POLITECNICO DI TORINO

III Facoltà di Ingegneria

Corso di Laurea in Ingegneria Elettronica

Tesi di Laurea Magistrale

Sviluppo del sottosistema di gestione del nanosatellite universitario AraMiS

Relatori:
prof. Leonardo Reyneri
prof. Claudio Passerone
ing. Juan Carlos De Los Rios Arbelaez

Candidato:
Stefano Iannone

Novembre 2011

Sommario

In ambito universitario la progettazione di satelliti di piccole dimensioni, detti anche nano-satelliti, ha avuto una crescita molto forte, soprattutto negli ultimi 8-10 anni e dovuta alla definizione dello standard CubeSat.
L'esplorazione dello spazio al Politecnico di Torino è iniziata nel 2004 con il progetto PiCPoT (Piccolo Cubo del Politecnico di Torino) che prevedeva la progettazione di un satellite, basato su CubeSat, i cui obiettivi erano verificare l'affidabilità di componenti commerciali in ambiente spaziale, scattare delle fotografie della Terra ed acquisire/trasmettere dati. Il satellite è stato completato e lanciato, purtroppo con esito negativo, nel 2006.
Poco dopo il lancio di PiCPoT ha preso il via il progetto del secondo satellite universitario del Politecnico di Torino, AraMiS. Con questo progetto, attualmente in fase di sviluppo, si vuole creare un nuovo approccio alla realizzazione di nano-satelliti.
L'acronimo AraMiS significa Architettura Modulare per Satelliti e indica lo scopo principale del progetto, ovvero realizzare un satellite modulare a basso costo, dove i moduli di cui è composto sono indipendenti e configurabili a seconda della missione. Ognuno di questi blocchi può essere progettato indipendentemente dagli altri, riutilizzato in progetti differenti ed è in grado di svolgere una determinata operazione in modo efficiente.
AraMiS è composto da due blocchi standard, detti tile, che contengono al loro interno un certo numero di moduli.

Power Management Tile : su queste tile sono installati i pannelli solari e le batterie, necessari ad alimentare il satellite. L'equipaggiamento di queste tile comprende anche i moduli di controllo d'orbita e d'assetto. Nella configurazione base, un satellite AraMiS è composto da 5 di queste tile.

Telecommunication Tile : permette la comunicazione tra il satellite e la stazione di Terra; utilizza due canali, alle frequenze di 437MHz e 2,4 GHz. Su questo blocco è installato l'On Board Computer (OBC), che ha il compito di gestire l'intero satellite.

L'argomento di questa tesi è il modulo di gestione dei dati di housekeeping del satellite AraMiS, indicato dal nome 1B43 Housekeeping Management, del quale saranno illustrati tutti gli step dalla progettazione al collaudo.
Il modulo 1B43 Housekeeping Management è una parte dell'On Board Computer ed ha lo scopo di acquisire ed analizzare le telemetrie di tutte le tile presenti nel satellite. Oltre ai vettori di housekeeping, il modulo può richiedere alle tile la loro configurazione o il loro stato, al fine di evitare errori in fase di analisi.
Nel caso in cui la configurazione del satellite abbia subito delle modifiche, il modulo ricalcola alcuni parametri necessari per analizzare i vettori di housekeeping. Data la complessità del calcolo, che richiede algoritmi di approssimazione numerica, il modulo ha la possibilità di richiedere l'aggiornamento dei parametri alla stazione di Terra.
Il progetto è stato sviluppato utilizzando il linguaggio UML, che permette di descrivere un sistema per mezzo di una serie di diagrammi. In particolare, i Class Diagram hanno permesso di descrivere i blocchi funzionali che compongono il sistema, gli Sequence Diagram sono stati utilizzati per mostrare come questi oggetti interagiscono tra loro mentre gli Use Case Diagram sono serviti ad illustrare le specifiche del sistema.

Il documento è strutturato come segue:

Capitolo 1 : descrive brevemente il mondo dei nano-satelliti e i due progetti del Politecnico di Torino;
Capitolo 2 : illustra le basi del linguaggio UML;
Capitolo 3 : descrive i concetti teorici utilizzati nel progetto;
Capitolo 4 : descrive il funzionamento del progetto;
Capitolo 5 : descrive i test effettuati sul sistema;
Appendice A : riporta gli acronimi utilizzati nel documento.

Indice

Capitolo 1

I satelliti di piccole dimensioni

Nell'ultimo decennio la ricerca nel campo dei satelliti di piccole dimensioni, detti anche mini-satelliti, ha avuto un forte incremento, dovuto alla spinta da parte delle università e dei centri di ricerca. Uno dei punti di forza dei satelliti di piccoli dimensioni sono i costi contenuti di lancio e di progettazione.
Infatti i mini-satelliti vengono solitamente realizzati con materiali Commercial Off-The-Shelf (COTS), che permettono di ridurre i costi di progettazione. Inoltre diminuendo le dimensioni, e di conseguenza il peso del satellite, anche i costi di lancio risulteranno più bassi. Le diverse fasce di peso in cui poter catalogare un mini-satellite sono mostrate nella seguente tabella.

Categoria	Peso (kg)
Mini	100 - 500
Micro	10 - 100
Nano	1 - 10
Pico	0.1 - 1
Femto	0.1

Sempre più spesso le università sviluppano dei satelliti di piccole dimensioni per fare della ricerca in ambito aerospaziale. Nel 2001 è stato definito dalla Stanford University e dalla California Polytechnic State University lo standard CubeSat [4, 5], pensato per la creazione di nano-satelliti. Molti dei satelliti universitari già realizzati, o in fase di realizzazione, si basano su questo standard. Anche i satelliti del Politecnico di Torino, PiCPoT [14] e AraMiS [1], seguono questo standard.
L'obiettivo di CubeSat è quello di definire specifiche e metodologie per facilitare la creazione e il lancio di satelliti di piccole dimensioni. Un satellite CubeSat, per rispettare lo standard, deve avere forma cubica con lato di 10 cm e massa non superiore a 1 kg. Avendo piccola massa è possibile scegliere un vettore di lancio più economico, che unito all'utilizzo di componenti COTS, permette di ridurre la spesa complessiva della missione.
Le piccole dimensioni permettono di diminuire, oltre al peso, anche i tempi di sviluppo e produzione del satellite; mediamente lo sviluppo di un satellite CubeSat dura circa due anni.

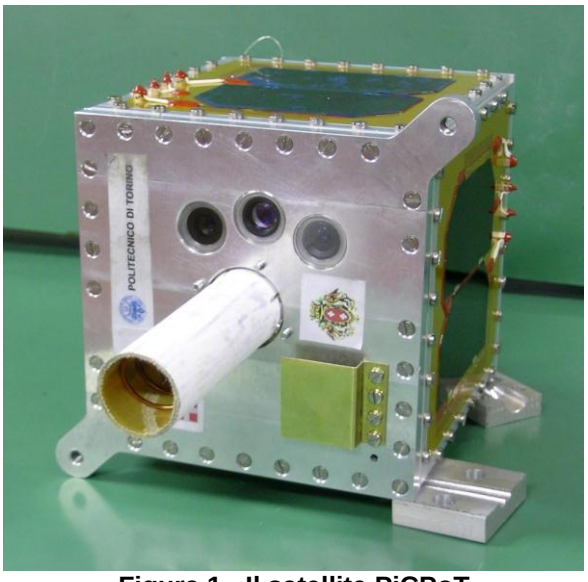


Figura 1 - Il satellite PiCPoT

1. PiCPoT

Nato dalla collaborazione tra il Dipartimento di Ingegneria Elettronica e il Dipartimento di Ingegneria Aerospaziale, il progetto PiCPoT (*Piccolo Cubo del Politecnico di Torino*) aveva lo scopo di progettare, realizzare e lanciare in orbita un nano-satellite sperimentale. Al progetto PiCPoT, partito nel 2004 e terminato nel 2006, hanno preso parte docenti, dottorandi e studenti di diversi dipartimenti del Politecnico.
Il nano-satellite PiCPoT è, proprio come dice il suo nome, un piccolo cubo di 13 cm di lato realizzato in lega di alluminio di tipo 5000 AlMn. Come ogni satellite, anche PiCPoT aveva la sua missione, che può essere riassunta nei tre punti seguenti:

- verificare il funzionamento e l'affidabilità dei componenti COTS in ambiente spaziale;
- acquisire e trasmettere immagini e misurazioni effettuate dai sensori di bordo alla Stazione di Terra;
- scattare delle fotografie della superficie terrestre.

L'immagine 2 a fondo pagina rappresenta il satellite visto da due angolazioni differenti, in modo da mostrare tutte le facce di PiCPoT, numerate da 0 a 5. Sulle facce dalla F_1 alla F_5 sono presenti dei pannelli solari, che servono come alimentazione e per ricaricarne le batterie, mentre sulla faccia F_0 sono montate tre fotocamere e le antenne per la comunicazione con la stazione di Terra.
Le batterie sono di due tipi: NiCd e NiHM; all'interno il satellite ospita una serie di moduli, ognuno realizzato su una scheda elettronica. Di seguito verranno descritte le caratteristiche di ciascun modulo.

Power Supply : Gestisce la prima parte dell'impianto di alimentazione, ovvero i pannelli solari, le batterie e i caricabatterie. Questo modulo ha il compito di ricaricare le batterie tramite i pannelli solari, e di tenere sotto controllo i parametri elettrici e le temperature delle batterie.

Power Switch : Genera le tensioni di alimentazione per le varie schede elettroniche a partire da quella delle batterie. Inoltre permette di attuare politiche di risparmio energetico, ovvero è in grado di scollegare l'alimentazione ai moduli non utilizzati per ridurre i consumi.

RxTx : È il modulo per la comunicazione tra la stazione di terra e il satellite. La comunicazione avviene tramite due canali half-duplex, uno a 437MHz e l'altro a 2,4 GHz, in modo da rendere il sistema ridondante e meno soggetto a guasti.

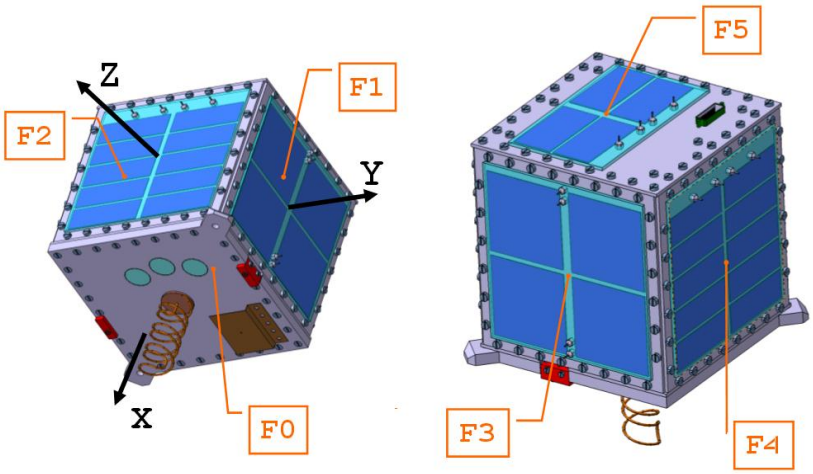


Figura 2 - Vista esterna di PiCPoT

ProcA e ProcB : "Semplicemente" il computer di bordo. Entrambe le schede ProcA e ProcB, realizzate con componenti e tecnologie differenti per aumentare la tolleranza ai guasti, contengono il computer di bordo del satellite. Questa ridondanza permette al sistema di funzionare anche in caso di guasti al computer di bordo. Le due schede lavorano indipendentemente l'una dall'altra ma mai contemporaneamente, per garantire il risparmio energetico. Tra i loro compiti ci sono la lettura delle telemetrie e la comunicazione con i moduli Payload e RxTx.

Payload : Costituito principalmente dalle tre telecamere. Il suo compito è di acquisire le immagini, comprimerle in formato JPEG e inviarle al computer di bordo, che provvederà a trasmetterle a terra.

Nello schema a blocchi 3 si può vedere com'è strutturato il satellite e come i blocchi presentati in precedenza interagiscono tra loro.

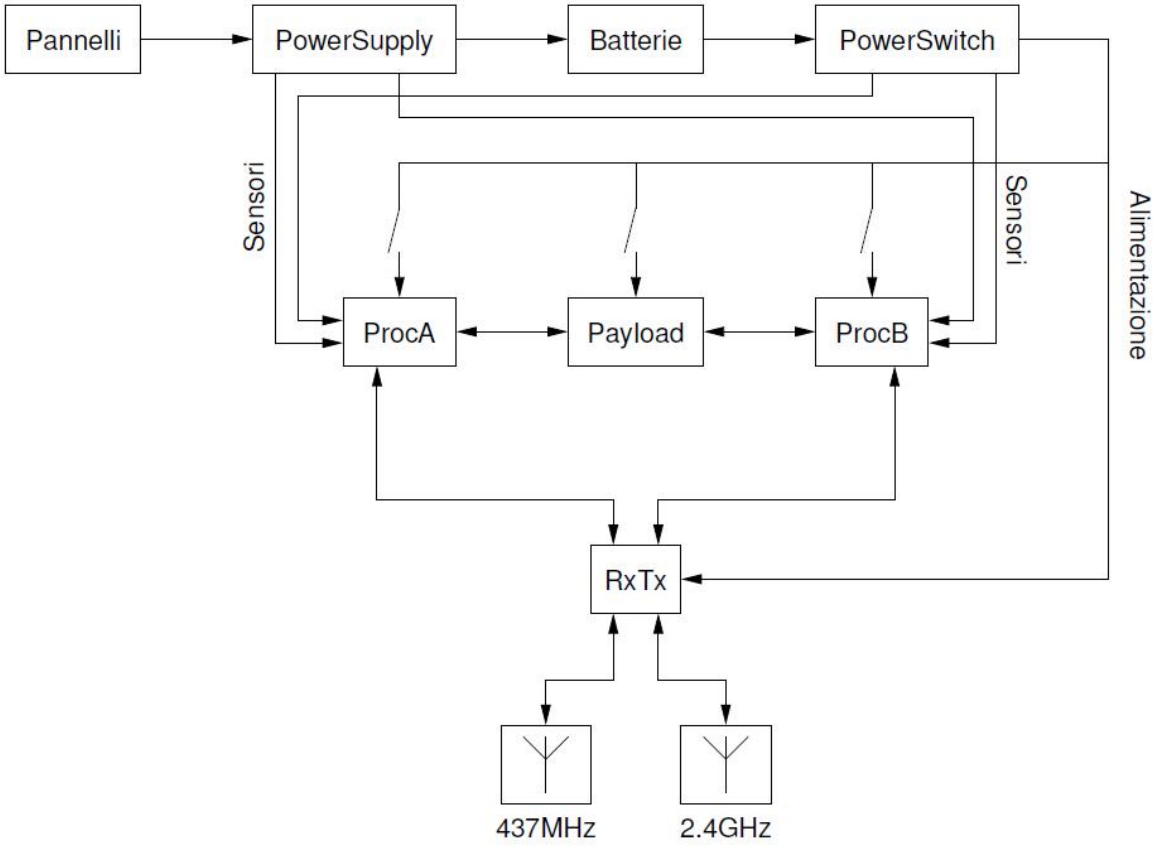


Figura 3 - Schema a blocchi del satellite PiCPoT

Il progetto PiCPoT si è concluso il 26 luglio 2006, con il lancio del satellite dalla base di Baikonur [2, 3] in Kazakistan, a bordo di un lanciatore Dnepr-1 [7]. Purtroppo, a causa di un malfunzionamento dell'unità di pompaggio idraulico di una camera di combustione del vettore, il lancio non è andato a buon fine.

2. AraMiS

Alcuni mesi dopo il lancio di PiCPoT prese il via il nuovo progetto del Politecnico di Torino sui nano-satelliti, AraMiS.

A questo nuovo progetto, il cui nome significa *Architettura Modulare per Satelliti*, partecipano oltre a docenti, dottorandi e studenti dei dipartimenti di Ingegneria Elettronica e di Ingegneria Aerospaziale del Politecnico di Torino, le aziende Neohm Componenti [13], Sky Technology [16] e Spin Electronics [17].
Il progetto AraMiS sta cercando di introdurre un nuovo approccio alla realizzazione di nano-satelliti. Il nuovo obiettivo, oltre a quelli ereditati da PiCPoT, è di realizzare un satellite a basso costo che possa adattarsi a più missioni, anche molto differenti tra loro. L'architettura del satellite è infatti suddivisa in blocchi funzionali ben definiti e indipendenti a livello meccanico, elettronico e di test. Hanno però un linguaggio di comunicazione comune e un'organizzazione gerarchica che consente di affidare la gestione di un modulo a quello di livello superiore.
L'utilizzo di un approccio modulare permette di ridurre oltre ai tempi di progetto, grazie al fatto che la configurazione dei blocchi funzionali può essere svolta da più gruppi di lavoro contemporaneamente, anche i costi di realizzazione, visto che i moduli possono essere riutilizzati in più missioni.
I moduli sono divisi in categorie, elencate sotto, in base al loro compito all'interno del satellite:

- 1B1 Power Management Subsystem;
- 1B2 Attitude and Orbit Subsystem;
- 1B3 TT&C Telecommunication Subsystem;
- 1B4 On-Board Data Handling Subsystem;
- 1B5 Payload Subsystem;
- 1B6 Mechanical Subsystem;
- 1B7 Controllers and Supervisor Subsystems.

I moduli elencati sopra vengono montati su degli appositi supporti chiamati tile (letteralmente "piastrella" o "tegola"), che andranno a costituire la struttura esterna del satellite. Esistono due tipi di tile, la 1B8 Power Management Tile e la 1B9 Telecomunication Tile, che differiscono per i moduli con cui sono equipaggiate. Di seguito vengono descritte le caratteristiche delle tile e di alcune parti fondamentali del satellite AraMiS.

1B8 Power Management Tile (PMT) : Ha il compito di fornire le alimentazioni al satellite tramite le batterie e i pannelli solari montati sul lato esterno. È inoltre dotata di sensori ed attuatori per il controllo attivo dell'assetto del satellite. Vi sono installati i moduli 1B1, 1B2 e 1B6.

1B9 Telecomunication Tile (TLC) : Permette la comunicazione con la stazione di terra. Ha le stesse caratteristiche del modulo RxTx di PiCPoT, ovvero due canali RF a 437MHz e 2,4 GHz half-duplex. L'utilizzo di due canali permette di avere ridondanza nelle comunicazioni oltre a prevenire i guasti. È composta dai moduli 1B3, 1B4, 1B6 e 1B7.

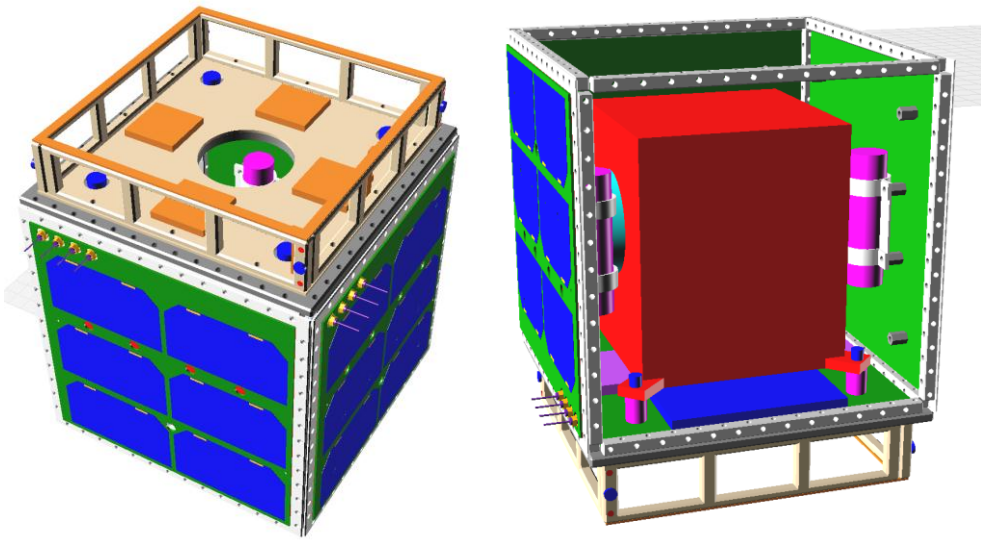


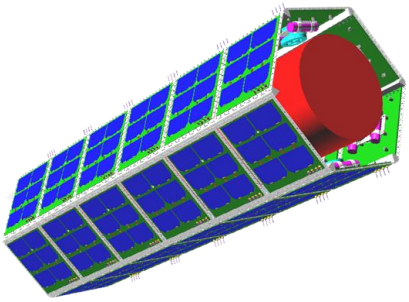
Figura 4 - Configurazione base del satellite AraMiS

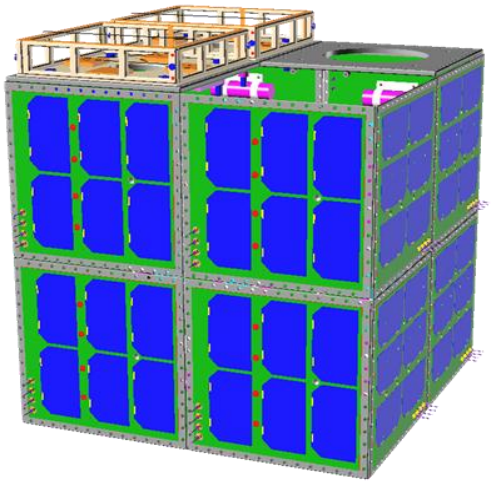
On Board Computer (OBC) : È il computer di bordo e corrisponde al modulo 1B4. Ha la gestione del sistema (compreso il payload) e delle comunicazioni da/verso Terra. Fisicamente è collocato sulla TLC tile.

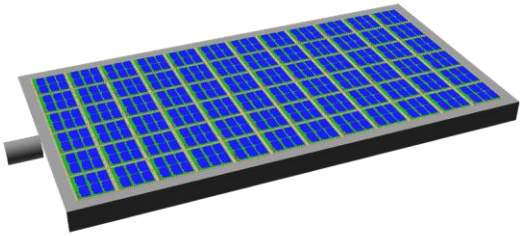
Payload : Lo scopo della missione. Può essere composto da tre telecamere, come nel satellite PiCPoT, oppure da un telescopio commerciale, come nell'esempio in figura. Corrisponde al modulo 1B5.

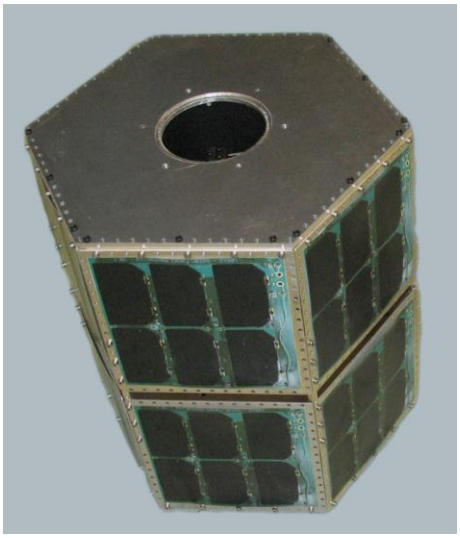
Tra i moduli descritti in precedenza ce ne sono alcuni, come l'1B4, che sono quasi completamente relativi alla parte software del sistema, mentre altri, ad esempio l'1B1, fortemente legati all'hardware; tra tutti, l'1B6 è un modulo particolare. Contiene infatti tutte le informazioni relative alle caratteristiche termo-meccaniche del satellite ed è proprio per questo motivo che è presente in entrambe le tile.
La configurazione base del satellite AraMiS (fig. 4) è un cubo, composto da 5 PMT tile e da una TLC tile, con lato di 16,5 cm e massa inferiore a 5 kg. I satelliti AraMiS sono destinati a missioni di medio-breve durata (circa 5 anni) in *Low Earth Orbit* (LEO), più precisamente ad orbite comprese tra i 600km e gli 800km. A queste quote, la *Total Ionizing Dose* (TID), ossia l'energia assorbita da un corpo dovuta alle radiazioni ionizzanti, del satellite è superiore a 15 krad (o 150 Gy).

Le figure nella pagina seguente mostrano altre possibili configurazioni di AraMiS.









Capitolo 2

Il linguaggio UML

La creazione e la gestione di applicazioni di alta qualità nell'ambito dell'*Information Technology* è un compito sempre più difficile, specie se lo sviluppo deve avvenire in tempi relativamente brevi. La metodologia *Visual Modeling* si propone come soluzione a questo problema.
Per prima cosa definiamo il concetto di *Modeling*. Con questo termine si indica la realizzazione di un modello del sistema da sviluppare che corrisponda alle richieste del cliente e che sia condiviso dal team di lavoro. Ovviamente bisognerà anche controllare che la costruzione del sistema segua i requisiti imposti dal modello creato.
Il *Visual Modeling* ha ormai quasi rimpiazzato il metodo utilizzato precedentemente, chiamato *Waterfall Method* (o metodo a cascata), grazie al fatto che lo sviluppo del sistema può essere fatto in parallelo sfruttando il modello realizzato in precedenza. Nel metodo a cascata lo sviluppo avveniva in modo sequenziale, cosa che rendeva il processo produttivo più lento e inefficiente.
Il *Visual Modeling* e il linguaggio UML (Unified Modelling Language), le cui basi verranno spiegate in questo capitolo, sono molto simili o, per meglio dire, l'UML è il linguaggio da utilizzare per realizzare un progetto utilizzando la metodologia *Visual Modeling*.
L'Unified Modelling Language (UML) è un linguaggio utilizzabile nella descrizione di un sistema di natura qualunque (anche se viene usato principalmente nello sviluppo di software) basato sul paradigma *Object-Oriented*. È un linguaggio grafico, nato nel 1995 e divenuto standard nel gennaio 2005 (ISO/IEC 19501:2005), creato proprio per fornire agli sviluppatori di software *Object Oriented* un metodo per modellare i loro progetti secondo regole comuni. Il fatto di comunicare nella stessa "lingua" aiuta ad evitare rischi di incomprensioni e quindi sprechi di tempo, in quanto l'interazione tra le risorse umane coinvolte nel progetto è più diretta e efficiente.
Un modello UML è costituito da una serie di diagrammi di vario tipo, tipicamente collegati tra di loro, composti da elementi grafici (ognuno con un significato univoco), elementi testuali formali ed elementi di testo libero. La semantica è molto precisa e conferisce a ciascun diagramma un grande potere comunicativo.
Il linguaggio UML permette di descrivere i tre aspetti principali di un sistema grazie ad un insieme di diagrammi:

Modello Funzionale : Rappresenta il punto di vista dell'utente. Serve a descrivere il funzionamento del sistema così com'è percepito dall'esterno, a prescindere dalla struttura interna dell'oggetto. Utilizza gli *Use Case Diagram.*

Modello ad Oggetti : Rappresenta la composizione interna del sistema, in termini di classi, oggetti e relazioni tra gli stessi. Utilizza i *Class Diagram*, gli *Object Diagram* e i *Deployment Diagram.*

Modello Dinamico : Rappresenta il comportamento degli oggetti nel sistema, ovvero come interagiscono tra loro e come evolvono nel tempo. Utilizza i *Sequence Diagram*, gli *Activity Diagram* e gli *Statechart Diagram*.

Il punto di forza del linguaggio UML sta nel fatto che chiunque sia coinvolto nel progetto, dai programmatori ai clienti, possa capire ed esaminare il sistema efficientemente. Nel seguente elenco vengono illustrati alcuni benefici derivanti dall'uso dell'UML:

- il sistema viene realizzato professionalmente e documentato ancor prima di scrivere il codice da parte degli sviluppatori;

- la scrittura del codice è resa più agevole ed efficiente, inoltre è più possibile scrivere codice riutilizzabile in progetti differenti;
- è più facile prevedere e anticipare eventuali "buchi" nel sistema;
- tutte le persone coinvolte nel progetto avranno una chiara idea di ciò che fa il sistema. Questo permette di sfruttare al meglio le risorse hardware;
- eventuali modifiche future risultano più semplici da applicare e da integrare con il progetto.

Nelle sezioni seguenti verranno mostrate le caratteristiche dei diagrammi UML utilizzati nello svolgimento di questa tesi.

1. Use Case Diagram

Un *diagramma di caso d'uso*, o *Use Case Diagram*, è un diagramma descrittivo del linguaggio UML il cui scopo è fornire un riepilogo delle funzionalità di un sistema. Ogni sistema può essere rappresentato come un insieme di attori, ovvero chiunque operi sull'oggetto (quindi non necessariamente solo esseri umani), e di casi d'uso. Ad ogni caso d'uso corrispondono una o più operazioni che il sistema è in grado di svolgere.
Questi diagrammi possono essere considerati come uno strumento per la rappresentazione dei requisiti funzionali del sistema oppure per modellare i servizi offerti da un certo modulo o sottosistema ad altri moduli. Un esempio di Use Case Diagram è mostrato in figura 5.

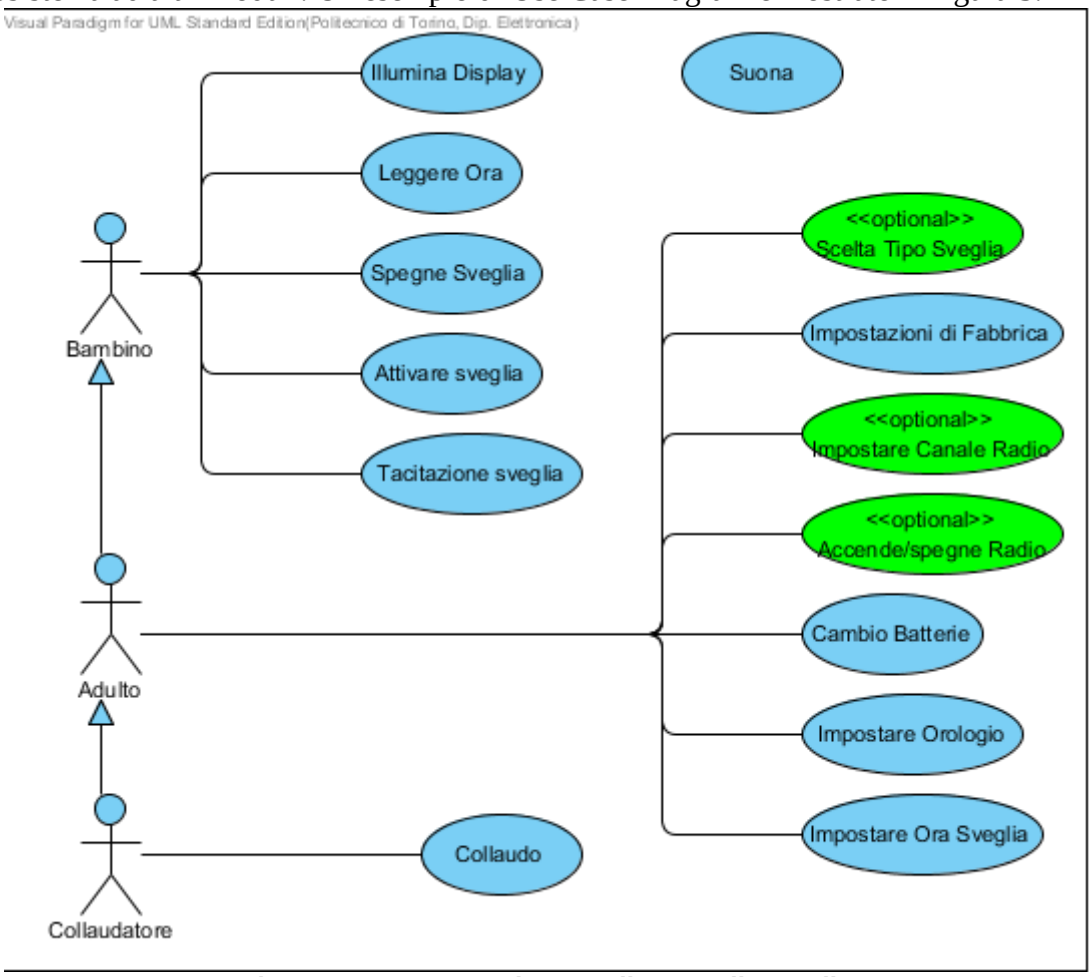


Figura 5 - Use Case Diagram di una radiosveglia

Come già citato all'inizio di questo capitolo, ogni elemento di un digramma UML ha un significato univoco e la semantica è molto precisa. In seguito verranno illustrate le

caratteristiche degli elementi che compongono un diagramma di caso d'uso e la semantica (ovvero le relazioni che legano gli elementi tra loro) specifica per ciascun oggetto. Un Use Case Diagram è composto principalmente da due elementi, ovvero i già citati casi d'uso e gli attori; esistono anche altri oggetti ma, poiché non sono stati utilizzati nello sviluppo del progetto, non verranno illustrati per non appesantire la trattazione.
Un attore (o actor) è, in generale, una qualunque entità (un utente, un software esterno, dispositivi hardware, ecc. . .) che prende parte ad una o più interazioni con il sistema. Nei diagrammi viene rappresentato da un uomo stilizzato, come si può vedere in figura 10. Tra gli attori una delle relazioni più comuni è la generalizzazione (o specializzazione), che serve a definire una gerarchia tra essi. Questa relazione viene rappresentata da una linea continua che termina in un triangolo vuoto tracciata dall'attore specializzato a quello più generico. Ad esempio, in figura 5 il collaudatore è un caso particolare dell'adulto, che è a sua volta una specializzazione del bambino. Gli attori specializzati ereditano tutti i casi d'uso degli attori generici da cui derivano, ma non viceversa.

Nel libro Use Case Modeling [20] è possibile trovare un'ottima definizione di caso d'uso:

Use case rapresent the things of value that the system performs for its actors. Use cases are not functions or features, and they cannot be decomposed. Use cases have a name and a brief description.They also have detailed descriptions that are essentially stories about how the actors use the system to do something they consider important, and what the system does to satisfy these needs.

di cui viene riportata la traduzione:

I casi d'uso rappresentano le cose di valore che il sistema esegue per i suoi attori. I casi d'uso non sono funzioni né caratteristiche, e non possono essere decomposti. I casi d'uso hanno un nome e una breve descrizione. Hanno anche delle descrizioni dettagliate che sono sostanzialmente storie su come gli attori usano il sistema per fare qualcosa che considerano importante, e cosa fa il sistema per soddisfare queste esigenze.

Gli Use Case sono rappresentati nei diagrammi UML da degli ellissi orizzontali contenenti il nome dell'oggetto; tra i casi d'uso le relazioni più comuni, mostrate in figura 6, sono quattro:

Generalizzazione : Anche per i casi d'uso esiste la questa relazione, che funziona in maniera analoga a quella degli attori. Un "sotto-caso d'uso" deve fornire lo stesso servizio generale del "super-caso d'uso", eventualmente seguendo un procedimento parzialmente diverso per ottenere il medesimo risultato, oppure producendo un valore aggiuntivo.

Inclusione : «Include è un rapporto diretto tra due casi d'uso, ciò implica che il comportamento dello Use Case incluso è inserito in quello del caso d'uso "includente"» [19, pp. 610-612]. La relazione è rappresentata con una freccia tratteggiata, contrassegnata dall'etichetta «*include*», che parte dal caso d'uso principale e termina in quello incluso.

Estensione : La relazione indica che un dato caso d'uso può estenderne un altro; specifica che il comportamento dell' "estensione" è in grado di essere inserito nel caso d'uso esteso sotto certe condizioni. Sono rappresentate da una freccia tratteggiata dall' "estensione" al caso d'uso esteso, con l'etichetta «*extend*». A questa relazione possono essere associate delle note o delle restrizioni in modo da illustrare le condizioni in cui viene eseguita l'estensione. La relazione «*extend*» è utilizzata per indicare Use Case "opzionali" rispetto a quello base.

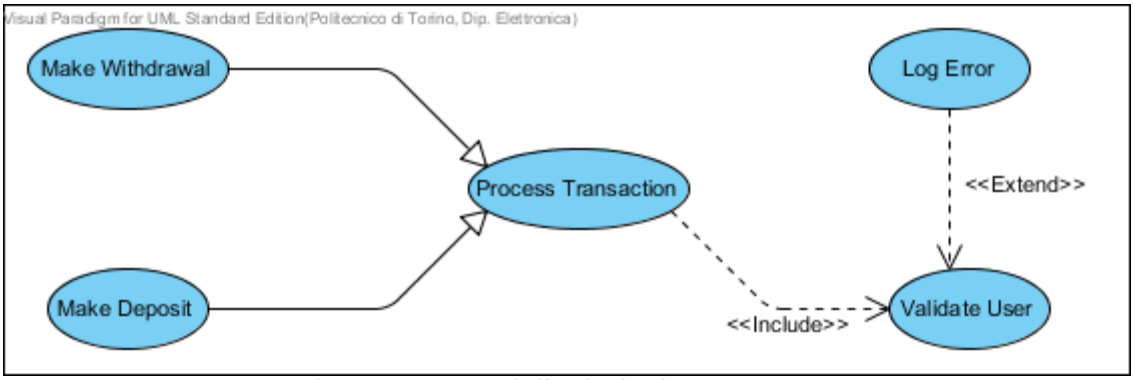


Figura 6 - Esempi di relazioni tra Use Case

Associazione : Un'associazione esiste quando un attore utilizza un servizio proposto da un caso d'uso. Sono rappresentate da una linea che collega un caso d'uso ad un attore e, opzionalmente, con una freccia ad uno dei due capi. La freccia indica il principale attore del caso d'uso o la direzione della prima invocazione del rapporto.

Le etichette poste sulle frecce in alcuni tipi di relazione o in alcuni casi d'uso, come si può vedere in figura 5 a pagina 8, prendono il nome di *stereotypes*. Gli stereotipi, utilizzati in tutti i diagrammi UML, servono a caratterizzare un oggetto, o ad estenderne il significato; nell'esempio 5 lo stereotipo «*optional*» indica, com'è facile intuire, che la funzionalità descritta non è strettamente necessaria per il funzionamento corretto del sistema.

2. Class Diagram

Tra tutti i diagrammi presenti nello standard UML, i più utilizzati sono i *Class Diagram*, o *diagrammi delle classi*. Questo tipo di diagramma rappresenta la struttura del sistema utilizzando degli oggetti, chiamati classi, e come essi sono relazionati tra loro.
I Class Diagram hanno una doppia funzionalità. Permettono infatti di descrivere in modo generico la struttura del sistema, ma consentono anche, rendendo il modello più dettagliato, di tradurre il diagramma in codice sorgente. Le classi contenute nel diagramma rappresentano sia gli oggetti di cui è costituito il sistema, sia gli oggetti che dovranno essere programmati. Per poter comprendere a pieno il resto dell'elaborato, conviene definire e spiegare il concetto di classe e le caratteristiche di questo elemento così importante nel linguaggio UML.

Una classe è, nella programmazione Object Oriented, un costrutto utilizzato come modello per creare altri oggetti, o istanze. Una classe rappresenta l'astrazione di un oggetto e, tramite le sue operazioni e i suoi attributi, ne descrive il comportamento e definisce lo stato. Nei diagrammi UML, una classe è rappresentata, come si può vedere in figura 7 a pagina 12, da un rettangolo, tipicamente diviso in tre parti. La parte superiore, al contrario delle altre due, è sempre presente e contiene il nome della classe e, se ce ne sono, gli stereotypes; nella seconda parte sono contenuti gli attributi e nell'ultima le operazioni (dette anche metodi). A tutti i membri di una classe, come gli attributi e i metodi, sono aggiunte alcune informazioni supplementari; quelle descritte di seguito riguardano la visibilità di ciascun membro nel sistema.

Public : Indicato con il simbolo +, specifica che il membro è visibile a tutte le altre classi nel diagramma.
Private : L'oggetto è visibile solo all'interno della classe di appartenenza ed è indicato con il simbolo −.

Protected : Indica che l'elemento è visibile solo alla classe a cui appartiene e alle sue sottoclassi. È rappresentato dal simbolo #.

Un'altra delle informazioni supplementari riguarda lo scope, ovvero l'ambito in cui l'elemento opera; può avere due valori, la cui definizione è diversa per attributi e metodi:

Istance : Se un attributo è definito istance, significa che il suo valore può variare tra le diverse istanze. Quando un metodo è definito istance, la sua invocazione influenza gli attributi dell'istanza.

Classifier : Un attributo classifier mantiene lo stesso valore in tutte le istanze. Allo stesso modo un metodo definito classifier lascia inalterati gli attributi quando viene invocato. Nei diagrammi delle classi i membri con questo scope appaiono sottolineati. Quando i Class Diagram vengono tradotti in codice, tutti gli oggetti con scope classifier vengono riconosciuti come static dalla maggior parte dei linguaggi di programmazione.

Dopo aver descritto il concetto di classe, è necessario spiegare anche le relazioni tra le classi che vengono utilizzate nei Class Diagram.

Association : È la relazione su cui sono basate tutte le altre, oltre ad essere una delle più utilizzate nei diagrammi di classe. Vi è un'associazione tra due (o più) classi quando una possiede un attributo definito come un'istanza di un'altra. Nell'esempio in figura 7 nella pagina seguente le classi Orologio e Contatore sono associate in quanto nella prima vi sono tre istanze della seconda. Le associazioni possono essere unidirezionali o bidirezionali e sono indicate con una linea.

Aggregation : Rappresenta una relazione più forte della semplice associazione. L'aggregazione implica che una classe è un insieme (o un contenitore) di altre classi, che però non sono essenziali per l'esistenza del contenitore o, come si dice in gergo, non hanno una forte dipendenza del ciclo di vita del contenitore. Ciò significa che se il contenitore viene distrutto, il contenuto continua a esistere. Questa relazione viene indicata graficamente con un rombo vuoto dal lato del contenitore che è collegato con una linea alle classi in esso contenute.

Composition : La composizione è una variante dell'associazione ancora più forte rispetto al caso precedente. Infatti, al contrario di quanto avviene nell'aggregazione, le classi contenute sono correlate al ciclo di vita del contenitore. Ovvero se il contenitore cessa di esistere, anche le classi in esso contenute verranno distrutte. Il simbolo utilizzato per questa relazione è del tutto simile a quello descritto per l'aggregazione; la solo differenza sta nel rombo che in questo caso è pieno, come si vede in figura 7.

