博士論文

再利用性と軌道上再構成能力に優れた
衛星ソフトウエア・アーキテクチャに関する研究

東京大学 大学院 工学系研究科 航空宇宙工学専攻
滝澤潤一 (学籍番号 37-127059)

指導教官: 中須賀真一教授

2014 年 12 月 1 日 提出

論文要旨

本論文は衛星搭載ソフトウェアの開発手法に関するものであり，衛星搭載ソフトウェアに高い再利用性とこれまで存在していなかった柔軟な軌道上再構成能力を付加する体系的な枠組みを構築し，その適用と評価を行っている．

近年，大きさ 50cm 立方以下，質量 50kg 以下の超小型衛星の開発はますます活発化しており，国内・国外を問わず様々な研究機関・企業が開発・運用に参入している．超小型衛星が初めて打ち上がった 2003 年からこれまで，超小型衛星といえば技術実証や教育を目的とするものが中心であったが，近年では地球観測や宇宙科学といった「実用」に主眼を置く超小型衛星が主流となってきている．

従来の大型衛星に対する超小型衛星の利点は「短期間」かつ「低コスト」でミッションを実現できることにあり，この利点をいかに実現するかが非常に重要である．近年主流となりつつある実用超小型衛星では従来の超小型衛星と比較して，ミッションの高度化に伴う「高性能化」と「運用の長期化」が強く求められており，これらの実現には衛星の信頼性を従来にも増して高めることが必要となる．一般に短期・低コスト化と高性能化・運用の長期化は相反する傾向にあり，実用超小型衛星ではこれらをいかに両立するかが問題となっている．

この問題の解決には「高い再利用性」と「柔軟な軌道上再構成能力」の 2 つが重要となる．まず，過去の超小型衛星で動作が確認された実績品を繰り返し使用する再利用は様々な観点で信頼性を向上させ，開発の短期・低コスト化を実現する．また，打ち上げ後の軌道上で衛星の機能を柔軟に再構成することが可能であれば軌道上で発生し得る地上で予期しなかった異常事態への対応能力が向上するだけでなく，従来地上で実施してきた試験の一部を軌道上で実施することも可能となり短期・低コストでの開発が実現する．

これまでの超小型衛星の搭載ソフトウェアは衛星毎の個別開発が基本であり，再利用性が高いとはいえず，また軌道上再構成に関してはこれを活用しようとする研究はあるものの，具体的な実現方法については議論が行われていない．搭載ソフトウェアの再利用と軌道上再構成を実現するためには闇雲に開発を行う従来の方法ではなく，まず明確な枠組みを構築した上で，その枠組に基いて個々の衛星の搭載ソフトウェアを開発していく環境の整備が必要である．

本研究ではまず，過去に開発された超小型衛星の搭載ソフトウェアの調査・整理を実施し，衛星の動きを実現するために搭載ソフトウェアに必要な「コマンド処理機能」「モード管理機能」「イベント処理機能」の 3 機能を「必須機能」と定義する．次に，それらが全て「予め用意された処理

の塊を展開し，決まったタイミングで実行する」という同一のパターンを共有していることを見出し，これに着目して全ての機能をコマンド処理機能，その中でも特に「ブロックコマンド機能」に集約することで，見通しが良く，高い再利用性と軌道上再構成能力を備えた搭載ソフトウェア・アーキテクチャ「Command Centric Architecture (C2A)」を提案している．

従来，打ち上げ後に実施できる衛星の再構成は制御ゲインの変更のような予め用意されたパラメータの変更か，これで対処できない場合は搭載計算機のメモリ内容を直接書き換えるプログラム書き換えかの極端な 2 択しか存在しなかった．プログラム書き換えは非常に高い対処能力を備える反面，その内容に誤りがあった場合は衛星システムの全損につながる可能性も高く，事前に周到な準備が必要な実施負荷の高い方式であり，容易に選択できるものではなかった．これに対して C2A では，前述の通り衛星搭載ソフトウェアの必須機能をブロックコマンド機能に集約して実現することで，モード遷移の変更や定義の追加，イベントへの対応内容の変更など，状況に応じて様々なレベルの再構成を地上での開発段階と同様の手順・粒度で選択可能な柔軟な軌道上再構成能力を実現している．

C2A では個々の衛星で独自に必要となる機能をアプリケーションと呼び，C2A の必須機能がこれらアプリケーションを呼び出すためのインターフェースを規定している．それぞれの機能をアプリケーションに分割して実装することで，C2A を採用した異なる衛星間で同様の機能が必要となった場合の再利用性が高まると同時に，明確なインターフェースを規定したことで，搭載ソフトウェアにアプリケーションを実装する際の手順が共通化され，見通しの良い開発環境が実現している．加えて，実装の手順が共通化された部分についてはソフトェアの自動生成についても取り組んでいる．

本研究では構築した C2A を実際に適用して搭載ソフトウェアの開発を行い，実用超小型衛星である「UNIFORM-1」「ほどよし 3 号機」「ほどよし 4 号機」に搭載した．UNIFORM-1 は 2014 年 5 月 24 日，ほどよし 3 号機・4 号機は 2014 年 6 月 19 日にそれぞれ打ち上げられ，現在も運用中である．運用の過程ではいくつかのトラブルが生じたが，C2A が提供する軌道上再構成能力を活用することで柔軟に対処し克服している．本研究では，上記 3 衛星に加え 2014 年 12 月 3 日に打ち上げられた超小型深宇宙探査機「PROCYON」にも C2A を適用して搭載ソフトウェアの開発・運用を実施している．PROCYON の開発期間はほぼ 1 年であり，これまでの超小型衛星と比べても短期間で開発が完了した宇宙機である．搭載ソフトウェアに関しては C2A の効果を最大限活用することで 5.5 ヶ月での開発を実現している．本論文では，これら実際の開発・運用結果を元に C2A のもたらす利点について評価を行い，その有効性を確認している．

以上の成果は，これまで明確な開発指針が存在しなかった超小型衛星の搭載ソフトウェアに体系的な枠組みを与え，高い再利用性と柔軟な軌道上再構成能力を実現するものであり，今後ますます発展が見込まれる超小型衛星の開発・運用に大きく貢献すると期待される．

目次

図目次

表目次

第 1 章

序論

1.1　超小型衛星

近年，超小型衛星を取り巻く状況ははますます活発化しており，国内・国外を問わず様々な研究機関・企業が開発・運用に参入している．

超小型衛星は「短期・低コスト」で開発できることが最大の特徴であり，この特徴を活かして従来の大型衛星に比べ，より高頻度により挑戦的なミッションを実現することを目指している．

これまで超小型衛星といえば技術実証や教育を目的としたものが主流であったが，2003 年に世界初の超小型衛星の打ち上げが成功して以降 10 年以上が経過し，近年では地球観測や宇宙科学といった実用を目的とした超小型衛星の開発が増加してきている．

実用超小型衛星では従来の技術実証や教育を目的とした超小型衛星と比較して，ミッション要求の高度化と運用の長期化が求められる傾向にある．従来の技術実証型の超小型衛星は新規技術の軌道上実証を主目的とするため，打ち上げ後数週間から数ヶ月程度でそのミッションが完了するのに対し，衛星の継続利用を前提とする実用型の超小型衛星はミッション完了まで通常数年単位の長期にわたる継続運用が求められる．これを実現するには超小型衛星の「機能性・信頼性」を従来にも増して高めることが必要となって来ている．

一方で，最初に述べたとおり超小型衛星は短期・低コストでの開発を実現することが従来の大型衛星に対する最大の特徴であり，この特徴を維持したままで要求される高い信頼性をいかに実現するかが課題となっている．

1.2　再利用性と軌道上再構成能力

短期・低コストでの開発と高い機能性・信頼性という一見相反する要求の両立には

- 高い再利用性
- 柔軟な軌道上再構成能力

の 2 つが重要となる．

1.2.1　再利用性

衛星を構成する各要素の再利用性を高め，積極的に再利用することは大型・超小型といった衛星の種類に関係なく短期・低コストの開発と高い機能性・信頼性を両立させる上で重要な役割を果たす．

既に以前に開発・運用した衛星で動作が確認されたいわゆる実績品は新規の衛星においても以前と同様に動作することが高い確度で期待できるので，これを再利用することは単純に衛星の信頼性向上につながると考えられる．同時に，実績品を再利用することは衛星の中の新規要素の数を削減し，開発や試験の省略・簡素化につながるため，信頼性向上の面だけでなく衛星全体のコストと開発期間の圧縮に大きく寄与することとなる．

また，たとえ以前の衛星で問題があった要素であっても，その原因を特定し対策を行ったうえで再利用していくというプロセスを回すことができれば，その要素の信頼性を継続的に向上させることが可能であり，これも衛星の信頼性向上に寄与すると考えられる．

1.2.2　軌道上再構成能力

再利用性を高めることに加えて，短期・低コスト開発と高い信頼性を両立させる上で重要となるもう一つの柱が柔軟な軌道上再構成能力である．

これまで，衛星の搭載ソフトウェアは打ち上げ前の段階でその内容を固定し，試験・検証を十分に行った上で打ち上げ，運用に臨むのが通常であった．打ち上げ後に搭載ソフトウェアを変更することは基本的に想定されておらず，万一これを実施するとなれば搭載計算機のメモリ内容に直接パッチを適用すると言った機械語レベルでの対処が必要となった．

これに対し，本論文で提唱する柔軟な軌道上再構成能力とは，衛星の搭載ソフトウェア自体にその動作を打ち上げ後も変更できる仕組みを積極的に用意することで，状況に応じた柔軟な運用を実現しようとする考え方である．

柔軟な軌道上再構成能力を衛星に付与することで，打ち上げ後に生じ得る不測の事態に柔軟に対処できるだけでなく，これまで地上で実施してきた試験の一部を軌道上で実施することで地上での開発期間を短縮するといったような衛星の不具合対応だけにとどまらない幅広い効果が期待できる．

1.2.3　両者のもたらす利点

高い再利用性と柔軟な再構成能力を実現することで超小型衛星には

- 打ち上げ時期への柔軟な対応が可能となる
- 新規性が高く野心的なミッションへの積極的な挑戦が可能となる

といった利点が生じる．

まず，衛星の構成要素の大部分を再利用によって構築することができ，さらに搭載ソフトウェアの大部分を再構成能力によって軌道上でも調整可能となれば，地上での開発期間を大幅に短縮し，衛星をハードウェア開発完了後の早い段階で打ち上げ可能な状態とすることができる．

大型衛星を中心とする従来の衛星開発の基本方針は，不測の事態が生じない設計を行い，全ての機能を事前に地上で確認することである．これを実現するために，費用・期間を投じてでも十分な試験環境・試験期間を確保して試験・検証が行われる．超小型衛星においても無論この方針が重要であることは変わらない．しかし，打ち上げ機会の大部分を超小型衛星自体の開発期間とは無関係な大型衛星との相乗りに頼る現状では，超小型衛星は，たとえ大型衛星のスケジュールによって短期の開発期間しか確保できない場合であっても，打ち上げ機会の確保を優先しなければならない状況にあることを考慮すると，開発期間の短期化によって大型衛星の打ち上げスケジュールに適合する可能性を高めることは非常に重要で大きな意義を持つ．

また，打ち上げ後に軌道上で生じる様々な状況を搭載ソフトウェアの柔軟な軌道上再構成能力によって対処できるのであれば，ある程度は失敗のリスクを恐れずにミッションに挑戦することが可能となる．さらに，より挑戦的なミッションとして，深宇宙探査や自律判断のような事前に状況・結果を完全に予測することが困難・不可能なミッションの実施も考えられる．これは短期・低コストという特徴を活かして，高頻度に新しいミッションや技術実証に挑戦するという超小型衛星の特性と親和性が非常に高い．

新規性の高いミッションは同時に新規開発要素の比重が高いことを意味しており，ハードウェアの開発段階で不具合が発生してスケジュールを圧迫する可能性も高く，搭載ソフトウェアまで含めた十分な検証期間が打ち上げまでに確保できない場合も想定される．このような事態に対しても本研究で提唱する軌道上での再構成能力が有効に機能すると予想される．

このように，高い再利用性と柔軟な軌道上再構成能力は超小型衛星の可能性を大きく広げると考えられる．

1.3　本研究の目的

本論文では「高い再利用性」と「柔軟な再構成能力」の 2 つを衛星の搭載ソフトウェアとしていかに実現するかが議論の中心となる．これには以下のような過程を踏む．

まず，過去の衛星の搭載ソフトウェアの調査を行って衛星の搭載ソフトウェアの必須機能を洗い出し，それらを整理・考察することで，柔軟な再構成能力を備えた搭載ソフトェアの枠組みを導出する．本論文ではこの枠組を Command Centric Architecture (C2A) と呼ぶ．

次に，導出した C2A を実際の搭載ソフトウェアとして実装する．この過程では搭載ソフトウェアの高い再利用性を実現するために考慮すべき点について議論を行う．

最後に，導出した C2A を用いて開発した衛星搭載ソフトウェアを複数機の超小型衛星に実際に搭載し，その開発・運用を通じて得られた結果を元にその評価を行う．

1.4 論文構成

本論文は合計 8 章からなり，その構成は以下の通りである．

まず，第 2 章では本論文の主題となる搭載ソフトウェアの再利用性と軌道上での再構成能力について議論を深め，本研究で克服すべき技術的な課題を整理する．

第 3 章では過去の衛星搭載ソフトウェアの調査を通じてその必須機能を整理し，C2A という体系的で柔軟な衛星搭載ソフトウェア・アーキテクチャを導出する．続く第 4 章では，第 3 章で導出した C2A を実際の搭載ソフトウェアとして実装するにあたって必要となる機能とそのパラメータについて整理し，第 5 章では本研究で提案する C2A を用いて実際に衛星の搭載ソフトウェアを開発する場合に考慮すべき点について述べる．

第 6 章では本研究で提案する C2A を実際の衛星搭載ソフトウェアの開発に利用することによって衛星搭載ソフトウェアの再利用性と再構成能力の強化以外に生じる副次的な効果について議論する．

第 7 章では提案した C2A を実際に複数の超小型衛星に適用した結果を評価し，最後に第 8 章で本論文の内容をまとめ，結論と今後について述べる．

論文中では提案する搭載ソフトウェアの枠組みについて，概念的な議論を中心に行うが，より詳細な実装に近い部分に関しては付録の形で巻末に掲載することとする．

第 2 章

超小型衛星の搭載ソフトウェア開発

2.1　超小型衛星の特徴

近年開発が活発化している超小型衛星は従来の大型衛星が抱える問題点に対する一つの解決策として出現した衛星である．本節ではまず大型衛星の問題点を整理し，超小型衛星がどのような特徴を活かしてその解決を目指しているかをまとめる．

これまでに開発された人工衛星はそのミッション要求が次第に複雑化・高度化する過程で大型化の道を歩んできた．人工衛星の大型化は複雑・高度なミッションを達成してきた反面，その開発コストの増大と開発期間の長期化が問題として指摘されている [1)2)]．

近年の大型衛星の開発コストは 1 機あたり数百億円，開発期間は 5 年以上となっており，失敗した場合の損失・リスクもそれに伴って増大している．このような環境下では，実績のある枯れた技術を用い，搭載機器の冗長化等を行って信頼性を確保する非常に保守的な設計・開発方針を採用せざるを得ない．機器の冗長化は信頼性を高める上で有効であるもののシステム全体の複雑化を招き，それに伴う新たなミスの混入や開発期間の長期化といった負の連鎖を生じ得る点が問題である．また，枯れた技術のみを採用する方針では，新たな機器・技術の実証機会が奪われることとなり，衛星全体の進歩の硬直化につながる恐れもある．

このような大型衛星の問題点に対し，近年の電子部品の集積技術等を積極的に活用して衛星を重量 50kg 程度以下，大きさ 50cm 立方程度以下に小型化すると同時に開発コストと信頼度の関係を慎重に見極め，適切な信頼性を維持しつつもコストの増大を回避することで，1 機あたりの開発コストを数億円程度，開発期間を 2〜3 年に抑えた超小型衛星が登場してきている [3)]．

短期・低コストで開発できる超小型衛星はその特徴により，ある程度は失敗のリスクを恐れず高頻度に新しいミッションや技術実証に挑戦できるため，これまで多くの大学や研究機関がその開発に参加しており，技術実証や教育を中心に継続的な技術革新や利用拡大が期待されている．

2.2　近年の超小型衛星の動向

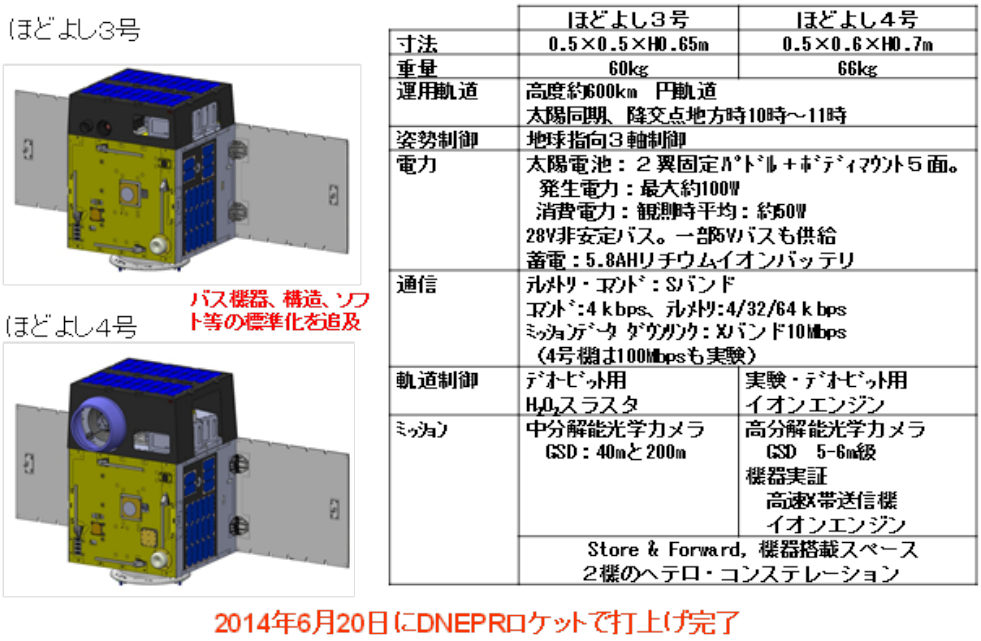
前節のような背景で発達してきた超小型衛星であるが，近年では技術実証や教育を目的とするだけでなく，より実用的なミッションを実現しようという動きが活発化している．

この具体的な例として図 2.1 に筆者の所属する東京大学中須賀・船瀬研究室（以下「当研究室」と記す）がこれまでに開発・運用してきた超小型衛星とその種類をまとめる．

図 2.1　超小型衛星の歴史 (当研究室の例)

図 2.1 に示すように，当研究室の超小型衛星は最も初期の実現可能性の実証衛星「XI-IV，XI-V」[4]，それに続く技術実証衛星「PRISM」[5] を経て実用段階である理学観測衛星「Nano-JASMINE」[6] へとそのミッションを発展してきた．

また，実用超小型衛星に焦点を当てると，国内では 2010 年度から 2013 年度まで，内閣府の最先端研究開発支援プログラムの一環として，「日本発の「ほどよし信頼性工学」を導入した超小型衛星による新しい宇宙開発・利用パラダイムの構築」[3] (以下「ほどよしプロジェクト」と記す) として，4 機の超小型衛星の開発が行われた．そのうちの 2 機にあたる「ほどよし 3・4 号機」は図 2.2 に示すような実用ミッションを主体とした超小型衛星であり，2014 年に打ち上げられ，現在初期運用が進められている．



	ほどよし３号	ほどよし４号
寸法	0.5×0.5×H0.65m	0.5×0.6×H0.7m
重量	60kg	66kg
運用軌道	高度約600km　円軌道 太陽同期、降交点地方時10時～11時	
姿勢制御	地球指向３軸制御	
電力	太陽電池：２翼固定ﾊﾟﾄﾞﾙ＋ﾎﾞﾃﾞｨﾏｳﾝﾄ５面。 発生電力：最大約100W 消費電力：観測時平均：約50W 28V非安定バス。一部5Vバスも供給 蓄電：5.8AHリチウムイオンバッテリ	
通信	ﾃﾚﾒﾄﾘ・ｺﾏﾝﾄﾞ：Sバンド ｺﾏﾝﾄﾞ:4kbps、ﾃﾚﾒﾄﾘ:4/32/64kbps ﾐｯｼｮﾝﾃﾞｰﾀ ﾀﾞｳﾝﾘﾝｸ：Xバンド10Mbps （4号機は100Mbpsも実験）	
軌道制御	ﾃﾞｵｰﾋﾞｯﾄ用 H_2O_2スラスタ	実験・ﾃﾞｵｰﾋﾞｯﾄ用 イオンエンジン
ミッション	中分解能光学カメラ GSD：40mと200m	高分解能光学カメラ GSD　5-6m級 機器実証 高速X帯送信機 イオンエンジン
	Store & Forward，機器搭載スペース ２機のヘテロ・コンステレーション	

図 2.2　ほどよし 3・4 号機 諸元

2.3　超小型衛星の抱える問題点

前節の通り活発な開発が続く超小型衛星であるが，いくつか問題点も抱えている．本節ではこれらについて整理する．

2.3.1　技術・知見の蓄積・共有の不完全さ

近年開発が盛んな超小型衛星であるが，現状は個別衛星の開発・運用のみに終始しており，個々の衛星を超えた技術・知見の蓄積・共有は不完全な状況にある．過去の超小型衛星の運用実績を調査すると，計画通りにミッションを達成した衛星が存在する一方で，ミッションを全く達成できなかった衛星も存在しており，両者の差が大きく開いている．また，超小型衛星の打ち上げ機数は年々増加する傾向にあるが，そのミッションの成功率は打ち上げ機数の増加にとともに改善はしておらず，一定の割合にとどまっている事も過去の超小型衛星の実績調査から指摘されている [7)8)]．

このような状況が今後も続けば，超小型衛星の利点や有益性に疑問が生じることは避けられず，問題となっている．

2.3.2　高性能化と運用の長期化

超小型衛星の開発が活発化するに従い衛星の高性能化と運用期間の長期化が求められている点も問題である．

超小型衛星の高性能化は実現するミッションの高度化に伴って生じている．超小型衛星の有用性が認識されるにつれミッションの内容も高度化が進んでおり，例えば地球観測では，より高解像度の画像が要求される傾向にある．この実現には高性能のミッション機器を開発・搭載するだけでは不十分で，衛星自体の姿勢の安定度や指向精度を向上させる必要があり，その他にも撮影した大量のデータを処理し地上へ高速に伝送する技術など衛星の基本機能全体の性能を高める必要が生じている．

運用期間の長期化は特に実用を目指す実用超小型衛星に関して顕著である．これは，目的とする技術が実証できた時点でミッション完了となる従来の技術実証超小型衛星と異なり，実用超小型衛星では衛星を継続的に利用することが重要となるためである．例えば地球観測を目的とする実用超小型衛星であれば，軌道上で観測機器の性能を確認するまでは準備期間であり，確認完了後にその機器を活用して継続的に各地点を観測することが当然求められる．

具体的な例として，図 2.3 に当研究室が打ち上げた技術実証型の超小型衛星 PRISM と実用超小型衛星であるほどよし 3, 4 号機の姿勢制御性能と運用期間をまとめて示す．

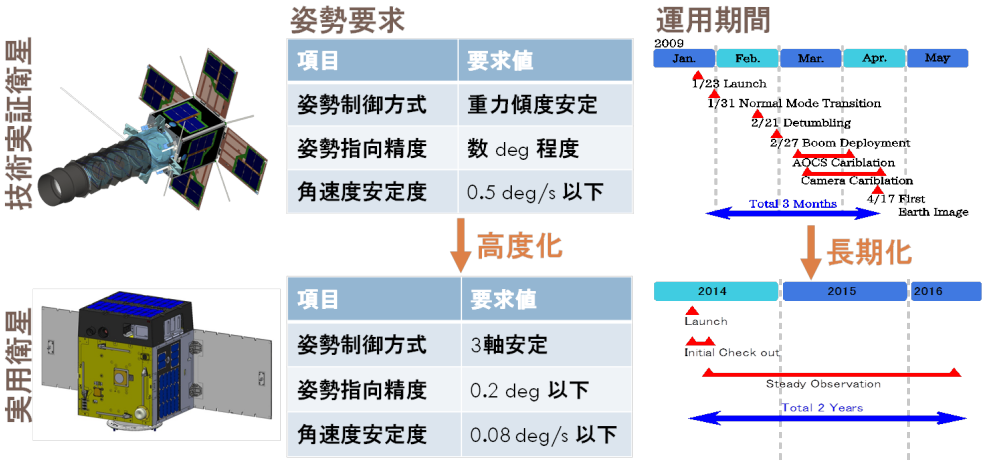


図 2.3　技術実証超小型衛星と実用超小型衛星の比較

ミッション達成に必要となる姿勢制御性能に関しては，PRISM と比較して高い解像度での地球観測を目的とするほどよし 3, 4 号機の方がより高度な姿勢制御方式を採用し，その指向精度と角速度安定度に関してもおおよそ一桁高い水準が求められていることが読み取れる．

運用期間については技術実証衛星である PRISM は打ち上げ後約 3 か月が経過した撮影完了の時点でミッション達成となったのに対し，ほどよし 3, 4 号機では打ち上げ後約 2 ヶ月の初期チェックアウト期間で観測性能を確認した後，ミッション完了まで少なくとも 2 年の継続運用が求められ

ている．

高性能化や運用の長期化は超小型衛星の利用を広げる観点で当然実現すべき項目であるが，現状ではこれに伴って開発や地上試験が複雑・長期化し，本来の超小型衛星の特徴である短期・低コスト性が失われる傾向にあり，問題である．

2.3.3 開発期間の超短期化

現在，各国で主力として用いられているロケットは大型衛星の打ち上げを念頭に開発されており，超小型衛星単体の打ち上げにはその能力や価格の観点で適さない構成となっている．このため，多くの超小型衛星ではその打ち上げ機会を大型衛星打ち上げ時のロケット余剰能力を活用する相乗りに求める場合が多い．

相乗りによる打ち上げは単独での打ち上げに比べ安価にその機会が得られる反面，あくまで打ち上げの目的は主衛星である大型衛星の打ち上げであるため，そのスケジュールは主衛星に左右される事となる．

