ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδίαση και Υλοποίηση διεπαφής για τον έλεγχο των δεσμών των σωματιδίων του Βόρειου Τμήματος του επιταχυντή SPS στο CERN

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΣΤΑΜΑΤΕΛΟΥ-ΜΑΥΡΟΜΙΧΑΛΗ ΧΡΗΣΤΟΥ

Επιβλέπων : Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2009

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

Σχεδίαση και Υλοποίηση διεπαφής για τον έλεγχο των δεσμών των σωματιδίων του Βόρειου Τμήματος του επιταχυντή SPS στο CERN

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΣΤΑΜΑΤΕΛΟΥ-ΜΑΥΡΟΜΙΧΑΛΗ ΧΡΗΣΤΟΥ

Επιβλέπων : Τιμολέων Σελλής
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή τον Οκτώβριο 2009.

.....	.....	.....
Τιμολέων Σελλής	Νεκτάριος Κοζύρης	Νικόλαος Παπασπύρου
Καθηγητής Ε.Μ.Π.	Αναπλ. Καθηγητής Ε.Μ.Π.	Επίκ. Καθηγητής Ε.Μ.Π.

Αθήνα, Οκτώβριος 2009

.....

ΣΤΑΜΑΤΕΛΟΣ-ΜΑΥΡΟΜΙΧΑΛΗΣ ΧΡΗΣΤΟΣ

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

Πρόλογος - Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στον ερευνητικό οργανισμό Φυσικής CERN, κατά την διάρκεια πρακτικής άσκησης που διεκπεραίωσα την χρονική περίοδο 1/4/2008-15/11/2008. Στενή υπήρξε και η συνεργασία με το Εργαστήριο Συστημάτων Βάσεων και Γνώσεων Δεδομένων (ΕΒΓΔ) του τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου.

Κατ' αρχάς, θέλω να ευχαριστήσω τον οργανισμό CERN που με δέχτηκε με υποτροφία και συγκεκριμένα τον καθηγητή κ. Η. Ευθυμιόπουλο, ο οποίος με προσέλαβε ως επιβλέπων μου από την πλευρά του CERN και μου παρείχε σημαντικότατη βοήθεια και υποστήριξη κατά την διάρκεια της παραμονής μου στον οργανισμό. Ακόμα, ευχαριστώ πολύ τον καθηγητή V. Baggiolini, Μηχανικό Λογισμικού στο CERN, για την πολύτιμη καθοδήγηση που μου προσέφερε καθώς και όλους τους άλλους εργαζόμενους του CERN με τους οποίους συνεργάστηκα.

Επίσης, θα ήθελα να ευχαριστώ θερμά τον καθηγητή κ. Τ. Σελλή για την επίβλεψη της εργασίας μου από την πλευρά του Πολυτεχνείου και για την πολύτιμη βοήθειά του ώστε να καταφέρω να εκπονήσω την διπλωματική αυτή εργασία στο CERN. Ταυτόχρονα, ευχαριστώ και όλους τους υπόλοιπους καθηγητές του ΕΜΠ που με ενθάρρυναν και βοήθησαν στο εγχείρημά μου αυτό.

Τέλος, δεν θα μπορούσα να μην ευχαριστήσω και την οικογένειά μου, η οποία με υποστήριξε καθ' όλη τη διάρκεια των σπουδών μου.

Οκτώβριος 2009

Σταματέλος-Μαυρομιχάλης Χρήστος

Περίληψη

Σκοπός της διπλωματικής εργασίας είναι η ανάλυση των απαιτήσεων, η σχεδίαση και η υλοποίηση μέρους του προγράμματος CESAR (“Cern Experimental areas SoftwAre Renovation”), που αναπτύσσει τα τελευταία χρόνια ο διεθνής οργανισμός CERN (Ευρωπαϊκό Κέντρο Πυρηνικής Έρευνας). Το CERN είναι ο μεγαλύτερος πόλος έρευνας στην Φυσική και συγκεκριμένα στον τομέα των Στοιχειωδών Σωματιδίων ή Υψηλών Ενεργειών.

Το πρόγραμμα CESAR έχει ως στόχο τον έλεγχο των δεσμών των σωματιδίων στον επιταχυντή SPS, έναν από τους σημαντικότερους επιταχυντές του οργανισμού. Συγκεκριμένα, η εργασία μου επικεντρώθηκε στο επίπεδο της παρουσίασης – διεπαφής με τους χρήστες (User Interface) και μπορεί να χωριστεί σε τρεις τομείς:

α) Σχεδίαση του γενικού framework μιας εφαρμογής (ονόματι επίσης CESAR)
που θα επιτρέπει σε κάποιους χρήστες να ελέγχουν τις δέσμες των σωματιδίων,

β) Παρουσίαση και διαχείριση της κατάστασης του εξοπλισμού μιας δέσμης και:

γ) Παρουσίαση και διαχείριση των αρχείων μιας δέσμης.

Τα (β), (γ) αναπτύχθηκαν ανεξάρτητα από το framework της εφαρμογής (α), αλλά προστέθηκαν στην εφαρμογή ως modules της.

Λέξεις Κλειδιά: CERN, Φυσική Στοιχειωδών Σωματιδίων, Επιταχυντής SPS, Δέσμες σωματιδίων, User Interface, Framework, Εξοπλισμός δέσμης, Αρχεία δέσμης, Modules

Abstract

The purpose of the diploma thesis is the requirements' specification, design and implementation of a part of the CESAR system ("Cern Experimental areas SoftwAre Renovation"), that the international organization CERN (European Organization for Nuclear Research) is developing the last few years. CERN is the largest place of research in Physics and in particular, in the field of Particle or High Energy Physics.

CESAR project aims at the beam line's control in SPS accelerator, one of the most important accelerators of the organization. Specifically, my work was concentrated on the presentation – user interface layer and can be separated in 3 fields:

a) Design of the general framework of an application (named CESAR as well) that would allow some users to control the beams of particles,

b) Presentation and management of the status of various equipment of a beam line and:

c) Presentation and management of beam files of a beam line.

Points (b) and (c) were developed independently from the framework of the application (a), but were added to the application as modules.

Keywords: CERN, Particle Physics, SPS accelerator, Particle Beams, User Interface, Framework, Beam equipment, Beam files, Modules

Πίνακας περιεχομένων

1

Εισαγωγή

Στο παρόν κεφάλαιο, στην παράγραφο 1.1 κάνουμε μια γενική περιγραφή του χώρου εφαρμογής της παρούσας διπλωματικής εργασίας, στην παράγραφο 1.2 αναλύουμε το αντικείμενο της διπλωματικής εργασίας και στην παράγραφο 1.3 γίνεται μια επισκόπηση της οργάνωσης του τόμου που κρατάτε στα χέρια σας.

1.1 Physics Computing

Τα τελευταία χρόνια, η έρευνα στο χώρο της Φυσικής έχει επικεντρωθεί στον κλάδο της Φυσικής Στοιχειωδών Σωματιδίων ή διαφορετικά, της Φυσικής Υψηλών Ενεργειών. Ο τομέας αυτός, όπως προδίδει το όνομά του, μελετά τα στοιχειώδη σωματίδια που συγκροτούν την ύλη καθώς και την συμπεριφορά (ακτινοβολία), και τις αλληλεπιδράσεις μεταξύ τους.

Όσο σημαντική είναι η έρευνα σε θεωρητικό επίπεδο, τόσο είναι και η πειραματική διαδικασία που επιτελείται σε ερευνητικά κέντρα, όπως το CERN, η οποία επιβεβαιώνει ή απορρίπτει διάφορες φυσικές θεωρίες, αλλά και δίνει κατευθύνσεις πολλές φορές στην ίδια την θεωρητική μελέτη της Φυσικής.

Η πειραματική αυτή διαδικασία απαιτεί πολύπλοκα και δαπανηρά συστήματα επιταχυντών σωματιδίων. Τα συστήματα αυτά είναι στην πραγματικότητα ειδικές μηχανικές διατάξεις, οι οποίες παράγουν πανίσχυρα ηλεκτρικά και μαγνητικά πεδία, ώστε τελικά να επιταχύνουν δέσμες στοιχειωδών σωματιδίων σε ταχύτητες που πλησιάζουν την ταχύτητα του φωτός. Τα πειράματα που πραγματοποιούνται σε τέτοιου είδους ερευνητικά κέντρα, μπορούν να

περιγραφούν, σε γενικές γραμμές, ως εξής: αφού επιταχυνθούν αρκετά δύο δέσμες σωματιδίων σε αντίθετες μεταξύ τους κατευθύνσεις μέσα σε έναν επιταχυντή, τότε αυτές συγκρούονται σε κάποιο σημείο του επιταχυντή. Το αποτέλεσμα αυτής της σύγκρουσης είναι άλλου τύπου σωματίδια, τα οποία «αναγνωρίζονται» από ειδικές διατάξεις, τους ανιχνευτές σωματιδίων, στο σημείο της σύγκρουσης. Έτσι, από αυτά τα πειράματα μπορούν να προκύψουν άγνωστα ως τώρα σωματίδια αλλά και συμπεράσματα για την συμπεριφορά γνωστών σωματιδίων.

Τέτοια πειράματα, ωστόσο, έχουν σημαντικές απαιτήσεις σε Πληροφοριακά Συστήματα και άλλα είδη εφαρμογών Πληροφορικής, όπως για παράδειγμα Προσομοίωση και Εξόρυξη Δεδομένων. Και αυτό για δύο κύριους λόγους. Αφ' ενός, ο όγκος των δεδομένων που προκύπτουν από τα πειράματα είναι πραγματικά τεράστιος (ενώ τα χρήσιμα δεδομένα είναι λίγα και διασκορπισμένα παντού σε όλον τον όγκο των αποτελεσμάτων των πειραμάτων). Και αφ' ετέρου ο έλεγχος του εξοπλισμού και η προσομοίωση φυσικών πειραμάτων απαιτούν σύγχρονα υπολογιστικά εργαλεία.

Επίσης, δεν είναι τυχαίο ότι μέσα από τέτοια πειράματα έχουν προκύψει μεγάλα οφέλη και για την ίδια την Πληροφορική, όπως η ανακάλυψη του World Wide Web (WWW) από τον Tim Berners-Lee το 1989 στο CERN αλλά και η διάδοση της τεχνολογίας Grid στις αρχές του 21^{ου} αιώνα για τις απαιτήσεις των πειραμάτων του κατασκευαζόμενου επιταχυντή LHC του ίδιου οργανισμού.

Όλα τα παραπάνω, έχουν καταστήσει την Πληροφορική άρρηκτα συνδεδεμένη με το χώρο της Φυσικής των Υψηλών Ενεργειών, με αποτέλεσμα η έρευνα στον τομέα αυτό της Φυσικής να χρησιμοποιεί πολλές μοντέρνες τεχνολογίες αλλά και να παράγει καινούργιες.

1.2 Αντικείμενο διπλωματικής

Η παρούσα Διπλωματική Εργασία εκπονήθηκε στο CERN, το μεγαλύτερο ερευνητικό κέντρο Φυσικής αυτήν την στιγμή στον κόσμο και επικεντρώθηκε σε ένα από τα θέματα που αναλύθηκαν στην προηγούμενη παράγραφο: τον έλεγχο του εξοπλισμού των επιταχυντών. Συγκεκριμένα, σκοπός μας είναι η ανάπτυξη μιας εφαρμογής, με τη βοήθεια της οποίας μπορεί να γίνει έλεγχος της δέσμης των σωματιδίων σε έναν από τους επιταχυντές του ερευνητικού κέντρου CERN (ονόματι SPS). Ο έλεγχος αυτός γίνεται μέσω του εξοπλισμού του επιταχυντή (για παράδειγμα ισχυροί μαγνήτες) και τον κάνουν οι πειραματικοί φυσικοί πριν εκπονήσουν διάφορα πειράματα, ώστε να καθορίσουν τις επιθυμητές ιδιότητες των δεσμών των σωματιδίων που θα χρησιμοποιήσουν (για παράδειγμα, τι είδους σωματίδια θέλουν να έχουν σε κάθε σημείο, την ενέργεια της δέσμης κλπ). Περισσότερες, όμως λεπτομέρειες θα δοθούν στο κεφάλαιο 3 του παρόντος τόμου.

Με τον όρο έλεγχο του εξοπλισμού του επιταχυντή, εννοούμε για παράδειγμα, να μπορούν οι πειραματικοί φυσικοί του οργανισμού να έχουν την εποπτεία των διαφόρων μηχανημάτων της δέσμης. Αυτό σημαίνει ότι εφαρμογές σαν την δική μας κάνουν διάφορες μετρήσεις (monitoring) πάνω στις παραμέτρους του εξοπλισμού και παρουσιάζουν με ακρίβεια και αξιοπιστία τα αποτελέσματα. Επίσης, πρέπει να φαίνεται και η κατάσταση, στην οποία βρίσκονται τα μηχανήματα αυτά. Επιπροσθέτως, οι χρήστες των εφαρμογών αυτών πρέπει να είναι σε θέση και να αλλάζουν (setting) τις παραμέτρους του εξοπλισμού, να δημιουργούν και να διαχειρίζονται αρχεία με αποθηκευμένες τις επιθυμητές για τα πειράματά τους παραμέτρους (αρχεία δέσμης) κλπ. Τέλος, όλες αυτές οι δυνατότητες (καθώς και άλλες πιο ειδικές) οφείλουν να είναι διαθέσιμες μέσα από ένα ενιαίο πρόγραμμα (software application), φιλικό και κατανοητό προς τους χρήστες, οι οποίοι δεν είναι τεχνολογικά ή επιστημονικά καταρτισμένοι ώστε να καταλάβουν περίπλοκες προγραμματιστικές ή φυσικές έννοιες.

Είναι σημαντικό να τονίσουμε ότι η εργασία μας επικεντρώθηκε στο υψηλό προγραμματιστικό επίπεδο της παρουσίασης, δηλαδή στην διεπαφή των διαθέσιμων προγραμματιστικών εργαλείων με τους απλούς χρήστες (φυσικούς). Έτσι, χρησιμοποιήσαμε αρκετό υπάρχοντα κώδικα στα πλαίσια του προγράμματος CESAR του οργανισμού, που μας προσέφερε έτοιμο API (Application Programming Interface), δηλαδή, μεθόδους χαμηλών επιπέδου ελέγχου (handlers) όπως αυτές που αλλάζουν τις παραμέτρους στον πραγματικό εξοπλισμό (hardware) ή τα αρχεία δεσμών που είναι αποθηκευμένα σε διάφορες βάσεις δεδομένων (server). Επίσης, χρησιμοποιήσαμε εργαλεία γραφικής απεικόνισης (GUI - Graphical User Interface) που είχαν αναπτυχθεί από συναδέλφους Μηχανικούς Λογισμικού που συμμετείχαν στο ίδιο πρόγραμμα.

Όλες αυτές τις απαιτήσεις ελέγχου του εξοπλισμού της δέσμης αναλάβαμε να υλοποιήσουμε μέσω μοντέρνων προγραμματιστικών εργαλείων και με τη βοήθεια του έμπειρου προσωπικού του CERN (διαφόρων ειδικοτήτων – κυρίως προγραμματιστών και ειδικευμένων φυσικών), οι οποίοι μας προσέφεραν την απαραίτητη γνώση όσον αφορά την Φυσική αλλά και υπάρχοντα κώδικα ως βάση ή παράδειγμα για την εργασία μας.

1.2.1 Συνεισφορά

Η συνεισφορά της διπλωματικής συνοψίζεται ως εξής:

1. Μελετήσαμε εφαρμογές παρόμοιων συστημάτων ελέγχου, το API που μας ήταν διαθέσιμο στα πλαίσια του CESAR καθώς και προηγούμενες εκδόσεις του προγράμματος CESAR.
2. Αλιεύσαμε πληροφορίες και γνώση από ειδικευμένους φυσικούς για τις απαιτήσεις του συστήματός μας, τον τρόπο χρήσης των διαφόρων μηχανημάτων και αρχείων

(καθώς και διάφορους περιορισμούς χρήσης τους) και οποιαδήποτε άλλη γνώση που θα έπρεπε να συμπεριλάβουμε στο πρόγραμμά μας.

3. Μοντελοποιήσαμε τον εξοπλισμό.
4. Σχεδιάσαμε και υλοποιήσαμε ένα φιλικό προς το χρήστη και αποδοτικό σύστημα.
5. Ελέγξαμε την ορθότητα του συστήματος αυτού.
6. Το προτείναμε στους ειδικευμένους φυσικούς για έγκριση στην μορφή του και περαιτέρω έλεγχο.

1.3 Οργάνωση κειμένου

Το Κεφάλαιο 2 της παρούσας Διπλωματικής Εργασίας αναφέρεται σε σχετικές με την δική μας εργασίες και προγράμματα. Το Κεφάλαιο 3 παρέχει τις θεωρητικές γνώσεις πάνω σε Φυσική, που πρέπει να έχει κάποιος για να καταλάβει πλήρως τη σημασία των προγραμμάτων που αναπτύξαμε. Το Κεφάλαιο 4 αναλύει τις απαιτήσεις του συστήματος που υλοποιήσαμε. Στο Κεφάλαιο 5 παρουσιάζουμε τον σχεδιασμό του συστήματος και στο Κεφάλαιο 6 αναλύουμε μέρος της υλοποίησής του. Το Κεφάλαιο 7 αναφέρει ένα σενάριο ελέγχου που εφαρμόσαμε προκειμένου να διαπιστώσουμε την συμφωνία του αποτελέσματος με την επιθυμητή συμπεριφορά. Το Κεφάλαιο 8 συνοψίζει την εργασία μας και το Κεφάλαιο 9 παρουσιάζει τις πηγές από όπου αντλήσαμε χρήσιμες πληροφορίες για την εκπόνηση της διπλωματικής μας εργασίας .

2

Σχετικές εργασίες

Στο παρόν κεφάλαιο, θα αναφέρουμε πολύ συνοπτικά δύο άλλα control systems, όπως αυτό στο οποίο εργαζόμαστε, δηλαδή το CESAR του ερευνητικού οργανισμού CERN. Τα δύο αυτά συστήματα, σκοπίμως επιλέγουμε να χρησιμοποιούνται από άλλα ερευνητικά κέντρα, ώστε να δώσουμε μια πιο σφαιρική εικόνα για το Λογισμικό που έχει αναπτυχθεί για τον έλεγχο των επιταχυντών στο πεδίο της Φυσικής Υψηλών Ενεργειών τα τελευταία χρόνια. Επίσης, είναι πιο μεγάλης κλίμακας και πιο πολύπλοκα από το δικό μας και επιτρέπουν και περισσοτέρων ειδών υπολογιστικές εφαρμογές.

2.1 EPICS

Το EPICS (Experimental Physics and Industrial Control System) είναι ένα έτοιμο εργαλείο - Λογισμικό το οποίο είναι γραμμένο σε C και C++ και επιτρέπει την διαχείριση και χρησιμοποίηση οποιουδήποτε Control System. Χρησιμοποιείται από διάφορους οργανισμούς, μεταξύ των οποίων και το ερευνητικό κέντρο Φυσικής Στοιχειωδών Σωματιδίων DESY στην Γερμανία.

2.2 ACNET

Πρόκειται για ενοποιημένο Control System για ολόκληρο το σύμπλεγμα επιταχυντών του ερευνητικού οργανισμού Fermilab, ο οποίος βρίσκεται στις Ηνωμένες Πολιτείες και μαζί με

το CERN αποτελούν τα δύο μεγαλύτερα ερευνητικά κέντρα στον τομέα της πειραματικής Φυσικής τα τελευταία 50 χρόνια. Πάνω στο σύστημα αυτό, έχουν αναπτυχθεί από τους Μηχανικούς Λογισμικού και τους Φυσικούς του Fermilab πληθώρα εφαρμογών που κάνουν monitoring αλλά και έλεγχο των διαφόρων επιταχυντών του οργανισμού (όπως ακριβώς κάνουμε και στο δικό μας σύστημα). Οι εφαρμογές αυτές είναι στην πλειονότητά τους γραμμένες σε γλώσσα Java, όπως και στην δική μας περίπτωση, αλλά και σε άλλες γλώσσες.

Συγκρίνοντας τα δύο παραπάνω συστήματα με το CESAR, μπορούμε να εξάγουμε τις παρακάτω διαφορές:

- ACNET
  - 3 επίπεδα
  - Κεντρικές Υπηρεσίες
  - Client/Server
  - Ιδιοκτησία ενός οργανισμού
    - Περισσότερος έλεγχος
    - Λιγότερη βοήθεια
  - Πληθώρα ειδών εφαρμογών
- EPICS
  - 2 επίπεδα
  - Κατανεμημένες υπηρεσίες
  - Client/Server
  - Συνεργασία οργανισμών
    - Περισσότερη βοήθεια
    - Λιγότερος έλεγχος
  - Μια GUI εφαρμογή μπορεί να χρησιμοποιηθεί για διάφορες απεικονίσεις
- CESAR
  - 3 επίπεδα
  - Κεντρικές υπηρεσίες
  - Client/Server
  - Ιδιοκτησία ενός οργανισμού + μικρή ομάδα προγραμματιστών
    - Περισσότερη βοήθεια
    - Λιγότερος έλεγχος
  - Μια GUI εφαρμογή μπορεί να χρησιμοποιηθεί για διάφορες απεικονίσεις

3

Θεωρητικό υπόβαθρο

Η Σωματιδιακή Φυσική έχει γίνει τελευταία το επίκεντρο της θεωρητικής έρευνας της Φυσικής Επιστήμης. Έτσι, μέσα από θεωρητικές μελέτες αλλά και πειραματικές παρατηρήσεις έχουν εξαχθεί διάφορα μοντέλα που σχετίζονται με την ταυτότητα και αλληλεπίδραση των διάφορων σωματιδίων καθώς και με την δομή, την ιστορία και την συμπεριφορά του σύμπαντος.

Με τον όρο στοιχειώδη σωματίδια, εννοούμε σωματίδια που δεν έχουν εσωτερική δομή, δεν αποτελούνται δηλαδή από άλλα σωματίδια. Τέτοια σωματίδια είναι τα φερμιόνια (κουάρκ και λεπτόνια) και τα μποζόνια (φωτόνια, W και Z μποζόνια και γλουόνια). Για παράδειγμα, δεν θεωρούνται στοιχειώδη σωματίδια ούτε τα άτομα, αφού αποτελούνται από τον πυρήνα (νετρόνια και πρωτόνια) και ηλεκτρόνια, ούτε τα πρωτόνια, αφού αποτελούνται από κουάρκ, ούτε πληθώρα άλλων γνωστών μας σωματιδίων. Μετά από πολλά χρόνια ερευνών, οι επιστήμονες έχουν καταλήξει στο Καθιερωμένο Μοντέλο, το οποίο απαριθμεί τα γνωστά ως σήμερα, στοιχειώδη σωματίδια και τις δυνάμεις που δρουν μεταξύ τους.

Τα πειράματα στον τομέα αυτό, για τα οποία αναφέραμε στην παράγραφο 1.1 είναι αναγκαία για την επαλήθευση ή κατάρρευση των διάφορων θεωρητικών μοντέλων που προτείνονται από τους θεωρητικούς Φυσικούς καθώς και την ευκαιρία προώθησης της θεωρητικής έρευνας μέσω παρατηρήσεων που απορρέουν από τα πειράματα.

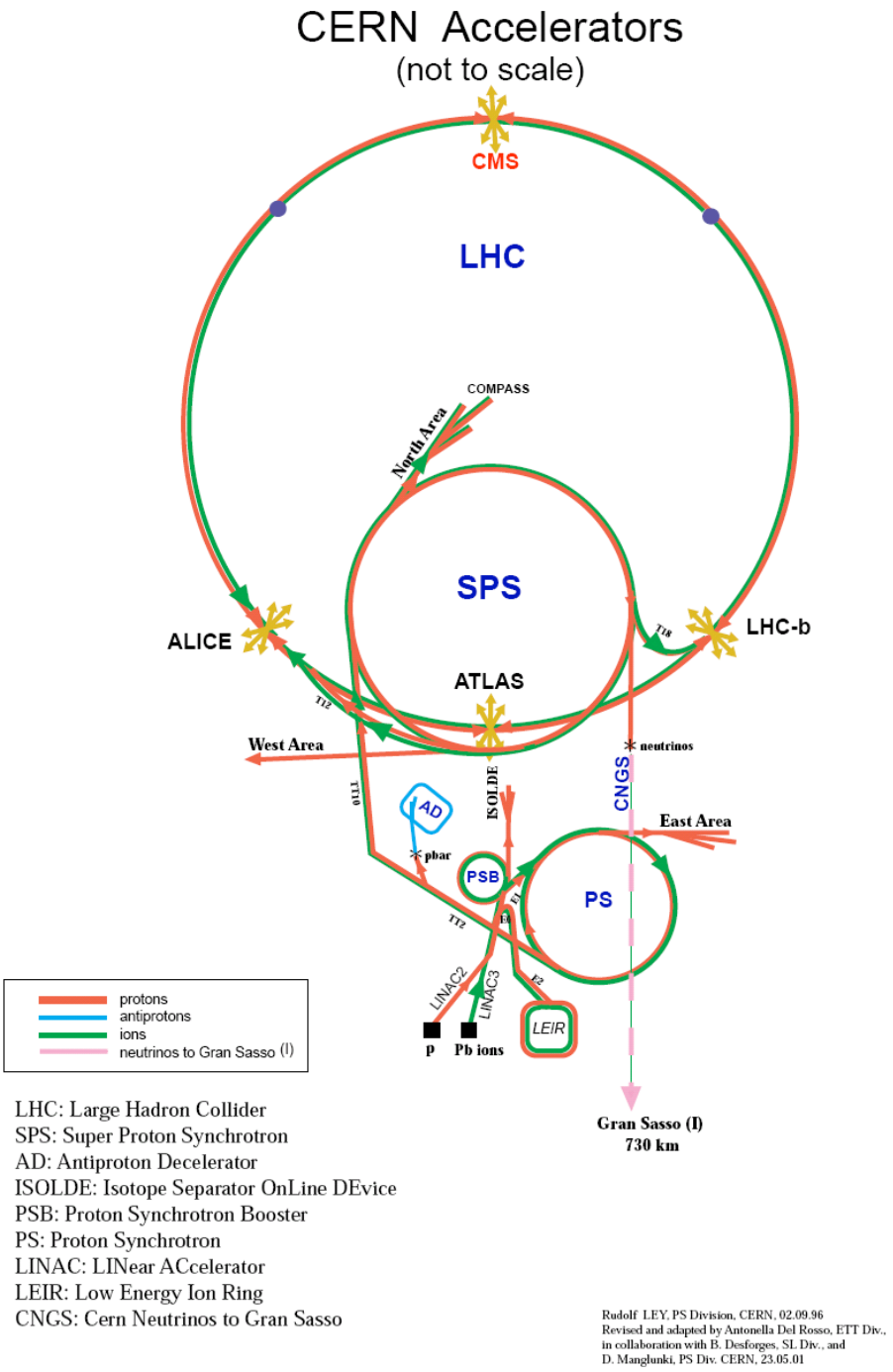
3.1 CERN

Το CERN (Ευρωπαϊκός Οργανισμός Πυρηνικής Έρευνας- European Organization for Nuclear Research) είναι το μεγαλύτερο ερευνητικό κέντρο στον τομέα της Σωματιδιακής Φυσικής στον κόσμο και βρίσκεται στα βορειοδυτικά προάστια της Γενεύης, ακριβώς πάνω στα Γαλλο-Ελβετικά σύνορα. Ο οργανισμός αριθμεί 20 κράτη-μέλη (μεταξύ τους και η Ελλάδα και μάλιστα ως ένα από τα ιδρυτικά μέλη) και είναι ο χώρος εργασίας για περίπου 2.600 εργαζόμενους πλήρους απασχόλησης. Εκτός αυτού, κατά προσέγγιση άλλοι 7.900 επιστήμονες και μηχανικοί (εκπροσωπώντας 50 πανεπιστήμια και 80 εθνικότητες), που αποτελούν την σχεδόν την μισή κοινότητα της Φυσικής Υψηλών Ενεργειών, συμμετέχουν σε προγράμματα του CERN.

Όπως σε όλα τα ερευνητικά κέντρα του τομέα, έτσι και στο CERN, γίνονται πειράματα σαν και αυτά που ήδη έχουμε περιγράψει στο κεφάλαιο 1. Η διαφορά του CERN είναι ότι εδώ, τα σωματίδια συγκρούονται με πολύ μεγαλύτερες ταχύτητες (άρα και ενέργειες) από οπουδήποτε αλλού (περίπου 10πλάσιες από οποιαδήποτε άλλη μηχανή) με αποτέλεσμα να δημιουργούνται οι συνθήκες που επικράτησαν στο σύμπαν κλάσματα δευτερολέπτου μετά το Big Bang (Μεγάλη Έκρηξη). Έτσι, η ανάλυση των δεδομένων των πειραμάτων, αν και αρκετά χρονοβόρα λόγω του τεράστιου όγκου των δεδομένων που παράγονται, αναμένεται να αποκαλύψει αρκετά από τα μυστικά του σύμπαντος. Μια άλλη διαφορά του CERN είναι ότι εδώ γίνονται συγκρούσεις μεταξύ σωματιδίων που έχουν προκύψει από πρωτόνια.

Εικόνα 1: Λογότυπο του CERN

Το σύμπλεγμα των επιταχυντών του CERN φαίνεται στην εικόνα 2:



Εικόνα 2: Σύμπλεγμα των επιταχυντών του CERN

Όπως βλέπουμε στην εικόνα, υπάρχουν 3 κύριοι (κυκλικοί) επιταχυντές:

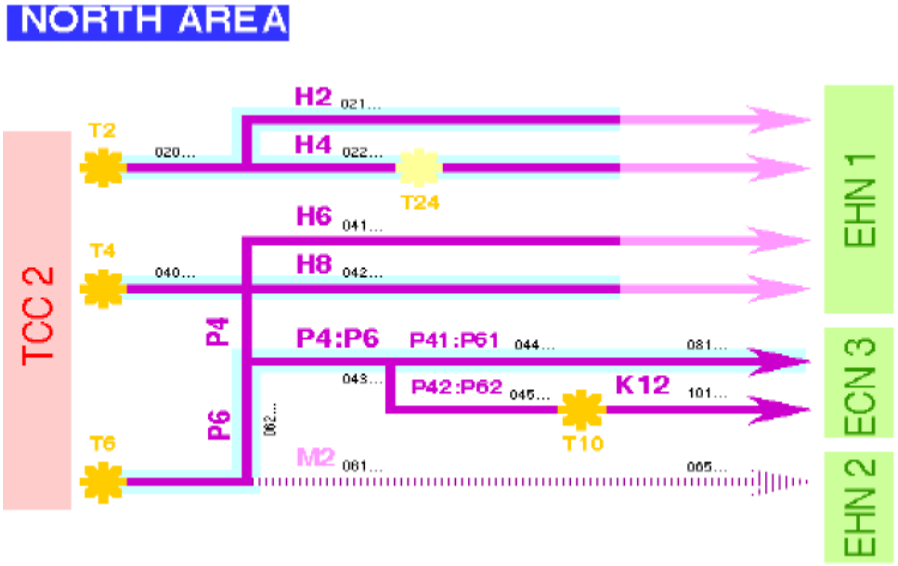
- PS: Είναι ο πιο μικρός από τους 3 (έχει περιφέρεια 628m). Σε αυτόν παράγονται τα πρωτόνια και επιταχύνονται ελαφρώς (έως τα 28 GeV). Στη συνέχεια περνάνε στον επιταχυντή SPS.

- SPS: Είναι ο αμέσως μεγαλύτερος (περιφέρεια 7 km). Εδώ τα διάφορα σωματίδια επιταχύνονται περαιτέρω ως τα 450 GeV και είτε καταλήγουν στο Βόρειο Τμήμα του επιταχυντή σε διάφορα πειράματα, όπως το Compass και το NA-48, είτε συνεχίζουν στον LHC.
- LHC: Είναι ο πιο μεγάλος (περιφέρεια 27 km). Εδώ βρίσκονται τα πιο διάσημα πειράματα του οργανισμού, τα οποία αρχίζουν την λειτουργία τους το φθινόπωρο του 2009. Αυτά τα πειράματα είναι το ATLAS, το CMS, το ALICE και το LHCb, τα οποία και φαίνονται στην εικόνα 2. Στον LHC, τα σωματίδια επιταχύνονται ως την τεράστια ενέργεια των 14 TeV και τελικά, δυο δέσμες σωματιδίων συγκρούονται μετωπικά σε κάποιο από τα 4 πειράματα που αναφέραμε. Εκεί, γίνεται η ανίχνευση των σωματιδίων που προκύπτουν με την βοήθεια τεράστιων ανιχνευτών. Κάθε πείραμα, έχει στόχο την εξερεύνηση διαφορετικών περιοχών έρευνας.

Εμείς εργαστήκαμε στο Βόρειο Τμήμα του επιταχυντή SPS, στο section AB-ATB-SBA, το οποίο ασχολείται με την συντήρηση και ανάπτυξη της δέσμης των σωματιδίων του επιταχυντή, ώστε να υποστηρίζει τις απαιτήσεις των πειραμάτων στο Βόρειο Τμήμα του SPS. Πιο συγκεκριμένα, εργαστήκαμε στο project CESAR, το οποίο έχει ως στόχο τον πλήρη έλεγχο της δέσμης στον SPS μέσω της ομώνυμης εφαρμογής που έχει αναπτυχθεί τα τελευταία χρόνια από Φυσικούς και Μηχανικούς Λογισμικού στο CERN. Περισσότερες πληροφορίες για το εν λόγω project και τις απαιτήσεις του συστήματος που εμείς υλοποιήσαμε δίνονται στο 4^{ο} κεφάλαιο.

3.2 Βόρειο Τμήμα του Επιταχυντή SPS

Το Βόρειο Τμήμα του SPS είναι ένα σύστημα από διάφορες δέσμες σωματιδίων και διάφορα πειράματα. Οι δέσμες σωματιδίων είναι αυτές που έχουν επιταχυνθεί σε όλο το μήκος του επιταχυντή SPS και τελικά έχουν καταλήξει στο Βόρειο Τμήμα του, όπου και θα οδηγηθούν στα πειράματα. Σχηματικά, το σύστημα αυτό φαίνεται στην εικόνα 3:

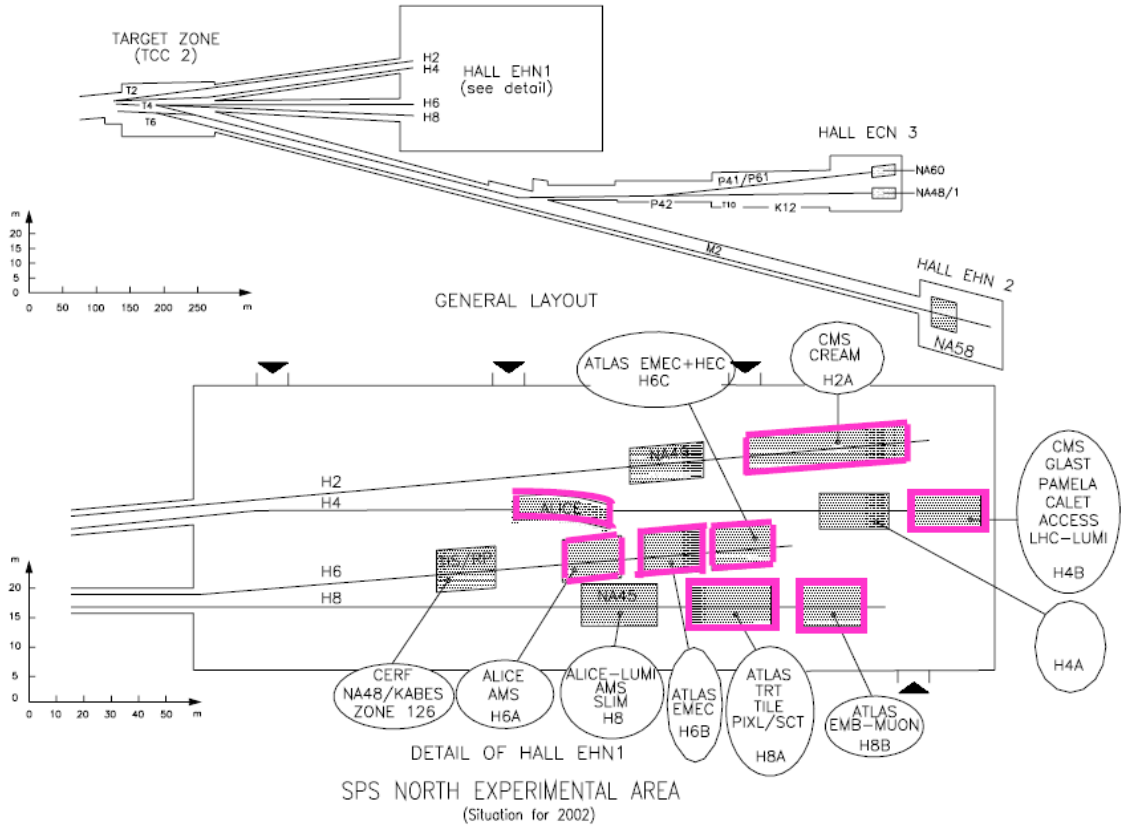


Εικόνα 3: Βόρειο Τμήμα του επιταχυντή SPS

Αναλύοντας το παραπάνω σχήμα, μπορούμε να πούμε τα εξής:

- Παρατηρούμε διάφορες δέσμες σωματιδίων (beam lines), οι οποίες αναπαρίστανται με ένα λατινικό γράμμα και έναν αριθμό (H2, H4, H6, H8, P42, K12 κλπ) και φαίνονται με το μοβ χρώμα στο σχήμα. Τις δέσμες σωματιδίων μπορούμε να τις φανταστούμε ως διαφορετικούς «δρόμους» στους οποίους «τρέχουν» τα σωματίδια υπό μορφή ευθύγραμμης δέσμης. Οι δέσμες μπορεί σε άλλα σημεία να περικλείονται από λεπτούς σωλήνες και σε άλλα να βρίσκονται απλά στον αέρα. Συνολικά, οι δέσμες αυτές έχουν μήκος περίπου 6,3 km. Τέλος, είναι προφανές ότι δεν χρειάζεται (και μερικές φορές είναι αδύνατο) να λειτουργούν όλες οι δέσμες ταυτόχρονα.
- Παρατηρούμε ακόμα τους διάφορους στόχους (targets), οι οποίοι αναλύουν μία δέσμη, όταν προσπίπτει κάθετα σε αυτούς, σε επιμέρους δέσμες σωματιδίων, με διαφορετικές ορμές. Οι στόχοι είναι με το κίτρινο χρώμα στην εικόνα (T2, T4, T6 και T10).
- Το ροζ ορθογώνιο (TCC2) είναι ένα κτήριο, μέσα στο οποίο βρίσκονται οι στόχοι T2, T4 και T6.
- Τα 3 πράσινα κτίρια (EHN1, ECN2 και EHN3) είναι 3 κτίρια στα οποία γίνονται τα διάφορα πειράματα. Τα πειράματα αυτά έχουν διάφορα ονόματα, που προδίδουν και σε ποια δέσμη υπάγονται (π.χ. H4A, H8C κλπ). Κάθε πείραμα ανήκει σε ακριβώς μία δέσμη, ενώ κάθε δέσμη μπορεί να έχει ένα ή περισσότερα πειράματα. Στην πραγματικότητα, τα πειράματα αυτά δεν είναι τίποτα άλλο από ζώνες μέσα στα 3 παραπάνω κτήρια.

- Αν και στην εικόνα 3 δεν φαίνεται, κατά μήκος των διαφόρων δεσμών υπάρχει πλήθος εξοπλισμού, ο οποίος ελέγχει τις δέσμες. Για αυτόν ακριβώς τον έλεγχο είναι υπεύθυνο το πρόγραμμα στο οποίο συμμετείχαμε (CESAR).



Εικόνα 4: Ο πειραματικός χώρος EHN1

Γίνεται φανερό πλέον, ότι δική μας ευθύνη είναι ο έλεγχος της δέσμης των σωματιδίων στην περιοχή αμέσως πριν το χώρο των πειραμάτων μέσω του εξοπλισμού που βρίσκεται εκεί. Αυτός ο εξοπλισμός δεν είναι παρά κάποια εξειδικευμένα μηχανήματα, όπως μαγνήτες, κατευθυντήρες, ανιχνευτές και πολλά ακόμα. Ο έλεγχος της δέσμης είναι απαραίτητος για τους πειραματικούς Φυσικούς, δηλαδή τους Φυσικούς που δουλεύουν στα πειράματα του Βόρειου Τμήματος του SPS. Αυτοί είναι και οι κύριοι χρήστες του λογισμικού που αναπτύξαμε. Εκτός, όμως από αυτούς, το CESAR το χρησιμοποιούν και οι Φυσικοί που συντηρούν την δέσμη, δηλαδή οι Φυσικοί του section στο οποίο εργαστήκαμε. Μάλιστα, οι συντηρητές της δέσμης πρέπει να έχουν περισσότερα προνόμια (privileges) στις δυνατότητες του λογισμικού από τους απλούς χρήστες.

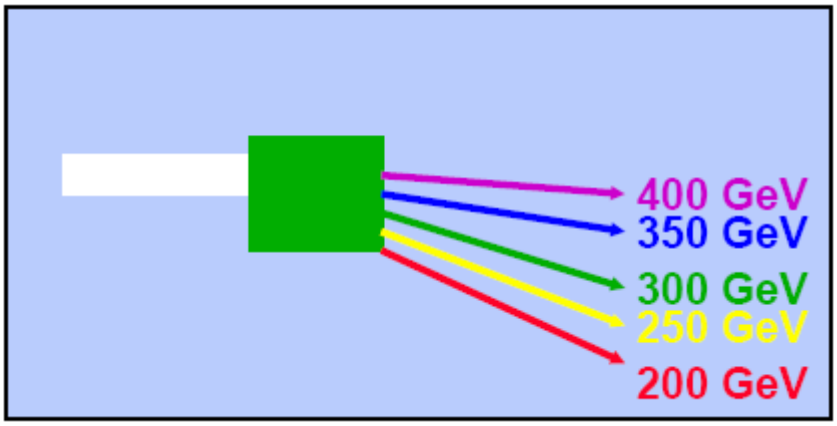
Πριν προχωρήσουμε στο ίδιο το πρόγραμμα CESAR, είναι απαραίτητο να πούμε δυο λόγια για τον εξοπλισμό των διαφόρων δεσμών, τον οποίο ελέγχει το πρόγραμμα.

Ο εξοπλισμός είναι απαραίτητος ώστε να καθοδηγήσουμε την δέσμη προς τις επιθυμητές κατευθύνσεις, να αλλάξουμε την ορμή της (άρα και την ενέργεια και ταυτόχρονα και τα σωματίδια που την αποτελούν), να αναγνωρίσουμε τα διάφορα σωματίδια που προκύπτουν, να εξασφαλίσουμε την ασφάλεια των ανθρώπων που δουλεύουν στα πειράματα κλπ.

Κάποια παραδείγματα μηχανημάτων αναφέρονται παρακάτω (με τα αγγλικά τους ονόματα καθώς αυτά θα συναντήσουμε και στον κώδικα της εφαρμογής μας) ταξινομώντας τα κατά κατηγορία:

- **Καθοδήγηση δέσμης:**

  1) Magnets: Είναι τα πιο σημαντικά μηχανήματα. Χωρίζονται σε 2 κατηγορίες:
  -Δίπολα (Dipoles): Για παράδειγμα Bends και Trims. Διαθλούν την δέσμη σε διάφορες γωνίες, ανάλογα με το ρεύμα που τα διαρρέουν. Έτσι, η μία δέσμη αναλύεται σε πολλές δέσμες διαφόρων ορμών και επομένως διαφόρων τύπων σωματιδίων (λειτουργία ανάλογη με ένα πρίσμα).



  Εικόνα 5: Λειτουργία δίπολου

  -Τετράπολα (Quadrupoles): Προκαλεί ταυτόχρονα εστίαση και απόκλιση στις δύο διαστάσεις (λειτουργία οπτικού φακού).

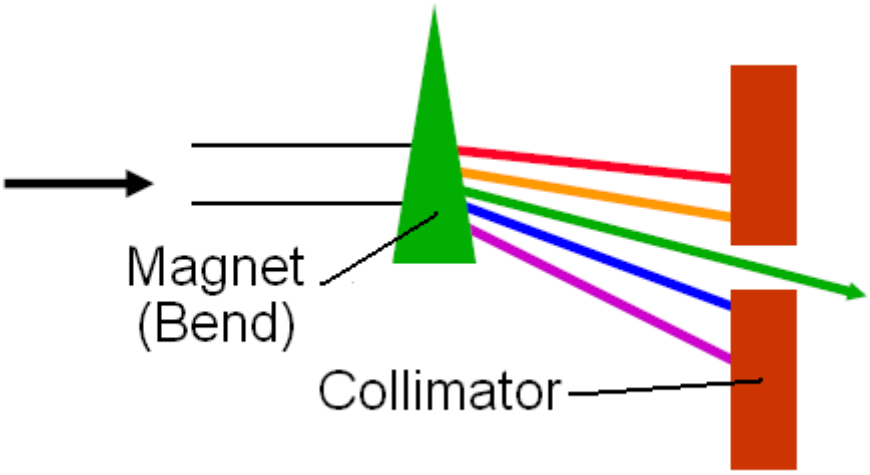
  κ.α.

- **Μέτρηση ορμής:**

  Calorimeters κ.α.

- **«Καθάρισμα» δέσμης:**

  1) Collimators: Απομονώνει ορισμένα από όλα τα σωματίδια που βρίσκονται στη δέσμη:



Εικόνα 6: Λειτουργία των Magnets και Collimators

2) Scrapers

κ.α.

- **Ασφάλεια:**

  1) Dumps: Είναι μεγάλοι όγκοι από βαρέα μέταλλα που ασφαλίζουν την περιοχή γύρω από τα πειράματα από την ακτινοβολία της δέσμης.

  2) Taxes: Είναι κινούμενα εμπόδια που μπαίνουν μπροστά από την δέσμη, όποτε χρειαστεί να διακοπεί.

  3) Access Doors: Είναι πόρτες που οδηγούν στα πειράματα και οι οποίες πρέπει να ασφαλίζονται όταν υπάρχει δέσμη σε κάποιο πείραμα, ώστε η πρόσβαση ανθρώπων σε αυτό να είναι αδύνατη.

  κ.α.

4

Ανάλυση Απαιτήσεων Συστήματος

Το πρόγραμμα στο οποίο εργαστήκαμε, δηλαδή το CESAR, έχει ως αντικείμενο τον έλεγχο της δέσμης των σωματιδίων στο Βόρειο Τμήμα του επιταχυντή SPS στο CERN. Πιο συγκεκριμένα, οι χρήστες του, οι οποίοι είναι πειραματικοί φυσικοί του ίδιου του οργανισμού, μπορούν μέσα από αυτό το πρόγραμμα, να μεταβάλλουν την τιμή σε διάφορες παραμέτρους του εξοπλισμού μίας δέσμης, να αποθηκεύσουν τις προτιμήσεις τους για κάποια δέσμη και πείραμα σε αρχεία (αρχεία δέσμης), να μπορούν να τροποποιήσουν τα αρχεία αυτά αλλά και να επαναφέρουν παλαιές εκδοχές τους, να φορτώσουν όποιο αρχείο επιθυμούν στον πραγματικό εξοπλισμό της δέσμης (δηλαδή να φορτώσουν τις τιμές του αρχείου για τα διάφορα μηχανήματα στον πραγματικό εξοπλισμό), να μπορούν να δουν σε ποια κατάσταση βρίσκεται ο εξοπλισμός, στοιχεία της ιστορίας (πρόσφατες αλλαγές) του εξοπλισμού, κ.α. Να σημειώσουμε ότι ένα αρχείο δέσμης ανήκει σε ένα συγκεκριμένο πείραμα (μέλη του οποίου το δημιούργησαν), ενώ κάποιο μηχάνημα ανήκει σε συγκεκριμένη δέσμη.

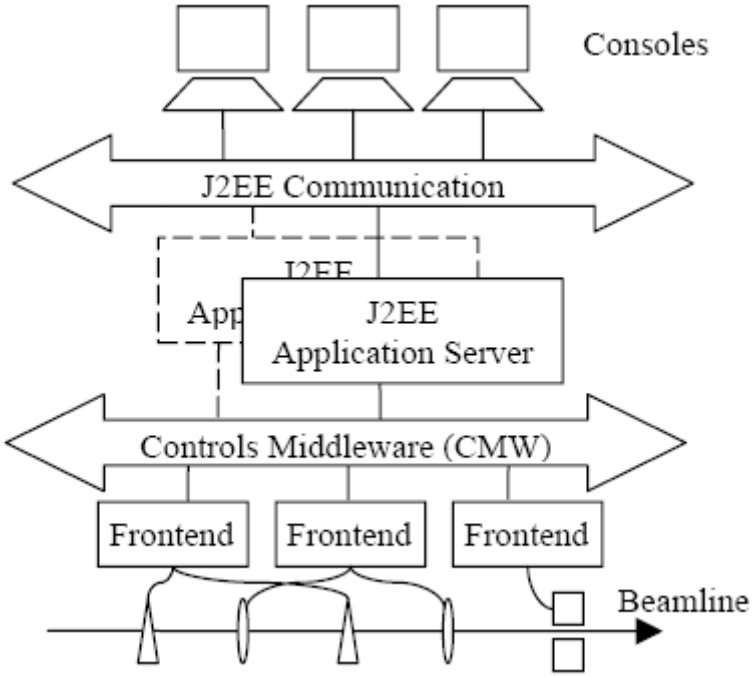
Όπως έχουμε ήδη αναφέρει στην περίληψη του παρόντος τόμου, η εργασία μας επικεντρώθηκε σε τρεις τομείς: την γενική εφαρμογή CESAR για όλο τον έλεγχο που επιζητούμε στην δέσμη, τα αρχεία της δέσμης και την κατάσταση του εξοπλισμού μιας δέσμης. Είναι, επίσης, χρήσιμο να τονίσουμε από τώρα, ότι στον κόσμο του CESAR, εμφανίζονται 3 ειδών τιμές εξοπλισμού: (α) αναφορικές τιμές αρχείων (file references), δηλαδή τιμές για τα μηχανήματα που είναι αποθηκευμένες στα αρχεία, (β) αναφορικές τιμές δέσμης (beam references), δηλαδή τιμές που θα πρέπει να έχει ο εξοπλισμός, ανάλογα με το αρχείο που έχει φορτωθεί στη δέσμη μία δεδομένη στιγμή (με άλλα λόγια τα file references

του αρχείου που είναι φορτωμένο κάθε φορά στη δέσμη) και (γ) αναγνωσμένες τιμές (readings), δηλαδή οι τιμές που έχει ο εξοπλισμός στην πραγματικότητα κάθε στιγμή.

Για την υλοποίηση λειτουργιών, όπως αυτές που ήδη αναφέραμε, κληθήκαμε να αναπτύξουμε μια σειρά επιμέρους προγραμμάτων και τελικά να τα συνδυάσουμε, χρησιμοποιώντας μοντέρνα εργαλεία λογισμικού με απώτερο σκοπό την αποτελεσματικότητα στον έλεγχο του εξοπλισμού αλλά και την ελκυστική εμφάνιση και την ευκολία στη χρήση ενός ενιαίου προγράμματος, ονόματι CESAR, το οποίο θα είναι υπεύθυνο για όλες τις πτυχές ελέγχου των δεσμών στο Βόρειο Τμήμα του SPS.

4.1 Αρχιτεκτονική

Το project CESAR, το οποίο αποτελεί πτυχή της γενικότερης δραστηριότητας του CERN με σκοπό την ανανέωση και αναβάθμιση του λογισμικού του οργανισμού, είναι ένα καλά δομημένο σύστημα ελέγχου και έχει βασιστεί στην αρχιτεκτονική 3 επιπέδων (3-Tier Architecture), όπως φαίνεται και στην εικόνα 7.



Εικόνα 7: Αρχιτεκτονική του συστήματος ελέγχου CESAR

Στο κατώτερο σημείο έχουμε την δέσμη των σωματιδίων (beam line), όπου βρίσκεται και ολόκληρος ο εξοπλισμός, όπως μαγνήτες (magnets), κατευθυντήρες (collimators), ανιχνευτές σωματιδίων (detectors) κ.α. Ο εξοπλισμός αυτός ελέγχεται από τους μετωπικούς (front-end) υπολογιστές του σχήματος, οι οποίοι με τη σειρά τους συνδέονται με το εξατομικευμένο λογισμικό ελέγχου (Controls Middleware-CMW). Ακριβώς ένα επίπεδο πιο πάνω, έχουμε την καρδιά του συστήματος, δύο J2EE (Java 2 Enterprise Edition) servers, που τρέχουν σε 2

UNIX servers. Οι 2 servers προσδίδουν αξιοπιστία στο σύστημα: αν ένας από τους δύο σταματήσει να λειτουργεί σωστά, αναλαμβάνει ο άλλος. Το βέλος "J2EE Communication" αναπαριστά τα εργαλεία επικοινωνίας των J2EE servers. Στο ανώτατο επίπεδο βρίσκονται οι προσωπικοί υπολογιστές (consoles), που είναι τοποθετημένοι στο κέντρο ελέγχου του CERN και στους χώρους των πειραμάτων. Τα παραπάνω 3 επίπεδα από κάτω προς τα πάνω, είναι τα γνωστά "Resource Tier", "Middle Tier" και "Presentation Tier" της αρχιτεκτονικής των 3 επιπέδων.

Η παρούσα Διπλωματική Εργασία επικεντρώνεται στο ανώτερο επίπεδο και σχετίζεται με εφαρμογές γραμμένες σε γλώσσα Java και οι οποίες είναι σχεδιασμένες έτσι ώστε να εξυπηρετούν τους πειραματικούς φυσικούς.

Για αυτό το ανώτερο επίπεδο, η εργασία μας χωρίστηκε σε 3 επιμέρους υποσυστήματα:

(α): Γενικό Framework μιας εφαρμογής για τους χρήστες (Framework for client-side Application). Αποτελεί τον κορμό της εφαρμογής, η οποία θα ονομάζεται και αυτή "CESAR".

(β): Διαχείριση Αρχείων Δέσμης (Beam Files Management)

(γ): Κατάσταση Εξοπλισμού (Equipment Status)

Τα (β) και (γ) θα αναπτυχθούν ανεξάρτητα από το (α), αλλά ταυτόχρονα θα πρέπει να μπορούν να ενσωματωθούν σε αυτό. Συνεπώς, η εφαρμογή CESAR θα περιέχει και τα Αρχεία Δέσμης και την Κατάσταση Εξοπλισμού, όμως και τα δύο αυτά θα πρέπει να μπορούν να εκτελεστούν και ανεξάρτητα από το CESAR. Η ανεξάρτητη εκτέλεση των (β), (γ) από το CESAR δεν απευθύνεται στους απλούς χρήστες οι οποίοι χρησιμοποιούν μόνο την εφαρμογή CESAR ως ενιαίο σύνολο αλλά στους προγραμματιστές και τους ειδικούς της δέσμης που θέλουν να εξετάζουν και να ελέγχουν τροποποιήσεις και νέες εκδόσεις των (β) και (γ), και αν τελικά το επιθυμούν, να τις ενσωματώνουν στο CESAR.

Τέλος, να διασαφηνίσουμε ότι η εφαρμογή CESAR, υπήρχε τα παλιότερα χρόνια. Ωστόσο, έπρεπε να επανασχεδιαστεί για διάφορους λόγους. Ένας από αυτούς είναι ότι είχε βασιστεί εξ' ολοκλήρου σε μια τεχνολογία-πλατφόρμα της αγοράς, που ονομάζεται Netbeans. Αυτό δεν ήταν επιτρεπτό, καθώς τα τελευταία χρόνια η πλατφόρμα αυτή έπαψε να υποστηρίζει συγκεκριμένα χαρακτηριστικά τα οποία ήταν απαραίτητα στο CESAR, με αποτέλεσμα αυτό να μένει προσκολλημένο σε παλαιότατες εκδόσεις της πλατφόρμας κάνοντας το ανήμπορο να ανανεωθεί και να επεκταθεί.

4.2 Περιγραφή Λειτουργιών

Όπως ήδη έχουμε αναφέρει, η εργασία μας χωρίστηκε σε τρεις τομείς. Οι λειτουργίες προς υλοποίηση για καθέναν από αυτούς, αναλύεται στη συνέχεια

4.2.1 Εφαρμογή

Το γενικό Framework της εφαρμογής πρέπει να πληροί κάποιες προϋποθέσεις ώστε να εξυπηρετεί τον σκοπό μας, που δεν είναι άλλος από τον έλεγχο των δεσμών των σωματιδίων στον επιταχυντή SPS. Γι' αυτό αλλά και για άλλους λόγους, το framework της εφαρμογής μας πρέπει να έχει τα παρακάτω χαρακτηριστικά:

- Οργάνωση σε Workspaces:

Αφού ο χώρος των πειραμάτων χωρίζεται σε δέσμες, όπως είδαμε στο κεφάλαιο 3, η εφαρμογή μας θα πρέπει να είναι οργανωμένη σε χώρους εργασίας (workspaces), δηλαδή χώρους στους οποίους εργαζόμαστε ανεξάρτητα και καθένας υποδηλώνει μία δέσμη. Κάθε φορά, ένα workspace πρέπει να βρίσκεται στο προσκήνιο (ενεργή δέσμη). Την εναλλαγή των ενεργών δεσμών θα πρέπει να την κάνουμε με χρήση καρτελών (tabs). Τέλος, πρέπει να μπορούμε να ανοίξουμε και να κλείσουμε συγκεκριμένα workspaces (δέσμες).

- Επιμέρους panels:

Στην εκάστοτε ενεργή δέσμη, θα πρέπει ο χρήστης να ανοίγει εσωτερικά στο workspace παράθυρα (internal frames) που δεν είναι άλλο από panels με συγκεκριμένες πληροφορίες. Τέτοια μπορεί να είναι panels που, για παράδειγμα, παρουσιάζουν την κατάσταση του εξοπλισμού ή διαχειρίζονται τα αρχεία της δέσμης. Η δυνατότητα αυτή θα πρέπει να είναι διαθέσιμη μέσα από κάποιο menu ή/ και toolbar σε κάποιο σημείο της εφαρμογής (επιλέγοντας την κατάλληλη ενέργεια).

- Ασφάλεια:

Πολύ σημαντική στην εφαρμογή μας είναι η ασφάλεια, καθώς δεν πρέπει οποιοσδήποτε χρήστης να έχει πρόσβαση στον εξοπλισμό και τα αρχεία ενώ διαφορετικοί χρήστες είναι καλό να έχουν και διαφορετικά δικαιώματα (privileges). Έτσι, στο CESAR πρέπει να υπάρχουν δυνατότητες σύνδεσης και αποσύνδεσης (login, logout) σε /από το σύστημα με χρήση έγκυρων λογαριασμών χρηστών.

Το CESAR πρέπει να υποστηρίζει 4 ειδών χρήστες:

Α) Guest: Επισκέπτης. Χρήστης που δεν έχει κάνει επιτυχημένο login στο σύστημα.

Β) Beam Parasitic User: Παρασιτικός χρήστης δέσμης (κάποιου πειράματος).

Γ) Beam Main User: Κύριος χρήστης μιας δέσμης (κάποιου πειράματος).

Δ) Super User: Συντηρητής της δέσμης.

Γενικά, το πλήθος των δικαιωμάτων έχει την εξής ιεραρχία (από αριστερά προς τα δεξιά αυξάνονται τα δικαιώματα):

Guest < Beam Parasitic User < Beam Main User < Super User

Για παράδειγμα, κάθε χρήστης μπορεί να έχει πρόσβαση (ανάγνωσης ή αλλαγής τιμών) μόνο σε συγκεκριμένες δέσμες:

Α) Guest: Σε καμία δέσμη.

Β) Beam Parasitic User: Στην δέσμη του πειράματός του.

Γ) Beam Main User: Στην δέσμη του πειράματός του.

Δ) Super User: Σε όλες τις δέσμες.

Αλλά και μέσα σε μία δέσμη, διαφορετικοί χρήστες δεν έχουν τα ίδια δικαιώματα.

Κατά την εκκίνηση του προγράμματος πρέπει να γίνεται αυτόματο login (autologin), σύμφωνα με τον υπολογιστή από τον οποίο εκκινείται το πρόγραμμα. Αν δεν αναγνωριστεί ως κάποιου άλλου τύπου, ο χρήστης συνδέεται ως Guest. Στη συνέχεια, ο χρήστης θα πρέπει να μπορεί να κάνει όσα login επιθυμεί (π.χ. έχει διάφορους λογαριασμούς ή κάποιος άλλος χρήστης χρησιμοποιεί προσωρινά τον υπολογιστή). Όταν γίνεται logout ο προηγούμενα συνδεδεμένος χρήστης γίνεται ενεργός (εκτός αν δεν υπάρχει προηγούμενος).

Όταν αλλάζει ο ενεργός χρήστης, τότε αυτόματα και ανάλογα με τα δικαιώματα του νέου χρήστη πρέπει να γίνονται τα εξής:

- Επιλογές σε menu ή toolbar μπορεί να καταστούν ενεργές ή ανενεργές.

- Μερικά workspaces μπορεί να κλείσουν, άλλα να ανοίξουν και άλλα να μείνουν στη θέση τους.

- Τα εσωτερικά παράθυρα των workspaces που μένουν στην θέση τους, πρέπει να ενημερώνονται για την αλλαγή στον ενεργό χρήστη και να προσαρμόζονται σύμφωνα με τα δικαιώματα του νέου χρήστη.

Οι επιλογές (ενέργειες) του framework, μαζί με το ποιος χρήστης έχει το δικαίωμα να τις χρησιμοποιήσει και από πού, φαίνονται παρακάτω:

Γενικές ενέργειες:

ΕΝΕΡΓΕΙΑ	ΕΞΟΥΣΙΟΔΟΤΗΜΕΝΟΙ ΧΡΗΣΤΕΣ	ΑΠΟ ΠΟΥ	ΕΠΕΞΗΓΗΣΗ
Add workspaces	ΟΛΟΙ (ο καθένας τις εξουσιοδοτημένες δέσμες του)	toolbar/menu “Workspace” ή κατά τη διάρκεια του autologin	Άνοιγμα ενός ή περισσοτέρων workspaces

Login/Logout	ΟΛΟΙ	menu “EA/Login” ή κατά τη διάρκεια του autologin	Σύνδεση / αποσύνδεση στο σύστημα με χρήση κάποιου λογαριασμού
Remove workspaces	ΟΛΟΙ	Δεξί κλικ στην καρτέλα ενός workspace -> Remove ή Remove All	Κλείσιμο ενός ή όλων των ανοιχτών workspaces

Αρχεία δέσμης:

ΕΝΕΡΓΕΙΑ	ΕΞΟΥΣΙΟΔΟΤΗΜΕΝΟΙ ΧΡΗΣΤΕΣ	ΑΠΟ ΠΟΥ	ΕΠΕΞΗΓΗΣΗ
Browse all Beam Files	Beam Main User, Beam Parasitic User, Super User	toolbar/menu “Files”	Εμφάνιση του panel με όλα τα αρχεία του πειράματος του ενεργού χρήστη
Browse Beam Files by Experiment	Super User	menu “Files”	Εμφάνιση του panel με όλα τα αρχεία πειράματος που επιλέγει ο χρήστης
Browse Beam Files filtered by Wobbling	Super User	menu “Files”	Εμφάνιση του panel με τα αρχεία που είναι σύμφωνα με τωρινή διαμόρφωση δέσμης
Save Actual Refs to last loaded file	Beam Main User, Beam Parasitic User, Super User	toolbar/menu “Files”	Ενέργεια “Save Actual BeamRefs” των panels των Αρχείων της δέσμης για το τελευταία φορτωμένο αρχείο στη δέσμη (περισσότερες πληροφορίες σε 4.2.2)

Κατάσταση εξοπλισμού:

ΕΝΕΡΓΕΙΑ	ΕΞΟΥΣΙΟΔΟΤΗΜΕΝΟΙ ΧΡΗΣΤΕΣ	ΑΠΟ ΠΟΥ	ΕΠΕΞΗΓΗΣΗ
Browse a Status Panel	Beam Main User, Beam Parasitic User, Super User	toolbar/menu “Status” και “Detectors”	Εμφάνιση panel με την κατάσταση συγκεκριμένου είδους εξοπλισμού

- **Ανεξαρτησία και Αφαίρεση:**

Το framework της εφαρμογής πρέπει να είναι ανεξάρτητο από τεχνολογίες που εύκολα αλλάζουν ή καταργούνται. Πρέπει δηλαδή να εξαρτώνται από standard γλώσσα προγραμματισμού και όχι κάποια άλλη εξεζητημένη έτοιμη πλατφόρμα που έχει αναπτυχθεί πάνω σε αυτή τη γλώσσα. Η απαίτηση αυτή πηγάζει από το γεγονός ότι τα προηγούμενα χρόνια, το πρόγραμμα CESAR είχε βασιστεί πάνω στην τεχνολογία NetBeans, η οποία με τον καιρό έπαψε να υποστηρίζει ορισμένα απαραίτητα για το CESAR χαρακτηριστικά (όπως τα workspaces), όπως αναφέραμε και στην προηγούμενη παράγραφο.