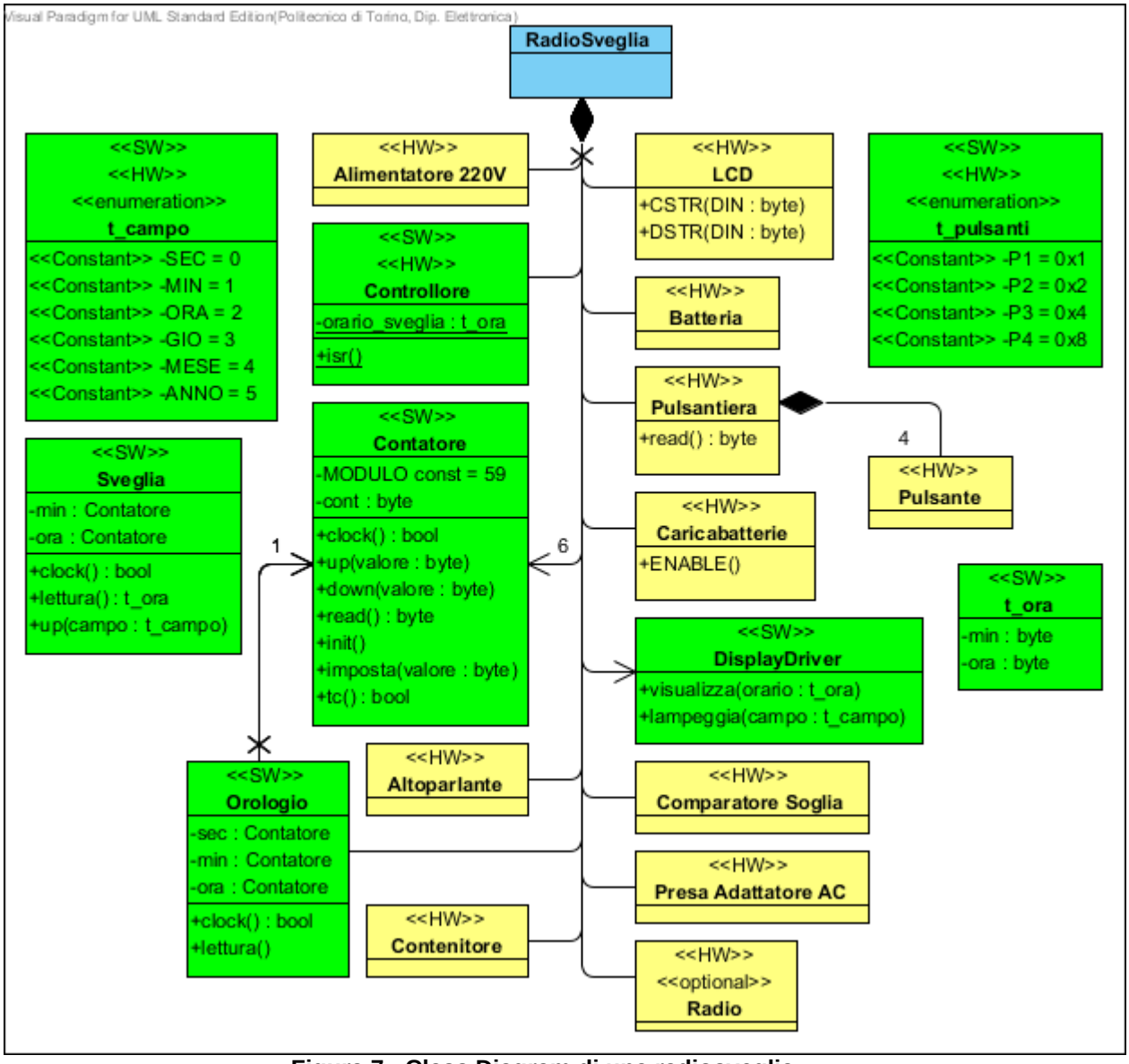


Figura 7 - Class Diagram di una radiosveglia

Generalization : La generalizzazione si comporta in modo analogo a come spiegato nella sezione precedente. Trattandosi però di classi, verranno ereditati i metodi e gli attributi.

Realization : Nella modellazione UML, la realizzazione è rapporto tra le classi del modello, in cui uno o più elementi (detti clienti) implementano il comportamento specificato dall'altro (chiamato fornitore). Questa relazione è rappresentata da una linea tratteggiata con una freccia vuota che punta dal cliente al fornitore. È possibile definire una realizzazione solo nei Class Diagram e nei Component Diagram.

Dependecy : Le dipendenze sono una forma debole di relazione, che indicano che una classe dipende da un'altra. La dipendenza esiste perché uno dei metodi di una classe ha come parametro o variabile locale un'altra classe.

Per tutte le relazioni è possibile definire la *Multiplicity*, ovvero il numero delle istanze che sono coinvolte nell'associazione.

3. Sequence Diagram

Una fase importante della modellazione di un sistema è la descrizione delle interazioni tra i processi; a questo proposito l'UML mette a disposizione i *Sequence Diagram*, o *diagrammi di sequenza*, che permettono di modellare la comunicazione tra un oggetto ed un altro in relazione al trascorrere del tempo. L'idea chiave è che le interazioni tra i processi avvengano seguendo un ordine ben preciso e che tale sequenza avvenga, nel tempo, dall'inizio alla fine.
Un Sequence Diagram si presenta come una serie di rettangoli (ognuno con un nome), che rappresentano gli oggetti che prendono parte all'interazione, da cui partono delle linee tratteggiate verso il basso, denominate "*linee della vita*" (*lifelines*). Le operazioni svolte vengono rappresentate da delle frecce orizzontali che collegano le lifeline degli oggetti coinvolti nell'interazione e vengono chiamate "*messaggi*". Questo tipo di diagramma consente di descrivere in maniera grafica dei semplici scenari di runtime.
Un esempio di Sequence Diagram è mostrato in figura 8 nella pagina successiva. Una lifeline viene disegnata con una linea tratteggiata lungo la quale si può trovare un piccolo rettangolo chiamato "*attivazione*" (*activation*) che rappresenta l'esecuzione di un'operazione di cui l'oggetto si fa carico mentre la durata dell'activation è rappresentata dalla lunghezza del rettangolo.
Un messaggio viene disegnato a partire dalla lifeline dell'oggetto da cui parte e arriva sulla lifeline dell'oggetto a cui è diretto; si può anche verificare il caso in cui un oggetto mandi un messaggio a se stesso, cioè un messaggio che parte e arriva alla stessa lifeline. Tale tipo di comportamento viene definito ricorsione.
Un messaggio può essere sincrono, ovvero l'oggetto attende che gli venga restituita una risposta al suo messaggio stesso prima di poter continuare con altre operazioni, o asincrono, se un oggetto non attende che gli venga inviata alcuna risposta prima di continuare con altre operazioni.
In un diagramma di sequenza lo scorrere del tempo viene fatto partire alla base di ogni oggetto per proseguire verso il basso. Ad esempio, un messaggio che si trovi più in alto nel diagramma (nella direzione verticale) rispetto ad un altro, si verificherà prima nel tempo. Tipicamente viene utilizzo il simbolo dell'actor per rappresentare l'inizio della sequenza, anche se questo simbolo non appartiene a quelli propri dei Sequence Diagram, ma viene "preso in prestito" dai Use Case Diagram.
I Sequence Diagram offrono la possibilità di descrivere istruzioni condizionali e cicli while, che nel diagramma sono rappresentati da un riquadro che racchiude la condizione di iter, racchiusa tra parentesi quadre, e i messaggi da iterare.

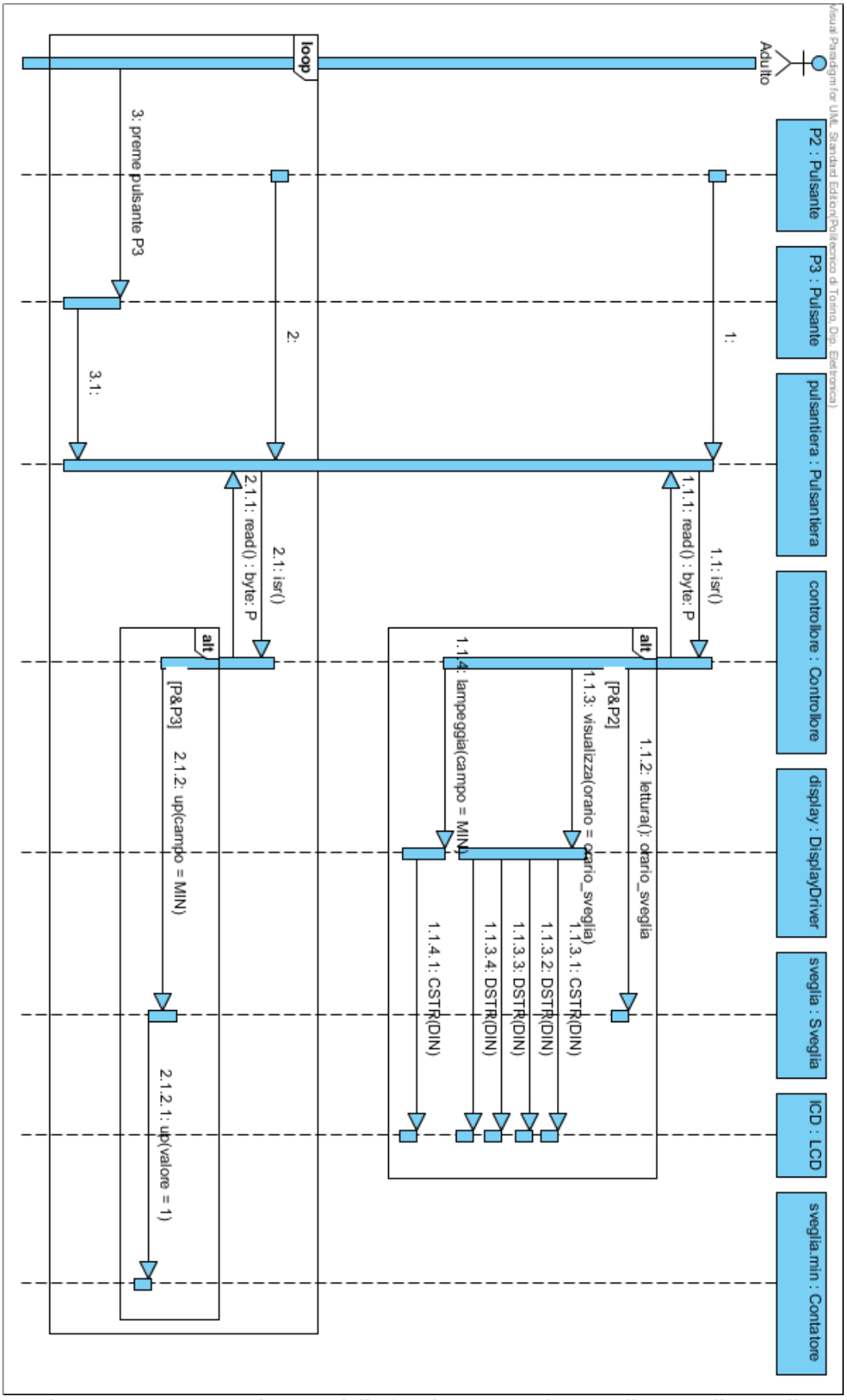


Figura 8 - Sequence Diagram della funzione "Regola ora" di una radiosveglia

Capitolo 3

Cenni teorici

In questo capitolo verranno illustate tutte le caratteristiche del satellite AraMiS che sono state utilizzate per la realizzazione del progetto 1B43 Housekeeping Management. Inoltre, saranno spiegati i concetti matematici di matrice pseudo inversa e di frazione continua, che sono stati utilizzati per la realizzazione di alcuni metodi.

Per lo sviluppo del modulo 1B43 Housekeeping Management si è scelto di considerare il satellite AraMiS nella sua configurazione base, ovvero un cubo composto da sei tile per lato, come già descritto nel capitolo 1. I parametri relativi alla configurazione base del satellite sono stati salvati in un'apposita classe nel progetto *Bk1A1_Common_Codes* e riutilizzati in questo progetto; in seguito verranno descritti brevemente questi valori (per una descrizione più dettagliata si rimanda alla sezione relativa alla classe *mechanicalConfiguration* nel prossimo capitolo).
Ad ogni tile è stato associato un indirizzo, in modo da permette al computer di bordo di comunicare con le tile in modo univoco; gli indirizzi sono definiti in un vettore la cui dimensione dipende, com'è facile immaginare, dal numero di tile. Il numero delle tile influenza molti altri parametri, come ad esempio i vettori che indicano la presenza dei sensori o le matrici dei coseni direttori, di cui verrà parlato in seguito, in quanto sono utilizzate per il calcolo delle matrici pseudo inverse.

Le matrici dei coseni direttori descrivono l'orientamento del sistema di riferimento della tile rispetto a quello posto al centro del satellite, come mostrato in figura 9.

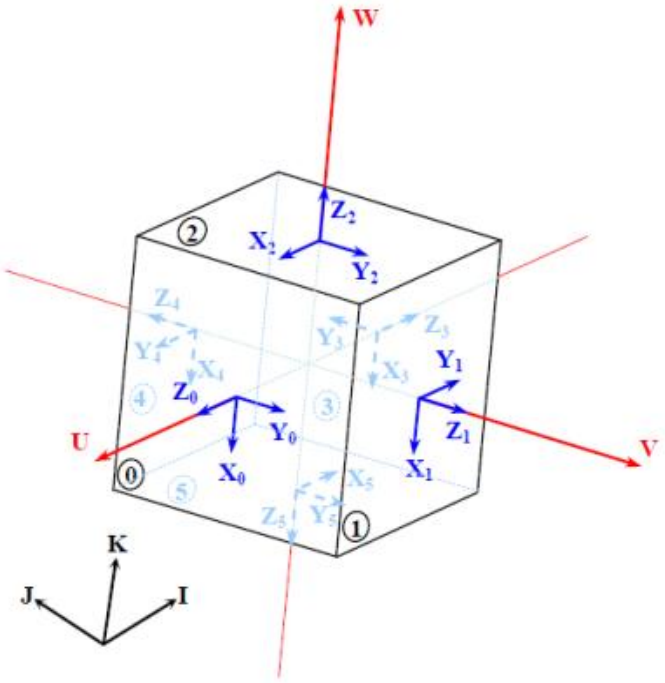


Figura 9 - Sistema di riferimento di un satellite cubico

Di seguito sono scritte le matrici dei coseni direttori delle tile nel caso di un satellite cubico, come quello in figura 9; si ricorda che per tutta la trattazione si è supposto che la tile 5 sia una TLC tile.

$$T_0 = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad T_1 = \begin{bmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_3 = \begin{bmatrix} 0 & 0 & -1 \\ 0 & -1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \quad T_4 = \begin{bmatrix} 0 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \quad T_5 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

1. Calcolo delle matrici pseudo-inverse

Le matrici pseudo-inverse sono utilizzate, nel progetto 1B43 Housekeeping Management, per calcolare grandezze vettoriali come il campo magnetico terrestre o lo spin del satellite.
La pseudo-inversa utilizzata per il calcolo del vettore campo magnetico terrestre, ad esempio, è una matrice *3*2T*, dove *T* indica il numero di tile equipaggiate con i magnetometri; questa è costruita a partire dalle matrici dei coseni direttori di ciascuna tile.
Come si può intuire, la pseudo-inversa va ricalcolata ogniqualvolta la configurazione del satellite varia, ovvero quando qualche sensore smette di funzionare. Per fortuna questi eventi non sono molto frequenti.

In algebra lineare si definisce matrice pseudo-inversa A^+ la generalizzazione della matrice inversa nel caso in cui la matrice *A* non sia quadrata; il tipo di matrice pseudo-inversa più conosciuto è quella di Moore-Penrose [15].
La matrice pseudo-inversa, chiamata anche inversa generalizzata, è definita e unica per tutte le matrici costituite de numeri reali o complessi. Questo tipo di matrici hanno le seguenti proprietà:

- Se gli elementi di *A* sono numeri reali, allora lo sono anche quelli di A^+;
- Se *A* è invertibile, la pseudo-inversa e l'inversa coincidono: $A^+ = A^{-1}$;
- La pseudo-inversa di una matrice nulla è la sua trasposta;
- La pseudo-inversa di una pseudo-inversa è la matrice originaria $(A^+)^+ = A$;
- La pseudo-inversa di uno scalare è il suo reciproco $(\alpha A)^+ = \alpha^{-1} A^+$ *per* $\alpha \neq 0$;

Un metodo computazionalmente semplice e accurato per calcolare le matrici pseudoinverse è la Singular Value Decomposition (SVD). La *Decomposizione ai Valori Singolari* è un tipo di fattorizzazione di una matrice reale o complessa, basata sull'utilizzo di autovalori e autovettori, espressa nella forma

$$A = U\Sigma V^*$$

dove *U* è una matrice unitaria *m*m*, Σ è una matrice diagonale non negativa *m*n* e *V** (la trasposta coniugata di *V*) è una matrice unitaria *n*n*.
Gli elementi posti sulla diagonale di Σ sono chiamati *valori singolari* di *A*, mentre le *m* colonne di *U* e le *n* colonne di *V* sono chiamate *vettori singolari sinistri* e *vettori singolari destri* di *A*, rispettivamente.
Una volta decomposta la matrice *A* tramite il metodo SVD, la sua pseudo-inversa è calcolabile come

$$A^+ = V\Sigma^+ U^*$$

dove Σ^+ è la pseudo-inversa di Σ, che si ottiene sostituendo ogni elemento non nullo con il suo reciproco e trasponendo la matrice risultante.

2. Approssimazione razionale di numeri reali

La maggior parte di microcontrollori non è equipaggiata con un'unità aritmetico-logica a virgola mobile; di conseguenza questo tipo di operazioni viene svolto da una routine assembler che causa un calo delle performance. Questo è uno dei motivi per cui si cerca di evitare l'uso di variabili floating point, senza contare che un aumento nel tempo di attività del processore causa anche un aumento dei consumi, che per alcune applicazioni non può essere tollerato.
Nel progetto AraMiS esistono due tipi di dato che permettono di evitare l'uso delle variabili *float*; questi tipi sono *t_floatFactorByte* e *t_floatFactorShort*. La classe *t_floatFactorByte* (e in maniera analoga *t_floatFactorShort*) è composta da due attributi, M e D, che rappresentano il numeratore e il denominatore della frazione utilizzata per approssimare un determinato valore in floating point.

L'algoritmo ha come obiettivo la conversione di una variabile float in un rapporto tra 2 interi, ovvero sotto forma di frazione. Questa operazione viene eseguita sfruttando il concetto matematico di frazione continua [6].
Le frazioni continue sono delle espressioni del tipo:

$$x = a_0 \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \mathrm{K}}}}$$

dove a_0 è un intero e tutti gli altri numeri a_n sono interi positivi.
Poiché le frazioni continue hanno una scrittura poco pratica, molto spesso vengono usate delle abbreviazioni. Una delle notazioni più usate è la seguente

$$[a_0; a_1, a_2 a_3]$$

dove a_0, a_1, a_2, a_3 sono i termini della frazione continua. È consuetudine sostituire la prima virgola con un punto e virgola.
Le frazioni continue hanno alcune proprietà interessanti, ad esempio:

- La frazione continua di un numero è finita se e solo se il numero è razionale;
- La frazione continua dei numeri razionali semplici è breve;
- La frazione continua dei numeri irrazionali è unica;
- La frazione continua di un numero razionale è quasi unica: ci sono esattamente due frazioni continue per ogni numero razionale, che sono uguali, tranne per il fatto che una termina con *...a,1]* e l'altra con *...a + 1]*;
- Troncando la frazione continua di un numero *x* si ottiene un'approssimazione razionale di *x* che in un certo senso è la "migliore possibile".

In particolare l'ultima proprietà è molto importante, ed è uno dei punti fondamentali su cui basa l'algoritmo. Infatti questa garantisce che l'approssimazione sia sempre quella più vicina al numero reale.

Il calcolo delle frazioni continue è un processo iterativo in due fasi: si prende la parte intera del numero e si fa il reciproco di quella frazionaria; l'algoritmo si ferma solo quando la parte frazionaria è uguale a 0.
Nella tabella seguente è mostrata l'approssimazione del numero 3,245. Nella prima colonna sono elencati i coefficienti a_n della frazione continua, nella seconda viene ricavata la parte frazionaria e nella terza si calcola il reciproco. La matrice continua del numero 3,245 risulta essere [3; 4, 12, 4].

Coefficienti	Parte Frazionaria	Reciproco
3	3,245 - 3 = 0,245	$\frac{1}{0.245} = 4.082$
4	4,082 - 4 = 0,082	$\frac{1}{0.082} = 12.250$
12	12,250 - 12 = 0,250	$\frac{1}{0.250} = 4.000$
4	4,000 - 4 = 0,000	

Una volta ottenuti i coefficienti della frazione continua, è possibile ricavare un numero razionale utilizzando la seguente formula:

$$\frac{N_n}{D_n} = \frac{a_n N_{n-1} + N_{n-2}}{a_n D_{n-1} + D_{n-2}}$$

con valori iniziali

$$N_{-1} = 0 \qquad D_{-1} = 1$$
$$N_0 = 1 \qquad D_0 = 0$$

Ad esempio, i primi quattro risultati che si ottengono per una frazione continua sono

$$\frac{a_0}{1} \quad \frac{a_0 a_1 + 1}{a_1} \quad \frac{a_2(a_0 a_1 + 1) + a_0}{a_2 a_1 + 1} \quad \frac{a_3(a_2(a_0 a_1 + 1) + a_0) + (a_0 a_1 + 1)}{a_3(a_2 a_1 + 1) + a_1}$$

che, sostituendo i valori [3; 4, 12, 4]

$$\frac{3}{1} = 3 \quad \frac{13}{4} = 3.2500 \quad \frac{159}{49} = 3.2449 \quad \frac{649}{200} = 3.2450$$

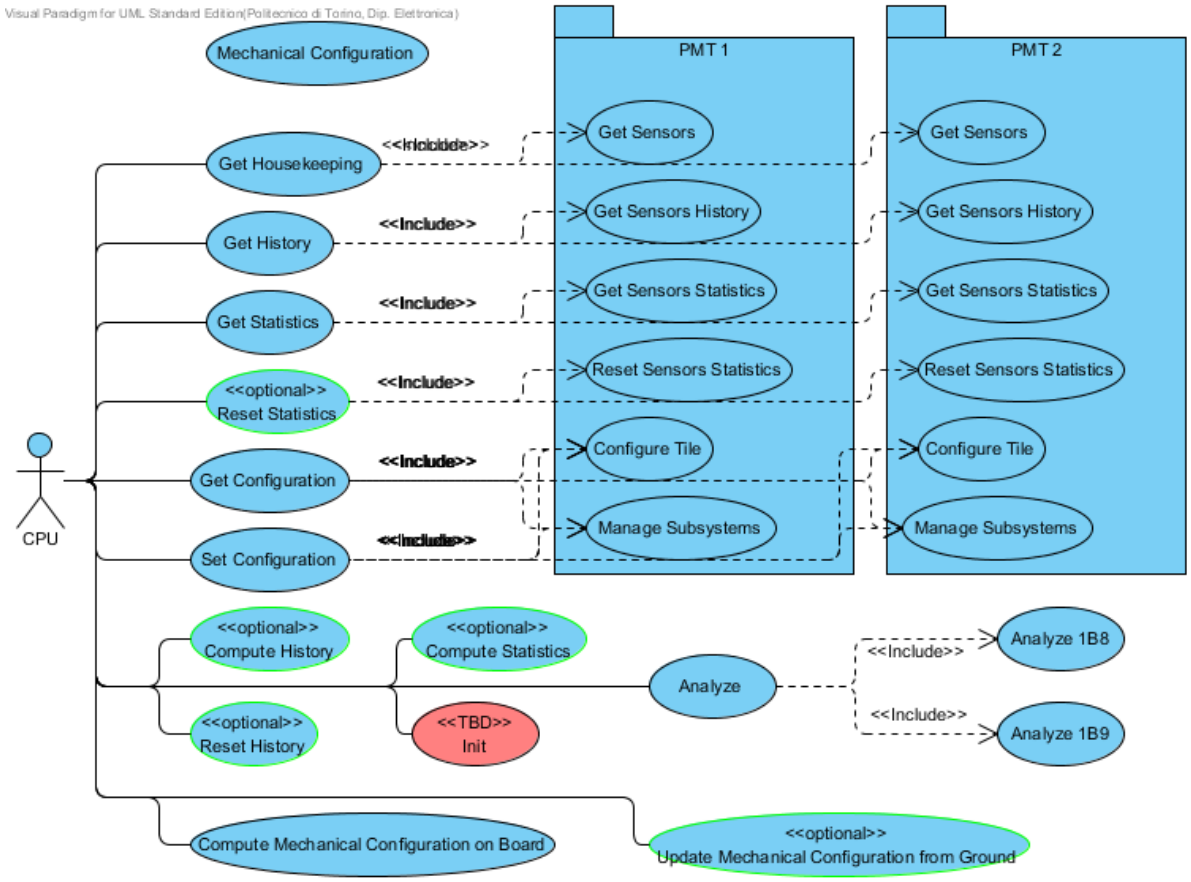
restituiscono il valore 3,245.
Purtroppo, questo algoritmo non può essere utilizzato con dei numeri in virgola mobile, in quanto dei piccoli errori nella parte frazionaria possono creare grandi differenze nel termine successivo per via dell'inversione. In realtà viene utilizzato ugualmente in quanto sono state inserite alcune "protezioni", anche se non è la definizione più corretta, che servono a limitare questo problema. L'algoritmo come prima cosa tronca il dato da convertire alla terza cifra decimale, in modo da eliminare tutte le cifre decimali spurie e non necessarie, e successivamente la costruzione della frazione è limitata al massimo valore possibile per i tipi di dato di numeratore e denominatore.

Capitolo 4

1B43 Housekeeping Management

In questo capitolo verranno mostrati e spiegati tutti i diagrammi UML relativi al modulo 1B43 Housekeeping Management.
Il primo diagramma mostrato sarà quello dei casi d'uso, che avrà lo scopo di introdurre tutte le funzioni che il modulo è in grado di svolgere.
A questo punto verrà descritto il cuore del progetto, i Class Diagram; i diagrammi che verranno illustrati sono due: Bk1B43 Housekeeping Management e Bk1B43-K GSE. In particolare il secondo è un diagramma di supporto al primo, in quanto viene utilizzato solo per la simulazione del sistema.
Alcune operazioni saranno correlate da dei Sequence Diagram in modo da spiegare il funzionamento di queste funzioni chiave più chiaramente.
Durante la lettura si può notare che la parte relativa alle tile TLC è piuttosto carente. Ciò è dovuto al fatto che lo sviluppo di questi moduli è ancora alle prime fasi, se paragonato a quello delle tile 1B8 (PMT). Si è deciso di creare comunque gli oggetti per le tile 1B9 per facilitare il compito di chi andrà a completare questa parte del progetto, quando le tile TLC saranno più mature.

Use Case Diagram



In questa sezione viene descritto il diagramma dei casi d'uso relativo al modulo 1B43 Housekeeping Management. Il diagramma, mostrato nella figura alla pagina precedente, illustra tutte le funzionalità del sistema sotto forma di casi d'uso.
Nel diagramma alcuni use case sono contenuti in un package (rappresentato da un rettangolo azzurro); ciò significa che questi appartengono ad un altro modulo del progetto AraMiS. Molti dei casi d'uso del modulo 1B43 Housekeeping Management utilizzano delle funzionalità di altri progetti, in particolare quando è coinvolta la comunicazione tra le tile.

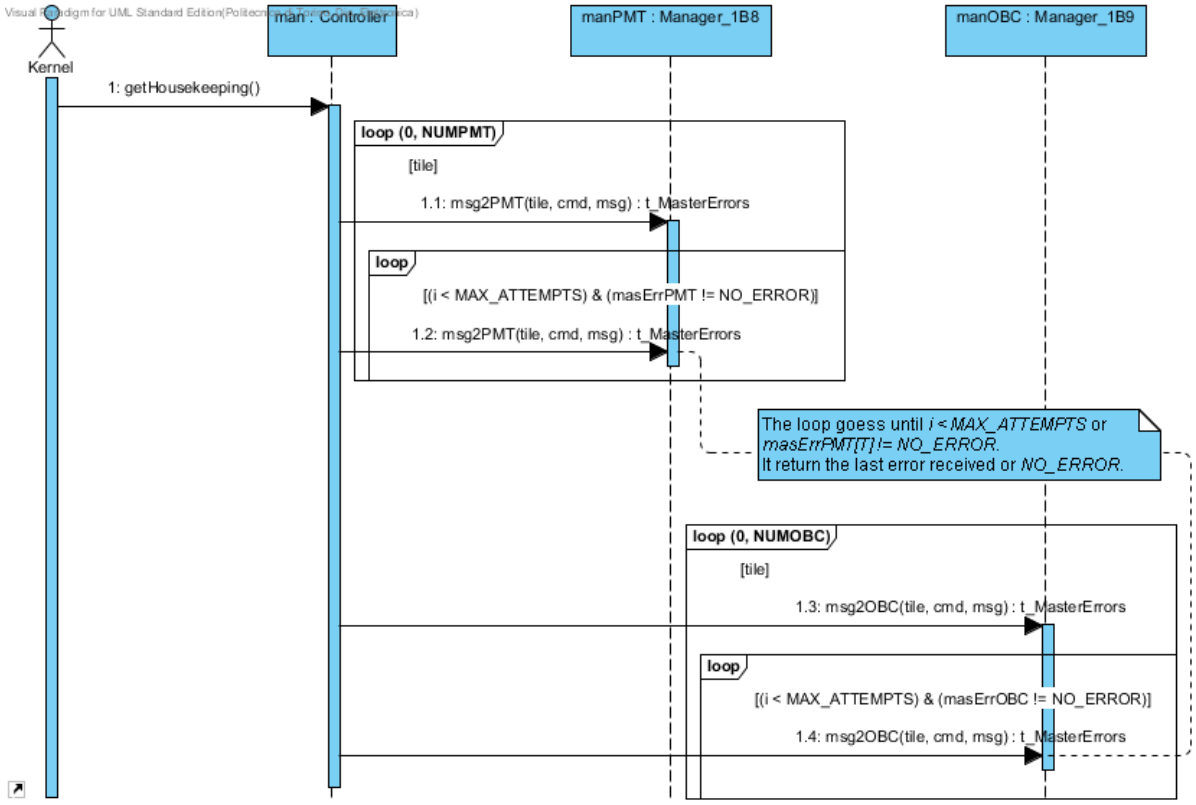
1.1. Actor - CPU

Name	Value
Name	CPU
Documentation	The **CPU** actor is a processor willing to communicate (exchange data and commands) with the **System**. The **CPU** communicates with the **System** via a 4-wires SPI-like interface plus a few other control and status signals. The **CPU** is the SPI master, while the **System** is an SPI slave. Slave addressing is via an individual chip select (**CS**) pin for each slave. The **CPU**, via the SPI interface, can at least: 1. send **Designer**-defined messages to the **System** 2. receive **Designer**-defined messages the **System** 3. acquire **Designer**-defined housekeeping information from the **System** (e.g. internal voltages, currents, temperatures) set the module to sleep mode or wake it up.

1.2. Use Case - Get Housekeeping

Name	Value
Name	Get Housekeeping
Stereotypes	UseCase
Documentation	Collect Housekeeping data from all tiles in the system, by calling the Get Sensors use case. Housekeeping data for PMT (1B8) tiles are collected into vectors hkPMT of class Manager_1B8. Housekeeping data for OBC (1B9) tiles are collected into vectors hkOBC of class Manager_1B9. This use case is invoked by calling operation getHousekeeping of class Controller.

Sequence Diagram - getHousekeeping

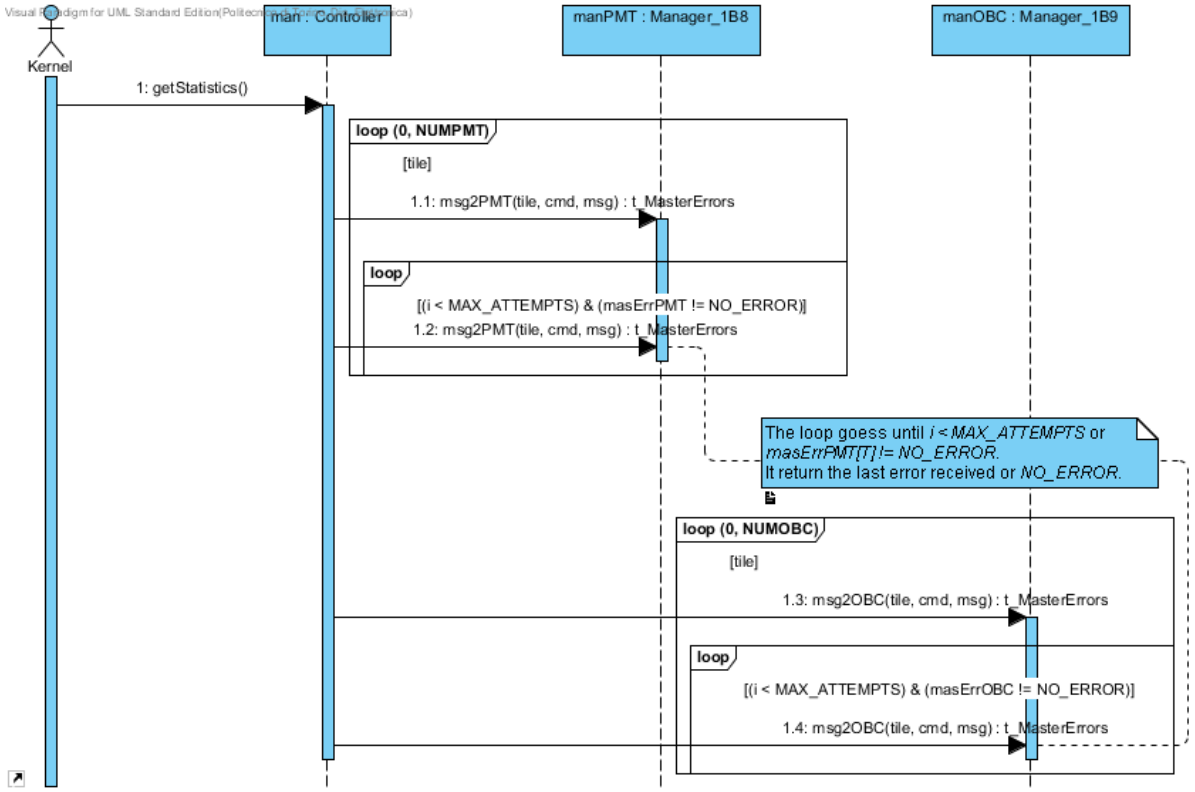


Questo Sequence Diagram illustra il funzionamento del metodo getHousekeeping della classe Controller.
Questa operazione viene utilizzata per acquisire il vettore di housekeeping da tutte le tile presenti nel satellite. La comunicazione con ogni tile viene tentata fino a quando non sono stati riscontrati errori e comunque fino ad un massimo di 5 volte. La scansione viene fatta a partire dalle PMT tile per poi continuare con le TLC tile; per la lettura delle telemetrie viene utilizzato il comando CMD_GET_HOUSEKEEPING della classe t_Commands.

1.3. Use Case - Get Statistics

Name	Value
Name	Get Statistics
Stereotypes	UseCase
Documentation	Collect Statistics from all tiles in the system, by calling Get Sensors Statistics use case. Statistics for PMT (1B8) tiles are collected into vectors statPMT. Statistics for OBC (1B9) tiles are collected into vectors statOBC. This use case is invoked by calling operation getStatistics of class Controller.

Sequence Diagram - getStatistics

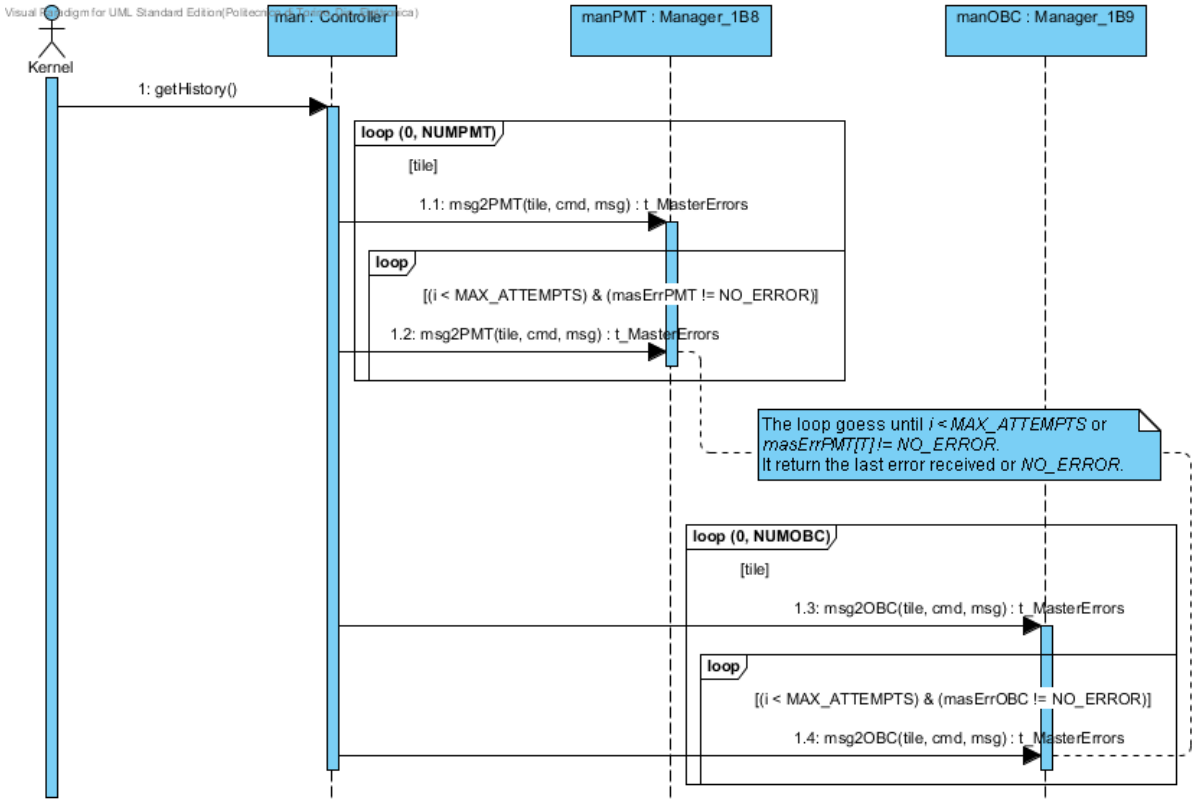


Questo Sequence Diagram illustra il funzionamento del metodo getStatistics della classe Controller.
Questa operazione viene utilizzata per acquisire il vettore delle statistiche da tutte le tile presenti nel satellite. La comunicazione con ogni tile viene tentata fino a quando non sono stati riscontrati errori e comunque fino ad un massimo di 5 volte. La scansione viene fatta a partire dalle PMT tile per poi continuare con le TLC tile; per la lettura delle statistiche viene utilizzato il comando CMD_GET_STATISTICS della classe t_Commands.

1.4. Use Case - Get History

Name	Value
Name	Get History
Stereotypes	UseCase
Documentation	Collect History matrix from all tiles in the system, by calling Get Sensors History use case. The size of history matrix depends by DEPHisPMT (or DEPHisOBC) and LENHisPMT (or LENHisOBC). History data for PMT (1B8) tiles are collected into vectors hisPMT. History data for OBC (1B9) tiles are collected into vectors hisOBC. This use case is invoked by calling operation getHistory of class Controller.

Sequence Diagram - getHistory



Questo Sequence Diagram illustra il funzionamento del metodo getHistory della classe Controller.
Questa operazione viene utilizzata per acquisire la matrice storia da tutte le tile presenti nel satellite. La comunicazione con ogni tile viene tentata fino a quando non sono stati riscontrati errori e comunque fino ad un massimo di 5 volte. La scansione viene fatta a partire dalle PMT tile per poi continuare con le TLC tile; per la lettura delle matrici storia viene utilizzato il comando CMD_GET_HISTORY della classe t_Commands.

1.5. Use Case - Reset Statistics

Name	Value
Name	Reset Statistics
Stereotypes	UseCase, optional
Documentation	Resets Statistics for each tile in the system by calling Reset Sensors Statistics. This use case is invoked by calling operation resetStatistics of class Controller.

1.6. Use Case - Reset History

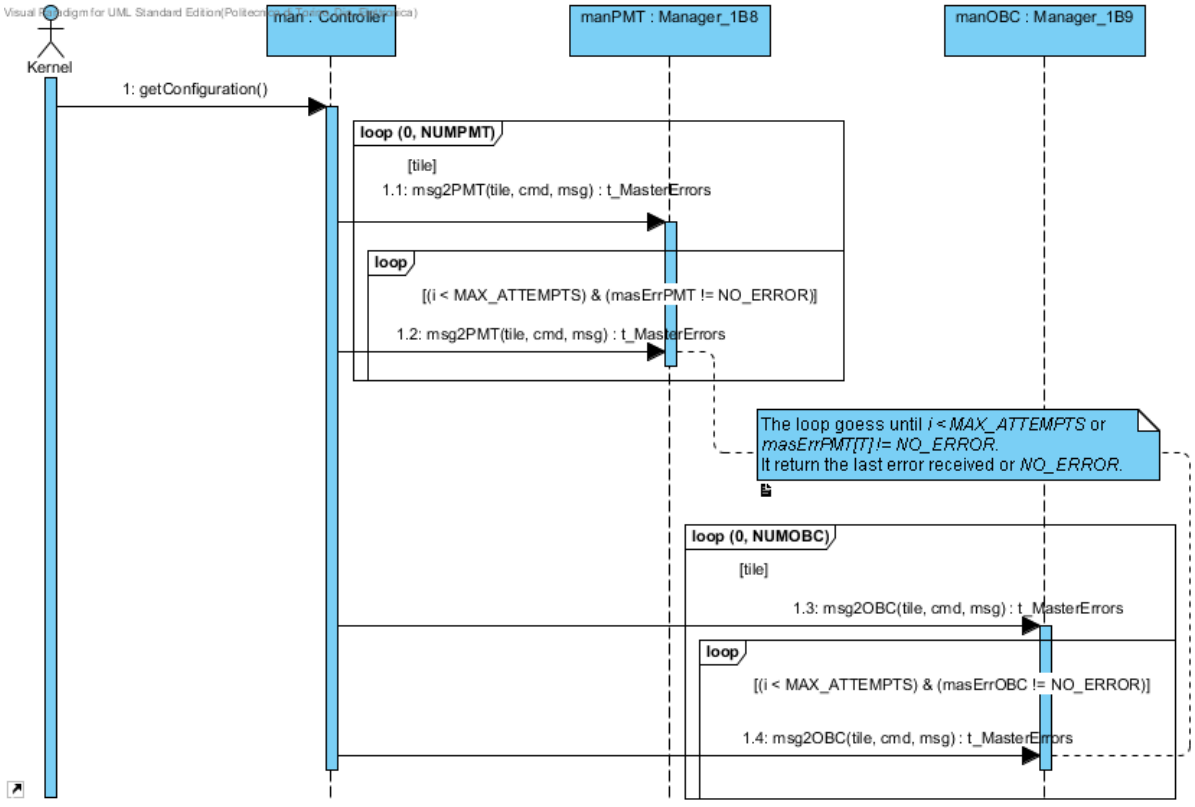
Name	Value
Name	Reset History
Stereotypes	UseCase, optional

Documentation	Clears History data for each tile in the system by calling Reset Sensors Statistics. This use case is invoked by calling operation clearHistory of class Controller.