また，相乗り打ち上げの機会はそもそもその頻度が少ない一方で，その公募は超小型衛星の開発期間とは無関係に行われる場合が多く，超小型衛星が打ち上げ機会を獲得するためには，短期の開発期間しか確保できない場合であっても打ち上げ機会の確保を優先しなければならない状況にある．

具体的な例として，図 2.4 に，これまで H-IIA ロケットの打ち上げ時に相乗り衛星として打ち上げられた超小型衛星の公募スケジュール [9)10)11)12)13)14)15)] をまとめて示す．図は横軸に公募開始時点で予告された打ち上げ期間の開始位置を起点とする日数をとり，それぞれの相乗り機会毎の公募の開始日，搭載の決定日，実際の打ち上げ日を示している．

実際の打ち上げ日ではなく，公募開始時点での予告打ち上げ期間の開始位置を起点としている理由は公募への応募者は応募時点では予告された打ち上げ期日以前に超小型衛星の開発を完了し，搭載側に引き渡しが可能なスケジュールを提示する必要があるためである．

図 2.4 より，相乗りの公募開始時点で，予告された打ち上げ期間の開始位置までの期間は 2 年から 1 年の間に分布していることが読み取れる．公募に応募するためには，超小型衛星をこの期間で開発できる現実的なスケジュールを提示する必要があるが，これは前述した超小型衛星の目指す開発期間 2〜3 年より短い．

実際の開発期間は応募が採択され搭載が決定した時点から実際の打ち上げ日までの期間になるが，これに関しても長くて 2 年程度，本論文で扱う超小型深宇宙探査機 PROCYON を含む「はやぶさ 2」の相乗りのような短いものではほぼ 1 年となっている．通常は打ち上げの数ヶ月前の時点で開発を終え，ロケット側に引き渡して打ち上げの準備を行わなければならないため，実際の開発期間は図 2.4 よりもさらに短くなる．

このように，現状では超小型衛星の開発は本来超小型衛星が狙っていた開発期間よりも更に短い期間で実現する必要に迫られており，問題となっている．

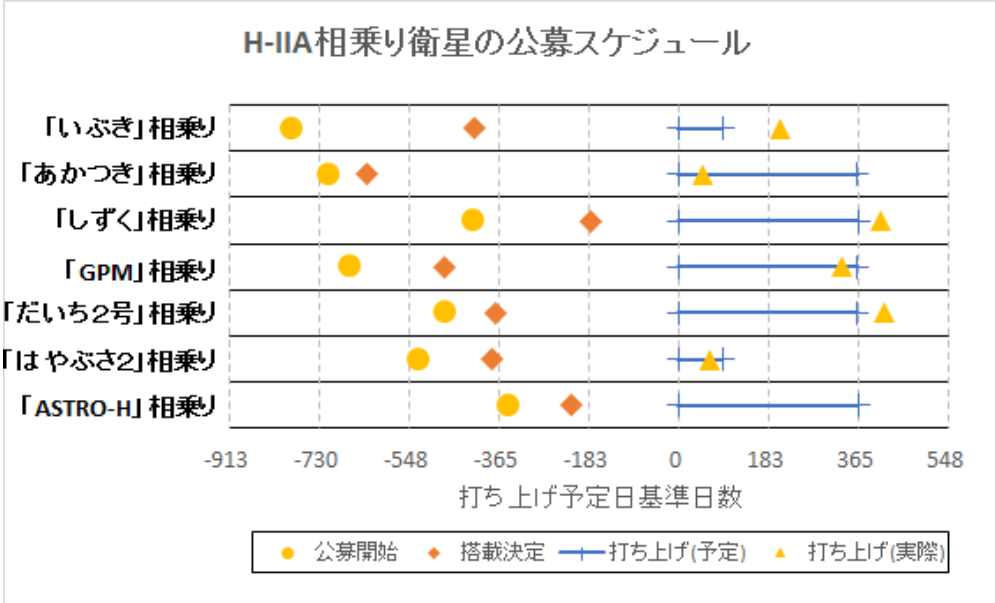


図 2.4　H-IIA 相乗り超小型衛星の採択スケジュール

2.4　再利用性と軌道上再構成能力の強化がもたらす利点

前述の問題点を超小型衛星の特徴である短期・低コストを維持したまま解決するためには超小型衛星の再利用性と軌道上再構成能力を高めることが有力な手段になると予想される．

再利用とは過去の衛星で使用した実績品を，場合によっては改良しながら，繰り返し使用する活動を指し，過去の衛星での技術・知見を次の衛星で活用する複数の衛星間で実施される取り組みである．再利用は主に衛星の開発段階に適用され，その対象はハードウェアとソフトウェアの両方が対処となる．

軌道上再構成とは衛星を打ち上げた後に軌道上で衛星の動きを変える活動を指し，個々の衛星内で実施される取り組みである．軌道上の衛星に対してハードウェアの再構成を実施することは現状困難であり，この取り組みは専らソフトウェアを対象に実施される．

以下ではこれらの 2 つの手段がそれぞれの問題点に対しどのような利点をもたらすかを整理する．

2.4.1　技術・知見の蓄積・共有の不完全さ

過去の衛星で使用した実績品を繰り返し使用する再利用は過去の技術・知見を蓄積・共有して活用する活動に他ならず，これを積極的に実施することで超小型衛星の継続的な成功率の向上が期待できる．

また，軌道上再構成によって衛星の柔軟な運用が可能であれば，軌道上で不具合や不測の事態が

発生した場合にも原因の究明や対策の検討を柔軟に実施でき，続く衛星に向けた知見をより多く獲得することが可能になると予想される．これは高頻度での打ち上げを目指す超小型衛星全体の成功率を高めていく上で大きな利点となる．

2.4.2　高性能化と運用の長期化

再利用性と軌道上再構成能力の強化は超小型衛星の高性能化と運用の長期化に対しても様々な利点をもたらす．

まず再利用については，実績品を改良しながら開発を進めることで衛星の継続的な性能向上が可能になると予想される．これは高性能化が求められている超小型衛星に対して大きな利点となる．また，実績品の再利用によって安定した衛星のバス機能を容易に実現することが可能になると考えられる．これは超小型衛星の運用長期化に対して大きな利点である．

次に軌道上再構成能力の強化については，これを積極的に活かして打ち上げ後に，例えば姿勢制御系のゲインといったパラメータの推定や調整を実施することで衛星の性能を最適化するような，軌道上調整が可能になると予想される．超小型衛星に求められる性能が高度化するに伴い，地上での試験・検証は複雑・大規模化する傾向にあり，全てを地上で実施するのは試験期間や費用の増加といった点で超小型衛星の利点と相反する部分も多い．軌道上を衛星にとって最高の試験環境と考え，軌道上で試験を行うことでこれらを削減することは短期・低コストでの開発を目指す超小型衛星にとって大きな利点となる．また，軌道上再構成能力の強化によって，打ち上げ後に生じる不測の事態への柔軟な対応が可能となることは衛星の軌道上での生存能力向上に大きく貢献すると予想される．これは特に近年運用の長期化が進む超小型衛星について大きな利点である．

2.4.3　開発期間の超短期化

再利用性と軌道上再構成能力の強化は衛星の開発期間の短期化に直結する．これは特に開発期間が外的要因によって左右される超小型衛星において大きな強みとなる．

再利用性を高めることで，既に過去の衛星で実績のあるハードウェアやソフトウェアの再利用が可能となれば，衛星開発の中で大きな割合を占める要素開発や試験の期間を大幅に短縮出来る．これは超小型衛星の開発期間全体を短縮する上で非常に重要な利点である．

軌道上再構成能力を高めることも，開発期間の短期化に貢献する．軌道上再構成能力の強化によって従来地上で行っていた試験の一部を軌道上で実施することが可能となれば，ハードウェアの開発が完了した早い段階で衛星を打ち上げ可能な状態とすることが出来る．また，新規技術に積極的に挑戦する場合が多い超小型衛星においては，開発期間全体の中でハードウェアの開発に割ける期間が増加することは大きな利点となる．

2.5　従来の取り組みの整理

前節で述べたとおり，再利用性と軌道上再構成能力の強化は超小型衛星の抱える問題点の解決に適した重要な手段であると考えられる．本節ではこれらを積極的に活用するためにこれまで実施されてきた各種取り組みを整理し，現状を確認すると本研究の主張に対して不足している点の明確化を行う．

2.5.1　再利用性の強化に関する取り組み

超小型衛星の再利用に関する取り組み

前述した通り，衛星開発における再利用は衛星の搭載ハードウェア面と搭載ソフトウェア面の両面で実施される活動である．ここでは超小型衛星におけるそれぞれの取り組みを整理する．

まず搭載ハードウェア面に関しては，過去の超小型衛星開発を通じて多くの実績部品・コンポーネントやそれらをいかに搭載し利用するかの知見が蓄積されてきている．

前述したほどよしプロジェクトでは，「超小型衛星の開発を支える国内のサプライチェーンネットワークの構築」をプロジェクト目標の 1 つと位置づけ，各種搭載コンポーネントの開発とその供給網の整備が行われている [16]．また，近年では超小型衛星向けに実績のある搭載コンポーネントを販売する企業も登場してきており，これらの製品を購入することで実績品の再利用が可能になりつつある．

超小型衛星の搭載ハードウェア面では，各搭載コンポーネントの開発だけではなく，超小型衛星全体とそのコンポーネントの試験を統一的に実施出来る施設の整備や，そこでの知見を活かした環境試験条件の標準化に向けた取り組みも進んでいる [17]．このような取り組みも搭載コンポーネントの再利用を促進する上で重要な役割を果たしている．

搭載ハードウェア面についてはここまでで述べたような再利用を実現する環境の整備が行われているだけでなく，実際にこれらを活用して開発した超小型衛星も登場してきている．例えば，ほどよしプロジェクトの後に開発された超小型深宇宙探査機 PROCYON ではほどよしプロジェクトで開発された各種搭載コンポーネントを積極的に活用して開発期間の短期化が実現されている [18]．

以上のように，搭載ハードウェアに関しては超小型衛星として再利用を可能とする環境構築に向けて活発な活動が展開されている状況にあり，これらを活用した成果も得られつつある．

これに対して搭載ソフトウェア面に関しては，これまでの超小型衛星開発を通じてその構成の共通化や部品化などは行われておらず，各衛星ごとに独立して開発が行われているのが現状である．

搭載ソフトウェアの再利用が実現できていない理由として，問題点としても挙げた通り，各衛星個別で開発が行われているためその内容が衛星のミッションや搭載ハードウェア構成に大きく依存した設計・実装となっていることが考えられる．搭載ソフトウェアの再利用性を向上するためには搭載ソフトウェアを衛星ごとに異なる部分と共通な部分で区別し，体系的に整理した枠組みが必要となるが現状このような枠組みは存在していない．

前節で述べた超小型衛星の問題点を解決するためには搭載ハードウェアだけでなく搭載ソフトウェアについても再利用性を高めることが必要であり，本研究ではこの実現を目指す．

組み込み開発でのソフトウェア再利用に関する取り組み

衛星の搭載ソフトウェアは衛星という機器に組み込まれて動作するソフトウェアであり，分類としては組み込みソフトウェアに該当する．組み込みソフトウェアは衛星だけにとどまらず航空機や自動車，各種家電製品など様々な分野で開発が行われており，これらの分野においては再利用に関する取り組みが進められている．

本節では，このような一般の組み込み開発で行われているソフトウェアの再利用に関連する取り組みとして，多くの分野で共通に実施されているハードウェアの抽象化とリアルタイム処理の実現手法を取り上げる．

ハードウェアの抽象化とはソフトウェアの構成を階層化し，ハードウェア毎の違いを隠蔽することで，ソフトウェアの移植性を高める手法である．ソフトウェアの構成をソフトウェア内での情報の流れに応じて階層化して実装し，階層間で情報をやりとりするインターフェースを明確化することで，ソフトウェア内部でハードウェアの差異が影響を及ぼす箇所を特定の階層内に局所化し上位層から隠蔽する．ハードウェア構成が変化した場合でも，その違いを階層内で吸収し，上位層とは同一のインターフェースを保つことで，上位層にはその影響が波及せず，高い移植性を実現している．

このような階層化のアプローチの具体例としては，様々な機器が接続される OS に実装されている HAL (Hardware Abstraction Layer) や，複数の機器を相互に接続する際のモデルである OSI(Open System Interconnection) 参照モデルなどが存在し，一般にソフトウェア開発全般で広く採用されている方式である．

ハードウェアの抽象化に加えて，多くの組み込みソフトウェア開発で再利用性を高める目的で実施されている取り組みとして，リアルタイム処理の実現手法の整理が挙げられる．

組み込みソフトウェアの多くでは，ソフトウェアが実現する処理にリアルタイム性が要求される場合が多い．リアルタイム性とは，ソフトウェアの処理自体が正しく実行されることに加えて，処理を完了させる時刻に対する要求が明確に現実の時刻と対応付けて規定される性質を指す．リアルタイム性は処理が指定された時刻までに完了しなかった場合の影響によって大きくハードリアルタイム性とソフトリアルタイム性の 2 つに区分される．処理が指定された時刻までに完了しなかった場合に致命的な影響が生じる場合はソフトウェアはハードリアルタイム性を有し，逆に致命的な影響が生じない場合はソフトウェアはソフトリアルタイム性を有す．衛星を含め各種制御など時間制約の厳しい処理を実現する組み込みソフトウェアは多くの場合ハードリアルタイム性が要求される．

これまで組み込みソフトウェアにおけるリアルタイム処理の実現手法を図 2.5 にまとめる．図ではリアルタイム処理の実現手法が Phase 1 から Phase 3 の大きく 3 つの段階を経て発展してきたことを示している．

第 1 段階は最も初期の組み込みソフトウェアの開発状況に対応する．この段階では組み込みソフ

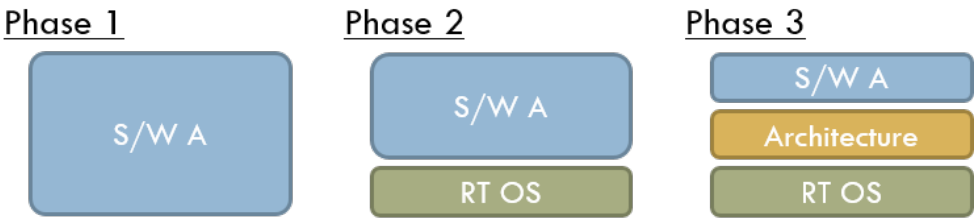


図 2.5　リアルタイム処理の実現手法の変遷

トウェアは各目的に応じて独自のソフトウェアが開発され，組み込みソフトウェアが実現すべき機能をリアルタイム性も含めて一体で実現していた．この段階では組み込みソフトウェアはそれぞれの目的に完全に特化されており，ソフトウェアの再利用は容易には行えない状態である．

第 2 段階では組み込みソフトウェア全体がリアルタイム OS と組み込みソフトウェア固有の機能の 2 つに分離された．第 1 段階においてリアルタイム性を有する組み込みソフトウェアが多数開発される過程を通じてリアルタイム性を実現するのに必要となる時間資源管理などの機能が共通化されることが見出され，これらをまとめたリアルタイム OS が開発された．リアルタイム OS の出現により，第 1 段階ではこれまで個別に実現していたリアルタイム性に関する機能はリアルタイム OS の機能を呼び出すことで実現可能となり，各ソフトウェア固有の機能の開発に注力することが可能となった．また，異なる実行環境であっても同一のリアルタイム OS を用意することができればリアルタイム性に関する部分については容易に移植が行える環境が構築された．リアルタイム OS の具体例としては TOPPERS[19] や VxWorks[20]，Linux RTAI[21] 等を挙げることが出来る．

第 3 段階では，リアルタイム OS と組み込みソフトウェア固有の機能の間にそれぞれの開発分野に特化した共通機能をまとめた階層を追加したソフトウェア・アーキテクチャが構築された．リアルタイム OS がソフトウェアがリアルタイム性を実現するのに一般的に必要とする機能を抽出した汎用的なものであるのに対し，ソフトウェア・アーキテクチャはリアルタイム OS を活用しつつ，各利用分野に特化した構成となっている点が特徴である．ソフトウェア・アーキテクチャではまず，多くの種類が存在するリアルタイム OS を抽象化してリアルタイム OS と組み込みソフトウェア固有の機能との間のインターフェースを規定し，異なるリアルタイム OS 上であってもソフトウェア・アーキテクチャが同じであれば同一の固有の機能を再利用できる環境を構築している．これに加えて，例えばコンポーネント間の通信プロトコルといったような，ソフトウェア・アーキテクチャが利用される分野毎に共通に必要とされる機能を抽出してソフトウェア・アーキテクチャ自体の機能として整備することで，組み込みソフトウェア固有の機能として開発が必要となる部分を第 2 段階よりも更に限定して，固有の機能の開発に注力できる環境を構築している．このようなソフトウェア・アーキテクチャの具体例としては自動車分野で用いられている AUTOSAR[22][23] や航空機分野で用いられている ARINC-653[24] 等を挙げることが出来る．分野内で同一のソフトウェア・アーキテクチャを利用する事によって，図 2.6 に Phase 3 と Phase 3’ として示すように，これまで複数の機器で実現していた機能を一つの機器に統合したり，逆に分割したりといった操作

が容易になる．近年では計算機の処理能力向上にともなって，多くの処理を一つの機器の集約して実現する傾向にある．具体的な例としては，航空機の IMA(Integrated Modular Avionics) 等が挙げられる．

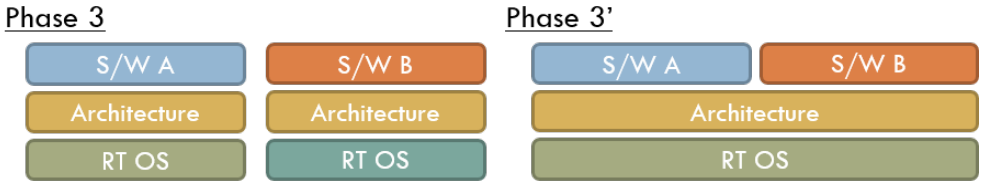


図 2.6　アーキテクチャによる機能統合

ここまでで述べたように，一般の組み込みソフトウェア開発ではソフトウェアの再利用に対する取り組みが活発に行われており，これらを超小型衛星の搭載ソフトウェア開発に取り入れることは有意義であると考えられる．

中・大型衛星の搭載ソフトウェア再利用に関する取り組み

本節では中・大型衛星において実施されている搭載ソフトウェアの再利用に関連する取り組みを整理する．

衛星の搭載ソフトウェアの開発においても，前節のリアルタイム処理の実現手法の箇所で述べたのと同様に，初期では衛星毎に独立に搭載ソフトウェアの開発が行われていた．このような段階で採用出来る最も単純な再利用としては，新たに衛星の搭載ソフトウェアを開発する際に，過去の衛星の搭載ソフトェアをそのまま流用し，構成が異なる部分にパッチを当てて対処する手法が考えられる．このような手法は新たに開発する衛星と流用元の衛星の構成がほぼ同一の場合には単純な変更のみで容易に対処が可能であるが，ソフトウェアの内部構成が十分に整理されていないため，大規模な構成変更に対応することは難しく，加えて変更を重ねる毎にソフトウェアが複雑化し，保守が困難になる点が問題となる．

このような問題点を解決するためにまず行われた取り組みが，搭載ソフトウェアの階層化である．一般の組み込みソフトウェアと同様に衛星の搭載ソフトウェアの構成を情報の流れに沿って階層化して実装し，搭載ソフトウェアの中で機器の構成に依存する箇所を局所化することで，搭載機器の変更といったような衛星のハードウェア構成の差異がより上位に位置する姿勢決定・制御計算等の処理に影響を与えない環境を構築し，再利用性を高める取り組みが行われている [25]．

超小型衛星の搭載ソフトウェアの開発においてもこのような取り組みを取り入れることは再利用性を高める上で有意義と考えられる．

搭載ソフトウェアの構成の階層化に加えて，中・大型衛星の搭載ソフトウェア開発では，一般の組み込みソフトウェア開発での取り組みとして前節で取り上げたようなソフトウェア・アーキテクチャの構築も行われている．特に近年活発に進められているのが SpaceWire[26] を基本とするハードウェア・ソフトウェアを合わせた衛星アーキテクチャの構築である．SpaceWire は

ESA(European Space Agency) を中心に策定が進められている宇宙機内のデータ通信インターフェースであり，この取り組では衛星に搭載する機器同士の接続を全てを SpaceWire に統一することで開発の効率化を目指している．SpaceWire を適用した具体的な例としては，国内では宇宙航空研究開発機構 (JAXA) 宇宙科学研究所 (ISAS) での取り組み [27]，海外では米国の Space Plug and Play Avionics(SPA)[28] 等を挙げることが出来る．

ISAS/JAXA で実施されている取り組みでは，まずハードウェア面として，宇宙機内部の機器同士の通信インターフェースを SpaceWire に統一し [27]，これらの機器を SpaceWire を通じて管理する計算機の開発が行われている [29]．ソフトウェア面では，従来地上で活用されてきたリアルタイム OS を元に，宇宙機に適応出来る機能と信頼性を有するリアルタイム OS の開発が行われ [30][31][32]，SpaceWire 上での通信プロトコルとして RMAP(Remote Memory Access Protocol)[33] を統一的に採用し，これらのに関連するソフトウェアを共通機能として用意したソフトウェア・アーキテクチャの構築が進められている [34][35]．

このようなアーキテクチャの採用は，衛星に搭載する機器の通信インターフェースを全て SpaceWire に統一出来る中・大型衛星では利点が大きいと考えられるが，宇宙用ではない民生機器などを積極的に採用して開発を行うことが多い超小型衛星ではこのような統一的な環境を構築することは難しく，直接は適用できない点が問題となる．中・大型衛星では SpaceWire に対応していない機器については通信方式を変換する機器を追加することでアーキテクチャに適合させる場合も存在するが，容積や電力といったリソースに大きな制約がある超小型衛星ではこのような方式の採用も難しい．

この他の再利用に関する取り組みとして，衛星機能のモデル化を挙げることが出来る．これは，運用者等の視点で外側から見た衛星の振る舞いを体系的に記述するモデルを構築し，機能間の連携や地上からの運用をそのモデルを用いて管理することを目指す取り組みである．多くの衛星を表現出来るモデルを構築することは，衛星が共有する動作の概念を抽出することであり，モデルを構築し活用することでこの概念を多数の衛星で再利用することが可能になる．

衛星のモデル化の具体例としては，ISAS/JAXA で進められている宇宙機の機能モデルを挙げることが出来る [36][37]．この取組では，宇宙機を構成する各機能を表現するモデルとして FMS(Functional Model of Spacecraft)[38] を構築し，このモデルで記述された各機能に対するコマンド操作やテレメトリ情報を表現するプロトコルとして SMCP(Satellite Monitor and Control Protocol)[39] を規定している．これらに基いて記述された衛星のモデルは SIB(Satellite Information Base)[40] と呼ばれるデータベースに格納され，この SIB に基いて衛星の試験や運用を統合的に行う GSTOS(Generic Spacecraft Test and Operation Softoware)[41] と呼ばれるソフトウェアが開発され，実際の衛星開発に用いられている．

このようなモデル化の取り組みは衛星に必要とされる機能を整理する観点では非常に参考になるが，モデルで表現される機能を実際にどのように搭載ソフトウェアとして実現すべきかについては扱われておらず，具体的な搭載ソフトウェアの開発を行う観点では追加の検討が必要である．

2.5.2　軌道上再構成能力の強化に関する取り組み

軌道上での再構成は打ち上げ後に行う操作であるためハードウェア的な対応は難しく，再構成の対象は通常はデータの書き換えのみで可能なソフトウェアに限定される．以下では超小型衛星と中・大型衛星における軌道上再構成能力の強化に関連する取り組みをまとめる．

超小型衛星での軌道上再構成に関する取り組み

超小型衛星の分野では高い性能が求められる実用衛星を中心に，短期開発と高性能の両立を目指し，姿勢系ソフトウェアの各種パラメータ推定を打ち上げ後の軌道上で実施する手法が提案されている [43]．

ただし，議論されている軌道上再構成の内容はパラメータ調整による衛星性能のチューニングが中心であり，その効果の範囲は本研究が主張する軌道上再構成の利点を実現するには限定的である点が問題である．また，議論の中心は軌道上での各パラメータ推定手順の効率化に重点が置かれており，実際に再構成を実現するために搭載ソフトウェアに必要となる仕組みについては議論が行われていない点も問題である．

中・大型衛星での軌道上再構成に関する取り組み

従来の中・大型衛星では搭載ソフトウェアに対しても搭載ハードウェアと同様に打ち上げ前に地上で十分な試験・検証を実施することが前提であり，軌道上での再構成に関してはこれを前提とした開発に関する議論は行われていない．地上での試験・検証を確実にすることを目的に，開発の指針となる開発標準の整理 [44] や適用 [45][46]，V&V や IV&V といった検証・有効性確認の整理 [47] や実施といった観点を中心に各種取り組みが行われている状況にある [48][48][50]．

超小型衛星でも当然このような取り組みを開発に取り入れていくべきであるが，現状の超小型衛星が抱える問題点を考慮するとこの厳密な実施は開発期間などの面で難しく，これとは別のアプローチも必要になる．

深宇宙探査では，中・大型であっても探査機に自動化自律化機能として，異常の発生条件とその対処を打ち上げ後も地上から組み替えられる形で用意している例が存在する [51][52]．ただし，再構成の範囲は異常対処のみに限定されておりその効果は超小型衛星の場合と同様やはり限定的である点が問題である．

中・大型衛星でも軌道上で搭載ソフトウェアに予め用意したコマンドでは対処できないような不具合が発生した場合には，計算機のプログラム書き換えによる軌道上再構成が行われるが，これは衛星搭載計算機のメモリ内容に直接パッチを当てるといった機械語レベルの低層での処置となるのが通常であり，手順を誤ると衛星自体を失う可能性も高いため簡単には選択できず，実施する場合には事前に周到な準備が必要となる非常に実施負荷の高い手法となっている．

軌道上再構成の手段が安全ではあるが対処範囲も限定され，不測の事態には対処できない予め用意されたコマンドと，対処範囲は広いが実施負荷の高いプログラム書き換えという両極端な 2 種類

