πρέπει να παρέχει στο χρήστη έναν ενιαίο τρόπο παραμετροποίησης της επικοινωνίας με την κάθε εφαρμογή εισόδου κατά την αρχικοποίηση του συστήματος.

Τέλος, το σύστημα πρέπει να υποστηρίζει την ταυτόχρονη σύνδεση με περισσότερες από μία εφαρμογές εισόδου. Με αυτόν τον τρόπο το σύστημα μπορεί να χρησιμοποιηθεί για τη σύνδεση περισσοτέρων του ενός ζεύγους εφαρμογών ταυτόχρονα καθώς και το συνδυασμό εισερχόμενων μηνυμάτων για την παραγωγή των μηνυμάτων προς τις εφαρμογές εξόδου. Για την ομαλή λειτουργία του συστήματος όταν είναι συνδεδεμένο με περισσότερες από μία εφαρμογές εισόδου, οι καθυστερήσεις στην επικοινωνία με μία από αυτές δεν πρέπει να επηρεάζει την απόδοση της επικοινωνίας με τις υπόλοιπες εφαρμογές.

Με βάση τα ανωτέρω μπορούν να εξαχθούν οι προδιαγραφές εισερχόμενων μηνυμάτων όπως περιγράφονται στον πίνακα 2.2.

Πίνακας 2.2: Προδιαγραφές εισερχόμενων μηνυμάτων

Κωδικός	Προδιαγραφή
ΠΕΙΜ-1	Η επικοινωνία με τις εφαρμογές εισόδου πρέπει να γίνεται με το πρωτόκολλο επικοινωνίας το οποίο χρησιμοποιούν οι εφαρμογές.
ΠΕΙΜ-2	Ο χρήστης πρέπει να έχει τη δυνατότητα να ορίζει νέα πρωτόκολλα επικοινωνίας με τις εφαρμογές εισόδου με έναν δυναμικό τρόπο, χωρίς την απαίτηση μετατροπών στο σύστημα.
ΠΕΙΜ-3	Ο χρήστης πρέπει να έχει τη δυνατότητα παραμετροποίησης του κάθε πρωτοκόλλου της προδιαγραφής ΠΕΙΜ-2 κατά την αρχικοποίηση του συστήματος.
ΠΕΙΜ-4	Το σύστημα πρέπει να επιτρέπει την ταυτόχρονη επικοινωνία με έναν απροσδιόριστο αριθμό εφαρμογών εισόδου, οι οποίες μπορεί να χρησιμοποιούν διαφορετικά πρωτόκολλα επικοινωνίας.
ΠΕΙΜ-5	Οι καθυστερήσεις στην επικοινωνία με κάποια εφαρμογή εισόδου δεν πρέπει να επηρεάζουν την απόδοση της επικοινωνίας με τις υπόλοιπες εφαρμογές εισόδου.

2.1.3. Προδιαγραφές επεξεργασίας μηνυμάτων

Εδώ καταγράφονται οι απαιτήσεις του συστήματος οι οποίες είναι σχετικές με την επεξεργασία των εισερχόμενων μηνυμάτων που εκτελεί το σύστημα (προδιαγραφή ΠΓΧ-2).

Σύμφωνα με την προδιαγραφή ΠΓΧ-2, το σύστημα επιτρέπει την επεξεργασία των εισερχόμενων μηνυμάτων για την ομαλή σύνδεση των εφαρμογών εισόδου με τις εφαρμογές εξόδου. Λόγω του διαφορετικού είδους πληροφορίας από διαφορετικές εφαρμογές εισόδου, καθώς και για χάρη της δυναμικότητας του συστήματος, ο χρήστης πρέπει να έχει τη δυνατότητα να ορίζει τον τρόπο με τον οποίο γίνεται η επεξεργασία των εισερχόμενων μηνυμάτων χωρίς την απαίτηση μετατροπών στο σύστημα. Για τον ίδιο λόγο ο χρήστης πρέπει επίσης να έχει τη δυνατότητα παραμετροποίησης του κάθε τρόπου επεξεργασίας που έχει ορίσει για κάθε εφαρμογή εισόδου κατά την αρχικοποίηση του συστήματος.

Επειδή η επεξεργασία ενός εισερχόμενου μηνύματος με κάποιον τρόπο επεξεργασίας μπορεί να διαρκεί ένα αρκετά μεγάλο χρονικό διάστημα, υπάρχει η περίπτωση μία εφαρμογή εισόδου να στείλει το επόμενο εισερχόμενο μήνυμα που πρέπει να επεξεργαστεί πριν τελειώσει η επεξεργασία του προηγούμενου. Σε αυτήν την περίπτωση το νέο μήνυμα δεν θα χάνεται, αλλά το σύστημα θα το αποθηκεύει και θα το επεξεργάζεται μόλις τελειώσει την επεξεργασία του προηγούμενου. Για την αποφυγή της υπερβολικής χρήσης μνήμης στην οποία μπορεί να οδηγήσει αυτή η συμπεριφορά, το σύστημα θα αποθηκεύει μόνο έναν ορισμένο αριθμό μηνυμάτων, τον οποίο θα

ορίζει ο χρήστης για κάθε στάδιο επεξεργασίας. Τα περαιτέρω μηνύματα που λαμβάνει το σύστημα θα αγνοούνται.

Για μεγαλύτερη ευελιξία του συστήματος, το αποτέλεσμα της επεξεργασίας ενός μηνύματος με έναν από τους τρόπους επεξεργασίας τους οποίους έχει ορίσει ο χρήστης, μπορεί να δέχεται περαιτέρω επεξεργασία με έναν ή περισσότερους από τους υπόλοιπους τρόπους επεξεργασίας. Με αυτόν τον τρόπο ο χρήστης μπορεί να ορίσει μόνο κάποιους βασικούς τρόπους επεξεργασίας και μετά να δημιουργήσει αλυσίδες αυτών, οι οποίες θα επιφέρουν το επιθυμητό αποτέλεσμα.

Τέλος, το σύστημα πρέπει να παρέχει τη δυνατότητα επεξεργασίας κάθε εισερχόμενου μηνύματος με περισσότερους από δύο τρόπους ταυτόχρονα, για την ταυτόχρονη παραγωγή διαφορετικών αποτελεσμάτων. Με αυτόν τον τρόπο είναι δυνατή η προώθηση των μηνυμάτων μιας εφαρμογής εισόδου σε περισσότερες από μία εφαρμογές εξόδου, οι οποίες χρησιμοποιούν διαφορετικά πρωτόκολλα επικοινωνίας.

Με βάση τα ανωτέρω μπορούν να εξαχθούν οι προδιαγραφές επεξεργασίας των μηνυμάτων όπως περιγράφονται στον πίνακα 2.3.

Πίνακας 2.3: Προδιαγραφές επεξεργασίας των μηνυμάτων

Κωδικός	Προδιαγραφή
ΠΕΠΜ-1	Ο χρήστης του πρέπει να έχει τη δυνατότητα να ορίζει νέους τρόπους επεξεργασίας των εισερχόμενων μηνυμάτων με ένα δυναμικό τρόπο, χωρίς την απαίτηση μετατροπών στο σύστημα.
ΠΕΠΜ-2	Ο χρήστης πρέπει να έχει τη δυνατότητα παραμετροποίησης του τρόπου επεξεργασίας της προδιαγραφής ΠΕΠΜ-1.
ΠΕΠΜ-3	Στην περίπτωση που η επεξεργασία ενός μηνύματος χρειάζεται αρκετό χρόνο και η εφαρμογή εισόδου στέλνει ένα καινούργιο μήνυμα, το σύστημα πρέπει να περιμένει να τελειώσει η επεξεργασία του παλιού μηνύματος και μετά να επεξεργαστεί το καινούργιο.
ΠΕΠΜ-4	Ο μέγιστος αριθμός των μηνυμάτων σε αναμονή της προδιαγραφής ΠΕΠΜ-3 πρέπει να καθορίζεται από το χρήστη. Περαιτέρω μηνύματα θα αγνοούνται.
ΠΕΠΜ-5	Το σύστημα πρέπει να έχει τη δυνατότητα να επεξεργαστεί το κάθε εισερχόμενο μήνυμα με περισσότερους από έναν τρόπους, με αποτέλεσμα την ταυτόχρονη παραγωγή διαφορετικών αποτελεσμάτων.
ΠΕΠΜ-6	Η καθυστέρηση στην επεξεργασία με έναν από τους τρόπους της προδιαγραφής ΠΕΠΜ-5 δεν πρέπει να επηρεάζει την απόδοση της επεξεργασίας με τους υπόλοιπους τρόπους.

Κωδικός	Προδιαγραφή
ΠΕΠΜ-7	Το αποτέλεσμα της επεξεργασίας ενός μηνύματος με έναν από τους τρόπους επεξεργασίας μπορεί να δέχεται περαιτέρω επεξεργασία με άλλους τρόπους επεξεργασίας που έχει ορίσει ο χρήστης.
ΠΕΠΜ-8	Ο χρήστης πρέπει να ορίζει τη ροή επεξεργασίας των εισερχόμενων μηνυμάτων (προδιαγραφές ΠΕΠΜ-5 και ΠΕΠΜ-7) κατά την αρχικοποίηση του συστήματος.

2.1.4. Προδιαγραφές εξερχόμενων μηνυμάτων

Εδώ καταγράφονται οι απαιτήσεις του συστήματος σχετικές με τα μηνύματα προς τις εφαρμογές εξόδου (προδιαγραφή ΠΓΧ-3).

Για τους ίδιους λόγους με τις εφαρμογές εισόδου, το σύστημα πρέπει να επικοινωνεί με τις εφαρμογές εξόδου με τα πρωτόκολλα επικοινωνίας τα οποία χρησιμοποιούν. Όπως και με τα πρωτόκολλα επικοινωνίας με τις εφαρμογές εισόδου, ο χρήστης πρέπει να έχει τη δυνατότητα να ορίζει νέα πρωτόκολλα επικοινωνίας με τις εφαρμογές εξόδου, χωρίς την απαίτηση μετατροπών στον κώδικα του συστήματος, όπως επίσης πρέπει να έχει τη δυνατότητα της παραμετροποίησής τους κατά την αρχικοποίηση του συστήματος.

Όπως αναφέρθηκε στην προδιαγραφή ΠΕΙΜ-4 το σύστημα μπορεί να συνδέεται με περισσότερες από μία εφαρμογές εισόδου ταυτόχρονα. Παρόμοια πρέπει να έχει τη δυνατότητα να συνδέεται επίσης και με περισσότερες από μία εφαρμογές εξόδου. Με αυτόν τον τρόπο το σύστημα μπορεί να χρησιμοποιηθεί για τη σύνδεση περισσοτέρων του ενός ζεύγους εφαρμογών. Όπως και με τις εφαρμογές εισόδου, οι καθυστερήσεις στην επικοινωνία με μία εφαρμογή εξόδου δεν πρέπει να επηρεάζουν την επικοινωνία με τις υπόλοιπες εφαρμογές εξόδου.

Η αποστολή ενός μηνύματος σε μια εφαρμογή εξόδου, ανάλογα με το πρωτόκολλο επικοινωνίας, μπορεί να διαρκέσει ένα αρκετά μεγάλο χρονικό διάστημα. Στην περίπτωση που το σύστημα θέλει να στείλει ένα δεύτερο μήνυμα στην εφαρμογή αυτή πριν η αποστολή του πρώτου μηνύματος έχει εκτελεστεί, το νέο μήνυμα δεν πρέπει να χάνεται, αλλά να αποστέλλεται αμέσως μετά την αποστολή του παλιού. Για την αποφυγή υπερβολικής χρήσης της μνήμης, όπως και κατά την επεξεργασία των μηνυμάτων, ο χρήστης μπορεί να ορίζει ένα μέγιστο αριθμό μηνυμάτων που θα αναμένουν να αποσταλούν για κάθε εφαρμογή εξόδου, και περαιτέρω μηνύματα θα αγνοούνται.

Τέλος, για μεγαλύτερη ευελιξία του συστήματος, το κάθε εισερχόμενο ή επεξεργασμένο μήνυμα μπορεί να αποστέλλεται σε περισσότερες από μία εφαρμογές εξόδου. Με αυτόν τον τρόπο η έξοδος μιας εφαρμογής εισόδου μπορεί να προωθείται σε περισσότερες από μία ανεξάρτητες

εφαρμογές για περαιτέρω επεξεργασία.

Με βάση τα ανωτέρω μπορούν να εξαχθούν οι προδιαγραφές εξερχόμενων μηνυμάτων όπως περιγράφονται στον πίνακα 2.4.

Πίνακας 2.4: Προδιαγραφές εξερχόμενων μηνυμάτων

Κωδικός	Προδιαγραφή
ΠΕΞΜ-1	Η επικοινωνία με τις εφαρμογές εξόδου πρέπει να γίνεται με το πρωτόκολλο επικοινωνίας των εφαρμογών.
ΠΕΞΜ-2	Ο χρήστης πρέπει να έχει τη δυνατότητα να ορίζει νέα πρωτόκολλα επικοινωνίας με τις εφαρμογές εξόδου με έναν δυναμικό τρόπο, χωρίς την απαίτηση μετατροπών στο σύστημα.
ΠΕΞΜ-3	Ο χρήστης πρέπει να έχει τη δυνατότητα παραμετροποίησης του κάθε πρωτοκόλλου της προδιαγραφής ΠΕΞΜ-2.
ΠΕΞΜ-4	Το σύστημα πρέπει να επιτρέπει την ταυτόχρονη επικοινωνία με έναν απροσδιόριστο αριθμό εφαρμογών εξόδου, οι οποίες μπορεί να χρησιμοποιούν διαφορετικά πρωτόκολλα επικοινωνίας.
ΠΕΞΜ-5	Οι καθυστερήσεις στην επικοινωνία με κάποια εφαρμογή εξόδου δεν πρέπει να επηρεάζουν την απόδοση της επικοινωνίας με της υπόλοιπες εφαρμογές εξόδου.
ΠΕΞΜ-6	Στην περίπτωση που η αποστολή ενός μηνύματος σε κάποια εφαρμογή εξόδου χρειάζεται αρκετό χρόνο και το σύστημα είναι έτοιμο να στείλει ένα καινούργιο μήνυμα στην ίδια εφαρμογή, το σύστημα πρέπει να περιμένει να αποσταλεί το παλιό μήνυμα και μετά να αποστείλει το καινούργιο.
ΠΕΞΜ-7	Ο μέγιστος αριθμός των μηνυμάτων σε αναμονή της προδιαγραφής ΠΕΞΜ-6 πρέπει να καθορίζεται από το χρήστη. Περαιτέρω μηνύματα θα αγνοούνται.
ΠΕΞΜ-8	Το σύστημα πρέπει να έχει τη δυνατότητα προώθησης του κάθε εισερχόμενου ή επεξεργασμένου μηνύματος σε περισσότερες από μία εφαρμογές εξόδου.

2.1.5. Προδιαγραφές διεπαφών του χρήστη

Εδώ καταγράφονται οι απαιτήσεις του συστήματος σχετικές με τις διεπαφές του χρήστη (προδιαγραφή ΠΓΧ-5).

Μετά την εκκίνηση του συστήματος ο χρήστης πρέπει να βλέπει ένα διάγραμμα, το οποίο θα δείχνει τις συνδέσεις με τις εφαρμογές εισόδου και εξόδου, τους τρόπους επεξεργασίας που έχει ορίσει ο χρήστης, καθώς και τα μονοπάτια που ακολουθούν τα εισερχόμενα μηνύματα από την παραλαβή τους από τις εφαρμογές εισόδου για την επεξεργασία τους και την αποστολή τους στις

εφαρμογές εξόδου. Το κάθε στοιχείο του διαγράμματος πρέπει να χαρακτηρίζεται από κάποιο όνομα για να το ξεχωρίζει ο χρήστης, το οποίο θα ορίζεται κατά την αρχικοποίηση του συστήματος.

Για μεγαλύτερη ευκολία, ο χρήστης πρέπει να έχει τη δυνατότητα να έχει επιλεγμένη κάθε στιγμή είτε τη σύνδεση με μια εφαρμογή εισόδου ή εξόδου, είτε ένα στάδιο επεξεργασίας. Το σύστημα θα παρουσιάζει στο χρήστη μία περιγραφή της λειτουργίας της επιλογής του. Η επιλογή του χρήστη πρέπει να φαίνεται στο διάγραμμα και, όταν η επιλογή είναι μια σύνδεση με εφαρμογή εισόδου ή ένα στάδιο επεξεργασίας, πρέπει επίσης να φαίνονται στο διάγραμμα οι εφαρμογές εξόδου και τα στάδια επεξεργασίας όπου οδηγούνται τα μηνύματα της επιλογής. Στην περίπτωση που η επιλογή είναι ένα στάδιο επεξεργασίας ή μια σύνδεση με εφαρμογή εξόδου, στο διάγραμμα πρέπει επίσης να φαίνεται από που δέχεται τα μηνύματα η επιλογή του χρήστη.

Τέλος, για κάθε εφαρμογή εισόδου, ο χρήστης πρέπει να βλέπει μια περιγραφή των μηνυμάτων που λαμβάνει το σύστημα. Το ίδιο πρέπει να συμβαίνει και μετά από κάθε στάδιο επεξεργασίας, ώστε ο χρήστης να βλέπει τις τιμές του επεξεργασμένου μηνύματος.

Με βάση τα ανωτέρω μπορούν να εξαχθούν οι προδιαγραφές διεπαφών του χρήστη όπως περιγράφονται στον πίνακα 2.5.

Πίνακας 2.5: Προδιαγραφές διεπαφών του χρήστη

Κωδικός	**Προδιαγραφή**
ΠΔΧ-1	Το σύστημα πρέπει να παρουσιάζει στο χρήστη ένα διάγραμμα το οποίο θα δείχνει τις συνδέσεις με τις εφαρμογές εισόδου και εξόδου, τους τρόπους επεξεργασίας των μηνυμάτων και τα μονοπάτια που ακολουθούν τα μηνύματα.
ΠΔΧ-2	Τα στοιχεία του διαγράμματος πρέπει να έχουν κάποιο όνομα το οποίο θα ορίζει ο χρήστης κατά την αρχικοποίηση του συστήματος.
ΠΔΧ-3	Στο διάγραμμα μπορεί να είναι επιλεγμένη κάθε στιγμή είτε η σύνδεση με μια εφαρμογή εισόδου ή εξόδου, είτε ένα στάδιο επεξεργασίας των μηνυμάτων.
ΠΔΧ-4	Το σύστημα θα παρουσιάζει στο χρήστη μια περιγραφή του επιλεγμένου στοιχείου του διαγράμματος.
ΠΔΧ-5	Όταν το επιλεγμένο στοιχείο είναι μια σύνδεση με μια εφαρμογή εισόδου ή ένα στάδιο επεξεργασίας, στο διάγραμμα πρέπει να φαίνονται οι εφαρμογές εξόδου και τα στάδια επεξεργασίας όπου οδηγούνται τα μηνύματα μετά το επιλεγμένο στοιχείο.

Κωδικός	Προδιαγραφή
ΠΔΧ-6	Όταν το επιλεγμένο στοιχείο είναι ένα στάδιο επεξεργασίας ή μια σύνδεση με εφαρμογή εξόδου, στο διάγραμμα πρέπει να φαίνεται από που δέχεται τα μηνύματα το επιλεγμένο στοιχείο.
ΠΔΧ-7	Ο χρήστης πρέπει να βλέπει μια περιγραφή της τιμής των μηνυμάτων που λαμβάνει το σύστημα για κάθε εφαρμογή εισόδου.
ΠΔΧ-8	Ο χρήστης πρέπει να βλέπει μια περιγραφή της τιμής των μηνυμάτων μετά από κάθε στάδιο επεξεργασίας.

2.2. Περιπτώσεις χρήσεως

Σε αυτήν την ενότητα εντοπίζονται οι περιπτώσεις χρήσεως του συστήματος, βασισμένες στις λειτουργικές προδιαγραφές της προηγούμενης ενότητας. Με τον όρο περίπτωση χρήσεως εννοούμε τις ενέργειες τις οποίες πρέπει να υποστηρίζει το σύστημα. Εδώ οι περιπτώσεις χρήσεως απλώς εντοπίζονται και καταγράφονται με μια μικρή περιγραφή. Στην επόμενη ενότητα θα αναλυθούν σε σενάρια.

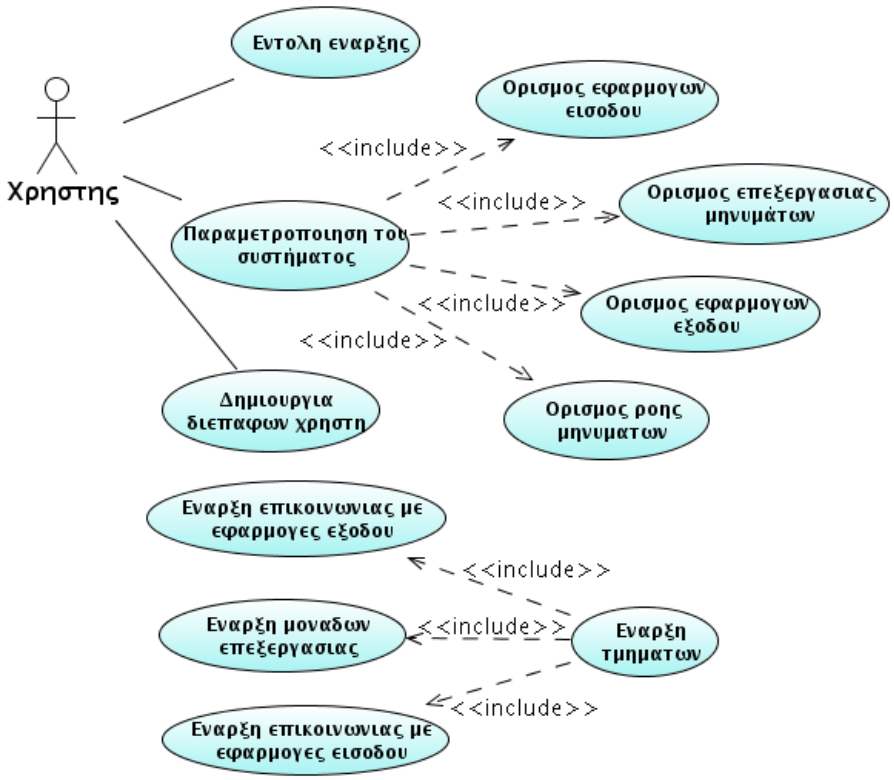
2.2.1. Περιπτώσεις χρήσεως αρχικοποίησης του συστήματος

Σε αυτήν την κατηγορία κατατάσσονται οι περιπτώσεις χρήσεως σχετικές με την αρχικοποίηση του συστήματος. Με τον όρο αρχικοποίηση του συστήματος εννοούμε τις ενέργειες που χρειάζονται ώστε να ξεκινήσει το σύστημα να αναμένει μηνύματα από τις εφαρμογές εισόδου για επεξεργασία και αποστολή στις εφαρμογές εξόδου. Οι περιπτώσεις χρήσεως αυτής της κατηγορίας καθώς και οι σχέσεις μεταξύ τους φαίνονται στο σχήμα 2.1.

Όπως φαίνεται και στο σχήμα, η αρχικοποίηση του συστήματος μπορεί να διαιρεθεί στην εντολή έναρξης, στην παραμετροποίηση του συστήματος, στη δημιουργία των διεπαφών του χρήστη και στην έναρξη των διάφορων τμημάτων του συστήματος.

2.2.1.1. Εντολή έναρξης

Η εντολή έναρξης είναι η ενέργεια που εκτελεί ο χρήστης για να εκκινήσει το σύστημα. Η ενέργεια αυτή είναι ιδιαίτερα απλή (ο χρήστης απλά εισάγει την εντολή για εκκίνηση του συστήματος). Σε αυτό το σημείο ο χρήστης θα μπορεί να επιλέξει αν θέλει να χρησιμοποιήσει το σύστημα με τις διεπαφές επικοινωνίας ενεργοποιημένες, ή αν θέλει να χρησιμοποιήσει το σύστημα χωρίς διεπαφές, για μεγιστοποίηση της επίδοσης (προδιαγραφή ΠΓΧ-7).



Σχήμα 2.1: Περιπτώσεις χρήσεως αρχικοποίησης του συστήματος

2.2.1.2. Παραμετροποίηση του συστήματος

Η παραμετροποίηση του συστήματος πηγάζει από την προδιαγραφή ΠΓΧ-4 σύμφωνα με την οποία ο χρήστης πρέπει να προσδιορίζει κατά την αρχικοποίηση του συστήματος τις εφαρμογές εισόδου, την επεξεργασία των εισερχόμενων μηνυμάτων και τις εφαρμογές εξόδου, καθώς και από τις προδιαγραφές ΠΕΠΜ-5, ΠΕΠΜ-7, ΠΕΠΜ-8 και ΠΕΞΜ-8, οι οποίες προσδιορίζουν τη ροή που ακολουθούν τα μηνύματα στο σύστημα.

Ο ορισμός κάθε εφαρμογής εισόδου πρέπει να περιλαμβάνει το όνομα που θα χρησιμοποιηθεί για τις διεπαφές του χρήστη (προδιαγραφή ΠΔΧ-2), το πρωτόκολλο επικοινωνίας που θα χρησιμοποιηθεί για την επικοινωνία (προδιαγραφή ΠΕΙΜ-1), καθώς και τις παραμέτρους που θέλει να δώσει ο χρήστης για την παραμετροποίηση του πρωτοκόλλου (προδιαγραφή ΠΕΙΜ-3).

Ο ορισμός της επεξεργασίας των μηνυμάτων πρέπει να περιλαμβάνει το όνομα που θα χρησιμοποιηθεί για τις διεπαφές του χρήστη (προδιαγραφή ΠΔΧ-2), τον τρόπο με τον οποίο θα επεξεργαστεί το σύστημα τα μηνύματα (προδιαγραφή ΠΕΠΜ-1), καθώς και τις παραμέτρους που θέλει να δώσει ο χρήστης για την παραμετροποίηση του κάθε τρόπου επεξεργασίας (προδιαγραφή ΠΕΠΜ-2). Επίσης ο ορισμός της επεξεργασίας των μηνυμάτων περιλαμβάνει τον ορισμό του

μεγίστου αριθμού μηνυμάτων που μπορούν να είναι σε αναμονή για επεξεργασία για κάθε συγκεκριμένο τρόπο τρόπο επεξεργασίας (προδιαγραφή ΠΕΠΜ-5).

Ο ορισμός κάθε εφαρμογής εξόδου πρέπει να περιλαμβάνει το όνομα που θα χρησιμοποιηθεί για τις διεπαφές του χρήστη (προδιαγραφή ΠΔΧ-2), το πρωτόκολλο επικοινωνίας που θα χρησιμοποιηθεί (προδιαγραφή ΠΕΞΜ-1) καθώς και τις παραμέτρους που θέλει να δώσει ο χρήστης για την παραμετροποίηση του πρωτοκόλλου (προδιαγραφή ΠΕΞΜ-3). Επίσης ο ορισμός των εφαρμογών εξόδου περιλαμβάνει τον ορισμό του μεγίστου αριθμού μηνυμάτων που μπορούν να είναι σε αναμονή για αποστολή για κάθε εφαρμογή εξόδου (προδιαγραφή ΠΕΞΜ-7).

Τέλος, κατά την παραμετροποίηση του συστήματος πρέπει να ορίζεται και η ροή που θα ακολουθούν τα εισερχόμενα μηνύματα από τις εφαρμογές εισόδου για την επεξεργασία και αποστολή τους στις εφαρμογές εξόδου, όπως περιγράφεται στις προδιαγραφές ΠΕΠΜ-5, ΠΕΠΜ-7, ΠΕΠΜ-8 και ΠΕΞΜ-8. Πιο συγκεκριμένα, ο χρήστης πρέπει να ορίζει για την κάθε εφαρμογή εισόδου τους τρόπους επεξεργασίας και τις εφαρμογές εξόδου στις οποίες θα προωθούνται τα μηνύματά της, και για τον κάθε τρόπο επεξεργασίας τους περαιτέρω τρόπους επεξεργασίας και τις εφαρμογές εξόδου στις οποίες θα προωθούνται τα μηνύματα που επεξεργάσθηκαν.

2.2.1.3. Δημιουργία διεπαφών του χρήστη

Η δημιουργία των διεπαφών επικοινωνίας με το χρήστη είναι μια ενέργεια η εκτέλεση ή όχι της οποίας εξαρτάται από την επιλογή του χρήστη κατά την εντολή εκτέλεσης. Κατά τη διάρκεια αυτής της ενέργειας το σύστημα πρέπει να δημιουργεί και να παρουσιάζει στο χρήστη το διάγραμμα της προδιαγραφής ΠΔΧ-1. Η ενέργεια αυτή θα αναλυθεί περισσότερο στα κεφάλαια που καλύπτουν τη σχεδίαση και την υλοποίηση του συστήματος.

2.2.1.4. Έναρξη των τμημάτων του συστήματος

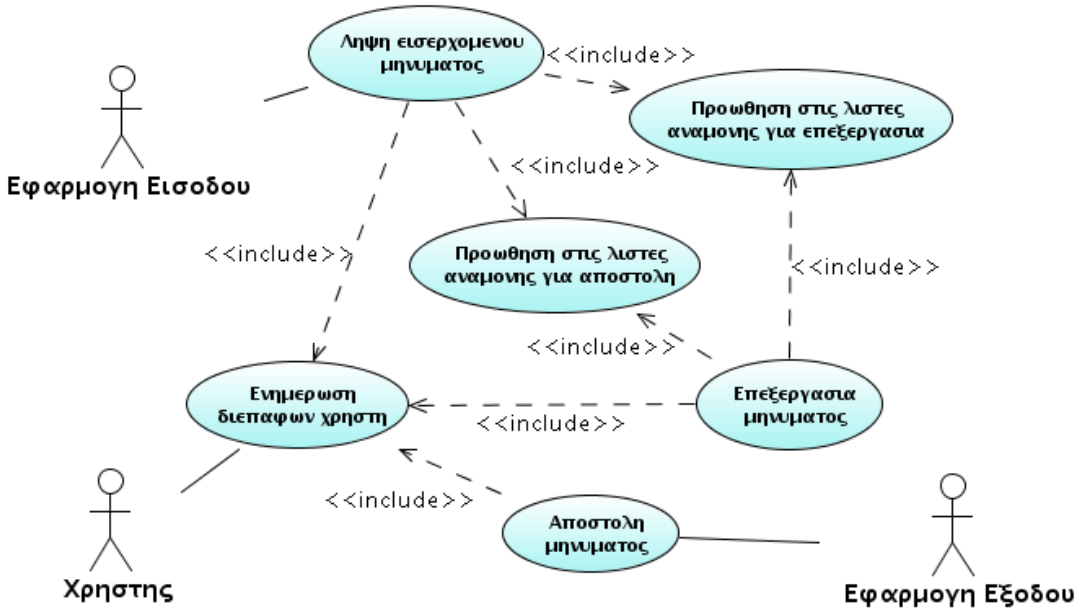
Τέλος, για την επιτυχή αρχικοποίηση, το σύστημα πρέπει να ξεκινήσει την επικοινωνία με τις εφαρμογές εισόδου και εξόδου καθώς και όλα τα στάδια επεξεργασίας. Η έναρξη αυτή θα αποτελείται από απαιτητικές σε χρόνο ενέργειες, η εκτέλεση των οποίων απαιτείται μία μόνο φορά, καθώς και από τις απαραίτητες ενέργειες ώστε το σύστημα να είναι έτοιμο να λάβει τα εισερχόμενα μηνύματα από τις εφαρμογές εισόδου. Για παράδειγμα, αν η επικοινωνία με μια εφαρμογή εξόδου γίνεται μέσω ενός αρχείου, το σύστημα μπορεί να ανοίξει το αρχείο για γράψιμο μία μόνο φορά κατά την έναρξη της επικοινωνίας, ή αν η επικοινωνία με μια εφαρμογή εισόδου γίνεται μέσω TCP/IP μηνύματα, κατά την έναρξη της επικοινωνίας το σύστημα μπορεί να ξεκινάει ένα TCP/IP server.

Για την ομαλή λειτουργία του συστήματος και για την αποφυγή απώλειας μηνυμάτων, το σύστημα

πρέπει πρώτα να ξεκινάει την επικοινωνία με τις εφαρμογές εξόδου, μετά τα στάδια επεξεργασίας και τέλος την επικοινωνία με τις εφαρμογές εισόδου.

2.2.2. Περιπτώσεις χρήσεως μετάδοσης μηνυμάτων

Σε αυτήν την κατηγορία κατατάσσονται οι περιπτώσεις χρήσεως σχετικές με τις ενέργειες που εκτελεί το σύστημα για τη λήψη, επεξεργασία και αποστολή των μηνυμάτων από τις εφαρμογές εισόδου στις εφαρμογές εξόδου. Οι περιπτώσεις χρήσεως αυτής της κατηγορίας καθώς και οι σχέσεις μεταξύ τους φαίνονται στο σχήμα 2.2.



Σχήμα 2.2: Περιπτώσεις χρήσεως μετάδοσης μηνυμάτων

Όπως φαίνεται και στο σχήμα, η μετάδοση ενός μηνύματος μπορεί να διαιρεθεί στη λήψη του εισερχόμενου μηνύματος από την εφαρμογή εισόδου, στην επεξεργασία του μηνύματος και στην αποστολή του εξερχόμενου μηνύματος προς την εφαρμογή εξόδου.

2.2.2.1. Λήψη εισερχόμενου μηνύματος

Η ενέργεια αυτή εκτελείται όταν μία εφαρμογή εισόδου στέλνει ένα μήνυμα στο σύστημα. Το σύστημα πρέπει να λαμβάνει το εισερχόμενο μήνυμα σύμφωνα με το πρωτόκολλο επικοινωνίας της εφαρμογής εισόδου και, όπως φαίνεται στο σχήμα, πρέπει να προωθεί το μήνυμα στις λίστες αναμονής για επεξεργασία ή για απευθείας αποστολή στις εφαρμογές εξόδου, όπως έχει ορίσει ο χρήστης κατά την αρχικοποίηση του συστήματος (προδιαγραφές ΠΓΧ-2, ΠΓΧ-3 και ΠΕΠΜ-5).

Επίσης, αν το σύστημα έχει εκκινηθεί με την υποστήριξη διεπαφών επικοινωνίας με το χρήστη, το διάγραμμα που παρουσιάζει το σύστημα στο χρήστη πρέπει να ενημερώνεται για την καινούργια τιμή του μηνύματος (προδιαγραφή ΠΔΧ-7). Αμέσως μετά από αυτές τις ενέργειες, το σύστημα πρέπει να αναμένει για το επόμενο εισερχόμενο μήνυμα από τη συγκεκριμένη εφαρμογή εισόδου.

2.2.2.2. Επεξεργασία μηνύματος

Κατά την ενέργεια αυτή το σύστημα πρέπει να επεξεργάζεται τα μηνύματα που είναι σε αναμονή για επεξεργασία, σύμφωνα με τους τρόπους που έχει ορίσει ο χρήστης (προδιαγραφή ΠΕΠΜ-1). Όταν το σύστημα τελειώσει την επεξεργασία ενός μηνύματος πρέπει να προωθεί το αποτέλεσμα της επεξεργασίας στις λίστες αναμονής για περαιτέρω επεξεργασία ή για αποστολή στις εφαρμογές εξόδου, όπως έχει ορίσει ο χρήστης κατά την αρχικοποίηση του συστήματος (προδιαγραφές ΠΓΧ-3 και ΠΕΠΜ-5). Επίσης, αν το σύστημα έχει εκκινηθεί με την υποστήριξη διεπαφών επικοινωνίας με το χρήστη, το διάγραμμα που παρουσιάζει το σύστημα στο χρήστη πρέπει να ενημερώνεται για την τιμή του αποτελέσματος της επεξεργασίας (προδιαγραφή ΠΔΧ-8). Μετά την εκτέλεση των ανωτέρω ενεργειών, το σύστημα πρέπει να ξεκινά την επεξεργασία του επόμενου μηνύματος στη λίστα αναμονής για επεξεργασία.

2.2.2.3. Αποστολή μηνύματος

Κατά την ενέργεια αυτή το σύστημα πρέπει να στέλνει τα μηνύματα που είναι σε αναμονή για αποστολή στις εφαρμογές εξόδου, όπως έχει ορίσει ο χρήστης κατά την αρχικοποίηση του συστήματος. Μετά την αποστολή ενός μηνύματος από τη λίστα το σύστημα πρέπει να συνεχίζει με την αποστολή του επόμενου μηνύματος στη λίστα αναμονής.