1.7. Use Case - Get Configuration

Name	Value
Name	Get Configuration
Stereotypes	UseCase
Documentation	Collect Configuration data from all tiles in the system, by calling Configure Tile and Manage Subsystems. Housekeeping data for PMT (1B8) tiles are collected into vectors confPMT. Housekeeping data for OBC (1B9) tiles are collected into vectors confOBC. This use case is invoked by calling operation getConfiguration of class Controller.

Sequence Diagram - getConfiguration



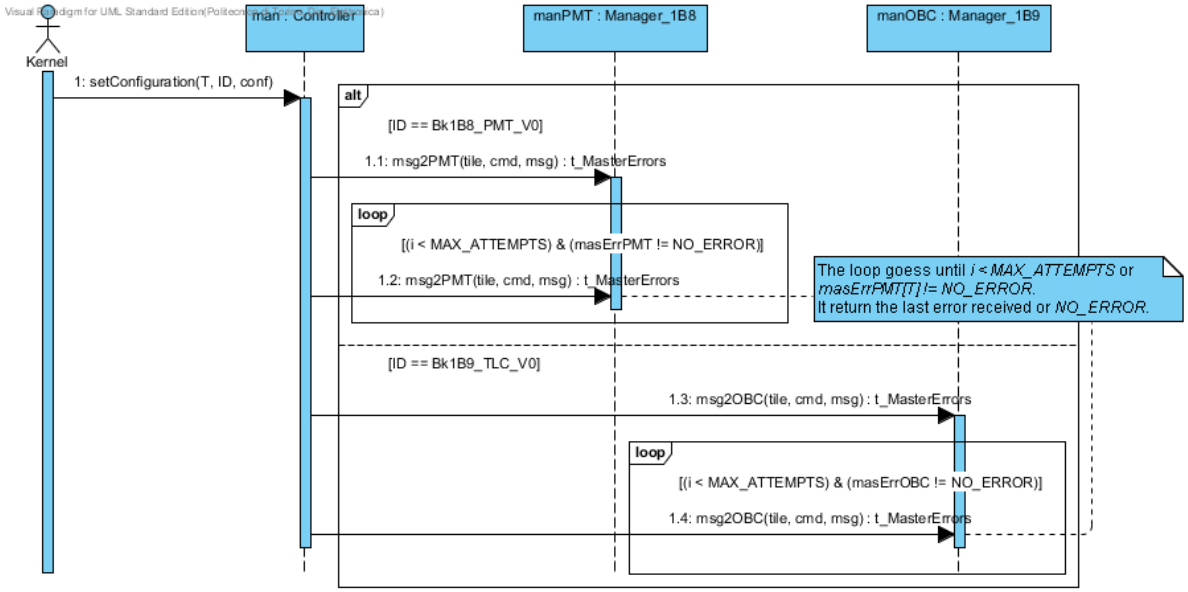
Questo Sequence Diagram illustra il funzionamento del metodo getConfiguration della classe Controller.

Questa operazione viene utilizzata per acquisire il vettore delle configurazioni da tutte le tile presenti nel satellite. La comunicazione con ogni tile viene tentata fino a quando non sono stati riscontrati errori e comunque fino ad un massimo di 5 volte. La scansione viene fatta a partire dalle PMT tile per poi continuare con le TLC tile; per la lettura delle configurazioni viene utilizzato il comando CMD_GET_CONFIGURATION della classe t_Commands.

1.8. Use Case - Set Configuration

Name	Value
Name	Set Configuration
Stereotypes	UseCase
Documentation	Send Configuration data to a tile in the system, by calling Configure Tile and Manage Subsystems. This use case is invoked by calling operation setConfiguration of class Controller.

Sequence Diagram - setConfiguration



Questo Sequence Diagram illustra il funzionamento del metodo setConfiguration della classe Controller.
Questa operazione viene utilizzata per inviare un nuovo vettore di configurazione ad una tile specifica. La comunicazione viene tentata fino a quando non sono stati riscontrati errori e comunque fino ad un massimo di 5 volte. È necessario specificare il tipo di tile (tra quelli presenti nella classe ID_Codes) a cui la nuova configurazione è destinatata, oltre ovviamente al numero della tile di quel tipo. Per questa operazione viene utilizzato il comando CMD_SET_CONFIGURATION della classe t_Commands.

1.9. Use Case - Compute Statistics

Name	Value
Name	Compute Statistics

Stereotypes	UseCase, optional
Documentation	Update the Statistics vector, for each tile in the system, based on the last housekeeping vector acquired by calling Get Housekeeping use case. Statistics for PMT (1B8) tiles are collected into vectors statPMT. Statistics for OBC (1B9) tiles are collected into vectors statOBC. This use case is invoked by calling operation computeStatistics of class Controller.

1.10. Use Case - Compute History

Name	Value
Name	Compute History
Stereotypes	UseCase, optional
Documentation	Update an history matrix, for each tile in the system, based on the last DEPHisPMT (or DEPHisOBC) housekeeping vector acquired by calling Get Housekeeping use case. History data for PMT (1B8) tiles are collected into vectors hisPMT. History data for OBC (1B9) tiles are collected into vectors hisOBC. This use case is invoked by calling operation computeHistory of class Controller.

1.11. Use Case - Analyze

Name	Value
Name	Analyze
Stereotypes	UseCase
Documentation	Computes a set of statistics, stored in the attributes of classes Result_1B8 and Result_1B9, which give a representation of the global status of the system. For more detatils, see Analyze 1B8 and Analyze 1B9 use cases documentation. This use case is invoked by calling operation analyze of class Controller.

1.12. Use Case - Analyze 1B8

Name	Value

Name	Analyze 1B8
Stereotypes	UseCase
Documentation	Analizes the status of all PMT (1B8) tiles using data collected by calling Get Housekeeping. This use case is invoked by calling operation analyzePMT of class Manager_1B8. analyzePMT computes the following values: Magnetic Field, by calling operation getMagField of class Manager_1B8. Spin of the satellite, by calling operation getSpin of class Manager_1B8. The position of the satellite in relation of a celestial object, by calling operation getPosition of class Manager_1B8. The average of a specific field. The standard deviation of a specific field, by calling operation StdDev of class Manager_1B8. The maximum and the minumum value of a specific field, by calling operation minMAX of class Manager_1B8. Values are saved in class Result_1B8.

1.13. Use Case - Analyze 1B9

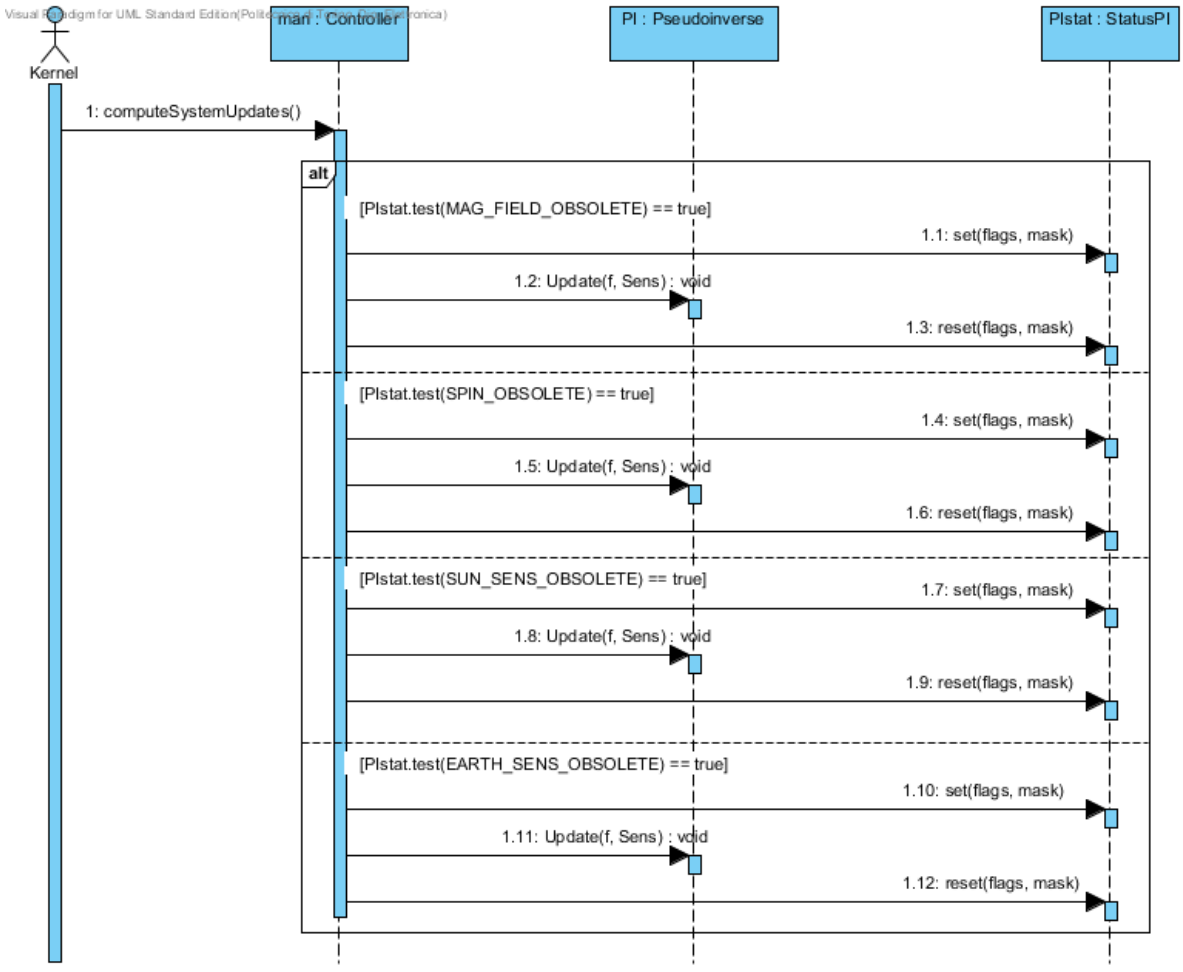
Name	Value
Name	Analyze 1B9
Stereotypes	UseCase
Documentation	Analizes the status of all OBC (1B9) tiles using data collected by calling Get Housekeeping. This use case is invoked by calling operation analyzeOBC of class Manager_1B9. analyzeOBC computes the following values: // TBD Values are saved in class Result_1B9.

1.14. Use Case - Compute Mechanical Configuration on Board

Name	Value
Name	Compute Mechanical Configuration on Board

Stereotypes	UseCase
Documentation	Processes the matrices of direction cosines, contained in class mechanicalConfiguration, of all PMT (1B8) tiles according to the state of the system. The calculation results will be used for the analysis of the housekeeping vectors. Results are collected in : Bpi ; Zpi ; Spi ; Epi of class Pseudoinverse. This use case is invoked by calling operation computeSystemUpdates of class Controller.

Sequence Diagram - computeSystemUpdate



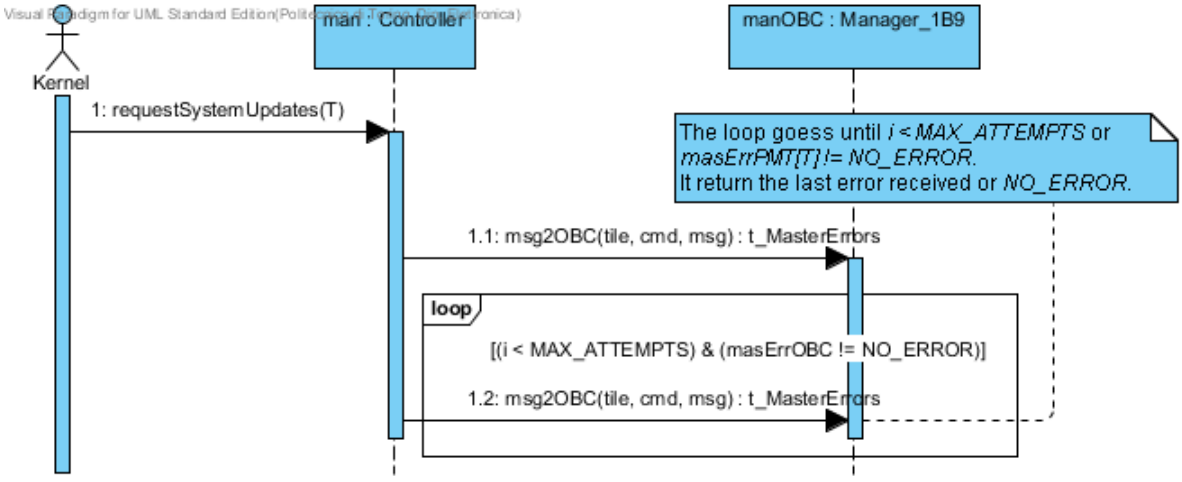
Questo Sequence Diagram illustra il funzionamento del metodo computeSystemUpdate della classe Controller.
Questa operazione viene utilizzata per ricalcolare le matrici di configurazione utilizzate per l'analisi dello stato del sistema. Queste matrici vengono aggiornate solo se ci sono delle modifiche nella configurazione del sistema; la tecnica utilizzata per il calcolo è spiegata nel

capitolo seguente. Data la complessità del calcolo, prima di iniziare l'operazione viene settato un flag per impedire ad altre operazioni di utilizzare la matrice che si sta aggiornando; una volta terminato l'update i flag vengono resettati.

1.15. Use Case - Update Mechanical Configuration from Ground

Name	Value
Name	Update Mechanical Configuration from Ground
Stereotypes	UseCase, optional
Documentation	Sends to the tiles OBC (1B9) a request of renovation of the configuration. The received data are collected in : Bpi ; Zpi ; Spi ; Epi of class Pseudoinverse. This use case is invoked by calling operation requestSystemUpdates of class Controller.

Sequence Diagram - requestSystemUpdate



Questo Sequence Diagram illustra il funzionamento del metodo requestSystemUpdate della classe Controller.
Questa operazione viene utilizzata per richiedere alla Ground Station un aggiornamento delle matrici di configurazione utilizzate per l'analisi dello stato del sistema. La funzione invia ad una TLC tile lo stato del satellite; sarà compito della tile inviare la richiesta alla stazione di terra. La comunicazione viene tentata fino a quando non sono stati riscontrati errori e comunque fino ad un massimo di 5 volte. Per l'invio dello stato del satellite viene utilizzato il comando CMD_WRITE_DATA_0 della classe t_Commands.

1.16. Use Case - Mechanical Configuration

Name	Value
Name	Mechanical Configuration
Stereotypes	UseCase
Documentation	Satellite configuration is defined in class mechanicalConfiguration (from package 1A), which contains: number of tiles, tile orientation and other relevant parameters.

I seguenti casi d'uso non sono originari del modulo 1B43, ma sono appartengono ad altri moduli, in particolare dell'1B8 Power Management Tile e dell'1B45 Subsystem Serial Data Bus.

1.17. Use Case - Get Sensors

Name	Value
Name	Get Sensors
Stereotypes	UseCase
Documentation	This use case gathers all functions related with acquiring sensor measurements from 1B8 Power Management Tile. Note that sensors are acquired at a fixed rate (see use case Housekeeping) independently of the call to any Get Sensors use cases. Not also that all Get Sensors use cases return all the housekeeping measurements, so it is suggested that the OBC calls the Get Sensors use case only once and saves values into a local copy for all its users. Immediately returns the last stored value of a number of housekeeping parameters (voltages, currents, temperatures, etc.). Among them: Solar Panels voltage, current, temperature Battery pack voltage, current, temperature Internal voltages, currents, temperatures Status of power switches Number of latchups, SEU, watchdog Attitude measurements Status word(s) Should also contain an indication of how long ago it has been measured

1.18. Use Case - Get Sensors Statistics

Name	Value
Name	Get Sensors Statistics
Stereotypes	UseCase
Documentation	Returns statistics (min, max, average, std deviation) of a few user-selected housekeeping parameter from 1B8 Power Management Tile. Note that sensors are acquired at a fixed rate (see use case Housekeeping) independently of the call to any Get Sensors Statistics use cases. Not also that Get Sensors Statistics use case returns history of all the housekeeping measurements, so it is suggested that the OBC calls the Get Sensors Statistics use case only once and saves values into a local copy for all its users. This Use Case makes use of Get Module Housekeeping Statistics use case of 1B45 package.

1.19. Use Case - Get Sensors History

Name	Value
Name	Get Sensors History
Stereotypes	UseCase
Documentation	Returns an history record of a few user-selected housekeeping parameter from 1B8 Power Management Tile. Note that sensors are acquired at a fixed rate (see use case Housekeeping) independently of the call to any Get Sensors History use cases. Not also that Get Sensors History use case returns history of all the housekeeping measurements, so it is suggested that the OBC calls the Get Sensors History use case only once and saves values into a local copy for all its users. This Use Case makes use of Get Module Housekeeping History use case of 1B45 package.

1.20. Use Case - Reset Sensors Statistics

Name	Value
Name	Reset Sensors Statistics
Stereotypes	UseCase
Documentation	Resets statistics records and statistics of all housekeeping parameters. This Use Case makes use of Reset Module Statistics use case of

	1B45 package.

1.21. Use Case - Configure Tile

Name	Value
Name	Configure Tile
Stereotypes	UseCase
Documentation	This use case gathers all functions related with configuring the 1B8 Power Management Tile and in particular: "detaching" a subsystem whenever it is identified as faulty. "attaching" a subsystem in case it has been improperly detached. A subsystem is considered "attached" if it is physically present on the tile (see flags of (model element not found)(model element not found)) and it has not been detached (see corresponding use cases). If the wheel is attached and enabled, any further action of reaction wheel will be carried on. These operation are risky therefore they should be done under ground control (Central Mission Controller actor). This Use Case makes use of either Set Module Configuration or Reset Module Configuration use case of 1B45 package, with the corresponding bit of configuration word ((model element not found)). The difference between Attach Wheel and Enable Xxx is that the former has effects only is the wheel is physically present on the tile, while the latter has effects only if the wheel is attached, therefore the former enables the latter. The Attach Wheel use case is intended only in rare occasions, to recover from an erroneous Detach Wheel operation.

1.22. Use Case - Manage Subsystems

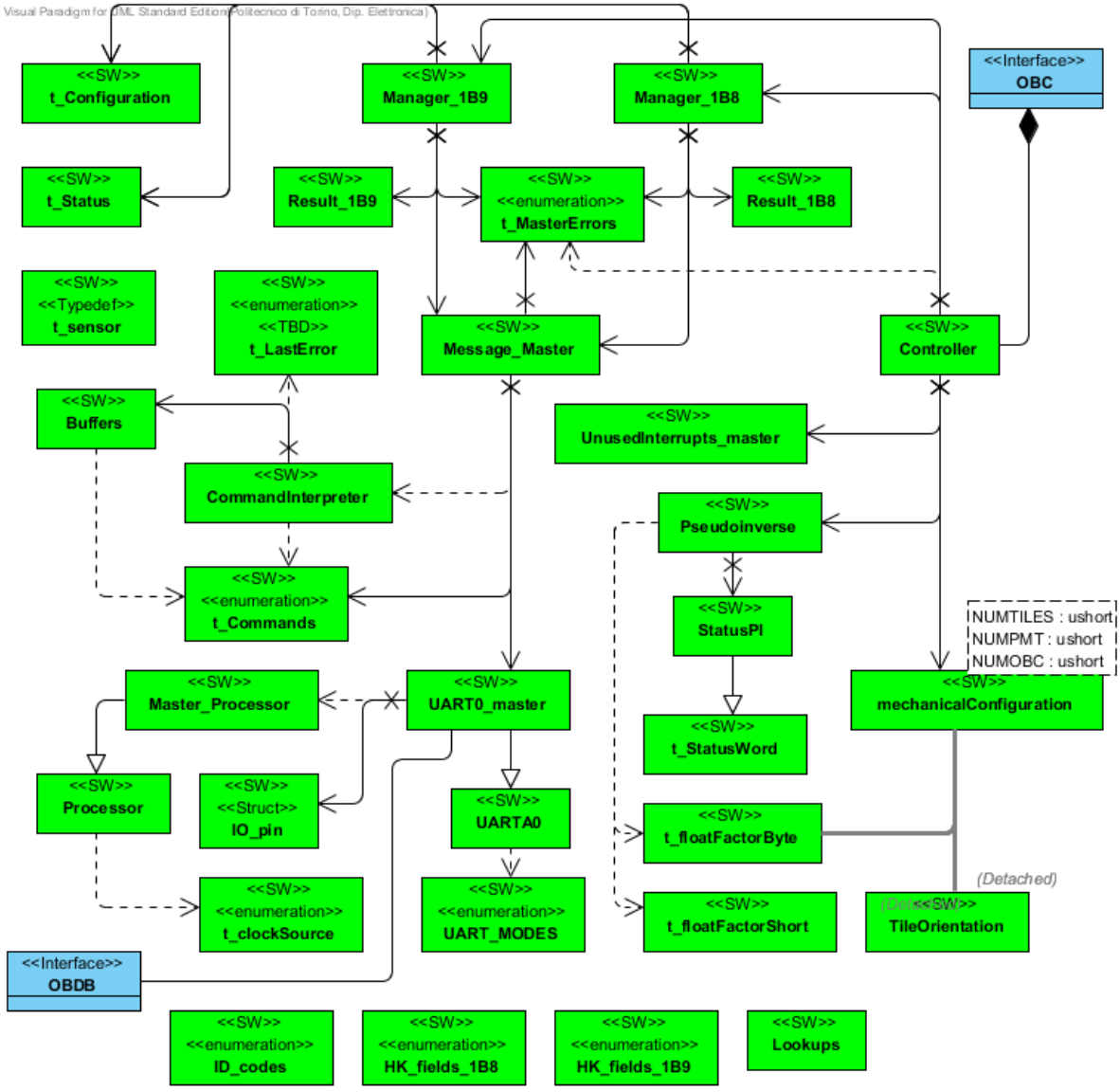
Name	Value
Name	Manage Subsystems
Stereotypes	UseCase
Documentation	This use case gathers all functions related with managing each individual subsystem of the 1B8 Power Management Tile and in particular: "enabling" each subsystem to enable all its functions; "disabling" each subsystem so that any further request will not be honored. Disabling a subsystem is intended to reduce power dissipation to a minimum whenever a subsystem is not used.

1.23. Use Case - Mechanical Configuration

Name	Value
Name	Mechanical Configuration
Stereotypes	UseCase
Documentation	Satellite configuration is defined in class mechanicalConfiguration (from package 1A), which contains: number of tiles, tile orientation and other relevant parameters.

Class Diagram

2. Bk1B43 Housekeeping Management



Il Class Diagram Bk1B43 Housekeeping Management è probabilmente la parte più complessa dell'intero progetto. Questo diagramma definisce la struttura del sistema e il modo in cui i casi d'uso vengono implementati.

Per ognuna delle classi presenti nel diagramma in figura è presente la descrizione delle proprietà dell'oggetto, oltre a quella dei metodi e degli attributi. Date le dimensioni del diagramma, si è scelto di nascondere gli attributi e le operazioni delle classi poiché altrimenti l'immagine sarebbe stata illeggibile.

2.1. Class - t_sensor

Name	Value
Name	t_sensor
Stereotypes	SW, Typedef
Documentation	Type to be used as much as possible to store and transfer telemetry data and actuator values. It corresponds to a 16-bits signed integer. Its value can be properly scaled by means of the t_scaling13 class.

2.2. Class - TileOrientation

Name	Value
Name	TileOrientation
Stereotypes	SW
Documentation	Define for each tile the direction cosine matrix of the XYZ tile reference frame with respect to the UVW satellite reference frame.

Attributes

tileO	
Signature	+tileO : float[3][3]
Visibility	public
Scope	instance
Documentation	Direction cosine matrix of a tile. Defines XYZ tile coordinate system with respect to the UVW satellite reference frame. First row are the coordinates of X axis on the UVW frame (U is first column; V is second column; W is third column). Second and third row are the coordinates of Y and Z axis, respectively. Each row must have 2-norm equal to 1.

2.3. Class - Controller

Name	Value
Name	Controller

Stereotypes	SW
Documentation	This class allow the acquisition and the analysis of the telemetry of each tile (1B8 and 1B9) in the satellite. It can also update (or ask to the **Ground Station** for updates) the configuration of the system, according to its status. The acquisition is attempted at most MAX_ATTEMPTS for each tile. If the error still after MAX_ATTEMPTS attempts, it starts the acquisition with the next tile. Any communication error is saved in masErrPMT (of class Manager_1B8) or masErrOBC (of class Manager_1B9) depending on the tile's type.

Attributes

NUMTILES	
Signature	+NUMTILES : ushort const = 6
Visibility	public
Scope	classifier

NUMPMT	
Signature	+NUMPMT : ushort const = 5
Visibility	public
Scope	classifier

NUMOBC	
Signature	+NUMOBC : ushort const = 1
Visibility	public
Scope	classifier

MAX_ATTEMPTS	
Signature	+MAX_ATTEMPTS : byte const = 5
Visibility	public
Scope	classifier
Documentation	The maximum number of communication attempts.

manPMT	
Signature	+manPMT : Manager_1B8
Visibility	public
Scope	classifier

manOBC	
Signature	+manOBC : Manager_1B9
Visibility	public
Scope	classifier

config	
Signature	-config : mechanicalConfiguration
Visibility	private
Scope	classifier

PI	
Signature	-PI : Pseudoinverse
Visibility	private
Scope	classifier

isr	
Signature	-isr : UnusedInterrupts_master
Visibility	private
Scope	classifier

Operations

getHousekeeping	
Signature	getHousekeeping()
Visibility	public
Scope	classifier
Documentation	Reads the housekeeping arrays from tiles in the satellite. Data are stored in hkPMT of Manager_1B8 class for PMT (1B8) tiles. Data are stored in hkOBC of Manager_1B9 class for OBC (1B9) tiles.

Code Body	#ifdef __DEBUG
	for(int T = 0; T < config.NUMPMT; T++)
	  for(int i = 0; i < config.LENHkPMT; i++)
	  manPMT.hkPMT[T][i] =0x0001;
	
	// Tile 0
	manPMT.hkPMT[0][0] = 0x8000; // EARTH_SENSOR_X
	manPMT.hkPMT[0][1] = 0x8000; // EARTH_SENSOR_Y
	manPMT.hkPMT[0][2] = 0x8000; // SUN_SENSOR_X
	manPMT.hkPMT[0][3] = 0x8000; // SUN_SENSOR_Y
	manPMT.hkPMT[0][6] = 3; // SOLAR_TEMPERATURE_CENTER

```
manPMT.hkPMT[0][7] = 3;                 //
SOLAR_TEMPERATURE_SIDE
manPMT.hkPMT[0][11] = 333; // BATTERY_TEMPERATURE
manPMT.hkPMT[0][12] = 318; //
POWERCIRCUITS_TEMPERATURE
manPMT.hkPMT[0][17] = 120; // INTERNAL_BUS_VOLTAGE
manPMT.hkPMT[0][20] = 500; // SPIN_Z
manPMT.hkPMT[0][23] = -11732;       // MAGNETIC_FIELD_X
manPMT.hkPMT[0][24] = -16032;       // MAGNETIC_FIELD_Y
manPMT.hkPMT[0][25] = 335; // ACS_TEMPERATURE
// Tile 1
manPMT.hkPMT[1][0] = 0x8000;        // EARTH_SENSOR_X
manPMT.hkPMT[1][1] = 0x8000;        // EARTH_SENSOR_Y
manPMT.hkPMT[1][2] = 6938; // SUN_SENSOR_X
manPMT.hkPMT[1][3] = 4033; // SUN_SENSOR_Y
manPMT.hkPMT[1][6] = 500;   //
SOLAR_TEMPERATURE_CENTER
manPMT.hkPMT[1][7] = 500;   // SOLAR_TEMPERATURE_SIDE
manPMT.hkPMT[1][11] = 348; // BATTERY_TEMPERATURE
manPMT.hkPMT[1][12] = 343; //
POWERCIRCUITS_TEMPERATURE
manPMT.hkPMT[1][17] = 110; // INTERNAL_BUS_VOLTAGE
manPMT.hkPMT[1][20] = 0;    // SPIN_Z
manPMT.hkPMT[1][23] = 16032;        // MAGNETIC_FIELD_X
manPMT.hkPMT[1][24] = 4670;         // MAGNETIC_FIELD_Y
manPMT.hkPMT[1][25] = 352; // ACS_TEMPERATURE
// Tile 2
manPMT.hkPMT[2][0] = 0x8000;        // EARTH_SENSOR_X
manPMT.hkPMT[2][1] = 0x8000;        // EARTH_SENSOR_Y
manPMT.hkPMT[2][2] = -8672;         // SUN_SENSOR_X
manPMT.hkPMT[2][3] = -6430;         // SUN_SENSOR_Y
manPMT.hkPMT[2][6] = 500;   //
SOLAR_TEMPERATURE_CENTER
manPMT.hkPMT[2][7] = 500;   // SOLAR_TEMPERATURE_SIDE
manPMT.hkPMT[2][11] = 353; // BATTERY_TEMPERATURE
manPMT.hkPMT[2][12] = 341; //
POWERCIRCUITS_TEMPERATURE
manPMT.hkPMT[2][17] = 117; // INTERNAL_BUS_VOLTAGE
manPMT.hkPMT[2][20] = 1000;         // SPIN_Z
manPMT.hkPMT[2][23] = -11732;       // MAGNETIC_FIELD_X
manPMT.hkPMT[2][24] = -4670;        // MAGNETIC_FIELD_Y
manPMT.hkPMT[2][25] = 348; // ACS_TEMPERATURE
// Tile 3
manPMT.hkPMT[3][0] = 0x8000;        // EARTH_SENSOR_X
manPMT.hkPMT[3][1] = 0x8000;        // EARTH_SENSOR_Y
manPMT.hkPMT[3][2] = -8760;         // SUN_SENSOR_X
manPMT.hkPMT[3][3] = 4458; // SUN_SENSOR_Y
manPMT.hkPMT[3][6] = 500;   //
SOLAR_TEMPERATURE_CENTER
manPMT.hkPMT[3][7] = 500;   // SOLAR_TEMPERATURE_SIDE
manPMT.hkPMT[3][11] = 345; // BATTERY_TEMPERATURE
manPMT.hkPMT[3][12] = 347; //
POWERCIRCUITS_TEMPERATURE
manPMT.hkPMT[3][17] = 115; // INTERNAL_BUS_VOLTAGE
manPMT.hkPMT[3][20] = -500;         // SPIN_Z
manPMT.hkPMT[3][23] = -11732;       // MAGNETIC_FIELD_X
manPMT.hkPMT[3][24] = 16032;        // MAGNETIC_FIELD_Y
manPMT.hkPMT[3][25] = 350; // ACS_TEMPERATURE
// Tile 4
manPMT.hkPMT[4][0] = 0x8000;        // EARTH_SENSOR_X
```

	manPMT.hkPMT[4][1] = 0x8000; // EARTH_SENSOR_Y
	manPMT.hkPMT[4][2] = 0x8000; // SUN_SENSOR_X
	manPMT.hkPMT[4][3] = 0x8000; // SUN_SENSOR_Y
	manPMT.hkPMT[4][6] = 3; // SOLAR_TEMPERATURE_CENTER
	manPMT.hkPMT[4][7] = 3; // SOLAR_TEMPERATURE_SIDE
	manPMT.hkPMT[4][11] = 331; // BATTERY_TEMPERATURE
	manPMT.hkPMT[4][12] = 323; // POWERCIRCUITS_TEMPERATURE
	manPMT.hkPMT[4][17] = 122; // INTERNAL_BUS_VOLTAGE
	manPMT.hkPMT[4][20] = 0; // SPIN_Z
	manPMT.hkPMT[4][23] = -16032; // MAGNETIC_FIELD_X
	manPMT.hkPMT[4][24] = 4670; // MAGNETIC_FIELD_Y
	manPMT.hkPMT[4][25] = 339; // ACS_TEMPERATURE
	
	for(int T = 0; T < config.NUMOBC; T++)
	for(int i = 0; i < config.LENHkOBC; i++)
	manOBC.hkOBC[T][i] =0x0001;
	
	#else
	for(int T = 0; T < config.NUMPMT; T++)
	{
	manPMT.masErrPMT[T] = manPMT.msg2PMT(T, Bk1A1_Common_Codes::CMD_GET_HOUSEKEEPING, (byte *)manPMT.hkPMT[T]);
	for(int i = 0; (i < MAX_ATTEMPTS) & (manPMT.masErrPMT[T] != Bk1B45_Master::NO_ERROR); i++)
	manPMT.masErrPMT[T] = manPMT.msg2PMT(T, Bk1A1_Common_Codes::CMD_GET_HOUSEKEEPING, (byte *)manPMT.hkPMT[T]);
	}
	
	for(int T = 0; T < config.NUMOBC; T++)
	{
	manOBC.masErrOBC[T] = manOBC.msg2OBC(T, Bk1A1_Common_Codes::CMD_GET_HOUSEKEEPING, (byte *)manOBC.hkOBC[T]);
	for(int i = 0; (i < MAX_ATTEMPTS) & (manOBC.masErrOBC[T] != Bk1B45_Master::NO_ERROR); i++)
	manOBC.masErrOBC[T] = manOBC.msg2OBC(T, Bk1A1_Common_Codes::CMD_GET_HOUSEKEEPING, (byte *)manOBC.hkOBC[T]);
	}
	#endif

getStatistics	
Signature	getStatistics()
Visibility	public
Scope	classifier
Documentation	Reads the statistics arrays from all tiles in the satellite. Values are stored in statPMT of Manager_1B8 class for PMT (1B8) tiles. Values are stored in statOBC of Manager_1B9 class for OBC (1B9) tiles.

Code Body	#ifdef def_CMD_GET_STATISTICS
	
	for(int T = 0; T < config.NUMPMT; T++)
	{
	  manPMT.masErrPMT[T] = manPMT.msg2PMT(T, Bk1A1_Common_Codes::CMD_GET_STATISTICS, (byte *)manPMT.statPMT[T]);
	  for(int i = 0; (i < MAX_ATTEMPTS) & (manPMT.masErrPMT[T] != Bk1B45_Master::NO_ERROR); i++)
	  manPMT.masErrPMT[T] = manPMT.msg2PMT(T, Bk1A1_Common_Codes::CMD_GET_STATISTICS, (byte *)manPMT.statPMT[T]);
	}
	
	for(int T = 0; T < config.NUMOBC; T++)
	{
	  manOBC.masErrOBC[T] = manOBC.msg2OBC(T, Bk1A1_Common_Codes::CMD_GET_STATISTICS, (byte *)manOBC.statOBC[T]);
	  for(int i = 0; (i < MAX_ATTEMPTS) & (manOBC.masErrOBC[T] != Bk1B45_Master::NO_ERROR); i++)
	  manOBC.masErrOBC[T] = manOBC.msg2OBC(T, Bk1A1_Common_Codes::CMD_GET_STATISTICS, (byte *)manOBC.statOBC[T]);
	}
	
	#endif

computeStatistics	
Signature	computeStatistics() : void
Visibility	public
Scope	classifier
Documentation	Computes a statistics array for each tile in the satellite. Statistics are updated every time the operation is called. Values are stored in statPMT of Manager_1B8 class for PMT (1B8) tiles. Values are stored in statOBC of Manager_1B9 class for OBC (1B9) tiles.

Code Body	for(int T = 0; T < config.NUMPMT; T++)
	  for(int i = 0; i < config.LENHkPMT; i++)
	  {
	  if(manPMT.statPMT[T][i] > manPMT.hkPMT[T][i]) // Ricerca del minimo
	  manPMT.statPMT[T][i] = manPMT.hkPMT[T][i];
	
	  if(manPMT.statPMT[T][config.LENHkPMT+i] < manPMT.hkPMT[T][i]) // Ricerca del massimo
	  manPMT.statPMT[T][config.LENHkPMT+i] = manPMT.hkPMT[T][i];
	  }
	
	for(int T = 0; T < config.NUMOBC; T++)
	  for(int i = 0; i < config.LENHkOBC; i++)
	  {

	if(manOBC.statOBC[T][i] > manOBC.hkOBC[T][i]) // Ricerca del minimo
	manOBC.statOBC[T][i] = manOBC.hkOBC[T][i];
	
	if(manOBC.statOBC[T][config.LENHkOBC+i] < manOBC.hkOBC[T][i]) // Ricerca del massimo
	manOBC.statOBC[T][config.LENHkOBC+i] = manOBC.hkOBC[T][i];
	}

getHistory	
Signature	getHistory()
Visibility	public
Scope	classifier
Documentation	Reads the history matrix from all tiles in the satellite. Values are stored in hisPMT of Manager_1B8 class for PMT (1B8) tiles. Values are stored in hisOBC of Manager_1B9 class for OBC (1B9) tiles.

Code Body	#ifdef def_CMD_GET_HISTORY
	
	for(int T = 0; T < config.NUMPMT; T++)
	{
	manPMT.masErrPMT[T] = manPMT.msg2PMT(T, Bk1A1_Common_Codes::CMD_GET_HISTORY, (byte *)manPMT.hisPMT[T]);
	for(int i = 0; (i < MAX_ATTEMPTS) & (manPMT.masErrPMT[T] != Bk1B45_Master::NO_ERROR); i++)
	manPMT.masErrPMT[T] = manPMT.msg2PMT(T, Bk1A1_Common_Codes::CMD_GET_HISTORY, (byte *)manPMT.hisPMT[T]);
	}
	
	for(int T = 0; T < config.NUMOBC; T++)
	{
	manOBC.masErrOBC[T] = manOBC.msg2OBC(T, Bk1A1_Common_Codes::CMD_GET_HISTORY, (byte *)manOBC.hisOBC[T]);
	for(int i = 0; (i < MAX_ATTEMPTS) & (manOBC.masErrOBC[T] != Bk1B45_Master::NO_ERROR); i++)
	manOBC.masErrOBC[T] = manOBC.msg2OBC(T, Bk1A1_Common_Codes::CMD_GET_HISTORY, (byte *)manOBC.hisOBC[T]);
	}
	
	#endif

computeHistory	
Signature	computeHistory()
Visibility	public
Scope	classifier

Documentation	Computes a history matrix for each tiles in the satellite. History data are updated every time the operation is called. Values are stored in hisPMT of Manager_1B8 class for PMT (1B8) tiles. Values are stored in hisOBC of Manager_1B9 class for OBC (1B9) tiles.

Code Body	if(manPMT.hisPMTcnt < config.DEPHisPMT)
	  manPMT.hisPMTcnt++;
	
	for(int T = 0; T < config.NUMPMT; T++)
	  for(int i = 0; i < manPMT.hisPMTcnt; i++)
	  for(int g = 0; g < config.LENHisPMT; g++)
	  if(manPMT.hisPMTcnt-1 < config.DEPHisPMT)
	  manPMT.hisPMT[T][manPMT.hisPMTcnt-1][g] = manPMT.hkPMT[T][g];
	  else if(i == manPMT.hisPMTcnt-1)
	  manPMT.hisPMT[T][i][g] = manPMT.hkPMT[T][g];
	  else
	  manPMT.hisPMT[T][i][g] = manPMT.hisPMT[T][i+1][g];
	
	if(manOBC.hisOBCcnt < config.DEPHisPMT)
	  manOBC.hisOBCcnt++;
	
	for(int T = 0; T < config.NUMOBC; T++)
	  for(int i = 0; i < manOBC.hisOBCcnt; i++)
	  for(int g = 0; g < config.LENHisPMT; g++)
	  if(manOBC.hisOBCcnt-1 < config.DEPHisPMT)
	  manOBC.hisOBC[T][manOBC.hisOBCcnt-1][g] = manOBC.hkOBC[T][g];
	  else if(i == manOBC.hisOBCcnt-1)
	  manOBC.hisOBC[T][i][g] = manOBC.hkOBC[T][g];
	  else
	  manOBC.hisOBC[T][i][g] = manOBC.hisOBC[T][i+1][g];

getConfiguration	
Signature	getConfiguration()
Visibility	public
Scope	classifier
Documentation	Reads the configuration of all tiles in the satellite. Configurations are stored in confPMT of Manager_1B8 class for PMT (1B8) tiles. Configurations are stored in confOBC of Manager_1B9 class for OBC (1B9) tiles.

Code Body	#ifdef def_CMD_GET_CONFIGURATION
	
	for(int T = 0; T < config.NUMPMT; T++)
	{
	  manPMT.masErrPMT[T] = manPMT.msg2PMT(T, Bk1A1_Common_Codes::CMD_GET_CONFIGURATION, (byte *)&manPMT.confPMT[T]);
	  for(int i = 0; (i < MAX_ATTEMPTS) & (manPMT.masErrPMT[T] != Bk1B45_Master::NO_ERROR); i++)

	manPMT.masErrPMT[T] = manPMT.msg2PMT(T, Bk1A1_Common_Codes::CMD_GET_CONFIGURATION, (byte *)&manPMT.confPMT[T]);
	}
	
	for(int T = 0; T < config.NUMOBC; T++)
	{
	manOBC.masErrOBC[T] = manOBC.msg2OBC(T, Bk1A1_Common_Codes::CMD_GET_CONFIGURATION, (byte *)&manOBC.confOBC[T]);
	for(int i = 0; (i < MAX_ATTEMPTS) & (manOBC.masErrOBC[T] != Bk1B45_Master::NO_ERROR); i++)
	manOBC.masErrOBC[T] = manOBC.msg2OBC(T, Bk1A1_Common_Codes::CMD_GET_CONFIGURATION, (byte *)&manOBC.confOBC[T]);
	}
	
	#endif

setConfiguration	
Signature	setConfiguration(T : byte, ID : byte, conf : byte)
Visibility	public
Scope	classifier
Documentation	Send a new configuration conf to tile T. Parameter ID defines tile's type according to the ones available in class ID_codes.

Code Body	#ifdef def_CMD_SET_CONFIGURATION
	
	if(ID == Bk1A1_Common_Codes::Bk1B8_PMT_V0)
	{
	manPMT.masErrPMT[T] = manPMT.msg2PMT(T, Bk1A1_Common_Codes::CMD_SET_CONFIGURATION, conf);
	
	for(int i = 0; (i < MAX_ATTEMPTS) & (manPMT.masErrPMT[T] != Bk1B45_Master::NO_ERROR); i++)
	manPMT.masErrPMT[T] = manPMT.msg2PMT(T, Bk1A1_Common_Codes::CMD_SET_CONFIGURATION, conf);
	}
	else if(ID == Bk1A1_Common_Codes::Bk1B9_TLC_V0)
	{
	manOBC.masErrOBC[T] = manOBC.msg2OBC(T, Bk1A1_Common_Codes::CMD_SET_CONFIGURATION, conf);
	
	for(int i = 0; (i < MAX_ATTEMPTS) & (manOBC.masErrOBC[T] != Bk1B45_Master::NO_ERROR); i++)
	manOBC.masErrOBC[T] = manOBC.msg2OBC(T, Bk1A1_Common_Codes::CMD_SET_CONFIGURATION, conf);
	}
	
	#endif

analyze	
Signature	analyze()
Visibility	public

Scope	classifier
Documentation	Computes, by calling analyzePMT and analyzeOBC, a set of statistics, stored in the attributes of classes Result_1B8 and Result_1B9, which give a representation of the global status of the system.
Code Body	manPMT.analyzePMT(); manOBC.analyzeOBC();

resetStatistics	
Signature	resetStatistics()
Visibility	public
Scope	classifier
Documentation	Resets all values in vectors statPMT and statOBC.

