ΑΡΙΣΤΟΤΕΛΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάλυση και Εφαρμογή Αρχιτεκτονικής Ανίχνευσης, Απομόνωσης και Αντιμετώπισης Βλαβών στον Νανοδορυφόρο AcubeSAT

Διπλωματική Εργασία

Θωμάς Κυριάκος Πραβινός

Α.Ε.Μ.: 9937

Επιβλέπων Καθηγητής: Χατζόπουλος Αλκιβιάδης, Α.Π.Θ.

Θεσσαλονίκη, Ιούνιος 2024

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κατάλογος Κώδικα

Μεταφράσεις ξενόγλωσσης ορολογίας

Boolean: Δυαδικός

Checking status: Κατάσταση ελέγχου

Constructor: Κατασκευαστής

Cubesat: Νανοδορυφόρος συγκεκριμένου μεγέθους και μορφής

Delta: Διαφορά, Δέλτα

Definition: Ορισμός

Detection: Εντοπισμός

Directory: Κατάλογος

Enumeration: απαρίθμηση

Event: Συμβάν

Expected value: Αναμενόμενη τιμή

Limit: Όριο

Mask: Μάσκα (κάλυψης δεδομένων)

Monitoring: Παρακολούθηση

Monitoring definition: Ορισμός παρακολούθησης

Parameter: Παράμετρος

Recovery: Ανάκτηση

Report: Αναφορά

Services: Υπηρεσίες

Standard: Πρότυπο

Threshold: Κατώφλι

Transition list: Λίστα μεταβάσεων

Transition reporting delay: Καθυστέρηση αναφοράς μετάβασης

Unit: Μονάδα

Unit tests: Δοκιμές μονάδας

Vector: Διάνυσμα

Ακρωνύμια

ADCS: Attitude Determination and Control System (Σύστημα Προσδιορισμού και Ελέγχου Θέσης)

CDR: Critical Design Review (Κρίσιμη Ανασκόπηση Σχεδίασης)

COMMS: Communications (Επικοινωνίες)

COTS: Commercial Off-The-Shelf (Εμπορικά Διαθέσιμα Προϊόντα)

ECSS: European Cooperation for Space Standardization (Ευρωπαϊκό Πρότυπο)

EPS: Electrical Power System (Σύστημα Ηλεκτρικής Ισχύος)

ETL: Embedded Template Library (Βιβλιοθήκη Ενσωματωμένων Προτύπων)

FMEA: Failure Modes and Effects Analysis (Ανάλυση Τρόπων και Επιδράσεων Αποτυχίας)

FDIR: Fault Detection, Isolation and Recovery (Εντοπισμός, Απομόνωση και Διόρθωση Αποτυχιών)

ID: Identifier (Αναγνωριστικό)

IDE: Integrated Development Environment (Ενσωματωμένο Περιβάλλον Ανάπτυξης)

LOC: Lab-on-a-Chip (Εργαστήριο σε Τσιπ)

MCU: Microcontroller Unit (Μικροελεγκτής)

OBDH: On-Board Data Handling (Διαχείριση Δεδομένων Επί του Σκάφους)

OBSW: On-Board Software (Λογισμικό Επί του Σκάφους)

OBC: On-Board Computer (Υπολογιστής Επί του Σκάφους)

OPS: Operations (Λειτουργίες)

PUS: Packet Utilization Standard (Πρότυπο Αξιοποίησης Πακέτων)

SU: System Unit (Μονάδα Συστήματος)

SYE: System Engineering (Μηχανική Συστήματος)

TC: Telecommand (Τηλε-εντολή)

TM: Telemetry (Τηλεμετρία)

Περίληψη

Η παρούσα διπλωματική εργασία επικεντρώνεται στην υλοποίηση μιας αρχιτεκτονικής Ανίχνευσης, Απομόνωσης και Αντιμετώπισης Βλαβών (FDIR) που βασίζεται στο Ευρωπαϊκό Πρότυπο Αξιοποίησης Πακέτων ECSS. Η υλοποίηση σε C++ περιλαμβάνει την ανάπτυξη της υπηρεσίας παρακολούθησης παραμέτρων ST[12], η οποία επιτρέπει τον ορισμό, την ενεργοποίηση και την απενεργοποίηση ελέγχων για την παρακολούθηση διαφόρων παραμέτρων του δορυφόρου. Η αρχιτεκτονική της υπηρεσίας είναι πλήρως διαμορφώσιμη, επιτρέποντας την προσθήκη, τροποποίηση ή αφαίρεση ελέγχων ανάλογα με τις ανάγκες της αποστολής. Επιπλέον, η εργασία περιγράφει αναλυτικά τη δομή του κώδικα της υπηρεσίας ST[12], τις βασικές μεθόδους που χρησιμοποιεί η υπηρεσία, καθώς και τη λογική πίσω από τους διάφορους τύπους ελέγχων. Το προτεινόμενο σύστημα FDIR αποσκοπεί στην αύξηση της αξιοπιστίας και των πιθανοτήτων επιτυχίας της αποστολής AcubeSAT των φοιτητών του Αριστοτελείου Πανεπιστημίου Θεσσαλονίκης.

Abstract

Thomas Kyriakos Pravinos

Analysis and Implementation of Fault Detection, Isolation and Recovery Architecture for the AcubeSAT nanosatellite

Supervisor: Prof. Alkiviadis Hatzopoulos

Aristotle University of Thessaloniki

19 June 2024

This thesis focuses on the implementation of a Fault Detection, Isolation, and Remediation (FDIR) architecture based on the European ECSS Packet Utilization Standard. The C++ implementation includes the development of the parameter monitoring service ST[12], which allows for defining, enabling, and disabling checks to monitor various on-board parameters. The architecture of the service is fully configurable, allowing for the addition, modification, or removal of checks according to mission needs. Additionally, the thesis provides a detailed description of the code structure of the ST[12] service, the main methods it employs, and the logic behind the various types of checks. The proposed FDIR system aims to enhance the reliability and success probabilities of the AcubeSAT mission conducted by the students of Aristotle University of Thessaloniki.

Ευχαριστίες

Η παρούσα διπλωματική εργασία είναι το αποτέλεσμα μιας προσπάθειας που δεν θα ήταν δυνατή χωρίς την υποστήριξη και την καθοδήγηση ορισμένων ανθρώπων, στους οποίους οφείλω ιδιαίτερες ευχαριστίες.

Πρωτίστως, θα ήθελα να εκφράσω τις θερμότερες ευχαριστίες μου προς τον επιβλέποντα καθηγητή μου, κ. Αλκιβιάδη Χατζόπουλο, ο οποίος με έφερε σε επαφή με το συγκεκριμένο αντικείμενο, μου προσέφερε την ευκαιρία να ασχοληθώ με αυτό και τελικά με καθοδήγησε προς την ολοκλήρωση της παρούσας εργασίας.

Επιπλέον , θα ήθελα να ευχαριστήσω τους συμφοιτητές μου που συμμετέχουν σε αυτό το πρότζεκτ για τη συνεργασία και το έργο που έχουν προσφέρει και χρησιμοποιείται σε αυτήν την εργασία, ιδιαίτερα στον Αθανάσιο Θεοχάρη, τον Κωνσταντίνο Καναβούρα και την ομάδα του OBC.

Τέλος, θερμές ευχαριστίες οφείλω στην οικογένειά μου, στους φίλους μου και στον σημαντικό μου άνθρωπο, για την αμέριστη αγάπη, υποστήριξη και κατανόηση που μου επέδειξαν καθ' όλη τη διάρκεια εκπόνησης αυτής της διπλωματικής εργασίας.

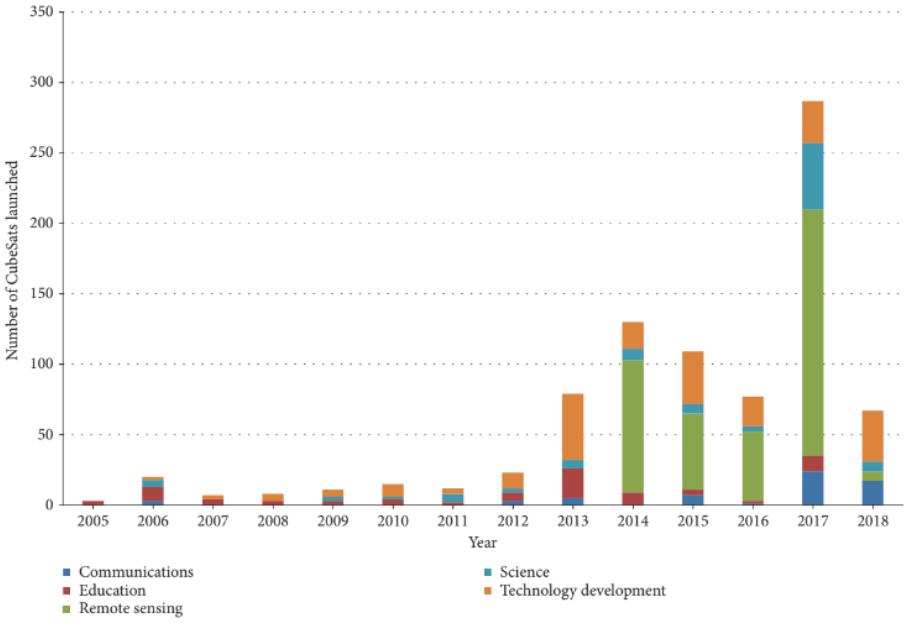
Νιώθω μεγάλη χαρά που μέσω αυτής της εργασίας είχα την ευκαιρία να συνεργαστώ με δημιουργικούς και ενθουσιώδεις ανθρώπους. Είμαι περήφανος που συνέβαλα σε αυτό το φιλόδοξο εγχείρημα και ανυπομονώ για τις μελλοντικές εξελίξεις και προκλήσεις.

1 Εισαγωγή

1.1 Επιστημονική Περιοχή

Η παρούσα διπλωματική εργασία εστιάζει στην Αεροδιαστημική Μηχανική και ειδικότερα στην ανάλυση και υλοποίηση αρχιτεκτονικής λογισμικού Ανίχνευσης, Απομόνωσης και Αντιμετώπισης Βλαβών. Οι αντίξοες συνθήκες του διαστήματος και η περιορισμένη δυνατότητα ανθρώπινης παρέμβασης καθιστούν την εφαρμογή αυτής της ευέλικτης αρχιτεκτονικής καίρια για την επιτυχία της αποστολής.

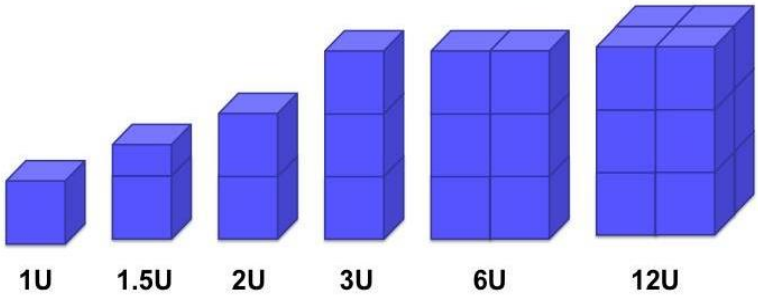
Σύμφωνα με τον ευρύτερο ορισμό τους, οι δορυφόροι τίθενται σε τροχιά γύρω από ουράνια σώματα για να εκτελέσουν ποικίλες λειτουργίες όπως οι τηλεπικοινωνίες, η παρατήρηση της Γης, η επιστημονική έρευνα ή και εκτέλεση μη-επανδρωμένων αποστολών. Τα παραπάνω αποτελούν πηγή δεδομένων για την καθημερινότητα του ανθρώπου με εφαρμογές στις μεταφορές, την πλοήγηση, την πρόσβαση στο διαδίκτυο, την πρόγνωση του καιρού και την γενικότερη εξέλιξη της σύγχρονης τεχνολογίας. Συνεπώς το φαινόμενο αυτό έχει οδηγήσει στην αύξηση του αριθμού και των ειδών των διαστημικών αποστολών. Την παρούσα στιγμή βρίσκονται εν ενεργεία περισσότεροι από 9.500 δορυφόροι στο διαστημικό χώρο [1], γεγονός το οποίο αναδεικνύει την αυξανόμενη ανάγκη βελτιστοποίησης των συστημάτων και ταυτόχρονα μείωσης των κοστών. Όσον αφορά την μοίρα των δορυφόρων μετά το πέρας της αποστολής τους, υπάρχουν δύο περιπτώσεις, εξαρτώμενες από τον αρχικό σχεδιασμό της αποστολής. Στη μία περίπτωση, παραμένουν στο διάστημα μετά το τέλος της αποστολής, επιβραδύνοντας σταδιακά, ώσπου τελικά εισέρχονται στην ατμόσφαιρα και καίγονται, ενώ στην άλλη παραμένουν στο διάστημα ως διαστημικά σκουπίδια [2].



Εικόνα 1.1: CubeSats που εκτοξεύθηκαν ανά έτος και εφαρμογή από το 2005 έως τις 31 Μαΐου 2018. [2]

Όπως είναι ευρέως γνωστό, οι αφιλόξενες συνθήκες που επικρατούν στο διάστημα, όπως οι ακραίες θερμοκρασίες, οι συνθήκες κενού, η ακτινοβολία, οι μηχανικές πιέσεις και η τεράστια απόσταση από την Γη καθιστούν την ανάπτυξη διαστημικής τεχνολογίας μια ιδιαίτερη πρόκληση για τον άνθρωπο. Οι σύγχρονες αεροδιαστημικές αποστολές απαιτούν εξειδίκευση σε διάφορους κλάδους μηχανικής όπως η μηχανική συστημάτων, η διασφάλιση προϊόντων, η μηχανική αξιοπιστίας και η συναρμολόγηση, η ενοποίηση και η επαλήθευση. Αυτή η εργασία εξετάζει συγκεκριμένα την ανίχνευση, την απομόνωση και την ανάκτηση σφαλμάτων, η οποία είναι αναγκαία για τα διαστημικά συστήματα που πρέπει να λειτουργούν αυτόνομα όσο είναι σε τροχιά.

Οι δορυφόροι αυτοί ποικίλλουν σημαντικά σε μέγεθος, από ογκώδεις γεωστατικούς δορυφόρους επικοινωνίας βάρους πολλών τόνων μέχρι πολύ μικρότερες μορφές, όπως οι νανοδορυφόροι. Αυτοί αντιπροσωπεύουν μια ειδική κατηγορία της οικογένειας των δορυφόρων, αποτέλεσμα της εξέλιξης της τεχνολογίας και της ανάγκης διεξαγωγής ουσιωδών διαστημικών αποστολών με χαμηλότερο κόστος και χαρακτηριστικά μικρότερο μέγεθος, συγκριτικά με αυτά των κλασσικών δορυφόρων. Οι CubeSats ξεχωρίζουν ως ένα βασικό υποσύνολο νανοδορυφόρων, έχοντας τυποποιημένο αρθρωτό σχεδιασμό με με δομικές μονάδες διαστάσεων 10 × 10 × 10 εκατοστά, που αναφέρονται ως "Units" με βάρος έως 1,33 κιλά η καθεμία. Οι τυπικές κατηγορίες νανοδορυφόρων προκύπτουν από στοιβάδες τέτοιων μονάδων με αποτέλεσμα δορυφόρους 1U, 1.5U, 2U, 3U, 6U και 12U [3]. Το εσωτερικό των CubeSats αποτελείται κατά βάση από εμπορικά ηλεκτρονικά εξαρτήματα χαμηλού κόστους, τα λεγόμενα Commercial Off-The-Shelf (COTS). Αξίζει να σημειωθεί πως τα CubeSats τοποθετούνται ως δευτερεύοντα φορτία σε εκτοξεύσεις μεγαλύτερων δορυφόρων και η διάρκεια δράσης τους κυμαίνεται από 1 ως 3 έτη.

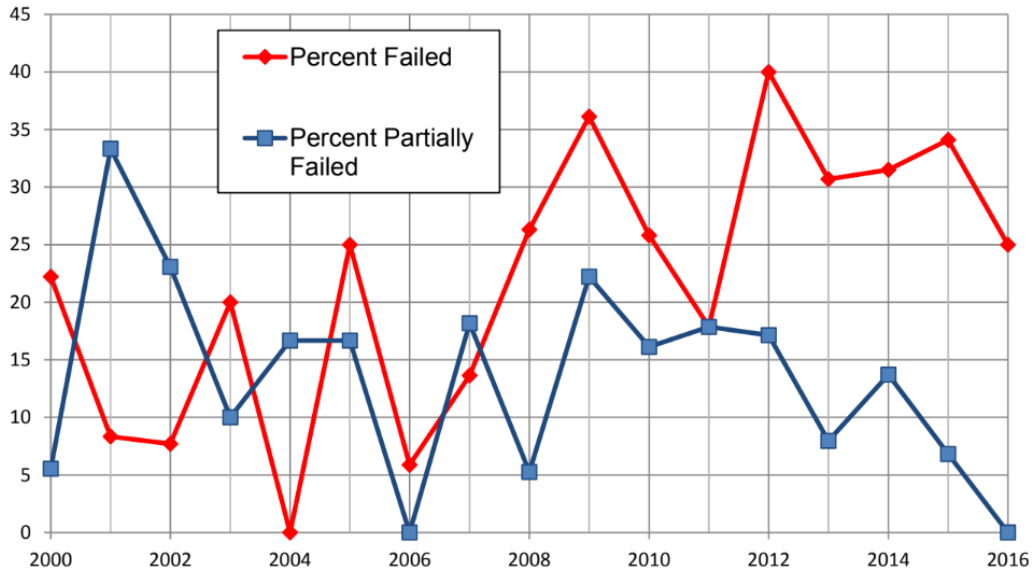


Εικόνα 1.2: Τα τυπικά μεγέθη Cubesat. [3]

Εικόνα 1.3: 3U Πλατφόρμα Cubesat της Endurosat. [4]

1.2 Σκοπός και συνεισφορά της διπλωματικής

Η παρούσα εργασία στοχεύει στην ανάλυση της υπάρχουσας έρευνας και εργασίας στο κομμάτι της αξιοπιστίας για τον νανοδορυφόρο AcubeSAT και την υλοποίηση τμήματος του λογισμικού που ευθύνεται για την Ανίχνευση, Απομόνωση και Αντιμετώπιση Βλαβών στα υποσυστήματα του. Κατά μέσο όρο, το 41,3% των CubeSats που εκτοξεύτηκαν μεταξύ 2000 και 2016 κατέληξαν είτε σε ολική είτε σε μερική αποτυχία της αποστολής, με το ποσοστό αυτό να αυξάνεται τα τελευταία χρόνια, με σχεδόν μία στις δύο αποστολές να αντιμετωπίζουν κάποιο βαθμό αποτυχίας. [5]



Εικόνα 1.4: Ποσοστό μικρών δορυφορικών αποστολών που απέτυχαν ολικά ή εν μέρει. [5]

Οι βασικές αιτίες αποτυχίας των αποστολών CubeSat σχετίζονται αρχικά με το περιορισμένο χρονοδιάγραμμα κατασκευής το οποίο οδηγεί σε παραλείψεις και σφάλματα κατά τη φάση του σχεδιασμού. Επιπλέον η αυξημένη πολυπλοκότητα του λογισμικού που χρησιμοποιείται σε αυτά τα συστήματα δεν συμβαδίζει με την ταχύτητα εξέλιξης των μεθόδων επαλήθευσης και επικύρωσης αυτού. Αυτό το χάσμα μεταξύ της πολυπλοκότητας του λογισμικού και των μεθόδων ελέγχου μπορεί να οδηγήσει σε μη ανιχνεύσιμα σφάλματα που προκαλούν αποτυχία της αποστολής. Ακόμη ένας σημαντικός παράγοντας που οδηγεί σε αποτυχία είναι η αυξανόμενη πολυπλοκότητα των ίδιων των αποστολών με το πέρασμα των χρόνων. Οι πρώιμες αποστολές είχαν απλούστερους στόχους, όπως η μετάδοση ενός σήματος ή η δοκιμή εξαρτημάτων στις συνθήκες διαστήματος ενώ οι σημερινοί CubeSats εκτελούν πολύ πιο φιλόδοξες αποστολές, όπως η μελέτη της διαστημικής ακτινοβολίας, ή η διεξαγωγή βιολογικών πειραμάτων που αποσκοπεί στην παρακολούθηση της συμπεριφοράς οργανισμών στο διάστημα. Συνεπώς η μεγαλύτερη πολυπλοκότητα οδηγεί και σε ανάλογα σημαντικότερη πιθανότητα αποτυχίας κάποιου τμήματος της αποστολής.

Ειδικά σε πανεπιστημιακές αποστολές, η έλλειψη εμπειρίας, τεχνογνωσίας και πόρων σε συνδυασμό με όλα τα προαναφερόμενα, μειώνουν αισθητά τον βαθμό αξιοπιστίας των νανοδορυφόρων.

Για αυτό το λόγο χρησιμοποιούνται δομημένα εργαλεία αξιολόγησης και βελτιστοποίησης για την ενίσχυση της αξιοπιστίας και του ποσοστού επιτυχίας των αποστολών, όπως Failure Mode and Effect Analysis (FMEA) και Fault Detection, Isolation and Recovery (FDIR) [6]. Επιπλέον δίνεται μεγάλη βαρύτητα σε ζητήματα αξιοπιστίας κατά το πρώτα στάδια του σχεδιασμού της αποστολής και ακολουθούνται ανοιχτά πρότυπα διαστημικής τεχνολογίας.

Στόχος αυτής της εργασίας είναι να ακολουθήσει τα ευρωπαϊκά πρότυπα ECSS και να ενσωματώσει την έρευνα των υπόλοιπων μελών της αποστολής στα μέτρα του νανοδορυφόρου AcubeSAT. Στις επόμενες σελίδες, πραγματοποιείται μια αναλυτική περιγραφή της υπηρεσίας ST[12] του ευρωπαϊκού προτύπου ECSS που συνδέεται με την παρακολούθηση παραμέτρων, αλλά και ο τρόπος που όλες οι απαραίτητες προδιαγραφές του προτύπου ECSS, μετατράπηκαν σε λειτουργικό λογισμικό για την υλοποίηση της υπηρεσίας.

Συγκεκριμένα, η συνεισφορά της διπλωματικής αυτής εργασίας εντοπίζεται στα εξής σημεία:

Εφαρμογή του Προτύπου ECSS στην Υπηρεσία ST[12]: Εξηγεί πώς οι προδιαγραφές του προτύπου ECSS εφαρμόζονται στην υπηρεσία ST[12], μετατρέποντάς τις σε πρακτικό λογισμικό. Αυτή η διαδικασία υλοποίησης προσφέρει ένα παράδειγμα καλών πρακτικών για την ανάπτυξη αξιόπιστων και προσαρμόσιμων συστημάτων λογισμικού σε διαστημικά οχήματα.

Πρακτική Εφαρμογή και Δοκιμή Κώδικα: Περιλαμβάνει την ανάπτυξη και τη δοκιμή του κώδικα σε C++, με ιδιαίτερη έμφαση στην παρακολούθηση και τον έλεγχο παραμέτρων. Η χρήση unit tests για την επαλήθευση της σωστής λειτουργίας του λογισμικού αποτελεί ένα σημαντικό κομμάτι για τη διασφάλιση της ποιότητας και της αξιοπιστίας του συστήματος.

Συμβολή στην Έρευνα της Ομάδας AcubeSAT: Η εργασία συνεισφέρει στη συλλογική προσπάθεια της ομάδας φοιτητών του Αριστοτελείου Πανεπιστημίου Θεσσαλονίκης, παρέχοντας χρήσιμα εργαλεία για την επιτυχή ολοκλήρωση της αποστολής AcubeSAT. Τέλος, η εργασία θέτει τις βάσεις για μελλοντικές βελτιώσεις και επεκτάσεις του συστήματος FDIR, προτείνοντας κατευθύνσεις για περαιτέρω έρευνα και ανάπτυξη στον τομέα της αεροδιαστημικής μηχανικής και της αξιοπιστίας διαστημικών αποστολών.

1.3 Δομή της διπλωματικής

Η εργασία ξεκινάει με μία σύντομη περιγραφή της αποστολής AcubeSAT στο Κεφάλαιο 2, στα πλαίσια της οποίας πραγματοποιείται η συγγραφή αυτής της εργασίας. Στη συνέχεια αναλύονται τα κύρια πρότυπα και η φιλοσοφία του FDIR, εστιάζοντας στις εφαρμογές στον νανοδορυφόρο AcubeSAT στο Κεφάλαιο 3. Παρακάτω στο Κεφάλαιο 4, γίνεται περιγραφή της υπηρεσίας ST[12] και της διαδικασίας οργάνωσης και υλοποίησης αυτής σε κώδικα. Στο Κεφάλαιο 5 αναλύεται η υλοποίηση των κεντρικών συναρτήσεων που χρησιμοποιεί η υπηρεσία μαζί με τις δοκιμές που έγιναν πάνω σε αυτές. Τέλος στο Κεφάλαιο 6 παρουσιάζεται μία πρωτότυπη υλοποίηση της λογικής των ελέγχων που θα εκτελεί η υπηρεσία στις παραμέτρους.

2 Η αποστολή AcubeSAT

2.1 Εισαγωγή

Ο νανοδορυφόρος AcubeSat είναι ένα εγχείρημα που βρίσκεται υπό σχεδιασμό ομάδας φοιτητών του Αριστοτελείου Πανεπιστημίου Θεσσαλονίκης από το 2016, και στοχεύει στην εκτέλεση ενός βιολογικού πειράματος. Πιο συγκεκριμένα, η ομάδα έχει σχεδιάσει ένα 3U CubeSat διαστάσεων 340,5x100x100 mm και μάζας 4.2 kg, που θα μελετήσει τις επιπτώσεις της μικροβαρύτητας και της κοσμικής ακτινοβολίας σε ανθρώπινα κύτταρα. Το 2020 το πρότζεκτ έγινε μέρος του προγράμματος `Fly Your Satellite! 3` του Ευρωπαϊκού Οργανισμού Διαστήματος (European Space Agency). Το έργο διατίθεται υπό την άδεια ανοικτού κώδικα, με το λογισμικό και τις κατασκευές που έχει αναπτύξει η ομάδα να είναι διαθέσιμα σε ανοιχτά αποθετήρια για ελεύθερη χρήση από το ευρύ κοινό.