Επίσης, το framework πρέπει να είναι αφηρημένο (abstract), ώστε να μπορεί να χρησιμοποιηθεί ως βάση και για άλλες εφαρμογές εκτός του CESAR, που παρουσιάζουν παρόμοια χαρακτηριστικά (π.χ. workspaces και ασφάλεια με login/logout). Η διαμόρφωση δηλαδή του framework πρέπει να γίνεται εξωτερικά από αυτό, ώστε να αποκτά επιθυμητά για το CESAR χαρακτηριστικά.

- **Σταθερό, φιλικό, αποδοτικό:**

Πρέπει η εφαρμογή να είναι σταθερή, δηλαδή να μην «κολλάει» και να αποτυγχάνει συχνά. Επίσης, πρέπει να είναι φιλική και κατανοητή προς το χρήστη όντας ευπαρουσίαστη, πρακτική (ακόμα και στα μικρά πράγματα που κάνει ο χρήστης) και καλά οργανωμένη, ώστε να προσφέρει ξεκάθαρα τις διαθέσιμες επιλογές. Ακόμα, το CESAR πρέπει να είναι όσο το δυνατόν πιο ελαφρύ (lightweight) σε χρήση της μνήμης και του επεξεργαστή του υπολογιστή στον οποίο θα τρέχει, καθώς η εφαρμογή πρόκειται να μένει ενεργή σε υπολογιστές επί μέρες. Έτσι, η εφαρμογή μας θα είναι, ταυτόχρονα και αρκετά γρήγορη εξυπηρετώντας καλύτερα τους χρήστες.

- **Συντήρηση:**

Για λόγους συντήρησης, το framework είναι πολύ σημαντικό να είναι καλά και κατανοητά γραμμένο και τεκμηριωμένο, ώστε να μπορεί να τροποποιηθεί και να επεκταθεί εύκολα από τους προγραμματιστές του project CESAR τα επόμενα χρόνια. Επίσης, πρέπει να μπορούν εύκολα να προστίθενται ή να αφαιρούνται στοιχεία του CESAR σε /από αυτό καθώς και να αλλάζουν εύκολα στοιχεία που είναι πιο πιθανό να πρέπει να τροποποιηθούν στο μέλλον, λόγω απαιτήσεων των φυσικών (π.χ. κάποιο tooltip σε κάποια επιλογή).

- **Κυρίως Παράθυρο:**

Πρέπει το κυρίως παράθυρο της εφαρμογής να έχει μία κονσόλα (προαιρετικά εμφανιζόμενη) στην οποία να γράφουμε χρήσιμα μηνύματα προς τους χρήστες (logging output) που έχουν σχέση με τις ενέργειες ή τα σφάλματα που λαμβάνουν χώρα κατά την εκτέλεση του προγράμματός μας. Επίσης, στο ίδιο παράθυρο, είναι καλό να υπάρχει μια μπάρα προόδου για όλες τις ενέργειες που απαιτούν αρκετό χρόνο για να εκτελεστούν. Τέλος, στο κάτω μέρος του παραθύρου πρέπει να υπάρχει και μία μπάρα κατάστασης (status line), στην οποία γράφουμε με ειδική χρωματική σήμανση διάφορα μηνύματα προς το χρήστη που είναι σημαντικά και πρέπει οπωσδήποτε να δει (η μπάρα είναι πάντα ορατή).

4.2.2 Αρχεία Δέσμης

Ένα από τα σημαντικότερα χαρακτηριστικά του CESAR είναι η διαχείριση των αρχείων της δέσμης (beam files management). Και αυτό γιατί οι φυσικοί πρέπει να αποθηκεύουν τις προτιμήσεις τους για την διαμόρφωση της δέσμης σε αρχεία (αρχεία δέσμης), έτσι ώστε να μπορούν εύκολα να ανακτούν τις τιμές που έχουν καθορίσει για τα επιμέρους μηχανήματα. Επίσης, θα πρέπει να έχουν διαθέσιμο ένα σύνολο από χρήσιμες λειτουργίες για την διαχείριση των αρχείων αυτών. Πρέπει να σημειωθεί ότι τα αρχεία οργανώνονται κατά δέσμη και πείραμα. Οι λειτουργίες που περιγράψαμε αναλύονται παρακάτω.

- **Παρουσίαση αρχείων:**

Ο χρήστης, αφού προεπιλέξει με κάποιον τρόπο την δέσμη (π.χ. H4) και το πείραμα που τον ενδιαφέρει (π.χ. H4A), μπορεί να φορτώσει ένα παράθυρο (panel) με τα αρχεία της δέσμης (του συγκεκριμένου πειράματος). Η επιλογή της δέσμης και του πειράματος γίνεται ως εξής:

- Αν το panel λανσάρεται μέσα από την εφαρμογή CESAR, τότε η δέσμη είναι η ενεργή δέσμη (workspace) και το πείραμα, το πείραμα του ενεργού χρήστη ή κάποιο άλλο που επιλέγει ο χρήστης.

- Αν το panel λανσάρεται εξωτερικά από την εφαρμογή, τότε ρωτάται ο χρήστης για ένα πείραμα π.χ. "H4A", ενώ η δέσμη είναι η δέσμη του πειράματος (η περίπτωση αυτή ενδείκνυται μόνο για προγραμματιστές).

Το panel με τα αρχεία της δέσμης πρέπει να περιέχει τα εξής (με σειρά από πάνω προς τα κάτω):

(α) μια μπάρα που λειτουργεί σαν τίτλος και περιέχει χρήσιμες πληροφορίες, όπως την δέσμη, το πείραμα, την ημερομηνία καθώς και το όνομα και άλλες χρήσιμες πληροφορίες του αρχείου που είναι κάθε στιγμή φορτωμένο στη δέσμη,

(β) ένα πίνακα με όλα τα υπάρχοντα αρχεία αυτής της δέσμης και αυτού του πειράματος (και στήλες του πίνακα το όνομα κάθε αρχείου, ένα σχόλιο, την ορμή και την τελευταία τροποποίηση του αρχείου),

(γ) διάφορα κουμπιά για τις ενέργειες (σύνολο 10 και περιγράφονται στη συνέχεια) που μπορούμε να εκτελέσουμε στα αρχεία, και:

(δ) logging output όπως αυτό του κεντρικού παραθύρου της εφαρμογής, δηλαδή μία κονσόλα που αποκρύπτεται και εμφανίζεται με το πάτημα κάποιου κουμπιού και στην οποία τυπώνονται πληροφορίες για τις ενέργειες που διενεργούνται αλλά και σφάλματα, προειδοποιήσεις και πληροφορίες για το debugging. Επίσης, πρέπει να έχουμε και μία status line όπου αναγράφονται ίδιου είδους αλλά πιο σημαντικά μηνύματα που φαίνονται συνέχεια στο χρήστη.

- **Τροποποίηση αρχείων:**

Η τροποποίηση των αρχείων αφορά 3 από τις ενέργειες που μπορούν να γίνουν πάνω σε ένα αρχείο:

(α) ενέργεια "View" (Προβολή), η οποία όταν εφαρμόζεται σε ένα αρχείο, φορτώνει ένα νέο παράθυρο όπου φαίνονται (σε 2 ξεχωριστές καρτέλες) τα μηχανήματα 2 τύπων εξοπλισμού μαζί με τις τιμές τους (file references). Οι 2 αυτοί τύποι είναι οι magnets (μαγνήτες) και οι collimators (κατευθυντήρες). Από αυτό το παράθυρο μπορούμε να αλλάξουμε αυτές τις τιμές (file references) αλλά και να αλλάξουμε και το σχόλιο του αρχείου.

(β) ενέργεια "Edit" (Τροποποίηση), η οποία ανοίγει ένα αρχείο Excel, όπου μπορούμε να αλλάξουμε και πάλι τα file references του εξοπλισμού.

(γ) ενέργεια "Save Actual BeamRefs" (Αποθήκευση πραγματικών αναφορικών τιμών), η οποία αποθηκεύει τις αναφορικές τιμές του εξοπλισμού (Beam References) στο αρχείο. Δηλαδή, αντικαθιστά τις αναφορικές τιμές του αρχείου (File References) με τις αναφορικές τιμές του εξοπλισμού (Beam References).

Όλες οι δυνατότητες θα πρέπει να είναι διαθέσιμες σε όλους τους χρήστες πλην Guest.

- **Διαγραφή αρχείων:**

Ο χρήστης μέσω της ενέργειας "Delete" (Διαγραφή), μπορεί να διαγράψει ένα αρχείο, αφού πρώτα ρωτηθεί αν είναι σίγουρος για την ενέργειά του αυτή. Η ενέργεια θα πρέπει να είναι διαθέσιμη σε όλους τους χρήστες πλην Guest.

- **Σύγκριση αρχείων:**

Αν ο χρήστης επιλέξει 2 αρχεία και εκτελέσει την ενέργεια "Compare" (Σύγκριση), τότε εμφανίζεται ένα παράθυρο με 2 καρτέλες με magnets και collimators, όπως και στο "View". Η διαφορά είναι ότι εδώ βρίσκονται δίπλα-δίπλα οι τιμές για τον εξοπλισμό (file references) για τα 2 αρχεία και έτσι ο χρήστης μπορεί εύκολα να τα συγκρίνει. Επίσης, στο ίδιο παράθυρο παρέχεται η δυνατότητα αντιγραφής όλων των τιμών από το ένα αρχείο στο άλλο. Η δυνατότητα "Σύγκριση" πρέπει να είναι διαθέσιμη σε όλους τους χρήστες πλην Guest.

- **Φόρτωση αρχείων:**

Με την ενέργεια "Load Beam File" (Φόρτωση Αρχείου Δέσμης), μπορούμε να φορτώσουμε κάποιο από τα αρχεία στη δέσμη. Με αυτόν τον τρόπο, οι τιμές του αρχείου περνάνε στον πραγματικό εξοπλισμό και άρα, τα file references του αρχείου αυτού γίνονται και beam references και readings για τον εξοπλισμό. Πρέπει να είναι διαθέσιμη μόνο σε Beam Main Users και Super Users.

- **Επαναφορά προηγούμενης εκδοχής αρχείων:**

Αφορά την ενέργεια "Restore Beam File" (Ανάκτηση Αρχείου Δέσμης). Κάθε φορά που αλλάζουμε ένα αρχείο δέσμης η νέα αλλά και η παλιά εκδοχή του (version) αποθηκεύονται στο σύστημα CVS (Concurrent Versions System) σε μορφή αρχείου κειμένου txt. Έτσι, με την ενέργεια "Restore Beam File" δίνουμε την δυνατότητα στο χρήστη να ανακτήσει κάποια εκδοχή (version) κάποιου αρχείου δέσμης από την δεξαμενή αρχείων του CVS, αφού δώσει το όνομα του αρχείου και το CVS version του αρχείου. Αυτό που πρέπει να κάνουμε είναι να διαβάσουμε το txt του CVS, να ανακτήσουμε όλες τις αναφορικές τιμές του αρχείου και να τις περάσουμε στην αντίστοιχη εγγραφή αρχείου δέσμης του server. Αν το όνομα του αρχείου που δώσει ο χρήστης δεν υπάρχει, ο χρήστης επιχειρεί να ανακτήσει ένα διαγραμμένο αρχείο. Έτσι, αν τα στοιχεία που δίνει ο χρήστης είναι σωστά, τότε δημιουργείται ένα νέο αρχείο, όμοιο με την εκδοχή του αρχείου που έχει επιλέξει. Αν το αρχείο που επιλέξει ο χρήστης υπάρχει ήδη, τότε αυτός ρωτάται αν θέλει να το αντικαταστήσει με την εκδοχή που έχει επιλέξει ή να ακυρώσει την επαναφορά της έκδοσης του αρχείου. Η ενέργεια "Restore Beam File" πρέπει να είναι διαθέσιμη μόνο σε Super Users.

- **Αντιγραφή αρχείων:**

Ο χρήστης έχει 2 δυνατότητες:

(α) με την ενέργεια "Copy" (Αντιγραφή), μπορεί να δημιουργήσει ένα ακριβές αντίγραφο ενός αρχείου, δίνοντας ένα όνομα για το νέο αρχείο, το πείραμα και το νέο σχόλιο για το υπό δημιουργία αρχείο. Αν το όνομα υπάρχει, τότε ο χρήστης ρωτάται αν θέλει να αντικαταστήσει το υπάρχον με το νέο αρχείο ή αν θέλει να ακυρώσει την διαδικασία.

(β) με την ενέργεια "Extrapolate", μπορεί να κάνει ακριβώς το ίδιο με την διαφορά ότι παρέχεται η δυνατότητα το νέο αρχείο να είναι διαφορετικής ορμής.

Όλες οι δυνατότητες οφείλουν να είναι διαθέσιμες σε όλους τους χρήστες πλην Guest.

- **Ανανέωση:**

Η ενέργεια "Refresh" (Ανανέωση) ανανεώνει τον πίνακα. Πρέπει να είναι διαθέσιμη σε όλους τους χρήστες.

4.2.3 Κατάσταση Εξοπλισμού

Οι χρήστες του CESAR πρέπει να έχουν την εποπτεία της κατάστασης του εξοπλισμού και να μπορούν να την επηρεάσουν. Κάθε μηχάνημα ανήκει σε μία δεδομένη δέσμη. Οι διαθέσιμες λειτουργίες του προγράμματός μας πρέπει να είναι οι εξής:

- **Παρουσίαση εξοπλισμού:**

Ο χρήστης, αφού επιλέξει με κάποιον τρόπο την δέσμη (π.χ. M2) και τον τύπο του εξοπλισμού (magnets, collimators, Obstacles, Taxes κ.α.- σύνολο 19 διαφορετικά είδη), μπορεί να δει ένα παράθυρο (panel) με την κατάσταση του εξοπλισμού.

Η επιλογή της δέσμης και του τύπου του εξοπλισμού γίνεται ως εξής:

- Αν το panel λανσάρεται από την εφαρμογή CESAR, τότε η δέσμη είναι η ενεργή δέσμη (workspace) και το είδος του εξοπλισμού καθορίζεται από την επιλογή του χρήστη (π.χ. επιλέγει το "Magnet Status" για να δει τους μαγνήτες).

- Αν το panel λανσάρεται εξωτερικά από την εφαρμογή, τότε ρωτάται ο χρήστης για ένα είδος εξοπλισμού π.χ. "Magnets" και για την δέσμη, π.χ. "H2" (η περίπτωση αυτή ενδείκνυται μόνο για προγραμματιστές).

Το panel με την κατάσταση του εξοπλισμού πρέπει να περιέχει τα εξής (με σειρά από πάνω προς τα κάτω):

(α) μια μπάρα που λειτουργεί σαν τίτλος όμοια με το Beam Files Panel,

(β) ένα πίνακα με όλον τον υπάρχοντα εξοπλισμό αυτού του τύπου και της δέσμης που επιλέχτηκε και στήλες του πίνακα το όνομα κάθε μηχανήματος, ένα σχόλιο, τα readings, τα beam references και άλλες χρήσιμες πληροφορίες ανάλογα με το είδος του εξοπλισμού. Στον πίνακα αυτό φαίνεται και η κατάσταση του εξοπλισμού με την βοήθεια κατάλληλης χρωματικής σήμανση (highlighting). Πιο συγκεκριμένα, ένα μηχάνημα μπορεί να έχει κατάσταση «καλή» (OK), «προειδοποίηση» (WARNING), «σφάλμα» (ERROR), «απασχολημένος εξοπλισμός» (BUSY), «μη ανανεωμένο» (NOT UPDATED) ή «αλλαγμένη ιδιότητα» (PROPERTY CHANGE). Αν η κατάσταση ενός μηχανήματος είναι "OK", τότε τα γράμματά του παραμένουν μαύρα και το φόντο άσπρο (όπως η προεπιλογή για έναν πίνακα). Αν η κατάστασή του είναι "WARNING", τότε τα γράμματά του γίνονται πορτοκαλί. Αν η κατάστασή του γίνει "ERROR", τα γράμματα γίνονται κόκκινα. Αν η κατάσταση μεταβεί στο "BUSY", τότε το φόντο γίνεται κίτρινο. Αν η κατάσταση γίνει "NOT UPDATED", τα γράμματα γίνονται μπλε. Τέλος, αν η κατάσταση γίνει "PROPERTY CHANGE", τα γράμματα γίνονται μοβ (αυτή η τελευταία κατάσταση χρησιμοποιείται μόνο για τύπο εξοπλισμού Scraper).

Το ποια κατάσταση επικρατεί για ένα μηχάνημα καθορίζεται με κανόνες που μας γνωστοποίησαν οι ειδικοί της δέσμης και που δεν είναι σκόπιμο να αναφέρουμε αναλυτικά εδώ. Ειδικά, η κατάσταση "NOT UPDATED" γίνεται ενεργή αν παρουσιάστηκε σφάλμα κατά την διάρκεια προσπάθειας ανάγνωσης μιας τιμής κάποιου μηχανήματος ή αν έχει μεσολαβήσει μεγάλο διάστημα (π.χ. 2 λεπτά), χωρίς το μηχάνημα να έχει στείλει reading στο panel μας.

Τα στοιχεία του πίνακα με τον εξοπλισμό ανανεώνονται σε πραγματικό χρόνο.

(γ) διάφορα κουμπιά για τις ενέργειες (διαφορετικές ανάλογα με το είδος του εξοπλισμού αλλά και μερικές κοινές για όλα τα είδη) που μπορούμε να εκτελέσουμε στα μηχανήματα

(δ) logging output όπως αυτό του Beam Files Panel.

- <u>Τροποποίηση εξοπλισμού:</u>

Υπάρχουν διάφορες ενέργειες, οι οποίες διαφέρουν ανάλογα με το είδος του εξοπλισμού, ώστε ο χρήστης να μπορεί να αλλάζει τις παραμέτρους του.

Όλες οι δυνατότητες πρέπει να είναι διαθέσιμες σε όλους τους χρήστες πλην Guest.

- <u>Monitoring εξοπλισμού:</u>

Ανάμεσα στις ενέργειες, πρέπει να υπάρχουν σε σχεδόν όλα τα Panels, ενέργειες που αφορούν την ανανεωσιμότητα του πίνακα με τον εξοπλισμό. Π.χ. πρέπει να υπάρχουν 2 κουμπιά με τα ονόματα "Run" (Τρέξιμο) και "Hold" (Σταμάτημα). Το 1^{ο} εκκινεί την

αυτόματη και σε πραγματικό χρόνο ανάγνωση των παραμέτρων του εξοπλισμού και την απεικόνισή τους στον πίνακα, ενώ το 2^{ο} σταματά αυτή τη διαδικασία. Με την φόρτωση του παραθύρου βρισκόμαστε ως προεπιλογή στην κατάσταση “Run”, ενώ μετά από μία ώρα η κατάσταση γίνεται αυτόματα “Hold” για λόγους που αφορούν την επεξεργαστική ισχύ του συστήματος. Φυσικά, ο χρήστης πρέπει να μπορεί να αλλάξει ανά πάσα στιγμή την κατάσταση στην οποία βρίσκεται το παράθυρο. Επίσης, υπάρχει και η ενέργεια “Refresh” (Ανανέωση) με δύο επιλογές: (i) “Refresh All” (Ανανέωση Όλων), που ανανεώνει όλον τον πίνακα και (ii) “Refresh Selected” (Ανανέωση Επιλεγμένων), που ανανεώνει μόνο τον επιλεγμένο εξοπλισμό. Όλες οι δυνατότητες πρέπει να είναι διαθέσιμες σε όλους τους χρήστες πλην Guest.

- **Εκτύπωση:**

Στα παράθυρα κατάστασης, καλό είναι να υπάρχει πάντα η δυνατότητα εκτύπωσης του πίνακα με τον εξοπλισμό μέσω ενός κουμπιού “Print”. Η επιλογή εκτύπωσης πρέπει να είναι διαθέσιμη σε όλους τους χρήστες πλην Guest.

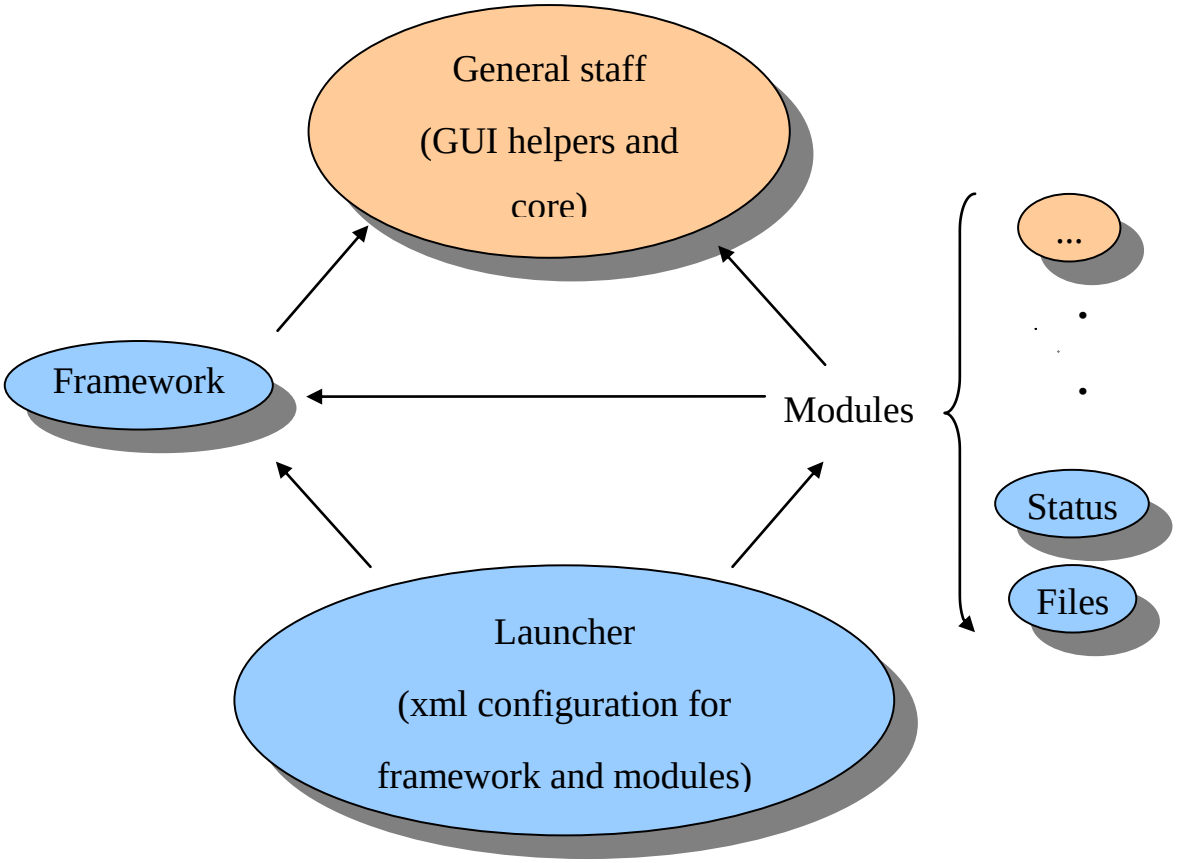
5

Σχεδίαση Συστήματος

Στο παρόν κεφάλαιο θα περιγράψουμε τη σχεδίαση του συστήματος που περιγράψαμε στο προηγούμενο κεφάλαιο. Αρχικά (παράγραφος 5.1) , θα αναφέρουμε την γενική αρχιτεκτονική ολόκληρου του συστήματος που θέλουμε να αναπτύξουμε. Με τον όρο αρχιτεκτονική εννοούμε την οργάνωση των κλάσεων που θα υλοποιηθούν στο επόμενο κεφάλαιο και την σημασία τους. Στη συνέχεια (παράγραφος 5.2), θα αναφέρουμε λεπτομέρειες για κάθε κλάση, δηλαδή μια γενική περιγραφή των λειτουργιών τους (ορισμένες μεθόδους τους).

5.1 Αρχιτεκτονική

Αρχικά, να θυμίσουμε ότι δουλεύουμε πάντα στο επίπεδο της παρουσίασης του προγράμματος CESAR. Έτσι, σύμφωνα με λογική που θα εξηγήσουμε παρακάτω, χωρίζουμε το σύστημά μας σε υποσυστήματα, όπως φαίνεται στο ακόλουθο σχήμα:



Σχήμα 1: Αρχιτεκτονική του συστήματος παρουσίασης του CESAR

Στο παραπάνω σχήμα βλέπουμε τα ακόλουθα:

- Με κύκλους αναπαριστούμε ξεχωριστά υποσυστήματα.
- Με μπλε χρώμα αναπαριστούμε τα υποσυστήματα που αναπτύξαμε εμείς και ασχολούμαστε με αυτά στην παρούσα Διπλωματική Εργασία, ενώ με πορτοκαλί τα υποσυστήματα που έχουν αναπτυχθεί από άλλους Μηχανικούς Λογισμικού του CERN.
- Με βέλη αναπαριστούμε τις εξαρτήσεις (dependencies) μεταξύ των υποσυστημάτων. Δηλαδή, δείχνουμε από ποια υποσυστήματα έχουμε αναφορές σε ποια.

Το περιεχόμενο των διαφόρων υποσυστημάτων εξηγείται ως εξής:

- Τα Status και Files είναι τα υποσυστήματα της κατάστασης του εξοπλισμού και των αρχείων της δέσμης, αντίστοιχα, των οποίων τις απαιτήσεις αναφέραμε στο προηγούμενο κεφάλαιο και τα οποία θα αναλύσουμε περαιτέρω ως προς την σχεδίαση στις επόμενες παραγράφους (5.1.2 και 5.1.3).
- Τα Framework και Launcher αντιστοιχούν στις απαιτήσεις της Εφαρμογής που αναφέραμε στο προηγούμενο κεφάλαιο. Το μεν Framework είναι το

γενικό interface μιας εφαρμογής η οποία έχει τα χαρακτηριστικά που αναφέραμε π.χ. είναι οργανωμένη σε workspaces. Το δε Launcher διαμορφώνει το Framework ώστε να παράγουμε το CESAR, δηλαδή την δική μας εφαρμογή. Επίσης, ενσωματώνει στο Framework τα υποσυστήματα Status και Files (και άλλα που δεν έχουμε αναπτύξει εμείς) και τέλος, το εκκινεί. Περισσότερες πληροφορίες για την εσωτερική σχεδίαση αυτών των υποσυστημάτων αναφέρουμε στις παραγράφους 5.1.1 και 5.1.4, που ακολουθούν.

- Το General Staff περιέχει προγραμματιστικά εργαλεία που έχουν αναπτυχθεί από άλλους Μηχανικούς Λογισμικού και τα οποία επαναχρησιμοποιήσαμε. Αυτά τα εργαλεία είναι κατά κύριο λόγο οι μέθοδοι (handlers) που έχουν απευθείας πρόσβαση στα χαμηλά επίπεδα του συστήματος CESAR, όπως τον πραγματικό εξοπλισμό (hardware) και τα αρχεία δέσμης (server). Επίσης, άλλα εργαλεία που χρησιμοποιήσαμε έχουν σχέση με GUI (Graphical User Interface), όπως για παράδειγμα τα παράθυρα με τα consoles που αναφέραμε στο κεφάλαιο των Απαιτήσεων. Περισσότερες πληροφορίες θα δοθούν στο κεφάλαιο της Υλοποίησης.

Είναι σημαντικό να επισημάνουμε ότι αυτή η αρχιτεκτονική που διαλέξαμε για το σύστημά μας εξυπηρετεί τις απαιτήσεις που αναφέραμε στο προηγούμενο κεφάλαιο, κυρίως όσον αφορά την γενική εφαρμογή.

Πρώτον, παρατηρούμε ότι το υποσύστημα Framework δεν εξαρτάται από υποσυστήματα τα οποία χαρακτηρίζουμε σαν modules, όπως το Status και το Files. Αυτό αποσκοπεί στη δυνατότητα εύκολης προσθήκης και αφαίρεσης των επιμέρους modules σε /από την εφαρμογή μας, κάτι που διευκολύνει την συντήρησή της από τους προγραμματιστές που θα δουλέψουν στο CESAR τα επόμενα χρόνια. Τον διαχωρισμό αυτόν τον πετύχαμε με την βοήθεια του Launcher, το οποίο λειτουργεί σαν ομπρέλα στην αρχιτεκτονική μας, καθώς, έχοντας εξαρτήσεις και προς το Framework και προς τα modules, επιτελεί τις λειτουργίες που αναφέραμε.

Δεύτερον, το framework είναι αφηρημένο (abstract) και συνεπώς επαναχρησιμοποιούμενο από άλλους Μηχανικούς Λογισμικού που θέλουν να αναπτύξουν μία εφαρμογή με παρόμοια χαρακτηριστικά με το CESAR. Έτσι, η διαμόρφωση του Framework για την δική μας εφαρμογή (CESAR) γίνεται ανεξάρτητα από αυτό (στο Launcher), ώστε η εφαρμογή να αποκτήσει τα CESAR-ειδικά (CESAR-specific) χαρακτηριστικά της. Το μόνο που έχουν δηλαδή να αλλάξουν άλλοι προγραμματιστές για την παραγωγή της δικής τους εφαρμογής είναι η διαμόρφωση του Framework. Έτσι, γλιτώνουν από τον κόπο να αναπτύξουν το Framework εξ' αρχής. Επίσης, στην αφαιρετική συμπεριφορά του Framework φυσικά

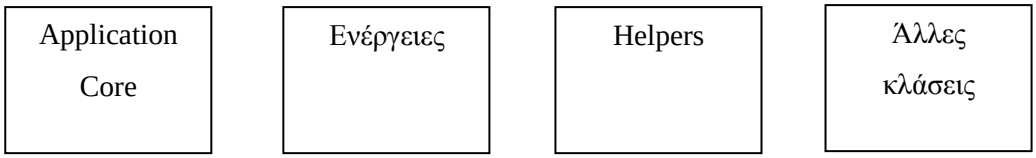
συμβάλλει και το γεγονός ότι το Framework δεν έχει εξάρτηση προς τα modules (που τα περισσότερα είναι CESAR-specific, όπως το Status και το Files), όπως αναφέραμε και προηγουμένως.

Τρίτον, παρατηρούμε ότι τα πιο ειδικά υποσυστήματα έχουν εξάρτηση προς πιο γενικά και όχι το αντίθετο. Για παράδειγμα, το Framework έχει εξάρτηση προς το General Stuff και τα modules προς το Framework και το General Stuff. Αυτό μας βολεύει στην επαναχρησιμοποίηση των στοιχείων που μας ενδιαφέρουν κάθε φορά και όχι άλλων που ενδεχομένως να μην χρειαζόμαστε.

Στη συνέχεια, θα αναφέρουμε την εσωτερική αρχιτεκτονική των υποσυστημάτων που περιγράψαμε παραπάνω, δηλαδή του Framework, του Launcher, της Διαχείρισης Αρχείων Δέσμης και της Κατάστασης Εξοπλισμού.

5.1.1 Framework

Το υποσύστημα Framework το χωρίζουμε στα εξής υποσυστήματα:



Σχήμα 2: Υποσυστήματα του συστήματος Framework

- Application Core: Είναι η καρδιά της εφαρμογής μας και περιέχει τις πλέον θεμελιώδεις κλάσεις αυτής.

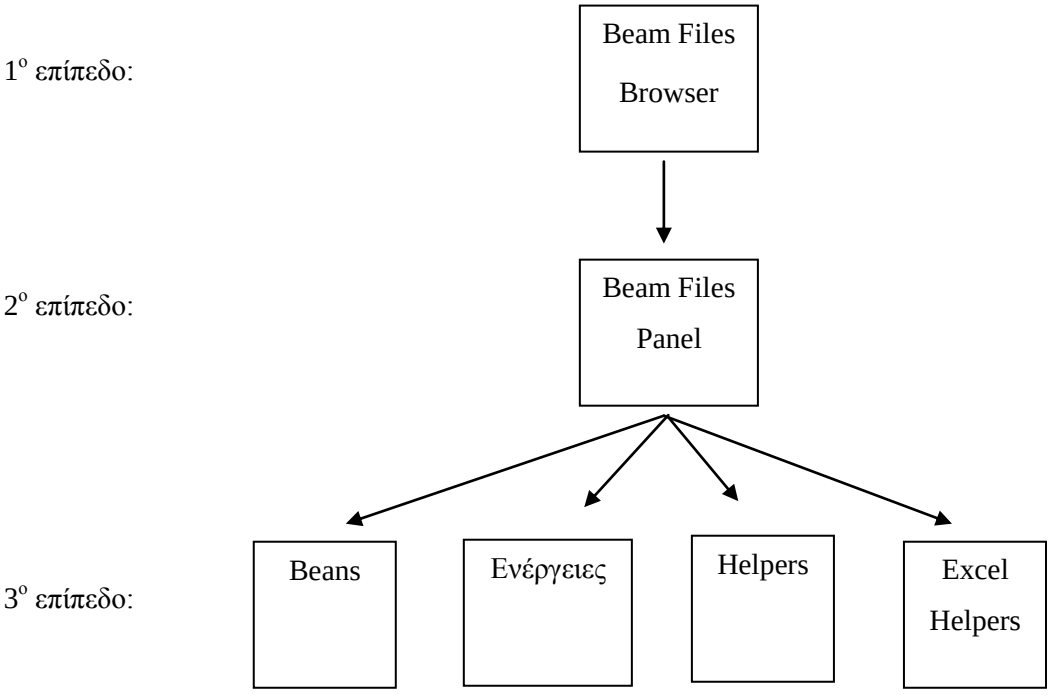
- Ενέργειες: Είναι όλες οι ενέργειες του Framework που πρέπει να υποστηρίζονται, όπως η προσθήκη ή η αφαίρεση κάποιου workspace από το χρήστη.

- Helpers: Είναι βοηθητικές κλάσεις που χρησιμοποιούνται και από το Framework και από τα modules της εφαρμογής μας και μας επιτρέπουν να χειριζόμαστε με ενιαίο τρόπο όλα τα μηνύματα (σφάλματος, προειδοποίησης, πληροφορίας κλπ) που εμφανίζουμε στο χρήστη, να εξασφαλίζουμε την ασφάλεια της εφαρμογής μας και να παρέχουμε στην εφαρμογή μας κάποια χρήσιμα εργαλεία, όπως ένα είδος πίνακα που χρειάζεται στα υποσυστήματα Status και Files για να παρουσιάζουμε τα μηχανήματα ή τα αρχεία της δέσμης.

- Άλλες κλάσεις: Κάποιες όχι και τόσο σημαντικές κλάσεις, όπως για παράδειγμα αυτήν που μας παρέχει ένα splash screen κατά την έναρξη της εφαρμογής μας μέχρι αυτή να φορτώσει.

5.1.2 Διαχείριση Αρχείων Δέσμης

Η γενική αρχιτεκτονική του υποσυστήματος είναι η εξής:



Σχήμα 3: Αρχιτεκτονική του υποσυστήματος Beam Files

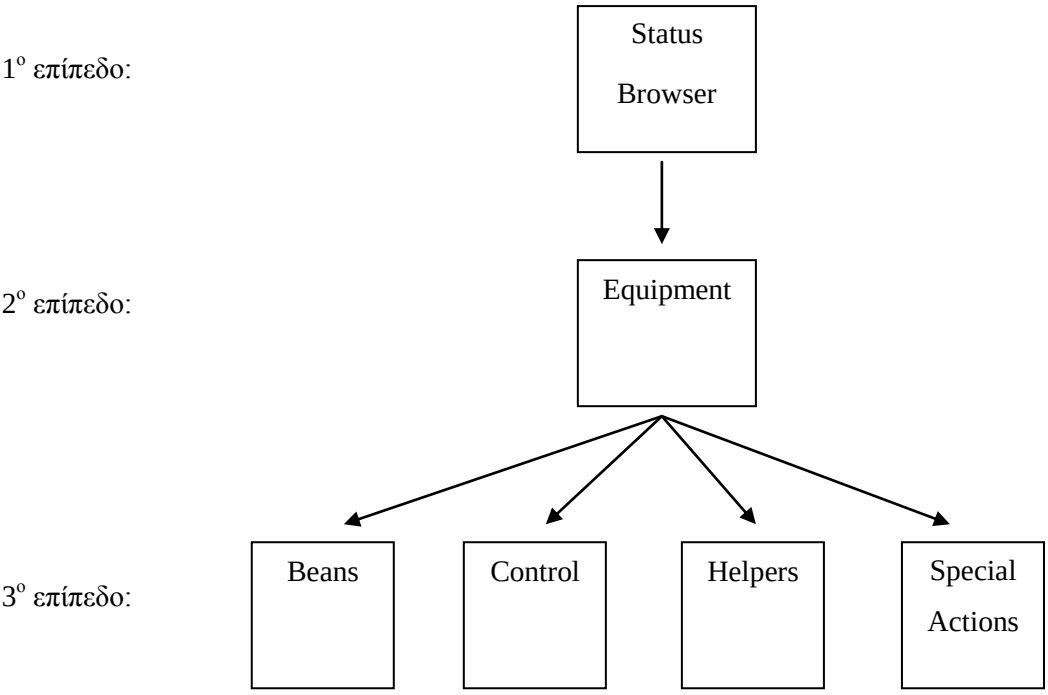
- 1ο επίπεδο: Παρατηρούμε το υποσύστημα Beam Files Browser, το οποίο, όπως θα δούμε, περιέχει κλάσεις που είναι υπεύθυνες για την φόρτωση του κυρίως παραθύρου του υποσυστήματος, με τα αρχεία μίας δέσμης και ενός πειράματος. Η εμφάνιση του παραθύρου αυτού μπορεί να γίνεται μέσα από την εφαρμογή CESAR ή ανεξάρτητα από αυτήν.

- 2ο επίπεδο: Έχουμε το υποσύστημα Beam Files Panel, το οποίο περιέχει διάφορες κλάσεις και αποτελεί το περιεχόμενο του κεντρικού παραθύρου (panel) με τα αρχεία της δέσμης αλλά και άλλων επιμέρους παραθύρων. Εδώ, περιέχονται όλες οι λειτουργίες που αναλύσαμε στο προηγούμενο κεφάλαιο, με την βοήθεια, βέβαια και των κλάσεων του 3ου επιπέδου.

- 3ο επίπεδο: Στην κατηγορία κλάσεων Beans, υπάρχουν κλάσεις, οι οποίες αναπαριστούν εξοπλισμό ή αρχείο δέσμης. Η κατηγορία Ενέργειες έχει τις ενέργειες (actions) του κεντρικού και των επιμέρους παραθύρων των αρχείων της δέσμης. Η κατηγορία Helpers παρέχει βοηθητικές κλάσεις, οι οποίες ελέγχουν κατευθείαν τα αρχεία και τον εξοπλισμό, παρέχουν χρήσιμη GUI (Graphical User Interface) υποστήριξη, παρέχουν το μηχανισμό για την εφαρμογή των διαθέσιμων ενεργειών κ.α. Τέλος, η κατηγορία Excel Helpers παρέχει κλάσεις που εκκινούν το πρόγραμμα Excel με συγκεκριμένες πληροφορίες προς επεξεργασία (βλ. προηγούμενο κεφάλαιο). Με την τελευταία αυτή κατηγορία δεν θα ασχοληθούμε, καθώς υπήρξε ως αποτέλεσμα δουλειάς άλλων εργαζομένων του CERN.

5.1.3 Κατάσταση Εξοπλισμού

Η γενική αρχιτεκτονική αυτού του υποσυστήματος μοιάζει με αυτήν του προηγούμενου και είναι η εξής:



Σχήμα 4: Αρχιτεκτονική του υποσυστήματος Equipment Status

- 1ο επίπεδο: Παρατηρούμε το υποσύστημα Status Browser, το οποίο, όπως θα δούμε, περιέχει κλάσεις που είναι υπεύθυνες για την φόρτωση ενός παραθύρου με τα μηχανήματα ενός είδους εξοπλισμού και μίας δέσμης. Η εμφάνιση του παραθύρου αυτού γίνεται μέσα από το πρόγραμμα CESAR ή ανεξάρτητα από αυτό.

- 2ο επίπεδο: Το υποσύστημα Equipment περιέχει διάφορες κλάσεις που μπορεί να χαρακτηριστούν ως οι βασικές κλάσεις αναπαράστασης της κατάστασης του εξοπλισμού. Περιέχει, δηλαδή, μία κλάση για κάθε τύπο εξοπλισμού, η οποία αποτελεί το περιεχόμενο του κεντρικού παραθύρου εξοπλισμού (panel) του τύπου αυτού με όλες τις πληροφορίες και τις λειτουργίες που απαιτούνται και έχουμε αναφέρει στο προηγούμενο κεφάλαιο. Συνεπώς, το υποσύστημα αυτό είναι υπεύθυνο για την παρουσίαση του εξοπλισμού (διάφορες ιδιότητες και την κατάστασή του) και την εφαρμογή διαφόρων ενεργειών στον εξοπλισμό. Για τον λόγο αυτό, χρησιμοποιεί και τα υποσυστήματα του 3ου επιπέδου.

- 3ο επίπεδο: Η κατηγορία κλάσεων Beans παρέχει τις λεπτομέρειες κάθε είδους εξοπλισμού, όπως τις ιδιότητες (κυρίως αριθμητικές παράμετροι), την κατάσταση («καλή κατάσταση» (OK), «προειδοποίηση» (WARNING), «σφάλμα» (ERROR), «απασχολημένος εξοπλισμός» (BUSY), «μη ανανεωμένο» (NOT UPDATED) και «αλλαγμένη ιδιότητα» (PROPERTY

CHANGE)), τον τρόπο που ανανεώνονται οι ιδιότητες και η κατάσταση κλπ.. Με άλλα λόγια, κάθε κλάση της κατηγορίας αυτής είναι μία αναπαράσταση (μοντελοποίηση) ενός είδους εξοπλισμού. Η κατηγορία Control παρέχει κάποιες κλάσεις που ελέγχουν κάποια είδη εξοπλισμού (εφαρμογή ενεργειών εδώ και στο επίπεδο 2). Η κατηγορία Helpers παρέχει βοηθητικές κλάσεις, που μπορούν να χρησιμοποιηθούν από πολλά είδη εξοπλισμού. Τέλος, η κατηγορία Special Actions παρέχει κλάσεις οι οποίες εφαρμόζουν λίγες και ειδικές ενέργειες στον εξοπλισμό ("Get Logbook information", "Quick", "Degauss") και δεν θα αναλυθεί καθώς δεν αναπτύχθηκε από εμάς.

5.1.4 Launcher

Το υποσύστημα αυτό δεν έχει ιδιαίτερα πολύπλοκη αρχιτεκτονική και μπορεί να χωριστεί σε δύο μέρη:

- Κλάσεις για την εκκίνηση της εφαρμογής: Παρέχει μία μέθοδο που εκκινεί (τρέχει) την εφαρμογή μας, η οποία είναι βασισμένη στο framework.

- Αρχεία για την διαμόρφωση του Framework: Σε ειδικά διαμορφωμένα αρχεία παρέχουμε μια περιγραφή (διαμόρφωση) για συγκεκριμένα πεδία (στοιχεία) της εφαρμογής (π.χ. το όνομά της, την υλοποίηση συγκεκριμένων αφηρημένων στοιχείων κλπ) και καθορίζουμε το περιεχόμενό της (modules της εφαρμογής και διαμόρφωση αυτών).

5.2 Περιγραφή Κλάσεων

Σε αυτήν την παράγραφο, περιγράφουμε συνοπτικά την γενική λειτουργία και σε κάποιες περιπτώσεις, τις επιμέρους μεθόδους των κλάσεών μας. Κλάσεις και μέθοδοι που δεν δημιουργήσαμε εμείς ή που δεν έχουν ζωτική σημασία για την εφαρμογή μας, δεν αναλύονται.

5.2.1 Framework

- Application Core:

- Application:
  Είναι η βασικότερη κλάση της εφαρμογής και αναπαριστά την ίδια την εφαρμογή. Η κλάση αυτή πρέπει να είναι singleton (και για αυτό ακολουθεί το σχεδιαστικό μόρφημα Singleton) και παρέχει μεθόδους για την διαχείριση και τροποποίηση του κύκλου ζωής της εφαρμογής. Βασικές μέθοδοι αυτής είναι:
  1. instance():
  Επιστρέφει το μοναδικό instance της κλάσης αυτής σε άλλες κλάσης που το

χρειάζονται για να τροποποιήσουν για κάποιο λόγο την εφαρμογή (π.χ. κάποια ενέργεια πρέπει να δημιουργήσει κάποιο παράθυρο στο ενεργό workspace), μέσω των μεθόδων της παρούσας κλάσης. Το instance της παρούσας κλάσης πρέπει να είναι μοναδικό, εφόσον, βέβαια, η εφαρμογή έχει φορτωθεί (δηλαδή έχουμε με κάποιο τρόπο δημιουργήσει instance της κλάσης αυτής).

2. isFrameOpen():

Επιστρέφει αληθή τιμή αν ένα παράθυρο με δοσμένη περιγραφή (σε κάθε παράθυρο που προσθέτουμε σε κάποιο workspace, δίνουμε και μία περιγραφή όπως θα δούμε παρακάτω) υπάρχει στο ενεργό workspace, διαφορετικά επιστρέφουμε την ψευδή τιμή. Αυτή η μέθοδος είναι χρήσιμη ώστε να μην προσθέτουμε παράθυρα που είναι ήδη ανοιχτά. Η περιγραφή κάθε είδος παραθύρου πρέπει να είναι μοναδική.

3. getActiveWorkspace():

Επιστρέφει το ενεργό workspace της εφαρμογής μας. Χρήσιμη μέθοδος για τις ενέργειες που προσθέτουν κάποιο παράθυρο στην εφαρμογή μας (π.χ. προσθήκη κάποιου παραθύρου Equipment Status ή Beam Files) ώστε να ξέρουν σε ποιο workspace να το προσθέσουν αλλά και το ποια είναι η δέσμη που μας ενδιαφέρει (θυμίζουμε ότι το όνομα του workspace συμπίπτει με το όνομα κάποιας δέσμης).

4. getAllWorkspaces():

Επιστρέφει όλα τα ανοιχτά workspaces. Χρήσιμο για να μην ανοίγουμε ξανά ήδη ανοιχτά workspaces.

5. getActiveWorkspaceName(), getWorkspacesNames():

Επιστρέφουν το όνομα του ενεργού workspace και τα ονόματα όλων των ανοιχτών workspaces, αντίστοιχα. Χρήσιμες μέθοδοι για τους ίδιους λόγους με τις μεθόδους 3 και 4.

6. addWorkspace(), addWorkspaces():

Προσθέτουν workspace ή workspaces (αντίστοιχα) με δοσμένο /α όνομα /ατα στην εφαρμογή μας. Χρήσιμο για την ενέργεια προσθήκης δέσμης στην εφαρμογή.

7. removeWorkspace(), removeWorkspaces():

Κλείνουν workspace ή workspaces (αντίστοιχα) με δοσμένο /α όνομα /ατα στην εφαρμογή μας. Χρήσιμο για την ενέργεια κλεισίματος δέσμης στην εφαρμογή.

8. removeAllWorkspaces():

Κλείνει όλα τα ανοιχτά workspaces. Χρήσιμο για την ενέργεια κλεισίματος όλων των workspaces της εφαρμογής.

9. start():

Εκκινεί την εφαρμογή μας κάνοντας αυτόματο login (autologin), εάν αναγνωριστεί η τοποθεσία από την οποία τρέχουμε την εφαρμογή ως ιδιοκτησία κάποιου χρήστη με έγκυρο λογαριασμό και καλώντας τις επόμενες τρεις μεθόδους.

10. refresh():

Κατασκευάζει το menu και το toolbar της εφαρμογής μας και τα προσθέτει στο κεντρικό παράθυρό της.

11. configureTabs():

Διαμορφώνει τις καρτέλες (tabs) της εφαρμογής μας, δίνοντάς τους το πλάτος που επιθυμούμε και προσθέτοντάς τους τις ενέργειες κλεισίματος ενός επιλεγμένου ή όλων των ανοιχτών workspaces (επιλογές που είναι διαθέσιμες ως pop-up menu με δεξί κλικ του ποντικιού πάνω σε μία καρτέλα).

12. launchMainFrame():

Εμφανίζει μεγιστοποιημένο το κεντρικό παράθυρο της εφαρμογής.

- ApplicationException:

  Είναι ένα είδος εξαίρεσης που συμβαίνει όταν κάποιο σημαντικό σφάλμα λαμβάνει χώρα κατά την ομαλή λειτουργία της εφαρμογής μας.

- WorkspaceContainerWithLogging:

  Είναι singleton και το instance της κλάσης αυτής περιέχει και χειρίζεται τα διάφορα workspaces της εφαρμογής (π.χ. ανοίγει ή κλείνει κάποια workspaces). Αυτήν την κλάση χρησιμοποιεί κατά κόρον η κλάση Application για να επιτελέσει πολλές από τις λειτουργίες (μεθόδους) που αναφέραμε.

- Ενέργειες:

- ActionGroup:

  Αναπαριστά ένα σύνολο από άλλα αντικείμενα της ίδιας κλάσης ή απλές ενέργειες. Αυτή η οργάνωση μας βολεύει στην κατασκευή των menus και toolbars, όπου πρέπει να ομαδοποιούμε τις ενέργειες μεταξύ τους. Για παράδειγμα. στο menu “Files”, πρέπει να υπάρχουν όλες οι ενέργειες που μπορεί να κάνει ο χρήστης από το κεντρικό παράθυρο του CESAR όσον αφορά τα αρχεία της δέσμης. Επίσης, πρέπει menu να μπορούν έχουν υπο-menu.

- BrowsePanelAction:

  Είναι μία ενέργεια που φορτώνει κάποιο panel στο ενεργό workspace σαν εσωτερικό παράθυρο σε αυτό. Η κλάση αυτή είναι abstract, άρα πρέπει να επεκταθεί ώστε να συγκεκριμενοποιήσουμε το panel για το οποίο μιλάμε. Παραδείγματα μιας τέτοιας υποκλάσης είναι οι StatusPanelAction και BeamFilesPanelAction, τις οποίες θα αναφέρουμε στις παραγράφους 5.2.2 και 5.2.3.

- GeneralAction:

  Είναι μια οποιαδήποτε ενέργεια που έχουμε στην εφαρμογή μας. Δηλαδή, για να έχουμε κάποια ενέργεια (π.χ. οι 5 κλάσεις που ακολουθούν στην παρούσα ενότητα

των Ενεργειών) πρέπει να επεκτείνουμε την κλάση αυτή ώστε να καθορίσουμε το τι θα κάνει η ενέργεια όταν καλείται. Η παρούσα κλάση είναι χρήσιμη καθώς προσφέρει σε μια ενέργεια την δυνατότητα να έχει όνομα, tooltip, εικόνα, συντόμευση πλήκτρων (key shortcut) και επίσης να «παρακολουθεί» για αλλαγή στον ενεργό χρήστη της εφαρμογής μας μέσω επιτυχημένων login/logout.

- BeamRefsToFileAction:

  Ενέργεια "Save Actual BeamRefs" των panels των Αρχείων της δέσμης για το τελευταία φορτωμένο αρχείο στη δέσμη. Σημαντικό είναι να τονίσουμε ότι λόγω του ότι η ενέργεια είναι CESAR-specific, δεν είναι συνδεμένη με τις υπόλοιπες κλάσεις του Framework (χρησιμοποιείται και προστίθεται στην εφαρμογή μέσω της διαμόρφωσής της στο υποσύστημα Launcher).

- NewWorkspaceAction:

  Ενέργεια προσθήκης ενός workspace στην εφαρμογή μας. Πριν την προσθήκη, ο χρήστης ρωτάται ποιες δέσμες επιθυμεί να προσθέσει μέσα από μια λίστα που καθορίζεται ως το σύνολο των γνωστών στο σύστημα δεσμών, μείον τις δέσμες όπου ο ενεργός χρήστης δεν έχει πρόσβαση, μείον της δέσμες που αντιστοιχούν σε ήδη ανοιγμένα workspaces στην εφαρμογή. Είναι σημαντικό να προσθέσουμε ότι η ενέργεια αυτή, αν και φαίνεται να είναι CESAR-specific, δεν είναι στην πραγματικότητα, γιατί, αφ' ενός, ότι έχει να κάνει με ονόματα προσβάσιμων workspaces το κρύβουμε μέσα στο interface WorkspaceSecurityManager (την αναφέρουμε λίγο πιο κάτω) και αφ' ετέρου διαχωρίζουμε την υλοποίηση αυτού του interface από τον κώδικα της παρούσας κλάσης (στην υλοποίηση του interface είναι προφανές ότι δεν μπορούμε να αποφύγουμε τα CESAR-specific στοιχεία αλλά την υλοποίηση αυτή την διαλέγουμε μέσω της διαμόρφωσης της εφαρμογής μας στο υποσύστημα Launcher).

- Autologin:

  Κάνει αυτόματο login του χρήστη ανάλογα με την τοποθεσία από την οποία τρέχουμε την εφαρμογή. Αν δεν βρεθεί γνωστός χρήστης για αυτή την τοποθεσία, συνδέουμε το χρήστη ως Guest στο σύστημα (ελάχιστα δικαιώματα). Δεν μπορούμε να κάνουμε logout τον χρήστη που έχει γίνει autologin (μπορούμε να κάνουμε όμως άλλα login).

- LoginAction:

  Ενέργεια απόπειρας login κάποιου χρήστη στο σύστημα. Μετά από επιτυχημένο login, ο ενεργός χρήστης της εφαρμογής αλλάζει αλλά κρατούνται στην μνήμη όλοι οι προηγούμενα συνδεδεμένοι χρήστες.

- LogoutAction:
  Κάνει logout τον ενεργό χρήστη του συστήματος και κάνει νέο ενεργό τον προηγούμενα συνδεδεμένο χρήστη της εφαρμογής μας.
- PrivilegedAction:
  Ενέργεια που είναι διαθέσιμη για τους χρήστες που αποτελούν προεπιλογή της interface SecurityService (θα μιλήσουμε στην παράγραφο Helpers για αυτή) και είναι υποκλάση της SecurityListeningAction που ακολουθεί. Χρησιμοποιείται από το module Status για τις ενέργειες που αλλάζουν τον πραγματικό εξοπλισμό.
- SecurityListeningAction:
  Ενέργεια που παρακολουθεί την εφαρμογή για αλλαγές στον ενεργό χρήστη (σχεδιαστικό μόρφημα Observer) και θέτει τον εαυτό της ενεργό ή ανενεργό, ανάλογα με τα δικαιώματα του εκάστοτε ενεργού χρήστη. Η κλάση αυτή πρέπει να επεκταθεί ώστε να καθορίσουμε ποιοι χρήστες είναι εξουσιοδοτημένοι για την εν λόγω ενέργεια.
- SelectionSecurityListeningAction:
  Ενέργεια που παρακολουθεί, όπως και η παραπάνω, τις αλλαγές σε ενεργό χρήστη αλλά και τις αλλαγές στην επιλογή σειρών κάποιου πίνακα και θέτει τον εαυτό της ενεργό /ανενεργό. Χρησιμοποιείται από τις ενέργειες του module Beam Files (π.χ. η ενέργεια compare πρέπει να είναι ενεργή μόνο όταν ακριβώς δυο στοιχεία του πίνακα με τα αρχεία της δέσμης είναι επιλεγμένα από το χρήστη αλλά και ταυτόχρονα ο ενεργός χρήστης να έχει τα απαραίτητα δικαιώματα για την ενέργεια αυτή).

- **Helpers:**

Να τονίσουμε εδώ ότι σε αυτήν την κατηγορία χρησιμοποιούμε πολύ συχνά ζευγάρια interface-υλοποίησης. Η υλοποίηση επιλέγεται από την διαμόρφωση του Framework, που γίνεται στο υποσύστημα Launcher. Η επιλογή μας αυτή έχει πολλαπλούς σκοπούς: Μπορούμε να αλλάξουμε εύκολα την υλοποίηση που έχουμε προεπιλέξει για κάποια σημεία της εφαρμογής μας, να τεστάρουμε εύκολα την εφαρμογή μας (ειδικά σε αυτό το σημείο θα αναφερθούμε στο κεφάλαιο του Ελέγχου) και τέλος, να διαχωρίσουμε CESAR-specific στοιχεία από την αφηρημένη εφαρμογή μας (τα στοιχεία αυτά θα εμφανίζονται στις υλοποιήσεις των interfaces αλλά αυτό δεν μας πειράζει επειδή αφ’ ενός ο κώδικας της εφαρμογής θα χρησιμοποιεί μόνο τα interfaces και αφ’ ετέρου η επιλογή των αντίστοιχων υλοποιήσεων θα γίνεται στο ξεχωριστό υποσύστημα Launcher κατά την διαμόρφωση της εφαρμογής).

Κατ’ αρχάς, έχουμε κάποιες γενικού σκοπού κλάσεις:

- ReflectiveActionHandler:
  Είναι ένας χειριστής ενεργειών που υποστηρίζει αντανάκλαση (reflection). Αυτό σημαίνει ότι μπορούμε να ορίσουμε τις μεθόδους, που χειρίζονται διάφορες ενέργειες. Έτσι, όταν επιλεγεί μία ενέργεια, τότε θα εκτελεστεί μια συγκεκριμένη μέθοδος (action handling method). Χρησιμοποιείται σαν εργαλείο στις ενέργειες των modules Status και Beam Files.
- MessageHandler και MessageHandlerImpl:
  Είναι μια interface και η υλοποίησή της και διαχειρίζονται μηνύματα που θέλουμε να εμφανίσουμε στο χρήστη. Με τις μεθόδους showError(), showWarning() και showInformation(), εμφανίζουμε έναν μήνυμα που πληροφορεί το χρήστη για σφάλμα, προειδοποίηση ή γενική πληροφορία, αντίστοιχα. Στην υλοποίηση, χρησιμοποιούμε pop-up διάλογους για την εμφάνιση αυτών των μηνυμάτων.
- DialogManager και DialogManagerImpl:
  Interface και υλοποίηση που διαχειρίζονται τους διάλογους (dialogs) με το χρήστη εμφανίζοντας κάποια panel με πεδία που πρέπει να συμπληρωθούν και τις επιλογές “Ok” και “Cancel”.
- ActionConfigurer και ActionConfigurerImpl:
  Interface και υλοποίηση που κατασκευάζουν το menu και το toolbar της εφαρμογής και τα θέτουν στο κεντρικό παράθυρο της εφαρμογής μας.
- SecurityListener:
  Ένα interface το οποίο κάνει τις κλάσεις που το υλοποιούν να «παρακολουθούν» για αλλαγές στον ενεργό χρήστη της εφαρμογής (σχεδιαστικό μόρφημα Observer). Όταν δηλαδή, γίνεται μια τέτοια αλλαγή καλείται η μέθοδος της παρούσας διαπροσωπίας (interface) activeUserChanged(), όπου η κλάση που την υλοποιεί κάνει όποιες ενέργειες απαιτούνται. Για παράδειγμα, το interface αυτό υλοποιούν πολλές ενέργειες, οι οποίες ενεργοποιούν/ απενεργοποιούν τον εαυτό τους, σύμφωνα με τα δικαιώματα του εκάστοτε ενεργού χρήστη.
- SecurityService και SecurityServiceImpl:
  Παρέχουν ένα σύνολο από μεθόδους που έχουν σχέση με την ασφάλεια της εφαρμογής. Για παράδειγμα, υπάρχουν οι μέθοδοι login, autologin και logout που κάνουν τις ομώνυμες αλλαγές στο σύστημα του CESAR και που χρησιμοποιούνται από τις αντίστοιχες ενέργειες του Framework που αναφέραμε (στο SecurityServiceImpl γίνεται η αναφορά στο σύστημα του CESAR). Το ενδιαφέρον είναι ότι η κλάση αυτή προσθέτει νέες λειτουργίες και τροποποιεί υπάρχουσες σε μια κλάση με το όνομα ClientSecurityService, η οποία υπήρχε σε προηγούμενες εκδόσεις

του CESAR. Για να το πετύχουμε αυτό χρησιμοποιήσαμε το σχεδιαστικό μόρφημα Decorator (Wrapper) στην κλάση μας (SecurityServiceImpl).

- WorkspacesSecurityManager και WorkspacesSecurityManagerImpl:
  Έχουν μία μέθοδο που επιστρέφει τα ονόματα των workspaces στα οποία έχει πρόσβαση ο ενεργός χρήστης. Η υλοποίηση έχει και άλλες λειτουργίες, όπως το να παρακολουθεί για αλλαγές στον ενεργό χρήστη και να προσθέτει ή να αφαιρεί κατάλληλα workspaces σε/ από την εφαρμογή, ανάλογα με τα δικαιώματα του χρήστη.
- TableExplorer:
  Η υλοποίηση ενός πίνακα, με προηγμένες δυνατότητες, όπως η ταξινόμηση των σειρών σύμφωνα με την τιμή κάποιας στήλης, η υποστήριξη highlighters και renderers, η δυνατότητα εκτύπωσης (print) και εύρεσης κάποιας λέξης (search) κ.α.
- PrintingDialog:
  Ο διάλογος (dialog) που εμφανίζεται όταν επιλεγεί η ενέργεια Εκτύπωση του παραπάνω πίνακα. Βρίσκεται εδώ γιατί η εκτύπωση είναι άρρηκτα συνδεδεμένη με τον πίνακα (π.χ. με το συνδυασμό πλήκτρων Ctrl-P). Δηλαδή, ο πίνακας έχει από μόνος του υποστήριξη εκτύπωσής του.
- StatusHeaderPanel:
  Μια μπάρα που χρησιμοποιείται τόσο στα Beam Files όσο και στα Status Panels (στο πάνω μέρος τους) και περιέχουν γενικές πληροφορίες για την δέσμη στην οποία αναφέρονται τα αντίστοιχα panels, όπως το όνομα της δέσμης, το αρχείο που είναι φορτωμένο σε αυτήν, την ενέργεια της δέσμης, την τρέχουσα ημερομηνία και ώρα κλπ.
- StatusHeaderCache:
  Είναι βοηθητική προς την παραπάνω κλάση και κρατάει και ανανεώνει κατάλληλα τις πληροφορίες της δέσμης που μας ενδιαφέρουν.

Δεύτερον, έχουμε κλάσεις οι οποίες παρέχουν την δυνατότητα στο χρήστη να εισάγει κάποια δεδομένα στην εφαρμογή, δηλαδή λειτουργούν ως διάλογοι (dialogs) με το χρήστη και εξυπηρετούν το γενικότερο Graphical User Interface (GUI) της εφαρμογής μας:

- LoginControlPanel:
  Είναι ένα panel το οποίο παρέχει στον χρήστη τη δυνατότητα να εισάγει το username και το password του, ώστε να επιχειρήσει login στο σύστημα. Χρησιμοποιείται προφανώς από την ενέργεια login που έχουμε ήδη αναφέρει.

- SingleInputDialog:

  Διάλογος με ένα πεδίο, όπου περιμένουμε ο χρήστης να πληκτρολογήσει μία αριθμητική τιμή. Χρησιμοποιείται από ενέργειες του module Equipment Status.

- SingleInputDialogWithCheckBox:

  Διάλογος ίδιος με τον προηγούμενο, με την προσθήκη ενός checkbox. Χρησιμοποιείται από τις ενέργειες των Status Panels ώστε ο χρήστης να εισάγει μία τιμή για κάποια παράμετρο του εξοπλισμού (αλλαγή πραγματικής τιμής εξοπλισμού) και με τσεκαρισμένο το checkbox να ανανεώνεται και η αναφορική τιμή του εξοπλισμού. Η παρούσα κλάση είναι υποκλάση της προηγούμενης.

- ComboBoxInputDialog:

  Είναι ένας άλλος διάλογος που ζητά από τον χρήστη να του δώσει μία τιμή ανάμεσα από έναν σύνολο δυνατών επιλογών. Χρησιμοποιείται από τις κλάσεις ObstacleStatus και TaxStatus του module Status.

- ComboBoxInputDialogWithCheckBox:

  Είναι ίδιος διάλογος με τον προηγούμενο, με την διαφορά ότι έχει και ένα checkbox. Είναι υποκλάση της προηγούμενης και καλείται από τις κλάσεις ObstacleStatus και TaxStatus του module Status. Αν το checkbox είναι τσεκαρισμένο, τότε εκτός από τις πραγματικές τιμές του εξοπλισμού, αλλάζουν και οι αναφορικές.