Code Body	for(int i = 0; i < config.NUMPMT; i++)
	  for(int g = 0; g < config.LENStatPMT; g++)
	  {
	  manPMT.statPMT[i][g] = 0x7FFF; // Valore massimo positivo
	  manPMT.statPMT[i][config.LENStatPMT+g] = 0x8000; // Valore minimo negativo
	  }
	
	for(int i = 0; i < config.NUMOBC; i++)
	  for(int g = 0; g < config.LENStatOBC; g++)
	  {
	  manOBC.statOBC[i][g] = 0x7FFF; // Valore massimo positivo
	  manOBC.statOBC[i][config.LENStatOBC+g] = 0x8000; // Valore minimo negativo
	  }

clearHistory	
Signature	clearHistory()
Visibility	public
Scope	classifier
Documentation	Clears vectors hisPMT and hisOBC. Resets flags hisPMTcnt and hisOBCcnt.

Code Body	for(int i = 0; i < config.NUMPMT; i++)
	  for(int g = 0; g < config.DEPHisPMT; g++)
	  for(int h = 0; h < config.LENHisPMT; h++)
	  manPMT.hisPMT[i][g][h] = 0;
	
	for(int i = 0; i < config.NUMOBC; i++)
	  for(int g = 0; g < config.DEPHisOBC; g++)
	  for(int h = 0; h < config.LENHisOBC; h++)
	  manOBC.hisOBC[i][g][h] = 0;
	
	manPMT.hisPMTcnt = 0;
	manOBC.hisOBCcnt = 0;

computeSystemUpdates	
Signature	computeSystemUpdates()
Visibility	public
Scope	classifier
Documentation	Updates pseudoinverse matrices, if necessary. Matrices' status are stored in the attribute PIstat of class Pseudoinverse.

Code Body	if(PI.PIstat.test(StatusPI::MAG_FIELD_OBSOLETE,
	StatusPI::MAG_FIELD_OBSOLETE))
	{
	// Pseudoinversa per Campo Magnetico obsoleta..
	PI.PIstat.set(StatusPI::MAG_FIELD_UPDATING,
	StatusPI::MAG_FIELD_UPDATING); // Starting Update..
	PI.Update(Bk1A_1B8_Codes::MAGNETIC_FIELD_X,
	StatusPI::magnetometer); // Updating pseudoinverse..
	PI.PIstat.reset(StatusPI::MAG_FIELD_OBSOLETE +
	StatusPI::MAG_FIELD_UPDATING,
	StatusPI::MAG_FIELD_OBSOLETE +
	StatusPI::MAG_FIELD_UPDATING); // Update complete..
	}
	
	if(PI.PIstat.test(StatusPI::SPIN_OBSOLETE,
	StatusPI::SPIN_OBSOLETE))
	{
	// Pseudoinversa per Spin obsoleta..
	PI.PIstat.set(StatusPI::SPIN_UPDATING,
	StatusPI::SPIN_UPDATING); // Starting Update..
	PI.Update(Bk1A_1B8_Codes::SPIN_Z, StatusPI::gyroscope);
	// Updating pseudoinverse..
	PI.PIstat.reset(StatusPI::SPIN_OBSOLETE +
	StatusPI::SPIN_UPDATING,
	StatusPI::SPIN_OBSOLETE +
	StatusPI::SPIN_UPDATING); // Update complete..
	}
	
	if(PI.PIstat.test(StatusPI::SUN_SENS_OBSOLETE,
	StatusPI::SUN_SENS_OBSOLETE))
	{
	// Pseudoinversa per Sensore di Sole obsoleta..
	PI.PIstat.set(StatusPI::SUN_SENS_UPDATING,
	StatusPI::SUN_SENS_UPDATING); // Starting Update..
	PI.Update(Bk1A_1B8_Codes::SUN_SENSOR_X, StatusPI::sun);
	// Updating pseudoinverse..
	PI.PIstat.reset(StatusPI::SUN_SENS_OBSOLETE +
	StatusPI::SUN_SENS_UPDATING,
	StatusPI::SUN_SENS_OBSOLETE +
	StatusPI::SUN_SENS_UPDATING); // Update complete..
	}
	
	if(PI.PIstat.test(StatusPI::EARTH_SENS_OBSOLETE,
	StatusPI::EARTH_SENS_OBSOLETE))
	{
	// Pseudoinversa per Sensore di Sole obsoleta..
	PI.PIstat.set(StatusPI::EARTH_SENS_UPDATING,
	StatusPI::EARTH_SENS_UPDATING); // Starting Update..
	PI.Update(Bk1A_1B8_Codes::EARTH_SENSOR_X,

	StatusPI::earth); // Updating pseudoinverse..
	PI.PIstat.reset(StatusPI::EARTH_SENS_OBSOLETE +
	StatusPI::EARTH_SENS_UPDATING,
	StatusPI::EARTH_SENS_OBSOLETE +
	StatusPI::EARTH_SENS_UPDATING); // Update complete..
	}

requestSystemUpdates	
Signature	requestSystemUpdates(T : byte)
Visibility	public
Scope	classifier
Documentation	Sends a request for updating of class Pseudoinverse to OBC (1B9) tile T.
Code Body	manOBC.masErrOBC[T] = manOBC.msg2OBC(T, Bk1A1_Common_Codes::CMD_WRITE_DATA_0, (byte *)&PI.PIstat); for(int i = 0; (i < MAX_ATTEMPTS) & (manOBC.masErrOBC[T] != Bk1B45_Master::NO_ERROR); i++) manOBC.masErrOBC[T] = manOBC.msg2OBC(T, Bk1A1_Common_Codes::CMD_WRITE_DATA_0, (byte *)&PI.PIstat);

2.4. Class - Lookups

Name	**Value**
Name	Lookups
Stereotypes	SW
Documentation	A class containing useful lookup tables to compute non-linear functions in a quick way.

Attributes

pi	
Signature	+pi : float const = 3.14159265
Visibility	public
Scope	classifier

RANGE_ANGLE	
Signature	+RANGE_ANGLE : ushort const = 4096
Visibility	public
Scope	classifier
Documentation	Resolution for angles. Angles range **0 = angle 2*pi** in units of

	2*pi/RANGE_ANGLE. RANGE_ANGLE should be a power of 2. MUST be RANGE_TRIGO > RANGE_ANGLE.

COEFF_ANGLE	
Signature	+COEFF_ANGLE : float const = 2.0*pi/RANGE_ANGLE
Visibility	public
Scope	classifier
Documentation	Multiplicative coefficient to convert any angle expressed in units of 1/RANGE_ANGLE into radians.

RANGE_TRIGO	
Signature	+RANGE_TRIGO : ushort const = 16384
Visibility	public
Scope	classifier
Documentation	The range of image of lookup tables. Trigonometric functions range between -1 and +1 in units of **1/RANGE_TRIGO**. RANGE_TRIGO should be a power of 2. MUST be RANGE_TRIGO > RANGE_ANGLE.

cos2pi	
Signature	+cos2pi : short const[RANGE_ANGLE] = {…}
Visibility	public
Scope	classifier
Documentation	Lookup table for cos(x), where x ranges 0 = x 2*pi in units of 2*pi/RANGE_ANGLE. The value is between -1 and +1 in units of 1/RANGE.

sin2pi	
Signature	+sin2pi : short const[RANGE_ANGLE] = {…}
Visibility	public
Scope	classifier
Documentation	Lookup table for sin(x), where x ranges 0 = x 2*pi in units of 2*pi/RANGE_ANGLE. The value is between -1 and +1 in units of 1/RANGE.

NaN	
Signature	+NaN const = 0x8000
Visibility	public
Scope	classifier
Documentation	Symbol indicating "not a number" for t_sensor types.

2.5. Class - t_MasterErrors

Name	Value
Name	t_MasterErrors
Stereotypes	SW, enumeration

Attributes

NO_ERROR	
Signature	-NO_ERROR
Visibility	private
Scope	instance

CRC_ERROR	
Signature	-CRC_ERROR
Visibility	private
Scope	instance

LENGTH_ERROR	
Signature	-LENGTH_ERROR
Visibility	private
Scope	instance

WRONG_DEST	
Signature	-WRONG_DEST
Visibility	private
Scope	instance

WRONG_LENGTH	
Signature	-WRONG_LENGTH
Visibility	private
Scope	instance

CHAN_BUSY	
Signature	-CHAN_BUSY
Visibility	private
Scope	instance

SLAVE_ERROR	
Signature	-SLAVE_ERROR
Visibility	private
Scope	instance

ID_ERROR	
Signature	-ID_ERROR
Visibility	private
Scope	instance

2.6. Class - Message_Master

Name	Value
Name	Message_Master
Stereotypes	SW

Attributes

TIMEOUT	
Signature	-TIMEOUT : ushort const = 1000
Visibility	private
Scope	classifier
Documentation	The timeout, in microseconds, after which a new message in queue can interrupt an ongoing (possibly corrupted) communication.

MAXPERIPHERALS	
Signature	+MAXPERIPHERALS : byte const = 3
Visibility	public
Scope	classifier
Documentation	The number of SPI channels which are available (only for SPI mode; not available in OBDB mode).

CRCread	
Signature	-CRCread : byte
Visibility	private
Scope	classifier
Documentation	The CRC computed on the fly while receiving data from the **Slave**. At the end of communication it must be 0.

CRCwrite	
Signature	-CRCwrite : byte
Visibility	private
Scope	classifier
Documentation	The CRC computed on the fly while transmitting data to the **Slave**. At the end of communication it must be 0.

id	
Signature	-id : byte
Visibility	private
Scope	classifier
Documentation	Contains the module ID received from the last addressed peripheral.

dest	
Signature	-dest : byte
Visibility	private
Scope	classifier
Documentation	Stores the address of communication (set by the user with the argument **dst** of **sendMessage** operation).

command	
Signature	-command : t_Commands
Visibility	private
Scope	classifier
Documentation	Stores the command code of the ongoing communication (set by the user with the argument **cmd** of **sendMessage** operation).

length	
Signature	-length : ushort*
Visibility	private
Scope	classifier
Documentation	Stores the length of data field of Write Data communications (set by the user with the argument **len** of **sendMessage** operation). Stores the length of data field of Read Data communications (as received from the **Slave**). Not used for Command Only protocol.

buffer

Signature	-buffer : byte*
Visibility	private
Scope	classifier
Documentation	Pointer to data buffer. Memory must be allocated by the user.

TXcounter	
Signature	-TXcounter : ushort
Visibility	private
Scope	classifier
Documentation	Counts up the number of bytes transmitted so far. During execution of TX interrupt routine it is not yet updated, therefore it indicates the number of bytes transmitted minus one.

RXcounter	
Signature	-RXcounter : ushort
Visibility	private
Scope	classifier
Documentation	Counts up the number of bytes received so far. During execution of RX interrupt routine it is not yet updated, therefore it indicates the number of bytes received minus one.

ack	
Signature	-ack : byte
Visibility	private
Scope	classifier
Documentation	Contains the **ack** byte received from the **Slave** (only for SPI mode; not found in OBDB mode).

error	
Signature	-error : t_MasterErrors
Visibility	private
Scope	classifier
Documentation	Stores details on the last detected error. See type **t_Masterrrors** for a list of errors.

spi	
Signature	-spi : UART0_master[MAXPERIPHERALS]
Visibility	private
Scope	classifier

Documentation	A vector of objects for SPI communication. They all share the same UART.

maxlen	
Signature	-maxlen : ushort
Visibility	private
Scope	classifier
Documentation	The maximum allowed length for received message. Any message byte received after maxlen bytes is not stored. Should match the allocated length of the buffer msg. Used only for Read Data Protocol.

Operations

sendMessage	
Signature	sendMessage(dest : byte, command : t_Commands, length : ushort, msg : byte, maxlen : ushort) : bool
Visibility	public
Scope	instance
Documentation	Transfers a command/message of type **cmd** to/from the **Peripheral** number **dst**. The type of protocol (Write Data, Read Data or Command Only) depends on the value of **cmd**. The type of command is indicated by parameter **cmd**. The length of the data field is indicated by parameter **len**. For Write Data protocol, **len** is written by the used when calling **sendMessage**. For Read Data protocol, **len** is set by the **Slave**, therefore **len** returns a value. For Command Only protocol, **len** is ignored. The message is pointed to by **msg**. The routine only starts communication, which then goes further autonomously. The **kernel** actor shall then check when communication terminates and if any error has occurred by polling the **ready()** operation. The routine first checks that **dst** is between 0 and **MAXSPI-1** and **length** is between 1 and 256. In the opposite case, case it immediately exits and sets the **error** flag, which can then be ready by means of the **ready()** operation. The routine then checks if communication channel (SPI) is free. Otherwise it waits **TIMEOUT** microseconds, then it aborts any ongoing communication and starts sending this new message. In case of any error, the **error** flag is set, which can be read by means of the **ready()** operation.

Code Body	error = NO_ERROR;
	
	if (dest>MAXSPI) {
	  error = WRONG_DEST;

	return false;
	}
	if ((length==0) || (length>256)) {
	error = WRONG_LENGTH;
	return false;
	}
	if (waitUntilFree (TIMEOUT) == false) {
	clearCommunication ();
	error = CHAN_BUSY;
	}
	
	TXcounter = 0;
	RXcounter = 0;
	CRCwrite = 0xC3;
	CRCread = 0x0F;
	this->dest = dest;
	this->command = command;
	this->length = &length;
	this->maxlen = maxlen;
	this->buffer = msg;
	
	spi[dest].enable (true, true);
	
	return true;

ready	
Signature	ready(dst : byte, slaveError : bool, CRC : ushort, destID : byte, masterError : t_MasterErrors) : bool
Visibility	public
Scope	instance
Documentation	Checks if communication to destination **dest** has terminated. Returns **false** if communication is still going on; **true** otherwise. Also returns boolean **slaveError** if slave has reported an error via its **Error** pin (if true, the caller shall issue a CMD_GET_STATUS command to get additional details on the error). It also returns **CRC** of last received message (should be 0 if communication has terminated; should be unpredictable if communication has not terminated yet; it might be 0 if an error has occurred during reception). Also returns **destID** containing the ID reported by the slave (valid only when communication has terminated). Also returns **masterError** to report any error detected by the master during communication (see **t_MasterErrors** class for further details).

Code Body	if (spi[dest].isEnabled() == true) {
	return false;
	}
	slaveError = spi[dest].getErrorPin();
	CRC = (ushort) CRCread;
	destID = id;
	masterError = error;
	return true;

isFree	
Signature	isFree() : bool
Visibility	public
Scope	instance
Documentation	Returns **true** if SPI communication channel is free; **false** otherwise, that is, if any communication is going on.

Code Body	byte i;
	for (i=0; i<MAXSPI; i++) {
	if (spi[i].isEnabled()) return false;
	}
	return true;

waitUntilFree	
Signature	waitUntilFree(timeout : ushort) : bool
Visibility	public
Scope	instance
Documentation	Waits until SPI communication channel is free, then it returns **true**. If the channel does not become free within **TIMEOUT** microseconds, it returns **false**.

Code Body	#if MSP430x42x || MSP430x43x || MSP430x54x
	volatile ulong cnt = (int)(((timeout*0.000001)*Bk1B45_Master::Master_Processor::CLOCK_FREQ)/(5)); // TBD
	#endif
	#if CHIPCON
	#endif
	
	while ((isFree() == false) & (cnt > 0)) {
	cnt--;
	}
	
	if (cnt != 0)
	return true;
	else
	return false;

clearCommunication	
Signature	clearCommunication() : void
Visibility	public
Scope	instance
Documentation	Stops a currently ongoing data exchange, by disabling the UART and lowering all CS pins. Data involved in the ongoing exchange may go lost.
Code Body	byte i; for (i=0; i<MAXSPI; i++)

	spi[i].disable();

isr_receivedByte	
Signature	isr_receivedByte() : void
Visibility	public
Scope	classifier
Documentation	Interrupt service routine which shall be called whenever the RX SPI buffer is full.

Code Body	if (Bk1B45_Slave::CommandInterpreter::usesWriteDataProtocol(command)) {
	  if (RXcounter == 1) {
	  id = spi[dest].readData (CRCread);
	  }
	  else if (RXcounter == 2) {
	  spi[dest].readData (CRCread);
	  }
	  else if (RXcounter == (*length+2)) {
	  spi[dest].disable ();
	  }
	  }
	else if
	(Bk1B45_Slave::CommandInterpreter::usesReadDataProtocol(command)) {
	  if (RXcounter == 1) {
	  id = spi[dest].readData (CRCread);
	  }
	  else if (RXcounter == 2) {
	  *length = 1 + spi[dest].readData (CRCread);
	  }
	  else if ((RXcounter < (*length + 3)) && ((RXcounter) < (maxlen))) {
	  buffer[RXcounter-3] = spi[dest].readData (CRCread);
	  }
	  else if (RXcounter == (*length + 3)) {
	  spi[dest].readData (CRCread);
	  }
	  else if (RXcounter == (*length + 5)) {
	  ack = spi[dest].readData (CRCread);
	  spi[dest].disable ();
	  }
	  }
	else if
	(Bk1B45_Slave::CommandInterpreter::usesCommandOnlyProtocol(command))
	{
	  if (RXcounter == 1) {
	  id = spi[dest].readData (CRCread);
	  }
	  else if (RXcounter == 2) {
	  spi[dest].readData (CRCread);
	  spi[dest].disable ();
	  }
	  }
	RXcounter++;

isr_TransmittingByte

Signature	isr_TransmittingByte() : void
Visibility	public
Scope	classifier
Documentation	Interrupt service routine to be called whenever the TX SPI buffer is free.

Code Body	if (Bk1B45_Slave::CommandInterpreter::usesWriteDataProtocol(command)) {
	  if (TXcounter == 0) {
	  spi[dest].writeData (command, CRCwrite);
	  }
	  else if (TXcounter == 1) {
	  spi[dest].writeData ((byte)(*length-1), CRCwrite);
	  }
	  else if (TXcounter < (*length+2)) {
	  spi[dest].writeData (buffer[TXcounter-2], CRCwrite);
	  }
	  else if (TXcounter == (*length+2)) {
	  spi[dest].writeData (CRCwrite);
	  }
	  else { // just for safety; should never come here
	  spi[dest].writeData (0);
	  }
	  }
	else if
	(Bk1B45_Slave::CommandInterpreter::usesReadDataProtocol(command)) {
	  if (TXcounter == 0) {
	  spi[dest].writeData (command, CRCwrite);
	  }
	  else if (TXcounter == 1) {
	  spi[dest].writeData (CRCwrite);
	  }
	  else {
	  spi[dest].writeData (0);
	  }
	  }
	else if
	(Bk1B45_Slave::CommandInterpreter::usesCommandOnlyProtocol(command))
	{
	  if (TXcounter == 0) {
	  spi[dest].writeData (command, CRCwrite);
	  }
	  else if (TXcounter == 1) {
	  spi[dest].writeData (CRCwrite);
	  }
	  else {
	  spi[dest].writeData (0);
	  }
	  }
	TXcounter++;

init	
Signature	init(id : byte, PORT : byte, BIT_CS : byte, BIT_ERR : byte) : void
Visibility	public
Scope	instance
Documentation	Configures USART **id** for either:

	• SPI mode, 3-wires, master mode, on port P3.1, P3.2, P3.3. (when def_SPI_MODE is defined) • OBDB mode(when def_OBDB_MODE is defined) MUST be called at boot before enabling interrupts !!!

Code Body	#if MSP430
	if ((id>=0) && (id<MAXSPI)) {
	spi[id].init(PORT, BIT_CS, BIT_ERR);
	}
	#endif
	
	#if CHIPCON
	
	#endif

getTelemetry	
Signature	getTelemetry(tile : byte, length : ushort, destID : byte, housekeeping : t_sensor) : t_MasterErrors
Visibility	public
Scope	instance
Documentation	Asks for telemetry to tile number **tile**, of type **destID**. Returns a **t_sensor housekeeping[]** of maximum length **length**. Returns **NO_ERROR** if telemetry has been successfully received. Returns **CRC_ERROR** or **LENGTH_ERROR** or **SLAVE_ERROR** or **ID_ERROR** in case of (respectively): CRC error during reception of data; error in length of telemetry (i.e. tile returns a message whose length does not match parameter **length**) ; a generic slave error (the caller in this case might then read error using GetStatus use case); the type of tile addressed by parameter **tile** does not match the parameter **ID**.

Code Body	int len = length;
	bool slaveError;
	ushort CRC;
	byte ID = destID;
	t_MasterErrors masterError;
	sendMessage(tile,Bk1A1_Common_Codes::CMD_GET_HOUSEKEEPING,length,(byte *)housekeeping,
	sizeof(t_sensor)*Bk1A_1B8_Codes::LENGTH_HOUSEKEEPING);
	while (ready(tile, slaveError, CRC, destID, masterError) == false) {}
	if (CRC != 0) {
	return CRC_ERROR;
	}
	if (slaveError) {
	sendMessage(tile,Bk1A1_Common_Codes::CMD_GET_STATUS,length,(byte *)&retmsg,sizeof(t_Status_1B8));
	while (ready(tile, slaveError, CRC, destID, masterError) == false) {}
	if (CRC != 0) {
	return CRC_ERROR;
	}
	return SLAVE_ERROR;
	}
	if (destID != ID) {
	return ID_ERROR;
	}
	if ((len > 0) && (len != length)) {

	return LENGTH_ERROR;
	} else {
	return NO_ERROR;
	}

2.7. Class - t_Commands

Name	Value
Name	t_Commands
Stereotypes	SW, enumeration
Documentation	Lists all available command codes and the corresponding value. There is one command for each use case, Removing a command from this class removes the corresponding use case. Adding additional commands requires adding the appropriate code to properly interpret and execute the command.

Attributes

CMD_NOP	
Signature	-CMD_NOP = 0x0
Visibility	private
Scope	instance
Documentation	No operation

CMD_SET_CONFIGURATION	
Signature	-CMD_SET_CONFIGURATION = 0x10
Visibility	private
Scope	instance

CMD_RESET_CONFIGURATION	
Signature	-CMD_RESET_CONFIGURATION = 0x11
Visibility	private
Scope	instance

CMD_CALIBRATE	
Signature	-CMD_CALIBRATE = 0x12
Visibility	private
Scope	instance
Documentation	Used to send calibration data to the **Slave**.

CMD_GET_STATUS	
Signature	-CMD_GET_STATUS = 0x20
Visibility	private
Scope	instance
Documentation	Used for **Get (Module) Status** use case.

CMD_GET_HOUSEKEEPING	
Signature	-CMD_GET_HOUSEKEEPING = 0x21
Visibility	private
Scope	instance
Documentation	Used for **Get (Module) Housekeeping** use case.

CMD_GET_STATISTICS	
Signature	-CMD_GET_STATISTICS = 0x22
Visibility	private
Scope	instance
Documentation	Used for **Get (Module) Housekeeping Statistics** use case.

CMD_GET_HISTORY	
Signature	-CMD_GET_HISTORY = 0x23
Visibility	private
Scope	instance
Documentation	Used for **Get (Module) Housekeeping History** use case.

CMD_GET_CONFIGURATION	
Signature	-CMD_GET_CONFIGURATION = 0x24
Visibility	private
Scope	instance
Documentation	Used to return module ID

CMD_RESET	
Signature	-CMD_RESET = 0x30
Visibility	private
Scope	instance
Documentation	Soft reset of the **System**. TBD.

CMD_STANDBY

Signature	-CMD_STANDBY = 0x31
Visibility	private
Scope	instance
Documentation	Used to put the System in standby mode.

CMD_WAKEUP	
Signature	-CMD_WAKEUP = 0x32
Visibility	private
Scope	instance
Documentation	Used to wake up the System from standby mode.

CMD_RESET_STATISTICS	
Signature	-CMD_RESET_STATISTICS = 0x33
Visibility	private
Scope	instance
Documentation	Used for **Reset (Module) Statistics** use case.

CMD_CLEAR_BUFFER	
Signature	-CMD_CLEAR_BUFFER = 0x34
Visibility	private
Scope	instance
Documentation	Clears last buffer read. See documentation of **Clear Buffer** use case in the **1B45** package for further details.

CMD_WRITE_DATA_0	
Signature	-CMD_WRITE_DATA_0 = 0x40
Visibility	private
Scope	instance
Documentation	Used for **Write (Module) Data** use case; write **Designer**-defined data of **Designer**-type 0.

CMD_WRITE_DATA_1	
Signature	-CMD_WRITE_DATA_1 = 0x41
Visibility	private
Scope	instance
Documentation	Used for **Write (Module) Data** use case; write **Designer**-defined data of **Designer**-type 1.

CMD_WRITE_DATA_2

Signature	-CMD_WRITE_DATA_2 = 0x42
Visibility	private
Scope	instance
Documentation	Used for **Write (Module) Data** use case; write **Designer**-defined data of **Designer**-type 2.

CMD_WRITE_DATA_3	
Signature	-CMD_WRITE_DATA_3 = 0x43
Visibility	private
Scope	instance
Documentation	Used for **Write (Module) Data** use case; write **Designer**-defined data of **Designer**-type 3.

CMD_WRITE_DATA_4	
Signature	-CMD_WRITE_DATA_4 = 0x44
Visibility	private
Scope	instance
Documentation	Used for **Write (Module) Data** use case; write **Designer**-defined data of **Designer**-type 4.

CMD_WRITE_DATA_5	
Signature	-CMD_WRITE_DATA_5 = 0x45
Visibility	private
Scope	instance
Documentation	Used for **Write (Module) Data** use case; write **Designer**-defined data of **Designer**-type 5.

CMD_WRITE_DATA_6	
Signature	-CMD_WRITE_DATA_6 = 0x46
Visibility	private
Scope	instance
Documentation	Used for **Write (Module) Data** use case; write **Designer**-defined data of **Designer**-type 6.

CMD_WRITE_DATA_7	
Signature	-CMD_WRITE_DATA_7 = 0x47
Visibility	private
Scope	instance
Documentation	Used for **Write (Module) Data** use case; write **Designer**-defined data of **Designer**-type 7.

CMD_READ_DATA_0	
Signature	-CMD_READ_DATA_0 = 0x80
Visibility	private
Scope	instance
Documentation	Used for **Read (Module) Data** use case; reads **Designer**-defined data of **Designer**-type 0.

CMD_READ_DATA_1	
Signature	-CMD_READ_DATA_1 = 0x81
Visibility	private
Scope	instance
Documentation	Used for **Read (Module) Data** use case; reads **Designer**-defined data of **Designer**-type 1.

CMD_READ_DATA_2	
Signature	-CMD_READ_DATA_2 = 0x82
Visibility	private
Scope	instance
Documentation	Used for **Read (Module) Data** use case; reads **Designer**-defined data of **Designer**-type 2.

CMD_READ_DATA_3	
Signature	-CMD_READ_DATA_3 = 0x83
Visibility	private
Scope	instance
Documentation	Used for **Read (Module) Data** use case; reads **Designer**-defined data of **Designer**-type 3.

CMD_READ_DATA_4	
Signature	-CMD_READ_DATA_4 = 0x84
Visibility	private
Scope	instance
Documentation	Used for **Read (Module) Data** use case; reads **Designer**-defined data of **Designer**-type 4.

CMD_READ_DATA_5	
Signature	-CMD_READ_DATA_5 = 0x85
Visibility	private

Scope	instance
Documentation	Used for **Read (Module) Data** use case; reads **Designer**-defined data of **Designer**-type 5.

CMD_READ_DATA_6	
Signature	-CMD_READ_DATA_6 = 0x86
Visibility	private
Scope	instance
Documentation	Used for **Read (Module) Data** use case; reads **Designer**-defined data of **Designer**-type 6.

CMD_READ_DATA_7	
Signature	-CMD_READ_DATA_7 = 0x87
Visibility	private
Scope	instance
Documentation	Used for **Read (Module) Data** use case; reads **Designer**-defined data of **Designer**-type 7.

CMD_COMMAND_0	
Signature	-CMD_COMMAND_0 = 0xC0
Visibility	private
Scope	instance
Documentation	Issues a **Designer**-defined command of **Designer**-type 0.

CMD_COMMAND_1	
Signature	-CMD_COMMAND_1 = 0xC1
Visibility	private
Scope	instance
Documentation	Issues a **Designer**-defined command of **Designer**-type 1.

CMD_COMMAND_2	
Signature	-CMD_COMMAND_2 = 0xC2
Visibility	private
Scope	instance
Documentation	Issues a **Designer**-defined command of **Designer**-type 2.

CMD_COMMAND_3	
Signature	-CMD_COMMAND_3 = 0xC3
Visibility	private

Scope	instance
Documentation	Issues a **Designer**-defined command of **Designer**-type 3.

CMD_COMMAND_4	
Signature	-CMD_COMMAND_4 = 0xC4
Visibility	private
Scope	instance
Documentation	Issues a **Designer**-defined command of **Designer**-type 4.

CMD_COMMAND_5	
Signature	-CMD_COMMAND_5 = 0xC5
Visibility	private
Scope	instance
Documentation	Issues a **Designer**-defined command of **Designer**-type 5.

CMD_COMMAND_6	
Signature	-CMD_COMMAND_6 = 0xC6
Visibility	private
Scope	instance
Documentation	Issues a **Designer**-defined command of **Designer**-type 6.

CMD_COMMAND_7	
Signature	-CMD_COMMAND_7 = 0xC7
Visibility	private
Scope	instance
Documentation	Issues a **Designer**-defined command of **Designer**-type 7.

2.8. Class - UART0_master

Name	**Value**
Name	UART0_master
Stereotypes	SW
Documentation	This class contains all drivers for a UART configured to be the master of a 4-wires SPI bus, with a non-standard CS signal active-high. CS shall be driven as a normal output pin.When disabled, all output pins are set to 0. When enabled, CS is set to 1; MOSI is unpredictable; MISO is in high impedance; clock is normally low and is generated only when a data is transmitted, then it returns to low.

	MISO is sampled on falling edge of SCK, while MOSI is updated on rising edge of SCK.

Attributes

PORT	
Signature	-PORT : byte*
Visibility	private
Scope	instance

BIT_CS	
Signature	-BIT_CS : byte
Visibility	private
Scope	instance

BIT_ERR	
Signature	-BIT_ERR : byte
Visibility	private
Scope	instance

CS_PIN	
Signature	-CS_PIN : IO_pin const = {&P1OUT,BIT2}
Visibility	private
Scope	classifier

ERR_PIN	
Signature	-ERR_PIN : IO_pin const = {&P1OUT,BIT4}
Visibility	private
Scope	classifier

Operations

UART0_master	
Signature	UART0_master()
Visibility	public
Scope	instance
Documentation	Configures USART 0 for SPI mode, 3-wires, master mode, on port P3.1, P3.2, P3.3.

isEnabled	
Signature	isEnabled() : bool
Visibility	public
Scope	instance
Code Body	#if MSP430 return ((*(this->PORT+(&P1IN-&P1OUT)) & this->BIT_CS) == this->BIT_CS); #endif #if CHIPCON #endif

getErrorPin	
Signature	getErrorPin() : bool
Visibility	public
Scope	instance
Documentation	Returns true if **Error Indicator** (pin **P1.1**) is high; false otherwise.
Code Body	#if MSP430 return ((*(this->PORT+(&P1IN-&P1OUT)) & this->BIT_ERR) == this->BIT_ERR); #endif #if CHIPCON #endif

init	
Signature	init(modePORT : byte, BIT_CS : byte, BIT_ERR : byte) : void
Visibility	public
Scope	instance
Documentation	Configures UART for either: • SPI mode, 3-wires, master mode, on port P3.1, P3.2, P3.3. (when def_SPI_MODE is defined) • OBDB mode(when def_OBDB_MODE is defined) MUST be called at boot before enabling interrupts !!!

Code Body	#if MSP430
	this->PORT = &PORT;
	this->BIT_CS = BIT_CS;
	this->BIT_ERR = BIT_ERR;
	#endif
	
	#if CHIPCON
	
	#endif

	init (mode)

2.9. Class - Master_Processor

Name	Value
Name	Master_Processor
Stereotypes	SW

Attributes

CLOCK_FREQ	
Signature	+CLOCK_FREQ : long const = 1048576
Visibility	public
Scope	classifier

2.10. Class - UARTA0

Name	Value
Name	UARTA0
Stereotypes	SW
Documentation	The low-level UART drivers fro SPI protocol, to read/write data and commands into UART and to enable/disable communication. This class is processor-dependent. At present, it supports both: MSP430FG439 MSP430F149 Chipcon ??? by using the corresponding IAR compiler. It handles the four pins of the SPI (**SCK**, **MISO**, **MOSI**, **CS**) and the CS input pin has to be internally connected to two pins of the **System** processor (**CS** and **CS_replica**)

Attributes

BAUD_RATE	
Signature	+BAUD_RATE : ulong const = 9600
Visibility	public
Scope	classifier
Documentation	Baud rate, in bps

PULSE_WIDTH	
Signature	+PULSE_WIDTH : ushort const = 100
Visibility	public
Scope	classifier
Documentation	Pulse width for IrDA, in ns

Operations

init	
Signature	init(clock_freq : ulong, mode : UART_MODES, baud_rate : ulong)
Visibility	public
Scope	classifier

Code Body	#if MSP430x43x
	  switch (mode) {
	  case SPI_MASTER_MODE:
	  U0CTL = CHAR | SYNC | SWRST | MM; // sets USART0 for 8-bit, synchronous, SPI mode, Master
	  U0TCTL = SSEL1 | STC; // sets USART0 for:
	  // clock polarity (writes on rising edge; reads on falling edge),
	  // clock phase (clock=0 when inactive)
	  // 3-wire mode
	  // SMCLK clock (master clock)
	  U0BR0 = (byte) (clock_freq / baud_rate); // baud rate
	  U0BR1 = (byte) (((short)(clock_freq / baud_rate)) >> 8); // baud rate
	  U0MCTL = 0x00; // must be 0 in SPI mode
	  U0CTL &= ~SWRST; // SW reset of SPI
	  break;
	  case SPI_SLAVE_MODE:
	  U0CTL = CHAR | SYNC | SWRST; // sets USART0 for 8-bit, synchronous, SPI mode, Slave
	  U0TCTL = 0; // sets USART0 for:
	  // clock polarity (writes on rising edge; reads on falling edge),
	  // clock phase (clock=0 when inactive)
	  // 4-wire mode
	  // UCLK clock (for slave)
	  U0BR0 = 0x00; // baud rate (0 for slave)
	  U0BR1 = 0x00; // baud rate (0 for slave)
	  U0MCTL = 0x00; // must be 0 in SPI mode
	  U0CTL &= ~SWRST; // SW reset of SPI
	  break;
	  case RS232_MODE:
	  //TBD!
	  U0CTL = CHAR | SYNC | SWRST | MM; // sets USART0 for 8-bit, synchronous, SPI mode, Master
	  U0CTL &= ~SWRST; // SW reset of SPI
	  break;
	  case I2C_MASTER_MODE:
	  U0CTL &= ~SWRST; // SW reset of SPI
	  break;
	  case I2C_SLAVE_MODE:
	  U0CTL &= ~SWRST; // SW reset of SPI
	  break;

```
  case IRDA_MODE:
    U0CTL &= ~SWRST;              // SW reset of SPI
    break;
  }
#endif

#if MSP430x24x || MSP430x54x
  if ((mode == SPI_MASTER_MODE) || (mode ==
SPI_SLAVE_MODE)) {
    UCA0CTL1 = UCSSEL1 | UCSSEL0 // clock from SMCLK
      | 0*UCRXEIE // erroneous characters rejected
         | 0*UCBRKIE // break characters set interrupt flag
           | 0*UCDORM // not dormant; all characters are received
             | 0*UCTXADDR // all frames are data; no address
               | 0*UCTXBRK // frames are not break
                    | UCSWRST; // SW reset
    UCA0CTL0 = 0*UCCKPH // writes on first (rising) edge of UCLK;
reads o nsecond (falling) edge
      | 0*UCCKPL // clock polarity (active high)
         | 0*UCMSB // LSB first
           | 0*UC7BIT // eight bits
             | UCMST // master
               | 0*UCMODE1 | 0*UCMODE0 // 3-pin SPI mode
                    | UCSYNC; // synchronous mode
    UCA0BR0 = (byte) (clock_freq / baud_rate);                // baud
rate
    UCA0BR1 = (byte) (((short)(clock_freq / baud_rate)) >> 8); //
baud rate
    UCA0MCTL = UCOS16; // Oversampling mode; temporarily
disabled modulation TBD
  }
// NOT IMPLEMENTED in UART A0
//                              if ((mode == I2C_MASTER_MODE) ||
(mode == I2C_SLAVE_MODE)) {
//                              UCA0CTL1 = UCSSEL1 | UCSSEL0 //
clock from SMCLK
//                              | UCTR // transmitter
//                              | 0*UCTXNACK // acknowledge select
(TBD)
//                              | 0*UCTXSTP // does not generates stop bit
//                              | 0*UCTXSTT // does not generates start bit
//                              | UCSWRST; // SW reset
//                              UCA0CTL0 = 0*UCA10 // 7-bits
masteraddressing
//                              | 0*UCSLA10 // 7-bits slave addressing
//                              | 0*UCMM // single master
//                              | UCMST // master
//                              | UCMODE1 | UCMODE0 // I2C mode
//                              | UCSYNC; // synchronous mode
//                              UCA0BR0 = (byte) (clock_freq /
baud_rate);                 // baud rate
//                              UCA0BR1 = (byte) (((short)(clock_freq /
baud_rate)) >> 8); // baud rate
//                              UCA0I2C0A = 0; // own address
//                              UCA0I2CSA = 0; // slave address
//                              }

  if ((mode == SPI_SLAVE_MODE) || (mode ==
I2C_SLAVE_MODE)) {
    UCA0CTL0 &= ~UCMST; // slave
```

	}
	if ((mode == RS232_MODE) || (mode == IRDA_MODE)) {
	UCA0CTL1 = UCSSEL1 | UCSSEL0 // clock from SMCLK
	| 0*UCRXEIE // erroneous characters rejected
	| 0*UCBRKIE // break characters reset interrupt flag
	| 0*UCDORM // not dormant; all characters are received
	| 0*UCTXADDR // all frames are data; no address
	| 0*UCTXBRK // frames are not break
	| UCSWRST; // SW reset
	UCA0CTL0 = UCPEN // parity enabled
	| 0*UCPAR // odd parity
	| 0*UCMSB // LSB first
	| 0*UC7BIT // eight bits
	| 0*UCSPB // one stop bit
	| 0; // UART mode
	UCA0BR0 = (byte) (clock_freq / baud_rate / 16); //
	baud rate
	UCA0BR1 = (byte) (((short)(clock_freq / baud_rate / 16)) >> 8); //
	baud rate
	UCA0MCTL = UCOS16; // Oversampling mode; temporarily
	disabled modulation TBD
	if ((mode == IRDA_MODE)) {
	UCA0IRTCTL = ((((PULSE_WIDTH * (clock_freq / 1000)) /
	1000000) & 0x3F) << 2) // pulse width for IrDA mode
	| UCIRTXCLK // IrDA pulse clock select (TBD)
	| UCIREN; // enable IrDA encoder/decoder
	UCA0IRRCTL = ((((PULSE_WIDTH/2 * (clock_freq / 1000)) /
	1000000) & 0x3F) << 2) // pulse width for IrDA mode
	| 0*UCIRRXPL // IrDA pulse is positive
	| UCIRRXFE; // enable IrDA receiver filter
	}
	}
	UCA0STAT = 0; // clear all status bits
	UCA0ABCTL = 0; // disable auto baudrate detector
	UCA0CTL1 &= ~UCSWRST; // remove SW reset
	#endif
	
	disable(mode); // Module disable

UARTA0	
Signature	UARTA0()
Visibility	public
Scope	instance
Documentation	Enables (if true) or disables (if false): UART TX interrupt (enTXinterrupt) UART RX interrupt (enRXinterrupt)

Code Body	disable(RS232_MODE);
	init(1000000, RS232_MODE);
	enableInterrupts(false, false);

writeData

Signature	writeData(data : byte) : void
Visibility	public
Scope	classifier
Documentation	It writes the value of data into the UART0/UARTA0 data register. Data is automatically sent when next data exchange starts, that is: for the master, as soon as the previous transmission has terminated; for the SPI slave, as soon as the master sends its first clock bit. If no data is written in time, the UART sends an unpredictable value. The UART has to be enabled to allow transmission (first call to init(), then to enable())

Code Body	#if MSP430x43x
	U0TXBUF = data;
	#endif
	
	#if MSP430x24x || MSP430x54x
	UCA0TXBUF = data;
	#endif

writeData	
Signature	writeData(data : byte, CRC : byte) : void
Visibility	public
Scope	classifier
Documentation	It first writes the value of data into the UART0/UARTA0 data register, as documented in writeData() operation. Then it updates the CRC argument according the the chosen algorithm (bit-wise EXOR).
Code Body	writeData(data); CRC ^= data;

readData	
Signature	readData() : byte
Visibility	public
Scope	classifier
Documentation	It reads and returns received data from UART0/UARTA0 data register. The UART has to be enabled to allow reception (first call to init(), then to enable()) The caller shall first check that a byte has been received and not yet read, by means of the isRXready() operation. If no data has been received yet, it returns an unpredictable value.
Code Body	#if MSP430x43x return U0RXBUF; #endif

	#if MSP430x24x || MSP430x54x return UCA0RXBUF; #endif

readData	
Signature	readData(CRC : byte) : byte
Visibility	public
Scope	classifier
Documentation	It first reads and returns received data from UART data register, as documented for the **readData()** operation. Then it updates the **CRC** register according to the chosen algorithm (bit-wise EXOR). If no data has been received yet, it returns an unpredictable value and updates CRC accordingly.
Code Body	byte tmp; tmp = readData(); CRC ^= tmp; return tmp;

isTXready	
Signature	isTXready() : bool
Visibility	public
Scope	classifier
Documentation	Returns **true** if transmitter buffer is ready to receive a new byte for transmission; **false** otherwise.

Code Body	#if MSP430x43x
	  return ((IFG1 & UTXIFG0) == UTXIFG0);
	#endif
	
	#if MSP430x24x
	  return ((IFG2 & UCA0TXIFG) == UCA0TXIFG);
	#endif
	
	#if MSP430x54x
	  return ((UCA0IFG & UCTXIFG) == UCTXIFG);
	#endif

isTXempty	
Signature	isTXempty() : bool
Visibility	public
Scope	classifier
Documentation	Returns **true** when all data in the TX buffer and in the TX shift register have been sent; **false** otherwise.

	The caller shall wait until isTXempty() returns true before calling disable(), otherwise data transmission can be interrupted before completion.