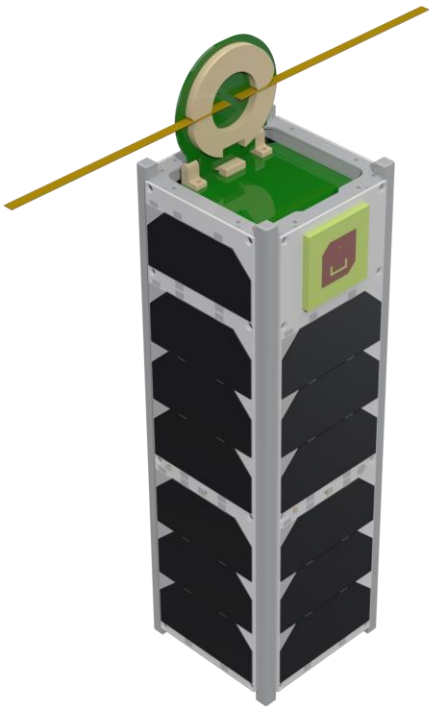
Εικόνα 2.1: Λογότυπο της αποστολής AcubeSAT [7]

2.2 Αποστολή

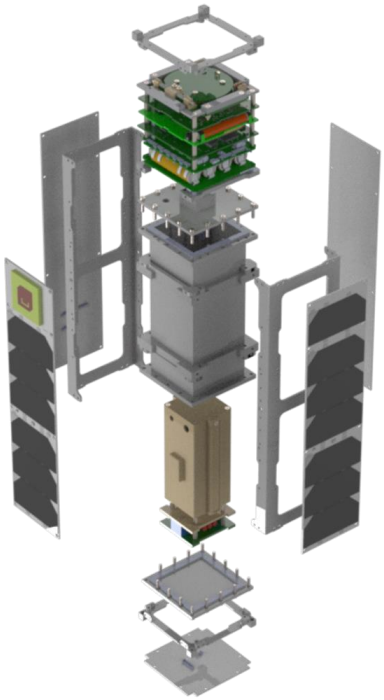
Η αποστολή έχει ως βασικό στόχο την μελέτη και ρύθμιση της μοριακής συμπεριφοράς των ευκαρυωτικών κυττάρων [7], υπό συνθήκες μικροβαρύτητας και ακτινοβολίας σε χαμηλή γήινη τροχιά (Low-Earth Orbit). Επιπρόσθετα, η αποστολή στοχεύει στην αξιοποίηση ενός μικρορευστικού τσιπ, παρουσιάζοντας αυτού του τύπου τις πλατφόρμες Lab-on-a-Chip (LOC) ως μια εφικτή και αρθρωτή εναλλακτική λύση για τη διεξαγωγή χαμηλού κόστους, υψηλής απόδοσης έρευνας διαστημικής βιολογίας. Πιο συγκεκριμένα η έμφυτη ικανότητα των συσκευών LOC , όπως η μικρογραφία, η παραλληλοποίηση και η αυτοματοποίηση, επιτρέπουν υψηλό χωροχρονικό έλεγχο, ενώ η ευελιξία τους επιτρέπει την ακριβή ρύθμιση των του μικροπεριβάλλοντος σε κυτταρικό επίπεδο.

Ο νανοδορυφόρος θα διαθέτει μια συμπιεσμένη εσωτερική δεξαμενή με μικροσκοπική διάταξη και ένα lab-on-a-chip ικανό να συντηρεί καλλιέργειες μυκήτων Saccharomyces cerevisiae για τη διερεύνηση των επιπτώσεων της ακτινοβολίας και των συνθηκών μικροβαρύτητας σε χαμηλή γήινη τροχιά. Για την επίτευξη αυτού του πειραματικού σκοπού, τα κύτταρα μυκήτων τροποποιούνται γενετικά ώστε να παράγουν φθορισμό κατά την έκφραση ενός γονιδίου ενδιαφέροντος. Οι επακόλουθες μελέτες θα εξετάσουν τα γονίδια σε κλίμακα έως και 100-200 φορές μεγαλύτερη από τις προηγούμενες αποστολές, καθώς και θα παρέχουν οπτικές παρατηρήσεις υψηλής ποιότητας για μια εκτεταμένη περίοδο 6 έως 9 μηνών ενώ βρίσκονται σε τροχιά.

Ζητούμενο της αποστολής είναι να χρησιμοποιηθούν τα ευρήματα αυτής της έρευνάς για τη διεξαγωγή συσχετίσεων, σε μοριακό επίπεδο, σχετικά με τις επιπτώσεις του διαστήματος στον άνθρωπο. Παράλληλα, τα κύτταρα που χρησιμοποιούνται αποτελούν ένα από τα σημαντικότερα βιοτεχνολογικά εργαλεία, με εφαρμογές που περιλαμβάνουν βιοκαύσιμα, φάρμακα και σύνθεση φαρμάκων.



Εικόνα 2.2: AcubeSAT με εγκατεστημένη την κεραία UHF [8]



Εικόνα 2.3: Αναλυτική προβολή του AcubeSAT [8]

2.3 Υποσυστήματα

Ο νανοδορυφόρος AcubeSAT αποτελείται από 11 υποσυστήματα, καθένα από τα οποία βρίσκεται υπό τη διαχείριση μίας ολιγομελούς ομάδας που είναι υπεύθυνη για το αντίστοιχο τμήμα. Λεπτομερής ανάλυση του σχεδιασμού και των λειτουργιών κάθε υποσυστήματος υπάρχει διαθέσιμη στα δημόσια αποθετήρια της ομάδας, στα έγγραφα του Critical Design Review (CDR) [10], ενώ οι λεπτομέρειες που αφορούν το FDIR αναλύονται στην διπλωματική εργασία μέλους της ομάδας που έχει πραγματοποιήσει εκτενής θεωρητική έρευνα πάνω στην αρχιτεκτονική του FDIR στα υποσυστήματα του AcubeSAT [11].

Τα υποσυστήματα του AcubeSAT είναι τα εξής:

Υποσύστημα Προσδιορισμού και Ελέγχου Προσανατολισμού (ADCS)

Η υποομάδα ADCS είναι υπεύθυνη για τον έλεγχο της προσανατολισμού του δορυφόρου καθώς βρίσκεται σε τροχιά. Το ADCS χρησιμοποιεί διάφορους αισθητήρες όπως γυροσκόπια και μαγνητόμετρα, καθώς και ενεργοποιητές όπως μαγνητικοί ενεργοποιητές 3 αξόνων και τροχός αντίδρασης 1 άξονα, για να προσδιορίσει και να ρυθμίσει την περιστροφή του δορυφόρου. Αυτό συνδυάζεται με αλγορίθμους ελέγχου και φιλτραρίσματος για αποτελεσματική λειτουργία.

Το σύστημα προσφέρει τρία διαφορετικά προφίλ στόχευσης:

- **Λειτουργία Σταθεροποίησης (Detumbling)**: Στοχεύει στην επιβράδυνση της γωνιακής ταχύτητας του δορυφόρου για να εξασφαλίσει σταθερή επικοινωνία και να αποφευχθούν τεχνικά προβλήματα.
- **Στόχευση στο Ναδίρ (Nadir Pointing)**: Ο δορυφόρος στρέφει την πλευρά +X προς τη Γη για άμεση οπτική επαφή κατά τη διάρκεια των περασμάτων πάνω από τον Σταθμό Βάσης.
- **Στόχευση στον Ήλιο (Sun Pointing)**: Προσανατολισμός προς τον ήλιο για μεγιστοποίηση της ενέργειας που λαμβάνουν τα ηλιακά πάνελ, χρήσιμο μεταξύ των περασμάτων από το Σταθμό Βάσης.

Υποσύστημα Τηλεπικοινωνιών (COMMS)

Το υποσύστημα COMMS του δορυφόρου είναι υπεύθυνο για τη μετάδοση δεδομένων μεταξύ της Γης και του δορυφόρου. Χρησιμοποιεί την πλακέτα SatNOGS COMMS για επικοινωνία, η οποία είναι ένας RF πομποδέκτης ανοικτού κώδικα που λειτουργεί στις ζώνες συχνοτήτων ISM, 436.5 MHz και 2.425 GHz. Το σύστημα μεταφέρει τηλε-εντολές, τηλεμετρία και πειραματικά επιστημονικά δεδομένα. Επιπλέον, το COMMS ασχολείται με την ανάλυση ηλεκτρομαγνητικής συμβατότητας και την κατασκευή του Σταθμού Βάσης, που είναι μέρος του παγκόσμιου δικτύου SatNOGS.

Υποσύστημα Ενέργειας (EPS)

Το υποσύστημα ηλεκτρικής ενέργειας (EPS) του AcubeSAT είναι ζωτικό για την παραγωγή, κατανομή και αποθήκευση ενέργειας στον δορυφόρο, καθώς αυτό εξασφαλίζει την ηλεκτρική τροφοδοσία πολλών άλλων υποσυστημάτων. Η κατασκευή του επικεντρώνεται στη χρήση προϊόντων COTS για αυξημένη αξιοπιστία και περιλαμβάνει ηλιακά πάνελ της EnduroSat για τη συλλογή ενέργειας, μια μονάδα ελέγχου και διανομής ισχύος (PCDU) από την NanoAvionics που προσφέρει πολλαπλά κανάλια και τάσεις, καθώς και μπαταρίες ιόντων λιθίου για αποθήκευση ενέργειας.

Υποσύστημα Διαχείρισης Δεδομένων (OBDH)

Το υποσύστημα OBDH του AcubeSAT είναι αφιερωμένο στον σχεδιασμό και τη διαχείριση των διεπαφών δεδομένων του δορυφόρου, καθώς και στην επίβλεψη των κρίσιμων λειτουργιών του μέσω της πλακέτας On-Board Computer (OBC). Η πλακέτα OBC βασίζεται σε έναν ανθεκτικό στην ακτινοβολία μικροελεγκτή Microchip SAMV71Q21RT και χρησιμοποιεί μνήμη MRAM για την αποθήκευση κρίσιμων δεδομένων. Επιπλέον, υποστηρίζει το υποσύστημα ADCS για εξοικονόμηση χώρου. Η επικοινωνία μεταξύ των υποσυστημάτων του δορυφόρου γίνεται μέσω ενός Controller Area Network (CAN) bus με ψυχρό πλεονασμό, επιλογή που αντικατοπτρίζει την υψηλή αξιοπιστία του συστήματος.

Υποσύστημα Λογισμικού (OBSW)

Το υποσύστημα OBSW του AcubeSAT είναι αρμόδιο για το σχεδιασμό και την ανάπτυξη του λογισμικού του νανοδορυφόρου. Το λογισμικό αναπτύσσεται σε μια περιορισμένη μορφή της γλώσσας C++, και ο κώδικας υπόκειται σε ελέγχους μέσω προτύπων, στατικών αναλυτών και unit tests, εκτελούμενος στο λειτουργικό σύστημα πραγματικού χρόνου `FreeRTOS`.

Υποσύστημα Χειρισμού (OPS)

Το υποσύστημα OPS του AcubeSAT είναι υπεύθυνο για τον σχεδιασμό και την επιβεβαίωση των λειτουργιών και διαδικασιών του δορυφόρου, εξασφαλίζοντας τη λειτουργικότητα, τον έλεγχο και την παρατηρησιμότητα του σκάφους τόσο πριν όσο και κατά τη διάρκεια της τροχιάς του. Κατά τη διάρκεια της αποστολής, το AcubeSAT υποστηρίζει πολλαπλές λειτουργικές καταστάσεις:

- **Λειτουργία Εκτόξευσης**: όπου ο δορυφόρος είναι πλήρως απενεργοποιημένος.
- **Λειτουργία Εκκίνησης**: για τις αρχικές διεργασίες μετά την εκτόξευση
- **Κανονική Λειτουργία**: για την καθημερινή λειτουργία και διαχείριση επιστημονικών δεδομένων

- **Επιστημονική Λειτουργία**: όπου πραγματοποιούνται τα επιστημονικά πειράματα.
- **Ασφαλής Λειτουργία**: για τη διαχείριση κρίσεων.

Υποσύστημα Δομής (Structural)

Το υποσύστημα δομής του CubeSat αποτελεί ένα κρίσιμο τμήμα της δορυφορικής κατασκευής, αναλαμβάνοντας την ανάλυση, διαμόρφωση, και συναρμολόγηση του σκελετού του δορυφόρου και του δοχείου πειράματος. Ο σκελετός 3U, κατασκευασμένος από αλουμίνιο, προσφέρει την απαραίτητη μηχανική υποστήριξη για να στεγάσει όλα τα εξαρτήματα του δορυφόρου, ενώ οι αναλύσεις δονήσεων διασφαλίζουν την ανθεκτικότητα του CubeSat κατά τις συνθήκες της εκτόξευσης.

Μηχανική Συστήματος (SYE)

Η υποομάδα του SYE έχει τον ρόλο της τεχνικής αρχής στην ανάπτυξη του δορυφόρου, επιβλέποντας τον συντονισμό των εργασιών και της επικοινωνίας μεταξύ των διαφορετικών υποσυστημάτων. Είναι υπεύθυνη για τη συμμόρφωση με τα διεθνή πρότυπα και τις τεχνικές προδιαγραφές, καθώς και για τον εντοπισμό και επίλυση τεχνικών ζητημάτων που ενδέχεται να προκύψουν κατά τον σχεδιασμό και την υλοποίηση του συστήματος. Επιπρόσθετα, η ομάδα SYE αναλαμβάνει ειδικά τεχνικά καθήκοντα που δεν καλύπτονται από άλλα υποσυστήματα, όπως η ανάλυση αξιοπιστίας (RAMS), η ανάλυση αποτυχιών και των αποτελεσμάτων τους (FMEA), η καλωδίωση, καθώς και τον σχεδιασμό και εκτέλεση του πλάνου κατασκευής και δοκιμών (MAIV).

Επιστημονική Μονάδα (SU)

Η υποομάδα SU του AcubeSAT είναι αφιερωμένη στο σχεδιασμό και υλοποίηση του επιστημονικού φορτίου της αποστολής, με στόχο τη μελέτη των επιδράσεων του περιβάλλοντος της Χαμηλής Γήινης Τροχιάς (LEO) σε ζυμομύκητες. Το φορτίο περιλαμβάνει ένα δοχείο πειραμάτων για την φιλοξενία των οργάνων και των δειγμάτων, ένα μικρορευστομηχανικό τσιπ για την ανάλυση μεγάλου αριθμού στελεχών του μύκητα Saccharomyces Cerevisiae, ένα υδραυλικό σύστημα για τη διαχείριση των υγρών, ένα οπτικό σύστημα που λειτουργεί ως μικροσκόπιο, καθώς και θερμαντικά σώματα και αισθητήρες για τον έλεγχο του θερμοκρασιακού περιβάλλοντος και άλλων περιβαλλοντικών παραμέτρων, εξασφαλίζοντας τις ιδανικές συνθήκες για την επιτυχία του επιστημονικού πειράματος.

Υποσύστημα Θερμικού Ελέγχου (Thermal)

Η υποομάδα Thermal του AcubeSAT είναι αρμόδια για τη θερμική ανάλυση του δορυφόρου, εστιάζοντας στην αντιμετώπιση των προκλήσεων που προκύπτουν από την εισερχόμενη ηλιακή ακτινοβολία και την απαγωγή θερμότητας των υποσυστημάτων. Σκοπός της είναι ο προσδιορισμός των ακραίων θερμοκρασιών, τόσο ψυχρών όσο και θερμών, στις οποίες μπορεί να εκτεθεί ο δορυφόρος. Από τα αποτελέσματα της θερμικής ανάλυσης, αναπτύσσονται και εφαρμόζονται μέθοδοι θερμικού ελέγχου, είτε παθητικοί είτε ενεργητικοί. Στο AcubeSAT, χρησιμοποιούνται ειδικά τρία ηλεκτρονικά ελεγχόμενα θερμαντικά σώματα για τη διατήρηση της θερμοκρασίας των μπαταριών, του πειραματικού τσιπ, και των βαλβίδων, διασφαλίζοντας τη σταθερότητα και απόδοση του συστήματος υπό διάφορες συνθήκες λειτουργίας.

Ανάλυση Τροχιάς (Trajectory)

Η υποομάδα Trajectory του AcubeSAT είναι υπεύθυνη για την ανάλυση της τροχιάς του διαστημικού σκάφους. Αυτή η ανάλυση περιλαμβάνει την αποτίμηση των επιπτώσεων της ακτινοβολίας στο διαστημικό περιβάλλον, τη συμμόρφωση με τους διεθνείς κανονισμούς σχετικά με τα διαστημικά απόβλητα και την εκτίμηση της προβλεπόμενης διάρκειας ζωής του δορυφόρου σε τροχιά. Το AcubeSAT θα ακολουθήσει μια τροχιά που καθορίζεται από την εκτόξευση και δεν θα διαθέτει τη δυνατότητα τροποποίησής της κατά τη διάρκεια της πτήσης λόγω της έλλειψης προωθητήρων. Η ομάδα πραγματοποιεί αναλύσεις ευαισθησίας για την κατανόηση των διαφόρων παραμέτρων που επηρεάζουν την τροχιά και για τον προσδιορισμό των επιτρεπόμενων τροχιών, χρησιμοποιώντας εργαλεία όπως το GMAT της NASA.

2.4 Περιβάλλον υλοποίησης

Για την ανάπτυξη λογισμικού, η ομάδα χρησιμοποιεί τη γλώσσα προγραμματισμού C++17, η οποία επιλέχθηκε λόγω της αντικειμενοστραφούς φύσης της και της δυνατότητάς της για χαμηλού επιπέδου χειρισμό μνήμης. Η συγγραφή του κώδικα πραγματοποιείται στο ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) CLion, το οποίο παρέχει προηγμένα εργαλεία για επεξεργασία, διόρθωση και διαχείριση του κώδικα, ενώ παράλληλα παρέχει ενσωματωμένες λειτουργίες της πλατφόρμας GitLab. Η τελευταία, χρησιμοποιείται για τη διαχείριση των εκδόσεων του κώδικα (version control) και την πραγματοποίηση τακτικών ελέγχων για την διασφάλιση της ποιότητας αυτού. Αξίζει να σημειωθεί πως κάθε κομμάτι κώδικα υπόκειται σε αυστηρό peer-review, διασφαλίζοντας ότι το τελικό αποτέλεσμα πληροί τις υψηλότερες προδιαγραφές ποιότητας και λειτουργικότητας.

2.5 Χρησιμοποιούμενα εργαλεία

Στο πλαίσιο του πολύπλοκου πρότζεκτ AcubeSAT, χρησιμοποιούνται πολλαπλά βοηθητικά εργαλεία για την υποστήριξη των τεχνικών και οργανωτικών απαιτήσεων του έργου. Τα εργαλεία αυτά βοηθούν στη διαχείριση κώδικα, στην τεκμηρίωση και στην οργάνωση της ομάδας.

Κάποια από τα βασικότερα είναι:

- **GitLab**: Χρησιμοποιείται ως πλατφόρμα για τον διαμοιρασμό και τη διαχείριση του κώδικα του πρότζεκτ.
- **LaTeX και Overleaf**: Εργαλεία που χρησιμοποιούνται για τη συγγραφή τεχνικής βιβλιογραφίας και τεκμηρίωσης.
- **Mattermost και Discord**: Εργαλεία για την οργάνωση και την επικοινωνία μεταξύ των μελών της ομάδας.

3 FDIR στο AcubeSAT

3.1 Πρότυπο αξιοποίησης Πακέτων ECSS

Το ECSS-E-ST-70-41C, επίσης γνωστό ως Packet Utilisation Standard (PUS) [12], είναι ένας βασικός κανονισμός του οργανισμού European Cooperation for Space Standardization (ECSS). Καθιερώνει πρότυπα για τις τηλε-εντολές (TC) και την τηλεμετρία (TM), με δυνατότητα εφαρμογής σε οποιαδήποτε διαστημική αποστολή. Το πρότυπο έχει ως στόχο να διασφαλίσει ότι τα διαστημικά σκάφη μπορούν να αλληλοεπιδρούν αποτελεσματικά με τους σταθμούς βάσης στη Γη, παρέχοντας μια τυποποιημένη «γλώσσα» για τη μεταφορά πληροφοριών και οδηγιών. Επιπλέον, ορίζει τις βασικές λειτουργίες του συστήματος, συμπεριλαμβανομένων των συνθηκών υπό τις οποίες εκτελούνται οι εντολές και διαχειρίζονται τα συμβάντα. Περιλαμβάνει επίσης ένα ευρύ φάσμα εφαρμογών και χαρακτηριστικών, επιτρέποντας στους σχεδιαστές αποστολών να επιλέγουν και να προσαρμόζουν τις καταλληλότερες υπηρεσίες για την εκάστοτε αποστολή τους, με σκοπό τον αποτελεσματικό έλεγχο και την παρακολούθηση των διαστημικών σκαφών.

Επιγραμματικά οι υπηρεσίες (services) που υλοποιούνται από την ομάδα του AcubeSAT, σύμφωνα με το ECSS-E-ST-70-41C είναι οι εξής:

- **ST[01] Request Verification:** Υπηρεσία υπεύθυνη για αναφορές επιβεβαίωσης ή αποτυχίας εκτελούμενων εντολών στους χειριστές.
- **ST[03] Housekeeping:** Παράγει περιοδικές αναφορές για τιμές παραμέτρων, συνθέτοντας του περιοδικούς ραδιοφάρους (RF beacons) του δορυφόρου.
- **ST[04] Parameter Statistics Reporting**: Υπηρεσία υπεύθυνη για την αναφορά στατιστικών στοιχείων ανά προκαθορισμένα χρονικά διαστήματα.
- **ST[05] Event Reporting:** Υπηρεσία υπεύθυνη για την ενημέρωση των χειριστών όταν πραγματοποιείται ένα γεγονός.
- **ST[06] Memory Management:** Επιτρέπει την απευθείας ανάγνωση και εγγραφή σε μονάδες μνήμης, και παράλληλα δίνει τη δυνατότητα διακίνησης αρχείων σε ένα σύστημα αρχείων.
- **ST[08] Function Management:** Παρέχει την δυνατότητα εκτέλεσης προκαθορισμένων και προσαρμόσιμών μεθόδων κατά την εκτέλεση πειραματικών ή και γενικών λειτουργιών.
- **ST[11] Time-based Scheduling:** Επιτρέπει τον χρονικό προγραμματισμό εκτέλεσης τηλε-εντολών.
- **ST[12] On-board Monitoring:** Ελέγχει τιμές παραμέτρων για να διασφαλίσει ότι παραμένουν εντός ρυθμιζόμενων ορίων. Σε περίπτωση παραβίασης των ορίων επικοινωνεί με την υπηρεσία ST[05] για την δημιουργία και ανάλυση αναφοράς.
- **ST[13] Large Packer Transfer:** Υπηρεσία υπεύθυνη για την τμηματοποίηση μεγάλων μηνυμάτων, ώστε να χωρέσουν στο μέγιστο αποδεκτό μήκος TC ή TM.

- **ST[15] On-board Storage and Retrieval:** Αποθηκεύει τις παραγόμενες αναφορές και επιτρέπει την ανάκτηση τους όταν ο δορυφόρος διέρχεται πάνω από τον σταθμό βάσης.
- **ST[17] Test:** Υπηρεσία υπεύθυνη για εντολές ελέγχου της υγείας του σκάφους.
- **ST[19] Event-action:** Παρέχει την δυνατότητα αυτόματης εκτέλεσης TCs κατά την πραγματοποίηση ενός γεγονότος ST[05].
- **ST[20] Parameter Management:** Υπηρεσία υπεύθυνη για τη διαχείριση των παραμέτρων του δορυφόρου και των τιμών τους.
- **ST[23] File Management:** Επιτρέπει την εκτέλεση διαχειριστικών εντολών σε συστήματα αρχείων στον δορυφόρο, όπως αντιγραφή, μετακίνηση, διαγραφή ή κλείδωμα.

Οι τηλε-εντολές κάθε υπηρεσίας, που έχουν επιλεχθεί για χρήση στην αποστολή και η πρόοδος της διαδικασίας υλοποίησης τους, βρίσκεται σε ξεχωριστό αποθετήριο της ομάδας στο GitLab, για το οποίο θα γίνει πιο αναλυτική αναφορά παρακάτω.

3.2 Βασικές Αρχές του FDIR

Το σύστημα FDIR στον AcubeSAT χρησιμοποιεί το πρότυπο χρήσης πακέτων ECSS-E-ST-70-41C και τις σχετικές υπηρεσίες, όπως συνιστάται από το SAVOIR-HB-003 [13]. Αυτή η προσέγγιση επιτρέπει μια αρχιτεκτονική συστήματος που μπορεί να καθοριστεί πριν από την εκτόξευση και να προσαρμοστεί κατά τη διάρκεια της πτήσης, ακολουθώντας κάποιες βασικές αρχές [14].

Συγκεκριμένα, διάφορες υπηρεσίες PUS διασυνδέονται με **αρθρωτό** τρόπο, δηλαδή υπάρχει μια δομημένη βάση δεδομένων για την υλοποίηση του λογισμικού, το οποίο δεν βασίζεται σε hard-coded τιμές και είναι ευέλικτο στις αλλαγές του σχεδιασμού. Επιπρόσθετα, η **αυτονομία** είναι μια κρίσιμη αρχή του FDIR, καθώς επιτρέπει την λειτουργία του σκάφους για μεγαλύτερο χρονικό διάστημα χωρίς χειροκίνητη παρέμβαση, ενώ ταυτόχρονα αποφεύγεται η πιθανότητα διάδοσης βλαβών σε όλο το σύστημα. Λόγω της απρόβλεπτης φύσης του διαστήματος, η **διαμορφωσιμότητα** είναι καίρια καθώς ο μηχανισμός του FDIR πρέπει να μπορεί να δέχεται επιπλέον ρύθμιση αν είναι αναγκαίο, συμπεριλαμβανομένων ενεργειών όπως απενεργοποίηση ελέγχων, μεταβολή ορίων και εγκατάσταση νέων ελέγχων. Η **παρατηρησιμότητα** ενισχύει τη δυνατότητα διερεύνησης συμβάντων με πρόσβαση σε εκτεταμένα δεδομένα αισθητήρων και καταστάσεις του συστήματος. Το λογισμικό του FDIR είναι σχεδιασμένο βασιζόμενο στις αποτυχίες, θεωρώντας κάθε σφάλμα πιθανό, έχοντας επαληθευμένες μεθόδους ανάκτησης λειτουργίας για όλα τα υποσυστήματα του σκάφους.