- ComboBoxInputDialogWithTextField:

  Είναι ίδιος διάλογος με τον προηγούμενο, με την διαφορά ότι αντί για checkbox, έχει ένα πεδίο για την συμπλήρωση μιας λέξης (υποκλάση της ComboBoxInputDialog). Καλείται από την κλάση StatusBrowser του module Status, έτσι ώστε στο σύνολο των δυνατών επιλογών να έχουμε τα είδη του εξοπλισμού και στο $2^{ο}$ πεδίο να εισάγεται το όνομα της δέσμης.

- Άλλες Κλάσεις:

Άλλες κλάσεις όπως οι SplashScreen, AbstractSplashScreen και ProgressMonitor μας επιτρέπουν να έχουμε μια εισαγωγική εικόνα (splash screen) κατά την διάρκεια της φόρτωσης της εφαρμογής.

5.2.2 Διαχείριση Αρχείων Δέσμης

Πριν από οτιδήποτε άλλο, θα πρέπει να αναφέρουμε ότι σε αυτό το υποσύστημα χρησιμοποιούμε τις κλάσεις TableExplorer και ReflectiveActionHandler, που περιγράψαμε στο υποσύστημα Framework. Η πρώτη χρησιμοποιείται για την υλοποίηση όλων των

πινάκων με αρχεία δέσμης ή εξοπλισμό, ενώ η δεύτερη για να προσθέσουμε σε όλες τις ενέργειες αντανάκλαση (reflection).

- **Beam Files Browser:**

- BeamFilesBrowser:
  Ζητά από το χρήστη να δώσει το όνομα του πειράματος (π.χ. "H2B", "M2A" κλπ) που τον ενδιαφέρει και στην συνέχεια, φορτώνει το κεντρικό παράθυρο της διαχείρισης των αρχείων δέσμης (εξωτερικά από το CESAR) με τα αρχεία του πειράματος που επελέγη. Συγκεκριμένα, ανακτά τα κατάλληλα αρχεία από μια βάση δεδομένων και κατασκευάζει ένα αντικείμενο της κλάσης FilePanel.
- BeamFilesPanelAction:
  Φορτώνει οποιουδήποτε είδους παράθυρο διαχείρισης αρχείων δέσμης μέσα από το CESAR. Είναι υπερκλάση των επόμενων τριών κλάσεων.
- AllBeamFilesPanelAction:
  Φορτώνει το κεντρικό παράθυρο της διαχείρισης των αρχείων δέσμης μέσα από το CESAR με τα αρχεία του πειράματος του ενεργού χρήστη και της ενεργής δέσμης του CESAR. Το παράθυρο προστίθεται στο ενεργό workspace του CESAR.
- BeamFilesByExperimentPanelAction:
  Φορτώνει το κεντρικό παράθυρο της διαχείρισης των αρχείων δέσμης μέσα από το CESAR με τα αρχεία οποιουδήποτε πειράματος επιλέξει ο χρήστης από τα πειράματα της ενεργής δέσμης του CESAR. Το παράθυρο προστίθεται στο ενεργό workspace του CESAR.
- BeamFilesFilteredByWobblingPanelAction:
  Φορτώνει το κεντρικό παράθυρο της διαχείρισης των αρχείων δέσμης μέσα από το CESAR με τα αρχεία οποιουδήποτε πειράματος της ενεργής δέσμης του CESAR και που είναι σύμφωνα με την εκάστοτε διαμόρφωση της δέσμης αυτής. Το παράθυρο προστίθεται στο ενεργό workspace του CESAR.

- **Beam Files Panel:**

- FilePanel:
  Μπορεί να χαρακτηριστεί ως η κεντρική κλάση του υποσυστήματος. Παρέχει όλα τα εποπτικά εργαλεία για την εμφάνιση του κεντρικού παραθύρου με τα αρχεία, έτσι όπως το περιγράψαμε στο προηγούμενο κεφάλαιο. Ορισμένες βασικές μέθοδοι της κλάσης αυτής είναι οι εξής:
  1. initComponents():

Δημιουργεί τα GUI χαρακτηριστικά του παραθύρου, όπως τον τίτλο-μπάρα (περιέχει χρήσιμες πληροφορίες όπως την δέσμη, το πείραμα, την τρέχουσα ημερομηνία και ώρα κλπ) στο πάνω μέρος, τον πίνακα με τα αρχεία ακριβώς από κάτω και τα κουμπιά των ενεργειών στο κάτω μέρος του παραθύρου.

2. initActions():

Δημιουργεί τις ενέργειες, που μπορούν να εφαρμοστούν στα αρχεία.

3. Action handling methods:

Μία σειρά από μέθοδοι που χειρίζονται τις ενέργειες πάνω στα αρχεία (βλ. προηγούμενο κεφάλαιο). Ο χειρισμός ενεργειών από μεθόδους γίνεται με την βοήθεια της κλάσης ReflectveActionHandler της κατηγορίας Helpers του υποσυστήματος Framework.

4. createNodeModel():

Δημιουργεί το μοντέλο του πίνακα , που είναι ένα αντικείμενο της κλάσης BeamFileNamesJB της κατηγορίας Beans με τις στήλες του πίνακα να αντιστοιχούν στις ιδιότητες της κλάσης αυτής (περισσότερα θα αναφερθούν παρακάτω).

5. updateTable():

Ανανεώνει όλον τον πίνακα. Καλείται από την action handling method refresh(), η οποία χειρίζεται την ενέργεια Refresh (Ανανέωση).

- ViewPanel:

  Είναι ένα άλλο panel, το οποίο εμφανίζεται με την ενέργεια View (Προβολή). Οι λειτουργίες της κλάσης είναι ανάλογες με την κλάση FilePanel, με λίγες μόνο διαφορές. Για παράδειγμα, εδώ έχουμε δύο καρτέλες με δύο διαφορετικούς πίνακες (για magnets και collimators) και διαφορετικές ενέργειες. Άλλη διαφορά είναι ότι εδώ χρησιμοποιούμε διαφορετική μπάρα-τίτλο, που είναι αντικείμενο της κλάσης BeamFileHeaderPanel της κατηγορίας Helpers. Επίσης, αξίζει να αναφέρουμε ότι το μοντέλο για τον πίνακα με τους magnets είναι η κλάση MagnetRefsBeanImpl ενώ για τον πίνακα με τους collimators η CollimRefsBeanImpl (και οι 2 στην κατηγορία Beans).

- ComparePanel:

  Είναι panel που έχει ως περιεχόμενο τις διαφορές που παρουσιάζουν οι τιμές των magnets αλλά και των collimators μεταξύ 2 αρχείων. Μοιάζει αρκετά με την προηγούμενη κλάση (2 καρτέλες). Οι μόνες διαφορές είναι ότι οι ενέργειες και για τις δύο καρτέλες είναι οι ίδιες και τα μοντέλα για τους πίνακες των magnets και collimators είναι οι κλάσεις MagnetCompRefsBeanImpl και CollimCompRefsBeanImpl, αντίστοιχα (κλάσεις κατηγορίας Beans).

- Beans:

Στην κατηγορία αυτή υπάγονται κλάσεις οι οποίες αναπαριστούν αρχεία δέσμης ή εξοπλισμό. Οι κλάσεις αυτές δεν είναι παρά τα μοντέλα στους διάφορους πίνακες, όπως ήδη αναφέραμε. Δηλαδή, κάθε αντικείμενο μιας τέτοιας κλάσης αποτελεί και ένα στοιχείο (γραμμή) σε κάποιον πίνακα. Επίσης, όλες οι κλάσεις της παρούσας κατηγορίας έχουν κάποιες ιδιότητες οι οποίες αντιστοιχούν στις στήλες των αντίστοιχων πινάκων. Το πώς υλοποιήσαμε όλη αυτήν την διαδικασία των beans θα αναφερθεί στο κεφάλαιο της Υλοποίησης.

- BeamFileNamesJB:

  Είναι το μοντέλο του πίνακα του κεντρικού παραθύρου των αρχείων και αναπαριστά αρχεία δέσμης. Οι ιδιότητές του bean αυτού είναι: Name (Όνομα αρχείου), Comment (Σχόλιο), Momentum (Ορμή) και Last Modification (Τελευταία Τροποποίηση).

- MagnetRefsBeanImpl:

  Είναι το μοντέλο του πίνακα του παραθύρου "View" του υποσυστήματος, και συγκεκριμένα της καρτέλας με τους magnets. Αυτό το bean, δηλαδή, αναπαριστά μαγνήτες ενός αρχείου. Οι ιδιότητές του είναι: Magnet (Όνομα μαγνήτη) και File Reference (Αναφορική τιμή αρχείου για το ρεύμα μαγνήτη).

- CollimRefsBeanImpl:

  Είναι το μοντέλο του πίνακα του παραθύρου "View" του υποσυστήματος, και συγκεκριμένα της καρτέλας με τους collimators. Αναπαριστά collimators ενός αρχείου και οι ιδιότητές του είναι: Collimator (Όνομα collimator), Jaw1 File Reference (Αναφορική τιμή αρχείου για την θέση Jaw1 ενός collimator) και Jaw2 File Reference (Αναφορική τιμή αρχείου για την θέση Jaw2 ενός collimator).

- MagnetCompRefsBeanImpl:

  Είναι το μοντέλο του πίνακα του παραθύρου "Compare" του υποσυστήματος, και συγκεκριμένα της καρτέλας με τους magnets. Αναπαριστά μαγνήτες της δέσμης στην οποία δουλεύουμε και τις τιμές κάποιων παραμέτρων 2 αρχείων για αυτούς. Οι ιδιότητές του είναι: Magnet (Όνομα μαγνήτη), File1 (Αναφορική τιμή $1^{ου}$ αρχείου για το ρεύμα μαγνήτη), File2 (Αναφορική τιμή $2^{ου}$ αρχείου για το ρεύμα μαγνήτη), File1-File2 (Διαφορά των 2 αναφορικών τιμών ρεύματος των 2 αρχείων) και File1/File2 (Πηλίκο των 2 αναφορικών τιμών).

- CollimCompRefsBeanImpl:

  Είναι το μοντέλο του πίνακα του παραθύρου "Compare" του υποσυστήματος, και συγκεκριμένα της καρτέλας με τους collimators. Αναπαριστά collimators της δέσμης στην οποία δουλεύουμε και τις τιμές κάποιων παραμέτρων 2 αρχείων για αυτούς. Οι ιδιότητές του είναι: Collimator (Όνομα collimator), File1 Jaw1 (Αναφορική τιμή $1^{ου}$

αρχείου για την θέση Jaw1), File1 Jaw2 (Αναφορική τιμή 1ου αρχείου για την θέση Jaw2), File2 Jaw1 (Αναφορική τιμή 2ου αρχείου για την θέση Jaw1), File2 Jaw2 (Αναφορική τιμή 2ου αρχείου για την θέση Jaw2), File1-File2 Jaw1 (Διαφορά των 2 αναφορικών τιμών θέσης Jaw1 των 2 αρχείων) και File1-File2 Jaw2 (Διαφορά των 2 αναφορικών τιμών θέσης Jaw2 των 2 αρχείων).

- **Ενέργειες:**

- ActualBeamRefsToFile:
  Ενέργεια "Save Actual BeamRefs" (Αποθήκευση πραγματικών αναφορικών τιμών).
- CopyAction:
  Ενέργεια "Copy" (Αντιγραφή).
- CompareAction:
  Ενέργεια "Compare" (Σύγκριση).
- DeleteAction:
  Ενέργεια "Delete" (Διαγραφή).
- EditAction:
  Ενέργεια "Edit" (Τροποποίηση).
- ExtrapolateAction:
  Ενέργεια "Extrapolate".
- LoadAction:
  Ενέργεια "Load Beam File" (Φόρτωση Αρχείου Δέσμης).
- RefreshAction:
  Ενέργεια "Refresh" (Ανανέωση).
- RestoreAction:
  Ενέργεια "Restore Beam File" (Επαναφορά Αρχείου Δέσμης).
- ViewAction:
  Ενέργεια "View" (Προβολή).

- **Helpers:**

- BeamFileLoadManager:
  Είναι η κλάση, που με την μέθοδό της loadBeamFile(), φορτώνει ένα αρχείο στην δέσμη. Χρησιμοποιείται από την ενέργεια "Load Beam File" (Φόρτωση Αρχείου Δέσμης).

- BeamFileManager:

  Βοηθά τις κλάσεις FilePanel, ViewPanel και ComparePanel να εκτελέσουν κάποιες ενέργειες στα αρχεία. Οι μέθοδοι που το κάνουν είναι οι εξής:

  1. deleteBeamFile():

  Διαγράφει ένα αρχείο (ενέργεια της κλάσης FilePanel).

  2. editBeamFile():

  Εξάγει πληροφορίες ενός αρχείου στο Excel προς επεξεργασία (ενέργεια της κλάσης FilePanel).

  3. editRefsCollim():

  Αλλάζει τις τιμές ενός αρχείου για τις θέσεις ενός collimator (ενέργεια της κλάσης ViewPanel).

  4. editRefsMagnets():

  Αλλάζει την τιμή ενός αρχείου για το ρεύμα ενός magnet (ενέργεια της κλάσης ViewPanel).

  5. changeComment():

  Αλλάζει το comment (σχόλιο) ενός αρχείου (ενέργεια της κλάσης ViewPanel).

  6. copyBeamFile():

  Αντιγράφει ένα αρχείο σε κάποιο νέο (ενέργεια της κλάσης FilePanel).

  7. extrapolateBeamFile():

  Εκτελεί την ενέργεια “Extrapolate” σε ένα αρχείο (ενέργεια της κλάσης FilePanel).

  8. saveBeam0():

  Εκτελεί την ενέργεια “Save Actual BeamRefs” σε ένα αρχείο (ενέργεια της κλάσης FilePanel).

- BeamfilesUtils:

  Κατασκευάζει τα στοιχεία των πινάκων.

- CopyChoicePanel:

  Είναι ένα panel που επιτρέπει στο χρήστη να δώσει χρήσιμες πληροφορίες για το νέο αρχείο που θα δημιουργηθεί, όταν εκτελείται η ενέργεια “Copy”.

- ExtrapChoicePanel:

  Είναι ένα panel που επιτρέπει στο χρήστη να δώσει χρήσιμες πληροφορίες για το νέο αρχείο που θα δημιουργηθεί, όταν εκτελείται η ενέργεια “Extrapolate”.

- ExtrapElectronDetailPanel, ExtrapHadronDetailPanel, ExtrapHeaderPanel:

  Είναι κλάσεις που χρησιμοποιούνται ως βοηθητικές για την κλάση ExtrapChoicePanel.

- RefSortOfChooser:

  Είναι μία βοηθητική κλάση για τις ενέργειες “Load Beam File” και “Save Actual

BeamRefs". Επιτρέπει στον χρήστη να διαλέξει ή μόνο magnets ή μόνο collimators ή και τους δύο τύπους εξοπλισμού. Έτσι, η αντίστοιχη ενέργεια θα αφορά μόνο τον (τους) επιλεγμένο (επιλεγμένους) τύπο (τύπους) εξοπλισμού.

- BeamFileChooser:
  Είναι ένας διάλογος, ώστε ο χρήστης να παρέχει ένα όνομα αρχείου και την CVS έκδοση αυτού (CVS version). Χρησιμοποιείται από την μέθοδο που χειρίζεται την ενέργεια "Restore Beam File" (Ανάκτηση Αρχείου Δέσμης), που ανήκει στην κλάση FilePanel.
- BeamFileHeaderPanel:
  Είναι μία μπάρα που την χρησιμοποιούμε στα παράθυρα "View" και "Compare" (στο πάνω μέρος τους) και παρέχουν χρήσιμες πληροφορίες για τα αρχεία δέσμης στα οποία αναφέρονται (όνομα, ενέργεια, περιγραφή, ημερομηνία κλπ).

Τέλος, να αναφέρουμε ότι έχουμε ξεχωριστές κλάσεις για τις ενέργειες που υπάρχουν στο κεντρικό παράθυρο (FilePanel), όπως είδαμε πιο πριν (στην παράγραφο Ενέργειες), και όλες είναι υποκλάσεις της κλάσης SelectionSecurityListeningAction του υποσυστήματος Framework. Και αυτό γιατί οι εν λόγω ενέργειες πρέπει να ενεργοποιούνται/ απενεργοποιούνται σύμφωνα και με το ποιος είναι ενεργός χρήστης του συστήματος και το πόσα είναι τα επιλεγμένα στοιχεία στον πίνακα του ίδιου παραθύρου. Για παράδειγμα, η ενέργεια "Load Beam File", που φορτώνει ένα αρχείο στην πραγματική δέσμη πρέπει να είναι ενεργοποιημένη μόνο αν ο ενεργός χρήστης είναι Super User ή Beam Main User, και ταυτόχρονα τα επιλεγμένα αρχεία στον πίνακα είναι ακριβώς ένα.

5.2.3 Κατάσταση Εξοπλισμού

Πριν προχωρήσουμε, θα πρέπει να αναφέρουμε ότι και σε αυτό το υποσύστημα χρησιμοποιούμε τις γνωστές μας κλάσεις TableExplorer και ReflectiveActionHandler του Framework.

- Status Browser:

- StatusBrowser:
  Ζητά από το χρήστη να δώσει ένα είδος εξοπλισμού και το όνομα της δέσμης που τον ενδιαφέρει και στην συνέχεια, ανακτά το ζητούμενο εξοπλισμό και φορτώνει (εκτός του προγράμματος CESAR) το κατάλληλο panel (παράθυρο) κατάστασης εξοπλισμού, δηλαδή κατασκευάζει αντικείμενο κάποιας από τις κλάσεις της κατηγορίας Equipment.

- StatusPanelAction:

  Φορτώνει οποιοδήποτε είδος παραθύρου κατάστασης εξοπλισμού μέσα από το CESAR. Είναι υπερκλάση των επόμενων κλάσεων της παρούσας κατηγορίας κλάσεων (Status Browser).

- MagnetStatusPanelAction:

  Φορτώνει το κεντρικό παράθυρο της κατάστασης εξοπλισμού μέσα από το CESAR με τον εξοπλισμό τύπου "Magnets" (μαγνήτες) της ενεργής δέσμης του CESAR. Το παράθυρο προστίθεται στο ενεργό workspace.

- CollimatorStatusPanelAction:

  Ομοίως με την προηγούμενη κλάση αλλά για είδος εξοπλισμού "Collimators".

- ……:

  Και άλλες τέτοιες κλάσεις (όσες και τα είδη εξοπλισμού, δηλαδή άλλες 17).

- **Equipment:**

Η κατηγορία αυτή περιέχει τις κλάσεις των panel κατάστασης κάθε είδους εξοπλισμού. Επιπλέον, υπάρχει και μία αφηρημένη υπερκλάση όλων αυτών που υλοποιεί κοινές λειτουργίες των status panels. Χρησιμοποιήσαμε, δηλαδή, το σχεδιαστικό μόρφημα Chain of Responsibility, για να ομαδοποιήσουμε τις κοινές λειτουργίες του εξοπλισμού. Όλες οι υποκλάσεις αυτής της κατηγορίας δέχονται από κάποια κλάση της κατηγορίας Status Browser, το σύνολο του εξοπλισμού κάποιου είδους και το όνομα της δέσμης (διότι δεν μπορούμε να μάθουμε από τον ίδιο τον εξοπλισμό σε ποια δέσμη βρισκόμαστε).

-Υπερκλάση:

- Status:

  Κατάσταση εξοπλισμού όλων των ειδών. Είναι η υπερκλάση των επόμενων κλάσεων.

-Υποκλάσεις:

- AccessStatus :

  Κατάσταση εξοπλισμού τύπου Access Doors.

- AnalogWireChamberStatus:

  Κατάσταση εξοπλισμού τύπου Analog Wire Chambers.

- CollimatorStatus:

  Κατάσταση εξοπλισμού τύπου Collimators.

- DelayWireChamberStatus:

  Κατάσταση εξοπλισμού τύπου Delay Wire Chambers.

- FiscStatus:
  Κατάσταση εξοπλισμού τύπου Fiscs.
- MagnetStatus:
  Κατάσταση εξοπλισμού τύπου Magnets.
- ObstacleStatus:
  Κατάσταση εξοπλισμού τύπου Obstacles και InOutPositionables.
- RectifierStatus:
  Κατάσταση εξοπλισμού τύπου Magnets (παρουσιάζει άλλες πληροφορίες από την κλάση MagnetStatus).
- ScalerStatus:
  Κατάσταση εξοπλισμού τύπου Scalers.
- ScrapersStatus:
  Κατάσταση εξοπλισμού τύπου Scrapers.
- SemStatus:
  Κατάσταση εξοπλισμού τύπου Sems.
- SetCollimatorStatus:
  Κατάσταση εξοπλισμού τύπου Collimators (εκτελεί συγκεκριμένη ενέργεια ως προεπιλογή).
- SetMagnetStatus:
  Κατάσταση εξοπλισμού τύπου Magnets (εκτελεί συγκεκριμένη ενέργεια ως προεπιλογή).
- TaxStatus:
  Κατάσταση εξοπλισμού τύπου Taxes.
- TdvStatus:
  Κατάσταση εξοπλισμού τύπου TDVs.
- TdxStatus:
  Κατάσταση εξοπλισμού τύπου TDXs.
- ThresholdStatus:
  Κατάσταση εξοπλισμού τύπου Thresholds.
- TriggerStatus:
  Κατάσταση εξοπλισμού τύπου Scintillators.
- VacuumStatus:
  Κατάσταση εξοπλισμού τύπου Pumps.

Επειδή οι λειτουργίες του επιμέρους εξοπλισμού ακολουθεί την ίδια λογική, θα περιγράψουμε τη λειτουργία μόνο της υπερκλάσης Status και μιας τυχαίας υποκλάσης XxxStatus:

- Status:

  1. initComponents():

  Δημιουργεί και εφαρμόζει κοινά GUI χαρακτηριστικά των επιμέρους status panels, όπως μία μπάρα που τοποθετείται στην κορυφή του panel και λειτουργεί ως τίτλος (περιέχει χρήσιμες πληροφορίες όπως την δέσμη, το πείραμα, την τρέχουσα ημερομηνία και ώρα και το όνομα και χρήσιμες πληροφορίες για το αρχείο που είναι κάθε στιγμή φορτωμένο στη δέσμη). Επίσης, η μέθοδος αυτή προσθέτει σε ένα σημείο στο κάτω μέρος του panel μία επιλογή εκτύπωσης του πίνακα του panel με τον εξοπλισμό.

  2. initActions():

  Δημιουργεί 2 ενέργειες που παρατηρούνται σε όλων των ειδών τα status panels και ονομάζονται “Get Additional Info” και “Get Logbook Info”.

  3. showAdditionalInfo():

  Χειρίζεται την εκτέλεση από το χρήστη της ενέργειας “Get Additional Info” (μέθοδος που χειρίζεται ενέργεια - action handling method). Ο χειρισμός των ενεργειών γίνεται με τον ReflectiveActionHandler του υποσυστήματος Framework, όπως έχουμε ήδη αναφέρει.

  4.showLogbooklInfo():

  Χειρίζεται την εκτέλεση από το χρήστη της ενέργειας “Get Logbook Info” (μέθοδος που χειρίζεται ενέργεια - action handling method). .

  5. subscribe():

  Εγγράφει όλον τον εξοπλισμό του panel στο σύστημα παρακολούθησης πραγματικού χρόνου (Monitoring), ώστε οι τιμές και η κατάσταση του εξοπλισμού να ανανεώνονται συνεχώς. Καλείται όταν πατηθεί ένα κουμπί με την ετικέτα “Run” (θα εξηγήσουμε από πού προέρχεται αυτό παρακάτω).

  6. unsubscribe():

  Κάνει το αντίθετο από την προηγούμενη μέθοδο. Καλείται όταν πατηθεί ένα κουμπί με την ετικέτα “Hold” (θα εξηγήσουμε από πού προέρχεται αυτό παρακάτω).

  6. subscribeSelectedNode():

  Εγγράφει το/α επιλεγμένο/α μηχάνημα/ατα στο σύστημα παρακολούθησης αν είμαστε σε κατάσταση “Run”. Καλούμε αυτή τη μέθοδο όταν ανανεώνουμε επιλεγμένο/α μηχάνημα/ατα (ενέργεια “Refresh”με την επιλογή “Refresh Selected ”).

7. launchQuickPanel():

Εμφανίζει παράθυρο που έχει διάφορες πληροφορίες για εξοπλισμό τύπου Scaler ή Scintillator. Συνεπώς, καλείται μόνο από τις κλάσεις ScalerStatus καιTriggerStatus, μέσω της ειδικής ενέργειας "Quick", την οποία δεν θα αναλύσουμε.

8. setHighlighters():

Δημιουργεί και εφαρμόζει τους highlighters, καθένας από τους οποίους είναι υπεύθυνος να απεικονίσει την κατάσταση κάθε μηχανήματος στον αντίστοιχο πίνακα (αλλαγή χρώματος γραμμάτων ή φόντου).

- XxxStatus:

1. initComponents():

Δημιουργεί και προσθέτει στο panel κατάστασης τον πίνακα του εξοπλισμού και τα κουμπιά για τις ενέργειες.

2. initActions():

Δημιουργεί τις ενέργειες, που μπορούν να εφαρμοστούν στον συγκεκριμένο τύπο εξοπλισμού. Οι ενέργειες που αλλάζουν τις παραμέτρους του πραγματικού εξοπλισμού είναι υποκλάσεις της κλάσης PrivilegedAction, που περιγράψαμε στο Framework.

3. Action handling methods:

Μία σειρά από μέθοδοι που χειρίζονται τις ενέργειες πάνω στον εξοπλισμό.

4. refresh():

Ανανεώνει όλον (αν είναι επιλεγμένο το κουμπί "Refresh All") ή μέρος (αν είναι επιλεγμένο το κουμπί "Refresh Selected") του πίνακα.

5. createNodeModel():

Δημιουργεί το μοντέλο του πίνακα (αντίστοιχη κλάση κατηγορίας Bean με στήλες που αντιστοιχούν σε ιδιότητες της κλάσης αυτής).

6. updateTable():

Ανανεώνει όλον τον πίνακα. Καλείται από την μέθοδο refresh(), εφόσον είμαστε σε κατάσταση "Refresh All".

7. updateSelectedNode():

Ανανεώνει το/α επιλεγμένο/α μηχάνημα/ατα. Καλείται από την μέθοδο refresh(), εφόσον είμαστε σε κατάσταση "Refresh Selected".

8. getDataFromXxx();

Δημιουργεί τα beans του πίνακα. Επιστρέφει, δηλαδή, ένα πίνακα με στοιχεία κάποια αντικείμενα της κλάσης της κατηγορίας Bean, η οποία αντιστοιχεί στο παρόν είδος εξοπλισμού.

9. getTable():

Επιστρέφει τον πίνακα. Χρησιμοποιείται από την υπερκλάση Status.

10. getTableModel():

Επιστρέφει το μοντέλο του πίνακα. Χρησιμοποιείται από την υπερκλάση Status.

11. getDataBeans():

Επιστρέφει τα beans του πίνακα. Χρησιμοποιείται από την υπερκλάση Status.

- **Beans:**

Το υποσύστημα αυτό περιέχει τις Bean κλάσεις κάθε είδους εξοπλισμού, σε ευθεία αναλογία με τα όσα είπαμε στο υποσύστημα Beam Files. Επιπλέον, οι εν λόγω κλάσεις συνδέονται σε ένα σύστημα λήψης γεγονότων από τον πραγματικό εξοπλισμό, ώστε να ανανεώνουν σε πραγματικό χρόνο τις παραμέτρους και την κατάσταση του εξοπλισμού.

Τις κλάσεις αυτής της κατηγορίας τις οργανώνουμε σε δύο επίπεδα, με μία κλάση η οποία είναι υπερκλάση όλων των υπόλοιπων. Η αφηρημένη αυτή υπερκλάση υλοποιεί κοινές λειτουργίες και χαρακτηριστικά όλων των τύπων εξοπλισμού.

Όλες οι υποκλάσεις αυτής της κατηγορίας δέχονται από την αντίστοιχη κλάση της κατηγορίας Equipment, ένα μηχάνημα συγκεκριμένου είδους, το σύνολο των ενεργειών (ώστε να εμφανίζονται με δεξί κλικ) και την default ενέργεια (ώστε να εκτελείται με διπλό κλικ) για αυτό το μηχάνημα. Περισσότερες πληροφορίες για τη λειτουργία αυτών των κλάσεων θα αναφέρουμε στο κεφάλαιο της Υλοποίησης.

-Υπερκλάση:

- StatusBeanImpl:

  Bean της κατάστασης εξοπλισμού όλων των ειδών. Είναι η υπερκλάση όλων των επόμενων κλάσεων.

-Υποκλάσεις:

- AccessBeanImpl:

  Bean εξοπλισμού τύπου Access Doors.

- AnalogWireChamberBeanImpl:

  Bean εξοπλισμού τύπου Analog Wire Chambers.

- CollimatorBeanImpl:

  Bean εξοπλισμού τύπου Collimators.

- DelayWireChamberBeanImpl:

  Bean εξοπλισμού τύπου Delay Wire Chambers.

- FiscBeanImpl:

  Bean εξοπλισμού τύπου Fiscs.

- InOutPositionableBeanImpl:
  Bean εξοπλισμού τύπου InOutPositionables (υπάρχει στο ObstacleStatus), TDVs και TDXs.
- MagnetBeanImpl:
  Bean εξοπλισμού τύπου Magnets (για την κλάση MagnetStatus).
- MagnetRectifierBeanImpl:
  Bean εξοπλισμού τύπου Magnets (για την κλάση RectifierStatus).
- ObstacleBeanImpl:
  Bean εξοπλισμού τύπου Obstacles.
- ObstaclePositionableBeanImpl:
  Bean εξοπλισμού τύπου Obstacles ή InOutPositionables.
- ScalerBeanImpl:
  Bean εξοπλισμού τύπου Scalers.
- ScraperBeanImpl:
  Bean εξοπλισμού τύπου Scrapers.
- SemBeanImpl:
  Bean εξοπλισμού τύπου Sems.
- TaxBeanImpl:
  Bean εξοπλισμού τύπου Taxes.
- TdvBeanImpl:
  Bean εξοπλισμού τύπου TDVs.
- TdxStatus:
  Bean εξοπλισμού τύπου TDXs.
- ThresholdStatus:
  Bean εξοπλισμού τύπου Thresholds.
- TriggerStatus:
  Bean εξοπλισμού τύπου Scintillators.
- VacuumStatus:
  Bean εξοπλισμού τύπου Pumps.

Επειδή οι λειτουργίες του επιμέρους εξοπλισμού ακολουθεί την ίδια λογική, θα περιγράψουμε τη λειτουργία μόνο της υπερκλάσης StatusBeanImpl και μιας τυχαίας υποκλάσης XxxBeanImpl:

- StatusBeanImpl:
  1. Ιδιότητες κλάσης Bean (Bean Properties):

Είναι οι παράμετροι του εξοπλισμού, όπως η τιμή του ρεύματος ενός μαγνήτη (αναφορική ή πραγματική τιμή), η θέση ενός Tax κλπ. Με άλλα λόγια, είναι απλά οι τιμές για τις στήλες του πίνακα με τον εξοπλισμό καθώς και οι τιμές που καθορίζουν την κατάσταση του εξοπλισμού. Αφού βρισκόμαστε στην υπερκλάση του υποσυστήματος Bean, οι ιδιότητες αυτές πρέπει να είναι κοινές για όλα τα είδη εξοπλισμού. Αυτές οι κοινές ιδιότητες είναι το comment (όλα τα panels έχουν στήλη με το όνομα "Comment" («Σχόλιο»), η οποία παρέχει χρήσιμες πληροφορίες για τον εξοπλισμό που ανανεώνουμε σε πραγματικό χρόνο), το busy (μας πληροφορεί αν το μηχάνημα είναι απασχολημένο) και το state (μας πληροφορεί αν η κατάσταση του μηχανήματος είναι OK, WARNING, ERROR, NOT UPDATED ή PROPERTY CHANGE).

2. Μέθοδοι για την λήψη των ιδιοτήτων:

Είναι διάφορες μέθοδοι λήψης (getter methods), που παρέχουν πρόσβαση σε ιδιότητες της κλάσης (τις ιδιότητες μιας κλάσης Bean και την χρησιμότητά αυτών αλλά και των μεθόδων πρόσβασης θα την αναλύσουμε στο επόμενο κεφάλαιο). Έχουν τα ονόματα getComment(), isBusy() και getState().

3. Μέθοδοι για ορισμό των ιδιοτήτων:

Είναι διάφορες μέθοδοι ορισμού (setter methods), που παρέχουν και πάλι πρόσβαση σε αυτές τις ιδιότητες. Καλούνται από την ίδια την κλάση αλλά και τις υποκλάσεις της κάθε φορά που θέλουμε να αλλάξουμε αυτές τις ιδιότητες. Π.χ. όταν λαμβάνουμε γνώση ότι ο εξοπλισμός απασχολείται από κάποιον χρήστη, θέτουμε την ιδιότητα busy σε αληθή τιμή (true) ή όταν λαμβάνουμε κάποιο σφάλμα, θέτουμε την ιδιότητα state στην τιμή κατάστασης ERROR και μπορούμε να προσθέσουμε το μήνυμα σφάλματος στην ιδιότητα comment.

4. getEqNames():

Χρήσιμη μέθοδος που δίνει το μηχάνημα που αναπαρίσταται από το παρόν Bean. Καλείται από την αντίστοιχη κλάση κατηγορίας Equipment, ώστε να εκτελεστούν ενέργειες στο μηχάνημα.

5. getBeam():

Χρήσιμη μέθοδος που καλείται από ορισμένες υποκλάσεις και επιστρέφει το όνομα της δέσμης στην οποία εργαζόμαστε.

6. getAdditionalInfo():

Μέθοδος που κατασκευάζει χρήσιμες πληροφορίες για το εν λόγω μηχάνημα. Καλείται από την ενέργεια "Get Additional Info" που είναι παρούσα σε όλα τα status panels.

7. getIsBeamRefAtFileRefString():

Επιστρέφει την τιμή μιας ιδιότητας που ονομάζουμε "f ", η οποία υπάρχει σε magnets

και collimators και την ορίζουμε με συγκεκριμένο τρόπο.

8. checkNullEq():

Ελέγχει αν οι παράμετροι του μηχανήματος έχουν αρχικοποιηθεί. Αν όχι, παρουσιάζεται μήνυμα σφάλματος.

9. isPropertyEventOk():

Ελέγχεται αν το γεγονός αλλαγής (property change event) είναι κάποια εξαίρεση (exception) ή όχι. Αν είναι, παρουσιάζουμε μήνυμα σφάλματος, το οποίο θέτουμε και στην ιδιότητα comment, ώστε να φαίνεται στον πίνακα με τον εξοπλισμό.

10. slowPropertyEventError():

Ελέγχει την λήψη από τον πραγματικό εξοπλισμό κάποιου γεγονότος σφάλματος, οπότε και θέτει την κατάσταση σε ERROR και τοποθετεί και πάλι, το μήνυμα σφάλματος στην ιδιότητα comment.

11. updateStatus():

Ανακτά την κατάσταση του μηχανήματος και αν υπάρξει πρόβλημα ανάκτησης, θέτει την κατάσταση σε NOT UPDATED.

12. handleStartEndEvents():

Αν φτάσει γεγονός έναρξης διαδικασίας στο μηχάνημα, θέτει την ιδιότητα busy σε αληθή τιμή, ενώ αν το γεγονός είναι λήξης, θέτει το busy σε ψευδή τιμή.

13. Μέθοδοι firePropertyChange():

Πυροδοτούν αλλαγή της κλάσης Bean όταν μια ιδιότητα αλλάξει. Έτσι, ο πίνακας ενημερώνεται ότι το μοντέλο του (αντικείμενο της κλάσης Bean) άλλαξε, ώστε να ανανεωθεί.

14. Listeners που είναι κοινοί για διάφορα είδη εξοπλισμού:

Αυτοί οι Listeners παρακολουθούν για αφίξεις (ασύγχρονα) γεγονότων από τον πραγματικό εξοπλισμό (σχεδιαστικό μόρφημα Observer) και αλλάζουν ιδιότητες είτε της ίδιας της κλάσης (όπως την ιδιότητα busy), είτε μιας υποκλάσης (όπως την τιμή του ρεύματος για τους magnets). Τα ονόματα αυτών των PropertyChangeListeners είναι StatusPCL (δέχεται γεγονότα τύπου κατάστασης του μηχανήματος και ανανεώνει όλες τις ιδιότητές του – χρησιμοποιείται σε πολλές υποκλάσεις), BusyPCL (δέχεται γεγονότα απασχόλησης του μηχανήματος και θέτει το busy στην κατάλληλη τιμή– χρησιμοποιείται σε πολλές υποκλάσεις), CountPCL (δέχεται γεγονότα τύπου Count και αλλάζει την ιδιότητα count των κλάσεων που έχουν τέτοια, δηλαδή των FiscBeanImpl, ScalerBeanImpl, SemBeanImpl και TriggerBeanImpl), InOutPosPCL (αλλάζει την ιδιότητα inOutPosition για όσες υποκλασεις την έχουν), InOutPosRefPCL (ομοίως για την ιδιότητα inOutPositionRef) και CurrentPCL (ομοίως για την ιδιότητα current).

- XxxBeanImpl:

  1. Ιδιότητες κλάσης Bean (Bean Properties):
  Ιδιότητες του συγκεκριμένου είδους εξοπλισμού.
  2. Μέθοδοι για την λήψη των ιδιοτήτων:
  Επιστρέφουν τις τιμές για τις ιδιότητες. Η χρησιμότητα αυτών θα αναφερθεί στο επόμενο κεφάλαιο.
  3. getEquipment():
  Επιστρέφει το μηχάνημα του bean. Καλείται από την αντίστοιχη κλάση κατηγορίας Equipment, ώστε να εκτελεστούν ενέργειες στο μηχάνημα.
  4. updateBeanProperties():
  Δέχεται την κατάσταση του μηχανήματος και επιστρέφει τις τιμές για τις διάφορες ιδιότητες του bean. Καλείται από ορισμένους Listeners.
  5. setPropertyToValue():
  Δέχεται ποια ιδιότητα θα αλλαχθεί και τη νέα τιμή της και την αλλάζει. Καλείται από τους Listeners της υπερκλάσης.
  6. updateImpl():
  Καλείται από την μέθοδο update της υπερκλάσης (η updateImpl είναι abstract μέθοδος της υπερκλάσης) και εκτελεί ανανεωτικές ενέργειες που είναι ειδικές για το κάθε είδος του εξοπλισμού.
  7. subscribe() και unsubscribe():
  Προσθέτει ή αφαιρεί, αντίστοιχα, το μηχάνημα σε ή από έναν ή περισσότερους Listeners.
  8. Και άλλοι (ειδικοί για τον εξοπλισμό) PropertyChangeListeners.
  9. Και άλλες (ειδικές για τον εξοπλισμό) μέθοδοι.

- **Control:**

Οι κλάσεις αυτής της κατηγορίας δέχονται ένα μηχάνημα και εκτελούν σε αυτό διάφορες ενέργειες. Οι ενέργειες αυτές έχουν το χαρακτηριστικό ότι είναι αρκετά κοινές στο υποσύστημά μας. Γίνεται, λοιπόν, φανερό ότι οι κλάσεις της παρούσας κατηγορίας χρησιμοποιούνται από μερικές μόνο κλάσεις της κατηγορίας Equipment για την εκτέλεση ορισμένων ενεργειών. Οι κλάσεις της κατηγορίας Control είναι οι εξής:

- DiscretePositionableControl:

  Χρησιμοποιείται από τις κλάσεις ObstacleStatus και TaxStatus και εκτελεί τις ενέργειες move() και setReference() (μέθοδοι της κλάσης αυτής) σε μηχάνημα τύπου

Obstacle ή Tax (εμπίπτουν και τα 2 στην κατηγορία εξοπλισμού DiscretePositionable).

- DiscretePositionableReferenceControl:

  Χρησιμοποιείται από τις κλάσεις ObstacleStatus και TaxStatus και εκτελεί την ενέργεια setReference() σε μηχάνημα τύπου Obstacle ή Tax.

- MovableControl:

  Χρησιμοποιείται από την κλάση FiscStatus και εκτελεί σε μηχάνημα τύπου Fisc (εμπίπτει στην κατηγορία του εξοπλισμού τύπου Movable) μία από τις ενέργειες: setToZero(), setToMin(), setToMax() ή setValue().

- PositionableControl:

  Χρησιμοποιείται από την κλάση TaxStatus και εκτελεί σε μηχάνημα τύπου Tax (εμπίπτει στην κατηγορία εξοπλισμού τύπου Positionable) την ενέργεια setBeamReferenceValue().

- TaxRangeControl:

  Χρησιμοποιείται από την κλάση TaxStatus και εκτελεί σε μηχάνημα τύπου Tax την ενέργεια setSelectedRange().

- VoltRestorableControl:

  Χρησιμοποιείται από τις κλάσεις AnalogWireChamberStatus, ThresholdStatus και TriggerStatus και εκτελεί τις ενέργειες restoreHV() και zeroHV() (μέθοδοι της κλάσης αυτής) στα αντίστοιχα μηχανήματα (εμπίπτουν στην κατηγορία εξοπλισμού BasicVoltControlable).

- Helpers:

Κατ' αρχάς, έχουμε βοηθητικές κλάσεις, οι οποίες χρησιμοποιούνται από όλες ή μερικές από τις κλάσεις της κατηγορίας Equipment για την εισαγωγή δεδομένων, μέσω κάποιου διαλόγου (dialog) ώστε να εφαρμοστούν στον εξοπλισμό (ενέργειες στον εξοπλισμό), εφόσον πατηθεί το OK του διαλόγου. Εκτός από τους διάλογους αυτούς, χρησιμοποιούμε και άλλους από το υποσύστημα Framework. Οι διάλογοι της παρούσας κατηγορίας είναι οι εξής:

- CollimSetJawPosDialog:

  Είναι ένας διάλογος που ζητά από το χρήστη τιμές για τις ιδιότητες jaw1 και jaw2 ενός Collimator. Επίσης, υπάρχει και ένα κουτί (checkbox) που αν είναι τσεκαρισμένο, τότε, όχι μόνο οι πραγματικές τιμές αλλά και οι αναφορικές τιμές του εξοπλισμού θα αλλάξουν. Είναι προφανές, ότι ο διάλογος αυτός χρησιμοποιείται μόνο από την κλάση CollimatorStatus.

- ScraperSetJawPosDialog:

  Είναι ένας διάλογος που ζητά από το χρήστη τιμές για τις ιδιότητες jaw1 και jaw2 ενός Scraper. Επίσης, υπάρχει και ένα κουτί (checkbox) που αν είναι τσεκαρισμένο, τότε, όχι μόνο οι πραγματικές τιμές αλλά και οι αναφορικές τιμές του εξοπλισμού θα αλλάξουν. Η κλάση αυτή χρησιμοποιείται μόνο από την κλάση ScraperStatus.

Ακόμα, έχουμε panels, που είναι εσωτερικά στα Status Panels και περιέχουν επιλογές «ανανεωσιμότητας» των τιμών των πινάκων με τον εξοπλισμό. Κάθε Status Panel μπορεί να έχει ένα από τα παρακάτω panels «ανανεωσιμότητας»:

- RefreshPanel:

  Είναι ένα panel με ένα κουμπί με την ετικέτα “Refresh” και τις επιλογές “Refresh All” και “Refresh Selected”. Η μέθοδος που χειρίζεται την ενέργεια “Refresh” βρίσκεται στις κλάσεις της κατηγορίας Equipment και ονομάζεται refresh(), όπως ήδη αναφέραμε. Αντικείμενο αυτής της κλάσης ή κάποιας υποκλάσης αυτής δημιουργεί κάθε status panel.

- RefreshRunHoldPanel:

  Είναι υποκλάση της παραπάνω και προσθέτει σε αυτήν τις επιλογές “Run” και “Hold”, που όταν πατηθούν καλούν τις μεθόδους subscribe() και unsubscribe() της κλάσης Status, αντίστοιχα.

- RunHoldPanel:

  Είναι υποκλάση της RefreshPanel και προσθέτει σε αυτήν τις επιλογές “Run” και “Hold”. Δεν χρησιμοποιείται από κάποια κλάση της κατηγορίας Equipment.

- RefreshOnEventPanel:

  Είναι υποκλάση της παραπάνω και εφαρμόζει την ενέργεια “Refresh” με την άφιξη συγκεκριμένου γεγονότος.

- RefreshOnEventRunHoldPanel:

  Είναι υποκλάση της RefreshRunHoldPanel και εφαρμόζει την ενέργεια “Refresh” με την άφιξη συγκεκριμένου γεγονότος.

Τέλος, έχουμε τους χρωματικούς highlighters που εφαρμόζονται στους πίνακες με τον εξοπλισμό ανάλογα με την κατάστασή του:

- BusyHighlighter
- ErrorHighlighter
- NotUpadatedHighlighter
- PropertyChangedHighlighter
- WarningHighlighter

Καθένας από αυτούς χαρακτηρίζεται και από ένα διαφορετικό χρώμα. Κάθε highlighter λαμβάνει την κατάσταση των beans του εξοπλισμού και ανάλογα ενεργοποιείται σε όποια γραμμή του πίνακα χρειάζεται (ο BusyHighlighter στα beans που η κατάστασή τους είναι "Busy" κ.ο.κ.).

5.2.4 Launcher

- Κλάσεις για την εκκίνηση της εφαρμογής:

- Main:

  Έχει μία main μέθοδο που λαμβάνει κάποια arguments και εκκινεί την εφαρμογή με την βοήθεια της επόμενης κλάσης.

- ApplicationLauncher:

  Λαμβάνει την διαδρομή (path) κάποιων αρχείων που περιέχουν κωδικοποιημένη την διαμόρφωση του Framework, τα επεξεργάζεται και τελικά εκκινεί την εφαρμογή. Δηλαδή, κατασκευάζει αντικείμενα όποιων κλάσεων του Framework και των modules χρειάζεται ώστε να εμφανιστεί το κεντρικό παράθυρο της εφαρμογής CESAR (π.χ. σίγουρα πρέπει να δημιουργήσουμε ένα instance της κλάσης Application του Framework και τις ενέργειες για εμφάνιση των Status ή Beam Files Panels των αντίστοιχων modules). Για όση ώρα φορτώνεται η εφαρμογή, η παρούσα κλάση φροντίζει να εμφανίζει ένα splash screen.

- Αρχεία για την διαμόρφωση του Framework:

Πρόκειται για κωδικοποιημένα αρχεία που περιέχουν την διαμόρφωση κάποιων κλάσεων του Framework και των modules, δηλαδή τιμές για συγκεκριμένα πεδία των κλάσεων αυτών, ώστε να προκύψει το τελικό πρόγραμμά μας (CESAR).

6

Υλοποίηση

Στο παρόν κεφάλαιο, θα αναλύσουμε κάποια σημεία της υλοποίησης του συστήματος που περιγράψαμε προηγουμένως. Στην παράγραφο 6.1, θα αναφέρουμε διάφορα προγραμματιστικά εργαλεία που χρησιμοποιήσαμε για την υλοποίηση της εφαρμογής μας. Στην παράγραφο 6.2, θα επικεντρωθούμε σε σημεία του συστήματος που είτε ήταν περίπλοκα στην υλοποίηση είτε θεωρούνται σημαντικά (π.χ. επειδή επαναλαμβάνονται ή παρουσιάζουν ένα πρωτότυπο τρόπο εργασίας).

6.1 Πλατφόρμες και προγραμματιστικά εργαλεία

Για την υλοποίηση του τμήματος του συστήματος CESAR, για το οποίο ήμασταν υπεύθυνοι, χρησιμοποιήσαμε μοντέρνα προγραμματικά εργαλεία, που βοηθούν στην ανάπτυξη εφαρμογών όπως η δική μας. Επίσης, δώσαμε ιδιαίτερο βάρος στο να είναι τα εργαλεία αυτά standard, δηλαδή να είναι ευρέως αποδεκτά και χρησιμοποιούμενα από την παγκόσμια κοινότητα της Πληροφορικής. Και αυτό ώστε να υπάρχει συνεχής υποστήριξη των εργαλείων αυτών από την πλευρά των εταιριών ή ομάδων ανθρώπων που τα ανέπτυξαν και να μην υπάρχει σοβαρή πιθανότητα εγκατάλειψης ή ματαίωσης της λειτουργίας των εργαλείων αυτών. Έτσι, μετά από προσεκτική μελέτη, καταλήξαμε στην χρησιμοποίηση των ελάχιστων δυνατών και αξιόπιστων, κατά την άποψή μας, παρακάτω εργαλείων:

- **Java:**

Επιλέξαμε την γλώσσα προγραμματισμού αυτή και συγκεκριμένα τον compiler JDK 6, λόγω του ότι, είναι μια πανίσχυρη γλώσσα όσον αφορά εφαρμογές που έχουν μεγάλες απαιτήσεις στην διεπαφή με το χρήστη (User Interface) καθώς και λόγω του ότι προτιμάται από την κοινότητα της Φυσικής για εφαρμογές που έχουν σχέση με τον έλεγχο εξοπλισμού. Επίσης, η γλώσσα Java έχει χρησιμοποιηθεί και τα προηγούμενα χρόνια στα πλαίσια του προγράμματος CESAR με αποτέλεσμα χρήσιμο API, που είχε αναπτυχθεί στα πλαίσια του προγράμματος και που χρησιμοποιήσαμε κατά κόρον, να είναι γραμμένο στη γλώσσα αυτή. Η επίσημη ιστοσελίδα της γλώσσας για προγραμματιστές είναι:

http://java.sun.com/

- **Eclipse:**

Πρόκειται για ένα πολύ χρήσιμο μέσο ανάπτυξης εφαρμογών τύπου IDE (Integrated Development Environment), κυρίως για Java, που προσφέρει μεγάλο αριθμό από δυνατότητες στον προγραμματιστή. Παρέχει, δηλαδή, σημαντικές ευκολίες σε κάποιον που αναπτύσσει λογισμικές εφαρμογές, όπως εύκολη διαχείριση επιμέρους projects (σύνολο από προγράμματα), εύκολο compile και debugging, συγγραφή και συντήρηση javadoc, συντομεύσεις για την αυτόματη παραγωγή μπλοκ κώδικα, χρήση του συστήματος CVS (Concurrent Version Control), δυνατότητα προσθαφαίρεσης διαφόρων ειδών plug-ins κ.α. Περισσότερα για το Eclipse, μπορούν να βρεθούν στην επίσημη ιστοσελίδα του:

http://www.eclipse.org/

- **Spring Framework:**

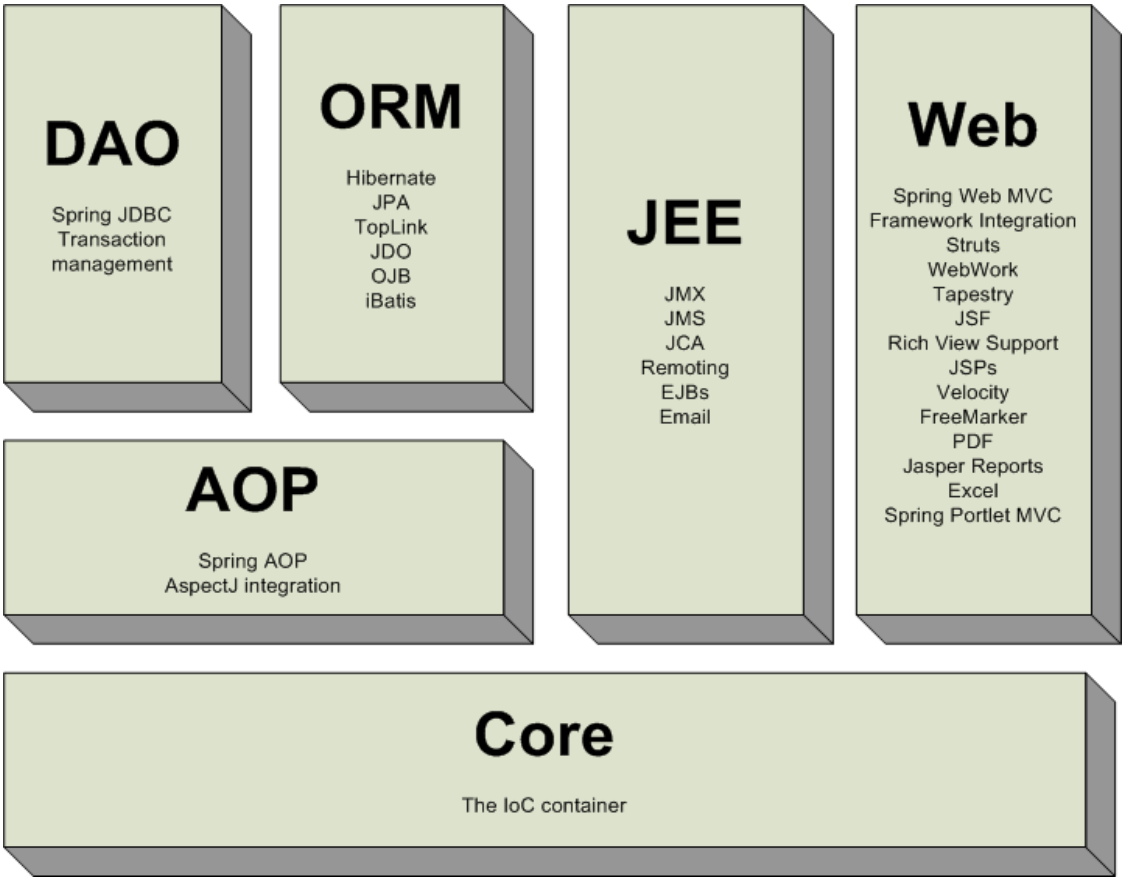
Είναι μια open-source πλατφόρμα ανάπτυξης λογισμικού (application framework) γραμμένη σε γλώσσα Java που βασίζεται στην περιγραφική γλώσσα XML και μας βοηθά στην οργάνωση των αντικειμένων των εφαρμογών μας. Αν και το Spring, σαν εργαλείο, δεν υποδεικνύει τον σκοπό της χρήσης του, έχει σήμερα καθιερωθεί ως μια από τις κυριότερες εναλλακτικές επιλογές του μοντέλου Enterprise JavaBeans- EJB (ή ακόμα και σαν συμπλήρωμα σε αυτό) για την ανάπτυξη ολοκληρωμένων web based ή standalone εφαρμογών. Ακόμα, το Spring υποστηρίζει μοντέρνες προγραμματιστικές τεχνικές όπως οι Dependency Injection, Aspect Oriented Programming και Inversion of Control, που κάνουν την διαμόρφωση των εφαρμογών μας πολύ πιο απλή διαδικασία συγκριτικά με άλλα παρόμοια εργαλεία (J2EE, .NET κλπ).

Το Dependency Injection είναι ιδιαίτερα σημαντική έννοια και χρησιμοποιείται από τα περισσότερα Frameworks της αγοράς αλλά και από άλλα εργαλεία, όπως το Unit Testing, το οποίο θα αναφέρουμε στο κεφάλαιο του Ελέγχου. Πρόκειται για την περίπτωση όπου έχουμε εξάρτηση ενός αντικειμένου μιας κλάσης από ένα εξωτερικό στοιχείο. Έχει τρεις βασικές υποπεριπτώσεις: Interface Injection, Setter Injection και Constructor Injection. Τις δύο πρώτες τις χρησιμοποιήσαμε και τις αναφέρουμε και εξηγούμε στην παράγραφο 6.2.4.

Μερικά πλεονεκτήματα του Spring Framework είναι τα παρακάτω:

- Ενθαρρύνει (χωρίς να υποβάλλει) σωστές προγραμματιστικές προσεγγίσεις σε διαφόρων ειδών εφαρμογές. Σε αυτό συνεισφέρει η οργάνωσή του, που βασίζεται σε POJOs (Plain Old Java Objects), όπως και πολλά άλλα εργαλεία της γλώσσας Java.
- Είναι απλό και εύκολο στην εκμάθηση.
- Υποστηρίζει μεγάλη γκάμα εφαρμογών, όπως εφαρμογές που έχουν να κάνουν με Web, Server και Client-side συστήματα κλπ. Αυτό φαίνεται και από το σχήμα 5, που ακολουθεί.
- Ελαχιστοποιεί το κόστος χρήσης interfaces.
- Αν χρησιμοποιηθεί σωστά, ελαχιστοποιεί και την εξάρτηση του κώδικά μας στο API του. Έτσι, ο κώδικας μιας εφαρμογής μας παραμένει στην ουσία του ανεξάρτητος από το Spring και μπορεί να πάψει να διαμορφώνεται από αυτό οποιαδήποτε στιγμή.
- Προσφέρει διασύνδεση με πλήθος άλλων εργαλείων.
- Διευκολύνει τον έλεγχο των εφαρμογών.

κ.α.



Σχήμα 5: Υποσυστήματα του Spring Framework

Στο παραπάνω σχήμα φαίνονται τα συστατικά του Spring. Το DAO επιτρέπει την επικοινωνία με βάσεις δεδομένων, το ORM προσφέρει διασύνδεση με κάποια άλλα frameworks, το JEE προσφέρει Java Enterprise Edition επιλογές, το Web περιέχει χρήσιμα εργαλεία για σχεδίαση web εφαρμογών, το AOP παρέχει Aspect Oriented Programming λύσεις στον σχεδιασμό της εφαρμογής μας και το Core είναι η καρδιά του Framework, δηλαδή το σύνολο των Beans (διαμορφωμένα αντικείμενα) της εφαρμογής μας μαζί με τις δυνατότητες που το Spring μας παρέχει για αυτά.

Περισσότερες πληροφορίες στην επίσημη ιστοσελίδα του project:

http://www.springframework.org/

- **SwingX:**

Το SwingX είναι ένα open-source εργαλείο που παρέχει GUI στοιχεία (πίνακες, κουμπιά, panels κλπ), τα οποία επεκτείνουν τις δυνατότητες των αντίστοιχων στοιχείων του Swing της Java. Δηλαδή, παρέχει τα ίδια εργαλεία αλλά με επιπλέον δυνατότητες, χωρίς να χρειαστεί να τις υλοποιήσουμε μόνοι μας (με κώδικα). Για παράδειγμα, μια δεντρική δομή του συγκεκριμένου εργαλείο είναι το JXTree, το οποίο είναι υποκλάση του JTree του κλασσικού Swing της Java και προσφέρει κάποιες επιπλέον δυνατότητες στον προγραμματιστή. Το

σχήμα αυτό είναι ιδιαίτερα βολικό, καθώς αν για οποιοδήποτε λόγο θέλουμε να απαλλαγούμε στο μέλλον από το SwingX, το μόνο που έχουμε να κάνουμε είναι να αντικαταστήσουμε τις κλάσεις του SwingX με τις αντίστοιχες του Swing (και ενδεχομένως να κάνουμε και κάποιες άλλες μικρές αλλαγές στον κώδικά μας). Έτσι, θα χάσουμε βέβαια τις επιπλέον δυνατότητες που αυτόματα (χωρίς δικό μας κώδικα) προφέρει το εν λόγω εργαλείο.

Περισσότερες πληροφορίες στις επίσημες ιστοσελίδες του project:

https://swingx.dev.java.net/ και http://swinglabs.org/

6.2 Λεπτομέρειες υλοποίησης

6.2.1 Framework

- Εργαλεία:

Σε αυτήν την παράγραφο θα αναφέρουμε γενικού σκοπού εργαλεία που αναπτύξαμε μαζί με την βοήθεια και άλλων εργαζομένων του CERN. Τα εργαλεία αυτά χρησιμοποιούνται όχι μόνο από την δική μας αλλά και από πλήθος άλλων εφαρμογών που αναπτύσσει το section AB-CO-AP του CERN, με το οποίο συνεργαστήκαμε. Τα εν λόγω εργαλεία βρίσκονται σε αυτό το υποσύστημα (Framework) καθώς χρησιμοποιούνται και από το ίδιο το υποσύστημα αλλά και τα επιμέρους modules της εφαρμογής μας (θυμίζουμε από το κεφάλαιο Σχεδίαση, ότι μπορούμε να έχουμε εξάρτηση από τα modules προς το Framework αλλά όχι και το αντίστροφο).

Το πρώτο από αυτά τα εργαλεία δεν είναι άλλο από τον πίνακα που χρησιμοποιούμε στα modules μας. Πρόκειται για την κλάση TableExplorer, η οποία χρησιμοποιεί και την κλάση PrintingDialog. Και οι δύο αυτές κλάσεις βρίσκονται στο υποσύστημα Framework.

Καταρχάς, η κλάση TableExplorer είναι υποκλάση της JXTable, η οποία ανήκει στο προγραμματιστικό εργαλείο SwingX. Το SwingX είναι ένα εργαλείο για το οποίο αναφέρουμε στην παράγραφο 6.1. Η κλάση TableExplorer είναι, με άλλα λόγια ένας κλασσικός πίνακας JTable του swing, μαζί με τις επεκτάσεις που προσφέρει το εργαλείο SwingX και τις επεκτάσεις που κάνουμε εμείς στην συγκεκριμένη κλάση.

Πριν αναφέρουμε διάφορες δυνατότητες του πίνακα αυτού, πρέπει να τονίσουμε ότι στοιχεία (γραμμές) του πίνακα είναι αντικείμενα της κλάσης Node (θα τα ονομάζουμε nodes ή κόμβους ή στοιχεία), η οποία αναπτύχθηκε από άλλους εργαζόμενους του CERN (και άρα ανήκει στην κατηγορία General Stuff της παραγράφου 5.1). Τα nodes έχουν χαρακτηριστικά, όπως όνομα, εικονίδιο, περιγραφή, ενέργειες, προκαθορισμένη ενέργεια κλπ.

Παραδείγματα των δυνατοτήτων που πραγματοποιούμε στην κλάση TableExplorer είναι:

1. Εφαρμογή Tool Tips:

Στις διάφορες εφαρμογές που περιέχουν κάποιον πίνακα, είναι καλό να υπάρχει ένας μηχανισμός, ώστε να εμφανίζονται Tool Tips, όταν το ποντίκι μένει για λίγη ώρα πάνω σε ένα στοιχείο του πίνακα.

2. Ενέργειες των nodes:

Ο πίνακας μας πρέπει να έχει γνώση των ενεργειών που είναι διαθέσιμες για κάθε στοιχείο του πίνακα (node), ώστε το πάτημα του δεξιού κλικ πάνω σε ένα node να παρουσιάζει τις διαθέσιμες ενέργειες σε αυτό και το διπλό αριστερό πάτημα του ποντικιού να εκτελεί την προκαθορισμένη ενέργεια του node (αν υπάρχει).

3. Ενέργειες του πίνακα:

Ο TableExplorer μας έχει και ενσωματωμένες 4 ενέργειες που μπορούν να εκτελεστούν στον ίδιο τον πίνακα. Τα ονόματα αυτών είναι “print” (εκτύπωση), “import” (εισαγωγή), “export” (εξαγωγή) και “find” (εύρεση). Η πρώτη (διαθέσιμη by default με το πάτημα των πλήκτρων Ctrl-P) εκτυπώνει τον πίνακα, η δεύτερη θέτει στον πίνακα τα δεδομένα που εισάγει από ένα αρχείο τύπου *.csv, η τρίτη εξάγει τα δεδομένα του πίνακα σε ένα αρχείο του ίδιου τύπου και η τελευταία (διαθέσιμη by default με το πάτημα των πλήκτρων Ctrl-F) βρίσκει μία λέξη που εισάγει ο χρήστης σκανάροντας τον πίνακα.

Ο TableExplorer παρέχει και άλλες δυνατότητες (όπως renderers) τις οποίες όμως δεν θα χρησιμοποιήσουμε.

Εκτός από τον TableExplorer, χρησιμοποιήσαμε και ένα σύνολο άλλων εργαλείων που έχουν σχέση με τα παράθυρα (frames) της εφαρμογής μας. Τις αντίστοιχες κλάσεις, δεν τις αναφέραμε στο κεφάλαιο Σχεδίαση, καθώς δεν αναπτύχθηκαν από εμάς, αλλά από άλλους εργαζομένους του CERN. Μπορούμε να πούμε ότι οι κλάσεις αυτές ανήκουν στην κατηγορία General Stuff, που αναφέραμε στην παράγραφο 5.1.