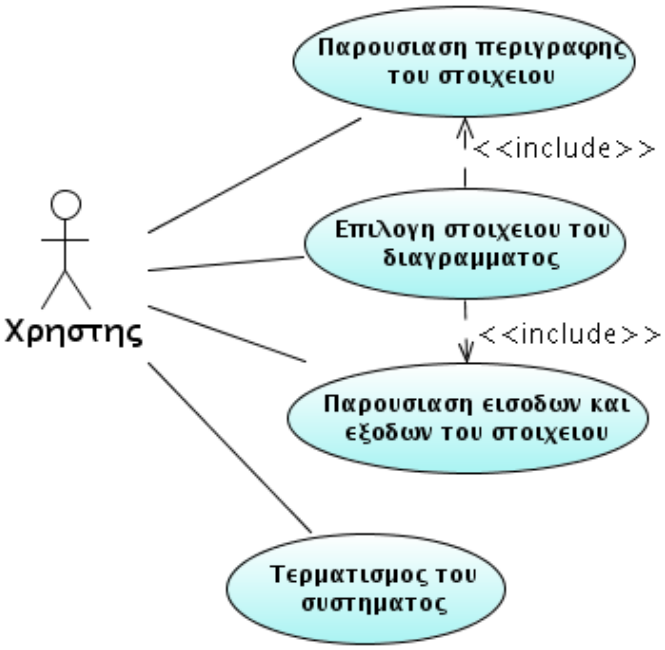
2.2.3. Περιπτώσεις χρήσεως αλληλεπίδρασης με το χρήστη

Εδώ περιγράφονται οι ενέργειες που πρέπει να υποστηρίζει το σύστημα, οι οποίες είναι σχετικές με αλληλεπιδράσεις του χρήστη με το σύστημα. Οι ενέργειες αυτές είναι στενά συνδεδεμένες με τις διεπαφές του χρήστη, οπότε ισχύουν μόνο στην περίπτωση όπου κατά την αρχικοποίηση του συστήματος ο χρήστης επέλεξε την εκτέλεση του συστήματος με διεπαφές. Οι περιπτώσεις χρήσεως αυτής της κατηγορίας, καθώς και οι σχέσεις μεταξύ τους, φαίνονται στο σχήμα 2.3.

Όπως φαίνεται και στο σχήμα, ο χρήστης μπορεί να αλληλεπιδράσει με το σύστημα με δύο τρόπους, την επιλογή ενός στοιχείου στο διάγραμμα και τον τερματισμό του συστήματος.

2.2.3.1. Επιλογή στοιχείου του διαγράμματος

Κατά τη διάρκεια εκτέλεσης του συστήματος, ο χρήστης μπορεί να επιλέξει ένα στοιχείο του



Σχήμα 2.3: Περιπτώσεις χρήσεως αλληλεπίδρασης με το χρήστη

διαγράμματος, δηλαδή τη σύνδεση με μια εφαρμογή εισόδου ή εξόδου ή ένα στάδιο επεξεργασίας (προδιαγραφή ΠΔΧ-3). Μετά την επιλογή αυτή, το σύστημα πρέπει να παρουσιάζει στο χρήστη την περιγραφή του επιλεγμένου στοιχείου (προδιαγραφή ΠΔΧ-4) καθώς και τις εισόδους και εξόδους του επιλεγμένου στοιχείου (προδιαγραφές ΠΔΧ-5 και ΠΔΧ-6).

2.2.3.2. Τερματισμός του συστήματος

Η ενέργεια αυτή είναι πολύ απλή. Ο χρήστης απλά επιλέγει να τερματίσει το σύστημα. Το σύστημα πρέπει να ζητάει από το χρήστη την επιβεβαίωση του τερματισμού (για την αποφυγή τερματισμού κατά λάθος) και, αν ο χρήστης επιβεβαιώσει τον τερματισμό, πρέπει να τερματίζει τη λειτουργία του.

2.3. Πιθανά σενάρια

Σε αυτήν την ενότητα αναλύονται οι περιπτώσεις χρήσεως της προηγούμενης ενότητας. Για κάθε μία περίπτωση χρήσης δίνεται αρχικά ένα σενάριο το οποίο αντιστοιχεί σε μια πετυχημένη εκτέλεση της χρήσης, και μετά αναλύονται εναλλακτικά σενάρια που παρουσιάζουν τη συμπεριφορά του συστήματος όταν δεν είναι δυνατή η αναμενόμενη εκτέλεση της χρήσης.

Τα βήματα των σεναρίων είναι αριθμημένα για ευκολότερη αναφορά σε αυτά στα επόμενα κεφάλαια. Όταν ένα βήμα περιγράφει μια περίπλοκη ενέργεια μπορεί να διαιρείται σε περισσότερα μικρότερα βήματα τα οποία αναλύουν περαιτέρω την ενέργεια. Αν κάποιο βήμα μπορεί να ακολουθήσει εναλλακτικά σενάρια, αυτά αριθμούνται με τα γράμματα της αλφαβήτου, ακολουθούμενα από τον αριθμό του εναλλακτικού βήματος.

2.3.1. Σενάρια εντολής έναρξης

2.3.1.1. Πετυχημένο σενάριο

Εδώ περιγράφεται το πετυχημένο σενάριο για την εντολή έναρξης του συστήματος. Αυτό είναι το σενάριο που το σύστημα θα ακολουθήσει αν δεν προκύψει κανένα πρόβλημα.

Πίνακας 2.6: Πετυχημένο σενάριο εντολής έναρξης

Βήμα	**Ενέργεια**
1.	Ο χρήστης εισάγει την εντολή έναρξης.
2.	Το σύστημα εξάγει από την εντολή έναρξης αν ο χρήστης θέλει να ενεργοποιήσει τις διεπαφές επικοινωνίας ή όχι.
3.	Η αρχικοποίηση συνεχίζεται με την παραμετροποίηση του συστήματος.

2.3.1.2. Εναλλακτικά σενάρια

Εδώ περιγράφονται τα σενάρια τα οποία θα εξελιχθούν στην περίπτωση που κάποιο βήμα του πετυχημένου σεναρίου δεν καταφέρει να ολοκληρωθεί με επιτυχία.

Πίνακας 2.7: Εναλλακτικά σενάρια εντολής έναρξης

Βήμα	**Ενέργεια**
2.α.	Ο χρήστης δεν έχει επιλέξει αν οι διεπαφές επικοινωνίας θα ενεργοποιηθούν ή όχι.
2.α.1.	Το σύστημα υποθέτει ότι ο χρήστης δεν θέλει να ενεργοποιήσει τις διεπαφές επικοινωνίας.
2.α.2.	Η εκτέλεση συνεχίζεται από το βήμα 3 του πετυχημένου σεναρίου.

2.3.2. Σενάρια παραμετροποίησης του συστήματος

2.3.2.1. Πετυχημένο σενάριο

Εδώ περιγράφεται το πετυχημένο σενάριο για την παραμετροποίηση του συστήματος. Αυτό είναι το σενάριο που το σύστημα θα ακολουθήσει αν δεν προκύψει κανένα πρόβλημα.

Πίνακας 2.8: Πετυχημένο σενάριο παραμετροποίησης του συστήματος

Βήμα	Ενέργεια
1.	Ο χρήστης ορίζει τις εφαρμογές εισόδου.
1.1.	Ο χρήστης ορίζει το όνομα που θα χρησιμοποιηθεί στις διεπαφές του χρήστη για την εφαρμογή εισόδου.
1.2.	Ο χρήστης ορίζει το πρωτόκολλο που θα χρησιμοποιηθεί για την επικοινωνία με την εφαρμογή.
1.3.	Ο χρήστης ορίζει τις παραμέτρους για την παραμετροποίηση του πρωτοκόλλου.
1.4.	Η εκτέλεση συνεχίζεται από το βήμα 1.1 μέχρι ο χρήστης να ορίσει όλες τις εφαρμογές εισόδου.
2.	Ο χρήστης ορίζει τους τρόπους επεξεργασίας των μηνυμάτων.
2.1.	Ο χρήστης ορίζει το όνομα που θα χρησιμοποιηθεί στις διεπαφές του χρήστη για τον τρόπο επεξεργασίας.
2.2.	Ο χρήστης ορίζει τη μέθοδο με την οποία το σύστημα θα επεξεργαστεί τα μηνύματα.
2.3.	Ο χρήστης ορίζει τις παραμέτρους για την παραμετροποίηση του τρόπου επεξεργασίας.
2.4.	Ο χρήστης ορίζει το μέγιστο αριθμό μηνυμάτων που μπορούν να είναι σε αναμονή για επεξεργασία για αυτόν τον τρόπο επεξεργασίας.
2.5.	Η εκτέλεση συνεχίζεται από το βήμα 2.1 μέχρι ο χρήστης να ορίσει όλους τους τρόπους επεξεργασίας.
3.	Ο χρήστης ορίζει τις εφαρμογές εξόδου.
3.1.	Ο χρήστης ορίζει το όνομα που θα χρησιμοποιηθεί στις διεπαφές του χρήστη για την εφαρμογή εξόδου.
3.2.	Ο χρήστης ορίζει το πρωτόκολλο που θα χρησιμοποιηθεί για την επικοινωνία με την εφαρμογή.
3.3.	Ο χρήστης ορίζει τις παραμέτρους για την παραμετροποίηση του πρωτοκόλλου.
3.4.	Ο χρήστης ορίζει το μέγιστο αριθμό μηνυμάτων που μπορούν να είναι σε αναμονή για αποστολή σε αυτήν την εφαρμογή εξόδου.
3.5.	Η εκτέλεση συνεχίζεται από το βήμα 3.1 μέχρι ο χρήστης να ορίσει όλες τις εφαρμογές εξόδου.
4.	Ο χρήστης ορίζει τη ροή των μηνυμάτων.
4.1.	Για κάθε εφαρμογή εισόδου, ο χρήστης ορίζει τα στάδια επεξεργασίας στα οποία θα προωθηθούν τα εισερχόμενα μηνύματα από αυτήν την εφαρμογή.
4.2.	Για κάθε εφαρμογή εισόδου, ο χρήστης ορίζει τις εφαρμογές εξόδου στις οποίες θα αποσταλούν χωρίς επεξεργασία τα εισερχόμενα μηνύματα.

Βήμα	Ενέργεια
4.3.	Για κάθε τρόπο επεξεργασίας, ο χρήστης ορίζει τα στάδια επεξεργασίας στα οποία θα προωθηθεί το αποτέλεσμα της επεξεργασίας για περαιτέρω επεξεργασία.
4.4.	Για κάθε τρόπο επεξεργασίας, ο χρήστης ορίζει τις εφαρμογές εξόδου στις οποίες θα αποσταλεί το αποτέλεσμα της επεξεργασίας.

2.3.2.2. Εναλλακτικά σενάρια

Εδώ περιγράφονται τα σενάρια τα οποία θα εξελιχθούν στην περίπτωση που κάποιο βήμα του πετυχημένου σεναρίου δεν καταφέρει να ολοκληρωθεί με επιτυχία.

Πίνακας 2.9: Εναλλακτικά σενάρια παραμετροποίησης του συστήματος

Βήμα	Ενέργεια
1.1.α.	Το όνομα που έδωσε ο χρήστης έχει ήδη χρησιμοποιηθεί για κάποιο άλλο στοιχείο.
1.1.α.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
1.1.α.2.	Η λειτουργία του συστήματος τερματίζεται.
1.1.β.	Ο χρήστης δεν έδωσε κανένα όνομα.
1.1.β.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
1.1.β.2.	Η λειτουργία του συστήματος τερματίζεται.
1.2.α.	Το πρωτόκολλο που έδωσε ο χρήστης δεν είναι πρωτόκολλο επικοινωνίας με εφαρμογή εισόδου.
1.2.α.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
1.2.α.2.	Η λειτουργία του συστήματος τερματίζεται.
1.2.β.	Ο χρήστης δεν έδωσε κάποιο πρωτόκολλο για την επικοινωνία.
1.2.β.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
1.2.β.2.	Η λειτουργία του συστήματος τερματίζεται.
1.3.α.	Ο χρήστης παρέλειψε κάποια παράμετρο η οποία είναι υποχρεωτική.
1.3.α.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
1.3.α.2.	Η λειτουργία του συστήματος τερματίζεται.
1.3.β.	Κάποια παράμετρος έχει τιμή ασύμβατη από αυτήν που περιμένει το πρωτόκολλο (πχ ο χρήστης έδωσε έναν αρνητικό αριθμό όταν το πρωτόκολλο δέχεται μόνο θετικούς).
1.3.β.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
1.3.β.2.	Η λειτουργία του συστήματος τερματίζεται.
2.1.α.	Το όνομα που έδωσε ο χρήστης έχει ήδη χρησιμοποιηθεί για κάποιο άλλο στοιχείο.
2.1.α.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
2.1.α.2.	Η λειτουργία του συστήματος τερματίζεται.

Βήμα	Ενέργεια
2.1.β.	Ο χρήστης δεν έδωσε κανένα όνομα.
2.1.β.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
2.1.β.2.	Η λειτουργία του συστήματος τερματίζεται.
2.2.α.	Η μέθοδος που έδωσε ο χρήστης δεν είναι μέθοδος επεξεργασίας μηνυμάτων.
2.2.α.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
2.2.α.2.	Η λειτουργία του συστήματος τερματίζεται.
2.2.β.	Ο χρήστης δεν έδωσε κάποια μέθοδο επεξεργασίας.
2.2.β.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
2.2.β.2.	Η λειτουργία του συστήματος τερματίζεται.
2.3.α.	Ο χρήστης παρέλειψε κάποια παράμετρο η οποία είναι υποχρεωτική.
2.3.α.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
2.3.α.2.	Η λειτουργία του συστήματος τερματίζεται.
2.3.β.	Κάποια παράμετρος έχει τιμή ασύμβατη από αυτήν που περιμένει ο τρόπος επεξεργασίας (πχ ο χρήστης έδωσε έναν αρνητικό αριθμό όταν ο τρόπος επεξεργασίας δέχεται μόνο θετικούς).
2.3.β.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
2.3.β.2.	Η λειτουργία του συστήματος τερματίζεται.
2.4.α.	Ο χρήστης έδωσε κάποιον αρνητικό αριθμό ή 0.
2.4.α.1.	Το σύστημα δεν θέτει όριο για τον αριθμό των μηνυμάτων σε αναμονή για το συγκεκριμένο τρόπο επεξεργασίας.
2.4.α.2.	Η εκτέλεση συνεχίζεται από το βήμα 2.5 του πετυχημένου σεναρίου.
2.4.β.	Η τιμή που έδωσε ο χρήστης δεν είναι κάποιος ακέραιος αριθμός.
2.4.β.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
2.4.β.2.	Η λειτουργία του συστήματος τερματίζεται.
2.4.γ.	Ο χρήστης δεν έδωσε όριο.
2.4.γ.1.	Το σύστημα δεν θέτει όριο για τον αριθμό των μηνυμάτων σε αναμονή για το συγκεκριμένο τρόπο επεξεργασίας.
2.4.γ.2.	Η εκτέλεση συνεχίζεται από το βήμα 3.5 του πετυχημένου σεναρίου.
3.1.α.	Το όνομα που έδωσε ο χρήστης έχει ήδη χρησιμοποιηθεί για κάποιο άλλο στοιχείο.
3.1.α.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
3.1.α.2.	Η λειτουργία του συστήματος τερματίζεται.
3.1.β.	Ο χρήστης δεν έδωσε κανένα όνομα.
3.1.β.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
3.1.β.2.	Η λειτουργία του συστήματος τερματίζεται.
3.2.α.	Το πρωτόκολλο που έδωσε ο χρήστης δεν είναι πρωτόκολλο επικοινωνίας με εφαρμογή

Βήμα	Ενέργεια
	εξόδου.
3.2.α.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
3.2.α.2.	Η λειτουργία του συστήματος τερματίζεται.
3.2.β.	Ο χρήστης δεν έδωσε κάποιο πρωτόκολλο για την επικοινωνία.
3.2.β.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
3.2.β.2.	Η λειτουργία του συστήματος τερματίζεται.
3.3.α.	Ο χρήστης παρέλειψε κάποια παράμετρο η οποία είναι υποχρεωτική.
3.3.α.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
3.3.α.2.	Η λειτουργία του συστήματος τερματίζεται.
3.3.β.	Κάποια παράμετρος έχει τιμή ασύμβατη από αυτήν που περιμένει το πρωτόκολλο (πχ ο χρήστης έδωσε έναν αρνητικό αριθμό όταν το πρωτόκολλο δέχεται μόνο θετικούς).
3.3.β.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
3.3.β.2.	Η λειτουργία του συστήματος τερματίζεται.
3.4.α.	Ο χρήστης έδωσε κάποιον αρνητικό αριθμό ή 0.
3.4.α.1.	Το σύστημα δεν θέτει όριο για τον αριθμό των μηνυμάτων σε αναμονή για τη συγκεκριμένη εφαρμογή εξόδου.
3.4.α.2.	Η εκτέλεση συνεχίζεται από το βήμα 3.5 του πετυχημένου σεναρίου.
3.4.β.	Η τιμή που έδωσε ο χρήστης δεν είναι κάποιος ακέραιος αριθμός.
3.4.β.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
3.4.β.2.	Η λειτουργία του συστήματος τερματίζεται.
3.4.γ.	Ο χρήστης δεν έδωσε όριο.
3.4.γ.1.	Το σύστημα δεν θέτει όριο για τον αριθμό των μηνυμάτων σε αναμονή για τη συγκεκριμένη εφαρμογή εξόδου.
3.4.γ.2.	Η εκτέλεση συνεχίζεται από το βήμα 3.5 του πετυχημένου σεναρίου.
4.1.α.	Κάποιο όνομα που έδωσε ο χρήστης για προώθηση δεν είναι στάδιο επεξεργασίας.
4.1.α.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
4.1.α.2.	Η λειτουργία του συστήματος τερματίζεται.
4.2.α.	Κάποιο όνομα που έδωσε ο χρήστης για προώθηση δεν είναι εφαρμογή εξόδου.
4.2.α.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
4.2.α.2.	Η λειτουργία του συστήματος τερματίζεται.
4.3.α.	Κάποιο όνομα που έδωσε ο χρήστης για προώθηση δεν είναι στάδιο επεξεργασίας.
4.3.α.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.
4.3.α.2.	Η λειτουργία του συστήματος τερματίζεται.
4.4.α.	Κάποιο όνομα που έδωσε ο χρήστης για προώθηση δεν είναι εφαρμογή εξόδου.
4.4.α.1.	Το σύστημα ενημερώνει το χρήστη για το πρόβλημα.

Βήμα	Ενέργεια
4.4.α.2.	Η λειτουργία του συστήματος τερματίζεται.

2.3.3. Σενάρια δημιουργίας διεπαφών επικοινωνίας με το χρήστη

2.3.3.1. Πετυχημένο σενάριο

Εδώ περιγράφεται το πετυχημένο σενάριο για τη δημιουργία των διεπαφών επικοινωνίας με το χρήστη. Αυτό είναι το σενάριο που το σύστημα θα ακολουθήσει αν δεν προκύψει κανένα πρόβλημα.

Πίνακας 2.10: Πετυχημένο σενάριο δημιουργίας διεπαφών του χρήστη

Βήμα	Ενέργεια
1.	Το σύστημα δημιουργεί το διάγραμμα της προδιαγραφής ΠΔΧ-1.
2.	Το σύστημα προσθέτει στο διάγραμμα ένα στοιχείο για κάθε εφαρμογή εισόδου.
3.	Το σύστημα προσθέτει στο διάγραμμα ένα στοιχείο για κάθε τρόπο επεξεργασίας.
4.	Το σύστημα προσθέτει στο διάγραμμα ένα στοιχείο για κάθε εφαρμογή εξόδου.
5.	Το σύστημα συνδέει το κάθε στοιχείο εφαρμογής εισόδου με τις εξόδους του.
6.	Το σύστημα συνδέει το κάθε στοιχείο τρόπου επεξεργασίας με τις εξόδους του.
7.	Το σύστημα παρουσιάζει το διάγραμμα στο χρήστη.

2.3.3.2. Εναλλακτικά σενάρια

Σε αυτό το στάδιο ανάλυσης του συστήματος δεν είναι δυνατός ο εντοπισμός εναλλακτικών σεναρίων για τη δημιουργία διεπαφών επικοινωνίας με το χρήστη.

2.3.4. Σενάρια έναρξης τμημάτων

2.3.4.1. Πετυχημένο σενάριο

Εδώ περιγράφεται το πετυχημένο σενάριο για την έναρξη των τμημάτων του συστήματος. Αυτό είναι το σενάριο που το σύστημα θα ακολουθήσει αν δεν προκύψει κανένα πρόβλημα.

Πίνακας 2.11: Πετυχημένο σενάριο έναρξης τμημάτων

Βήμα	Ενέργεια
1.	Το σύστημα ξεκινάει την επικοινωνία με τις εφαρμογές εξόδου.
2.	Το σύστημα ξεκινάει τα στάδια επεξεργασίας.
3.	Το σύστημα ξεκινάει την επικοινωνία με τις εφαρμογές εισόδου.

2.3.4.2. Εναλλακτικά σενάρια

Εδώ περιγράφονται τα σενάρια τα οποία θα εξελιχθούν στην περίπτωση που κάποιο βήμα του πετυχημένου σεναρίου δεν καταφέρει να ολοκληρωθεί με επιτυχία.

Πίνακας 2.12: Εναλλακτικά σενάρια έναρξης τμημάτων

Βήμα	Ενέργεια
1.1.	Η έναρξη της επικοινωνίας απέτυχε.
1.α.1.	Το σύστημα ενημερώνει το χρήστη.
1.α.2.	Το σύστημα συνεχίζει την κανονική εκτέλεση και από αυτό το σημείο αγνοεί τη συγκεκριμένη εφαρμογή εξόδου.
2.1.	Η έναρξη του σταδίου επεξεργασίας απέτυχε.
2.α.1.	Το σύστημα ενημερώνει το χρήστη.
2.α.2.	Το σύστημα συνεχίζει την κανονική εκτέλεση και από αυτό το σημείο αγνοεί το συγκεκριμένο στάδιο επεξεργασίας.
3.1.	Η έναρξη της επικοινωνίας απέτυχε.
3.α.1.	Το σύστημα ενημερώνει το χρήστη.
3.α.2.	Το σύστημα συνεχίζει την κανονική εκτέλεση και από αυτό το σημείο αγνοεί τη συγκεκριμένη εφαρμογή εισόδου.

2.3.5. Σενάρια λήψης εισερχόμενου μηνύματος

2.3.5.1. Πετυχημένο σενάριο

Εδώ περιγράφεται το πετυχημένο σενάριο για τη λήψη ενός εισερχόμενου μηνύματος από μια εφαρμογή εισόδου. Αυτό είναι το σενάριο που το σύστημα θα ακολουθήσει αν δεν προκύψει κανένα πρόβλημα.

Πίνακας 2.13: Πετυχημένο σενάριο λήψης εισερχόμενου μηνύματος

Βήμα	Ενέργεια
1.	Η εφαρμογή εισόδου αποστέλλει το εισερχόμενο μήνυμα.
2.	Το σύστημα λαμβάνει το εισερχόμενο μήνυμα σύμφωνα με το πρωτόκολλο επικοινωνίας.
3.	Το σύστημα προωθεί το μήνυμα στις λίστες αναμονής για επεξεργασία που έχει ορίσει ο χρήστης κατά την αρχικοποίηση.
4.	Το σύστημα προωθεί το μήνυμα στις λίστες αναμονής για αποστολή σε εφαρμογές εξόδου που έχει ορίσει ο χρήστης κατά την αρχικοποίηση.

Βήμα	Ενέργεια
5.	Το σύστημα ενημερώνει τις διεπαφές του χρήστη με την τιμή του εισερχόμενου μηνύματος.
6.	Το σύστημα ξεκινάει την αναμονή του επόμενου μηνύματος από την εφαρμογή εισόδου.

2.3.5.2. Εναλλακτικά σενάρια

Εδώ περιγράφονται τα σενάρια τα οποία θα εξελιχθούν στην περίπτωση που κάποιο βήμα του πετυχημένου σεναρίου δεν καταφέρει να ολοκληρωθεί με επιτυχία.

Πίνακας 2.14: Εναλλακτικά σενάρια λήψης εισερχόμενου μηνύματος

Βήμα	Ενέργεια
2.α.	Λόγω κάποιου προβλήματος το μήνυμα έφτασε λάθος.
2.α.1.	Το σύστημα ενημερώνει το χρήστη.
2.α.1.	Το σύστημα αγνοεί το μήνυμα.
2.α.2.	Η εκτέλεση συνεχίζεται από το βήμα 6 του πετυχημένου σεναρίου.
3.α.	Η λίστα αναμονής για επεξεργασία είναι ήδη γεμάτη.
3.α.1.	Το σύστημα αγνοεί το μήνυμα.
3.α.2.	Η εκτέλεση συνεχίζεται κανονικά.
4.α.	Η λίστα αναμονής για αποστολή είναι ήδη γεμάτη.
4.α.1.	Το σύστημα αγνοεί το μήνυμα.
4.α.2.	Η εκτέλεση συνεχίζεται κανονικά.
5.α.	Ο χρήστης έχει επιλέξει την εκτέλεση χωρίς διεπαφές επικοινωνίας.
5.α.1.	Η εκτέλεση συνεχίζεται από το βήμα6 του πετυχημένου σεναρίου.

2.3.6. Σενάρια επεξεργασίας μηνύματος

2.3.6.1. Πετυχημένο σενάριο

Εδώ περιγράφεται το πετυχημένο σενάριο για την επεξεργασία ενός μηνύματος. Αυτό είναι το σενάριο που το σύστημα θα ακολουθήσει αν δεν προκύψει κανένα πρόβλημα.

Πίνακας 2.15: Πετυχημένο σενάριο επεξεργασίας μηνύματος

Βήμα	Ενέργεια
1.	Το σύστημα παίρνει ένα μήνυμα από τη λίστα αναμονής για επεξεργασία.
2.	Το σύστημα επεξεργάζεται το μήνυμα σύμφωνα με τη μέθοδο που έχει ορίσει ο χρήστης.

Βήμα	Ενέργεια
3.	Το σύστημα προωθεί το αποτέλεσμα της επεξεργασίας στις λίστες αναμονής για επεξεργασία που έχει ορίσει ο χρήστης κατά την αρχικοποίηση.
4.	Το σύστημα προωθεί το αποτέλεσμα της επεξεργασίας στις λίστες αναμονής για αποστολή σε εφαρμογές εξόδου που έχει ορίσει ο χρήστης κατά την αρχικοποίηση.
5.	Το σύστημα ενημερώνει τις διεπαφές του χρήστη με την τιμή του αποτελέσματος της επεξεργασίας.
6.	Η εκτέλεση συνεχίζεται από το βήμα 1.

2.3.6.2. Εναλλακτικά σενάρια

Εδώ περιγράφονται τα σενάρια τα οποία θα εξελιχθούν στην περίπτωση που κάποιο βήμα του πετυχημένου σεναρίου δεν καταφέρει να ολοκληρωθεί με επιτυχία.

Πίνακας 2.16: Εναλλακτικά σενάρια επεξεργασίας μηνύματος

Βήμα	Ενέργεια
1.α.	Η λίστα αναμονής δεν περιέχει κανένα μήνυμα.
1.α.1.	Το σύστημα περιμένει μέχρι κάποιο μήνυμα να εισαχθεί στη λίστα.
1.α.2.	Το σύστημα παίρνει το μήνυμα από τη λίστα.
1.α.3.	Η εκτέλεση συνεχίζεται από το βήμα 2 του πετυχημένου σεναρίου.
2.α.	Η επεξεργασία του μηνύματος αποτυχαίνει.
2.α.1.	Το σύστημα ενημερώνει το χρήστη για το λάθος.
2.α.2.	Το σύστημα αγνοεί το μήνυμα.
2.α.3.	Η εκτέλεση συνεχίζεται από το βήμα 1 του πετυχημένου σεναρίου.
3.α.	Η λίστα αναμονής για επεξεργασία είναι ήδη γεμάτη.
3.α.1.	Το σύστημα αγνοεί το μήνυμα.
3.α.2.	Η εκτέλεση συνεχίζεται κανονικά.
4.α.	Η λίστα αναμονής για αποστολή είναι ήδη γεμάτη.
4.α.1.	Το σύστημα αγνοεί το μήνυμα.
4.α.2.	Η εκτέλεση συνεχίζεται κανονικά.
5.α.	Ο χρήστης έχει επιλέξει την εκτέλεση χωρίς διεπαφές επικοινωνίας.
5.α.1.	Η εκτέλεση συνεχίζεται από το βήμα6 του πετυχημένου σεναρίου.

2.3.7. Σενάρια αποστολής εξερχόμενου μηνύματος

2.3.7.1. Πετυχημένο σενάριο

Εδώ περιγράφεται το πετυχημένο σενάριο για την αποστολή ενός εξερχόμενου μηνύματος σε μια εφαρμογή εξόδου. Αυτό είναι το σενάριο που το σύστημα θα ακολουθήσει αν δεν προκύψει κανένα πρόβλημα.

Πίνακας 2.17: Πετυχημένο σενάριο αποστολής εξερχόμενου μηνύματος

Βήμα	Ενέργεια
1.	Το σύστημα παίρνει ένα μήνυμα από τη λίστα αναμονής για αποστολή στην εφαρμογή εξόδου.
2.	Το σύστημα στέλνει το μήνυμα στην εφαρμογή εξόδου σύμφωνα με το πρωτόκολλο επικοινωνίας που έχει ορίσει ο χρήστης κατά την αρχικοποίηση.
3.	Η εκτέλεση συνεχίζεται από το βήμα 1.

2.3.7.2. Εναλλακτικά σενάρια

Εδώ περιγράφονται τα σενάρια τα οποία θα εξελιχθούν στην περίπτωση που κάποιο βήμα του πετυχημένου σεναρίου δεν καταφέρει να ολοκληρωθεί με επιτυχία.

Πίνακας 2.18: Εναλλακτικά σενάρια αποστολής εξερχόμενου μηνύματος

Βήμα	Ενέργεια
1.α.	Η λίστα αναμονής δεν περιέχει κανένα μήνυμα.
1.α.1.	Το σύστημα περιμένει μέχρι κάποιο μήνυμα να εισαχθεί στη λίστα.
1.α.2.	Το σύστημα παίρνει το μήνυμα από τη λίστα.
1.α.3.	Η εκτέλεση συνεχίζεται από το βήμα 2 του πετυχημένου σεναρίου.
2.α.	Η αποστολή του μηνύματος αποτυχαίνει.
2.α.1.	Το σύστημα ενημερώνει το χρήστη για το λάθος.
2.α.2.	Το σύστημα αγνοεί το μήνυμα.
2.α.3.	Η εκτέλεση συνεχίζεται από το βήμα 1 του πετυχημένου σεναρίου.

2.3.8. Σενάρια επιλογής στοιχείου του διαγράμματος

2.3.8.1. Πετυχημένο σενάριο

Εδώ περιγράφεται το πετυχημένο σενάριο για την επιλογή ενός στοιχείου του διαγράμματος των διεπαφών του χρήστη. Αυτό είναι το σενάριο που το σύστημα θα ακολουθήσει αν δεν προκύψει

κανένα πρόβλημα.

Πίνακας 2.19: Πετυχημένο σενάριο επιλογής του διαγράμματος

Βήμα	Ενέργεια
1.	Ο χρήστης επιλέγει ένα στοιχείο στο διάγραμμα.
2.	Το σύστημα παρουσιάζει στο χρήστη την περιγραφή του στοιχείου.
3.	Το σύστημα παρουσιάζει στο χρήστη τα στοιχεία που αποτελούν την είσοδο του στοιχείου.
4.	Το σύστημα παρουσιάζει στο χρήστη τα στοιχεία που αποτελούν την έξοδο του στοιχείου.

2.3.8.2. Εναλλακτικά σενάρια

Στην παρούσα φάση ανάλυσης του συστήματος δεν υπάρχουν εναλλακτικά σενάρια για την επιλογή ενός στοιχείου του διαγράμματος των διεπαφών του χρήστη.

2.3.9. Σενάρια τερματισμού του συστήματος

2.3.9.1. Πετυχημένο σενάριο

Εδώ περιγράφεται το πετυχημένο σενάριο για τον τερματισμό του συστήματος από τον χρήστη. Αυτό είναι το σενάριο που το σύστημα θα ακολουθήσει αν δεν προκύψει κανένα πρόβλημα.

Πίνακας 2.20: Πετυχημένο σενάριο τερματισμού του συστήματος

Βήμα	Ενέργεια
1.	Ο χρήστης επιλέγει να τερματίσει το σύστημα.
2.	Το σύστημα τερματίζει τη λειτουργία του.

2.3.9.2. Εναλλακτικά σενάρια

Στην παρούσα φάση ανάλυσης του συστήματος δεν υπάρχουν εναλλακτικά σενάρια για τον τερματισμό του συστήματος από τον χρήστη.

2.4. Πλατφόρμες και προγραμματιστικά εργαλεία

Σε αυτήν την ενότητα παρουσιάζονται οι πλατφόρμες και τα προγραμματιστικά εργαλεία τα οποία χρησιμοποιήθηκαν για την υλοποίηση του συστήματος. Το κάθε ένα από αυτά συνοδεύεται από μια σύντομη επεξήγηση του λόγου επιλογής του.

2.4.1. Προγραμματιστική πλατφόρμα Java 1.6

Η προγραμματιστική πλατφόρμα που επιλέχθηκε για την υλοποίηση του συστήματος είναι η Java 1.6. Η επιλογή αυτή έγινε με βάση τα ακόλουθα κριτήρια:

- Η Java είναι μια γλώσσα προγραμματισμού η οποία είναι εστιασμένη στον αντικειμενοστραφή προγραμματισμό
- Η Java αποκρύπτει από τον προγραμματιστή το μεγαλύτερο μέρος της επικοινωνίας με το υλικό του υπολογιστή, επιτρέποντάς του να εστιαστεί στη σχεδίαση του συστήματος που θέλει να αναπτύξει
- Η Java παρέχει πολύ καλές και αποδοτικές βιβλιοθήκες στον τομέα της μεταφοράς μηνυμάτων μέσω δικτύων
- Η Java (από την έκδοση 1.5) παρέχει ένα ολοκληρωμένο, αρκετά αποδοτικό και εύκολο στη χρήση σύστημα διαχείρισης συλλογών αντικειμένων
- Η Java παρέχει βιβλιοθήκες για το χειρισμό XML αρχείων
- Η Java διατίθεται δωρεάν
- Ο πηγαίος κώδικας της γλώσσας είναι διαθέσιμος
- Υπάρχει πλούσια βιβλιογραφία σχετική με την Java
- Η Java είναι μια γλώσσα ανεξάρτητη από την πλατφόρμα υλοποίησης, με αποτέλεσμα το σύστημα να μπορεί να εκτελεστεί σε διαφορετικά λειτουργικά συστήματα

Τα παραπάνω κριτήρια οδήγησαν στην επιλογή της Java ως την καλύτερη επιλογή για την υλοποίηση του συστήματος.

2.4.2. Netbeans 6

Ως προγραμματιστικό εργαλείο για την ανάπτυξη του συστήματος θα χρησιμοποιηθεί το Netbeans 6. Το εργαλείο αυτό επιτρέπει τη δημιουργία διαγραμμάτων UML και παρέχει εργαλεία για την ευκολότερη δημιουργία Java κώδικα, καθώς επίσης και εργαλεία για τον έλεγχο και τον εντοπισμό λαθών στον κώδικα. Η επιλογή του Netbeans και όχι κάποιου παρόμοιου εργαλείου έγινε καθαρά από λόγους προτίμησης του συγγραφέα.

3. Σχεδίαση

Στο κεφάλαιο αυτό περιγράφεται η σχεδίαση του συστήματος. Η πρώτη ενότητα είναι εισαγωγική και περιγράφει κάποιες από τις ιδιότητες της Java, η γνώση των οποίων είναι απαραίτητη για την κατανόηση της σχεδίασης του συστήματος. Στις υπόλοιπες ενότητες του κεφαλαίου γίνεται ο εντοπισμός των κλάσεων του συστήματος, με στόχο την ικανοποίηση των λειτουργικών προδιαγραφών, των περιπτώσεων χρήσεως και των σεναρίων που εντοπίστηκαν στο κεφάλαιο της ανάλυσης.

3.1. Ιδιότητες της πλατφόρμας Java

Σε αυτήν την ενότητα περιγράφονται οι ιδιότητες της πλατφόρμας Java οι οποίες θα χρησιμοποιηθούν για την υλοποίηση του συστήματος, δηλαδή οι ιδιότητες τη γνώση των οποίων απαιτείται για την κατανόηση της σχεδίασης του συστήματος. Οι ιδιότητες αυτές δεν θα αναλυθούν σε μεγάλο βαθμό, παρά μόνο θα καλυφθούν όσο χρειάζεται για την κατανόηση του παρόντος κειμένου. Ο αναγνώστης που θέλει να τις κατανοήσει καλύτερα ή να διαβάσει περισσότερες πληροφορίες για αυτές μπορεί να απευθυνθεί στη βιβλιογραφία.

3.1.1. Αντικείμενα (objects)

Ο τρόπος με τον οποίο λειτουργούσαν οι πρώτες γλώσσες προγραμματισμού ήταν καθαρά

σειριακός. Το κάθε πρόγραμμα ήταν απλά μια σειρά εντολών τις οποίες εκτελούσε ο υπολογιστής για τον υπολογισμό του επιθυμητού αποτελέσματος. Η κατασκευή προγραμμάτων με τη χρήστη τέτοιων γλωσσών είναι αρκετά γρήγορη και απλή, οπότε για τη δημιουργία απλών και μικρών προγραμμάτων η χρήση τους είναι ακόμα επιθυμητή (αυτό εξηγεί και την ύπαρξη γλωσσών κωδικοποίησης σεναρίου – scripting languages). Στη σημερινή εποχή όμως, η πολυπλοκότητα και το μέγεθος των περισσότερων προγραμμάτων έχει πολλαπλασιαστεί, και η χρήστη μιας τέτοιας γλώσσας προγραμματισμού θα καθιστούσε την υλοποίησή τους υπερβολικά χρονοβόρα (αν όχι αδύνατη) και το αποτέλεσμα θα ήταν ένα χαοτικό σύνολο χιλιάδων εντολών, στο οποίο η εύρεση οποιουδήποτε λάθους θα ήταν πρακτικά αδύνατη.