Το AcubeSAT επωφελείται από μεγαλύτερους χρόνους σε τροχιά, αφού αυτό επιτρέπει το κατέβασμα μεγαλύτερου όγκου δεδομένων και περισσότερα επιστημονικά αποτελέσματα. Επιπλέον, οποιαδήποτε σύντομη διακοπή των υποπειραμάτων θα μπορούσε ενδεχομένως να οδηγήσει σε πλήρη αποτυχία τους. Ως εκ τούτου, το AcubeSAT ακολουθεί μία λειτουργική (fail-operational) και όχι ακίνδυνη (fail-safe) λογική, και προσπαθεί να διορθώσει βλάβες χωρίς να διακόψει τη ροή των λειτουργιών του.

3.3 Αρχιτεκτονική του FDIR

Όπως ήδη αναφέρθηκε στο Κεφάλαιο 3.2 η αρχιτεκτονική του FDIR στον AcubeSAT είναι βασισμένη στο πρότυπο ECSS-E-ST-70-41C. Η ροή της αρχιτεκτονικής [14] του συστήματος FDIR στον AcubeSAT ξεκινά με τη συνεχή παρακολούθηση των παραμέτρων επί του σκάφους, όπως η κατάσταση των εξαρτημάτων, οι χρονομετρητές και οι μετρητές βλαβών, που διαχειρίζεται η υπηρεσία διαχείρισης παραμέτρων ST[20]. Αυτές οι παράμετροι παρακολουθούνται σχολαστικά από την υπηρεσία ST[12] για τυχόν αποκλίσεις από προκαθορισμένα ή δυναμικά προσαρμοσμένα όρια. Κατά την ανίχνευση τυχόν ανωμαλιών, ενεργοποιείται η υπηρεσία αναφοράς συμβάντων ST[05], η οποία καταγράφει με ακρίβεια και αναφέρει τη βλάβη ως τηλεμετρία. Στη συνέχεια, αναλαμβάνει η υπηρεσία ST[19], η οποία εκτελεί συγκεκριμένες εντολές του διαστημικού σκάφους προσαρμοσμένες για τον μετριασμό και τη διόρθωση των ανιχνευμένων βλαβών. Αυτή η διαδικασία όχι μόνο εξασφαλίζει την ακεραιότητα του συστήματος αλλά και διατηρεί συνεχή λειτουργική κατάσταση, η οποία είναι προσβάσιμη τόσο στους επίγειους σταθμούς όσο και στους δέκτες του υποσυστήματος.

Εικόνα 3.1: Αρχιτεκτονική βασισμένη σε PUS για επίγειο διαμορφώσιμο αρθρωτό μηχανισμό FDIR, εμπνευσμένη από το SAVOIR-HB-003 [14]

Η εφαρμογή του FDIR στο AcubeSAT έχει σχεδιαστεί στρατηγικά σε τρία επίπεδα για να διασφαλίσει ολοκληρωμένη διαχείριση σφαλμάτων και ανθεκτικότητα του συστήματος. Σε επίπεδο **μονάδας**, οι μεμονωμένες αστοχίες εξαρτημάτων εντοπίζονται και διορθώνονται γρήγορα από το MCU κάθε υποσυστήματος, αποτρέποντας αποτελεσματικά την κλιμάκωση των προβλημάτων. Το επίπεδο **υποσυστήματος** επιτρέπει ευρύτερες διορθωτικές ενέργειες, όπως προσωρινές ή μόνιμες προσαρμογές, όπως επανεκκινήσεις ή επαναδιαμορφώσεις, για τη διατήρηση της ακεραιότητας των μεγαλύτερων στοιχείων του συστήματος. Το επίπεδο **συστήματος** εστιάζει στις γενικές λειτουργίες ασφάλειας και ανάκτησης, παρέχοντας ουσιαστική προστασία από εκτεταμένες βλάβες του συστήματος, αν και με κάποιο κίνδυνο. Αυτή η επεκτάσιμη προσέγγιση επιτρέπει ακριβείς και κλιμακούμενες αποκρίσεις που είναι ζωτικής σημασίας για τη μακροζωία και την απόδοση του διαστημικού σκάφους.

Ο καταμερισμός των αρμοδιοτήτων εντός του συστήματος FDIR είναι στενά καθορισμένος ώστε να ενισχύεται η γρήγορη και αποτελεσματική διαχείριση των βλαβών. Σε περίπτωση απομονωμένης βλάβης μονάδας, κάθε υποσύστημα είναι εξοπλισμένο για ανεξάρτητη ανίχνευση

και αντιμετώπιση βλαβών, ελαχιστοποιώντας τον χρόνο διακοπής λειτουργίας και αποτρέποντας τη διάδοση της βλάβης. Σε περίπτωση προβλήματος σε ολόκληρο το υποσύστημα ή σε μια βασική λειτουργία, το λογισμικό του ενσωματωμένου υπολογιστή (OBC) διαδραματίζει πρωταγωνιστικό ρόλο, αναλαμβάνοντας τα ηνία για την υλοποίηση των απαραίτητων ενεργειών αποκατάστασης.

Επιπλέον, για την προστασία από πιθανά μεμονωμένα σημεία αστοχίας, όπως μια δυσλειτουργία του OBC, τα σχέδια έκτακτης ανάγκης περιλαμβάνουν την ανάπτυξη ενός εναλλακτικού υποσυστήματος για την ανάληψη των αρμοδιοτήτων του FDIR. Αυτός ο ισχυρός μηχανισμός εξασφαλίζει ότι οι δραστηριότητες FDIR δεν είναι μόνο αξιόπιστες αλλά και προσαρμόσιμες, σύμφωνα με τα αυστηρά πρότυπα ασφαλείας που καθορίζονται στις επιχειρησιακές απαιτήσεις του διαστημικού σκάφους.

4 Η υπηρεσία ST[12]

4.1 Εισαγωγή στην Υπηρεσία ST[12]

Η υπηρεσία ST[12] παρέχει τη δυνατότητα παρακολούθησης των on-board παραμέτρων σε σχέση με τους προκαθορισμένους ελέγχους από το επίγειο σύστημα, με σκοπό την αναφορά τυχόν μεταβάσεων ελέγχου παραμέτρων στη Γη και την δημιουργία events όταν παραβιάζονται οι συνθήκες παρακολούθησης. Οι τύποι ελέγχου που μπορούν να εφαρμοστούν για μια παράμετρο επί του σκάφους εξαρτώνται από την παράμετρο και τον τύπο της. Τα τρία είδη ελέγχων είναι ο **έλεγχος ορίου**, ο **έλεγχος αναμενόμενης τιμής** και ο **έλεγχος δέλτα**, ενώ για κάθε παράμετρο και τον τύπο ελέγχου της καθορίζεται ένας ορισμός παρακολούθησης παραμέτρων (PMON ή parameter monitoring definition). Το πρότυπο ECSS-E-ST-70-41C δεν περιορίζει τον αριθμό των ελέγχων που μπορεί να εκτελεστεί για μια on-board παράμετρο, καθώς μπορεί για παράδειγμα μια παράμετρος την ίδια στιγμή να ελέγχεται ταυτόχρονα με έλεγχο ορίων και έλεγχο δέλτα, χρησιμοποιώντας δύο διαφορετικούς ορισμούς παρακολούθησης παραμέτρων. Επιπρόσθετα, η υπηρεσία παρέχει την δυνατότητα να περιλαμβάνει έναν έλεγχο υπό όρους σε ένα `PMON`, για περιπτώσεις απενεργοποίησης της παρακολούθησης μιας παραμέτρου όταν ο σχετικός εξοπλισμός είναι ανενεργός. Σε αυτή την περίπτωση, εάν ο έλεγχος υπό συνθήκη είναι ψευδής, ο έλεγχος παρακολούθησης παραμέτρων σε αυτόν τον ορισμό δεν εκτελείται.

Η υπηρεσία ST[12] συνδυάζεται με μια υπηρεσία αναφοράς συμβάντων (ST[05]), ζωτικής σημασίας για την καταγραφή των συμβάντων που ενεργοποιούνται στην υπηρεσία παρακολούθησης και την έκδοση των απαραίτητων ειδοποιήσεων. Τα συμβάντα αναγνωρίζονται μοναδικά μέσω ενός συνδυασμού του αναγνωριστικού της διεργασίας εφαρμογής που φιλοξενεί την υπηρεσία αναφοράς συμβάντων και ενός αναγνωριστικού ορισμού συμβάντος. Αυτή η ρύθμιση διασφαλίζει ότι κάθε υπηρεσία παρακολούθησης είναι πλήρως εξοπλισμένη για τον αποτελεσματικό χειρισμό τόσο της παρακολούθησης των παραμέτρων όσο και της διαχείρισης των συμβάντων του συστήματος.

4.1.1 Τύποι ελέγχων

Όπως αναφέρθηκε προηγουμένως, η υπηρεσία έχει σχεδιαστεί για να υποστηρίζει βασικούς τύπους ελέγχου για τη διασφάλιση της ακεραιότητας του συστήματος. Πρέπει να είναι σε θέση να εκτελεί δύο θεμελιώδεις τύπους ελέγχων, έλεγχο ορίων και έλεγχο αναμενόμενης τιμής. Σε έναν **έλεγχο ορίου**, η υπηρεσία επαληθεύει ότι η τιμή μιας παραμέτρου εμπίπτει εντός καθορισμένων οριακών τιμών, δηλώνοντας ότι ο έλεγχος είναι επιτυχής εάν η τιμή της παραμέτρου δεν υπερβαίνει το υψηλό όριο και δεν υπολείπεται του χαμηλού ορίου. Για τους **ελέγχους αναμενόμενης τιμής**, η υπηρεσία αξιολογεί εάν η τιμή που προκύπτει από την εφαρμογή μιας μάσκας bit ταιριάζει με μια αναμενόμενη τιμή, και η επιτυχία δηλώνεται όταν οι τιμές αυτές συμπίπτουν. Επιπλέον, η υπηρεσία

μπορεί να υποστηρίζει **ελέγχους δέλτα**, οι οποίοι περιλαμβάνουν τον υπολογισμό της διαφοράς μεταξύ δύο διαδοχικών τιμών παραμέτρων και τον προσδιορισμό του κατά πόσον αυτή η διαφορά βρίσκεται εντός ενός καθορισμένου οριακού εύρους. Αξιοσημείωτο είναι πως η ικανότητα εκτέλεσης αυτού του ελέγχου πρέπει να καθορίζεται κατά τον ορισμό της υπηρεσίας. Μία πρωτότυπη υλοποίηση της λογικής κάθε τύπου ελέγχου σε κώδικα μαζί με δοκιμές περιγράφεται στο Κεφάλαιο 6.

4.1.2 Ορισμός παρακολούθησης παραμέτρων

Ένας ορισμός παρακολούθησης παραμέτρων (PMON ή parameter monitoring definition) καθορίζει τα κριτήρια και τις συνθήκες για την παρακολούθηση και την αξιολόγηση μιας συγκεκριμένης παραμέτρου του συστήματος. Ο μέγιστος αριθμός PMON που μπορεί να αξιολογεί ταυτόχρονα η υπηρεσία δηλώνεται κατά τον προσδιορισμό της εν λόγω υπηρεσίας. Αυτός ο αριθμός αποτελεί και τον μέγιστο αριθμό εγγραφών που μπορεί να αποθηκεύσει η υπηρεσία σε μία λίστα που ονομάζεται "PMON list". Κάθε `PMON` πρέπει να περιέχει τις εξής πληροφορίες:

- Το αναγνωριστικό του `PMON`.
- Το αναγνωριστικό της ενσωματωμένης παραμέτρου που θα παρακολουθείται.
- Μια boolean συνθήκη για να ελεγχθεί αν η παρακολούθηση είναι ενεργοποιημένη.
- Κατάσταση Ελέγχου.
- Χρονικά διαστήματα παρακολούθησης.
- Έναν τύπο ελέγχου.

Σχετικά με τον τύπο ελέγχου, αυτός μπορεί να είναι ένας από τους προαναφερόμενους τρεις:

- Έλεγχος Ορίου (Limit Check).
- Έλεγχος Αναμενόμενης Τιμής (Expected Value Check).
- Έλεγχος Δέλτα (Delta Check).

Όταν ο τύπος ελέγχου είναι Ορίου ο ορισμός περιέχει επιπλέον:

- Υψηλό όριο (High Limit).
- Χαμηλό όριο (Low Limit).
- Το αναγνωριστικό ορισμού συμβάντος, εάν καθιερωθεί νέα κατάσταση ελέγχου "κάτω από το χαμηλό όριο" (Below Low Limit Event).
- Το αναγνωριστικό ορισμού συμβάντος, εάν καθιερωθεί νέα κατάσταση ελέγχου "άνω του υψηλού ορίου" (Above High Limit Event).

Όταν ο τύπος ελέγχου είναι Αναμενόμενης Τιμής ο ορισμός περιέχει επιπλέον:

- Αναμενόμενη τιμή (Expected Value).
- Τη μάσκα που εφαρμόζεται στην λαμβανόμενη τιμή (Mask).
- Το αναγνωριστικό ορισμού συμβάντος, εάν καθιερωθεί νέα κατάσταση ελέγχου "απροσδόκητης τιμής" (Unexpected Value Event).

Όταν ο τύπος ελέγχου είναι Δέλτα ο ορισμός περιέχει επιπλέον:

- Τον αριθμό των διαδοχικών τιμών δέλτα, κάθε μία από τις οποίες υπολογίζεται μεταξύ δύο διαδοχικών τιμών της παραμέτρου, που χρησιμοποιούνται για τον υπολογισμό της μέσης τιμής αυτών των διαδοχικών τιμών δέλτα (number Of Consecutive Delta Checks).
- Τιμή χαμηλού δέλτα κατωφλίου (Low Delta Threshold).
- Τιμή υψηλού δέλτα κατωφλίου (High Delta Threshold).
- Το αναγνωριστικό ορισμού συμβάντος, εάν καθιερωθεί νέα κατάσταση ελέγχου "κάτω από το χαμηλό κατώφλι (Below Low Threshold Event).
- Το αναγνωριστικό ορισμού συμβάντος, εάν καθιερωθεί νέα κατάσταση ελέγχου "άνω του υψηλού κατωφλίου" (Above High Threshold Event).

4.1.3 Καταστάσεις παρακολούθησης

Για κάθε ορισμό παρακολούθησης παραμέτρου, το σύστημα παρακολούθησης διατηρεί μια κατάσταση που δείχνει το αποτέλεσμα των ελέγχων που γίνονται στην παράμετρο αυτή. Αυτή η κατάσταση ονομάζεται "κατάσταση ελέγχου PMON" (PMON Checking Status).

- Για έναν έλεγχο ορίου, η κατάσταση ελέγχου `PMON` μπορεί να έχει οποιαδήποτε από τις ακόλουθες τιμές: "unchecked", "invalid", "within limits", "below low limit" ή "above high limit".
- Για έναν έλεγχο αναμενόμενης τιμής, ο έλεγχος `PMON` μπορεί να έχει οποιαδήποτε από τις ακόλουθες τιμές: "unchecked", "invalid", "expected value" ή "unexpected value".
- Για έναν έλεγχο δέλτα, η κατάσταση ελέγχου `PMON` μπορεί να έχει οποιαδήποτε από τις ακόλουθες τιμές: "unchecked", "invalid", "within threshold", "below low threshold" ή "above high threshold".

Η τιμή της κατάστασης ελέγχου `PMON` αλλάζει όταν ένας αριθμός διαδοχικών και συνεχών ελέγχων καθορίζουν μια νέα κατάσταση ελέγχου. Επίσης οι τιμές κατάστασης "unchecked" και "invalid" υποδεικνύουν ότι δεν υπάρχει καθορισμένη κατάσταση ελέγχου επί του παρόντος για την παράμετρο.

4.1.4 Λειτουργίες παρακολούθησης παραμέτρων

Οι βασικές λειτουργίες που εκτελεί η υπηρεσία με την μορφή τηλε-εντολών είναι επιγραμματικά:

- Ενεργοποίηση ορισμών παρακολούθησης παραμέτρων.
- Απενεργοποίηση ορισμών παρακολούθησης παραμέτρων.
- Αλλαγή της μέγιστης καθυστέρησης αναφοράς μετάβασης.

- Προσθήκη ορισμών παρακολούθησης παραμέτρων.
- Διαγραφή όλων των ορισμών παρακολούθησης παραμέτρων.
- Διαγραφή ορισμού παρακολούθησης παραμέτρου.
- Τροποποίηση ορισμών παρακολούθησης παραμέτρων.
- Αναφορά ορισμών παρακολούθησης παραμέτρων.

Αυτές είναι οι λειτουργίες έχουν κριθεί ως αναγκαίες από την ομάδα προς το παρόν και υλοποιήθηκαν ως συναρτήσεις σε κώδικα, η δομή και η λειτουργικότητα του οποίου αναλύεται στα επόμενα υποκεφάλαια.

4.2 Σχεδιασμός και Δομή του Κώδικα

4.2.1 Εισαγωγή

Η ομάδα έχει δημιουργήσει ένα ξεχωριστό αποθετήριο [15] για την υλοποίηση των υπηρεσιών ECSS στο περιβάλλον του GitLab, το οποίο είναι οργανωμένο σε καταλόγους (directories) που περιέχουν τα αρχεία κώδικα. Πιο αναλυτικά τα βασικα directories και τα περιεχόμενα τους είναι:

- **ci:** Αρχεία ρυθμίσεων Gitlab CI.
- **docs:** Έγγραφα ανάλυσης πηγαίου κώδικα και τεκμηρίωσης διαδικασίας ανάπτυξης.
- **inc:** Όλες οι επικεφαλίδες (headers) και οι βιβλιοθήκες (libraries) που χρησιμοποιούνται στον κώδικα.
- **lib:** Εξωτερικές βιβλιοθήκες που χρησιμοποιούνται στον κώδικα.
- **src:** Όλα τα πηγαία αρχεία περιλαμβάνονται εδώ.
- **test:** Υλοποίηση δοκιμών μονάδας (unit tests).

Η υλοποίηση της υπηρεσίας ST[12] περιλάμβανε την δημιουργία και τροποποίηση αρχείων σε όλα τα παραπάνω directories εκτός των docs και lib. Αρχικά, μέσα στο directory **inc**, και συγκεκριμένα στους φακέλους `Helpers` και `Services`, δημιουργήθηκαν τα αρχεία `PMON.hpp` και `OnBoardMonitoringService.hpp` αντίστοιχα, ενώ ενημερώθηκε το αρχείο `TypeDefinitios.hpp`. Γενικά τα αρχεία με κατάληξη `.hpp` (header files) περιέχουν τις δηλώσεις των κλάσεων, των συναρτήσεων, και των μεταβλητών. Αυτά τα αρχεία δεν περιλαμβάνουν την πραγματική υλοποίηση αλλά μόνο τις δηλώσεις, οι οποίες χρησιμεύουν ως διεπαφή προς τα άλλα μέρη του προγράμματος. Ουσιαστικά, τα header files ενημερώνουν τον μεταγλωττιστή (compiler) για το τι υπάρχει στον κώδικα (τύπους δεδομένων, κλάσεις, συναρτήσεις).

Ειδικότερα, το αρχείο `TypeDefinitios.hpp` ενεργεί ως ένας κεντρικός κατάλογος τύπων δεδομένων, για τη διαχείριση και τον ορισμό τύπων που χρησιμοποιούνται στις διάφορες υπηρεσίες και λειτουργίες του λογισμικού. Αυτή η διάταξη καθιστά εφικτή την ταχεία προσαρμογή και ενημέρωση των τύπων δεδομένων σε περίπτωση που αλλάξουν οι απαιτήσεις του συστήματος ή εντοπιστούν νέες βέλτιστες πρακτικές για τη διαχείριση δεδομένων, κάνοντας τον κώδικα αρθρωτό και επεκτάσιμο. Στο παρακάτω κομμάτι κώδικα φαίνεται ο ορισμός των τύπων δεδομένων για τις μεταβλητές που σχετίζονται με τους τύπους ελέγχων της υπηρεσίας ST[12], όπως προτείνονται από το πρότυπο ECSS.

```
using EventDefinitionId = uint16_t;
using ParameterId = uint16_t;

/**
 * The types used for the three Check Types and their variables in
OnBoardMonitoringService.
 */
 using PMONRepetitionNumber = uint16_t;
 using PMONLimit = double;
 using PMONExpectedValue = double;
 using PMONBitMask = uint64_t;
 using NumberOfConsecutiveDeltaChecks = uint16_t;
 using DeltaThreshold = double;
```

Κώδικας 4.1: Τύποι δεδομένων μεταβλητών των τύπων ελέγχων

Ο τύπος `uint16_t` είναι ένας 16-bit ακέραιος χωρίς πρόσημο, ιδανικός για την αποθήκευση μικρών αριθμητικών τιμών. Ο τύπος αυτός χρησιμοποιείται όταν είναι αναγκαία η οικονομία μνήμης και το εύρος των τιμών μιας παραμέτρου είναι περιορισμένο, κάτι σύνηθες στην περίπτωση υλοποίησης των υπηρεσιών ECSS. Από την άλλη ο τύπος `uint64_t` χρησιμοποιείται για μεγαλύτερα αριθμητικά στοιχεία ή δυαδικές μάσκες, προσφέροντας επαρκή χώρο για την αναπαράσταση μεγάλων τιμών και πολύπλοκων δεδομένων χωρίς τον κίνδυνο υπερχείλισης. Τέλος ο `double`, ένας τύπος πραγματικού αριθμού με διπλή ακρίβεια, είναι απαραίτητος όταν απαιτούνται ακριβείς υπολογισμοί και αποτελεσματική αναπαράσταση δεδομένων, και είναι ιδιαίτερα χρήσιμος για τις διεργασίες των μηχανισμών ελέγχου, όταν η ακρίβεια είναι κομβική. Με την δήλωση `using` ορίζεται ουσιαστικά ένα ψευδώνυμο τύπου δεδομένων για έναν υπάρχοντα τύπο, διευκολύνοντας την ανάγνωση, τη συντήρηση, καθώς και στην προσαρμογή του κώδικα σε μελλοντικές αλλαγές.

Το αρχείο `PMON.hpp` υλοποιεί τις αρχές του προτύπου ECSS για την παρακολούθηση παραμέτρων, μεταφράζοντας τις προαναφερόμενες προδιαγραφές σε απτές λειτουργίες μέσα σε ένα οργανωμένο σύνολο C++ κλάσεων.

4.2.2 Κλάση PMON

Η βασική κλάση `PMON` αποτελεί τον πυρήνα της διαχείρισης παρακολούθησης παραμέτρων, περιλαμβάνοντας τα θεμελιώδη στοιχεία για τον ορισμό και την παρακολούθηση της κατάστασης των παραμέτρων. Στους παρακάτω πίνακες παρουσιάζονται οι μεταβλητές και οι συναρτήσεις της κλάσης μαζί με την περιγραφή τους.

Πίνακας 4.1: Μεταβλητές κλάσης PMON.

Μεταβλητή	**Περιγραφή**
monitoredParameterId	Η ταυτότητα της παρακολουθούμενης παραμέτρου.
monitoredParameter	Αναφορά στην παρακολουθούμενη παράμετρο.
repetitionNumber	Ο αριθμός των επαναλήψεων που απαιτούνται για να οριστεί μια νέα κατάσταση παρακολούθησης.
repetitionCounter	Ο αριθμός των επαναλήψεων που έχουν ήδη πραγματοποιηθεί.
monitoringEnabled	Δείκτης που δηλώνει εάν η παρακολούθηση είναι ενεργοποιημένη ή απενεργοποιημένη.
checkingStatus	Η τρέχουσα κατάσταση ελέγχου της παρακολουθούμενης παραμέτρου.
checkTransitionList	Λίστα με τις πιθανές μεταβάσεις κατάστασης του ελέγχου.

Πίνακας 4.2: Συναρτήσεις κλάσης PMON.

Συνάρτηση	**Περιγραφή**
getRepetitionNumber()	Επιστρέφει τον αριθμό των επαναλήψεων που απαιτούνται για να οριστεί μια νέα κατάσταση.
isMonitoringEnabled()	Επιστρέφει True αν η παρακολούθηση είναι ενεργοποιημένη, διαφορετικά False.
getCheckType()	Επιστρέφει τον τρέχοντα τύπο ελέγχου.
getCheckingStatus()	Επιστρέφει την τρέχουσα κατάσταση ελέγχου της παρακολουθούμενης παραμέτρου.

Constructor της PMON

Ο `constructor` της κλάσης `PMON` παίζει σημαντικό ρόλο στην αρχικοποίηση των αντικειμένων της κλάσης. Ορίζεται ως προστατευμένος (protected), πράγμα που σημαίνει ότι η κλάση `PMON` προορίζεται να είναι βάση για άλλες κλάσεις (όπως οι υποκλάσεις που υλοποιούν συγκεκριμένους τύπους ελέγχων) και δεν μπορεί να δημιουργηθεί άμεσα.

Ο `constructor` δέχεται τρεις παραμέτρους: `monitoredParameterId`, `repetitionNumber`, και `checkType`. Αυτές οι παράμετροι αρχικοποιούν τις αντίστοιχες μεταβλητές της κλάσης, θέτοντας τις βασικές ιδιότητες του αντικειμένου κατά τη δημιουργία του. Αργότερα θα δούμε πως οι τιμές αυτών των παραμέτρων αρχικοποιούνται όταν δημιουργείται ένα νέο αντικείμενο της κλάσης `PMON` βάσει των περιεχομένων ενός αιτήματος-μηνύματος.

```
protected:
    /**
     * @param monitoredParameterId is assumed to be correct and not
checked.
     */
    PMON(ParameterId monitoredParameterId, PMONRepetitionNumber
repetitionNumber, CheckType checkType);
};
```

Κώδικας 4.2 Constructor κλάσης PMON.