しかない現状は柔軟とはいえず問題がある．

2.6　本研究で取り組む課題

本節ではここまでの議論を踏まえ，超小型衛星が抱える問題点を解決する再利用性と軌道上再構成能力を備えた搭載ソフトウェアを実現するために本研究で取り組む課題の明確化を行う．

まず，前節で取り上げた一般や中・大型衛星でのソフトウェアの再利用・軌道上再構成に関する取り組みを超小型衛星の観点で整理する．

再利用に関しては，一般や中・大型衛星のソフトウェア開発で実施されているソフトウェアの階層化やアーキテクチャの構築，モデル化など，超小型衛星でも参考にできる取り組みが多数存在している．既に行われているこれらの取り組みを超小型衛星に直接適用することは難しいが，これらを参考に適切な修正を加えることで活用は可能と考えられる．

一方で，軌道上再構成については現状，超小型衛星に適した取り組みが存在していない．一般の組み込みソフトウェアでは仮に再構成が必要となった場合には，一度システムを停止してソフトウェアの書き換えを実施するといった衛星ではプログラム書き換えに相当する対応を比較的容易に実現出来るため，衛星に求められるような柔軟な再構成能力は必要とされていない．また，中・大型衛星では地上での試験・検証を重視しており，軌道上再構成に関しては採用されるとしてもその効果の範囲は限定的な状況にある．超小型衛星の問題点を解決するためには，これとは違ったアプローチが必要となる．

超小型衛星の搭載ソフトウェアの軌道上再構成を実現するためには，まず前提として地上で搭載ソフトウェアを構成する手法が明確でなければならない．これを実現するためにはまず，搭載ソフトウェアが衛星の動きを実現する仕組みの明確化が必要であり，さらにその仕組みに沿って衛星の動きを搭載ソフトウェアで実現する手順が明確化されていなければならない．これら地上での搭載ソフトウェア構成方法の明確化に加えて，軌道上でも地上と同一の手順・粒度で搭載ソフトウェアが構成できる仕組みを用意することで柔軟な軌道上再構成能力が実現される．

以上の議論を踏まえると，本研究で超小型衛星の抱える問題点の解決策として主張している再利用性と軌道上再構成能力を備えた搭載ソフトウェアは，

1. **体系性** - 地上での開発を明確な枠組みに沿って行えること
2. **柔軟性** - 軌道上再構成を地上と同様の手順・粒度で行えること
3. **局所性** - 衛星毎の差異を特定の箇所に限定できること

の 3 つの性質を備える必要があると考えられる．

再利用と軌道上再構成を実現するためには多くの衛星の搭載ソフトウェアが共通の枠組みに沿って開発される必要がある．枠組みが不明瞭であれば，異なる衛星間での再利用は望めず，軌道上での再構成も不可能となる．搭載ソフトウェアが「体系性」を有していることは両者の実現に何より重要である．

軌道上再構成能力の強化とは単に軌道上不具合への対処能力を高めることではない．既に多くの

衛星に用意されているメモリ書き換えによる再構成機能は対処能力の観点ではほぼ万能といえる．問題はその実施負荷の高さにあり，必要なのは不具合が発生した場合にその重篤度に応じて様々な再構成能力を選択できる「柔軟性」である．

個々の衛星はそのミッションの違いによってそのハードウェア構成や運用に違いが生じる．これらの違いを乗り越えて搭載ソフトウェアの再利用性を高めるためには搭載ソフトウェアの枠組みが前述の体系性を有するだけでは不十分で，これと同時に衛星毎に違いが生じる箇所を狭い範囲に閉じ込める「局所性」を備える必要がある．

これらの性質を備えた搭載ソフトウェアを実現するために，本研究では

- 衛星の必須機能と再構成能力とを融合させる仕組みの導出
- 必須機能を実現するアーキテクチャの構築
- アーキテクチャに基づく開発方針の整理

の 3 つを課題とし，図 2.7 に示す流れに沿ってこの解決に取り組む．以下にそれぞれの課題と本研究での解決方針について詳細をまとめる．

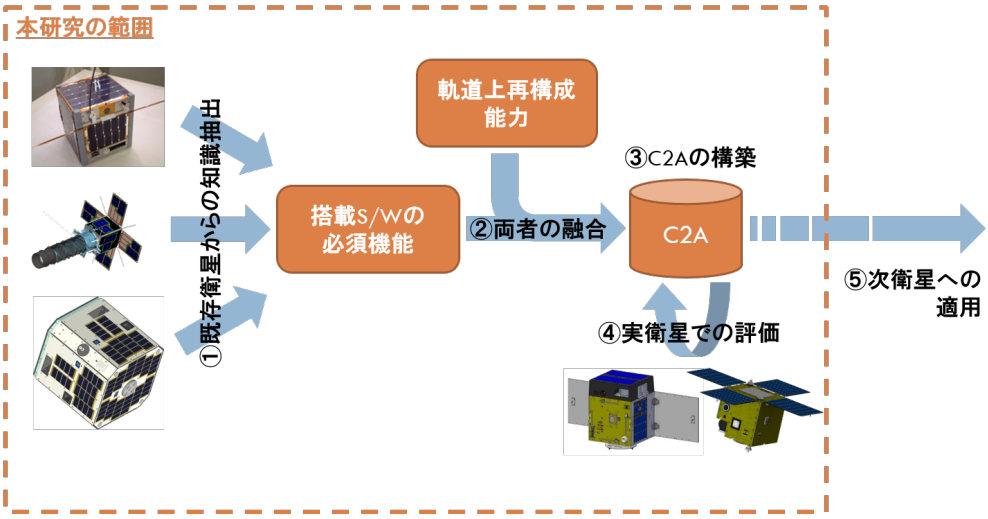


図 2.7　本研究の概要

2.6.1　衛星の必須機能と再構成能力とを融合させる仕組みの導出

明確な体系性を持った枠組みの元で開発を行うためにはまず，衛星の必須機能とは何かを明らかにする必要がある．ここで必須機能とは，個々の衛星の差異を超えて全ての衛星が普遍的に有す，衛星の動きを実現する機能を指す．このような機能を見つけ出して体系化することができれば様々な衛星の振る舞いをこの体系に沿って実現することが可能となる．

本研究では，過去に開発された衛星の搭載ソフトウェアの整理を行うことで，必須機能の抽出と体系化を行う (図 2.7-①)．

また，柔軟性の高い軌道上再構成能力を実現するためには再構成を実現する手段が必須機能と高い親和性を持つことが必要である．なぜなら，軌道上再構成能力とは衛星の動きを変化させる能力であり，この能力は衛星の動きを実現する必須機能に働きかけることで実現されるものと考えられるからである．

本研究では，必須機能の整理を通じて全ての必須機能が同一のパターンを共有することを見出し，この観点に基づいて必須機能全てをコマンド処理機能に集約することで，必須機能と再構成能力の融合を行う (図 2.7-②)．

2.6.2　必須機能を実現するアーキテクチャの構築

実際に衛星の搭載ソフトウェアを開発するためには，前節で述べたような概念的で抽象度の高い「仕組み」を実際の搭載ソフトウェア開発が行えるレベルまで具体化する必要がある．

本研究では，導出した必須機能に加えて，これを実現するために必要となる補助的な機能 (以降単に補助機能と書く) を洗い出した上でそれぞれの機能の要求を明確化し，両者を合わせた全体を Command Centric Architecture (C2A) として具体化する (図 2.7-③)．

2.6.3　アーキテクチャに基づく開発方針の整理

C2A の構築により，柔軟な軌道上再構成能力を有し体系化された開発環境がもたらされるが，その特性を活かし，再利用に必要となる局所性を実現するためには C2A の使い方についても考慮する必要がある．

本研究ではこの観点で，C2A に基いて構成される搭載ソフトウェアの「安全性」,「再構成能力」,「再利用性」を高める使い方を整理し，指針を示す．本研究では以上の内容で構成される手法の提案に加えて，C2A を実際の超小型衛星に適用して開発を行うことでその実証・評価も実施する (図 2.7-④)．

第 3 章

Command Centric Architecture (C2A) の導出

本章では再利用性と軌道上再構成能力の高い搭載ソフトウェアを実現する基礎となるアーキテクチャの導出を行う．本研究ではこのアーキテクチャを「Command Centric Architecture (C2A)」と名付け，以降論文中ではこの名称を使用する．

本章ではまず，これまでの衛星の搭載ソフトウェアで実現されてきた処理を具体例として示しながら衛星の搭載ソフトウェアが普遍的に有している機能を「必須機能」として導出し，それぞれの役割と特徴を整理する．

次に，導出した必須機能全体を俯瞰して整理することで必須機能の全てが「処理の塊を展開し決まったタイミングに実行する」という同一のパターンに集約されることを示す．この同一パターンへの集約が本研究で提唱する C2A の根幹部分であり，この観点に立って必須機能全てをコマンド処理機能という単一の機能に集約した C2A を導出する．

その後，必須機能を実現するために C2A が用意する必要のある補助機能について説明を加え，必須機能と補助機能を合わせた C2A の全体像を明らかにする．

最後に，必須機能全てをコマンド処理に集約する C2A を採用することで獲得されるメリットについて述べる．特に，搭載ソフトウェアの再構成能力が高まる点が重要であり，この点について重点的に議論する．

3.1 必須機能の抽出

必須機能とは個々の衛星の差異を超えて全ての衛星が普遍的に有す，衛星の振る舞いを実現する機能を指す．

本研究では過去の衛星の搭載ソフトウェアを調査し，必須機能として

- コマンド処理機能
- モード管理機能

- イベント処理機能

の 3 機能を抽出した．以下ではそれぞれの機能について詳細を述べる．

3.1.1 コマンド処理機能

衛星に望んだ動作を実行させるための指令をコマンドと呼ぶ．衛星にはその目的に応じて様々なコマンドが用意され，打ち上げ後はそれらを用いて運用が進められる．打ち上げ後，宇宙空間という地上から切り離された位置に置かれる衛星は，多くの場合，地上の運用局から電波を用いて送信するコマンドによって各種動作を指令し運用することになる．

衛星の運用を円滑に行うためには，運用に必要なコマンドを用意するだけではなく，その「実行の仕方」にいくつかの種類を用意する必要がある．コマンド処理機能はこの複数種類の実行の仕方を実現する機能である．

多くの衛星でコマンド処理機能として実装されるコマンドの実行の仕方は，コマンドの「実行タイミング」に関するものとコマンドの「実行方式」に関するものの 2 種類に区分される．実行タイミングに関する区分の中には「リアルタイムコマンド」と「タイムラインコマンド」の 2 種類があり，実行方式に関する区分の中には「シングルコマンド」と「ブロックコマンド」2 種類がある．実行タイミングによる区分と実行方式による区分とは直交する概念であり，コマンドの実行の仕方としては表 3.1 に示す通り，2×2 の合計 4 種類が存在する．これら 4 種類のコマンド実行の仕方は本論文と名称が異なる場合もあるが，多くの衛星において実装されているものである．

表 3.1　コマンド実行種類の一覧

		実行タイミング	
		リアルタイムコマンド	タイムラインコマンド
実行方式	シングルコマンド	シングルコマンド かつ リアルタイムコマンド	シングルコマンド かつ タイムラインコマンド
	ブロックコマンド	ブロックコマンド かつ リアルタイムコマンド	ブロックコマンド かつ タイムラインコマンド

以下それぞれのコマンドを実現する機能について詳細を述べる．

リアルタイムコマンド機能

最も単純に衛星の運用を実現するためには，少なくとも衛星側に地上側が送信したコマンドを受信したら即座に実行する機能が必要になる．

このようなコマンドの実行の仕方をリアルタイムコマンドと呼び，これを実現する機能がリアルタイムコマンド機能である．リアルタイムコマンドは衛星の運用に用いられるコマンド実行のタイミング制御として最も基本的な形式であり，衛星の可視運用中に多用される．

図 3.1 にリアルタイムコマンドの動作の概念図を示す．

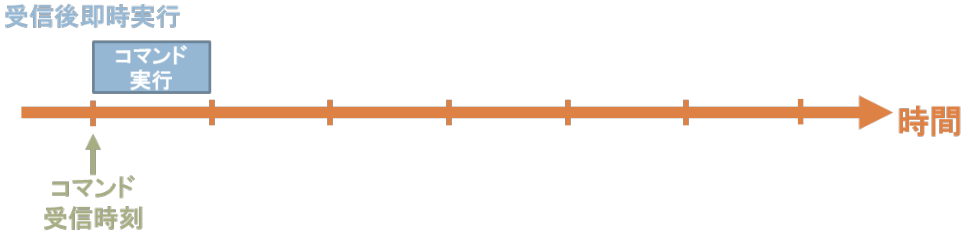


図 3.1　リアルタイムコマンドの動作概念

タイムラインコマンド機能

衛星の運用を行う上では，リアルタイムコマンドのように地上からのコマンドを受信したら即座に実行するのではなく，コマンドを指定した時刻に実行する機能が必要になる．

この機能が必要となる理由としては

- 実行時刻が重要となる運用の存在
- 地上から直接運用が行えない期間の運用の存在

の 2 つが挙げられる．

実行時刻が重要となる運用の具体例としては，地上の目標地点を撮影するための撮影コマンドを衛星が特定の位置にいる時刻で実行する場合や，衛星の軌道を変更するためのマニューバを適切なタイミングで実行する場合等が挙げられる．撮影の場合には，目標地点の位置と衛星の軌道・姿勢から撮影のタイミングが，また，軌道変更の場合には，現在の軌道と変更後の軌道との関係でマニューバのタイミングが場合によっては秒以下のオーダーで厳密に決まってくる．運用者のコマンド送信タイミングのズレや送信の間に存在する各種の遅延などを考えるとリアルタイムコマンドを用いて厳密なコマンド実行タイミングの制御を実現するのは大きな困難を伴う．また，仮にこれらの問題を解決出来たとしても，送信したコマンドが衛星側できちんと受信され，実行されることを確実に保証するのは特に無線を用いて通信を行っている衛星運用では難しい．

地上から直接運用が行えない期間中の運用の具体例としては，非可視中の運用が挙げられる．多くの衛星では地上側と衛星側とで通信を確立した状態での直接運用は常時行えるものではなく，その軌道や地上局の位置等によって決まる限定された時間帯にしか行えない．多くの超小型衛星が投入される低軌道であれば，地上局の位置にも左右されるが，多くの場合その運用回数は 1 日 4-5 回，1 回の運用時間は 10-15 分程度である．衛星のミッションをこの可視運用期間だけで実現することは通常不可能であり，それ以外にも例えば次の運用開始直前に地上局に対して電波の送出を開始するといったような可視運用の開始以前に実行したいコマンドも存在する．

これらを実現するためには，最初に述べたとおり，予め衛星側に地上から実行時刻を指定してコマンドを登録しておき，衛星側で指定された時刻にコマンドを実行する機能が必要になる．このよ

うなコマンド実行の仕方をタイムラインコマンドと呼び，これを実現する機能がタイムラインコマンド機能である．

図 3.2 にタイムラインコマンドの動作の概念図を示す．図 3.1 に示したリアルタイムコマンドと異なり，コマンド受信後，指定した時刻になるまでコマンドが実行されない点がタイムラインコマンドの特徴である．

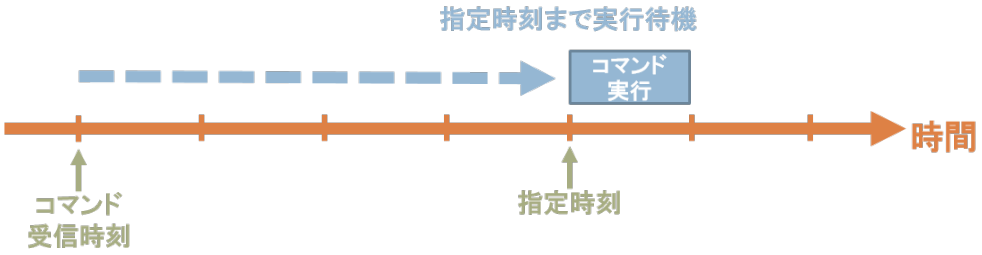


図 3.2　タイムラインコマンドの動作概念

シングルコマンド機能

コマンドを用いた衛星の操作を実現するためには，少なくとも特定の機器の電源の ON/OFF 操作や姿勢目標値の設定といったような衛星の操作したい機能に直接作用する基本的なコマンドが必要である．このようにコマンドの実行結果が衛星の特定の機能の操作と直接対応する単機能のコマンドをシングルコマンドと呼び，これを実現する機能がシングルコマンド機能である．

図 3.3 にシングルコマンドの動作の概念図を示す．

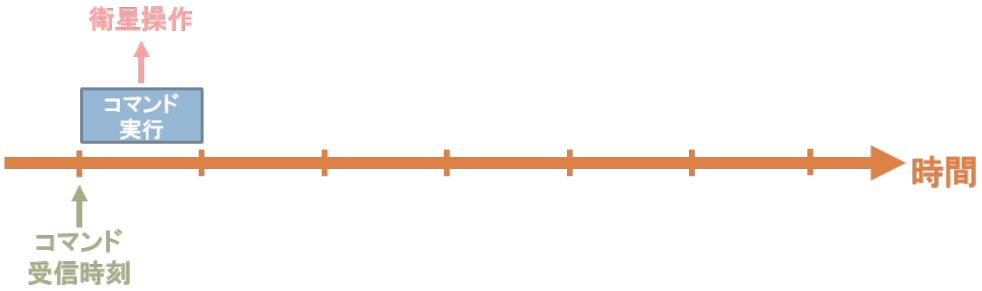


図 3.3　シングルコマンドの動作概念

ブロックコマンド機能

衛星の運用を行う上では，シングルコマンドのように単純に一つの操作だけを実行するのではなく，複数のコマンドを組み合わせてより複雑な操作を一つのコマンドとして実現する機能が必要になる．

この機能が必要となる理由としては

- 運用の効率化の実現
- 衛星の安全性の確保

の 2 つが挙げられる．

ブロックコマンドを用いて運用の効率化を実現する具体例としては，衛星運用の過程で生じる何度も繰り返し行うルーチン的な運用の効率化が挙げられる．多くの場合，衛星のコマンドは搭載される各機器や機能単位に用意されるため，このような例は特に複数機器にまたがる運用を行う場合に生じやすい．例えば，撮影を行う場合には，撮影に向けた目標姿勢の設定，機器の立ちあげ，撮影，撮影データの転送，機器の立ち下げ，姿勢の復帰といった一連の操作が考えられる．このような操作を毎回それぞれコマンドとして送信するのは単に面倒なだけでなく，これを何度も送信することで少ない運用時間が圧迫されるといった問題もある．

ブロックコマンドを用いて衛星の安全性を確保する具体例としては，運用上必ず組として実行する必要があるコマンドを確実に実行する場合や，原理的に複数の操作を 1 コマンドで完了させる必要がある場合などが挙げられる．

コマンドを組として実行する場合としては，常時 ON 状態とすると衛星の電力収支を維持できなくなるような大電力機器を限定させた期間のみ ON 状態とするために ON コマンドと OFF コマンドを組み合わせる操作が挙げられる．この場合，機器の OFF コマンドを ON コマンドとは別に送信する運用では，何らかの原因で OFF コマンドが衛星に届かなかった場合に電力収支が成り立たなくなるといった深刻な事態に発展するリスクがあり，シングルコマンドだけでの運用には問題がある．これに対してはタイムラインコマンドを利用して OFF コマンドから先行して登録していくような運用も考えられるが，手順としては煩雑で頻繁に実行する場合は効率化の観点でも両方が同時に登録される仕組みが存在することが望ましい．

原理的に操作を 1 コマンドで完了させる必要がある場合としては，衛星に複数のアンテナが存在する場合のアンテナ切り替え操作等が挙げられる．アンテナの切り替えを物理的なスイッチで行う場合には，特に複数のスイッチが存在するとスイッチ切り替えのタイミングで地上からの通信が物理的に途切れてしまうことで，衛星側が地上側からのコマンドを受け付けられない状態になる場合がある．このような場合は，シングルコマンドだけでの運用は原理的に不可能であり問題である．また，きちんとした手順を踏めば操作を完了できる場合であっても，手順を誤った場合に危険な状態となるリスクがある操作は毎回それぞれのコマンドを送るのではなく，予め操作をコマンド群として用意し安全性を確認した上で，それを繰り返し使用することが望ましい．

このような問題を解決して運用を行うためには最初に述べたとおり，衛星側で複数のコマンドをまとめて管理し，地上から 1 つのコマンドでそのコマンド群を呼び出して実行する機能が必要になる．このようなコマンド実行を実現する実行の仕方をブロックコマンドと呼び，これを実現する機能がブロックコマンド機能である．

図 3.4 にブロックコマンドの動作の概念図を示す．図 3.3 に示したシングルコマンドと異なり，ブロックコマンド自体は衛星操作に直結せず，複数のコマンドに展開され，展開された個々のコマンドが衛星操作を行う点がブロックコマンドの特徴である．

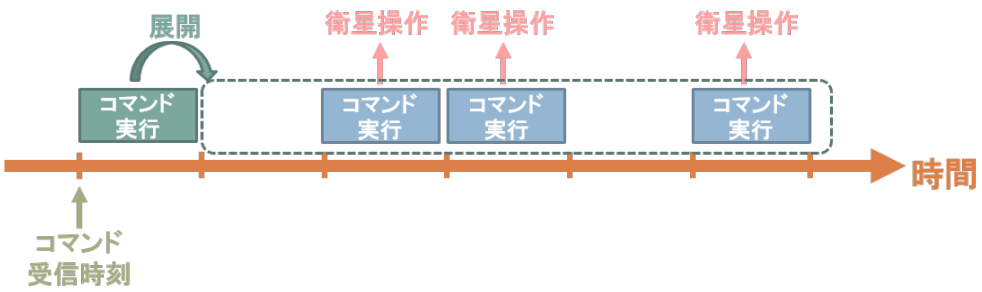


図 3.4　ブロックコマンドの動作概念

ここまでに挙げた具体例を整理すると，複数のコマンドをまとめて一括で呼び出すブロックコマンドには，

- ブロックコマンド内の各コマンドの実行タイミングを管理できること
- ブロックコマンドの内容を軌道上で書き換えられること

というより具体的な 2 つの要求が存在する．

まず，ブロックコマンドに登録される各コマンドは，そのブロックコマンドが呼び出された際に実行される順序とタイミングを指定できる必要がある．ここまで具体例として挙げてきた内容も全て，コマンドの実行順序と実行タイミングが指定できることを前提としている．例えば運用の効率化を実現するためにブロックコマンドを利用する場合には，ブロックコマンドとしてまとめられたコマンドが地上からそれぞれのコマンドを送信する場合と同様の順序・タイミングで実行されなければならない．

次に，ブロックコマンドの内容は打ち上げ後の軌道上でも地上から書き換えられる必要がある．特に運用の効率化については実際の運用で得られた経験に応じて運用の手順等が定まってくる部分もあり，打ち上げ後にこれらを柔軟に反映させるためには，ブロックコマンドの内容を状況に応じて書き換えられなければならない．

3.1.2　モード管理機能

軌道上という遠隔地にある衛星を安全に運用するためには，衛星の状態を常に管理し，望んだ状態に維持する必要がある．

多くの衛星ではこの衛星の状態管理を実現する手法として「モード」を用いた管理が行われている．本節ではまず，モードという考え方を整理し，モードによる衛星の状態管理を実現するためには「モード維持機能」と「モード遷移機能」の 2 つの機能が必要となることを述べ，その後それぞれの機能の詳細について明らかにする．

モードによる状態管理

衛星の状態は衛星に搭載された個々のコンポーネントの状態や衛星を構成する各サブシステムの状態などの組み合わせで定義される．コンポーネントの例として通信機を取り上げると，送信ビットレートや送信出力などがコンポーネントの状態であり，サブシステムの例として姿勢制御系を取り上げると，使用する姿勢決定則や姿勢制御則などがサブシステムの状態となる．最も単純に考えられる衛星状態の管理手法としては，これら全ての構成要素の状態をそれぞれ固定して管理する手法が考えられるが，衛星は多数のコンポーネントやサブシステムで構成されているため，この手法では状態の組み合わせが莫大な数となってしまい，実現は困難である．

衛星のモードとは，このように莫大な組み合わせを持つことになる衛星の状態を運用の目的に着目して，現実的かつ効率的に管理するために用いられる概念である．衛星各部の状態のうち，ある運用目的を達成するのに必要なコンポーネント・サブシステムの状態のみを抜き出して指定したものを衛星のモードとして定義し，衛星の状態管理を実現している．

図 3.5 に衛星モードの具体的な例を示す．図中には衛星のモードとして，「セーフモード」「地球撮影モード」「軌道制御モード」の 3 つが示されている．セーフモードとはともかく衛星を生存させることを目的としたモードであり，打ち上げ直後や異常事態が発生した場合などにひとまず衛星の安全を確保するために使用される．地球撮影モードと軌道制御モードはそれぞれその名の通り地球撮影と軌道制御の実施を目的としたモードである．

図ではセーフモードとして指定すべき状態として姿勢制御則を電力が最も確保できる太陽指向制御に，通信設定を地上で最も受信が行いやすい 4kbps の低速でかつ最大出力に指定している．一方，地球撮影モードでは姿勢制御則に撮影のため地心指向制御を指定し，ミッション機器のカメラを ON 状態とすることが指定されている．地球撮影モードではセーフモードにはあった通信設定の項目が無いが，これは地球撮影モードにでは通信設定は明示的に指定する必要がなく例えば通信速度の低速・高速関係なくどのような状態にあっても運用目的を達成できることを意味している．ここで示したものはあくまで単純化した例であり，実際にはより多くの指定が各モードに存在するが，それでも衛星構成要素の全状態を指定するよりは大幅に少なく，現実的かつ効率的な管理が実現されている．

図 3.5　モード定義の例

モードによる衛星の状態管理を実現するためには，衛星の搭載ソフトウェアに

- モードを維持するための処理

- モード間を遷移するための処理

の 2 つの処理を実現する機能が必要になる．

まず，衛星の搭載ソフトウェアには定義された各モードでそのモードを維持するために必要な処理を実行する機能が必要である．図 3.5 の例で言えば，太陽指向や地心指向といった姿勢を維持するために必要な処理や，各モードで電源が ON となっている機器のテレメトリ収集，機器へのコマンド配信等がモードを維持するために必要な処理として挙げられる．C2A ではこのモード維持を実現する仕組みを「モード維持機能」と呼ぶ．