Το πρόβλημα αυτό βρήκε λύση στις αντικειμενοστραφείς γλώσσες προγραμματισμού (object oriented languages). Ο τρόπος σκέψης κατά τον προγραμματισμό με τέτοιες γλώσσες είναι τελείως διαφορετικός απ' ότι στις παραδοσιακές γλώσσες. Το κάθε πρόγραμμα δεν αποτελείται πια από μια λίστα εντολών, παρά αποτελείται από ένα σύνολο αντικειμένων (objects), τα οποία αλληλεπιδρούν μεταξύ τους στέλνοντας μηνύματα το ένα στο άλλο. Με αυτόν τον τρόπο, οι σύγχρονες γλώσσες προγραμματισμού διαιρούν το κάθε πρόγραμμα σε μικρότερα κομμάτια (τα αντικείμενα), των οποίων η ανάλυση, σχεδίαση και υλοποίηση είναι αρκετά ευκολότερη.

Η Java είναι μια καθαρά αντικειμενοστραφής γλώσσα προγραμματισμού. Κατά τη δική της ορολογία, τα αντικείμενα που αναφέρθηκαν ανωτέρω ονομάζονται κλάσεις (classes). Οι κλάσεις αυτές αποτελούν τις οντότητες του συστήματος και θα προσδιοριστούν σε αυτό το κεφάλαιο.

3.1.2. Κληρονομικότητα (inheritance)

Η κάθε Java κλάση έχει κάποιες ιδιότητες οι οποίες την χαρακτηρίζουν. Για παράδειγμα, μια κλάση η οποία αντιπροσωπεύει ένα σχήμα μπορεί να έχει ένα εμβαδόν. Στην περίπτωση που ο χρήστης θέλει να δημιουργήσει μια πιο εξειδικευμένη κλάση, για παράδειγμα έναν κύκλο, η Java επιτρέπει στη νέα κλάση να κληρονομήσει τις ιδιότητες του σχήματος και απλά να προσθέσει νέες ιδιότητες, όπως για παράδειγμα την ακτίνα του κύκλου. Δηλαδή ο προγραμματιστής μπορεί να δημιουργήσει μια κλάση γενικότερης έννοιας, όπως το σχήμα, και μετά να δημιουργήσει εξειδικεύσεις αυτής της κλάσης, όπως τον κύκλο, το τετράγωνο, το τρίγωνο κτλ.

Η ιδιότητα αυτή των αντικειμενοστραφών γλωσσών δίνει πολύ μεγαλύτερη δύναμη στον προγραμματιστή από την προφανή, δηλαδή ότι γλυτώνει χρόνο μην έχοντας να γράψει τον κώδικα για τις βασικές ιδιότητες της γενικότερης κλάσης για κάθε εξειδικευμένη κλάση. Η πιο σημαντική ιδιότητα είναι ότι οι εξειδικευμένες κλάσεις συνεχίζουν να έχουν ακόμα την έννοια της γενικής κλάσης. Αυτό σημαίνει ότι ο κύκλος και το τετράγωνο συνεχίζουν να είναι και τα δύο σχήματα, αν και είναι διαφορετικά μεταξύ τους. Ο προγραμματιστής λοιπόν μπορεί να γράψει συναρτήσεις οι

οποίες θα χρησιμοποιούν την κλάση που αντιπροσωπεύει την γενικότερη έννοια του σχήματος, οι οποίες θα μπορούν να χειρίζονται όλες τις εξειδικευμένες κλάσεις. Αυτή η ιδιότητα είναι μια ιδιότητα των αντικειμενοστραφών γλωσσών η οποία θα χρησιμοποιηθεί εκτεταμένα για τη σχεδίαση του παρών συστήματος.

3.1.3. Νήματα εκτέλεσης (threads)

Για τη δυνατότητα υλοποίησης των πολύπλοκων σύγχρονων προγραμμάτων, τα διάφορα λειτουργικά συστήματα και οι γλώσσες προγραμματισμού παρέχουν στον προγραμματιστή τη δυνατότητα παράλληλου προγραμματισμού. Σε αυτήν την περίπτωση, δύο ή περισσότερα κομμάτια του προγράμματος τρέχουν παράλληλα, σαν να ήταν δύο διαφορετικά προγράμματα, και ανάλογα με τη γλώσσα προγραμματισμού που χρησιμοποιείται, υπάρχουν τεχνικές επικοινωνίας μεταξύ τους.

Η υποστήριξη του παράλληλου προγραμματισμού στη Java γίνεται με τη χρήση νημάτων εκτέλεσης (threads) και έχει υλοποιηθεί με λίγο διαφορετικό τρόπο από άλλες γλώσσες προγραμματισμού. Σε αντίθεση με άλλες γλώσσες, η Java δεν βασίζεται στην υποστήριξη του λειτουργικού συστήματος για την υλοποίηση παράλληλου προγραμματισμού. Αντιθέτως, η ίδια η γλώσσα παρέχει την απαραίτητη υποδομή για την υλοποίησή του. Η μέθοδος αυτή έχει και πλεονεκτήματα αλλά και μειονεκτήματα, η αναφορά των οποίων δεν είναι σκόπιμη στο παρόν κείμενο.

3.1.4. Διαχείριση σφαλμάτων (Exceptions handling)

Κάθε φορά που εκτελείται μια εντολή ενός προγράμματος υπάρχει η δυνατότητα να μην τερματίσει σωστά. Τις περισσότερες φορές τέτοιου είδους σφάλματα προέρχονται από τις εντολές που αλληλεπιδρούν με το υλικό του υπολογιστή, για παράδειγμα διάβασμα από ένα αρχείο που δεν υπάρχει, αλλά μπορεί να προέρχονται και από το λογισμικό, για παράδειγμα η λανθασμένη μετατροπή ενός αλφαριθμητικού σε αριθμό. Το κοινό χαρακτηριστικό που έχουν όλες αυτές οι περιπτώσεις είναι ότι η κανονική ροή του προγράμματος έχει διακοπεί και το πρόγραμμα είτε πρέπει να ανακάμψει από το σφάλμα είτε να τερματίσει τη λειτουργία του.

Η διαχείριση των σφαλμάτων κατά την εκτέλεση ενός προγράμματος είναι μια χρονοβόρα και δύσκολη εργασία. Γνωρίζοντας αυτό, οι δημιουργοί της Java συμπεριέλαβαν στη γλώσσα ένα πολύ καλό σύστημα διαχείρισης σφαλμάτων. Ο προγραμματιστής μπορεί να ορίσει τις ενέργειες για κάθε πιθανό σφάλμα για κάθε κομμάτι κώδικα είτε ώστε το πρόγραμμα να ανακάμψει είτε για να τερματίσει. Επίσης το σύστημα σφαλμάτων είναι επεκτάσιμο, ώστε ο προγραμματιστής να μπορεί να ορίσει τους δικούς του τύπους σφαλμάτων.

3.1.5. Προγραμματισμός οδηγούμενος από συμβάντα (events driven programming)

Ο παραδοσιακός τρόπος επικοινωνίας των διάφορων κομματιών ενός προγράμματος είναι με τη κλήση μεθόδων. Σε αυτήν την μέθοδο, κατά την εκτέλεση ενός κομματιού κώδικα, καλείται μια συγκεκριμένη μέθοδος ενός άλλου κομματιού. Σε αντίθεση με αυτόν τον τρόπο προγραμματισμού, στη Java είναι επίσης δυνατός ο προγραμματισμός οδηγούμενος από συμβάντα. Σε αυτήν την περίπτωση κάποιο κομμάτι κώδικα μπορεί να δημιουργήσει κάποιο συμβάν (event), για παράδειγμα όταν λάβει κάποια δεδομένα. Το συμβάν αυτό μπορεί να περιέχει οτιδήποτε είδους πληροφορία. Κάποια άλλα κομμάτια κώδικα δηλώνουν ότι θέλουν να ενημερώνονται για τα συμβάντα του συγκεκριμένου τύπου και, κάθε φορά που δημιουργείται ένα τέτοιο συμβάν εκτελούν κάποιες συγκεκριμένες εντολές.

Η ιδιότητα αυτή της Java θα χρησιμοποιηθεί στο παρών σύστημα για την επικοινωνία μεταξύ των διάφορων μονάδων του συστήματος.

3.2. Κλάσεις σχετικές με δεδομένα

Σε αυτήν την ενότητα εντοπίζονται και οι κλάσεις και οι σχέσεις μεταξύ τους, οι οποίες είναι σχετικές με τα δεδομένα τα οποία μετακινούνται στο σύστημα.

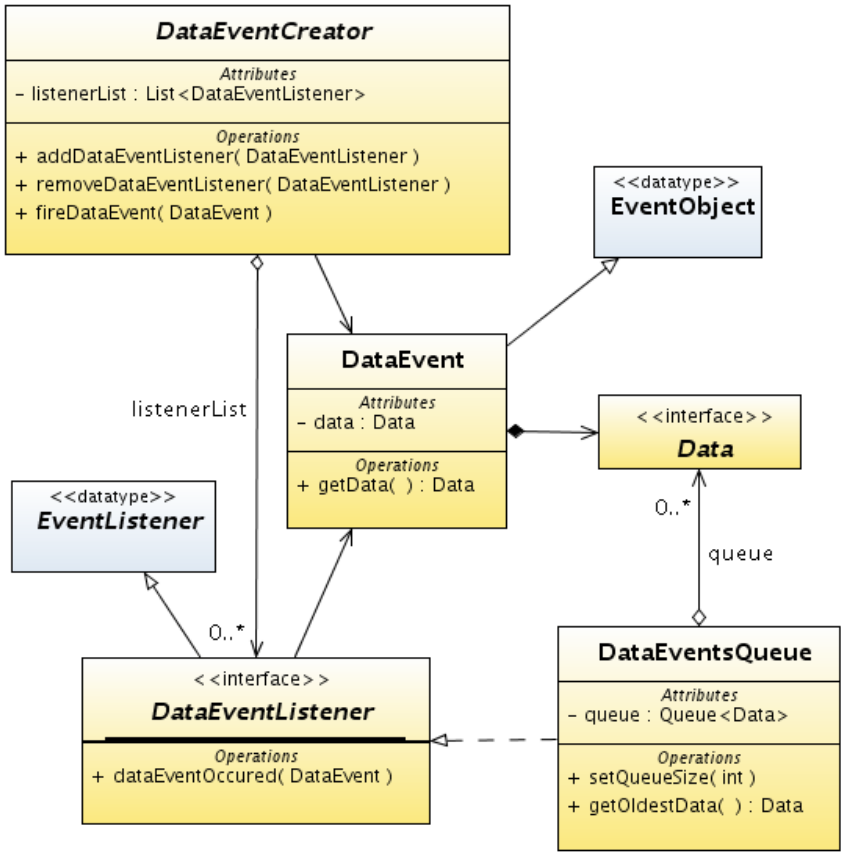
Σύμφωνα με της προδιαγραφές ΠΓΧ-1 και ΠΓΧ-3, το σύστημα πρέπει να έχει τη δυνατότητα να επικοινωνεί με ένα μεγάλο πλήθος διαφορετικών εφαρμογών. Το γεγονός αυτό έχει ως αποτέλεσμα ότι το σύστημα πρέπει να χειρίζεται ένα μεγάλο πλήθος διαφορετικών δεδομένων. Για την κατασκευή του συστήματος όμως απαιτείται να υπάρχει μία μόνο κλάση η οποία θα αντιπροσωπεύει τα δεδομένα. Για αυτόν τον λόγο θα χρησιμοποιηθεί η ιδιότητα της κληρονομικότητας, οπότε θα σχεδιαστεί μόνο μία διεπαφή (interface) η οποία θα αντιπροσωπεύει όλους τους τύπους δεδομένων. Ο χρήστης του συστήματος πρέπει να υλοποιεί κλάσεις οι οποίες θα κληρονομούν τη διεπαφή και θα παρέχουν τις μεθόδους για την αποθήκευση των δεδομένων που θέλει.

Η μεταφορά των δεδομένων μεταξύ των διάφορων στοιχείων του συστήματος (προδιαγραφές ΠΕΠΜ-5, ΠΕΠΜ-7 και ΠΕΞΜ-8) θα γίνεται με τη μέθοδο προγραμματισμού οδηγούμενου από συμβάντα. Για να υλοποιηθεί αυτό πρέπει να σχεδιαστούν τρεις κλάσεις. Η πρώτη είναι μια κλάση η οποία αντιπροσωπεύει το ίδιο το συμβάν. Η κλάση αυτή πρέπει να μεταφέρει τα δεδομένα και να παρέχει τη δυνατότητα ανάκτησης τους. Η δεύτερη κλάση είναι μια διεπαφή την οποία πρέπει να υλοποιούν όλες οι κλάσεις οι οποίες θέλουν να ενημερώνονται για τα συμβάντα. Η διεπαφή αυτή απαιτεί την υλοποίηση μιας μεθόδου, η οποία θα εκτελείται κάθε φορά που λαμβάνεται ένα συμβάν. Η τρίτη κλάση θα κληρονομείται από τις κλάσεις οι οποίες θέλουν να δημιουργούν

συμβάντα δεδομένων. Η κλάση αυτή πρέπει να διατηρεί μία λίστα με τις κλάσεις που πρέπει να ενημερωθούν για τα συμβάντα, μεθόδους για την προσθήκη και διαγραφή κλάσεων από τη λίστα αυτή, καθώς και μια μέθοδο η οποία θα καλείται κάθε φορά που δημιουργείται ένα συμβάν και θα ενημερώνει τις κλάσεις στη λίστα.

Τέλος, για την ικανοποίηση των προδιαγραφών ΠΕΠΜ-3 και ΠΕΞΜ-6, είναι απαραίτητη και η σχεδίαση μίας κλάσης η οποία θα υλοποιεί τη διεπαφή για να ενημερώνεται για συμβάντα που αναφέρθηκε προηγουμένως, η οποία επίσης θα υλοποιεί και μία λίστα αναμονής. Η κλάση αυτή θα διατηρεί μια ουρά (queue) και θα υλοποιεί τη μέθοδο της διεπαφής με τέτοιο τρόπο ώστε απλά να προσθέτει τα δεδομένα που μεταφέρονται στα συμβάντα στην ουρά. Για την εξαγωγή των δεδομένων από την ουρά θα παρέχει μια μέθοδο η οποία θα επιστρέφει το παλιότερο στοιχείο της λίστας. Επίσης θα παρέχει μια μέθοδο για τον ορισμό του μέγιστου αριθμού δεδομένων τα οποία θα μπορεί να αποθηκεύσει η ουρά.

Από τα ανωτέρω προκύπτει το σχήμα 3.1, το οποίο είναι το διάγραμμα κλάσεων σχετικών με δεδομένα.



Σχήμα 3.1: Διάγραμμα κλάσεων σχετικών με δεδομένα

Η διεπαφή (interface) Data είναι η διεπαφή που αντιπροσωπεύει τα δεδομένα. Η κλάση DataEvent αντιπροσωπεύει τα συμβάντα δεδομένων και υλοποιεί το EventObject της βιβλιοθήκης java.util της Java. Η διεπαφή DataEventListener είναι η διεπαφή που αντιπροσωπεύει τις κλάσεις οι οποίες ενημερώνονται για τα συμβάντα δεδομένων και υλοποιεί το EventListener της βιβλιοθήκης java.util της Java. Η κλάση DataEventCreator αντιπροσωπεύει τις κλάσεις οι οποίες δημιουργούν τα συμβάντα δεδομένων. Η κλάση DataEventsQueue είναι η υλοποίηση της διεπαφής DataEventListener η οποία υλοποιεί την ουρά αναμονής.

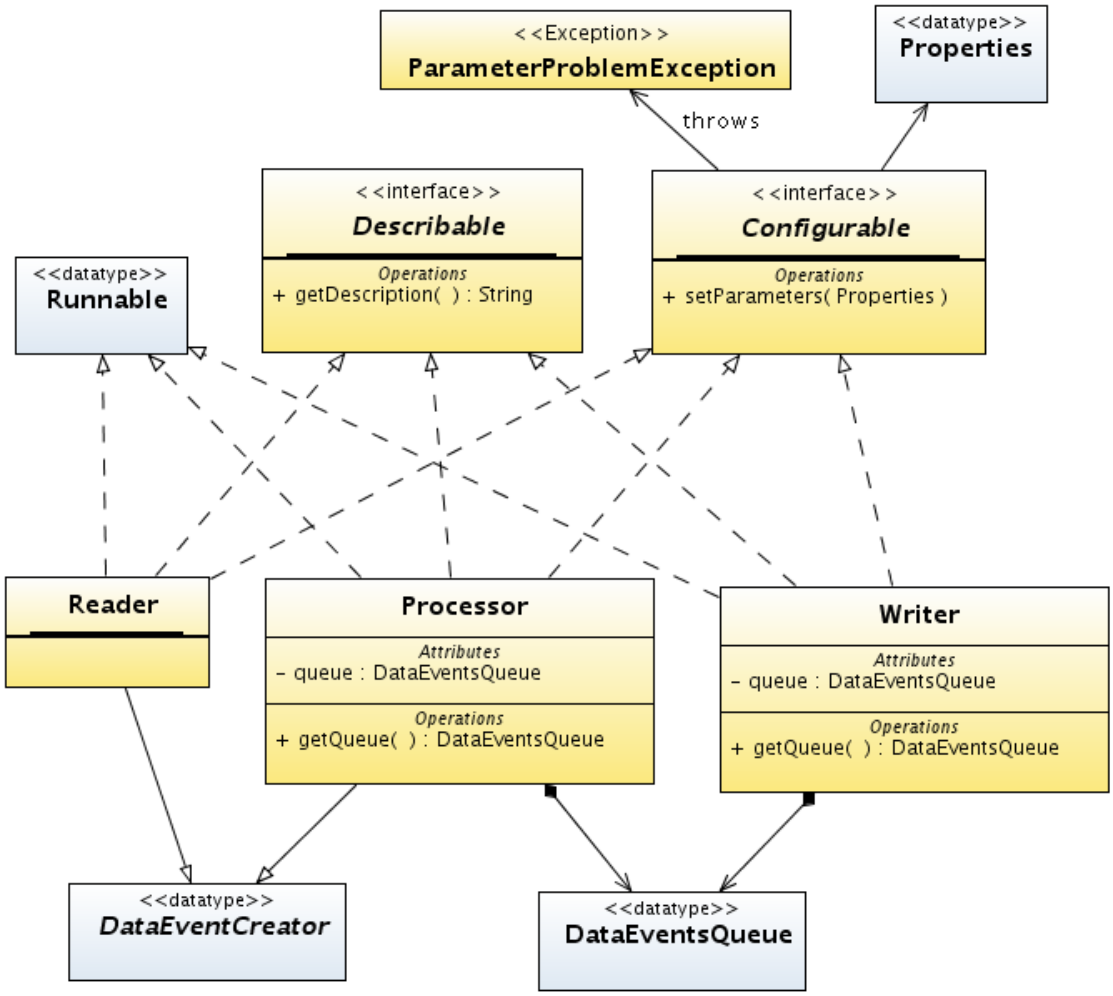
3.3. Κλάσεις σχετικές με τα στοιχεία του συστήματος

Όπως προκύπτει από τις προδιαγραφές ΠΓΧ-1, ΠΓΧ-2 και ΠΓΧ-3, το σύστημα πρέπει να αποτελείται από στοιχεία τριών ειδών. Τα στοιχεία τα οποία δέχονται τα μηνύματα από τις εφαρμογές εισόδου, τα στοιχεία τα οποία επεξεργάζονται τα μηνύματα και τα στοιχεία τα οποία στέλνουν τα μηνύματα στις εφαρμογές εξόδου. Από αυτά, τα στοιχεία εισόδου και τα στοιχεία επεξεργασίας πρέπει να υλοποιούν τη διεπαφή DataEventCreator, για να μπορούν να προωθούν μηνύματα σε άλλα στοιχεία του συστήματος. Παρομοίως, τα στοιχεία επεξεργασίας και τα στοιχεία εξόδου πρέπει να περιέχουν ένα αντικείμενο τύπου DataEventsQueue, για να μπορούν να δέχονται μηνύματα. Επίσης, για να είναι δυνατή η εκτέλεση του κάθε στοιχείου σε διαφορετικό νήμα εκτέλεσης (thread) όλα τα στοιχεία πρέπει να υλοποιούν τη διεπαφή Runnable.

Σύμφωνα με την προδιαγραφή ΠΓΧ-4, όλα τα στοιχεία πρέπει να έχουν μια περιγραφή της λειτουργίας που εκτελούν. Γι' αυτόν τον λόγο θα σχεδιαστεί μια διεπαφή την οποία θα υλοποιούν όλα τα στοιχεία, η οποία θα επιβάλει την υλοποίηση μιας μεθόδου η οποία θα επιστρέφει την περιγραφή του στοιχείου. Με αυτόν τον τρόπο η περιγραφή μπορεί να δημιουργείται δυναμικά και να κάνει χρήση των παραμέτρων που έχει δώσει ο χρήστης.

Τέλος, σύμφωνα με τις προδιαγραφές ΠΕΙΜ-3, ΠΕΠΜ-2 και ΠΕΞΜ-3, πρέπει να είναι δυνατή η παραμετροποίηση των στοιχείων του συστήματος με έναν ενιαίο τρόπο. Γι' αυτόν τον λόγο θα σχεδιαστεί ακόμα μια διεπαφή την οποία όλα τα στοιχεία θα υλοποιούν, η οποία θα τα υποχρεώνει να υλοποιούν μια μέθοδο η οποία θα δέχεται ένα αντικείμενο τύπου Properties (από την java.util βιβλιοθήκη) το οποίο θα περιέχει τις παραμέτρους, και με βάση αυτό θα πραγματοποιεί την παραμετροποίηση. Το σύστημα θα καλεί αυτήν την συνάρτηση πριν ξεκινήσει το νήμα εκτέλεσης του στοιχείου. Οι παράμετροι θα υπάρχουν στο Properties αντικείμενο σε ζευγάρια ονομάτων – τιμών, όπου τα ονόματα και οι τιμές θα είναι αλφαριθμητικά. Στην περίπτωση όπου κάποια παράμετρος λείπει ή δεν έχει το σωστό format, η μέθοδος θα δημιουργεί ένα σφάλμα, το οποίο θα περιγράφει το πρόβλημα.

Από τα ανωτέρω προκύπτει το σχήμα 3.2, το οποίο είναι το διάγραμμα κλάσεων σχετικών με τα στοιχεία του συστήματος.



Σχήμα 3.2: Διάγραμμα κλάσεων σχετικών με τα στοιχεία του συστήματος

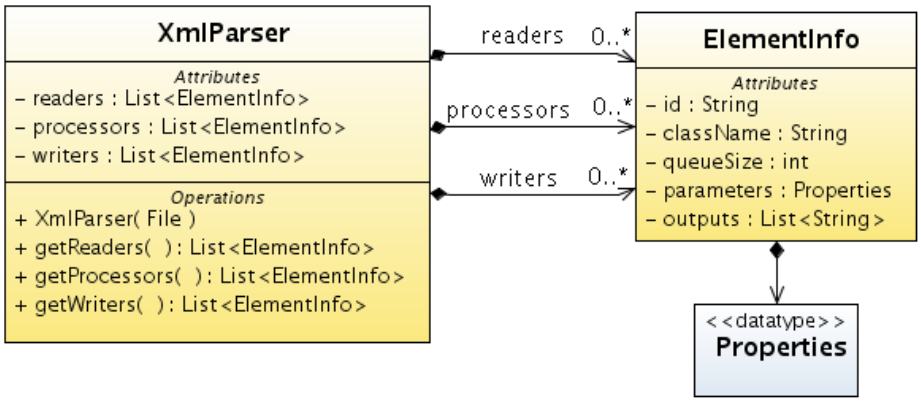
Οι κλάσεις Reader, Processor και Writer είναι οι κλάσεις για επικοινωνία με τις εφαρμογές εισόδου, για επεξεργασία των μηνυμάτων και για επικοινωνία με τις εφαρμογές εξόδου αντίστοιχα. Οι κλάσεις Reader και Processor κληρονομούν την κλάση DataEventCreator για να μπορούν να προωθούν δεδομένα σε άλλα στοιχεία. Οι κλάσεις Processor και Writer περιέχουν ένα αντικείμενο τύπου DataEventsQueue για να μπορούν να δέχονται μηνύματα από άλλα στοιχεία. Και οι τρεις κλάσεις υλοποιούν τη διεπαφή Runnable από τη βιβλιοθήκη java.util ώστε να μπορούν να εκτελεστούν σε διαφορετικά νήματα εργασίας. Η διεπαφή Describable επιβάλει την υλοποίηση της μεθόδου getDescription() η οποία επιστρέφει την περιγραφή του στοιχείου. Τέλος, η διεπαφή Configurable επιτρέπει την παραμετροποίηση των στοιχείων με τη μέθοδο setParameters().

3.4. Κλάσεις σχετικές με την αρχικοποίηση

Για την είσοδο της πληροφορίας που χρειάζεται το σύστημα για την αρχικοποίηση των στοιχείων του, θα χρησιμοποιηθεί ένα αρχείο XML. Λόγω της ομοιότητας των πληροφοριών για την αρχικοποίηση του στοιχείων εισόδου, επεξεργασίας και εξόδου, θα σχεδιαστεί μόνο μία κλάση για την αποθήκευση της πληροφορίας αυτής, η οποία θα χρησιμοποιείται σε όλες τις περιπτώσεις. Η κλάση αυτή θα περιέχει το όνομα του στοιχείου (προδιαγραφή ΠΔΧ-2), την όνομα της κλάσης η οποία κληρονομεί ένα από τα Reader, Processor ή Writer και υλοποιεί το στοιχείο, το μέγεθος της ουράς αναμονής (προδιαγραφές ΠΕΠΜ-4 και ΠΕΞΜ-7), τις παραμέτρους για την παραμετροποίηση του στοιχείου (προδιαγραφές ΠΕΙΜ-5, ΠΕΠΜ-2 και ΠΕΞΜ-3) και μία λίστα με τα ονόματα των στοιχείων στα οποία θα αποσταλούν τα μηνύματα από αυτό το στοιχείο.

Το διάβασμα του XML αρχείου θα γίνεται από μια ξεχωριστή κλάση. Η κλάση αυτή θα δέχεται το όνομα του XML αρχείου κατά τη δημιουργία της, θα το ανοίγει και θα δημιουργεί τρεις λίστες, οι οποίες θα αντιπροσωπεύουν τα στοιχεία του συστήματος. Το σύστημα θα δημιουργεί μια τέτοια κλάση και μετά θα τη χρησιμοποιεί ως καθοδήγηση για τη δημιουργία των στοιχείων.

Από τα ανωτέρω προκύπτει το σχήμα 3.3, το οποίο είναι το διάγραμμα κλάσεων σχετικών με την αρχικοποίηση του συστήματος.



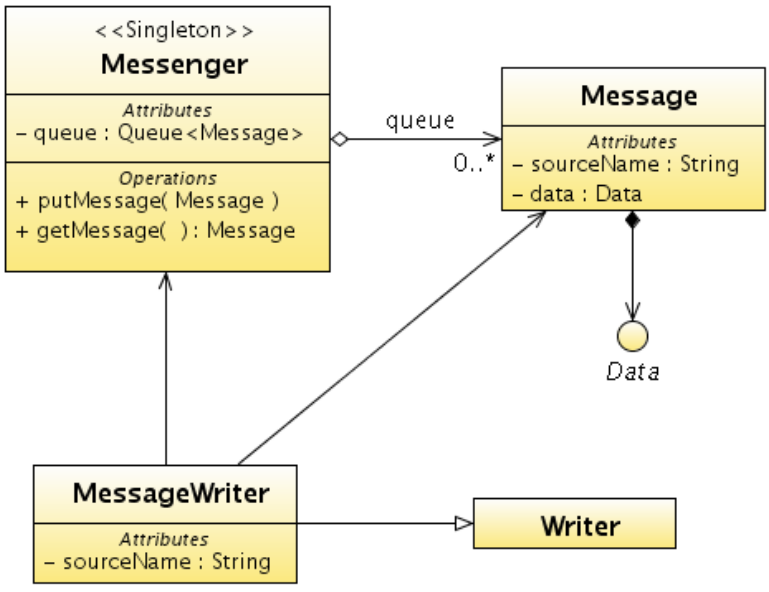
Σχήμα 3.3: Διάγραμμα κλάσεων σχετικών με την αρχικοποίηση

Η κλάση ElementInfo αντιπροσωπεύει την πληροφορία για την αρχικοποίηση ενός στοιχείου του συστήματος. Η κλάση XmlParser είναι η κλάση η οποία διαβάζει και επεξεργάζεται το XML αρχείο και περιέχει τις τρεις λίστες readers, processors και writers, στις οποίες αποθηκεύονται οι πληροφορίες για όλα τα στοιχεία.

3.5. *Κλάσεις σχετικές με ενσωμάτωση*

Σύμφωνα με τη προδιαγραφή ΠΓΧ-6, πρέπει να είναι δυνατή η ενσωμάτωση του συστήματος σε μελλοντικές εφαρμογές. Η πληροφορία την οποία χρειάζεται μια εφαρμογή η οποία ενσωματώνει το σύστημα μπορεί να χωριστεί σε δύο μέρη. Τη στατική πληροφορία, η οποία αποτελείται από τις πληροφορίες σχετικές με τα στοιχεία του συστήματος και τον τρόπο με τον οποίο συνδέονται (η πληροφορία η οποία υπάρχει στο XML αρχείο) και τη δυναμική πληροφορία, η οποία αποτελείται από τα μηνύματα που εισέρχονται στο σύστημα, καθώς και το αποτέλεσμα της επεξεργασίας τους. Για τη στατική πληροφορία δεν χρειάζεται κάποια νέα κλάση, επειδή μπορεί να χρησιμοποιηθεί η XMLParser που σχεδιάστηκε νωρίτερα.

Για τη δυναμική πληροφορία θα χρησιμοποιηθεί μια κλάση singleton. Μια κλάση singleton είναι η κλάση της οποίας μπορεί να δημιουργηθεί μόνο ένα αντικείμενο, στο οποίο αναφέρονται όλες οι υπόλοιπες κλάσεις του συστήματος. Με αυτόν τον τρόπο δύο κλάσεις οι οποίες χρησιμοποιούν την singleton κλάση μπορούν να ανταλλάζουν μηνύματα μεταξύ τους χωρίς να έχουν καμία γνώση η μία για την ύπαρξη της άλλης. Η singleton κλάση θα περιέχει μια ουρά στην οποία θα αποθηκεύονται τα μηνύματα και θα παρέχει μεθόδους για την εισαγωγή και εξαγωγή των μηνυμάτων. Το κάθε μήνυμα θα περιέχει επίσης το όνομα του στοιχείου του συστήματος από το οποίο προέρχεται.



Σχήμα 3.4: Διάγραμμα κλάσεων σχετικών με ενσωμάτωση

Για την ενημέρωση της singleton κλάσης θα σχεδιαστεί μία υλοποίηση ενός Writer. Η υλοποίηση

αυτή θα δέχεται τα μηνύματα και απλώς θα ενημερώνει το singleton. Όταν το σύστημα θα είναι ενσωματωμένο σε κάποια εφαρμογή, ο κάθε Reader και Processor θα στέλνει την έξοδό του σε αυτόν τον Writer και το singleton θα ενημερώνεται για το μήνυμα.

Από τα ανωτέρω προκύπτει το σχήμα 3.4, το οποίο είναι το διάγραμμα κλάσεων σχετικών με την ενσωμάτωση σε άλλες εφαρμογές.

Η κλάση Message αντιπροσωπεύει ένα μήνυμα και περιέχει το όνομα του στοιχείου που στέλνει το μήνυμα και το ίδιο το μήνυμα. Η κλάση Messenger είναι η singleton κλάση που περιέχει την ουρά. Η MessageWriter κλάση είναι η υλοποίηση του Writer για την αποστολή μηνυμάτων στο singleton.

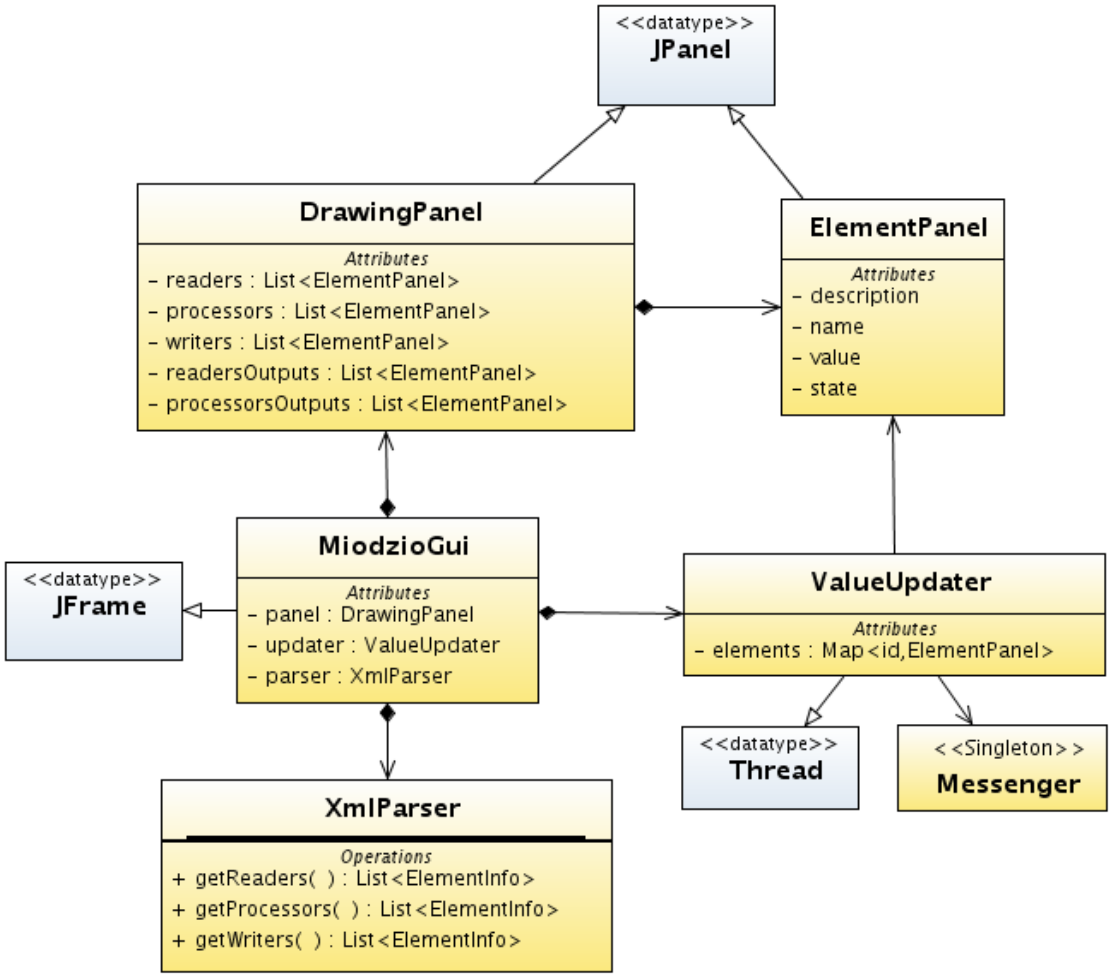
3.6. Κλάσεις σχετικές με τις διεπαφές του χρήστη

Για τη διατήρηση της ανεξαρτησίας μεταξύ του συστήματος και των διεπαφών του χρήστη, η επικοινωνία με τις διεπαφές του χρήστη θα υλοποιηθεί θεωρώντας τις διεπαφές του χρήστη ως μια ξεχωριστή εφαρμογή η οποία ενσωματώνει το σύστημα, δηλαδή με τη χρήση των κλάσεων της προηγούμενης ενότητας.

Για τη σχεδίαση του κάθε στοιχείου του συστήματος θα υπάρχει μια κλάση η οποία θα κληρονομεί το JPanel. Η κλάση αυτή θα επαναπροσδιορίζει τη συνάρτηση η οποία σχεδιάζει την κλάση ώστε να παρουσιάζει το στοιχείο και την τιμή των μηνυμάτων που διέρχονται από αυτό. Για την ομαδική παρουσίαση των στοιχείων θα σχεδιαστεί άλλη μια κλάση που θα κληρονομεί το JPanel, η οποία θα περιέχει όλες τις κλάσεις που αντιπροσωπεύουν τα στοιχεία και θα σχεδιάζει επίσης τις συνδέσεις μεταξύ τους.

Η ενημέρωση των δύο ανωτέρω κλάσεων θα γίνεται μέσω μιας τρίτης κλάσης η οποία θα εκτελείται σε ένα ξεχωριστό νήμα εκτέλεσης. Η κλάση αυτή θα επικοινωνεί με το singleton στο οποίο το σύστημα αποθηκεύει τα μηνύματα και θα ενημερώνει τα στοιχεία των διεπαφών ανάλογα. Τέλος, θα υπάρχει και μια κεντρική κλάση των διεπαφών, η οποία θα χρησιμοποιεί την κλάση XmlParser για τη δημιουργία των ανωτέρω.

Από τα ανωτέρω προκύπτει το σχήμα 3.5, το οποίο είναι το διάγραμμα κλάσεων σχετικών με τις διεπαφές επικοινωνίας με το χρήστη.



Σχήμα 3.5: Διάγραμμα κλάσεων σχετικών με τις διεπαφές του χρήστη

Η κλάση ElementPanel είναι το JPanel για το σχεδιασμό του κάθε στοιχείου. Η κλάση DrawingPanel είναι το JPanel για την οργάνωση των ElementPanel και το σχεδιασμό των συνδέσεων μεταξύ τους. Η κλάση ValueUpdater είναι η κλάση η οποία επικοινωνεί με το singleton και ενημερώνει τις διεπαφές για τις τιμές των μηνυμάτων. Η κλάση MiodzioGui είναι η κλάση που δημιουργεί και οργανώνει τις διεπαφές.