Παρόλο που η κλάση `PMON` παρέχει τις θεμελιώδεις λειτουργίες, στην πράξη, η παρακολούθηση των παραμέτρων απαιτεί πιο εξειδικευμένες προσεγγίσεις ανάλογα με τη φύση και τις απαιτήσεις των εκάστοτε παραμέτρων. Αυτή η ανάγκη οδηγεί στην υλοποίηση ειδικών υποκλάσεων εντός της `PMON`, οι οποίες διαχειρίζονται τους τύπους ελέγχων που χρησιμοποιεί η υπηρεσία ST[12]. Κάθε υποκλάση αναλαμβάνει να εφαρμόσει τις απαραίτητες λειτουργίες και διαδικασίες ειδικά σχεδιασμένες για τον έλεγχο που αντιπροσωπεύει, επιτρέποντας μια πιο ολοκληρωμένη και εξατομικευμένη διαχείριση των παρακολουθούμενων παραμέτρων.

4.2.2.1 Υποκλάση PMONLimitCheck

Η υποκλάση `PMONLimitCheck` επεκτείνει την κλάση `PMON`, κληρονομώντας τα χαρακτηριστικά και τις μεθόδους της, και παράλληλα διαχειρίζεται ελέγχους ορίων για την παρακολούθηση αν οι παράμετροι παραμένουν εντός εγκεκριμένων οριακών τιμών. Στους παρακάτω πίνακες παρουσιάζονται οι μεταβλητές και οι συναρτήσεις της υποκλάσης μαζί με την περιγραφή τους.

Πίνακας 4.3: Μεταβλητές υποκλάσης PMONLimitCheck.

Μεταβλητή	**Περιγραφή**
lowLimit	Το κατώτατο όριο για την παρακολούθηση.
highLimit	Το ανώτατο όριο για την παρακολούθηση.
belowLowLimitEvent	Ταυτότητα του γεγονότος που εκδηλώνεται όταν η τιμή είναι κάτω από το κατώτατο όριο.
aboveHighLimitEvent	Ταυτότητα του γεγονότος που εκδηλώνεται όταν η τιμή υπερβαίνει το ανώτατο όριο.

Πίνακας 4.4: Συναρτήσεις υποκλάσης PMONLimitCheck.

Συνάρτηση	**Περιγραφή**
getLowLimit()	Επιστρέφει το κατώτατο όριο.
getHighLimit()	Επιστρέφει το ανώτατο όριο.
getBelowLowLimitEvent()	Επιστρέφει την ταυτότητα του γεγονότος για το κατώτατο όριο.
getAboveHighLimitEvent()	Επιστρέφει την ταυτότητα του γεγονότος για το ανώτατο όριο.

Constructor της PMONLimitCheck

Ο `constructor` της υποκλάσης `PMONLimitCheck` καλεί τον `constructor` της βασικής κλάσης `PMON`, περνώντας τις παραμέτρους `monitoredParameterId`, `repetitionNumber`, και `CheckType::Limit`, καθορίζοντας τον τύπο του ελέγχου ως έλεγχο ορίων. Αυτό διασφαλίζει ότι όλες οι βασικές και εξειδικευμένες μεταβλητές αρχικοποιούνται σωστά, προσδίδοντας σταθερότητα και ακρίβεια στη δημιουργία των αντικειμένων `PMONLimitCheck`.

```
explicit PMONLimitCheck(ParameterId monitoredParameterId, PMONRepetitionNumber
repetitionNumber, PMONLimit lowLimit, EventDefinitionId belowLowLimitEvent,
PMONLimit highLimit, EventDefinitionId aboveHighLimitEvent)
    : PMON(monitoredParameterId, repetitionNumber, CheckType::Limit),
lowLimit(lowLimit), belowLowLimitEvent(belowLowLimitEvent),
highLimit(highLimit),
      aboveHighLimitEvent(aboveHighLimitEvent) {
}
```

Κώδικας 4.3: Constructor υποκλάσης PMONLimitCheck.

4.2.2.2 Υποκλάση PMONExpectedValueCheck

Η υποκλάση `PMONExpectedValueCheck` επεκτείνει την κλάση PMON, κληρονομώντας τα χαρακτηριστικά και τις μεθόδους της, και παράλληλα ειδικεύεται στους ελέγχους αναμενόμενων τιμών. Στους παρακάτω πίνακες παρουσιάζονται οι μεταβλητές και οι συναρτήσεις της υποκλάσης μαζί με την περιγραφή τους.

Πίνακας 4.5: Μεταβλητές υποκλάσης PMONExpectedValueCheck.

Μεταβλητή	**Περιγραφή**
expectedValue	Η αναμενόμενη τιμή για την παρακολούθηση.
mask	Μάσκα που εφαρμόζεται στην παρακολούμενη τιμή.
unexpectedValueEvent	Ταυτότητα του γεγονότος που εκδηλώνεται όταν η τιμή είναι αναμενόμενη.

Πίνακας 4.6: Συναρτήσεις υποκλάσης PMONExpectedValueCheck.

Συνάρτηση	**Περιγραφή**
getMask()	Επιστρέφει τη μάσκα που χρησιμοποιείται.
getExpectedValue()	Επιστρέφει την αναμενόμενη τιμή.
getUnexpectedValueEvent()	Επιστρέφει την ταυτότητα του γεγονότος για αναμενόμενη τιμή.

Constructor της PMONExpectedValueCheck

Με την ίδια λογική όπως στην προηγουμένη υποκλάση, εδώ ο `constructor` καλεί επίσης αυτόν της βασικής κλάσης `PMON`, περνώντας τις παραμέτρους `monitoredParameterId`, `repetitionNumber`, και `CheckType::ExpectedValue`, η οποία καθορίζει τον τύπο του ελέγχου ως έλεγχο αναμενόμενης τιμής. Έτσι δημιουργούνται τα αντικείμενα της PMONExpectedValueCheck με τις αρχικές του τιμές.

```
explicit PMONExpectedValueCheck(ParameterId monitoredParameterId,
PMONRepetitionNumber repetitionNumber, PMONExpectedValue expectedValue,
PMONBitMask mask, EventDefinitionId unexpectedValueEvent)
    : PMON(monitoredParameterId, repetitionNumber, CheckType::ExpectedValue),
      expectedValue(expectedValue),
      mask(mask),
      unexpectedValueEvent(unexpectedValueEvent) {
}
```

Κώδικας 4.4: Constructor υποκλάσης PMONExpectedValueCheck.

4.2.2.3 Υποκλάση PMONDeltaCheck

Η `PMONDeltaCheck` επεκτείνει την κλάση `PMON`, κληρονομώντας τα χαρακτηριστικά και τις μεθόδους της και είναι υπεύθυνη για τη διαχείριση των ελέγχων διακυμάνσεων, παρακολουθώντας τις αλλαγές στις τιμές των παραμέτρων κατά τη διάρκεια του χρόνου. Στους παρακάτω πίνακες παρουσιάζονται οι μεταβλητές και οι συναρτήσεις της υποκλάσης μαζί με την περιγραφή τους.

Πίνακας 4.7: Μεταβλητές υποκλάσης PMONDeltaCheck.

Μεταβλητή	**Περιγραφή**
numberOfConsecutiveDeltaChecks	Ο αριθμός των διαδοχικών διακυμάνσεων που εξετάζονται.
lowDeltaThreshold	Η χαμηλή κατωφλιακή τιμή για τις διακυμάνσεις.
highDeltaThreshold	Η υψηλή κατωφλιακή τιμή για τις διακυμάνσεις.
belowLowThresholdEvent	Ταυτότητα του γεγονότος που εκδηλώνεται όταν η διακύμανση είναι κάτω από το χαμηλό κατώφλι.
aboveHighThresholdEvent	Ταυτότητα του γεγονότος που εκδηλώνεται όταν η διακύμανση υπερβαίνει το υψηλό κατώφλι.

Πίνακας 4.8: Συναρτήσεις υποκλάσης PMONDeltaCheck.

Συνάρτηση	**Περιγραφή**
getNumberOfConsecutiveDeltaChecks()	Επιστρέφει τον αριθμό των απαιτούμενων διαδοχικών ελέγχων διακύμανσης.
getLowDeltaThreshold()	Επιστρέφει τη χαμηλή κατωφλιακή τιμή.
getHighDeltaThreshold()	Επιστρέφει τη υψηλή κατωφλιακή τιμή.
getBelowLowThresholdEvent()	Επιστρέφει την ταυτότητα του γεγονότος για το κάτω από το χαμηλό κατώφλι.
getAboveHighThresholdEvent()	Επιστρέφει την ταυτότητα του γεγονότος για το πάνω από το υψηλό κατώφλι.

Constructor της PMONDeltaCheck

Στην τελευταία από τις υποκλάσεις, ο `constructor` της καλεί αυτόν της κεντρικής κλάσης `PMON`, περνώντας τις παραμέτρους `monitoredParameterId`, `repetitionNumber`, και `CheckType::Delta`, για τον καθορισμό του τύπου του ελέγχου ως έλεγχο διακυμάνσεων (δέλτα). Αυτή η διαδικασία οδηγεί στην δημιουργία αρχικοποιημένων αντικειμένων της `PMONDeltaCheck`.

```
explicit PMONDeltaCheck(ParameterId monitoredParameterId, PMONRepetitionNumber
repetitionNumber, NumberOfConsecutiveDeltaChecks numberOfConsecutiveDeltaChecks,
DeltaThreshold lowDeltaThreshold, EventDefinitionId belowLowThresholdEvent,
DeltaThreshold highDeltaThreshold, EventDefinitionId aboveHighThresholdEvent)
    : PMON(monitoredParameterId, repetitionNumber, CheckType::Delta),
numberOfConsecutiveDeltaChecks(numberOfConsecutiveDeltaChecks),
lowDeltaThreshold(lowDeltaThreshold),
      belowLowThresholdEvent(belowLowThresholdEvent),
highDeltaThreshold(highDeltaThreshold),
      aboveHighThresholdEvent(aboveHighThresholdEvent){
}
```

Κώδικας 4.5: Constructor υποκλάσης PMONDeltaCheck.

4.2.3 Κλάση OnBoardMonitoringService

Η κλάση `OnBoardMonitoringService` επεκτείνει τη βασική κλάση `Service` και είναι υπεύθυνη για τη διαχείριση των ορισμών παρακολούθησης παραμέτρων (Parameter Monitoring Definitions), ορίζοντας τις απαραίτητες μεθόδους για την προσθήκη, διαγραφή και αναφορά των ελέγχων παραμέτρων αλλά και σημαντικές μεταβλητές. Παρακάτω παρουσιάζονται αναλυτικά τα βασικά σημεία του κώδικα. Ο κώδικας που αναλύεται παρακάτω βρίσκεται στο αρχείο `OnBoardMonitoringService.hpp` ενώ η υλοποίηση αυτών των μεθόδων βρίσκεται στο αντίστοιχο αρχείο `.cpp` που θα εξεταστεί στο υποκεφάλαιο 4.3.

Περιεχόμενα Header Files

Το αρχείο περιλαμβάνει μια σειρά από header files, τα οποία είναι απαραίτητα για τη λειτουργία της κλάσης:

- `ECSS_Definitions.hpp`: Περιέχει βασικούς τύπους δεδομένων και σταθερές που χρησιμοποιούνται σε όλες τις υπηρεσίες ECSS.
- `Helpers/PMON.hpp`: Περιέχει την υλοποίηση της κλάσης `PMON` και των υποκλάσεών της.
- `Helpers/Parameter.hpp`: Παρέχει τη δυνατότητα διαχείρισης των παραμέτρων.
- `Message.hpp` και `Service.hpp`: Περιέχουν τη βασική δομή για την ανταλλαγή μηνυμάτων και την υλοποίηση των υπηρεσιών.
- βιβλιοθήκες `etl`[16]: Χρησιμοποιούνται για την υλοποίηση container (π.χ., map, vector) με στατική κατανομή μνήμης, που είναι κατάλληλη για χρήση σε περιβάλλοντα με περιορισμένους πόρους, όπως τα embedded συστήματα.

Ιδιότητες της Κλάσης

Πίνακας 4.9: Ιδιωτικές ιδιότητες κλάσης OnBoardMonitoringService

Ιδιωτικές Ιδιότητες (Private Properties)	**Περιγραφή**
parameterMonitoringList	Ένας χάρτης που αποθηκεύει τους ορισμούς παρακολούθησης παραμέτρων, με κλειδιά τύπου `uint16_t`.
MaximumNumberOfChecksLimitCheck	Σταθερά που καθορίζει τον μέγιστο αριθμό ελέγχων για κάθε Limit Check.
MaximumNumberOfChecksExpectedValueCheck	Σταθερά που καθορίζει τον μέγιστο αριθμό ελέγχων για κάθε Expected Value Check.
MaximumNumberOfChecksDeltaCheck	Σταθερά που καθορίζει τον μέγιστο αριθμό ελέγχων για κάθε Delta Check.
limitChecks	Ένα etl::vector (διάνυσμα) που αποθηκεύει τους ορισμούς PMONLimitCheck πριν προστεθούν στον χάρτη.
expectedValueChecks	Ένα etl::vector (διάνυσμα) που αποθηκεύει τους ορισμούς PMONExpectedValueCheck πριν προστεθούν στον χάρτη.
deltaChecks	Ένα etl::vector (διάνυσμα) που αποθηκεύει τους ορισμούς PMONDeltaCheck πριν προστεθούν στον χάρτη.

Πίνακας 4.10: Δημόσιες ιδιότητες κλάσης OnBoardMonitoringService

Δημόσιες Ιδιότητες (Public Properties)	**Περιγραφή**
ServiceType	Σταθερά που ορίζει τον τύπο της υπηρεσίας (12 για την ST[12]).
MessageType	Λίστα τύπου Enum (απαρίθμηση) που περιέχει τους τύπους μηνυμάτων για την υπηρεσία παρακολούθησης παραμέτρων.
maximumTransitionReportingDelay	Η μέγιστη χρονική καθυστέρηση μεταξύ δύο αναφορών μετάβασης, σε μονάδες του ελάχιστου διαστήματος δειγματοληψίας παραμέτρου.
parameterMonitoringFunctionStatus	Boolean που δηλώνει αν η παρακολούθηση παραμέτρων είναι ενεργοποιημένη ή όχι.

Μέθοδοι της Κλάσης

Η κλάση `OnBoardMonitoringService` περιλαμβάνει μια σειρά από μεθόδους που επιτρέπουν την πλήρη διαχείριση των παραμέτρων παρακολούθησης. Παρακάτω παρατίθενται οι πιο σημαντικές μέθοδοι με αναλυτικές περιγραφές των λειτουργιών τους.

Constructor

Ο `constructor` αρχικοποιεί την υπηρεσία και ορίζει τον τύπο της υπηρεσίας στο `ServiceType`. Αυτό σημαίνει ότι η υπηρεσία θα αναγνωρίζεται ως τύπος 12, σύμφωνα με το πρότυπο ECSS.

```
inline static constexpr ServiceTypeNum ServiceType = 12;

...

OnBoardMonitoringService() {
    serviceType = ServiceType;
}
```

Κώδικας 4.6: Constructor κλάσης OnBoardMonitoringService.

Μέθοδοι Προσθήκης Ορισμών Παρακολούθησης

Αυτές οι μέθοδοι είναι υπεύθυνες για την προσθήκη νέων ορισμών παρακολούθησης στον χάρτη `parameterMonitoringList`. Η διαδικασία δεν είναι όμως τόσο απλή καθώς οι νέοι ορισμοί προστίθενται πρώτα σε ένα `etl::vector`, ανάλογα με τον τύπο ελέγχου τους και στη συνέχεια μέσω αυτού προστίθενται τελικά στον χάρτη τον ορισμών. Αυτή η ροή εξασφαλίζει ότι οι ορισμοί δεν χάνονται και διατηρούνται σε μια οργανωμένη δομή για εύκολη πρόσβαση και διαχείριση.

Παράμετροι των Μεθόδων

- `ParameterId PMONId`: Ο μοναδικός αναγνωριστικός αριθμός της παραμέτρου που παρακολουθείται.

Ανάλογα τον τύπο ελέγχου μία εκ των παρακάτω:

- `PMONLimitCheck& limitCheck`: Αναφορά σε αντικείμενο τύπου `PMONLimitCheck` που περιέχει τα όρια παρακολούθησης.
- `PMONExpectedValueCheck& expectedValueCheck`: Αναφορά σε αντικείμενο τύπου `PMONExpectedValueCheck` που περιέχει την αναμενόμενη τιμή παρακολούθησης.
- `PMONDeltaCheck& deltaCheck`: Αναφορά σε αντικείμενο τύπου `PMONDeltaCheck` που περιέχει τα όρια διακύμανσης παρακολούθησης.

Μέθοδος Προσθήκης Ορισμού Limit Check

Η μέθοδος `addPMONLimitCheck` προσθέτει έναν νέο ορισμό παρακολούθησης ορίων (limit check) στη λίστα παρακολούθησης. Αρχικά, η συνάρτηση `push_back` προσθέτει το αντικείμενο `limitCheck` στο τέλος του `etl::vector` `limitChecks`, διασφαλίζοντας ότι όλοι οι ορισμοί παρακολούθησης ορίων αποθηκεύονται σε μια διαδοχική δομή δεδομένων. Στη συνέχεια το `etl::pair` δημιουργεί ένα ζεύγος που αποτελείται από το `PMONId` και μια αναφορά στον τελευταίο ορισμό που προστέθηκε στον `limitChecks` (χρησιμοποιώντας `etl::ref`). Τελικά η συνάρτηση `insert` προσθέτει το ζεύγος κλειδιού-τιμής (key-value pair) στον χάρτη `parameterMonitoringList`, εξασφαλίζοντας ότι ο χάρτης περιέχει μια αναφορά στον ορισμό παρακολούθησης που αποθηκεύεται στον `limitChecks`, επιτρέποντας την εύκολη πρόσβαση και διαχείριση των ορισμών.

```
/**
 * Adds a new Parameter Monitoring Limit Check to the parameter monitoring list.
 */
void addPMONLimitCheck(ParameterId PMONId, PMONLimitCheck& limitCheck) {
    limitChecks.push_back(limitCheck);
    parameterMonitoringList.insert(etl::pair<const ParameterId,
etl::reference_wrapper<PMON>>(PMONId, etl::ref(limitChecks.back())));
}
```

Κώδικας 4.7: Μέθοδος Προσθήκης Ορισμού Limit Check.

Μέθοδος Προσθήκης Ορισμού Expected Value Check

Αντίστοιχα ακολουθώντας την ίδια λογική ακριβώς πραγματοποιείται η προσθήκη ενός ορισμού με τύπο ελέγχου αναμενόμενης τιμής

```
/**
 * Adds a new Parameter Monitoring Expected Value Check to the parameter
monitoring list.
 */
void addPMONExpectedValueCheck(ParameterId PMONId, PMONExpectedValueCheck&
expectedValueCheck) {
    expectedValueChecks.push_back(expectedValueCheck);
    parameterMonitoringList.insert(etl::pair<const ParameterId,
etl::reference_wrapper<PMON>>(PMONId, etl::ref(expectedValueChecks.back())));
}
```

Κώδικας 4.8: Μέθοδος Προσθήκης Ορισμού Expected Value Check

Μέθοδος Προσθήκης Ορισμού Delta Check

Ομοίως για την προσθήκη ενός ορισμού με τύπο ελέγχου δέλτα.

```
/**
 * Adds a new Parameter Monitoring Delta Check to the parameter monitoring
list.
 */
void addPMONDeltaCheck(ParameterId PMONId, PMONDeltaCheck& deltaCheck) {
    deltaChecks.push_back(deltaCheck);
    parameterMonitoringList.insert(etl::pair<const ParameterId,
etl::reference_wrapper<PMON>>(PMONId, etl::ref(deltaChecks.back())));
}
```

Κώδικας 4.9: Μέθοδος Προσθήκης Ορισμού Μέθοδος Προσθήκης Ορισμού Delta Check

Μέθοδοι Διαχείρισης Λίστας Παρακολούθησης

Οι μέθοδοι αυτοί αποτελούν εργαλεία για την εκκαθάριση της λίστας παραμέτρων, την ανάκτηση συγκεκριμένων ορισμών, τον έλεγχο αν η λίστα είναι κενή και την καταμέτρηση των στοιχείων στον χάρτη-λίστας.

Εκκαθάριση Λίστας Παρακολούθησης

Η μέθοδος `clearParameterMonitoringList` χρησιμοποιεί τη συνάρτηση `clear` της `etl::map`, η οποία αφαιρεί όλα τα στοιχεία από τον χάρτη `parameterMonitoringList`. Αυτή η λειτουργία επαναφέρει τη λίστα παρακολούθησης στην αρχική του κατάσταση, διαγράφοντας κάθε ορισμό παρακολούθησης που έχει προηγουμένως προστεθεί.

```
/**
 * This function clears the Parameter Monitoring List map.
 */
void clearParameterMonitoringList() {
    parameterMonitoringList.clear();
}
```

Κώδικας 4.10: Μέθοδος εκκαθάρισης λίστας παρακολούθησης παραμέτρων

Ανάκτηση Ορισμού Παρακολούθησης

Η μέθοδος `getPMONDefinition` χρησιμοποιεί τη συνάρτηση `at` της `etl::map` για να ανακτήσει ένα στοιχείο από τον χάρτη `parameterMonitoringList` με βάση το κλειδί `PMONId`. Η `at` επιστρέφει μια αναφορά στο στοιχείο που αντιστοιχεί στο δεδομένο κλειδί και αν το κλειδί δεν υπάρχει στον χάρτη, η `at` θα πετάξει μια εξαίρεση `std::out_of_range`. Αυτό εξασφαλίζει ότι η μέθοδος παρέχει

ασφαλή πρόσβαση στα στοιχεία του χάρτη, επιτρέποντας την ανάκτηση συγκεκριμένων ορισμών παρακολούθησης.

```
/**
 * @param PMONId
 * @return Parameter Monitoring definition
 */
etl::reference_wrapper<PMON> getPMONDefinition(ParameterId PMONId) {
    return parameterMonitoringList.at(PMONId);
}
```

Κώδικας 4.11: Μέθοδος ανάκτησης ορισμού παρακολούθησης

Έλεγχος Κενής Λίστας

Η μέθοδος `isPMONListEmpty` χρησιμοποιεί τη συνάρτηση `empty` της `etl::map`, η οποία επιστρέφει `true` αν ο χάρτης `parameterMonitoringList` είναι κενός, και `false` αν περιέχει στοιχεία. Αυτή η μέθοδος είναι χρήσιμη για τον έλεγχο της κατάστασης της λίστας παρακολούθησης, επιτρέποντας στον κώδικα να επιβεβαιώσει αν υπάρχουν ορισμοί παρακολούθησης αποθηκευμένοι στον χάρτη.

```
/**
 * @return true if PMONList is empty.
 */
bool isPMONListEmpty() {
    return parameterMonitoringList.empty();
}
```

Κώδικας 4.12: Μέθοδος ελέγχου κενής λίστας

Στο τέλος της κλάσης `OnBoardMonitoringService` ορίζονται οι συναρτήσεις που διαχειρίζονται την παρακολούθηση των παραμέτρων όπως ορίζονται στο πρότυπο ECSS. Αυτές οι μέθοδοι χειρίζονται τα μηνύματα που λαμβάνονται και ορίζουν τις αντίστοιχες ενέργειες που πρέπει να εκτελεστούν. Η αναλυτική τους λειτουργία και δομή περιγράφεται στο ακόλουθο κεφάλαιο.

```
/**
 * Enables the PMON definitions which correspond to the ids in TC[12,1].
 */
void enableParameterMonitoringDefinitions(Message& message);

/**
 * Disables the PMON definitions which correspond to the ids in TC[12,2].
 */
void disableParameterMonitoringDefinitions(Message& message);

...
```

Κώδικας 4.13: Ορισμός μεθόδων διαχείρισης παρακολούθησης

5 Ανάλυση Μεθόδων κλάσης OnBoardMonitoringService

5.1 Εισαγωγή

Στο αρχείο πηγαίου κώδικα `OnBoardMonitoringService.cpp`, υλοποιούνται οι μέθοδοι που παρέχουν τη λειτουργικότητα αυτής της αντίστοιχης κλάσης. Αυτές οι μέθοδοι χειρίζονται τις τηλε-εντολές για ενεργοποίηση, απενεργοποίηση, προσθήκη, διαγραφή, τροποποίηση και αναφορά των ορισμών παρακολούθησης παραμέτρων. Το αρχείο αυτό βρίσκεται στο subdirectory `src/Services` του αποθετηρίου των υπηρεσιών ECSS.

Περιεχόμενα Header Files

Το αρχείο περιλαμβάνει μια σειρά από header files, τα οποία είναι απαραίτητα για τη λειτουργία της κλάσης `OnBoardMonitoringService`:

- `ECSS_Configuration.hpp`: Περιλαμβάνει το αρχείο διαμόρφωσης ECSS που ορίζει τις παραμέτρους και τις ρυθμίσεις για την υλοποίηση των υπηρεσιών ECSS.
- `Message.hpp`: Περιλαμβάνει τον ορισμό της κλάσης `Message` για τη διαχείριση των μηνυμάτων τηλε-εντολών και τηλεμετρίας.
- `ServicePool.hpp`: Περιλαμβάνει τον ορισμό της κλάσης `ServicePool` που διαχειρίζεται τις διάφορες υπηρεσίες του συστήματος.
- `Services/OnBoardMonitoringService.hpp`: Περιλαμβάνει τον ορισμό της κλάσης OnBoardMonitoringService για την παρακολούθηση παραμέτρων.

Στα επόμενα υποκεφάλαια ακολουθεί η τεχνική περιγραφή της υλοποίησης των μεθόδων της κλάσης `OnBoardMonitoringService`. Η ανάπτυξη και ο σχεδιασμός του κώδικα αυτών των μεθόδων αποτέλεσε μια διαδικασία «μετάφρασης» των οδηγιών του πρότυπου ECSS, στη γλώσσα πραγματισμού C++, τηρώντας πάντα όλες τις κατευθυντήριες γραμμές της ομάδας.