次に，衛星の搭載ソフトウェアには，定義された複数のモードを切り替える機能が必要である．通常，衛星には多数の運用目的が存在し，どのような運用を行うかは状況に応じて変化する．このため衛星にはそれぞれの運用目的に対応するモードが存在し，状況に応じてあるモードから別のモードへと衛星のモードを遷移させながら運用が行われる．図 3.6 に具体的なモード遷移の例を示す．C2A ではこの遷移を実現するための仕組みを「モード遷移機能」と呼ぶ．

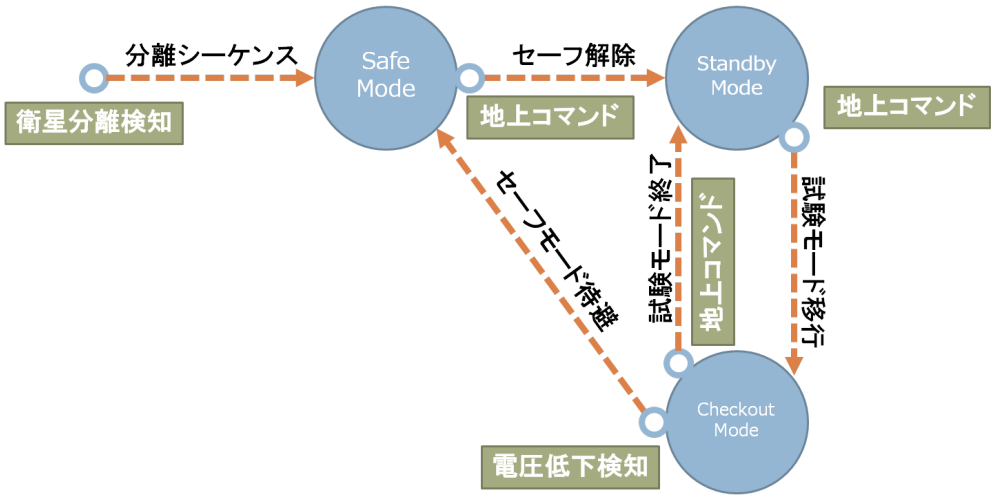


図 3.6　モード遷移の例

モード維持機能

衛星に定義されるそれぞれのモードにはその運用目的に応じて維持すべきいくつかの状態があり，衛星の搭載ソフトウェアには，これらを維持するために必要な処理が存在する．このような処理の実行を管理するの機能がモード維持機能である．

モード維持機能によって実行されるモード維持に必要な各種の処理は「周期を持った繰り返し」として表現される．この理由は，各モードには例えば「そのモードに遷移してから一定時間が経過した時点」といったような時間的に明確な終了条件が運用上存在しない場合が多く，「運用によって別モードへの遷移指示が出るまで」といった時間的に不明瞭な終了条件で運用がなされるためである．時間的に明確な終了条件が存在しないため，モードの維持に必要な処理は周期的に実行する

形で用意され，そのモードの開始から終了まで，この処理を繰り返し実行することでモードを維持することとなる．

モードを維持するためには通常多数の処理が 1 周期の中で実行されることになるが，この周期的に実行される処理群はその周期性を確実に保証するため，その「実行順序」と「実行タイミング」とを明確に管理する必要がある．

実行順序の管理が必要となる理由は，周期的に実行する処理同士に実行順序の依存関係が存在する場合が多いためである．例えば姿勢関連の処理では，センサ情報取得，姿勢決定，制御量計算，アクチュエータ出力といった形に処理の順序を維持する必要があり，この順序関係が崩れると望んだ結果を得られなくなる．

実行タイミングの管理が必要となる理由は，処理の周期性を維持するためである．衛星のモードを維持する処理は，指定された周期で実行できなかった場合にはその価値が失われる，ハードリアルタイム性の高い処理であり，モードを維持するために用意される多数の処理を 1 周期の中で確実に実行するためには，それぞれの処理について 1 周期の中で実行を開始するタイミングと，その処理に要する時間とを明確化し，すべての処理が 1 周期の中で確実に実行されることを保証する必要がある．

モード維持機能が実現する機能の概念を図 3.7 にまとめる．この図では 1 周期に処理 A～処理 B の 3 個の処理が実行されている．モード維持機能は，1 周期の処理を実行する際必ず処理 A，処理 B，処理 C の順で実行し，それぞれの処理を開始するタイミングも維持する必要がある．

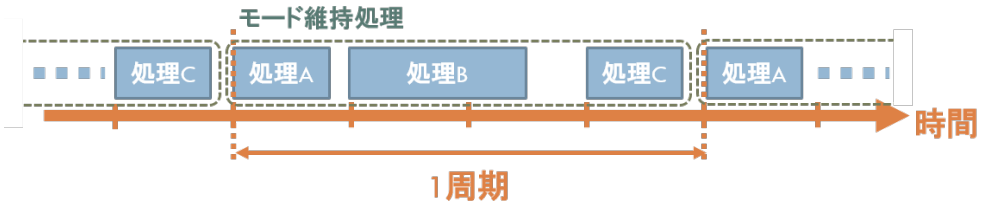


図 3.7　モード維持機能の動作概念

モード遷移機能

衛星の運用目的は状況に応じて変化するため，衛星のモードもこれに合わせて変化していくこととなる．衛星搭載ソフトウェアの中で，このようなあるモードから別のモードへの遷移処理を実現する機能がモード遷移機能である．

モード遷移処理は前述のモード維持処理とは対照的に，処理の開始と終了が明確な処理である．これはモード遷移処理が，遷移元のモードからスタートして予め定めた処理を順番に実行して遷移先のモードを開始する準備を整えた上で遷移先のモード維持処理をスタートさせるという一連のシーケンスとして実現されるためである．

モード遷移を実現するためには，モード遷移を開始する時点で遷移元と遷移先のモードから定ま

る予め用意した遷移処理のシーケンスを展開し，決められたタイミングで順次実行していく必要がある．

モード遷移処理の開始に遷移先だけでなく遷移元のモードが情報として必要となる理由は遷移先のモードが同じ場合であっても，遷移を開始するモードが異なれば必要な処理が異なってくるためである．例えば，異なる 2 つのモード，モード A とモード B からモード C に遷移する場合，モード A とモード B で機器の電源 ON/OFF 状態が異なれば，モード C へ遷移するために必要となる電源操作の種類や順序は当然異なってくる．

モード遷移処理において各処理の実行タイミングが重要となる理由は，多くの場合で遷移処理として実行される各処理の実行順序に依存関係が存在するためである．具体的には，ある機器を初期化する場合には当然その前に機器の電源を入れなければならないといった例が挙げられる．場合によっては電源を入れた後指定された時間が経過した後に初期化処理を実行する必要があるといったケースも考えられる．このような制約条件を確実に満たしてモード遷移を行うためには実行タイミングの制御が正確に行えることが重要となる．

モード遷移機能が実現する機能の概念を図 3.8 にまとめる．この図では遷移処理として，処理 A ～処理 D の 4 つが実行されている．モード遷移機能はこれらの遷移処理をモード遷移開始のタイミングで展開し，その実行順序とタイミングを守って実行する必要がある．

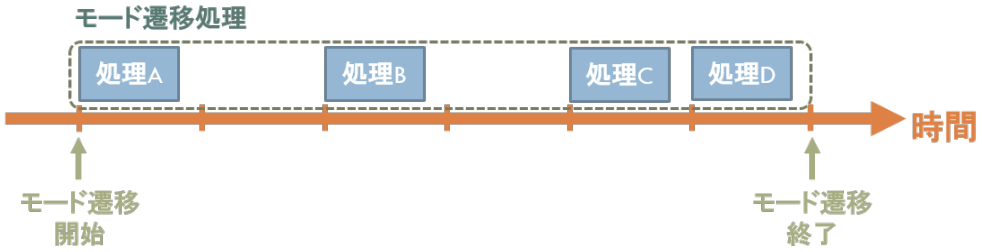


図 3.8　モード遷移機能の動作概念

3.1.3　イベント処理機能

衛星の安全性を確保するためには，発生時期が予測できない突発的な事象，いわゆるイベントが発生した場合に地上からの指示によらず自律的に対処する機能が必要である．この理由はコマンド処理機能の節でも触れた通り，衛星は常時地上から運用が行えるわけではなく，むしろそのタイミングは非常に限定された形となるのが通常で，多くの場合即座に地上から衛星に対して指令を送ることが不可能に近いためである．

このような機能が必要となる例としては，姿勢系機器の故障による姿勢制御異常や電力収支異常によるバッテリ電圧の低下等が挙げられる．これらのイベントが発生した場合には，衛星は故障した機器をリセットして復帰を試みたり，生存に不必要な機器を使用しないモードへ遷移して必要な電力を削減し衛星の安全を確保したりといった対処を自律的に実行できなければならない．

このような衛星の自律化を目的として用意される機能がイベント処理機能であり，衛星になんらかのイベントが発生した場合に，予め用意してある対応を自動的に実行し，地上に変わって対処を行う．

イベントへの対処は多くの場合複数の処理で構成される．先に例として挙げた機器のリセットであれば，電源の OFF，ON，初期化といったような処理のまとまりが考えられ，電力確保であれば，不必要な機器を順次 OFF し，モードを変更するといった処理のまとまりが考えられる．加えて，イベントに対処するためにはモード遷移処理と同様にこれらの処理を予め決めた順序・タイミングで実行しなければならない．

イベント処理機能が実現する機能の概念を図 3.9 にまとめる．この図ではイベントへの対処として処理 A〜処理 C の 3 つが実行されている．イベント処理機能はこの対処をイベント発生のタイミングで展開し，その実行順序とタイミングを守って実行する必要がある．

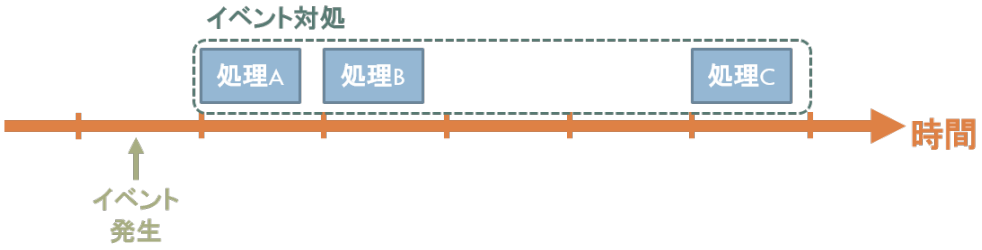


図 3.9　イベント処理機能の動作概念

3.2　必須機能のパターン集約

ここまでで，衛星搭載ソフトウェアの必須機能として

- コマンド処理機能
- モード管理機能
- イベント処理機能

の 3 機能を抽出した．

これらを俯瞰すると全ての必須機能が「予め用意された処理の塊を展開し，決まったタイミングで実行する」という同一のパターンを有していることが見出だせる．

まず，コマンド処理機能についてみると，ブロックコマンドは複数のコマンドをまとめて一つのコマンドとして呼び出す機能であり，この機能を実現するためには呼び出し対象である複数のコマンドを展開し，各コマンドを実行することが要求された．ブロックコマンドとしてまとめられるコマンド群は予め用意された処理の塊と考える事ができ，ブロックコマンドの呼び出しを実現するにはこの処理の塊を展開し，それぞれの処理を決まったタイミングで実行する必要があった．

次に，モード管理機能について考える．モード管理機能はモード維持機能とモード遷移機能で構成された．

モード維持機能はモードを維持するために必要な処理を周期的に実行する機能であり，この機能を実現するためにはモード維持に必要となる様々な処理を 1 周期の中で実行していくことが要求された．モードを維持するために 1 周期の中で呼び出される様々な処理は予め用意された処理の塊と考える事ができ，周期的な処理を実現するためにはこの処理の塊を展開し，それぞれの処理を決まったタイミングで実行する必要があった．

モード遷移機能は遷移元のモードから遷移先のモードへのモード遷移を実現する機能であり，この機能の実現にはモード遷移に必要となる様々な処理を順次実行していくことが要求された．モード遷移に必要な様々な処理は遷移元のモードと遷移先のモードによって定まる処理の塊と考える事ができ，適切に遷移を実現するためにはこの処理の塊を展開し，それぞれの処理を決まったタイミングで実行する必要があった．

最後に，イベント処理機能は発生したイベントに対して自律的な対処を実現する機能であり，この機能の実現には発生したイベント情報に基づいて適切な処理を順次実行することが要求された．ここでも発生したイベントへの対処は事前に用意された処理の塊と考える事ができ，適切に対処を実現するためにはこの処理の塊を展開し，それぞれの処理を決まったタイミングで実行していく必要があった．

以上のように，3 つの必須機能はいずれも本節の最初に述べた通り，「予め用意された処理の塊を展開し，決まったタイミングで実行する」という同一のパターンを共有している．

C2A では，このパターンの同一性に注目して衛星の 3 つの必須機能を搭載ソフトウェア上で単一の機能に集約して実装し，見通しの良い搭載ソフトウェアの枠組みを構築する．

これまで別々の機能として扱われてきた必須機能を搭載ソフトウェア上で単一の機能に集約してすることは

- 必須機能の実装が容易
- 必須機能の試験が容易
- 必須機能の理解が容易

といった利点を生む．

まず，搭載ソフトウェアとして必須機能を実装する観点からは，必須機能を単一の機能に集約することで実装が容易になるという利点が生じる．機能の集約により，必須機能それぞれを個別に実装するのに比べ，必須機能の実現に必要な実装の規模が小さくなることが期待できる．これは C2A を継続的に保守する観点でも利点である．

次に，必須機能を 1 つの機能のみに集約して実装することは実装した必須機能を試験する観点でも有利に働く．機能の共有により，実装規模が小さくなることでソフトウェアの中で確認すべき箇所も少なくなり，効率的に試験を実施できる．また，不具合が発見された場合も修正箇所の特定が容易となるだけでなく，修正は全ての必須機能に反映されるので，必須機能全体の信頼性を高めることが容易になると期待できる．

最後に，必須機能を利用して衛星それぞれに固有の機能を実装する観点では，必須機能の全てが単一の機能に集約されることで，それぞれの必須機能の挙動を理解することが容易になるという利点が生じる．衛星それぞれの固有機能は必須機能から呼び出されて動作するため，これを実装する際には必須機能の動作を理解しておくことが必要となるが，C2A では必須機能全てがソフトウェア上で単一の機能で実現されているため全体の見通しが良く，その理解が容易になると期待できる．

3.3　コマンド処理機能への集約

前節では，衛星の必須機能全てが同一のパターンを共有している事を示しこのパターンの同一性に注目して必須機能全てを単一の機能に集約して実装する C2A の方針とその利点について議論した．本節ではこの方針に基づいて必須機能を単一の機能に集約して実装する具体的な手法について述べる．

C2A ではモード管理機能とイベント処理機能の 2 つをコマンド処理機能を用いて実現することを考える．より具体的には，必須機能が共有する「予め用意された処理の塊を展開し，決まったタイミングで実行する」というパターンを実現する手段としてブロックコマンドを用いることを考える．

必須機能の実現の中心にコマンド処理機能，特にブロックコマンドを置く理由として，ブロックコマンドの動作自体が抽出したパターンとよく一致している事はもちろんであるが，さらに追加としてブロックコマンドに軌道上でその内容の書き換えを可能とする機能が備わっている点が挙げられる．

ブロックコマンドとは既に述べたとおり複数のコマンドをまとめて一つのコマンドとして呼び出す機能であり，まとめられるそれぞれのコマンドをある種の処理と考えれば「予め用意された処理の塊を展開し，決まったタイミングで実行する」という必須機能が持つ共通パターンに良く一致している．これと同様のパターンを共有するモード管理機能とイベント処理機能もその実装に多少の工夫を加える事でブロックコマンドを用いてその機能を実現できる可能性は高いと考えられる．

また，ブロックコマンドは軌道上でその内容を書き換える機能を備えている．モード管理機能とイベント処理機能をブロックコマンドを用いて実現することができれば，このブロックコマンドの書き換え機能を利用することでこれらの 2 つの機能についてもそれらが呼び出す処理の内容を軌道上で書き換えられる可能性は高い．これは本研究で主張している軌道上再構成能力の強化につながると考えられる．

以下ではこのような観点に基づいて，衛星搭載ソフトウェアの必須機能をコマンド処理機能に集約して実装する手法について述べる．まず全ての機能の基本となるブロックコマンド機能の仕組みについて述べ，その後にモード管理機能とイベント処理機能をこのブロックコマンドを中心とするコマンド処理機能を用いて実現する手法について述べる．

必須機能の全てをコマンド処理機能に集約する基本的な考え方は，モード管理機能とイベント処理機能が実行する処理の塊をブロックコマンドとして定義し，処理を実行する際にはそのブロック

コマンドを展開することでそれぞれの処理の実行順序とタイミングを制御するというものである．ブロックコマンド機能の仕組みに続くモード管理機能とイベント処理機能の詳細説明では，この考え方を実現するために必要となる

- ブロックコマンドで何を管理するか
- ブロックコマンドでどのように機能を実現するか
- ブロックコマンドで機能を実現するには何が必要か
- 再構成能力を高める観点でどのような操作を用意するか

の各項目を明らかにする．

3.3.1 ブロックコマンド機能

ブロックコマンド機能とは複数のコマンドをまとめて一つのコマンドとして呼び出す機能である．ブロックコマンドとしてまとめられる各コマンドはその実行タイミングを指定でき，これによりブロックコマンド呼び出し時の各コマンドの実行順序や実行タイミングを制御できる．また，ブロックコマンドとしてまとめられるコマンドの内容は軌道上でも編集することができる．

本節では，ブロックコマンド機能を構成する各コマンドの実行タイミング制御を実現する機能を「展開機能」，軌道上でブロックコマンドの内容編集を実現する機能を「登録機能」と呼び，それぞれの機能について説明する．登録機能としては主に複数のコマンドをブロックコマンドとしてまとめて管理する仕組みについて述べ，展開機能としてはブロックコマンドの実行タイミング制御をタイムラインコマンドを用いて実現する仕組みについて述べる．

登録機能

ブロックコマンドの登録機能は

- ブロックコマンド定義テーブル
- ブロックコマンド毎の登録コマンド数
- 各登録コマンドの相対実行時刻

の 3 つを用いてブロックコマンドを管理する．

ブロックコマンド定義テーブルはブロックコマンドとして登録されるコマンドを管理する図 3.10 に示すような縦方向に「ブロックコマンド番号」，横方向に「コマンド番号」をとったテーブルである．ブロックコマンド番号は衛星に登録できるブロックコマンドそれぞれに振られた一意の番号であり，この番号によって特定のブロックコマンドを指定できる．コマンド番号はブロックコマンドを構成する各コマンドに振られた番号であり，この番号を用いることでブロックコマンドを構成する特定のコマンドを指定できる．

それぞれのブロックコマンドに登録されるコマンドの個数はブロックコマンド毎に異なるため，登録機能は各ブロックコマンドに何個のコマンドが登録されているかを管理する必要がある．これ

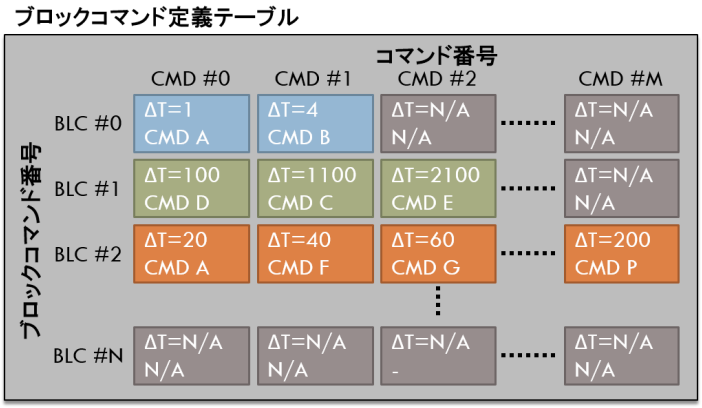


図 3.10　ブロックコマンド定義テーブルの概念

がブロックコマンド毎の登録コマンド数である．

ブロックコマンドとしてまとめられるそれぞれのコマンドは，ブロックコマンドが展開された際の実行タイミングを指定できなければならない．このため，登録される各コマンドにはブロックコマンドが展開された時刻からそのコマンドを実行するまでの相対実行時刻が設定される．この相対実行時刻の具体的な用途については続く展開機能の項で述べる．

ブロックコマンドへのコマンド登録はコマンドを登録する位置をブロックコマンド定義テーブル上で指定し，その位置にコマンドを書き込むことで実現される．ブロックコマンド定義テーブル上の位置はブロックコマンド番号とコマンド番号の 2 つを指定することで指定できる．

展開機能

展開機能は指定されたブロックコマンドを呼び出し，ブロックコマンドを構成する各コマンドをタイムラインコマンドに変換することで実行タイミング制御を実現する．

呼び出すブロックコマンドの指定にはブロックコマンド番号を用いる．展開機能は指定されたブロックコマンド番号に対応するブロックコマンドの内容をブロックコマンド定義テーブルから取得する．

ブロックコマンドを構成する各コマンドの実行タイミング制御は，ブロックコマンドが展開された時刻と各コマンドに設定された相対実行時刻を用いて各コマンドをタイムラインコマンドに変換して実行することで実現される．

展開機能はブロックコマンド定義テーブルを参照してブロックコマンドを構成する各コマンドの相対実行時刻を取得し，それに展開時刻である現在時刻を加算することで各コマンドの実行時刻を生成し，タイムラインコマンドとして登録する．タイムラインコマンドは登録されたコマンドを指定時刻に実行する機能であり，展開機能はこの機能を利用することで，ブロックコマンドを構成する各コマンドの実行タイミング制御を実現する．図 3.11 にこの展開機能の動作の概念図を示す．

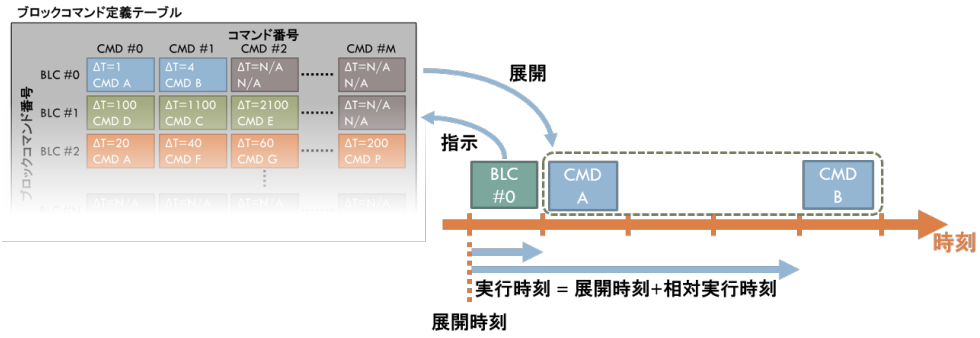


図 3.11　ブロックコマンド展開機能の動作概念

3.3.2　モード管理機能

モード管理機能は 3.1.2 節で述べた通り，モード維持機能とモード遷移機能の 2 つの機能で構成される．以下ではそれぞれの機能をブロックコマンドを用いて実現する仕組みについて説明する．

モード維持機能

モード維持機能はモードを維持するために必要な処理を周期的に実行する機能である．C2A のモード維持機能では，あるモードの 1 周期の中で実行する処理を全て 1 つのブロックコマンドに登録して管理する．具体的には，例えば搭載コンポーネントのテレメトリ取得処理や姿勢決定処理といった 1 周期の中で実行する処理を全てブロックコマンドとして登録する．

ブロックコマンドは複数のコマンドを一つにまとめる機能であり，モード維持機能が 1 周期の間に実行する処理をブロックコマンドを用いて管理するためには，モード維持に必要な処理全てをコマンドとして呼び出せる形で用意する必要がある．これについての詳細は別に補助機能として 3.4 節で説明する．

1 周期の間に実行する各処理の実行順序は各処理をコマンドとしてブロックコマンドに登録する際に指定する相対実行時刻によって規定する．例えば処理 A を処理 B よりも先に実行する必要がある場合，処理 A を呼び出すコマンドには処理 B を呼び出すコマンドよりも早い相対実行時刻を指定することで実行順序が規定される．

C2A のモード維持機能は各周期のはじめに 1 周期で実行する処理を定義したブロックコマンドを展開する．展開されたそれぞれのコマンドはタイムラインコマンドとして登録され，1 周期の中で指定されたタイミングに実行される．モード維持機能はこれを毎周期繰り返すことで，モード維持のために必要な周期的な処理をその実行順序を維持して決まったタイミングに実行する機能を実現する．

モードの維持に必要な処理はそれぞれのモードに応じて異なる場合が多い．C2A ではこの違いはモード毎に周期的に呼び出すブロックコマンドの違いとして表現される．モード維持機能はこれ

を管理するため，各モードで呼び出すブロックコマンド番号を図 3.12 に示すようなテーブルで管理する．このテーブルを「モード維持処理定義テーブル」と呼ぶ．

モード維持機能は自身が管理するモード維持処理定義テーブルの内容を軌道上でも地上の指示で書き換えられる機能をコマンドとして備える．この機能は搭載ソフトウェアの柔軟な軌道上再構成を実現するために重要であり，この詳細については 3.6 節で議論する．

モード遷移機能

モード遷移機能はあるモードから別のモードへの遷移を行う場合に必要な処理を実行する機能である．C2A のモード遷移機能では遷移元のモードから遷移先のモードへ遷移する際に実行する一連の処理を全て 1 つのブロックコマンドに登録して管理する．具体的には，機器の電源 OFF や電源 ON，遷移先モードでモード維持機能によって呼び出される各種処理の初期化等の処理をブロックコマンドとして登録する．

モード維持機能の場合と同様に，モード遷移機能がモード遷移時に実行する処理をブロックコマンドを用いて管理するためには，モード遷移で必要になるそれぞれの処理をコマンドとして呼び出せる形で用意する必要がある．これについての詳細は別に補助機能として 3.4 節の中で説明する．

遷移処理を構成する各処理の実行には順序関係がある．C2A ではそれぞれの処理の実行順序を各処理をコマンドとしてブロックコマンドに登録する際に指定する相対実行時刻によって規定する．モード維持機能の場合と同様，処理 A を処理 B よりも先に実行する必要がある場合は，処理 A を呼び出すコマンドに処理 B を呼び出すコマンドよりも早い相対実行時刻を指定することでその実行順序が規定される．