Code Body	#if MSP430x43x
	  return ((U0TCTL & TXEPT) == TXEPT);
	#endif
	
	#if MSP430x24x || MSP430x54x
	  return ((UCA0STAT & UCBUSY) == 0);
	#endif

isRXready	
Signature	isRXready() : bool
Visibility	public
Scope	classifier
Documentation	Returns **true** if receiver buffer is full, that is, a byte has been received but not yet read; **false** otherwise.

Code Body	#if MSP430x43x
	  return ((IFG1 & URXIFG0) == URXIFG0);
	#endif
	
	#if MSP430x24x
	  return ((IFG2 & UCA0RXIFG) == UCA0RXIFG);
	#endif
	
	#if MSP430x54x
	  return ((UCA0IFG & UCRXIFG) == UCRXIFG);
	#endif

enableInterrupts	
Signature	enableInterrupts(enTXinterrupt : bool, enRXinterrupt : bool) : void
Visibility	public
Scope	classifier
Documentation	Enables (if **true**) or disables (if **false**): • UART TX interrupt (**enTXinterrupt**) • UART RX interrupt (**enRXinterrupt**)

Code Body	#if MSP430x43x
	  if (enTXinterrupt) {
	  IE1 |= UTXIE0; // enables TX interrupt
	  }
	  else
	  {
	  IE1 &= ~UTXIE0; // disables TX interrupt
	  }
	  if (enRXinterrupt) {
	  IE1 |= URXIE0; // enables RX interrupt
	  }

	else
	{
	IE1 &= ~URXIE0; // disables RX interrupt
	}
	#endif
	
	#if MSP430x24x
	if (enTXinterrupt) {
	IE2 |= UCA0TXIE; // enables TX interrupt
	}
	else
	{
	IE2 &= ~UCA0TXIE; // disables TX interrupt
	}
	if (enRXinterrupt) {
	IE2 |= UCA0RXIE; // enables RX interrupt
	}
	else
	{
	IE2 &= ~UCA0RXIE; // disables RX interrupt
	}
	#endif
	
	#if MSP430x54x
	if (enTXinterrupt) {
	UCA0IE |= UCTXIE; // enables TX interrupt
	}
	else
	{
	UCA0IE &= ~UCTXIE; // disables TX interrupt
	}
	if (enRXinterrupt) {
	UCA0IE |= UCRXIE; // enables RX interrupt
	}
	else
	{
	UCA0IE &= ~UCRXIE; // disables RX interrupt
	}
	#endif

init	
Signature	init(clock_freq : ulong, mode : UART_MODES)
Visibility	public
Scope	classifier
Code Body	init (clock_freq, mode, BAUD_RATE);

msbFirst	
Signature	msbFirst()
Visibility	public
Scope	classifier
Code Body	#if MSP430x43x

	#endif
	
	#if MSP430x24x || MSP430x54x
	  UCA0CTL1 |= UCSWRST; // set SW reset
	  UCA0CTL0 |= UCMSB; // MSB first
	  UCA0CTL1 &= ~UCSWRST; // remove SW reset
	#endif

enable	
Signature	enable(mode : UART_MODES) : void
Visibility	public
Scope	classifier
Documentation	Activates communication using SPI protocol, configured as master: enables and resets UART (therefore clears TX and RX buffers) sets output pins (SCK, MOSI) to low impedance configures UART for transmitting and receiving SPI protocol • activates interrupt on transmitting byte (if **enTXinterrupt** is true) • activates interrupt on received byte (if **enRXinterrupt** is true)

Code Body	#if MSP430x43x
	  U0CTL |= SWRST; // resets the UART, therefore clears TX and RX buffers
	  if ((mode == SPI_MASTER_MODE) || (mode == SPI_SLAVE_MODE)) {
	  P3SEL |= (BIT1 | BIT2 | BIT3); // P3.1,2,3 set as normal input pins (MOSI, MISO, SCK)
	  }
	// NOT IMPLEMENTED in UART A0
	// if ((mode == I2C_MASTER_MODE) || (mode == I2C_SLAVE_MODE)) {
	// P3SEL |= (BIT1 | BIT2 | BIT3); // UART pins set for UART usage
	// }
	
	  if ((mode == RS232_MODE) || (mode == IRDA_MODE)) {
	  P3SEL |= (BIT1 | BIT2); // P3.1,2,3 set as normal input pins (MOSI, MISO, SCK)
	  }
	  ME1 |= USPIE0; // Module enable
	  U0CTL &= ~SWRST;
	#endif
	
	#if MSP430x24x || MSP430x54x
	  UCA0CTL1 |= UCSWRST; // SW reset
	  if ((mode == SPI_MASTER_MODE) || (mode == SPI_SLAVE_MODE)) {
	  P3SEL |= (BIT4 | BIT5 | BIT0); // UART pins set for UART usage
	  }
	// NOT IMPLEMENTED in UART A0
	// if ((mode == I2C_MASTER_MODE) || (mode == I2C_SLAVE_MODE)) {
	// P3SEL |= (BIT4 | BIT5 | BIT0); // UART pins

	set for UART usage
	// }
	
	  if ((mode == RS232_MODE) || (mode == IRDA_MODE)) {
	  P3SEL |= (BIT4 | BIT5); // UART pins set for UART usage
	  }
	  UCA0CTL1 &= ~UCSWRST; // SW reset
	#endif
	
	#if CHIPCON
	
	#endif

disable	
Signature	disable(mode : UART_MODES) : void
Visibility	public
Scope	classifier
Documentation	Deactivates communication using SPI protocol: • disables UART, interrupting communication. The user shall wait until all data have been exchanged (namely, until **isTXempty()** returns true). sets output pins (MISO) to high impedance deactivates interrupts on transmitting byte and received byte

Code Body	enableInterrupts (false, false);
	
	#if MSP430x43x
	  U0CTL |= SWRST; // resets the UART, therefore clears TX and RX buffers
	  ME1 &= ~USPIE0; // Module disable
	  if ((mode == SPI_MASTER_MODE) || (mode == SPI_SLAVE_MODE)) {
	  P3SEL &= ~(BIT1 | BIT2 | BIT3); // P3.1,2,3 set as normal inputs
	  P3DIR &= ~(BIT1 | BIT2 | BIT3); // P3.1,2,3 set as normal inputs
	  }
	// NOT IMPLEMENTED in UART A0
	// if ((mode == I2C_MASTER_MODE) || (mode == I2C_SLAVE_MODE)) {
	// P3SEL &= ~(BIT1 | BIT2 | BIT3); // P3.1,2,3 set as normal inputs
	// P3DIR &= ~(BIT1 | BIT2 | BIT3); // P3.1,2,3 set as normal inputs
	// }
	
	  if ((mode == RS232_MODE) || (mode == IRDA_MODE)) {
	  P3SEL &= ~(BIT1 | BIT2); // P3.1,2,3 set as normal inputs
	  P3DIR &= ~(BIT1 | BIT2); // P3.1,2,3 set as normal inputs
	  }
	  U0CTL &= ~SWRST;
	#endif
	
	#if MSP430x24x || MSP430x54x
	  UCA0CTL1 |= UCSWRST; // SW reset

	UCA0CTL1 &= ~UCSWRST; // SW reset
	if ((mode == SPI_MASTER_MODE) || (mode == SPI_SLAVE_MODE)) {
	P3SEL &= ~(BIT4 | BIT5 | BIT0); // UART pins set for normal usage
	P3DIR &= ~(BIT4 | BIT5 | BIT0); // UART pins set as normal inputs
	}
	// NOT IMPLEMENTED in UART A0
	// if ((mode == I2C_MASTER_MODE) || (mode == I2C_SLAVE_MODE)) {
	// P3SEL &= ~(BIT4 | BIT5 | BIT0); // UART pins set for normal usage
	// }
	
	if ((mode == RS232_MODE) || (mode == IRDA_MODE)) {
	P3SEL &= ~(BIT4 | BIT5); // UART pins set for normal usage
	P3DIR &= ~(BIT4 | BIT5); // UART pins set as normal inputs
	}
	#endif

radiationCheck	
Signature	radiationCheck() : bool
Visibility	public
Scope	classifier
Documentation	Checks that ionizing radiation (or any other unpredictable cause) has not affected UART configuration. It returns **true** in normal conditions; **false** when any configuration has changed. In the latter case, it calls the constructor again, trying to correct the anomaly.
Code Body	#if MSP430 // da rifare TBD !!! return true; #endif #if CHIPCON #endif

RX	
Signature	RX()
Visibility	public
Scope	instance

TX	
Signature	TX()
Visibility	public
Scope	instance

MISO	
Signature	MISO()
Visibility	public
Scope	instance

MOSI	
Signature	MOSI()
Visibility	public
Scope	instance

SCK	
Signature	SCK()
Visibility	public
Scope	instance

2.11. Class - CommandInterpreter

Name	Value
Name	CommandInterpreter
Stereotypes	SW

Attributes

buffers	
Signature	-buffers : Buffers
Visibility	private
Scope	classifier

Operations

interpret	
Signature	interpret(command : t_Commands, error : t_LastError)
Visibility	public
Scope	instance
Documentation	Is called at then end of data exchange, when the CS is deselected and the CRC has been received correctly, to allow the **Designer** to execute some **Designer**-specific code. It receives the **command**, an indication of the last **error** encountered. This allows also to execute the **Designer**-specific

	code even in case of error. It is therefore under the **Designer** responsibility to check that **error==NO_ERROR**, if required.

Code Body	byte * bufferWrite, * bufferRead;
	short i;
	ushort length;
	switch (command) {
	case Bk1A1_Common_Codes::CMD_WRITE_DATA_0:
	#ifdef DEBUG
	P5OUT ^= BIT1;
	#endif
	bufferWrite = buffers.get (command, length);
	if (bufferWrite != NULL) {
	bufferRead = buffers.lock
	(Bk1A1_Common_Codes::CMD_READ_DATA_0);
	if (bufferRead != NULL) {
	for (i=0; ((i<length)&&(i<256)); i++) {
	bufferRead[i] = bufferWrite[i];
	}
	}
	}
	buffers.release (bufferWrite);
	buffers.ready (Bk1A1_Common_Codes::CMD_READ_DATA_0,
	length);
	break;
	}

isReadData	
Signature	isReadData(command : t_Commands) : bool
Visibility	public
Scope	classifier
Documentation	Returns true if **command** is any **Read Data** command.
Code Body	return ((command & 0xC0) == 0x80);

isWriteData	
Signature	isWriteData(command : t_Commands) : bool
Visibility	public
Scope	classifier
Documentation	Returns true if **command** is any **Write Data** command.
Code Body	return ((command & 0xC0) == 0x40);

usesCommandOnlyProtocol	
Signature	usesCommandOnlyProtocol(command : t_Commands) : bool
Visibility	public
Scope	classifier
Documentation	Returns true if **command** uses **Command Only** protocol.
Code Body	return ((command & 0xC0) == 0xC0) || ((command & 0x30) ==

	0x30);

usesReadDataProtocol	
Signature	usesReadDataProtocol(command : t_Commands) : bool
Visibility	public
Scope	classifier
Documentation	Returns true if **command** uses **Read Data** protocol.
Code Body	return ((command & 0xC0) == 0x80) || ((command & 0x30) == 0x20);

usesWriteDataProtocol	
Signature	usesWriteDataProtocol(command : t_Commands) : bool
Visibility	public
Scope	classifier
Documentation	Returns true if **command** uses **Write Data** protocol.
Code Body	return ((command & 0xC0) == 0x40) || ((command & 0x30) == 0x10);

2.12. Class - Buffers

Name	**Value**
Name	Buffers
Stereotypes	SW
Documentation	This class implements a set of fixed-length buffers, to be used for **Write Module Data** and **Read Module Data** use cases, respectively. Up to **MAXBUFFERS** buffers of 256 bytes each can be allocated (provided that the processor has enough memory). Each buffer can be used for either **Write** or **Read** operation, although the class guarantees that at least one buffer is allocated for both **Write** and **Read** use case. In particular, for either **Write** (if **command** = CMD_WRITE_DATA_x) or **Read** (for **command** = CMD_READ_DATA_x): throws away any buffer of that type still being written but not yet complete (namely, for which the ready() operation has not yet been called, if any) finds the first available buffer of that type locks it and declares it being written; the user shall then fill the buffer and call the ready() operation when finished. After that, the buffer is queued... returns a pointer to it Returns **null** if no buffer of the chosen type is available or command is not supported.

Attributes

MAXBUFFERS	
Signature	-MAXBUFFERS : byte const = 4
Visibility	private
Scope	classifier
Documentation	Sets the number of buffers which are allocated. It must be 2 = **MAXBUFFERS** = 16. More buffers might optionally be allocated if the size of locked and busy is increased.

len	
Signature	-len : byte[MAXBUFFERS]
Visibility	private
Scope	classifier
Documentation	Vector of buffer lengths. They contain actual length of associated buffer MINUS 1.

buffs	
Signature	-buffs : byte[MAXBUFFERS][256]
Visibility	private
Scope	classifier
Documentation	The set of available buffers (between 2 and 16). buffs[0] can only be used for CMD_WRITE_DATA. buffs[1] can only be used for CMD_READ_DATA. The other buffers can be used by either (or other) commands.

locked	
Signature	-locked : ushort
Visibility	private
Scope	classifier
Documentation	Indicates which of the buffers are locked. A buffer is locked between the call to lock() up to the call to release(). Each bit of this vaiable is associated with a diffeent buffer: locked.0 --> buffs[0] locked.1 --> buffs[1] etc.

busyWrite	
Signature	-busyWrite : ushort

Visibility	private
Scope	classifier
Documentation	Indicates which of the buffers are busy for CMD_WRITE_DATA. A buffer is locked between the call to lock() up to the call to ready(). Each bit of this vaiable is associated with a diffeent buffer: busyWrite.0 --> buffs[0] busyWrite.1 --> buffs[1] etc.

busyRead	
Signature	-busyRead : ushort
Visibility	private
Scope	classifier
Documentation	Indicates which of the buffers are busy for CMD_READ_DATA. A buffer is locked between the call to lock() up to the call to ready(). Each bit of this vaiable is associated with a diffeent buffer: busyRead.0 --> buffs[0] busyRead.1 --> buffs[1] etc.

stackWrite	
Signature	-stackWrite : byte[MAXBUFFERS]
Visibility	private
Scope	classifier

stackRead	
Signature	-stackRead : byte[MAXBUFFERS]
Visibility	private
Scope	classifier

Operations

lock	
Signature	lock(command : t_Commands) : byte
Visibility	public
Scope	instance
Documentation	Allocates and locks one of the MAXBUFFERS available buffers. There are two types of buffers, to be used for **Write Module Data** and **Read Module Data** use cases, respectively. In particular, for either **Write** (if **command** = CMD_WRITE_DATA_x) or **Read** (for **command** =

	CMD_READ_DATA_x): throws away any buffer of that type still being written but not yet complete (namely, for which the ready() operation has not yet been called, if any) finds the first available buffer of that type locks it and declares it being written; the user shall then fill the buffer and call the ready() operation when finished. After that, the buffer is queued... returns a pointer to it Returns **null** if no buffer of the chosen type is available or command is not supported.

ready	
Signature	ready(command : t_Commands, length : ushort) : bool
Visibility	public
Scope	instance
Documentation	Declares that the buffer pointed to by **buffer** has been filled in with **length** bytes and is ready to be read Returns **false** if either no buffer or more than one buffer has been allocated. In the latter case declares that all of them are ready. Otherwise it return **true**.

get	
Signature	get(command : t_Commands, length : ushort) : byte
Visibility	public
Scope	instance
Documentation	Returns pointer to first buffer containing data: • written, if **command** == CMD_WRITE_DATA_x (with x = 0 through 7). • to be read, if **command** == CMD_READ_DATA_x (with x = 0 through 7). Returns NULL in case of no buffer containing data. The argument length also returns the amount of data actually written in that buffer

release	
Signature	release(buffer : byte) : bool
Visibility	public
Scope	instance
Documentation	Releases buffer identified by parameter **buffer**. No more reading can be done on that buffer, which is then available for further use (via the **lock()** operation) Returns false is no open buffer matches the parameter **buffer**. Otherwise it returns **true**.

reset	
Signature	reset() : void
Visibility	public
Scope	instance
Documentation	Clears all buffers, stacks and flags

Buffers	
Signature	Buffers()
Visibility	public
Scope	instance

2.13. Class - t_LastError

Name	Value
Name	t_LastError
Stereotypes	SW, enumeration, TBD
Documentation	Defines codes for the last error which occurred in the slave.

Attributes

CRC_ERROR	
Signature	-CRC_ERROR
Visibility	private
Scope	instance
Documentation	It indicates that an error has been detected in the CRC of the last message or command coming from **CPU** actor.

LENGTH_ERROR	
Signature	-LENGTH_ERROR
Visibility	private
Scope	instance
Documentation	It indicates that the lenght of the last message/command does not match the expected length. Either: • for **Command Only** messages, there were either more or less byte than the expected sequence. • for **Write Data** messages, the CS has terminated the transfer either too early or too late, when compared with the length field of the message. • for **Read Data** messages, it indicates that the **CPU** actor has terminated the communication either too early or too late to

	transfer the amount of dta declared by the **System**.

NO_BUFFER_AVAILABLE	
Signature	-NO_BUFFER_AVAILABLE
Visibility	private
Scope	instance
Documentation	During **Write Module Data** use case, it indicates that there is no buffer available fro storing data.

WRONG_HOUSEKEEPING	
Signature	-WRONG_HOUSEKEEPING
Visibility	private
Scope	instance

NO_DATA_AVAILABLE	
Signature	-NO_DATA_AVAILABLE
Visibility	private
Scope	instance

TIMEOUT	
Signature	-TIMEOUT
Visibility	private
Scope	instance

WRONG_LENGTH	
Signature	-WRONG_LENGTH
Visibility	private
Scope	instance

WRONG_COMMAND	
Signature	-WRONG_COMMAND
Visibility	private
Scope	instance

WRONG_CRC	
Signature	-WRONG_CRC
Visibility	private

Scope	instance

WRONG_BUFFER	
Signature	-WRONG_BUFFER
Visibility	private
Scope	instance

NO_ERROR	
Signature	-NO_ERROR = 0
Visibility	private
Scope	instance

UNRECOGNIZED_COMMAND	
Signature	-UNRECOGNIZED_COMMAND
Visibility	private
Scope	instance

2.14. Class - Result_1B8

Name	**Value**
Name	Result_1B8
Stereotypes	SW
Documentation	It contains the results of the analisis of the method analyzePMT of Manager_1B8.

Attributes

Buvw	
Signature	+Buvw : t_sensor[3]
Visibility	public
Scope	classifier
Documentation	Earth's magnetic field in UVW coordinates. Buvw[0] = U coordinate, Buvw[1] = V coordinate, Buvw[2] = W coordinate.

Spin	
Signature	+Spin : t_sensor[3]
Visibility	public

Scope	classifier
Documentation	Satellite's spin in UVW coordinates. Spin[0] = U coordinate, Spin[1] = V coordinate, spin[2] = W coordinate.

SunPos	
Signature	+SunPos : t_sensor[3]
Visibility	public
Scope	classifier
Documentation	Sun's position relative to the satellite's center, in UVW coordinates. SunPos[0] = U coordinate, SunPos[1] = V coordinate, SunPos[2] = W coordinate.

EarthPos	
Signature	+EarthPos : t_sensor[3]
Visibility	public
Scope	classifier
Documentation	Earth's center's position relative to the satellite's center, in UVW coordinates. EarthPos[0] = U coordinate, EarthPos[1] = V coordinate, EarthPos[2] = W coordinate.

IBVmean	
Signature	+IBVmean : t_sensor
Visibility	public
Scope	classifier
Documentation	Arithmetic mean of the values in INTERNAL_BUS_VOLTAGE.

IBVdev	
Signature	+IBVdev : t_sensor
Visibility	public
Scope	classifier
Documentation	Standard deviation of the values in INTERNAL_BUS_VOLTAGE.

IBVmin	
Signature	+IBVmin : t_sensor[2]
Visibility	public
Scope	classifier

Documentation	IBVmin[0] = minimum value of the field INTERNAL_BUS_VOLTAGE; IBVmin[1] = tile corrisponding to the minimum of INTERNAL_BUS_VOLTAGE.

IBVmax	
Signature	+IBVmax : t_sensor[2]
Visibility	public
Scope	classifier
Documentation	IBVmax[0] = maximum value of the field INTERNAL_BUS_VOLTAGE; IBVmax[1] = tile corrisponding to the maximum of INTERNAL_BUS_VOLTAGE.

STCmean	
Signature	+STCmean : t_sensor
Visibility	public
Scope	classifier
Documentation	Arithmetic mean of the values in SOLAR_TEMPERATURE_CENTER.

STSmean	
Signature	+STSmean : t_sensor
Visibility	public
Scope	classifier
Documentation	Arithmetic mean of the values in SOLAR_TEMPERATURE_SIDE.

BTmean	
Signature	+BTmean : t_sensor
Visibility	public
Scope	classifier
Documentation	Arithmetic mean of the values in BATTERY_TEMPERATURE.

PTmean	
Signature	+PTmean : t_sensor
Visibility	public
Scope	classifier
Documentation	Arithmetic mean of the values in POWERCIRCUITS_TEMPERATURE.

ACSTmean	
Signature	+ACSTmean : t_sensor
Visibility	public
Scope	classifier
Documentation	Arithmetic mean of the values in ACS_TEMPERATURE.

2.15. Class - ID_codes

Name	Value
Name	ID_codes
Stereotypes	SW, enumeration
Documentation	ID_codes uniquely identifying the type of tile. ID_codes do not identify a single tile, but its type. A satellite will likely contain a number of tiles of each type. Each tile will then be identified by a unique tile address.

Attributes

Bk1B8_PMT_V0	
Signature	-Bk1B8_PMT_V0 = 0x1
Visibility	private
Scope	instance
Documentation	ID_codes for Power Management and Attitude Control Tile (usually called 1B8 Power Management Tile); version 0.

Bk1B9_TLC_V0	
Signature	-Bk1B9_TLC_V0 = 0x11
Visibility	private
Scope	instance
Documentation	ID_codes for On Board Computer and Telecommunication Tile (usually called 1B9 Telecommunication Tile); version 0.

Bk1BA_PAYLOAD	
Signature	-Bk1BA_PAYLOAD = 0xF1
Visibility	private
Scope	instance
Documentation	ID_codes for a generic payload

2.16. Class - HK_fields_1B8

Name	Value
Name	HK_fields_1B8
Stereotypes	SW, enumeration
Documentation	Codes for housekeeping of 1B8 Power management Tile. Each code identifies the location of the corresponding value in the 1B8 housekeeping vector

Attributes

EARTH_SENSOR_X	
Signature	-EARTH_SENSOR_X
Visibility	private
Scope	instance
Documentation	Housekeeping code for the **x** coordinate of the unit-length versor pointing to the earth center. It is associated with use case **Get SunSensor** (see 1B8 project). It is expressed as a t_sensor type in units of 1/RANGE_TRIGO. If the earth is not visible or not easily detectable, it returns NaN.

EARTH_SENSOR_Y	
Signature	-EARTH_SENSOR_Y
Visibility	private
Scope	instance
Documentation	Housekeeping code for the **y** coordinate of the unit-length versor pointing to the earth center. It is associated with use case **Get SunSensor** (see 1B8 project). It is expressed as a t_sensor type in units of 1/RANGE_TRIGO. If the earth is not visible or not easily detectable, it returns NaN.

SUN_SENSOR_X	
Signature	-SUN_SENSOR_X
Visibility	private
Scope	instance
Documentation	Housekeeping code for the **x** coordinate of the unit-length versor pointing to the sun center. It is associated with use case **Get SunSensor** (see 1B8 project). It is expressed as a t_sensor type in units of 1/RANGE_TRIGO. If the earth is not visible or not easily detectable, it returns NaN.

SUN_SENSOR_Y	
Signature	-SUN_SENSOR_Y

Visibility	private
Scope	instance
Documentation	Housekeeping code for the **y** coordinate of the unit-length versor pointing to the sun center. It is associated with use case **Get SunSensor** (see 1B8 project). It is expressed as a t_sensor type in units of 1/RANGE_TRIGO. If the earth is not visible or not easily detectable, it returns NaN.

SOLAR_VOLTAGE	
Signature	-SOLAR_VOLTAGE
Visibility	private
Scope	instance
Documentation	Housekeeping code for measured voltage across solar panel. It is associated with use case **Get Solar Voltage** (see 1B8 project). It is expressed as a t_sensor type in units of 1/RANGE_TRIGO.

SOLAR_CURRENT	
Signature	-SOLAR_CURRENT
Visibility	private
Scope	instance
Documentation	HK code for measured average current from solar panel. Associated with use case **Get Housekeeping** (see 1B8 project).

SOLAR_TEMPERATURE_CENTER	
Signature	-SOLAR_TEMPERATURE_CENTER
Visibility	private
Scope	instance
Documentation	HK code for measured temperature close to the center of solar panel. Temperature is measured inside the Al, as close as possible to solar panels. Associated with use case **Get Housekeeping** (see 1B8 project).

SOLAR_TEMPERATURE_SIDE	
Signature	-SOLAR_TEMPERATURE_SIDE
Visibility	private
Scope	instance
Documentation	HK code for measured temperature close to the side of solar panel. Temperature is measured inside the Al, as close as possible to solar panels. Associated with use case **Get Housekeeping** (see 1B8 project).

BATTERY_VOLTAGE

Signature	-BATTERY_VOLTAGE
Visibility	private
Scope	instance
Documentation	HK code for measured voltage across the battery series. Associated with use case **Get Housekeeping** (see 1B8 project).

BATTERY_CURRENT_CHARGE	
Signature	-BATTERY_CURRENT_CHARGE
Visibility	private
Scope	instance
Documentation	HK code for measured current into battery from battery charger from the battery series. Associated with use case **Get Housekeeping** (see 1B8 project).

BATTERY_CURRENT_DISCHARGE	
Signature	-BATTERY_CURRENT_DISCHARGE
Visibility	private
Scope	instance
Documentation	HK code for measured current from battery from battery discharger from the battery series. Associated with use case **Get Housekeeping** (see 1B8 project).

BATTERY_TEMPERATURE	
Signature	-BATTERY_TEMPERATURE
Visibility	private
Scope	instance
Documentation	HK code for measured temperature of the batteries. Associated with use case **Get Housekeeping** (see 1B8 project).

POWERCIRCUITS_TEMPERATURE	
Signature	-POWERCIRCUITS_TEMPERATURE
Visibility	private
Scope	instance
Documentation	HK code for measured temperature of the tile. Associated with use case **Get Housekeeping** (see 1B8 project).

BATTERY_UNBALANCE	
Signature	-BATTERY_UNBALANCE
Visibility	private
Scope	instance

Documentation	HK code for difference between the voltage across the two batteries. Associated with use case **Get Housekeeping** (see 1B8 project).

BATTERY_CHARGE	
Signature	-BATTERY_CHARGE
Visibility	private
Scope	instance
Documentation	HK code for last estimated remaining charge on battery in units or mAh. Associated with use case **Get Housekeeping** (see 1B8 project).

BATTERY_STATUS	
Signature	-BATTERY_STATUS
Visibility	private
Scope	instance
Documentation	Bit 0: flag notICL_BAT_CHARGER of 1B113_Battery_Charger Bit 1: flag ACP_BAT_CHARGER of 1B113_Battery_Charger Bit 2: flag notFAULT_BAT_CHARGER of 1B113_Battery_Charger Bit 3: flag notBAT_CHARGED of 1B113_Battery_Charger Bit 4: flag notLATCHUP_ON of 1B113_Battery_Charger

BATTERY_BUS_CURRENT	
Signature	-BATTERY_BUS_CURRENT
Visibility	private
Scope	instance

INTERNAL_BUS_VOLTAGE	
Signature	-INTERNAL_BUS_VOLTAGE
Visibility	private
Scope	instance

INTERNAL_VOLTAGE_1	
Signature	-INTERNAL_VOLTAGE_1
Visibility	private
Scope	instance
Documentation	HK code for measured voltage at the output of one voltage regulator (TBD). Associated with use case **Get Housekeeping** (see 1B8 project).

INTERNAL_VOLTAGE_2	
Signature	-INTERNAL_VOLTAGE_2
Visibility	private
Scope	instance
Documentation	HK code for measured voltage at the output of one voltage regulator (TBD). Associated with use case **Get Housekeeping** (see 1B8 project).

SPIN_Z	
Signature	-SPIN_Z
Visibility	private
Scope	instance
Documentation	HK code for Tile spin around its **z-axis**. Associated with use case **Get Spin** (see 1B8 project).

REACTION_SPEED	
Signature	-REACTION_SPEED
Visibility	private
Scope	instance

REACTION_CURRENT	
Signature	-REACTION_CURRENT
Visibility	private
Scope	instance

MAGNETIC_FIELD_X	
Signature	-MAGNETIC_FIELD_X
Visibility	private
Scope	instance
Documentation	HK code for measured magnetic field along **x-axis**. Associated with use case **Get Magnetic Field** (see 1B8 project).

MAGNETIC_FIELD_Y	
Signature	-MAGNETIC_FIELD_Y
Visibility	private
Scope	instance
Documentation	HK code for measured magnetic field along **y-axis**. Associated with use case **Get Magnetic Field** (see 1B8 project). (model element not found) **MUST** be equal to (model element not

	found)+1.

ACS_TEMPERATURE	
Signature	-ACS_TEMPERATURE
Visibility	private
Scope	instance
Documentation	HK code for measured temperature of ACS subsystem of 1B8 tile. Associated with use case **Get Housekeeping**(see 1B8 project).

SHUNT_CURRENT	
Signature	-SHUNT_CURRENT
Visibility	private
Scope	instance
Documentation	HK code for measured current in magnetic asset coil. Associated with use case **Get Housekeeping** (see 1B8 project).

TOTAL_DOSE_OUTSIDE	
Signature	-TOTAL_DOSE_OUTSIDE
Visibility	private
Scope	instance
Documentation	HK code for last measured total dose outside the satellite (TID) from launch in units of 10 rads. Associated with use case **Get Housekeeping** (see 1B8 project).

TOTAL_DOSE_INSIDE	
Signature	-TOTAL_DOSE_INSIDE
Visibility	private
Scope	instance
Documentation	HK code for last measured total dose inside the satellite (TID) from launch in units of 10 rads. Associated with use case **Get Housekeeping** (see 1B8 project).

COIL_CURRENT	
Signature	-COIL_CURRENT
Visibility	private
Scope	instance
Documentation	HK code for measured current in magnetic asset coil. Associated with use case **Get Housekeeping** (see 1B8 project). (model element not found) **MUST** be either less than (model element not found)-2 or greater than (model element not found)+3,

	otherwise (model element not found) is measured while it is disabled to allow measuring (model element not found) and (model element not found).

REBOOTS	
Signature	-REBOOTS
Visibility	private
Scope	instance
Documentation	HK code for counted number of reboots. Associated with use case **Get Housekeeping** (see 1B8 project).

MEM_ERRORS	
Signature	-MEM_ERRORS
Visibility	private
Scope	instance
Documentation	HK code for counted number of memory errors. Associated with use case **Get Housekeeping** (see 1B8 project).

CRC_ERRORS	
Signature	-CRC_ERRORS
Visibility	private
Scope	instance

STACK_POINTER	
Signature	-STACK_POINTER
Visibility	private
Scope	instance

SWITCH_STATUS_1	
Signature	-SWITCH_STATUS_1
Visibility	private
Scope	instance
Documentation	HK code for status of various internal switches (TBD). Meaning of each bit is found in **t_Switches_1B8** enumeration. Associated with use case **Get Housekeeping** (see 1B8 project).

SWITCH_STATUS_2	
Signature	-SWITCH_STATUS_2
Visibility	private
Scope	instance

Documentation	HK code for status of various internal switches (TBD). Meaning of each bit is found in **t_Switches_1B8** enumeration. Associated with use case **Get Housekeeping** (see 1B8 project).

UP_TIME	
Signature	-UP_TIME
Visibility	private
Scope	instance
Documentation	HK code for time in min from last reboot. Associated with use case **Get Housekeeping** (see 1B8 project).

EXECUTED_COMMANDS	
Signature	-EXECUTED_COMMANDS
Visibility	private
Scope	instance
Documentation	HK code for number of commands executed from last reboot. Associated with use case **Get Housekeeping** (see 1B8 project).

MAGNETIC_ACTUATOR_ERRORS	
Signature	-MAGNETIC_ACTUATOR_ERRORS
Visibility	private
Scope	instance

LENGTH_HOUSEKEEPING	
Signature	+LENGTH_HOUSEKEEPING
Visibility	public
Scope	instance
Documentation	The length of the housekeeping vector, in number of **t_sensor** elements. It also indicates the number of measurements in the housekeeping vector.

EOC_INTEGRAL	
Signature	-EOC_INTEGRAL
Visibility	private
Scope	instance
Documentation	This is not an index to the housekeeping field, but it is used to index the computation of the angular momentum of the magnetic coil (with reference to command (model element not found))

SETRESET_MAGN

Signature	-SETRESET_MAGN
Visibility	private
Scope	instance
Documentation	This is not an index to the housekeeping field, but it is used to index the automatic set reset operation of the 1B21 Magnetic Attitude Subsystem.

2.17. Class - Result_1B9

Name	Value
Name	Result_1B9
Stereotypes	SW
Documentation	It contains the results of the analisis of the method analyzeOBC of class Manager_1B9.

2.18. Class - HK_fields_1B9

Name	Value
Name	HK_fields_1B9
Stereotypes	SW, enumeration
Documentation	Codes for housekeeping of 1B9 Telecommunication Tile. Each code identifies the location of the corresponding value in the 1B9 housekeeping vector

Attributes

CPU_TEMPERATURE	
Signature	-CPU_TEMPERATURE
Visibility	private
Scope	instance
Documentation	HK code for measured temperature of OBC CPU. Associated with use case **Get Housekeeping** (see 1B9 project).

PDB_VOLTAGE_1	
Signature	-PDB_VOLTAGE_1
Visibility	private
Scope	instance
Documentation	HK code for measured voltage at the Power Distribution Bus to which the CPU is connected. Associated with use case **Get Housekeeping** (see 1B9 project).

PDB_VOLTAGE_2	
Signature	-PDB_VOLTAGE_2
Visibility	private
Scope	instance
Documentation	HK code for measured voltage at the other Power Distribution Bus (the one to which the other CPU is connected). Associated with use case **Get Housekeeping** (see 1B9 project).

CPU_CURRENT	
Signature	-CPU_CURRENT
Visibility	private
Scope	instance
Documentation	HK code for measured current supply to the OBC CPU + memories and peripherals. Associated with use case **Get Housekeeping** (see 1B9 project).

RF_RSSI_437MHZ	
Signature	-RF_RSSI_437MHZ
Visibility	private
Scope	instance
Documentation	HK code for measured signal strength received by the 1B31A 437MHz RF module. Range is TBD.

RF_RSSI_2G4	
Signature	-RF_RSSI_2G4
Visibility	private
Scope	instance
Documentation	HK code for measured signal strength received by the 1B31B 2.4GHz RF module. Range is TBD.

PA_TEMP_437MHz	
Signature	-PA_TEMP_437MHz
Visibility	private
Scope	instance
Documentation	Temperature of PA of the 1B31A 437MHz link. Sensor is not linear.

PA_TEMP_2G4	
Signature	-PA_TEMP_2G4
Visibility	private

Scope	instance
Documentation	Temperature of PA of the 1B31B 2.4GHz link. Sensor is not linear.

PA_CURRENT_437MHz	
Signature	-PA_CURRENT_437MHz
Visibility	private
Scope	instance
Documentation	Supply current of the power pin of PA of the 1B31A 437MHz link. Range 0-2A (TBD).

PA_CURRENT_2G4	
Signature	-PA_CURRENT_2G4
Visibility	private
Scope	instance
Documentation	Supply current of the power pin of PA of the 1B31B 2.4GHz link. Range 0-2A (TBD).

PA_VOLTAGE_437MHz	
Signature	-PA_VOLTAGE_437MHz
Visibility	private
Scope	instance
Documentation	Supply voltage of the power pin of PA of the 1B31A 437MHz link. Range 0-5V (TBD).

PA_VOLTAGE_2G4	
Signature	-PA_VOLTAGE_2G4
Visibility	private
Scope	instance
Documentation	Supply voltage of the power pin of PA of the 1B31B 2.4GHz link. Range 0-5V (TBD).

RF_VOLTAGE_437MHz	
Signature	-RF_VOLTAGE_437MHz
Visibility	private
Scope	instance
Documentation	Supply voltage of the 1B31A 437MHz RF module.

RF_VOLTAGE_2G4	
Signature	-RF_VOLTAGE_2G4

Visibility	private
Scope	instance
Documentation	Supply voltage of the 1B31B 2.4GHz RF module.

REBOOTS	
Signature	-REBOOTS
Visibility	private
Scope	instance
Documentation	HK code for counted number of reboots. Associated with use case **Get Housekeeping** (see 1B8 project).

MEM_ERRORS	
Signature	-MEM_ERRORS
Visibility	private
Scope	instance
Documentation	HK code for counted number of memory errors. Associated with use case **Get Housekeeping** (see 1B8 project).

CRC_ERRORS	
Signature	-CRC_ERRORS
Visibility	private
Scope	instance

STACK_POINTER	
Signature	-STACK_POINTER
Visibility	private
Scope	instance

BACKDOOR	
Signature	-BACKDOOR
Visibility	private
Scope	instance
Documentation	HK code for status of RF backdoor. Associated with use case **Get Housekeeping** (see 1B8 project).

UP_TIME	
Signature	-UP_TIME
Visibility	private
Scope	instance

Documentation	HK code for time in min from last reboot. Associated with use case **Get Housekeeping** (see 1B8 project).

EXECUTED_COMMANDS	
Signature	-EXECUTED_COMMANDS
Visibility	private
Scope	instance
Documentation	HK code for number of commands executed from last reboot. Associated with use case **Get Housekeeping** (see 1B8 project).

LENGTH_HOUSEKEEPING	
Signature	-LENGTH_HOUSEKEEPING
Visibility	private
Scope	instance
Documentation	The length of the housekeeping vector, in number of **t_sensor** elements. It also indicates the number of measurements in the housekeeping vector.

2.19. Class - Pseudoinverse

Name	**Value**
Name	Pseudoinverse
Stereotypes	SW
Documentation	This class contains all attributes and operations related to the calculation of pseudoinverse matrices. The maximum dimension allowed for the matrix to pseudoinvert is m * n.

Attributes

Bpi	
Signature	+Bpi : t_floatFactorByte[3][2*mechanicalConfiguration<6,5,1>::NUMPMT]
Visibility	public
Scope	classifier
Documentation	Pseudoinverse matrix used to calculate the magnetic field vector.

Zpi	
Signature	+Zpi : t_floatFactorByte[3][mechanicalConfiguration<6,5,1>::NUMPMT]
Visibility	public

Scope	classifier
Documentation	Pseudoinverse matrix used to calculate the spin vector.

Spi	
Signature	+Spi : t_floatFactorByte[3][2*mechanicalConfiguration<6,5,1>::NUMPMT]
Visibility	public
Scope	classifier
Documentation	Pseudoinverse matrix used to calculate the Sun's position vector.

Epi	
Signature	+Epi : t_floatFactorByte[3][2*mechanicalConfiguration<6,5,1>::NUMPMT]
Visibility	public
Scope	classifier
Documentation	Pseudoinverse matrix used to calculate the Earth's position vector.

PIstat	
Signature	+PIstat : StatusPI
Visibility	public
Scope	classifier
Documentation	Indicates the status of pseudoinverse matrices.

n	
Signature	-n : byte const = 3
Visibility	private
Scope	classifier
Documentation	Indicates the number of columns of the matrix A.

m	
Signature	-m : byte const = 2*mechanicalConfiguration<6,5,1>::NUMPMT
Visibility	private
Scope	classifier
Documentation	Indicates the maximum number of rows of the matrix A.

A	
Signature	-A : float[m][n]
Visibility	private

Scope	classifier
Documentation	The matrix to pseudoinvert. Operation Update fills its fields according to f parameter.

Ai	
Signature	-Ai : float[n][m]
Visibility	private
Scope	classifier
Documentation	Pseudoinverse of A.

U	
Signature	-U : float[m][m]
Visibility	private
Scope	classifier
Documentation	Its columns are left-singular vectors for the corresponding singular values, contained in S.

S	
Signature	-S : float[n][m]
Visibility	private
Scope	classifier
Documentation	A n*m matrix in which its diagonal entries are equal to the singular values of A.

V0	
Signature	-V0 : float[n][n]
Visibility	private
Scope	classifier
Documentation	Its columns are right-singular vectors for the corresponding singular values, contained in S.

Operations

Update	
Signature	Update(f : ushort, Sens : byte) : void
Visibility	public
Scope	instance
Documentation	Builds A from the matrix of direction cosines of each tile, according to f parameter. Then calls Compute operation.

	Data in Ai are converted from **float** to t_floatFactorByte using float2floatFactorByte and are saved in: - Bpi, if f is MAGNETIC_FIELD_X; - Zpi, if f is SPIN_Z; - Spi, if f is SUN_SENSOR_X; - Epi, if f is EARTH_SENSOR_X.

Code Body	byte cnt = 0;
	
	// Azzero matrici A, Ai, S...
	for(int i = 0; i < m; i++)
	for(int j = 0; j < n; j++)
	{
	A[i][j] = 0.0;
	Ai[j][i] = 0.0;
	S[j][i] = 0.0;
	}
	
	// Azzero matrice U
	for(int i = 0; i < m; i++)
	for(int j = 0; j < m; j++)
	U[i][j] = 0.0;
	
	// Azzero matrice V0
	for(int i = 0; i < n; i++)
	for(int j = 0; j < n; j++)
	V0[i][j] = 0.0;
	
	// Generazione matrice A
	for(int i = 0; i < m/2; i++)
	if((bool)Sens[i])
	{
	if(f == Bk1A_1B8_Codes::SPIN_Z)
	{
	for(int j = 0; j < n; j++)
	A[cnt][j] =
	mechanicalConfiguration<6,5,1>::orientation[i].tileO[2][j];
	cnt++;
	}
	else if ((f == Bk1A_1B8_Codes::MAGNETIC_FIELD_X) |
	(f == Bk1A_1B8_Codes::SUN_SENSOR_X) |
	(f == Bk1A_1B8_Codes::EARTH_SENSOR_X))
	{
	for(int j = 0; j < n; j++)
	{
	A[cnt][j] =
	mechanicalConfiguration<6,5,1>::orientation[i].tileO[0][j];
	A[cnt+1][j] =
	mechanicalConfiguration<6,5,1>::orientation[i].tileO[1][j];
	}
	cnt += 2;
	}
	}
	
	Compute(cnt);
	
	// Conversione float --> t_floatFactorByte
	for(int i = 0; i < n; i++)
	for(int j = 0; j < cnt; j++)

	if(f == Bk1A_1B8_Codes::SPIN_Z) Zpi[i][j] = float2floatFactor<GlobalTypes::t_floatFactorByte> (Ai[i][j]); else if(f == Bk1A_1B8_Codes::MAGNETIC_FIELD_X) Bpi[i][j] = float2floatFactor<GlobalTypes::t_floatFactorByte> (Ai[i][j]); else if(f == Bk1A_1B8_Codes::SUN_SENSOR_X) Spi[i][j] = float2floatFactor<GlobalTypes::t_floatFactorByte> (Ai[i][j]); else if(f == Bk1A_1B8_Codes::EARTH_SENSOR_X) Epi[i][j] = float2floatFactor<GlobalTypes::t_floatFactorByte> (Ai[i][j]);

Compute	
Signature	Compute(m1 : byte) : void
Visibility	public
Scope	classifier
Documentation	Computes the pseudoinverse of matrix A using SVD operation. Results are saved in Ai.