Ένα από αυτά frames είναι ένα είδος παραθύρου που χρησιμοποιήσαμε για το κυρίως παράθυρο της εφαρμογής μας. Το frame αυτό είναι μία κλάση που ονομάζεται ExternalFrame, η οποία εκτός από τις συνήθεις λειτουργίες ενός παραθύρου, παρέχει και μια κονσόλα όπου εμφανίζονται τα διάφορα μηνύματα που θέλουμε να παρουσιάσουμε στους χρήστες ή στους προγραμματιστές (logging output). Επίσης, υπάρχει μια μπάρα που εμφανίζει μηνύματα σφάλματος ή προειδοποίησης στο κάτω μέρος του παραθύρου (status line) και μια μπάρα προόδου για χρονοβόρες διεργασίες που επιτελούνται στην εφαρμογή μας (progress bar). Τέλος, διευκολύνει την προσθήκη κάποιου μενού (menu) στο πάνω μέρος του παραθύρου ή και κάποιας μπάρας με ενέργειες (toolbar). Τμήμα του interface του ExternalFrame που φανερώνει αυτές τις δυνατότητες είναι το εξής:

```
public void setConsoleVisible(boolean visible);
public void setJMenuBar(JMenuBar menuBar);
```

```
public void setToolBar(Object[] actionIDs);
public void setToolBar(JToolBar toolBar);
```

Η πρώτη μέθοδος εμφανίζει ή εξαφανίζει την κονσόλα του παραθύρου, ανάλογα με την παράμετρο visible. Η δεύτερη θέτει στο παράθυρο ένα menu, ενώ οι 2 τελευταίες θέτουν στο παράθυρο κάποιο toolbar.

Εκτός από την κλάση ExternalFrame, χρησιμοποιούμε και την InternaFrame, η οποία αποτελεί ένα άλλο είδος παραθύρου. Αυτό το παράθυρο είναι εσωτερικό ενός workspace. Έχει και αυτό μια κονσόλα και μια μπάρα μηνυμάτων όπως και η κλάση ExternalFrame. Πρέπει εδώ να σημειώσουμε ότι οι κλάσεις InternaFrame και ExternalFrame είναι υποκλάσεις της AppFrame, η οποία αναπαριστά οποιοδήποτε είδος παραθύρου σε μία εφαρμογή.

Ακόμα, χρησιμοποιούμε έναν FrameManager, ο οποίος χειρίζεται τα διάφορα παράθυρα της εφαρμογής μας. Ορισμένες μέθοδοι της κλάσης αυτής φαίνονται παρακάτω:

```
public static FrameManager getInstance();
public ExternalFrame getMainFrame(String title, JComponent
          rootComponent);
public ExternalFrame createExternalFrame(String title, JComponent
                                                  rootComponent);
public InternalFrame createInternalFrame(String title, JComponent
                                                  rootComponent);
public void closeFrame(AppFrame frame);
```

Η πρώτη επιστρέφει το μοναδικό instance της κλάσης (την πρώτη φορά που καλείται το κατασκευάζει ενώ τις επόμενες επιστρέφει αυτό που έχει κατασκευαστεί). Η δεύτερη κατασκευάζει το κυρίως παράθυρο της εφαρμογής μας και το επιστρέφει. Η τρίτη κατασκευάζει κάποιο External Frame και η τέταρτη κάποιο Internal Frame. Αυτές οι δύο μέθοδοι παίρνουν ως ορίσματα τον τίτλο του παραθύρου (title) και το περιεχόμενο του παραθύρου (rootComponent). Τέλος, η πέμπτη μέθοδος κλείνει ένα οποιοδήποτε παράθυρο που έχουμε κατασκευάσει.

Χρήση των παραπάνω κάνουμε όταν δημιουργούμε το κυρίως παράθυρο της εφαρμογής μας στην μέθοδο launchMainFrame() της κλάσης Application του Framework:

```
ExternalFrame mainFrame = FrameManager.getInstance().getMainFrame(
                        applicationName, workspaceContainer);
mainFrame.setVisible(true);
```

Στο παραπάνω κομμάτι κώδικα, στην μεταβλητή applicationName έχουμε αποθηκεύσει το όνομα της εφαρμογής μας, δηλαδή το String “CESAR”, ενώ στην workspaceContainer ένα (μοναδικό στην εφαρμογή μας) instance της κλάσης WorkspaceContainer του Framework. Αρχικά, κατασκευάζουμε το κυρίως παράθυρο της εφαρμογής μας και στην συνέχεια το εμφανίζουμε στην οθόνη.

Επίσης, χρήση των frames αυτών κάνουμε και στην μέθοδο addInternalFrameToWorkspace της κλάσης Application. Συγκεκριμένα, η χρήση γίνεται ως εξής:

```
InternalFrame frame =
FrameManager.getInstance().createInternalFrame(title, rootPanel);
workspace.add(frame);
frame.setVisible(true);
```

Στον παραπάνω κώδικα, κατασκευάζουμε το Internal Frame, στη συνέχεια, το προσθέτουμε σε κάποιο workspace και τέλος το εμφανίζουμε. Στην μεταβλητή title έχουμε αποθηκεύσει τον τίτλο του Internal Frame, στην rootPanel το περιεχόμενο του Internal Frame και στην workspace το workspace στο οποίο θέλουμε να προσθέσουμε το Internal Frame.

Με χρήση των frames που αναφέραμε, μπορούμε σε οποιαδήποτε κλάση, η οποία αποτελεί το περιεχόμενο του παραθύρου, να χρησιμοποιήσουμε την κονσόλα του παραθύρου ως logging output. Αυτό γίνεται εφικτό αν εισάγουμε στην κλάση μας τον ακόλουθο κώδικα (παράδειγμά μας η κλάση FilePanel του module Αρχεία Δέσμης):

```
public class FilePanel extends JPanel implements LogSource {
private Log logger = LogFactory.getLog(FilePanel.class);
private StatusLine statusLine = new
                    StatusLineLogAdapter(this.logger);
// -- implements LogSrc ------------------------------------
      public void setLogger(Log newLogger) {
            this.logger = newLogger;
      }
      public void setStatusLine(StatusLine newStatusLine) {
            this.statusLine = newStatusLine;
      }
```

Οπότε, κάθε φορά που εμφανίζουμε κάποιο μήνυμα με τις μεθόδους logger.error("…"), logger.warning("…"), logger.debug("…") ή logger.info("…"), τότε το μήνυμα γίνεται ορατό στην κονσόλα με το αντίστοιχο χρώμα που φανερώνει το χαρακτήρα του μηνύματος (π.χ. κόκκινο για σφάλμα). Αντίστοιχα, μπορούμε να εμφανίσουμε διάφορα μηνύματα στην μπάρα του παραθύρου με τις μεθόδους statusLine.warn("…") και statusLine.error("…").

Πέρα από τα διάφορα frames, είναι καλό να αναφερθούμε και στο το πώς διαχειριστήκαμε τις ενέργειες στα διάφορα modules της εφαρμογής μας. Συγκεκριμένα, χρησιμοποιήσαμε την κλάση ReflectiveActionHandler (βρίσκεται στο Framework), η οποία λειτουργεί ως διαχειριστής ενεργειών (action manager), που υποστηρίζει αντανάκλαση (reflection). Με τον όρο αντανάκλαση εννοούμε την δυνατότητα να αναθέτουμε σε μεθόδους κάποιας συγκεκριμένης κλάσης την εκτέλεση μιας συγκεκριμένης ενέργειας (action handling methods). Για παράδειγμα, χρήση αυτού του διαχειριστή ενεργειών γίνεται με τον παρακάτω κώδικα, τον οποίον λάβαμε από την κλάση FilePanel (του module Αρχεία Δέσμης) και αφορά την ενέργεια View:

```
public String ACTION_COMMAND_VIEW = "view";
private TargetableAction[] actions = new TargetableAction[10];
private JButton[] buttons = new JButton[10];
```

```
ImageIcon viewIcon = new
            ImageIcon(ImageResource.class.getResource("View.gif"));
actions[0] = new ViewAction(table, "View", ACTION_COMMAND_VIEW,
            viewIcon,"View the beamfile contents(ALT+V)");
buttons[0] = new JButton(actions[0]);
ReflectiveActionHandler handler = new ReflectiveActionHandler(table);
handler.addAction(ACTION_COMMAND_VIEW, this, "viewSettings");
TargetManager.getInstance().addTarget(handler,true);
```

Στην 1η γραμμή του παραπάνω κώδικα, διαλέγουμε ένα μοναδικό αναγνωριστικό όνομα για την ενέργειά μας (View). Στην 2η, δημιουργούμε το σύνολο των ενεργειών του FilePanel, που είναι αντικείμενα της κλάσης TargetableAction του εργαλείου SwingX. Οι ενέργειες αυτού του τύπου μας επιτρέπουν να έχουμε κάποιο στόχο για αυτές, δηλαδή μια μέθοδο που τις χειρίζεται. Στην 3η γραμμή, δημιουργούμε το σύνολο των κουμπιών για τις ενέργειες. Στην 4η και 5η γραμμή δημιουργούμε το εικονίδιο της ενέργειας View. Εν συνεχεία, δημιουργούμε την ενέργεια View, που είναι αντικείμενο της κλάσης ViewAction και ακολούθως, το κουμπί της ενέργειάς μας. Αμέσως μετά, δημιουργούμε τον χειριστή των ενεργειών με βάση τον πίνακά του panel και προσθέτουμε σε αυτόν την ενέργεια View με την μέθοδο addAction του χειριστή. Μπορούμε να διευκρινίσουμε εδώ ότι το 2ο όρισμα της μεθόδου addAction είναι η κλάση της μεθόδου στην οποία ανήκει η μέθοδος που χειρίζεται την ενέργεια και το 3ο αποτελεί το όνομα της εν λόγω μεθόδου. Τέλος, δημιουργούμε έναν TargetManager, ο οποίος είναι κλάση και πάλι του swingX και ο οποίος χειρίζεται κλάσεις που περιέχουν στόχους, και προσθέτουμε σε αυτόν τον χειριστή ενεργειών μας.

Έτσι, μπορούμε τώρα να έχουμε στην ίδια την κλάση που περιέχει το παραπάνω τμήμα κώδικα μια μέθοδο με το όνομα viewSettings , η οποία να καλείται κάθε φορά που επιλέγεται από το χρήστη η ενέργεια View (πάτημα αντίστοιχου κουμπιού ή Alt + V):

```
public void viewSettings(){
      //code that launches View Panel with specific info
      //of the selected beam file
}
```

- <u>Abstraction:</u>

Είναι σημαντικό να δείξουμε πως κρατήσαμε αφηρημένο το υποσύστημα που εξετάζουμε (Framework). Έτσι, αναφέρουμε, για αρχή, την κλάση GeneralAction, η οποία αναπαριστά μία οποιαδήποτε ενέργεια. Η κλάση αυτή είναι abstract, με αποτέλεσμα, για να έχουμε μια συγκεκριμένη ενέργεια, να πρέπει να δημιουργήσουμε μία υποκλάση αυτής και να ορίσουμε μία μέθοδο με το όνομα actionPerformed (είναι abstract μέθοδος της GeneralAction). Στην μέθοδο actionPerformed θα βρίσκεται ο κώδικας που εκτελείται κάθε φορά που εκτελούμε την συγκεκριμένη ενέργεια. Το αξιοσημείωτο με αυτήν την κλάση είναι τα πεδία της, που είναι τα:

```
private String name;
private String iconPath;
private String iconName;
private String description;
private String keyShortcut;
private SecurityService securityService;
```

Το 1ο είναι το όνομα της ενέργειας, το 2ο η τοποθεσία του αρχείου που χρησιμοποιούμε ως εικονίδιο για την ενέργεια, το 3ο είναι το όνομα του αρχείου μιας προκαθορισμένης τοποθεσίας με εικόνες (image repository), το οποίο χρησιμοποιούμε ως εικονίδιο για την ενέργεια (ή το 2ο ή το 3ο πεδίο χρειάζεται να παρέχουμε), το 4ο είναι μια αναλυτική περιγραφή της ενέργειας (το χρησιμοποιούμε σαν tooltip), το 5ο μια συντόμευση πλήκτρων για την ενέργεια και το τελευταίο μια υπηρεσία που μας βοηθά να καθορίζουμε τα δικαιώματα των διαφόρων χρηστών ως προς την ενέργειά μας.

Τα πεδία αρχικοποιούνται μέσω του υποσυστήματος Launcher, όπως θα δούμε στην αντίστοιχη παράγραφο. Επίσης, δεν χρειάζεται όλα να έχουν τιμή. Η ενέργεια αποκτά τα αντίστοιχα στοιχεία των πεδίων αυτών (όνομα, περιγραφή κλπ) μέσω μιας μεθόδου init που ορίζουμε στην κλάση GeneralAction και καλείται κατά την κατασκευή ενός αντικειμένου της κλάσης (η κατασκευή γίνεται από το Launcher, όπως θα δούμε).

Σημαντικότερο από όλα είναι το πεδίο securityService, που είναι τύπου SecurityService (interface). Αυτό σημαίνει ότι το πεδίο αυτό θα αρχικοποιηθεί (μέσω του Launcher), αν χρειάζεται, με ένα αντικείμενο κλάσης που υλοποιεί το interface SecurityService. Αυτή η λειτουργία μας δείχνει κάποια σημεία που έχουμε καταστήσει σημαντικά για την εφαρμογή μας στα προηγούμενα κεφάλαια όπως ο διαχωρισμός λειτουργικότητας-υλοποίησης του framework. Εδώ, δηλαδή, διαχωρίζεται η λειτουργικότητα της κλάσης GeneralAction από την υλοποίηση που διαλέγουμε κάθε φορά για την interface SecurityService. Ταυτόχρονα, με αυτή την λογική που ακολουθούμε, απαλείφονται CESAR-specific στοιχεία από το framework. Π.χ. το interface SecurityService το κρατάμε χωρίς CESAR χαρακτηριστικά και την υλοποίηση με CESAR χαρακτηριστικά. Το ότι το interface δεν έχει CESAR χαρακτηριστικά φαίνεται από τον κώδικά του:

```
public interface SecurityService {
	public void addSecurityListener(SecurityListener listener);
	public boolean isAuthorized();
	public boolean isAuthorized(String resource);
	public void fireSecurityEvent();
	public void login(String username, String password) throws
		FailedLoginException, LoginException ;
	public String logout();
	public void autoLogin();
	public boolean canLogout();
	public String getActiveUserName();
}
```

Επιστρέφοντας στην κλάση GeneralAction, αν το πεδίο securityService δεν είναι κενό, τότε η κλάση μας προσθέτει τον εαυτό της (κατά την αρχικοποιητική μέθοδο init) στην λίστα με τις κλάσεις που «παρακολουθούν» τις αλλαγές στον ενεργό χρήστη του συστήματος. Αυτό σημαίνει βέβαια ότι η κλάση GeneralAction υλοποιεί το interface SecurityListener, ώστε να μπορεί να ειδοποιηθεί σε κάθε περίπτωση επιτυχών login, autologin ή logout. Έτσι, σε κάθε τέτοια περίπτωση, η κλάση μας ζητά την βοήθεια του securityService και ανάλογα με τον ενεργό χρήστη ενεργοποιεί ή απενεργοποιεί τον εαυτό της. Αυτό γίνεται με την εντολή:

```
securityService.isAuthorized();
```

ή την:

```
securityService.isAuthorized(str);
```

,όπου str κάποιο String. Η 1η μέθοδος εξετάζει το ποιος είναι ο ενεργός χρήστης και επιστρέφει την αληθή τιμή αν αυτός ανήκει σε μια λίστα από προκαθορισμένους δικαιωματικούς χρήστες (default authorized users). Η 2η εξετάζει το ποιος είναι ο ενεργός χρήστης και επιστρέφει την αληθή τιμή αν αυτός συμφωνεί με την περιγραφή που αντιστοιχεί στο String str. Αυτή η περιγραφή είναι μιας στάνταρ μορφής πεδίο (resource) που περιγράφει ποιος χρήστης ή σύνολο χρηστών έχουν δικαίωμα να κάνουν μια ενέργεια. Γι' αυτό το resource διαλέγουμε, στην κλάση που υλοποιεί το interface SecurityService, την μορφή ονομάτων χρηστών χωρισμένα με κάθετο, δηλαδή: "/όνομα χρήστη1/ όνομα χρήστη2/…./" . Έτσι, το "/SUPER/" σημαίνει ότι μια ενέργεια πρέπει να είναι διαθέσιμη μόνο σε χρήστες τύπου SUPER USER. Αυτή η λογική του resource κάνει το interface SecurityService ανεξάρτητο του CESAR, καθώς το resource είναι ένα απλό String. Επίσης, κρατάει την περιγραφή των δικαιωμάτων για τις διάφορες ενέργειες αρκετά απλή, καθώς δεν χρειάζεται να έχουμε σε κάποιο σημείο όλες τις ενέργειες (οι οποίες είναι πάρα πολλές) μαζί με τα δικαιώματα των χρηστών για κάθε ενέργεια.

Είναι σημαντικό να παρατηρήσουμε ότι στο υποσύστημα Framework έχουμε απλό κώδικα Java και η διαμόρφωσή του μπορεί να γίνει κατά οποιαδήποτε τρόπο και με οποιοδήποτε εργαλείο θέλουμε στο υποσύστημα Launcher. Στο Launcher, όπως θα δούμε, διαλέξαμε το εργαλείο Spring (το οποίο αναφέραμε στην παράγραφο 6.1), αλλά θα μπορούσαμε να είχαμε διαλέξει οποιοδήποτε άλλο (ή αν θέλουμε, μπορούμε να το αλλάξουμε στο μέλλον). Αυτό διευκολύνει την συντήρηση της εφαρμογής μας καθώς τα διάφορα προγραμματιστικά εργαλεία, όπως το Spirng, μπορεί να τροποποιούνται κατά την διάρκεια του χρόνου (με τα διάφορα releases τους) αλλά και καμιά φορά να ξεπερνιόνται και να αντικαθίστανται από άλλα καλύτερα και πιο μοντέρνα.

Το γεγονός ότι στο framework έχουμε απλό κώδικα Java οφείλεται στην έννοια Dependency Injection του Spring, που θα αναλύσουμε στο υποσύστημα Launcher στο παρόν κεφάλαιο.

6.2.2 Αρχεία Δέσμης

- Beans:

Όπως έχουμε ήδη αναφέρει, ορίζουμε το μοντέλο του πίνακά μας ως ένα bean, δηλαδή μια κλάση η οποία υλοποιεί ένα interface, κατάλληλο για τα δεδομένα που θέλουμε να απεικονίσουμε στον πίνακα. Ας δούμε, όμως, βήμα-βήμα αυτή την διαδικασία, παίρνοντας ως παράδειγμα τον πίνακα με τους collimators στο παράθυρο που εμφανίζεται με την ενέργεια View σε ένα αρχείο δέσμης. Στην κλάση ViewPanel, έχουμε το τμήμα του κώδικα:

```
private NodeTableAdapter table2Model;
private TableExplorer table2;
private JPanel createCollimPanel() throws SpseaException {
      this.table2Model = createNodeModelCollim(
      CollimRefsBeanImpl.class);
      this.table2 = new TableExplorer(this.table2Model);
}
private NodeTableAdapter createNodeModelCollim(Class beanClass) {
      return new NodeTableAdapter(beanClass, // class of the bean
            // Column headers and property names
            new String[] { "Collimator","Jaw1 File Reference","Jaw2
            File Reference"}, new String[] {
                  "physicName","jaw1","jaw2" });
}
```

Στον παραπάνω κώδικα, η μεταβλητή table2Model είναι το μοντέλο του πίνακα με τους collimators, ενώ η table2 είναι ο ίδιος ο πίνακας. Στην μέθοδο createCollimPanel δημιουργούμε το μοντέλο του πίνακα με την βοήθεια της μεθόδου createNodeModelCollim και στην συνέχεια δημιουργούμε τον πίνακα με βάση αυτό. Στην μέθοδο createNodeModelCollim, παρατηρούμε ότι χρησιμοποιούμε μια βοηθητική κλάση, την NodeTableAdapter, η οποία αποτελεί την υλοποίηση της έννοιας του μοντέλου του πίνακα. Έτσι, το μοντέλο που δημιουργήσαμε θέτει στον πίνακα τρεις στήλες με ονόματα “Collimator", "Jaw1 File Reference" και "Jaw2 File Reference". Οι τιμές για τις στήλες του πίνακα αντιστοιχούν μία προς μία στις ιδιότητες της κλάσης CollimRefsBeanImpl (ανήκει στην κατηγορία Beans) με τα ονόματα “physicName”, “jaw1” και “jaw2”. Η κλάση CollimRefsBeanImpl υλοποιεί το interface CollimBean. Οι ιδιότητες της κλάσης CollimRefsBeanImpl ορίζονται ως τα πεδία για τα οποία έχουμε getter methods που υλοποιούν το interface CollimBean. Δηλαδή:

```
public interface CollimBean {
      public static final String PHYSIC_NAME_PROPERTY = "physicName";
      public static final String JAW1_PROPERTY = "jaw1";
      public static final String JAW2_PROPERTY = "jaw2";
      public static final String
                  OFFICIAL_NAME_PROPERTY="officialName";
      public String getPhysicName();
      public double getJaw1();
      public double getJaw2();
```

```
	public String getOfficialName();
}
```

Αυτό σημαίνει ότι η κλάση CollimRefsBeanImpl πρέπει να υλοποιεί τις μεθόδους getPhysicName, getJaw1, getJaw2 και getOfficialName. Αυτές καθορίζουν ότι οι ιδιότητες του bean (δηλαδή της κλάσης CollimRefsBeanImpl) είναι τέσσερις και έχουν ως τύπο τον τύπο που επιστρέφουν οι αντίστοιχες μέθοδοι και ονόματα τα ονόματα των μεθόδων χωρίς το get και με το πρώτο γράμμα πεζό (φαίνεται και από τα ονόματα που έχουμε δώσει στις πρώτες γραμμές του interface). Οι 3 πρώτες ιδιότητες χρησιμοποιήθηκαν, όπως είδαμε σαν στήλες του πίνακα. Η 4η (officialName) αποτελεί ένα εναλλακτικό όνομα για τους collimators (υπάρχει για όλους τους τύπους εξοπλισμού) και το χρησιμοποιούμε ως περιγραφή των στοιχείων του πίνακα (node description) και άρα ως tooltip στα στοιχεία του πίνακα.

Η υλοποίηση του παραπάνω interface είναι η εξής:

```
public class CollimRefsBeanImpl implements CollimBean {
	private final PropertyChangeSupport propertyChangeSupport;
	private final CollimatorReferences references;
	public CollimRefsBeanImpl (CollimatorReferences references) {
		this.references = references;
		propertyChangeSupport = new PropertyChangeSupport(this);
	}
	public String getPhysicName() {
		return references.getPhysicName();
	}
	public String getEqName() {
		return references.getEquipmentName();
	}
	public double getJaw1() {
		return references.getJaw1().doubleValue();
	}
	public double getJaw2() {
		return references.getJaw2().doubleValue();
	}
	public String getOfficialName() {
		return references.getOfficialName();
	}
}
```

Παρατηρούμε την χρησιμοποίηση της κλάσης CollimatorReference η οποία αποτελεί το σύνολο των file references για κάποιον collimator, που ανακτώνται από τον server. Η κλάση αυτή δεν αναπτύχθηκε από εμάς, δηλαδή εμπίπτει στην κατηγορία κλάσεων General Stuff της παραγράφου 5.1.

Τέλος, ο καθορισμός των δεδομένων στον πίνακα γίνεται με την κλήση της μεθόδου:

```
public void updateTable2() throws SpseaException {
	table2Model.setModel(BeamfilesUtils.createCollimNodes(bfr,new
	String[]{ACTION_COMMAND_WRITE2,
	ACTION_COMMAND_CHANGE},ACTION_COMMAND_WRITE2));
}
```

Οι μεταβλητές ACTION_COMMAND_WRITE2 και ACTION_COMMAND_CHANGE περιέχουν τα ονόματα (αναγνωριστικά) των ενεργειών που μπορούν να εκτελεστούν στους collimators, όπως είδαμε και στην παράγραφο 6.2.1 για κάποιες άλλες ενέργειες. Οι ενέργειες

αυτές είναι η αλλαγή των file references για κάποιον collimator και η αλλαγή του σχόλιου του αρχείου, αντίστοιχα.

Έτσι, καλείται η μέθοδος:

```
public static Node[] createCollimNodes(BeamFileRecord bfr,String []
      actions,String defaultAction) throws SpseaException {
            CollimRefsBeanImpl[] r = getDataFromCollimators(bfr);
            Node[] nodes = new Node[r.length];
            for (int i = 0; i < r.length; i++) {
                  nodes[i] = createNode(r[i],actions,defaultAction);
            }
            return nodes;
}
```

Η μέθοδος αυτή παίρνει ως ορίσματα το αρχείο που έχει ανοιχτεί (bfr), το σύνολο των ενεργειών (actions) και την προκαθορισμένη ενέργεια (defaultAction) και κατασκευάζει και επιστρέφει τα στοιχεία του πίνακα (nodes). Χρησιμοποιούμε, όπως φαίνεται από τον κώδικα, τις 2 επόμενες μεθόδους: την getDataFromCollimators και την createNodes.

Η πρώτη μέθοδος δημιουργεί και επιστρέφει τα beans. Αρχικά ανακτά τα file references του συγκεκριμένου αρχείου για τα collimators του αρχείου (κλάση CollimatorReference που αναφέραμε παραπάνω), με την βοήθεια του BeamFileHandler. Ο BeamFileHandler είναι κλάση που χειρίζεται κατευθείαν τα αρχεία της δέσμης, που είναι αποθηκευμένα σε server του συστήματος CESAR. Αυτή η κλάση δεν αναπτύχθηκε από εμάς και ανήκει στην κατηγορία General Stuff της παραγράφου 5.1. Έτσι, η μέθοδός μας, τελικά, κατασκευάζει από τα file references του αρχείου bfr και επιστρέφει το σύνολο των beans που προορίζονται για τον πίνακά μας.

Η δεύτερη μέθοδος δημιουργεί και επιστρέφει ένα node πίνακα, με στήλες τις ιδιότητες ενός bean από αυτά που δημιούργησε η προηγούμενη μέθοδος, περιγραφή (για το tooltip) το Official Name του bean, ενέργειες τις ενέργειες με αναγνωριστικά αυτά που έχουμε αποθηκεύσει στην μεταβλητή actions και προκαθορισμένη ενέργεια την ενέργεια με αναγνωριστικό ότι έχουμε αποθηκεύσει στην defaultAction . Οι ενέργειες γίνονται διαθέσιμες με δεξί κλικ πάνω στο node ενώ η προκαθορισμένη ενέργεια εκτελείται με διπλό αριστερό κλικ.

- **Ενέργειες:**

Σε αυτήν την παράγραφο θα αναλύσουμε το πώς ορίσαμε κάποιες ενέργειες, που παρουσιάζουν ενδιαφέρον.

Στην παράγραφο 6.2.1 αναφέραμε την έννοια των μεθόδων που χειρίζονται ενέργειες (action handling methods). Έτσι, εδώ μπορούμε απλά να παρουσιάσουμε και να εξηγήσουμε μια ενδεικτική μέθοδο από αυτές.

Παίρνουμε ως παράδειγμα την ενέργεια “Write to file reference (magnets)”, η οποία εκτελείται από το παράθυρο της ενέργειας View και η οποία αλλάζει την τιμή του ρεύματος

του επιλεγμένου magnet στον αντίστοιχο πίνακα του ίδιου παραθύρου για το αρχείο που έχει προεπιλεγεί από τον κεντρικό πίνακα με τα αρχεία. Η αντίστοιχη action handling method είναι η εξής:

```
public void writeFileRefsTable1() throws SpseaException {
      int magnetRow=table1.getSelectedRow();
      if (magnetRow!=-1) {
                  String answer= JOptionPane.showInputDialog
                  (null,"Enter value current:");
                  if (answer!=null) {
                        double current=Double.valueOf(answer);
                        BeamFileHandler bfh =
                              StaticEnvironment.getBeamFileHandler();
                        String fileName = bfr.getName();
                        References[]  refs =
                                    bfh.getFileRef(fileName);
                        RefsByType rbt = new RefsByType(refs);
                        References[] refs2 =
                              rbt.getByType(MagnetReferences.class);
                        MagnetReferences magRef=new
                              MagnetReferences(refs2[magnetRow]);
                        BeamFileManager bfm=new BeamFileManager();
                        bfm.editRefsMagnets(magRef,current,bfr,
                              refs2[magnetRow].getPhysicName(),logger
                        );
                        updateTable1();
                  }
                  else {
                        logger.info("Change value of current " +
                                                + "cancelled");
                  }
            }
            else {
                  statusLine.warn("You must select a magnet!");
            }
      }
```

Παρατηρούμε ότι η παραπάνω μέθοδος λαμβάνει τον επιλεγμένο magnet του πίνακα με τους magnets (table1), με την βοήθεια της μεθόδου getSelectedRow του πίνακα. Στη συνέχεια, ζητά από το χρήστη μια νέα τιμή ρεύματος γι' αυτόν τον magnet. Την τιμή αυτή αποθηκεύουμε στην μεταβλητή current. Ακολούθως, θα λάβουμε τις αναφορές (κλάση References) του αρχείου για το οποίο έχει κατασκευαστεί το αντικείμενο της κλάσης που αναλύουμε (ViewPanel). Οι αναφορές αυτές περιέχουν όλες τις αποθηκευμένες παραμέτρους του αρχείου και για να τις ανακτήσουμε, χρησιμοποιούμε τον γνωστό μας BeamFileHandler και το όνομα του αρχείου μας. Το όνομα του αρχείου το εξάγουμε από το ίδιο το αρχείο, το οποίο είναι αποθηκευμένο στο πεδίο bfr της κλάσης μας και είναι τύπου BeamFileRecord. Αφού λάβουμε τις αναφορές του αρχείου και τις αποθηκεύσουμε στην μεταβλητή refs, λαμβάνουμε τις αναφορές των magnets μόνο και τις αποθηκεύουμε στην μεταβλητή refs2. Για να το κάνουμε αυτό, χρησιμοποιήσαμε την κλάση RefsByType και συγκεκριμένα την μέθοδο αυτής getByType, η οποία παίρνει ως παράμετρο την κλάση που περιγράφει τις αναφορές συγκεκριμένου είδους (εδώ την MagnetReference για τους magnets, η οποία

φυσικά είναι υποκλάση της References). Παίρνοντας μόνο τις αναφορές των magnets, μπορούμε χωρίς πρόβλημα να δημιουργήσουμε αντικείμενο της κλάσης MagnetReferences για τον επιλεγμένο magnet (με το όνομα magRef). Τέλος, καλούμε την μέθοδο editRefsMagnets της κλάσης BeamFileManager (η κλάση αυτή παρέχει βοηθητικές μεθόδους για τον χειρισμό τον αρχείων μας), η οποία θα αλλάξει την αναφορά του επιλεγμένου magnet για το ρεύμα του ώστε να πάρει τιμή ίση με αυτήν που έδωσε ο χρήστης (μεταβλητή current). Η μέθοδος αυτή φαίνεται παρακάτω:

```
public void editRefsMagnets(MagnetReferences magRefs,double value
                        ,BeamFileRecord bfr,String name,Log logger) {
      log=logger;
      magRefs.setCurrent(value);
      BeamFileHandler bfh = StaticEnvironment.getBeamFileHandler();
      try {
            bfh.setFileRef(magRefs.getFileName(), magRefs);
            log.info("Current of magnet "+name+" of beam file "
                        +bfr.getName()+" is changed");
      } catch (SpseaException e) {
            log.error(e, e);
      }
}
```

Το σημαντικό αυτής της μεθόδου είναι η κλήση της μεθόδου setCurrent της κλάσης MagnetReferences, η οποία παίρνει ως παράμετρο την νέα τιμή του ρεύματος και την αποθηκεύει στη δομή MangetReferences. Έτσι, τροποποιούμε το αντικείμενο magRefs, ώστε να έχει την τιμή που έδωσε ο χρήστης (value) ως τιμή ρεύματος. Στη συνέχεια, το μόνο που έχουμε να κάνουμε είναι να εφαρμόσουμε αυτό το magRef στο αρχείο μας. Αυτό το κάνουμε με την μέθοδο setFileRef της κλάσης BeamFileHandler, η οποία παίρνει ως παραμέτρους το όνομα του αρχείου και τις αναφορές (αντικείμενο τύπου References), που θα εφαρμόσει στο αρχείο. Έτσι, το αρχείο μας αλλάζει την τιμή που έχει για το ρεύμα του magnet που επιλέξαμε.

6.2.3 Κατάσταση Εξοπλισμού

- Beans:

Η λογική των κλάσεων της κατηγορίας ακολουθεί την λογική των αντίστοιχων του υποσυστήματος της Διαχείρισης των Αρχείων. Επίσης, εξυπηρετεί και τους ίδιους σκοπούς, δηλαδή την αναπαράσταση κάποιου είδους εξοπλισμού και την δημιουργία του μοντέλου του πίνακα του αντίστοιχου status panel. Η μόνη διαφορά είναι ότι η κατηγορία αυτή οργανώνεται σε κλάσεις (όσες και τα είδη του εξοπλισμού) και την υπερκλάση αυτών, η οποία είναι η StatusBeanImpl, όπως αναφέραμε και στο κεφάλαιο της Σχεδίασης. Έτσι, οι ιδιότητες του bean κάποιου είδους εξοπλισμού χωρίζονται σε 2 επίπεδα: τις ιδιότητες της υποκλάσης, οι οποίες είναι ειδικές για το συγκεκριμένο είδος εξοπλισμού και τις ιδιότητες

της υπερκλάσης, τις οποίες έχουν όλα τα είδη εξοπλισμού. Για παράδειγμα, χαρακτηριστικά της υπερκλάσης είναι η κατάσταση του εξοπλισμού και ο τρόπος που την αλλάζουμε αλλά και την ανακτούμε όποτε αυτό χρειαστεί (π.χ. στους highlighters).

Έτσι, έχουμε το interface StatusBean, το οποίο είναι υπερκλάση όλων των υπόλοιπων interfaces (των διαφόρων ειδών εξοπλισμού):

```
public interface StatusBean {
	//bean properties
	public static final String COMMENT_PROPERTY ="comment";
	public static final String BUSY_PROPERTY="busy";
	public static final String STATE_PROPERTY="state";
	//getter methods
	public String getComment();
	public boolean isBusy();
	public int getState();
}
```

Βλέπουμε, δηλαδή, ότι οι κοινές ιδιότητες όλων των ειδών εξοπλισμού είναι οι ιδιότητες comment (σχόλιο), busy (απασχολημένο) και state (κατάσταση). Το πρώτο πεδίο μας δίνει την στήλη των πινάκων "Comment" η οποία έχει διάφορες πληροφορίες για τα μηχανήματα ή πιθανά μηνύματα σφάλματος και υπάρχει σε κάθε πίνακα εξοπλισμού. Τα δύο άλλα, μας δίνουν την κατάσταση του εξοπλισμού που χρησιμοποιείται από τους highlighters. Το μεν busy είναι μία λογική τιμή που γίνεται αληθής μόνο όταν κάποιο μηχάνημα είναι απασχολημένο, ενώ το state αποκτά ακέραιες τιμές που αναπαριστούν όλες τις άλλες καταστάσεις (σφάλμα, προειδοποίηση κλπ).

Αυτές τις ιδιότητες υλοποιούμε στην υπερκλάση όλων των υλοποιήσεων των interfaces (δηλαδή όλων των μελών της κατηγορίας Bean). Η υπερκλάση αυτή (που όπως είπαμε είναι η StatusBeanImpl), περιέχει την κωδικοποίηση των καταστάσεων σε ακεραίους και την μέθοδο getState():

```
//state constants
protected final static int STATE_OK              =  0;
protected final static int STATE_NOT_UPDATED     =  1;
protected final static int STATE_PROPERTY_CHANGE =  2;
protected final static int STATE_ERROR           =  3;
protected final static int STATE_WARNING         =  4;
//the state
private int state=STATE_OK;
public int getState() {
	return state;
}
```

Επίσης, περιέχει έναν ενιαίο τρόπο αλλαγής της κατάστασης που καλείται από τις υποκλάσεις, που είναι η μέθοδος:

```
protected void setState(int newState) {
	switch (newState) {
		case STATE_OK:
			state=STATE_OK; break;
		case STATE_NOT_UPDATED:
			state=STATE_NOT_UPDATED; break;
```

```
            case STATE_PROPERTY_CHANGE:
                  state=STATE_PROPERTY_CHANGE; break;
            case STATE_ERROR:
                  state=STATE_ERROR; break;
            case STATE_WARNING:
                  state=STATE_WARNING; break;
            default:
                  state=STATE_OK; break;
      }
      firePropertyChange(null,null);
   }
```

Η μέθοδος firePropertyChange ενημερώνει τον πίνακα που χρησιμοποιεί αυτήν την κλάση (ή υποκλάσεις της) ως μοντέλο ότι, κάποια τιμή στις ιδιότητες του bean άλλαξε (δεν καθορίζει συγκεκριμένη ιδιότητα, αλλά επιλέγει να ανανεώσει όλες τις ιδιότητες). Έτσι, ο πίνακας ανανεώνει την αντίστοιχη τιμή σε κάποια στήλη ή την εφαρμογή των highlighters. Οι λεπτομέρειες της υλοποίησης της μεθόδου αυτής δεν κρίνεται χρήσιμο να αναφερθούν.

Εντελώς αντίστοιχη είναι η λογική για τις ιδιότητες busy και comment.

Εκτός από την υπερκλάση StatusBeanImpl, που αναλύσαμε, έχουμε και τις υποκλάσεις, οι οποίες αναπαριστούν κάθε είδος εξοπλισμού της δέσμης. Σε αυτές τις κλάσεις, λοιπόν, συμπεριλαμβάνουμε τα ιδιαίτερα χαρακτηριστικά που ισχύουν για τα διάφορα είδη εξοπλισμού (π.χ. το ρεύμα των magnets ή την θέση των taxes) και επίσης χειριζόμαστε κατάλληλα τις κοινές ιδιότητες της υπερκλάσης που αναφέραμε (π.χ. κατάσταση του εξοπλισμού).

Πριν προχωρήσουμε στην ανάλυση των υποκλάσεων καλό είναι να αναφέρουμε την μοντελοποίηση που έχουμε χρησιμοποιήσει για τα διάφορα είδη εξοπλισμού και την κατάστασή τους. Οι αντίστοιχες κλάσεις δεν αναπτύχθηκαν από εμάς και άρα ανήκουν στην κατηγορία General Stuff της παραγράφου 5.1. Και εδώ έχει ακολουθηθεί η οργάνωση των δύο επιπέδων. Δηλαδή, για τα είδη του εξοπλισμού έχουμε την γενική κλάση Equipment, η οποία αναπαριστά κάθε είδους εξοπλισμό και υποκλάσεις αυτής όπως Magnet, Collimator, Tax, Door κλπ, που αναπαριστούν τα επιμέρους είδη. Εντελώς αντίστοιχα, μοντελοποιείται η κατάσταση του εξοπλισμού. Η γενική κλάση Status αναπαριστά την κατάσταση όλου του εξοπλισμού και υποκλάσεις αυτής όπως MagnetStatus, CollimatorStatus, TaxStatus κλπ, την κατάσταση συγκεκριμένων ειδών εξοπλισμού. Επίσης, και για τον εξοπλισμό και για την κατάσταση αυτού, υπάρχουν και ενδιάμεσες κατηγορίες δηλαδή υποκλάσεις των Equipment και Status που περιέχουν άλλες υποκλάσεις κ.ο.κ. με τελικές όμως υποκλάσεις τα διάφορα είδη εξοπλισμού (Magnet, Collimator, Tax κλπ) και τις καταστάσεις αυτών (MagnetStatus, CollimatorStatus, TaxStatus κλπ), αντίστοιχα. Συνεπώς, μπορούμε με αυτόν τον τρόπο να ομαδοποιήσουμε τον εξοπλισμό, π.χ. αυτόν που είναι κινητός ή που κινείται με έναν συγκεκριμένο τρόπο (διακριτά, συνεχώς, μηχανικά κλπ).

Επανερχόμενοι στις κλάσεις της κατηγορίας Bean, το interface που περιγράφει ένα συγκεκριμένο είδος εξοπλισμού, για παράδειγμα τους taxes (κλάση TaxBean, που είναι υποκλάση της StatusBean), έχει την εξής μορφή:

```
//bean properties
public static final String PHYSIC_NAME_PROPERTY ="physicName";
public static final String READ_PROPERTY ="read";
public static final String BEAM_REF_PROPERTY ="beamRef";
public static final String MIN_PROPERTY ="min";
public static final String MAX_PROPERTY ="max";
public static final String ACTUAL_RANGE_PROPERTY ="actualRange";
public static final String SELECTED_RANGE_PROPERTY ="selectedRange";
//getter methods
public String getPhysicName();
public Double getRead();
public Double getBeamRef();
public Double getMin();
public Double getMax();
public String getActualRange();
public String getSelectedRange();
```

Οι ιδιότητες του bean αυτού, όπως το όνομα (PhysicName), η τιμή της θέσης του tax που διαβάζεται (Read), η αναφορική τιμή της θέση (BeamRef) κλπ, αποτελούν τις στήλες του πίνακα του panel κατάστασης για τους taxes (ο οποίος βέβαια έχει και την στήλη "Comment", όπως όλα τα panels). Η υλοποίηση αυτών των μεθόδων από τις οποίες λαμβάνουμε τις διάφορες ιδιότητες του bean, βρίσκεται στην κλάση TaxBeanImpl. Εκεί, για παράδειγμα, βλέπουμε πως λαμβάνουμε την πραγματική τιμή για την ιδιότητα "Read" (την αποθηκεύουμε σε ένα πεδίο με το όνομα position):

```
private Double position;
public Double getRead() {
	return position;
}
```

Φυσικά έχει προηγηθεί αρχικοποίηση της τιμής της μεταβλητής στην μέθοδο updateBeanProperties (η οποία καλείται από τον κατασκευαστή της κλάσης), ως εξής:

```
protected void updateBeanProperties(Status status) {
	TaxStatus taxStatus=(TaxStatus) status;
	try {
		DiscretePositionableStatus posStatus =
			taxStatus.getPositionableStatus();
		position = new
			Double(posStatus.getNumericPosition());
		//various other commands
	} catch (Exception ex) {
		log.error(ex, ex);
		setComment(ex.getMessage());
		setState(STATE_NOT_UPDATED);
	}
}
```

Στο παραπάνω τμήμα κώδικα, παρατηρούμε την παρουσία της μεθόδου updateBeanProperties, η οποία είναι αφηρημένη (abstract) στην υπερκλάση StatusBeanImpl και είναι υπεύθυνη για την ανάκτηση και αποθήκευση των ιδιοτήτων κάθε bean. Η μέθοδος αυτή παίρνει ως παράμετρο μία μεταβλητή τύπου Status, που υποδηλώνει την κατάσταση

οποιουδήποτε εξοπλισμού και αφού το μηχάνημα που μας ενδιαφέρει είναι Tax, μετατρέπουμε το αντικείμενο της κλάσης Status σε αντικείμενο της κλάσης TaxStatus (υποκλάση της Status για εξοπλισμό τύπου Tax). Από αυτό, παίρνουμε μια υποκλάση DiscretePositionableStatus (υποδηλώνει την κατάσταση του εξοπλισμού που μετακινούνται σε διακριτές τιμές), η οποία είναι υποκλάση της κλάσης PositionableStatus (κατάσταση εξοπλισμού που μετακινείται), η οποία με την σειρά της είναι υποκλάση της Status (κατάσταση οποιουδήποτε εξοπλισμού). Από την κλάση DiscretePositionableStatus του μηχανήματός μας, λαμβάνουμε την τιμή της θέσης του και την αποθηκεύουμε στο πεδίο position (αυτό επιστρέφουμε σαν ιδιότητα του bean). Εάν υπάρξει κάποια εξαίρεση στην διαδικασία της ανάκτησης της ζητούμενης τιμής, αναφέρουμε το μήνυμα της εξαίρεσης στην κονσόλα του παραθύρου και στην στήλη "Comment" και επιπλέον, θέτουμε την κατάσταση του μηχανήματος που εξετάζουμε σε «μη ανανεωμένη», με τον τρόπο (μέθοδο setState) που έχουμε αναλύσει, έτσι ώστε τα γράμματα του αντίστοιχου στοιχείου του πίνακα να γίνουν μπλε (με το highlighting, το οποίο θα αναφέρουμε παρακάτω).

Στην ίδια μέθοδο, εκτός από την ανάκτηση των ιδιοτήτων του bean, μπορούμε να κάνουμε και άλλους χρήσιμους ελέγχους και να ανανεώσουμε το highlighting μας. Π.χ. :

```
if (!taxStatus.isOperational()) {
      setComment(MSG_NOT_OPERATIONAL);
      setState(STATE_WARNING);
}
```

Η μέθοδος isOperational μας επιστρέφει αληθή τιμή μόνο αν το Tax είναι λειτουργήσιμο. Αν μας επιστρέψει false, τότε θέτουμε στην στήλη "Comment" την σταθερά MSG_NOT_OPERATIONAL (η οποία στην κλάση StatusBeanConstants ορίζεται ως "Warning" και συνιστά προσοχή) και επίσης ορίζουμε την κατάσταση ως «προειδοποίηση» (κάνοντας τα γράμματα πορτοκαλί με χρήση του highlighting).

Όμως, όπως έχουμε αναφέρει, οι ιδιότητες ενός bean δεν αρχικοποιούνται απλά κατά την έναρξη του προγράμματός μας αλλά ανανεώνονται σε πραγματικό χρόνο. Έτσι, έχουμε διάφορους Listeners, οι οποίοι δέχονται γεγονότα συγκεκριμένων ειδών από τον πραγματικό εξοπλισμό και ανανεώνουν τις αντίστοιχες τιμές του εξοπλισμού. Οι Listeners ενεργοποιούνται και απενεργοποιούνται μέσα στις μεθόδους subscribe και unsubscribe, αντίστοιχα, των διαφόρων beans. Για παράδειγμα, στους taxes, που εξετάζουμε έχουμε μόνο έναν Listener, ο οποίος ονομάζεται PositionRefPCL και ευθύνεται για την ανανέωση της ιδιότητας positionBeamRef της bean κλάσης μας.

Ο Listener αυτός (PositionRefPCL) βρίσκεται στην κλάση TaxBeanIml, την οποία εξετάζουμε και ορίζεται ως εξής:

```
private class PositionRefPCL implements PropertyChangeListener {
      public void propertyChange(PropertyChangeEvent
                                    propertyChangeEvent) {
            if (isPropertyChangeEventOk(propertyChangeEvent)) {
                  positionBeamRef = (Double)
```

```
                        propertyChangeEvent.getNewValue();
                  firePropertyChange(null, null);
            }
      }
}
```

Παρατηρούμε ότι όταν έρχεται κάποιο γεγονός αλλαγής της αναφορικής τιμής της θέσης, του Tax, τότε καλείται η μέθοδος propertyChange, μέσα στην οποία αλλάζουμε την τιμή του αντίστοιχου πεδίου (positionBeamRef) και ενημερώνουμε το μοντέλο του πίνακα ότι η αντίστοιχη ιδιότητα του bean άλλαξε.

Αντίστοιχη είναι η λογική για όλες τις άλλες κλάσεις της κατηγορίας Bean.

- <u>Εργαλεία:</u>

Στο module Κατάσταση Εξοπλισμού, χρησιμοποιήσαμε τον πίνακα TableExplorer του Framework, όπως βέβαια κάναμε και για το module Αρχεία Δέσμης. Εδώ όμως κάναμε χρήση της δυνατότητας εκτύπωσης που προσφέρει ο πίνακας TableExplorer και που εξηγήσαμε στην παράγραφο 6.2.1, παρέχοντας ειδικό κουμπί εκτύπωσης. Επίσης, ένα άλλο εργαλείο που δεν χρησιμοποιήσαμε στην διαχείριση των αρχείων και το κάναμε εδώ, είναι οι highlighters. Και αυτό γιατί θέλουμε το χρώμα των γραμμάτων και του φόντου στα στοιχεία του πίνακα να μας δείχνουν την κατάσταση του εξοπλισμού (σφάλμα, προειδοποίηση, απασχολημένο κλπ). Η κλάσεις που χρησιμοποιήσαμε ως highlighters βρίσκονται στην κατηγορία Helpers και επεκτείνουν την κλάση AbstractHighlighter του SwingX. Από αυτές τις κλάσεις έχουν ενδιαφέρον ο κατασκευαστής (constructor) της κλάσης και η μέθοδος doHighlight. Για παράδειγμα ο κατασκευαστής του highlighter της κατάστασης σφάλματος είναι:

```
public ErrorHighlighter(){
       HighlightPredicate predicate = new HighlightPredicate() {
            public boolean isHighlighted(Component
                  renderer,ComponentAdapter adapter) {
                        JComponent comp=(JComponent)
                              adapter.getComponent();
                        Object bean = null;
                        if (comp instanceof JXTable) {
                              JXTable table = (JXTable) comp;
                              int row=adapter.row;
                              TableModel tableModel = ((JTable)
                                                comp).getModel();
                              if (tableModel instanceof
                                          AbstractTableAdapter) {
                                    bean = ((AbstractTableAdapter)
            tableModel).getRow(table.convertRowIndexToModel(row));
                              } else {
                                    bean =
            tableModel.getValueAt(adapter.row, adapter.column);
                              }
                        }
                        Node node = (Node) bean;
                        StatusBean s = (StatusBean)
```

```
                                    node.getNodeBean();
                        //Highlight if status is "error"
                        if (s.getState()==STATE_ERROR) {
                          return true;
                        }
                        return false;
                        }
                };
                super.setHighlightPredicate(predicate);
        }
```

Παρατηρούμε ότι στον κατασκευαστή αυτόν, θέτουμε στον highlighter μας ένα "predicate" (αντικείμενο της κλάσης HighlightPredicate του SwingX), δηλαδή μια συνθήκη, η οποία μόνο αν είναι αληθής για ένα οποιοδήποτε node του πίνακα με τον εξοπλισμό, εφαρμόζει το highlighting στο node. Στην αρχή αυτού του predicate, εξάγουμε από τον πίνακά μας το node το οποίο θέλουμε να εξετάσουμε. Στην συνέχεια, λαμβάνουμε το αντικείμενο της αντίστοιχης κλάσης της κατηγορίας Bean (δηλαδή ένα μηχάνημα της δέσμης μοντελοποιημένο ως το interface StatusBean, όπως είδαμε στην προηγούμενη παράγραφο) του node αυτού και ακολούθως, εξετάζουμε αν η συνθήκη του highlighting είναι αληθής. Η συνθήκη που επιλέγουμε για το highlighting είναι, φυσικά, αν ο εξοπλισμός είναι σε κατάσταση σφάλματος. Συνεπώς, πρέπει από το bean να λάβουμε την ιδιότητα "state" του, μέσω της μεθόδου (accessor) getState. Η μέθοδος αυτή επιστρέφει, όπως είδαμε στην προηγούμενη παράγραφο, ένα αναγνωστικό για κάθε κατάσταση (state) ενός bean. Αν το αναγνωριστικό αυτό αντιστοιχεί στην κατάσταση σφάλματος, τότε εφαρμόζουμε το highlighting, δηλαδή η μέθοδος isHighlighted του predicate επιστρέφει αληθή τιμή.

Η μέθοδος doHighlight της ίδιας κλάσης (ErrorHiglighter) είναι η εξής:

```
@Override
protected Component doHighlight(Component component,
      ComponentAdapter adapter) {
            //set fonts' style to Italic
            Font font=((JComponent) component).getFont();
            font=font.deriveFont(Font.ITALIC);
                ((JComponent) component).setFont(font);
            //set fonts' color to Red
            ((JComponent) component).setForeground(new Color(238,
                        0, 0));
            return component;
}
```

Στην παραπάνω μέθοδο ορίζουμε τις ιδιότητες του highlighting, δηλαδή επιλέγουμε να γίνονται τα γράμματα ενός node του πίνακά μας κόκκινα και πλαγιαστά (στυλ Italic) σε περίπτωση που η συνθήκη του highlighting είναι αληθής.

Τέλος, δημιουργούμε και εφαρμόζουμε στον πίνακα τους διάφορους highlighters με την εντολή:

```
Highlighter[] highlighters = new Highlighter[] {new
      PropertyChangedHighlighter(), new WarningHighlighter(), new
      ErrorHighlighter(), new NotUpdatedHighlighter(), new
      BusyHighlighter()};
```

```
getTable().setHighlighters(highlighters);
```

Η μέθοδος getTable επιστρέφει τον πίνακα με τον εξοπλισμό του παραθύρου μας και η μέθοδος setHighlighters εφαρμόζει στον πίνακά μας όλους τους διαθέσιμους highlighters, τους οποίους και κατασκευάζουμε.

- **Ενέργειες:**

Η υλοποίηση των ενεργειών γίνεται και εδώ με τον τρόπο που έχουμε αναλύσει στις προηγούμενες παραγράφους. Έχουμε δηλαδή, για κάθε ενέργεια μία μέθοδο που την χειρίζεται. Ενδιαφέρον, δηλαδή, έχει η υλοποίηση των μεθόδων αυτών αλλά και το γεγονός ότι και πάλι έχουμε οργάνωση δύο επιπέδων.

Έτσι, στην γενική κλάση της κατηγορίας Equipment (Status), έχουμε δυο κοινές ενέργειες για όλον τον εξοπλισμό, οι οποίες ονομάζονται “Get LogBook Info” και “Get Additional Info”. Η πρώτη μας δείχνει την «ιστορία» κάποιου μηχανήματος, δηλαδή τις τροποποιήσεις που έχει υποστεί και τον χρόνο κατά τον οποίο έγιναν αυτές. Η ενέργεια αυτή δεν θα αναλυθεί καθώς δεν ήμασταν εμείς που την αναπτύξαμε. Η δεύτερη επίσης δεν θα αναλυθεί καθώς δεν παρουσιάζει κάποιο ενδιαφέρον (απλά παρουσιάζει κάποιες πληροφορίες για το επιλεγμένο μηχάνημα).

Εκτός από αυτές τις δύο ενέργειες, υπάρχουν και άλλες για τα επιμέρους είδη εξοπλισμού και συνεπώς ανήκουν στις υποκλάσεις της κατηγορίας Equipment. Παράδειγμα τέτοιας ενέργειας είναι η “Move In” (Μετακίνηση εντός) του εξοπλισμού τύπου Scintillators, η οποία ανήκει στην κλάση TriggerStatus. Η μέθοδος που την χειρίζεται είναι η εξής:

```
public void moveIn() {
	//get selected rows
	int row[] = table.getSelectedRows();
	int len = row.length;
	if (len > 0) {
		for (int i = 0; i < len; i++) {
			//get selected node
			Node node = (Node) ((NodeTableAdapter)
				table.getModel()).getRow
					(table.convertRowIndexToModel(row[i]));
			TriggerBeanImpl bean = (TriggerBeanImpl)
					node.getNodeBean();
			InOutPositionable inOutPositionable =
				(InOutPositionable) bean.getEquipment();
			//perform action to equipment
			try {
				inOutPositionable.setPositionIn();
			} catch (SpseaException ex) {
				logger.error(ex);
				messageHandler.showError(ex);
			}
		}
	} else {
		statusLine.warn(MSG_SELECTION);
	}
```

```
}
```

Στον παραπάνω κώδικα, το πρώτο πράγμα που κάνουμε είναι να λάβουμε τα επιλεγμένα στοιχεία του πίνακά μας και για κάθε ένα από αυτά εφαρμόζουμε την ενέργειά μας (επανάληψη του βρόχου for). Συγκεκριμένα, λαμβάνουμε το bean που αναπαριστά το στοιχείο του πίνακα, που εδώ είναι ένα αντικείμενο της κλάσης TriggerBeanImpl. Στη συνέχεια, ανακτάμε από το bean αυτό, το αντίστοιχο μηχάνημα. Το μηχάνημα είναι είδους Scintillator, αλλά αρκεί στην περίπτωσή μας να λάβουμε τύπο εξοπλισμού InOutPositionable, που είναι υπερκλάση της κλάσης Scintillator και συμβολίζει όλα τα μηχανήματα που μπορούν να μετακινηθούν μέσα και έξω. Στην συνέχεια, εφαρμόζουμε την ενέργεια "Move In" στον επιλεγμένο εξοπλισμό με την μέθοδο (handler) setPositionIn(). Αν υπάρξει εξαίρεση στην διαδικασία εφαρμογής της ενέργειας, αναφέρουμε το σφάλμα στην κονσόλα αλλά και σε ένα μήνυμα που εμφανίζεται σε ξεχωριστό αναδυόμενο παράθυρο (pop-up window).

Αντίστοιχες μέθοδοι υπάρχουν για όλα τα είδη εξοπλισμού, ενώ σε μερικά από αυτά χρησιμοποιούμε και κλάσεις της κατηγορίας Control.

6.2.4 Launcher

- <u>Διαμόρφωση Framework:</u>

Όπως έχουμε αναφέρει και στις Απαιτήσεις και στην Σχεδίαση του συστήματός μας, το υποσύστημα Framework, που είναι η καρδιά της εφαρμογής μας, πρέπει να είναι αφηρημένο (abstract) ώστε να είναι κατάλληλο ως βάση όχι μόνο για την εφαρμογή την δική μας (CESAR) αλλά και για οποιαδήποτε άλλη εφαρμογή, που παρουσιάζει παρόμοια χαρακτηριστικά. Έτσι, το Framework έχει μόνο γενικά στοιχεία, τα οποία τα διαμορφώνουμε κατάλληλα για το CESAR μέσω του υποσυστήματος Launcher. Αυτή η διαμόρφωση γίνεται με την βοήθεια του εργαλείου Spring (όχι για όλες τις κλάσεις μας αλλά για επιλεγμένες). Συγκεκριμένα, κατασκευάζουμε αρχεία στα οποία έχουμε εντολές γλώσσας XML, με την βοήθεια των οποίων παρέχουμε την διαμόρφωση που περιγράψαμε. Αυτό που κάνουμε ακριβώς, μέσα σε αυτά τα XML αρχεία, είναι να ορίζουμε διάφορα αντικείμενα (beans), τα οποία και διαμορφώνουμε. Στη συνέχεια, το Spring διαβάζει αυτά τα αρχεία και κατασκευάζει άλλα που περιέχουν όλες τις πληροφορίες που έχουμε περιγράψει στα αρχικά αρχεία. Τα νέα αρχεία ονομάζονται contexts. Ταυτόχρονα, το Spring κατασκευάζει αντικείμενα των διαφόρων beans που έχουμε ορίσει (η προκαθορισμένη λειτουργία του είναι να φτιάχνει μοναδικό αντικείμενο για κάθε κλάση που αντιστοιχεί σε bean – singleton beans).

Τα contexts είναι ιδιαίτερα σημαντικά και μπορούμε να έχουμε ένα ή περισσότερα στα πλαίσια της εφαρμογής μας. Καλό είναι, βέβαια, να χωρίζουμε με λογικό τρόπο την εφαρμογή μας σε contexts, ώστε να τα χειριζόμαστε ευκολότερα. Για παράδειγμα, εμείς

χρησιμοποιήσαμε 3 contexts: ένα για την εκκίνηση της εφαρμογής (startup context), ένα για το κύριο μέρος της (application context) και ένα για τις ενέργειές της (actions context). Το πρώτο είναι ιδιαίτερα μικρό και εμφανίζει ουσιαστικά μόνο το splash screen κατά την εκκίνηση της εφαρμογής. Το δεύτερο περιέχει σημαντικές για το Framework κλάσεις (beans), όπως την Application και είναι μεσαίου μεγέθους. Το τρίτο είναι το μεγαλύτερο και περιέχει ενέργειες τόσο του Framework, όπως πρόσθεση ενός workspace και login, όσο και των επιμέρους modules, όπως άνοιγμα ενός Status ή Beam Files Panel στην ενεργή δέσμη (ενεργό workspace).

Κάθε context μπορεί να παράγεται από ένα ή περισσότερα XML αρχεία. Ένα ή λίγα αρχεία έχουν το πλεονέκτημα ότι έχουν όλες τις πληροφορίες που μας ενδιαφέρουν συγκεντρωμένες και το μειονέκτημα ότι για μεγάλες εφαρμογές τα αρχεία αυτά μπορεί να είναι πολύ μεγάλου μεγέθους (αχανή). Αντίθετα, τα πολλά αρχεία έχουν το πλεονέκτημα ότι είναι μικρά και ευκολοδιάβαστα αλλά και το μειονέκτημα ότι μια συγκεκριμένη πληροφορία που ψάχνουμε, δεν είναι εύκολο να προβλέψουμε σε ποιο αρχείο θα βρίσκεται. Εμείς, λόγω του μεγέθους και της διάρθρωσης της εφαρμογής μας, διαλέξαμε να έχουμε ένα αρχείο για το startup context, ένα για το application context και έξι για το actions context.