C2A のモード遷移機能はモード遷移開始時に，現在のモードから遷移先のモードへの遷移処理に対応するブロックコマンドを展開する．展開されたそれぞれのコマンドはブロックコマンドの展開時刻，すなわちモード遷移の開始時刻を起点とするタイムラインコマンドとして登録され，モード遷移に必要な処理を予め設定された実行順序とタイミングで実行していく．

遷移元のモードから遷移先のモードへ遷移する処理は，遷移先のモードだけではなく遷移元のモードによっても異なる場合が多い．C2A ではこの違いはモード遷移処理毎に呼び出すブロックコマンドの違いとして表現される．モード遷移機能は遷移元と遷移先の 2 つモードから適切なモード遷移処理に対応するブロックコマンドを選択する必要があり，これを実現するために各モード遷移で呼び出すべきブロックコマンド番号を図 3.12 に示すような遷移元モードと遷移先モードを縦横にとった「モード遷移定義テーブル」で管理する．このテーブルによって遷移元のモードと遷移先のモードから適切なブロックコマンドを選択することが可能となる．

モード遷移機能は自身が管理するモード遷移定義テーブルの内容を軌道上でも地上の指示で書き換えられる機能をコマンドとして備える．この機能は搭載ソフトウェアの柔軟な軌道上再構成を実現するために重要であり，この詳細については 3.6 節で議論する．

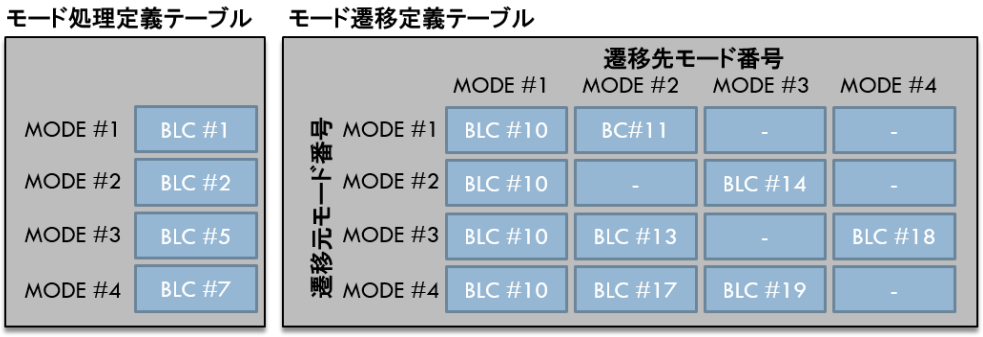


図 3.12　モード処理定義テーブルとモード遷移定義テーブルの概念

3.3.3　イベント処理機能

イベント処理機能は 3.1.3 で述べた通り，衛星で発生したイベントに対して適切な処理を自律的に実行する機能である．C2A のイベント処理機能では，イベント発生時に実行する一連の処理を 1 つのブロックコマンドとして定義する．具体的には異常が発生した機器のリセット操作や電力確保のための不必要な機器の電源 OFF 操作等の処理をブロックコマンドとして登録することになる．

モード管理機能で述べたのと同様に，イベント処理機能がイベント発生時に実行する処理をブロックコマンドを用いて管理するためには，それぞれの処理をコマンドとして呼び出せる形で用意する必要がある．これについての詳細は別に補助機能として 3.4 節の中で説明する．

イベントへの対応として実行される処理にはその実行に順序関係が存在する．C2A ではモード管理機能と同様にイベント処理機能でもこの実行順序関係を各処理をコマンドとしてブロックコマンドに登録する際に指定する相対実行時刻で規定する．処理 A を処理 B よりも先に実行する必要がある場合は，処理 A を呼び出すコマンドに処理 B を呼び出すコマンドよりも早い相対実行時刻を指定することで処理の実行順序が規定される．

C2A のイベント処理機能はイベント発生時にそのイベントに対応するブロックコマンドを展開する．展開されたそれぞれのコマンドはブロックコマンドの展開時刻，すなわちイベント対処の開始時刻を起点とするタイムラインコマンドとして登録され，対処に必要な処理を予め設定された実行順序とタイミングで実行していく．

イベントへの対処として実行する処理は発生するイベント毎に異なる．C2A ではこの違いをブロックコマンドの違いとして表現する．イベント処理機能は発生したイベントに応じて対処として展開するブロックコマンドを選択する必要があり，これを実現するためにイベントとそれに応じて展開するブロックコマンド番号をテーブルで管理する．このテーブルを「イベント処理定義テーブル」と呼ぶ．

イベント処理機能は自身が管理するイベント処理定義テーブルの内容を軌道上でも地上の指示で書き換えられる機能をコマンドとして備える．この機能は搭載ソフトウェアの柔軟な軌道上再構成

を実現するために重要であり，この詳細については 3.6 節で議論する．

3.4 補助機能の抽出

ここまでで C2A の必須機能の概念について説明した．必須機能の動作を実現するためには必須機能の他にいくつか補助的な機能が必要であり，本節ではこの部分について説明を加える．

C2A では必須機能を実現するために必要となる機能を補助機能と呼ぶ．この補助機能には，

- 時刻管理機能
- アプリケーション管理機能
- イベント記録機能

の 3 機能が存在する．

時刻管理機能は C2A の動作に必要となる時刻情報を維持・管理する機能である．この機能が管理する時刻情報はタイムラインコマンド機能が指定した時刻にコマンドを実行するために必要不可欠な情報である．C2A では全ての必須機能が呼び出す処理の塊をブロックコマンドで管理しており，これを展開することで処理を実現している．ブロックコマンドは展開時にタイムラインコマンドとして登録され実行されることとなるので，このタイムラインコマンドを実現する時刻管理機能は C2A の動作を支える重要な機能である．

アプリケーション管理機能は，必須機能が呼び出す処理をコマンドとして用意する機能である．必須機能は呼び出す処理の塊をブロックコマンドとして管理するため，各処理をコマンドの形で呼び出せる仕組みを用意する必要があり，アプリケーション管理機能がこれを実現している．

イベント記録機能は搭載ソフトウェア動作中に発生する各種イベントを記録する機能であり，イベント処理機能が適切なイベント対応を実現するのに必要な補助を行う．

それぞれの機能の詳細を以下に述べる．

3.4.1 時刻管理機能

全ての必須機能は「予め用意された処理の塊を展開し，決まったタイミングで実行する」という同一のパターンを共有していることを述べたが，このパターンに基づいて必須機能を実現するためには「決まったタイミング」を明確に定義する必要がある．この役割を担うのが時刻管理機能である．

C2A の必須機能を動作させるためには，大小 2 つの時間粒度が必要になる．一つはモード維持機能が周期的に処理を実現するために必要になる一定の周期で繰り返される時間粒度であり，もう一つは，モード遷移機能やイベント処理機能がモード遷移やイベント処理を実現するために必要になるモード維持機能の周期を超えた，より大きな時間粒度である．

C2A の時刻管理機能では必須機能の動作に必要となる時刻情報を「周期的に繰り返す細かな時間粒度」と，「その周期を何度繰り返したかをカウントする大きな時間粒度」の 2 つの粒度で管理

する．細かな時間粒度を「ステップ」と呼び，大きな時間粒度を「サイクル」と呼ぶ．モード遷移機能やイベント処理機能が処理を実現するために利用する時間粒度がサイクルであり，モード維持機能が 1 サイクル内で様々な処理を実行するために利用する，1 つのサイクルをより細かい時間間隔で刻んだ時間粒度がステップである．我々が普段使用している秒と分の時間粒度に例えると，サイクルが分に相当し，ステップが秒に相当する形となる．

この 2 つの時間粒度の概念を図 3.13 に示す．

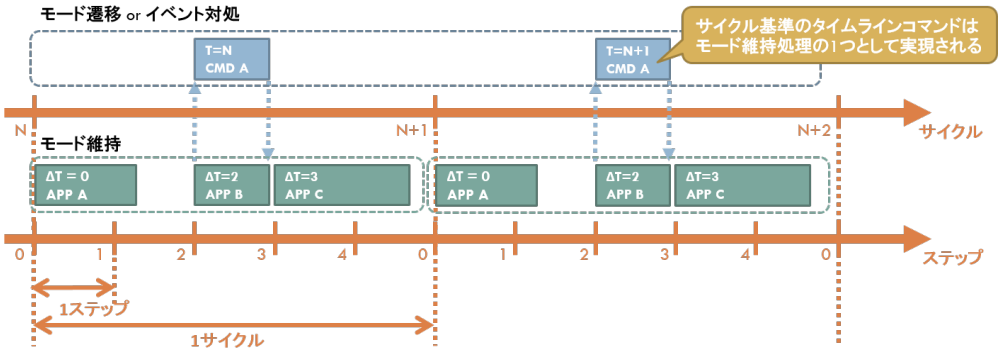


図 3.13　時刻管理機能の動作概念

C2A の必須機能はブロックコマンドを利用して管理される．このブロックコマンドは実行時には展開処理を経てタイムラインコマンドに変換され実行される．必須機能がステップとサイクルの 2 つの時間粒度を必要とするため，必須機能の各処理を実行するタイムラインコマンドにも，ステップに対応したものとサイクルに対応したものの 2 種類が必要になる．

モード維持機能は，モード維持に必要な処理が定義されたブロックコマンドを，各周期の初めにステップに対応したタイムラインコマンドとして展開することで周期的な処理を実現し，モード遷移機能やイベント処理機能は必要な処理が定義されたブロックコマンドを，サイクルに対応したタイムラインコマンドとして展開することでそれぞれの処理を実現する．

時刻管理機能は細かい時間粒度であるステップ毎に時刻情報を更新する必要がある．各ステップの間で C2A はモード維持に必要となる処理を実行しているため，ステップの計測はこの処理とは独立に実行する必要がある．これを実現するための実装は衛星の搭載計算機毎に異なるが，具体的な例としては多くの計算機にハードウェアとして付属するカウンタ機能を用いて時刻を計測し，ステップ幅毎に割り込みを発生させることで時刻情報を更新するといった実装が考えられる．C2A ではこの時刻管理機能のみがハードウェアに依存する箇所であり，この機能をそれぞれのハードウェアに合わせて実装することで，多数の衛星に C2A を適用することが可能となる．

3.4.2　アプリケーション管理機能

C2A によって用意される衛星の必須機能は全ての衛星が共有する，衛星の振る舞いを実現する枠組みであり，この枠組みの上で各衛星のミッションに応じた独自の振る舞いを実現するために

は，衛星毎にその振る舞いを実現する機能を実装し必須機能に登録して呼び出す必要がある．

例えば衛星の姿勢制御を行う場合には，まずそれに対応する機能を用意し，それを必須機能に登録して呼び出すことでその動作が実現する．より具体的には，センサからの信号を読み取る機能，姿勢決定を行う機能，制御出力を計算する機能，アクチュエータに出力を指令する機能といったものを用意し，それぞれをモード維持処理として必須機能にブロックコマンドの形で登録することで，モード維持機能がこれらの機能を呼び出して望んだ動作が実現する．

個々の衛星で必要となる独自の機能が必須機能から呼び出せるためには，衛星の個々の機能と必須機能との間のインターフェースを定める必要がある．C2A の必須機能はここまでで述べたとおり，呼び出す処理の塊をブロックコマンドとして定義し，それを展開することで，必要な処理を実行している．必須機能が処理の塊を定義するために用いるブロックコマンドは複数のコマンドを一つにまとめて実行する機能であり，処理の塊をブロックコマンドとして登録するためには，それぞれの処理がコマンドとして呼び出せなければならない．C2A では必須機能が呼び出す必要のある処理をコマンドとして用意することが必須機能と衛星個々の機能とをつなぐインターフェースであり，これを実現する補助機能がアプリケーション管理機能である．

必須機能が呼び出す処理

アプリケーション管理機能の詳細に入る前に，衛星の必須機能である，モード維持機能，モード遷移機能，イベント処理機能が呼び出す処理について整理する．

まず，モード維持機能とは，モードの維持に必要な処理を周期的に繰り返し呼び出す機能であり，このモード維持機能によって呼び出される処理が個々の衛星の振る舞いを実現している．C2A では，このモード維持機能によって呼び出される個々の処理のことを「アプリケーション」と呼ぶ．この言葉を用いると，モード維持機能とは，モードの維持に必要なアプリケーションを周期的に呼び出す機能であり，衛星の振る舞いはこのアプリケーションによって実現されている．

一方，モード遷移機能やイベント処理機能によって呼び出される処理は，

- アプリケーションに対する指示
- 必須機能に対する指示

の 2 種類に分類される．

アプリケーションに対する指示とは，モード遷移処理やイベント処理の例として挙げた機器の電源状態の変更処理等が対応する．この処理はモード維持処理によって実行されている電源管理アプリケーションへの指示である．指示を受けたアプリケーションがモード維持処理によって呼び出された際に指示に対応する機器の電源操作を実行することで，この動作が完了する．

必須機能に対する指示とは，イベント処理の例として挙げた，異常事態発生時に自動的にモードを変更し衛星の安全を確保するといった処理が対応する．この処理は，必須機能であるモード管理機能への指示である．指示を受けたモード管理機能が指示に対応するモード遷移を実行することでこの動作が完了する．

以上の整理の結果より，必須機能が呼び出す処理にはアプリケーションに関する物が多く含まれ

ることから，必須機能が呼び出す処理をコマンドとして整理するためには，アプリケーション自身の呼び出しとそれに対する指示を整理してコマンド化することが重要となることが分かる．これを実現するのがアプリケーション管理機能の役割である．

アプリケーションの構成

アプリケーション管理機能は必須機能が呼び出す処理をコマンドとして整理するため，アプリケーションを構成する要素を

- 属性
- イベント情報

の 2 種類のデータと，

- 実行内容
- 属性初期化
- 属性設定

の 3 種類の処理で定義する．この定義にあたっては衛星の機能モデル (Functional Model of Spacecraft / FMS)[38] の内容も参考にしている．

「属性」はアプリケーションの動作と関連する変数群であり，例えば姿勢決定系が決定した姿勢情報のようなアプリケーションの出力情報や，姿勢制御系のゲインのようなアプリケーションの動作を調整する入力情報を格納する．

「イベント情報」はアプリケーションがイベント処理機能に対して通知する信号である．アプリケーションが発生するイベント毎に一意の番号を割り当て，イベントの識別を可能にする．

「実行内容」はモード維持機能によって呼び出される衛星の振る舞いを実現する処理のことである．この処理は実行時に属性の値を参照でき，この値に応じて処理を変化させることができる．また，属性値の書き換えも可能であり，これによって処理の様々な状態を属性値に反映できる．

「属性初期化」はアプリケーションの実行前に属性値を適切な値に設定する処理である．通常では C2A の起動直後，アプリケーションの実行が開始される前に C2A によって一度だけ呼び出される．起動直後以外のタイミングで呼び出すことで，意図的に属性値を初期状態に再設定することも可能である．

「属性設定」は属性値を外部から書き換える処理である．この処理を呼び出して外部から属性値を書き換えることで，アプリケーションに指示を出すことが可能になる．

以上の構成要素の関係をまとめると図 3.14 のようになる．

C2A に基づく衛星搭載ソフトウェアの開発では，衛星毎の機能をアプリケーションとして実装することになるが，その際にはここで定義したアプリケーションの構成にしたがって，各要素を用意することになる．

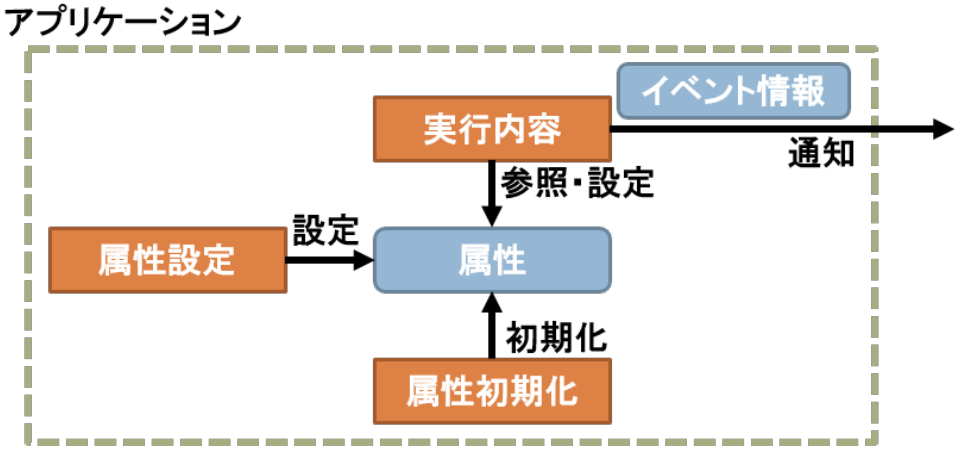


図 3.14　アプリケーションの構成要素

アプリケーションの管理

前節でアプリケーションの構成を定義したことによって，アプリケーションとそれに付随する要素が明確化された．アプリケーション管理機能はこの定義に基いて，必須機能が呼び出す処理を

- アプリケーション定義テーブル
- コマンド定義テーブル
- テレメトリ定義テーブル

を用いて管理する．

「アプリケーション管理テーブル」はアプリケーションとその初期化処理を管理するテーブルである．アプリケーション管理機能は，用意されたアプリケーション全てに一意の識別番号を割り振り管理を行う．アプリケーション管理テーブルはこの識別番号と対応するアプリケーションおよびその初期化処理を対応付けて管理する．

アプリケーション管理機能はアプリケーション管理テーブルに対して，識別番号を指定して初期化処理の呼び出しを行うコマンドとアプリケーションの呼び出しを行うコマンドを用意する．これらのコマンドを用いることで，アプリケーションとその初期化処理を必須機能から呼び出すことが可能になる．

「コマンド定義テーブル」はコマンドを管理するテーブルである．アプリケーション管理機能は，用意されたコマンド全てに一意の識別番号を割り振り管理を行う．コマンド定義テーブルはこの識別番号とコマンドとを対応付けて管理する．

コマンド定義テーブルに登録された処理は，地上局から識別番号を指定して実行することが可能な，いわゆる通常のコマンドとして動作する．C2A では必須機能に対するコマンドもこのテーブルに登録することで同様に呼び出しを実現する．

アプリケーションの動作状況を地上で確認するためには，アプリケーションの属性値を地上へ送信する「テレメトリ」が必要になる．C2A では属性値からテレメトリを生成する処理を「テレメトリ生成処理」と呼ぶ．通常，衛星には多種類のテレメトリが用意される．複数のアプリケーションの属性値を一つのテレメトリにまとめる場合も多いことから，テレメトリとアプリケーションは一対一の関係ではなく，個々のアプリケーションが備えるべき要素としてテレメトリ生成処理を規定することはできないが，この処理はアプリケーションの属性値と密接に関連する処理であることから，アプリケーション管理機能はテレメトリ生成処理を「テレメトリ定義テーブル」で管理する．

テレメトリ定義テーブルはテレメトリ生成処理を管理するテーブルである．アプリケーション管理機能は，用意されたテレメトリ生成処理全てに一意の識別番号を割り振り管理を行う．テレメトリ定義テーブルはこの識別番号とテレメトリ生成処理とを対応付けて管理するテーブルである．アプリケーション管理機能はテレメトリ定義テーブルに対して，識別番号を指定してテレメトリ生成処理の呼び出しを行うコマンドを用意する．これにより，各アプリケーションのテレメトリをコマンドによって生成することが可能になる．

以上のようなテーブルを用いてアプリケーションの構成要素を管理することで，衛星毎に実装されるアプリケーションとそれに対する指示をコマンドとして呼び出すことが可能となる．

加えて，アプリケーション管理機能は各テーブルの内容を書き換える機能をコマンドとして用意する．この機能は搭載ソフトウェアの柔軟な軌道上再構成を実現するために重要であり，この詳細については 3.6 節で議論する．

3.4.3 イベント記録機能

必須機能の一つであるイベント処理機能は，搭載ソフトウェアの実行中に発生したイベントに対して適切な対処を呼び出す機能であるが，この動作を実現するためには発生したイベントの履歴を記録することが重要となる．ここで説明するイベント記録機能はイベント処理機能に対してこの機能を提供するものである．

C2A では搭載ソフトウェア実行中に発生したイベントは直接イベント処理機能に引き渡されるのではなく，一旦イベント記録機能に引き渡され，全て記録される．イベント処理機能はイベント記録機能に記録されたイベント情報を参照して対処が必要な場合には適切な処理を呼び出して対処する．

このように搭載ソフトウェアの実行中に発生したイベントを直接イベント処理機能に渡すのではなく，一度イベント記録機能に記録してイベント処理を行う理由には，

- 高度なイベント処理の実現
- 異常発生時の原因解明

の 2 つが存在する．

高度なイベント処理の実現の観点では，イベント発生の時間履歴が対処の選択に重要な意味を持つ場合が多いことが履歴を記録する理由として挙げられる．履歴を記録することで得られる利点を

示す簡単な例としては，例えばセンサの出力値の異常を判定する場合に，異常値が一度だけ記録された場合は過渡的な変化やノイズによるゆらぎの影響として無視するが，同一の異常値が複数回連続で記録された場合は異常発生と判断して対処を行うといったものが挙げられる．このような機能を実現するためにはイベントの発生履歴を記録することが必要となる．さらに，イベントの発生履歴を参照することで，複数の異常記録から原因を判断し対処を行うといったより高度な処理も実現可能である．

異常発生時の原因解明の観点では，搭載ソフトウェア動作中のイベントの発生履歴が地上での異常原因調査に重要となる場合が多いことが履歴を記録する理由として挙げられる．イベント処理機能に高度なイベント判断を実装しない場合であっても，運用中に衛星の動作に何らかの異常が発生した場合は地上でその原因調査が行われる．この際にイベント記録機能によって記録されたイベントの発生履歴は，異常発生時の衛星の挙動を確認する上で重要な手がかりとなる．

図 3.15 にイベント記録機能とイベント処理機能を合わせた動作の概略図を示す．

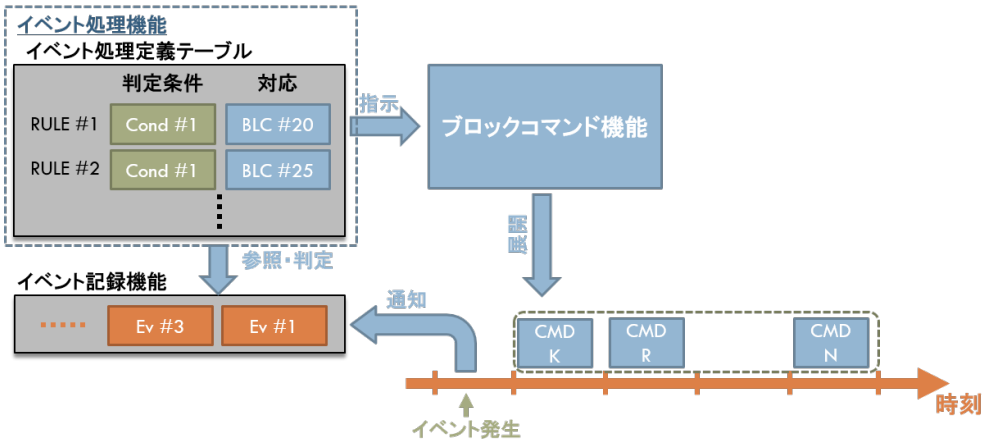


図 3.15　イベント記録機能とイベント処理機能の動作概念

3.5　C2A の全体構成

C2A の必須機能と補助機能をまとめた全体像を図 3.16 に示す．

ここまでで述べてきたように，搭載ソフトウェア全体は衛星に共通の必須機能と補助機能からなる C2A のコア機能部と個々の衛星の目的に応じた機能を実現するアプリケーションからなる衛星固有部の 2 つに明確に分離されている．

図中の緑色の矢印は C2A を適用した開発の流れを示し，青色の矢印は C2A の実行時の動作を示している．

C2A を適用した開発ではまず，各アプリケーションの開発を行い，それらをアプリケーション管理機能の「アプリケーション定義テーブル」「コマンド定義テーブル」「テレメトリ定義テーブル」に登録する．この登録によって，アプリケーションの各機能をコマンドとして呼び出すことが

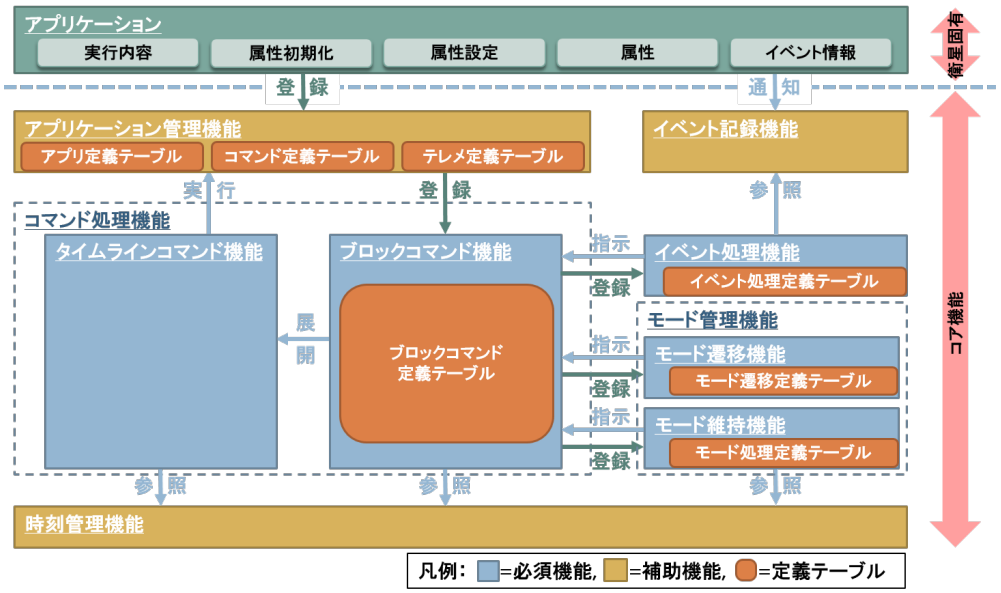


図 3.16　C2A の全体構成

可能となり，必須機能が呼び出す処理の塊をブロックコマンドとして「ブロックコマンド定義テーブル」に登録することが可能となる．ブロックコマンドとして登録された各種処理の塊は，モード管理機能とイベント処理機能の「モード処理定義テーブル」「モード遷移定義テーブル」「イベント処理定義テーブル」にそれぞれ登録され，C2A が動作する環境が整う．