Code Body	float temp[m][n];
	
	// Clear temp
	for(int i = 0; i < n; i++)
	for(int j = 0; j < m; j++)
	temp[j][i] = 0;
	
	SVD(m1);
	
	for(int i = 0; i < n; i++)
	for(int j = 0; j < m1; j++)
	for(int k = 0; k < n; k++)
	temp[i][j] += V0[i][k]*S[k][j];
	
	for(int i = 0; i < n; i++)
	for(int j = 0; j < m1; j++)
	for(int k = 0; k < n; k++)
	Ai[i][j] += temp[i][k]*U[j][k];
	
	/*
	The calculation on A should be done using the transpose of U in the line
	Ai[i][j] += temp[i][k]*U[k][j];
	To avoid creating another float m*m matrix the code was changed as follows
	Ai[i][j] += temp[i][k]*U[j][k];
	The transposition of U is done during the calculation.
	*/

SVD	
Signature	SVD(m1 : byte) : void
Visibility	public
Scope	classifier

Documentation	Performs *Singular Values Decomposition* (SVD) algorithm. It uses U, S and V0 matrix as support for calculation.

Code Body	int i, j, k, l, l1, iter;
	float c, f, g, h, s, x, y, z;
	float e[n], q[n];
	int withu = 1;
	int withv = 1;
	float eps = 0.001;
	float tol = 0.001;
	
	for (i = 0; i < n; i++){
	  e[i] = 0;
	  q[i] = 0;
	}
	
	/* Copy 'A' to 'U' */
	for (i = 0; i < m1; i++)
	  for (j = 0; j < n; j++)
	  U[i][j] = A[i][j];
	
	/* Householder's reduction to bidiagonal form. */
	g = x = 0.0;
	for (i=0;i < n;i++)
	{
	  e[i] = g;
	  s = 0.0;
	  l = i+1;
	  for (j=i;j<m1;j++)
	  s += (U[j][i]*U[j][i]);
	  if (s < tol)
	  g = 0.0;
	  else
	  {
	  f = U[i][i];
	  g = (f < 0) ? sqrt(s) : -sqrt(s);
	  h = f * g - s;
	  U[i][i] = f - g;
	  for (j=l;j<n;j++) {
	  s = 0.0;
	  for (k=i;k<m1;k++)
	  s += (U[k][i] * U[k][j]);
	  f = s / h;
	  for (k=i;k<m1;k++)
	  U[k][j] += (f * U[k][i]);
	  } /* end j */
	  } /* end s */
	  q[i] = g;
	  s = 0.0;
	  for (j=l;j<n;j++)
	  s += (U[i][j] * U[i][j]);
	  if (s < tol)
	  g = 0.0;
	  else {
	  f = U[i][i+1];
	  g = (f < 0) ? sqrt(s) : -sqrt(s);
	  h = f * g - s;
	  U[i][i+1] = f - g;
	  for (j=l;j<n;j++)
	  e[j] = U[i][j]/h;

```
    for (j=l;j<m1;j++) {
      s = 0.0;
      for (k=l;k<n;k++)
          s += (U[j][k] * U[i][k]);
      for (k=l;k<n;k++)
          U[j][k] += (s * e[k]);
    } /* end j */
  } /* end s */
  y = fabs(q[i]) + fabs(e[i]);
  if (y > x)
    x = y;
} /* end i */

/* accumulation of right-hand transformations */
if (withv) {
  for (i=n-1;i>=0;i--) {
    if (g != 0.0) {
      h = U[i][i+1] * g;
      for (j=l;j<n;j++)
          V0[j][i] = U[i][j]/h;
      for (j=l;j<n;j++) {
          s = 0.0;
          for (k=l;k<n;k++)
            s += (U[i][k] * V0[k][j]);
          for (k=l;k<n;k++)
            V0[k][j] += (s * V0[k][i]);

      } /* end j */
    } /* end g */
    for (j=l;j<n;j++)
      V0[i][j] = V0[j][i] = 0.0;
    V0[i][i] = 1.0;
    g = e[i];
    l = i;
  } /* end i */

} /* end withv, parens added for clarity */

/* accumulation of left-hand transformations */
if (withu)
{
  for (i=n;i<m1;i++) {
    for (j=n;j<m1;j++)
      U[i][j] = 0.0;
    U[i][i] = 1.0;
  }
}
if (withu)
{
  for (i=n-1;i>=0;i--) {
    l = i + 1;
    g = q[i];
    for (j=l;j<m1;j++)  /* upper limit was 'n' */
      U[i][j] = 0.0;
    if (g != 0.0) {
      h = U[i][i] * g;
      for (j=l;j<m1;j++) { /* upper limit was 'n' */
          s = 0.0;
          for (k=l;k<m1;k++)
            s += (U[k][i] * U[k][j]);
```

```
          f = s / h;
          for (k=i;k<m1;k++)
            U[k][j] += (f * U[k][i]);
      } /* end j */
      for (j=i;j<m1;j++)
          U[j][i] /= g;
    } /* end g */
    else {
      for (j=i;j<m1;j++)
          U[j][i] = 0.0;
    }
    U[i][i] += 1.0;
  } /* end i*/
} /* end withu, parens added for clarity */

/* diagonalization of the bidiagonal form */
eps *= x;
for (k=n-1;k>=0;k--) {
  iter = 0;
test_f_splitting:
  for (l=k;l>=0;l--) {
    if (fabs(e[l]) <= eps) goto test_f_convergence;
    if (fabs(q[l-1]) <= eps) goto cancellation;
  } /* end l */

  /* cancellation of e[l] if l > 0 */
cancellation:
  c = 0.0;
  s = 1.0;
  l1 = l - 1;
  for (i=l;i<=k;i++) {
    f = s * e[i];
    e[i] *= c;
    if (fabs(f) <= eps) goto test_f_convergence;
    g = q[i];
    h = q[i] = sqrt(f*f + g*g);
    c = g / h;
    s = -f / h;
    if (withu) {
      for (j=0;j<m1;j++) {
          y = U[j][l1];
          z = U[j][i];
          U[j][l1] = y * c + z * s;
          U[j][i] = -y * s + z * c;
      } /* end j */
    } /* end withu, parens added for clarity */
  } /* end i */
test_f_convergence:
  z = q[k];
  if (l == k) goto convergence;

  /* shift from bottom 2x2 minor */
  iter++;
  if (iter > 30) {
    break;
  }
  x = q[l];
  y = q[k-1];
  g = e[k-1];
  h = e[k];
```

```
  f = ((y-z)*(y+z) + (g-h)*(g+h)) / (2*h*y);
  g = sqrt(f*f + 1.0);
  f = ((x-z)*(x+z) + h*(y/((f<0)?(f-g):(f+g))-h))/x;
  /* next QR transformation */
  c = s = 1.0;
  for (i=l+1;i<=k;i++) {
    g = e[i];
    y = q[i];
    h = s * g;
    g *= c;
    e[i-1] = z = sqrt(f*f+h*h);
    c = f / z;
    s = h / z;
    f = x * c + g * s;
    g = -x * s + g * c;
    h = y * s;
    y *= c;
    if (withv) {
      for (j=0;j<n;j++) {
          x = V0[j][i-1];
          z = V0[j][i];
          V0[j][i-1] = x * c + z * s;
          V0[j][i] = -x * s + z * c;
      } /* end j */
    } /* end withv, parens added for clarity */
    q[i-1] = z = sqrt(f*f + h*h);
    c = f/z;
    s = h/z;
    f = c * g + s * y;
    x = -s * g + c * y;
    if (withu) {
      for (j=0;j<m1;j++) {
          y = U[j][i-1];
          z = U[j][i];
          U[j][i-1] = y * c + z * s;
          U[j][i] = -y * s + z * c;
      } /* end j */
    } /* end withu, parens added for clarity */
  } /* end i */
  e[l] = 0.0;
  e[k] = f;
  q[k] = x;
  goto test_f_splitting;
convergence:
  if (z < 0.0) {
    /* q[k] is made non-negative */
    q[k] = - z;
    if (withv)
    {
      for (j=0;j<n;j++)
          V0[j][k] = -V0[j][k];
    } /* end withv, parens added for clarity */
  } /* end z */
} /* end k */


// Salvo i valori della matrice S
for(i=0; i<n; i++)
  if(q[i] != 0)
    S[i][i]=1/q[i];
```

float2floatFactorByte	
Signature	float2floatFactorByte(k : float) : t_floatFactorByte
Visibility	public
Scope	classifier
Documentation	Converts a **float** into t_floatFactorByte.

Code Body	float j;
	byte const m = 8;
	short a[m];
	short N0, N1, N2, D, D0, D1;
	bool neg = false;
	byte n = 0;
	GlobalTypes::t_floatFactorByte M;
	
	if (k > 127.0)
	{
	  M.M = 127;
	  M.D = 1;
	}
	else if (k < -128.0) {
	  M.M = -128;
	  M.D = 1;
	}
	else{
	
	  if(k < 0)
	  {
	  neg = true;
	  k = -k;
	  }
	
	  k = (ceilf(k*1000.0))/1000.0 ;
	
	  n = 0;
	  for(int i = 0; i < m; i++)
	  {
	  a[i] = (short) k;
	  j = k - (float) a[i];
	  if (j != 0)
	  k = 1/j;
	  else{
	  n++;
	  break;
	  }
	  n++;
	  }
	
	  N0 = 1;
	  N1 = 0;
	  N2 = 1;
	  D = 1;
	  D1 = 0;
	  D0 = -a[0] + 1;
	  for(int i = 0; i < n; i++)
	  {
	  N0 = a[i]*N0 + N1;

	if(i != 0)
	  D = a[i]*D + D0;
	
	// Controllo se esco dai limiti del byte/char...
	if((N0 > 0x7F) || (D > 0xFF))
	{
	  N0 = N2;
	  D = D1;
	  break;
	}
	
	N1 = N2;
	N2 = N0;
	D0 = D1;
	D1 = D;
	}
	
	if(neg)
	  M.M = -N0;
	else
	  M.M = N0;
	
	M.D = D;
	}
	return M;

float2floatFactorShort	
Signature	float2floatFactorShort(k : float) : t_floatFactorShort
Visibility	public
Scope	classifier
Documentation	Converts a **float** into t_floatFactorShort.

Code Body	float j;
	byte const m = 8;
	short a[m];
	short N0, N1, N2, D, D0, D1;
	bool neg = false;
	byte n = 0;
	GlobalTypes::t_floatFactorShort M;
	
	if (k > 32767.0)
	{
	M.M = 32767;
	M.D = 1;
	}
	else if (k < -32768.0) {
	M.M = -32768;
	M.D = 1;
	}
	else{
	
	if(k < 0)
	{
	neg = true;
	k = -k;
	}

	k = (ceilf(k*1000.0))/1000.0 ;
	
	n = 0;
	for(int i = 0; i < m; i++)
	{
	a[i] = (short) k;
	j = k - (float) a[i];
	if (j != 0)
	k = 1/j;
	else{
	n++;
	break;
	}
	n++;
	}
	
	N0 = 1;
	N1 = 0;
	N2 = 1;
	D = 1;
	D1 = 0;
	D0 = -a[0] + 1;
	for(int i = 0; i < n; i++)
	{
	N0 = a[i]*N0 + N1;
	if(i != 0)
	D = a[i]*D + D0;
	
	N1 = N2;
	N2 = N0;
	D0 = D1;
	D1 = D;
	}
	
	if(neg)
	M.M = -N0;
	else
	M.M = N0;
	
	M.D = D;
	}
	return M;

float2floatFactor	
Signature	float2floatFactor(k : float) : retType
Visibility	public
Scope	instance
Documentation	Converts a **float** into t_floatFactorByte or t_floatFactorShort.

Code Body	float j;
	byte const m = 8;
	short a[m];
	short N0, N1, N2, D, D0, D1;
	bool neg = false;
	byte n = 0;
	short s, h, l;
	retType M;

```
s = sizeof(M.M);
h =  (s << (s * 7))-1;
l = -(s << (s * 7));

if (k > (float) h)
{
  M.M = 127;
  M.D = 1;
}
else if (k < (float) l) {
  M.M = -128;
  M.D = 1;
}
else{

  if(k < 0)
  {
    neg = true;
    k = -k;
  }

  k = (ceilf(k*1000.0))/1000.0 ;

  n = 0;
  for(int i = 0; i < m; i++)
  {
    a[i] = (short) k;
    j = k - (float) a[i];
    if (j != 0)
      k = 1/j;
    else{
      n++;
      break;
    }
    n++;
  }

  N0 = 1;
  N1 = 0;
  N2 = 1;
  D = 1;
  D1 = 0;
  D0 = -a[0] + 1;
  for(int i = 0; i < n; i++)
  {
    N0 = a[i]*N0 + N1;
    if(i != 0)
      D = a[i]*D + D0;

    if((N0 > h) || (D > (1 << (s*8))-1))
    {
      N0 = N2;
      D  = D1;
      break;
    }

    N1 = N2;
    N2 = N0;
    D0 = D1;
```

	D1 = D;
	}
	
	if(neg)
	M.M = -N0;
	else
	M.M = N0;
	
	M.D = D;
	}
	return M;

2.20. Class - StatusPI

Name	Value
Name	StatusPI
Stereotypes	SW
Documentation	Status information for the pseudoinverse matrices. It also contains a vector for each type of sensor equipped in the satellite. Those vectors indicate the presence of the sensor (or if it's working) in each tile.

Attributes

MAG_FIELD_OBSOLETE	
Signature	+MAG_FIELD_OBSOLETE : byte const = 0x01
Visibility	public
Scope	classifier
Documentation	Indicates that the pseudoinverse matrix for magnetic field is obsolete.

MAG_FIELD_UPDATING	
Signature	+MAG_FIELD_UPDATING : byte const = 0x02
Visibility	public
Scope	classifier
Documentation	Indicates that the system is updating the pseudoinverse matrix for magnetic field.

SPIN_OBSOLETE	
Signature	+SPIN_OBSOLETE : byte const = 0x04
Visibility	public
Scope	classifier
Documentation	Indicates that the pseudoinverse matrix for spin is obsolete.

SPIN_UPDATING	
Signature	+SPIN_UPDATING : byte const = 0x08
Visibility	public
Scope	classifier
Documentation	Indicates that the system is updating the pseudoinverse matrix for spin.

SUN_SENS_OBSOLETE	
Signature	+SUN_SENS_OBSOLETE : byte const = 0x10
Visibility	public
Scope	classifier
Documentation	Indicates that the pseudoinverse matrix for Sun's position is obsolete.

SUN_SENS_UPDATING	
Signature	+SUN_SENS_UPDATING : byte const = 0x20
Visibility	public
Scope	classifier
Documentation	Indicates that the system is updating the pseudoinverse matrix for Sun's sensor.

EARTH_SENS_OBSOLETE	
Signature	+EARTH_SENS_OBSOLETE : byte const = 0x40
Visibility	public
Scope	classifier
Documentation	Indicates that the pseudoinverse matrix for Earth's position is obsolete.

EARTH_SENS_UPDATING	
Signature	+EARTH_SENS_UPDATING : byte const = 0x80
Visibility	public
Scope	classifier
Documentation	Indicates that the system is updating the pseudoinverse matrix for Earth's sensor.

magnetometer	
Signature	+magnetometer : byte[mechanicalConfiguration<6,5,1>::NUMPMT]
Visibility	public
Scope	classifier

Documentation	Indicates the presence/absence of the magnetometers in each PMT tile.

gyroscope	
Signature	+gyroscope : byte[mechanicalConfiguration<6,5,1>::NUMPMT]
Visibility	public
Scope	classifier
Documentation	Indicates the presence/absence of a gyroscope in each PMT tile.

sun	
Signature	+sun : byte[mechanicalConfiguration<6,5,1>::NUMPMT]
Visibility	public
Scope	classifier
Documentation	Indicates the presence/absence of a Sun sensor in each PMT tile.

earth	
Signature	+earth : byte[mechanicalConfiguration<6,5,1>::NUMPMT]
Visibility	public
Scope	classifier
Documentation	Indicates the presence/absence of a Earth sensor in each PMT tile.

Operations

StatusPI	
Signature	StatusPI()
Visibility	public
Scope	instance
Documentation	Constructor of class StatusPI.

Code Body	for(int i = 0; i < mechanicalConfiguration<6,5,1>::NUMPMT; i++)
	{
	  gyroscope[i] = mechanicalConfiguration<6,5,1>::gyroscope[i];
	  magnetometer[i] =
	mechanicalConfiguration<6,5,1>::magnetometer[i];
	  sun[i] = mechanicalConfiguration<6,5,1>::sun[i];
	  earth[i] = mechanicalConfiguration<6,5,1>::earth[i];
	}

2.21. Class - Manager_1B8

Name	**Value**
Name	Manager_1B8

Stereotypes	SW
Documentation	This class allow the communication with a PMT (1B8) tile. It contains, and can analyze, the telemetry vectors for each PMT tile.

Attributes

NUMPMT	
Signature	-NUMPMT : short const = mechanicalConfiguration<6,5,1>::NUMPMT
Visibility	private
Scope	classifier
Documentation	Number of PMT (1B8) tiles in the satellite.

LENHkPMT	
Signature	-LENHkPMT : short const = mechanicalConfiguration<6,5,1>::LENHkPMT
Visibility	private
Scope	classifier
Documentation	The length of PMT (1B8) tiles' housekeeping vector.

LENStatPMT	
Signature	-LENStatPMT : short const = mechanicalConfiguration<6,5,1>::LENStatPMT
Visibility	private
Scope	classifier
Documentation	The length of PMT (1B8) tiles' statistics vector.

LENHisPMT	
Signature	-LENHisPMT : short const = mechanicalConfiguration<6,5,1>::LENHisPMT
Visibility	private
Scope	classifier
Documentation	The length of PMT (1B8) tiles' history vector.

DEPHisPMT	
Signature	-DEPHisPMT : short const = mechanicalConfiguration<6,5,1>::DEPHisPMT
Visibility	private
Scope	classifier

Documentation	The depth of PMT (1B8) tiles' history vector.

LENStatusPMT	
Signature	-LENStatusPMT : ushort const = 16
Visibility	private
Scope	classifier
Documentation	The length of PMT (1B8) tiles' status vector.

LENConfPMT	
Signature	-LENConfPMT : ushort const = 16
Visibility	private
Scope	classifier
Documentation	The length of PMT (1B8) tiles' configuration vector.

hkPMT	
Signature	+hkPMT : t_sensor[NUMPMT][LENHkPMT]
Visibility	public
Scope	classifier
Documentation	Contains all housekeeping vectors of all PMT (1B8) tiles.

statPMT	
Signature	+statPMT : t_sensor[NUMPMT][LENStatPMT]
Visibility	public
Scope	classifier
Documentation	Contains all statistics vectors of all PMT (1B8) tiles.

hisPMT	
Signature	+hisPMT : t_sensor[NUMPMT][DEPHisPMT][LENHisPMT]
Visibility	public
Scope	classifier
Documentation	Contains all history vectors of all PMT (1B8) tiles.

hisPMTcnt	
Signature	+hisPMTcnt : byte = 0
Visibility	public
Scope	classifier

confPMT

Signature	+confPMT : t_Configuration[NUMPMT][LENConfPMT]
Visibility	public
Scope	classifier
Documentation	Contains all configuration parameters of all PMT (1B8) tiles.

masErrPMT	
Signature	+masErrPMT : t_MasterErrors[NUMPMT]
Visibility	public
Scope	classifier
Documentation	Contains all communication errors (master side) of all PMT (1B8) tiles.

slaErrPMT	
Signature	+slaErrPMT : t_Status[NUMPMT][LENStatusPMT]
Visibility	public
Scope	classifier
Documentation	Contains the status of all PMT (1B8) tiles.

resPMT	
Signature	-resPMT : Result_1B8
Visibility	private
Scope	classifier
Documentation	Contains the results of the elaboration on PMT (1B8) tiles data.

msg	
Signature	-msg : Message_Master
Visibility	private
Scope	classifier

Operations

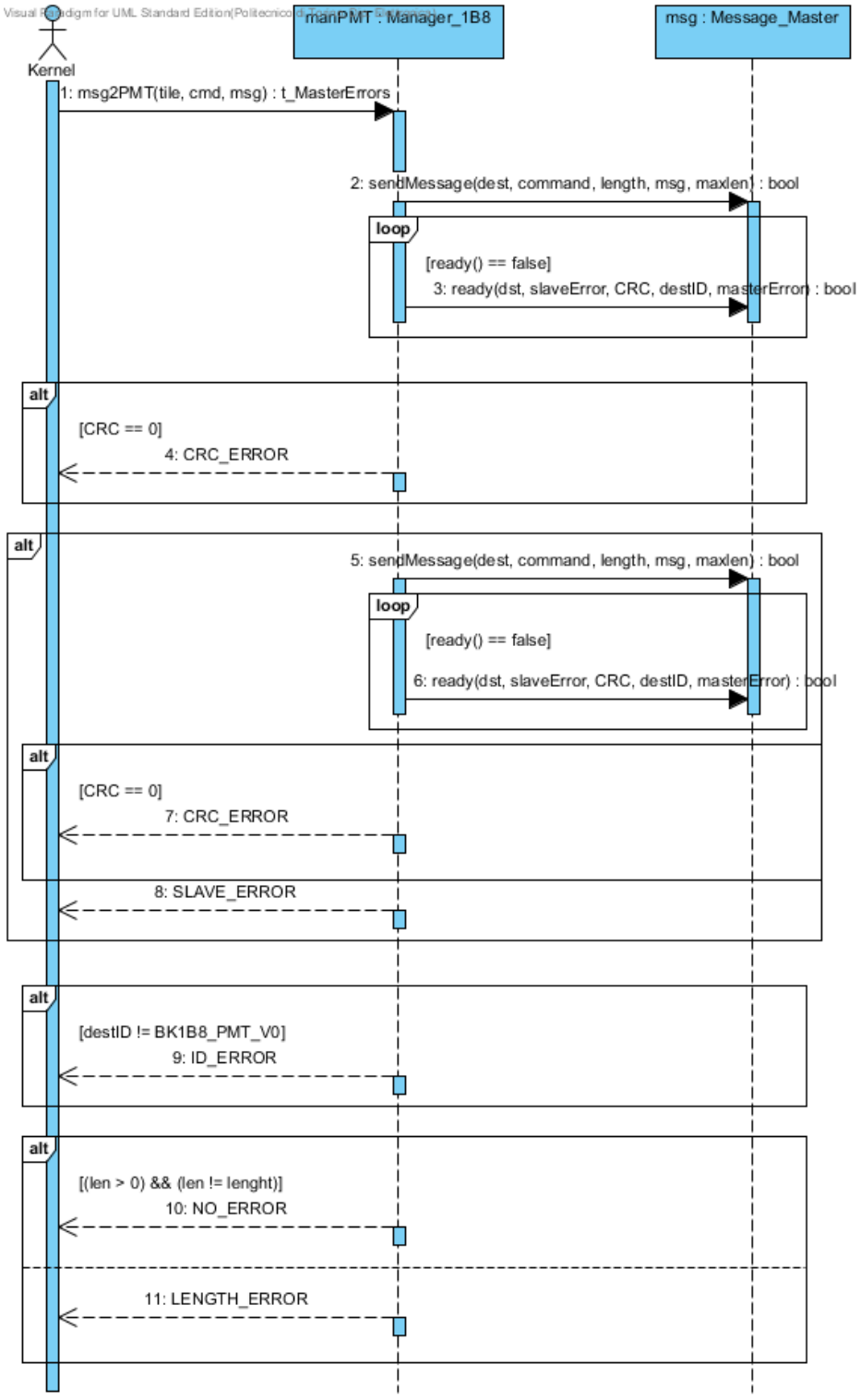
msg2PMT	
Signature	msg2PMT(tile : byte, cmd : t_Commands, msg : byte) : t_MasterErrors
Visibility	public
Scope	classifier
Documentation	Allows the sending of the command cmd between a 1B8 slave tile and master. The message sent/received is contained in msg. The method returns any t_MasterErrors error encountered by the master, or

	NO_ERROR.

Code Body	ushort maxlen = 0;
	ushort lenght = 0;
	ushort len = 0;
	bool slaveErr = false;
	byte destID = 0;
	ushort CRC = 0;
	Bk1B45_Master::t_MasterErrors maserr;
	
	// Imposta il valore di maxlen e lenght
	
	if(cmd == Bk1A1_Common_Codes::CMD_GET_HOUSEKEEPING){
	maxlen = sizeof(GlobalTypes::t_sensor)*LENHkPMT;
	lenght = LENHkPMT;
	}else if(cmd == Bk1A1_Common_Codes::CMD_GET_STATISTICS){
	maxlen = sizeof(GlobalTypes::t_sensor)*LENStatPMT;
	lenght = LENStatPMT;
	}else if(cmd == Bk1A1_Common_Codes::CMD_GET_HISTORY){
	maxlen = sizeof(GlobalTypes::t_sensor)*(LENHisPMT*DEPHisPMT);
	lenght = LENHisPMT*DEPHisPMT;
	}else if((cmd == Bk1A1_Common_Codes::CMD_GET_CONFIGURATION) |
	(cmd == Bk1A1_Common_Codes::CMD_SET_CONFIGURATION)) {
	maxlen = sizeof(t_Configuration);
	lenght = sizeof(t_Configuration);
	}
	
	len = lenght;
	
	Manager_1B8::msg.sendMessage(mechanicalConfiguration<6,5,1>::addressPMT[tile], cmd, lenght, msg, maxlen);
	
	while(Manager_1B8::msg.ready(mechanicalConfiguration<6,5,1>::addressPMT[tile], slaveErr, CRC, destID, maserr) == false) {}
	
	if(CRC != 0)
	return Bk1B45_Master::CRC_ERROR;
	
	if(slaveErr)
	{
	
	Manager_1B8::msg.sendMessage(mechanicalConfiguration<6,5,1>::addressPMT[tile], Bk1A1_Common_Codes::CMD_GET_STATUS,lenght,(byte *)&slaErrPMT[tile],sizeof(t_Status));
	
	while (Manager_1B8::msg.ready(mechanicalConfiguration<6,5,1>::addressPMT[tile], slaveErr, CRC, destID, maserr) == false) {}
	
	if (CRC != 0)
	return Bk1B45_Master::CRC_ERROR;
	
	return Bk1B45_Master::SLAVE_ERROR;
	}
	
	if (destID != Bk1A1_Common_Codes::Bk1B8_PMT_V0)
	return Bk1B45_Master::ID_ERROR;
	
	if ((len > 0) && (len != lenght))
	return Bk1B45_Master::LENGTH_ERROR;

	else return Bk1B45_Master::NO_ERROR;

Sequence Diagram - msg2PMT



Questo Sequence Diagram illustra il funzionamento del metodo msg2PMT della classe Manager_1B8.
Questo metodo permette la comunicazione tra l'OBC e le PMT tile. I blocchi alt sono utilizzati per identificare il tipo di errore riscontrato durante la comunicazione.

analyzePMT	
Signature	analyzePMT() : void
Visibility	public
Scope	classifier
Documentation	Analyzes the housekeeping vector hkPMT and save results in resPMT.

Code Body	// Analizzo la configurazione del sistema
	for(int i = 0; i < NUMPMT; i++)
	{
	  if(/*
	confPMT[i].config_1B2.test(t_ConfigWord_1B2::ATTACH_MAGNETOMETER) &
	confPMT[i].mechanical.test(t_MechanicalConfiguration_1B8::HAS_MAGNETOM
	ETER) & */
	  (bool)Pseudoinverse::PIstat.magnetometer[i])
	  Pseudoinverse::PIstat.magnetometer[i] = 1;
	  else
	  {
	  Pseudoinverse::PIstat.magnetometer[i] = 0;
	  Pseudoinverse::PIstat.set(StatusPI::MAG_FIELD_OBSOLETE,
	StatusPI::MAG_FIELD_OBSOLETE);
	  }
	
	  if(/* confPMT[i].config_1B2.test(t_ConfigWord_1B2::ATTACH_GYROSCOPE) &
	confPMT[i].mechanical.test(t_MechanicalConfiguration_1B8::HAS_GYROSCOP
	E) & */
	  (bool)Pseudoinverse::PIstat.gyroscope[i])
	  Pseudoinverse::PIstat.gyroscope[i] = 1;
	  else
	  {
	  Pseudoinverse::PIstat.gyroscope[i] = 0;
	  Pseudoinverse::PIstat.set(StatusPI::SPIN_OBSOLETE,
	StatusPI::SPIN_OBSOLETE);
	  }
	
	  if(/* confPMT[i].config_1B2.test(t_ConfigWord_1B2::ATTACH_SUNSENSOR) &
	confPMT[i].mechanical.test(t_MechanicalConfiguration_1B8::HAS_SUNSENSO
	R) & */
	  (bool)Pseudoinverse::PIstat.sun[i])
	  Pseudoinverse::PIstat.sun[i] = 1;
	  else
	  {
	  Pseudoinverse::PIstat.sun[i] = 0;
	  Pseudoinverse::PIstat.set(StatusPI::SUN_SENS_OBSOLETE,
	StatusPI::SUN_SENS_OBSOLETE);
	  }
	
	  if(/*
	confPMT[i].config_1B2.test(t_ConfigWord_1B2::ATTACH_EARTHSENSOR) &
	confPMT[i].mechanical.test(t_MechanicalConfiguration_1B8::HAS_EARTHSENS
	OR) & */
	  (bool)Pseudoinverse::PIstat.earth[i])
	  Pseudoinverse::PIstat.earth[i] = 1;
	  else
	  {
	  Pseudoinverse::PIstat.earth[i] = 0;

```
    Pseudoinverse::PIstat.set(StatusPI::EARTH_SENS_OBSOLETE,
StatusPI::EARTH_SENS_OBSOLETE);
  }
}

// Calcolo del vettore Campo Magnetico
if(Pseudoinverse::PIstat.test(StatusPI::MAG_FIELD_OBSOLETE +
StatusPI::MAG_FIELD_UPDATING,
                              StatusPI::MAG_FIELD_OBSOLETE +
StatusPI::MAG_FIELD_UPDATING))
{
  getMagField(resPMT.Buvw);
#ifdef __DEBUG
  printf("Campo magnetico\n x\t%i\n y\t%i\n z\t%i\n\n", resPMT.Buvw[0],
resPMT.Buvw[1], resPMT.Buvw[2]);
#endif
}

// Calcolo del vettore Spin
if(Pseudoinverse::PIstat.test(StatusPI::SPIN_OBSOLETE +
StatusPI::SPIN_UPDATING,
                              StatusPI::SPIN_OBSOLETE +
StatusPI::SPIN_UPDATING))
{
  getSpin(resPMT.Spin);
#ifdef __DEBUG
  printf("Spin del satellite\n x\t%i\n y\t%i\n z\t%i\n\n", resPMT.Spin[0],
resPMT.Spin[1], resPMT.Spin[2]);
#endif
}

// Calcolo del vettore Posizione del Sole
if(Pseudoinverse::PIstat.test(StatusPI::SUN_SENS_OBSOLETE +
StatusPI::SUN_SENS_UPDATING,
                              StatusPI::SUN_SENS_OBSOLETE +
StatusPI::SUN_SENS_UPDATING))
{
  getPosition(Bk1A_1B8_Codes::SUN_SENSOR_X, resPMT.SunPos);
#ifdef __DEBUG
  printf("Posizione del Sole\n x\t%i\n y\t%i\n z\t%i\n\n", resPMT.SunPos[0],
resPMT.SunPos[1], resPMT.SunPos[2]);
#endif
}

// Calcolo del vettore Posizione della Terra
if(Pseudoinverse::PIstat.test(StatusPI::EARTH_SENS_OBSOLETE +
StatusPI::EARTH_SENS_UPDATING,
                              StatusPI::EARTH_SENS_OBSOLETE +
StatusPI::EARTH_SENS_UPDATING))
{
  getPosition(Bk1A_1B8_Codes::EARTH_SENSOR_X, resPMT.EarthPos);
#ifdef __DEBUG
  printf("Posizione della Terra\n x\t%i\n y\t%i\n z\t%i\n\n", resPMT.EarthPos[0],
resPMT.EarthPos[1], resPMT.EarthPos[2]);
#endif
}


// Calcolo medie artimetiche
for(int i = 0; i < NUMPMT; i++)
```

	{
	  resPMT.STCmean +=
	hkPMT[i][Bk1A_1B8_Codes::SOLAR_TEMPERATURE_CENTER];
	  resPMT.STSmean +=
	hkPMT[i][Bk1A_1B8_Codes::SOLAR_TEMPERATURE_SIDE];
	  resPMT.BTmean +=
	hkPMT[i][Bk1A_1B8_Codes::BATTERY_TEMPERATURE];
	  resPMT.PTmean +=
	hkPMT[i][Bk1A_1B8_Codes::POWERCIRCUITS_TEMPERATURE];
	  resPMT.ACSTmean += hkPMT[i][Bk1A_1B8_Codes::ACS_TEMPERATURE];
	}
	resPMT.STCmean /= NUMPMT;
	resPMT.STSmean /= NUMPMT;
	resPMT.BTmean /= NUMPMT;
	resPMT.PTmean /= NUMPMT;
	resPMT.ACSTmean /= NUMPMT;
	
	// Campo Internal_Bus_Voltage - Deviazione Standard e Valori min/MAX
	resPMT.IBVdev = StdDev(Bk1A_1B8_Codes::INTERNAL_BUS_VOLTAGE,
	resPMT.IBVmean);
	minMAX(Bk1A_1B8_Codes::INTERNAL_BUS_VOLTAGE, resPMT.IBVmin,
	resPMT.IBVmax);
	
	#ifdef __DEBUG
	  printf("STC mean\t%i\nSTS mean\t%i\nBT mean\t%i\n", resPMT.STCmean,
	resPMT.STSmean, resPMT.BTmean);
	  printf("IBV mean\t%i\nIBV dev\t%i\nIBV min\t%i - %i\nIBV max\t%i - %i\n"
	  , resPMT.IBVmean, resPMT.IBVdev, resPMT.IBVmin[0],
	resPMT.IBVmin[1], resPMT.IBVmax[0], resPMT.IBVmax[1]);
	  printf("PT mean\t%i\nACST mean\t%i\n", resPMT.PTmean,
	resPMT.ACSTmean);
	#endif

getMagField	
Signature	getMagField(Buvw : t_sensor) : bool
Visibility	public
Scope	classifier
Documentation	Calculate the magnetic field vector, refered to the center of the satellite.

Code Body	// Definizione delle variabili
	GlobalTypes::t_sensor PHIt[2*NUMPMT];
	byte cnt = 0;
	byte n = 0;
	
	// Verifico la presenza dei magnetometri
	for(int i = 0; i < NUMPMT; i++)
	  if((bool)Pseudoinverse::PIstat.magnetometer[i])
	  n++;
	
	if (n > 1)
	{
	  // Inizializzo il vettore PHIt
	  for(int i = 0; i < NUMPMT; i++)
	  if((bool)Pseudoinverse::PIstat.magnetometer[i])
	  {
	  PHIt[cnt++] =

	hkPMT[i][Bk1A_1B8_Codes::MAGNETIC_FIELD_X];
	PHIt[cnt++] =
	hkPMT[i][Bk1A_1B8_Codes::MAGNETIC_FIELD_Y];
	}
	
	for(int i = 0; i < 3; i++)
	Buvw[i] = 0;
	
	for(int i = 0; i < 3; i++)
	for(int j = 0; j < 2*n; j++)
	Buvw[i] += Pseudoinverse::Bpi[i][j].multiply(PHIt[j]);
	
	return 0;
	}else
	return 1;

getSpin	
Signature	getSpin(Spin : t_sensor) : bool
Visibility	public
Scope	classifier
Documentation	Calculate the spin vector, refered to the center of the satellite.

Code Body	// Definizione delle variabili
	GlobalTypes::t_sensor OMEGAt[NUMPMT];
	byte n = 0;
	byte cnt = 0;
	
	// Verifico la presenza dei giroscopi
	for(int i = 0; i < NUMPMT; i++)
	if((bool)Pseudoinverse::PIstat.gyroscope[i])
	n++;
	
	// Inizializzo il vettore OMEGAt
	for(int i = 0; i < NUMPMT; i++)
	if((bool)Pseudoinverse::PIstat.gyroscope[i])
	OMEGAt[cnt++] = hkPMT[i][Bk1A_1B8_Codes::SPIN_Z];
	
	for(int i = 0; i < 3; i++)
	Spin[i] = 0;
	
	for(int i = 0; i < 3; i++)
	for(int j = 0; j < n; j++)
	Spin[i] += Pseudoinverse::Zpi[i][j].multiply(OMEGAt[j]);
	
	return 0;

getPosition	
Signature	getPosition(f : HK_fields_1B8, r : t_sensor) : bool
Visibility	public
Scope	classifier
Documentation	Calculate the the position relative to another celestial body (i.e. Sun, Earth, costellations, etc..), refered to the center of the satellite.
Code Body	// Definizione delle variabili

	GlobalTypes::t_sensor Pt[2*NUMPMT];
	byte cnt = 0;
	byte n = 0;
	
	for(int i = 0; i < NUMPMT; i++)
	  if((f == Bk1A_1B8_Codes::SUN_SENSOR_X) &
	(bool)Pseudoinverse::PIstat.sun[i])
	  n++;
	  else if ((f == Bk1A_1B8_Codes::EARTH_SENSOR_X) &
	(bool)Pseudoinverse::PIstat.earth[i])
	  n++;
	
	if(n > 1)
	{
	  // Inizializzo il vettore Pt
	  for(int i = 0; i < NUMPMT; i++)
	  if ((f == Bk1A_1B8_Codes::SUN_SENSOR_X) &
	(bool)Pseudoinverse::PIstat.sun[i])
	  {
	  Pt[cnt++] = Lookups::sin2pi[hkPMT[i][f]] *
	Lookups::cos2pi[hkPMT[i][f+1]];
	  Pt[cnt++] = Lookups::sin2pi[hkPMT[i][f]] *
	Lookups::sin2pi[hkPMT[i][f+1]];
	  }else if ((f == Bk1A_1B8_Codes::EARTH_SENSOR_X) &
	(bool)Pseudoinverse::PIstat.earth[i])
	  {
	  Pt[cnt++] = Lookups::sin2pi[hkPMT[i][f]] *
	Lookups::cos2pi[hkPMT[i][f+1]];
	  Pt[cnt++] = Lookups::sin2pi[hkPMT[i][f]] *
	Lookups::sin2pi[hkPMT[i][f+1]];
	  }
	
	  for(int i = 0; i < 3; i++)
	  r[i] = 0;
	
	  for(int i = 0; i < 3; i++)
	  for(int j = 0; j < 2*n; j++)
	  if(f == Bk1A_1B8_Codes::SUN_SENSOR_X)
	  r[i] += Pseudoinverse::Spi[i][j].multiply(Pt[j]);
	  else if (f == Bk1A_1B8_Codes::EARTH_SENSOR_X)
	  r[i] += Pseudoinverse::Epi[i][j].multiply(Pt[j]);
	  return 0;
	}else
	  return 1;

StdDev	
Signature	StdDev(f : HK_fields_1B8, m : t_sensor) : t_sensor
Visibility	public
Scope	classifier
Documentation	Compute the Standard Deviation of the values in vector hkPMT indicated by f. This method also compute the mean of those values and save it in parameter m.

Code Body	long s = 0, s2 = 0;
	long sum2 = 0, sumc = 0;
	
	for(int i = 0; i < NUMPMT; i++)

	m += hkPMT[i][f];
	m /= NUMPMT;
	
	for(int i = 0; i < NUMPMT; i++)
	{
	sum2 += (hkPMT[i][f] - m)*(hkPMT[i][f] - m);
	sumc += (hkPMT[i][f] - m);
	}
	
	s2 = (sum2 - (sumc*sumc)/NUMPMT) / NUMPMT;
	s = sqrt(s2);
	
	return (GlobalTypes::t_sensor)s;

minMAX	
Signature	minMAX(f : HK_fields_1B8, min : t_sensor, MAX : t_sensor) : void
Visibility	public
Scope	classifier
Documentation	Compute the minumum and the maximum of a set of values, indicated by parameter f. Indicates also in which tile has occurred the minumum and the maximum.

Code Body	MAX[0] = 0;
	MAX[1] = 0;
	min[0] = 0xFF;
	min[1] = 0;
	
	for(int i = 0; i < NUMPMT; i++)
	{
	if(MAX[0] < hkPMT[i][f])
	{
	MAX[0] = hkPMT[i][f];
	MAX[1] = i;
	}
	if(min[0] > hkPMT[i][f])
	{
	min[0] = hkPMT[i][f];
	min[1] = i;
	}
	}

sqrt	
Signature	sqrt(value : long) : t_sensor
Visibility	private
Scope	classifier
Documentation	Compute the square root of the parameter value.

Code Body	// Dato l'intero positivo value, calcola la parte intera della
	// sua radice quadrata principale e il relativo resPMTto;
	// ritorna la radice
	
	long x, r, dp1, L, g[10], j, y,yn;
	
	// separa coppie di cifre e calcola numero delle cifre della radice

	L = 0;
	while(value > 0)
	{
	  g[L++] = value % 100;
	  value /= 100;
	}
	
	// corsa per individuare le successive cifre della radice
	x = r = 0;
	for(j = L-1; j >= 0; j--)
	{
	  r = r * 100 + g[j]; //somma al resto precedente moltiplicato per 100 il nuovo gruppo di 2 cifre
	  y = 0; // determina cifra
	  for(dp1 = 1; dp1 < 10; dp1++)
	  {
	  yn = dp1 * (20 * x + dp1);
	  if(yn <= r)
	  y=yn;
	  else
	  break;
	  }
	  x = x * 10 + dp1 - 1;
	  r -= y;
	}
	return (GlobalTypes::t_sensor)x;

2.22. Class - Manager_1B9

Name	Value
Name	Manager_1B9
Stereotypes	SW
Documentation	This class allow the communication with a OBC (1B9) tile. It contains, and can analyze, the telemetry vectors for each OBC tile.

Attributes

NUMOBC	
Signature	-NUMOBC : short const = mechanicalConfiguration<6,5,1>::NUMOBC
Visibility	private
Scope	classifier
Documentation	Number of OBC (1B9) tiles in the satellite.