5.2 Μέθοδος ενεργοποίησης ορισμών παρακολούθησης παραμέτρων

Το πρότυπο ECSS στην ενότητα 6.12.3.6.1 περιγράφει την διαδικασία ενεργοποίησης των ορισμών παρακολούθησης παραμέτρων. Συγκεκριμένα, το πρότυπο λέει ότι η υπηρεσία παρακολούθησης παραμέτρων πρέπει να παρέχει τη δυνατότητα ενεργοποίησης των ορισμών παρακολούθησης παραμέτρων μέσω μηνυμάτων τύπου "TC[12,1] enable parameter monitoring definitions". Κάθε αίτημα πρέπει να περιέχει μία ή περισσότερες εντολές για την ενεργοποίηση των ορισμών παρακολούθησης, και κάθε εντολή πρέπει να περιλαμβάνει τον αναγνωριστικό κωδικό του

ορισμού παρακολούθησης παραμέτρων. Η υπηρεσία πρέπει να απορρίπτει οποιαδήποτε εντολή εάν ο αναγνωριστικός κωδικός δεν υπάρχει στη λίστα `PMON` ή αν ο ορισμός παρακολούθησης προστατεύεται από λειτουργικό υποσύστημα παρακολούθησης. Για κάθε έγκυρη εντολή, η υπηρεσία πρέπει να μηδενίζει τον μετρητή επαναλήψεων και να θέτει την κατάσταση `PMON` του ορισμού σε "enabled". Στην υλοποίηση του κώδικα, αυτά επιτυγχάνονται ως εξής:

```
void OnBoardMonitoringService::enableParameterMonitoringDefinitions(Message&
message) {
    if (!message.assertTC(ServiceType, EnableParameterMonitoringDefinitions))
{
        return;
    }
```

Κώδικας 5.1: Έλεγχος εγκυρότητας τηλε-εντολής ενεργοποίησης παρακολούθησης παραμέτρων

Η μέθοδος ξεκινάει ελέγχοντας αν το μήνυμα είναι έγκυρη τηλε-εντολή του τύπου "TC[12,1] enable parameter monitoring definitions" για την υπηρεσία `ServiceType` η οποία έχει αρχικοποιηθεί στην κλάση `OnBoardMonitoringService` με την τιμή 12. Αν η τηλε-εντολή δεν είναι έγκυρη, η μέθοδος επιστρέφει αμέσως, όπως απαιτεί το πρότυπο.

```
uint16_t const numberOfPMONDefinitions = message.readUint16();
    for (uint16_t i = 0; i < numberOfPMONDefinitions; i++) {
       const ParameterId currentId = message.read<ParameterId>();
       auto definition = parameterMonitoringList.find(currentId);
       if (definition == parameterMonitoringList.end()) {
          ErrorHandler::reportError(
              message,
ErrorHandler::ExecutionStartErrorType::GetNonExistingParameterMonitoringDefinition);
          continue;
       }
       definition->second.get().repetitionNumber = 0;
       definition->second.get().monitoringEnabled = true;
    }
}
```

Κώδικας 5.2: Λογική ενεργοποίησης παρακολούθησης παραμέτρων ανά επανάληψη

Στη συνέχεια, διαβάζει τον αριθμό των ορισμών παρακολούθησης παραμέτρων από το μήνυμα και τον αποθηκεύει στη σταθερά `numberOfPMONDefinitions`. Ένας βρόχος `for` επαναλαμβάνεται τόσες φορές όσες και το πλήθος των ορισμών PMON. Για κάθε επανάληψη, διαβάζεται το ID του

τρέχοντος ορισμού παρακολούθησης από το μήνυμα και αποθηκεύεται στη μεταβλητή `currentId`. Χρησιμοποιώντας αυτό, αναζητείται ο αντίστοιχος ορισμός στη λίστα παρακολούθησης παραμέτρων `parameterMonitoringList`. Αν ο ορισμός δεν βρεθεί, η μέθοδος απορρίπτει την εντολή και αναφέρει ένα σφάλμα μέσω της κλάσης `ErrorHandler` και συνεχίζεται η επόμενη επανάληψη του βρόχου , όπως απαιτεί το πρότυπο.

Η κλάση `ErrorHandler` είναι υπεύθυνη για τη διαχείριση και αναφορά των σφαλμάτων που προκύπτουν κατά την εκτέλεση των εντολών. Παρέχει μεθόδους για την αναφορά διαφόρων τύπων σφαλμάτων και εξασφαλίζει ότι τα σφάλματα καταγράφονται και αναφέρονται με έναν τυποποιημένο τρόπο και χρησιμοποιείται και από άλλες υπηρεσίες.

Αν ο ορισμός βρεθεί, η μέθοδος μηδενίζει τον μετρητή επαναλήψεων `repetitionNumber` του ορισμού και ενεργοποιεί την παρακολούθηση ρυθμίζοντας την `monitoringEnabled` σε `true`. Ο βρόχος ολοκληρώνεται και η μέθοδος τελειώνει, έχοντας ενεργοποιήσει όλους τους ορισμούς παρακολούθησης που αναφέρονται στο μήνυμα.

5.3 Μέθοδος απενεργοποίησης ορισμών παρακολούθησης παραμέτρων

Το πρότυπο ECSS στην ενότητα 6.12.3.6.2 περιγράφει την διαδικασία απενεργοποίησης των ορισμών παρακολούθησης παραμέτρων. Ειδικότερα, το πρότυπο αναφέρει ότι η υπηρεσία παρακολούθησης παραμέτρων πρέπει να παρέχει τη δυνατότητα απενεργοποίησης των ορισμών παρακολούθησης παραμέτρων μέσω μηνυμάτων τύπου "TC[12,2] disable parameter monitoring definitions". Κάθε αίτημα πρέπει να περιέχει μία ή περισσότερες εντολές για την απενεργοποίηση των ορισμών παρακολούθησης, και κάθε εντολή πρέπει να περιλαμβάνει τον αναγνωριστικό κωδικό του ορισμού παρακολούθησης παραμέτρων. Επιπλέον, η υπηρεσία πρέπει να απορρίπτει οποιαδήποτε εντολή εάν ο αναγνωριστικός κωδικός δεν υπάρχει στη λίστα `PMON` ή αν ο ορισμός παρακολούθησης προστατεύεται από λειτουργικό υποσύστημα παρακολούθησης. Για κάθε έγκυρη εντολή, η υπηρεσία πρέπει να θέτει την κατάσταση `PMON` του ορισμού σε "disabled" και την κατάσταση ελέγχου σε "unchecked". Στην υλοποίηση του κώδικα, αυτά επιτυγχάνονται ως εξής:

```
void OnBoardMonitoringService::disableParameterMonitoringDefinitions(Message&
message) {
    if (!message.assertTC(ServiceType, DisableParameterMonitoringDefinitions)) {
        return;
    }
```

Κώδικας 5.3: Έλεγχος εγκυρότητας τηλε-εντολής απενεργοποίησης παρακολούθησης παραμέτρων

Η μέθοδος αρχίζει ελέγχοντας αν το μήνυμα αποτελεί έγκυρη τηλε-εντολή του τύπου "TC[12,2] disable parameter monitoring definitions" για την υπηρεσία `ServiceType` (12). Αν η τηλε-εντολή δεν είναι έγκυρη, η μέθοδος επιστρέφει αμέσως, όπως απαιτεί το πρότυπο.

```
     uint16_t const numberOfPMONDefinitions = message.readUint16();
    for (uint16_t i = 0; i < numberOfPMONDefinitions; i++) {
       const ParameterId currentId = message.read<ParameterId>();
       auto definition = parameterMonitoringList.find(currentId);
       if (definition == parameterMonitoringList.end()) {
          ErrorHandler::reportError(
              message,
ErrorHandler::ExecutionStartErrorType::GetNonExistingParameterMonitoringDefinition);

          continue;
       }

       definition->second.get().monitoringEnabled = false;
       definition->second.get().checkingStatus = PMON::Unchecked;
    }
```

Κώδικας 5.4: Λογική απενεργοποίησης παρακολούθησης παραμέτρων ανά επανάληψη

Στη συνέχεια, διαβάζει το πλήθος των ορισμών παρακολούθησης παραμέτρων από το μήνυμα και τον αποθηκεύει στη σταθερά `numberOfPMONDefinitions`. Ένας βρόχος `for` επαναλαμβάνεται τόσες φορές, όσες και ο αριθμός των ορισμών `PMON` και σε κάθε επανάληψη διαβάζει το ID του τρέχοντος ορισμού παρακολούθησης από το μήνυμα αποθηκεύοντας το στη μεταβλητή `currentId`. Χρησιμοποιώντας το `currentId`, αναζητείται ο αντίστοιχος ορισμός στη λίστα παρακολούθησης παραμέτρων `parameterMonitoringList`. Αν ο ορισμός δεν βρεθεί, η μέθοδος απορρίπτει την εντολή και αναφέρει ένα σφάλμα μέσω της `ErrorHandler` και συνεχίζεται η επόμενη επανάληψη του βρόχου, όπως απαιτεί το πρότυπο.

Αν ο ορισμός βρεθεί, η μέθοδος θέτει την κατάσταση παρακολούθησης `monitoringEnabled` σε `false` και την κατάσταση ελέγχου `checkingStatus` σε `Unchecked`, σύμφωνα με το πρότυπο. Ο βρόχος ολοκληρώνεται και η μέθοδος τελειώνει, έχοντας απενεργοποιήσει όλους τους ορισμούς παρακολούθησης που αναφέρονται στο μήνυμα.

5.4 Μέθοδος αλλαγής μέγιστης καθυστέρησης αναφοράς μετάβασης.

Το ECSS πρότυπο στην ενότητα 6.12.3.6.3 περιγράφει την διαδικασία αλλαγής της μέγιστης καθυστέρησης αναφοράς μεταβάσεων. Πιο αναλυτικά, το πρότυπο λέει ότι η υπηρεσία

παρακολούθησης παραμέτρων πρέπει να παρέχει τη δυνατότητα αλλαγής της μέγιστης καθυστέρησης αναφοράς μεταβάσεων μέσω μηνυμάτων τύπου "TC[12,3] change the maximum transition reporting delay". Κάθε αίτημα πρέπει να περιέχει ακριβώς μία εντολή για την αλλαγή της μέγιστης καθυστέρησης αναφοράς μεταβάσεων, και κάθε εντολή πρέπει να περιλαμβάνει την τιμή της νέας μέγιστης καθυστέρησης αναφοράς μεταβάσεων. Τελικά, η υπηρεσία πρέπει να ορίσει τη μέγιστη καθυστέρηση αναφοράς μεταβάσεων στην τιμή που καθορίζεται στην εντολή. Στην υλοποίηση του κώδικα, αυτά επιτυγχάνονται ως εξής:

```
void OnBoardMonitoringService::changeMaximumTransitionReportingDelay(Message&
message) {
    if (!message.assertTC(ServiceType, ChangeMaximumTransitionReportingDelay)) {
        return;
    }
    maximumTransitionReportingDelay = message.readUint16();
}
```

Κώδικας 5.5: Μέθοδος changeMaximumTransitionReportingDelay

Η μέθοδος `changeMaximumTransitionReportingDelay` ξεκινάει ελέγχοντας αν το μήνυμα είναι έγκυρη τηλε-εντολή του τύπου "TC[12,3] change the maximum transition reporting delay" για την υπηρεσία `ServiceType` (12). Σε περίπτωση που η τηλε-εντολή δεν είναι έγκυρη, η μέθοδος επιστρέφει αμέσως, όπως προβλέπεται.

Έπειτα, η μέθοδος διαβάζει την τιμή της μέγιστης καθυστέρησης αναφοράς μεταβάσεων από το μήνυμα και την αποθηκεύει στη μεταβλητή `maximumTransitionReportingDelay`. Με αυτό τον απλό τρόπο η μέθοδος ολοκληρώνεται έχοντας αλλάξει τη μέγιστη καθυστέρηση αναφοράς μεταβάσεων στην τιμή που παρέχεται από το μήνυμα, σύμφωνα με τις οδηγίες του προτύπου ECSS.

5.5 Μέθοδος διαγραφής όλων των ορισμών παρακολούθησης παραμέτρων

Το πρότυπο ECSS στην ενότητα 6.12.3.9.2 περιγράφει τη διαδικασία διαγραφής όλων των ορισμών παρακολούθησης παραμέτρων, εξηγώντας ότι η υπηρεσία παρακολούθησης παραμέτρων πρέπει να παρέχει τη δυνατότητα διαγραφής όλων των ορισμών παρακολούθησης παραμέτρων μέσω μηνυμάτων τύπου "TC[12,4] delete all parameter monitoring definitions". Κάθε αίτημα πρέπει να περιέχει ακριβώς μία εντολή για τη διαγραφή όλων των ορισμών παρακολούθησης παραμέτρων και η υπηρεσία πρέπει να απορρίπτει οποιοδήποτε αίτημα αν ένα ή περισσότερα `PMON` χρησιμοποιούνται από το `functional monitoring subservice`. Η υπηρεσία ST[12] πρέπει να διαγράφει όλες τις καταχωρήσεις στη λίστα `PMON` και να αναφέρει οποιοδήποτε σφάλμα.

Στην υλοποίηση του κώδικα, αυτά επιτυγχάνονται ως εξής:

```
void OnBoardMonitoringService::deleteAllParameterMonitoringDefinitions(const
Message& message) {
    if (!message.assertTC(ServiceType,
DeleteAllParameterMonitoringDefinitions)) {
        return;
    }
```

Κώδικας 5.6: Έλεγχος εγκυρότητας τηλε-εντολής διαγραφής ορισμών παρακολούθησης παραμέτρων

Η μέθοδος `deleteAllParameterMonitoringDefinition`s ξεκινάει ελέγχοντας αν το μήνυμα είναι έγκυρη τηλε-εντολή του τύπου "TC[12,4] delete all parameter monitoring definitions" για την υπηρεσία `ServiceType` (12). Αν η τηλε-εντολή δεν είναι έγκυρη, η μέθοδος επιστρέφει αμέσως, όπως απαιτεί το πρότυπο.

```
if (parameterMonitoringFunctionStatus ) {
    ErrorHandler::reportError(
        message,

ErrorHandler::ExecutionStartErrorType::InvalidRequestToDeleteAllParameterMon
itoringDefinitions);
    return;
}
parameterMonitoringList.clear();
```

Κώδικας 5.7: Λογική διαγραφής ορισμών παρακολούθησης παραμέτρων

Η μέθοδος ελέγχει αν η κατάσταση λειτουργίας `PMON` είναι "enabled" βάσει της κατάστασης της boolean μεταβλητής `parameterMonitoringFunctionStatus` η οποία είναι δηλωμένη στην κλάση `OnBoardMonitoringService.hpp`. Αν η τιμή της είναι `true` η μέθοδος αναφέρει σφάλμα και επιστρέφει, απορρίπτοντας το αίτημα, όπως απαιτείται από το πρότυπο. Αντίθετα, η μέθοδος χρησιμοποιεί την μέθοδο `clear` της `etl::map` για την διαγραφή όλων των ορισμών παρακολούθησης παραμέτρων.

5.6 Μέθοδος προσθήκης ορισμών παρακολούθησης

Το πρότυπο ECSS στην ενότητα 6.12.3.9.1 περιγράφει τη διαδικασία προσθήκης ορισμών παρακολούθησης παραμέτρων. Συγκεκριμένα, το πρότυπο λέει ότι η υπηρεσία παρακολούθησης παραμέτρων πρέπει να παρέχει τη δυνατότητα προσθήκης νέων ορισμών παρακολούθησης παραμέτρων μέσω μηνυμάτων τύπου "TC[12,5] add parameter monitoring definitions". Κάθε αίτημα πρέπει να περιέχει μία ή περισσότερες εντολές για μία προσθήκη και κάθε εντολή πρέπει να περιλαμβάνει το περιεχόμενο του ορισμού παρακολούθησης παραμέτρων. Επιπλέον η υπηρεσία παρακολούθησης παραμέτρων πρέπει να απορρίπτει οποιαδήποτε εντολή αν:

- Η λίστα `PMON` είναι πλήρης.
- Ο αναγνωριστικός κωδικός υπάρχει ήδη στη λίστα `PMON`.
- Η παράμετρος που πρέπει να παρακολουθείται δεν είναι προσβάσιμη.
- Η παράμετρος εγκυρότητας δεν είναι προσβάσιμη.
- Το υψηλό όριο είναι χαμηλότερο από το χαμηλό όριο (για ελέγχους ορίων).
- Το υψηλό κατώφλι είναι χαμηλότερο από το χαμηλό κατώφλι (για ελέγχους διακυμάνσεων).
- Η υπηρεσία παρακολούθησης παραμέτρων πρέπει να προσθέτει κάθε νέο ορισμό παρακολούθησης στη λίστα `PMON`, να θέτει την κατάσταση ελέγχου σε "unchecked" και την κατάσταση `PMON` σε "disabled".

Στην υλοποίηση του κώδικα, αυτά επιτυγχάνονται ως εξής:

```
void OnBoardMonitoringService::addParameterMonitoringDefinitions(Message&
message) {
    message.assertTC(ServiceType, AddParameterMonitoringDefinitions);

    uint16_t numberOfIds = message.readUint16();

    for (uint16_t i = 0; i < numberOfIds; i++) {
       ParameterId currentPMONId = message.read<ParameterId>();
       ParameterId currentMonitoredParameterId = message.read<ParameterId>();
       PMONRepetitionNumber currentPMONRepetitionNumber =
message.read<PMONRepetitionNumber>();
       uint16_t checkTypeValue = message.readEnum8();
       auto currentCheckType = static_cast<PMON::CheckType>(checkTypeValue);
```

Κώδικας 5.8: Έλεγχος εγκυρότητας τηλε-εντολής και ανάγνωση παραμέτρων

Αρχικά η μέθοδος `addParameterMonitoringDefinitions` ελέγχει αν το μήνυμα είναι έγκυρη τηλε-εντολή του τύπου "TC[12,5] add parameter monitoring definitions" για την υπηρεσία `ServiceType` (12), επιστρέφοντας άμεσα αν η τηλε-εντολή δεν είναι έγκυρη. Στη συνέχεια, η μέθοδος διαβάζει τον αριθμό των ορισμών παρακολούθησης παραμέτρων από το μήνυμα και τον αποθηκεύει στη μεταβλητή `numberOfIds`. Ένας βρόχος `for` επαναλαμβάνεται τόσες φορές όσες και ο αριθμός των ορισμών `PMON`. Για κάθε επανάληψη, διαβάζονται οι σχετικές παράμετροι από το μήνυμα και αποθηκεύονται στις αντίστοιχες μεταβλητές `currentPMONId`, `currentMonitoredParameterId`, `currentPMONRepetitionNumber`, και `currentCheckType`.

```
auto parameterToBeAdded =
Services.parameterManagement.getParameter(currentMonitoredParameterId);
if (!parameterToBeAdded) {
    ErrorHandler::reportError(message,
ErrorHandler::ExecutionStartErrorType::GetNonExistingParameterMonitoringDefi
nition);
    continue;
}
if (parameterMonitoringList.find(currentPMONId) !=
parameterMonitoringList.end()) {
    ErrorHandler::reportError(message,
ErrorHandler::ExecutionStartErrorType::AddAlreadyExistingParameter);
    continue;
}
if (parameterMonitoringList.full()) {
    ErrorHandler::reportError(message,
ErrorHandler::ExecutionStartErrorType::ParameterMonitoringListIsFull);
    continue;
}
```

Κώδικας 5.9: Έλεγχος προσβασιμότητας και διαθεσιμότητας ορισμών παρακολούθησης

Η μέθοδος ελέγχει αν η παράμετρος που πρόκειται να προστεθεί είναι προσβάσιμη και αν δεν είναι, αναφέρει σφάλμα και συνεχίζει με την επόμενη εντολή. Επιπλέον, ελέγχει αν ο αναγνωριστικός κωδικός υπάρχει ήδη στη λίστα `PMON` με τη χρήση της μεθόδου `find` που παίρνει ως παράμετρο το αναγνωριστικό `currentId` ή αν η λίστα είναι πλήρης εφαρμόζοντας την συνάρτηση `full` στον χάρτη `parameterMonitoringList`. Σε κάθε τέτοια περίπτωση, αναφέρεται σφάλμα και συνεχίζει με την επόμενη εντολή.

```
switch (currentCheckType) {
       case PMON::CheckType::Limit: {
          PMONLimit lowLimit = message.read<PMONLimit>();
          EventDefinitionId belowLowLimitEventId =
message.read<EventDefinitionId>();
          PMONLimit highLimit = message.read<PMONLimit>();
          EventDefinitionId aboveHighLimitEventId =
message.read<EventDefinitionId>();
          if (highLimit <= lowLimit) {
             ErrorHandler::reportError(
                 message,
ErrorHandler::ExecutionStartErrorType::HighLimitIsLowerThanLowLimit);
             continue;
          }
          PMONLimitCheck limitCheck(currentMonitoredParameterId,
currentPMONRepetitionNumber,
                                    lowLimit, belowLowLimitEventId,
highLimit, aboveHighLimitEventId);
          limitCheck.checkingStatus = PMON::Unchecked;
          limitCheck.monitoringEnabled = false;
          addPMONLimitCheck(currentPMONId, limitCheck);
          break;
       }
```

Κώδικας 5.10: Προσθήκη ορισμού παρακολούθησης για τύπο Limit

Στην πρώτη γραμμή του *Κώδικα 5.10* ξεκινάει μία δήλωση `switch` που ελέγχει την τιμή του τύπου ελέγχου `currentCheckType`. Αν η τιμή είναι `PMON::CheckType::Limit`, τότε εκτελείται το επακόλουθο μπλοκ κώδικα μέσα στην `case` δήλωση. Ακολουθεί η ανάγνωση των παραμέτρων από το μήνυμα και η αποθήκευση των τιμών τους σε τοπικές μεταβλητές με τα αντίστοιχα ονόματα και εν συνεχεία ελέγχεται αν το υψηλό όριο είναι μικρότερο ή ίσο από το χαμηλό.

Αν η συνθήκη αυτή είναι αληθής, τότε αναφέρεται σφάλμα μέσω της κλάσης `ErrorHandler` και η τρέχουσα εντολή παραλείπεται με τη χρήση του `continue`. Στην αντίθετη περίπτωση, δημιουργείται ένα αντικείμενο `limitCheck` της υποκλάσης `PMONLimitCheck` με αρχικές τιμές αυτές που έχουν αποθηκευτεί από το μήνυμα προηγουμένως ενώ παράλληλα ορίζεται το `checkingStatus` ως `PMON::Unchecked` και το `monitoringEnabled` ως `false`, σύμφωνα με τις οδηγίες του προτύπου. Τέλος, ο νέος ορισμός `PMONLimitCheck` προστίθεται στη λίστα των ελέγχων παραμέτρων με τη χρήση της μεθόδου `addPMONLimitCheck`. Η `switch` δήλωση τερματίζεται με τη χρήση της εντολής `break`, η οποία σταματά την εκτέλεση της υπόλοιπης `switch` δήλωσης.

```
case PMON::CheckType::ExpectedValue: {
    PMONBitMask mask = message.read<PMONBitMask>();
    PMONExpectedValue expectedValue = message.read<PMONExpectedValue>();
    EventDefinitionId unexpectedValueEvent =
message.read<EventDefinitionId>();
    PMONExpectedValueCheck expectedValueCheck(currentMonitoredParameterId,
currentPMONRepetitionNumber,
                                              expectedValue, mask,
unexpectedValueEvent);
    expectedValueCheck.checkingStatus = PMON::Unchecked;
    expectedValueCheck.monitoringEnabled = false;
    addPMONExpectedValueCheck(currentPMONId, expectedValueCheck);
    break;
}
```

Κώδικας 5.11: Προσθήκη ορισμού παρακολούθησης για τύπο ExpectedValue

Στην δεύτερη περίπτωση η τιμή του `currentCheckType` είναι `PMON::CheckType::ExpectedValue`, τότε εκτελείται το μπλοκ κώδικα που φαίνεται παραπάνω στον *Κώδικα 5.11*, όπου αρχικά γίνεται ανάγνωση των παραμέτρων από το μήνυμα και η αποθήκευση των τιμών τους σε τοπικές μεταβλητές με τα αντίστοιχα ονόματα. Ακολουθεί η αρχικοποίηση ενός αντικειμένου της υποκλάσης `PMONExpectedValueCheck` με τις τιμές που λήφθηκαν από το μήνυμα και ο ορισμός του `checkingStatus` ως `PMON::Unchecked` και του `monitoringEnabled` ως `false`, σύμφωνα το πρότυπο. Τέλος, ο νέος ορισμός ` PMONExpectedValueCheck ` προστίθεται στη λίστα των ελέγχων παραμέτρων με τη χρήση της μεθόδου `addPMONExpectedValueCheck`. Η `switch` δήλωση τερματίζεται με τη χρήση της εντολής `break`, η οποία σταματά την εκτέλεση της υπόλοιπης `switch` δήλωσης.

```
case PMON::CheckType::Delta: {
    DeltaThreshold lowDeltaThreshold = message.read<DeltaThreshold>();
    EventDefinitionId belowLowThresholdEventId =
message.read<EventDefinitionId>();
    DeltaThreshold highDeltaThreshold = message.read<DeltaThreshold>();
    EventDefinitionId aboveHighThresholdEventId =
message.read<EventDefinitionId>();
    NumberOfConsecutiveDeltaChecks numberOfConsecutiveDeltaChecks =
message.read<NumberOfConsecutiveDeltaChecks>();
    if (highDeltaThreshold <= lowDeltaThreshold) {
       ErrorHandler::reportError(
           message,
ErrorHandler::ExecutionStartErrorType::HighThresholdIsLowerThanLowThreshold);
       continue;
    }
    PMONDeltaCheck deltaCheck(currentMonitoredParameterId,
currentPMONRepetitionNumber,
                              numberOfConsecutiveDeltaChecks, lowDeltaThreshold,
belowLowThresholdEventId, highDeltaThreshold, aboveHighThresholdEventId);
    deltaCheck.checkingStatus = PMON::Unchecked;
    deltaCheck.monitoringEnabled = false;
    addPMONDeltaCheck(currentPMONId, deltaCheck);
    break;
}
```

Κώδικας 5.12: Προσθήκη ορισμού παρακολούθησης για τύπο Delta

Στην τρίτη και τελευταία περίπτωση η τιμή του `currentCheckType` είναι `PMON::CheckType::ExpectedValue`, τότε εκτελείται το μπλοκ κώδικα που φαίνεται παραπάνω στον *Κώδικα 5.12*, όπου αρχικά γίνεται ανάγνωση των παραμέτρων από το μήνυμα και η αποθήκευση των τιμών τους σε τοπικές μεταβλητές με τα αντίστοιχα ονόματα. Στη συνέχεια πραγματοποιείται έλεγχος αν το χαμηλό κατώφλι είναι μεγαλύτερο από το υψηλό και αν η συνθήκη αυτή είναι αληθής τότε αναφέρεται σφάλμα μέσω της κλάσης `ErrorHandler` και η τρέχουσα εντολή παραλείπεται με τη χρήση του `continue`. Εάν η συνθήκη είναι αναληθής τότε αρχικοποιείται ένα αντικείμενο της υποκλάσης `PMONDeltaCheck` με τις τιμές που λήφθηκαν από το μήνυμα και ορίζονται το `checkingStatus` ως `PMON::Unchecked` και το `monitoringEnabled` ως `false`, σύμφωνα το πρότυπο. Τελικά ο νέος ορισμός `PMONDeltaCheck` προστίθεται στη λίστα των ελέγχων παραμέτρων με τη χρήση της μεθόδου `addPMONDeltaCheck `. Η `switch` δήλωση τερματίζεται με τη χρήση της εντολής `break`, η οποία σταματά την εκτέλεση της υπόλοιπης `switch` δήλωσης.