C2A の実行ではまず，モード管理機能やイベント処理機能が補助機能である時刻管理機能やイベント記録機能の情報を参照し，状況に応じた処理の塊を展開するようブロックコマンド機能に指示を出す．ブロックコマンド機能は指示されたブロックコマンドを時刻情報を参照しながらタイムラインコマンドとして展開し，タイムラインコマンド機能がこれらのコマンドを指定時刻に実行していくことで衛星全体の動作が実現される．

C2A では衛星の動作は全て各種定義テーブルによって実現されている．この定義テーブルを軌道上で書き換える機能を備えることで，軌道上でも地上と同様の手順・粒度で衛星の動作の柔軟な再構成が可能となっている．これに関しては次節でより詳細な議論を行う．

3.6　軌道上再構成能力の実現

C2A では衛星搭載ソフトウェアの必須機能が同じ処理のパターンを共有している点に着目し，全ての必須機能をコマンド処理機能，特にブロックコマンドに集約して実現する．必須機能をブロックコマンドに集約する理由として 3.3 節では，パターンの同一性だけではなく，柔軟な再構成能力の獲得を挙げた．本節では C2A が必須機能をブロックコマンドに集約することで具体的にどのような軌道上再構成能力が獲得されるかを説明する．

C2A では全ての必須機能をブロックコマンドを中心に実装したことで，

- パラメータ変更
- ブロックコマンド変更
- 各種定義テーブル変更
- メモリ部分書き換え
- メモリ全書き換え

の 5 段階の軌道上再構成が可能となっている．以下それぞれの再構成能力の詳細について詳細を述べる．

3.6.1 パラメータ変更

パラメータ変更とは，地上からのコマンド操作によって衛星の動作に関連するパラメータを調整する再構成手段である．具体的には，搭載カメラの露出時間の調整や姿勢制御系の制御ゲインの調整といったものが挙げられる．

C2A ではモード維持機能がモード維持に必要なアプリケーションを周期的に呼び出すことで衛星の動作を実現している．パラメータ変更はアプリケーションの属性値を変更することに対応し，アプリケーションのコマンドを呼び出すことで実現される．

この再構成手段は従来の衛星でも通常用いられている一般的な内容である．

3.6.2 ブロックコマンド変更

C2A ではブロックコマンドの内容を変更することが再構成手段の 1 つとなる．C2A ではそれぞれの必須機能が呼び出す処理の塊をブロックコマンドとして管理しているため，処理の塊に対応するブロックコマンドの内容を変更することで必須機能によって呼び出される処理の内容を変更することが可能になる．

より具体的には，ブロックコマンドの変更により図 3.17 に示すように必須機能から呼び出される処理の

- 実行タイミングの調整
- 追加や削除

が実現できる．

処理の実行タイミングの調整はブロックコマンドを構成している各コマンドの相対実行時刻情報を編集することで可能となる．ブロックコマンド定義テーブルを操作して各コマンドの相対実行時刻を書き換えることで処理と処理との間隔や処理の実行順序を調整することができる．

処理の追加と削除はブロックコマンドの定義内容を編集することで可能になる．ブロックコマンド定義テーブルを編集することで，展開されるブロックコマンドの中に新たな処理を加えたり，逆に特定の処理を取り除くことができる．

以下ではモード管理機能とイベント処理機能それぞれに対し，ブロックコマンドの内容を変更す

ることでどのような再構成が可能となるかまとめる．

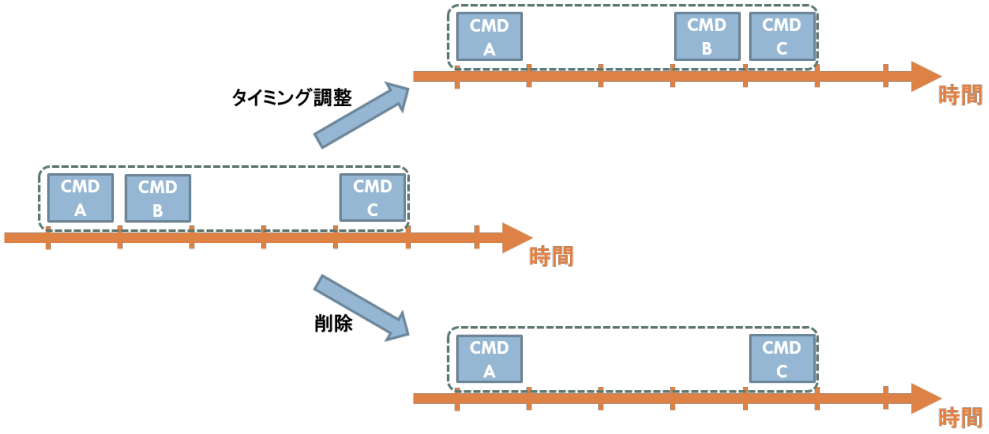


図 3.17　ブロックコマンド変更による再構成の例

モード管理機能

モード維持機能は，各モードで 1 周期の間に実行するアプリケーションの塊をブロックコマンドとして管理している．このため，対応するブロックコマンドを編集することで，1 周期の間に実行されるアプリケーションの実行タイミングを調整したり，1 周期の中で呼び出されるアプリケーションの追加や削除が可能になる．例えば，1 周期の中で実行されるアプリケーション同士の実行間隔が近く，直前の処理が次の処理の開始までに終了しない場合にはブロックコマンドの相対実行時刻を変更してアプリケーションの実行タイミングを調整することが可能である．

モード遷移機能は遷移元のモードから遷移先のモードへ遷移する際に実行する処理の塊をブロックコマンドとして管理している．このため，対応するブロックコマンドを編集することで，遷移処理の実行タイミングを調整したり，遷移処理として呼び出す処理の追加や削除が可能になる．例えば，モード遷移処理として機器の電源 ON と初期化の 2 つの処理を実施する場合に，両者の実行間隔が近すぎて電源 ON 操作の完了前に初期化処理が実行されてしまう場合にはブロックコマンドの相対実行時刻を変更することで実行間隔を調整可能である．また，モード遷移処理に例えばある機器の電源 ON といった操作を加えたい場合にはブロックコマンドにその処理に対応するコマンドを加えることで処理を追加できる．

イベント処理機能

イベント処理機能は，イベント発生時に呼び出す処理の塊をブロックコマンドとして管理している．このため，対応するブロックコマンドを編集することで，イベント発生時に呼び出される処理の実行タイミングを調整したり，呼び出される処理の追加や削除が可能になる．例えば，発生電力低下時の対処として不必要な機器を OFF にする操作を考えると，ブロックコマンドとして登録されたコマンドの相対時刻を変更することでそれぞれの機器を OFF にするタイミングの調整が可能

であり，またブロックコマンドに新たに機器を OFF にするコマンドを追加することで OFF にする機器を追加したりといった変更が可能である．

3.6.3　各種定義テーブル変更

C2A の必須機能は展開する処理の塊をブロックコマンドとして定義すると同時に，展開するブロックコマンドを状況に応じて選択するために必要な情報を各種の定義テーブルとして管理している．モード管理機能ではモード維持処理定義テーブルとモード遷移定義テーブルが，イベント処理機能ではイベント処理定義テーブルがこれにあたる．

3.3.2 節や 3.3.3 節で各必須機能をブロックコマンドを中心に実現する仕組みを説明する過程で述べたとおり，C2A はそれぞれの定義テーブルの内容をコマンド操作で変更できる機能を備えている．この機能を利用して定義テーブルの内容を書き換えることで搭載ソフトウェアの再構成が可能となっている．

以下では，モード管理機能とイベント処理機能において定義テーブルの書き換えを行うことでどのような再構成が実現されるかその詳細について述べる．

モード管理機能機能

モード維持機能はモード維持処理定義テーブルで衛星に定義されたモードとそれぞれのモードで周期的に実行するアプリケーションを定義したブロックコマンド番号を管理している．このテーブルの内容を書き換えることで，各モードで呼び出されるブロックコマンドを別のブロックコマンドへ切り替えることが可能となる．ただし，各モードで実行されるアプリケーションの修正はモード維持処理管理テーブルを書き換えなくとも，前節で述べた通りブロックコマンドの内容を変更することで実現できるため，この機能単体で得られる効果は大きくない．モード維持処理定義テーブルの書き換えは，次に述べるモード遷移定義テーブルの書き換えと合わせて実施することで，より大きな効果が生まれる．

モード遷移機能はモード遷移定義テーブルで遷移元のモードと遷移先のモードの組と遷移処理を定義したブロックコマンド番号との対応関係を管理している．このテーブルを書き換えることで衛星のモード遷移図を軌道上で修正することが可能となる．

まず，モード処理定義テーブル上で既に定義されたブロックコマンド番号を書き換えることで，モード遷移実行時に呼び出すブロックコマンドを別のブロックコマンドに切り替えることが可能となる．これによって遷移時に呼び出される処理の内容を別のブロックコマンドで定義された内容に変更することができるが，この変更自体はもともと遷移処理を定義していたブロックコマンドを直接書き換えることでも実現できる内容である．

モード管理機能の再構成で重要になるのは，モード遷移定義テーブル上で打ち上げ時点では未定義となっていた箇所にブロックコマンド番号を書き込んだ場合である．モード遷移定義テーブルで未定義となっている箇所は対応する遷移元モードから遷移先モードへの遷移が存在しない，すなわち遷移が禁止されている状態に対応するが，この内容を書き換えて新たにブロックコマンド番号を

割り当てることで，図 3.18 に示すような，これまで実行できなかった遷移が可能となる．逆にこれまで定義されていたブロックコマンド番号を未定義に設定することで，特定のモード遷移を無効化することも可能である．これはモード遷移定義テーブルを書き換えることで，衛星のモード遷移図を修正できることを意味している．

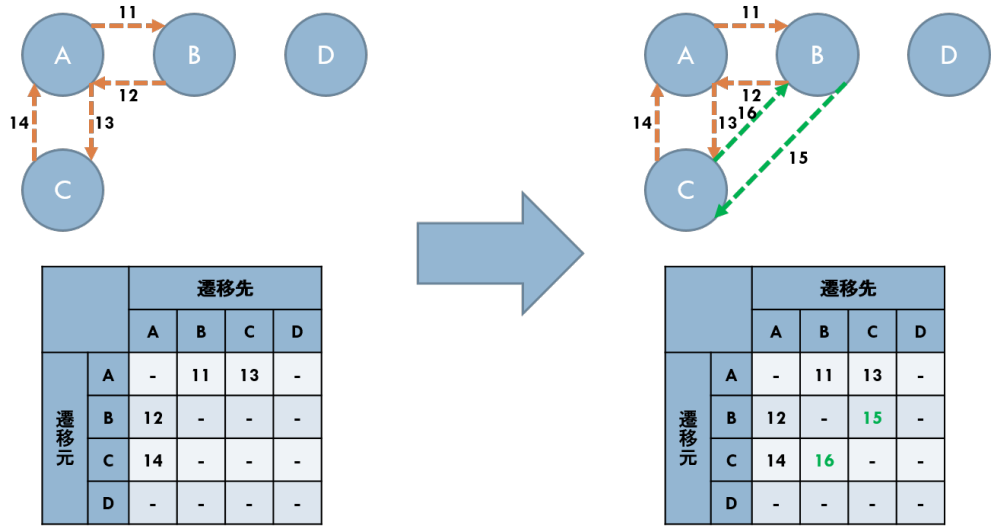


図 3.18　再構成によるモード遷移追加の例

さらに，予め打ち上げ前に衛星のモードとして通常は使用しない予備的なモードを用意しておくことで，打ち上げ後の衛星に新たなモードを定義することも可能となる．打ち上げ直後はどのモードからも遷移できないモードを予備として用意しておき，打ち上げ後に新たなモード定義が必要となった場合にそのモードで周期的に実行する処理とそのモードへの遷移処理とをブロックコマンドとして定義し，モード維持処理定義テーブルとモード遷移定義テーブルにそれらのブロックコマンド番号を登録することで図 3.19 に示すような新たなモードが定義できる．

イベント処理機能

イベント処理機能はイベント処理定義テーブルでイベントの対処条件とそれに応じて呼び出す処理を定義したブロックコマンド番号との対応関係を管理している．このテーブルを書き換えることで衛星のイベントに対する対処を変更することが可能になる．

イベント処理テーブルのブロックコマンド番号を変更することで，イベント発生時に呼び出すブロックコマンドを別のものに入れ替えることが可能となる．ただし，この変更は予め用意されていたブロックコマンドを直接編集することでも実現可能である．

対処条件を書き換えることで，ブロックコマンドを呼び出すイベントを変更することが可能になる．また，モード管理機能の場合と同様に，イベント処理定義テーブルに打ち上げの段階では使用しない予備的なイベント対処を用意しておき，対処条件とブロックコマンド番号を打ち上げ後に書き込むことで，打ち上げ前の段階では対処を予定していなかったイベントに対する対処を追加した

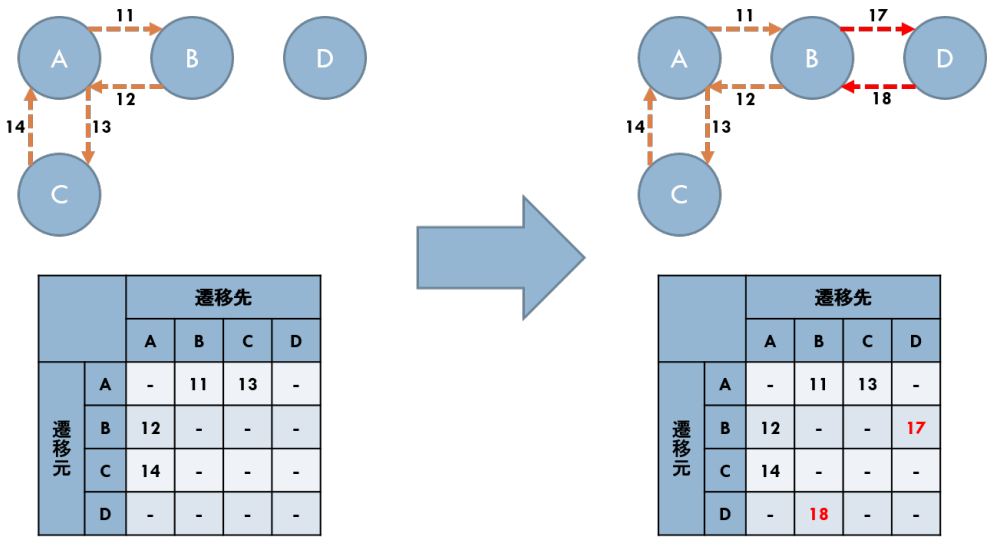


図 3.19　再構成によるモード追加の例

り，逆に打ち上げ後に対処不要と判断したイベント対応を削除したりといった変更も可能になる．

3.6.4　メモリ部分書き換え

ここまでで述べた再構成機能では対処できない事態に対しては衛星の搭載ソフトウェアを直接書き換える，メモリ書き換えによる再構成が必要となる．C2A ではメモリ書き換えについても「部分書き換え」と「全書き換え」の 2 種類が存在する．

打ち上げ後に修正や追加が必要な機能のみをメモリ書き換えで追加し，もともと衛星に搭載されている C2A から呼び出して実行することを C2A では部分書き換えと呼ぶ．

C2A ではそれぞれの必須機能が処理の塊をブロックコマンドとして定義している．このため，打ち上げ後であっても，コマンドとして呼び出せる形で衛星に処理を追加できれば C2A 自体を変更しなくても衛星の搭載ソフトウェアに機能を追加することが可能である．

C2A では各必須機能が呼び出す処理は全てコマンドとして呼び出せる形で用意する必要があり，これを実現しているのがアプリケーション管理機能であった．アプリケーション管理機能は，アプリケーションに関連する要素をアプリケーション定義テーブル，コマンド定義テーブル，テレメトリ定義テーブルの 3 つのテーブルで管理し，C2A の必須機能がそれらをコマンドとして呼び出す仕組みを実現している．

アプリケーション管理機能は，各テーブルの内容を書き換える機能を備えており，打ち上げ後も，アプリケーションやコマンド処理の内容を衛星のメモリに書き込んだ上で，各テーブルにそれらを呼び出すのに必要な情報を書き込むことで，追加した処理を C2A から呼び出すことが可能である．

これは，C2A によってアプリケーションとそれに付随する要素が明確化されたことで可能となる再構成である．

3.6.5　メモリ全書き換え

メモリ全書き換えは，部分書き換えとは異なり，衛星に搭載したプログラムの内容全てを書き換える操作である．この操作は衛星に C2A に備わる搭載ソフトウェアの再構成能力では解決できないような事態が生じた場合に使用される最後の手段となる．

3.6.6　従来の搭載ソフトウェアとの比較

ここまで述べてきたように，C2A では必須機能をブロックコマンドに集約して実装することで，複数の再構成能力を獲得している．ここでは，従来の衛星の再構成能力と C2A が実現する再構成能力とを比較し，その優位性を明らかにする．

C2A に基づいて実現される搭載ソフトウェアが従来の搭載ソフトウェアとくらべて再構成の観点で優れている点は，再構成の選択肢が広く，状況に応じて柔軟な再構成を実現可能な点にある．C2A では，ここまでに説明したように必須機能をブロックコマンドに集約して実装することで，表 3.2 に示す 5 段階で構成される柔軟な再構成手段を実現している．

表 3.2　C2A による再構成能力の階層化

レベル	再構成の種類	実施リスク
0	パラメータ変更	低
1	ブロックコマンド変更	
2	定義テーブル変更	中
3	メモリ部分書き換え	
4	メモリ全書き換え	高

図 3.20 に従来の衛星搭載ソフトウェアでの再構成と C2A を用いる衛星搭載ソフトウェアでの再構成との比較を示す．従来の衛星搭載ソフトウェアは，打ち上げ後にその動作を変更する手段はコマンドによるパラメータ調整のみに限定されており，これで対処が不可能な場合は搭載プログラムの書き換えを行うしかないという 2 択の状態にあった．パラメータ変更に対して搭載ソフトウェアの書き換えは当然対応範囲は広いものの，同時に事前準備などの実施負荷が非常に高い方法であり，この 2 つの間には大きな隔たりがあった．これに対して C2A は必須機能をブロックコマンドに集約して実装することで，従来はメモリ書き換えで実現するしかなかった再構成の一部をブロックコマンドに関連する操作で実施可能とし，柔軟な再構成手段を実現している．

従来のメモリ書き換えによる再構成と C2A によって実現される再構成を比較した場合に，C2A による再構成の利点として，

- 運用との親和性の高さ
- 理解のしやすさ

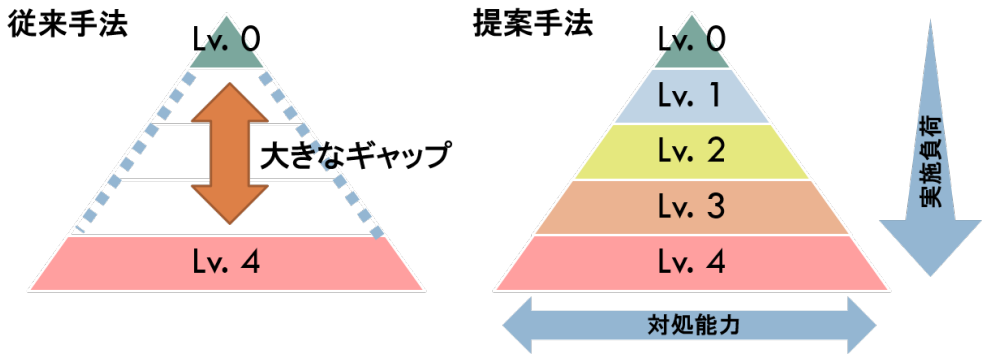


図 3.20　再構成能力の階層化

- 実施のしやすさ

の 3 点が挙げられる．

まず，C2A によって用意される再構成にはメモリ書き換えと比べて運用との親和性が高いという利点がある．C2A ではブロックコマンドの書き換えという通常の運用で使用する操作と同じ操作で衛星の挙動を再構成できる．これは，メモリ書き換えという通常の運用では利用しない特殊な手順で再構成を実現する従来の再構成方式と比較して，運用との親和性の高い方式と言うことができる．これは，衛星の必須機能の動作をブロックコマンドに集約した事によって得られた利点である．

次に，C2A で実現される再構成にはメモリの書き換えに比べて人が理解しやすいという利点がある．搭載計算機のメモリを直接書き換えて実施する再構成が高リスクと考えられる原因は，この操作とそれによって得られる効果が直感的に理解できないためと考えられる．メモリの書き換えを実施する際には地上から搭載計算機のメモリに書き込む内容を送信することになるが，この内容は機械語やデータで構成される単なる数値の羅列であり，人が内容を見ても理解することは困難である．このため，メモリ書き換えの操作は確認が行いづらく，実施リスクが高い操作と考えられている．これに対して C2A が実現する再構成は，パラメータ変更とメモリ書き換えとの間に，ブロックコマンドの編集と定義テーブルの編集による再構成を加えている．これらを実施する際の操作はメモリ書き換えとは異なり通常のコマンドで構成されるため，C2A の仕組みを理解していればその内容も容易に理解できる形となっている．この再構成の見通しの良さは C2A という明確な枠組みを構築したことによって得られる利点である．

また，C2A での再構成の実施しやすさについては，C2A で行う再構成には実施する際にアップリンクが必要なデータ量が少ないという利点がある．C2A ではブロックコマンドの書き換えや定義テーブルの書き換えによって衛星の挙動を再構成することが可能となっている．ブロックコマンドの変更は呼び出す処理を指定するものであり，定義テーブルの変更は呼び出すブロックコマンド番号を変更することで行われる．これらは衛星内部に既に用意された処理を呼び出すコマンドであり，実行する処理自体をアップリンクするメモリ書き換えに対して極めて少ないデータ量で実現で

きる．衛星と地上との通信は遠距離の無線通信であり，通信の速度は一般にかなり制限されているため，これまではメモリ書き換えで実現する必要のあった処理の一部が少ないデータ量のアップリンクで実現できることは大きな利点である．

以上のように，提案する C2A によって見通しが良く，柔軟な再構成が可能な衛星の搭載ソフトウェアが実現される．

第 4 章

C2A の実装

前章ではまず，これまでの衛星搭載ソフトェアが実現してきた処理の整理を通じて C2A の必須機能と補助機能を導出した．その後，必須機能が同一の処理パターンを共有している点に着目し，必須機能をブロックコマンドに集約することで，全体の見通しの良さと高い再構成能力を兼ね備える C2A の概念を提示した．

前章で述べた通り，C2A の機能は

- 必須機能
  - コマンド処理機能
    - リアルタイムコマンド機能
    - タイムラインコマンド機能
    - シングルコマンド機能
    - ブロックコマンド機能
  - モード管理機能
    - モード維持機能
    - モード遷移機能
  - イベント処理機能
- 補助機能
  - 時刻管理機能
  - アプリケーション管理機能
  - イベント記録機能

で構成される．

本章では前章で提示した C2A の概念を具体化し，C2A の各機能が満たすべき要求として整理することで，それぞれの機能が C2A の中で担う役割を明確化する．

本章では，まず初めに必須機能を支える役割を担う各補助機能に対する要求をまとめ，その後に補助機能を利用して機能を実現している各必須機能に対する要求をまとめる．前章が C2A を導出するためにトップダウン的な構成をとったのに対し，本章は各機能を積み上げて C2A を構築する

ボトムアップ的な構成となっている．前章の内容が C2A の基本思想と動作のイメージを理解するのに適した形式であるのに対して，本章の内容は実際に C2A を衛星搭載ソフトウェアとして実装する場合に適した形式である．

本章では，各機能が満たすべき要求を

- 必要な処理
- 必要なコマンド
- 必要なテレメトリ
- 衛星毎に調整するパラメータ

の 4 項目に整理して提示する．

「必要な処理」では各機能が衛星の内部で実現すべき機能，「必要なコマンド」では各機能を地上から操作するために用意すべきコマンド，「必要なテレメトリ」では各機能の動作確認やコマンドによる操作を行うために用意すべきテレメトリをそれぞれ整理する．また，「衛星毎に調整するパラメータ」では実際に C2A を適用して各衛星の搭載ソフトウェアを構築する場合に，各衛星の特性・実情に合わせて搭載ソフトウェアの動作を調整するために，C2A が備えるべきパラメータを整理する．

本章ではコマンド処理機能については，C2A を実装する上で重要となるタイムラインコマンド機能とブロックコマンド機能に絞ってその要求を明確化する．また，モード管理機能については前章ではモード維持機能とモード遷移機能を独立して扱ったが，本章は両者をまとめてモード管理機能として要求を明確化する．

本章で示した各機能の要求を具体化した，より実際の実装に近いソースコードと対応したレベルでの C2A の構成については，本論文末尾に付録として収録する．必要に応じて適宜参照されたい．

4.1 補助機能の実装

4.1.1 時刻管理機能

必要な処理

1. 時刻情報の保持
   - 時刻管理機能は C2A の動作の基本となる時刻情報を管理する機能を備える．
   - 時刻情報はステップとサイクルの 2 つの時間粒度で構成される．
   - 時刻情報は時刻管理機能自身の操作かコマンド操作によってのみ変更可能とする．
   - 時刻情報は C2A のその他の機能と各アプリケーションから自由に参照できる．
2. 時刻情報の初期化
   - 時刻管理機能は C2A の動作開始時に自身が保持する時刻情報を初期化する機能を備える．
   - 本機能は値はステップ，サイクルともに初期値として 0 を設定する．

3. 時刻情報の更新
    - 時刻管理機能は自身が保持する時刻情報をステップ幅毎に更新する機能を備える．
    - 本機能はステップ幅に相当する時間が経過する毎に自身が保持するステップ値を 1 加算して更新する．
    - 本機能は更新したステップ値がステップ値の上限に達した場合は，サイクル値を 1 加算して更新し，ステップ値を 0 に再設定する．
    - 本機能はサイクル更新時にサイクル値が上限に達する場合はサイクル数を 0 に設定する．

必要なコマンド

1. サイクル値の設定
    - 時刻管理機能はサイクル値を指定された値に設定するコマンドを備える．
    - 本コマンドは指定されたサイクル値がサイクル値の上限を超過する場合はサイクル値の設定を行わず，異常を通知する．