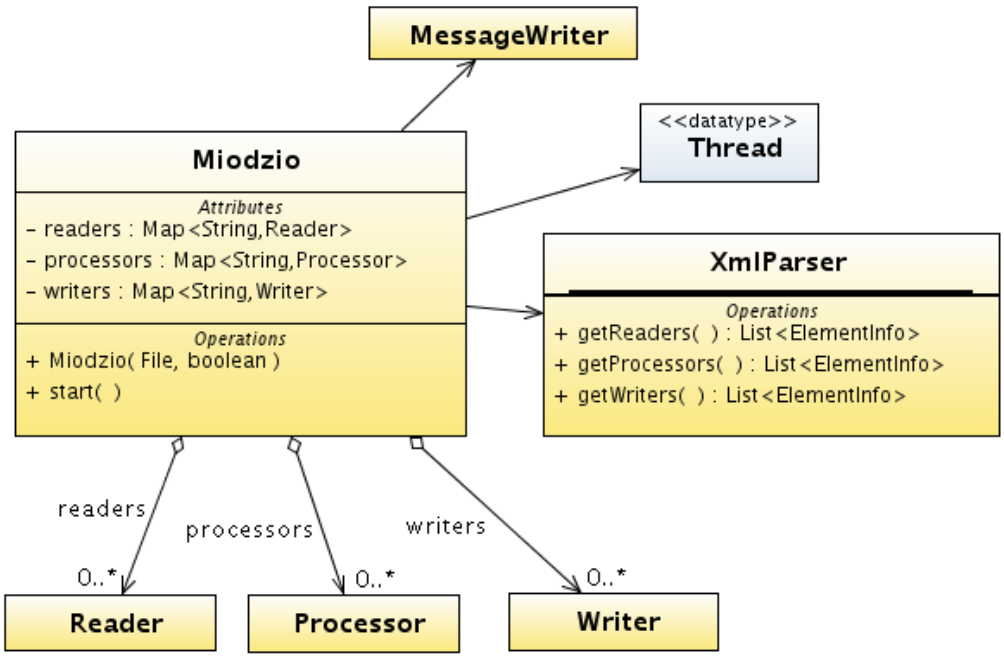
3.7. Κεντρική κλάση του συστήματος

Η τελευταία κλάση που θα σχεδιαστεί για το σύστημα είναι μια κεντρική κλάση, η οποία έχει ως σκοπό την οργάνωση όλων των προηγούμενων. Η κλάση αυτή θα δέχεται κατά την κατασκευή της το όνομα του XML αρχείου με τις επιλογές του χρήστη και μια σημαία εάν το σύστημα

χρησιμοποιείται ως ενσωματωμένο σε άλλη εφαρμογή.

Κατά την αρχικοποίηση θα χρησιμοποιεί την κλάση XMLParser και θα δημιουργεί τρεις λίστες με τα στοιχεία του συστήματος (μια για τις κλάσεις τύπου Reader, μια για τις κλάσεις τύπου Processor και μια για τις κλάσεις τύπου Writer). Επίσης θα συνδέει τα διάφορα στοιχεία μεταξύ τους, όπως περιγράφεται στο XML αρχείο που έχει δώσει ο χρήστης. Εάν το σύστημα χρησιμοποιείται ως ενσωματωμένο σε άλλη εφαρμογή, θα προστίθεται σε κάθε Reader και Processor ως έξοδος και μια κλάση τύπου MessageWriter για την ενημέρωση του singleton. Τέλος, η κλάση θα παρέχει και μια μέθοδο με την οποία θα γίνεται η εκκίνηση όλων των στοιχείων του συστήματος.

Από τα ανωτέρω προκύπτει το σχήμα 3.6, το οποίο είναι το διάγραμμα της κεντρικής κλάσης του συστήματος.



Σχήμα 3.6: Διάγραμμα κεντρικής κλάσης του συστήματος

4. Υλοποίηση

Στο κεφάλαιο αυτό περιγράφεται η υλοποίηση του συστήματος, δηλαδή η υλοποίηση των κλάσεων οι οποίες εντοπίστηκαν στο κεφάλαιο της σχεδίασης (για την ακρίβεια υπάρχουν και μερικές διεπαφές (interfaces) οι οποίες για λόγους ομοιογένειας θα ονομάζονται επίσης κλάσεις). Για την καλύτερη οργάνωση του κώδικα, οι κλάσεις θα χωριστούν σε Java πακέτα (packages), τα οποία θα ακολουθούν την κατηγοριοποίηση που έγινε στο κεφάλαιο της σχεδίασης. Τα ονόματα των πακέτων θα ακολουθούν την ονοματολογία που προτείνει η Java και όλα θα έχουν το πρόθεμα "*gr.ntua.ece.miodzio*" το οποίο θα παραλείπεται στο υπόλοιπο κείμενο για λόγους ευκολίας. Επίσης, στα κομμάτια κώδικα που θα παρουσιαστούν παραλείπονται αρκετά κομμάτια σχολίων ή κώδικα και παρουσιάζονται μόνο τα απαραίτητα, για ευκολότερη ανάγνωση.

4.1. Πακέτο data

Αυτό το πακέτο περιέχει τις κλάσεις οι οποίες είναι σχετικές με τα δεδομένα που διακινούνται στο σύστημα (σχήμα 3.1 του προηγούμενου κεφαλαίου).

4.1.1. Κλάση Data

Η κλάση *Data* είναι μία διεπαφή (interface) την οποία πρέπει να υλοποιούν όλες οι μελλοντικές κλάσεις οι οποίες θα αντιπροσωπεύουν τα δεδομένα που θα διακινούνται στο σύστημα. Ο λόγος

ύπαρξης αυτής της επαφής δεν είναι η δημιουργία κάποιων κοινών χαρακτηριστικών στις κλάσεις που την υλοποιούν, αλλά ο χαρακτηρισμός τους ως κλάσεις δεδομένων. Με αυτόν τον τρόπο το σύστημα θα μπορεί να αντιμετωπίσει όλες αυτές τις κλάσεις με έναν ενιαίο τρόπο. Από την παραπάνω περιγραφή γίνεται φανερό ότι η διεπαφή αυτή δεν θα περιέχει ούτε μέλη ούτε μεθόδους, παρά μόνο θα δηλώνεται ως *interface*. Ο κώδικας που υλοποιεί την κλάση *Data* είναι ο ακόλουθος:

```
package gr.ntua.ece.miodzio.data;

public interface Data {

}
```

Η πρώτη γραμμή στον κώδικα απλώς δηλώνει σε ποιο πακέτο ανήκει η κλάση. Σε κάθε κλάση του συστήματος θα υπάρχει μια αντίστοιχη δήλωση και στο υπόλοιπο του κειμένου θα παραλείπεται για χάρη απλότητας.

Το πιο σημαντικό στη διεπαφή *Data*, δεν είναι ο κώδικας που την υλοποιεί αλλά η Java τεκμηρίωσή της (javadoc) η οποία περιγράφει πως οι κλάσεις που την υλοποιούν πρέπει να συμπεριφέρονται. Εάν μία κλάση που υλοποιεί την *Data* δεν ακολουθήσει αυτούς τους κανόνες, τότε το σύστημα ναι μεν θα λειτουργήσει, αλλά δεν θα είναι εγγυημένη η ασφάλεια του συστήματος. Αυτό οφείλεται στο ότι οι κανόνες συμπεριφοράς των *Data* κλάσεων αποσκοπούν στο να μην επιτρέπουν αλλαγές στα δεδομένα που αποθηκεύονται μετά τη δημιουργία ενός αντικειμένου (τέτοιες κλάσεις λέγονται immutable στη Java).

Οι κανόνες που πρέπει να ακολουθούν όλες οι *Data* κλάσεις είναι οι ακόλουθοι:

1. Τα δεδομένα που αντιπροσωπεύει η κλάση πρέπει να αποθηκεύονται σε αντικείμενα μέλη της, τα οποία να μην είναι προσβάσιμα από οποιαδήποτε άλλη κλάση.

2. Η κλάση δεν πρέπει να επιτρέπει τη μετατροπή των δεδομένων που περιέχει μετά την αρχικοποίησή της.

3. Η κλάση πρέπει να παρέχει για την αρχικοποίηση των αντικειμένων της έναν constructor ο οποίος θα δέχεται ως ορίσματα τα δεδομένα που αντιπροσωπεύει η κλάση.

4. Η κλάση πρέπει να παρέχει μεθόδους για την ανάκληση των τιμών των δεδομένων που αντιπροσωπεύει. Ιδιαίτερη προσοχή πρέπει να δίνεται στα επιστρεφόμενα αντικείμενα, ώστε να μην είναι δυνατή η μετατροπή των δεδομένων μέσω αυτών.

5. Η μέθοδος *toString()* πρέπει να επαναγράφεται για να αντιπροσωπεύει τα δεδομένα της κλάσης.

6. Στη Java τεκμηρίωση της κλάσης (javadoc) πρέπει να περιγράφεται με ακρίβεια το είδος των δεδομένων που αντιπροσωπεύει η κλάση.

Από τους ανωτέρω κανόνες, οι πρώτοι δύο (κανόνες 1 και 2) υπάρχουν για να εξασφαλίσουν την ασφάλεια του συστήματος. Στην περίπτωση που κάποια *Data* κλάση δεν τους ικανοποιεί, το σύστημα θα λειτουργήσει, αλλά θα είναι δυνατή η μεταβολή των δεδομένων που χειρίζεται το σύστημα από κάποιον κακόβουλο χρήστη. Αυτοί είναι και οι δύο πιο σημαντικοί κανόνες από τους έξι και καμία κλάση η οποία δεν τους ακολουθεί δεν πρέπει να χρησιμοποιείται.

Οι επόμενοι δύο κανόνες (κανόνες 3 και 4) είναι αναγκαίοι λόγω των πρώτων δύο για να είναι η κλάση χρήσιμη. Αν μια *Data* κλάση δεν τους ακολουθεί, τότε ναι μεν δεν μπορεί να βλάψει με κανέναν τρόπο το σύστημα, αλλά είτε δεν θα είναι δυνατόν να οριστούν σωστά τα δεδομένα που αντιπροσωπεύει η κλάση, είτε δεν θα είναι δυνατόν να εξαχθούν από αυτήν, οπότε θα είναι άχρηστη. Αν και οι κανόνες αυτοί θα μπορούσαν να παραληφθούν, αναφέρονται στην τεκμηρίωση της κλάσης για τη μεγιστοποίηση της διευκόλυνσης των χρηστών οι οποίοι επιθυμούν να υλοποιήσουν *Data* κλάσεις.

Ο πέμπτος κανόνας υπάρχει για να γίνει εφικτή η ικανοποίηση των προδιαγραφών ΠΔΧ-7και ΠΔΧ-8, κατά τις οποίες επιβάλλεται η παρουσίαση της τιμής των μηνυμάτων στο χρήστη. Η μη συμμόρφωση με αυτόν τον κανόνα δεν θα βλάψει το σύστημα με κανέναν τρόπο, αλλά οι τιμές που θα παρουσιάζονται στις διεπαφές του χρήστη ή σε οποιαδήποτε μηνύματα προς τον χρήστη δεν θα έχουν κανένα νόημα.

Τέλος ο έκτος κανόνας είναι ιδιαίτερα σημαντικός για τη δυνατότητα επαναχρησιμοποίησης της προς υλοποίηση *Data* κλάσης και, αν και δεν επηρεάζει με κανέναν τρόπο τη λειτουργία του συστήματος, δεν πρέπει να παραβλέπεται.

Για τη σωστή ικανοποίηση των ανωτέρω κανόνων ιδιαίτερη προσοχή πρέπει να δοθεί στα αντικείμενα μέλη των *Data* κλάσεων τα οποία είναι mutable (δηλαδή μπορούν να μεταβληθούν μετά τη δημιουργία τους). Τέτοιου είδους αντικείμενα θα πρέπει να αντιγράφονται στον constructor καθώς και θα πρέπει να επιστρέφεται ένα αντίγραφό τους από τις μεθόδους για την ανάκλησή τους, ώστε να μην είναι δυνατή η μεταβολή τους έξω από την κλάση. Αυτό οφείλεται στο ότι στη Java όλα τα ορίσματα στις μεθόδους είναι περασμένα by reference και όχι by value. Το πρόβλημα αυτό δεν υπάρχει με τα αντικείμενα που είναι immutable επειδή δεν μεταβάλλονται μετά τη δημιουργία τους, ούτε με τους βασικούς τύπους δεδομένων επειδή περνιούνται στις μεθόδους by value.

4.1.2. Κλάση DataEvent

Όπως έχει αναφερθεί σε προηγούμενη ενότητα, η διακίνηση των μηνυμάτων μέσα στο σύστημα θα γίνεται με προγραμματισμό οδηγούμενο από συμβάντα. Η κλάση *DataEvent* είναι η πρώτη κλάση που είναι απαραίτητη γι' αυτήν την σχεδίαση και είναι η κλάση η οποία αντιπροσωπεύει τα συμβάντα και θα μεταφέρει τα δεδομένα μεταξύ των στοιχείων του συστήματος.

Η υλοποίηση της κλάσης αυτής είναι αρκετά απλή και ακολουθεί την τυπική υλοποίηση συμβάντων της Java. Για τη μεταφορά των δεδομένων χρησιμοποιεί ένα αντικείμενο μέλος της κλάσης τύπου *Data* το οποίο παρουσιάστηκε προηγουμένως. Ο κώδικας που υλοποιεί την κλάση είναι ο ακόλουθος:

```
public class DataEvent extends EventObject {

    private Data data;

    public DataEvent(Object source, Data data) {
        super(source);
        this.data = data;
    }

    public Data getData() {
        return data;
    }
}
```

Όπως φαίνεται στην πρώτη γραμμή, η κλάση *DataEvent* κληρονομεί την κλάση *EventObject*. Η κλάση αυτή ανήκει στη βιβλιοθήκη *java.util* της Java και πρέπει να κληρονομείται από όλες τις κλάσεις οι οποίες αντιπροσωπεύουν συμβάντα.

Η κλάση έχει ένα μέλος τύπου *Data*, στο οποίο αποθηκεύονται τα δεδομένα του συμβάντος. Οι δύο μέθοδοι είναι ένας constructor ο οποίος δέχεται ως ορίσματα το αντικείμενο που δημιουργεί το συμβάν και τα δεδομένα, και μια κλάση για την ανάκληση των δεδομένων για περαιτέρω χρήση. Εδώ είναι αναγκαίο να παρατηρηθεί ότι το αντικείμενο τύπου *Data* χειρίζεται ως immutable λόγω των κανόνων που επιβάλλονται στη δημιουργία των αντικειμένων *Data*.

4.1.3. Κλάση DataEventListener

Η δεύτερη κλάση που χρειάζεται για την υλοποίηση του προγραμματισμού οδηγούμενου από συμβάντα είναι η διεπαφή *DataEventListener*. Αυτή είναι μια διεπαφή την οποία πρέπει να υλοποιούν όλες οι κλάσεις οι οποίες θα έχουν τη δυνατότητα να ενημερώνονται για συμβάντα τύπου *DataEvent*. Ο κώδικας που υλοποιεί τη διεπαφή είναι ο ακόλουθος:

```
public interface DataEventListener extends EventListener {

    public void dataEventOccured(DataEvent evt);

}
```

Όπως φαίνεται στην πρώτη γραμμή, η διεπαφή *DataEventListener* κληρονομεί την *EventListener* του πακέτου java.util της Java. Αυτό επιβάλλεται από την τυπική υλοποίηση του προγραμματισμού οδηγούμενου από συμβάντα στη Java. Το μοναδικό που επιβάλλεται στις κλάσεις που θα υλοποιήσουν τη διεπαφή είναι η υλοποίηση της συνάρτησης *dataEventOccured()*, η οποία θα καλείται κάθε φορά που η κλάση θα λαμβάνει ένα νέο συμβάν. Για να μην υπάρχουν καθυστερήσεις στο σύστημα, κάθε ενέργεια αυτής της συνάρτησης η οποία είναι απαιτητική σε χρόνο πρέπει να εκτελείται σε ένα ξεχωριστό νήμα εκτέλεσης (thread).

4.1.4. Κλάση DataEventCreator

Η τρίτη και τελευταία κλάση που χρειάζεται για την υλοποίηση του προγραμματισμού οδηγούμενου από συμβάντα είναι η *DataEventCreator*. Αυτή είναι η κλάση την οποία πρέπει να κληρονομούν όλες οι κλάσεις οι οποίες θέλουν να δημιουργούν συμβάντα τύπου *DataEvent*.

Η κλάση αυτή παρέχει τη δυνατότητα διατήρησης μιας λίστας από αντικείμενα τύπου *DataEnentListener* τα οποία ειδοποιεί για τα συμβάντα που δημιουργεί. Η υλοποίησή της φαίνεται στον ακόλουθο κώδικα:σ

```
public abstract class DataEventCreator {

    private final List<DataEventListener> listenerList =
            new ArrayList<DataEventListener>();

    public final void addDataEventListener(DataEventListener listener) {
        listenerList.add(listener);
    }

    public final void removeDataEventListener(
                                    DataEventListener listener) {
        listenerList.remove(listener);
    }

    protected final void fireDataEvent(DataEvent evt) {
        // Go through the listenerList and notify all the listeners
        for (DataEventListener listener : listenerList) {
            listener.dataEventOccured(evt);
        }
    }
}
```

Όπως φαίνεται και στον κώδικα, όλα τα αντικείμενα μέλη και οι μέθοδοι της κλάσης είναι τύπου

final, το μέλος *listenerList* είναι τύπου *private* και η μέθοδος *fireDataEvent()* είναι *protected*. Με αυτόν τον τρόπο δεν επιτρέπεται η αλλαγή τους και ελαχιστοποιείται η πρόσβαση σε αυτά από οποιαδήποτε κλάση που θα κληρονομήσει την *DataEventCreator*, ενισχύοντας έτσι την ασφάλεια του συστήματος.

Το αντικείμενο μέλος *listenerList* είναι μία λίστα τύπου *ArrayList* της βιβλιοθήκης java.util της Java και είναι η λίστα στην οποία αποθηκεύονται τα αντικείμενα τα οποία θα ενημερώνονται για τα συμβάντα. Η διαχείριση αυτής της λίστας γίνεται με τις μεθόδους *addDataEventListener()* και *removeDataEventListener()*, οι οποίες προσθέτουν και αφαιρούν μέλη στη λίστα αντίστοιχα.

Η μέθοδος *fireDataEvent()* είναι η μέθοδος η οποία στέλνει τα συμβάντα στους Listeners. Ο τρόπος με τον οποίο μπορούν να στέλνουν συμβάντα οι κλάσεις που κληρονομούν την *DataEventCreator* είναι καλώντας αυτήν την μέθοδο. Το μοναδικό που κάνει αυτή η μέθοδος είναι να καλεί τη μέθοδο *dataEventOccured()* του κάθε listener που υπάρχει στη λίστα *listenerList*. Εδώ πρέπει να τονισθεί ότι λόγω της σχεδίασης του συστήματος, οι συναρτήσεις *dataEventOccured()* εκτελούνται αρκετά γρήγορα, οπότε οι listeners ενημερώνονται στο ίδιο νήμα εκτέλεσης (thread). Οι δοκιμές που έγιναν έδειξαν ότι η προσέγγιση αυτή έχει ικανοποιητική απόδοση και μάλιστα καλύτερη από την εναλλακτική του να ενημερώνεται κάθε listener σε διαφορετικό νήμα εκτέλεσης (σε αυτήν την περίπτωση το σύστημα επιβαρύνεται και από την επιπλέον διαχείριση των νημάτων εκτέλεσης).

4.1.5. Κλάση DataEventsQueue

Η τελευταία κλάση στο πακέτο *data* είναι η *DataEventsQueue* και η ανάγκη της υλοποίησής της, όπως αναφέρθηκε και προηγουμένως, προκύπτει από τις προδιαγραφές ΠΕΠΜ-3 και ΠΕΞΜ-6. Η κλάση αυτή είναι μια υλοποίηση της διεπαφής *DataEventListener*, η οποία αποθηκεύει τα συμβάντα που λαμβάνει σε μια ουρά και παρέχει τη δυνατότητα ανάκλησής τους σε χρονολογική σειρά. Η υλοποίησή της γίνεται από τον ακόλουθο κώδικα:

```
public class DataEventsQueue implements DataEventListener {

    private LinkedBlockingQueue<Data> queue = new
                                        LinkedBlockingQueue<Data>();

    public void setQueueSize(int size) {
        if (size > 0) {
            queue = new LinkedBlockingQueue<Data>(size);
        } else {
            queue = new LinkedBlockingQueue<Data>();
        }
    }
```

```
    public void dataEventOccured(DataEvent evt) {
        // Add the data to the queue
        queue.add(evt.getData());
    }

    public Data getOldestData() {
        Data result = null;
        try {
            result = queue.poll(10000, TimeUnit.MILLISECONDS);
        } catch (InterruptedException ex) {
        }
        return result;
    }
}
```

Στην πρώτη γραμμή του κώδικα γίνεται η δήλωση ώστε η κλάση *DataEventsQueue* να υλοποιεί τη διεπαφή *DataEventListener*. Η κλάση έχει ένα private μέλος, το *queue*, το οποίο είναι μία ουρά *LinkedBlockingQueue* της βιβλιοθήκης java.util.concurrent της Java, στην οποία αποθηκεύονται τα εισερχόμενα συμβάντα. Η επιλογή του συγκεκριμένου τύπου ουράς έγινε επειδή επιτρέπει τη ρύθμιση του μέγιστου αριθμού στοιχείων, πράγμα που απαιτείται από τις προδιαγραφές ΠΕΠΜ-4 και ΠΕΞΜ-7.

Για τη ρύθμιση του μεγέθους της ουράς η κλάση παρέχει τη μέθοδο *setQueueSize()*. Εάν η παράμετρος της μεθόδου είναι ένας θετικός αριθμός τότε το μέγεθος της ουράς ορίζεται σύμφωνα με αυτόν τον αριθμό, ενώ εάν είναι μηδέν ή κάποιος αρνητικός αριθμός δεν θέτεται όριο στο μέγεθος της ουράς (στην πραγματικότητα το όριο είναι 2147483647, όπου είναι ο μεγαλύτερος αριθμός που μπορεί να αντιπροσωπεύει μία μεταβλητή τύπου *int* στη Java). Όπως φαίνεται και στον κώδικα, η διαδικασία ορισμού του μεγέθους της ουράς δημιουργεί ένα νέο αντικείμενο *LinkedBlockingQueue*. Αυτό έχει ως αποτέλεσμα οτιδήποτε τιμές προϋπήρχαν στην ουρά να χάνονται. Αυτό δεν αποτελεί πρόβλημα για το σύστημα, επειδή το μέγεθος της ουράς θα ορίζεται μόνο μια φορά κατά την αρχικοποίηση του συστήματος, πριν ληφθούν μηνύματα, οπότε η ουρά θα είναι άδεια.

Η μέθοδος *dataEventOccured()* είναι η υλοποίηση της μεθόδου της διεπαφής *DataEventListener*. Κάθε φορά που εκτελείται απλώς προσθέτει το αντικείμενο τύπου *Data* που μεταφέρει το συμβάν στην ουρά.

Τέλος, η μέθοδος *getOldestData()* επιστρέφει το παλαιότερο αντικείμενο από την ουρά. Στην περίπτωση που η ουρά είναι άδεια, η μέθοδος δεν επιστρέφει αμέσως, αλλά περιμένει για δέκα δευτερόλεπτα πριν επιστρέψει την τιμή null. Εάν σε αυτήν την χρονική περίοδο ληφθεί κάποιο καινούργιο συμβάν, θα επιστραφεί η τιμή του και όχι null. Αυτό γίνεται με τέτοιον τρόπο ώστε δεν χρησιμοποιείται υπολογιστική ισχύς κατά την αναμονή. Η προσέγγιση αυτή έγινε για τη

διευκόλυνση στην υλοποίηση των κλάσεων που θα λαμβάνουν τα δεδομένα από την κλάση *DataEventsQueue*.

4.2. Πακέτο elements

Αυτό το πακέτο περιέχει τις κλάσεις οι οποίες είναι σχετικές με τα στοιχεία του συστήματος (σχήμα 3.2 του προηγούμενου κεφαλαίου).

4.2.1. Κλάση Describable

Η κλάση *Describable* είναι μία διεπαφή η οποία είναι αναγκαία για την ικανοποίηση της προδιαγραφής ΠΓΧ-4. Η διεπαφή αυτή επιβάλει έναν ομοιόμορφο τρόπο περιγραφής των στοιχείων, επιβάλλοντας την υλοποίηση μιας μεθόδου που επιστρέφει την περιγραφή αυτή. Ο κώδικας που την υλοποιεί είναι ο ακόλουθος:

```
public interface Describable {

    public String getDescription();

}
```

Όπως φαίνεται και στον κώδικα, η μέθοδος η οποία επιστρέφει την περιγραφή των *Describable* αντικειμένων είναι η *getDescription()*.

4.2.2. Κλάση Configurable

Η δημιουργία της κλάσης *Configurable*, η οποία είναι μία διεπαφή, επιβάλλεται από τις προδιαγραφές ΠΕΙΜ-3, ΠΕΠΜ-2 και ΠΕΞΜ-3. Σύμφωνα με αυτές τις προδιαγραφές, όλα τα στοιχεία του συστήματος πρέπει να είναι παραμετροποιήσιμα με κάποιον κοινό τρόπο. Η διεπαφή *Configurable* υποχρεώνει τις κλάσεις που την υλοποιούν να χρησιμοποιούν μία κοινή μέθοδο για την παραμετροποίησή τους. Ο κώδικας που υλοποιεί τη διεπαφή είναι ο ακόλουθος:

```
public interface Configurable {

    public void setParameters(Properties parameters) throws
                                                ParameterProblemException;

}
```

Η μέθοδος που χρησιμοποιείται για την παραμετροποίηση των κλάσεων που υλοποιούν την *Configurable* είναι η *setParameters()*. Η μέθοδος αυτή δέχεται ως όρισμα το αντικείμενο *parameters*, το οποίο είναι τύπου *Properties* της βιβλιοθήκης java.util της Java. Το αντικείμενο αυτό περιέχει τις παραμέτρους σε ζεύγη ονομάτων – τιμών και παρέχει μεθόδους για την

πρόσβασή τους μέσω του ονόματός τους.

Τα ονόματα και οι τιμές που είναι αποθηκευμένες στο *properties* είναι τύπου String, το οποίο είναι ιδανικό για την συγκεκριμένη εφαρμογή, επειδή ο τύπος των παραμέτρων δεν είναι γνωστός και μπορεί να αλλάζει σε κάθε υλοποίηση των στοιχείων του συστήματος. Η κάθε υλοποίηση είναι υπεύθυνη να μετατρέπει τις παραμέτρους που δέχεται στη σωστή μορφή. Στην περίπτωση που κάποια παράμετρος έχει λάθος μορφή ή μη αποδεκτή τιμή, ή στην περίπτωση που κάποια υποχρεωτική παράμετρος δεν συμπεριλαμβάνεται στο properties, η μέθοδος πρέπει να δημιουργεί ένα σφάλμα τύπου *ParameterProblemException* που να περιγράφει το πρόβλημα, το οποίο αναλύεται στην αμέσως επόμενη ενότητα.

4.2.3. Κλάση ParameterProblemException

Όπως αναφέρθηκε προηγουμένως η κλάση *ParameterProblemException* δημιουργείται όταν υπάρχει κάποιο σφάλμα κατά την παραμετροποίηση των στοιχείων του συστήματος. Η υλοποίηση της κλάσης αυτής ακολουθεί τον τρόπο διαχείρισης σφαλμάτων της Java, και η υλοποίησή της είναι ιδιαίτερα απλή, όπως φαίνεται και στον ακόλουθο κώδικα:

```
public class ParameterProblemException extends Exception {

    public ParameterProblemException() {
        super();
    }

    public ParameterProblemException(String msg) {
        super(msg);
    }
}
```

Η Java παρέχει την κλάση *Exception*, την οποία πρέπει να κληρονομούν όλες οι κλάσεις που αντιπροσωπεύουν σφάλματα. Από αυτήν κληρονομούνται κάποια ιδιαίτερα χαρακτηριστικά, ώστε η κλάση να μπορεί να διαχειριστεί ως ένα σφάλμα. Η κατασκευή της κλάσης μπορεί να γίνει είτε χωρίς κανένα όρισμα είτε με ένα String το οποίο περιέχει κάποιο μήνυμα που περιγράφει το σφάλμα.

4.2.4. Κλάση Reader

Η κλάση *Reader* είναι η κλάση την οποία πρέπει να κληρονομούν όλα τα στοιχεία του συστήματος τα οποία προορίζονται για επικοινωνία με τις εφαρμογές εισόδου. Όπως φαίνεται και στον ακόλουθο κώδικα, δεν προσθέτει κάποια καινούργια μέλη ή μεθόδους, αλλά απλώς ορίζει την κλάση που πρέπει να κληρονομείται και τις διεπαφές που πρέπει να υλοποιούνται.

```
public abstract class Reader extends DataEventCreator implements
                                    Runnable, Configurable, Describable {

}
```

Όπως φαίνεται στον κώδικα, η κλάση *Reader* κληρονομεί την *DataEventCreator* του πακέτου data. Αυτό γίνεται επειδή αφού λάβει κάποια δεδομένα από μία εφαρμογή εισόδου, θα τα προωθήσει σε άλλα στοιχεία του συστήματος. Η διεπαφή *Runnable* της βιβλιοθήκης java.lang της Java πρέπει να υλοποιείται για να είναι δυνατή η εκτέλεση του κάθε *Reader* σε διαφορετικό νήμα εκτέλεσης. Τέλος οι διεπαφές *Configurable* και *Describable* είναι αυτές που περιγράφηκαν προηγουμένως και πρέπει να υλοποιούνται για να είναι οι κλάσεις *Reader* παραμετροποιήσιμες και να παρέχουν την περιγραφή τους.

Όπως και η κλάση *Data* του πακέτου data, έτσι και η κλάση *Reader* έχει σχεδιαστεί για υλοποίηση από τους χρήστες, οπότε η java τεκμηρίωσή της είναι εξίσου σημαντική. Σε αυτήν περιγράφονται τρεις κανόνες για την ορθή υλοποίηση μιας κλάσης που κληρονομεί την *Reader* ως εξής:

1. Πρέπει να υλοποιείται τη μέθοδος *setParameters()* της διεπαφής *Configurable* για να καθορίζει τις παραμέτρους της. Εάν κάποια παράμετρος λείπει ή η τιμή της δεν είναι σωστή πρέπει να δημιουργείται ένα σφάλμα τύπου *ParameterProblemException* το οποίο να περιγράφει το πρόβλημα.

2. Πρέπει να υλοποιείται η μέθοδος *run()* της διεπαφής *Runnable* με ένα άπειρα επαναλαμβανόμενο loop, στο οποίο θα δέχεται μηνύματα από τις εφαρμογές εισόδου, θα δημιουργεί ένα αντικείμενο *Data* από αυτά και θα στέλνει ένα *DataEvent* στις κλάσεις που περιμένουν για συμβάντα χρησιμοποιώντας τη συνάρτηση *fireDataEvent()*.

3. Πρέπει να υλοποιείται η μέθοδος *getDescription()* της διεπαφής *Describable* ώστε να επιστρέφει μια περιγραφή της λειτουργίας της κλάσης.

Για θέματα ασφαλείας, κατά τη δημιουργία της μεθόδου *setParameters()*, οι διάφορες παράμετροι πρέπει να αποθηκεύονται εσωτερικά στην κλάση με τέτοιο τρόπο ώστε να μην υπάρχει πρόσβαση σε αυτές. Οι τιμές των παραμέτρων μπορούν να χρησιμοποιηθούν και στη συνάρτηση που επιστρέφει την περιγραφή της κλάσης, για να αποδοθεί μια πιο πλήρης περιγραφή. Για παράδειγμα μία κλάση που περιμένει μηνύματα σε κάποιο port μπορεί να γράφει στην περιγραφή της το νούμερο του port στο οποίο ακούει.

4.2.5. Κλάση Processor

Η κλάση *Processor* είναι η κλάση την οποία πρέπει να κληρονομούν όλα τα στοιχεία του

συστήματος τα οποία προορίζονται για την επεξεργασία των μηνυμάτων. Όπως φαίνεται και στον κώδικα που ακολουθεί, επιβάλλει τις ίδιες υλοποιήσεις με την κλάση *Reader*, αλλά επιπλέον περιέχει και ένα αντικείμενο μέλος τύπου *DataEventsQueue*. Ο κώδικας που υλοποιεί την κλάση είναι ο ακόλουθος:

```
public abstract class Processor extends DataEventCreator implements
                                    Runnable, Configurable, Describable {

    protected DataEventsQueue queue = new DataEventsQueue();

    public DataEventsQueue getQueue() {
        return queue;
    }
}
```

Όπως φαίνεται και στον κώδικα, η κλάση *Processor* περιέχει και τη μέθοδο *getQueue()* η οποία επιστρέφει την ουρά των μηνυμάτων. Το σύστημα χρησιμοποιεί αυτήν την μέθοδο κατά την αρχικοποίηση του συστήματος και θέτει την ουρά της κλάσης να περιμένει για συμβάντα από *Readers* ή άλλους *Processors*. Δηλαδή η ίδια η κλάση *Processor* δεν ενημερώνεται για τα συμβάντα, παρά μόνο η λίστα της.

Η Java τεκμηρίωση της κλάσης είναι εξίσου σημαντική με της κλάσης *Reader* και περιγράφει τους ίδιους τρεις κανόνες, με κάποιες διαφορές στον δεύτερο. Στην περίπτωση των Processors, στο loop της συνάρτησης *run()* τα δεδομένα δεν έρχονται από κάποια εφαρμογή εισόδου, αλλά από την ουρά του αντικειμένου με την κλήση της συνάρτησης *queue.getOldestData()*. Όταν καλείται αυτή η συνάρτηση πρέπει να δίνεται ιδιαίτερη προσοχή στην τιμή που επιστρέφει, επειδή αν η ουρά είναι άδεια και δεν υπάρχει κανένα εισερχόμενο μήνυμα για περισσότερο από 10 δευτερόλεπτα από την στιγμή που καλείται η συνάρτηση, επιστρέφει την τιμή null. Αφού ληφθούν κάποια δεδομένα θα γίνεται η επεξεργασία τους και θα ενημερώνονται γι' αυτό όλες οι κλάσεις που αναμένουν για συμβάντα, μέσω της μεθόδου *fireDataEvent()*.

4.2.6. Κλάση Writer

Η τελευταία κλάση του πακέτου elements είναι η *Writer*, την οποία πρέπει να κληρονομούν όλες οι κλάσεις που προορίζονται για επικοινωνία με τις εφαρμογές εξόδου. Η υλοποίηση της κλάσης αυτής είναι όμοια με της *Processor*, αλλά δεν κληρονομεί την κλάση *DataEventCreator*, αφού δεν χρειάζεται να δημιουργεί συμβάντα. Ο κώδικας που την υλοποιεί είναι ο ακόλουθος:

```
public abstract class Writer implements Runnable, Configurable,
                                                    Describable {

    protected DataEventsQueue queue = new DataEventsQueue();
```

```
    public DataEventsQueue getQueue() {
        return queue;
    }
}
```

Στην Java τεκμηρίωση της κλάσης αναφέρονται οι ίδιοι τρεις κανόνες. Η μοναδική διαφορά με τους κανόνες της κλάσης *Processor* είναι ότι στη μέθοδο *run()*, αφού ληφθεί κάποιο μήνυμα από την ουρά, μετατρέπεται σε κάποια μορφή κατάλληλη για την μορφή εξόδου και αποστέλλεται σε αυτήν.

4.3. Πακέτο config

Το πακέτο αυτό περιέχει τις κλάσεις οι οποίες είναι σχετικές με την παραμετροποίηση του συστήματος (σχήμα 3.3 του προηγούμενου κεφαλαίου). Όπως αναφέρθηκε σε προηγούμενη ενότητα, η παραμετροποίηση του συστήματος θα γίνεται μέσω ενός XML αρχείου. Οι ακόλουθες κλάσεις είναι υπεύθυνες για το διάβασμα του αρχείου και τη μετατροπή των πληροφοριών που περιέχει σε μια μορφή κατάλληλη για χρήση από το υπόλοιπο σύστημα. Σε αυτήν την ενότητα, λόγω της σχετικότητας με τις κλάσεις του πακέτου, παρουσιάζεται και το αρχείο miodzio.dtd, το οποίο προσδιορίζει τη σύνταξη την οποία πρέπει να έχουν τα XML αρχεία που θα χρησιμοποιηθούν.