5.6 Μέθοδος διαγραφής ορισμών παρακολούθησης

Το πρότυπο ECSS στην ενότητα 6.12.3.9.3 περιγράφει τη διαδικασία διαγραφής ορισμών παρακολούθησης παραμέτρων, καθώς η υπηρεσία ST[12] πρέπει να παρέχει τη δυνατότητα διαγραφής ορισμών παρακολούθησης παραμέτρων μέσω μηνυμάτων τύπου "TC[12,6] delete parameter monitoring definitions". Κάθε αίτημα πρέπει να περιέχει μία ή περισσότερες εντολές για τη διαγραφή ορισμών παρακολούθησης παραμέτρων και η υπηρεσία πρέπει να απορρίπτει οποιοδήποτε αίτημα αν ο ορισμός παρακολούθησης παραμέτρων δεν βρίσκεται στη λίστα `PMON`, αν ο ορισμός έχει κατάσταση "enabled" ή αν χρησιμοποιείται από κάποιον ορισμό λειτουργικής παρακολούθησης. Στην υλοποίηση του κώδικα, αυτά επιτυγχάνονται ως εξής:

```
void OnBoardMonitoringService::deleteParameterMonitoringDefinitions(Message&
message) {
    message.assertTC(ServiceType, DeleteParameterMonitoringDefinitions);
    uint16_t numberOfIds = message.readUint16();
    for (uint16_t i = 0; i < numberOfIds; i++) {
       ParameterId currentPMONId = message.read<ParameterId>();

       if (parameterMonitoringList.find(currentPMONId) ==
parameterMonitoringList.end()) {
          ErrorHandler::reportError(message,
ErrorHandler::InvalidRequestToDeleteParameterMonitoringDefinition);
          continue;
       }

       if (getPMONDefinition(currentPMONId).get().monitoringEnabled) {
          ErrorHandler::reportError(message,
ErrorHandler::InvalidRequestToDeleteParameterMonitoringDefinition);
          continue;
       }

       parameterMonitoringList.erase(currentPMONId);
    }
}
```

Κώδικας 5.13: Έλεγχος εγκυρότητας τηλε-εντολής και διαγραφή ορισμών παρακολούθησης παραμέτρων

Σε πρώτο χρόνο η μέθοδος `deleteParameterMonitoringDefinitions` ελέγχει αν το μήνυμα είναι έγκυρη τηλε-εντολή του τύπου "TC[12,6] delete parameter monitoring definitions" για την υπηρεσία `ServiceType`. Αν η τηλε-εντολή δεν είναι έγκυρη, η μέθοδος επιστρέφει αμέσως, όπως απαιτεί το πρότυπο.

Σε δεύτερο χρόνο, διαβάζει τον αριθμό των ορισμών παρακολούθησης παραμέτρων από το μήνυμα και το αποθηκεύει στη σταθερά `numberOfIds`. Ένας βρόχος `for` επαναλαμβάνεται τόσες φορές όσες και το πλήθος των ορισμών `PMON`. Για κάθε επανάληψη, διαβάζεται το ID του τρέχοντος ορισμού παρακολούθησης από το μήνυμα και αποθηκεύεται στη μεταβλητή `currentPMONId`.

Η μέθοδος χρησιμοποιεί τη συνάρτηση `find` της `etl::map` για να αναζητήσει τον αντίστοιχο ορισμό στη λίστα παρακολούθησης παραμέτρων `parameterMonitoringList`. Αν ο ορισμός δεν βρεθεί στη λίστα, αναφέρεται σφάλμα μέσω της κλάσης `ErrorHandler` και η μέθοδος συνεχίζει στην επόμενη επανάληψη του βρόχου.

Τέλος, η μέθοδος ελέγχει αν η κατάσταση λειτουργίας του ορισμού είναι "enabled", και αναφέρει σφάλμα εάν όχι, συνεχίζοντας χρησιμοποιεί τη συνάρτηση `erase` της `etl::map` για να διαγράψει τον ορισμό παρακολούθησης παραμέτρων από τη λίστα `parameterMonitoringList`. Με αυτόν τον τρόπο, η μέθοδος αφαιρεί μεμονωμένους ορισμούς παρακολούθησης παραμέτρων, σύμφωνα με τις οδηγίες του προτύπου ECSS.

5.7 Μέθοδος τροποποίησης ορισμών παρακολούθησης παραμέτρων

Το πρότυπο ECSS στην ενότητα 6.12.3.9.4 περιγράφει τη διαδικασία τροποποίησης ορισμών παρακολούθησης παραμέτρων, εξηγώντας ότι η υπηρεσία παρακολούθησης παραμέτρων πρέπει να παρέχει τη δυνατότητα τροποποίησης ορισμών παρακολούθησης παραμέτρων μέσω μηνυμάτων τύπου "TC[12,7] modify parameter monitoring definitions". Κάθε αίτημα πρέπει να περιέχει μία ή περισσότερες εντολές για την τροποποίηση ορισμών παρακολούθησης παραμέτρων και η υπηρεσία πρέπει να απορρίπτει οποιοδήποτε αίτημα αν δεν πληροί τις συγκεκριμένες προϋποθέσεις που καθορίζονται στο πρότυπο. Στην υλοποίηση του κώδικα, αυτά επιτυγχάνονται ως εξής:

```
void
OnBoardMonitoringService::modifyParameterMonitoringDefinitions(Message&
message) {
    message.assertTC(ServiceType, ModifyParameterMonitoringDefinitions);

    uint16_t numberOfIds = message.readUint16();

    for (uint16_t i = 0; i < numberOfIds; i++) {

        ParameterId currentPMONId = message.read<ParameterId>();
        ParameterId currentMonitoredParameterId =
message.read<ParameterId>();
        PMONRepetitionNumber currentPMONRepetitionNumber =
message.read<PMONRepetitionNumber>();
        uint16_t currentCheckType = message.readEnum8();
```

Κώδικας 5.13: Έλεγχος εγκυρότητας τηλε-εντολής τροποποίησης ορισμών παρακολούθησης παραμέτρων

Η μέθοδος `modifyParameterMonitoringDefinitions` ξεκινάει ελέγχοντας αν το μήνυμα είναι έγκυρη τηλε-εντολή του τύπου "TC[12,7] modify parameter monitoring definitions" για την υπηρεσία `ServiceType` (12). Αν η τηλε-εντολή δεν είναι έγκυρη, η μέθοδος επιστρέφει αμέσως, όπως απαιτεί το πρότυπο. Ακολουθεί η ανάγνωση και αποθήκευση του αριθμού των ορισμών παρακολούθησης παραμέτρων από το μήνυμα στην σταθερά `numberOfIds`, και ξεκινάει ένας βρόχος `for` που επαναλαμβάνεται τόσες φορές όσες και το πλήθος των ορισμών PMON. Σε κάθε επανάληψη διαβάζονται το αναγνωριστικό του τρέχοντος ορισμού παρακολούθησης, αυτό του παρακολουθούμενου ορισμού, ο αριθμός επαναλήψεων, ο τύπος ελέγχου και αποθηκεύονται στις αντίστοιχες μεταβλητές.

```
auto it = parameterMonitoringList.find(currentPMONId);

if (it == parameterMonitoringList.end()) {
    ErrorHandler::reportError(
        message,
ErrorHandler::ExecutionStartErrorType::ModifyParameterNotInTheParameterMon
itoringList);
    return;
}

if (getPMONDefinition(currentPMONId).get().monitoredParameterId !=
currentMonitoredParameterId) {
    ErrorHandler::reportError(message,
ErrorHandler::ExecutionStartErrorType::

DifferentParameterMonitoringDefinitionAndMonitoredParameter);
    return;
}
```

Κώδικας 5.14: Έλεγχος ύπαρξης του ορισμού στη λίστα PMON και επιβεβαίωση ταυτότητας παραμέτρου

Η μέθοδος χρησιμοποιεί τη συνάρτηση `find` της `etl::map` για να αναζητήσει τον αντίστοιχο ορισμό στη λίστα παρακολούθησης παραμέτρων `parameterMonitoringList`. Αν ο ορισμός δεν βρεθεί στη λίστα, αναφέρεται σφάλμα μέσω της `ErrorHandler` και η μέθοδος επιστρέφει, όπως απαιτείται από το πρότυπο. Έπειτα, ελέγχεται αν ο τρέχων ορισμός ταιριάζει με τον παρακολουθούμενο ορισμό και εάν δεν ταιριάζει, αναφέρεται σφάλμα και η μέθοδος επιστρέφει.

```
PMON& pmon = it->second.get();
pmon.repetitionCounter = 0;
pmon.repetitionNumber = currentPMONRepetitionNumber;
pmon.checkingStatus = PMON::Unchecked;
```

Κώδικας 5.15: Επαναφορά της κατάστασης του ορισμού

Η γραμμή `PMON& pmon = it->second.get();` χρησιμοποιείται για την ανάκτηση της αναφοράς στον ορισμό παρακολούθησης παραμέτρων από τη λίστα. Η `etl::map` αποθηκεύει ζεύγη κλειδιού-τιμής όπου η τιμή είναι ένας `reference_wrapper` που περιέχει αναφορά σε ένα αντικείμενο PMON. Η `it->second` αναφέρεται στη δεύτερη τιμή του ζεύγους (δηλαδή στην αναφορά του reference_wrapper), και η κλήση της μεθόδου get() στον reference_wrapper επιστρέφει την αρχική αναφορά στο αντικείμενο PMON. Η μέθοδος επαναφέρει τον μετρητή επαναλήψεων και τον αριθμό επαναλήψεων του ορισμού, ενώ θέτει την κατάσταση ελέγχου του ορισμού σε `Unchecked`, καλύπτοντας τις απαιτήσεις του προτύπου για την επαναφορά της κατάστασης του ορισμού.

```
switch (static_cast<PMON::CheckType>(currentCheckType)) {
    case PMON::CheckType::Limit: {
       PMONLimit lowLimit = message.read<PMONLimit>();
       EventDefinitionId belowLowLimitEventId =
message.read<EventDefinitionId>();
       PMONLimit highLimit = message.read<PMONLimit>();
       EventDefinitionId aboveHighLimitEventId =
message.read<EventDefinitionId>();

       if (highLimit <= lowLimit) {
          ErrorHandler::reportError(
              message,
ErrorHandler::ExecutionStartErrorType::HighLimitIsLowerThanLowLimit);
          continue;
       }

       auto& limitCheck = static_cast<PMONLimitCheck&>(pmon);
       limitCheck.lowLimit = lowLimit;
       limitCheck.belowLowLimitEvent = belowLowLimitEventId;
       limitCheck.highLimit = highLimit;
       limitCheck.aboveHighLimitEvent = aboveHighLimitEventId;
       break;
    }
```

Κώδικας 5.16: Τροποποίηση ορισμών παρακολούθησης τύπου Limit

Γενικά η μέθοδος τροποποιεί τον ορισμό παρακολούθησης παραμέτρων ανάλογα με τον τύπο ελέγχου που καθορίζεται από την εντολή. Για τον τύπο `Limit`, διαβάζονται τα χαμηλά και υψηλά όρια και τα αντίστοιχα `IDs` των γεγονότων από το μήνυμα. Αν το υψηλό όριο είναι μικρότερο ή ίσο με το χαμηλό όριο, αναφέρεται σφάλμα και η εντολή απορρίπτεται. Αντιστοίχως, τροποποιούνται οι ορισμοί ExpectedValue και Delta με τα κατάλληλα πεδία.

Πιο αναλυτικά, , για να μπορέσουμε να τροποποιήσουμε συγκεκριμένα πεδία που αφορούν μόνο τον τύπο `Limit`, είναι απαραίτητο να κάνουμε `cast` το γενικό αντικείμενο `PMON` που ανακτήθηκε προηγουμένως στο πιο εξειδικευμένο `PMONLimitCheck`. Η γραμμή `auto& limitCheck = static_cast<PMONLimitCheck&>(pmon);` πραγματοποιεί ακριβώς αυτή τη διαδικασία, αφού η χρήση του `static_cast` εξασφαλίζει ότι το αντικείμενο `pmon` μετατρέπεται σωστά στον τύπο `PMONLimitCheck`. Το `auto&` χρησιμοποιείται για να δηλώσει μια αναφορά στο αντικείμενο `PMONLimitCheck`, επιτρέποντας την άμεση τροποποίηση των πεδίων του. Μόλις η μετατροπή ολοκληρωθεί, μπορούμε να τροποποιήσουμε τα πεδία του PMONLimitCheck.

Τελικά γίνεται η ενημέρωση των πεδίων `lowLimit`, `belowLowLimitEvent`, `highLimit` και `aboveHighLimitEvent` με τις νέες τιμές που έχουν διαβαστεί από το μήνυμα όπως φαίνεται για παράδειγμα στην γραμμή `limitCheck.lowLimit = lowLimit;`.

```
case PMON::CheckType::ExpectedValue: {
        PMONBitMask mask = message.read<PMONBitMask>();
        PMONExpectedValue expectedValue =
message.read<PMONExpectedValue>();
        EventDefinitionId unexpectedValueEvent =
message.read<EventDefinitionId>();

        auto& expectedValueCheck =
static_cast<PMONExpectedValueCheck&>(pmon);
        expectedValueCheck.mask = mask;
        expectedValueCheck.expectedValue = expectedValue;
        expectedValueCheck.unexpectedValueEvent = unexpectedValueEvent;
        break;
    }
```

Κώδικας 5.17: Τροποποίηση ορισμών παρακολούθησης τύπου ExpectedValue

Ακριβώς η ίδια λογική ακολουθείται για την τροποποίηση τον ορισμών παρακολούθησης με τύπο ελέγχου `ExpectedValue`, όπως φαίνεται στον παραπάνω κώδικα.

```
case PMON::CheckType::Delta: {
    DeltaThreshold lowDeltaThreshold = message.read<DeltaThreshold>();
    EventDefinitionId belowLowThresholdEventId =
message.read<EventDefinitionId>();
    DeltaThreshold highDeltaThreshold = message.read<DeltaThreshold>();
    EventDefinitionId aboveHighThresholdEventId =
message.read<EventDefinitionId>();
    NumberOfConsecutiveDeltaChecks numberOfConsecutiveDeltaChecks =
message.read<NumberOfConsecutiveDeltaChecks>();

    if (highDeltaThreshold <= lowDeltaThreshold) {
       ErrorHandler::reportError(
           message,
ErrorHandler::ExecutionStartErrorType::HighThresholdIsLowerThanLowThreshol
d);
       continue;
    }

    auto& deltaCheck = static_cast<PMONDeltaCheck&>(pmon);
    deltaCheck.numberOfConsecutiveDeltaChecks =
numberOfConsecutiveDeltaChecks;
    deltaCheck.lowDeltaThreshold = lowDeltaThreshold;
    deltaCheck.belowLowThresholdEvent = belowLowThresholdEventId;
    deltaCheck.highDeltaThreshold = highDeltaThreshold;
    deltaCheck.aboveHighThresholdEvent = aboveHighThresholdEventId;
    break;
}
```

Κώδικας 5.18: Τροποποίηση ορισμών παρακολούθησης τύπου Delta

Παρομοίως για τον τύπο ελέγχου `Delta`. Διαβάζονται τα χαμηλά και υψηλά κατώφλια και τα αντίστοιχα `IDs` των γεγονότων από το μήνυμα. Αν το υψηλό κατώφλι είναι μικρότερο ή ίσο με το χαμηλό, αναφέρεται σφάλμα και η εντολή απορρίπτεται. Το αντικείμενο `PMON` που ανακτήθηκε προηγουμένως γίνεται cast σε `PMONDeltaCheck` και τροποποιούνται τα πεδία του με τις νέες τιμές που διαβάστηκαν από το μήνυμα.

5.8 Μέθοδος αναφοράς ορισμών παρακολούθησης παραμέτρων

Το πρότυπο ECSS στην ενότητα 6.12.3.10 περιγράφει τη διαδικασία αναφοράς ορισμών παρακολούθησης παραμέτρων, εξηγώντας ότι η υπηρεσία παρακολούθησης παραμέτρων πρέπει να παρέχει τη δυνατότητα αναφοράς ορισμών παρακολούθησης παραμέτρων μέσω μηνυμάτων τύπου "TC[12,8] report parameter monitoring definitions". Κάθε αίτημα πρέπει να περιέχει μία ή περισσότερες εντολές για την αναφορά ορισμών παρακολούθησης παραμέτρων ή μία εντολή για την αναφορά όλων των ορισμών παρακολούθησης παραμέτρων. Στην υλοποίηση του κώδικα, αυτά επιτυγχάνονται ως εξής:

```
void
OnBoardMonitoringService::reportParameterMonitoringDefinitions(Message&
message) {
    message.assertTC(ServiceType, ReportParameterMonitoringDefinitions);
    Message pmonDefinitionReport(ServiceType,
MessageType::ParameterMonitoringDefinitionReport, Message::TM,
ApplicationId);
    pmonDefinitionReport.appendUint16(maximumTransitionReportingDelay);
    uint16_t numberOfIds = message.readUint16();
    pmonDefinitionReport.appendUint16(numberOfIds);
```

Κώδικας 5.19: Έλεγχος εγκυρότητας τηλε-εντολής αναφοράς ορισμών παρακολούθησης παραμέτρων

Η μέθοδος `reportParameterMonitoringDefinitions` ξεκινάει ελέγχοντας αν το μήνυμα είναι έγκυρη τηλε-εντολή του τύπου "TC[12,8] report parameter monitoring definitions" για την υπηρεσία `ServiceType` (12) και όπως στις προηγούμενες μεθόδους αν η τηλε-εντολή δεν είναι έγκυρη, η μέθοδος επιστρέφει αμέσως. Στη συνέχεια, δημιουργείται ένα μήνυμα αναφοράς `pmonDefinitionReport` και αποθηκεύεται σ' αυτό η μέγιστη καθυστέρηση αναφοράς μεταβάσεων και αμέσως μετά διαβάζεται και αποθηκεύεται ο αριθμός των ορισμών παρακολούθησης παραμέτρων από το μήνυμα στην σταθερά `numberOfIds`.

Για ακόμα μία φορά ο βρόχος `for` ξεκινάει για επαναλήψεις ίσες με το πλήθος των ορισμών `PMON`, όπου σε κάθε επανάληψη διαβάζεται το αναγνωριστικό του τρέχοντος ορισμού παρακολούθησης από το μήνυμα και αναζητείται στη λίστα `parameterMonitoringList` χρησιμοποιώντας τη συνάρτηση `find` της `etl::map`. Αν ο ορισμός δεν βρεθεί στη λίστα, αναφέρεται σφάλμα μέσω της `ErrorHandler` και η μέθοδος συνεχίζει στην επόμενη επανάληψη, αντίθετα ανακτάται η αναφορά του ορισμού παρακολούθησης χρησιμοποιώντας τη γραμμή `PMON& pmon = it->second.get();`. Ακολούθως, η μέθοδος προσθέτει στο μήνυμα αναφοράς `pmonDefinitionReport` το αναγνωριστικό του τρέχοντος ορισμού παρακολούθησης, το αναγνωριστικό του παρακολουθούμενου ορισμού, την κατάσταση ενεργοποίησης της παρακολούθησης, τον αριθμό επαναλήψεων και τον τύπο ελέγχου.

```
for (uint16_t i = 0; i < numberOfIds; i++) {
    auto currentPMONId = message.read<ParameterId>();

    auto it = parameterMonitoringList.find(currentPMONId);
    if (it == parameterMonitoringList.end()) {
       ErrorHandler::reportError(message,
ErrorHandler::ReportParameterNotInTheParameterMonitoringList);
       continue;
    }

    PMON& pmon = it->second.get();

    pmonDefinitionReport.append<ParameterId>(currentPMONId);
    pmonDefinitionReport.append<ParameterId>(pmon.monitoredParameterId);
    pmonDefinitionReport.appendBoolean(pmon.monitoringEnabled);

pmonDefinitionReport.append<PMONRepetitionNumber>(pmon.repetitionNumber);

    auto checkTypeValue = pmon.checkType;
    pmonDefinitionReport.append<PMON::CheckType>(checkTypeValue);
```

Κώδικας 5.20: Αναζήτηση ορισμών στη λίστα PMON και προσθήκη δεδομένων στο μήνυμα αναφοράς

Ανάλογα με τον τύπο ελέγχου του ορισμού, η μέθοδος αναφέρει τα σχετικά δεδομένα. Για τον τύπο `Limit`, γίνεται `cast` του γενικού αντικειμένου `PMON` σε `PMONLimitCheck` και προστίθενται στο μήνυμα αναφοράς τα χαμηλά και υψηλά όρια, καθώς και τα αντίστοιχα `IDs` των γεγονότων.

```
switch (pmon.checkType) {
    case PMON::CheckType::Limit: {
       auto& limitCheck = static_cast<PMONLimitCheck&>(pmon);
       pmonDefinitionReport.append<PMONLimit>(limitCheck.getLowLimit());

pmonDefinitionReport.append<EventDefinitionId>(limitCheck.getBelowLowLimit
Event());
       pmonDefinitionReport.append<PMONLimit>(limitCheck.getHighLimit());

pmonDefinitionReport.append<EventDefinitionId>(limitCheck.getAboveHighLimi
tEvent());

       break;
    }
```

Κώδικας 5.21: Αναφορά ορισμών παρακολούθησης τύπου Limit

Ακολουθείται η ίδια λογική για την αναφορά των ορισμών παρακολούθησης με τύπο ελέγχου `ExpectedValue`, όπως φαίνεται στον παρακάτω κώδικα.

```
case PMON::CheckType::ExpectedValue: {
    auto& expectedValueCheck = static_cast<PMONExpectedValueCheck&>(pmon);

pmonDefinitionReport.append<PMONBitMask>(expectedValueCheck.getMask());

pmonDefinitionReport.append<PMONExpectedValue>(expectedValueCheck.getExpec
tedValue());

pmonDefinitionReport.append<EventDefinitionId>(expectedValueCheck.getUnexp
ectedValueEvent());

    break;
}
```

Κώδικας 5.22: Αναφορά ορισμών παρακολούθησης τύπου ExpectedValue

Αντίστοιχα για τον τύπο ελέγχου `Delta`, γίνεται `cast` του γενικού αντικειμένου `PMON` σε `PMONDeltaCheck` και προστίθενται στο μήνυμα αναφοράς τα χαμηλά και υψηλά όρια διαφοράς, καθώς και τα αντίστοιχα `IDs` των γεγονότων και ο αριθμός των συνεχόμενων ελέγχων διαφοράς.

```
case PMON::CheckType::Delta: {
            auto& deltaCheck = static_cast<PMONDeltaCheck&>(pmon);

pmonDefinitionReport.append<DeltaThreshold>(deltaCheck.getLowDeltaThreshol
d());

pmonDefinitionReport.append<EventDefinitionId>(deltaCheck.getBelowLowThres
holdEvent());

pmonDefinitionReport.append<DeltaThreshold>(deltaCheck.getHighDeltaThresho
ld());

pmonDefinitionReport.append<EventDefinitionId>(deltaCheck.getAboveHighThre
sholdEvent());

pmonDefinitionReport.append<NumberOfConsecutiveDeltaChecks>(deltaCheck.get
NumberOfConsecutiveDeltaChecks());

            break;
        }
    }
    storeMessage(pmonDefinitionReport);
}
```

Κώδικας 5.23: Αναφορά ορισμών παρακολούθησης τύπου Delta

Τελικά, το μήνυμα αναφοράς `pmonDefinitionReport` αποθηκεύεται χρησιμοποιώντας τη συνάρτηση `storeMessage` της κλάσης `Service`, ολοκληρώνοντας τη διαδικασία αναφοράς σύμφωνα με τις απαιτήσεις του προτύπου ECSS.

5.9 Μέθοδος execute

Η μέθοδος `execute` της κλάσης `OnBoardMonitoringService` είναι υπεύθυνη για την εκτέλεση των κατάλληλων ενεργειών βάσει του τύπου μηνύματος που λαμβάνεται. Η λειτουργία της βασίζεται στη χρήση μιας δήλωσης `switch`, η οποία επιτρέπει την εκτέλεση διαφορετικών τμημάτων κώδικα ανάλογα με την τιμή της μεταβλητής `messageType` που περιέχεται στο μήνυμα.

Σε περίπτωση που το μήνυμα δεν ταιριάζει με κανέναν από τους προκαθορισμένους τύπους, η μέθοδος καλεί την `ErrorHandler` για να αναφέρει ένα εσωτερικό σφάλμα, διασφαλίζοντας ότι όλες οι πιθανές τιμές του `messageType` καλύπτονται και ότι οποιαδήποτε απροσδόκητη τιμή αντιμετωπίζεται κατάλληλα.