必要なテレメトリ

1. 現在のサイクル値
    - 時刻管理機能が保持するサイクル値はテレメトリとして用意する．

衛星毎に調整するパラメータ

1. ステップと実時間との対応関係
    - 1 ステップの時間幅を実時間の何秒に対応付けるかは衛星毎に調整可能なパラメータとする．
    - この値は衛星搭載ソフトウェアに要求される時間の計測精度や C2A を動作させる計算機の処理能力によって決定すべき値である．
2. サイクルとステップとの対応関係
    - 1 サイクルを何ステップに対応づけるかは衛星毎に調整可能なパラメータとする．
    - この値はモード維持処理の 1 周期を決定する値であり，最も高頻度に実行すべきアプリケーションの実行周期によって決定すべき値である．
3. サイクルの最大値
    - サイクルが取りうる最大値は衛星毎の調整可能なパラメータとする．
    - この値は衛星の運用期間および運用方針によって決定すべき値である．

4.1.2 アプリケーション管理機能

必要な処理

1. アプリケーションの管理
   - アプリケーション管理機能は C2A 上で動作するアプリケーションを管理する機能を備える．
   - アプリケーションの管理はアプリケーション定義テーブルを用いて実現される．
   - アプリケーション定義テーブルはアプリケーションの ID とアプリケーションの初期化処理・周期処理との対応関係を管理する．
   - アプリケーション定義テーブルの内容はアプリケーション管理機能自身の操作かコマンド操作によってのみ変更可能とする．
2. アプリケーション定義テーブルの初期化
   - アプリケーション管理機能は C2A の動作開始時にアプリケーション定義テーブルの内容を初期化する機能を備える．
   - 本機能は搭載ソフトウェアに用意されたアプリケーションをアプリケーション定義テーブルに登録し，呼び出し可能な状態を用意する．
   - 本機能は C2A の動作開始時点でアプリケーションが定義されないアプリケーション ID については，その内容を未定義に設定する．
3. 起動時の全アプリケーション初期化
   - アプリケーション管理機能はアプリケーション定義テーブルの初期化完了後，登録された各アプリケーションの初期化を実行する機能を備える．
   - 本機能はアプリケーション定義テーブルに登録された各アプリケーションの初期化処理を呼び出すことで実現される．
4. コマンドの管理
   - アプリケーション管理機能は衛星に対する全コマンドを管理する機能を備える．
   - コマンドの管理はコマンド定義テーブルを用いて実現される．
   - コマンド定義テーブルはコマンドの ID とコマンドとして呼び出す処理との対応関係を管理する．
   - コマンド定義テーブルの内容はアプリケーション管理機能自身の操作かコマンド操作によってのみ変更可能とする．
5. コマンド定義テーブルの初期化
   - アプリケーション管理機能は C2A の動作開始時にコマンド定義テーブルの内容を初期化する機能を備える．
   - 本機能は搭載ソフトウェアに用意されたコマンドをコマンド定義テーブルに登録し，呼び出し可能な状態を用意する．
   - 本機能は C2A の動作開始時点でコマンドが定義されないコマンド ID については，そ

の内容を未定義に設定する．

6. コマンド実行
    - アプリケーション管理機能はコマンド呼び出しに対応してコマンドを実行する機能を備える．
    - 呼び出すコマンドはコマンド ID で指定する．
    - コマンドを呼び出す際には，コマンド ID と合わせてコマンドに引き渡す追加のパラメータを指定できる．
    - コマンドの実行はコマンド定義テーブルを参照し，指定されたコマンド ID に対応するコマンドを呼び出すことで実現する．呼び出しの際，追加のパラメータは実行するコマンドに引き渡される．
7. テレメトリ定義テーブルの保持
    - アプリケーション管理機能は地上に送信するテレメトリを管理する機能を備える．
    - テレメトリの管理はテレメトリ定義テーブルを用いて実現される．
    - テレメトリ定義テーブルはテレメトリの ID とテレメトリを生成する処理との対応関係を管理する．
    - テレメトリ定義テーブルの内容はアプリケーション管理機能自身の操作かコマンド操作によってのみ変更可能とする．
8. テレメトリ定義テーブルの初期化
    - アプリケーション管理機能は C2A の動作開始時にテレメトリ定義テーブルの内容を初期化する機能を備える．
    - 本機能は搭載ソフトウェアに用意されたテレメトリ生成処理をテレメトリ定義テーブルに登録し，呼び出し可能な状態を用意する．
    - 本機能は C2A の動作開始時点でテレメトリ生成処理が定義されないテレメトリ ID については，その内容を未定義に設定する．

必要なコマンド

1. アプリケーションの登録
    - アプリケーション管理機能はアプリケーション定義テーブルにアプリケーションの初期化処理と周期処理を登録するコマンドを備える．
    - アプリケーション定義テーブル上の登録先はアプリケーション ID で指定する．
    - 本コマンドは指定されたアプリケーション ID に対応するアプリケーション定義テーブルの内容を指定された内容に変更する．
    - 本コマンドは指定されたアプリケーション ID がアプリケーションの登録上限数を超過する場合，アプリケーションの追加を行わず異常を通知する．
2. アプリケーションの初期化
    - アプリケーション管理機能は指定されたアプリケーションの初期化処理を呼び出すコマ

ンドを備える．
    - 初期化を行うアプリケーションはアプリケーション ID で指定する．
    - 本コマンドはアプリケーション定義テーブルを参照し，指定されたアプリケーション ID に対応する初期化処理を呼び出す．
    - 本コマンドは指定されたアプリケーション ID に対応する初期化処理が未定義の場合，初期化処理を呼び出さず，異常を通知する．
    - 本コマンドは指定されたアプリケーション ID がアプリケーションの登録上限数を超過する場合，初期化処理を呼び出さず，異常を通知する．
3. アプリケーションの実行
    - アプリケーション管理機能は指定されたアプリケーションを実行するコマンドを備える．
    - 実行するアプリケーションはアプリケーション ID で指定する．
    - 本コマンドはアプリケーション定義テーブルを参照し，指定されたアプリケーション ID に対応するアプリケーションを実行する．
    - 本コマンドは指定されたアプリケーション ID に対応するアプリケーションが未定義の場合，アプリケーションの実行を行わず，異常を通知する．
    - 本コマンドは指定されたアプリケーション ID がアプリケーションの登録上限数を超過する場合，アプリケーションの実行を行わず，異常を通知する．
4. コマンドの登録
    - アプリケーション管理機能はコマンド定義テーブルにコマンド処理を登録するコマンドを備える．
    - コマンド定義テーブル上の登録先はコマンド ID で指定する．
    - 本コマンドは指定されたコマンド ID に対応するコマンド定義テーブルの内容を指定された内容に変更する．
    - 本コマンドは指定されたコマンド ID がコマンドの登録上限数を超過する場合，コマンドの登録を行わず，異常を通知する．
5. テレメトリの登録
    - アプリケーション管理機能はテレメトリ定義テーブルにテレメトリ生成処理を登録するコマンドを備える．
    - テレメトリ定義テーブル上の登録先はテレメトリ ID で指定する．
    - 本コマンドは指定されたテレメトリ ID に対応するコマンド定義テーブルの内容を指定された内容に変更する．
    - 本コマンドは指定されたテレメトリ ID がテレメトリの登録上限数を超過する場合，テレメトリの登録を行わず，異常を通知する．
6. テレメトリの生成
    - アプリケーション管理機能は指定されたテレメトリを生成するコマンドを備える．
    - 生成するテレメトリはテレメトリ ID で指定する．

- 本コマンドはテレメトリ定義テーブルを参照し，指定されたテレメトリ ID に対応するテレメトリ生成処理を実行する．
- 本コマンドは指定されたテレメトリ ID に対応するテレメトリ生成処理が未定義の場合，テレメトリ生成処理の実行を行わず，異常を通知する．
- 本コマンドは指定されたテレメトリ ID がテレメトリの登録上限数を超過する場合，テレメトリ生成処理の実行を行わず，異常を通知する．

必要なテレメトリ

1. アプリケーション定義テーブルの内容
   - アプリケーション管理機能はアプリケーション定義テーブルの内容をテレメトリとして用意する．
2. コマンド定義テーブルの内容
   - アプリケーション管理機能はコマンド定義テーブルの内容をテレメトリとして用意する．
3. テレメトリ定義テーブルの内容
   - アプリケーション管理機能はテレメトリ定義テーブルの内容をテレメトリとして用意する．

衛星毎に調整するパラメータ

1. アプリケーションの登録上限値
   - アプリケーション定義テーブルに登録できるアプリケーションの上限値は衛星毎に調整可能なパラメータとする．
   - この値は衛星のミッションを達成するために必要となるアプリケーションの総数に基いて決定すべき値である．
2. コマンドの登録上限値
   - コマンド定義テーブルに登録できるコマンドの上限値は衛星毎に調整可能なパラメータとする．
   - この値はアプリケーションによって必要となるコマンドの総数に基いて決定すべき値である．
3. テレメトリの登録上限値
   - テレメトリ定義テーブルに登録できるテレメトリの上限値は衛星毎に調整可能なパラメータとする．
   - この値は衛星運用に必要となるテレメトリの総数に基いて決定すべき値である．

4.1.3 イベント記録機能

必要な処理

1. イベント発生履歴の保持
   - イベント記録機能は C2A の動作中に発生するイベントの発生履歴を保持する機能を備える．
   - イベント発生履歴はイベント発生情報の履歴と総イベント発生数で構成される．
   - イベント発生情報は発生したイベントの種類と発生時刻で構成される．
   - イベント発生履歴の内容はイベント記録機能自身の操作かコマンドによる操作によってのみ変更可能とする．
   - 本機能が保持するイベント発生履歴は，本機能以外にイベント処理機能からも参照可能とする．
2. イベント発生履歴の初期化
   - イベント記録機能は C2A の動作開始時にイベント発生履歴を初期化する機能を備える．
   - 本機能はイベント発生履歴の内容を空に設定し，総イベント発生数を 0 に設定する．
3. イベント発生情報の登録
   - イベント記録機能はイベント発生履歴にイベント発生情報を登録する機能を備える．
   - 本機能はイベント発生履歴にイベント発生情報を登録し，同時に総イベント発生数の値を 1 加算する．
   - 本機能はイベントを登録することでイベント発生履歴に記録されたイベント数が記録上限に達する場合は，代わりに記録上限に達したことを示すイベントを登録し，以降のイベント登録は受け付けない．
   - 本機能は C2A 上の全ての機能・アプリケーションから実行できる．

必要なコマンド

1. イベント発生履歴のクリア
   - イベント管理機能はイベント履歴をクリアするコマンドを備える．
   - 本コマンドはイベント発生履歴の内容を空に設定し，総イベント発生数を 0 に設定する．
2. イベント発生情報の登録
   - イベント管理機能はイベント発生履歴にイベント発生情報を登録するコマンドを備える．
   - 本コマンドはコマンド実行時刻に指定されたイベントが発生したというイベント発生情報をイベント履歴に登録する．
   - 本コマンドは地上でのイベント処理機能の動作確認を意図して用意されたものである．

必要なテレメトリ

1. イベント発生履歴の内容
    - イベント記録機能はイベント発生履歴の内容をテレメトリとして用意する．

衛星毎に調整するパラメータ

1. イベント発生履歴の記録上限数
    - イベント発生履歴に記録できるイベントの数は衛星毎に調整可能なパラメータとする．
    - この値は衛星の運用の頻度や計算機の記憶領域の容量を考慮して決定すべき値である．

4.2 必須機能の実装

4.2.1 タイムラインコマンド機能

必要な処理

1. コマンドの蓄積
    - タイムラインコマンド機能は実行時刻が指定されたタイムラインコマンドを衛星内部に蓄積する機能を備える．
    - 本機能には複数のタイムラインコマンドを蓄積することが可能である．
    - 本機能は蓄積されているタイムラインコマンドの蓄積総数を保持する．
    - 本機能に蓄積されるタイムラインコマンドは常に実行時刻でソートされた状態で管理される．
    - 蓄積内容はタイムラインコマンド機能自身の操作かコマンドによる操作によってのみ変更可能とする．
2. 蓄積内容の初期化
    - タイムラインコマンド機能は C2A の動作開始時にタイムラインコマンドの蓄積機能を初期化する機能を備える．
    - 本機能は蓄積機能の蓄積状態を登録コマンドの無い空の状態に設定し，蓄積総数の値を 0 に設定する．
3. コマンドの登録
    - タイムラインコマンド機能は蓄積機能にタイムラインコマンドを登録する機能を備える．
    - 本機能は登録対象のタイムラインコマンドの実行時刻が既に蓄積機能に登録されているタイムラインコマンドの実行時刻と同一の場合は，タイムラインコマンドの登録を行わず，異常を通知する．
    - 本機能は指定された実行時刻が既に過去となっているタイムラインコマンドの場合は，

タイムラインコマンドの登録を行わず，異常を通知する．
- 本機能はタイムラインコマンドの蓄積総数が上限値に達している場合は，タイムラインコマンドの登録を行わず，異常を通知する．
- 本機能はタイムラインコマンドを蓄積機能に登録する際に，蓄積総数の値を 1 加算する．

4. 指定時刻でのコマンド実行
    - タイムラインコマンド機能は蓄積機能に蓄積されたコマンドを指定時刻に実行する機能を備える．
    - 本機能は時刻管理機能が提供する現在時刻と蓄積機能に蓄積されたタイムラインコマンドの直近の実行時刻とを比較し，両者が一致した場合に実行時刻に対応するタイムラインコマンドを実行する．
    - 本機能は蓄積機能に登録されたタイムラインコマンドの直近の実行時刻が時刻管理機能の提供する現在時刻より過去の場合，速やかに対応するタイムラインコマンドを実行し，同時に異常を通知する．
    - 本機能はタイムラインコマンドを実行した場合，蓄積総数の値を 1 減算する．
    - 本機能は実行したタイムラインコマンドの実行結果を保存する．
    - C2A は，ステップとサイクルの 2 種類の時刻粒度それぞれに対応するタイムラインコマンド機能を備える．

必要なコマンド

1. 蓄積コマンドの全消去
    - タイムラインコマンド機能は蓄積機能が蓄積しているタイムラインコマンド全てを消去するコマンドを備える．
    - 本コマンドは，蓄積機能の蓄積状態を登録コマンドの無い空の状態に設定し，蓄積総数の値を 0 に設定する．
2. 蓄積コマンドの指定消去
    - タイムラインコマンド機能は蓄積機能が蓄積しているタイムラインコマンドの中から特定のコマンドを消去する機能を備える．
    - 消去対象のタイムラインコマンドはその実行時刻で指定する．
    - 本コマンドは指定された実行時刻に対応するタイムラインコマンドを蓄積機能から消去し，蓄積総数の値を 1 減算する．
    - 本コマンドは指定された実行時刻に対応するタイムラインコマンドが蓄積機能の中に存在しない場合，消去を行わず，異常を通知する．

必要なテレメトリ

1. コマンドの蓄積状況

- タイムラインコマンド機能は蓄積機能に蓄積されているタイムラインコマンドの蓄積状況をテレメトリとして用意する．
- 本テレメトリの内容は，蓄積されているタイムラインコマンドの種類と実行時刻の対応関係を把握するのに十分な情報を含む．

2. コマンドの実行結果
   - タイムラインコマンド機能は実行したタイムラインコマンドの実行結果をテレメトリとして用意する．

衛星毎に調整するパラメータ

1. 蓄積できるタイムラインコマンドの上限値
   - 蓄積機能に蓄積できるコマンドの上限値は衛星毎に調整可能なパラメータとする．
   - この値は衛星の運用上必要となるタイムラインコマンドの個数に基いて決定すべき値である．
   - ブロックコマンドは展開時にタイムラインコマンドに変換されるため，この値はブロックコマンドの最大コマンド数よりも大きい値である必要がある．

4.2.2 ブロックコマンド機能

必要な処理

1. ブロックコマンドの内容管理
   - ブロックコマンド機能はブロックコマンドの内容を管理する機能を備える．
   - ブロックコマンドの内容はブロックコマンド定義テーブルで管理する．
   - ブロックコマンド定義テーブルは，ブロックコマンドを構成するコマンドの位置とその位置に登録されるコマンド情報との対応関係を管理する．
   - ブロックコマンドを構成するコマンドの位置はブロックコマンド番号とコマンド番号の組で表現される．
   - ブロックコマンド定義テーブルに登録されるコマンド情報は，コマンドの内容と，ブロックコマンド展開時の相対実行時刻の組で構成される．
   - ブロックコマンド定義テーブルは各ブロックコマンドに登録されているコマンドの登録個数を管理する．
   - ブロックコマンド定義テーブルは各ブロックコマンドの展開が可能かどうかを表す展開有効フラグを管理する．
   - ブロックコマンド定義テーブルの内容はブロックコマンド機能自身の操作とコマンド操作によってのみ変更可能とする．
2. ブロックコマンド書き込み位置の管理
   - ブロックコマンド機能はブロックコマンド定義テーブルに登録するコマンドの書き込み

位置を管理する機能を備える．

- 書き込み位置の情報はブロックコマンドテーブルのブロックコマンド番号とコマンド番号の組で表現される．
- ブロックコマンド書き込み位置の内容はブロックコマンド機能自身の操作とコマンド操作によってのみ変更可能とする．

3. ブロックコマンド定義テーブルの初期化
   - ブロックコマンド機能は C2A の動作開始時にブロックコマンド定義テーブルの内容を初期化する機能を備える．
   - 本機能は搭載ソフトウェアに用意されたブロックコマンドの内容をブロックコマンド定義テーブルに登録し，呼び出し可能な状態を用意する．
   - 本機能は C2A の動作開始時点で内容が定義されないブロックコマンドについては，コマンド登録個数を 0 に設定し，展開有効フラグを無効に設定する．
4. ブロックコマンドの登録
   - ブロックコマンド機能はブロックコマンド定義テーブルにコマンドを書き込む機能を備える．
   - 本機能は登録コマンドをブロックコマンド機能が管理する書き込み位置情報で指定される位置に書き込む．
   - 本機能は書き込み位置情報が指定する，ブロックコマンド番号かコマンド番号がその最大値を超えている場合，コマンド登録を行わず，異常を通知する．
   - 本機能は書き込み位置情報が指定する位置の直前にコマンドが登録されていない場合，コマンド登録を行わず，異常を通知する．
   - 本機能は登録コマンドを書き込んだ場合，書き込み位置情報のコマンド番号の値を 1 加算する．

必要なコマンド

1. ブロックコマンド書き込み位置の指定
   - ブロックコマンド機能はブロックコマンド定義テーブルの書き込み位置を設定するコマンドを備える．
   - 書き込み位置はブロックコマンド番号とコマンド番号で指定される．
   - 本コマンドは指定されたブロックコマンド番号かコマンド番号がその最大値を超えている場合，書き込み位置の設定を行わず，異常を通知する．
2. ブロックコマンドの登録解除
   - ブロックコマンド機能はブロックコマンド定義テーブルに定義されたブロックコマンドの登録内容を解除するコマンドを備える．
   - 登録解除を行うブロックコマンドのはブロックコマンド番号で指定される．
   - 本コマンドは指定されたブロックコマンド番号に対応するブロックコマンド定義テーブ

ルのコマンド登録数を 0 に設定し，展開有効フラグを無効に設定する．
- 本コマンドは指定されたブロックコマンド番号がその最大値を超えている場合，登録解除処理を行わず，異常を通知する．
- 本コマンドは登録解除を実行した場合，ブロックコマンド書き込み位置のブロックコマンド番号を指定されたブロックコマンド番号に設定し，コマンド番号を 0 に設定する．

3. ブロックコマンドの展開有効化
   - ブロックコマンド機能はブロックコマンド定義テーブルに登録されたブロックコマンドの展開を有効化するコマンドを備える．
   - 本コマンドは，ブロックコマンドの書き込み位置のブロックコマンド番号に対応するブロックコマンドの展開有効フラグを有効に設定する．
4. ブロックコマンドの展開
   - ブロックコマンド機能はブロックコマンド定義テーブルに定義されたブロックコマンドを展開するコマンドを備える．
   - 展開するブロックコマンドはブロックコマンド番号で指定される．
   - 本コマンドは，時刻管理機能が提供する現在時刻とブロックコマンド定義テーブルが保持する各コマンドの相対実行時刻とを足しあわせて各コマンドの実行時刻を計算し，各コマンドをタイムラインコマンドとして登録する．
   - 本コマンドは展開処理で各コマンドを登録する際に，タイムラインコマンドに登録できるコマンドの残数が展開するブロックコマンドの構成コマンド数よりも少ない場合は，タイムラインコマンドの登録内容を全て解除してブロックコマンドを展開し，異常を通知する．
   - 本コマンドは展開処理で各コマンドを登録する際に計算した実行時刻が既にタイムラインコマンドとして登録されているコマンドと一致してしまった場合は計算した実行時刻の値を 1 ずつ加算して直近の時刻で登録を行い，異常を通知する．
   - 本コマンドは指定されたブロックコマンド番号がその最大値を超えている場合，展開処理を行わず，異常を通知する．
   - 本コマンドは指定されたブロックコマンド番号に対応するブロックコマンドの展開有効フラグが無効の場合，展開処理を行わず，異常を通知する．
   - C2A ではタイムラインコマンドに，ステップを参照するタイムラインコマンドとサイクルを参照するタイムラインコマンドの 2 種類が存在し，ブロックコマンド展開時には展開先をいずれかに指定して展開できる．

必要なテレメトリ

1. ブロックコマンド定義テーブルの内容
   - ブロックコマンド機能はブロックコマンド定義テーブルの内容をテレメトリとして用意する．

- 本テレメトリの内容は，登録されているタイムラインコマンドを構成するコマンドの種類，相対実行時刻，登録個数，展開有効フラグの状態を確認するのに十分な情報を含む．

2. ブロックコマンド定義テーブルの書き込み位置
   - ブロックコマンド機能はブロックコマンド定義テーブルの書き込み位置をテレメトリとして用意する．

衛星毎に調整するパラメータ

1. ブロックコマンド番号の上限値
   - ブロックコマンド定義テーブルに登録できるブロックコマンドの上限値は衛星毎に調整可能なパラメータとする．
   - この値は C2A の必須機能に必要なブロックコマンドの数，衛星運用に必要なブロックコマンドの数と計算機の記憶領域の容量を考慮して決定すべき値である．
2. コマンド番号の最大値
   - ブロックコマンド定義テーブルに登録できる，ブロックコマンドを構成するコマンドの上限値は衛星毎に調整可能なパラメータとする．
   - この値は，登録するブロックコマンドの内容と計算機の記憶領域の容量を考慮して決定すべき値である．

4.2.3 モード管理機能

必要な処理

1. モード情報の管理
   - モード管理機能は C2A 実行中のモード情報を管理する機能を備える．
   - モード情報は現在のモード番号と直前のモード番号，モード状態で構成される．
   - モード状態はモード遷移処理を実行中かどうかを示す情報で，遷移中と遷移完了のいずれかの値をとる．
   - モード情報はモード管理機能自身の操作かコマンドによる操作のみによって変更可能とする．
2. モード維持処理の管理
   - モード管理機能は C2A 上に用意された各モードのモード維持処理を管理する機能を備える．
   - モード維持処理の管理はモード維持処理定義テーブルを用いて実現する．
   - モード維持処理定義テーブルはモードとそのモードに対応するモード維持処理を定義したブロックコマンド番号との対応関係を管理する．
   - モード維持処理定義テーブルの内容はモード管理機能自身の操作かコマンドによる操作のみによって変更可能とする．

3. モード遷移処理の管理
    - モード管理機能は C2A 上に用意された各モード間のモード遷移処理を管理する機能を備える．
    - モード遷移処理の管理はモード遷移定義テーブルを用いて実現する．
    - モード遷移定義テーブルは遷移元と遷移先のモードの組とその遷移に必要な処理を定義したブロックコマンド番号との対応関係を管理する．
    - モード遷移定義テーブルの内容はモード管理機能自身の操作かコマンドによる操作のみによって変更可能とする．
4. モード維持処理定義テーブルの初期化
    - モード管理機能は C2A の動作開始時にモード維持処理定義テーブルの内容を初期化する機能を備える．
    - 本機能は搭載ソフトウェアに用意された各モードのモード維持処理を定義するブロックコマンド番号をモード維持処理定義テーブルに登録し，呼び出し可能な状態を用意する．
5. モード遷移定義テーブルの初期化
    - モード管理機能は C2A の動作開始時にモード遷移定義テーブルの内容を初期化する機能を備える．
    - 本機能は搭載ソフトウェアに用意された各モード間のモード遷移処理を定義するブロックコマンド番号をモード遷移定義テーブルに登録し，呼び出し可能な状態を用意する．
    - 本機能は C2A の動作開始時点でモード間の遷移が定義されない箇所については，その内容を未定義に設定する．
6. モード維持処理の実行
    - モード管理機能は各モードのモード維持処理を周期的に実行する機能を備える．
    - 本機能は時刻管理機能が提供する時刻情報を参照し，ステップが 0 になった時点でモード維持処理定義テーブルに定義されたブロックコマンドをステップ参照するタイムラインコマンドとして展開する．
    - 本機能はステップ時刻が 0 となった時点で，ステップを参照するタイムラインコマンドにコマンドが登録されている場合は，一度タイムラインコマンドの内容を全消去してモード維持処理定義テーブルに定義されたブロックコマンドを展開し，異常を通知する．

必要なコマンド

1. モード維持処理の登録
    - モード管理機能はモード維持処理定義テーブルにモード維持処理を定義したブロックコマンド番号を登録するコマンドを備える．
    - モード遷移処理定義テーブル上の登録位置はモード番号で指定する．

- 本コマンドは指定されたモード番号に対応するモード維持処理定義テーブルの内容を指定されたブロックコマンド番号に変更する．
- 本コマンドは指定されたモード番号がその最大値を超えている場合は，登録処理を行わず，異常を通知する．