4.3.1. Αρχείο miodzio.dtd

Το αρχείο αυτό ορίζει τη σύνταξη που πρέπει να ακολουθούν τα XML αρχεία για την παραμετροποίηση του συστήματος. Το αρχικό στοιχείο του XML αρχείου πρέπει να ονομάζεται MIODZIO και πρέπει να περιέχει τρία υποστοιχεία με ονόματα READERS, PROCESSORS και WRITERS, τα οποία θα περιέχουν τις πληροφορίες για τις κλάσεις που επικοινωνούν με τις εφαρμογές εισόδου, που επεξεργάζονται τα μηνύματα και που επικοινωνούν με τις εφαρμογές εξόδου αντίστοιχα.

```
<!ELEMENT MIODZIO (READERS,PROCESSORS,WRITERS)>
<!ELEMENT READERS (READER*)>
<!ELEMENT PROCESSORS (PROCESSOR*)>
<!ELEMENT WRITERS (WRITER*)>
```

Το κάθε στοιχείο READER, το οποίο αντιπροσωπεύει μία κλάση επικοινωνίας με κάποια εφαρμογή εισόδου, πρέπει να έχει ως χαρακτηριστικά (attributes) το όνομα της κλάσης μαζί με το java path της (για παράδειγμα gr.ntua.ece.miodzio.example.ExampleReader) ώστε το σύστημα να ξέρει ποια κλάση να χρησιμοποιήσει, και κάποιο αναγνωριστικό String για περαιτέρω αναφορά. Ως υποστοιχεία, πρέπει να έχει δύο λίστες, μία με τις παραμέτρους για την παραμετροποίηση της

κλάσης και μία με τα στοιχεία του συστήματος τα οποία θα ενημερώνονται για τα συμβάντα από αυτήν την κλάση.

```
<!ELEMENT READER (PARAMETERS,OUTPUTS)>
<!ATTLIST READER OBJECTID ID #REQUIRED>
<!ATTLIST READER CLASS CDATA #REQUIRED>
```

Το κάθε στοιχείο PROCESSOR, το οποίο αντιπροσωπεύει μία κλάση επεξεργασίας μηνυμάτων, πρέπει να έχει τα ίδια χαρακτηριστικά και υποστοιχεία με τα στοιχεία READER, αλλά επιπλέον μπορεί να έχει και ένα παραπάνω χαρακτηριστικό, το οποίο θα ορίζει το μέγεθος της ουράς αναμονής. Το χαρακτηριστικό αυτό δεν είναι υποχρεωτικό και στην περίπτωση που λείπει, η ουρά θα έχει απεριόριστο μέγεθος.

```
<!ELEMENT PROCESSOR (PARAMETERS,OUTPUTS)>
<!ATTLIST PROCESSOR OBJECTID ID #REQUIRED>
<!ATTLIST PROCESSOR CLASS CDATA #REQUIRED>
<!ATTLIST PROCESSOR QUEUESIZE CDATA #IMPLIED>
```

Τα στοιχεία WRITER, τα οποία αντιπροσωπεύουν κλάσεις επικοινωνίας με τις εφαρμογές εξόδου, έχουν τα ίδια χαρακτηριστικά και υποστοιχεία με τα στοιχεία PROCESSOR, με τη διαφορά ότι δεν έχουν λίστα με στοιχεία τα οποία θα ενημερώνουν για συμβάντα, αφού δεν δημιουργούν συμβάντα.

```
<!ELEMENT WRITER (PARAMETERS)>
<!ATTLIST WRITER OBJECTID ID #REQUIRED>
<!ATTLIST WRITER CLASS CDATA #REQUIRED>
<!ATTLIST WRITER QUEUESIZE CDATA #IMPLIED>
```

Η κάθε λίστα παραμέτρων περιέχει στοιχεία PARAMETER, τα οποία αντιπροσωπεύουν τις παραμέτρους της κλάσης. Κάθε παράμετρος έχει ως χαρακτηριστικό το όνομά της και περιέχει ένα αλφαριθμητικό, όπου είναι η τιμή της παραμέτρου.

```
<!ELEMENT PARAMETERS (PARAMETER*)>
<!ELEMENT PARAMETER (#PCDATA)>
<!ATTLIST PARAMETER NAME CDATA #REQUIRED>
```

Τέλος, η κάθε λίστα με στοιχεία για ενημέρωση για συμβάντα θα περιέχει στοιχεία OUTPUT, τα οποία απλώς περιέχουν αλφαριθμητικά με τα αναγνωριστικά των κλάσεων προς ενημέρωση στο XML αρχείο.

```
<!ELEMENT OUTPUTS (OUTPUT*)>
<!ELEMENT OUTPUT (#PCDATA)>
```

4.3.2. Κλάση ElementInfo

Λόγω της πληθώρας των χαρακτηριστικών που έχουν τα στοιχεία του συστήματος, επιβάλλεται η δημιουργία μιας κλάσης η οποία θα τα αντιπροσωπεύει. Επειδή οι readers, processors και writers έχουν σχεδόν ίδια στοιχεία, αρκεί η δημιουργία μίας κλάσης. Η κλάση αυτή είναι η *ElementInfo* και διατηρεί πληροφορίες για το όνομα, τον τύπο της κλάσης, το μέγεθος της ουράς, τις παραμέτρους και τα στοιχεία εξόδου. Ο κώδικας που την υλοποιεί είναι ο ακόλουθος:

```
public class ElementInfo {

    private String id;
    private String className;
    private int queueSize;
    private Properties parameters;
    private List<String> outputs;

    public ElementInfo(String id, String className, int queueSize,
                       Properties parameters,List<String> outputs) throws
                                                    IllegalArgumentException {
        ......
    }

    public String getId() {
        return id;
    }

    public String getClassName() {
        return className;
    }

    public int getQueueSize() {
        return queueSize;
    }

    public Properties getParameters() {
        // Copy the properties and return them
        Properties toReturn = new Properties();
        for (Map.Entry entry : parameters.entrySet()) {
            toReturn.setProperty((String)entry.getKey(),
                                                (String)entry.getValue());
        }
        return toReturn;
    }

    public List<String> getOutputs() {
        // Copy the outputs and return them
        return new ArrayList<String>(outputs);
    }
}
```

Τα μέλη της κλάσης *id*, *className*, *queueSize*, *parameters* και *outputs* διατηρούν τις ανάλογες τιμές και παρέχονται μέθοδοι για την ανάκλησή τους. Στην περίπτωση των *getParameters()* και

getOutputs() τα αντικείμενα *parameters* και *outputs* αντιγράφονται και επιστρέφεται το αντίγραφό τους. Αυτό γίνεται για να αυξηθεί η ασφάλεια του συστήματος, ώστε να μην είναι δυνατή η μεταβολή της κλάσης μετά τη δημιουργία τους.

Η κατασκευή της κλάσης γίνεται δίνοντας τα ως παραμέτρους τιμές για όλα τα μέλη της κλάσης. Ο κώδικας παραλήφθηκε προηγουμένως για συντομία και είναι ο ακόλουθος:

```
    public ElementInfo(String id, String className, int queueSize,
                       Properties parameters,List<String> outputs) throws
                                                    IllegalArgumentException {

        // Check that the parameters and the outputs are not null
        if (parameters == null) {
            throw new IllegalArgumentException(
                                    "Argument parameters cannot be null");
        }
        if (outputs == null) {
            throw new IllegalArgumentException(
                                      "Argument outputs cannot be null");
        }

        // Set locally the ID
        this.id = id;

        // Set locally the class name
        this.className = className;

        // Set locally the queue size
        this.queueSize = queueSize;

        // Set locally the parameters
        this.parameters = new Properties();
        for (Map.Entry entry : parameters.entrySet()) {
            this.parameters.setProperty((String)entry.getKey(),
                                                    (String)entry.getValue());
        }

        // Set locally the outputs
        this.outputs = new ArrayList<String>(outputs);
    }
```

Όπως φαίνεται και στον κώδικα, η κλάση δεν επιτρέπεται να δημιουργηθεί αν το αντικείμενο με τις παραμέτρους ή το αντικείμενο με τις εξόδους έχουν την τιμή null. Σε αυτήν την περίπτωση δημιουργείται ένα σφάλμα τύπου *IllegalArgumentException* της βιβλιοθήκης java.lang της Java και το αντικείμενο δεν δημιουργείται. Στην περίπτωση που κάποιο στοιχείο του συστήματος δεν έχει καμία παράμετρο πρέπει να δοθεί ως όρισμα ένα αντικείμενο *Properties* το οποίο να μην περιέχει κανένα ζευγάρι ονόματος – τιμής και αντίστοιχα, όταν δεν υπάρχει καμία έξοδος πρέπει να δίνεται μία κενή λίστα.

Για την ασφάλεια του συστήματος, κατά τη δημιουργία ενός αντικειμένου τύπου *ElementInfo* γίνεται αντιγραφή του αντικειμένου *Properties* και της λίστας που δίνονται στον constructor. Με αυτόν τον τρόπο δεν υπάρχει τρόπος να αλλαχτούν οι τιμές τους εξωτερικά από την κλάση. Αυτό σε συνδυασμό με την αντιγραφή που γίνεται στις μεθόδους *getParameters()* και *getOutputs()* κάνει την κλάση immutable και αυξάνει την ασφάλεια του συστήματος. Οι ενέργειες αυτές καταναλώνουν περισσότερο χρόνο κατά την εκτέλεση του συστήματος, αλλά οι συγκεκριμένες μέθοδοι χρησιμοποιούνται μόνο κατά την αρχικοποίηση του συστήματος, οπότε δεν βλάπτουν την απόδοσή του (αν και ο χρόνος αυτός έτσι κι αλλιώς είναι ιδιαίτερα μικρός για να είναι σημαντικότερος από την ασφάλεια του συστήματος).

4.3.3. Κλάση XmlParser

Η κλάση *XmlParser* διαβάζει ένα αρχείο XML και δημιουργεί τρεις λίστες με τις πληροφορίες για τα στοιχεία του συστήματος. Η Java παρέχει διάφορες βιβλιοθήκες για την προσπέλαση XML αρχείων. Από αυτές επιλέχθηκε η μέθοδος SAX. Αυτή η μέθοδος διαβάζει σειριακά το αρχείο και δημιουργεί συμβάντα ανάλογα με τα στοιχεία XML που βρίσκει. Αυτό την κάνει κατάλληλη στην περίπτωση που ένα αρχείο θα διαβαστεί μία φορά για την εξαγωγή των δεδομένων του και δεν γίνονται ενημερώσεις στο αρχείο (που είναι και η περίπτωση χρήσης στο σύστημα). Ο κώδικας που υλοποιεί την κλάση *XmlParser* είναι ο ακόλουθος:

```
public class XmlParser {

    List<ElementInfo> readers = new ArrayList<ElementInfo>();
    List<ElementInfo> processors = new ArrayList<ElementInfo>();
    List<ElementInfo> writers = new ArrayList<ElementInfo>();

    public XmlParser(File xmlFile) throws SAXException, IOException,
            ParserConfigurationException {
        // Use the default (non-validating) parser
        SAXParserFactory factory = SAXParserFactory.newInstance();

        // Parse the input
        SAXParser saxParser = factory.newSAXParser();
        saxParser.parse(xmlFile, new XmlHandler());
    }

    private class XmlHandler extends DefaultHandler {
         ...
    }

    public List<ElementInfo> getReaders() {
        return new ArrayList<ElementInfo>(readers);
    }

    public List<ElementInfo> getProcessors() {
        return new ArrayList<ElementInfo>(processors);
```

```
    }

    public List<ElementInfo> getWriters() {
        return new ArrayList<ElementInfo>(writers);
    }
}
```

Τα αντικείμενα μέλη *readers, processors* και *writers* είναι οι λίστες στις οποίες αποθηκεύονται οι πληροφορίες για τα στοιχεία του συστήματος. Οι τρεις συναρτήσεις *getReaders()*, *getProcessors()* και *getWriters()* επιστρέφουν αντίγραφα των λιστών αντίστοιχα. Η επιστροφή αντιγράφων και όχι των ίδιων των λιστών γίνεται για τους λόγους ασφαλείας που έχουν αναφερθεί νωρίτερα.

Κατά τη δημιουργία της κλάσης γίνεται το διάβασμα ενός αρχείου το οποίο δίνεται από το χρήστη και αρχικοποιούνται οι λίστες. Κατά το διάβασμα του αρχείου χρησιμοποιείται η μέθοδος SAX και η κλάση *XmlHandler*, η οποία είναι εσωτερική κλάση της *XmlParser* πρώτον επειδή δεν χρησιμοποιείται πουθενά αλλού και δεύτερον για να έχει πρόσβαση στις λίστες *readers, processors* και *writers*. Ο κώδικας που υλοποιεί την κλάση αυτήν την κλάση είναι ο ακόλουθος:

```
    private class XmlHandler extends DefaultHandler {

        private String className = null;
        private String id = null;
        private int queueSize = 0;
        private Properties parameters = new Properties();
        private List<String> outputs = new ArrayList<String>();
        private String lastElement = null;
        private String parameterName = null;
        private String parameterValue = null;
        private String outputValue = null;

        public void startElement(String uri, String localName,
               String qName, Attributes attributes) throws SAXException {
             ....
        }

        public void characters(char[] ch, int start, int length)
                                                    throws SAXException {
            ....
        }

        public void endElement(String uri, String localName,
                                    String qName) throws SAXException {
            ....
        }
    }
```

Οι συναρτήσεις *startElement()*, *characters()* και *endElement()* καλούνται όταν συναντάται η αρχή ενός στοιχείου XML, ένα σύνολο κειμένου και το τέλος ενός στοιχείου XML αντίστοιχα κατά τη διάρκεια του διαβάσματος του αρχείου. Οι παράμετροι που δέχονται οι συναρτήσεις έχουν τις

ανάλογες τιμές που διαβάστηκαν. Όλα τα αντικείμενα μέλη της κλάσης είναι βοηθητικά και χρησιμοποιούνται για την προσωρινή αποθήκευση των πληροφοριών από το αρχείο ώστε να είναι διαθέσιμες κατά τη δημιουργία των αντικειμένων που προστίθενται στις λίστες *readers*, *procerrors* και *writers*.

Ο κώδικας που υλοποιεί τη μέθοδο *startElement()*, ο οποίος καλείται κάθε φορά που συναντάται η αρχή ενός XML στοιχείου, είναι ο ακόλουθος:

```
        public void startElement(String uri, String localName,
                String qName, Attributes attributes) throws SAXException {
            // Set the last element variable
            lastElement = qName;

            // Check if we have WRITER, READER or processor and if
            // yes save the
            // class name, the ID and the queue size (if it exists)
            if (qName.equals("READER") || qName.equals("PROCESSOR") ||
                    qName.equals("WRITER")) {
                className = attributes.getValue("CLASS");
                id = attributes.getValue("OBJECTID");
                String queueSizeString =
                                        attributes.getValue("QUEUESIZE");
                if (queueSizeString != null) {
                    try {
                        queueSize = Integer.parseInt(queueSizeString);
                    } catch (Exception ex) {
                        throw new SAXException("Queue size for " + id +
                                               " is not " + "an integer");
                    }
                } else {
                    queueSize = 0;
                }
            }

            // Check if we have PARAMETER and save the parameter name
            if (qName.equals("PARAMETER")) {
                parameterName = attributes.getValue("NAME");
            }
        }
```

Όπως φαίνεται και στον κώδικα, όταν συναντάται η αρχή ενός XML στοιχείου αποθηκεύεται τοπικά ο τύπος του στοιχείου και εάν το στοιχείο είναι τύπου READER, PROCESSOR ή WRITER αποθηκεύονται επίσης το όνομα της κλάσης, το αναγνωριστικό του στοιχείου καθώς και το μέγεθος της ουράς. Στην περίπτωση που είναι τύπου PARAMETER αποθηκεύεται το όνομα της παραμέτρου.

Ο κώδικας που υλοποιεί τη μέθοδο *characters()*, ο οποίος καλείται κάθε φορά που συναντάται ένα σύνολο κειμένου, είναι ο ακόλουθος:

```
        public void characters(char[] ch, int start, int length)
                                                    throws SAXException {
            // Check if we have a parameter value
            if (lastElement.equals("PARAMETER")) {
                parameterValue = new String(ch).substring(start,
                                                        start + length);
            }

            // Check if we have output value
            if (lastElement.equals("OUTPUT")) {
                outputValue = new String(ch).substring(start,
                                                        start + length);
            }
        }
```

Στην περίπτωση που το XML στοιχείο που περιέχει το κείμενο είναι ένα στοιχείο τύπου PARAMETER ή OUTPUT αποθηκεύεται τοπικά το κείμενο για χρήση από την τελευταία μέθοδο.

Τέλος, ο κώδικας που υλοποιεί τη μέθοδο *endElement()*, η οποία καλείται όταν συναντάται το τέλος ενός XML στοιχείου, είναι ο ακόλουθος:

```
        public void endElement(String uri, String localName,
                               String qName) throws SAXException {
            // If we have a parameter add it in the parameters
            if (qName.equals("PARAMETER")) {
                parameters.setProperty(parameterName, parameterValue);
            }

            // If we have an output add it in the outputs
            if (qName.equals("OUTPUT")) {
                outputs.add(outputValue);
            }

            // If we have a reader add it and zeroset the parameters and
            // the outputs
            if (qName.equals("READER")) {
                readers.add(new ElementInfo(id, className, queueSize,
                                            parameters, outputs));
                parameters = new Properties();
                outputs = new ArrayList<String>();
            }

            // If we have a processor add it and zeroset the parameters
            // and the outputs
            if (qName.equals("PROCESSOR")) {
                processors.add(new ElementInfo(id, className, queueSize,
                                            parameters, outputs));
                parameters = new Properties();
                outputs = new ArrayList<String>();
            }

            // If we have a writer add it and zeroset the parameters and
            // the outputs
            if (qName.equals("WRITER")) {
```

```
                writers.add(new ElementInfo(id, className, queueSize,
                                                parameters, outputs));
                parameters = new Properties();
                outputs = new ArrayList<String>();
            }
        }
```

Αυτή η μέθοδος ελέγχει τι τύπου είναι το XML στοιχείο που τελειώνει και ανάλογα δημιουργεί αντικείμενα τύπου *ElementInfo* χρησιμοποιώντας τις τοπικές τιμές (οι οποίες έχουν οριστεί από τις άλλες δύο συναρτήσεις) και τα προσθέτει στις λίστες *readers*, *processors* και *writers*.

4.4. Πακέτο messenger

Στο πακέτο *messenger* περιέχονται όλες οι κλάσεις οι οποίες είναι σχετικές με την ενσωμάτωση του συστήματος σε άλλες εφαρμογές (σχήμα 3.4 του προηγούμενου κεφαλαίου).

4.4.1. Κλάση Message

Η κλάση *Message* αντιπροσωπεύει ένα μήνυμα από το σύστημα προς τις εφαρμογές που το ενσωματώνουν. Οι εφαρμογές αυτές είναι αρκετό να δέχονται μηνύματα κατά τη λήψη ενός εισερχόμενου μηνύματος από μία εφαρμογή εισόδου και κατά το πέρας της επεξεργασίας ενός μηνύματος. Οποιαδήποτε άλλη πληροφορία από το σύστημα είναι περιττή και μπορεί να αποκρυφτεί. Οι πληροφορίες που περιέχει η κλάση *Message* είναι τα δεδομένα του μηνύματος και η προέλευσή του, δηλαδή το στοιχείο του συστήματος που τα δημιούργησε. Ο κώδικας που υλοποιεί την κλάση είναι ο ακόλουθος:

```
public class Message {

    private String sourceName;
    private Data data;

    public Message(String sourceName, Data data) {
        this.sourceName = sourceName;
        this.data = data;
    }

    public String getSourceName() {
        return sourceName;
    }

    public Data getData() {
        return data;
    }
}
```

Όπως φαίνεται και στον κώδικα, η κλάση έχει δύο αντικείμενα μέλη, τα *sourceName* τύπου String

και data τύπου *Data*, στα οποία διατηρούνται τα δεδομένα και η προέλευση του μηνύματος. Οι τιμές για αυτά τα αντικείμενα ορίζονται κατά τη δημιουργία των *Message* αντικειμένων, μέσω ενός constructor ο οποίος δέχεται τις αντίστοιχες τιμές ως ορίσματα. Τέλος, οι δύο μέθοδοι *getSourceName()* και *getData()* επιστρέφουν την προέλευση και τα δεδομένα του μηνύματος αντίστοιχα.

Εδώ πρέπει να παρατηρηθεί ότι εάν τα αντικείμενα *Data* που μεταφέρονται στο μήνυμα έχουν κατασκευαστεί σύμφωνα με τους κανόνες που αναφέρθηκαν προηγουμένως (δηλαδή είναι immutable), τότε η κλάση *Message* θα είναι και αυτή immutable. Αυτή είναι και η σχεδίαση του συστήματος, οπότε δεν γίνονται αντιγραφές στις μεθόδους επιστροφής των δεδομένων της κλάσης.

4.4.2. Κλάση Messenger

Όπως αναφέρθηκε στη σχεδίαση του συστήματος, η επικοινωνία με τις εφαρμογές οι οποίες θα ενσωματώνουν το σύστημα θα γίνεται μέσω μιας κλάσης singleton. Η κλάση αυτή είναι η *Messenger* και χαρακτηρίζεται από το ότι δεν μπορούν να δημιουργηθούν αντικείμενα αυτού του τύπου. Ο κώδικας που την υλοποιεί είναι ο ακόλουθος:

```
public class Messenger {

    private static LinkedBlockingQueue<Message> queue =
            new LinkedBlockingQueue<Message>();

    private Messenger() {
    }

    public static void putMessage(Message message) throws
                                                NullPointerException {
        queue.offer(message);
    }

    public static Message getMessage() throws InterruptedException {
        return queue.take();
    }
}
```

Η κλάση αυτή έχει ένα αντικείμενο μέλος, την ουρά *queue* στην οποία διατηρούνται τα μηνύματα για τα οποία πρέπει να ενημερωθεί η εφαρμογή που ενσωματώνει το σύστημα. Το αντικείμενο αυτό είναι τύπου *static*, το οποίο σημαίνει ότι την πρώτη φορά που χρησιμοποιείται η κλάση *Messenger* δημιουργείται το αντικείμενο και τις επόμενες φορές χρησιμοποιείται το ίδιο. Ο κενός constructor τύπου *private* μετατρέπει την κλάση σε singleton, αφού απαγορεύεται η πρόσβαση σε αυτόν έξω από την κλάση. Τέλος, οι δύο μέθοδοι *putMessage()* και *getMessage()* επιτρέπουν την εισαγωγή και την εξαγωγή μηνυμάτων από την ουρά.

Κατά τη διάρκεια της εκτέλεσης του συστήματος ως ενσωματωμένο σε κάποια εφαρμογή, η μέθοδος *putMessage()* καλείται από το σύστημα και η ουρά ενημερώνεται με τα κατάλληλα μηνύματα. Η εφαρμογή πρέπει να καλεί τη μέθοδο *getMessage()* και να παίρνει τα μηνύματα για περαιτέρω επεξεργασία. Στην περίπτωση που δεν υπάρχει κανένα μήνυμα στην ουρά, η μέθοδος δεν θα επιστρέφει, αλλά θα περιμένει το σύστημα να εισάγει κάποιο μήνυμα, οπότε και θα το επιστρέφει.

4.4.3. Κλάση MessengerWriter

Η ενημέρωση του singleton *Messenger* θα γίνεται μέσω κάποιων *Writers* οι οποίοι θα δέχονται τις εξόδους από τους *Readers* και τους *Processors* του συστήματος. Η κλάση που υλοποιεί τον συγκεκριμένο *Writer* είναι η *MessengerWriter*. Για την υλοποίησή της ακολουθούνται οι κανόνες που αναφέρθηκαν στην ενότητα που αναλύει την κλάση *Writer*.

```
public class MessengerWriter extends Writer {

    private String sourceName = null;

    public void run() {
        // Start the infinite loop
        while (true) {
            // get the data from the queue
            Data data = queue.getOldestData();

            // Check that the queue wasn't empty. If it was skip the
            // rest of the loop and try to read again from the queue.
            if (data == null) {
                continue;
            }

            // Write the data to the messanger
            Messenger.putMessage(new Message(sourceName, data));
        }
    }

    public void setParameters(Properties parameters) throws
                                            ParameterProblemException {
        // Set the source name
        sourceName = parameters.getProperty("sourceName");
        if (sourceName == null) {
            throw new ParameterProblemException("Parameter sourceName is
                                                    not set");
        }
    }

    public String getDescription() {
        return "Writes the input data to the Messenger for the element "
                                                        + sourceName;
    }
}
```

Η κλάση *MessengerWriter* έχει μία παράμετρο, το αναγνωριστικό του στοιχείου του συστήματος για το οποίο ενημερώνει το singleton. Η παράμετρος αυτή αποθηκεύεται τοπικά στο αλφαριθμητικό *sourceName* και παίρνει την τιμή της μέσω της μεθόδου *setParameters()*. Στο άπειρο loop της μεθόδου *run()*, αρχικώς εξάγεται το τελευταίο αντικείμενο *Data* από την ουρά, ελέγχεται αν η ουρά δεν ήταν άδεια και τελικώς ενημερώνεται το singleton με ένα αντικείμενο τύπου *Message*. Η υλοποίηση της κλάσης τελειώνει με τη μέθοδο *getDescription()* η οποία επιστρέφει ένα αλφαριθμητικό που εξηγεί για ποιο στοιχείο του συστήματος ενημερώνεται το singleton.

4.5. Πακέτο miodzio

Στο πακέτο αυτό ανήκει μόνο μία κλάση, η *Miodzio*, η οποία είναι η κεντρική κλάση του συστήματος (σχήμα 3.6 του προηγούμενου κεφαλαίου). Ο σκελετός της κλάσης είναι ο ακόλουθος:

```
public class Miodzio {

    private Map<String, Writer> writers = new HashMap<String, Writer>();
    private Map<String, Reader> readers = new HashMap<String, Reader>();
    private Map<String, Processor> processors = new HashMap<String,
                                                          Processor>();

    public Miodzio(File xmlFile, boolean embedded) throws
                        ClassNotFoundException, InstantiationException,
                    IllegalAccessException, SAXException, IOException,
              ParserConfigurationException, ParameterProblemException {
        ...
    }

    public void start() {
        ...
    }

    public static void main(String[] args) {
        ....
    }
}
```

Η κλάση περιέχει τέσσερα μέλη τύπου *Map* (τα οποία λειτουργούν σαν λίστες που διατηρούν ζευγάρια ονομάτων-τιμών), στα οποία αποθηκεύονται τοπικά τα στοιχεία του συστήματος. Κατά την αρχικοποίηση της κλάσης δημιουργούνται και συνδέονται αυτά τα στοιχεία σύμφωνα με τις πληροφορίες που περιέχονται σε ένα XML αρχείο και με τη συνάρτηση *start()* ξεκινάει η λειτουργία τους.

4.5.1. Constructor της κλάσης

Όπως αναφέρθηκε προηγουμένως κατά την αρχικοποίηση της κλάσης εξάγονται οι πληροφορίες από ένα XML αρχείο. Αυτό γίνεται μέσω ενός αντικειμένου *XmlParser*, το οποίο αναλύθηκε νωρίτερα, το οποίο διαβάζει το αρχείο και εξάγει όλες τις χρήσιμες πληροφορίες από αυτό.

```
XmlParser configParser = new XmlParser(xmlFile);
```

Μετά τη δημιουργία του αντικειμένου *configParser*, το τελευταίο περιέχει όλες τις πληροφορίες για τα στοιχεία του συστήματος. Με αυτές τις πληροφορίες, αρχικώς το σύστημα κατασκευάζει όλα τα στοιχεία που το αποτελούν.

```
for (ElementInfo config : configParser.getReaders()) {
    // Create the reader
    Reader reader = (Reader)
                    (Class.forName(config.getClassName()).newInstance());
    // Set the readers parameters
    reader.setParameters(config.getParameters());
    // Add the reader to the readers map
    readers.put(config.getId(), reader);
}

// Go through the processors and create them
for (ElementInfo config : configParser.getProcessors()) {
    // Create the processor
    Processor processor = (Processor)
                    (Class.forName(config.getClassName()).newInstance());
    // Set the waiting list maximum size
    processor.getQueue().setQueueSize(config.getQueueSize());
    // Set the processors parameters
    processor.setParameters(config.getParameters());
    // Add the processor to the processors map
    processors.put(config.getId(), processor);
}

// Go through the writers and create them
for (ElementInfo config : configParser.getWriters()) {
    // Create the writer
    Writer writer = (Writer)
                    (Class.forName(config.getClassName()).newInstance());
    // Set the waiting list maximum size
    writer.getQueue().setQueueSize(config.getQueueSize());
    // Set the writers parameters
    writer.setParameters(config.getParameters());
    // Add the writer to the writers map
    writers.put(config.getId(), writer);
}
```

Όπως φαίνεται και στον κώδικα, για κάθε λίστα που επιστρέφουν οι μέθοδοι *getReaders()*, *getProcessors()* και *getWriters()*, τα *ElementInfo* αντικείμενα χρησιμοποιούνται για την κατασκευή των στοιχείων του συστήματος. Η αρχικοποίηση των στοιχείων γίνεται μέσω της μεθόδου

forName() της κλάσης *Class*. Αυτός ο τρόπος αρχικοποίησης χρησιμοποιείται όταν το java όνομα της κλάσης που θα χρησιμοποιηθεί δεν είναι γνωστό πριν την εκτέλεση του συστήματος, όπως στην συγκεκριμένη περίπτωση. Η μέθοδος *forName()* θα ψάξει όλες τις κλάσεις ου υπάρχουν στο classpath της Java και θα επιστρέψει την ανάλογη, οπότε ο χρήστης του συστήματος πρέπει απλώς να εισάγει τις κλάσεις του στο classpath πριν εκτελέσει το σύστημα.

Αφού αρχικοποιηθούν τα αντικείμενα που αντιπροσωπεύουν τα στοιχεία του συστήματος, θέτονται ανάλογα οι παράμετροί τους, καθώς και το μέγεθος της ουράς για τους processors και τους writers, και προστίθενται στις τοπικές λίστες, απ' όπου θα χρησιμοποιηθούν κατά την υπόλοιπη εκτέλεση του συστήματος.

Μόλις αυτή η διαδικασία τελειώσει, το σύστημα συνδέει τα στοιχεία μεταξύ τους, όπως φαίνεται στον παρακάτω κώδικα. Για κάθε reader και processor ελέγχονται οι λίστες των processors και των writers και αν βρεθεί κάποιο ζευγάρι που πρέπει να συνδεθεί (σύμφωνα με τις πληροφορίες από το XML αρχείο), η ουρά τύπου *DataEventsQueue* του προορισμού θέτεται να ενημερώνεται για τα συγκεκριμένα συμβάντα.

```
for (ElementInfo reader : configParser.getReaders()) {
    // Get the reader ID
    String readerId = reader.getId();
    // Go through the outputs and connect
    for (String output : reader.getOutputs()) {
        // Check if the output is a processor
        Processor outputprocessor = processors.get(output);
        if (outputprocessor != null) {
            readers.get(readerId).addDataEventListener(
                                            outputprocessor.getQueue());
        }
        // Check if the output is a writer
        Writer outputWriter = writers.get(output);
        if (outputWriter != null) {
            readers.get(readerId).addDataEventListener(
                                               outputWriter.getQueue());
        }
    }
}

for (ElementInfo processor : configParser.getProcessors()) {
    // Get the processor ID
    String processorId = processor.getId();
    // Go through the outputs and connect
    for (String output : processor.getOutputs()) {
        // Check if the output is a processor
        Processor outputprocessor = processors.get(output);
        if (outputprocessor != null) {
            processors.get(processorId).addDataEventListener(
                                            outputprocessor.getQueue());
        }
```

```
        // Check if the output is a writer
        Writer outputWriter = writers.get(output);
        if (outputWriter != null) {
            processors.get(processorId).addDataEventListener(
                                                outputWriter.getQueue());
        }
    }
}
```

Όταν όλα τα στοιχεία του συστήματος έχουν δημιουργηθεί και έχει τελειώσει η σύνδεσή τους, εάν το σύστημα λειτουργεί ως ενσωματωμένο σε κάποια εφαρμογή, δημιουργούνται για κάθε reader και processor αντικείμενα τύπου *MessengerWriter* και συνδέονται με αυτά.

```
if (embedded) {
    // First go through the readers
    for (ElementInfo reader : configParser.getReaders()) {
        // Get the reader ID
        String readerId = reader.getId();

        // Create the MessengerWriter and set it up
        MessengerWriter writer = new MessengerWriter();
        Properties parameters = new Properties();
        parameters.setProperty("sourceName", readerId);
        writer.setParameters(parameters);

        // Set the writer as an output of the reader
        readers.get(readerId).addDataEventListener(writer.getQueue());

        // Start the writer
        new Thread(writer).start();
    }

    // Go through the processors
    for (ElementInfo processor : configParser.getProcessors()) {
        // Get the processor ID
        String processorId = processor.getId();

        // Create tne MessengerWriter and set it up
        MessengerWriter writer = new MessengerWriter();
        Properties parameters = new Properties();
        parameters.setProperty("sourceName", processorId);
        writer.setParameters(parameters);

        // Set the writer as an output of the processor
        processors.get(processorId).addDataEventListener(
                                                    writer.getQueue());

        // Start the writer
        new Thread(writer).start();
    }
}
```

Όπως φαίνεται στον κώδικα, η κατασκευή του κάθε *MessengerWriter* γίνεται με την αρχικοποίησή

του και τον ορισμό μιας παραμέτρου με το όνομα του στοιχείου από το οποίο δέχεται μηνύματα. Η σύνδεσή του με το στοιχείο γίνεται με τον ίδιο τρόπο όπως οι συνδέσεις των στοιχείων του συστήματος μεταξύ του. Αφού συνδεθεί, δημιουργείται ένα νέο νήμα εργασίας στο οποίο εκτελείται ο *MessengerWriter*.

4.6. Πακέτο gui

Στο πακέτο *gui* περιλαμβάνονται όλες οι κλάσεις οι οποίες είναι σχετικές με τις διεπαφές του χρήστη (σχήμα 3.5 του προηγούμενου κεφαλαίου). Η περιγραφή της κατασκευής αυτών των κλάσεων διαφέρει λίγο από τις προηγούμενες διότι χρησιμοποιήθηκε το WYSIWYG (What You See Is What You Get) εργαλείο της πλατφόρμας Netbeans. Γι' αυτό το λόγο ο κώδικας που δημιουργήθηκε αυτόματα δεν θα παρουσιαστεί (λόγω του ιδιαίτερα μεγάλου όγκου του), αλλά θα περιγραφούν τα Swing στοιχεία που χρησιμοποιήθηκαν και η παραμετροποίησή τους.

4.6.1. Κλάση ElementPanel

Η κλάση *ElementPanel* είναι ένα *Jpanel* το οποίο χρησιμοποιείται για τη σχεδίαση του κάθε στοιχείου του συστήματος. Περιέχει δύο Swing αντικείμενα, δύο *Jlabels*, τα οποία δείχνουν το όνομα του στοιχείου και την τιμή της εξόδου του.

Σχήμα 4.1: ElementPanel

Για να είναι σωστή η σχεδίαση του panel έχει οριστεί το χρώμα του φόντου στο άσπρο και το μέγεθός του στα 250X50 pixel. Το τελευταίο γίνεται ορίζοντας τις παραμέτρους του panel maximumSize, minimumSize και preferredSize όλες σε αυτήν την τιμή. Το σταθερό μέγεθος του panel είναι ιδιαίτερα σημαντικό για τον τρόπο με τον οποίο θα υλοποιηθεί η μέθοδος *print()*.

Το label το οποίο παρουσιάζει το όνομα του στοιχείου έχει το όνομα *nameLabel* και είναι το αριστερό στο σχήμα. Για να εμφανίζεται όπως στο σχήμα έχει οριστεί η γραμματοσειρά σε Dialog 12 Bold, η στοίχιση του κειμένου στο κέντρο και το χρώμα του φόντου στο RGB=(245,245,245) το οποίο είναι μία γκρι απόχρωση. Το χρώμα του φόντου θα μεταβάλλεται κατά τη διάρκεια της εκτέλεσης του συστήματος σύμφωνα με τις προδιαγραφές ΠΔΧ-3, ΠΔΧ-5 και ΠΔΧ-6. Το κείμενο του label το οποίο φαίνεται στο σχήμα είναι εικονικό και το πραγματικό ορίζεται κατά τη διάρκεια

της αρχικοποίησης του συστήματος, όταν δηλαδή είναι διαθέσιμα τα ονόματα των στοιχείων από το XML αρχείο.

Το label το οποίο παρουσιάζει την έξοδο του στοιχείου έχει το όνομα *valueLabel* και έχει οριστεί η γραμματοσειρά του σε Dialog 11 Plain. Το κείμενο αυτού του label θα ενημερώνεται κατά τη διάρκεια εκτέλεσης του συστήματος με τις ανάλογες τιμές μέσω του singleton *Messenger*.