Ένα αρχείο XML, το οποίο θα διαβάσει το Spring, πρέπει να έχει την μορφή:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
      "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
      <!-- BEAN_1 -->
      <!-- BEAN_2 -->
      <!-- . . .  -->
      <!-- . . .  -->
</beans>
```

Να θυμίσουμε ότι στην γλώσσα XML, η σύνταξη για τα σχόλια είναι: <!--ΣΧΟΛΙΟ -->. Έτσι, με το παραπάνω τμήμα κώδικα, με τα BEAN_1, BEAN_2 κλπ, εννοούμε ότι στα αντίστοιχα σημεία προσθέτουμε τον ορισμό κάποιων beans (όσων θέλουμε). Κάθε bean έχει την παρακάτω μορφή:

```
<bean id="ONOMA BEAN"
      class="ΚΛΑΣΗ">
            <!—ΙΔΙΟΤΗΤΑ_1 -->
            <!—ΙΔΙΟΤΗΤΑ_2 -->
            <!-- . . . .  -->
            <!-- . . . .  -->
</bean>
```

Στην πρώτη γραμμή δίνουμε το όνομα του bean, ενώ στην δεύτερη την κλάση στην οποία αντιστοιχεί. Αν αυτή η κλάση υλοποιεί ένα interface, τότε έχουμε καθορισμό, μέσω του bean, της υλοποίησης ενός interface. Η διαδικασία αυτή λέγεται Interface Injection και είναι υποπερίπτωση της Dependency Injection. Στις επόμενες γραμμές, δίνουμε τιμές για ορισμένες ιδιότητες του bean. Η διαδικασία αυτή ονομάζεται Setter Injection και είναι επίσης,

υποπερίπτωση της Dependency Injection. Αυτές οι ιδιότητες δεν είναι τίποτα άλλο από πεδία (fields) της αντίστοιχης κλάσης και έχουν την μορφή:

```
<property name="ONOMA ΙΔΙΟΤΗΤΑΣ">
      <!-- ΤΙΜΗ ΙΔΙΟΤΗΤΑΣ -->
</property>
```

Οι τιμές μιας ιδιότητας, ανάλογα με τον τύπο που έχουμε ορίσει για το αντίστοιχο πεδίο, μπορεί να είναι κάποιος αριθμός (double ή int), κάποια συμβολοσειρά (String) ή αναφορά σε κάποια άλλη κλάση, την οποία κωδικοποιούμε επίσης ως bean. Αν είναι ένα από τα δύο πρώτα, τότε για την τιμή της ιδιότητας, γράφουμε:

```
<value>ΤΙΜΗ</value>
```

, όπου ΤΙΜΗ κάποιος αριθμός, π.χ. 1.0, 3 κλπ, ή κάποια συμβολοσειρά, π.χ. “CESAR”.

Αν έχουμε αναφορά σε άλλο bean (το ορίζουμε σε κάποια τοποθεσία), γράφουμε:

```
<ref bean="ONOMA BEAN"/>
```

Αξίζει να αναφέρουμε ότι για τις ιδιότητες ενός bean, πρέπει να έχουμε accessors (setter methods) στα αντίστοιχα πεδία της κλάσης, στην οποία αναφερόμαστε. Μάλιστα, το όνομα της ιδιότητας πρέπει να συμφωνεί με το όνομα της setter method (ώστε το Spring να ξέρει σε ποιο πεδίο αναφερόμαστε), με την εξής λογική: αν έχουμε ένα όνομα για μια ιδιότητα, έστω ONOMA, τότε η setter method στην οποία αντιστοιχεί, πρέπει να ονομάζεται setONOMA.

Επίσης, οι ιδιότητες ενός bean μπορεί να αντιστοιχούν σε πεδία της αντίστοιχης κλάσης ή υπερκλάσης αυτής. Επιπροσθέτως, είναι πασιφανές ότι κάποια πεδία στα οποία έχουμε accessor, μπορούν να οριστούν από κάποιο bean (μέσω τιμής σε ιδιότητα) ή και να μην οριστούν (αρκεί η έλλειψη αρχικοποίησής τους από το Spring να μην προκαλεί κάποια εξαίρεση ή λογική δυσλειτουργία).

Ακόμα, να σημειώσουμε ότι δεν διαλέγουμε όλα τα πεδία μιας κλάσης ως ιδιότητες του αντίστοιχου bean καθώς κάτι τέτοιο δεν θα ήταν χρήσιμο. Διαλέγουμε, για παράδειγμα, τα πεδία που είτε αλλάζουν συχνά ή είναι πιθανό να αλλάξουν στο μέλλον και άρα πρέπει να εντοπίζονται και να αλλάζουν εύκολα. Επίσης, διαλέγουμε συνήθως και τα πεδία των οποίων ο τύπος είναι interface και των οποίων η υλοποίηση πρέπει να αλλάζει συχνά ή αναμένεται να αλλάξει κάποια στιγμή. Όλα αυτά διευκολύνουν την συντήρηση της εφαρμογής μας.

Τέλος, όταν έχουμε αναφορά από ένα bean σε κάποιο άλλο, συνήθως το πεδίο στο οποίο αναφέρεται η αντίστοιχη ιδιότητα είναι τύπου interface και το 2^{ο} αυτό bean είναι μια κλάση που υλοποιεί την interface. Αυτό το κάνουμε ώστε να διαχωρίσουμε την λειτουργικότητα από την υλοποίηση στα προγράμματά μας.

Όλα τα παραπάνω τα εξηγούμε μέσα από ένα παράδειγμα. Είναι αυτονόητο, πως πρέπει οπωσδήποτε να κατασκευάσουμε αντικείμενο της καρδιάς του Framework, δηλαδή της κλάσης Application. Το bean αυτό, το οποίο ανήκει στο application context, όπως θα αναμενόταν, είναι το παρακάτω:

```
<bean id="application"
      class="cern.spsea.gui.application.Application">
      <property name="name">
            <value>CESAR</value>
      </property>
      <property name="version">
            <value>1.0</value>
      </property>
      <property name="actionConfigurer">
            <ref bean="actionConfigurer"/>
      </property>
      <property name="workspaceContainer">
            <ref bean="workspaceContainer"/>
      </property>
      <property name="messageHandler">
            <ref bean="messageHandler"/>
      </property>
      <property name="dialogManager">
            <ref bean="dialogManager"/>
      </property>
      <property name="securityService">
            <ref bean="securityService"/>
      </property>
</bean>
```

Όπως φαίνεται από τον παραπάνω ορισμό, το όνομα του bean είναι "application" και η αντίστοιχη κλάση είναι η Application. Από εκεί και έπειτα, έχουμε τον ορισμό των ιδιοτήτων του bean.

Αρχικά, η ιδιότητα name είναι το όνομα της εφαρμογής και την θέτουμε ίση με "CESAR", ενώ η ιδιότητα version είναι ο αριθμός έκδοσης της εφαρμογής μας και την θέτουμε ίση με 1.0. Από αυτές τις δύο ιδιότητες, γίνεται φανερό το πώς καταφέραμε να υλοποιήσουμε (με την βοήθεια του Spring) την σχεδιαστική απαίτησή μας να είναι το Framework αφηρημένο και να το διαμορφώνουμε ως "CESAR" στο υποσύστημα Launcher.

Στη συνέχεια, έχουμε τον ορισμό κάποιων άλλων ιδιοτήτων, οι οποίες αναφέρονται σε άλλα beans. Για παράδειγμα, η ιδιότητα messageHandler είναι ο διαχειριστής των μηνυμάτων προς το χρήστη. Το πεδίο στο οποίο αναφερόμαστε είναι τύπου MessageHandler, που είναι interface του υποσυστήματος Framework και παρέχει τις εξής βασικές μεθόδους: showError, showWarning, showInformation και showConfirmation. Όλες αυτές οι μέθοδοι παίρνουν ως όρισμα ένα String, που αποτελεί μήνυμα προς το χρήστη και στη συνέχεια το παρουσιάζουν με διαφορετικό κάθε φορά τρόπο. Το bean messageHandler το ορίζουμε, τώρα, ως εξής:

```
<bean id="messageHandler"
class="cern.spsea.gui.application.helpers.impl.MessageHandlerImpl"/>
```

Το MessageHandlerImpl είναι κλάση που υλοποιεί το interface MessageHandler και εκεί μπορούμε πλέον να ορίσουμε το σώμα τον μεθόδων που αναφέραμε και άρα τον τρόπο με τον οποίο εμφανίζονται τα διάφορα μηνύματα. Εμείς, στην εν λόγω κλάση, επιλέγουμε να τα εμφανίζουμε ως pop-up παράθυρα/ διάλογους προς τον χρήστη. Με αυτό το παράδειγμα, γίνεται αντιληπτό πως διαχωρίζουμε την λειτουργικότητα μιας κλάσης (εδώ της Application)

από την υλοποίησή της (εδώ υλοποίηση της διαχείρισης των μηνυμάτων) με την βοήθεια του Spring και συγκεκριμένα του Interface Injection.

Η διαδικασία που αναλύσαμε ονομάζεται, όπως έχουμε αναφέρει, Dependency Injection και είναι περίπτωση του σχεδιαστικού μορφήματος Inversion of Control. Και αυτό γιατί, όπως είδαμε παραπάνω, ο κώδικας που διαμορφώνει το Spring (εδώ ο κώδικας των κλάσεων του Framework που επιλέξαμε), μπορεί, με σωστή χρήση, να μην εξαρτάται από το API του Spring. Είδαμε, δηλαδή, πως ο κώδικας «επιτρέπει» την διαμόρφωσή του από το Spring, παρέχοντάς του μόνο απλές setter μεθόδους. Συνεπώς, μιλάμε για μια αντιστροφή ελέγχου, αφού το Spring πηγαίνει και διαμορφώνει την εφαρμογή μας και όχι το αντίστροφο.

Μπορούμε εδώ να τονίσουμε, ότι το Spring δεν επιβάλλει συγκεκριμένες προγραμματιστικές τεχνικές. Το μόνο που κάνει είναι να ενθαρρύνει κάποιες σωστές προγραμματιστικές τεχνικές, ενώ παράλληλα, προσφέρει πληθώρα λύσεων σε διάφορα προβλήματα που μπορούμε να συναντήσουμε. Συνεπώς την λογική των προγραμμάτων μας την επιλέξαμε εμείς ενώ το Spring μπορεί να χρησιμοποιηθεί και με άλλους τρόπους.

Εκτός από αυτό το παράδειγμα του bean application, αξίζει να αναφέρουμε και κάποια άλλα. Ενδιαφέρον, λοιπόν, παρουσιάζει το πώς οργανώνουμε τα menus και τα toolbars, μέσω του παρόντος υποσυστήματος. Έτσι, αναφέρουμε το τμήμα της διαμόρφωσης που δημιουργεί το menu Workspace:

```
<bean id="menuBar"
      class="cern.spsea.gui.application.actions.ActionGroup">
      <property name="members">
            <list>
                  <ref bean="workspace"/>
                  <ref bean="statusForMenu"/>
                  <ref bean="filesForMenu"/>
                  <ref bean="detectors"/>
                  <ref bean="ea"/>
            </list>
      </property>
      <property name="groupId">
            <value>menuBar</value>
      </property>
</bean>
<bean id="workspace"
      class="cern.spsea.gui.application.actions.ActionGroup">
      <property name="members">
            <list>
                  <ref bean="newWorkspaceAction1"/>
                  <ref bean="print"/>
            </list>
      </property>
      <property name="groupId">
            <value>Workspace</value>
      </property>
</bean>
<bean id="print"
      class="cern.spsea.gui.application.actions.ActionGroup">
```

```
      <property name="members">
            <list>
                  <ref bean="printStatusTableAction1"/>
            </list>
      </property>
      <property name="groupId">
            <value>Print</value>
      </property>
</bean>
<bean id="newWorkspaceAction1"
class="cern.spsea.gui.application.actions.specific.NewWorkspaceAction
" init-method="init">
      <property name="name">
            <value>New Workspace</value>
      </property>
      <property name="iconName">
            <value>Workspaces.gif</value>
      </property>
      <property name="description">
            <value>Add a new workspace</value>
      </property>
      <property name="keyShortcut">
            <value>ctrl N</value>
      </property>
      <property name="workspacesSecurityManager">
            <ref bean="workspacesSecurityManager"/>
      </property>
</bean>
<bean id="printStatusTableAction1"
class="cern.spsea.gui.status.equipment.helpers.PrintStatusTableAction
      " init-method="init"/>
```

Τα παραπάνω beans βρίσκονται σε διαφορετικά XML αρχεία αλλά ανήκουν όλα στο actions context και μας δείχνουν το πώς δομούμε μέρος του menu μας σε υπο-menus και ενέργειες.

Παρατηρούμε ότι τα menus και υπο-menus είναι αντικείμενα της κλάσης ActionGroup. Το ίδιο ισχύει και για τα toolbars και υπο-toolbars. Επίσης, οι ενέργειες, όπως η NewWorkspaceAction και η PrintStatusTableAction, είναι υποκλάσεις της GeneralAction. Για τις ενέργειες έχουμε ήδη αναφέρει αρκετά.

Όσον αφορά τα αντικείμενα της ActionGroup, δηλαδή τα menus, παρατηρούμε ότι έχουν δύο ιδιότητες. Η $1^{η}$ είναι μια λίστα με υπο-menus ή / και ενέργειες, που είναι «παιδιά» του menu αυτού. Η $2^{η}$ είναι ένα αναγνωριστικό (όνομα) για το menu.

Με όλα αυτά υπόψη καθώς και το τμήμα XML που παραθέσαμε προηγουμένως, είναι φανερό πως το γενικό menu της εφαρμογής μας θα έχει τα υπο-menus με τα ονόματα: “Workspace”, “Status”, “Files”, “Detectors” και “EA”. Το menu “Workspace” θα έχει την ενέργεια με όνομα “New Workspace” καθώς και το υπο-menu με το όνομα “Print”, το οποίο με την σειρά του περιέχει την ενέργεια του bean printStatusTableAction1 (έχει όνομα “Print Status Table”). Τα διάφορα menus φαίνονται σε screenshots στο κεφάλαιο του Ελέγχου.

- **Διαμόρφωση Modules:**

Η διαμόρφωση των modules αλλά και η προσάρτησή τους στην εφαρμογή μας γίνεται επίσης μέσω του εργαλείου Spring. Αυτό που κάνουμε είναι να δημιουργούμε ένα bean (και άρα ένα αντικείμενο μιας κλάσης) για κάθε ενέργεια που ανοίγει παράθυρο κάποιου module στην εφαρμογή μας. Το bean το ορίζουμε στο υποσύστημα Launcher, όπως παραπάνω. Η κλάση ανήκει στο υποσύστημα του module. Δεν την βάζουμε στο υποσύστημα Framework γιατί η κλάση αυτή θα έχει σίγουρα dependencies στο module (κατασκευάζει τουλάχιστον ένα αντικείμενο κλάσης του – π.χ. για το module Beam Files, κατασκευάζεται σίγουρα το FilePanel που είναι το κεντρικό panel των αρχείων της δέσμης) και θέλουμε το Framework να μην εξαρτάται από τα modules, όπως έχουμε αναφέρει στη Σχεδίαση του συστήματός μας.

Για παράδειγμα, έχουμε την ενέργεια "Browse Beam Files Filtered By Experiment", όπως αναφέρουμε στις Απαιτήσεις του CESAR. Το bean αυτής της ενέργειας είναι το εξής:

```
<bean id="beamFilesByExperimentPanelAction1"
class="cern.spsea.gui.beamfiles.browse.BeamFilesByExperimentPanelActi
on"   init-method="init">
            <property name="name">
                  <value>Browse Beam Files by Experiment</value>
            </property>
            <property name="iconName">
                  <value>Files.gif</value>
            </property>
            <property name="description">
                  <value>Browse Beam Files of any Experiment</value>
            </property>
            <property name="securityService">
                  <ref bean="securityService"/>
            </property>
      </bean>
```

Οι ιδιότητες που ορίζουμε εδώ είναι της κλάσης GeneralAction του Framework, η οποία είναι υπερκλάση της BeamFilesByExperimentPanelAction (dependencies από τα modules στο Framework επιτρέπεται να έχουμε). Η ιδιότητα name είναι το όνομα της ενέργειας, η ιδιότητα iconName είναι το όνομα του αρχείου στο repository των εικόνων, που διαλέγουμε ως εικονίδιο για την ενέργειά μας, η description είναι η αναλυτική περιγραφή της ενέργειας, την οποία χρησιμοποιούμε ως tooltip και τέλος, η securityService είναι ένα πεδίο τύπου SecurityService (interface του Framework) που, όπως έχουμε αναφέρει στο προηγούμενο κεφάλαιο, μας παρέχει ευκολίες στον καθορισμό των δικαιωμάτων που έχουν οι διάφοροι χρήστες ως προς τις ενέργειες της εφαρμογής. Όσον αφορά αυτήν την τελευταία ιδιότητα, αν αρχικοποιηθεί σε ένα bean (όπως εδώ), τότε η αντίστοιχη κλάση ενέργειας προσθέτει τον εαυτό της στην λίστα με τις κλάσεις που «παρακολουθούν» τις αλλαγές στον ενεργό χρήστη στο CESAR. Το ότι αυτό το πεδίο της κλάσης αρχικοποιείται μέσω του bean σημαίνει ότι η ενέργειά μας δεν είναι διαθέσιμη σε όλους τους χρήστες.

- **Εκκίνηση:**

Η εκκίνηση του προγράμματος CESAR γίνεται κατασκευάζοντας ένα αντικείμενο της κλάσης Application του Framework. Η κλάση αυτή, διαμορφώνεται (και κατασκευάζεται ταυτόχρονα αντικείμενό της) από το εργαλείο Spring μέσω αρχείων γραμμένων σε γλώσσα xml, όπως έχουμε ήδη αναφέρει. Την κατασκευή των contexts την κάνουμε μέσω κώδικα και με την βοήθεια βιβλιοθηκών του Spring στις κλάσεις Main και ApplicationLauncher του υποσυστήματος Launcher.

Τέλος, είναι προφανές ότι χρησιμοποιήσαμε και το Spring αλλά και την κλασσική τακτική δημιουργίας αντικειμένων της Java.

7

Έλεγχος

Στο παρόν κεφάλαιο, κάνουμε μια σύντομη αξιολόγηση του συστήματός μας. Στην παράγραφο 7.1, παρουσιάζουμε τις μεθόδους αξιολόγησης που χρησιμοποιήσαμε, ενώ στην παράγραφο 7.2, παρουσιάζουμε ένα σενάριο λειτουργίας του προγράμματός μας που ελέγχει και αυτό την ορθότητά του.

7.1 Μεθοδολογία ελέγχου

- Unit Testing:

Πρόκειται για τον έλεγχο μικρών κομματιών κώδικα (units) που παρουσιάζουν αρκετά μεγάλη αλγοριθμική πολυπλοκότητα. Έτσι, ελέγχουμε αν σε κάθε δυνατή περίπτωση, το κομμάτι αυτό του κώδικα βγάζει το σωστό αποτέλεσμα.

- Functionality & Usability Testing:

Το Usability Testing έχει ως στόχο να ελέγξουμε αν το CESAR είναι αρκετά φιλικό προς το χρήστη.

Το Functionality Testing έχει ως στόχο τον έλεγχο της ορθής λογικής λειτουργίας του προγράμματός μας, π.χ. να αλλάζει το ρεύμα ενός μαγνήτη στον πραγματικό εξοπλισμό και

να το θέτει όσο επιλέγει ο χρήστης. Γενικά, το πρόγραμμα οφείλει να κάνει ότι ακριβώς έχουμε περιγράψει στις Απαιτήσεις του συστήματός μας.

Για αυτόν τον $2^{ο}$ έλεγχο, θα παρουσιάσουμε στην παράγραφο 7.2 ένα σενάριο λειτουργίας του CESAR, το οποίο θα μας δείχνει όσο πιο πολλές γίνεται από τις λειτουργίες που αυτό πρέπει να επιτελεί. Το σενάριο αυτό έχει ως εξής:

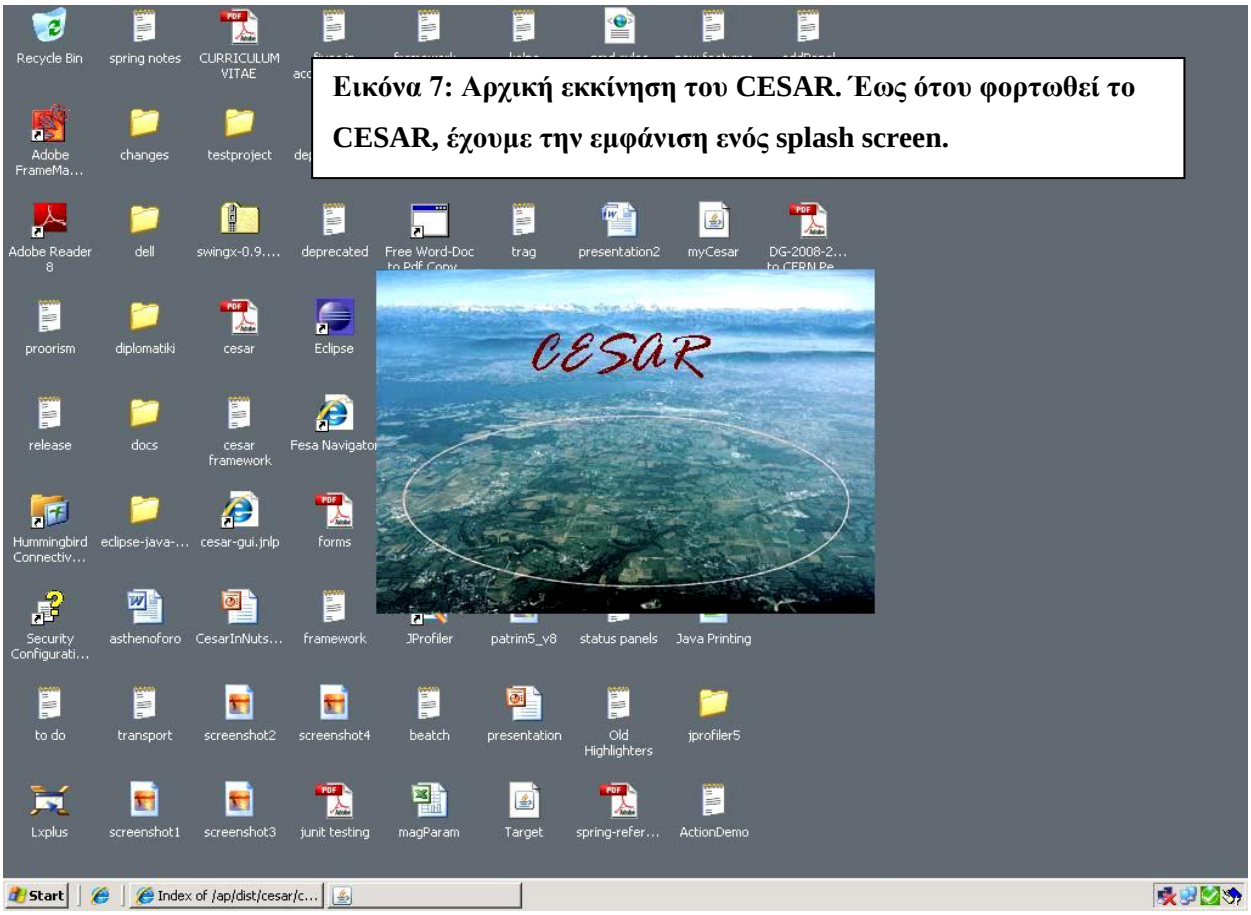
Εκκινείται το CESAR από έναν υπολογιστή, ο οποίος δεν αποτελεί αναγνωρίσιμη τοποθεσία για το σύστημα του CESAR. Παρατηρούμε το γενικό interface της εφαρμογής και ποιες δυνατότητες είναι διαθέσιμες σε αυτό. Κάνουμε εσφαλμένο login. Στην συνέχεια, κάνουμε επιτυχημένο login ως BEAM PARASITIC USER και ανοίγουμε κάποια παράθυρα Αρχείων Δέσμης και Κατάστασης Εξοπλισμού. Παρατηρούμε ποιες δυνατότητες μας είναι διαθέσιμες σε αυτά τα παράθυρα. Κάνουμε login ως BEAM MAIN USER και παρατηρούμε ποιες αλλαγές επέρχονται. Κάνουμε login ως SUPER USER και παρατηρούμε και πάλι ποιες αλλαγές επέρχονται. Εκτελούμε κάποιες δυνατότητες σε αρχεία δέσμης και εξοπλισμό. Τέλος, παρατηρούμε το logging output, το οποίο είναι διαθέσιμο στο χρήστη και κλείνουμε την εφαρμογή.

- <u>Performance Testing:</u>

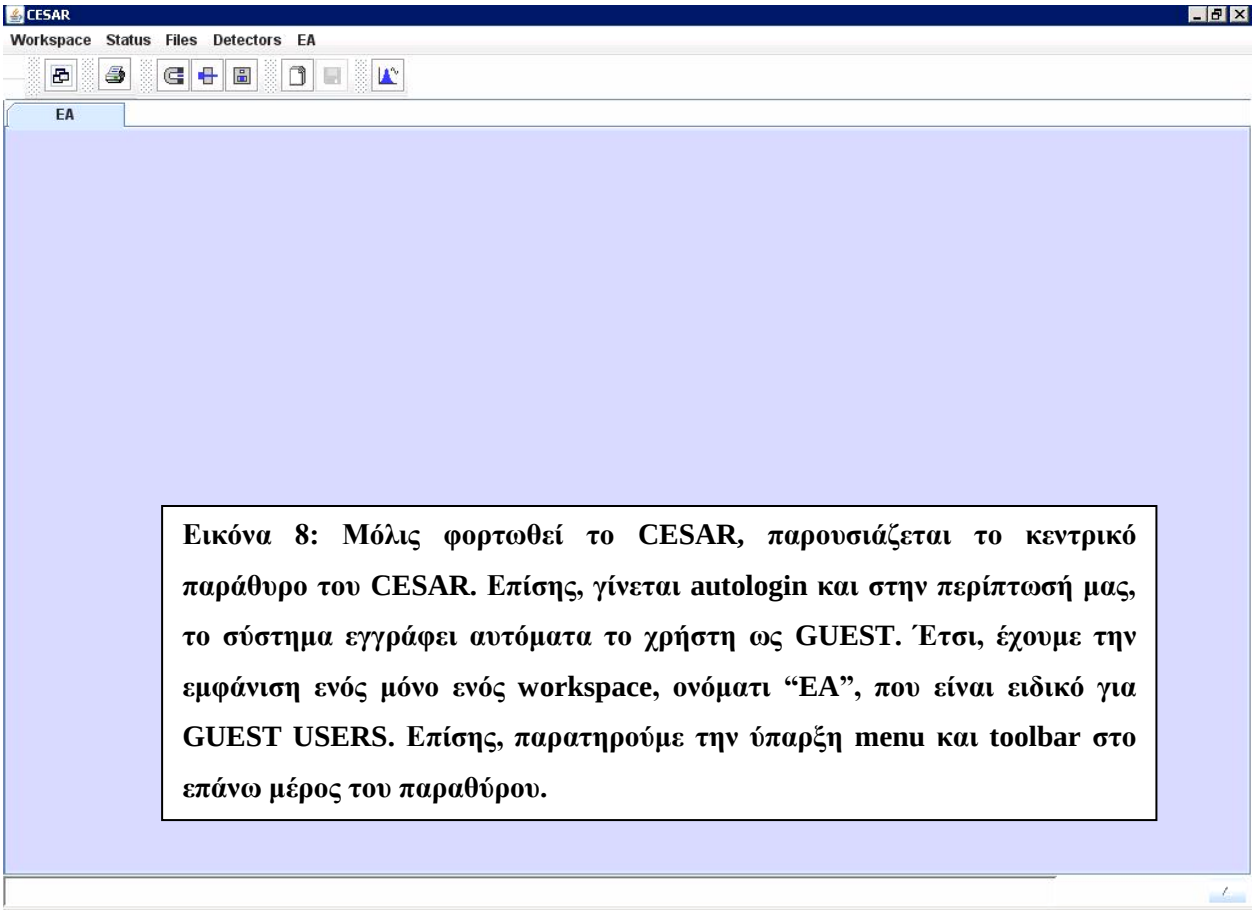
Με χρήση κατάλληλου βοηθητικού λογισμικού (ονόματι JProfiler), κάναμε ανάλυση της χρήσης της CPU και της μνήμης του υπολογιστή στον οποίο τρέχουμε την εφαρμογή μας από την ίδια την εφαρμογή. Κυριότερο βάρος δώσαμε στην χρήση της μνήμης, καθώς είναι πολύ σημαντικό να μην έχουμε διαρροές μνήμης (memory leaks). Και αυτό γιατί, το CESAR μπορεί να μένει ανοιχτό σε έναν υπολογιστή κάποιου πειράματος επί ημέρες.

7.2 Αναλυτική παρουσίαση ελέγχου

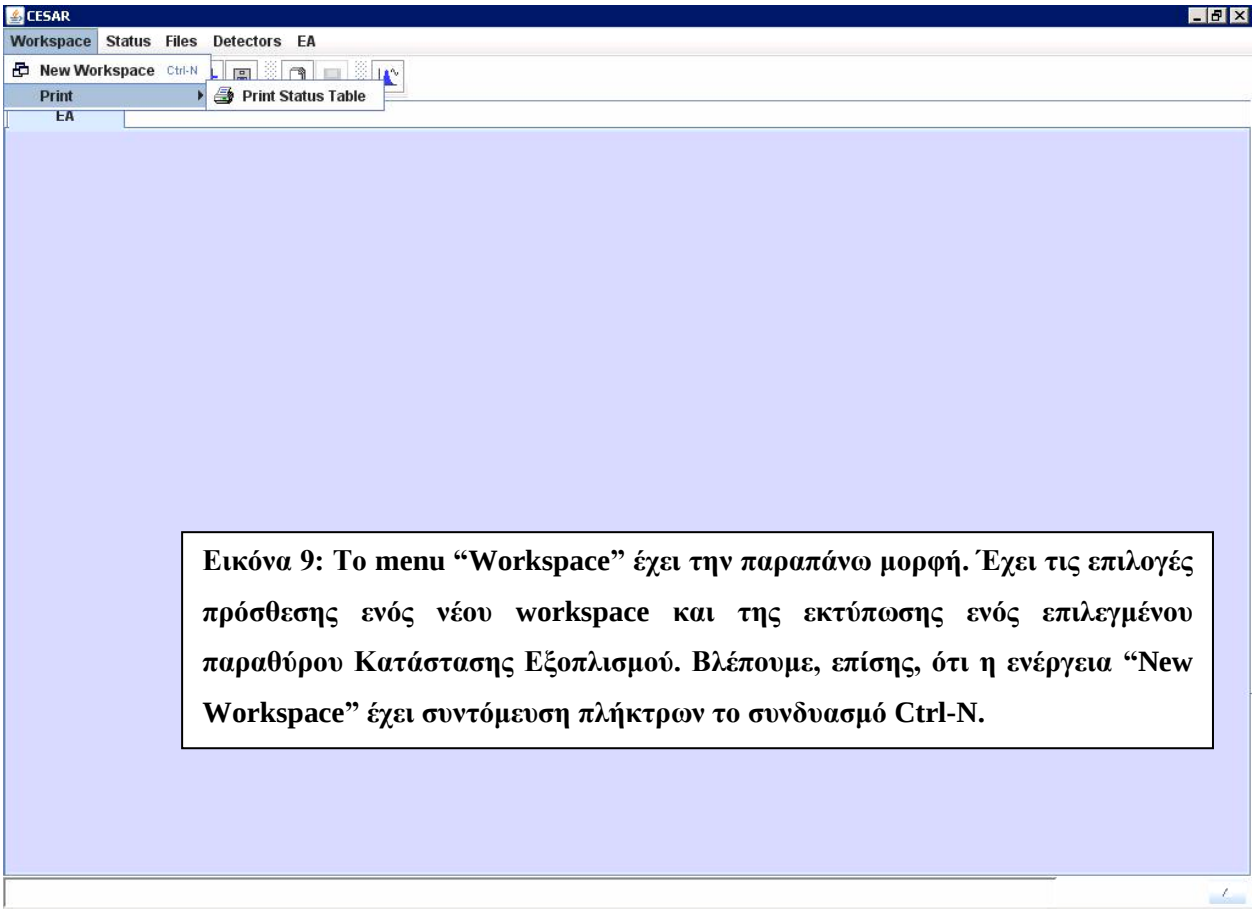
Στην ενότητα αυτή παρουσιάζουμε αναλυτικά τον έλεγχο του συστήματος σύμφωνα με το σενάριο που περιγράφηκε στην προηγούμενη ενότητα, με την βοήθεια αλληλουχίας από screenshots. Έτσι, ελέγχουμε κάποιες από τις σημαντικότερες λειτουργίες του συστήματός μας (όχι όλες διότι είναι πάρα πολλές) και ταυτόχρονα αποκτάμε μία εποπτική εμπειρία του CESAR.



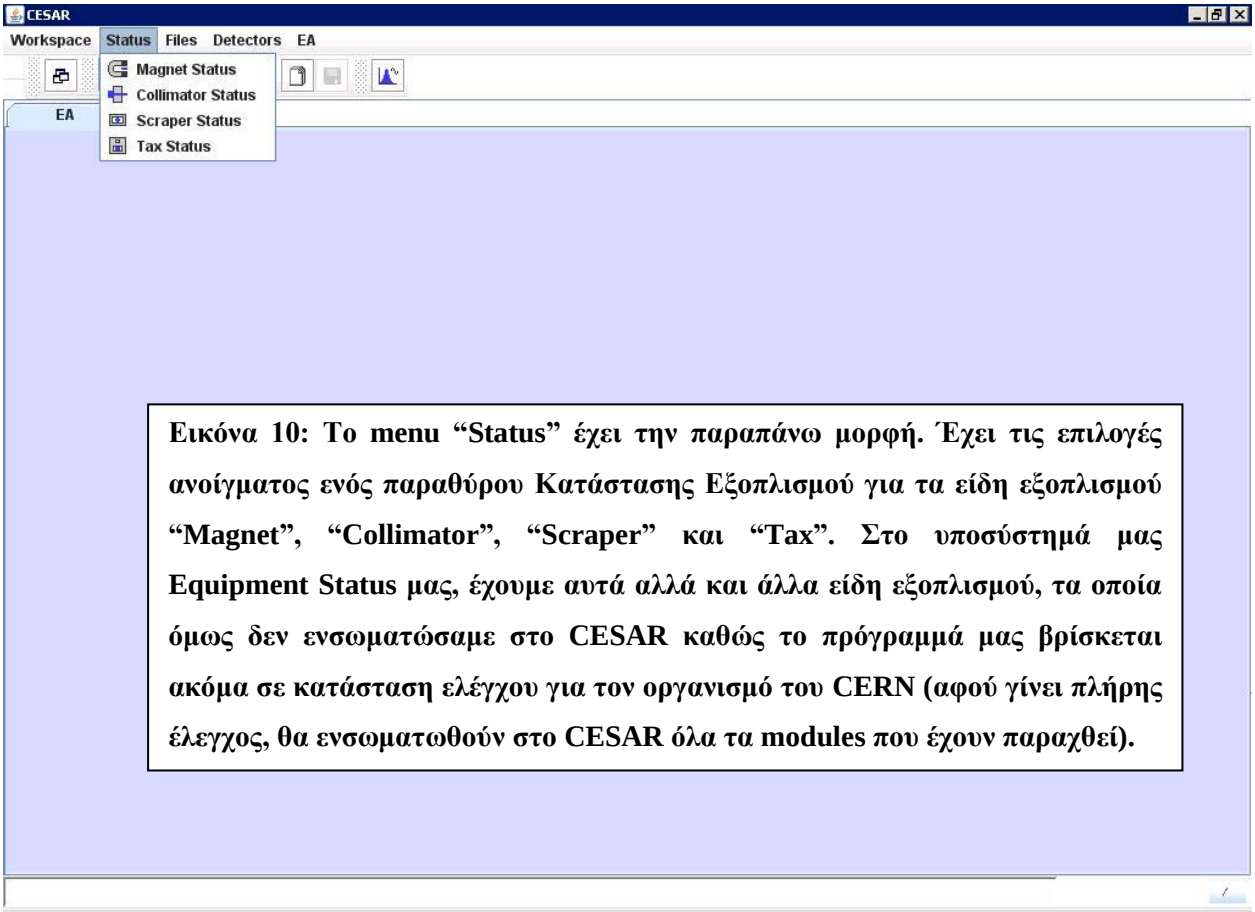
Εικόνα 7: Αρχική εκκίνηση του CESAR. Έως ότου φορτωθεί το CESAR, έχουμε την εμφάνιση ενός splash screen.



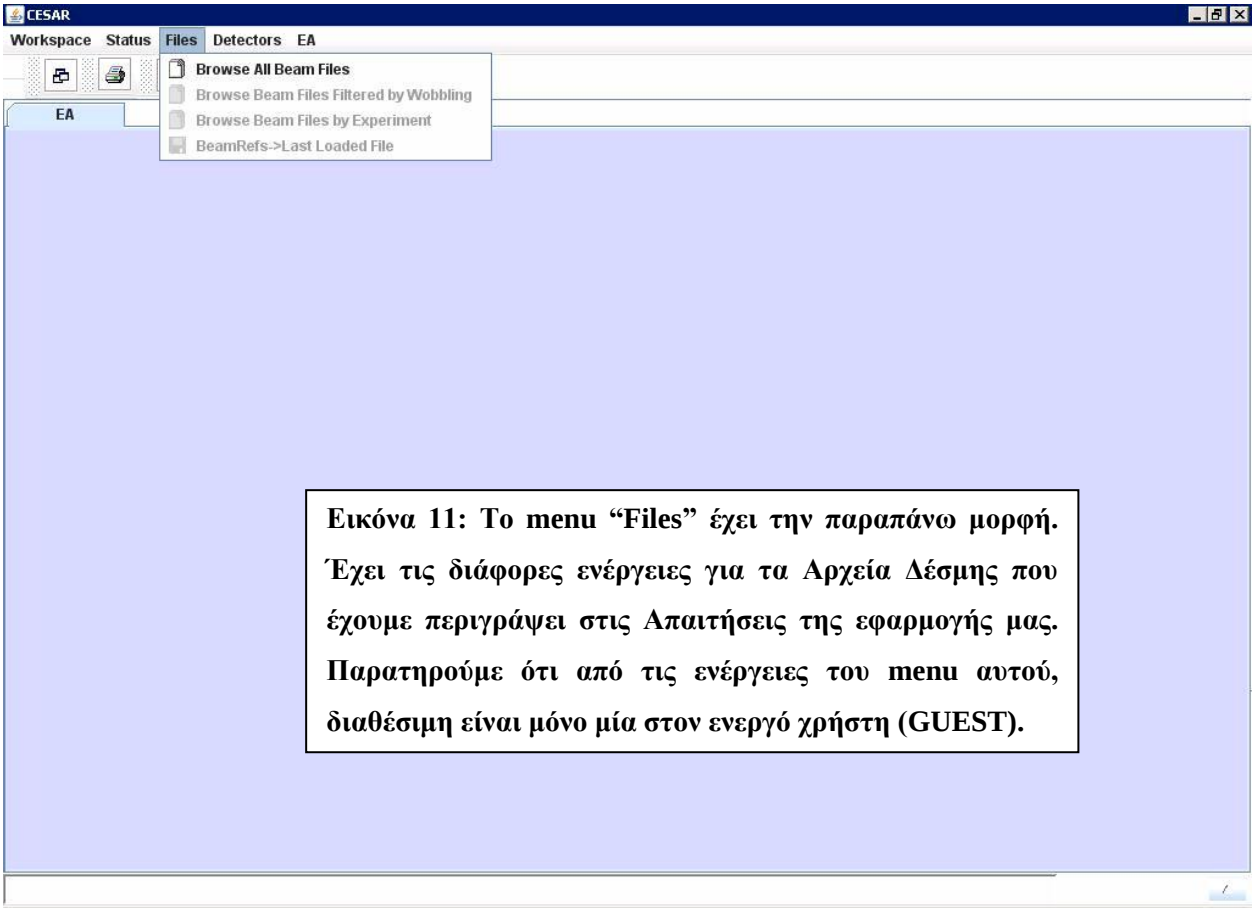
Εικόνα 8: Μόλις φορτωθεί το CESAR, παρουσιάζεται το κεντρικό παράθυρο του CESAR. Επίσης, γίνεται autologin και στην περίπτωσή μας, το σύστημα εγγράφει αυτόματα το χρήστη ως GUEST. Έτσι, έχουμε την εμφάνιση ενός μόνο ενός workspace, ονόματι "EA", που είναι ειδικό για GUEST USERS. Επίσης, παρατηρούμε την ύπαρξη menu και toolbar στο επάνω μέρος του παραθύρου.



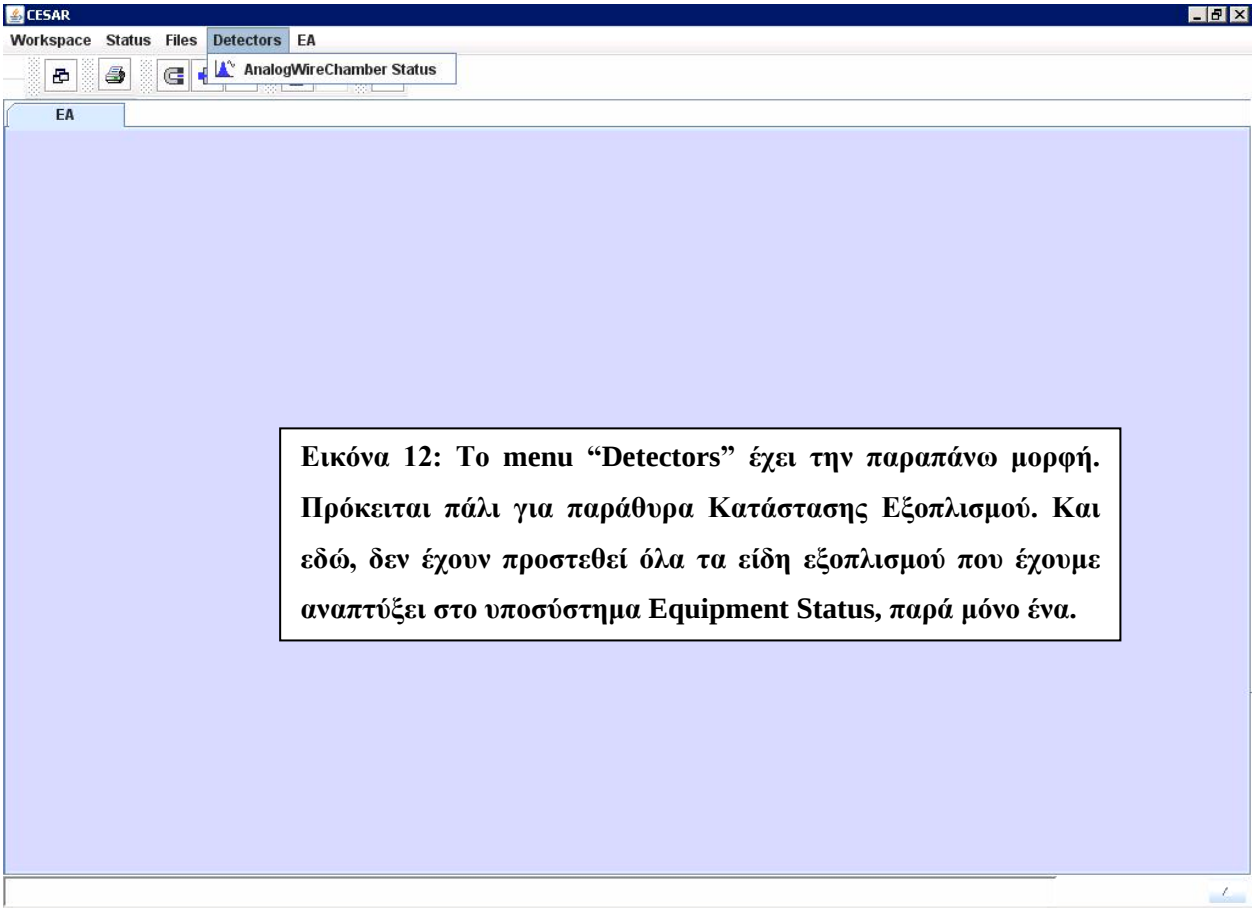
Εικόνα 9: Το menu "Workspace" έχει την παραπάνω μορφή. Έχει τις επιλογές πρόσθεσης ενός νέου workspace και της εκτύπωσης ενός επιλεγμένου παραθύρου Κατάστασης Εξοπλισμού. Βλέπουμε, επίσης, ότι η ενέργεια "New Workspace" έχει συντόμευση πλήκτρων το συνδυασμό Ctrl-N.



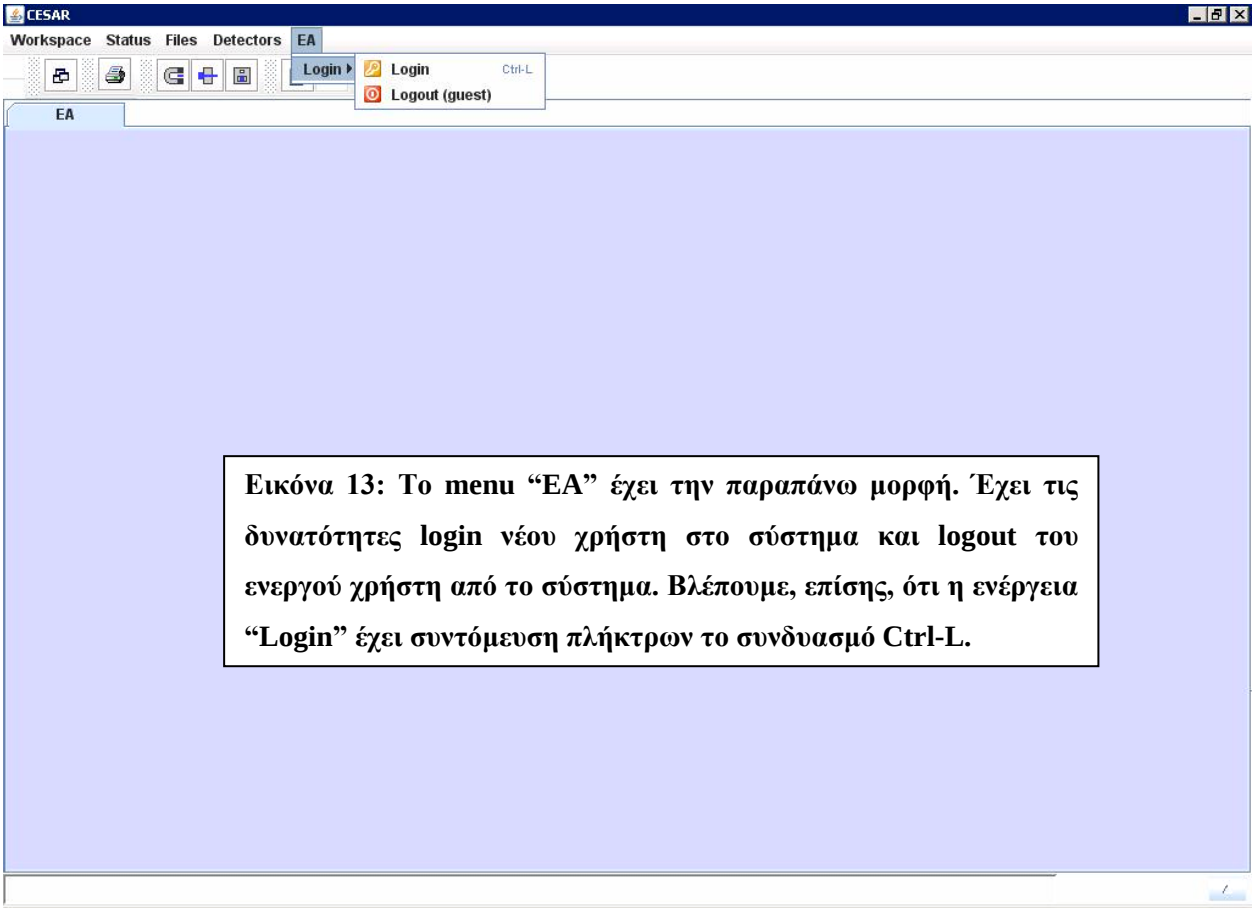
Εικόνα 10: Το menu "Status" έχει την παραπάνω μορφή. Έχει τις επιλογές ανοίγματος ενός παραθύρου Κατάστασης Εξοπλισμού για τα είδη εξοπλισμού "Magnet", "Collimator", "Scraper" και "Tax". Στο υποσύστημά μας Equipment Status μας, έχουμε αυτά αλλά και άλλα είδη εξοπλισμού, τα οποία όμως δεν ενσωματώσαμε στο CESAR καθώς το πρόγραμμά μας βρίσκεται ακόμα σε κατάσταση ελέγχου για τον οργανισμό του CERN (αφού γίνει πλήρης έλεγχος, θα ενσωματωθούν στο CESAR όλα τα modules που έχουν παραχθεί).



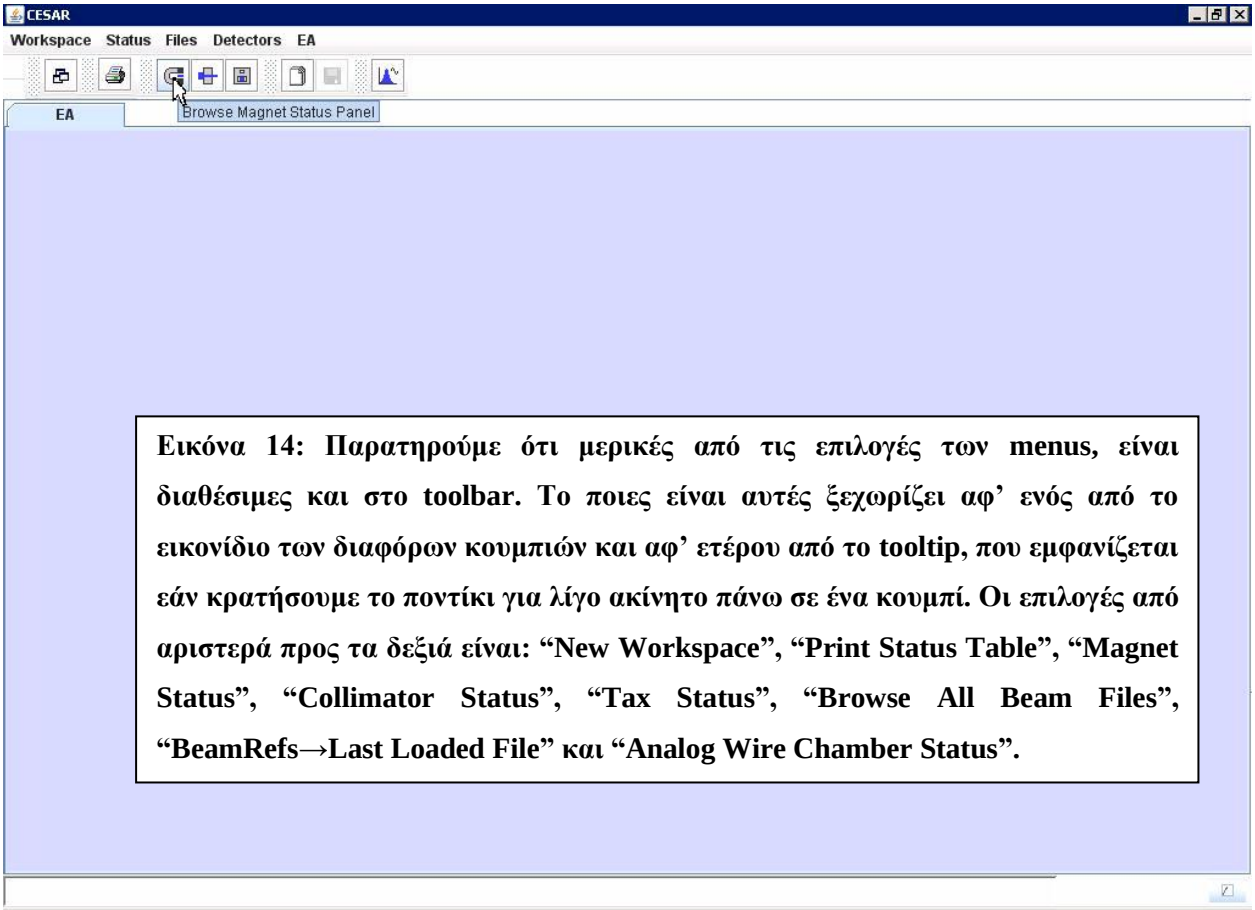
Εικόνα 11: Το menu "Files" έχει την παραπάνω μορφή. Έχει τις διάφορες ενέργειες για τα Αρχεία Δέσμης που έχουμε περιγράψει στις Απαιτήσεις της εφαρμογής μας. Παρατηρούμε ότι από τις ενέργειες του menu αυτού, διαθέσιμη είναι μόνο μία στον ενεργό χρήστη (GUEST).



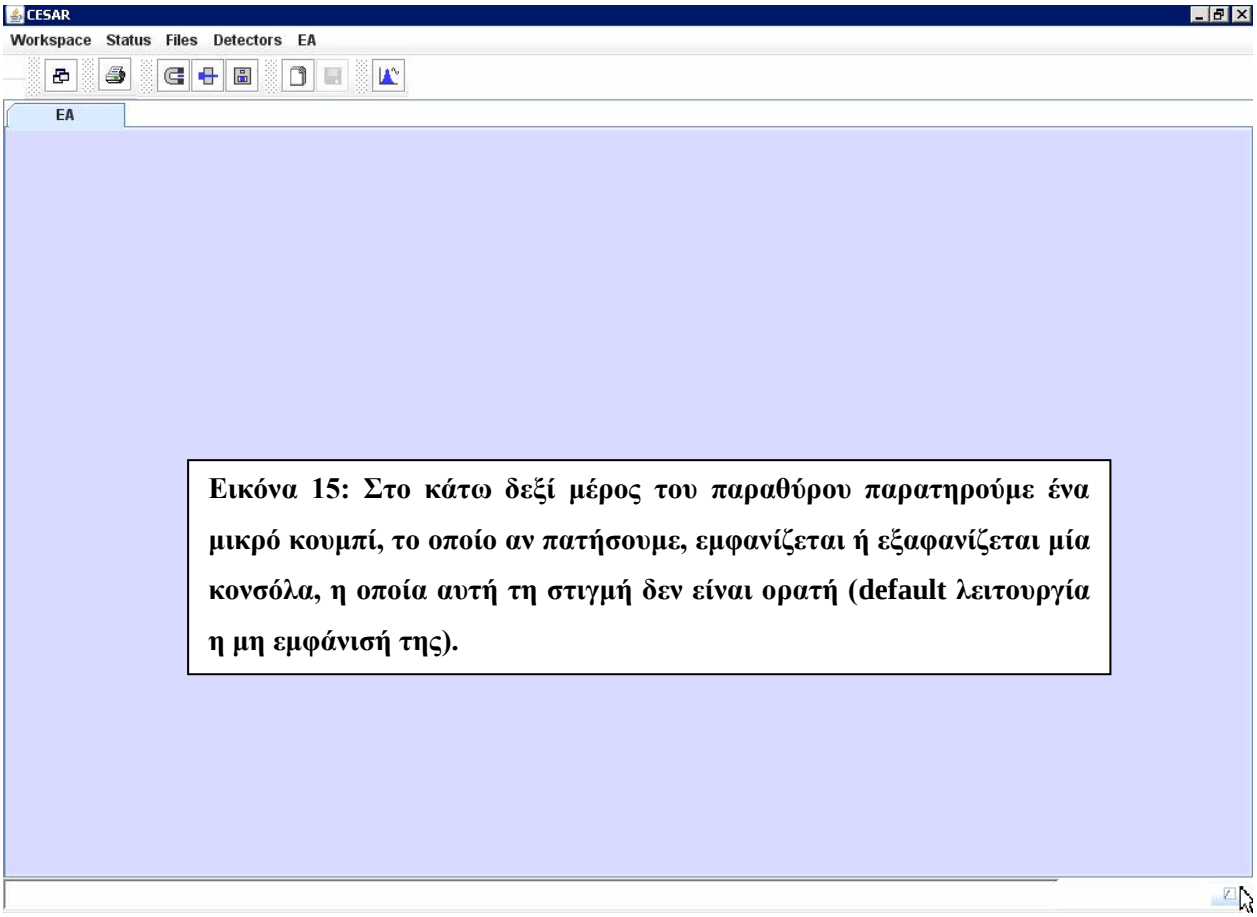
Εικόνα 12: Το menu "Detectors" έχει την παραπάνω μορφή. Πρόκειται πάλι για παράθυρα Κατάστασης Εξοπλισμού. Και εδώ, δεν έχουν προστεθεί όλα τα είδη εξοπλισμού που έχουμε αναπτύξει στο υποσύστημα Equipment Status, παρά μόνο ένα.



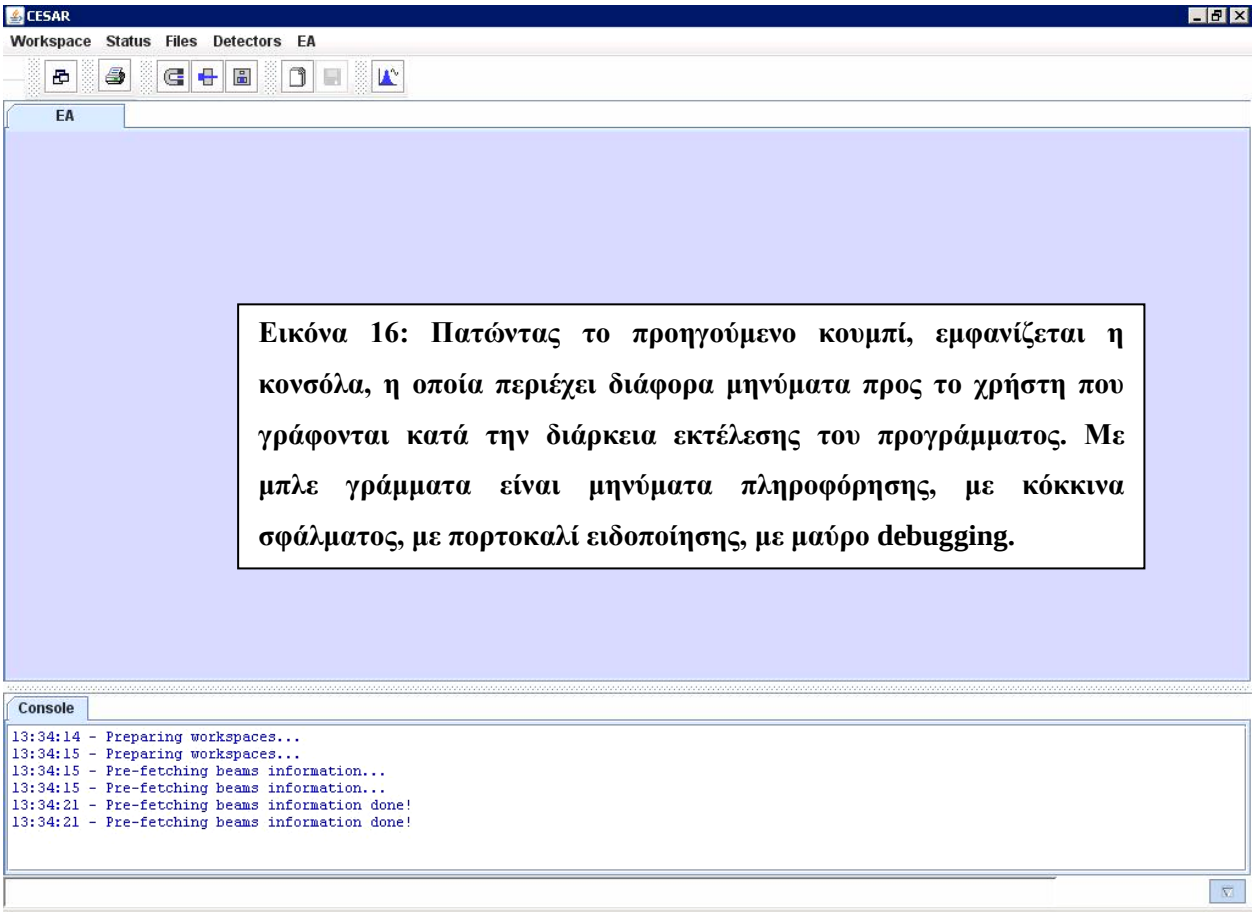
Εικόνα 13: Το menu "EA" έχει την παραπάνω μορφή. Έχει τις δυνατότητες login νέου χρήστη στο σύστημα και logout του ενεργού χρήστη από το σύστημα. Βλέπουμε, επίσης, ότι η ενέργεια "Login" έχει συντόμευση πλήκτρων το συνδυασμό Ctrl-L.



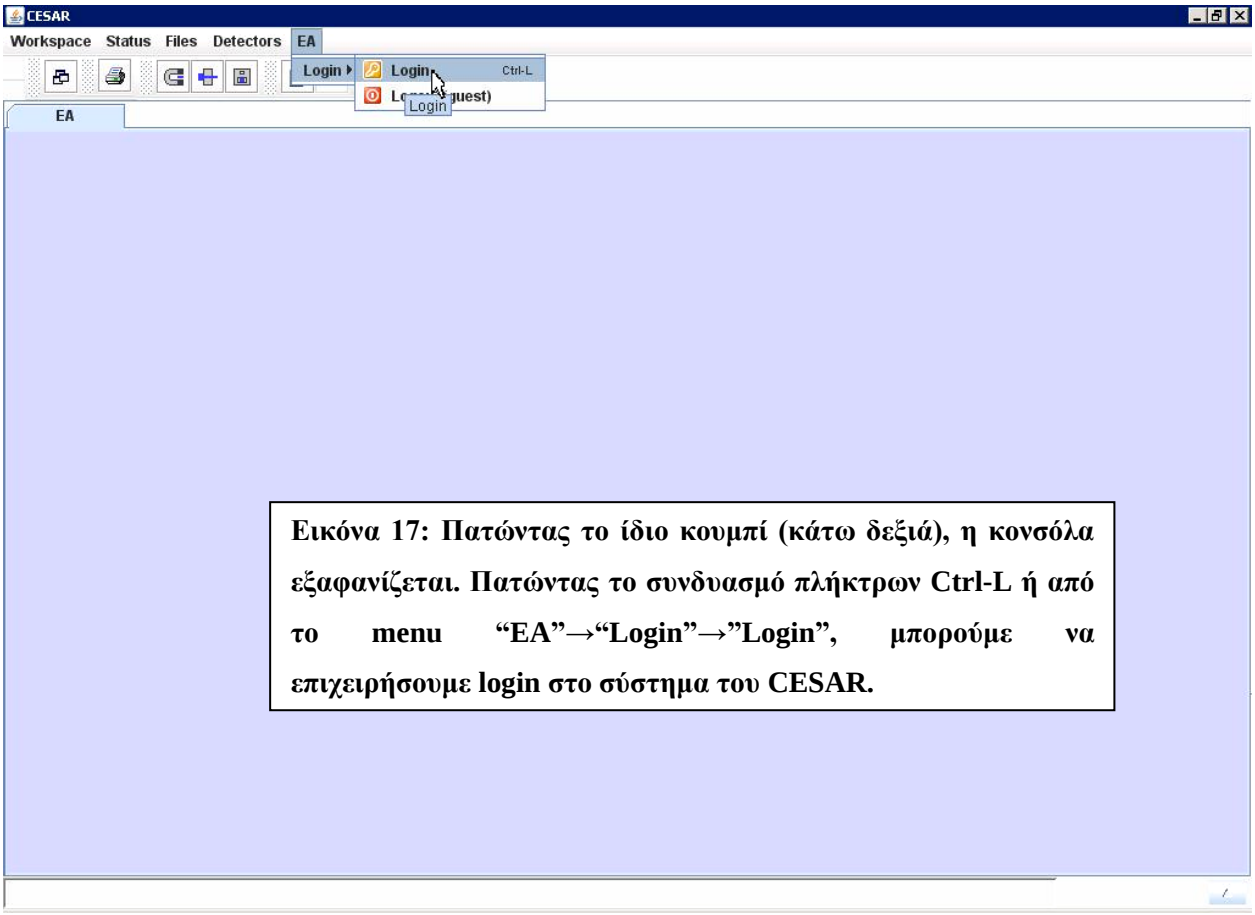
Εικόνα 14: Παρατηρούμε ότι μερικές από τις επιλογές των menus, είναι διαθέσιμες και στο toolbar. Το ποιες είναι αυτές ξεχωρίζει αφ' ενός από το εικονίδιο των διαφόρων κουμπιών και αφ' ετέρου από το tooltip, που εμφανίζεται εάν κρατήσουμε το ποντίκι για λίγο ακίνητο πάνω σε ένα κουμπί. Οι επιλογές από αριστερά προς τα δεξιά είναι: "New Workspace", "Print Status Table", "Magnet Status", "Collimator Status", "Tax Status", "Browse All Beam Files", "BeamRefs→Last Loaded File" και "Analog Wire Chamber Status".



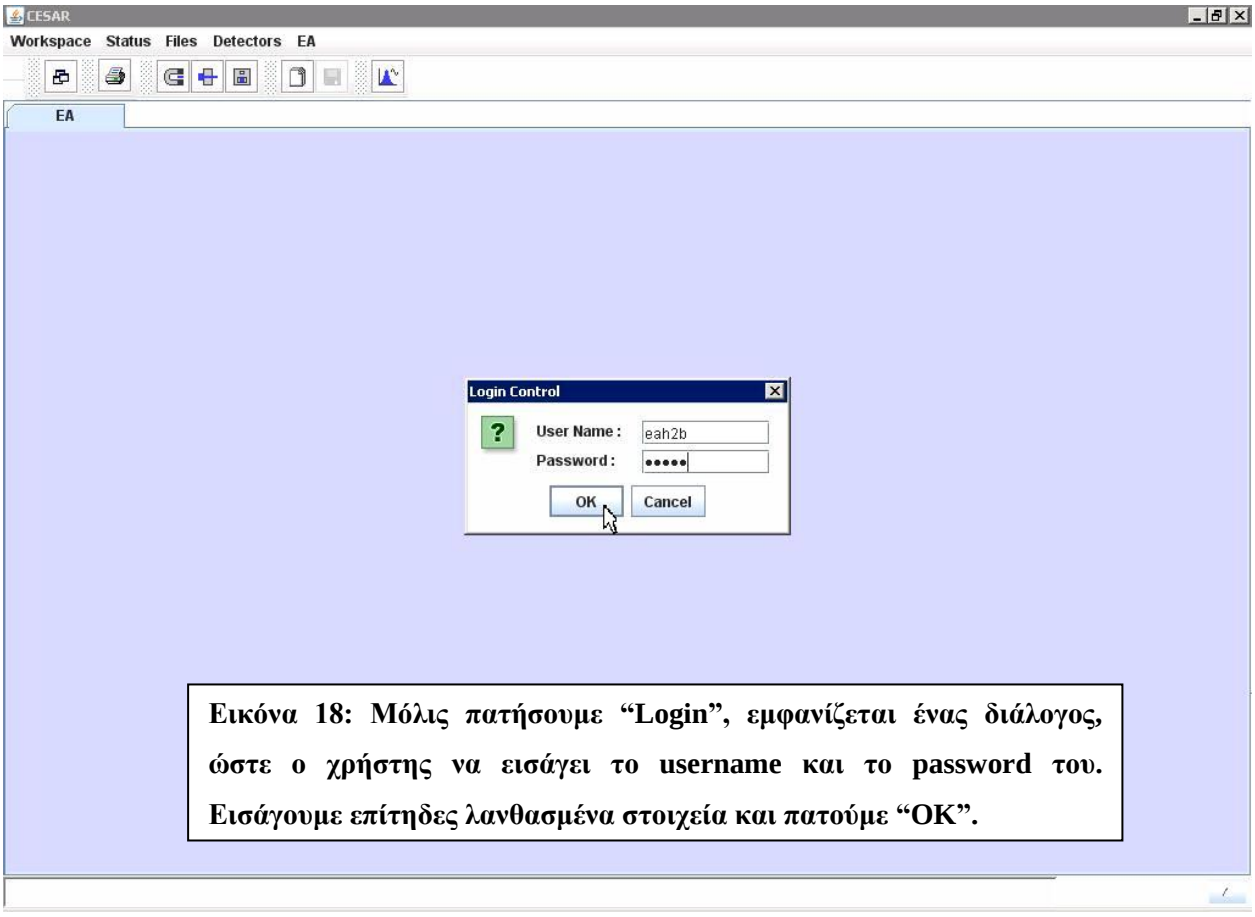
Εικόνα 15: Στο κάτω δεξί μέρος του παραθύρου παρατηρούμε ένα μικρό κουμπί, το οποίο αν πατήσουμε, εμφανίζεται ή εξαφανίζεται μία κονσόλα, η οποία αυτή τη στιγμή δεν είναι ορατή (default λειτουργία η μη εμφάνισή της).



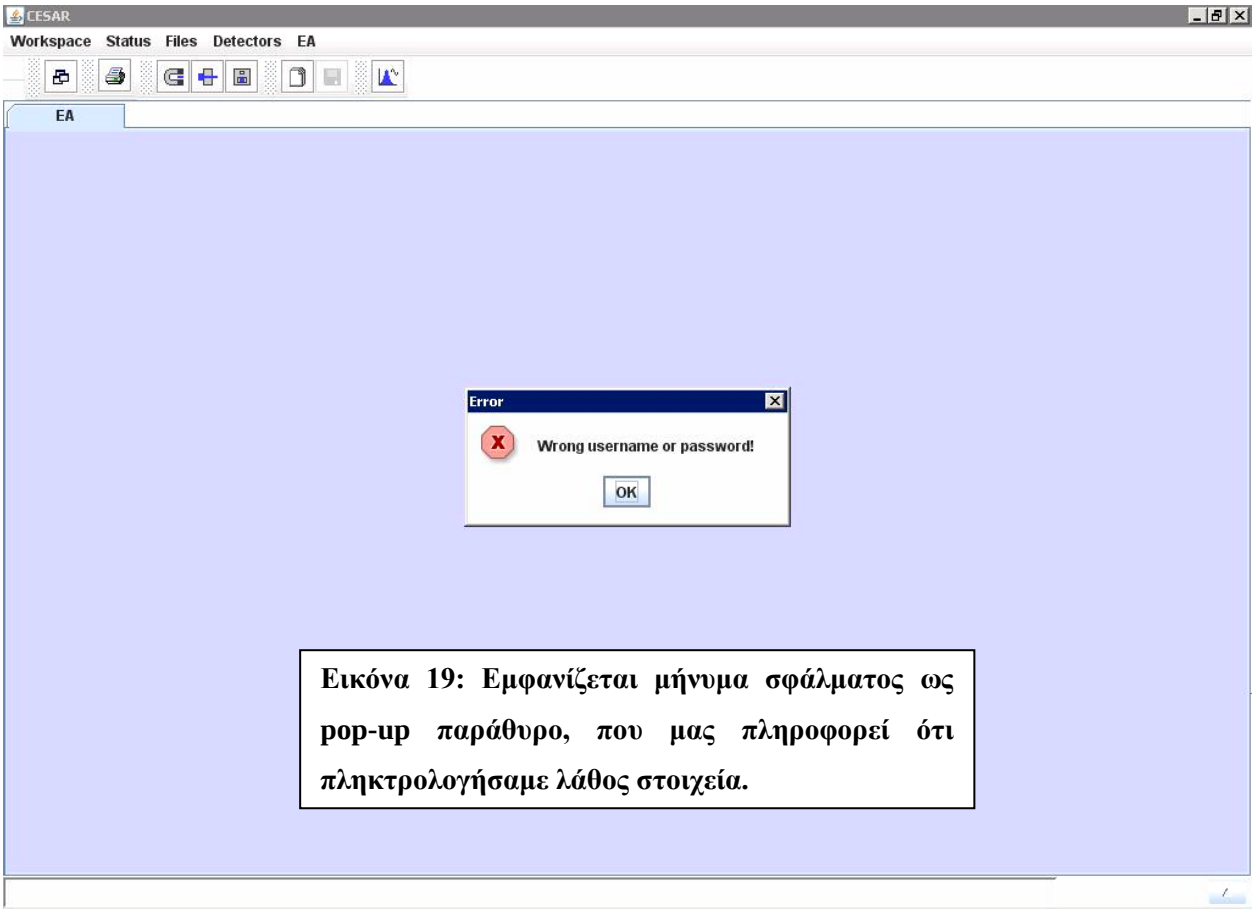
Εικόνα 16: Πατώντας το προηγούμενο κουμπί, εμφανίζεται η κονσόλα, η οποία περιέχει διάφορα μηνύματα προς το χρήστη που γράφονται κατά την διάρκεια εκτέλεσης του προγράμματος. Με μπλε γράμματα είναι μηνύματα πληροφόρησης, με κόκκινα σφάλματος, με πορτοκαλί ειδοποίησης, με μαύρο debugging.



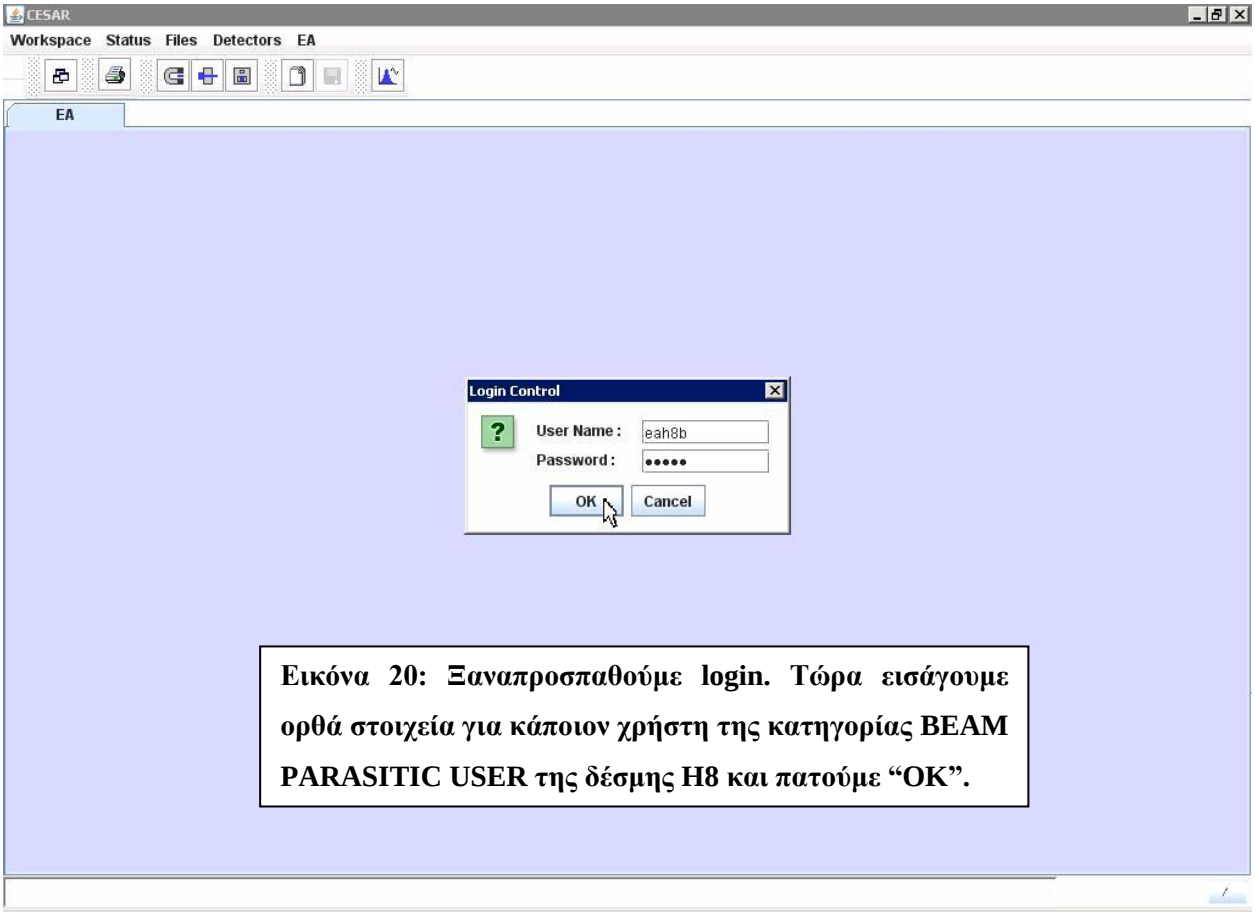
Εικόνα 17: Πατώντας το ίδιο κουμπί (κάτω δεξιά), η κονσόλα εξαφανίζεται. Πατώντας το συνδυασμό πλήκτρων Ctrl-L ή από το menu "EA"→"Login"→"Login", μπορούμε να επιχειρήσουμε login στο σύστημα του CESAR.



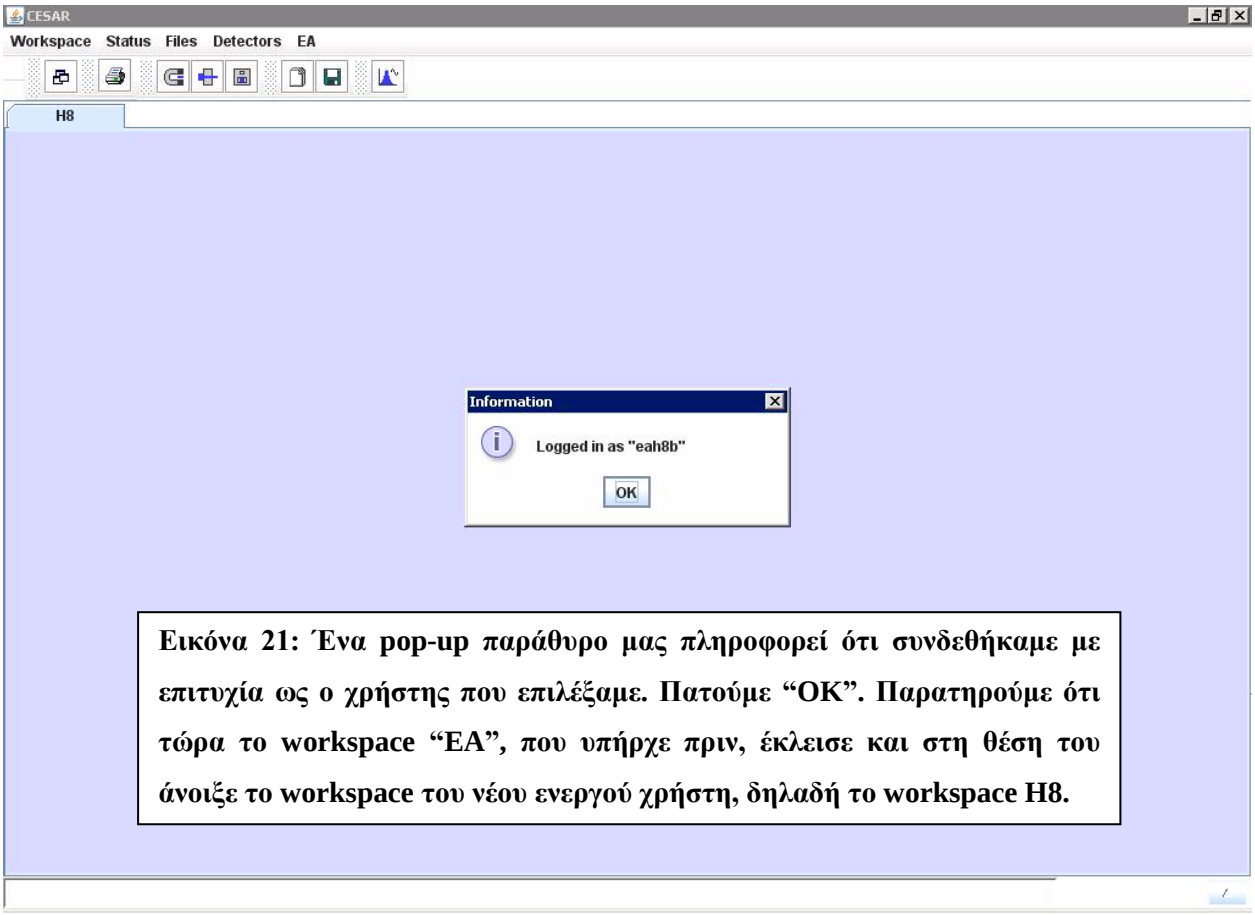
Εικόνα 18: Μόλις πατήσουμε "Login", εμφανίζεται ένας διάλογος, ώστε ο χρήστης να εισάγει το username και το password του. Εισάγουμε επίτηδες λανθασμένα στοιχεία και πατούμε "OK".



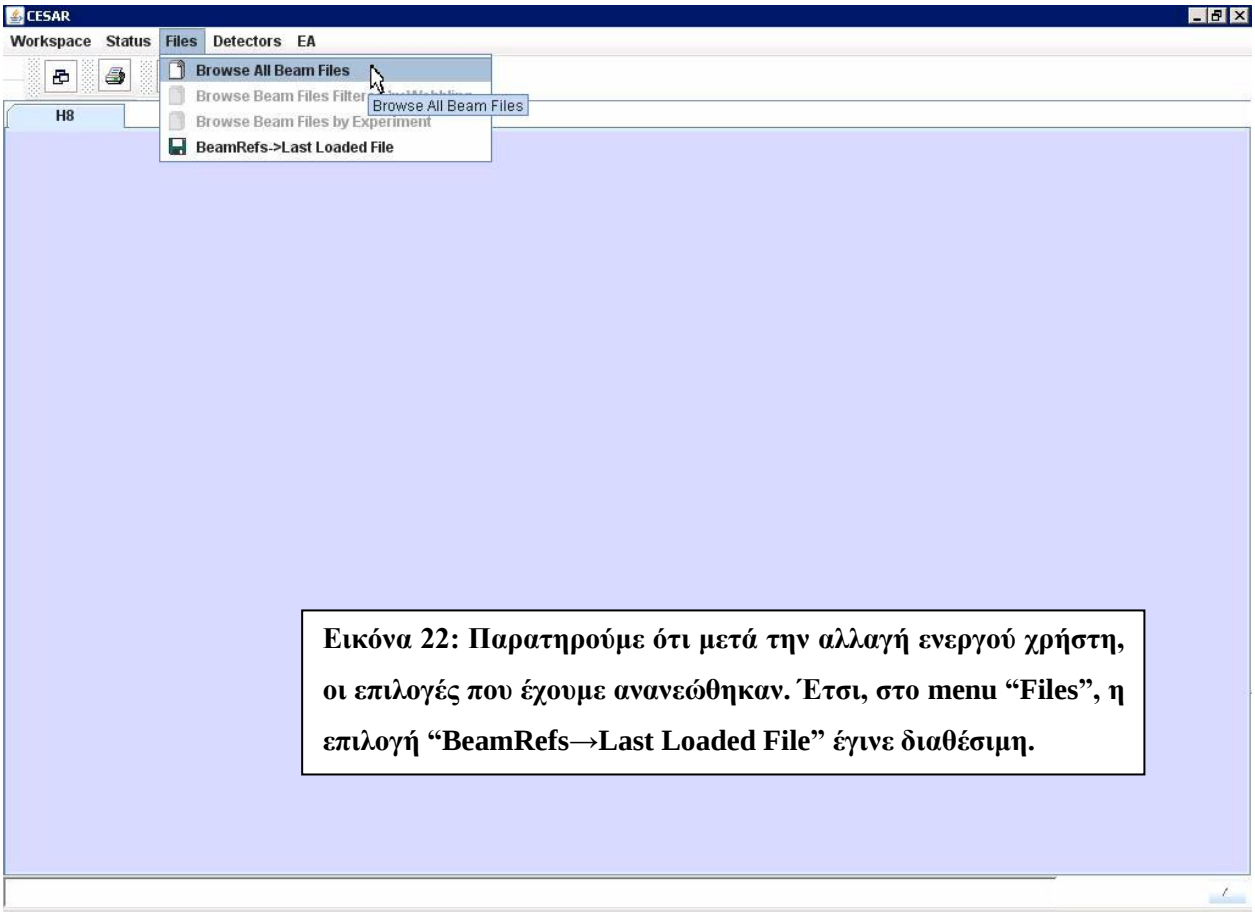
Εικόνα 19: Εμφανίζεται μήνυμα σφάλματος ως pop-up παράθυρο, που μας πληροφορεί ότι πληκτρολογήσαμε λάθος στοιχεία.



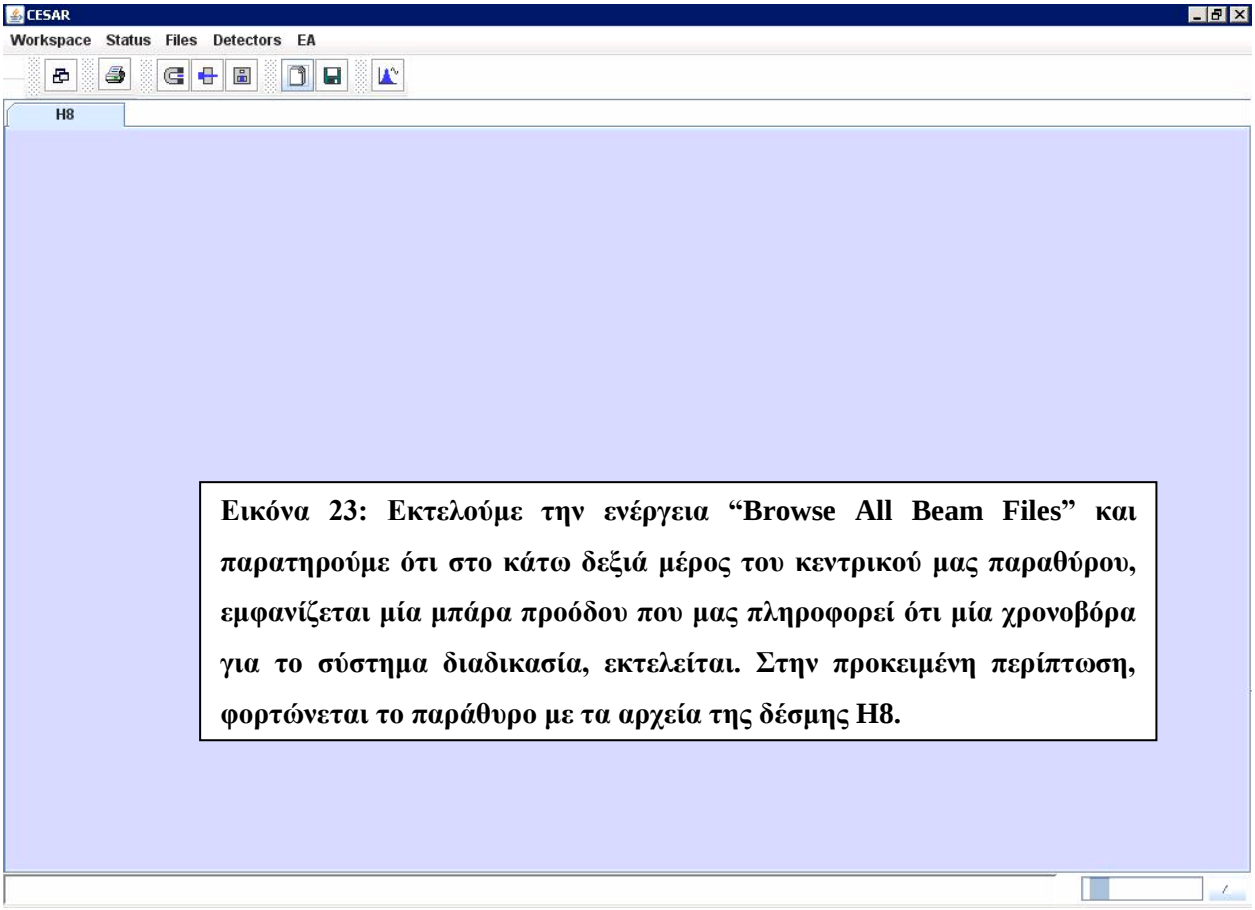
Εικόνα 20: Ξαναπροσπαθούμε login. Τώρα εισάγουμε ορθά στοιχεία για κάποιον χρήστη της κατηγορίας BEAM PARASITIC USER της δέσμης H8 και πατούμε "OK".



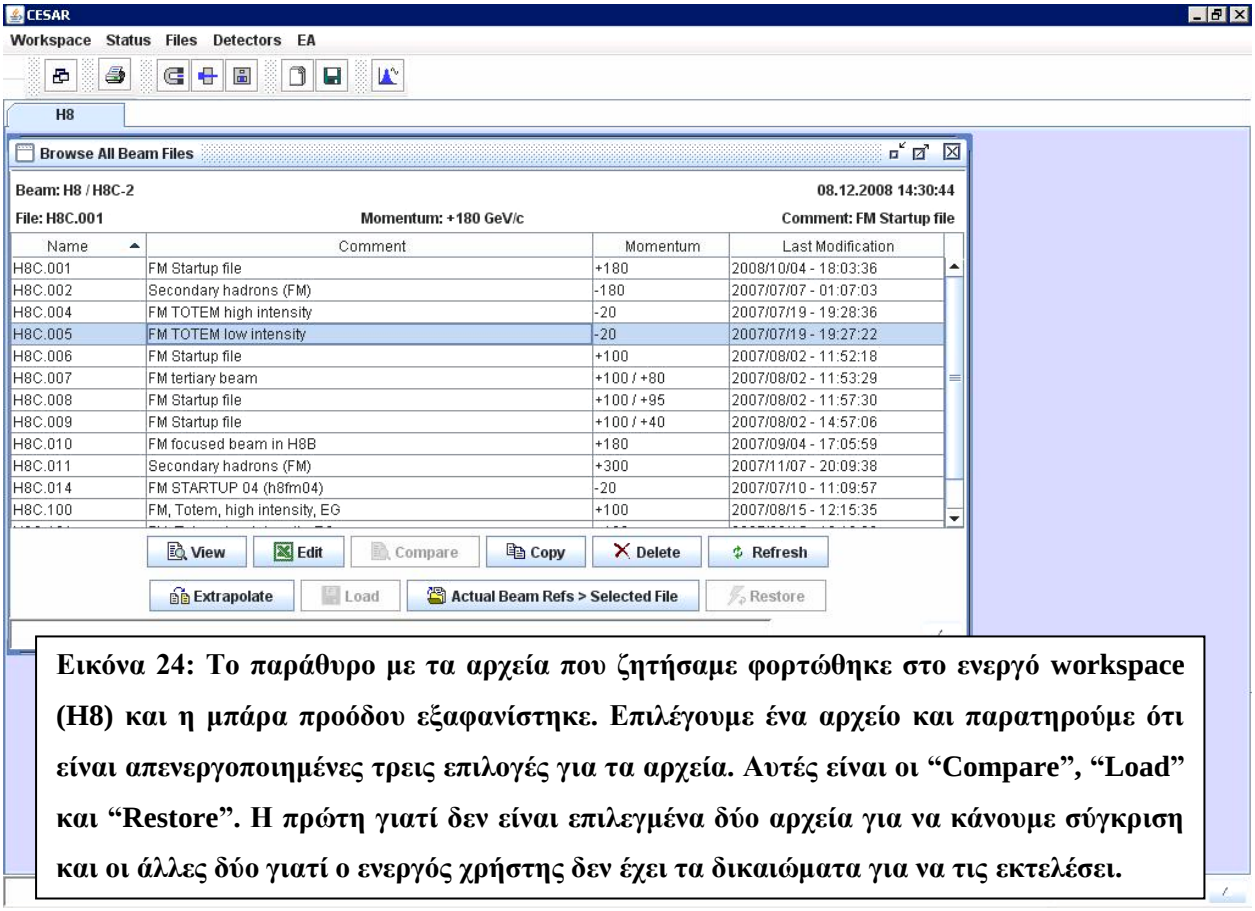
Εικόνα 21: Ένα pop-up παράθυρο μας πληροφορεί ότι συνδεθήκαμε με επιτυχία ως ο χρήστης που επιλέξαμε. Πατούμε "OK". Παρατηρούμε ότι τώρα το workspace "EA", που υπήρχε πριν, έκλεισε και στη θέση του άνοιξε το workspace του νέου ενεργού χρήστη, δηλαδή το workspace H8.



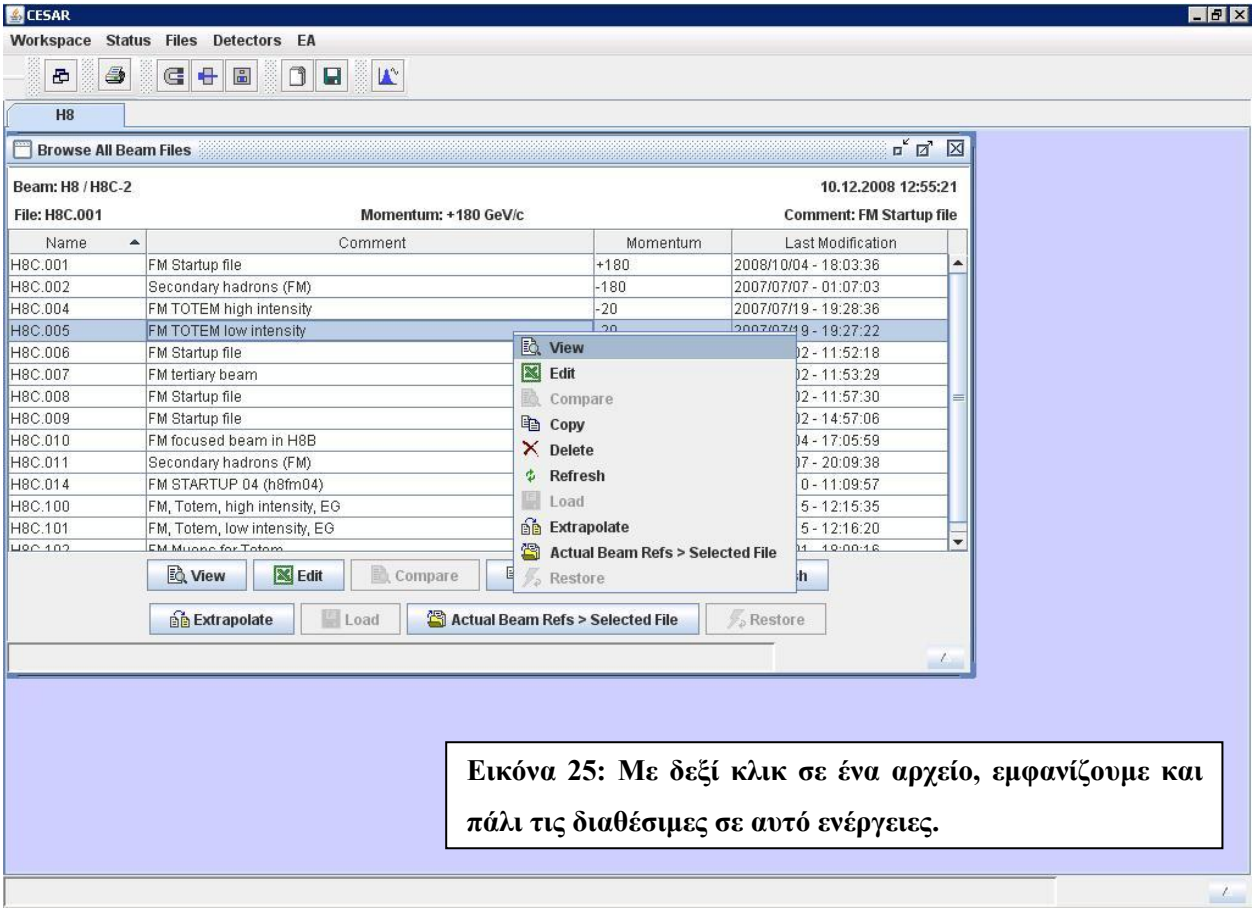
Εικόνα 22: Παρατηρούμε ότι μετά την αλλαγή ενεργού χρήστη, οι επιλογές που έχουμε ανανεώθηκαν. Έτσι, στο menu "Files", η επιλογή "BeamRefs→Last Loaded File" έγινε διαθέσιμη.



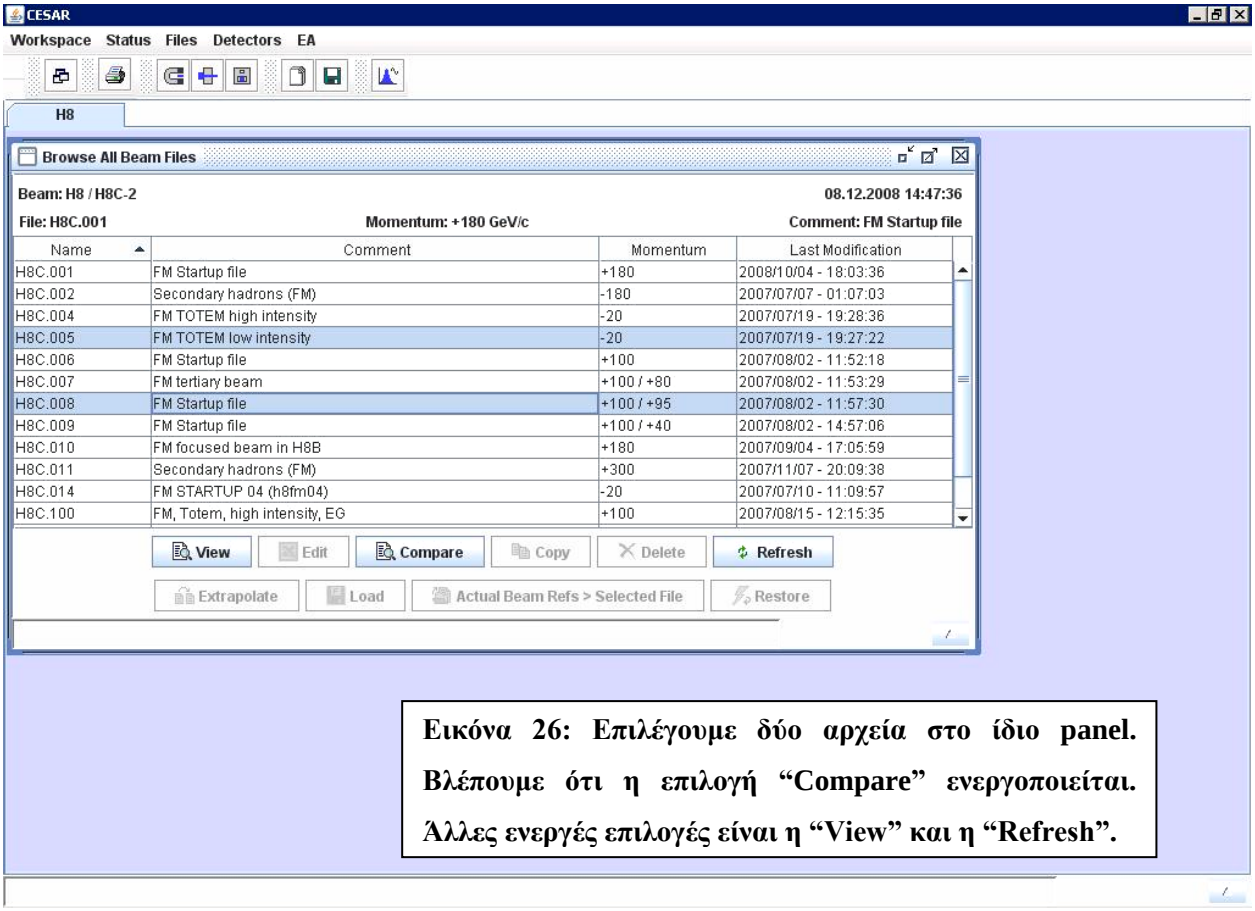
Εικόνα 23: Εκτελούμε την ενέργεια "Browse All Beam Files" και παρατηρούμε ότι στο κάτω δεξιά μέρος του κεντρικού μας παραθύρου, εμφανίζεται μία μπάρα προόδου που μας πληροφορεί ότι μία χρονοβόρα για το σύστημα διαδικασία, εκτελείται. Στην προκειμένη περίπτωση, φορτώνεται το παράθυρο με τα αρχεία της δέσμης H8.



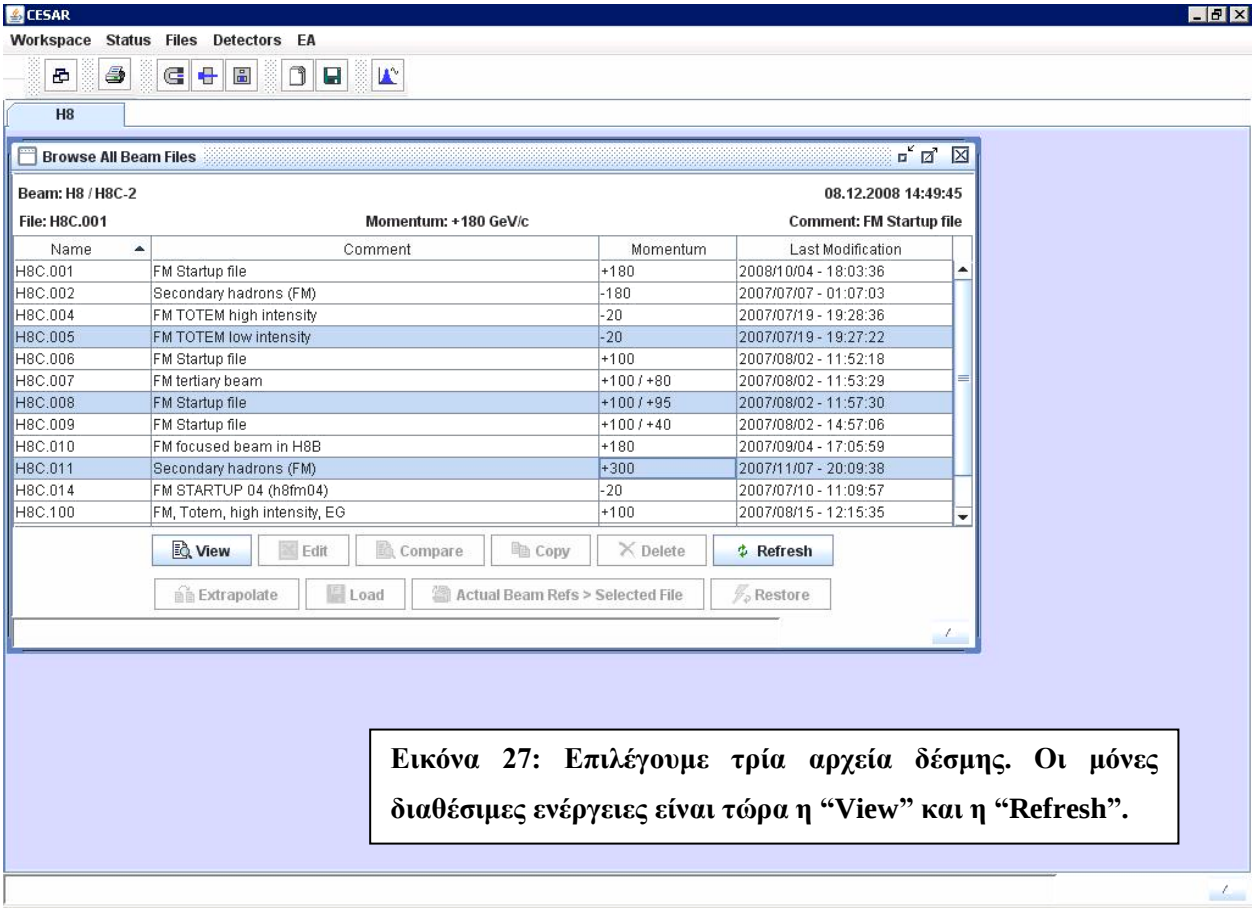
Εικόνα 24: Το παράθυρο με τα αρχεία που ζητήσαμε φορτώθηκε στο ενεργό workspace (H8) και η μπάρα προόδου εξαφανίστηκε. Επιλέγουμε ένα αρχείο και παρατηρούμε ότι είναι απενεργοποιημένες τρεις επιλογές για τα αρχεία. Αυτές είναι οι "Compare", "Load" και "Restore". Η πρώτη γιατί δεν είναι επιλεγμένα δύο αρχεία για να κάνουμε σύγκριση και οι άλλες δύο γιατί ο ενεργός χρήστης δεν έχει τα δικαιώματα για να τις εκτελέσει.