Η αρχιτεκτονική αυτή επιτρέπει την εύκολη διαχείριση και επέκταση της λογικής χειρισμού των μηνυμάτων στην κλάση `OnBoardMonitoringService`, καθιστώντας την ευέλικτη και εύκολη στη συντήρηση. Για την προσθήκη νέων τύπων μηνυμάτων, απαιτείται απλώς η προσθήκη μιας νέας περίπτωσης στη δήλωση `switch` και η υλοποίηση της αντίστοιχης μεθόδου χειρισμού.

```
void OnBoardMonitoringService::execute(Message& message) {
    switch (message.messageType) {
       case EnableParameterMonitoringDefinitions:
          enableParameterMonitoringDefinitions(message);
          break;

                              .
                              .
                              .

       case ReportParameterMonitoringDefinitions:
          reportParameterMonitoringDefinitions(message);
          break;
       default:

ErrorHandler::reportInternalError(ErrorHandler::OtherMessageType);
    }
}
```

Κώδικας 5.24: Μέθοδος execute της κλάσης OnBoardMonitoringService

5.10 Unit Tests για την OnBoardMonitoringService

5.10.1 Λογική των Unit Tests

Για την δοκιμή των μεθόδων της κλάσης `OnBoardMonitoringService`, υλοποιήθηκαν `unit tests` χρησιμοποιώντας τη βιβλιοθήκη `Catch2` στο αρχείο OnBoardMonitoringServiceTests.cpp` που ανήκει στον υποφάκελο `test/Services` του πρότζεκτ. Η `Catch2` είναι μια μοντέρνα βιβλιοθήκη C++ συγγραφής `unit tests` και προσφέρει μια σειρά από μακροεντολές για τον ορισμό και την εκτέλεση των δοκιμών. Τα βασικά στοιχεία που χρησιμοποιούνται στα tests περιλαμβάνουν τις μακροεντολές `TEST_CASE`, `SECTION` και `CHECK`.

Η λογική που ακολουθήθηκε για τη δημιουργία των δοκιμών ήταν να καλυφθούν όλα τα πιθανά σενάρια χρήσης των μεθόδων της κλάσης `OnBoardMonitoringService`, περιλαμβάνοντας τόσο έγκυρα όσο και μη έγκυρα σενάρια για κάθε μέθοδο.

- **`TEST_CASE`**: Χρησιμοποιείται για να ορίσει μια ομάδα δοκιμών που σχετίζονται με μια συγκεκριμένη λειτουργία ή μέθοδο. Κάθε `TEST_CASE` μπορεί να περιέχει πολλά `SECTION`.
- **`SECTION`**: Χρησιμοποιείται για να χωρίσει ένα `TEST_CASE` σε διακριτά σενάρια ή περιπτώσεις δοκιμής, με κάθε `SECTION` να εκτελείται ανεξάρτητα.
- **`CHECK`**: Χρησιμοποιείται για να ελέγξει αν μια συνθήκη είναι αληθής. Αν η συνθήκη δεν είναι αληθής, καταγράφεται σφάλμα αλλά η εκτέλεση της δοκιμής συνεχίζεται.

5.10.2 Αρχικοποίηση δοκιμαστικών δεδομένων

Για την απλοποίηση των δοκιμών, δημιουργήθηκε μια δομή `Fixtures` που περιλαμβάνει προεπιλεγμένους ορισμούς παρακολούθησης παραμέτρων. Η δομή αυτή βοηθά στη γρήγορη και εύκολη αρχικοποίηση των δεδομένων που απαιτούνται για τα tests.

```
struct Fixtures {
    PMONExpectedValueCheck monitoringDefinition1 = PMONExpectedValueCheck(monitoredParameterId:8, repetitionNumber:5,
        expectedValue:10, mask:8, unexpectedValueEvent:0);
    PMONLimitCheck monitoringDefinition2 = PMONLimitCheck(monitoredParameterId:6, repetitionNumber:5,
        lowLimit:2, belowLowLimitEvent:1, highLimit:9, aboveHighLimitEvent:2);
    PMONDeltaCheck monitoringDefinition3 = PMONDeltaCheck(monitoredParameterId:9, repetitionNumber:5,
        numberOfConsecutiveDeltaChecks:5, lowDeltaThreshold:3, belowLowThresholdEvent:3, highDeltaThreshold:11, aboveHighThresholdEvent:4);
    PMONDeltaCheck monitoringDefinition4 = PMONDeltaCheck(monitoredParameterId:7, repetitionNumber:5,
        numberOfConsecutiveDeltaChecks:5, lowDeltaThreshold:3, belowLowThresholdEvent:3, highDeltaThreshold:11, aboveHighThresholdEvent:4);
```

Κώδικας 5.25: Δομή δοκιμαστικών δεδομένων Fixtures

Ειδικότερα, η παραπάνω δομή περιλαμβάνει τέσσερις ορισμούς παρακολούθησης παραμέτρων `monitoringDefinition1` με τύπο ελέγχου `ExpectedValue`, `monitoringDefinition2` με τύπο ελέγχου `Limit`, `monitoringDefinition3` και `monitoringDefinition4` με τύπο `Delta`. Επιπλέον, στον `constructor` της `Fixtures` αρχικοποιείται η ιδιότητα `monitoringEnabled` του πρώτου ορισμού σε `true`, προετοιμάζοντας το για τις δοκιμές.

Στη συνέχεια, η συνάρτηση `initialiseParameterMonitoringDefinitions` χρησιμοποιείται για την αρχικοποίηση των δεδομένων της δομής `Fixtures`, καθώς προσθέτει τους ορισμούς παρακολούθησης στην υπηρεσία `onBoardMonitoringService` χρησιμοποιώντας τις μεθόδους `addPMONExpectedValueCheck`, `addPMONLimitCheck` και `addPMONDeltaCheck`. Έτσι, οι δοκιμές ξεκινούν πάντα με τα ίδια δεδομένα, διασφαλίζοντας την επαναληψιμότητα και την αξιοπιστία των αποτελεσμάτων.

```
void initialiseParameterMonitoringDefinitions() {
    // Reset fixtures to the defaults set up by the
constructor
    new (&fixtures) Fixtures();

    onBoardMonitoringService.addPMONExpectedValueCheck(0,
etl::ref(fixtures.monitoringDefinition1));
    onBoardMonitoringService.addPMONLimitCheck(1,
etl::ref(fixtures.monitoringDefinition2));
    onBoardMonitoringService.addPMONDeltaCheck(2,
etl::ref(fixtures.monitoringDefinition3));
    onBoardMonitoringService.addPMONDeltaCheck(3,
etl::ref(fixtures.monitoringDefinition4));
}
```

Κώδικας 5.25: Συνάρτηση initialiseParameterMonitoringDefinitions

5.10.3 Ανασκόπηση των Unit Tests

Παρακάτω ακολουθεί μια σύντομη περιγραφή κάποιων δοκιμαστικών σεναρίων για κάθε βασική μέθοδο της υπηρεσίας `OnBoardMonitoringService`. Οι δοκιμές έχουν σχεδιαστεί ώστε να καλύπτουν όλες τις πιθανές περιπτώσεις χρήσης, περιλαμβάνοντας τόσο σενάρια ορθής λειτουργίας όσο και σενάρια με σφάλματα για κάθε μέθοδο. Το σύνολο των σεναρίων για κάθε μέθοδο είναι διαθέσιμο στο αρχείο `test/Service/OnBoardMonitoringServiceTests.cpp` του πρότζεκτ, το οποίο βρίσκεται στον σύνδεσμο που αναφέρεται στον Πηγαίο Κώδικα.

```
TEST_CASE("Enable Parameter Monitoring Definitions") {
    SECTION("3 valid requests to enable Parameter Monitoring Definitions")
{
       initialiseParameterMonitoringDefinitions();

       Message request =
           Message(OnBoardMonitoringService::ServiceType,

OnBoardMonitoringService::MessageType::EnableParameterMonitoringDefinition
s, Message::TC, 0);
       uint16_t numberOfIds = 3;
       request.appendUint16(numberOfIds);
       etl::array<ParameterId, 3> PMONIds = {0, 1, 2};
       request.append<ParameterId>(PMONIds[0]);
       request.append<ParameterId>(PMONIds[1]);
       request.append<ParameterId>(PMONIds[2]);

       MessageParser::execute(request);
       CHECK(ServiceTests::count() == 0);


CHECK((onBoardMonitoringService.getPMONDefinition(PMONIds[0]).get().monito
ringEnabled == true));

CHECK((onBoardMonitoringService.getPMONDefinition(PMONIds[1]).get().monito
ringEnabled == true));

CHECK((onBoardMonitoringService.getPMONDefinition(PMONIds[2]).get().monito
ringEnabled == true));

CHECK(onBoardMonitoringService.getPMONDefinition(PMONIds[0]).get().repetit
ionCounter == 0);

CHECK(onBoardMonitoringService.getPMONDefinition(PMONIds[1]).get().repetit
ionCounter == 0);

CHECK(onBoardMonitoringService.getPMONDefinition(PMONIds[2]).get().repetit
ionCounter == 0);

       ServiceTests::reset();
       Services.reset();
    }
```

Κώδικας 5.26: Σενάριο επιτυχημένης τριών έγκυρων αιτημάτων για ενεργοποίηση παρακολούθησης ορισμών

Το παραπάνω `TEST_CASE` περιγράφει τις δοκιμές για την ενεργοποίηση των ορισμών παρακολούθησης παραμέτρων. Στην ενότητα SECTION("3 valid requests to enable Parameter

Monitoring Definitions"), αρχικοποιούνται οι προεπιλεγμένοι ορισμοί παρακολούθησης παραμέτρων με την κλήση της συνάρτησης `initialiseParameterMonitoringDefinitions` και μετά δημιουργείται ένα αντικείμενο `Message` για να προσομοιωθεί ένα αίτημα ενεργοποίησης ορισμών παρακολούθησης παραμέτρων.

Η κατασκευή του `Message` γίνεται με την κλήση του `constructor` του, όπου ορίζεται ο τύπος της υπηρεσίας (ServiceType), ο τύπος του μηνύματος (EnableParameterMonitoringDefinitions), η κατηγορία του μηνύματος (Message::TC), και το αναγνωριστικό της εφαρμογής (0). Αυτός ο τρόπος κατασκευής διασφαλίζει ότι το μήνυμα έχει τα σωστά μεταδεδομένα για την εκτέλεση της αντίστοιχης εντολής.

Ακολούθως, το μήνυμα συμπληρώνεται με τα απαραίτητα δεδομένα χρησιμοποιώντας τη μέθοδο `request.append`, η οποία προσθέτει τα δεδομένα στο μήνυμα. Η μέθοδος `request.appendUint16(numberOfIds)` προσθέτει τον αριθμό των ορισμών που πρόκειται να ενεργοποιηθούν, και η `request.append<ParameterId>(PMONIds[0])` προσθέτει τα αναγνωριστικά των ορισμών αυτών.

Η εκτέλεση του μηνύματος γίνεται με την κλήση της `MessageParser::execute(request)`, και ακολουθούν έλεγχοι με τις μακροεντολές CHECK για να διασφαλιστεί ότι οι ορισμοί έχουν ενεργοποιηθεί σωστά και ότι οι μετρητές έχουν επαναφερθεί. Σε αυτό αλλά και σε κάθε `SECTION` γίνεται επαναφορά της αρχικής κατάστασης.

```
TEST_CASE("Change Maximum Transition Reporting Delay") {
    initialiseParameterMonitoringDefinitions();
    Message request =
        Message(OnBoardMonitoringService::ServiceType,

OnBoardMonitoringService::MessageType::ChangeMaximumTransitionReportingDel
ay, Message::TC, 0);
    uint16_t newMaximumTransitionReportingDelay = 10;
    request.appendUint16(newMaximumTransitionReportingDelay);
    MessageParser::execute(request);
    CHECK(ServiceTests::count() == 0);
    CHECK(onBoardMonitoringService.maximumTransitionReportingDelay ==
newMaximumTransitionReportingDelay);

    ServiceTests::reset();
    Services.reset();
}
```

Κώδικας 5.27: Σενάριο αιτήματος αλλαγής μέγιστης καθυστέρησης αναφοράς μεταβάσεων

Σε αυτή τη δοκιμή, αρχικοποιούνται οι ορισμοί παρακολούθησης παραμέτρων και δημιουργείται ένα αίτημα για την αλλαγή της μέγιστης καθυστέρησης αναφοράς μεταβάσεων. Το μήνυμα συμπληρώνεται με τη νέα τιμή της καθυστέρησης και εκτελείται με τη χρήση της `MessageParser::execute(request)` διασφαλίζοντας ότι η αλλαγή έχει εφαρμοστεί σωστά.

```
TEST_CASE("Delete all Parameter Monitoring Definitions") {
    SECTION("Valid request to delete all Parameter Monitoring Definitions") {
       initialiseParameterMonitoringDefinitions();
       onBoardMonitoringService.parameterMonitoringFunctionStatus = false;
       Message request =
           Message(OnBoardMonitoringService::ServiceType,

OnBoardMonitoringService::MessageType::DeleteAllParameterMonitoringDefinitions,
Message::TC, 0);
       MessageParser::execute(request);
       CHECK(ServiceTests::count() == 0);
       CHECK(onBoardMonitoringService.isPMONListEmpty());

       ServiceTests::reset();
       Services.reset();
    }
```

Κώδικας 5.28: Σενάριο έγκυρου αιτήματος για διαγραφή όλων των ορισμών παρακολούθησης

Αυτό το σενάριο ελέγχει τη διαγραφή όλων των ορισμών παρακολούθησης παραμέτρων. Αφού αρχικοποιηθούν οι ορισμοί, δημιουργείται και εκτελείται ένα αίτημα για τη διαγραφή όλων των ορισμών και τελικά οι έλεγχοι διασφαλίζουν ότι η λίστα `PMON `είναι κενή μετά την εκτέλεση της εντολής με την χρήση της μεθόδου `isPMONListEmpty`.

```
SECTION("Add Parameter Monitoring Definition with a non-existing
parameter") {
    uint16_t numberOfIds = 1;
    PMONRepetitionNumber repetitionNumber = 5;

    Message request =
        Message(OnBoardMonitoringService::ServiceType,

OnBoardMonitoringService::MessageType::AddParameterMonitoringDefinitions,
Message::TC, 0);
    request.appendUint16(numberOfIds);
    request.append<PMONRepetitionNumber>(repetitionNumber);
    request.append<PMON::CheckType>(PMON::CheckType::ExpectedValue);

    MessageParser::execute(request);
    CHECK(ServiceTests::count() == 1);

CHECK(ServiceTests::countThrownErrors(ErrorHandler::GetNonExistingParamete
rMonitoringDefinition) == 1);
    ServiceTests::reset();
    Services.reset();
}
```

Κώδικας 5.29: Σενάριο προσθήκης ορισμού παρακολούθησης με απουσία παρακολουθούμενης παραμέτρου

Στο παραπάνω σενάριο, δοκιμάζεται η προσθήκη ενός νέου ορισμού παρακολούθησης παραμέτρων που αναφέρεται σε μη υπάρχουσα παράμετρο. Η δημιουργία και εκτέλεση του μηνύματος γίνεται όπως και στα προηγούμενα σενάρια χρησιμοποιώντας τη `request.append` η οποία προσθέτει τις απαραίτητες τιμές στο μήνυμα, όπως τον αριθμό των ορισμών και τις σχετικές παραμέτρους, με την παρακολουθούμενη παράμετρο όμως να λείπει από το μήνυμα. Μετά την εκτέλεση της εντολής, οι έλεγχοι `CHECK` επιβεβαιώνουν ότι έχει αναφερθεί ένα σφάλμα μέσω της `ErrorHandler` για την απόπειρα προσθήκης ενός ορισμού με μη υπάρχουσα παράμετρο.

```
SECTION("Invalid request to report Parameter Monitoring Definitions") {
    initialiseParameterMonitoringDefinitions();
    uint16_t numberOfIds = 1;
    uint16_t PMONId = 5;
    Message request =
        Message(OnBoardMonitoringService::ServiceType,

OnBoardMonitoringService::MessageType::ReportParameterMonitoringDefinition
s, Message::TC, 0);
    request.appendUint16(numberOfIds);
    request.appendEnum16(PMONId);
    MessageParser::execute(request);
    CHECK(ServiceTests::count() == 2);

CHECK(ServiceTests::countThrownErrors(ErrorHandler::ReportParameterNotInTh
eParameterMonitoringList) == 1);
    ServiceTests::reset();
    Services.reset();
}
```

Κώδικας 5.30: Σενάριο μη έγκυρου αιτήματος αναφοράς ορισμού παραμέτρου

Σε αυτή την περίπτωση δοκιμάζεται η αναφορά ενός ορισμού παρακολούθησης παραμέτρων που δεν υπάρχει στη λίστα. Αρχικά, αρχικοποιούνται οι ορισμοί παρακολούθησης και δημιουργείται ένα μήνυμα με το αναγνωριστικό ενός μη υπάρχοντος ορισμού στην δομή `Fixtures`. Η `request.appendUint16(numberOfIds)` προσθέτει τον αριθμό των ορισμών που πρόκειται να αναφερθούν, και η `request.appendEnum16(PMONId)` προσθέτει το αναγνωριστικό του ορισμού. Μετά την εκτέλεση της εντολής, οι έλεγχοι `CHECK` επιβεβαιώνουν ότι αναφέρθηκαν δύο σφάλματα, εκ των οποίων το ένα αφορά στην απόπειρα αναφοράς ενός μη υπάρχοντος ορισμού μέσω της ErrorHandler.

6 Υλοποίηση λογικής ελέγχων

Σε αυτό το κεφάλαιο αναλύεται η ενσωμάτωση της λογικής των τύπων ελέγχων που περιεγράφηκαν στο υποκεφάλαιο 4.1.1 στον κώδικα που έχει ήδη σχολιαστεί. Η υλοποίηση αυτή αποτελεί μία αρχική προσπάθεια δημιουργίας των διαδικασιών που εκτελούνται κατά τη διάρκεια των ελέγχων. Στόχος είναι να παρέχει μία προσαρμόσιμη βάση που θα διευκολύνει τη διεξαγωγή δοκιμών σε επίπεδο λογισμικού, αλλά και σε περιβάλλον μικροελεγκτή.

6.1 Μέθοδος performCheck

Αρχικά, η κλάση `PMON` απέκτησε τη μέθοδο `performCheck`, μια καθαρά εικονική μέθοδο, η οποία επιτρέπει στις υποκλάσεις να ορίζουν τη δική τους λογική ελέγχου ανάλογα με τον τύπο τους, ακολουθώντας τις προδιαγραφές του προτύπου ECSS.

```
virtual void performCheck() = 0;
```

Κώδικας 6.1: Εικονική μέθοδος performCheck

6.1.1 Έλεγχος Ορίων

Στην υλοποίηση της μεθόδου `performCheck` της υποκλάσης `PMONLimitCheck`, πρώτα ανακτάται η τρέχουσα κατάσταση ελέγχου και αποθηκεύεται σε μια τοπική μεταβλητή `previousStatus`. Στη συνέχεια, η τρέχουσα τιμή της παραμέτρου λαμβάνεται ως `double` και συγκρίνεται με τα καθορισμένα όρια. Αν η τιμή είναι μικρότερη από το χαμηλό όριο, η κατάσταση ελέγχου ορίζεται σε `BelowLowLimit`, ενώ εάν η τιμή είναι μεγαλύτερη από το υψηλό όριο, η κατάσταση ελέγχου ορίζεται σε `AboveHighLimit`. Στο σενάριο που η τιμή είναι εντός των ορίων, η κατάσταση ελέγχου ορίζεται σε `WithinLimits`. Μετά την ολοκλήρωση του ελέγχου, η τρέχουσα κατάσταση ελέγχου συγκρίνεται με την προηγούμενη, όπου αν έχει παραμείνει ίδια, ο μετρητής επαναλήψεων αυξάνεται, αλλιώς επαναφέρεται στη τιμή 1.

```
void performCheck() override {
    auto previousStatus = checkingStatus;
    auto currentValue = monitoredParameter.get().getValueAsDouble();
    if (currentValue < getLowLimit()) {
       checkingStatus = BelowLowLimit;
    } else if (currentValue > getHighLimit()) {
       checkingStatus = AboveHighLimit;
    } else {
       checkingStatus = WithinLimits;
    }

    if (checkingStatus == previousStatus) {
       repetitionCounter++;
    } else {
       repetitionCounter = 1;
    }
}
```

Κώδικας 6.2: Μέθοδος performCheck για τον έλεγχο οριών

6.1.2 Έλεγχος Αναμενόμενης Τιμής

Για την `performCheck` της υποκλάσης `PMONExpectedValueCheck`, αρχικά ανακτάται η τρέχουσα κατάσταση ελέγχου και αποθηκεύεται σε μια τοπική μεταβλητή `previousStatus` και η τρέχουσα τιμή της παραμέτρου λαμβάνεται και μετατρέπεται σε `uint64_t`. Ακολούθως, η τρέχουσα τιμή υφίσταται τη λογική πράξη AND με την προκαθορισμένη μάσκα bit, η οποία διατηρεί μόνο τα bit που είναι ενεργοποιημένα και στις δύο τιμές, με το αποτέλεσμα να συγκρίνεται με την προκαθορισμένη αναμενόμενη τιμή. Στην περίπτωση που η νέα τιμή είναι ίση με την αναμενόμενη η κατάσταση ελέγχου ορίζεται σε `ExpectedValue`, αντίθετα σε `UnexpectedValue`. Τελικά, η τρέχουσα κατάσταση ελέγχου συγκρίνεται με την προηγούμενη, όπου αν είναι ίδια, ο μετρητής επαναλήψεων αυξάνεται, αλλιώς επαναφέρεται στη τιμή 1.

```
void performCheck() override {
    auto previousStatus = checkingStatus;
    auto currentValueAsUint64 =
monitoredParameter.get().getValueAsUint64();
    uint64_t maskedValue = currentValueAsUint64 & getMask();

    if (maskedValue == getExpectedValue()) {
       checkingStatus = ExpectedValue;
    } else {
       checkingStatus = UnexpectedValue;
    }

    if (checkingStatus == previousStatus) {
       repetitionCounter++;
    } else {
       repetitionCounter = 1;
    }
}
```

Κώδικας 6.3: Μέθοδος performCheck για τον έλεγχο αναμενόμενης τιμής

6.1.3 Έλεγχος Δέλτα

Η μέθοδος `performCheck` υλοποιεί τον έλεγχο δέλτα στην υποκλάση `PMONDeltaCheck`, υπολογίζοντας τον ρυθμό μεταβολής της διαφοράς μεταξύ δύο διαδοχικών τιμών για την αντίστοιχη χρονική τους διαφορά. Για την υλοποίηση αυτής της λογικής χρειάστηκαν και άλλες βοηθητικές μέθοδοι, όπως η `updatePreviousValueAndTimestamp`, η οποία ενημερώνει την προηγούμενη τιμή και το χρονικό στίγμα με τις τρέχουσες τιμές, διευκολύνοντας την ορθή παρακολούθηση των μεταβολών των τιμών των παραμέτρων, `getDeltaPerSecond` που υπολογίζει τη διαφορά ανά δευτερόλεπτο μεταξύ της τρέχουσας και της προηγούμενης τιμής και τέλος η μέθοδος `hasOldValue` που ελέγχει την ύπαρξη προηγούμενης τιμής, επιτρέποντας την ορθή εκτέλεση του ελέγχου δέλτα.

```
void updatePreviousValueAndTimestamp(double newValue, const
Time::DefaultCUC& newTimestamp) {
    previousValue = newValue;
    previousTimestamp = newTimestamp;
}
```

Κώδικας 6.4: Μέθοδος updatePreviousValueAndTimestamp

```
private:
    double previousValue;
    etl::optional<Time::DefaultCUC>
previousTimestamp;

. . .

bool hasOldValue() const {
    return previousTimestamp.has_value();
}
```

Κώδικας 6.5: Ιδιωτικές μεταβλητές και μέθοδος hasOldValue

Αξίζει να σημειωθεί ότι η μέθοδος `hasOldValue` αποτελεί ένα δημόσιο μέσο πρόσβασης στην ιδιωτική μεταβλητή `previousTimestamp`, διασφαλίζοντας την ενθυλάκωση και την ασφάλεια των δεδομένων της κλάσης.

```
double getDeltaPerSecond(double currentValue) const {
    if (previousTimestamp.has_value()) {
       double delta = currentValue - previousValue;
       auto duration = TimeGetter::getCurrentTimeDefaultCUC() -
*previousTimestamp;
       double deltaTime = std::chrono::duration<double>(duration).count();

       if (deltaTime == 0) {
          return 0;
       }
       return delta / deltaTime;
    }
    return 0;
}
```

Κώδικας 6.5: Μέθοδος getDeltaPerSecond

Η μέθοδος `getDeltaPerSecond` αρχικά ελέγχει αν υπάρχει διαθέσιμη προηγούμενη χρονική τιμή με την βοήθεια της `has_value` και αν δεν υπάρχει επιστρέφει τιμή 0 για την αποφυγή εμφάνισης μηδενικού στον παρανομαστή της διαίρεσης. Αν όμως βρεθεί προηγούμενη τιμή υπολογίζει τη διαφορά τιμής (delta) μεταξύ της τρέχουσας και της προηγούμενης μέτρησης, και τη χρονική διαφορά (deltaTime) σε δευτερόλεπτα. Ειδικότερα, η χρονική διαφορά (duration) υπολογίζεται ως η διαφορά μεταξύ της τρέχουσας χρονικής στιγμής λαμβανόμενη από τη συνάρτηση `getCurrentTimeDefaultCUC` της κλάσης `TimeGetter` και της προηγούμενης χρονικής στιγμής.