Το χρώμα του φόντου, όπως αναφέρθηκε προηγουμένως θα μεταβάλλεται σύμφωνα με τις επιλογές του χρήστη. Οι δυνατές καταστάσεις είναι τέσσερις. Ένα στοιχείο μπορεί να είναι επιλεγμένο, μπορεί να στέλνει μηνύματα στο επιλεγμένο στοιχείο, να δέχεται μηνύματα από αυτό ή να μην έχει καμία σχέση με το επιλεγμένο. Γι' αυτό το λόγο έχουν οριστεί τέσσερις σταθερές οι οποίες αντιπροσωπεύουν τις καταστάσεις του συστήματος, καθώς και τέσσερις σταθερές με τα χρώματα τα οποία θα χρησιμοποιηθούν. Ο ορισμός της κατάστασης του στοιχείου γίνεται μέσω της μεθόδου *setState()* η οποία ρυθμίζει ανάλογα τον φόντο του στοιχείου.

```
    private static final Color COLOR_NORMAL =
                                        new Color(245, 245, 245);
    private static final Color COLOR_SELECTED =
                                        new Color(180, 245, 245);
    private static final Color COLOR_INPUT =
                                        new Color(245, 180, 180);
    private static final Color COLOR_OUTPUT =
                                        new Color(180, 245, 180);
    public static final int STATE_NORMAL = 0;
    public static final int STATE_SELECTED = 1;
    public static final int STATE_INPUT = 2;
    public static final int STATE_OUTPUT = 3;

    public void setState(int state) {
        switch (state) {
            case STATE_NORMAL:
                namePanel.setBackground(COLOR_NORMAL);
                break;
            case STATE_SELECTED:
                namePanel.setBackground(COLOR_SELECTED);
                break;
            case STATE_INPUT:
                namePanel.setBackground(COLOR_INPUT);
                break;
            case STATE_OUTPUT:
                namePanel.setBackground(COLOR_OUTPUT);
                break;
        }
    }
```

Για τη διαχείριση του ονόματος του στοιχείου παρέχονται δύο μέθοδοι, οι *getName()* και *setName()*. Οι μέθοδοι αυτές ελέγχουν το κείμενο του label *nameLabel*.

```
    public String getName() {
        return nameLabel.getText();
    }

    public void setName(String name) {
        nameLabel.setText(name);
    }
```

Με παρόμοιο τρόπο γίνεται και η διαχείριση της τιμής εξόδου του στοιχείου, μέσω των μεθόδων *getValue()* και *setValue()*, οι οποίες ελέγχουν το κείμενο του label *valueLabel*.

```
    public String getValue() {
        return valueLabel.getText();
    }

    public void setValue(String value) {
        valueLabel.setText(value);
    }
```

Σύμφωνα με τη προδιαγραφή ΠΔΧ-8, ο χρήστης πρέπει να βλέπει περιγραφές των στοιχείων του συστήματος. Γι' αυτό το λόγο η κλάση *ElementPanel* διατηρεί μία μεταβλητή τύπου String, την *description*, στην οποία αποθηκεύεται η περιγραφή του στοιχείου. Η διαχείριση της μεταβλητής γίνεται μέσω των μεθόδων *getDescription()* και *setDescription()*.

```
    private String description = null;

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }
```

Επειδή τα στοιχεία του συστήματος πρέπει να συνδεθούν μεταξύ τους με γραμμές, θα υλοποιηθεί και η μέθοδος *paint()*, στην οποία θα ζωγραφιστούν οι γραμμές εισόδου και εξόδου του συστήματος. Οι γραμμές αυτές θα είναι δύο ή τρεις ανάλογα με την περίπτωση. Η πρώτη γραμμή θα ενώνει την αριστερή μεριά του *nameLabel* με την αριστερή μεριά του panel και είναι η γραμμή εισόδου του στοιχείου. Η δεύτερη γραμμή θα ενώνει τη δεξιά μεριά του *nameLabel* με τη δεξιά μεριά του panel και είναι η γραμμή εξόδου του στοιχείου. Η τρίτη γραμμή θα χρησιμοποιείται μόνο σε στοιχεία τύπου processor και στην περίπτωση που κάποια έξοδος είναι πάλι τύπου processor (οι έξοδοι τύπου writer θα συνδέονται στη δεύτερη γραμμή) και θα ενώνει τη δεξιά μεριά του *nameLabel* με την αριστερή μεριά του panel. Η πληροφορία για το αν χρειάζεται η τρίτη γραμμή ή όχι θα ορίζεται με τη μέθοδο *setOutputBack()* και θα αποθηκεύεται τοπικά στη μεταβλητή *outputBack*.

```
    private boolean outputBack;

    public boolean isOutputBack() {
        return this.outputBack;
    }

    public void setOutputBack(boolean outputBack) {
        this.outputBack = outputBack;
    }

    public void paint(Graphics g) {
        super.paint(g);

        g.setColor(new Color(51, 51, 51));
        g.drawLine(0, 25, 12, 25);
        g.drawLine(142, 25, 250, 25);

        if (this.isOutputBack()) {
            g.drawLine(150, 25, 150, 45);
            g.drawLine(150, 45, 0, 45);
        }
    }
```

Εδώ πρέπει να σημειωθεί η κλήση της συνάρτησης *super.paint()* στην αρχή της συνάρτησης *paint()*. Αυτή η κλήση θα σχεδιάσει τα περιεχόμενα του panel, δηλαδή τα *nameLabel* και *valueLabel* στη σωστή τους θέση πριν σχεδιαστούν οι γραμμές.

4.6.2. Κλάση DrawingPanel

Η κλάση *DrawingPanel* είναι μία κλάση τύπου *Jpanel* στην οποία θα παρουσιάζονται όλα τα στοιχεία του συστήματος, καθώς και οι συνδέσεις μεταξύ τους. Το panel αυτό θα περιέχει τρία άλλα panels, τα *readersPanel*, *processorsPanel* και *writersPanel*, τα οποία θα δείχνουν τα αντίστοιχα στοιχεία του συστήματος. Η διάταξή τους θα είναι το ένα δίπλα στο άλλο από αριστερά προς τα δεξιά, με τη σειρά που αναφέρθηκαν. Τα στοιχεία του συστήματος θα παρουσιάζονται στα panel το ένα κάτω από το άλλο, με τη σειρά που θα εισάγονται σε αυτά. Για να γίνει αυτό ορίστηκε το layout τους ως *BoxLayout*, το οποίο ρυθμίστηκε να στοιβάζει τα αντικείμενα που περιέχει το panel στον Y άξονα (δηλαδή κατακόρυφα). Επίσης ορίστηκε το χρώμα του φόντου να είναι λευκό.

Επειδή τα τρία αυτά panels θα έχουν συγκεκριμένο μέγεθος στον X άξονα (το μέγεθος του *ElementPanel*, δηλαδή 250 pixels) για να είναι δυνατή η μεταβολή του μεγέθους του *DrawingPanel* εισήχθησαν ανάμεσά τους δύο άλλα panels, τα *spacePanel1* και *spacePanel2*, τα οποία δεν θα περιέχουν τίποτα, αλλά ορίστηκαν να μεταβάλλεται το μέγεθός τους για να καλύπτει τους κενούς χώρους. Με αυτόν τον τρόπο η αριστερή μεριά του *readersPanel* θα είναι η ίδια με την αριστερή μεριά του *DrawingPanel*, το *processorsPanel* θα είναι πάντα ακριβώς στη μέση και η δεξιά μεριά του *writersPanel* θα είναι η ίδια με τη δεξιά μεριά του *DrawingPanel*. Αυτή η διάταξη

είναι ιδιαίτερα σημαντική κατά τη σχεδίαση των συνδέσεων μεταξύ των στοιχείων.

Για την αρχικοποίησή της, η κλάση *DrawingPanel* παρέχει μία μέθοδο, την *initialize()*. Η μέθοδος αυτή δέχεται ως παραμέτρους τρεις λίστες οι οποίες περιέχουν τα *ElementPanels* για τους readers, processors και writers, καθώς και τέσσερις λίστες με τις συνδέσεις των readers με τους processors, των readers με τους writers, των processors με τους writers και των processors με τους processors.. Οι τελευταίες λίστες είναι λίστες οι οποίες περιέχουν λίστες από αντικείμενα *Integer*. Για παράδειγμα, αν το δεύτερο στοιχείο της λίστας συνδέσεων των readers με τους processors είναι η λίστα [3,6], τότε ο δεύτερος reader συνδέεται με τον τρίτο και τον έκτο processor.

```
    List<List<Integer>> readerToProcessorOutputs = null;
    List<List<Integer>> readerToWriterOutputs = null;
    List<List<Integer>> processorToWriterOutputs = null;
    List<List<Integer>> processorToProcessorOutputs = null;

    public void initialize(List<ElementPanel> readers,
                           List<ElementPanel> processors,
                           List<ElementPanel> writers,
                           List<List<Integer>> readersOutputs,
                           List<List<Integer>> readerToWriterOutputs,
                           List<List<Integer>> processorsOutputs,
                       List<List<Integer>> processorToprocessorOutputs) {

        if (readers != null) {
            for (ElementPanel reader : readers) {
                readersPanel.add(reader);
            }
        }

        if (processors != null) {
            for (ElementPanel processor : processors) {
                processorsPanel.add(processor);
            }
        }

        if (writers != null) {
            for (ElementPanel writer : writers) {
                writersPanel.add(writer);
            }
        }

        this.readerToProcessorOutputs = readersOutputs;
        this.readerToWriterOutputs = readerToWriterOutputs;
        this.processorToWriterOutputs = processorsOutputs;
        this.processorToProcessorOutputs = processorToprocessorOutputs;
    }
```

Όπως φαίνεται και στον κώδικα, η μέθοδος *initialize()* αρχικά εισάγει στα τρία panels τους readers, τους processors και τους writers και μετά αποθηκεύει τοπικά τις συνδέσεις μεταξύ τους σε τέσσερις τοπικές μεταβλητές για χρήση στη μέθοδο paint(). Η μέθοδος αυτή αρχικά καλεί τη

συνάρτηση *super.paint()* και σχεδιάζονται όλα τα στοιχεία του συστήματος. Αφού σχεδιαστούν τα στοιχεία η μέθοδος σχεδιάζει τις συνδέσεις μεταξύ τους.

Αρχικά σχεδιάζονται οι συνδέσεις μεταξύ των writers και των processors. Επειδή για όλες τις συνδέσεις οι αρχικές και τελικές συντεταγμένες στον άξονα X είναι σταθερές, αυτές υπολογίζονται ξεχωριστά μόνο μία φορά. Για τον υπολογισμό τους χρησιμοποιείται η συνάρτηση *getLocation()* του *Jpanel*, η οποία επιστρέφει τις συντεταγμένες της πάνω αριστερής γωνίας του panel. Η αρχική X συντεταγμένη υπολογίζεται από τον τύπο:

αρχικό X = X συντεταγμένη του *readersPanel* + μήκος του *readersPanel*

και η τελική X συντεταγμένη από τον τύπο:

τελικό X = X συντεταγμένη του *processorsPanel*

Αφού υπολογιστούν αυτές οι δύο τιμές, για κάθε σύνδεση που πρέπει να γίνει υπολογίζονται οι Y συντεταγμένες της εξόδου του reader και της εισόδου του processor χρησιμοποιώντας το γεγονός ότι η Y διάσταση του *ElementPanel* είναι 50 pixel και τις θέσεις στις οποίες σχεδιάστηκαν οι είσοδοι και οι έξοδοι στην *paint()* μέθοδο του *ElementPanel*. Από αυτά προκύπτει ότι η αρχική Y συντεταγμένη υπολογίζεται από τον τύπο:

αρχικό Y = πλήθος προηγούμενων readers * 50 + 25

και η τελική Y συντεταγμένη από τον τύπο:

τελικό Y = πλήθος προηγούμενων processors * 50 + 25

Τα πλήθη των προηγούμενων στοιχείων είναι εύκολο να βρεθούν, αφού ξέρουμε τη θέση του συγκεκριμένου στοιχείου στη λίστα.

```
// the starting and ending X
int startX = (int)(readersPanel.getLocation().getX() +
                readersPanel.getSize().getWidth());
int endX = (int)(processorsPanel.getLocation().getX());

// Go through the reader outputs and paint the lines
if (readerToProcessorOutputs != null) {
    for (int i = 0; i < readerToProcessorOutputs.size(); i++) {
       for (int j = 0; j < readerToProcessorOutputs.get(i).size(); j++){
           g.drawLine(startX, i*50 + 25, endX,
                       readerToProcessorOutputs.get(i).get(j)*50 + 25);
       }
    }
}
```

Αμέσως μετά γίνονται οι συνδέσεις των readers με τους writers. Αυτή η περίπτωση είναι λίγο πιο

περίπλοκη, επειδή υπάρχει περίπτωση να υπάρχει ένας processor ανάμεσα στα δύο στοιχεία που συνδέονται. Αυτό το πρόβλημα λύνεται σχεδιάζοντας μία οριζόντια παράκαμψη 5 pixel πάνω από τον processor που βρίσκεται ακριβώς δεξιά του reader και συνεχίζοντας τη σύνδεση από το μεσαίο σημείο του κενού panel ανάμεσα στα *processorsPanel* και *readersPanel*. Ο υπολογισμός της X συντεταγμένης αυτού του σημείου γίνεται με τον τύπο:

X = X συντεταγμένη του *processorsPanel* + μήκος του *processorsPanel* +
+1/2 * μήκος κενού panel

όπου το μήκος του κενού panel υπολογίζεται από τον τύπο:

μήκος κενού panel = (μήκος *DrawingPanel* – μήκος *readersPanel* – μήκος *processorsPanel* -
μήκος *writersPanel*) / 4

```
// the starting, middle and ending X
startX = (int)(readersPanel.getLocation().getX() +
                                          readersPanel.getWidth());
int middleX = (int)(processorsPanel.getLocation().getX() +
                processorsPanel.getWidth() + ((this.getWidth() -
                readersPanel.getWidth() - processorsPanel.getWidth() -
                writersPanel.getWidth()) / 4));
endX = (int)(writersPanel.getLocation().getX());

if (readerToWriterOutputs != null) {
    for (int i = 0; i < readerToWriterOutputs.size(); i++) {
         for (int j = 0; j < readerToWriterOutputs.get(i).size(); j++) {
              g.drawLine(startX, i*50 + 25, startX, i*50 + 5);
              g.drawLine(startX, i*50 + 5, middleX, i*50 + 5);
              g.drawLine(middleX, i*50 + 5,
              endX, readerToWriterOutputs.get(i).get(j)*50 + 25);
         }
    }
}
```

Με τον ίδιο τρόπο σχεδιάζονται και οι συνδέσεις μεταξύ των processors και των writers και των processors μεταξύ τους, όπως φαίνεται στον παρακάτω κώδικα. Περαιτέρω επεξήγηση του κώδικα είναι περιττή , αφού οι υπολογισμοί είναι οι ίδιοι που ήδη έχουν αναφερθεί.

```
// Draw the lines between the processors and the writers
// the starting and ending X
startX = (int)(processorsPanel.getLocation().getX() +
                                    processorsPanel.getSize().getWidth());
endX = (int)(writersPanel.getLocation().getX());

// Go through the processor outuputs and paint the lines
if (processorToWriterOutputs != null) {
    for (int i = 0; i < processorToWriterOutputs.size(); i++) {
       for (int j = 0; j < processorToWriterOutputs.get(i).size(); j++){
           g.drawLine(startX, i*50 + 25, endX,
```

```
                        processorToWriterOutputs.get(i).get(j)*50 + 25);
        }
    }
}

// Draw the lines between the processors and the processors
// the starting and ending X
startX =(int)(processorsPanel.getLocation().getX() - ((this.getWidth() -
                readersPanel.getWidth() - processorsPanel.getWidth() -
                writersPanel.getWidth()) / 4));
endX = (int)(processorsPanel.getLocation().getX());

if (processorToProcessorOutputs != null) {
    for (int i = 0; i < processorToProcessorOutputs.size(); i++) {
        for (int j = 0; j < processorToProcessorOutputs.get(i).size();
                                                                j++){
            g.drawLine(startX, i*50 + 45, endX, i*50 + 45);
            g.drawLine(startX, i*50 + 45, endX,
                    processorToProcessorOutputs.get(i).get(j)*50 + 25);
        }
    }
}
```

4.6.3. *Κλάση ValueUpdater*

Η κλάση *ValueUpdater* είναι η κλάση η οποία επικοινωνεί με το singleton *Messenger* και ενημερώνει ανάλογα τις διεπαφές του χρήστη. Κατά τη κατασκευή της δέχεται ένα αντικείμενο *Map* το οποίο αποθηκεύει τοπικά στη μεταβλητή *elements* το οποίο περιέχει τα αντικείμενα ElementPanel για όλα τα στοιχεία του συστήματος, και ένα αντικείμενο τύπου *JtextArea* το οποίο θα ενημερώνεται εάν η τιμή του επιλεγμένου στοιχείου μεταβληθεί. Η κλάση αυτή κληρονομεί την κλάση *Thread* και εκτελείται σε ένα δικό της νήμα εκτέλεσης.

```
public class ValueUpdater extends Thread {

    private Map<String, ElementPanel> elements;
    private JTextArea outputTextArea = null;

    public ValueUpdater(Map<String, ElementPanel> elements,
                                            JTextArea outputTextArea) {
        if (elements == null) {
            throw new IllegalArgumentException("Parameter elements
                                                    cannot be null");
        }
        this.elements = elements;

        if (outputTextArea == null) {
            throw new IllegalArgumentException("Parameter outputTextArea
                                                  cannot be null");
        }
        this.outputTextArea = outputTextArea;
    }
```

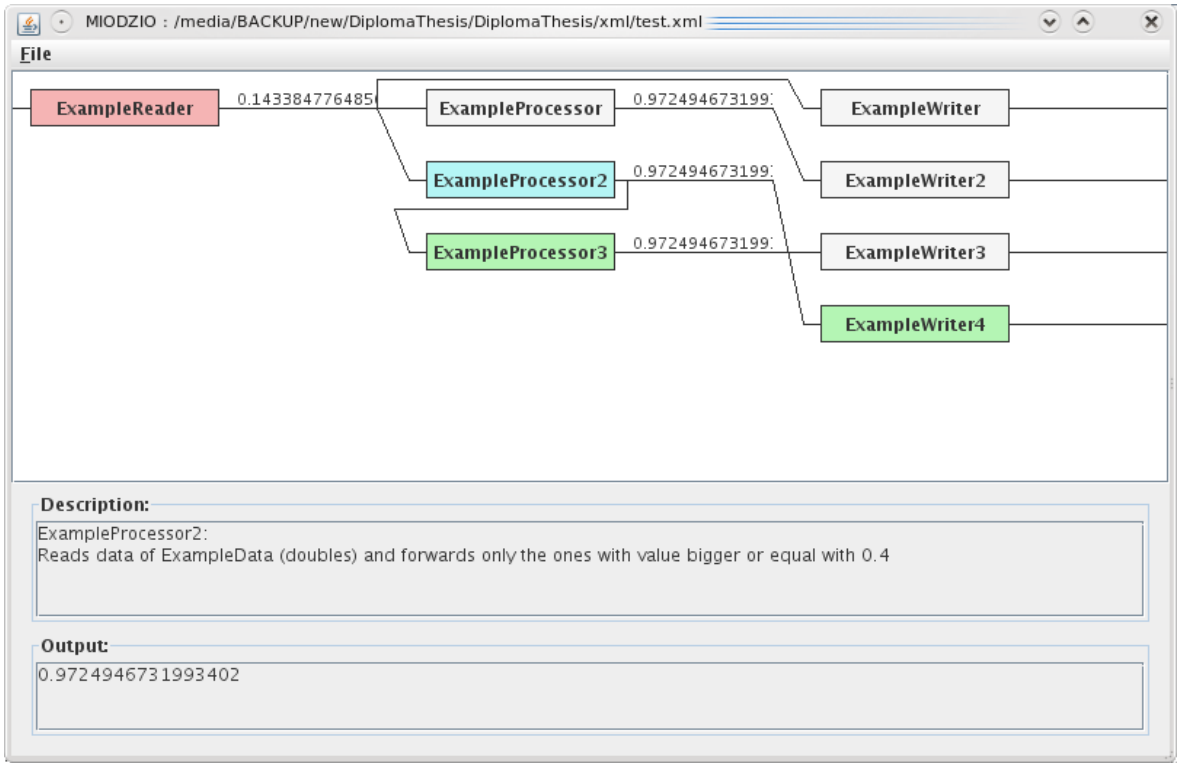
```
    public void run() {
        while (true) {
            Message message = null;
            try {
                message = Messenger.getMessage();
            } catch (InterruptedException ex) {
                continue;
            }
            if (message != null) {
                String elementName = message.getSourceName();
                String elementValue = message.getData().toString();
                elements.get(elementName).setValue(elementValue);
                if (elementName.equals(selectedElement)) {
                    outputTextArea.setText(elementValue);
                }
            }
        }
    }

    private String selectedElement = null;

    public void setSelectedElement(String selectedElement) {
        this.selectedElement = selectedElement;
    }
}
```

Η μεταβλητή *selectedElement* διατηρεί το όνομα του στοιχείου το οποίο είναι διαλεγμένο και θέτεται μέσω της συνάρτησης *setSelectedElement()*. Όπως φαίνεται και στον κώδικα η λειτουργία της κλάσης είναι αρκετά απλή. Για κάθε μήνυμα που δέχεται από τον *Messenger* ενημερώνει το κατάλληλο στοιχείο και, εάν αυτό είναι το επιλεγμένο, ενημερώνει και το αντικείμενο JtextArea.

4.6.4. Κλάση MiodzioGui

Η κλάση αυτή είναι η βασική κλάση των διεπαφών του χρήστη και χρησιμοποιείται για τη σύνδεση όλων των προηγούμενων. Τα Swing στοιχεία που αποτελούν την κλάση είναι ένα μενού (το οποίο απλώς περιέχει την επιλογή Exit και δεν θα γίνει παραπέρα αναφορά σε αυτό), ένα αντικείμενο DrawingPanel και δύο J*textAreas*, μία η οποία παρουσιάζει την περιγραφή του επιλεγμένου αντικειμένου και μία η οποία παρουσιάζει την τιμή της εξόδου του. Τα ανωτέρω φαίνονται στο σχήμα 4.2.

Όπως φαίνεται και στο σχήμα, το επιλεγμένο στοιχείο εμφανίζεται μπλε, ενώ το στοιχείο που συνδέεται μαζί του ως είσοδος είναι κόκκινο και τα στοιχεία που συνδέονται μαζί του ως έξοδος είναι πράσινα. Στην περιοχή κειμένου για την περιγραφή εμφανίζεται μια σύντομη περιγραφή της ενέργειας που εκτελεί το επιλεγμένο στοιχείο και στην περιοχή κειμένου για την τιμή εμφανίζεται η τιμή εξόδου του. Επίσης στο διάγραμμα εμφανίζονται οι έξοδοι όλων των στοιχείων του συστήματος.



Σχήμα 4.2: MiodzioGui

Κατά την αρχικοποίηση της κλάσης, μέσω της χρήσης της κλάσης *XmlParser* δημιουργούνται τοπικές λίστες με τις πληροφορίες από το XML αρχείο, ακριβώς ίδιες με αυτές που παρουσιάστηκαν στην κλάση *DrawingPanel*.

```
    private List<ElementPanel> readersElements;
    private List<ElementPanel> processorsElements;
    private List<ElementPanel> writersElements;
    private List<List<Integer>> readersOutputs;
    private List<List<Integer>> readerToWriterOutputs;
    private List<List<Integer>> processorsOutputs;
    private List<List<Integer>> processorToProcessorOutputs;
```

Η δημιουργία των *readersElements*, *processorsElements* και *writersElements* γίνεται με παρόμοιο τρόπο ώς εξής. Αρχικώς αρχικοποιείται η λίστα ως μια νέα κενή λίστα. Μετά, χρησιμοποιώντας τις συναρτήσεις *getReaders()*, *getProcessors()* και *getWriters()* της κλάσης *XmlParser*, δημιουργούνται τα αντίστοιχα αντικείμενα *ElementPanels* και θέτεται το όνομά τους από τις πληροφορίες στο αρχείο και η τιμή τους στο String “Not Available Yet”, αφού πριν ξεκινήσει το σύστημα δεν υπάρχει καμία τιμή εξόδου από κανένα στοιχείο. Για τον ορισμό της περιγραφής των *ElementPanel* στοιχείων απαιτείται η δημιουργία των αντίστοιχων *Readers*, *Processors* και *Writers*, τα οποία παρέχουν τη μέθοδο *getDescription()* η οποία επιστρέφει την πιο κατάλληλη

περιγραφή.

```
// Initialize the readers modules list
readersElements = new ArrayList<ElementPanel>();
for (ElementInfo config : configParser.getReaders()) {
    ElementPanel readerElement = new ElementPanel();
    readerElement.setName(config.getId());
    readerElement.setValue("Not Available Yet");
    Reader reader = (Reader
                    (Class.forName(config.getClassName()).newInstance());
    reader.setParameters(config.getParameters());
    readerElement.setDescription(reader.getDescription());
    readersElements.add(readerElement);
    readerElement.addMouseListener(this);
}

// Initialize the processors modules list
processorsElements = new ArrayList<ElementPanel>();
for (ElementInfo config : configParser.getProcessors()) {
    ElementPanel processorElement = new ElementPanel();
    processorElement.setName(config.getId());
    processorElement.setValue("Not Available Yet");
    Processor processor = (Processor)
                    (Class.forName(config.getClassName()).newInstance());
    processor.setParameters(config.getParameters());
    processorElement.setDescription(processor.getDescription());
    processorsElements.add(processorElement);
    processorElement.addMouseListener(this);
}

// Initialize the writers modules list
writersElements = new ArrayList<ElementPanel>();
for (ElementInfo config : configParser.getWriters()) {
    ElementPanel writerElement = new ElementPanel();
    writerElement.setName(config.getId());
    writerElement.setValue("");
    Writer writer = (Writer)
                    (Class.forName(config.getClassName()).newInstance());
    writer.setParameters(config.getParameters());
    writerElement.setDescription(writer.getDescription());
    writersElements.add(writerElement);
    writerElement.addMouseListener(this);
}
```

Όπως φαίνεται στον κώδικα, η κλάση *MiodzioGui* ορίζεται να δέχεται συμβάντα ποντικιού από όλα τα στοιχεία *ElementPanel*. Αυτό γίνεται όπως θα εξηγηθεί αργότερα για τη δυνατότητα επιλογής κάποιου στοιχείου με τη χρήση του ποντικιού.

Μετά την κατασκευή των λιστών με τα στοιχεία *ElementPanel*, το σύστημα προχωράει με την κατασκευή των λιστών που περιέχουν τις συνδέσεις μεταξύ τους. Η διαδικασία αυτή είναι ιδιαίτερα απλή. Για κάθε *Reader* και *Processor* στοιχείο το σύστημα ελέγχει τις εξόδους του και τις προσθέτει στις λίστες ανάλογα. Η μοναδική παρατήρηση άξια αναφοράς είναι ότι εάν κάποιο

στοιχείο *Processor* συνδέεται με κάποιο άλλο στοιχείο *Processor* καλείται η συνάρτηση *setOutputBack()* με παράμετρο true, ώστε να σχεδιαστεί σωστά το στοιχείο.

```
// Initialize the readers outputs list
readersOutputs = new ArrayList<List<Integer>>();
readerToWriterOutputs = new ArrayList<List<Integer>>();
for (ElementInfo config : configParser.getReaders()) {
    List<Integer> processorOutputs = new ArrayList<Integer>();
    List<Integer> writerOutputs = new ArrayList<Integer>();
    for (String outputName : config.getOutputs()) {
        for (ElementPanel processorElement : processorsElements) {
            if (processorElement.getName().equals(outputName)) {
                processorOutputs.add(processorsElements.indexOf(
                                                    processorElement));
                break;
            }
        }
        for (ElementPanel writerElement : writersElements) {
            if (writerElement.getName().equals(outputName)) {
                writerOutputs.add(writersElements.indexOf(
                                                      writerElement));
                break;
            }
        }
    }
    readersOutputs.add(processorOutputs);
    readerToWriterOutputs.add(writerOutputs);
}

// Initialize the processors outputs list
processorsOutputs = new ArrayList<List<Integer>>();
processorToProcessorOutputs = new ArrayList<List<Integer>>();
for (ElementInfo config : configParser.getProcessors()) {
    List<Integer> processorOutputs = new ArrayList<Integer>();
    List<Integer> writerOutputs = new ArrayList<Integer>();
    for (String outputName : config.getOutputs()) {
        for (ElementPanel processorElement : processorsElements) {
            if (processorElement.getName().equals(outputName)) {
                processorsElements.get(processorToProcessorOutputs.
                                        size()).setOutputBack(true);
                processorOutputs.add(processorsElements.indexOf(
                                                    processorElement));
                break;
            }
        }
        for (ElementPanel writerElement : writersElements) {
            if (writerElement.getName().equals(outputName)) {
                writerOutputs.add(writersElements.indexOf(
                                                      writerElement));
                break;
            }
        }
    }
    processorsOutputs.add(writerOutputs);
    processorToProcessorOutputs.add(processorOutputs);
```

```
}
```

Με την ολοκλήρωση του ανωτέρω κώδικα τελειώνει το διάβασμα του αρχείου και το σύστημα έχει όλες τις πληροφορίες που χρειάζεται για τη συνέχιση της αρχικοποίησής του. Αυτό γίνεται με την αρχικοποίηση του *DrawingPanel*, τη δημιουργία ενός *ValueUpdater* ο οποίος θα ενημερώνει τις τιμές των στοιχείων και τη δημιουργία ενός αντικειμένου *Miodzio* ορισμένο να λειτουργεί ως ενσωματωμένο σε εφαρμογή.

```
// Initialize the drawing panel
drawingPanel.initialize(readersElements,  processorsElements,
                        writersElements, readersOutputs,
                        readerToWriterOutputs, processorsOutputs,
                        processorToProcessorOutputs);

// Create the value updater
Map <String,ElementPanel> elements = new HashMap<String,ElementPanel>();
for (ElementPanel element : readersElements) {
    elements.put(element.getName(), element);
}
for (ElementPanel element : processorsElements) {
    elements.put(element.getName(), element);
}
for (ElementPanel element : writersElements) {
    elements.put(element.getName(), element);
}
valueUpdater = new ValueUpdater(elements, outputTextArea);
valueUpdater.start();

// Create the miodzio object and start it
try {
    miodzio = new Miodzio(xmlFile, true);
    miodzio.start();
} catch (Exception e) {
    e.printStackTrace();
    System.exit(0);
}
```

Όπως αναφέρθηκε προηγουμένως, όλα τα *ElementPanel* στοιχεία ενημερώνουν την κλάση *MiodzioGui* για συμβάντα ποντικιού. Για να γίνει αυτό η κλάση *MiodzioGui* πρέπει να υλοποιεί τη διεπαφή *MouseListener*. Αυτό προϋποθέτει την υλοποίηση αρκετών μεθόδων οι οποίες καλούνται ανάλογα με τον τύπο του συμβάντος ποντικιού, δηλαδή με την κίνηση, την είσοδο ή έξοδο του ποντικιού σε κάποιο Swing στοιχείο, το πάτημα ενός κουμπιού κτλ. Όλες αυτές οι μέθοδοι έχουν υλοποιηθεί ως κενές μέθοδοι (δεν περιέχουν κανέναν κώδικα οπότε δεν εκτελούν καμία ενέργεια) εκτός από μία, την *mouseEntered()*. Αυτή η συνάρτηση καλείται όταν το ποντίκι μπει σε κάποιο στοιχείο και θα ορίσει το στοιχείο που είναι επιλεγμένο από το χρήστη, καθώς και ποια στοιχεία είναι οι είσοδοι και οι έξοδοι του.

Αρχικά θέτονται οι καταστάσεις όλων των στοιχείων ως *NORMAL*, για να αναιρεθούν οι καταστάσεις από την προηγούμενη κλήση.

```
for (ElementPanel module : readersElements) {
    module.setState(ElementPanel.STATE_NORMAL);
}
for (ElementPanel module : processorsElements) {
    module.setState(ElementPanel.STATE_NORMAL);
}
for (ElementPanel module : writersElements) {
    module.setState(ElementPanel.STATE_NORMAL);
}
```

Αφού οριστούν οι καταστάσεις των στοιχείων ως *NORMAL*, το σύστημα εξάγει το όνομα του στοιχείου στο οποίο μπήκε το ποντίκι. Αυτό γίνεται μέσω του αντικειμένου *MouseEvent* που δέχεται η μέθοδος *mouseEntered()*. Το αντικείμενο αυτό παρέχει τη μέθοδο *getSource()*, η οποία επιστρέφει το αντικείμενο που δημιούργησε το συμβάν.

```
// Get the name of the element the mouse just entered
ElementPanel source = (ElementPanel)e.getSource();
String sourceName = source.getName();
```

Ξέροντας το όνομα του στοιχείου στο οποίο μπήκε το ποντίκι, θέτεται η κατάστασή του ως *SELECTED*. Επίσης ενημερώνεται και το αντικείμενο *valueUpdater* για να ενημερώνει ανάλογα την περιοχή κειμένου με την έξοδο του επιλεγμένου στοιχείου και ενημερώνονται κατάλληλα οι δύο περιοχές κειμένου.

```
// Set the state of the selected element as SELECTED
source.setState(ElementPanel.STATE_SELECTED);

// Tell to the value updater which element is the currently selected
valueUpdater.setSelectedElement(sourceName);

// Update the two text areas for the new selection
infoTextArea.setText(sourceName + ":" +
                             System.getProperty("line.separator") +
                             source.getDescription());
outputTextArea.setText(source.getValue());
```

Τέλος, ελέγχεται ο τύπος του στοιχείου (αν είναι reader, processor ή writer) και ανάλογα θέτονται οι καταστάσεις των εισόδων και των εξόδων του. Η διαδικασία αυτή φαίνεται στον ακόλουθο κώδικα:

```
//If we have a reader set the states of the output elements
int sourceIndex = readersElements.indexOf(source);
if (sourceIndex != -1) {
    for (Integer outputIndex : readersOutputs.get(sourceIndex)) {
        processorsElements.get(outputIndex).setState(ElementPanel.STATE_
```

```
OUTPUT);
    }
    for (Integer outputIndex : readerToWriterOutputs.get(sourceIndex)) {
        writersElements.get(outputIndex).setState(ElementPanel.STATE_OUT
PUT);
    }
}

// If we have a processor set the states of the output and input
elements
sourceIndex = processorsElements.indexOf(source);
if (sourceIndex != -1) {
    for (Integer outputIndex : processorsOutputs.get(sourceIndex)) {
        writersElements.get(outputIndex).setState(ElementPanel.STATE_OUT
PUT);
    }
    for (Integer outputIndex :
processorToProcessorOutputs.get(sourceIndex)) {
        processorsElements.get(outputIndex).setState(ElementPanel.STATE_
OUTPUT);
    }
    for (int i = 0; i < readersOutputs.size(); i++) {
        if (readersOutputs.get(i).contains(sourceIndex)) {
            readersElements.get(i).setState(ElementPanel.STATE_INPUT);
        }
    }
    for (int i = 0; i < processorToProcessorOutputs.size(); i++) {
        if (processorToProcessorOutputs.get(i).contains(sourceIndex)) {
            processorsElements.get(i).setState(ElementPanel.STATE_INPUT)
;
        }
    }
}

// If we have a writer set the states of input elements
sourceIndex = writersElements.indexOf(source);
if (sourceIndex != -1) {
    for (int i = 0; i < readerToWriterOutputs.size(); i++) {
        if (readerToWriterOutputs.get(i).contains(sourceIndex)) {
            readersElements.get(i).setState(ElementPanel.STATE_INPUT);
        }
    }
    for (int i = 0; i < processorsOutputs.size(); i++) {
        if (processorsOutputs.get(i).contains(sourceIndex)) {
            processorsElements.get(i).setState(ElementPanel.STATE_INPUT)
;
        }
    }
}
```

4.7. Πακέτο example

Το πακέτο *example* δεν αναλύθηκε στα προηγούμενα κεφάλαια, επειδή δεν είναι ακριβώς κομμάτι

του συστήματος (το σύστημα δεν εξαρτάται με κανέναν τρόπο από αυτό). Το πακέτο αυτό περιέχει παραδείγματα υλοποιήσεων των τεσσάρων κλάσεων που πρέπει να υλοποιούν οι χρήστες, δηλαδή των *Data*, *Reader*, *Processor* και *Writer*. Οι υλοποιήσεις αυτές ακολουθούν τους κανόνες που αναφέρθηκαν στις προηγούμενες ενότητες και μπορούν να χρησιμοποιηθούν ως σκελετός για τις υλοποιήσεις των χρηστών.

4.7.1. Κλάση ExampleData

Η κλάση αυτή είναι ένα παράδειγμα υλοποίησης της διεπαφής *Data*. Τα δεδομένα που αντιπροσωπεύει είναι ένας δεκαδικός αριθμός διπλής ακρίβειας. Ο κώδικας που υλοποιεί την κλάση είναι ο ακόλουθος:

```
public class ExampleData implements Data {

    private double value;

    public ExampleData(double value) {
        this.value = value;
    }

    public double getValue() {
        return value;
    }

    public String toString() {
        return Double.toString(value);
    }
}
```

Όπως φαίνεται και στον κώδικα, η κλάση χρησιμοποιεί ένα αντικείμενο μέλος τύπου *double* στο οποίο αποθηκεύεται τοπικά η τιμή του αριθμού. Κατά την κατασκευή του αντικειμένου και κατά την ανάκληση της τιμής του μέσω της μεθόδου *getValue()* γίνεται απλώς ανάθεση και όχι κάποια πιο περίπλοκη αντιγραφή, επειδή οι μεταβλητές τύπου *double* είναι βασικές μεταβλητές και η Java τις αντιγράφει “by value”. Τέλος, η μέθοδος *toString()* επιστρέφει ένα αλφαριθμητικό το οποίο αντιπροσωπεύει την τιμή της μεταβλητής *value*.

4.7.2. Κλάση ExampleReader

Η κλάση *ExampleReader* είναι ένα παράδειγμα υλοποίησης της κλάσης *Reader*. Η κλάση αυτή δημιουργεί τυχαίους αριθμούς από το μηδέν έως το ένα ανά περίοδο την οποία δέχεται ως παράμετρο. Ο σκελετός της κλάσης είναι ο ακόλουθος:

```
public class ExampleReader extends Reader {

    private long period = 0;
```

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

    public void setParameters(Properties parameters) throws
                                                ParameterProblemException {
        ....
    }

    public String getDescription() {
        ....
    }
}
```

Το αντικείμενο μέλος της κλάσης *period* διατηρεί την περίοδο σε msec κατά την οποία δημιουργούνται οι τυχαίοι αριθμοί. Ο ορισμός της γίνεται μέσω της μεθόδου *setParameters()*. Η μέθοδος αυτή ελέγχει το αντικείμενο *parameters* για την τιμή της περιόδου και την αποθηκεύει στη μεταβλητή *period* αφού πρώτα τη μετατρέψει σε τύπο *double*. Στην περίπτωση που η παράμετρος δεν περιέχεται στο αντικείμενο *parameters* ή η τιμή της δεν είναι κάποιος δεκαδικός αριθμός δημιουργούνται τα ανάλογα σφάλματα.