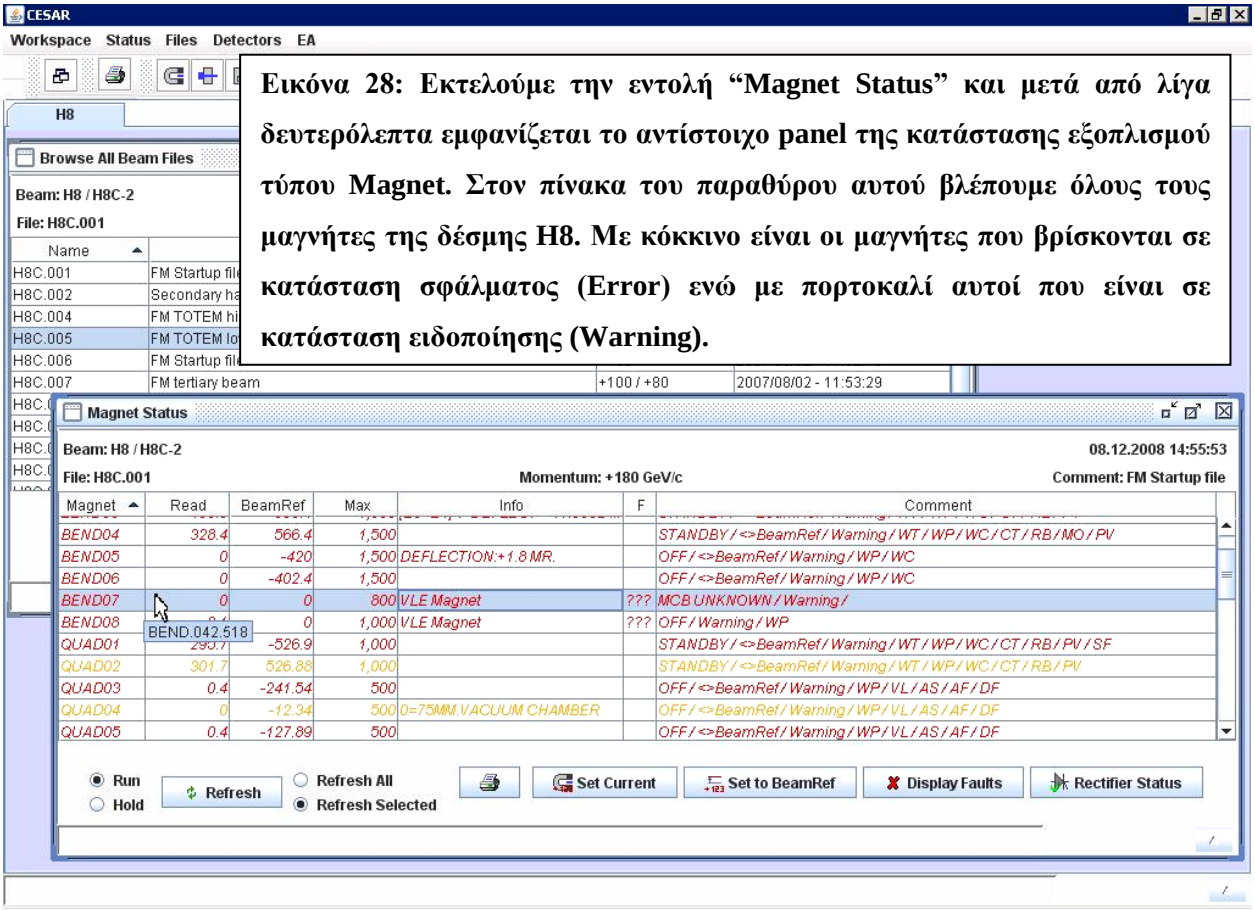
Εικόνα 25: Με δεξί κλικ σε ένα αρχείο, εμφανίζουμε και πάλι τις διαθέσιμες σε αυτό ενέργειες.



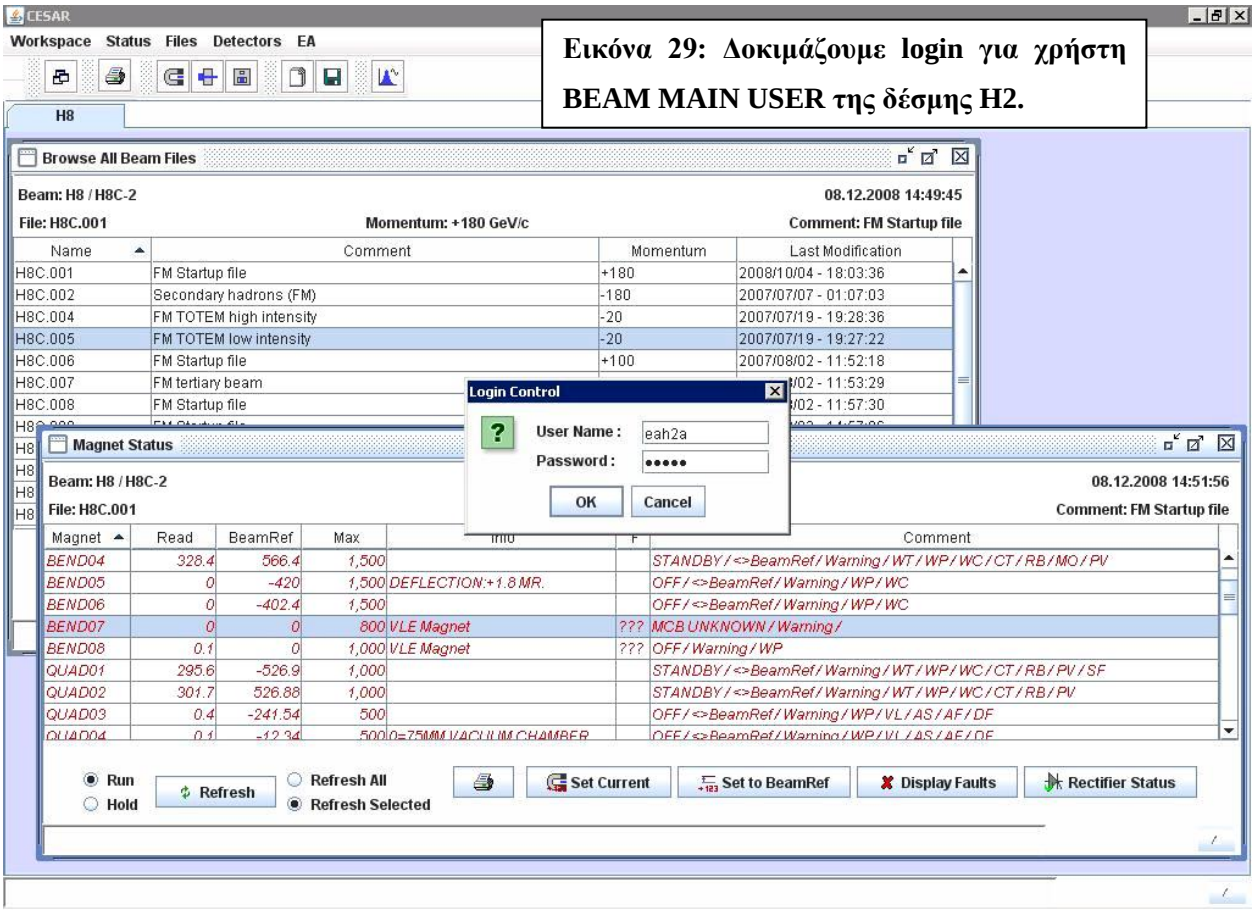
Εικόνα 26: Επιλέγουμε δύο αρχεία στο ίδιο panel. Βλέπουμε ότι η επιλογή "Compare" ενεργοποιείται. Άλλες ενεργές επιλογές είναι η "View" και η "Refresh".



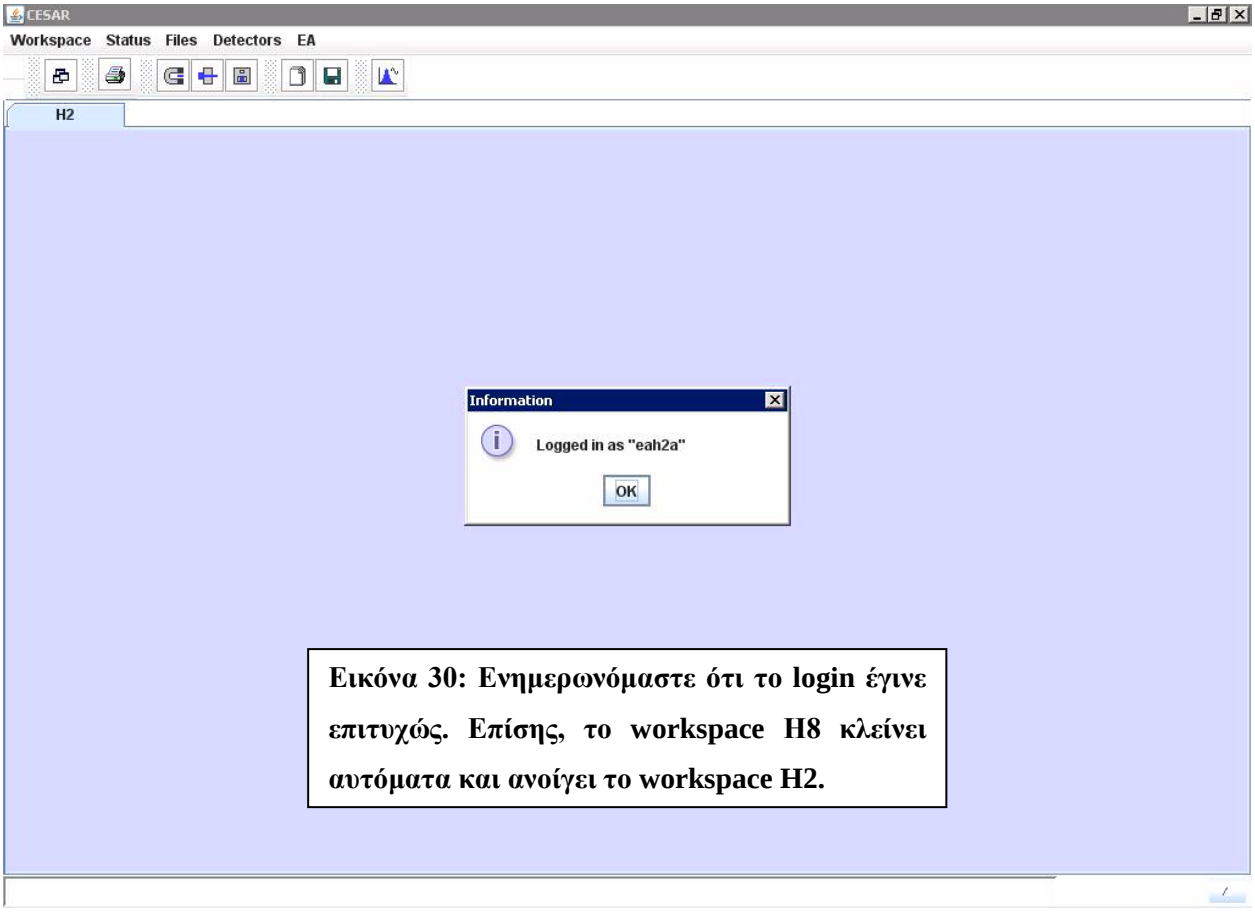
Εικόνα 27: Επιλέγουμε τρία αρχεία δέσμης. Οι μόνες διαθέσιμες ενέργειες είναι τώρα η "View" και η "Refresh".



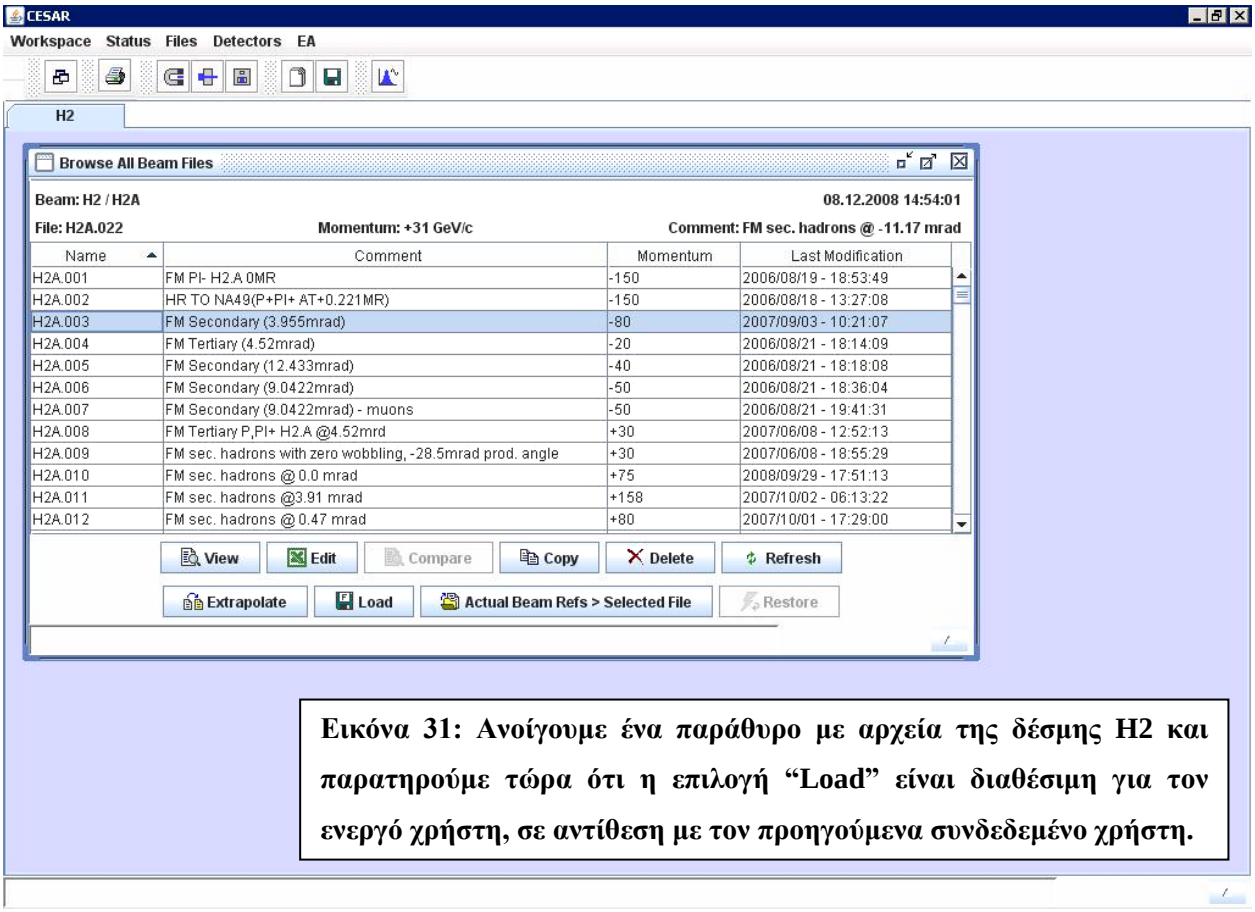
Εικόνα 28: Εκτελούμε την εντολή "Magnet Status" και μετά από λίγα δευτερόλεπτα εμφανίζεται το αντίστοιχο panel της κατάστασης εξοπλισμού τύπου Magnet. Στον πίνακα του παραθύρου αυτού βλέπουμε όλους τους μαγνήτες της δέσμης H8. Με κόκκινο είναι οι μαγνήτες που βρίσκονται σε κατάσταση σφάλματος (Error) ενώ με πορτοκαλί αυτοί που είναι σε κατάσταση ειδοποίησης (Warning).



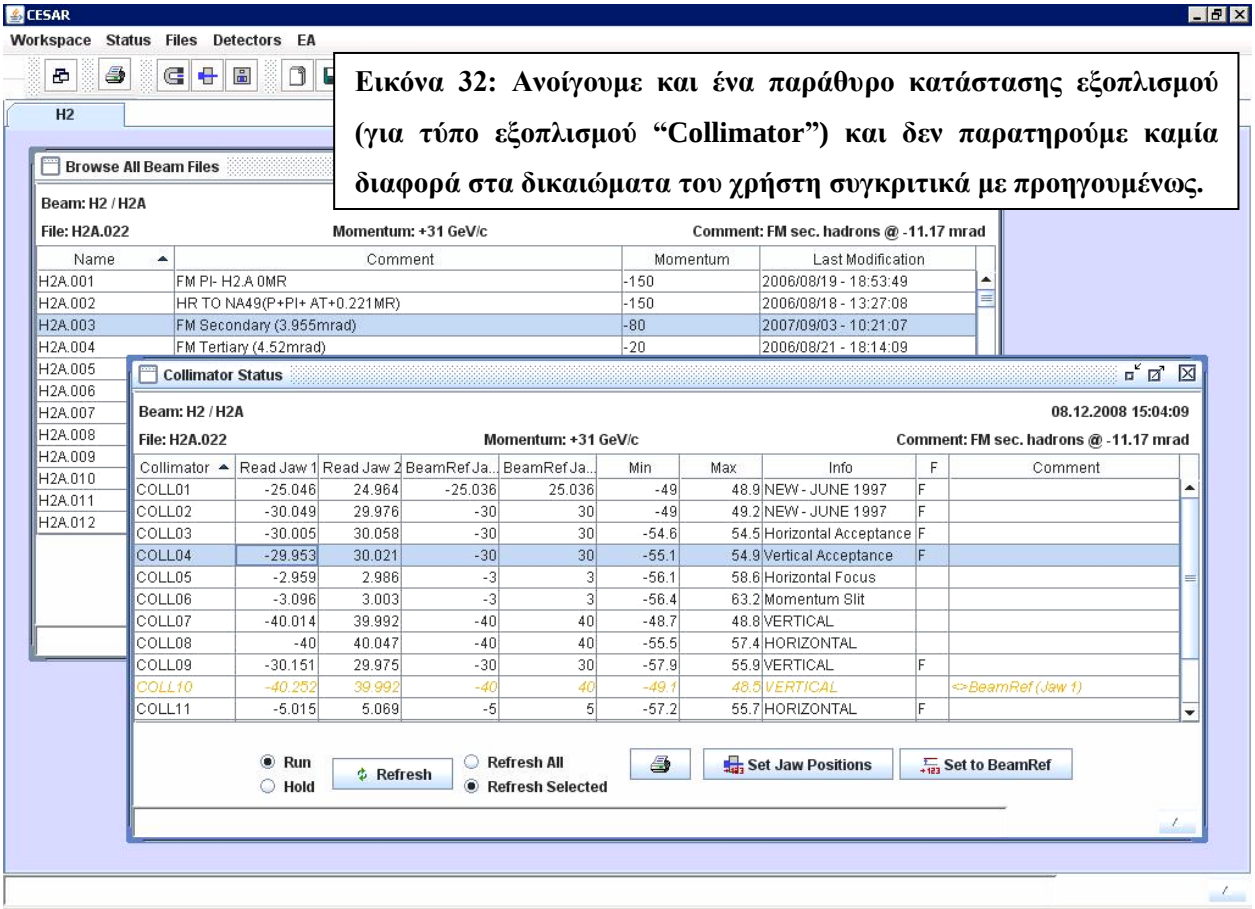
Εικόνα 29: Δοκιμάζουμε login για χρήστη BEAM MAIN USER της δέσμης H2.



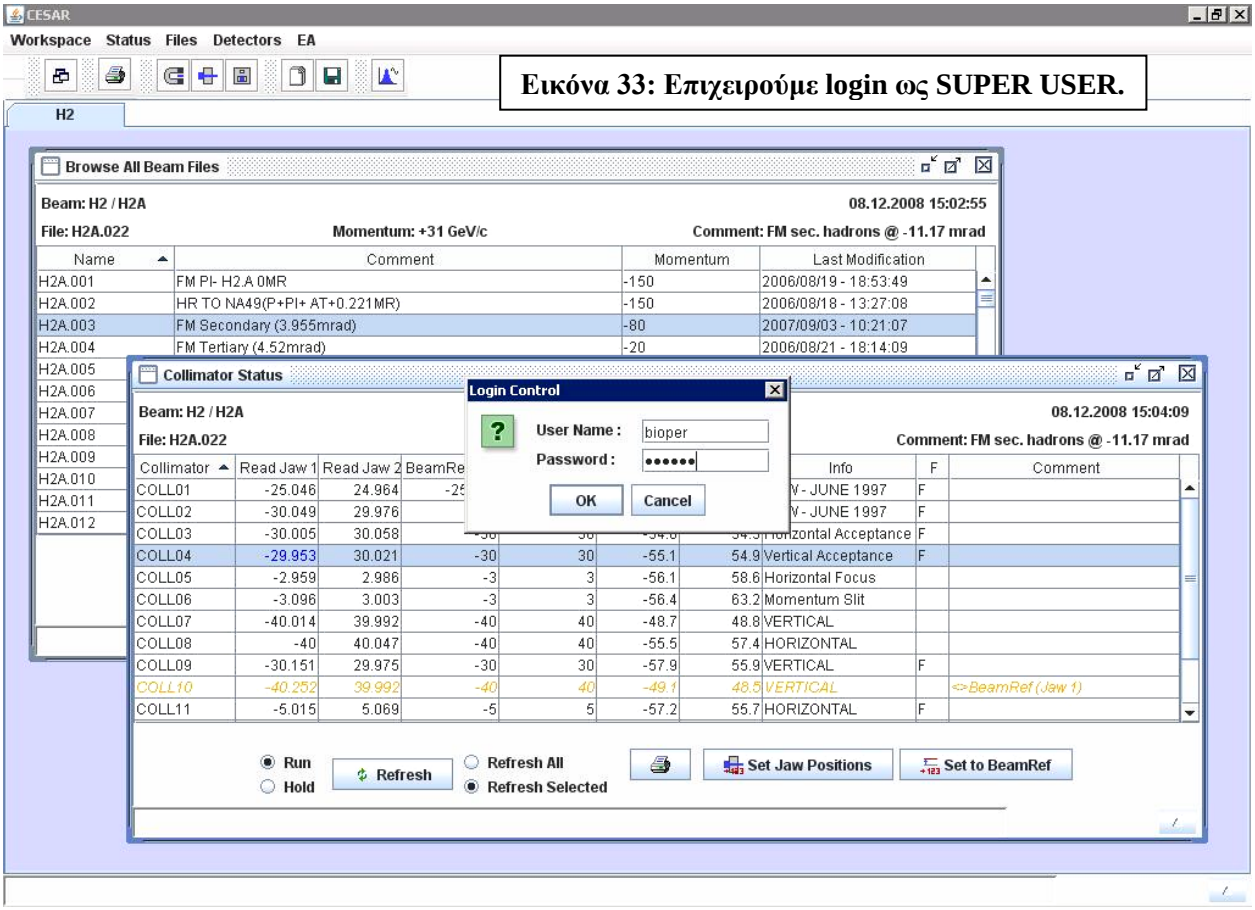
Εικόνα 30: Ενημερωνόμαστε ότι το login έγινε επιτυχώς. Επίσης, το workspace H8 κλείνει αυτόματα και ανοίγει το workspace H2.



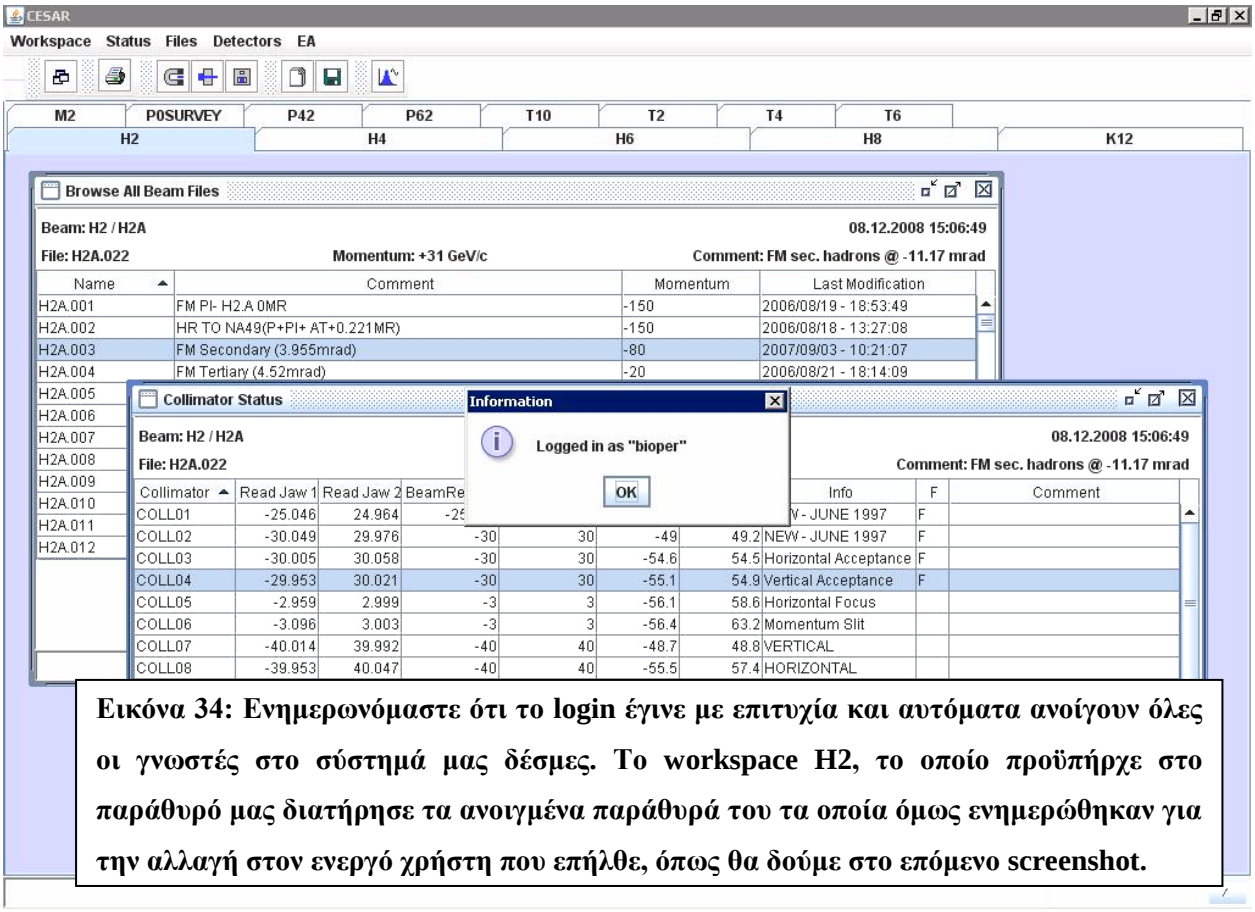
Εικόνα 31: Ανοίγουμε ένα παράθυρο με αρχεία της δέσμης H2 και παρατηρούμε τώρα ότι η επιλογή "Load" είναι διαθέσιμη για τον ενεργό χρήστη, σε αντίθεση με τον προηγούμενα συνδεδεμένο χρήστη.



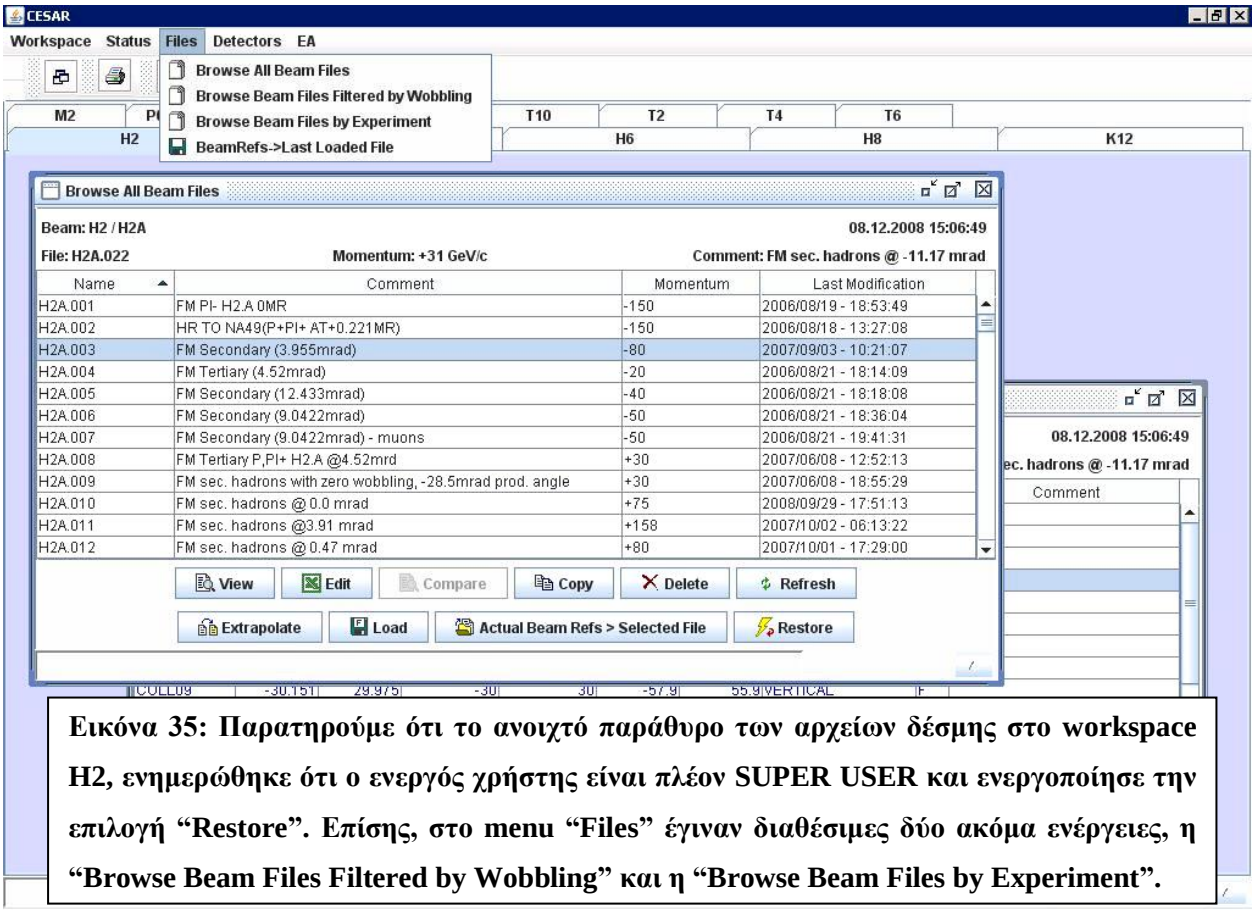
Εικόνα 32: Ανοίγουμε και ένα παράθυρο κατάστασης εξοπλισμού (για τύπο εξοπλισμού "Collimator") και δεν παρατηρούμε καμία διαφορά στα δικαιώματα του χρήστη συγκριτικά με προηγουμένως.



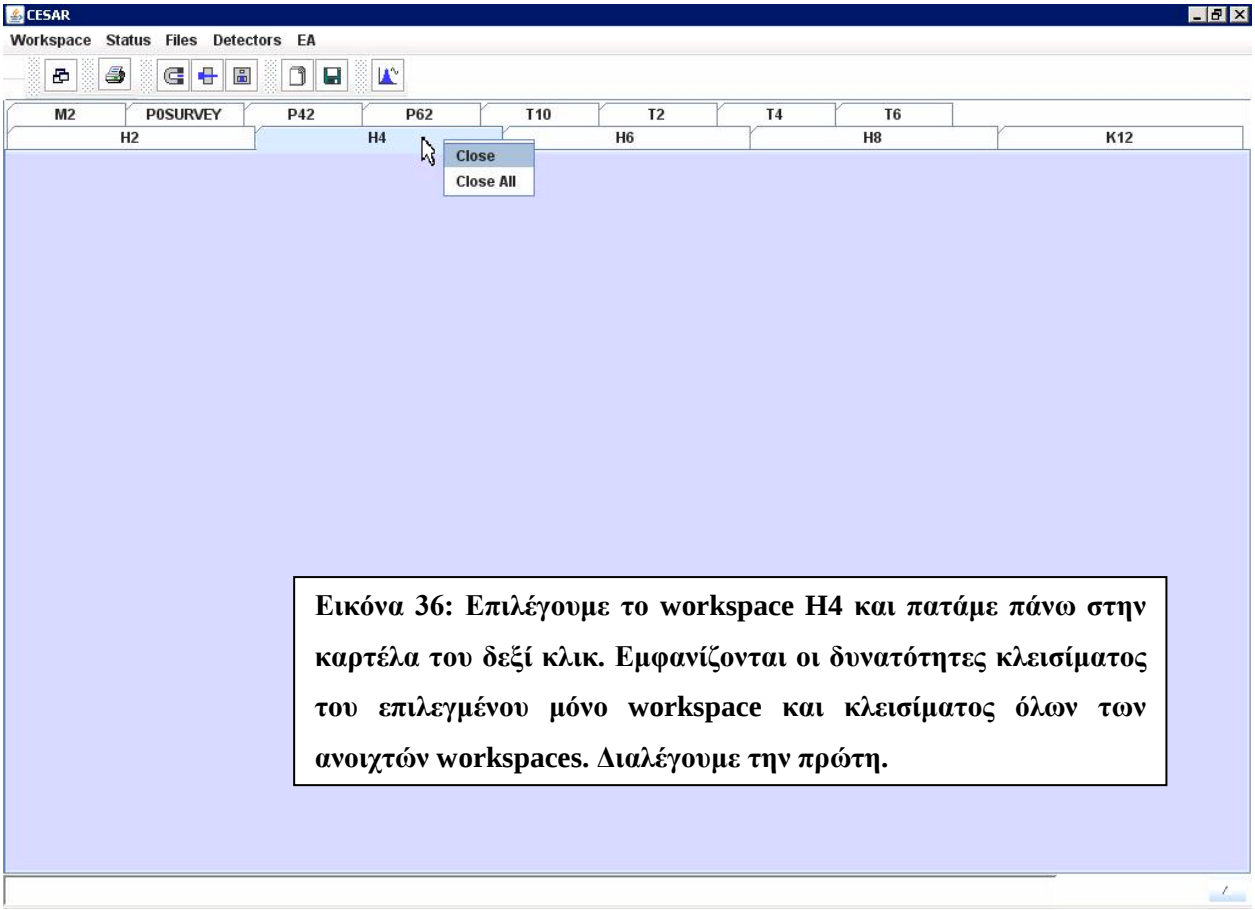
Εικόνα 33: Επιχειρούμε login ως SUPER USER.



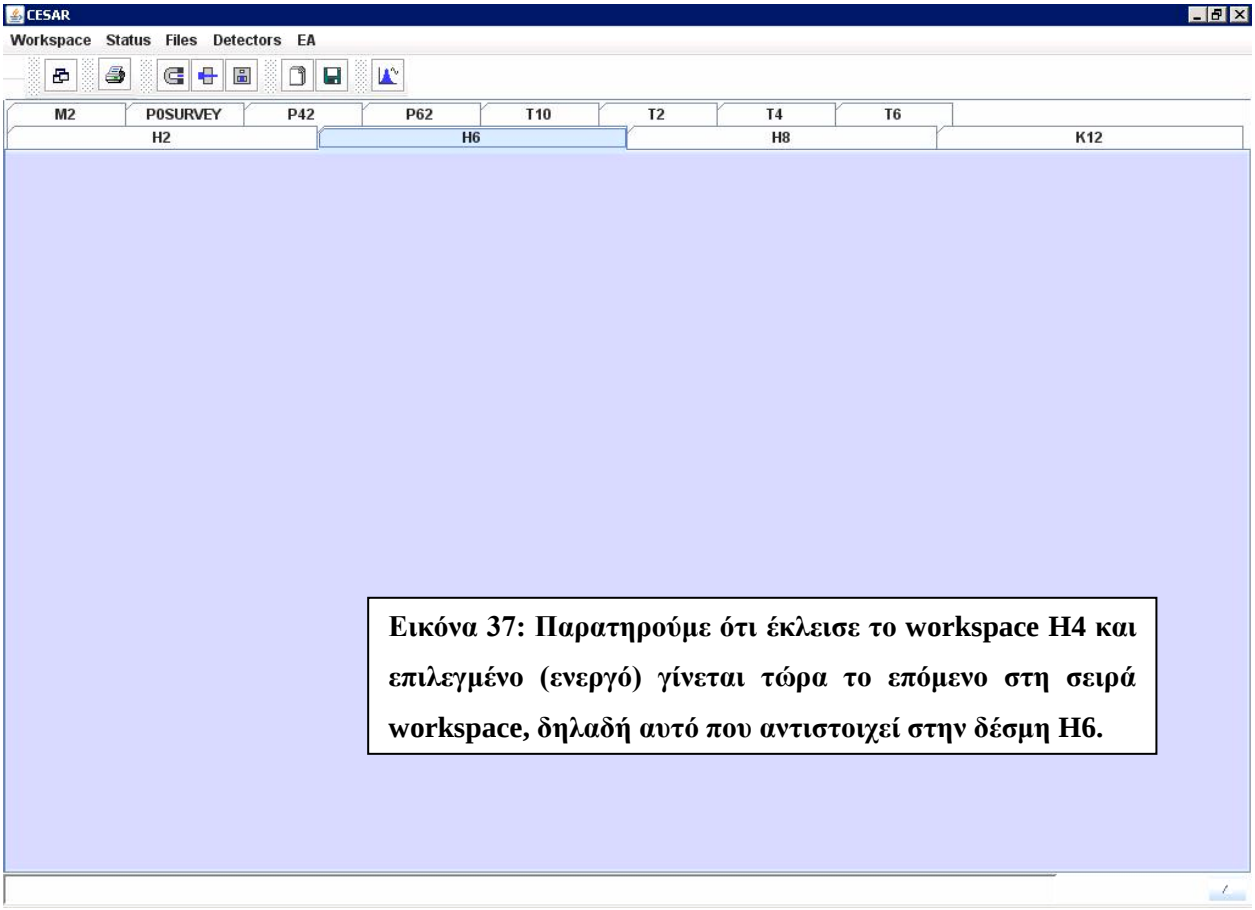
Εικόνα 34: Ενημερωνόμαστε ότι το login έγινε με επιτυχία και αυτόματα ανοίγουν όλες οι γνωστές στο σύστημά μας δέσμες. Το workspace H2, το οποίο προϋπήρχε στο παράθυρό μας διατήρησε τα ανοιγμένα παράθυρά του τα οποία όμως ενημερώθηκαν για την αλλαγή στον ενεργό χρήστη που επήλθε, όπως θα δούμε στο επόμενο screenshot.



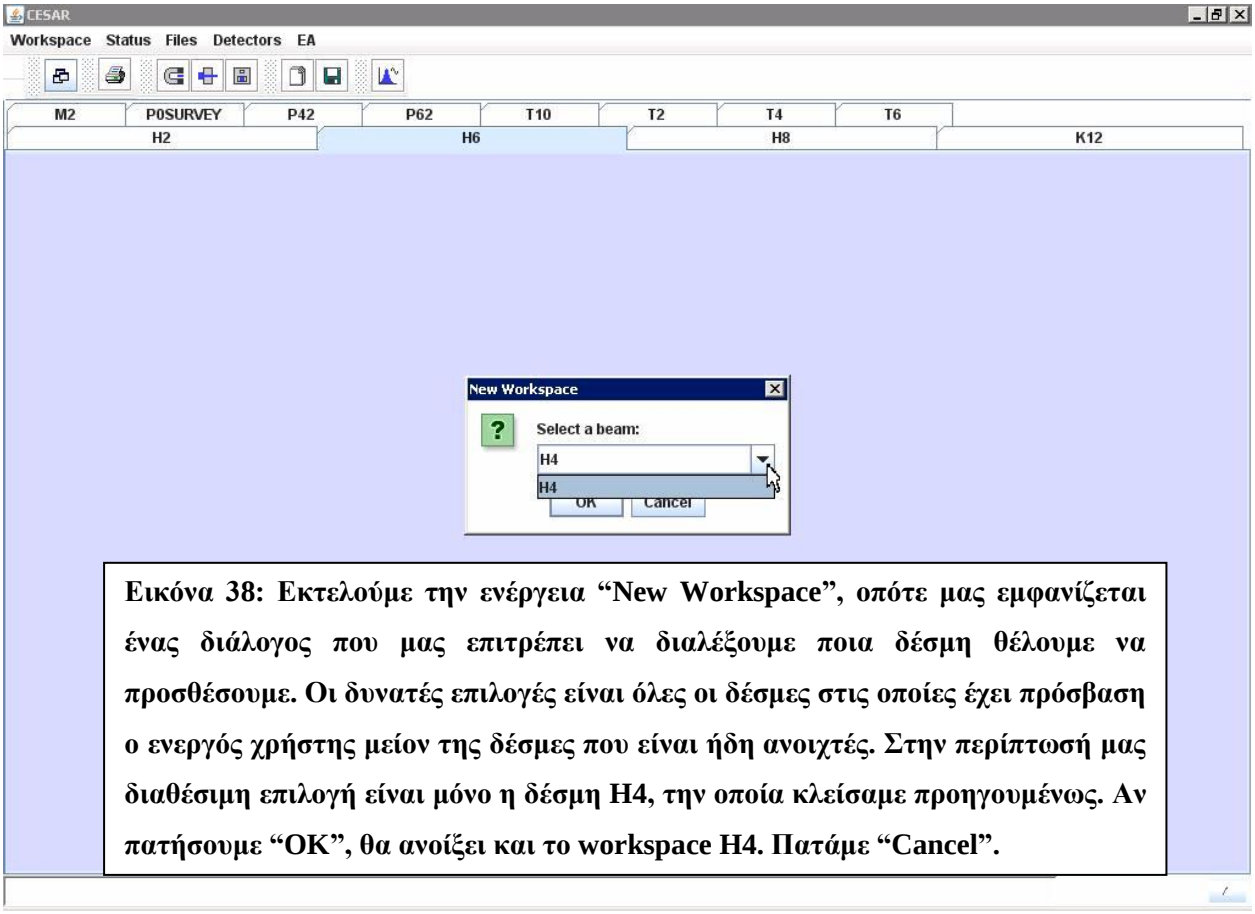
Εικόνα 35: Παρατηρούμε ότι το ανοιχτό παράθυρο των αρχείων δέσμης στο workspace H2, ενημερώθηκε ότι ο ενεργός χρήστης είναι πλέον SUPER USER και ενεργοποίησε την επιλογή "Restore". Επίσης, στο menu "Files" έγιναν διαθέσιμες δύο ακόμα ενέργειες, η "Browse Beam Files Filtered by Wobbling" και η "Browse Beam Files by Experiment".



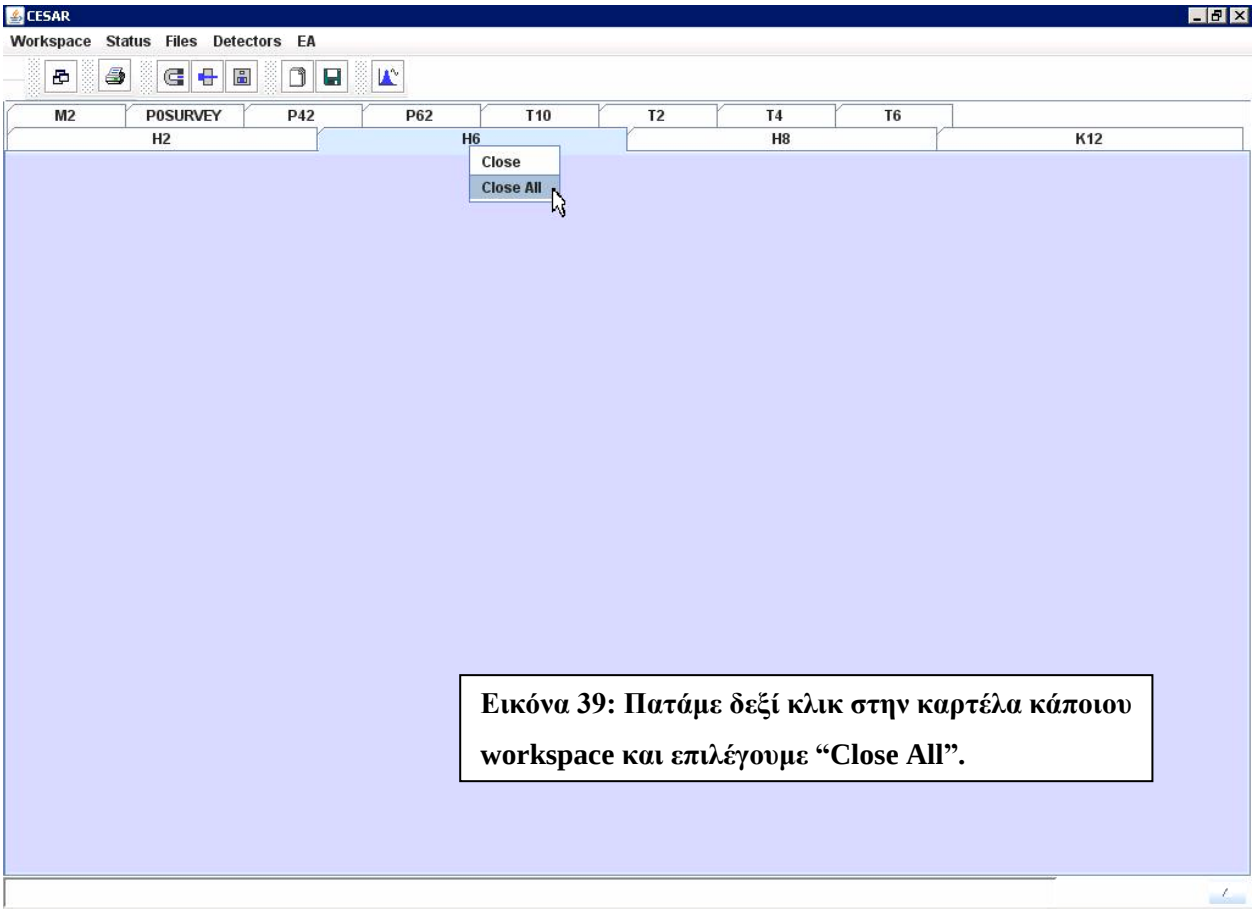
Εικόνα 36: Επιλέγουμε το workspace H4 και πατάμε πάνω στην καρτέλα του δεξί κλικ. Εμφανίζονται οι δυνατότητες κλεισίματος του επιλεγμένου μόνο workspace και κλεισίματος όλων των ανοιχτών workspaces. Διαλέγουμε την πρώτη.



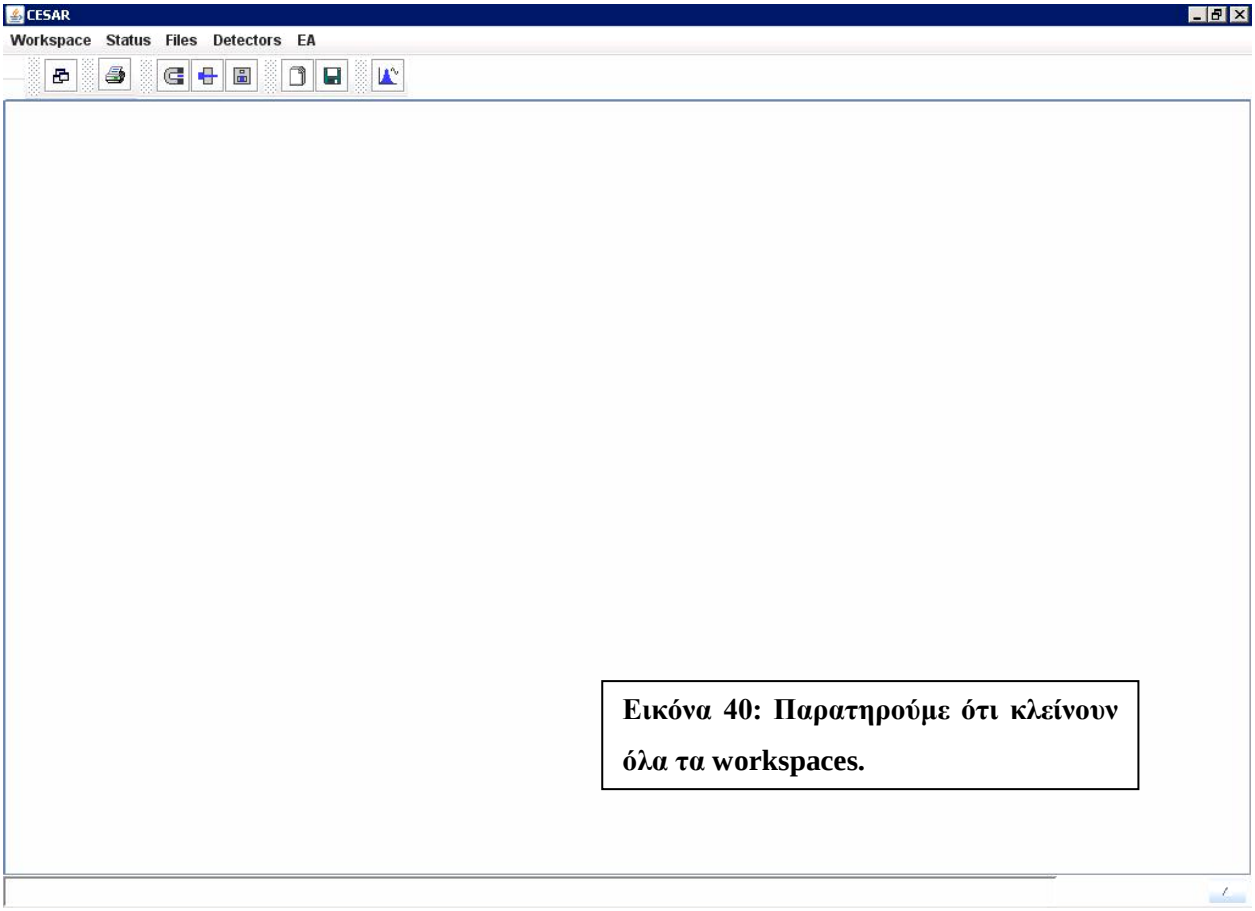
Εικόνα 37: Παρατηρούμε ότι έκλεισε το workspace H4 και επιλεγμένο (ενεργό) γίνεται τώρα το επόμενο στη σειρά workspace, δηλαδή αυτό που αντιστοιχεί στην δέσμη H6.



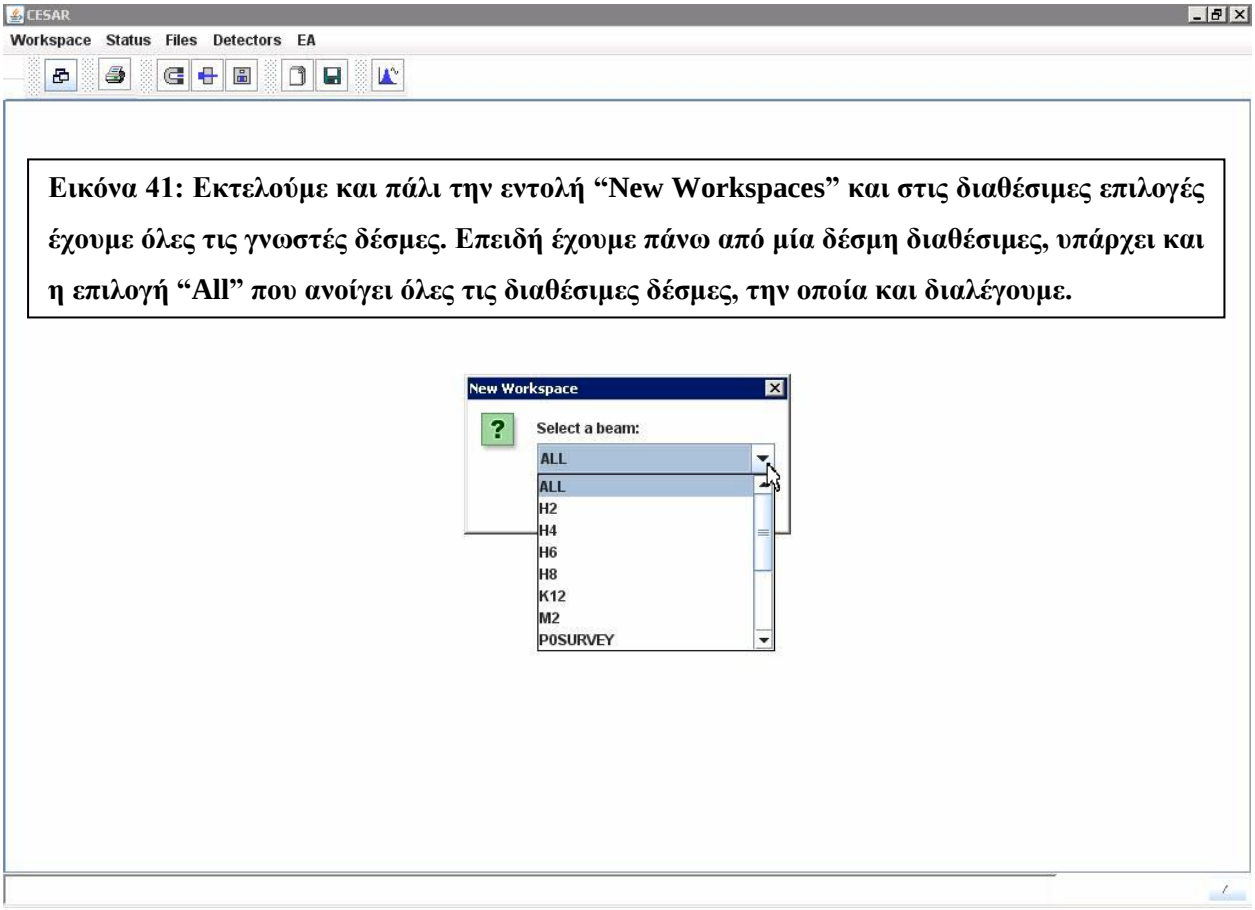
Εικόνα 38: Εκτελούμε την ενέργεια "New Workspace", οπότε μας εμφανίζεται ένας διάλογος που μας επιτρέπει να διαλέξουμε ποια δέσμη θέλουμε να προσθέσουμε. Οι δυνατές επιλογές είναι όλες οι δέσμες στις οποίες έχει πρόσβαση ο ενεργός χρήστης μείον της δέσμες που είναι ήδη ανοιχτές. Στην περίπτωσή μας διαθέσιμη επιλογή είναι μόνο η δέσμη H4, την οποία κλείσαμε προηγουμένως. Αν πατήσουμε "OK", θα ανοίξει και το workspace H4. Πατάμε "Cancel".



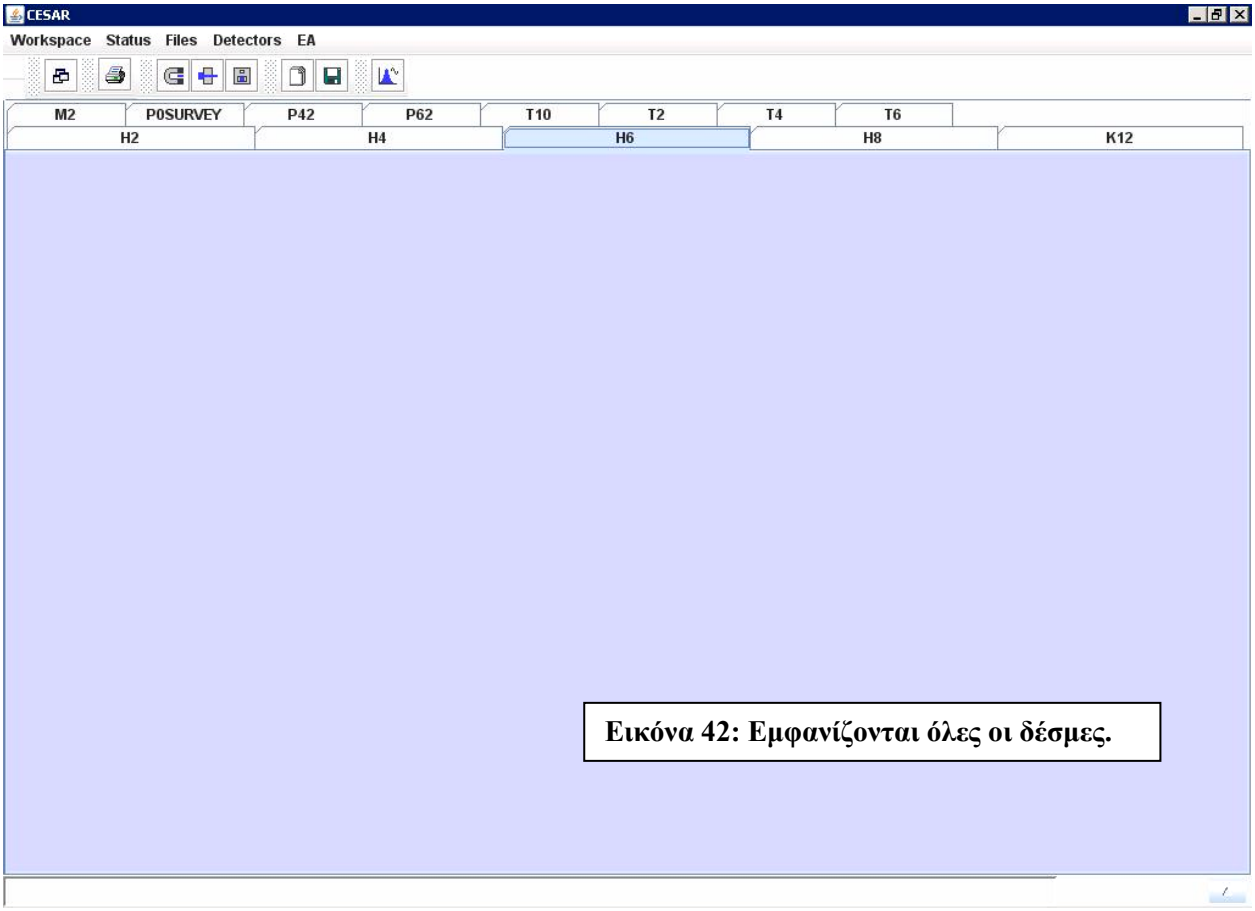
Εικόνα 39: Πατάμε δεξί κλικ στην καρτέλα κάποιου workspace και επιλέγουμε "Close All".



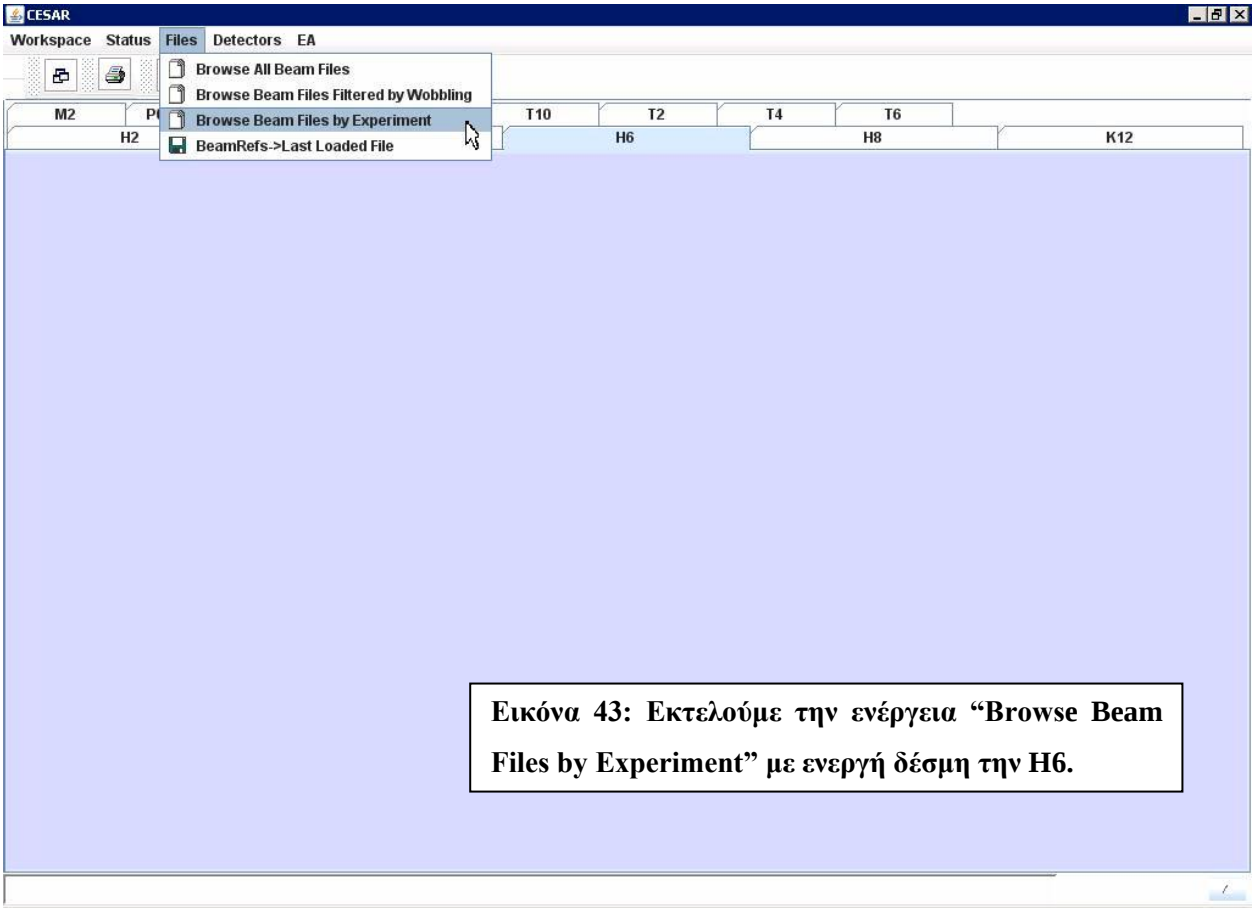
Εικόνα 40: Παρατηρούμε ότι κλείνουν όλα τα workspaces.



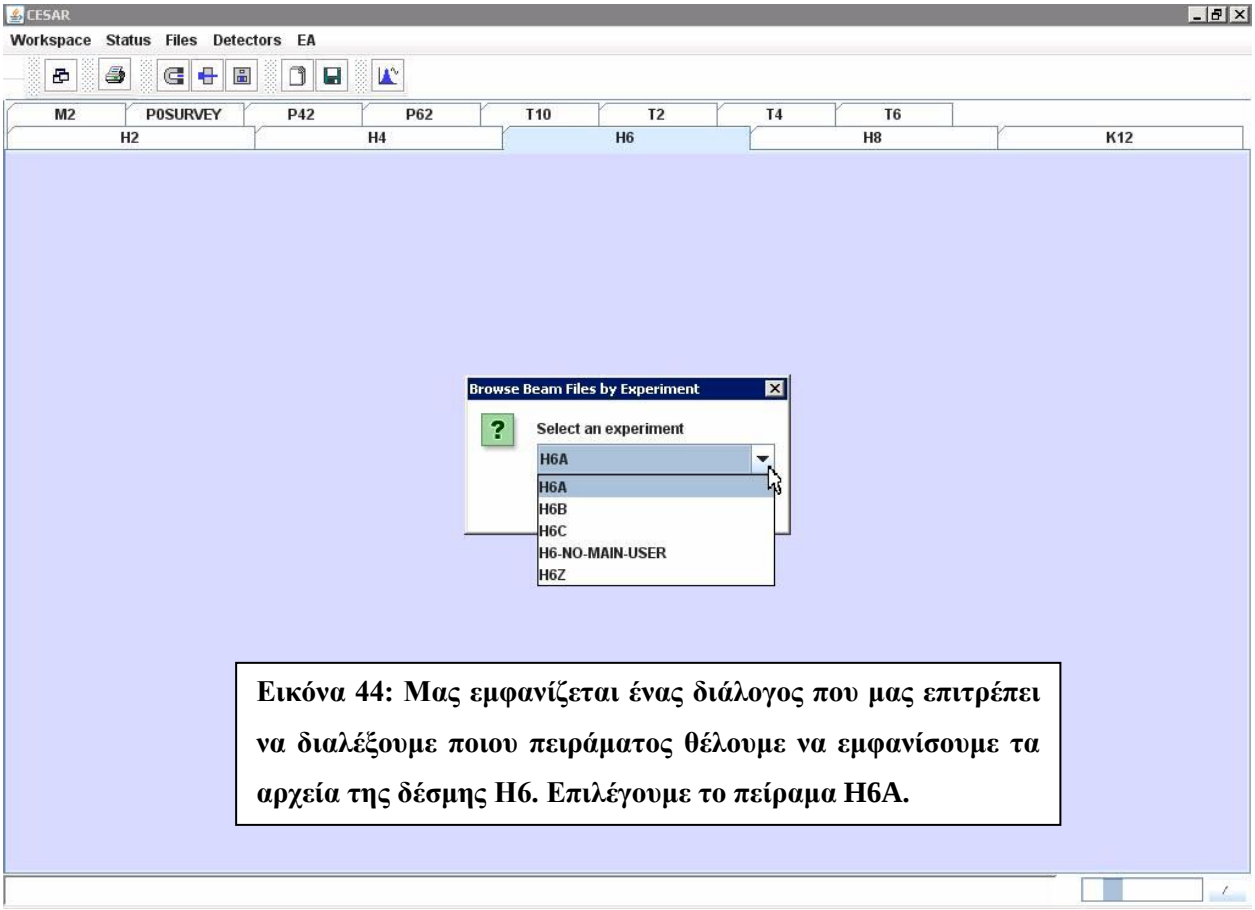
Εικόνα 41: Εκτελούμε και πάλι την εντολή "New Workspaces" και στις διαθέσιμες επιλογές έχουμε όλες τις γνωστές δέσμες. Επειδή έχουμε πάνω από μία δέσμη διαθέσιμες, υπάρχει και η επιλογή "All" που ανοίγει όλες τις διαθέσιμες δέσμες, την οποία και διαλέγουμε.



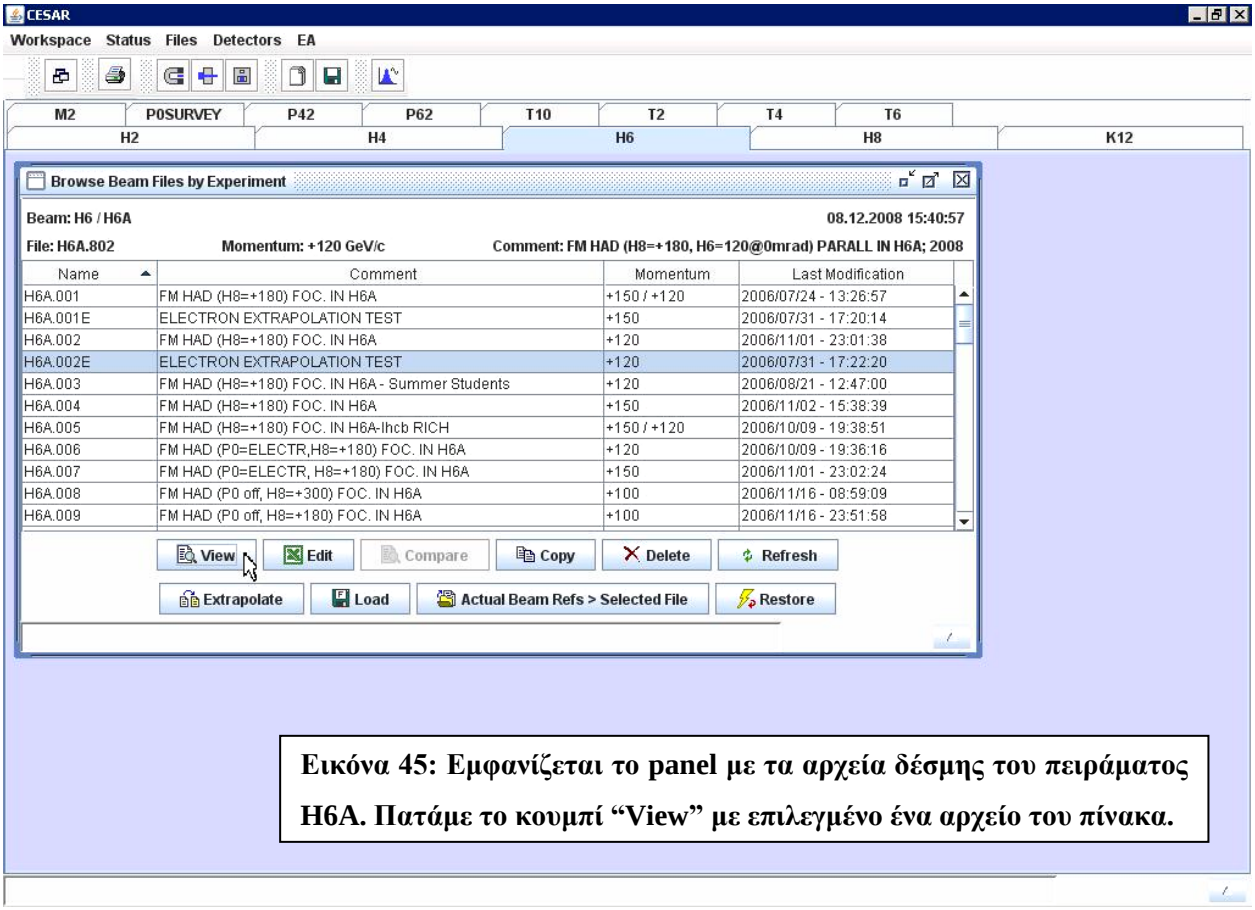
Εικόνα 42: Εμφανίζονται όλες οι δέσμες.



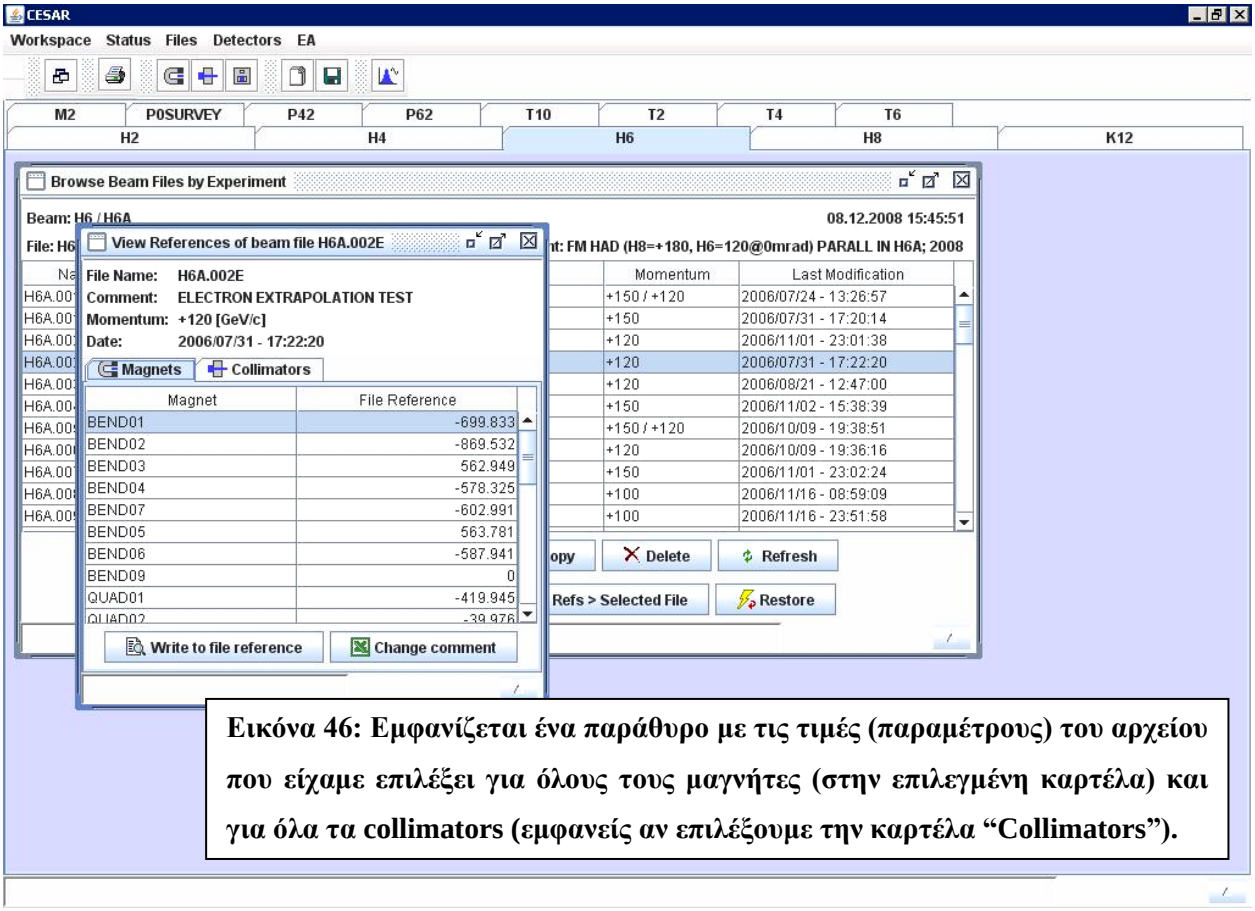
Εικόνα 43: Εκτελούμε την ενέργεια "Browse Beam Files by Experiment" με ενεργή δέσμη την H6.



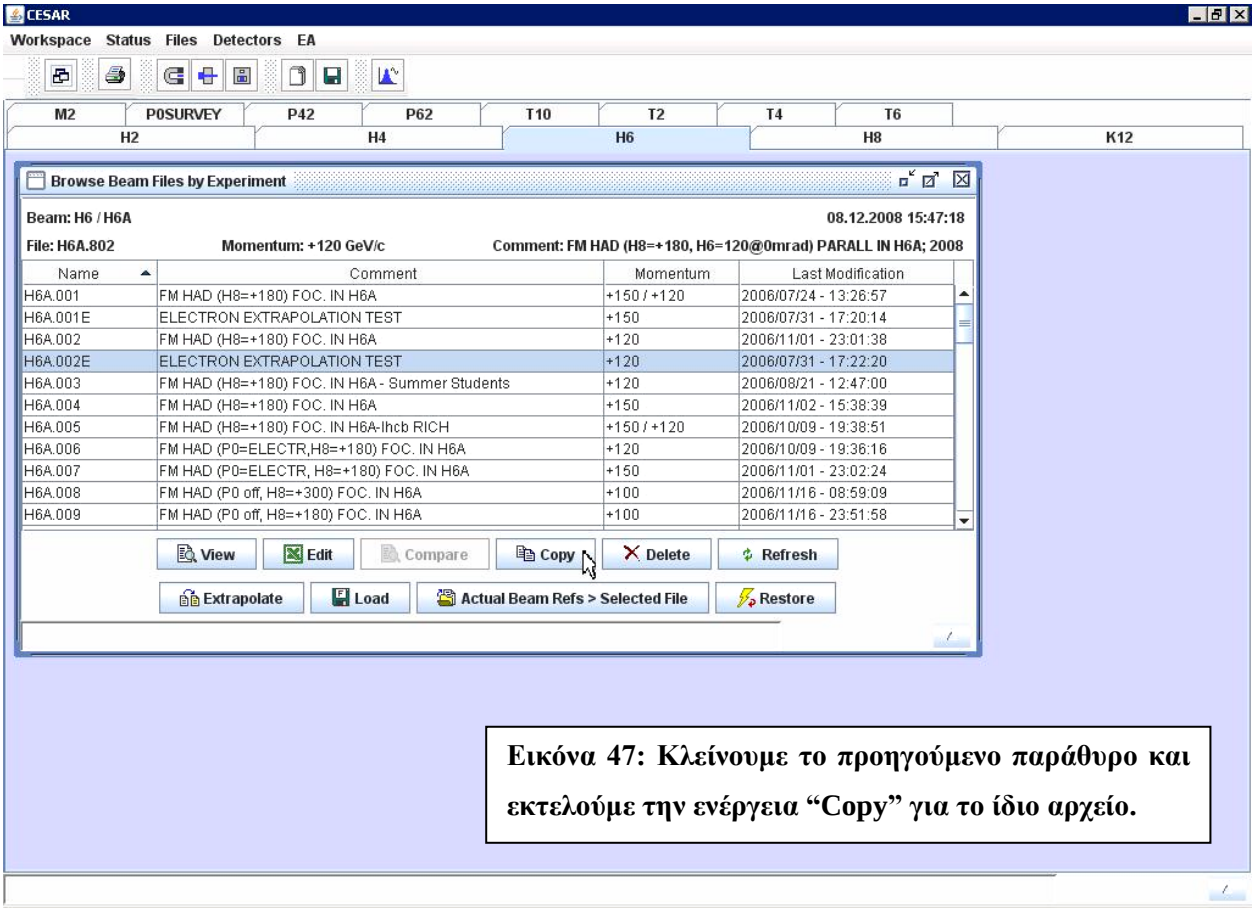
Εικόνα 44: Μας εμφανίζεται ένας διάλογος που μας επιτρέπει να διαλέξουμε ποιου πειράματος θέλουμε να εμφανίσουμε τα αρχεία της δέσμης H6. Επιλέγουμε το πείραμα H6A.



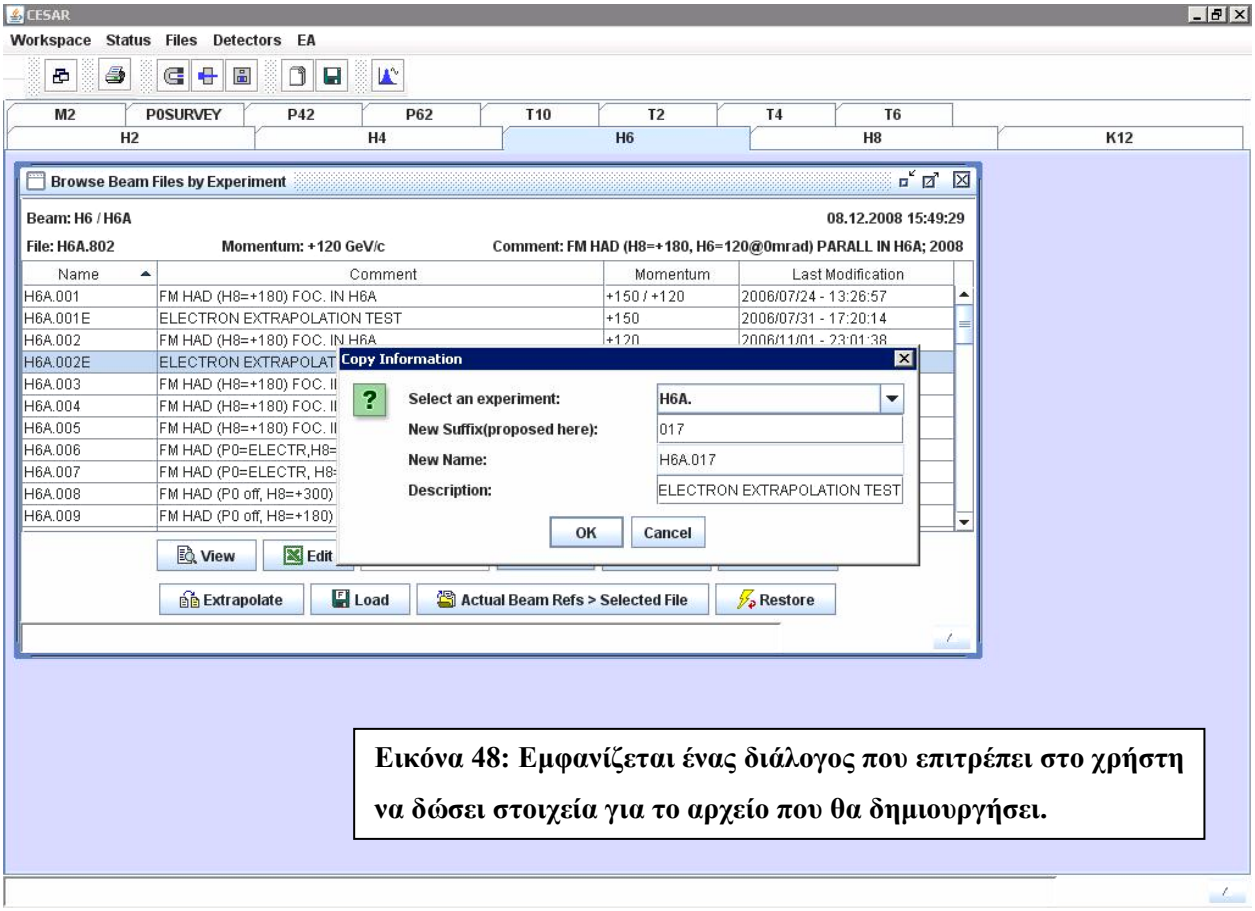
Εικόνα 45: Εμφανίζεται το panel με τα αρχεία δέσμης του πειράματος H6A. Πατάμε το κουμπί "View" με επιλεγμένο ένα αρχείο του πίνακα.



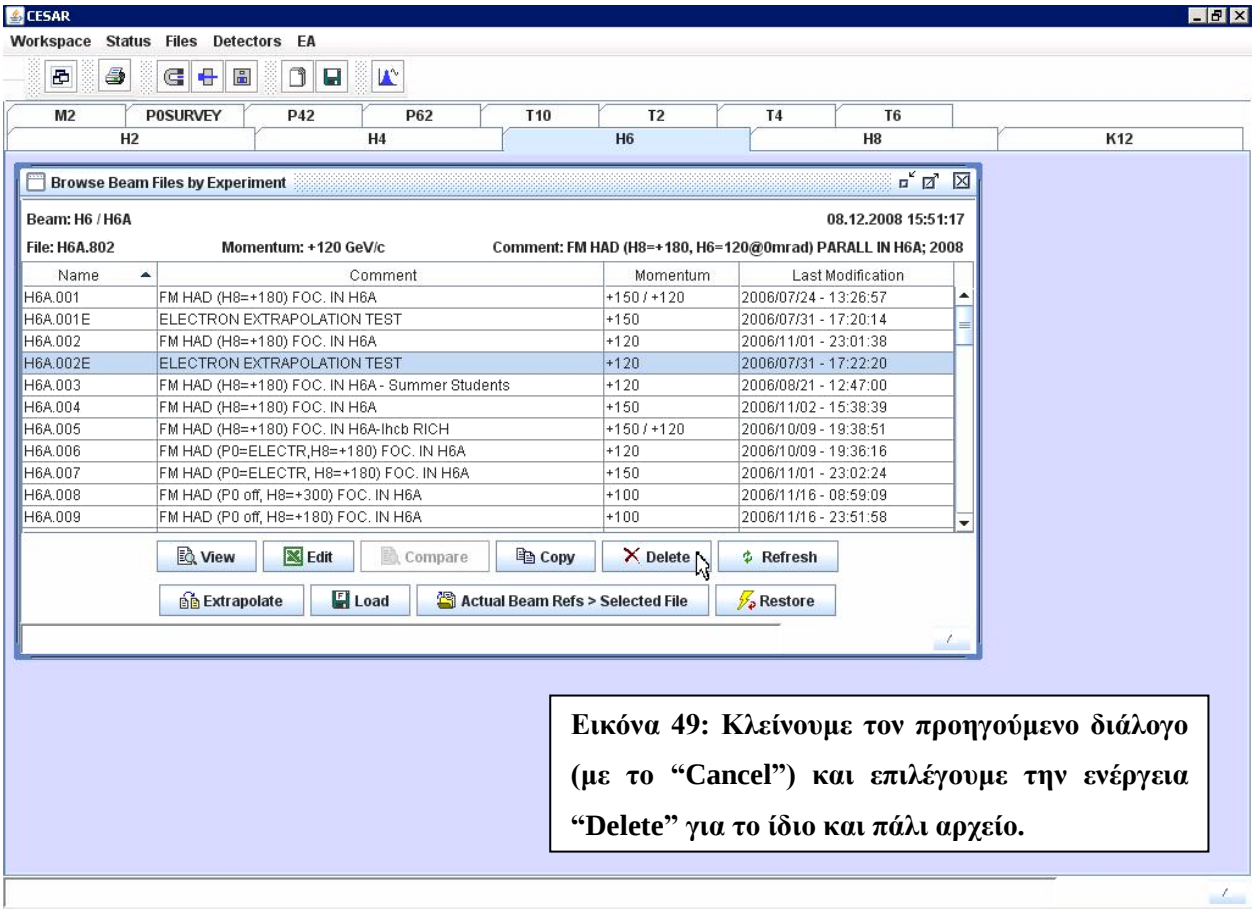
Εικόνα 46: Εμφανίζεται ένα παράθυρο με τις τιμές (παραμέτρους) του αρχείου που είχαμε επιλέξει για όλους τους μαγνήτες (στην επιλεγμένη καρτέλα) και για όλα τα collimators (εμφανείς αν επιλέξουμε την καρτέλα "Collimators").



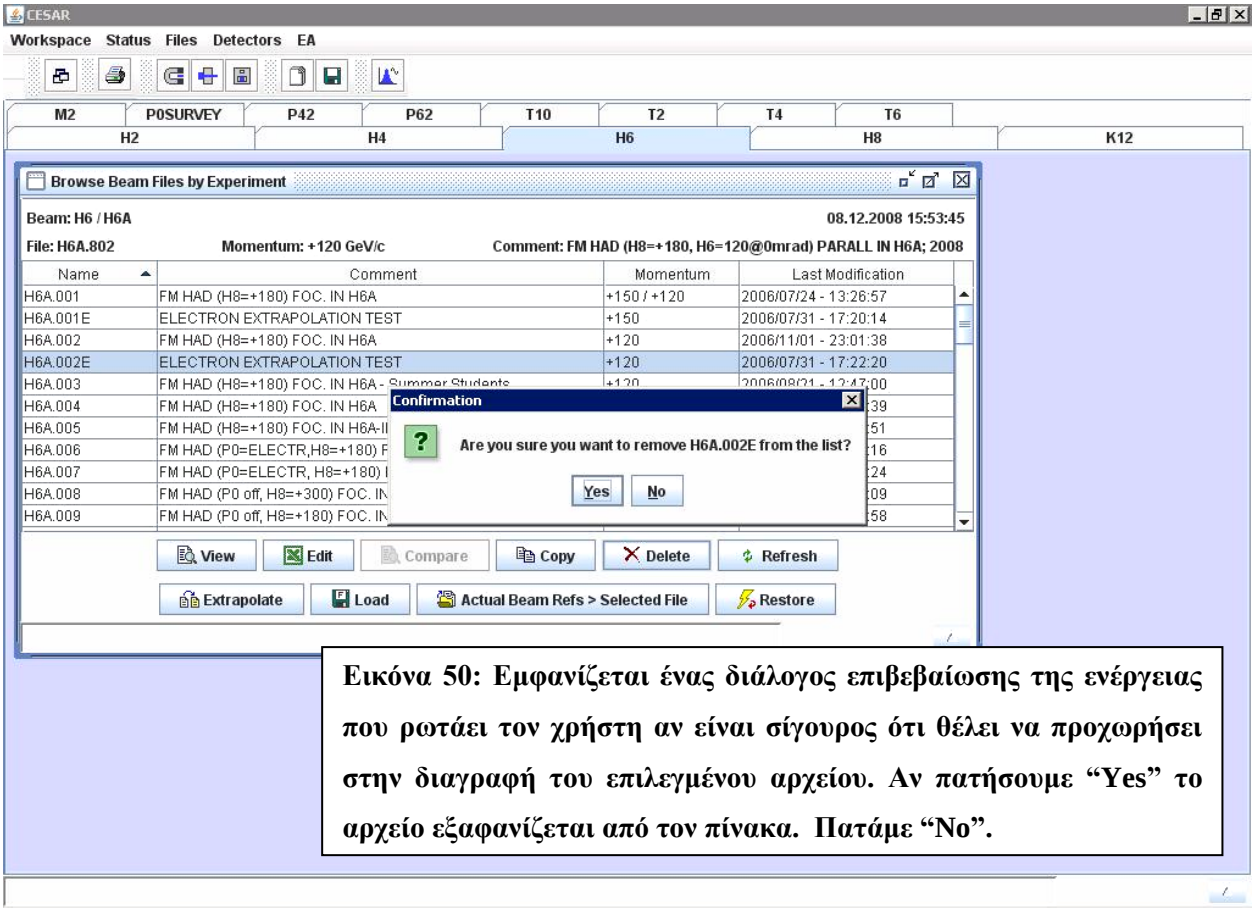
Εικόνα 47: Κλείνουμε το προηγούμενο παράθυρο και εκτελούμε την ενέργεια "Copy" για το ίδιο αρχείο.



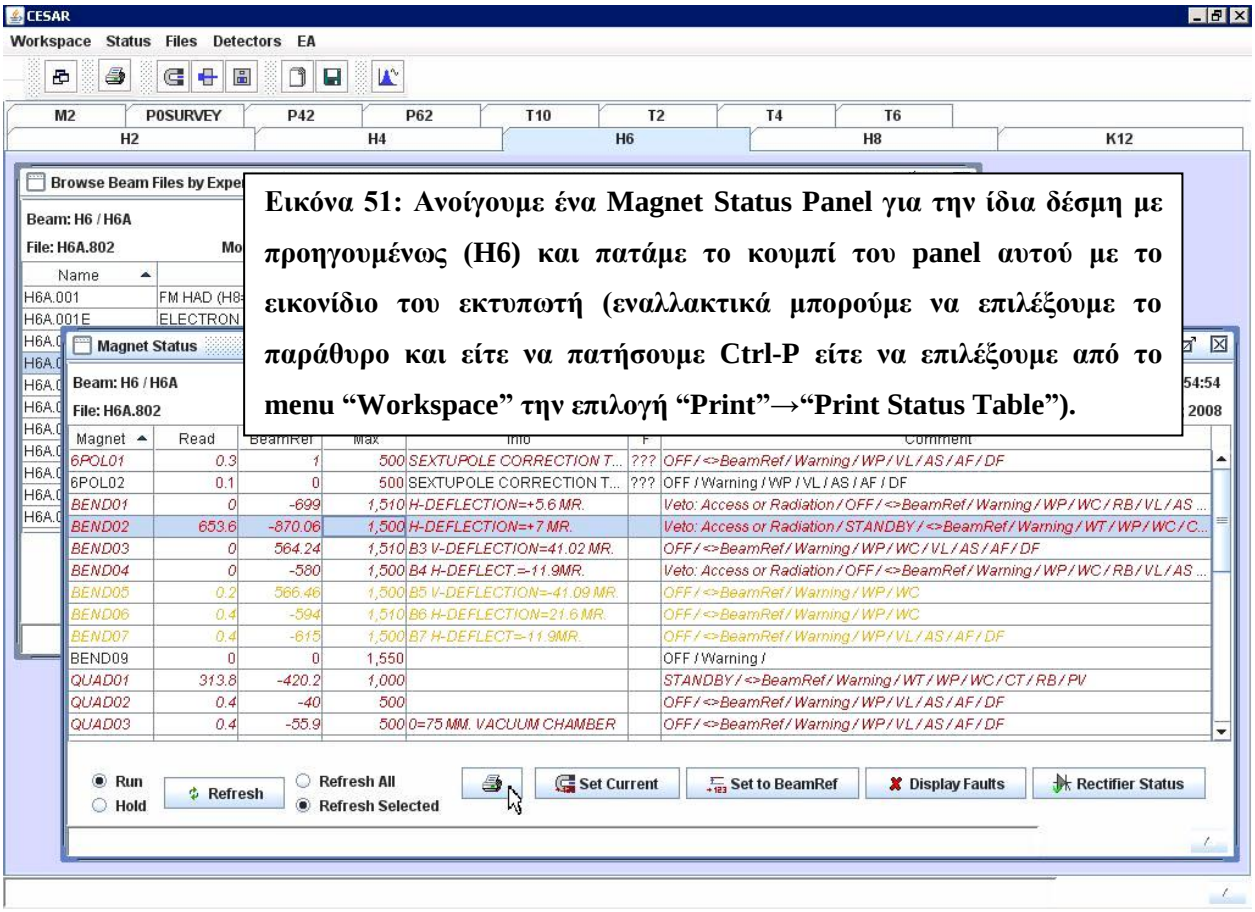
Εικόνα 48: Εμφανίζεται ένας διάλογος που επιτρέπει στο χρήστη να δώσει στοιχεία για το αρχείο που θα δημιουργήσει.



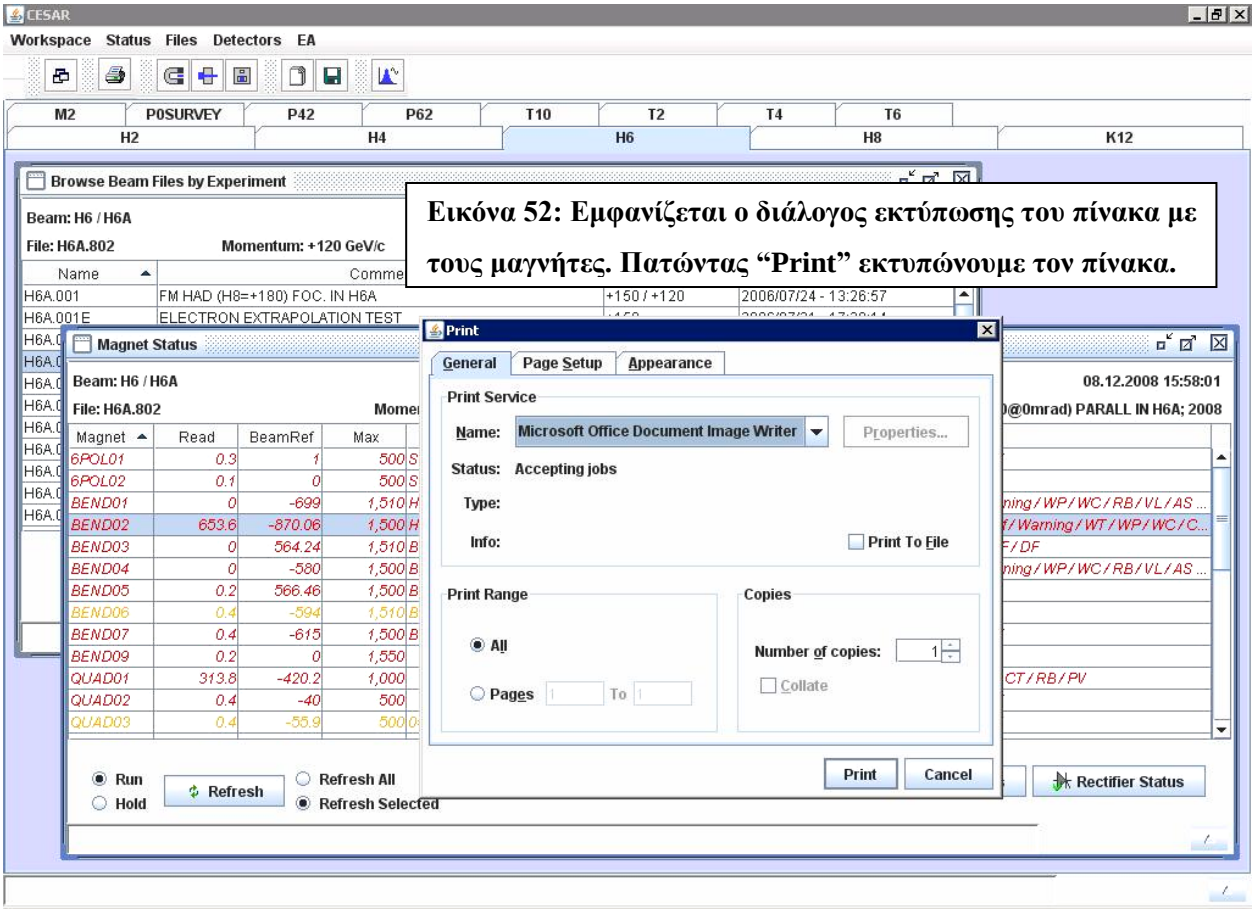
Εικόνα 49: Κλείνουμε τον προηγούμενο διάλογο (με το "Cancel") και επιλέγουμε την ενέργεια "Delete" για το ίδιο και πάλι αρχείο.



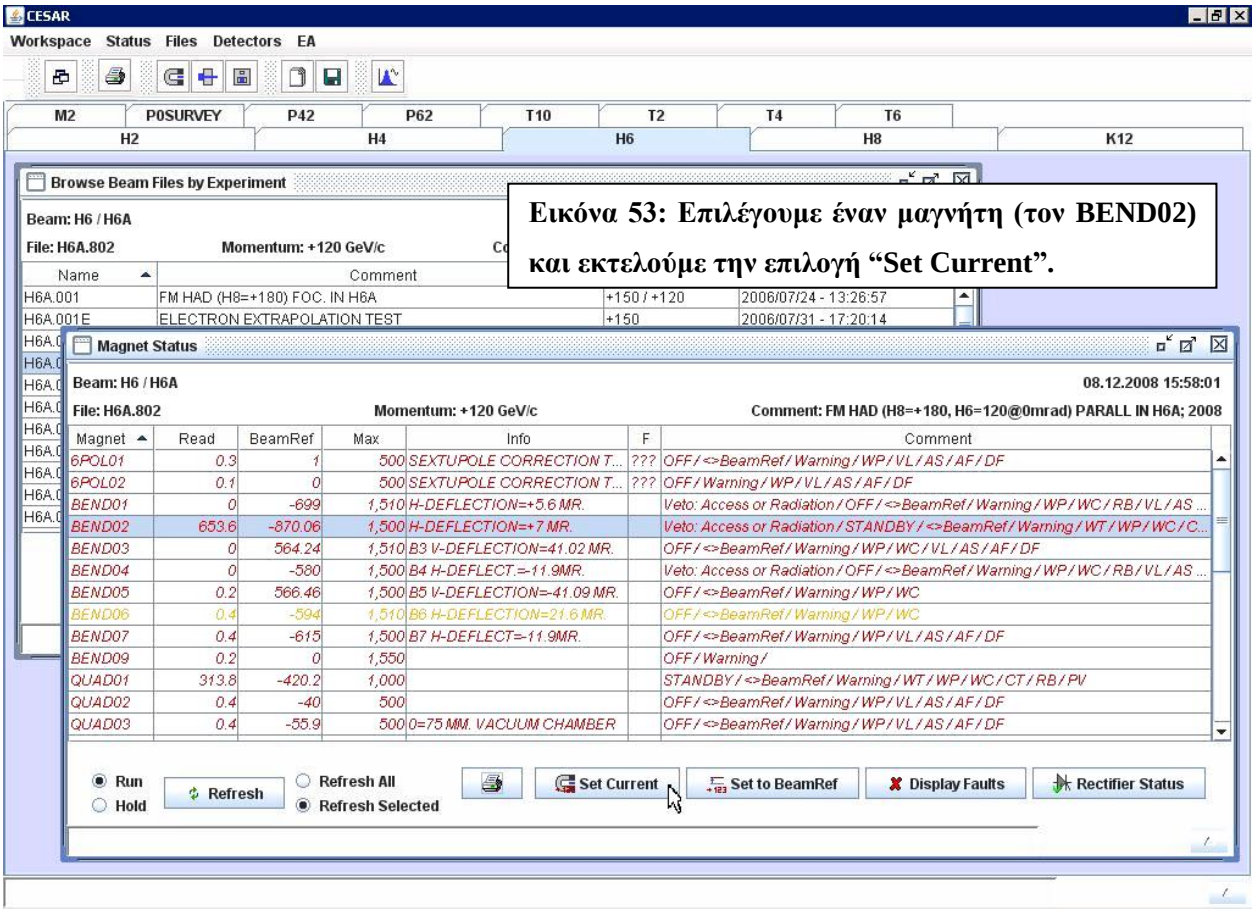
Εικόνα 50: Εμφανίζεται ένας διάλογος επιβεβαίωσης της ενέργειας που ρωτάει τον χρήστη αν είναι σίγουρος ότι θέλει να προχωρήσει στην διαγραφή του επιλεγμένου αρχείου. Αν πατήσουμε "Yes" το αρχείο εξαφανίζεται από τον πίνακα. Πατάμε "No".



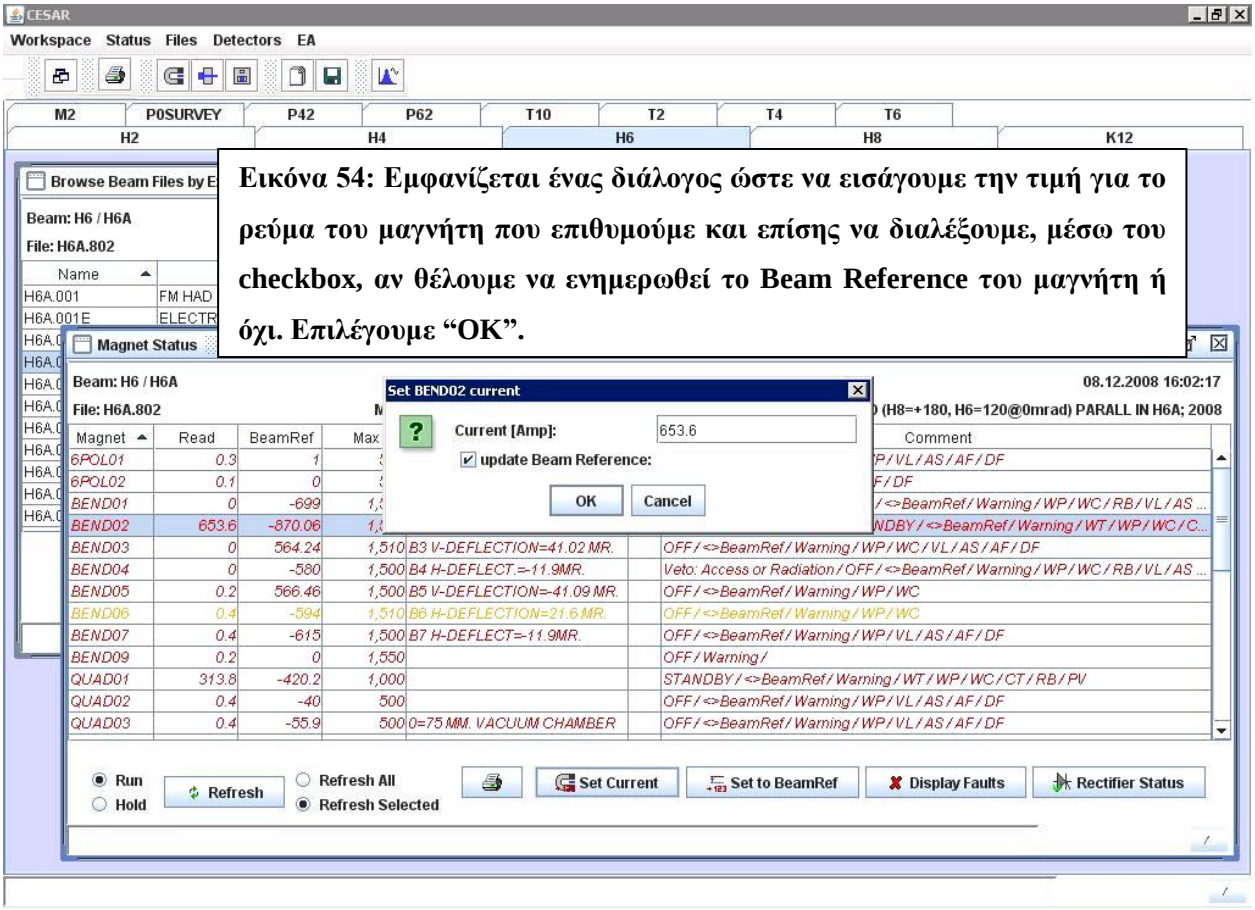
Εικόνα 51: Ανοίγουμε ένα Magnet Status Panel για την ίδια δέσμη με προηγουμένως (H6) και πατάμε το κουμπί του panel αυτού με το εικονίδιο του εκτυπωτή (εναλλακτικά μπορούμε να επιλέξουμε το παράθυρο και είτε να πατήσουμε Ctrl-P είτε να επιλέξουμε από το menu "Workspace" την επιλογή "Print"→"Print Status Table").



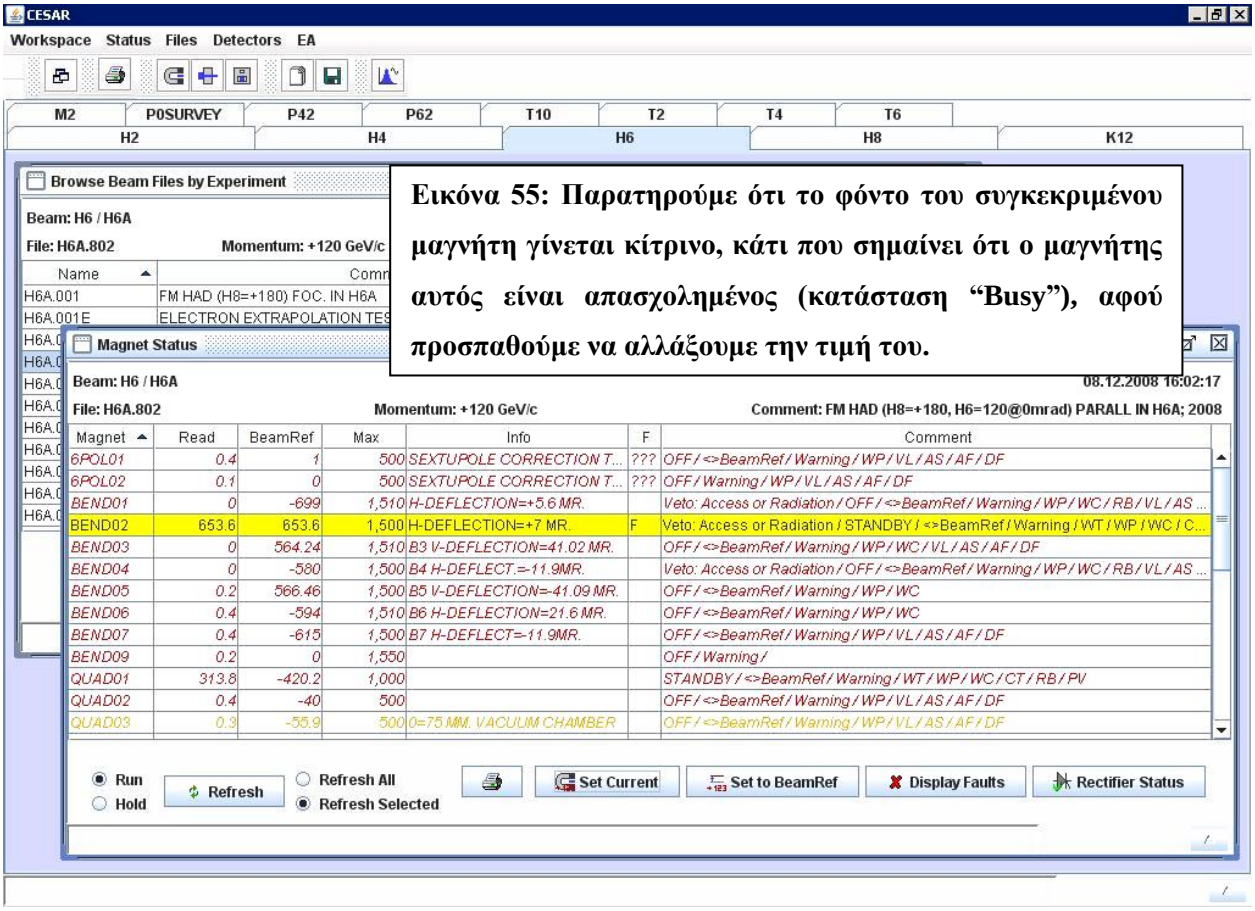
Εικόνα 52: Εμφανίζεται ο διάλογος εκτύπωσης του πίνακα με τους μαγνήτες. Πατώντας "Print" εκτυπώνουμε τον πίνακα.



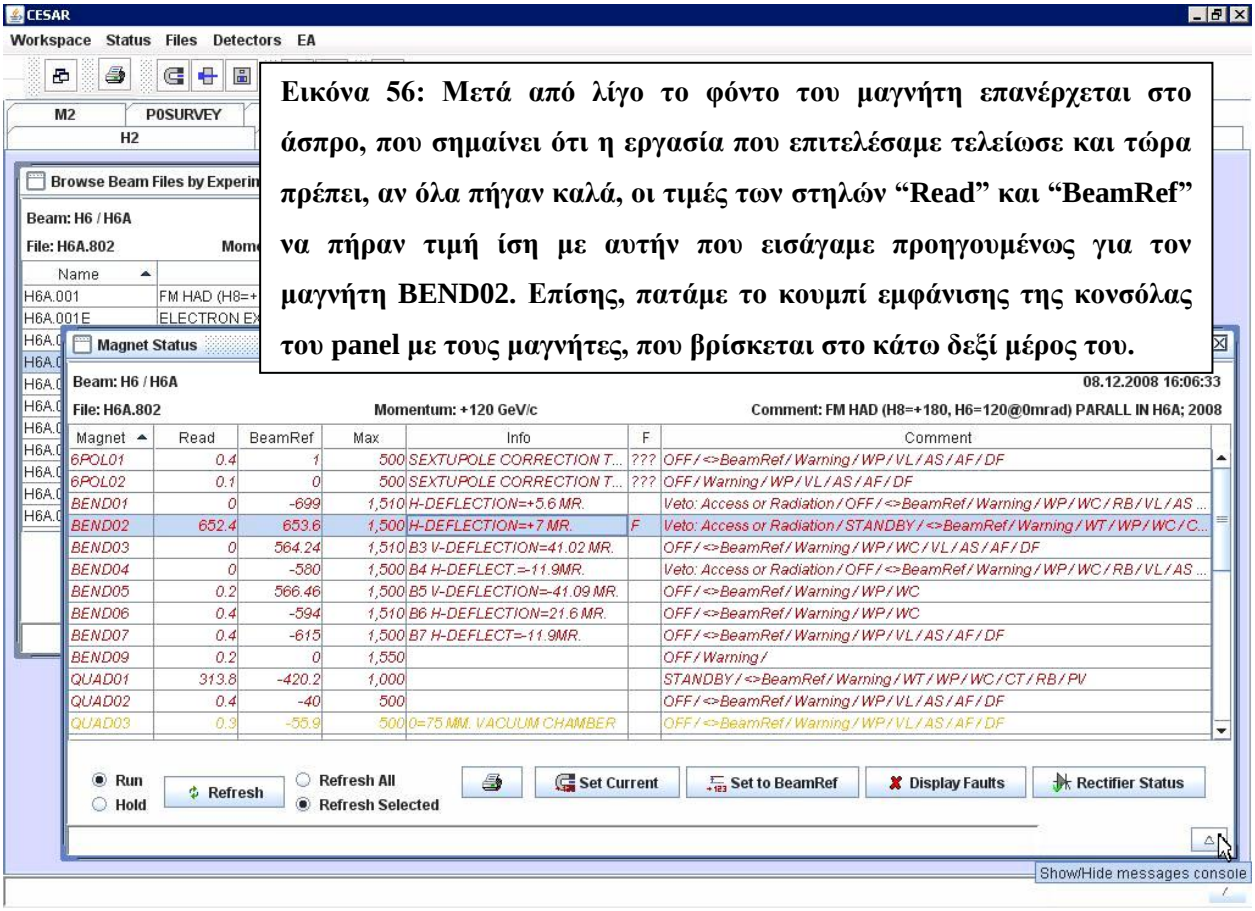
Εικόνα 53: Επιλέγουμε έναν μαγνήτη (τον BEND02) και εκτελούμε την επιλογή "Set Current".



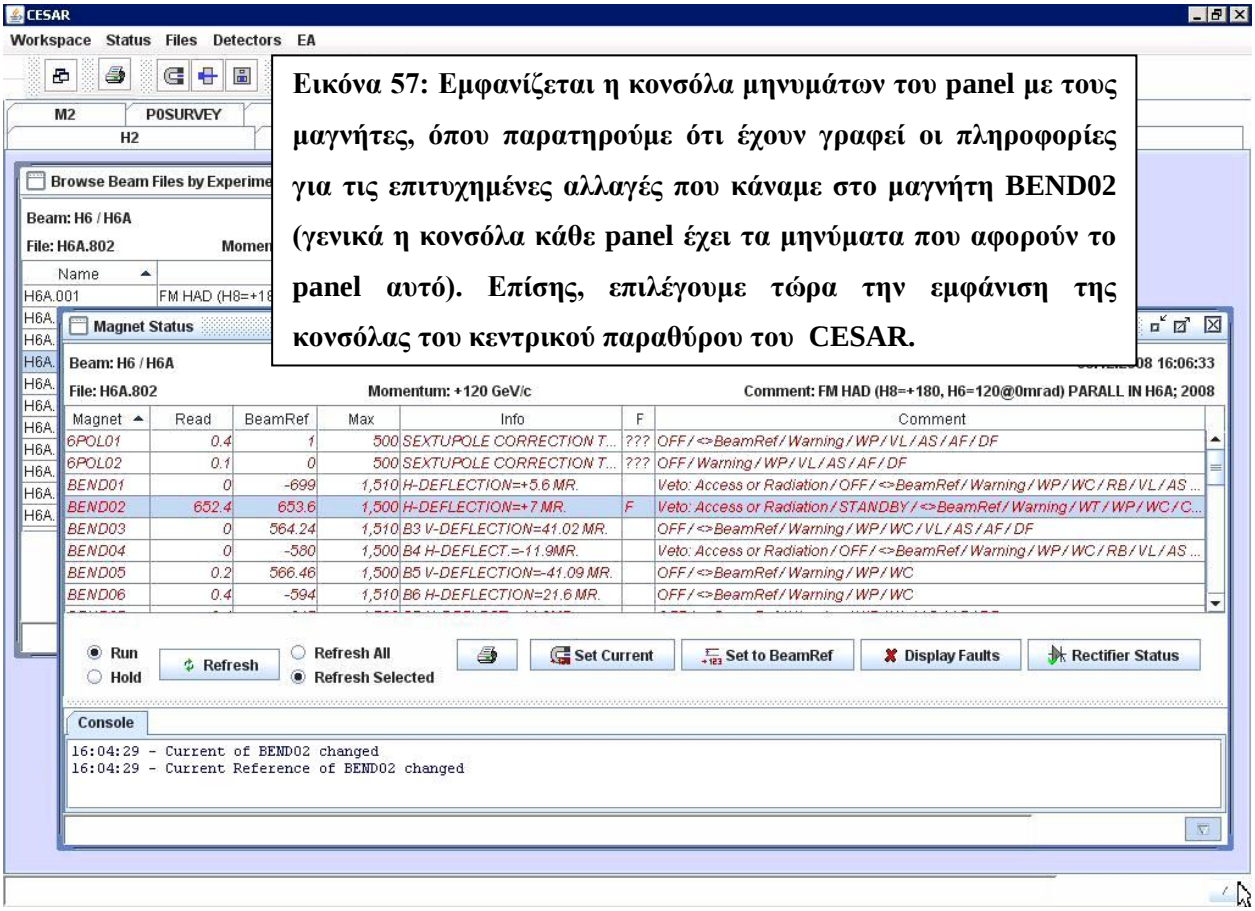
Εικόνα 54: Εμφανίζεται ένας διάλογος ώστε να εισάγουμε την τιμή για το ρεύμα του μαγνήτη που επιθυμούμε και επίσης να διαλέξουμε, μέσω του checkbox, αν θέλουμε να ενημερωθεί το Beam Reference του μαγνήτη ή όχι. Επιλέγουμε "OK".



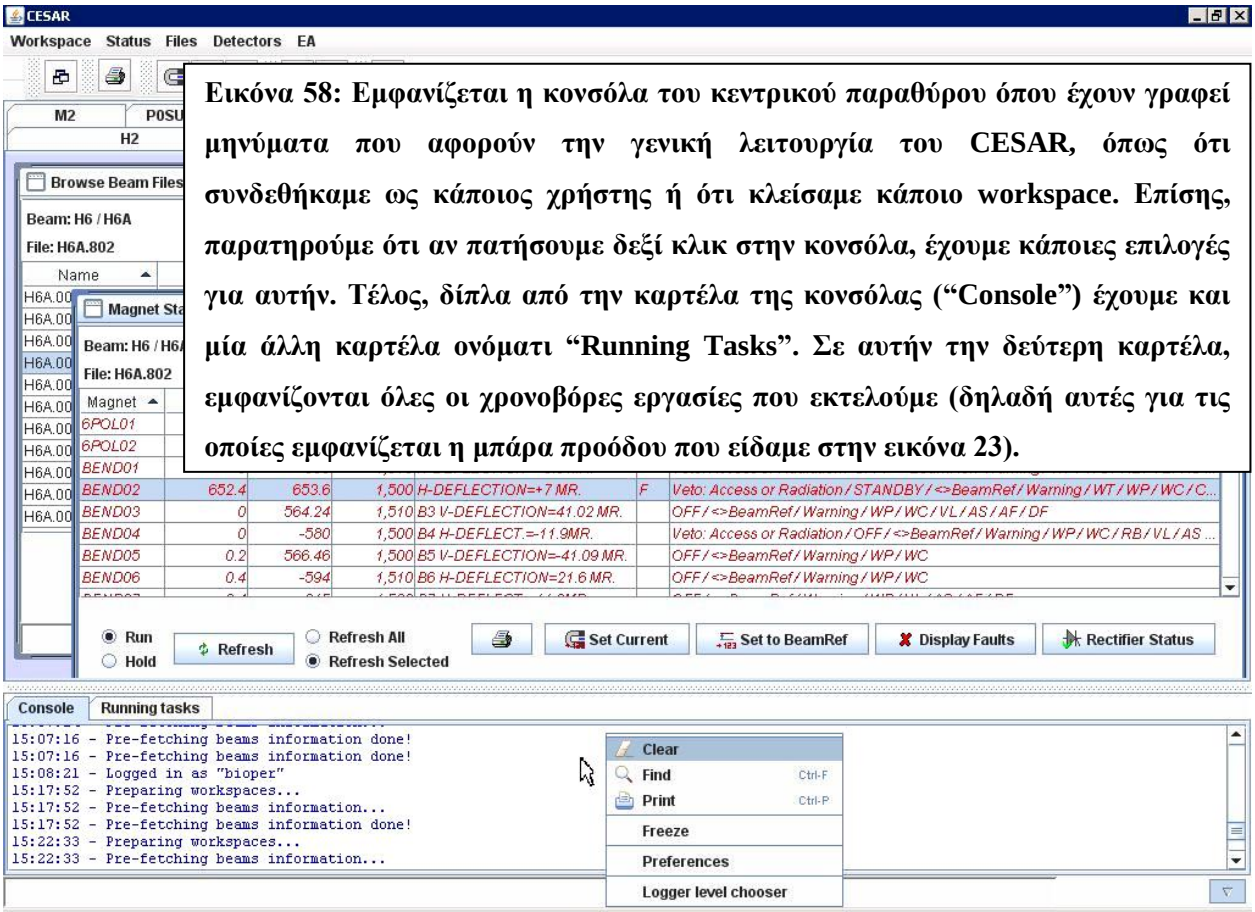
Εικόνα 55: Παρατηρούμε ότι το φόντο του συγκεκριμένου μαγνήτη γίνεται κίτρινο, κάτι που σημαίνει ότι ο μαγνήτης αυτός είναι απασχολημένος (κατάσταση "Busy"), αφού προσπαθούμε να αλλάξουμε την τιμή του.



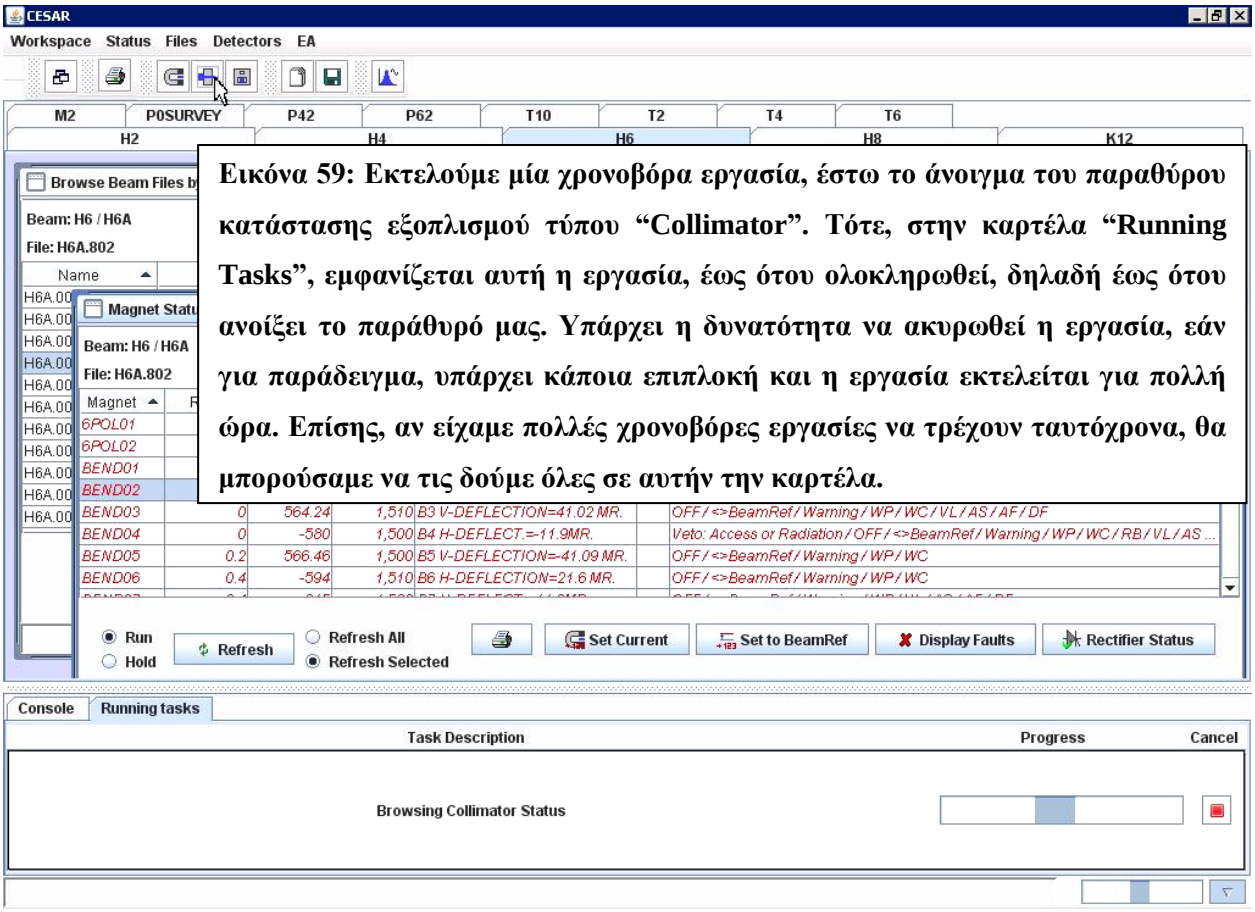
Εικόνα 56: Μετά από λίγο το φόντο του μαγνήτη επανέρχεται στο άσπρο, που σημαίνει ότι η εργασία που επιτελέσαμε τελείωσε και τώρα πρέπει, αν όλα πήγαν καλά, οι τιμές των στηλών "Read" και "BeamRef" να πήραν τιμή ίση με αυτήν που εισάγαμε προηγουμένως για τον μαγνήτη BEND02. Επίσης, πατάμε το κουμπί εμφάνισης της κονσόλας του panel με τους μαγνήτες, που βρίσκεται στο κάτω δεξί μέρος του.



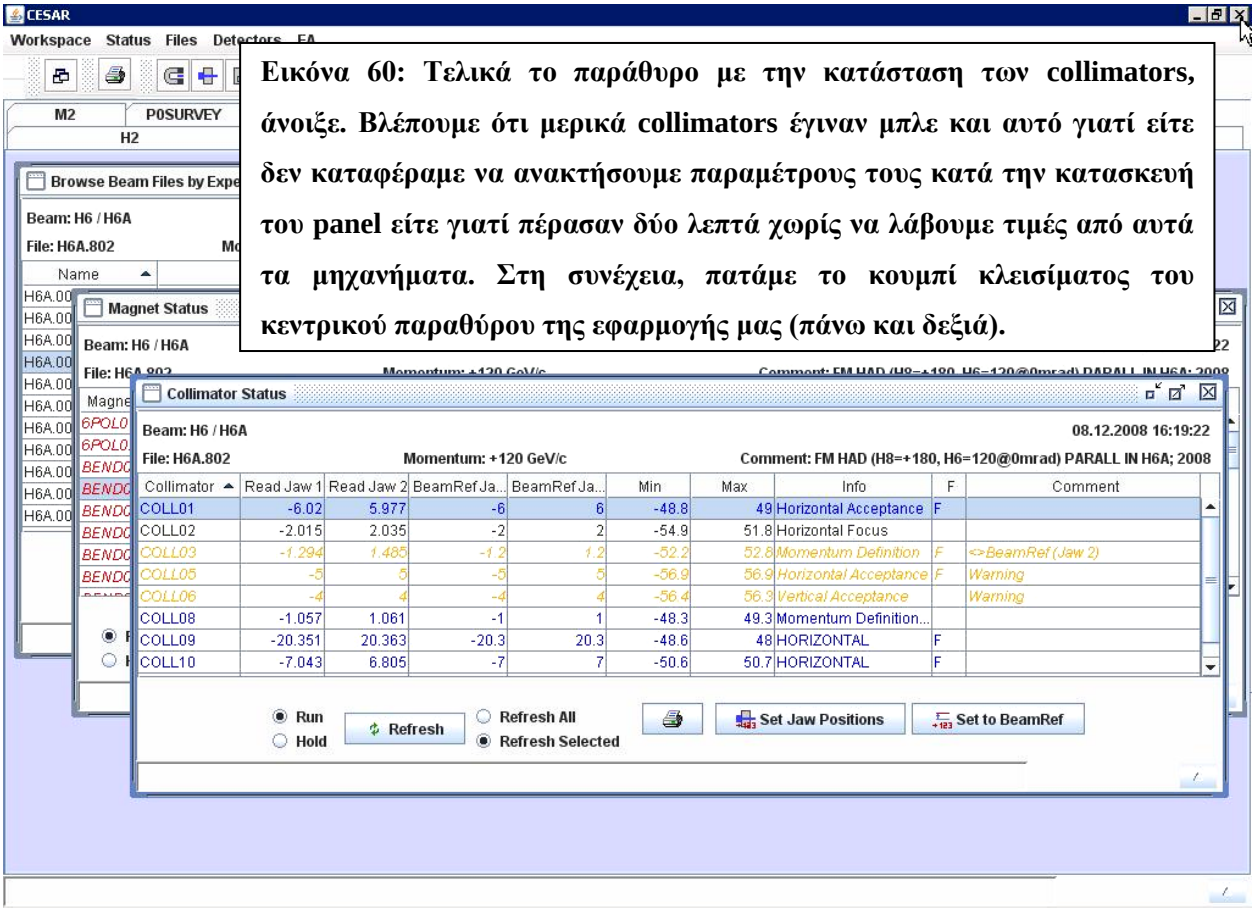
Εικόνα 57: Εμφανίζεται η κονσόλα μηνυμάτων του panel με τους μαγνήτες, όπου παρατηρούμε ότι έχουν γραφεί οι πληροφορίες για τις επιτυχημένες αλλαγές που κάναμε στο μαγνήτη BEND02 (γενικά η κονσόλα κάθε panel έχει τα μηνύματα που αφορούν το panel αυτό). Επίσης, επιλέγουμε τώρα την εμφάνιση της κονσόλας του κεντρικού παραθύρου του CESAR.



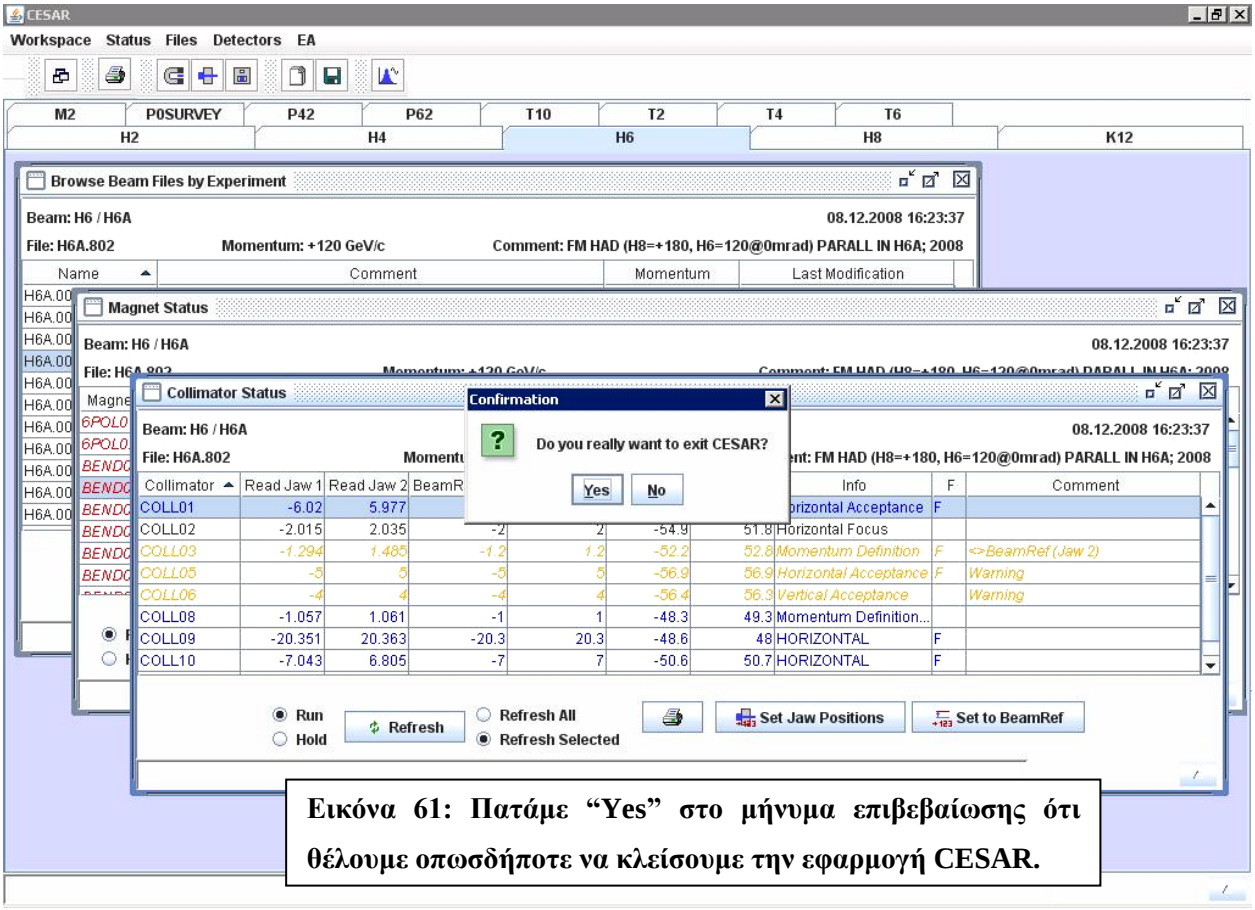
Εικόνα 58: Εμφανίζεται η κονσόλα του κεντρικού παραθύρου όπου έχουν γραφεί μηνύματα που αφορούν την γενική λειτουργία του CESAR, όπως ότι συνδεθήκαμε ως κάποιος χρήστης ή ότι κλείσαμε κάποιο workspace. Επίσης, παρατηρούμε ότι αν πατήσουμε δεξί κλικ στην κονσόλα, έχουμε κάποιες επιλογές για αυτήν. Τέλος, δίπλα από την καρτέλα της κονσόλας ("Console") έχουμε και μία άλλη καρτέλα ονόματι "Running Tasks". Σε αυτήν την δεύτερη καρτέλα, εμφανίζονται όλες οι χρονοβόρες εργασίες που εκτελούμε (δηλαδή αυτές για τις οποίες εμφανίζεται η μπάρα προόδου που είδαμε στην εικόνα 23).



Εικόνα 59: Εκτελούμε μία χρονοβόρα εργασία, έστω το άνοιγμα του παραθύρου κατάστασης εξοπλισμού τύπου "Collimator". Τότε, στην καρτέλα "Running Tasks", εμφανίζεται αυτή η εργασία, έως ότου ολοκληρωθεί, δηλαδή έως ότου ανοίξει το παράθυρό μας. Υπάρχει η δυνατότητα να ακυρωθεί η εργασία, εάν για παράδειγμα, υπάρχει κάποια επιπλοκή και η εργασία εκτελείται για πολλή ώρα. Επίσης, αν είχαμε πολλές χρονοβόρες εργασίες να τρέχουν ταυτόχρονα, θα μπορούσαμε να τις δούμε όλες σε αυτήν την καρτέλα.



Εικόνα 60: Τελικά το παράθυρο με την κατάσταση των collimators, άνοιξε. Βλέπουμε ότι μερικά collimators έγιναν μπλε και αυτό γιατί είτε δεν καταφέραμε να ανακτήσουμε παραμέτρους τους κατά την κατασκευή του panel είτε γιατί πέρασαν δύο λεπτά χωρίς να λάβουμε τιμές από αυτά τα μηχανήματα. Στη συνέχεια, πατάμε το κουμπί κλεισίματος του κεντρικού παραθύρου της εφαρμογής μας (πάνω και δεξιά).



Εικόνα 61: Πατάμε "Yes" στο μήνυμα επιβεβαίωσης ότι θέλουμε οπωσδήποτε να κλείσουμε την εφαρμογή CESAR.

8

Επίλογος

Στην ενότητα αυτή συνοψίζουμε τα αποτελέσματα τις διπλωματικής εργασίας και παραθέτουμε κάποιες μελλοντικές επεκτάσεις που έχουν νόημα και ενδιαφέρον να υλοποιηθούν.

8.1 Σύνοψη και συμπεράσματα

Γενικά, η εργασία μας είχε ως αποτέλεσμα την παραγωγή μιας ολοκληρωμένης client-side εφαρμογής, με την βοήθεια της οποίας οι χρήστες (φυσικοί του CERN) μπορούν να ελέγξουν εύκολα τις δέσμες των σωματιδίων του Βόρειου Τμήματος του επιταχυντή SPS.

Για αυτό τον σκοπό, είχαμε ως αφετηρία ένα συγκεκριμένο σύνολο κώδικα, ο οποίος ήταν ήδη έτοιμος. Αυτός ο κώδικας είναι διαφόρων ειδών. Για παράδειγμα, είχαμε API, το οποίο επιτρέπει άμεση πρόσβαση στον εξοπλισμό της δέσμης (hardware) και στα αρχεία της δέσμης (server). Επίσης, είχαμε κώδικα από προηγούμενες εκδόσεις του CESAR, που μας βοήθησε να αντιληφθούμε την λογική της εφαρμογής μας. Σε αυτό συνέβαλαν και Φυσικοί αλλά και Μηχανικοί Λογισμικού που δουλεύουν στο CERN. Τέλος, είχαμε έτοιμα σύνολα προγραμμάτων, γραμμένα από άλλους προγραμματιστές, τα οποία ήταν καλό να επαναχρησιμοποιήσουμε. Και αυτό γιατί έτσι είχαμε λιγότερο κώδικα να γράψουμε και επίσης, είναι επιθυμητό να είναι κοινός ο κώδικας που παράγει ένα συγκεκριμένο αποτέλεσμα στα πλαίσια του group του CERN, όπου δουλέψαμε.