LENHkOBC	
Signature	-LENHkOBC : short const = mechanicalConfiguration<6,5,1>::LENHkOBC
Visibility	private

Scope	classifier
Documentation	The length of OBC (1B9) tiles' housekeeping vector.

LENStatOBC	
Signature	-LENStatOBC : short const = mechanicalConfiguration<6,5,1>::LENStatOBC
Visibility	private
Scope	classifier
Documentation	The length of OBC (1B9) tiles' statistics vector.

LENHisOBC	
Signature	-LENHisOBC : short const = mechanicalConfiguration<6,5,1>::LENHisOBC
Visibility	private
Scope	classifier
Documentation	The length of OBC (1B9) tiles' history vector.

DEPHisOBC	
Signature	-DEPHisOBC : short const = mechanicalConfiguration<6,5,1>::DEPHisOBC
Visibility	private
Scope	classifier
Documentation	The depth of OBC (1B9) tiles' history vector.

LENStatusOBC	
Signature	-LENStatusOBC : ushort const = 16
Visibility	private
Scope	classifier
Documentation	The length of OBC (1B9) tiles' status vector.

LENConfOBC	
Signature	-LENConfOBC : ushort const = 16
Visibility	private
Scope	classifier
Documentation	The length of OBC (1B9) tiles' configuration vector.

hkOBC	
Signature	+hkOBC : t_sensor[NUMOBC][LENHkOBC]
Visibility	public

Scope	classifier
Documentation	Contains all housekeeping vectors of all OBC (1B9) tiles.

statOBC	
Signature	+statOBC : t_sensor[NUMOBC][LENStatOBC]
Visibility	public
Scope	classifier
Documentation	Contains all statistics vectors of all OBC (1B9) tiles.

hisOBC	
Signature	+hisOBC : t_sensor[NUMOBC][DEPHisOBC][LENHisOBC]
Visibility	public
Scope	classifier
Documentation	Contains all history vectors of all OBC (1B9) tiles.

hisOBCcnt	
Signature	+hisOBCcnt : byte = 0
Visibility	public
Scope	classifier

confOBC	
Signature	+confOBC : t_Configuration[NUMOBC][LENConfOBC]
Visibility	public
Scope	classifier
Documentation	Contains all configuration parameters of all OBC (1B9) tiles.

masErrOBC	
Signature	+masErrOBC : t_MasterErrors[NUMOBC]
Visibility	public
Scope	classifier
Documentation	Contains all communication errors (master side) of all OBC (1B9) tiles.

slaErrOBC	
Signature	+slaErrOBC : t_Status[NUMOBC][LENStatusOBC]
Visibility	public
Scope	classifier
Documentation	Contains the status of all OBC (1B9) tiles.

resOBC	
Signature	-resOBC : Result_1B9
Visibility	private
Scope	classifier
Documentation	Contains the results of the elaboration on OBC (1B9) tiles data.

msg	
Signature	-msg : Message_Master
Visibility	private
Scope	classifier

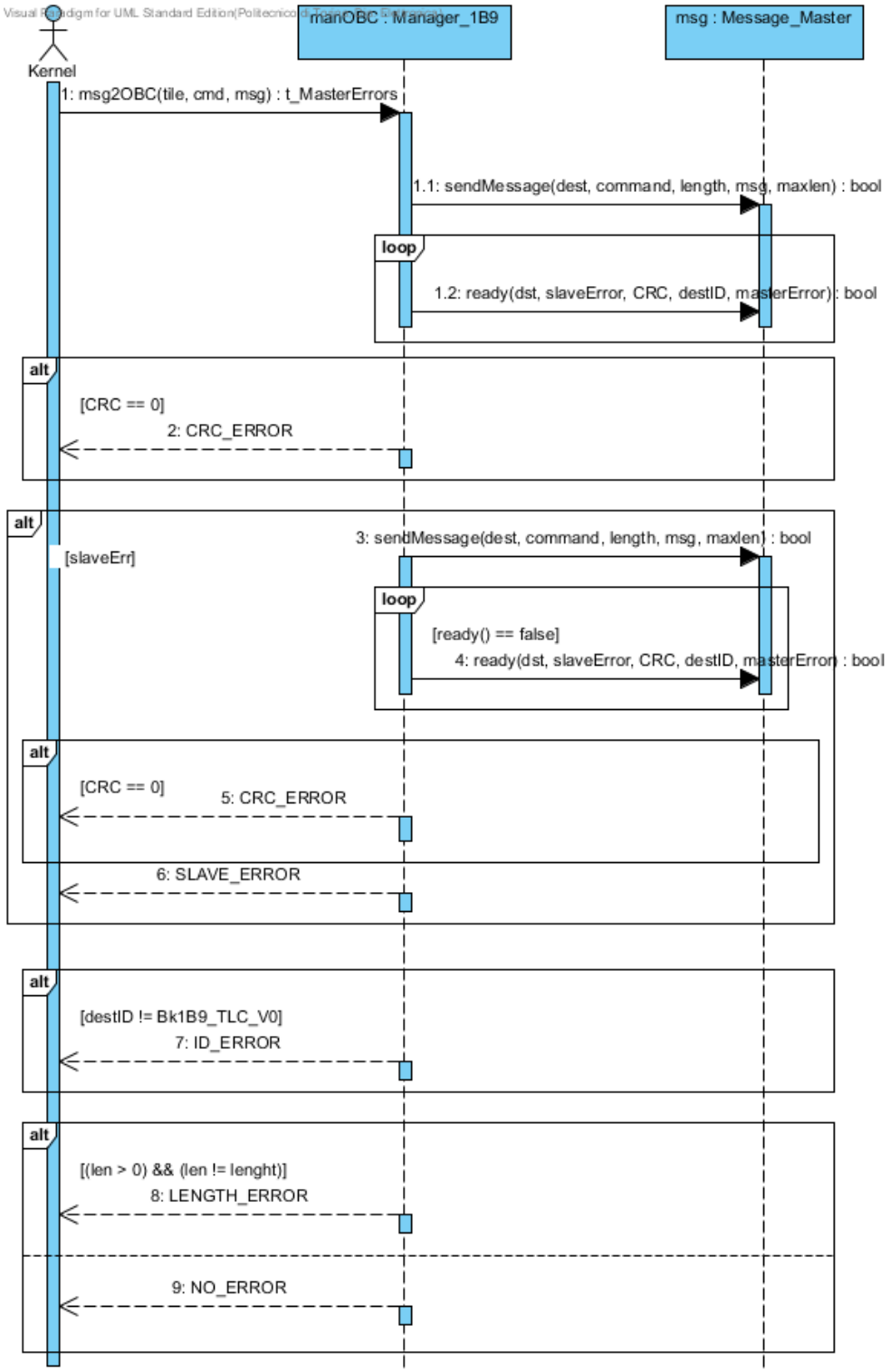
Operations

msg2OBC	
Signature	msg2OBC(tile : byte, cmd : t_Commands, msg : byte) : t_MasterErrors
Visibility	public
Scope	classifier
Documentation	Allows the sending of the command cmd between a 1B9 slave tile and master. The message sent/received is contained in msg. The method returns any t_MasterErrors error encountered by the master, or NO_ERROR.

Code Body	ushort maxlen = 0;
	ushort lenght = 0;
	ushort len = 0;
	bool slaveErr = false;
	byte destID = 0;
	ushort CRC = 0;
	Bk1B45_Master::t_MasterErrors maserr;
	
	// Imposta il valore di maxlen e lenght
	
	if(cmd == Bk1A1_Common_Codes::CMD_GET_HOUSEKEEPING){
	maxlen = sizeof(GlobalTypes::t_sensor)*LENHkOBC;
	lenght = LENHkOBC;
	}else if(cmd == Bk1A1_Common_Codes::CMD_GET_STATISTICS){
	maxlen = sizeof(GlobalTypes::t_sensor)*LENStatOBC;
	lenght = LENStatOBC;
	}else if(cmd == Bk1A1_Common_Codes::CMD_GET_HISTORY){
	maxlen = sizeof(GlobalTypes::t_sensor)*(LENHisOBC*DEPHisOBC);
	lenght = LENHisOBC*DEPHisOBC;
	}else if(cmd == Bk1A1_Common_Codes::CMD_GET_CONFIGURATION){
	maxlen = sizeof(t_Configuration);
	lenght = sizeof(t_Configuration);
	}else if(cmd == Bk1A1_Common_Codes::CMD_WRITE_DATA_0){
	maxlen = sizeof(StatusPI);
	lenght = sizeof(StatusPI);

```
}

len = lenght;

Manager_1B9::msg.sendMessage(mechanicalConfiguration<6,5,1>::addressOBC[tile],
cmd, lenght, msg, maxlen);

while(Manager_1B9::msg.ready(mechanicalConfiguration<6,5,1>::addressOBC[tile],
slaveErr, CRC, destID, maserr) == false) {}

if(CRC != 0)
  return Bk1B45_Master::CRC_ERROR;

if(slaveErr)
{

Manager_1B9::msg.sendMessage(mechanicalConfiguration<6,5,1>::addressOBC[tile],
Bk1A1_Common_Codes::CMD_GET_STATUS,lenght,(byte
*)&slaErrOBC[tile],sizeof(t_Status));

  while (Manager_1B9::msg.ready(mechanicalConfiguration<6,5,1>::addressOBC[tile],
slaveErr, CRC, destID, maserr) == false) {}

  if (CRC != 0)
    return Bk1B45_Master::CRC_ERROR;

  return Bk1B45_Master::SLAVE_ERROR;
}

if (destID != Bk1A1_Common_Codes::Bk1B9_TLC_V0)
  return Bk1B45_Master::ID_ERROR;


if ((len > 0) && (len != lenght))
  return Bk1B45_Master::LENGTH_ERROR;
else
  return Bk1B45_Master::NO_ERROR;
```

Sequence Diagram - msg2OBC



Questo Sequence Diagram illustra il funzionamento del metodo msg2OBC della classe Manager_1B9.
Questo metodo permette la comunicazione tra l'OBC e le TLC tile. I blocchi alt sono utilizzati per identificare il tipo di errore riscontrato durante la comunicazione.

analyzeOBC	
Signature	analyzeOBC()

Visibility	public
Scope	classifier
Documentation	Analyzes the housekeeping vector hkOBC and save results in resOBC.
Code Body	/// TBD

2.23. Class - UnusedInterrupts_master

Name	Value
Name	UnusedInterrupts_master
Stereotypes	SW
Documentation	Disables all unused interrupts (if any) and traps any of them in case that, for any reason, they get enabled by mistake. The watchdog is also disabled. The constructor disables all interrupts. Each interrupt service routine disables the corresponding interrupt enable flag. If an interrupt becomes used, the Designer shall: duplicate this class and name it appropriately remove the corresponding interrupt service (or watchdog) routine from the duplicated class to the relevant class remove its disabling from the constructor

Attributes

DISABLED_INTERRUPTS	
Signature	-DISABLED_INTERRUPTS : ushort const = 0xFCBF
Visibility	private
Scope	classifier
Documentation	Set (respectively reset) each bit of this variable if the corresponding isr_ is present in this class (respectively, has been moved into another class), according to the following correspondence:

DISABLED_INTERRUPTS	isr_ routine
0x0001	isr_basictimer()
0x0002	isr_P2()
0x0004	
0x0008	isr_dac12_dma()
0x0010	isr_P1()
0x0020	isr_timerA1()
0x0040	isr_timerA0()
0x0080	isr_adc12()
0x0100	isr_uart0tx()
0x0200	isr_uart0rx()

	0x0400	isr_wdt()
	0x0800	isr_comparatorA()
	0x1000	isr_timerB1()
	0x2000	isr_timerB0()
	0x4000	isr_nmi() associated with NMI
	0x8000	isr_nmi() associated with NMI, OFIFG, ACCVIFG

DISABLED_WDT	
Signature	-DISABLED_WDT : ushort const = 0x0
Visibility	private
Scope	classifier
Documentation	Set this variable to 0xFFFF if the Watch Dog Timer (WDT) is not used (respectively, used).

Operations

isr_P2	
Signature	isr_P2() : void
Visibility	public
Scope	classifier

isr_dummy2	
Signature	isr_dummy2() : void
Visibility	public
Scope	classifier

isr_timerA0	
Signature	isr_timerA0() : void
Visibility	public
Scope	classifier

isr_P1	
Signature	isr_P1() : void
Visibility	public
Scope	classifier
Documentation	Handles interrupts on PORT1 by calling a specific function for each enabled pin. It is called for each transition on pins, as enabled by the constructor

	InterruptHandler(). The following interrupts are honored: • Interrupt on BIT0 is associated with either rising edge (message header; calls MessageHandler::MessageHeader()) or falling edge (message footer ; calls MessageHandler::MessageFooter()) of CS signal.

isr_adc12	
Signature	isr_adc12() : void
Visibility	public
Scope	classifier

isr_wdt	
Signature	isr_wdt() : void
Visibility	public
Scope	classifier

isr_timerB1	
Signature	isr_timerB1() : void
Visibility	public
Scope	classifier

isr_timerB0	
Signature	isr_timerB0() : void
Visibility	public
Scope	classifier

isr_nmi	
Signature	isr_nmi() : void
Visibility	public
Scope	classifier

checkCode	
Signature	checkCode() : bool
Visibility	public
Scope	instance
Documentation	Computes signature of a block of code (256bytes). Every time it is called, it checks a different block of code, starting at address TBD, down to address TBD, then repeating forever.

UnusedInterrupts_master	
Signature	UnusedInterrupts_master()
Visibility	public
Scope	instance
Documentation	Disables all interrupts and the watchdog.

resetWDT	
Signature	resetWDT()
Visibility	public
Scope	classifier
Documentation	Resets Watch Dog Timer. It must be called by the user periodically, namely before the WDT triggers a reboot.
Code Body	WDTCTL = WDTPW + (WDTCTL & 0xFF) + WDTCNTCL; // resets WDT counter

configureWDT	
Signature	configureWDT(period : byte)
Visibility	public
Scope	classifier
Documentation	Configures WDT for counting SMCLK ticks depending on the value of **period** (0-3).
Code Body	WDTCTL = WDTPW + period & (WDTIS0+WDTIS1); // configures WDT for clock on SMCLK/32768/8192/512/64, depending on period=0/1/2/3

2.24. Class - IO_pin

Name	Value
Name	IO_pin
Stereotypes	SW, Struct
Documentation	Class describing a single pin of an IO port of the microcontroller. It has two attributes: port storing the address of the IO port pinmask storing the pin, as a mask with a 1 in the position corresponding to the desired pin. Pin operations can be implemented as follows: •

Attributes

port	
Signature	-port : byte*
Visibility	private
Scope	instance

pinmask	
Signature	-pinmask : byte
Visibility	private
Scope	instance

Operations

set	
Signature	set()
Visibility	public
Scope	instance
Code Body	(*port) |= pinmask;

2.25. Class - UART_MODES

Name	Value
Name	UART_MODES
Stereotypes	SW, enumeration

Attributes

SPI_MASTER_MODE	
Signature	-SPI_MASTER_MODE
Visibility	private
Scope	instance

SPI_SLAVE_MODE	
Signature	-SPI_SLAVE_MODE
Visibility	private
Scope	instance

RS232_MODE

Signature	-RS232_MODE
Visibility	private
Scope	instance

I2C_MASTER_MODE	
Signature	-I2C_MASTER_MODE
Visibility	private
Scope	instance

I2C_SLAVE_MODE	
Signature	-I2C_SLAVE_MODE
Visibility	private
Scope	instance

IRDA_MODE	
Signature	-IRDA_MODE
Visibility	private
Scope	instance

2.26. Class - mechanicalConfiguration

Name	**Value**
Name	mechanicalConfiguration
Stereotypes	SW
Documentation	The number of tiles in the satellite.

Attributes

addressPMT	
Signature	+addressPMT : byte const[NUMPMT] = {1,2,3,4,5}
Visibility	public
Scope	classifier
Documentation	The physical address of each tile on the OBDB

addressOBC	
Signature	+addressOBC : byte const[NUMOBC] = {0xFF}
Visibility	public
Scope	classifier

DEPHisOBC	
Signature	+DEPHisOBC : short const = TBD
Visibility	public
Scope	classifier
Documentation	The depth of OBC (1B9) tiles' history vector.

DEPHisPMT	
Signature	+DEPHisPMT : short const = TBD
Visibility	public
Scope	classifier
Documentation	The depth of PMT (1B8) tiles' history vector.

LENHisOBC	
Signature	+LENHisOBC : short const = Bk1A_1B8_Codes::LENGTH_HOUSEKEEPING
Visibility	public
Scope	classifier
Documentation	The length of OBC (1B9) tiles' history vector.

LENHisPMT	
Signature	+LENHisPMT : short const = Bk1A_1B8_Codes::LENGTH_HOUSEKEEPING
Visibility	public
Scope	classifier
Documentation	The length of PMT (1B8) tiles' history vector.

LENHkOBC	
Signature	+LENHkOBC : short const = Bk1A_1B9_Codes::LENGTH_HOUSEKEEPING
Visibility	public
Scope	classifier
Documentation	The length of OBC (1B9) tiles' housekeeping vector.

LENHkPMT	
Signature	+LENHkPMT : short const = Bk1A_1B8_Codes::LENGTH_HOUSEKEEPING
Visibility	public
Scope	classifier

Documentation	The length of PMT (1B8) tiles' housekeeping vector.

LENStatOBC	
Signature	+LENStatOBC : short const = 2*Bk1A_1B9_Codes::LENGTH_HOUSEKEEPING
Visibility	public
Scope	classifier
Documentation	The length of OBC (1B9) tiles' statistics vector.

LENStatPMT	
Signature	+LENStatPMT : short const = 2*Bk1A_1B8_Codes::LENGTH_HOUSEKEEPING
Visibility	public
Scope	classifier
Documentation	The length of PMT (1B8) tiles' statistics vector.

orientation	
Signature	+orientation : TileOrientation const[NUMTILES] = { {{{0,0,-1},{0,1,0},{1,0,0}}} , {{{0,0,-1},{-1,0,0},{0,1,0}}} , {{{1,0,0},{0,1,0},{0,0,1}}} , {{{0,0,-1},{0,-1,0},{-1,0,0}}} , {{{0,0,-1},{1,0,0},{0,-1,0}}} , {{{-1,0,0},{0,-1,0},{0,0,-1}}} }
Visibility	public
Scope	classifier
Documentation	Direction cosine matrix of each tile.

Wx	
Signature	+Wx : t_floatFactorByte const[NUMPMT][3] = { {{0,1},{1,1},{0,1}} , {{0,1},{0,1},{1,1}} , {{-1,1},{0,1},{0,1}} , {{0,1},{-1,1},{0,1}} , {{0,1},{0,1},{-1,1}} , }
Visibility	public
Scope	classifier
Documentation	??? Indicates the weight of each gyroscope on the total gyroscope measurement. It is based on the orientation of each tile, divided by the number of tiles along the same direction. It is the coefficient by which to multiply each gyroscopic measurement in order to get the total satellite spin, Each tile is associated with a vector which indicates the orientation of the positive z axis of the tile in the satellite reference frame, divided by the number of tiles along that direction.

Wy

Signature	+Wy : t_floatFactorByte const[NUMPMT][3] = { {{0,1},{1,1},{0,1}} , {{0,1},{0,1},{1,1}} , {{-1,1},{0,1},{0,1}} , {{0,1},{-1,1},{0,1}} , {{0,1},{0,1},{-1,1}}}
Visibility	public
Scope	classifier
Documentation	Indicates the weight of each gyroscope on the total gyroscope measurement. It is based on the orientation of each tile, divided by the number of tiles along the same direction. It is the coefficient by which to multiply each gyroscopic measurement in order to get the total satellite spin, Each tile is associated with a vector which indicates the orientation of the positive z axis of the tile in the satellite reference frame, divided by the number of tiles along that direction.

Wz	
Signature	+Wz : t_floatFactorByte const[NUMPMT][3] = { {{0,1},{1,1},{0,1}} , {{0,1},{0,1},{1,1}} , {{-1,1},{0,1},{0,1}} , {{0,1},{-1,1},{0,1}} , {{0,1},{0,1},{-1,1}}}
Visibility	public
Scope	classifier
Documentation	Indicates the weight of each gyroscope on the total gyroscope measurement. It is based on the orientation of each tile, divided by the number of tiles along the same direction. It is the coefficient by which to multiply each gyroscopic measurement in order to get the total satellite spin, Each tile is associated with a vector which indicates the orientation of the positive z axis of the tile in the satellite reference frame, divided by the number of tiles along that direction.

eigenvalues	
Signature	+eigenvalues : float const[3] = {1,1,1}
Visibility	public
Scope	classifier

magnetometer	
Signature	+magnetometer : byte[NUMPMT] = {1,1,1,1,1}
Visibility	public
Scope	classifier
Documentation	Indicates the presents of magnetometers on each PMT tile. - 0 : magnetometers are not present, - 1 : magnetometers are present.

reaction	
Signature	+reaction : byte[NUMPMT] = {1,1,1,1,1}
Visibility	public
Scope	classifier
Documentation	Indicates the presents of a reaction wheel on each PMT tile. - 0 : reaction wheel is not present, - 1 : reaction wheel is present.

sun	
Signature	+sun : byte[NUMPMT] = {1,1,1,1,1}
Visibility	public
Scope	classifier
Documentation	Indicates the presents of a sun sensor on each PMT tile. - 0 : Sun sensor is not present, - 1 : Sun sensor is present.

earth	
Signature	+earth : byte[NUMPMT] = {1,1,1,1,1}
Visibility	public
Scope	classifier
Documentation	Indicates the presents of a sun sensor on each PMT tile. - 0 : Sun sensor is not present, - 1 : Sun sensor is present.

gyroscope	
Signature	+gyroscope : byte[NUMPMT] = {1,1,1,1,1}
Visibility	public
Scope	classifier
Documentation	Indicates the presents of a gyroscope on each PMT tile. - 0 : gyroscope is not present, - 1 : gyroscope is present.

inertia	
Signature	+inertia : float const[3][3] = { {1,0,0} , {0,1,0} , {0,0,1}}
Visibility	public
Scope	classifier
Documentation	Indicates the presents of a inertia wheel on each PMT tile. - 0 : inertia wheel is not present, - 1 : inertia wheel is present.

2.27. Class - t_floatFactorByte

Name	Value
Name	t_floatFactorByte
Stereotypes	SW
Documentation	A class used to express non-integer multiplicative factors K by means of two byte coefficients. The factor is defined as the ratio between the M attribute and the D attribute: K = M / D.

Attributes

M	
Signature	+M : char
Visibility	public
Scope	instance

D	
Signature	+D : byte
Visibility	public
Scope	instance

Operations

multiply	
Signature	multiply(value : byte) : byte
Visibility	public
Scope	instance
Documentation	Multiplies argument value by factor K = M / D and returns the result.
Code Body	return (byte)(((short)value * M) / D);

divide	
Signature	divide(value : byte) : byte
Visibility	public
Scope	instance
Documentation	Multiplies argument value by factor K = M / D and returns the result.
Code Body	return (byte)(((short)value * D) / M);

multiply	
Signature	multiply(value : char) : char
Visibility	public
Scope	instance
Documentation	Multiplies argument value by factor K = M / D and returns the result.
Code Body	return (char)(((short)value * M) / D);

divide	
Signature	divide(value : char) : char
Visibility	public
Scope	instance
Documentation	Multiplies argument value by factor K = M / D and returns the result.
Code Body	return (char)(((short)value * D) / M);

multiply	
Signature	multiply(value : ushort) : ushort
Visibility	public
Scope	instance
Documentation	Multiplies argument value by factor K = M / D and returns the result.
Code Body	return (ushort)(((long)value * M) / D);

divide	
Signature	divide(value : ushort) : ushort
Visibility	public
Scope	instance
Documentation	Multiplies argument value by factor K = M / D and returns the result.
Code Body	return (ushort)(((long)value * D) / M);

multiply	
Signature	multiply(value : short) : short
Visibility	public
Scope	instance
Documentation	Multiplies argument value by factor K = M / D and returns the result.
Code Body	return (short)(((long)value * M) / D);

divide	
Signature	divide(value : short) : short

Visibility	public
Scope	instance
Documentation	Multiplies argument value by factor K = M / D and returns the result.
Code Body	return (short)(((long)value * D) / M);

2.28. Class - t_floatFactorShort

Name	Value
Name	t_floatFactorShort
Stereotypes	SW
Documentation	A class used to express non-integer multiplicative factors K by means of two short coefficients. The factor is defined as the ratio between the M attribute and the D attribute: K = M / D

Attributes

M	
Signature	+M : short
Visibility	public
Scope	instance

D	
Signature	+D : ushort
Visibility	public
Scope	instance

Operations

multiply	
Signature	multiply(value : byte) : byte
Visibility	public
Scope	instance
Documentation	Multiplies argument value by factor K = M / D and returns the result
Code Body	return (byte)(((short)value * M) / D);

divide	
Signature	divide(value : byte) : byte

Visibility	public
Scope	instance
Documentation	Divides argument value by factor K = M / D and returns the result
Code Body	return (byte)(((short)value * D) / M);

multiply	
Signature	multiply(value : char) : char
Visibility	public
Scope	instance
Documentation	Multiplies argument value by factor K = M / D and returns the result
Code Body	return (char)(((short)value * M) / D);

divide	
Signature	divide(value : char) : char
Visibility	public
Scope	instance
Documentation	Divides argument value by factor K = M / D and returns the result
Code Body	return (char)(((short)value * D) / M);

multiply	
Signature	multiply(value : ushort) : ushort
Visibility	public
Scope	instance
Documentation	Multiplies argument value by factor K = M / D and returns the result
Code Body	return (ushort)(((long)value * M) / D);

divide	
Signature	divide(value : ushort) : ushort
Visibility	public
Scope	instance
Documentation	Divides argument value by factor K = M / D and returns the result
Code Body	return (ushort)(((long)value * D) / M);

multiply	
Signature	multiply(value : short) : short
Visibility	public
Scope	instance

Documentation	Multiplies argument value by factor K = M / D and returns the result
Code Body	return (short)(((long)value * M) / D);

divide	
Signature	divide(value : short) : short
Visibility	public
Scope	instance
Documentation	Divides argument value by factor K = M / D and returns the result
Code Body	return (short)(((long)value * D) / M);

2.29. Class - t_Status

Name	**Value**
Name	t_Status
Stereotypes	SW
Documentation	Status information for the 1B8 Power Management Tile. It contains one status word for each of its major subsystems: (model element not found) for 1B1 Power Management Subsystem (model element not found) for 1B2 Attitude and Orbit Control System (model element not found) for 1B45 Inter-tile communication package It also contains the (model element not found) attribute for additional details on the last reported error.

2.30. Class - t_Configuration

Name	**Value**
Name	t_Configuration
Stereotypes	SW
Documentation	Configuration information for the 1B8 Power Management Tile. It contains one configuration word for each of its major subsystems: (model element not found) for 1B1 Power Management Subsystem (model element not found) for 1B2 Attitude and Orbit Control System It also contains other constant configuration data: (model element not found) containing the SW revision of the Tile (model element not found) containing the HW revision of the Tile All above information can be retrieved at run-time by means of the 1B45 package. The (model element not found) also contains other constant

	configuration data with additional geometrical, mechanical and electrical parameters on the tile, which can only be accessed at compile-time. Details on these are found in the attribute section.

2.31. Class - t_StatusWord

Name	Value
Name	t_StatusWord
Stereotypes	SW
Documentation	Generic status word for ARAMIS tiles. It is composed of a status attribute, plus all operations to: get the content of status word; set the content of individual bits of status word; reset the content of individual bits of status word; test the content of individual bits of status word; write the content of the whole status word;

Attributes

status	
Signature	-status : byte
Visibility	private
Scope	instance
Documentation	Generic status word for 1B45-compatible peripheral. Each bit indicates a different status condition, as enumerated by the other constant attributes of this class.

Operations

get	
Signature	get() : byte
Visibility	public
Scope	instance
Documentation	Returns status word.
Code Body	return status;

set	
Signature	set(flags : ushort, mask : byte)
Visibility	public
Scope	instance

Documentation	Sets all bits of status word corresponding to bits set to 1 in argument **mask**.
Code Body	status |= mask;

reset	
Signature	reset(flags : ushort, mask : byte)
Visibility	public
Scope	instance
Documentation	Resets all bits of status word corresponding to bits set to 1 in argument **mask**.
Code Body	status &= ~mask;

write	
Signature	write(mask : byte)
Visibility	public
Scope	instance
Documentation	Overwrites the whole status word with the argument **mask**.
Code Body	status = mask;

test	
Signature	test(flags : ushort, mask : byte)
Visibility	public
Scope	instance
Documentation	Returns true if all bits of status word corresponding to bits set to 1 in argument **mask** are set.
Code Body	return ((status & mask) == mask);

2.32. Class - Processor

Name	**Value**
Name	Processor
Stereotypes	SW

Operations

Processor	
Signature	Processor()
Visibility	public
Scope	instance

Code Body	init();

init	
Signature	init()
Visibility	public
Scope	classifier

Code Body	#if MSP430x43x
	// configures FLL
	SCFI0 = 0; // set clock divide to 1 in FLL
	// configures SMCLK = DCO/1; the other clocks are irrelevant
	FLL_CTL0 |= DCOPLUS;
	// configures XTAL2 to off; SMCLK on; MCLK = DCO; SMCLK = DCO
	FLL_CTL1 &= ~(SMCLKOFF | XT2OFF | SELM0 | SELM1 | SELS | FLL_DIV0 | FLL_DIV1);
	FLL_CTL1 |= XT2OFF;
	
	//setDCO(CALBC1_8MHZ, CALDCO_8MHZ);
	setACLK(XT1CLK, 1);
	setMCLK(XT1CLK, 1);
	setSMCLK(XT1CLK, 1);
	resetFlags();
	IE1 &= ~OFIE;
	IFG1 &= ~OFIFG;
	#endif
	
	#if MSP430x24x
	  BCSCTL2 = 0*SELM0 | 0*DIVM0 | 0*SELS | 0*DIVS0 | 0*DCOR;
	  BCSCTL3 = 0*XT2S0 | 0*LFXT1S0 | 0*XCAP0;
	  setDCO(CALBC1_8MHZ, CALDCO_8MHZ);
	  setACLK(XT1CLK, 1);
	  setMCLK(XT1CLK, 1);
	  setSMCLK(XT1CLK, 1);
	  resetFlags();
	  IE1 &= ~OFIE;
	  IFG1 &= ~OFIFG;
	#endif
	
	#if MSP430x54x
	  UCSCTL6 = 0*XT2DRIVE0 + 0*XT2BYPASS + XT2OFF + 0*XT1DRIVE0 + 0*XTS + 0*XT1BYPASS + 0*XCAP0 + SMCLKOFF + XT1OFF;
	  setACLK(XT1CLK, 1);
	  setMCLK(XT1CLK, 1);
	  setSMCLK(XT1CLK, 1);
	  resetFlags();
	  SFRIE1 &= ~OFIE;
	  SFRIFG1 &= ~OFIFG;
	#endif

activateXT1	
Signature	activateXT1(freq : ulong)
Visibility	public

Scope	classifier

Code Body	#if MSP430x43x
	// configures XTAL2 to off; SMCLK on; MCLK = DCO; SMCLK = DCO
	//FLL_CTL1 &= ~(XT1OFF);
	#endif
	
	// RSELx = 11; DCOx = 3; MODx = 0
	#if MSP430x24x
	  BCSCTL1 &= ~(XT1DRIVE0 | XT1DRIVE1 | XTS | XT1BYPASS | XT1OFF);
	  if (freq < 10000) {}
	  else if (freq < 8000000)
	    BCSCTL1 |= XTS;
	  else if (freq < 16000000)
	    BCSCTL1 |= (XTS | 1*XT1DRIVE0);
	  else if (freq < 24000000)
	    BCSCTL1 |= (XTS | 2*XT1DRIVE0);
	  else
	    BCSCTL1 |= (XTS | 3*XT1DRIVE0);
	#endif
	
	#if MSP430x54x
	  P7SEL |= BIT0 | BIT1;
	  UCSCTL6 &= ~(XT1DRIVE0 | XT1DRIVE1 | XTS | XT1BYPASS | XT1OFF);
	  if (freq < 10000) {}
	  else if (freq < 8000000)
	    UCSCTL6 |= XTS;
	  else if (freq < 16000000)
	    UCSCTL6 |= (XTS | 1*XT1DRIVE0);
	  else if (freq < 24000000)
	    UCSCTL6 |= (XTS | 2*XT1DRIVE0);
	  else
	    UCSCTL6 |= (XTS | 3*XT1DRIVE0);
	#endif

activateXT2	
Signature	activateXT2(freq : ulong)
Visibility	public
Scope	classifier

Code Body	#if MSP430x43x
	//TBD...
	FLL_CTL1 &= ~XT2OFF;
	#endif
	
	#if MSP430x24x
	  BCSCTL1 &= ~(XT2BYPASS | XT2OFF);
	#endif
	
	#if MSP430x54x
	  P5SEL |= BIT2 | BIT3;
	  UCSCTL6 &= ~(XT2BYPASS | XT2OFF);
	#endif

setDCO	
Signature	setDCO(range : ushort, freqFactor : ushort)
Visibility	public
Scope	classifier

Code Body	#if MSP430x43x
	///TBD ...
	SCFQCTL = 0;
	#endif
	
	#if MSP430x24x
	DCOCTL = freqFactor;
	BCSCTL1 |= ((range & 0x7) * RSEL0);
	#endif
	
	#if MSP430x54x
	UCSCTL0 = MOD0 * (freqFactor & 0x3FF);
	UCSCTL1 = DCORSEL0 * (range & 0x7);
	if ((freqFactor & 0x1F) == 0)
	UCSCTL1 |= DISMOD;
	#endif
	
	#if CHIPCON
	
	#endif

setFLL	
Signature	setFLL(divide1 : ushort, divide2 : ushort, multiply : ushort, source : t_clockSource) : bool
Visibility	public
Scope	classifier

Code Body	#if MSP430x43x
	//TBD ...
	#endif
	
	#if MSP430x24x
	TBD ...
	#endif
	
	#if MSP430x54x
	UCSCTL2 = 0;
	if (multiply < 2) return false;
	if (multiply > 4096) return false;
	UCSCTL2 |= ((multiply-1) & 0x3FF) * FLLN0;
	switch (divide2) {
	case 1: break;
	case 2: UCSCTL2 |= 1*FLLD0; break;
	case 4: UCSCTL2 |= 2*FLLD0; break;
	case 8: UCSCTL2 |= 3*FLLD0; break;
	case 16: UCSCTL2 |= 4*FLLD0; break;
	case 32: UCSCTL2 |= 5*FLLD0; break;

	default: return false;
	}
	
	UCSCTL3 = 0;
	switch (divide1) {
	case 1: break;
	case 2: UCSCTL3 |= 1*FLLREFDIV0; break;
	case 4: UCSCTL3 |= 2*FLLREFDIV0; break;
	case 8: UCSCTL3 |= 3*FLLREFDIV0; break;
	case 12: UCSCTL3 |= 4*FLLREFDIV0; break;
	case 16: UCSCTL3 |= 5*FLLREFDIV0; break;
	default: return false;
	}
	switch (source) {
	case XT1CLK: UCSCTL3 |= SELREF0 * 0; break;
	case REFOCLK: UCSCTL3 |= SELREF0 * 0; break;
	case XT2CLK: UCSCTL3 |= SELREF0 * 0; break;
	default: return false;
	}
	#endif
	
	return true;

setACLK	
Signature	setACLK(source : t_clockSource, divide : ushort) : bool
Visibility	public
Scope	classifier

Code Body	#if MSP430x43x
	#endif
	
	#if MSP430x24x
	BCSCTL2 &= 7*SELA0;
	switch (source) {
	case XT1CLK: break;
	case VLOCLK: BCSCTL2 |= 1*SELA0; break;
	case REFOCLK: BCSCTL2 |= 2*SELA0; break;
	case DCOCLK: BCSCTL2 |= 3*SELA0; break;
	case DCOCLKDIV: BCSCTL2 |= 4*SELA0; break;
	case XT2CLK: BCSCTL2 |= 5*SELA0; break;
	default: return false;
	}
	BCSCTL1 &= 7*(DIVA0);
	switch (divide) {
	case 1: break;
	case 2: BCSCTL1 |= 1*(DIVA0); break;
	case 4: BCSCTL1 |= 2*(DIVA0); break;
	case 8: BCSCTL1 |= 3*(DIVA0); break;
	case 16: BCSCTL1 |= 4*(DIVA0); break;
	case 32: BCSCTL1 |= 5*(DIVA0); break;
	default: return false;
	}
	#endif
	
	#if MSP430x54x
	UCSCTL4 &= 7*SELA0;

	switch (source) {
	case XT1CLK: break;
	case VLOCLK: UCSCTL4 |= 1*SELA0; break;
	case REFOCLK: UCSCTL4 |= 2*SELA0; break;
	case DCOCLK: UCSCTL4 |= 3*SELA0; break;
	case DCOCLKDIV: UCSCTL4 |= 4*SELA0; break;
	case XT2CLK: UCSCTL4 |= 5*SELA0; break;
	default: return false;
	}
	
	UCSCTL5 &= 7*(DIVPA0+DIVA0);
	switch (divide) {
	case 1: break;
	case 2: UCSCTL5 |= 1*(DIVPA0+DIVA0); break;
	case 4: UCSCTL5 |= 2*(DIVPA0+DIVA0); break;
	case 8: UCSCTL5 |= 3*(DIVPA0+DIVA0); break;
	case 16: UCSCTL5 |= 4*(DIVPA0+DIVA0); break;
	case 32: UCSCTL5 |= 5*(DIVPA0+DIVA0); break;
	default: return false;
	}
	#endif
	
	return true;

setMCLK	
Signature	setMCLK(source : t_clockSource, divide : ushort) : bool
Visibility	public
Scope	classifier

Code Body	#if MSP430x43x
	FLL_CTL1 &= 3*SELM0;
	switch (source) {
	case VLOCLK: FLL_CTL1 |= 3*SELM0; break;
	case DCOCLK: FLL_CTL1 |= 0*SELM0; break;
	case XT2CLK: FLL_CTL1 |= 2*SELM0; break;
	default: return false;
	}
	#endif
	
	#if MSP430x24x
	BCSCTL2 &= 7*SELM0;
	switch (source) {
	case XT1CLK: break;
	case VLOCLK: BCSCTL2 |= 1*SELM0; break;
	case REFOCLK: BCSCTL2 |= 2*SELM0; break;
	case DCOCLK: BCSCTL2 |= 3*SELM0; break;
	case DCOCLKDIV: BCSCTL2 |= 4*SELM0; break;
	case XT2CLK: BCSCTL2 |= 5*SELM0; break;
	default: return false;
	}
	BCSCTL1 &= 7*(DIVM0);
	switch (divide) {
	case 1: break;
	case 2: BCSCTL1 |= 1*(DIVM0); break;
	case 4: BCSCTL1 |= 2*(DIVM0); break;
	case 8: BCSCTL1 |= 3*(DIVM0); break;

	case 16: BCSCTL1 |= 4*(DIVM0); break;
	case 32: BCSCTL1 |= 5*(DIVM0); break;
	default: return false;
	}
	#endif
	
	#if MSP430x54x
	UCSCTL4 &= 7*SELM0;
	switch (source) {
	case XT1CLK: break;
	case VLOCLK: UCSCTL4 |= 1*SELM0; break;
	case REFOCLK: UCSCTL4 |= 2*SELM0; break;
	case DCOCLK: UCSCTL4 |= 3*SELM0; break;
	case DCOCLKDIV: UCSCTL4 |= 4*SELM0; break;
	case XT2CLK: UCSCTL4 |= 5*SELM0; break;
	default: return false;
	}
	
	UCSCTL5 &= 7*(DIVM0);
	switch (divide) {
	case 1: break;
	case 2: UCSCTL5 |= 1*(DIVM0); break;
	case 4: UCSCTL5 |= 2*(DIVM0); break;
	case 8: UCSCTL5 |= 3*(DIVM0); break;
	case 16: UCSCTL5 |= 4*(DIVM0); break;
	case 32: UCSCTL5 |= 5*(DIVM0); break;
	default: return false;
	}
	#endif
	
	return true;

setSMCLK	
Signature	setSMCLK(source : t_clockSource, divide : ushort) : bool
Visibility	public
Scope	classifier

Code Body	#if MSP430x43x
	FLL_CTL1 &= 1*SELS;
	switch (source) {
	case DCOCLK: FLL_CTL1 |= 0*SELM0; break;
	case XT2CLK: FLL_CTL1 |= 1*SELM0; break;
	default: return false;
	}
	#endif
	
	#if MSP430x24x
	BCSCTL2 &= 7*SELS0;
	switch (source) {
	case XT1CLK: break;
	case VLOCLK: BCSCTL2 |= 1*SELS0; break;
	case REFOCLK: BCSCTL2 |= 2*SELS0; break;
	case DCOCLK: BCSCTL2 |= 3*SELS0; break;
	case DCOCLKDIV: BCSCTL2 |= 4*SELS0; break;
	case XT2CLK: BCSCTL2 |= 5*SELS0; break;
	default: return false;

	}
	BCSCTL1 &= 7*(DIVS0);
	switch (divide) {
	case 1: break;
	case 2: BCSCTL1 |= 1*(DIVS0); break;
	case 4: BCSCTL1 |= 2*(DIVS0); break;
	case 8: BCSCTL1 |= 3*(DIVS0); break;
	case 16: BCSCTL1 |= 4*(DIVS0); break;
	case 32: BCSCTL1 |= 5*(DIVS0); break;
	default: return false;
	}
	#endif
	
	#if MSP430x54x
	UCSCTL4 &= 7*SELS0;
	switch (source) {
	case XT1CLK: break;
	case VLOCLK: UCSCTL4 |= 1*SELS0; break;
	case REFOCLK: UCSCTL4 |= 2*SELS0; break;
	case DCOCLK: UCSCTL4 |= 3*SELS0; break;
	case DCOCLKDIV: UCSCTL4 |= 4*SELS0; break;
	case XT2CLK: UCSCTL4 |= 5*SELS0; break;
	default: return false;
	}
	
	UCSCTL5 &= 7*(DIVS0);
	switch (divide) {
	case 1: break;
	case 2: UCSCTL5 |= 1*(DIVS0); break;
	case 4: UCSCTL5 |= 2*(DIVS0); break;
	case 8: UCSCTL5 |= 3*(DIVS0); break;
	case 16: UCSCTL5 |= 4*(DIVS0); break;
	case 32: UCSCTL5 |= 5*(DIVS0); break;
	default: return false;
	}
	UCSCTL6 &= ~SMCLKOFF;
	#endif
	
	return true;

resetFlags	
Signature	resetFlags()
Visibility	public
Scope	classifier

Code Body	#if MSP430x43x
	FLL_CTL0 &= ~0xF;
	#endif
	
	#if MSP430x24x
	TBD ...
	#endif
	
	#if MSP430x54x
	UCSCTL7 = 0; // resetting fault flags;
	UCSCTL8 = 0; // resetting conditional requests for clocks

	#endif

2.33. Class - t_clockSource

Name	Value
Name	t_clockSource
Stereotypes	SW, enumeration

Attributes

XT1CLK	
Signature	-XT1CLK
Visibility	private
Scope	instance

VLOCLK	
Signature	-VLOCLK
Visibility	private
Scope	instance

REFOCLK	
Signature	-REFOCLK
Visibility	private
Scope	instance

DCOCLK	
Signature	-DCOCLK
Visibility	private
Scope	instance

DCOCLKDIV	
Signature	-DCOCLKDIV
Visibility	private
Scope	instance

XT2CLK	
Signature	-XT2CLK
Visibility	private

Scope	instance

2.34. Class - OBC

Name	Value
Name	OBC
Stereotypes	Interface
Documentation	On Board Computer.