```
public void setParameters(Properties parameters) throws
                                                ParameterProblemException {
    // get the period and check if it exists
    String periodString = parameters.getProperty("period");
    if (periodString == null) {
        throw new ParameterProblemException("Parameter period is not
                                                                    set");
    }

    // Convert the period to long
    try {
        period = Long.parseLong(periodString);
    } catch (Exception e) {
        throw new ParameterProblemException("Period wasn't a long");
    }
}
```

Η συνάρτηση *run()* ξεκινάει ένα άπειρο loop, μέσα στο οποίο αρχικά περιμένει για τόσα msec όσο είναι η τιμή της μεταβλητής *period*, δημιουργεί ένα αντικείμενο *ExampleData* το οποίο περιέχει μια τυχαία τιμή από το μηδέν έως το ένα, η οποία δημιουργείται με τη χρήση της συναρτησης *Math.random()*, και δημιουργεί ένα συμβάν για το οποίο ενημερώνονται όλες οι κατάλληλες κλάσεις.

```
public void run() {
    // Start the infinite loop
    while (true) {
        // Sleep for period
```

```
        try {
            Thread.currentThread().sleep(period);
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }

        // Generate the random number
        ExampleData data = new ExampleData(Math.random());

        fireDataEvent(new DataEvent(this, data));
    }
}
```

Τέλος, η μέθοδος *getDescription()* επιστρέφει ένα αλφαριθμητικό που περιγράφει τη λειτουργία της κλάσης.

```
public String getDescription() {
    return "Every " + period + " miliseconds it creates an ExampleData "
           + "(double) that contains a random value between 0 and 1";
}
```

4.7.3. Κλάση ExampleProcessor

Η κλάση *ExampleProcessor* είναι ένα παράδειγμα υλοποίησης ενός *Processor*. Η λειτουργία της κλάσης είναι να δέχεται αντικείμενα τύπου *ExampleData* και, αν η τιμή τους είναι μεγαλύτερη από κάποιο κατώφλι τις προωθεί στην έξοδό της, ενώ αν η τιμή τους είναι μικρότερη από το κατώφλι τα μηνύματα αγνοούνται. Ο κώδικας που υλοποιεί την κλάση είναι ο ακόλουθος:

```
public class ExampleProcessor extends Processor {

    private double trigger;

    public void run() {
        ....
    }

    public void setParameters(Properties parameters) throws
                                                ParameterProblemException {
        // Get the trigger from the parameters and check if it was set
        String triggerString = parameters.getProperty("trigger");
        if (triggerString == null) {
            throw new ParameterProblemException("Parameter trigger is
                                                        not set");
        }

        // Convert the trigger to double and store it locally
        try {
            trigger = Double.parseDouble(triggerString);
        } catch (Exception e) {
            throw new ParameterProblemException("Trigger wasn't a
                                                        double");
```

```
        }
    }

    public String getDescription() {
        return "Reads data of ExampleData (doubles) and forwards only
                the ones with value bigger or equal with " + trigger;
    }
}
```

Ο τρόπος με τον οποίο γίνεται ο χειρισμός για την παράμετρο *trigger* (η οποία αντιπροσωπεύει το κατώφλι) και η υλοποίηση της μεθόδου *getDescription()* γίνεται ακριβώς όπως στην κλάση *ExampleData* και δεν θα αναλυθεί περαιτέρω.

Η μέθοδος *run()* περιέχει ένα άπειρο loop, το οποίο δέχεται τα δεδομένα από την ουρά, τα μετατρέπει σε τύπο *ExampleData*, ελέγχει αν είναι μεγαλύτερα από το κατώφλι, και εάν είναι, δημιουργεί ένα συμβάν για το οποίο ενημερώνονται τα στοιχεία εξόδου.

```
public void run() {
    // Start the infinite loop
    while (true) {
        // get the data from the queue
        Data data = queue.getOldestData();

        // Check that the queue wasn't empty. If it was skip the rest of
        // the loop and try to read again from the queue.
        if (data == null) {
            continue;
        }

        // Check if the data is smaller than the trigger and if it is
        // skip the rest
        if (((ExampleData)data).getValue() < trigger) {
            continue;
        }

        // Here we know the data should be forwarded, so do it
        fireDataEvent(new DataEvent(this, data));
    }
}
```

4.7.4. Κλάση ExampleWriter

Η κλάση *ExampleWriter* είναι ένα παράδειγμα υλοποίησης *Writer*. Η κλάση αυτή έχει μία παράμετρο η οποία αποθηκεύεται στη μεταβλητή *prefix* και τυπώνει στην οθόνη τα μηνύματα που δέχεται, προσθέτοντας στην αρχή την τιμή της μεταβλητής *prefix*. Ο κώδικας που υλοποιεί την κλάση είναι ο ακόλουθος:

```
public class ExampleWriter extends Writer {
```

```
    private String prefix;

    public void run() {
        // Start the infinite loop
        while (true) {
            // get the data from the queue
            Data data = queue.getOldestData();

            // Check that the queue wasn't empty. If it was skip the
            // rest of the loop and try to read again from the queue.
            if (data == null) {
                continue;
            }

            // Print the data
            System.out.println(prefix + data);
        }
    }

    public void setParameters(Properties parameters) {
        prefix = parameters.getProperty("prefix");
        if (prefix == null) {
            prefix = "";
        }
    }

    public String getDescription() {
        return "It prints the ExampleData it gets to the screen with the
                                                prefix \"" + prefix + "\"";
    }
}
```

Η περαιτέρω επεξήγηση του κώδικα είναι περιττή, αφού οι διαδικασίες είναι όμοιες με με τις κλάσεις *ExampleReader* και *ExampleProcessor*. Η μοναδική διαφορά είναι ότι εάν η παράμετρος *prefix* δεν περιέχεται στο αντικείμενο *parameters* θέτεται αυτόματα σε ένα κενό αλφαριθμητικό και δεν δημιουργείται κανένα σφάλμα. Αυτό μετατρέπει την παράμετρο *prefix* σε προαιρετική, οπότε η τιμή της μπορεί να λείπει από το XML αρχείο.

4.7.5. Παράδειγμα XML αρχείου

Χρησιμοποιώντας τις παραπάνω κλάσεις ακολουθεί ένα παράδειγμα XML αρχείου για την αρχικοποίηση του συστήματος. Το παράδειγμα αυτό περιέχει ένα *ExampleReader* ο οποίος δημιουργεί τιμές οι οποίες στέλνονται σε δύο *ExampleProcessors*. Οι *processors* αυτοί επιτρέπουν να περάσουν τιμές μεγαλύτερες από 0,3 και 0,7 αντίστοιχα, και συνδέονται με δύο *ExampleWriters* οι οποίοι τυπώνουν στην οθόνη τα αποτελέσματα.

```
<?xml version="1.0"?>
<!DOCTYPE MIODZIO SYSTEM "miodzio.dtd">
```

```
<MIODZIO>
    <READERS>
        <READER CLASS="gr.ntua.ece.miodzio.example.ExampleReader"
                                                OBJECTID="ExampleReader">
            <PARAMETERS>
                <PARAMETER NAME="period">100</PARAMETER>
            </PARAMETERS>
            <OUTPUTS>
                <OUTPUT>ExampleProcessor1</OUTPUT>
                <OUTPUT>ExampleProcessor2</OUTPUT>
            </OUTPUTS>
        </READER>
    </READERS>
    <PROCESSORS>
        <PROCESSOR CLASS="gr.ntua.ece.miodzio.example.ExampleProcessor"
                                            OBJECTID="ExampleProcessor1">
            <PARAMETERS>
                <PARAMETER NAME="trigger">0.3</PARAMETER>
            </PARAMETERS>
            <OUTPUTS>
                <OUTPUT>ExampleWriter1</OUTPUT>
            </OUTPUTS>
        </PROCESSOR>
        <PROCESSOR CLASS="gr.ntua.ece.miodzio.example.ExampleProcessor"
                                            OBJECTID="ExampleProcessor2">
            <PARAMETERS>
                <PARAMETER NAME="trigger">0.7</PARAMETER>
            </PARAMETERS>
            <OUTPUTS>
                <OUTPUT>ExampleWriter2</OUTPUT>
            </OUTPUTS>
        </PROCESSOR>
    </PROCESSORS>
    <WRITERS>
        <WRITER CLASS="gr.ntua.ece.miodzio.example.ExampleWriter"
                                              OBJECTID="ExampleWriter1">
            <PARAMETERS>
                <PARAMETER NAME="prefix">Writer 1: </PARAMETER>
            </PARAMETERS>
        </WRITER>
        <WRITER CLASS="gr.ntua.ece.miodzio.example.ExampleWriter"
                                               BJECTID="ExampleWriter2">
            <PARAMETERS>
                <PARAMETER NAME="prefix">Writer 2: </PARAMETER>
            </PARAMETERS>
        </WRITER>
    </WRITERS>
</MIODZIO>
```

Ένα δείγμα της εξόδου της οθόνης κατά την εκτέλεση του συστήματος με το συγκεκριμένο XML αρχείο είναι η ακόλουθη:

```
Writer 1: 0.9008952306630278
Writer 2: 0.9008952306630278
Writer 1: 0.3609729434573501
```

```
Writer 1: 0.6529979401883561
Writer 1: 0.9539171011812374
Writer 2: 0.9539171011812374
Writer 1: 0.7446224123079154
Writer 2: 0.7446224123079154
Writer 1: 0.8627108164041641
Writer 2: 0.8627108164041641
Writer 2: 0.7389151189695994
Writer 1: 0.7389151189695994
```

Όπως φαίνεται από την έξοδο του συστήματος, ο Writer 1 τυπώνει τις τιμές που είναι μεγαλύτερες από 0,3 και ο Writer 2 τις τιμές που είναι μεγαλύτερες από 0,7 όπως αναμενόταν.

5. Εφαρμογές του συστήματος

Στα προηγούμενα κεφάλαια παρουσιάστηκαν η ανάλυση, η σχεδίαση και η υλοποίηση του συστήματος. Τα ανωτέρω έχουν ως αποτέλεσμα ένα σύστημα το οποίο ανταποκρίνεται στις λειτουργικές προδιαγραφές που τέθηκαν. Σε αυτό το κεφάλαιο αναλύεται η χρήση του συστήματος για τη διασύνδεση τριών εφαρμογών του Image, Video and Multimedia Systems Lab (IVML) του Εθνικού Μετσόβιου Πολυτεχνείου.

5.1. Εφαρμογές προς διασύνδεση

Οι τρεις εφαρμογές που συνδέθηκαν είναι οι Components, Torque και Callas AMS. Η εφαρμογή Components επικοινωνεί με μία κάμερα και παρατηρεί τις κινήσεις των χεριών όποιου κάθεται μπροστά από την κάμερα. Από αυτές τις κινήσεις εξάγει διάφορες παραμέτρους από τις οποίες στη συγκεκριμένη περίπτωση ενδιαφερόμαστε μόνο για μία, την ταχύτητα της κίνησης. Η επικοινωνία με την εφαρμογή αυτή γίνεται μέσω μιας TCP/IP σύνδεσης.

Η εφαρμογή Torque είναι ένας τρισδιάστατος κόσμος. Μέσω μίας TCP/IP σύνδεσης είναι δυνατόν να αλλάξουν διάφορες μεταβλητές της εφαρμογής όπως είναι η ταχύτητα με την οποία κινείται ο χρήστης. Τέλος η εφαρμογή Callas AMS είναι μία εφαρμογή η οποία παίζει μουσική, αλλά μέσω μίας TCP/IP σύνδεσης είναι δυνατή η μεταβολή της ταχύτητας ή της χροιάς του τρόπου παιξίματος.

Η διασύνδεση των ανωτέρω εφαρμογών είχε ως σκοπό τον έλεγχο της ταχύτητας της κίνησης του χρήστη στο πρόγραμμα Torque και τον ταυτόχρονο έλεγχο της ταχύτητας εκτέλεσης του κομματιού από το πρόγραμμα Callas AMS, μέσω της ταχύτητας της κίνησης των χεριών του χρήστη ο οποίος χρησιμοποιεί το πρόγραμμα Components. Δηλαδή η εφαρμογή Components είναι η εφαρμογή εισόδου του συστήματος και οι εφαρμογές Torque και Callas AMS είναι οι εφαρμογές εξόδου.

5.2. Απαραίτητες κλάσεις

Για την υλοποίηση της σύνδεσης είναι απαραίτητη η δημιουργία των ακόλουθων κλάσεων.

5.2.1. Υλοποιήσεις Data

- *TorqueData*: Η κλάση αυτή υλοποιεί την κλάση *Data* και αντιπροσωπεύει τα δεδομένα ενός μηνύματος για επικοινωνία με την εφαρμογή Torque.
- *DoubleData*: Η κλάση αυτή υλοποιεί την κλάση *Data* και αντιπροσωπεύει ένα δεκαδικό αριθμό τύπου *double*. Η κλάση αυτή έχει ήδη υλοποιηθεί στο πακέτο *example* του συστήματος και απλώς μεταφέρθηκε με άλλο όνομα σε άλλο πακέτο για να είναι πιο λογική η θέση της, οπότε δεν θα γίνει περαιτέρω ανάλυσή της.
- *IntegerData*: Η κλάση αυτή είναι όμοια με την *DoubleData* με τη διαφορά ότι αντιπροσωπεύει έναν ακέραιο αντί για δεκαδικό.

5.2.2. Υλοποιήσεις Readers

- ComponentsReader: Η κλάση αυτή υλοποιεί την κλάση Reader και δημιουργεί ένα TCP/IP server ο οποίος δέχεται μηνύματα από την εφαρμογή Components και δημιουργεί συμβάντα με αντικείμενα τύπου *TorqueData*.

5.2.3. Υλοποιήσεις Processors

- *CounterProcessor*: Η κλάση αυτή υλοποιεί έναν *Processor* ο οποίος μετράει τα μηνύματα που δέχεται (ανεξαρτήτως του τύπου τους) και προωθεί στην έξοδό του τον αριθμό τους ως *IntegerData*
- *SpeedAverage*: Η κλάση αυτή υλοποιεί έναν *Processor* ο οποίος δέχεται μηνύματα τύπου *TorqueData* και προωθεί στην έξοδό του επίσης μηνύματα τύπου *TorqueData* με το μέσο όρο της ταχύτητας των εισερχόμενων μηνυμάτων

- *DivideSpeed*: Η κλάση αυτή υλοποιεί έναν *Processor* ο οποίος δέχεται μηνύματα τύπου *TorqueData* και τα προωθεί στην έξοδό του με διαιρεμένη την ταχύτητα με κάποιον αριθμό που ορίζει ο χρήστης

- *AmsProcessor*: Η κλάση αυτή υλοποιεί έναν *Processor* ο οποίος δέχεται μηνύματα τύπου *TorqueData* και δημιουργεί μηνύματα τύπου *DoubleData*, επεξεργάζοντας κατάλληλα την τιμή της ταχύτητας από τα εισερχόμενα μηνύματα

5.2.4. Υλοποιήσεις Writers

- *TorqueWriter*: Η κλάση αυτή υλοποιεί έναν *Writer* ο οποίος δέχεται μηνύματα τύπου *TorqueData* και στέλνει μηνύματα ελέγχου σε μία εφαρμογή Torque μέσω ενός TCP/IP client.

- *AmsWriter*: Η κλάση αυτή υλοποιεί έναν *Writer* ο οποίος δέχεται μηνύματα τύπου *DoubleData* και στέλνει μηνύματα ελέγχου σε μία εφαρμογή Callas AMS μέσω ενός TCP/ IP client.

5.3. Java και TCP/IP επικοινωνία

Όπως φαίνεται από τους ανωτέρω *Readers* και *Writers*, το σύστημα θα επικοινωνεί με τις εφαρμογές εισόδου και εξόδου μέσω TCP/IP μηνυμάτων. Γι' αυτόν τον λόγο κρίνεται σκόπιμο σε αυτό το σημείο να γίνει μία σύντομη αναφορά στον προγραμματισμό τέτοιων μηνυμάτων χρησιμοποιώντας τη γλώσσα προγραμματισμού Java.

Σύμφωνα με το TCP/IP πρωτόκολλο η σύνδεση και η ανταλλαγή μηνυμάτων γίνεται μεταξύ δύο κόμβων, του server και του client. Αρχικώς ο server αναμένει για συνδέσεις από clients σε κάποιο συγκεκριμένο port. Ο client, γνωρίζοντας την IP διεύθυνση του server και τον αριθμό του port στο οποίο αναμένει, επικοινωνεί με τον server και ξεκινάει την επικοινωνία μεταξύ τους. Αφού δημιουργηθεί η σύνδεση, ο client και ο server αρχίζουν την ανταλλαγή πακέτων μεταξύ τους, η οποία είναι αμφίδρομη. Το πρωτόκολλο της πληροφορίας που περιέχεται στα πακέτα εξαρτάται από τις εφαρμογές και ο server και ο client πρέπει να χρησιμοποιούν το ίδιο πρωτόκολλο για να επικοινωνήσουν σωστά.

Η Java, ως γλώσσα υψηλού επιπέδου, αποκρύπτει τις λεπτομέρειες του πρωτοκόλλου (όπως είναι το μέγεθος των πακέτων που μεταφέρονται, ο έλεγχος για χαμένα πακέτα κτλ) από το χρήστη οπότε δεν θα γίνει περαιτέρω αναφορά σε αυτές. Μάλιστα, προχωράει ένα ακόμα επίπεδο απόκρυψης των λεπτομερειών παραπάνω από τις άλλες γλώσσες και επιτρέπει στο χρήστη να

διαχειρίζεται τις TCP/IP συνδέσεις όπως όλα τα υπόλοιπα streams για είσοδο – έξοδο (I/O). Αυτό σημαίνει ότι η αποστολή και παραλαβή TCP/IP μηνυμάτων γίνεται με τον ίδιο τρόπο που διαβάζεται/εγγράφεται ένα αρχείο ή τυπώνεται ένα μήνυμα στην οθόνη, κάνοντας τον προγραμματισμό τέτοιων συνδέσεων ιδιαίτερα εύκολο, αφού ο προγραμματιστής μπορεί να εστιαστεί μόνο στην υλοποίηση του πρωτοκόλλου το οποίο υλοποιούν οι εφαρμογές.

Για την επίτευξη των ανωτέρω η Java παρέχει δύο κλάσεις, μία η οποία χρησιμοποιείται στον server και μία η οποία χρησιμοποιείται στον client. Οι κλάσεις αυτές είναι οι *ServerSocket* και η *Socket* αντίστοιχα.

Τα αντικείμενα τύπου *ServerSocket* δέχονται κατά τη δημιουργία τους τον αριθμό του port στο οποίο θα αναμένουν για συνδέσεις από τους clients και μετά από κάθε επιτυχημένη σύνδεση επιστρέφουν δύο stream, για εγγραφή και ανάγνωση των μηνυμάτων από τον client. Εδώ πρέπει να τονισθεί ότι μόλις δημιουργηθεί η σύνδεση με κάποιον client το αντικείμενο *ServerSocket* μπορεί να ξεκινήσει αμέσως να αναμένει για περαιτέρω συνδέσεις, επιτρέποντας έτσι την ταυτόχρονη επικοινωνία με περισσότερους από έναν clients.

Τα αντικείμενα τύπου *Socket* λειτουργούν με αντίστοιχο τρόπο. Δέχονται κατά την αρχικοποίησή τους την IP διεύθυνση και τον αριθμό του port του server και αφού δημιουργηθούν παρέχουν δύο stream για εγγραφή και ανάγνωση. Η IP του server δεν χρειάζεται καν να είναι στην αριθμητική της μορφή, αφού η Java θα μεταφράσει οποιοδήποτε URL επικοινωνώντας με κάποιον DNS server (ενέργεια η οποία αποκρύπτεται εντελώς από τον χρήστη).

Τα προγράμματα που χρησιμοποιούν τις ανωτέρω κλάσεις ενημερώνονται για οποιοδήποτε σφάλμα κατά την έναρξη της σύνδεσης ή κατά τη διάρκεια της ανταλλαγής μηνυμάτων με το σύστημα σφαλμάτων της Java.

5.4. Κλάση TorqueData

Η κλάση *TorqueData* υλοποιεί τη διεπαφή *Data* και αναπαριστά τα δεδομένα τα οποία περιέχονται στα TCP/IP μηνύματα επικοινωνίας του προγράμματος Torque. Αν και κατά τη συγκεκριμένη υλοποίηση θα χρησιμοποιηθεί μόνο η παράμετρος της ταχύτητας η κλάση αυτή περιέχει όλες τις πληροφορίες των μηνυμάτων για να γίνει ευκολότερη η μετέπειτα χρήση της στην περίπτωση που το σύστημα χρησιμοποιηθεί για τον έλεγχο και άλλων παραμέτρων.

Για την τοπική αποθήκευση της πληροφορίας του μηνύματος η κλάση περιέχει ένα αντικείμενο μέλος για κάθε παράμετρο του TCP/IP μηνύματος. Οι παράμετροι, οι οποίοι φαίνονται στον ακόλουθο κώδικα, εξάχθηκαν παρατηρώντας τα μηνύματα κατά την εκτέλεση των προγραμμάτων

εισόδου. Κάθε μία από αυτές ελέγχει και κάποια παράμετρο του προγράμματος Torque. Όλα τα τοπικά αντικείμενα είναι αλφαριθμητικά επειδή αυτός είναι ο τρόπος με τον οποίο περιέχονται και στα TCP/IP μηνύματα.

```
    private String refresh = null;
    private String clientGameConnection = null;
    private String speedMultiplier = null;
    private String sizeMultiplier = null;
    private String lightning = null;
    private String lightningDuration = null;
    private String ambush = null;
    private String numOfEnemies = null;
    private String animation = null;
```

Για την περαιτέρω χρήση των πληροφοριών που περιέχονται στην κλάση, παρέχονται μέθοδοι οι οποίες επιστρέφουν τις τιμές των παραμέτρων. Επειδή τα αλφαριθμητικά στη Java είναι immutable δεν χρειάζεται να γίνει αντιγραφή της τιμής τους και απλώς επιστρέφονται.

```
    public String getRefresh() {
        return refresh;
    }

    public String getClientGameConnection() {
        return clientGameConnection;
    }

    public String getSpeedMultiplier() {
        return speedMultiplier;
    }

    public String getSizeMultiplier() {
        return sizeMultiplier;
    }

    public String getLightning() {
        return lightning;
    }

    public String getLightningDuration() {
        return lightningDuration;
    }

    public String getAmbush() {
        return ambush;
    }

    public String getNumOfEnemies() {
        return numOfEnemies;
    }

    public String getAnimation() {
        return animation;
```

```
    }
```

Για την κατασκευή των αντικειμένων τύπου *TorqueData* παρέχεται ένας constructor ο οποίος δέχεται ως παραμέτρους τις τιμές για όλες τις τοπικές μεταβλητές. Επειδή ένα μήνυμα μπορεί να μην περιέχει όλες τις παραμέτρους, επιτρέπεται να ανατεθεί η τιμή null στις μεταβλητές. Οι συγκεκριμένες μεταβλητές στο εξής θα αγνοούνται.

```
public TorqueData(String refresh, String clientGameConnection,
                  String speedMultiplier, String sizeMultiplier,
                  String lightning, String lightningDuration,
                  String ambush, String numOfEnemies, String animation){
    this.refresh = refresh;
    this.clientGameConnection = clientGameConnection;
    this.speedMultiplier = speedMultiplier;
    this.sizeMultiplier = sizeMultiplier;
    this.lightning = lightning;
    this.lightningDuration = lightningDuration;
    this.ambush = ambush;
    this.numOfEnemies = numOfEnemies;
    this.animation = animation;
}
```

Για την ολοκλήρωση της υλοποίησης χρειάζεται να υλοποιηθεί η μέθοδος *toString()*. Λόγω της φύσης της κλάσης (δηλαδή ότι αντιπροσωπεύει ένα TCP/IP μήνυμα) η μέθοδος αυτή υλοποιήθηκε με τέτοιον τρόπο ώστε να επιστρέφει το κείμενο όπως πρέπει να αποσταλεί στα TCP/IP μηνύματα. Όπως φαίνεται και από τον κώδικα τα μηνύματα απλώς περιέχουν όλες τις παραμέτρους και τις τιμές τους, χωρισμένες με κενά.

```
    public String toString() {
        StringBuilder toReturn = new StringBuilder();
        if (refresh != null) {
            toReturn.append("refresh ");
            toReturn.append(refresh);
            toReturn.append(" ");
        } else {
            toReturn.append("clientGameConnection 1265(eXiler) ");
        }
        if (speedMultiplier != null) {
            toReturn.append("speed_multiplier ");
            toReturn.append(speedMultiplier);
            toReturn.append(" ");
        }
        if (sizeMultiplier != null) {
            toReturn.append("size_multiplier ");
            toReturn.append(sizeMultiplier);
            toReturn.append(" ");
        }
        if (lightning != null) {
            toReturn.append("lightning ");
            toReturn.append(lightning);
```

```
            toReturn.append(" ");
        }
        if (lightningDuration != null) {
            toReturn.append("lightning_duration ");
            toReturn.append(lightningDuration);
            toReturn.append(" ");
        }
        if (ambush != null) {
            toReturn.append("ambush ");
            toReturn.append(ambush);
            toReturn.append(" ");
        }
        if (numOfEnemies != null) {
            toReturn.append("num_of_enemies ");
            toReturn.append(numOfEnemies);
            toReturn.append(" ");
        }
        if (animation != null) {
            toReturn.append("animation ");
            toReturn.append(animation);
            toReturn.append(" ");
        }

        return toReturn.toString();
    }
```

5.5. Κλάση DoubleData

Η κλάση DoubleData έχει ήδη αναλυθεί προηγουμένως (κλάση *ExampleData*) και δεν θα αναλυθεί περαιτέρω. Η μόνη παρατήρηση που πρέπει να γίνει είναι ότι λόγω του γενικού χαρακτήρα της έχει τοποθετηθεί στο πακέτο gr.ntua.ece.miodzio.dataimpl για να είναι ευκολότερη η χρήση της από τους χρήστες του συστήματος.

5.6. Κλάση IntegerData

Η κλάση IntegerData είναι παρόμοια με την κλάση DoubleData, με τη διαφορά ότι αντί να αντιπροσωπεύει ένα δεκαδικό αριθμό, αντιπροσωπεύει έναν ακέραιο. Λόγω του γενικού της χαρακτήρα βρίσκεται στο ίδιο πακέτο με την DoubleData. Επειδή η υλοποίηση αυτών των δύο κλάσεων είναι σχεδόν ίδια δεν γίνεται περαιτέρω ανάλυσή της, αλλά απλώς παρουσιάζεται ο κώδικας που την υλοποιεί.

```
public class IntegerData implements Data {

    private int value;

    public IntegerData(int value) {
        this.value = value;
```

```
    }

    public int getValue() {
        return value;
    }

    public String toString() {
        return Integer.toString(value);
    }
}
```

5.7. Κλάση ComponentsReader

Η κλάση αυτή υλοποιεί την κλάση *Reader* και δημιουργεί ένα TCP/IP server στον οποίο μπορεί να συνδεθεί και να στείλει μηνύματα η εφαρμογή Components. Για τη δυνατότητα ρύθμισης του port στο οποίο θα περιμένει η κλάση για συνδέσεις από τους clients, ο αριθμός του port θα δίνεται ως παράμετρος. Ο αριθμός αυτός θα αποθηκεύεται τοπικά σε μια ακέραια μεταβλητή. Η υλοποίηση της μεθόδου *setParameters()* φαίνεται στον ακόλουθο κώδικα.

```
private int port = -1;

public void setParameters(Properties parameters) throws
                                            ParameterProblemException {
    // get the port and check if it exists
    String portString = parameters.getProperty("port");
    if (portString == null) {
        throw new ParameterProblemException("Parameter port is not
                                                            set");
    }
    // Convert the port to int
    try {
        port = Integer.parseInt(portString);
    } catch (Exception e) {
        throw new ParameterProblemException("Port wasn't an integer");
    }
}
```

Για την ορθή υλοποίηση της κλάσης *Reader* πρέπει να υλοποιηθεί και η συνάρτηση *getDescription()*. Η συνάρτηση αυτή απλώς επιστρέφει ένα κείμενο στο οποίο αναφέρει τον αριθμό του port στο οποίο αναμένει για συνδέσεις.

```
public String getDescription() {
    return "Listens at port " + port + " for component clients and
            creates a new thread to listen the messages from each
            connected client. It forwards to the processors all the
            messages from all the clients.";
}
```

Το πιο σημαντικό κομμάτι της κλάσης είναι η υλοποίηση της μεθόδου *run()* η οποία είναι

υπεύθυνη για να δημιουργήσει τον TCP/IP server και να διαχειρίζεται τις συνδέσεις σε αυτόν. Ο κώδικας που την υλοποιεί είναι ο ακόλουθος:

```
public void run() {
    // Create the server socket and bound to the port
    ServerSocket serverSocket = null;
    try {
        serverSocket = new ServerSocket(port);
    } catch (IOException ex) {
        System.out.println("Failed to listen at port " + port);
            ex.printStackTrace();
    }

    // Start an infinite loop where new clients will be connected
    while (true) {
        // Wait until a client is connected
        Socket socket = null;
        try {
            socket = serverSocket.accept();
        } catch (IOException ex) {
            ex.printStackTrace();
        }

        ComponenteaderThread thread =
                                new ComponentReaderThread(this, socket);
        thread.start();
    }
}
```

Όπως φαίνεται στον κώδικα, αρχικώς δημιουργείται ένα αντικείμενο τύπου *ServerSocket*. Κατά τη δημιουργία του αντικειμένου αυτού δίνεται ως παράμετρος ο αριθμός του port που θα χρησιμοποιηθεί και, αν δεν υπάρχει πρόβλημα χρήσης του συγκεκριμένου port το λειτουργικό σύστημα το παραχωρεί και δημιουργείται η κλάση. Στην περίπτωση που το port που ζητείται από το λειτουργικό σύστημα δεν είναι διαθέσιμο (είτε χρησιμοποιείται από κάποια άλλη εφαρμογή, είτε ο χρήστης δεν επιτρέπεται να το χρησιμοποιήσει) το αντικείμενο δεν θα δημιουργηθεί και το σύστημα θα τερματιστεί αναφέροντας το πρόβλημα.

Αφού δημιουργηθεί το αντικείμενο *serverSocket* καλείται η συνάρτηση *accept()*. Η συνάρτηση αυτή θα αναμείνει μέχρι κάποιος client να επιχειρήσει να συνδεθεί. Οι λεπτομέρειες της σύνδεσης αυτής εκτελούνται από τη Java και επιστρέφεται ένα αντικείμενο τύπου *Socket* από το οποίο μπορούν να εξαχθούν τα streams για την επικοινωνία με τον client.

Η διαχείριση αυτού του stream (οπότε και της σύνδεσης με το συγκεκριμένο client) γίνεται σε ένα καινούργιο νήμα εργασίας τύπου *ComponentReaderThread* (το οποίο παρουσιάζεται παρακάτω). Με αυτόν τον τρόπο το τρέχον νήμα εργασίας ελευθερώνεται και το αντικείμενο *serverSocket* χρησιμοποιείται ξανά για να αναμένει σύνδεση από κάποιον καινούργιο client. Δηλαδή θα είναι

δυνατή η παράλληλη σύνδεση περισσοτέρων των ενός clients και η ταυτόχρονη επεξεργασία των μηνυμάτων από αυτούς.

5.7.1. Κλάση ComponentReaderThread

Η κλάση *ComponentReaderThread* είναι υπεύθυνη για τη διαχείριση της σύνδεσης από κάθε έναν από τους clients του προγράμματος Components. Η λειτουργία της είναι να δέχεται τα μηνύματα από τον client, να εξάγει από αυτά τις πληροφορίες που μεταφέρουν και να χρησιμοποιεί τη μέθοδο *fireDataEvent()* της κλάσης *ComponentReader* για την προώθησή τους στα υπόλοιπα στοιχεία του συστήματος.

Κατά την αρχικοποίησή της η κλάση δέχεται δύο παραμέτρους. Η μία είναι το αντικείμενο τύπου *ComponentReader* του οποίου η μέθοδος *fireDataEvent()* θα χρησιμοποιηθεί και η δεύτερη είναι το αντικείμενο τύπου *Socket* για επικοινωνία με τον client. Επειδή στη συγκεκριμένη περίπτωση η επικοινωνία είναι μονόδρομη (από τον client προς τον server), από το αντικείμενο *socket* εξάγεται μόνο το stream εισόδου το οποίο αποθηκεύεται τοπικά στη μεταβλητή *in* για περαιτέρω χρήση.

```
private ComponentReader componentReader = null;
private InputStream in = null;

/** Creates a new instance of ComponentReaderRunnable */
public ComponentReaderThread(ComponentReader reader, Socket socket) {
    this.componentReader = reader;
    try {
        this.in = socket.getInputStream();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

Κατά την εκτέλεσή της, η κλάση *ComponentReaderThread* διαβάζει τα μηνύματα από το stream διαβάζοντας έναν έναν τους χαρακτήρες και προσθέτοντάς τους σε ένα προσωρινό αλφαριθμητικό. Ο τερματισμός της σύνδεσης από τον client εντοπίζεται όταν διαβαστεί ο χαρακτήρας με τιμή -1, οπότε και το νήμα εκτέλεσης τερματίζεται.

```
while (c == 0) {
    c = in.read();
}
if (c == -1) {
    break;
}
StringBuffer message = new StringBuffer();
while (c != 0 && c != -1) {
    message.append((char) c);
    if (in.available() == 0) {
        c = 0;
```

```
    } else {
        c = in.read();
    }
}
String line = message.toString();
```

Τα μηνύματα που στέλνονται από το πρόγραμμα Components είναι όμοια με τα μηνύματα του προγράμματος Torque τα οποία αναλύθηκαν σε προηγούμενο κεφάλαιο, δηλαδή περιέχουν τις παραμέτρους refresh, clientGameConnection, speed_multiplier, size_multiplier, lightning, lightning_duration, ambush, num_of_enemies, και animation, καθώς και τις τιμές τους, χωρισμένες με κενά. Το πρόγραμμα εξάγει την τιμή της κάθε παραμέτρου και την αποθηκεύει σε μια τοπική μεταβλητή παίρνοντας ανά δύο τις λέξεις του μηνύματος (το όνομα και η τιμή της παραμέτρου) και επαναλαμβάνοντας μέχρι να ελέγξει όλο το μήνυμα.

```
while (!line.equals("")) {
    // If there is no space stop reading the line
    if (line.indexOf(' ') == -1) {
        break;
    }
    String attribute = line.substring(0, line.indexOf(' '));
    line = line.substring(line.indexOf(' ') + 1);

    String value = null;
    if (line.indexOf(' ') != -1) {
        value = line.substring(0, line.indexOf(' '));
        line = line.substring(line.indexOf(' ') + 1);
    } else {
        value = line;
        line = "";
    }

    if (attribute.equals("refresh")) {
        refresh = value;
    }
    if (attribute.equals("clientGameConnection")) {
        clientGameConnection = value;
    }
    if (attribute.equals("speed_multiplier")) {
        speedMultiplier = value;
    }
    if (attribute.equals("size_multiplier")) {
        sizeMultiplier = value;
    }
    if (attribute.equals("lightning")) {
        lightning = value;
    }
    if (attribute.equals("lightning_duration")) {
        lightningDuration = value;
    }
    if (attribute.equals("ambush")) {
        ambush = value;
```

```
    }
    if (attribute.equals("num_of_enemies")) {
        numOfEnemies = value;
    }
    if (attribute.equals("animation")) {
        animation = value;
    }
}
```

Αφού εξαχθεί ό,τι πληροφορία περιέχει το μήνυμα, οι τοπικές μεταβλητές χρησιμοποιούνται για τη δημιουργία ενός αντικειμένου τύπου *TorqueData*, και το τελευταίο αποστέλλεται σε όλα τα στοιχεία του συστήματος που πρέπει να ενημερωθούν, μέσω της μεθόδου *fireDataEvent()*.

```
TorqueData data = new TorqueData(refresh, clientGameConnection,
                                 speedMultiplier, sizeMultiplier, lightning,
                                 lightningDuration, ambush, numOfEnemies,
                                 animation);

componentReader.fireDataEvent(new DataEvent(componentReader, data));
```

Το νήμα εκτέλεσης θα συνεχίσει να διαβάζει τα εισερχόμενα μηνύματα και να επαναλαμβάνει την ανωτέρω διαδικασία μέχρι να διαβάσει τον χαρακτήρα με τιμή -1, που σημαίνει ότι η σύνδεση τερματίστηκε από τον client, οπότε και τερματίζει την εκτέλεσή του.