Αυτή η διαφορά μετατρέπεται σε δευτερόλεπτα χρησιμοποιώντας τη συνάρτηση `std::chrono::duration<double>(duration).count()`

Εν τέλει, η `performCheck` της `PMONDeltaCheck` ξεκινάει ανακτώντας και αποθηκεύοντας την κατάσταση ελέγχου, την τρέχουσα τιμή και τη χρονική στιγμή της μέτρησης στις κατάλληλες τοπικές μεταβλητές. Χρησιμοποιεί την βοηθητική μέθοδο `hasOldValue` για να ελέγξει αν υπάρχει διαθέσιμη η προηγούμενη τιμή ορίζοντας την κατάσταση ελέγχου σε `Invalid` στην αρνητική περίπτωση. Στην θετική, καλεί την `getDeltaPerSecond` για την τρέχουσα τιμή και βάσει αποτελέσματος ορίζει την κατάσταση σε `BelowLowThreshold` αν η επιστρεφόμενη τιμή είναι μικρότερη από το χαμηλό κατώφλι, ή σε `AboveHighThreshold` αν η διαφορά είναι μεγαλύτερη από το υψηλό κατώφλι. Προφανώς, εάν η τιμή είναι εντός των ορίων, η κατάσταση ελέγχου ορίζεται σε `WithinThreshold`. Τέλος, ενημερώνονται οι προηγούμενες τιμές με τη βοήθεια της ` updatePreviousValueAndTimestamp` και η τρέχουσα κατάσταση ελέγχου συγκρίνεται με την προηγούμενη. Αν είναι ίδια, ο μετρητής επαναλήψεων αυξάνεται, αλλιώς επαναφέρεται στην τιμή 1.

```
void performCheck() override {
    auto previousStatus = checkingStatus;
    auto currentValue = monitoredParameter.get().getValueAsDouble();
    auto currentTimestamp = TimeGetter::getCurrentTimeDefaultCUC();

    if (hasOldValue()) {
       double deltaPerSecond = getDeltaPerSecond(currentValue);
       if (deltaPerSecond < getLowDeltaThreshold()) {
          checkingStatus = BelowLowThreshold;
       } else if (deltaPerSecond > getHighDeltaThreshold()) {
          checkingStatus = AboveHighThreshold;
       } else {
          checkingStatus = WithinThreshold;
       }
    } else {
       checkingStatus = Invalid;
    }

    updatePreviousValueAndTimestamp(currentValue, currentTimestamp);

    if (checkingStatus == previousStatus) {
       repetitionCounter++;
    } else {
       repetitionCounter = 1;
    }
}
```

Κώδικας 6.6: Μέθοδος performCheck για τον έλεγχο δέλτα

6.2 Μέθοδος checkAll

Στην κλάση `OnBoardMonitoringService` έγινε η προσθήκη της μεθόδου `checkAll`, η οποία είναι υπεύθυνη για τον έλεγχο όλων των αντικειμένων `PMON` που υπάρχουν στη λίστα παρακολούθησης παραμέτρων, διασφαλίζοντας ότι εκτελούνται οι κατάλληλοι έλεγχοι για όλες τις ενεργοποιημένες παραμέτρους, ακολουθώντας τη λογική που έχει οριστεί στις επιμέρους μεθόδους `performCheck` των υποκλάσεων της `PMON`.

Η δήλωση της μεθόδου έγινε στο αρχείο `OnBoardMonitoringService.hpp` και η υλοποίηση της στο αντίστοιχο `OnBoardMonitoringService.cpp` ακολουθώντας τα εξής βήματα:

- Χρησιμοποιεί έναν βρόχο `for` για να διατρέξει κάθε στοιχείο στη λίστα `parameterMonitoringList`.
- Για κάθε στοιχείο της λίστας, ελέγχει αν η παράμετρος είναι ενεργοποιημένη (isMonitoringEnabled).
- Εάν η παράμετρος είναι ενεργοποιημένη, καλείται η μέθοδος performCheck της αντίστοιχης υποκλάσης της PMON.

```
void OnBoardMonitoringService::checkAll() const {
    for (const auto& entry : parameterMonitoringList) {
        auto& pmon = entry.second.get();
        if (pmon.isMonitoringEnabled()) {
            pmon.performCheck();
        }
    }
}
```

Κώδικας 6.7: Μέθοδος checkAll

Για να οριστεί η συχνότητα κλήσης της μεθόδου checkAll, προστέθηκε μια σταθερά στο αρχείο `ECSS_Definitions.hpp`, η `ECSSMonitoringFrequency`. Η προεπιλεγμένη τιμή έχει οριστεί στα 60 δευτερόλεπτα, αλλά μπορεί να τροποποιηθεί αν χρειαστεί. Αυτή η προσθήκη εξασφαλίζει ότι οι έλεγχοι παραμέτρων εκτελούνται σε τακτά χρονικά διαστήματα, προσφέροντας συνεχή παρακολούθηση της κατάστασης των παραμέτρων του συστήματος.

```
inline constexpr std::chrono::seconds ECSSMonitoringFrequency(60);
```

Κώδικας 6.7: Σταθερά συχνότητας κλήσης της μεθόδου checkAll

6.3 Unit Tests για την λογική των ελέγχων

Οι δοκιμές για την επαλήθευση της σωστής λειτουργίας των συναρτήσεων `performCheck` ανά τύπο ελέγχου και της `checkAll` υλοποιήθηκαν στο ίδιο αρχείο με τις δοκιμές που περιεγράφηκαν στο υποκεφάλαιο 5.10, το `OnBoardMonitoringServiceTests.cpp`, ακολουθώντας την ίδια μεθοδολογία. Όπως και προηγουμένως, έγινε χρήση της δομής `Fixtures` και της συνάρτησης `initialiseParameterMonitoringDefinitions` για την αρχικοποίηση δοκιμαστικών δεδομένων.

Τα σενάρια εξέτασης ανά τύπο ελέγχου αξιολόγησαν:

- Την ορθή ενημέρωση του `checkingStatus` των ορισμών,
- Την σωστή ενημέρωση του `repetionCounter`,
- Τη γενικότερη συμπεριφορά των ορισμών παρακολούθησης όταν υποβάλλονται σε διαδοχικούς ελέγχους.

Επιπλέον, εξετάστηκε η ορθή λειτουργία της μεθόδου `checkAll`, εστιάζοντας στον έλεγχο μόνο των ορισμών των οποίων η μεταβλητή `monitoringEnabled` είχε την τιμή `true`. Το σύνολο των δοκιμών μπορεί να βρεθεί στον Πηγαίο Κώδικα. Παρακάτω θα αναλυθούν κάποιες αντιπροσωπευτικές περιπτώσεις από αυτά τα σενάρια.

```
TEST_CASE("Limit Check Behavior") {
    SECTION("Value within limits") {
        initialiseParameterMonitoringDefinitions();
        auto& pmon = fixtures.monitoringDefinition2;
        auto& param = static_cast<Parameter<unsigned
char>&>(pmon.monitoredParameter.get());

        param.setValue(5);
        pmon.performCheck();
        CHECK(pmon.getCheckingStatus() == PMON::WithinLimits);
        CHECK(pmon.getRepetitionCounter() == 1);

        pmon.performCheck();
        CHECK(pmon.getRepetitionCounter() == 2);

        ServiceTests::reset();
        Services.reset();
    }
```

Κώδικας 6.8: Σενάριο Δοκιμής για τον Έλεγχο Ορίων - Τιμή Εντός Ορίων

Στο παραπάνω σενάριο δοκιμής, γίνεται κλήση της `initialiseParameterMonitoringDefinitions` για την αρχικοποίηση των δεδομένων και επιλέγεται η παράμετρος `monitoringDefinition2` από τη δομή `Fixtures`. Στη συνέχεια, γίνεται μετατροπή της αναφοράς του αντικειμένου `ParameterBase` στην εξειδικευμένη κλάση Parameter<unsigned char>, επιτρέποντας την άμεση πρόσβαση και τροποποίηση των δεδομένων της παραμέτρου. Η παράμετρος ορίζεται στην εντός ορίων τιμή 5, με τη βοήθεια της συνάρτησης `setValue`. Η `performCheck` καλείται και ελέγχεται αν η κατάσταση ελέγχου (checkingStatus) είναι `WithinLimits` και αν ο μετρητής επαναλήψεων (repetitionCounter) έχει αυξηθεί σωστά. Η διαδικασία επαναλαμβάνεται για να διασφαλιστεί ότι η κατάσταση και ο μετρητής επαναλήψεων ενημερώνονται σωστά κατά τη διάρκεια διαδοχικών ελέγχων.

```
TEST_CASE("Expected Value Check Behavior") {
    SECTION("Value matches expected value") {
       initialiseParameterMonitoringDefinitions();
       auto& pmon = fixtures.monitoringDefinition1;
       auto& param = static_cast<Parameter<unsigned
char>&>(pmon.monitoredParameter.get());

       param.setValue(10);
       pmon.mask = 0xFF;

       pmon.performCheck();
       CHECK(pmon.getCheckingStatus() ==
PMON::ExpectedValue);
       CHECK(pmon.getRepetitionCounter() == 1);

       pmon.performCheck();
       CHECK(pmon.getRepetitionCounter() == 2);

       ServiceTests::reset();
       Services.reset();
    }
```

Κώδικας 6.9: Σενάριο Δοκιμής για τον Έλεγχο Αναμενόμενης Τιμής - Τιμή ταιριάζει με την αναμενόμενη

Σε αυτό το σενάριο, αρχικά επαναλαμβάνεται η ίδια διαδικασία με το παραπάνω σενάριο και ορίζεται η τιμή της παραμέτρου σε 10 και η μάσκα σε 0xFF, η οποία σημαίνει ότι όλα τα `bits` είναι ενεργοποιημένα. Κατά την πρώτη κλήση της `performCheck`, ελέγχεται αν η κατάσταση ελέγχου είναι `ExpectedValue` και αν ο μετρητής επαναλήψεων έχει αυξηθεί σωστά. Η διαδικασία επαναλαμβάνεται για να διασφαλιστεί η ορθή ενημέρωση κατά τη διάρκεια διαδοχικών ελέγχων.

```
TEST_CASE("Delta Check Perform Check") {
    SECTION("Delta threshold checks including negative delta") {
        initialiseParameterMonitoringDefinitions();
        auto& pmon = fixtures.monitoringDefinition3;
        auto& param = static_cast<Parameter<unsigned
char>&>(pmon.monitoredParameter.get());

        param.setValue(10);
        ServiceTests::setMockTime(UTCTimestamp(2024, 4, 10, 10, 15, 0));
        pmon.performCheck();
        CHECK(pmon.getCheckingStatus() == PMON::Invalid);
        CHECK(pmon.getRepetitionCounter() == 1);


        ServiceTests::setMockTime(UTCTimestamp(2024, 4, 10, 10, 15, 15));
        param.setValue(180);
        pmon.performCheck();
        CHECK(pmon.getCheckingStatus() == PMON::AboveHighThreshold);
        CHECK(pmon.getRepetitionCounter() == 1);

        ServiceTests::setMockTime(UTCTimestamp(2024, 4, 10, 10, 15, 30));
        param.setValue(5);
        pmon.performCheck();
        CHECK(pmon.getCheckingStatus() == PMON::BelowLowThreshold);
        CHECK(pmon.getRepetitionCounter() == 1);

        ServiceTests::reset();
        Services.reset();
    }
```

Κώδικας 6.10: Σενάριο Δοκιμής για τον Έλεγχο Δέλτα

Για το σενάριο δοκιμής του ελέγχου Δέλτα, γίνεται αρχικοποίηση των δεδομένων και επιλέγεται η παράμετρος `monitoringDefinition3` από τη δομή `Fixtures`. Στη συνέχεια, καλύπτονται τρεις διαφορετικές περιπτώσεις για διαφορετικές τιμές της παραμέτρου και χρονικές στιγμές.

```
ServiceTests::setMockTime(UTCTimestamp(year:2024, month:4, day:10, hour:10, minute:15, second:0));
```

Κώδικας 6.11: Ορισμός Πεδίων της UTCTimestamp με την setMockTime

Πρώτα, η τιμή της παραμέτρου ορίζεται σε 10 και η τρέχουσα χρονική στιγμή καθορίζεται με την κλήση της `ServiceTests::setMockTime(UTCTimestamp(2024, 4, 10, 10, 15, 0))`. Κατά την πρώτη κλήση της `performCheck`, δεν υπάρχει προηγούμενη τιμή, οπότε η κατάσταση ελέγχου ορίζεται σε `Invalid` και ο μετρητής επαναλήψεων αυξάνεται σε 1.Στη συνέχεια, αλλάζει η τιμή της παραμέτρου σε 180 και η χρονική στιγμή καθορίζεται σε 15 δευτερόλεπτα αργότερα. Η δεύτερη κλήση της `performCheck` υπολογίζει τη διαφορά της τιμής ανά δευτερόλεπτο και την συγκρίνει

με τα καθορισμένα όρια. Εφόσον η μεταβολή είναι μεγαλύτερη από το ανώτερο όριο, η κατάσταση ελέγχου ορίζεται σε `AboveHighThreshold` και ο μετρητής επαναλήψεων επαναφέρεται σε 1.

Τέλος, η τιμή της παραμέτρου αλλάζει σε 5 και η χρονική στιγμή καθορίζεται σε 15 δευτερόλεπτα αργότερα από την προηγούμενη μέτρηση. Η τρίτη κλήση της `performCheck` υπολογίζει ξανά τη διαφορά τιμής ανά δευτερόλεπτο και τη συγκρίνει με τα καθορισμένα όρια. Από τη στιγμή που η μεταβολή είναι μικρότερη από το κατώτερο όριο, η κατάσταση ελέγχου ορίζεται σε `BelowLowThreshold` και ο μετρητής επαναλήψεων επαναφέρεται σε 1.

```
TEST_CASE("Check All Behavior") {
    SECTION("monitoringDefinition1 and 2 enabled") {
        initialiseParameterMonitoringDefinitions();
        auto& pmon1 = onBoardMonitoringService.getPMONDefinition(0).get();
        auto& pmonExpected = static_cast<PMONExpectedValueCheck&>(pmon1);
        auto& pmon2 = onBoardMonitoringService.getPMONDefinition(1).get();
        auto& pmonLimit = static_cast<PMONLimitCheck&>(pmon2);
        auto& pmon3 = onBoardMonitoringService.getPMONDefinition(2).get();
        auto& pmonDelta = static_cast<PMONDeltaCheck&>(pmon3);

        auto& param1 = static_cast<Parameter<unsigned
char>&>(pmon1.monitoredParameter.get());
        auto& param2 = static_cast<Parameter<unsigned
char>&>(pmon2.monitoredParameter.get());
        auto& param3 = static_cast<Parameter<unsigned
char>&>(pmon3.monitoredParameter.get());

        param1.setValue(10);
        pmonExpected.monitoringEnabled = true;
        pmonExpected.mask = 0xFF;
        param2.setValue(5);
        pmonLimit.monitoringEnabled = true;
        param3.setValue(100);

        CHECK(pmonExpected.getCheckingStatus() == PMON::Unchecked);
        CHECK(pmonLimit.getCheckingStatus() == PMON::Unchecked);
        CHECK(pmonDelta.getCheckingStatus() == PMON::Unchecked);

        onBoardMonitoringService.checkAll();

        CHECK(pmonExpected.getCheckingStatus() == PMON::ExpectedValue);
        CHECK(pmonLimit.getCheckingStatus() == PMON::WithinLimits);
        CHECK(pmonDelta.getCheckingStatus() == PMON::Unchecked);

        ServiceTests::reset();
        Services.reset();
    }
```

Κώδικας 6.12: Σενάριο Δοκιμής για τη Μέθοδο checkAll

Σε αυτό το σενάριο, δοκιμάζεται η λειτουργία της `checkAll` για να ελεγχθεί αν μόνο οι ενεργοποιημένοι ορισμοί παραμέτρων ελέγχονται. Αρχικά, οι ορισμοί παρακολούθησης αρχικοποιούνται, ακολούθως αποθηκεύονται σε μεταβλητές σύμφωνα με τη θέση τους στην λίστα `parameterMonitoringList` (getPMONDefinition().get()) και οι τιμές ορίζονται κατάλληλα για την `monitoringDefinition1`, `monitoringDefinition2`, και `monitoringDefinition3`. Κατόπιν, η `checkAll` καλείται και ελέγχεται αν οι καταστάσεις ελέγχου έχουν ενημερωθεί σωστά μόνο για τους ενεργοποιημένους ορισμούς, επιβεβαιώνοντας ότι η `checkAll` λειτουργεί όπως αναμένεται.

7 Σύνοψη και Μελλοντική Εργασία

7.1 Σύνοψη υλοποίησης

Η παρούσα διπλωματική εργασία παρουσίασε την υλοποίηση μιας αρχιτεκτονικής Ανίχνευσης, Απομόνωσης και Αντιμετώπισης Βλαβών (FDIR) για τον νανοδορυφόρο AcubeSAT, βασιζόμενη στο Ευρωπαϊκό Πρότυπο Αξιοποίησης Πακέτων ECSS και στο έργο των υπόλοιπων φοιτητών που ασχολούνται με αυτό το εγχείρημα.

Στα δύο πρώτα κεφάλαια έγινε ένας σύντομος σχολιασμός του επιστημονικού τομέα της εργασίας και μια παρουσίαση της αποστολής του AcubeSAT, των στόχων αυτής καθώς και μια αναφορά στα υποσυστήματα του νανοδορυφόρου. Στο Κεφάλαιο 3 έγινε γενική περιγραφή της αρχιτεκτονικής FDIR και των αρχών στις οποίες βασίζεται.

Ακολουθεί η ανάλυση της υπηρεσίας ST[12], η οποία είναι υπεύθυνη για την παρακολούθηση on-board παραμέτρων, στο Κεφάλαιο 4. Επιπλέον, σε αυτό το σημείο περιγράφονται οι τύποι ελέγχων που υποστηρίζει η υπηρεσία, οι ορισμοί παρακολούθησης παραμέτρων και οι καταστάσεις που μπορεί να βρεθούν κατά τη διάρκεια της παρακολούθησης. Στη συνέχεια, παρουσιάζεται η δομή του κώδικα, οι βασικές κλάσεις με τις ιδιότητές και τις συναρτήσεις τους, αλλά και ο τρόπος που είναι οργανωμένο το πρότζεκτ στο αποθετήριο

Στο Κεφάλαιο 5, περιγράφεται αναλυτικά η υλοποίηση των τηλε-εντολών που εκτελεί η υπηρεσία σε κώδικα σε σχέση με τις προδιαγραφές του προτύπου ECSS, μαζί με τα unit tests που πραγματοποιήθηκαν για την επαλήθευση της λειτουργίας του κώδικα σε διάφορες περιπτώσεις. Τελικά, στο Κεφάλαιο 6, παρουσιάζεται μια αρχική υλοποίηση της λογικής των ελέγχων ανάλογα με τον τύπο του ορισμού και των δοκιμών που πραγματοποιήθηκαν σε επίπεδο λογισμικού.

Συμπερασματικά, η ανάπτυξη της υπηρεσίας ST[12] είναι θεμελιώδης για την αξιοπιστία του FDIR, καθώς παρέχει τη δυνατότητα αυτοματοποιημένης και συνεχιζόμενης παρακολούθησης και διαχείρισης των παραμέτρων του δορυφόρου. Αυτό ενισχύει την αυτονομία του συστήματος και μειώνει την ανάγκη για χειροκίνητες παρεμβάσεις από τον επίγειο σταθμό, κάτι που είναι ιδιαίτερα σημαντικό στις αποστολές στο διάστημα όπου οι συνθήκες είναι απρόβλεπτες και η ανθρώπινη παρέμβαση περιορισμένη. Οι τύποι ελέγχων που υποστηρίζει η υπηρεσία, όπως οι έλεγχοι ορίου, αναμενόμενης τιμής και δέλτα, επιτρέπουν την ανίχνευση διαφόρων τύπων αποκλίσεων από τις κανονικές τιμές των παραμέτρων του συστήματος.

7.2 Μελλοντική εργασία

Η παρούσα υλοποίηση αποτελεί ένα σημαντικό βήμα προς την κατεύθυνση της δημιουργίας ενός αξιόπιστου προσαρμόσιμου συστήματος FDIR για τον AcubeSAT. Στο μέλλον, προτείνονται οι εξής βελτιώσεις και επεκτάσεις:

- **Δημιουργία και Διαχείριση Συμβάντων**: Αυτή η διαδικασία περιλαμβάνει τη σύνδεση με την υπηρεσία ST[05] Event Reporting, η οποία είναι υπεύθυνη για την αναφορά των συμβάντων. Κάθε φορά που μια παράμετρος υπερβαίνει τα όρια, θα πρέπει να δημιουργείται ένα συμβάν με τον κατάλληλο ορισμό και σοβαρότητα, το οποίο θα αναφέρεται μέσω της ST[05]. Η σοβαρότητα του συμβάντος και τα συνοδευτικά δεδομένα θα πρέπει να καθοριστούν με ακρίβεια, ώστε να παρέχουν όλες τις απαραίτητες πληροφορίες για την ανάλυση και την αντιμετώπιση του συμβάντος.

- **Λίστες Μεταβάσεων (Transition Lists)**: Οι λίστες μεταβάσεων θα πρέπει να υλοποιηθούν και να ελεγχθούν για να διασφαλιστεί η σωστή διαχείριση των αλλαγών κατάστασης των παραμέτρων. Οι λίστες αυτές θα παρακολουθούν τις μεταβάσεις των παραμέτρων από μια κατάσταση σε άλλη, και θα ενεργοποιούν τις κατάλληλες ενέργειες και συμβάντα ανάλογα με την αλλαγή. Η υλοποίηση των λιστών μεταβάσεων θα διασφαλίσει ότι κάθε αλλαγή κατάστασης καταγράφεται και αντιμετωπίζεται με τον σωστό τρόπο, παρέχοντας μια ολοκληρωμένη εικόνα της κατάστασης του συστήματος.

- **Δοκιμή του Κώδικα σε Πραγματικό Μικροελεγκτή**: Για να διασφαλιστεί η σωστή λειτουργία της υπηρεσίας ST[12] σε συνθήκες πραγματικού χρόνου, ο κώδικας θα πρέπει να μεταφερθεί και να δοκιμαστεί σε πραγματικό μικροελεγκτή. Η διαδικασία αυτή περιλαμβάνει τη σύνδεση του μικροελεγκτή με διάφορους αισθητήρες, όπως αισθητήρες θερμοκρασίας, και την ενσωμάτωση των τιμών τους ως ορισμούς παρακολούθησης για να ελεγχθούν από τους τύπους ελέγχου.

- **Συνεχής Ανανέωση του Κώδικα**: Ο κώδικας θα πρέπει να ανανεώνεται συνεχώς με βάση νέα δεδομένα και σενάρια που πρέπει να ελεγχθούν. Αυτό περιλαμβάνει την ενσωμάτωση των τελευταίων τεχνολογικών εξελίξεων και την προσαρμογή σε νέες απαιτήσεις της αποστολής. Η συνεχής ενημέρωση του κώδικα θα διασφαλίσει ότι η υπηρεσία ST[12] παραμένει ενημερωμένη και αποτελεσματική.

Α ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Orbiting now, Available:https://orbit.ing-now.com/

[2] T. Villela et al., "Towards the Thousandth CubeSat: A Statistical Overview," Special Issue: CubeSats and Small Satellites, vol. 2019, Article ID 5063145, Jan. 2019. [Online]. Available: https://doi.org/10.1155/2019/5063145

[3] S. Mane, "Theoretical Overview on CubeSat Technology," International Journal of All Research Education and Scientific Methods (IJARESM), vol. 12, no. 1, pp. 1107, January 2024. ISSN: 2455-6211.[Online].Available:https://www.researchgate.net/publication/377663430_Theoretical_Overview_on_CubeSat_Technology

[4] Endurosat, 3U CubeSat Platform https://www.endurosat.com/cubesat-store/cubesat-platforms/3u-cubesat-platform/

[5] S. A. Jacklin, “Small-Satellite Mission Failure Rates”, NASA/TM-2018-220034, Mar. 1, 2019. [Online]. Available: https://ntrs.nasa.gov/citations/20190002705

[6] A. Menchinelli, F. Ingiosi, L. Pamphili, P. Marzioli, R. Patriarca, F. Costantino, and F. Piergentili, “A Reliability Engineering Approach for Managing Risks in CubeSats”, Aerospace, vol. 5, no. 4, σ. 121, 4 Dec. 2018. : 10.3390/aerospace5040121. [Online]. Available: https://www.mdpi.com/2226-4310/5/4/121

[7] ACubeSat, “About ACubeSat”. [Online]. Available: https://acubesat.spacedot.gr/about/.

[8] European Space Agency, "Meet the team AcubeSAT". [Online]. Available: https://www.esa.int/Education/CubeSats_-_Fly_Your_Satellite/Meet_the_team_AcubeSAT.

[9] A. Zaras, K. Kapoglis, M. Georgousi, T. Papafotiou, M. Chadolias, A. Anthopoulos, A.-F. Retselis, A. Arampatzis, K.-O. Xenos, and E. Christidou, AcubeSAT Mission Description & Operations Plan, 2021. [Online]. Available: https://gitlab.com/acubesat/documentation/cdr-_public/-/blob/master/MDO%20file/MDO.pdf.

[10] AcubeSAT, "CDR Public," GitLab. [Online]. Available: https://gitlab.com/acubesat/documentation/cdr-public.

[11] K. Kanavouras, "Design of Fault Detection, Isolation and Recovery in the AcubeSAT Nanosatellite," Aristotle University of Thessaloniki, Thessaloniki, Greece, June 28, 2021. [Online]. Available: https://ikee.lib.auth.gr/record/332377/files/Konstantinos_Kanavouras_Undergraduate_Thesis.pdf

[12] ECSS Secretariat, “ECSS-E-ST-70-41C – Telemetry and telecommand packet utilization”, European Space Agency, Apr. 15, 2016. [Online]. Available: https://ecss.nl/standard/ecss-e-st-70-41cspace-engineering-telemetry-and-telecommand-packet-utilization-15-april-2016/

[13] Space Avionics Open interface Architecture, “SAVOIR FDIR Handbook”, European Space Agency, SAVOIR-HB-003, Oct. 2019. [Online]. Available: https://essr.esa.int/project/savoir

[14] A.-F. Retselis and K. Kanavouras, AcubeSAT FMEA File, 2021. [Online]. Available: https://gitlab.com/acubesat/documentation/cdr-public/-/blob/master/FMEA%20file/FMEA.pdf

[15] ACubeSat, "ECSS Services," GitLab. [Online]. Available: https://gitlab.com/acubesat/obc/ecss-services.

[16] ETLCPP, "Embedded Template Library," [Online]. Available: https://www.etlcpp.com/home.html.

Β ΠΗΓΑΙΟΣ ΚΩΔΙΚΑΣ

https://github.com/Pravinos/AcubeSat-ECSS-ST12-Service

Το πρότζεκτ είναι επίσης διαθέσιμο στο δημόσιο αποθετήριο των ECSS υπηρεσιών της αποστολής [15]. Ωστόσο, στο μέλλον πιθανότατα θα υπάρξουν αλλαγές, καθώς ο κώδικας μπορεί να ενημερωθεί ή να επεκταθεί από μέλη της ομάδας.