Το πιο σημαντικό και επίπονο κομμάτι στην διαδικασία παραγωγής ήταν ο σχεδιασμός της εφαρμογής αυτής. Και αυτό διότι το CESAR πρέπει να είναι εύκολο στην συντήρησή του, η οποία απαιτεί πάντα μεγαλύτερο όγκο εργασίας από την ίδια την παραγωγή ενός λογισμικού. Έτσι, πρέπει οι Μηχανικοί Λογισμικού του CERN να μπορούν με ευκολία να αλλάζουν, να προσθέτουν και να αφαιρούν στοιχεία της εφαρμογής μας κατά τη διάρκεια των επόμενων χρόνων λειτουργίας του CESAR, τα οποία αναμένεται να είναι αρκετά. Ο καλός σχεδιασμός της εφαρμογής μας έγινε με την βοήθεια του εργαλείου Spring, το οποίο με σωστή χρήση, μας επιτρέπει να οργανώσουμε μια εφαρμογή. Επίσης, ιδιαίτερη σημασία είχε να οργανώσουμε σωστά τις κλάσεις μας και τις μεθόδους τους και ακόμα, να τις τεκμηριώσουμε με σχόλια και javadoc, ώστε προγραμματιστές του CERN να μπορούν να καταλάβουν την δομή και λειτουργία του κώδικά μας.

Επίσης, εφαρμογές όπως η δική μας, πρέπει να είναι φιλικές προς το χρήστη. Έτσι, δώσαμε βάση σε μικρά πράγματα, τα οποία όμως κάνουν την εφαρμογή πολύ εύχρηστη στους Φυσικούς. Ακόμα, έπρεπε η εφαρμογή μας να μην έχει πολλές διαφορές με τις προηγούμενες εκδόσεις του CESAR στο επίπεδο της εμφάνισης, της βασικής λειτουργίας και της οργάνωσης.

Τέλος, καταφέραμε η εφαρμογή μας να είναι όσο πιο αποδοτική γίνεται κυρίως με την χρήση της μνήμης. Αυτό ήταν ένα σημαντικό στοιχείο, καθώς το CESAR μπορεί να τρέχει σε κάποιον υπολογιστή επί μέρες και δεν πρέπει να «καταναλώνει» μνήμη του υπολογιστή για ανύπαρκτους λόγους, δηλαδή δεν πρέπει να υπάρχουν διαρροές μνήμης (memory leaks).

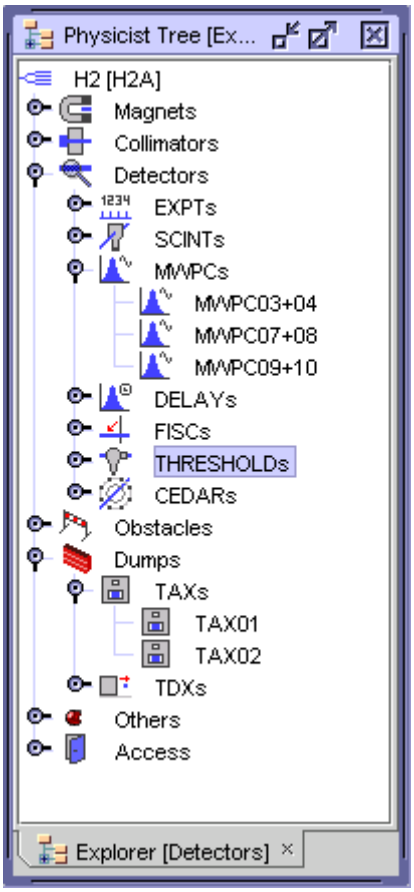
8.2 Μελλοντικές επεκτάσεις

Είναι φανερό πως η επέκταση του CESAR μπορεί να γίνει εύκολα με την προσθήκη κάποιου ή κάποιων modules. Αυτά τα modules, τα οποία παράγονται ανεξάρτητα από το CESAR, μπορεί να έχουν διάφορες χρησιμότητες. Δυο modules που εμείς σχεδιάσαμε είναι τα Αρχεία Δέσμης και η Κατάσταση Εξοπλισμού. Τα Αρχεία Δέσμης τα ενσωματώσαμε πλήρως στην εφαρμογή CESAR ενώ την Κατάσταση Εξοπλισμού την ενσωματώσαμε για λίγα μόνο είδη εξοπλισμού. Και αυτό διότι, το framework της εφαρμογής που σχεδιάσαμε είναι σε φάση ελέγχου από τους υπεύθυνους του project, οπότε, μόνο όταν αυτός ο έλεγχος τελειώσει, θα προστεθούν όλα τα απαιτούμενα στοιχεία του CESAR σε αυτό.

Επίσης, πολλά άλλα modules, που έχουν ήδη παραχθεί από προγραμματιστές του CERN και υπήρχαν σε προηγούμενες εκδόσεις του CESAR, δεν τα έχουμε ενσωματώσει ακόμα στην δική μας εφαρμογή. Αυτά δεν τα παρουσιάσαμε ούτε τα ενσωματώσαμε στο γενικό framework της εφαρμογής μας, καθώς, αφ’ ενός δεν θέλαμε να συμπεριλάβουμε στην Διπλωματική Εργασία αυτή, την δουλεία άλλων εργαζομένων του οργανισμού και αφ’ ετέρου

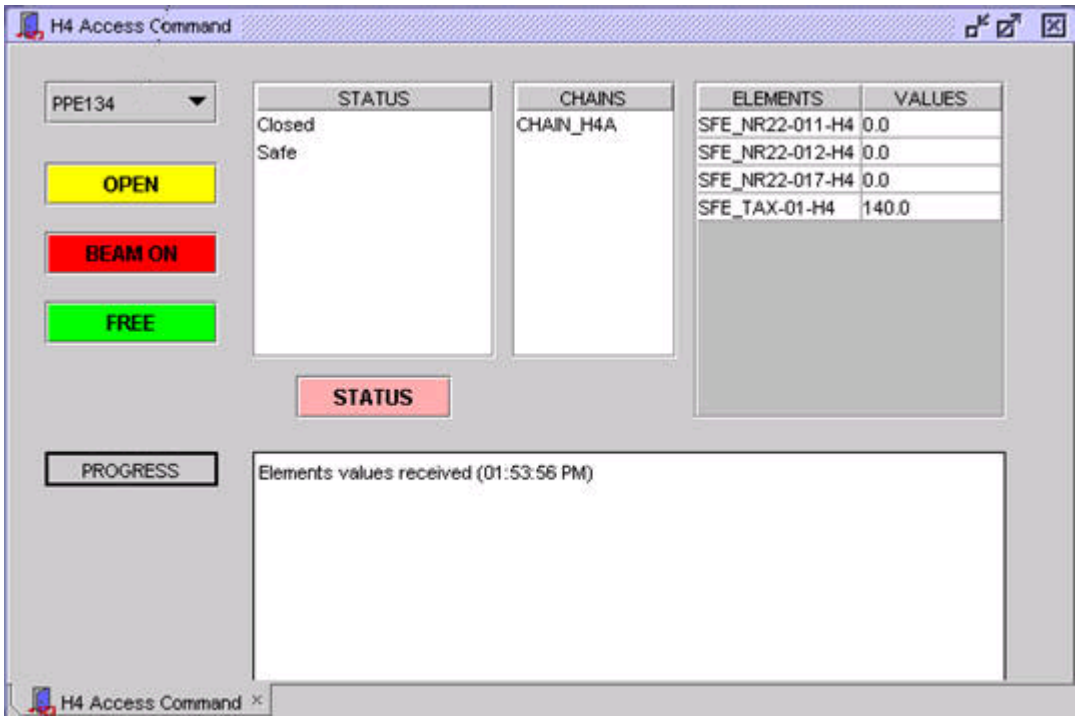
το framework που σχεδιάσαμε είναι σε φάση ελέγχου, όπως έχουμε αναφέρει. Μερικά μόνο από τα modules αυτά είναι:

- Ένα δέντρο το οποίο καλούν οι Φυσικοί "Physicist Tree" και παρουσιάζει όλα τα είδη εξοπλισμού. Από το δέντρο αυτό μπορεί κάποιος χρήστης να εμφανίσει το παράθυρο Κατάστασης συγκεκριμένου είδους Εξοπλισμού (π.χ. με διπλό αριστερό κλικ στον εξοπλισμό). Το εν λόγω δέντρο, σε παλαιότερες εκδόσεις του CESAR, είχε την μορφή:



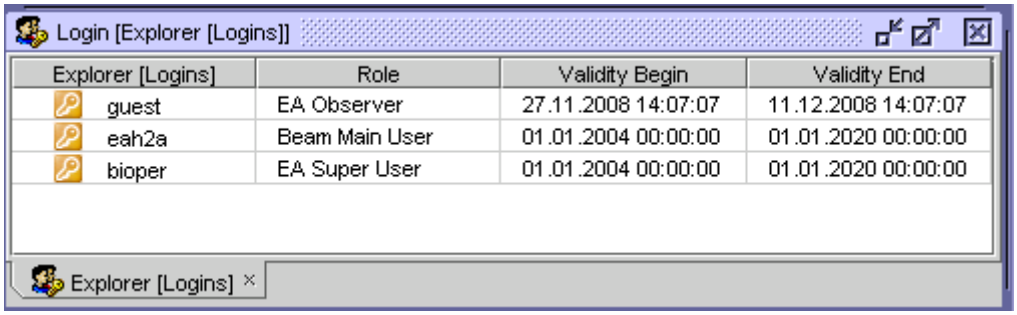
Εικόνα 62: Το δέντρο με τα διάφορα είδη εξοπλισμού της δέσμης H2.

- Ένα panel που ελέγχει την προσβασιμότητα ανθρώπων στην δέσμη (access) και είχε σε παλαιές εκδόσεις του CESAR την εξής μορφή:



Εικόνα 63: Το panel προσβασιμότητας ανθρώπων στην δέσμη H4.

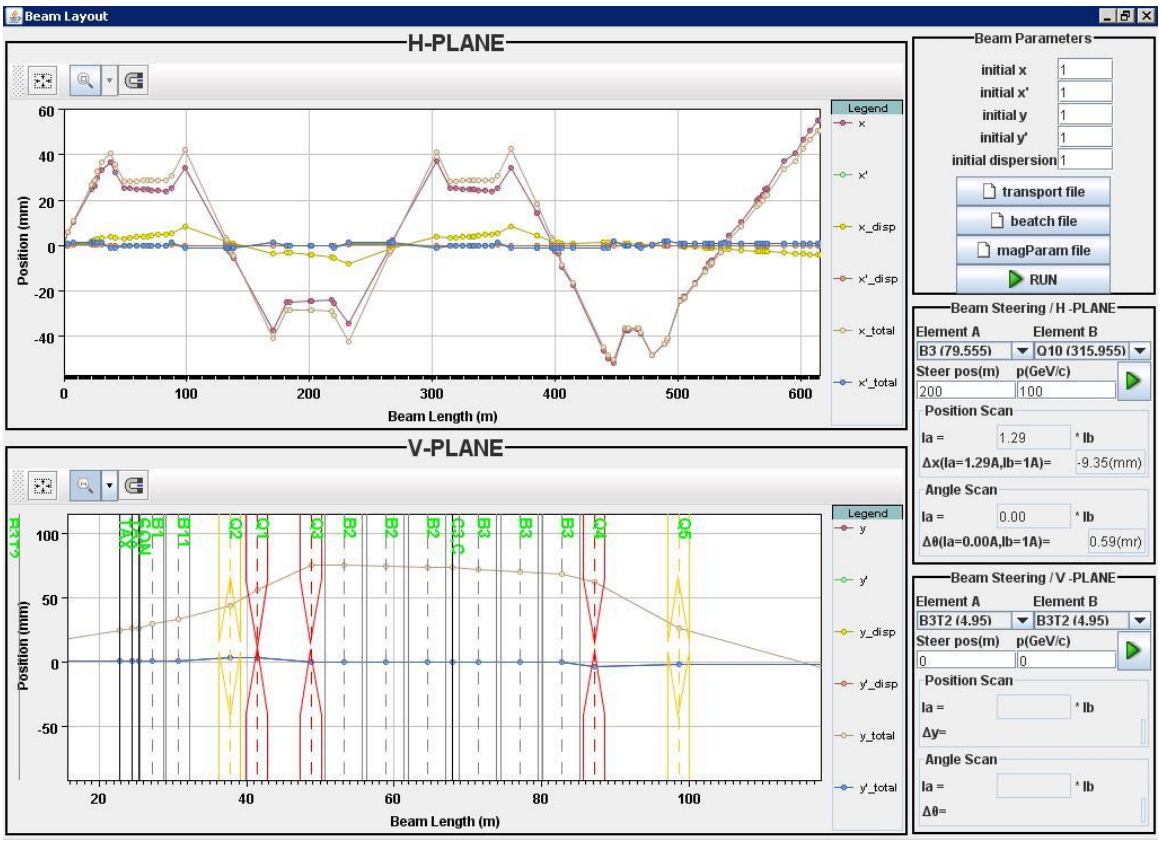
- Όπως έχουμε αναφέρει, μπορούμε να έχουμε στοίβες από logins. Αυτό συμβαίνει όταν κάνουμε πολλές φορές επιτυχημένα logins. Έτσι, κάνοντας logout, περνάμε στον προηγούμενα συνδεδεμένο χρήστη. Συνεπώς, καλό είναι να υπάρχει ένας πίνακας που να μας δείχνει όλους τους χρήστες που έχουν κάνει login στο παρελθόν κατά την διάρκεια της χρήσης του CESAR. Αυτός ο πίνακας ήταν σε προηγούμενη έκδοση του CESAR, ο εξής:



Εικόνα 64: Ο πίνακας με όλα τα logins που έχουν γίνει.

Εκτός από όσα είχε το CESAR στο παρελθόν, όμως, μπορούμε να προσθέσουμε στην εφαρμογή μας και νέα στοιχεία. Για το ποια μπορεί να είναι αυτά, υπεύθυνοι να απαντήσουν είναι περισσότερο οι ίδιοι οι Φυσικοί που χρησιμοποιούν το CESAR. Από συνομιλίες, ωστόσο, που είχαμε με αυτούς, συμπεράναμε ότι το CESAR θα μπορούσε κάποια στιγμή να αποκτήσει ένα ακόμα φιλικότερο και απλουστευμένο περιβάλλον προς τον χρήστη, δίνοντας περισσότερο έμφαση στην εποπτεία των διαφόρων δεσμών. Θα μπορούσε δηλαδή να έχει γραφήματα τα οποία να μας δείχνουν με ακρίβεια την

γεωμετρία που έχει μία δέσμη στο χώρο των τριών διαστάσεων και ταυτόχρονα ο χρήστης να μπορεί, αλλάζοντας διάφορες παραμέτρους του εξοπλισμού, να βλέπει αυτόματα τις αλλαγές που επέρχονται σε αυτήν την γεωμετρία. Παράδειγμα μιας τέτοιας λειτουργίας είναι ένα panel, το οποίο εμείς, επίσης, αναπτύξαμε, αλλά όχι στα πλαίσια του CESAR και το οποίο δείχνει ως εξής:



Εικόνα 65: Το panel που δείχνει την γεωμετρία μιας δέσμης.

Το παραπάνω panel διαβάζει κάποια (ειδικά διαμορφωμένα) αρχεία κειμένου τύπου txt, τα οποία περιέχουν τις παραμέτρους της δέσμης καθώς και κάποιες αρχικές τιμές για αυτήν. Στη συνέχεια, σχεδιάζει την μορφή που έχει η δέσμη σε δύο γραφήματα. Τα δύο αυτά γραφήματα απεικονίζουν την δέσμη των σωματιδίων αλλά και άλλες παραμέτρους της, στο οριζόντιο και κατακόρυφο επίπεδο. Στο κάτω γράφημα έχουμε κάνει zoom-in και έχουμε εμφανίσει ειδικά σχήματα που αντιστοιχούν στον εξοπλισμό κατά μήκος της δέσμης.

Κάποιο ανάλογο panel θα μπορούσε να υπάρχει στο CESAR, χωρίς όμως να εισάγουμε τα δεδομένα της δέσμης μέσω ειδικών txt αρχείων, που είναι αρκετά πολύπλοκη διαδικασία για τον μέσο χρήστη, αλλά να παίρνουμε τα δεδομένα αυτά από την διαμόρφωση που έχει γίνει μέσω CESAR στον πραγματικό εξοπλισμό. Έτσι, παρακολουθώντας συνεχώς τον πραγματικό εξοπλισμό και τις ενέργειες του χρήστη, θα μπορούμε να ενημερώνουμε τα γραφήματά μας, ώστε να έχει ο χρήστης και εποπτική δυνατότητα στις αλλαγές που κάνει στην δέσμη του.

9

Βιβλιογραφία

1 V. Baggiolini, P. Bailly, B. Chauchaix, F. Follin, J. Fullerton, P. Malacarne, L. Mateos-Miret; L. Pereira: "The CESAR Project using J2EE for Accelerator Controls", Proceedings of ICALEPCS2003, Gyeongju, Korea

2 Sharon Lackey: "ACNET vs EPICS at Fermilab", Accelerator Physics and Technology Seminar, Fermilab, 14/3/2006

3 http://www.theserverside.com/, Άρθρο "Introduction to the Spring Framework", Rod Johnson

4 http://www.wikipedia.org/, Διάφορα άρθρα

5 http://www.springframework.org/, documentation και forum

6 https://swingx.dev.java.net/

7 http://www.cern.ch/

8 http://ab-div-atb-ea.web.cern.ch/ab-div-atb-ea/

9 http://java.sun.com/docs/books/tutorial/

Παράρτημα

ΓΛΩΣΣΑΡΙ

Accessor	Μια μέθοδος που παρέχει πρόσβαση σε ένα πεδίο, δηλαδή μια getter ή setter μέθοδος
Action Handling Method	Μέθοδος που καλείται όταν εκτελείται μία ενέργεια
Aspect Oriented Programming	Προγραμματιστική τεχνική σύμφωνα με την οποία η λειτουργία κάποιου προγράμματος χωρίζεται σε τμήματα και πριν ή μετά την εκτέλεση τους μπορούμε να επέμβουμε
Beam File	Αρχείο Δέσμης
Bean (ή JavaBean)	Ειδικά διαμορφωμένο αντικείμενο κλάσης. Στην εργασία μας, μιλάμε για κλάσεις που αποτελούν το μοντέλο κάποιου στοιχείου με ορισμένα χαρακτηριστικά, π.χ. το μοντέλο ενός μηχανήματος ή αρχείου
Browser	Πρόγραμμα ή οποιοδήποτε στοιχείο που εκκινεί μία εφαρμογή ή υποσύστημα
Business Logic	Η ανταλλαγή πληροφοριών μεταξύ μιας βάσης δεδομένων και ενός user interface
CERN	Ερευνητικός Οργανισμός στον τομέα Φυσικής Στοιχειωδών Σωματιδίων
CESAR	Πρόγραμμα ελέγχου των δεσμών των σωματιδίων στο Βόρειο Τμήμα του επιταχυντή SPS στο CERN
Client-side εφαρμογή	Εφαρμογή που έχει να κάνει με την πλευρά των χρηστών και την διεπαφή αυτών με την καρδιά της εφαρμογής
Compile	Μεταγλώττιση κώδικα
Console	Μια κονσόλα στην οποία μπορούμε να γράφουμε διάφορα μηνύματα
Control System	Σύστημα Ελέγχου
CVS (Concurrent Versions System)	Σύστημα που αποθηκεύει όλες τις εκδόσεις κάποιων αρχείων
Debug	Διαδικασία εύρεσης σφαλμάτων σε κώδικα
Dependency Injection	Η εξάρτηση ενός αντικειμένου μιας κλάσης από κάποιο εξωτερικό στοιχείο. Είναι υποπερίπτωση της προγραμματιστικής τεχνικής Inversion of Control
EJB (Enterprise JavaBean)	Ένα server-side μοντέλο για την ενσωμάτωση ενός business logic σε μία

	εφαρμογή
Equipment Status	Κατάσταση Εξοπλισμού
Framework	Το γενικό περιβάλλον και η δομή ενός προγράμματος σε Η/Υ. Μπορεί επίσης να είναι και κάποια τεχνική σχεδιασμού ενός προγράμματος σε Η/Υ.
Getter μέθοδος	Μια μέθοδος η οποία επιστρέφει ένα πεδίο μιας κλάσης
Handler	Μια μέθοδος η οποία χειρίζεται κάποιο στοιχείο, π.χ. ένα μηχάνημα ή ένα αρχείο
Helper	Μια κλάση η οποία είναι βοηθητική για ένα γενικό σκοπό και έχει συγκεκριμένη λειτουργία
Highlighter	Ένας τύπος απεικόνισης κάποιου στοιχείου (π.χ. ενός πίνακα) με ένα ξεχωριστό στυλ, συνήθως χρωματικό
IDE (Integrated Development Environment)	Ένα περιβάλλον εργασίας για προγραμματιστές που διευκολύνει την ανάπτυξη προγραμμάτων γραμμένων σε διάφορες γλώσσες, όπως Java, C++ κ.α.
Interface (ή User Interface)	Η διεπαφή των χρηστών με ένα σύστημα μέσω μιας εφαρμογής
Interface	Η διαπροσωπία της γλώσσας Java
Inversion of Control	Προγραμματιστική τεχνική κατά την οποία η ροή ελέγχου αντιστρέφεται συγκριτικά με την πιο συνηθισμένη περίπτωση
Javadoc	Είδος τεκμηρίωσης κώδικα γραμμένου σε Java
Launcher	Πρόγραμμα ή οποιοδήποτε στοιχείο που εκκινεί μία εφαρμογή
Logging Output	Γραπτά μηνύματα σε μία εφαρμογή που αναφέρονται στο χρήστη ή στους προγραμματιστές της
Login	Εγγραφή κάποιου χρήστη σε ένα σύστημα, που ταυτοποιεί χρήστες
Logout	Έξοδος κάποιου χρήστη από ένα σύστημα, που ταυτοποιεί χρήστες
Menu	Ένα μενού με διάφορες ενέργειες σε μία εφαρμογή
Monitoring	Παρακολούθηση κάποιων παραμέτρων ή τιμών
Node	Γραμμή (στοιχείο) ενός πίνακα
Panel	Το περιεχόμενο ενός παραθύρου σε μία εφαρμογή
Password	Κωδικός σε έναν λογαριασμό χρήστη
POJO (Plain Old Java Object)	Αντικείμενο κλάσης Java το οποίο χαρακτηρίζεται ως «καθαρό», δηλαδή δεν είναι κάποιο Enterprise JavaBean
Pop-up παράθυρο	Ένα παράθυρο που εμφανίζεται αυτόματα κάποια χρονική στιγμή κατά την εκτέλεση μιας εφαρμογής
Progress bar	Μπάρα προόδου μιας χρονοβόρας διεργασίας σε μία εφαρμογή
Renderer	Τύπος απεικόνισης ενός στοιχείου (π.χ. ενός

	πίνακα) εφαρμόζοντας κάποια εικόνα στο στοιχείο αυτό
Screenshot	Η εικόνα που εμφανίζει η οθόνη του υπολογιστή κάποια χρονική στιγμή
Server-side εφαρμογή	Εφαρμογή που έχει να κάνει με κάποιον server, δηλαδή με κάποια βάση δεδομένων
Setter μέθοδος	Μια μέθοδος που αλλάζει την τιμή ενός πεδίου μιας κλάσης
Splash screen	Μια εικόνα που εμφανίζεται στο προσκήνιο κατά την διάρκεια εκκίνησης μιας εφαρμογής, έως ότου αυτή φορτωθεί
Standalone εφαρμογή	Εφαρμογή που δεν χρειάζεται κάποια σύνδεση στο WEB για να τρέξει
Toolbar	Μια μπάρα με κουμπιά, τα οποία αντιστοιχούν σε ενέργειες
Username	Όνομα χρήστη σε έναν λογαριασμό χρήστη
WEB	Το Διαδίκτυο με την σημερινή του μορφή
WEB-based εφαρμογή	Εφαρμογή που τρέχει χρησιμοποιώντας κάποια σύνδεση στο WEB
Workspace	Ανεξάρτητος χώρος εργασίας σε μία εφαρμογή