5.8. Κλάση CounterProcessor

Η κλάση *CounterProcessor* είναι μία υλοποίηση Processor γενικού χαρακτήρα. Η λειτουργία της είναι να μετράει τα μηνύματα που δέχεται και να προωθεί στην έξοδό της τον αριθμό τους ως *IntegerData*. Ο κώδικας που την υλοποιεί είναι ο ακόλουθος:

```
public class CounterProcessor extends Processor {

    private int counter = 0;

    public void run() {
        // Start the infinite loop
        while (true) {
            // get the data from the queue
            Data data = queue.getOldestData();

            // Check that the queue wasn't empty. If it was skip the
            // rest of the loop and try to read again from the queue.
            if (data == null) {
                continue;
            }

            // Increase the counter
            counter ++;
```

```
            // Fire the event with the counter
            fireDataEvent(new DataEvent(this, new
                                                IntegerData(counter)));
        }
    }

    public void setParameters(Properties parameters) {
        // Do nothing
    }

    public String getDescription() {
        return "It counts the data ariving";
    }
}
```

Όπως φαίνεται και στον κώδικα, η υλοποίηση της κλάσης είναι ιδιαίτερα απλή. Η μεταβλητή *counter* αυξάνεται κατά ένα κάθε φορά που η κλάση δέχεται ένα μήνυμα και δημιουργείται ένα συμβάν που περιέχει ένα αντικείμενο *IntegerData* με την τιμή της.

5.9. Κλάση SpeedAverage

Η κλάση SpeedAverage είναι ένας Processor ο οποίος χρησιμοποιείται για τον υπολογισμό του μέσου όρου της ταχύτητας των torque μηνυμάτων. Πιο συγκεκριμένα, η κλάση δέχεται μηνύματα τύπου *TorqueData* και αντικαθιστά την παράμετρο της ταχύτητας με το μέσο όρο της ταχύτητας των τελευταίων *n* μηνυμάτων, όπου ο αριθμός *n* ορίζεται από το χρήστη μέσω μίας παραμέτρου. Η κλάση αυτή είναι αναγκαία για την ορθή επικοινωνία των εφαρμογών Components και Torque επειδή η ταχύτητα στα μηνύματα που στέλνει η εφαρμογή Components μεταβάλλονται απότομα και ακανόνιστα (αφού στέλνεται η ταχύτητα της κίνησης των χεριών του χρήστη) και είναι απαραίτητη η εξομάλυνσή τους.

Για να υπολογίζεται ο μέσος όρος των ταχυτήτων τελευταίων *n* μηνυμάτων πρέπει με κάποιο τρόπο αυτές να αποθηκεύονται τοπικά στην κλάση *SpeedAverage*. Γι' αυτόν τον σκοπό δημιουργήθηκε μια εσωτερική κλάση, η *Storage*, η οποία περιέχει μία λίστα από αντικείμενα τύπου *double* και τις κατάλληλες μεθόδους για τη διαχείριση της λίστας.

```
private class Storage {

    double[] list;
    int size;
    int counter = 0;

    public Storage(int size) {
        list = new double[size];
        this.size = size;
```

```
    }

    public void addValue(double value) {
        list[counter] = value;
        counter++;
        if (counter == size) {
            counter = 0;
        }
    }

    public double getAverage() {
        double average = 0;
        for (int i = 0; i < size; i++) {
            average += list[i];
        }
        average = average / size;
        return average;
    }
}
```

Το αντικείμενο *list* είναι η λίστα στην οποία αποθηκεύονται οι τιμές της παραμέτρου της ταχύτητας των τελευταίων *n* μηνυμάτων. Όπως φαίνεται στη μέθοδο *addValue()* οι καινούργιες τιμές αποθηκεύονται κυκλικά στις επόμενες θέσεις της λίστας και δεν γίνονται μετακινήσεις των παλαιότερων στοιχείων στη λίστα, παρά μόνο το παλαιότερο αντικαθίσταται με το καινούργιο. Με αυτόν τον τρόπο η λίστα δεν είναι σωστά διατεταγμένη χρονολογικά, όμως επειδή χρησιμοποιούνται όλα τα στοιχεία της λίστας για τον υπολογισμό του μέσου όρου (όπως φαίνεται στη συνάρτηση *getAverage()*) το γεγονός αυτό δεν επηρεάζει τη λειτουργία της κλάσης (παρά μόνο την κάνει γρηγορότερη από κάποια που θα μετακινούσε τα στοιχεία για να τα βάλει στη σωστή σειρά).

Όπως αναφέρθηκε νωρίτερα, η κλάση *SpeedAverage* δέχεται μία παράμετρο, η οποία είναι το μήκος της λίστας του *Storage* αντικειμένου. Η τιμή της παραμέτρου αποθηκεύεται στην ακέραια μεταβλητή *noOfMessages* και ο τρόπος υλοποίησης της μεθόδου *setParameters()* για την απόδοση της τιμής της είναι όμοιος με όλων των προηγούμενων κλάσεων τύπου Reader, Processor και Writer, οπότε η παρουσίασή της είναι περιττή.

Στην υλοποίηση της μεθόδου *run()* ακολουθείται η εξής διαδικασία. Αρχικώς το εισερχόμενο μήνυμα μετατρέπεται σε αντικείμενο τύπου *TorqueData*. Εάν το μήνυμα δεν περιέχει την παράμετρο της ταχύτητας προωθείται αμέσως, ενώ εάν την περιέχει εξάγεται η τιμή της και προστίθεται στο αντικείμενο *storage*. Μετά δημιουργείται ένα νέο αντικείμενο τύπου *TorqueData* του οποίου όλες οι παράμετροι έχουν τις ίδιες τιμές με του εισερχόμενου, εκτός από την ταχύτητα, όπου έχει την τιμή που επιστρέφει η συνάρτηση *getAverage()* του αντικειμένου *storage*, και προωθείται. Τα ανωτέρω φαίνονται στον ακόλουθο κώδικα.

```
TorqueData data = (TorqueData) queue.getOldestData();

if (data == null) {
    continue;
}

// If we don't have speed multiplier send the message
if (data.getSpeedMultiplier() == null) {
    fireDataEvent(new DataEvent(this, data));
    continue;
}

storage.addValue(Double.parseDouble(data.getSpeedMultiplier()));
TorqueData newData = new TorqueData(data.getRefresh(),
        data.getClientGameConnection(),
        String.valueOf(storage.getAverage()),
        data.getSizeMultiplier(), data.getLightning(),
        data.getLightningDuration(), data.getAmbush(),
        data.getNumOfEnemies(), data.getAnimation());

// Here we know the data should be forwarded, so do it
fireDataEvent(new DataEvent(this, newData));
```

5.10. Κλάση DivideSpeed

Η κλάση *DivideSpeed* είναι ένας Processor ο οποίος χρησιμοποιείται για τη διαίρεση της παραμέτρου της ταχύτητας στα μηνύματα του προγράμματος torque με κάποιον αριθμό που δίνει ο χρήστης. Η υλοποίηση αυτής της κλάσης είναι απαραίτητη επειδή το εύρος των τιμών που στέλνει το πρόγραμμα Components είναι αρκετά μεγαλύτερο από αυτό που χρησιμοποιεί το πρόγραμμα torque.

Η κλάση δέχεται μία παράμετρο τύπου *double*, την divideBy, η οποία ορίζει τον αριθμό με τον οποίο θα διαιρούνται οι τιμές της ταχύτητας. Ο κώδικας που υλοποιεί τη μέθοδο *setParameters()* είναι όμοιος με τον κώδικα των προηγούμενων κλάσεων, οπότε δεν θα παρουσιαστεί. Η υλοποίηση της μεθόδου *run()* είναι ιδιαίτερα απλή. Όταν η κλάση δέχεται ένα μήνυμα *TorqueData* δημιουργεί ένα νέο με τις ίδιες τιμές, αντικαθιστώντας μόνο την παράμετρο της ταχύτητας (όπως και στην προηγούμενη κλάση.

```
TorqueData data = (TorqueData) queue.getOldestData();

if (data == null) {
    continue;
}

// If we don't have speed multiplier send the message
if (data.getSpeedMultiplier() == null) {
    fireDataEvent(new DataEvent(this, data));
```

```
    continue;
}

// Calculate the new speed multiplier
double newValue = (Double.parseDouble(data.getSpeedMultiplier())) /
                                                        divideBy;
TorqueData newData = new TorqueData(data.getRefresh(),
                    data.getClientGameConnection(),
                    String.valueOf(newValue),
                    data.getSizeMultiplier(), data.getLightning(),
                    data.getLightningDuration(), data.getAmbush(),
                    data.getNumOfEnemies(), data.getAnimation());

// Here we know the data should be forwarded, so do it
fireDataEvent(new DataEvent(this, newData));
```

5.11. Κλάση AmsProcessor

Η κλάση *AmsProcessor* είναι ένας processor ο οποίος δέχεται μηνύματα τύπου *TorqueData*, εξάγει την τιμή της ταχύτητας, και τη μετατρέπει σε τιμή κατάλληλη για το πρόγραμμα Callas AMS. Το πρόγραμμα Callas AMS δέχεται τιμές στο εύρος 0,5 έως 6, όπου η τιμή 0,5 αντιστοιχεί σε αργό παίξιμο της μουσικής και η τιμή 6 σε γρήγορο. Η εξίσωση για τον υπολογισμό της τιμής που στέλνεται είναι η ακόλουθη:

ταχύτητα AMS = 6 – (ταχύτητα torque / εύρος ταχύτητας) * 6

όπου το εύρος της ταχύτητας είναι η αναμενόμενη μέγιστη τιμή της ταχύτητας στα μηνύματα torque που δέχεται η κλάση και ορίζεται από το χρήστη ως παράμετρος (για να μπορεί να χρησιμοποιηθεί ο processor και με άλλα προγράμματα εκτός από το Components, με διαφορετικό εύρος τιμών). Η υλοποίηση της μεθόδου run είναι η ακόλουθη:

```
Data data = queue.getOldestData();

if (data == null) {
    continue;
}

// Get the speed multiplier value
TorqueData torqueData = (TorqueData)data;
String speedMulitplierString = torqueData.getSpeedMultiplier();
if (speedMulitplierString == null) {
    continue;
}
double speedMultiplier = Double.parseDouble(speedMulitplierString);

// Calculate the value to send to the AMS program
speedMultiplier = (speedMultiplier / inputRange) * 6.;
speedMultiplier = 6 - speedMultiplier;
```

```
speedMultiplier += 0.5;
if (speedMultiplier > 6) {
    speedMultiplier = 6;
}

// Create the data to send
Data speedData = new DoubleData(speedMultiplier);

// Here we know the data should be forwarded, so do it
fireDataEvent(new DataEvent(this, speedData));
```

Όπως φαίνεται στον κώδικα, αν η τιμή που υπολογιστεί είναι μικρότερη από 0,5 ή μεγαλύτερη από 6, στέλνονται οι ακραίες αυτές τιμές. Αφού υπολογιστεί η ταχύτητα, δημιουργείται ένα νέο αντικείμενο τύπου *DoubleData* και προωθείται με τη χρήση της συνάρτησης *fireDataEvent()*.

5.12. Κλάση TorqueWriter

Η κλάση *TorqueWriter* είναι ένας writer για την αποστολή μηνυμάτων στο πρόγραμμα Torque. Η κλάση δέχεται δύο παραμέτρους για τον ορισμό του host στον οποίο εκτελείται το πρόγραμμα Torque και το port στο οποίο αναμένει για μηνύματα.

Όπως έχει αναφερθεί προηγουμένως, για την επικοινωνία με το πρόγραμμα Torque θα χρησιμοποιηθεί ένα αντικείμενο τύπου *Socket*, το οποίο θα δημιουργήσει τη σύνδεση με τον server και θα παρέχει τα κατάλληλα streams για τη μεταφορά των μηνυμάτων. Η σύνδεση αυτή γίνεται στη μέθοδο initializeConnection(), η οποία αρχικοποιεί τη σύνδεση και το stream που θα χρησιμοποιηθεί. Στην περίπτωση που κάποια ενέργεια αποτύχει, οπότε η σύνδεση δεν είναι επιτυχής, η μέθοδος επιστρέφει την τιμή *false*.

```
private boolean initializeConnection() {
    try {
        socket = new Socket(host, port);
    } catch (UnknownHostException ex) {
        return false;
    } catch (IOException ex) {
        return false;
    }
    try {
        out = new PrintWriter(socket.getOutputStream(), true);
    } catch (IOException ex) {
        return false;
    }
    return true;
}
```

Η υλοποίηση της μεθόδου *run()* είναι αρκετά απλή. Αρχικά χρησιμοποιεί τη μέθοδο *initializeConnection()* για να συνδεθεί με το server. Στην περίπτωση που η σύνδεση δεν είναι

επιτυχής, το σύστημα περιμένει για ένα δευτερόλεπτο και επιχειρεί να συνδεθεί ξανά. Αυτή η διαδικασία συνεχίζεται μέχρι να επιτευχθεί η σύνδεση. Αμέσως μετά δημιουργείται ένα άπειρο loop στο οποίο διαβάζονται τα μηνύματα από την ουρά *queue* και στέλνονται στο server μέσω του stream *out*. Λόγω του τρόπου υλοποίησης της μεθόδου *toString()* της κλάσης *TorqueData* η αποστολή του μηνύματος γίνεται με την εντολή *out.println(data.toString())*.

```
while (!initializeConnection()) {
    try {
        Thread.sleep(1000);
    } catch (InterruptedException ex) {
        ex.printStackTrace();
    }
}

// Start the infinite loop
while (true) {
    // get the data from the queue
    Data data = queue.getOldestData();

    // Check that the queue wasn't empty. If it was skip the rest of
    // the loop and try to read again from the queue.
    if (data == null) {
        continue;
    }

    // Send the data throught the socket
    out.println(data.toString());
}
```

5.13. Κλάση AmsWriter

Η κλάση *AmsWriter* είναι ένας writer για την αποστολή μηνυμάτων στο πρόγραμμα Callas AMS. Η λογική της κλάσης είναι παρόμοια με της προηγούμενης, αφού και οι δύο υλοποιούν το client κομμάτι της TCP/IP επικοινωνίας, με μικρές διαφορές. Ο τρόπος σύνδεσης είναι ο ίδιος, μόνο που αλλάζει ο τρόπος κατασκευής του μηνύματος που στέλνεται.

Τα μηνύματα που δέχεται η κλάση είναι τύπου *DoubleData* και αρχικώς μετατρέπονται σε αλφαριθμητικά. Επειδή δεν χρειάζονται περισσότερα από δύο δεκαδικά ψηφία, τα υπόλοιπα αφαιρούνται.

```
String valueString = Double.toString(((DoubleData) data).getValue());
if (valueString.indexOf(".") + 3 <= valueString.length()) {
    valueString = valueString.substring(0,valueString.indexOf(".") + 3);
}
```

Το επόμενο πρόβλημα είναι ότι το πρόγραμμα Callas AMS αναγνωρίζει το χαρακτήρα που χωρίζει

τα δεκαδικά ψηφία ανάλογα με την έκδοση του λειτουργικού συστήματος. Δηλαδή εάν το πρόγραμμα εκτελείται σε έναν υπολογιστή με αγγλική έκδοση του λειτουργικού χρησιμοποιεί την τελεία, ενώ εάν το λειτουργικό είναι ελληνική έκδοση χρησιμοποιεί το κόμμα. Επειδή δεν είναι δυνατόν να αναγνωριστεί η έκδοση του λειτουργικού συστήματος του απομακρυσμένου host, ο χρήστης πρέπει να παρέχει αυτήν την πληροφορία μέσω μιας παραμέτρου, την decimalSeparator.

```
valueString = valueString.replace('.', decimalSeparator.charAt(0));
```

Τέλος, δημιουργείται και αποστέλλεται το μήνυμα στον server.

```
// Create the XML String to send
String xmlString = "play <AMSCase Name=\"Miodzio\">" +
        "<PADModel Pleasure=\"-1\" Arousal=\"-1\" Dominance=\"-1\" />" +
        "<MusicSample SamplePath=\"miodzio.mid\" Pitch=\"0\" Tempo=" +
        "\"" + valueString + "\" " +
        "Volume=\"100\" />" +
        "</AMSCase>";

// Send the data throught the socket
out.println(xmlString);
```

6. Επίλογος

6.1. Σύνοψη και συμπεράσματα

Στα προηγούμενα κεφάλαια παρουσιάστηκε η ανάλυση, η σχεδίαση και η υλοποίηση ενός συστήματος διακίνησης και επεξεργασίας μηνυμάτων μεταξύ εφαρμογών. Κατά τη διάρκεια της ανάλυσης ορίστηκαν οι λειτουργικές προδιαγραφές του συστήματος και εντοπίστηκαν από αυτές οι περιπτώσεις χρήσεως και τα πιθανά σενάρια. Κατά τη διάρκεια της σχεδίασης, με τη χρήση του αποτελέσματος της ανάλυσης, εντοπίστηκαν οι κλάσεις οι οποίες είναι απαραίτητες για την υλοποίηση του συστήματος, καθώς και οι σχέσεις μεταξύ τους. Τέλος παρουσιάζεται ο τρόπος υλοποίησης των κλάσεων αυτών με τη χρήση της προγραμματιστικής πλατφόρμας Java.

Το αποτέλεσμα των ανωτέρω είναι ένα σύστημα το οποίο ικανοποιεί πλήρως τις προσδοκίες που υπήρχαν για το σύστημα πριν τη διεκπεραίωση της διπλωματικής. Αυτό προκύπτει και από τη χρήση του συστήματος για τη διακίνηση μηνυμάτων μεταξύ τριών ήδη υπαρχουσών εφαρμογών. Τα οφέλη της χρήσης του συστήματος σε σχέση με τον παραδοσιακό τρόπο επικοινωνίας των εφαρμογών είναι κυρίως τρία. Η επικοινωνία εφαρμογών οι οποίες χρησιμοποιούν διαφορετικά πρωτόκολλα επικοινωνίας, η προώθηση των μηνυμάτων μιας εφαρμογής σε περισσότερες από μία διαφορετικές εφαρμογές και η στοιχειώδης επεξεργασία των μηνυμάτων.

Όλα τα ανωτέρω, καθώς και η απλότητα χρήσης του συστήματος, καθιστούν το σύστημα ως ένα

χρήσιμο εργαλείο στη διάθεση οποιουδήποτε επιθυμεί να μεταφέρει μηνύματα μεταξύ εφαρμογών οι οποίες δεν έχουν σχεδιαστεί γι' αυτόν τον σκοπό.

6.2. Μελλοντικές επεκτάσεις

Λόγω της φύσης του συστήματος, οι επεκτάσεις που μπορούν να υλοποιηθούν είναι απεριόριστες. Στην πραγματικότητα κάθε χρήση του συστήματος θα αποτελεί και μία επέκταση αυτού, αφού θα έχει ως αποτέλεσμα τη δημιουργία νέων στοιχείων Readers, Processors και Writers τα οποία θα είναι επαναχρησιμοποιήσιμα. Γι' αυτό το λόγο είναι απαραίτητο να τονιστεί η σπουδαιότητα της σωστής συμπλήρωσης της java τεκμηρίωσης όλων των κλάσεων που υλοποιεί ο χρήστης.

Ο τομέας στον οποίο μπορούν να γίνουν επεκτάσεις στο κύριο κομμάτι του συστήματος είναι οι διεπαφές του χρήστη. Οι διεπαφές αυτές μπορούν να αποκτήσουν ένα πιο δυναμικό χαρακτήρα. Για παράδειγμα ο χρήστης θα μπορούσε να επιλέγει κάποιο στοιχείο κατά τη διάρκεια εκτέλεσης του συστήματος και να του δίνεται η επιλογή να ξεκινήσει ή να σταματήσει τη λειτουργία του. Χρήσιμη θα ήταν επίσης η δυνατότητα να μπορεί ο χρήστης να προσθέτει και να αφαιρεί συνδέσεις μεταξύ των στοιχείων. Τέλος ενδιαφέρον έχει και η ιδέα να μπορεί ο χρήστης να προσθέτει και να αφαιρεί στοιχεία κατά τη διάρκεια εκτέλεσης του συστήματος χωρίς να είναι απαραίτητη η επανεκκίνησή του.

7. Παραρτήματα

Παράρτημα Α : Εγχειρίδιο Χρήσης

Για την επιτυχή χρήση του συστήματος, πρέπει αρχικώς να συλλεχθούν πληροφορίες για τα πρωτόκολλα επικοινωνίας που χρησιμοποιούν οι εφαρμογές προς σύνδεση, καθώς και πληροφορίες για τον τύπο των μηνυμάτων που στέλνουν ή δέχονται οι εφαρμογές αυτές. Με αυτές τις πληροφορίες θα κατασκευαστούν ορισμένες Java κλάσεις καθώς και ένα XML αρχείο με τη χρήση των οποίων το σύστημα θα συνδέσει τις εφαρμογές.

Α.1. Κλάσεις τύπου Data

Για κάθε τύπο μηνύματος από ή προς κάποια εφαρμογή προς σύνδεση είναι απαραίτητη η υλοποίηση μιας κλάσης η οποία θα υλοποιεί τη διεπαφή *gr.ntua.ece.miodzio.data*. Οι κλάσεις αυτές θα αντιπροσωπεύουν το περιεχόμενο των μηνυμάτων εσωτερικά στο σύστημα και κάθε μία από αυτές πρέπει να περιέχει τις ίδιες πληροφορίες με τα μηνύματα.

Οι κανόνες που πρέπει να ακολουθεί κάθε κλάση τύπου *Data* είναι οι ακόλουθοι:

1. Τα δεδομένα που αντιπροσωπεύει η κλάση πρέπει να αποθηκεύονται σε αντικείμενα μέλη της, τα οποία να μην είναι προσβάσιμα από οποιαδήποτε άλλη κλάση.

2. Η κλάση δεν πρέπει να επιτρέπει τη μετατροπή των δεδομένων που περιέχει μετά την αρχικοποίησή της.

3. Η κλάση πρέπει να παρέχει για την αρχικοποίηση των αντικειμένων της έναν constructor ο οποίος θα δέχεται ως ορίσματα τα δεδομένα που αντιπροσωπεύει η κλάση.

4. Η κλάση πρέπει να παρέχει μεθόδους για την ανάκληση των τιμών των δεδομένων που αντιπροσωπεύει. Ιδιαίτερη προσοχή πρέπει να δίνεται στα επιστρεφόμενα αντικείμενα, ώστε να μην είναι δυνατή η μετατροπή των δεδομένων μέσω αυτών.

5. Η μέθοδος *toString()* πρέπει να επαναγράφεται για να αντιπροσωπεύει τα δεδομένα της κλάσης.

6. Στη Java τεκμηρίωση της κλάσης (javadoc) πρέπει να περιγράφεται με ακρίβεια το είδος των δεδομένων που αντιπροσωπεύει η κλάση.

Ένα παράδειγμα υλοποίησης μιας *Data* κλάσης είναι η κλάση *ExampleData* που βρίσκεται στο πακέτο *gr.ntua.ece.miodzio.example* και προτείνεται η χρήση αυτής της κλάσης ως σκελετό για τη δημιουργία νέων *Data* κλάσεων.

Α.2. Κλάσεις τύπου Reader

Για την επικοινωνία με κάθε εφαρμογή η οποία στέλνει μηνύματα στο σύστημα είναι απαραίτητη η υλοποίηση μιας κλάσης που θα κληρονομεί την κλάση *gr.ntua.ece.miodzio.elements.Reader*. Οι κλάσεις αυτές είναι υπεύθυνες για την υλοποίηση του πρωτοκόλλου επικοινωνίας με τις εφαρμογές και τη μετατροπή της πληροφορίας των μηνυμάτων από αυτές σε αντικείμενα τύπου *Data*. Οποιαδήποτε παραμετροποίηση του πρωτοκόλλου επικοινωνίας πρέπει να γίνεται μέσω της μεθόδου *setParameters()* η οποία χρησιμοποιείται από το σύστημα με τις τιμές που δίνονται στο XML αρχείο που θα αναφερθεί αργότερα.

Οι κανόνες που πρέπει να ακολουθεί κάθε κλάση τύπου *Reader* είναι οι ακόλουθοι:

1. Πρέπει να υλοποιείται η μέθοδος *setParameters()* της διεπαφής *Configurable* για να καθορίζει τις παραμέτρους της. Εάν κάποια παράμετρος λείπει ή η τιμή της δεν είναι σωστή πρέπει να δημιουργείται ένα σφάλμα τύπου *ParameterProblemException* το οποίο να περιγράφει το πρόβλημα.

2. Πρέπει να υλοποιείται η μέθοδος *run()* της διεπαφής *Runnable* με ένα άπειρα επαναλαμβανόμενο loop, στο οποίο θα δέχεται μηνύματα από τις εφαρμογές εισόδου, θα δημιουργεί ένα αντικείμενο *Data* από αυτά και θα στέλνει ένα *DataEvent* στις κλάσεις που

περιμένουν για συμβάντα, χρησιμοποιώντας τη συνάρτηση *fireDataEvent()*.

3. Πρέπει να υλοποιείται η μέθοδος *getDescription()* της διεπαφής *Describable* ώστε να επιστρέφει μια περιγραφή της λειτουργίας της κλάσης.

Ένα παράδειγμα υλοποίησης μιας *Writer* κλάσης είναι η κλάση *ExampleWriter* που βρίσκεται στο πακέτο *gr.ntua.ece.miodzio.example* και προτείνεται η χρήση αυτής της κλάσης ως σκελετό για τη δημιουργία νέων *Writer* κλάσεων.

Α.3. Κλάσεις τύπου Processor

Για κάθε περίπτωση που είναι απαραίτητη η επεξεργασία ενός μηνύματος πριν την αποστολή του προς κάποια εφαρμογή εξόδου, πρέπει να υλοποιείται μια κλάση η οποία κληρονομεί την κλάση *gr.ntua.ece.miodzio.elements.Processor*. Οι περιπτώσεις κατά τις οποίες χρειάζεται η χρήση κάποιου *Processor* μπορεί να είναι απλές, όπως είναι η μετατροπή ενός τύπου *Data* σε κάποιον άλλο, ή αρκετά πιο σύνθετες, όπως είναι η εξαγωγή στατιστικών στοιχείων. Οι διάφορες κλάσεις τύπου *Processor* μπορούν επίσης να συνδυάζονται για την εκτέλεση περίπλοκων επεξεργασιών.

Οι κανόνες που πρέπει να ακολουθεί κάθε κλάση τύπου *Processor* είναι οι ακόλουθοι:

1. Πρέπει να υλοποιείται η μέθοδος *setParameters()* της διεπαφής *Configurable* για να καθορίζει τις παραμέτρους της. Εάν κάποια παράμετρος λείπει ή η τιμή της δεν είναι σωστή πρέπει να δημιουργείται ένα σφάλμα τύπου *ParameterProblemException* το οποίο να περιγράφει το πρόβλημα.

2. Πρέπει να υλοποιείται η μέθοδος *run()* της διεπαφής *Runnable* με ένα άπειρα επαναλαμβανόμενο loop, στο οποίο θα δέχεται μηνύματα από άλλα στοιχεία του συστήματος με τη χρήση της συνάρτησης *queue.getOldestData()*, θα δημιουργεί ένα αντικείμενο *Data* από αυτά και θα στέλνει ένα *DataEvent* στις κλάσεις που περιμένουν για συμβάντα, χρησιμοποιώντας τη συνάρτηση *fireDataEvent()*.

3. Πρέπει να υλοποιείται η μέθοδος *getDescription()* της διεπαφής *Describable* ώστε να επιστρέφει μια περιγραφή της λειτουργίας της κλάσης.

Ένα παράδειγμα υλοποίησης μιας *Processor* κλάσης είναι η κλάση *ExampleProcessor* που βρίσκεται στο πακέτο *gr.ntua.ece.miodzio.example* και προτείνεται η χρήση αυτής της κλάσης ως σκελετό για τη δημιουργία νέων *Processor* κλάσεων.

Α.4. Κλάσεις τύπου Writer

Για την επικοινωνία με κάθε εφαρμογή η οποία δέχεται μηνύματα από το σύστημα είναι

απαραίτητη η υλοποίηση μιας κλάσης που θα κληρονομεί την κλάση *gr.ntua.ece.miodzio.elements.Writer*. Οι κλάσεις αυτές είναι υπεύθυνες για την υλοποίηση του πρωτοκόλλου επικοινωνίας με τις εφαρμογές και την αποστολή μηνυμάτων σε αυτές.

Οι κανόνες που πρέπει να ακολουθεί κάθε κλάση τύπου *Writer* είναι οι ακόλουθοι:

1. Πρέπει να υλοποιείται η μέθοδος *setParameters()* της διεπαφής *Configurable* για να καθορίζει τις παραμέτρους της. Εάν κάποια παράμετρος λείπει ή η τιμή της δεν είναι σωστή πρέπει να δημιουργείται ένα σφάλμα τύπου *ParameterProblemException* το οποίο να περιγράφει το πρόβλημα.

2. Πρέπει να υλοποιείται η μέθοδος *run()* της διεπαφής *Runnable* με ένα άπειρα επαναλαμβανόμενο loop, στο οποίο θα δέχεται μηνύματα από άλλα στοιχεία του συστήματος με τη χρήση της συνάρτησης *queue.getOldestData()* και θα τα στέλνει στις εφαρμογές χρησιμοποιώντας το πρωτόκολλό τους.

3. Πρέπει να υλοποιείται η μέθοδος getDescription() της διεπαφής Describable ώστε να επιστρέφει μια περιγραφή της λειτουργίας της κλάσης.

Ένα παράδειγμα υλοποίησης μιας *Writer* κλάσης είναι η κλάση *ExampleWriter* που βρίσκεται στο πακέτο *gr.ntua.ece.miodzio.example* και προτείνεται η χρήση αυτής της κλάσης ως σκελετό για τη δημιουργία νέων *Writer* κλάσεων.

Α.5. Αρχείο XML

Ο τρόπος σύνδεσης των ανωτέρω κλάσεων μεταξύ τους για τον ορισμό της πορείας που ακολουθούν τα μηνύματα δίνεται στο σύστημα μέσω ενός XML αρχείου.

Το αρχικό στοιχείο του XML αρχείου πρέπει να ονομάζεται MIODZIO και πρέπει να περιέχει τρία υποστοιχεία με ονόματα READERS, PROCESSORS και WRITERS, τα οποία θα περιέχουν τις πληροφορίες για τις κλάσεις που επικοινωνούν με τις εφαρμογές εισόδου, που επεξεργάζονται τα μηνύματα και που επικοινωνούν με τις εφαρμογές εξόδου αντίστοιχα.

Το κάθε στοιχείο READER, το οποίο αντιπροσωπεύει μία κλάση επικοινωνίας με κάποια εφαρμογή εισόδου, πρέπει να έχει ως χαρακτηριστικά (attributes) το όνομα της κλάσης μαζί με το java path της (για παράδειγμα gr.ntua.ece.miodzio.example.ExampleReader) ώστε το σύστημα να ξέρει ποια κλάση να χρησιμοποιήσει, και κάποιο αναγνωριστικό String για περαιτέρω αναφορά. Ως υποστοιχεία, πρέπει να έχει δύο λίστες, μία με τις παραμέτρους για την παραμετροποίηση της κλάσης και μία με τα στοιχεία του συστήματος τα οποία θα ενημερώνονται για τα συμβάντα από αυτήν την κλάση.

Το κάθε στοιχείο PROCESSOR, το οποίο αντιπροσωπεύει μία κλάση επεξεργασίας μηνυμάτων, πρέπει να έχει τα ίδια χαρακτηριστικά και υποστοιχεία με τα στοιχεία READER, αλλά επιπλέον μπορεί να έχει και ένα παραπάνω χαρακτηριστικό, το οποίο θα ορίζει το μέγεθος της ουράς αναμονής. Το χαρακτηριστικό αυτό δεν είναι υποχρεωτικό και στην περίπτωση που λείπει, η ουρά θα έχει απεριόριστο μέγεθος.

Τα στοιχεία WRITER, τα οποία αντιπροσωπεύουν κλάσεις επικοινωνίας με τις εφαρμογές εξόδου, έχουν τα ίδια χαρακτηριστικά και υποστοιχεία με τα στοιχεία PROCESSOR, με τη διαφορά ότι δεν έχουν λίστα με στοιχεία τα οποία θα ενημερώνουν για συμβάντα, αφού δεν δημιουργούν συμβάντα.

Η κάθε λίστα παραμέτρων περιέχει στοιχεία PARAMETER, τα οποία αντιπροσωπεύουν τις παραμέτρους της κλάσης. Κάθε παράμετρος έχει ως χαρακτηριστικό το όνομά της και περιέχει ένα αλφαριθμητικό, όπου είναι η τιμή της παραμέτρου.

Τέλος, η κάθε λίστα με στοιχεία για ενημέρωση για συμβάντα θα περιέχει στοιχεία OUTPUT, τα οποία απλώς περιέχουν αλφαριθμητικά με τα αναγνωριστικά των κλάσεων προς ενημέρωση στο XML αρχείο.

Ένας σκελετός που μπορεί να χρησιμοποιηθεί για την δημιουργία νέων αρχείων είναι ο ακόλουθος:

```
<?xml version="1.0"?>
<!DOCTYPE MIODZIO SYSTEM "miodzio.dtd">

<MIODZIO>
    <READERS>
        <READER CLASS="" OBJECTID="">
            <PARAMETERS>
                <PARAMETER NAME=""></PARAMETER>
            </PARAMETERS>
            <OUTPUTS>
                <OUTPUT></OUTPUT>
            </OUTPUTS>
        </READER>
    </READERS>
    <PROCESSORS>
        <PROCESSOR CLASS="" OBJECTID="" QUEUESIZE="">
            <PARAMETERS>
                <PARAMETER NAME=""></PARAMETER>
            </PARAMETERS>
            <OUTPUTS>
                <OUTPUT></OUTPUT>
            </OUTPUTS>
        </PROCESSOR>
    </PROCESSORS>
```

```
    <WRITERS>
        <WRITER CLASS="" OBJECTID="" QUEUESIZE="">
            <PARAMETERS>
                <PARAMETER NAME=""></PARAMETER>
            </PARAMETERS>
        </WRITER>
    </WRITERS>
</MIODZIO>
```

A.6. Εκτέλεση του συστήματος

Για την εκτέλεση του συστήματος μετά τη δημιουργία των ανωτέρω πρέπει να εκτελεστούν τα επόμενα βήματα:

1. Τοποθέτηση του αρχείου miodzio.jar σε κάποιον γνωστό φάκελο
2. Τοποθέτηση των κλάσεων του χρήστη σε κάποιον γνωστό φάκελο
3. Τοποθέτηση του XML αρχείου μαζί με το αρχείο miodzio.dtd σε κάποιον γνωστό φάκελο
4. Για εκτέλεση χωρίς διεπαφές του χρήστη:
   ```
   java -cp <PATH για miodzio.jar>:<PATH για κλάσεις χρήστη> \
   -Dmiodzio.xmlFile=<PATH για αρχείο XML> \
   gr.ntua.ece.miodzio.Miodzio
   ```
5. Για εκτέλεση με διεπαφές του χρήστη:
   ```
   java -cp <PATH για miodzio.jar>:<PATH για κλάσεις χρήστη> \
   -Dmiodzio.xmlFile=<PATH για αρχείο XML> \
   gr.ntua.ece.miodzio.gui.MiodzioGui
   ```

8. Βιβλιογραφία

[Blo01] Joshua Bloch, “Effective Java”, Addison-Wesley, 2001, ISBN: 0201310058.

[Boo05] Grady Booch, “The Unified Modeling Language User Guide Second Edition”, Addison-Wesley, 2005, ISBN: 0321267974.

[CWH+01] Mary Campione, Kathy Walrath, Alison Huml, Tutorial team, “The Java Tutorial, Third Edition: A Short Course on the Basics”, Addison-Wesley, 2001, ISBN: 0201703939. Also available online at <http://java.sun.com/docs/books/tutorial/index.html>.

[CWH+98] Mary Campione, Kathy Walrath, Alison Huml, Tutorial Team, “The Java Tutorial Continued: The Rest of the JDK”, Addison-Wesley, 1998, ISBN: 0201485583. Also available online at <http://java.sun.com/docs/books/tutorial/index.html>.

[Eck05] Bruce Eckel, “Thinking in Java, 4th Edition”, Prentice Hall, 2005, ISBN: 0131872486.

[Har04] Elliotte Rusty Harold, “Java Network Programming”, O'Reilly, 2004, ISBN: 0596007213.

[HRF+07] David Hunter, Jeff Rafter, Joe Fawcett, Eric van der Vlist DannyAyers, Jon Duckett, “Beginning XML, 4th Edition”, John Wiley & Sons, 2007, ISBN: 0470114878.

[HSH99] Merlin Hughes, Michael Shoffner, Derek Hamner, “Java Network Programming”, Manning, 1999, ISBN: 188477749X.

[LF04] Brett McLaughlin, David Flanagan, “Java 5.0 Tiger: A Developer's Notebook”, O'Reilly, 2004, ISBN: 0596007388.

[Lin04] Peter van der Linden, “Just Java 2 Sixth Edition”, Prentice Hall, 2004, ISBN: 0131482114.

[Pil06] Dan Pilone, “UML 2.0 Pocket Reference”, O'Reilly, 2006, ISBN: 0596102089.

[Ray03] ET Ray, “Learning XML, 2nd Edition”, O'Reilly, 2003, ISBN: 0596004206.

[Ste90] W. Richard Stevens, “Unix Network Programming”, Prentice Hall, 1990, ISBN: 0139498761.

[Sun04] Sun Microsystems Inc., “Java 2 Platform Standard Edition 5.0 API Specification”, available online at <http://java.sun.com/j2se/1.5.0/docs/api>, 2004.

[WCHZ04] Kathy Walrath, Mary Campione, Alison Huml, Sharon Zakhour, “The JFC Swing Tutorial: A Guide to Constructing GUIs, Second Edition”, Addison-Wesley, 2004, ISBN: 0201914670. Also available online at <http://java.sun.com/docs/books/tutorial/index.html>.