2.35. Class - OBDB

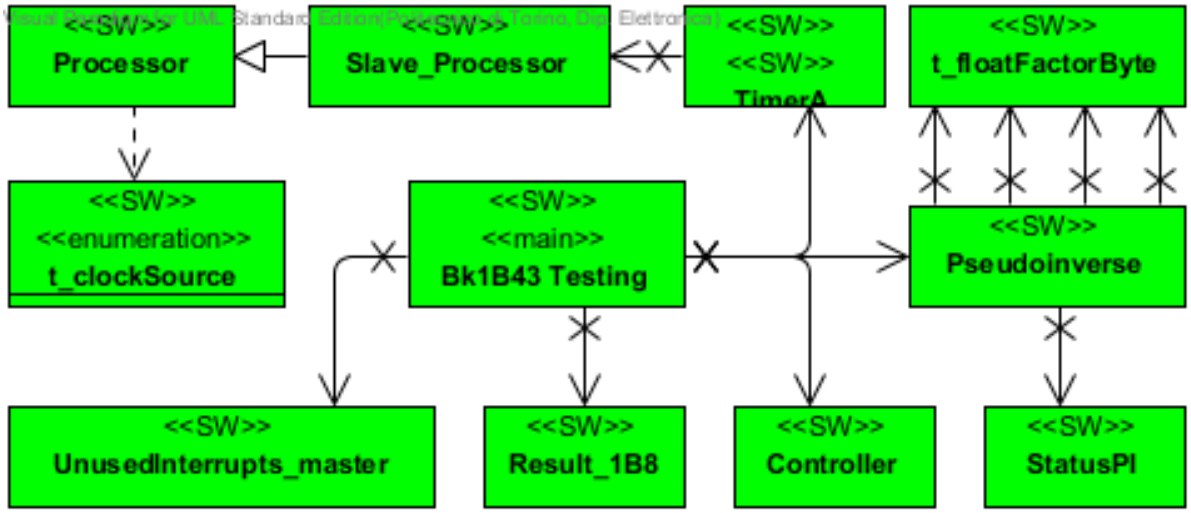
Name	Value
Name	OBDB
Stereotypes	Interface
Documentation	On Board Data Bus.

Capitolo 5

Test effettuati

Un progetto, di qualunque natura sia, non si può dire completo se prima non ne viene verificato il funzionamento. Il collaudo è infatti parte integrante dello sviluppo del progetto ed è molto importante.
Per il collaudo del progetto 1B43 Housekeeping Management sono state preparate tre prove, che verranno descritte nel dettaglio nei paragrafi seguenti; questi test hanno lo scopo di verificare il corretto funzionamento del codice sorgente delle operazioni più importanti.
Il primo test serve a verificare il funzionamento dell'operazione **float2floatFactorByte()** della classe Pseudoinverse che, come visto in precedenza, ha come scopo la conversione di dati a virgola mobile in variabili di tipo *t_floatFactorByte*.
Nel secondo verrà collaudato il metodo **Update()** della classe Pseudoinverse, il cui compito è aggiornare le matrici pseudoinverse in base allo stato del satellite. Queste matrici sono utilizzate durante l'analisi per il calcolo di parametri vettoriali quali, ad esempio, la posizione del satellite rispetto al Sole.
In ultimo, il modulo verrà messo in comunicazione con una tile 1B8 fittizia, in modo da poter collaudare il protocollo di comunicazione in una caso reale e verificare se l'algoritmo di analisi fornisce dei valori corretti.
Come nel capitolo precedente, le tile 1B9 (TLC) non vengono utilizzate nella trattazione per via dello sviluppo in fase embrionale. Si fa comunque notare che le operazioni utilizzate nella comunicazione (tutte le operazioni **getXX()** della classe Manager_1B9) hanno il medesimo codice sorgente di quelle utilizzate dalle PMT tile; per cui, a meno di errori nell'hardware, il funzionamento su un tipo di tile garantisce in prima approssimazione anche quello sull'altro.
Ovviamente non appena verrà resa disponibile una versione più evoluta delle tile 1B9 sarà necessario fare delle prove di comunicazione con il modulo 1B43 Housekeeping Management e verificarne il corretto funzionamento.

1. Class Diagram - Bk1B43-K GSE



In questa sezione verrà illustrato il Class Diagram Bk1B43-K GSE, un diagramma "di supporto" al Bk1B43 Housekeeping Management, come già anticipato all'inizio del capitolo. Infatti lo schema oggetto di questo paragrafo è stato utilizzato per simulare il corretto

funzionamento del diagramma principale del progetto, oltre che per sottoporlo ad alcuni test, descritti in questo capitolo.
Si è scelto di non includere tutte le dipendenze della classe Controller per rendere il diagramma più piccolo e di facile lettura, ma questo ha portato un piccolo inconveniente. Infatti il tool Istant Generator di Visual Paradigm genererà solo il codice delle classi nel contenute diagramma, ma non le loro dipendenze; di conseguenza per simulare il modulo è necessario "avvertire" il tool di generare anche il codice del diagramma Bk1B43 Housekeeping Management.
Anche in questo caso, le classi verranno descritte singolarmente, saltando ovviamente quelle illustrate nella sezione precedente. Di seguito sono elencate le classi oggetto di questo paragrafo:

- Bk1B43 Testing
- TimerA
- Slave_Processor

La classe Bk1B43 Testing contiene il metodo *main* ed è l'elemento centrale del sistema di simulazione, oltre a fornire un esempio di utilizzo del modulo 1B43 Housekeeping Management. La classe TimerA viene invece utilizzata per temporizzare il sistema, infatti la letture a l'analisi delle telemetrie vengono effettuate ogni minuto; l'attivazione viene fatta tramite *interrupt*. Nel tempo che intercorre tra una lettura e l'altra, se non è necessario aggiornare le matrici pseudoinverse, il microprocessore viene messo in modalità risparmio energetico.

1.1. Class - Bk1B43 Testing

Name	Value
Name	Class - Bk1B43 Testing
Stereotypes	SW, main

Attributes

man	
Signature	-man : Controller
Visibility	private
Scope	classifier

res	
Signature	-res : Result_1B8
Visibility	private
Scope	classifier

PI	
Signature	-PI : Pseudoinverse
Visibility	private
Scope	classifier

T	
Signature	-T : TimerA
Visibility	private
Scope	classifier

isr	
Signature	-isr : UnusedInterrupts_master
Visibility	private
Scope	classifier

Operations

main	
Signature	main()
Visibility	public
Scope	instance

Code Body	#ifdef __DEBUG
	int M,S,G,E = 0;
	float a;
	GlobalTypes::t_floatFactorByte b;
	#endif
	
	__enable_interrupt();
	
	// Disable watchdog timer
	WDTCTL = WDTPW + WDTHOLD;
	
	#ifdef __DEBUG
	a = 1.980;
	b = PI.float2floatFactorByte(a);
	printf("Test 1.1 : %f <-> %i / %i\n", b.M, b.D);
	a = -1.980;
	b = PI.float2floatFactorByte(a);
	printf("Test 1.2 : %f <-> %i / %i\n", b.M, b.D);
	a = 3.245;
	b = PI.float2floatFactorByte(a);
	printf("Test 2.1 : %f <-> %i / %i\n", b.M, b.D);
	a = -3.245;
	b = PI.float2floatFactorByte(a);
	printf("Test 2.2 : %f <-> %i / %i\n", b.M, b.D);
	a = 130;
	b = PI.float2floatFactorByte(a);
	printf("Test 3.1 : %f <-> %i / %i\n", b.M, b.D);
	a = -130;
	b = PI.float2floatFactorByte(a);
	printf("Test 3.2 : %f <-> %i / %i\n", b.M, b.D);
	a = 0.0003;
	b = PI.float2floatFactorByte(a);
	printf("Test 4.1 : %f <-> %i / %i\n", b.M, b.D);

```
a  = -0.0003;
b = PI.float2floatFactorByte(a);
printf("Test 4.2 : %f <-> %i / %i\n", b.M, b.D);

PI.Update(Bk1A_1B8_Codes::MAGNETIC_FIELD_X,
StatusPI::magnetometer);
PI.Update(Bk1A_1B8_Codes::SPIN_Z, StatusPI::gyroscope);
PI.Update(Bk1A_1B8_Codes::SUN_SENSOR_X, StatusPI::sun);
PI.Update(Bk1A_1B8_Codes::EARTH_SENSOR_X,
StatusPI::earth);
#endif

T.init();
T.start();

while(1)
{
  if(PI.PIstat.get() != 0)
  {
    man.computeSystemUpdates();

#ifdef __DEBUG
    for(int j = 0; j < mechanicalConfiguration<6,5,1>::NUMPMT; j++)
    {
      if((bool)PI.PIstat.magnetometer[j])
          M++;
      if((bool)PI.PIstat.gyroscope[j])
          G++;
      if((bool)PI.PIstat.sun[j])
          S++;
      if((bool)PI.PIstat.earth[j])
          E++;
    }

    printf("Mag Field\n");
    for(int i = 0; i < 3; i++)
      for(int j = 0; j < 2*M; j++)
          printf("%i-%u => %i/%i\n", i, j, PI.Bpi[i][j].M, PI.Bpi[i][j].D);

    printf("\nSpin\n");
    for(int i = 0; i < 3; i++)
      for(int j = 0; j < G; j++)
          printf("%i-%u => %i/%i\n", i, j, PI.Zpi[i][j].M, PI.Zpi[i][j].D);

    printf("\nSun Sensor\n");
    for(int i = 0; i < 3; i++)
      for(int j = 0; j < 2*S; j++)
          printf("%i-%u => %i/%i\n", i, j, PI.Spi[i][j].M, PI.Spi[i][j].D);

    printf("\nEarth Sensor\n");
    for(int i = 0; i < 3; i++)
    for(int j = 0; j < 2*E; j++)
    printf("%i-%u => %i/%i\n", i, j, PI.Epi[i][j].M, PI.Epi[i][j].D);
#endif

  }else
    __low_power_mode_0();      // Low Power Mode 0  => CPU off
}
```

1.2. Class - Slave_Processor

Name	Value
Name	Class - Slave_Processor
Stereotypes	SW

Attributes

CLOCK_FREQ	
Signature	+CLOCK_FREQ : long const = 1048576
Visibility	public
Scope	classifier

1.3. Class - TimerA

Name	Value
Name	Class - TimerA
Stereotypes	SW, SW
Documentation	The hardware and driver routines for TimerA.

Attributes

proc	
Signature	-proc : Slave_Processor
Visibility	private
Scope	classifier

CLOCK_FREQ	
Signature	+CLOCK_FREQ : ulong const = 8000000
Visibility	public
Scope	classifier
Documentation	Clock frequency of processor, on SMCLK input; in Hz

TIMER_FREQ	
Signature	+TIMER_FREQ : ulong const = 100
Visibility	public
Scope	classifier
Documentation	Frequency (in Hz) at which TimerA calls interrupt service routine isr_timerA0. Actual frequency might be slightly different, as

	permitted by the chosen processor clock frequency. Frequency accuracy depends on accuracy of processor clock.

TIMER_COUNT	
Signature	-TIMER_COUNT : ushort = 0
Visibility	private
Scope	classifier
Documentation	Numbers of times which TimerA calls interrupt service routine isr_timerA0.

TIMER_COUNT2	
Signature	-TIMER_COUNT2 : byte = 0
Visibility	private
Scope	classifier

cnt	
Signature	-cnt : byte = 5
Visibility	private
Scope	classifier

Operations

TimerA	
Signature	TimerA()
Visibility	public
Scope	instance
Documentation	Configures TimerA as from operation init.
Code Body	init(); reset();

init	
Signature	init()
Visibility	public
Scope	classifier
Documentation	Initializes timer to have clock from SMCLK (having frequency CLOCK_FREQ Hz) divided by 8 and to operate in compare up mode and to generate interrupt (when enabled) at rate TIMER_FREQ Hz; interrupt disabled. For MSP430, TIMER_FREQ (CLOCK_FREQ / 2^16 / 8).

Code Body	#if MSP430x43x || MSP430x24x
	// Timer A control register
	// configures Timer A for up mode, clock on SMCLK divided by 8; interrupt disabled; timer frequency = TIMER_FREQ
	TACTL = TASSEL1 // source clock from SMCLK
	  | ID1 | ID0 // Input divide by 8
	  | MC0 // up mode (counts from 0 up to TACCR0 included)
	  | TACLR ; // reset
	// Interrupt disabled
	TACCTL0 = 0; // Capture mode = no capture
	// output mode = output bit value (normal port operation)
	// other bits are immaterial
	TACCR0 = (unsigned int) ((CLOCK_FREQ / TIMER_FREQ / 8) - 1); // count period such as to trigger at TIMER_FREQ Hz, supposing processor clock frequency at CLOCK_FREQ Hz.
	#endif
	
	#if MSP430x54x
	// Timer A control register
	// configures Timer A for up mode, clock on SMCLK divided by 8; interrupt disabled; timer frequency = TIMER_FREQ
	TA0CTL = TASSEL1 // source clock from SMCLK
	  | ID1 | ID0 // Input divide by 8
	  | MC0 // up mode (counts from 0 up to TACCR0 included)
	  | TACLR ; // reset
	// Interrupt disabled
	TA0CCTL0 = 0; // Capture mode = no capture
	// output mode = output bit value (normal port operation)
	// other bits are immaterial
	TA0CCR0 = (unsigned int) ((CLOCK_FREQ / TIMER_FREQ / 8) - 1); // count period such as to trigger at TIMER_FREQ Hz, supposing processor clock frequency at CLOCK_FREQ Hz.
	#endif
	
	reset();

reset	
Signature	reset()
Visibility	public
Scope	classifier
Documentation	Resets TimerA counting. The first interrupt is generated at the end of counting, that is, after 1/TIMER_FREQ seconds.
Code Body	#if MSP430x43x || MSP430x24x TACTL |= TACLR; TACCTL0 &= CCIFG; #endif #if MSP430x54x TA0CTL |= TACLR; TA0CCTL0 &= CCIFG; #endif

start	
Signature	start()
Visibility	public
Scope	classifier
Documentation	Resets and starts TimerA counting. The first interrupt is generated at the end of counting, that is, after 1/TIMER_FREQ seconds.

Code Body	reset();
	
	#if MSP430x43x || MSP430x24x
	// enables interrupt
	TACCTL0 |= CCIE;
	#endif
	
	#if MSP430x54x
	// enables interrupt
	TA0CCTL0 |= CCIE;
	#endif

disable	
Signature	disable()
Visibility	public
Scope	classifier
Documentation	Disables TimerA interrupts. No more interrupt is therefore generated.
Code Body	#if MSP430x43x || MSP430x24x // disables interrupt TACCTL0 &= ~CCIE; #endif #if MSP430x54x // disables interrupt TA0CCTL0 &= ~CCIE; #endif

read	
Signature	read() : ushort
Visibility	public
Scope	classifier
Documentation	Reads TimerA counting. Counting starts from 0, therefore an immediate call to read right after start will return either 0 or a very small number.
Code Body	#if MSP430x43x || MSP430x24x return TAR; #endif #if MSP430x54x

	return TA0R; #endif

isr_timerA0	
Signature	isr_timerA0()
Visibility	public
Scope	classifier
Documentation	Interrupt service routine for TimerA, which is periodically invoked at frequency TIMER_FREQ Hz.

Code Body	__low_power_mode_off_on_exit();
	
	if(TIMER_COUNT < 60*TIMER_FREQ -1)
	  TIMER_COUNT++;
	else
	{
	 disable();
	 Controller::getHousekeeping();
	#ifdef __DEBUG
	 for(int i = 0; mechanicalConfiguration<6,5,1>::LENHkPMT; i++)
	  printf("hk[0][%i]\t%i\n", i, Controller::manPMT.hkPMT[0][i]);
	
	 Controller::computeStatistics();
	 Controller::computeHistory();
	  if(TIMER_COUNT2 < 10)
	  TIMER_COUNT2++;
	  else
	  {
	  TIMER_COUNT2 = 0;
	  StatusPI::magnetometer[2] = 0;
	  StatusPI::gyroscope[2] = 0;
	  }
	#else
	 Controller::getStatistics();
	 Controller::getHistory();
	 Controller::getConfiguration();
	#endif
	 Controller::analyze();
	 TIMER_COUNT = 0;
	}
	
	__enable_interrupt();
	start();

radiationCheck	
Signature	radiationCheck() : bool
Visibility	public
Scope	classifier
Documentation	Checks if any of the TimerA configuration registers (which should be constant) has been corrupted by a SEU.
Code Body	// TBD TO BE REWRITTEN !!!

```
#if MSP430x43x || MSP430x24x
//TBD
if ( ! (
((TACTL & ~(TACLR | TAIFG)) == (TASSEL1 | ID1 | ID0 | MC0))
&&
((TACCTL0 & ~(CCIE | CCI | OUT | COV | SCCI)) == 0) &&
(TACCR0 == (unsigned int) ((CLOCK_FREQ / TIMER_FREQ / 8) -
1))
) ) {
TimerA();
return false;
}
return true;
#endif

#if MSP430x54x
//TBD
if ( ! (
((TA0CTL & ~(TACLR | TAIFG)) == (TASSEL1 | ID1 | ID0 | MC0))
&&
((TA0CCTL0 & ~(CCIE | CCI | OUT | COV | SCCI)) == 0) &&
(TA0CCR0 == (unsigned int) ((CLOCK_FREQ / TIMER_FREQ / 8) -
1))
) ) {
TimerA();
return false;
}
return true;
#endif

#if CHIPCON

#endif
```

2. Il metodo *float2floatFactorByte*

Il primo test è relativo alla verifica della bontà dell'algoritmo per l'approssimazione razionale di numeri reali, che è ciò su cui si basano le operazioni **float2floatFactorByte()** e **float2floatFactorShort()** della classe Pseudoinverse.
Queste operazioni sono state pensate per limitare l'uso di variabili di tipo *float*, le quali presentano alcuni inconvenienti, soprattutto se il codice è installato su microprocessori come quelli della famiglia MSP430 [10–12].

Per verificare il corretto funzionamento di questo algoritmo si utilizza la funzione **rats** del linguaggio MATLAB; questa funzione ricava l'approssimazione razionale del dato fornito come parametro.
Sono stati fatti 4 test, ognuno dei quali è composto da due prove, che coprono tutti casi possibili ai cui l'algoritmo può operare; i risultati sono visibili nella tabella alla pagina seguente.
Tolto il test 1, dove i valori attesi e calcolati coincidono, è necessario spiegare i risultati ottenuti negli altri tre test.

Per il test 2 il problema è dovuto all'accuratezza dell'approssimazione. Infatti l'algoritmo controlla ad ogni iterazione se il dato è rappresentabile su 8 bit, con e senza segno. Se è possibile, raffina il risultato con una nuova iterazione, altrimenti si ferma; quindi i valori calcolati possono essere considerati esatti, soltanto meno accurati di quelli attesi.
Gli errori nei test 3 e 4 sono dovuti al fatto che il risultato eccede il massimo (o minimo) valore rappresentabile da una variabile *t_floatFactorByte*, come tutti i numeri non compresi nell'intervallo $-128 \leq x \leq 127$ (come nel caso del test 3) oppure quelli più piccoli di *1/255* (test 4). Per questo motivo quando il valore è troppo grande l'algoritmo fissa il risultato al massimo valore possibile su un *signed char*, mentre quando è troppo piccolo restituisce *0*.

Test	Numero Reale	MATLAB	*float2floatFactorByte()*
1	1.980	$99/50$	$99/50$
	-1.980	$-99/50$	$-99/50$
2	3.245	$649/200$	$13/4$
	-3.245	$-649/200$	$-13/4$
3	130.0	$130/1$	$127/1$
	-130.0	$-130/1$	$-127/1$
4	0.0003	$3/1000$	$0/1$
	-0.0003	$-3/1000$	$0/1$

3. Il metodo *Update*

Nonostante quanto detto in precedenza, il calcolo delle matrici pseudo-inverse viene fatto utilizzando delle variabili di tipo float. Si è scelta questa soluzione perché l'operazione **Update()** viene invocata solo in casi eccezionali e per questo incide poco sui consumi del microprocessore.
L'operazione **Update()** della classe Pseudoinverse ha il compito di costruire una nuova matrice da invertire in base alla configurazione del satellite e calcolarne la pseudo-inversa, utilizzando il metodo di Decomposizione ai Valori Singolari (SVD). La matrice ottenuta verrà poi "convertita" da float a *t_floatFactorByte* tramite la funzione **float2floatFactorByte()**.

Anche in questo caso i risultati dell'elaborazione verranno confrontati con quelli ottenuti utilizzando la funzione **pinv** del linguaggio MATLAB.
L'algoritmo che implementa il metodo SVD è contenuto nell'omonima operazione della classe Pseudoinverse ed essendo un algoritmo di tipo numerico ha bisogno, per mantenere una buona precisione nei valori calcolati, di utilizzare delle variabili di tipo *float*. Il codice contenuto nell'operazione **SVD()** è un adattamento di quello scaricabile dal sito Internet **www.crbond.com/linear.htm**.

Il primo test è relativo al calcolo di una matrice pseudo-inversa *3*2T* che, come detto all'inizio del paragrafo, serve al calcolo del vettore campo magnetico terrestre. La matrice *A1*

da pseudo-invertire ha dimensioni *2T*3* e viene costruita dall'operazione **Update()** utilizzando le prime due righe delle matrici $T_0 - T_4$. Vengono prese le prime due righe perché ogni PMT tile è equipaggiata con 2 magnetometri, uno utilizzato per misurare la componente *x* e l'altro per la componente *y*.

$$A_1 = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix}$$

Il risultato dell'operazione **Update()** è la matrice $A^+_{1\ 1B43}$ mentre la matrice $A^+_{1\ MATLAB}$ è stata calcolata con l'applicativo MATLAB.

$$A^+_{1\_1B43} = \begin{bmatrix} 0.00 & 0.00 & 0.00 & -0.33 & 0.33 & 0.00 & 0.00 & 0.00 & 0.00 & 0.33 \\ 0.00 & 0.33 & 0.00 & 0.00 & 0.00 & 0.33 & 0.00 & -0.33 & 0.00 & 0.00 \\ -0.25 & 0.00 & -0.25 & 0.00 & 0.00 & 0.00 & -0.25 & 0.00 & -0.25 & 0.00 \end{bmatrix}$$

$$A^+_{1\_MATLAB} = \begin{bmatrix} 0.00 & 0.00 & 0.00 & -0.33 & 0.33 & 0.00 & 0.00 & 0.00 & 0.00 & 0.33 \\ 0.00 & 0.33 & 0.00 & 0.00 & 0.00 & 0.33 & 0.00 & -0.33 & 0.00 & 0.00 \\ -0.25 & 0.00 & -0.25 & 0.00 & 0.00 & 0.00 & -0.25 & 0.00 & -0.25 & 0.00 \end{bmatrix}$$

Sempre utilizzando come possibile applicazione dell'algoritmo il calcolo del vettore campo magnetico, si è fatto un secondo test per verificare il funzionamento dell'algoritmo nel caso di malfunzionamento di una tile. Alla matrice A_2, rispetto ad A_1, sono state tolte le righe corrispondenti alla tile T_2, che viene considerata guasta.

$$A_2 = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix}$$

Anche in questo caso i risultati dell'elaborazione sono stati confrontati con i valori attesi calcolati con l'applicativo MATLAB.

$$A^+_{2\_1B43} = \begin{bmatrix} 0.00 & 0.00 & 0.00 & -0.50 & 0.00 & 0.00 & 0.00 & 0.50 \\ 0.00 & 0.50 & 0.00 & 0.00 & 0.00 & -0.50 & 0.00 & 0.00 \\ -0.25 & 0.00 & -0.25 & 0.00 & -0.25 & 0.00 & -0.25 & 0.00 \end{bmatrix}$$

$$A^{+}_{2\_MATLAB} = \begin{bmatrix} 0.00 & 0.00 & 0.00 & -0.50 & 0.00 & 0.00 & 0.00 & 0.50 \\ 0.00 & 0.50 & 0.00 & 0.00 & 0.00 & -0.50 & 0.00 & 0.00 \\ -0.25 & 0.00 & -0.25 & 0.00 & -0.25 & 0.00 & -0.25 & 0.00 \end{bmatrix}$$

Negli ultimi due test l'algoritmo verrà utilizzato per il calcolo del vettore di spin del satellite. La matrice A_3 ha dimensioni *T*3* ed è stata costruita utilizzando l'ultima riga delle matrici dei coseni direttori delle PMT tile $T_0 - T_4$. Si utilizza solo l'ultima riga perché ogni tile è equipaggiata con un giroscopio in grado di misurare la rotazione dell'asse *z*.

$$A_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

Le matrici $A^{+}_{3\ 1B43}$ e $A^{+}_{3\ MATLAB}$, mostrate in seguito, rappresentano rispettivamente i risultati dell'elaborazione ed i valori attesi.

$$A^{+}_{3\_1B43} = \begin{bmatrix} 0.50 & 0.00 & 0.00 & -0.50 & 0.00 \\ 0.00 & 0.50 & 0.00 & 0.00 & -0.50 \\ 0.00 & 0.00 & 1.00 & 0.00 & 0.00 \end{bmatrix}$$

$$A^{+}_{3\_MATLAB} = \begin{bmatrix} 0.50 & 0.00 & 0.00 & -0.50 & 0.00 \\ 0.00 & 0.50 & 0.00 & 0.00 & -0.50 \\ 0.00 & 0.00 & 1.00 & 0.00 & 0.00 \end{bmatrix}$$

L'ultimo test sull'operazione **Update()** è sempre relativo al calcolo del vettore di spin del satellite ma, analogamente al test 2, verrà simulato un guasto ad una tile. Anche in questo caso la PMT tile non funzionante sarà la T_2.

$$A_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix}$$

I risultati di questo test sono mostrati nelle seguenti matrici.

$$A^{+}_{3\_1B43} = \begin{bmatrix} 0.50 & 0.00 & -0.50 & 0.00 \\ 0.00 & 0.50 & 0.00 & -0.50 \\ 0.00 & 0.00 & 0.00 & 0.00 \end{bmatrix}$$

$$A^{+}_{3\_1B43} = \begin{bmatrix} 0.50 & 0.00 & -0.50 & 0.00 \\ 0.00 & 0.50 & 0.00 & -0.50 \\ 0.00 & 0.00 & 0.00 & 0.00 \end{bmatrix}$$

4. Simulazione del modulo 1B43 Housekeeping Management

Quest'ultimo test è servito per verificare il corretto funzionamento del protocollo *On Board Data Bus* (OBDB) e per effettuare una simulazione più realistica del modulo 1B43 Housekeeping Management.
Per questo test sono stati utilizzati due processori MSP430F5438A [9], uno in configurazione *slave* e l'altro in configurazione *master*. Il codice installato sul processore slave ha il compito di simulare una PMT tile, il cui vettore di housekeeping è stato definito costante in fase di compilazione, in modo da poter verificare facilmente se ci sono stati errori durante la trasmissione. Il processore master ha invece il compito di eseguire il codice del modulo 1B43 Housekeeping Management.
Il vettore di housekeeping di prova Hk_{TX} è composto da 39 elementi, per questo motivo è stato diviso in 4 parti per occupare meno spazio. Il significato di ogni elemento è descritto nella sezione relativa alla classe HK_fields_1B8.

$$Hk_{TX}[0-9] = \begin{bmatrix} 0x8000 \\ 0x8000 \\ 0x8000 \\ 0x8000 \\ 0x0001 \\ 0x0002 \\ 3 \\ 3 \\ 0x0003 \\ 0x0004 \end{bmatrix} \qquad \backslash Hk_{TX}[10-19] = \begin{bmatrix} 0x0005 \\ 33 \\ 31 \\ 0x0006 \\ 0x0007 \\ 0x0008 \\ 0x0009 \\ 120 \\ 0x0010 \\ 0x0011 \end{bmatrix}$$

$$Hk_{TX}[20-29] = \begin{bmatrix} 500 \\ 0x0011 \\ 0x0012 \\ -11732 \\ -16032 \\ 35 \\ 0x0013 \\ 0x0014 \\ 0x0015 \\ 0x0016 \end{bmatrix} \qquad Hk_{TX}[30-38] = \begin{bmatrix} 0x0017 \\ 0x0018 \\ 0x0019 \\ 0x0020 \\ 0x0021 \\ 0x0022 \\ 0x0023 \\ 0x0024 \\ 0x0025 \end{bmatrix}$$

L'avvio della comunicazione tra i processori è regolata dal TimerA ed avviene ogni minuto; una volta terminate la comunicazione e l'analisi dei dati, il master entra in modalità *risparmio energetico* (o *low power*) fino a quando non viene nuovamente "svegliato" dal timer.
Di seguito viene mostrato il vettore Hk_{RX} ricevuto dal master al termine della comunicazione con lo slave.

$$Hk_{RX}[0-9] = \begin{bmatrix} 0x8000 \\ 0x8000 \\ 0x8000 \\ 0x8000 \\ 0x0001 \\ 0x0002 \\ 3 \\ 3 \\ 0x0003 \\ 0x0004 \end{bmatrix} \qquad Hk_{RX}[10-19] = \begin{bmatrix} 0x0005 \\ 333 \\ 318 \\ 0x0006 \\ 0x0007 \\ 0x0008 \\ 0x0009 \\ 120 \\ 0x0010 \\ 0x0011 \end{bmatrix}$$

$$Hk_{RX}[20-29] = \begin{bmatrix} 500 \\ 0x0011 \\ 0x0012 \\ -11732 \\ -16032 \\ 335 \\ 0x0013 \\ 0x0014 \\ 0x0015 \\ 0x0016 \end{bmatrix} \qquad Hk_{RX}[30-38] = \begin{bmatrix} 0x0017 \\ 0x0018 \\ 0x0019 \\ 0x0020 \\ 0x0021 \\ 0x0022 \\ 0x0023 \\ 0x0024 \\ 0x0025 \end{bmatrix}$$

A questo punto è necessario verificare se le analisi fatte dal modulo sono corrette. Anche in questo caso si suppone che il satellite sia di forma cubica (come descritto nel capitolo 3) e che la tile 5 sia una TLC tile. La funzione che svolge le analisi dei dati viene chiamata dal timer subito dopo aver acquisito il vettore di housekeeping da tutte le tile del sistema. In questo caso, potendo simulare una sola tile, si considerano i dati del vettore Hk_{RX} appartenenti alla tile T_0, mentre per le altre PMT tile sono state create delle telemetrie ad-hoc. Nei vettori di housekeeping delle tile $T_1 - T_4$ tutti i campi non interessati dall'operazione **analyzePMT()** per le analisi sono stati fissati al valore *0x0001*, mentre per tutti gli altri campi sono stati utilizzati i valori scritti nella tabella seguente.

Campo	**Tile 1**	**Tile 2**	**Tile 3**	**Tile 4**	
Hk[0] : EARTH_SENSOR_X	0x8000	0x8000	0x8000	0x8000	10^3 rad
Hk[1] : EARTH_SENSOR_Y	0x8000	0x8000	0x8000	0x8000	10^3 rad
Hk[2] : SUN_SENSOR_X	6938	-8672	-8760	0x8000	10^3 rad
Hk[3] : SUN_SENSOR_Y	4033	-6430	4458	0x8000	10^3 rad
Hk[6] : SOLAR_TEMPERATURE_CENTER	500	500	500	3	K
Hk[7] : SOLAR_TEMPERATURE_SIDE	500	500	500	3	K
Hk[11] : BATTERY_TEMPERATURE	348	353	345	331	K
Hk[12] : POWERCIRCUITS_TEMPERATURE	343	341	347	323	K
Hk[17] : INTERNAL_BUS_VOLTAGE	110	117	115	122	V
Hk[20] : SPIN_Z	0	1000	-500	0	10^3 rad/s

Hk[23] : MAGNETIC_FIELD_X	16032	-11732	-11732	-16032	10 nT
Hk[24] : MAGNETIC_FIELD_Y	4670	-4670	16032	4670	10 nT
Hk[25] : ACS_TEMPERATURE	352	348	350	339	K

L'analisi fornisce una serie di valori, descritti nel caso d'uso *Analyze 1B8* e rappresentati sotto forma di attributi nella classe Result_1B8. I valori calcolati sono:

- il vettore campo magnetico terrestre;
- lo spin del satellite;
- la posizione del satellite rispetto al Sole;
- la posizione del satellite rispetto al centro della Terra;
- la media aritmetica dei campi:
    - SOLAR_TEMPERATURE_CENTER;
    - SOLAR_TEMPERATURE_SIDE;
    - BATTERY_TEMPERATURE;
    - POWERCIRCUITS_TEMPERATURE;
    - INTERNAL_BUS_VOLTAGE;
    - ACS_TEMPERATURE;
- la deviazione standard del campo INTERNAL_BUS_VOLTAGE;
- i valori minimo e massimo del campo INTERNAL_BUS_VOLTAGE.

Ognuno di questi valori viene memorizzato in uno specifico attributo della classe Result_1B8. Per ognuno dei risultati dell'analisi è stato calcolato il valore atteso tramite il linguaggio MATLAB; nelle seguenti tabelle verranno messi a confronto i valori calcolati dall'algoritmo e quelli attesi per verificare il corretto funzionamento del software.
Nella tabella seguente le colonne indicano rispettivamente l'attributo della classe Result_1B8 al quale i valori sono associati, i valori calcolati e quelli attesi.

Result_1B8	**1B43**	**MATLAB**	
Buvw[0]	-3910	-3911	10 nT
Buvw[1]	-12244	-12245	10 nT
Buvw[2]	5866	5866	10 nT
Spin[0]	500	500	10^3 rad/s
Spin[1]	0	0	10^3 rad/s
Spin[2]	1000	1000	10^3 rad/s
SunPos[0]	0x8000	-4235	10^3 rad
SunPos[1]	0x8000	-3629	10^3 rad
SunPos[2]	0x8000	456	10^3 rad
EarthPos[0]	0x8000	0x8000	10^3 rad
EarthPos[1]	0x8000	0x8000	10^3 rad
EarthPos[2]	0x8000	0x8000	10^3 rad
IBVmean	116	116,80	V
IBVdev	4	4,6583	
IBVmin[0]	110	110	V
IBVmin[1]	1	1	-
IBVmax[0]	122	122	V
IBVmax[1]	4	4	-
STCmean	301	301	K
STSmean	301	301	K
BTmean	342	342	K
PTmean	334	334,4	K
ACSTmean	344	344,8	K

Come si può vedere dalla tabella, gli errori riscontrati sono per la maggior parte dei casi di piccola entità, per cui si ritengono le misure tutto sommato corrette. Per quanto riguarda le misurazioni sui sensori di Sole sono state riscontrate maggiori difficoltà che purtroppo non si è riusciti a risolvere.

Capitolo 6

Conclusioni

La progettazione del modulo 1B43 Housekeeping Management, utilizzato per gestire i dati di telemetria delle tile di un satellite AraMiS, ha consentito di utilizzare le conoscenze acquisite durante il periodo trascorso al Politecnico di Torino.
Grazie a questo progetto il candidato è entrato in possesso di nuove conoscenze, in particolare riguardo alla applicazioni embedded e alle problematiche legate all'ambito aerospaziale; inoltre è stato possibile rafforzare le proprie conoscenze dei linguaggi UML e C++.
Nella prima fase del progetto sono state riscontrate delle difficoltà, dovute maggiormente al fatto di entrare a far parte di un progetto della grandezza e complessità di AraMiS. È stato necessario spendere alcune settimane per comprendere il funzionamento di tutti i moduli con cui il progetto 1B43 Housekeeping Management interagisce per poter iniziare seriamente lo sviluppo.
In fase di progettazione si è scelto di modellare il sistema utilizzando da subito il linguaggio UML. L'utilizzo dell'UML ha permesso di realizzare il modello del sistema facilmente, rendendo più agevole la scrittura del codice sorgente e la modifica della stessa struttura del modulo. Le operazioni contenute nella classe Pseudoinverse sono quelle che hanno presentato le maggiori difficoltà per via della loro base teorica "esotica", come appunto le matrici pseudo-inverse e le frazioni continue, che erano concetti sconosciuti al candidato prima di iniziare lo sviluppo di questo progetto.
Nonostante il progetto sia completamente software, c'è stato modo di "scendere di livello" fino alla parte hardware, prima scrivendo il codice sorgente di alcune classi del modulo 1B45 Subsystem Serial Data Bus per una nuova famiglia di microprocessori e poi saldando i connettori sulla socket board del kit di sviluppo del processore MSP430F5438A.
Infine, tutti i test effettuati ha fornito esito positivo, anche se resta da verificare la parte relativa alle TLC tile.
Il modulo 1B43 Housekeeping Management ha comunque dei margini per poter essere migliorato, soprattutto per quanto riguarda la protezione dei dati. In particolare le matrici pseudo-inverse, essendo oggetti che si presume di non ricalcolare per lunghi periodi, andrebbero protette dalle radiazioni, ad esempio salvandole nella memoria flash del processore oppure tramite duplicazione o triplicazione dei dati.

Appendice A

Acronimi

AraMiS *Architettura Modulare per Satelliti*
AraMiS è il secondo nano-satellite sviluppato dal Politecnico di Torino. Il progetto è iniziato nell'autunno del 2006, dopo il lancio di PiCPoT, ed è ancora in corso. AraMiS è un satellite modulare, ovvero è possibile adattare la struttura del satellite alla missione assemblando e configurando in modo opportuno dei moduli standard.

COTS *Commercial Off-The-Shelf*
Componentistica elettronica commerciale a basso costo.

LEO *Low Earth Orbit*
Orbite con altitudine compresa tra 200km e 2000 km. Un corpo che orbita in una LEO compie una rivoluzione in circa 90 minuti, a una velocità di 27400 km/h.

OBC *On Board Computer*
È il computer di bordo del satellite AraMiS ed è posizionato sulla Telecomunication Tile.

OBDB *On Board Data Bus*
È il bus di comunicazione utilizzato dalle tile del satellite AraMiS.

PiCPoT *Piccolo Cubo del Politecnico di Torino*
PiCPoT è il primo nano-satellite sviluppato dal Politecnico di Torino tra il 2004 e il 2006. Lo scopo del progetto PiCPoT era di sviluppare e realizzare un nano-satellite per l'acquisizione di immagini della superficie terrestre e per verificare l'affidabilità di componenti COTS in ambiente spaziale.

PMT *1B8 Power Management Tile*
Modulo standard del satellite AraMiS. Gestisce le alimentazioni del satellite ed il suo assetto.

SEU *Single Event Upset*
Un single event upset è un cambio di stato logico causato dall'impatto di una particella ad alta energia in un nodo sensibile con un dispositivo micro-elettronico, come un microprocessore, una memoria o un transistor di potenza.

SVD *Singular Value Decomposition*
La Singular Value Decomposition (o Decomposizione ai Valori Singolari) è un tipo di fattorizzazione di una matrice reale o complessa, basata sull'utilizzo di autovalori e autovettori.

TID *Total Ionizing Dose*
La TID (o dose assorbita) è una grandezza fisica che misura la quantità di energia per unità di massa assorbita da un mezzo a seguito di esposizione a radiazioni ionizzanti. La sua unità di misura nel Sistema Internazionale è il

Gray (Gy), che ha sostituito il *Radiation Absorbed Dose* (rad); 1 Gy = 100 rad.

TLC *1B9 Telecomunication Tile*
Modulo standard del satellite AraMiS. Permette le comunicazioni da/verso Terra tramite due canali RF.

UML *Unified Modelling Language*
È un linguaggio standard di modellazione nell'ambito dell'ingegneria del software object-oriented, ma può essere utilizzato in altri campi. L'UML permette di descrivere a livello strutturale e comportamentale un sistema per mezzo di grafici e diagrammi, rendendo la comprensione e l'analisi più intuitive.

Bibliografia

[1] *AraMiS Website*. url: **http://polimage.polito.it/aramis**.
[2] *Baikonur Cosmodrome*. url: **http://www.russianspaceweb.com/baikonur.html**.
[3] *Wikipedia: Baikonur Cosmodrome*. url: **http://en.wikipedia.org/wiki/Baikonur_Cosmodrome.**
[4] *CubeSat Community Website*. url: **http://cubesat.calpoly.edu**.
[5] J. A. Schaffner. "The Electronic System Design, Analysis, Integration, and Construction of the Cal Poly State University CP1 CubeSat". In: *16th AIAA/USU Conference on Small Satellites* (2002), pp. 1–12.
[6] *Wikipedia: Continued Fraction*. url: **http://en.wikipedia.org/wiki/Continued_fraction**.
[7] *Wikipedia: Dnepr-1 rocket*. url: **http://en.wikipedia.org/wiki/Dnepr-1**.
[8] *Datasheet MSP430F2418*. url: **http://focus.ti.com/lit/ds/symlink/msp430f2418.pdf**.
[9] *Datasheet MSP430F5438A*. url: **http://focus.ti.com/lit/ds/symlink/msp430f5438A.pdf**.
[10] *Application Note MSP430x24x*. url: **http://focus.ti.com/lit/ug/slau144g/slau144g.pdf**.
[11] *Application Note MSP430x43x*. url: **http://focus.ti.com/lit/ug/slau056j/slau056j.pdf**.
[12] *Application Note MSP430x54x*. url: **http://focus.ti.com/lit/ug/slau208h/slau208h.pdf**.
[13] *Neohm Componenti*. url: **http://www.neohmcomponenti.com/**.
[14] *PiCPoT Website*. url: **http://polimage.polito.it/picpot**.
[15] *Wikipedia: Moore-Penrose Pseudoinverse*. url: **http://en.wikipedia.org/wiki/Moore-Penrose_pseudoinverse**.
[16] *Sky Technology*. url: **http://www.skytechnology.it/**.
[17] *Spin Electronics*. url: **http://www.spinelectronics.eu/**.
[18] Manuel Bonjean. "Progetto AraMiS: Progetto del software indirizzato alla determinazione ed al controllo dell'assetto del satellite". Tesi di Laurea Magistrale. Politecnico di Torino, 2010.
[19] Object Management Group. *Unified Modelling Language Superstructure*. 2010. url: **http://www.omg.org/spec/UML/2.3/Superstructure/PDF**.
[20] Kurt Bittner & Ian Spence. *Use Case Modeling*. Addison-Wesley Professional, 2003.