2. モード遷移処理の登録
   - モード管理機能はモード遷移定義テーブルにモード遷移処理を定義したブロックコマンド番号を登録するコマンドを備える．
   - モード遷移定義テーブル上の登録位置は遷移元のモード番号と遷移先のモード番号の組で指定する．
   - 本コマンドは指定された遷移に対応するモード遷移定義テーブルの内容を指定されたブロックコマンド番号に変更する．
   - 本コマンドは指定されたモード番号のいずれかがその最大値を超えている場合は，登録処理を行わず，異常を通知する．
3. モード遷移の開始
   - モード管理機能は現在のモードから別のモードへのモード遷移を開始するコマンドを備える．
   - 遷移先のモードはモード番号で指定する．
   - 本コマンドはモード遷移定義テーブルを参照し，現在のモード番号と指定されたモード番号から遷移処理を定義したブロックコマンド番号を特定し，その番号に対応するブロックコマンドをサイクルを参照するタイムラインコマンドに展開する．
   - 本コマンドはモード状態が遷移中の場合は，モード遷移処理を行わず，異常を通知する．
   - 本コマンドは指定されたモード番号がその最大値を超えている場合は，遷移処理を行わず，異常を通知する．
   - 本コマンドはモード遷移定義テーブル上で現在のモード番号と指定されたモード番号に対応する遷移が未定義となっている場合は，遷移処理を行わず，異常を通知する．
   - 本コマンドはモード遷移処理を行った場合，モード状態を遷移中に設定する．
4. モード遷移の終了
   - モード管理機能は実行中のモード遷移処理を終了するコマンドを備える．
   - 本コマンドは現在のモード番号を遷移先のモード番号に設定し，モード状態の内容を遷移完了に設定する．
   - 本コマンドはモード状態の内容が遷移中でない場合は，終了処理を行わず，異常を通知する．
   - 本コマンドはモード遷移処理を定義するブロックコマンドの最後のコマンドとして呼び出すことを想定している．

必要なテレメトリ

1. モード情報の内容
   - モード管理機能はモード情報の内容をテレメトリとして用意する．
2. モード維持処理定義テーブルの内容
   - モード管理機能はモード維持処理定義テーブルの内容をテレメトリとして用意する．
3. モード遷移定義テーブルの内容
   - モード管理機能はモード遷移定義テーブルの内容をテレメトリとして用意する．

衛星毎に調整するパラメータ

1. モード番号の最大値
   - モード管理機能が管理するモード番号の最大値は衛星毎に調整可能なパラメータとする．
   - この値は衛星運用に必要となるモードの数に基づいて決定すべき値である．

4.2.4 イベント処理機能

必要な処理

1. イベント処理情報の管理
   - イベント処理機能はイベントに対して C2A が自律的に実行するイベント処理を管理する機能を備える．
   - イベント処理の管理はイベント処理定義テーブルを用いて実現する．
   - イベント処理定義テーブルは，イベント処理 ID とイベント処理定義の対応関係を管理する．
   - イベント処理定義の内容はイベント処理の実行条件とその実行条件を満たした場合に実行する処理を定義したブロックコマンド番号で構成される．
   - イベント処理定義テーブルは，各イベント処理 ID に対応するイベント処理が有効かどうかを表す有効フラグを管理する．
   - イベント処理定義テーブルの内容はイベント処理自身の操作かコマンドによる操作のみによって変更可能とする．
2. イベント処理定義テーブルの初期化
   - イベント処理機能は C2A の動作開始時にイベント処理定義テーブルの内容を初期化する機能を備える．
   - 本機能は搭載ソフトウェアに用意されたイベント処理定義をイベント処理定義テーブルに登録し，呼び出し可能な状態を用意する．
   - 本機能は C2A の動作開始時点でイベント処理が定義されないイベント処理 ID につい

ては，その有効フラグを無効に設定する．

3. イベント処理の実行
    - イベント処理機能はイベント発生時に対応するイベント処理を呼び出す機能を備える．
    - 本機能はイベント記録機能が提供するイベント発生履歴とイベント処理定義テーブルを参照し，発生履歴の内容に一致するイベント処理の実行条件があるか検索する．この検索は有効フラグが有効のイベント処理 ID を対象に実行する．
    - 本機能は発生履歴が実行条件に一致した場合は実行する処理を定義したブロックコマンドをサイクルを参照するタイムラインコマンドに展開する．
    - 本機能はイベント処理を実行した場合，実行したイベント処理の有効フラグを無効に設定する．

必要なコマンド

1. イベント処理定義の登録
    - イベント処理機能はイベント処理定義テーブルにイベント処理定義を登録するコマンドを備える．
    - イベント処理定義テーブル上の登録位置はイベン処理 ID で指定する．
    - 本コマンドは指定されたイベント処理 ID に対応するイベント処理定義テーブルのイベント処理定義を指定された内容に変更する．
    - 本コマンドは指定されたイベント処理 ID の値がその最大値を超えている場合は，登録処理を行わず，異常を通知する．
    - 本コマンドは登録処理を実行した場合，対象のイベント処理 ID に対応する有効フラグを無効に設定する．
2. イベント処理の有効化
    - イベント処理機能はイベント処理定義テーブルのイベント処理定義を有効化するコマンドを備える．
    - 有効化対象のイベント処理定義はイベント処理 ID で指定する．
    - 本コマンドは指定されたイベント処理 ID に対応する有効フラグを有効に設定する．
    - 本コマンドは指定されたイベント処理 ID の値がその最大値を超えている場合は，有効化を行わず，異常を通知する．
3. イベント処理の無効化
    - イベント処理機能はイベント処理定義テーブルのイベント処理定義を無効化するコマンドを備える．
    - 無効化対象のイベント処理定義はイベント処理 ID で指定する．
    - 本コマンドは指定されたイベント処理 ID に対応する有効フラグを無効に設定する．
    - 本コマンドは指定されたイベント処理 ID の値がその最大値を超えている場合は，無効化を行わず，異常を通知する．

必要なテレメトリ

1. イベント処理定義テーブルの内容
   - イベント処理機能はイベント処理定義テーブルの内容をテレメトリとして用意する.
   - 本テレメトリの内容は，イベント処理 ID と対応するイベント処理定義の内容，有効フラグの状態を確認するのに十分な情報を含む.
2. 実行したイベント処理の記録
   - イベント処理機能は実行したイベント処理の記録をテレメトリとして用意する.
   - 本テレメトリの内容は，実行したイベント処理 ID と実行した時刻を含む.

衛星毎に調整するパラメータ

1. 登録できるイベント処理定義の最大値
   - イベント処理定義テーブルに登録できるイベント処理定義の最大値は衛星毎に調整可能なパラメータとする.
   - この値は衛星に必要となるイベント対処の総数を考慮して決定すべき値である.

第 5 章

C2A の利用指針

前章では実際の衛星搭載ソフトウェアとして C2A が備えるべき機能の整理を行った．ここで整理した機能を搭載ソフトウェアとして実装することで，衛星の必須機能が高い再構成能力を備えた形で実現される．

本章では実際に C2A を用いて搭載ソフトウェアの開発を行う場合に考慮すべき点について議論する．C2A に基いて開発される搭載ソフトウェアが C2A の能力を十分に発揮するためには，衛星を開発する際に

- 安全性を高める実装
- 再構成能力を高める実装
- 再利用性を高める実装

といった点を考慮する必要がある．

C2A によって実現される搭載ソフトウェアの軌道上再構成は再構成によって衛星を失うリスクを伴う．「安全性を高める実装」では，これを軽減するための手法について議論する．

C2A によって実現される再構成能力を軌道上で活用するためには，予め搭載ソフトェアに様々な余裕を用意することが重要となる．「再構成能力を高める実装」ではこの余裕について議論する．

C2A は時刻管理機能のみが衛星のハードウェアに依存しており，この部分だけを搭載計算機に合わせて実装することで様々な衛星に適用できる．C2A を様々な衛星で利用するためには，C2A の上で動作するアプリケーションについてもその再利用性を高める工夫が必要である．「再利用性を高める実装」ではこの実現方法について議論する．

5.1　安全性を高める実装

C2A は衛星搭載ソフトウェアの軌道上再構成能力を強化することで，打ち上げ後の衛星に不測の事態が生じた場合にも柔軟な対処を可能にする．このような軌道上再構成能力の強化は衛星の生存性を高める観点で非常に有益であるが，その半面，誤った再構成を実施することにより衛星を失うリスクを高めているとも考えられる．

C2A によって強化される軌道上再構成能力は 3.6.6 節で既に述べた通り，衛星の必須機能をコマンド処理機能に集約することで，従来の衛星搭載ソフトウェアではメモリ書き換えというリスクの高い手段でしか行えなかったような再構成をコマンド操作という比較的リスクの低い手段で実現可能にしている．しかし，従来と比較して再構成のリスクが低減されているものの，C2A によって実現される再構成に全くリスクが存在しないというわけではなく，誤った再構成によって衛星を失う可能性は依然として存在している．具体例としてブロックコマンドの編集によってモード維持処理の再構成を実施する場合を考えると，誤ってコマンド処理を担うアプリケーションに対応するコマンドをブロックコマンドから削除してしまった場合には，再構成以降は衛星はコマンド処理を行わなくなり，地上から一切の操作が行えなくなるといった事態が発生する可能性も考えられる．

以上の点を考えると，C2A を実際の衛星搭載ソフトウェアに適用するためには，軌道上再構成による衛星喪失という最悪の事態を避けるための仕組みを用意する必要がある．この仕組は，衛星に搭載ソフトウェアとは別に搭載計算機をリセット出来る手段を用意することと，リセット後に衛星が失われない生存性の高い機体設計を行うことの 2 点で構成される．以下それぞれについて詳細を述べる．

5.1.1 計算機リセット手段の用意

軌道上再構成によって最悪の場合，地上から搭載ソフトウェアの操作が全く行えなくなる可能性が存在することを考えると，搭載ソフトウェアに不具合が生じた場合に復帰させる手段として，搭載計算機をその搭載ソフトウェアによらず外部からリセット出来る手段を用意しておく必要がある．これを実現する具体的な例としては，通信機への特別コマンドの用意や別ハードウェアによるウォッチドッグ機能の用意が挙げられる．

通信機の特別コマンドとは衛星の搭載通信機が直接解釈するコマンドである．通常のコマンドは通信機を介して搭載計算機に入力され，搭載計算機が内容を解釈して処理を実行するのに対し，特別コマンドは地上からのある特定の信号が入力された場合に，通信機自身がそれを解釈して処理を行う．搭載計算機の状態によらず通信機のみでコマンド処理が完結するため，搭載計算機が異常状態となった場合にも強制的なリセットを実行できる．

ウォッチドッグ機能とは，内部にカウンタ機能と搭載計算機のリセット機能とを有し，カウンタが外部からリセットされずに一定時間が経過すると搭載計算機をリセットする機能である．通常は搭載計算機の搭載ソフトウェアがウォッチドッグ機能のカウンタを定期的にリセットして搭載計算機のリセット発生を防止するが，搭載ソフトウェアによって何らかの異常が発生してこの定期リセットが停止した場合はウォッチドッグ機能によって一定時間経過後に搭載計算機のリセットが実行される．

通信機の特別コマンドは衛星の搭載ソフトウェアがコマンド処理を行えなくなった場合の復帰手段として有効であり，ウォッチドッグ機能は衛星の搭載ソフトウェアが動作を停止した場合の復帰手段として有効である．通信機の特別コマンドは，搭載ソフトウェアの動作が停止した場合にも対処可能であるが，リセットを実行するためには地上からのコマンドが必要である．これに対して

ウォッチドッグ機能は，搭載ソフトウェアの状態を常時監視しており，停止時には自動的にリセットが実行されるという特徴があり，搭載ソフトウェアの停止時の対処に関してはウォッチドッグ機能の方が適している．

ここで述べたような，搭載計算機とは独立した別のハードウェアとして搭載計算機をリセット出来る手段を用意することで万一軌道上再構成に失敗した場合もに復帰が可能となる．

5.1.2 生存性の高い機体設計の採用

搭載ソフトウェア再構成の安全性を確保するためには，搭載計算機が外部からリセットできることに加えて，このリセット後に衛星が地上からのコマンドなしで生存できる事が重要となる．

この点に関しては，そもそも衛星の打ち上げ直後の初回起動時には地上局からコマンドを送ることが不可能である場合が多いことに加え，衛星のリセットはここで議論している搭載計算機の再構成によって生じ得る搭載ソフトウェアの異常以外にも例えば放射線による機能障害等様々な要因によって発生することが想定されるものであり，再構成能力とは関係なく，衛星の設計時点でその仕組が構築されるべきものである．

地上からのコマンドによらず衛星の生存性を高める方針としては，生存に不可欠である電力・通信・熱の姿勢依存性を下げることが重要となる．超小型衛星においては既に，軌道上での姿勢系のチューニングを実現することを目的にこのような生存性を高める手法について議論が行われている[43]．このような手法を取り入れて設計を行うことは本研究で提案する軌道上再構成能力の強化に対しても有益である．

5.2 再構成能力を高める実装

C2A によって高い再構成能力を有した搭載ソフトウェアが実現されるが，軌道上でこの再構成能力を十分に活用するためには，打ち上げ時点では搭載ソフトウェアにある程度の余裕を持たせておくことが重要となる．本節ではこの点について議論する．

C2A に基いて開発された搭載ソフトウェアの余裕とは具体的には各種定義テーブルの余裕であり，

- ブロックコマンド登録数 (ブロックコマンド定義テーブル)
- モード登録数 (モード処理・モード遷移定義テーブル)
- イベント処理登録数 (イベント処理定義テーブル)
- アプリケーション登録数 (アプリケーション・コマンド・テレメトリ定義テーブル)

の 4 項目を挙げる事ができる．

ブロックコマンド登録数の余裕とは，打ち上げの時点では特に内容を定義しない，未使用のブロックコマンドをブロックコマンド定義テーブルに確保しておく事を意味する．この余裕を確保しておくことで，軌道上で地上で用意していない新規のブロックコマンドの追加によって不測の事態

に対する対処能力を高めることが可能となる．ブロックコマンドは各種必須機能の処理を定義する中心となる機能であるので，ブロックコマンドの登録数に余裕を確保することは，以下で述べるその他の余裕を活用する上でも重要となる．

モード登録数の余裕とは，打ち上げの時点ではどのモードからも遷移できない，未定義のモードをモード維持処理定義テーブルとモード遷移定義テーブルに確保しておくことを意味する．この余裕を確保しておくことで，軌道上でこれらのテーブルを書き換えることによって衛星に新たなモードを定義することが可能となる．モード登録数に余裕がない場合は，C2A の再構成能力では既存のモード間の新たな遷移を追加することや，逆にあるモードへの遷移を全て無効化することでそのモードを削除することが可能であるものの，現状のモード構成を維持したまま新規のモードを追加することはできず，再構成能力が不完全となる．軌道上でのモードの再構成では，地上で動作を確認した既存のモード定義は残したまま新たにモードを追加するケースの方が多いと予想され，かつ実施に必要な手順もモード追加のみの方が簡単であるため，予め余裕を確保しておくことが重要である．

イベント処理登録数の余裕とは，打ち上げの時点ではどのイベントにも対応しない未定義のイベント処理をイベント処理定義テーブルに確保しておくことを意味する．この余裕を確保しておくことで，軌道上で地上では対処を想定していなかったイベントに対して何らかの対処を行う必要が生じた場合もこのテーブルに対応関係を追加することで，対処が可能となる．モード登録数の余裕と同様，イベント処理の登録数に余裕が無い場合は，処理の変更や削除は可能であるが，追加はできない状況となるため，再構成の能力が不完全となる．イベント処理の柔軟性を確保するためにはこの余裕を確保しておくことが重要となる．

アプリケーション登録数の余裕とは，打ち上げの時点では定義されていない未定義の領域をアプリケーション関連の情報を登録するアプリケーション定義テーブル，コマンド定義テーブル，テレメトリ定義テーブルに確保しておくことを意味する．この余裕は軌道上で衛星に予め用意したアプリケーションでは対処が不可能な事態が発生した場合に，メモリの部分書き換えによって新規のアプリケーションを追加する場合に必要となる．この余裕が存在せず，アプリケーション管理テーブルが全て必要なアプリケーションで埋まっている場合は，地上からのアプリケーション追加が行えず，再構成手段の中で実施リスクの最も高いメモリの全書き換えによる対処が必要となる．このような事態を避けるためには，この余裕を確保しておくことが重要になる．

5.3 再利用性を高める実装

C2A では 3.5 節に示したとおり，搭載ソフトウェアの構成が衛星に共通の必須機能と補助機能をまとめたコア機能部と衛星毎のアプリケーション群で構成される衛星固有部とに明確に分離される．

C2A に基づく搭載ソフトウェアの再利用性について考えると，コア機能部については計算機などのハードウェアに依存する部分は補助機能である時刻管理機能のみであり，この機能を衛星の環境に合わせて修正するのみで多くの環境に対応が可能である．また，4 章で述べた通り，C2A

を様々な衛星に適用する際に調整が必要となる箇所は C2A のパラメータとして明確化を行っており，衛星毎の違いにも容易に対応することが可能となっている．以上の点から，C2A のコア機能部については複数衛星での再利用性が確保されている．

衛星固有部については，C2A では衛星固有の機能をアプリケーションと呼ぶ単位で管理を行う．アプリケーションと C2A のコア機能との間のインターフェースはアプリケーション管理機能によって明確に規定されているため，各機能をアプリケーションに分割して実装することで，同様のアプリケーションが C2A を採用した別の衛星で必要とされる場合に再利用を行いやすい環境が整備されると予想される．

このようなアプリケーションの再利用を円滑に実施するためには，単純にアプリケーションを用意するだけではなく，アプリケーションを分割する際の構成に注意を払う必要がある．具体的には衛星以外にも多くの組み込みソフトウェアで行われているのと同様に，衛星の機能を衛星内部でのデータの流れに沿って分割して構成することでハードウェアの差異を各アプリケーションに局在化させることが重要であり，これによってハードウェアの差異に柔軟に対応が可能となりアプリケーションの再利用性が強化される．

C2A ではアプリケーションの分割の指針として図 5.1 の右側に示すような，

- 「ミドルウェア層（Middleware Layer）」
  搭載計算機ごとの差異を吸収し統一された操作手段を提供する
- 「ドライバ層（Driver Layer）」
  搭載コンポーネント依存の情報と操作を隠蔽し，抽象化を行う
- 「アプリケーション層（Application Layer）」
  衛星運用に必要な各種処理をモジュールとして実装する

の 3 層からなる分類を用いる．

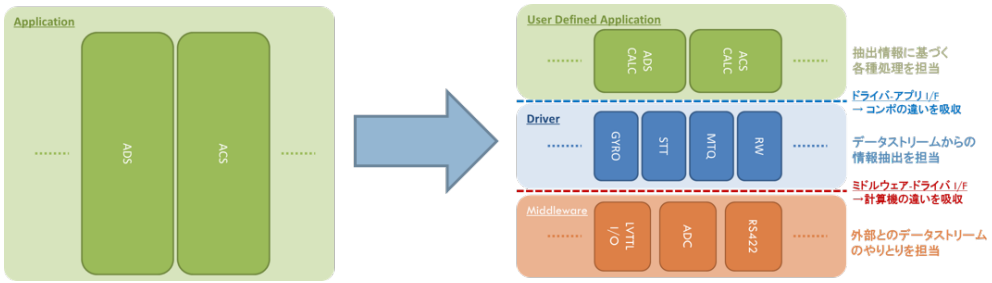


図 5.1　アプリケーションの階層化

例えば，姿勢決定や姿勢制御の機能を実装する場合に図 5.1 の左側のような「搭載センサからデータを取得して姿勢を決定するアプリケーション」や「衛星の姿勢から各アクチュエータへの指令値を計算し出力するアプリケーション」としてアプリケーションを実装してしまうと，この機能を別の衛星で再利用する場合に例えば使用する姿勢センサの種類が異なると，アプリケーションの

中でその違いが影響する範囲が明確で無いため，結局多くの箇所を修正しなければならないといった事態が発生し得る．

予め図 5.1 の右側のようにアプリケーションを「計算機外部と通信するアプリケーション」，「受信したセンサデータを解析するアプリケーション」，「センサデータから姿勢を決定するアプリケーション」といった形に分割して構成することで，例えば衛星に搭載するハードウェアが同一機能を持つ別のハードウェアに変更された場合もその違いが影響するアプリケーションが明確化され，その部分だけを変更することで他のアプリケーションはそのままの形で再利用することが可能となる．

実際に本節で示したような指針に基いて搭載ソフトウェアの開発を行うことで，異なる計算機上で同一機能の搭載ソフトウェアを動作させる場合にソフトウェアの差異を局所化でき，再利用性が高まることを確認している [53)54)]．

第 6 章

C2A による副次的効果

前章までで，C2A という明確な枠組に基づいて衛星の搭載ソフトウェアを開発することによって高い再利用性と再構成能力を備えた搭載ソフトウェアが実現されることを述べた．

C2A の提供する明確な枠組みは再利用性と再構成能力の高さ以外にも

- 開発者間の意思疎通への貢献
- 搭載ソフトウェア構築のパターン化
- 搭載ソフトウェア構築の自動化
- SILS 環境の構築への貢献

といった様々な効果をもたらす．本章では，このような C2A を利用することで再利用性と再構成能力の高さ以外に得られる副次的な効果について議論する．

6.1 開発者同士の意思疎通の円滑化

C2A に基づいて衛星の搭載ソフトウェアを開発することで，様々なレベルで開発者同士の意思疎通が容易になると考えられる．これは個々の開発者が C2A という明確な枠組みを共有して搭載ソフトウェアを開発することで，C2A が開発者間の意思疎通を行うための共通の言語として作用する事で得られる効果である．

まず，C2A によってソフトウェア開発チーム内での意思疎通が円滑になる効果が期待される．衛星の搭載ソフトウェアは全てを一人で開発するには規模が大きすぎるため，多くの場合，姿勢系やミッション系といったサブシステムごと，もしくはより細かな区分で開発者を割り当てて複数人で開発が進められる．このような開発体制下ではチーム内の開発者同士で意思疎通が円滑に行えることは搭載ソフトウェアの信頼性を高める観点で極めて重要になる．

C2A によって開発者間で衛星搭載ソフトウェアの認識を共通化できることで，開発チームの分散も容易になると予想される．例えば，搭載ソフトウェア全ての開発を一箇所で行うのではなく，複数の組織で得意な箇所を分担して開発して統合するといった作業も容易に実現出来るようになると考えられ，適切な分担が行えれば，開発期間の短縮といった効果が期待できる．

また，C2A を複数の衛星で用いることで，個々の衛星の開発チーム内部の意思疎通が円滑になるだけではなく，複数の衛星開発チーム同士での意思疎通も容易になると予想される．これは C2A に基づいて搭載ソフトウェアの開発を行うことで，別々の衛星であっても搭載ソフトウェアの構成が同一となることで得られる効果である．これにより，衛星を開発中のチーム同士で例えば相互レビューといったような意見交換が可能となるだけでなく，過去に C2A に基づいて開発された衛星の搭載ソフトウェアの理解も容易となる．これらは信頼性や再利用性向上の観点で有利に働くと期待される．

6.2 開発作業のパターン化

C2A に基づいて衛星の搭載ソフトウェアを開発することで，搭載ソフトウェアの開発作業において，その作業手順やソースコードの構成がパターン化されることが期待できる．これは C2A が搭載ソフトウェアの構成を明確に規定していることで得られる効果である．

C2A は衛星の動作の枠組みをコマンド処理機能，モード管理機能，イベント処理機能の 3 つ必須機能で規定する．衛星毎に必要となる機能はアプリケーションとして実装・登録され，C2A の枠組みを介して呼び出される．必須機能からの呼び出しを実現するためにアプリーケーションとして用意すべき項目は C2A のアプリケーション管理機能によって規定されているため，開発者はこの規定に沿う形でアプリケーションを開発すれば良い．

搭載ソフトウェアの中で実装すべき項目が C2A によって明確化されることで，作業手順もパターン化される．開発者は衛星に機能を追加する際に毎回全てを考える必要はなく，用意された枠組みに当てはめる形で作業を進める事が可能になる．開発すべき項目が明確に規定され，作業の手順がパターン化されることで，複数の機能を実装する際の負担が軽減し，ミスも減少すると同時に，初めて C2A で機能を開発する場合にもその理解が容易になるといった効果が生じることが期待される．

また，実装すべき項目の明確化によって，そのソースコードの構成もパターン化される．C2A によってアプリケーションを呼び出す方法が明確に規定されるため，異なるアプリケーションであっても，それぞれを実装するソースコード中の定数・変数や関数の定義や記法は同一のパターンに沿うことになる．これにより，ソースコードの開発や保守・管理が容易になるだけでなく，例えば過去の衛星で使用されたソースコードのような，開発者本人以外の他の開発者が作成したソースコードの理解も容易になる．これはソフトウェアの再利用性を高める上で重要な効果である．

6.3 開発作業の自動化

前節では，C2A に基づいて搭載ソフトウェアを開発することで，作業手順やソースコードの構成がパターン化されることを述べた．これをさらに進めることで実現されるのが搭載ソフトウェア構築の自動化である．

C2A によってパターン化が実現された箇所に関しては，その生成をソフトウェアによって自動

化することが可能となる．特にパターン化によって発生するソースコードの定形部分に対してはコードの自動生成を導入することで作業が効率化するだけでなく，人間が作業したことによってしばしば混入する単純ミスを防ぐことにもつながると考えられる．

特に C2A ではアプリケーション管理機能によって，衛星に実装される各アプリケーションのコマンドとテレメトリが統一的に管理され，その登録作業がパターン化される．これにより，衛星内部のソフトウェアだけでなく，衛星の運用に用いる地上局との連携についても自動化が可能となる．衛星の運用を行うためにはテレメトリとコマンドの定義を衛星と地上局との両方で一致させることが必須となる．C2A で実現されるソースコードのパターン化を活かして，衛星と地上局との連携を自動化することでこの部分のミスを軽減することは作業の効率化の観点で非常に重要である．

6.4 SILS 環境構築への貢献

打ち上げ後に修理や機能の追加が困難な衛星開発では，打ち上げ前の検証が重要となる．検証は実際の軌道上環境を模擬して行うことが望ましいが，特に姿勢制御系の検証については衛星と環境とが複雑に関係するため模擬が困難であり，多くの場合，衛星の姿勢運動と各種姿勢制御機器の入出力をシミュレータで模擬する環境を構築し，この環境上で姿勢制御ソフトウェアを動作させることで姿勢制御系の検証が行われる．

この検証方式には大きく分けて姿勢制御ソフトウェア自体もシミュレータ上で動作させる Software In the Loop Simulator (以下 SILS) と，姿勢運動と各種機器の状態のみをシミュレータ上で模擬し，姿勢制御ソフトウェア自体は実際の搭載計算機上で動作させる Hardware In the Loop Simurator (以下 HILS) の 2 種類が存在する．

SILS 環境は搭載計算機自体の挙動を模擬できないという欠点はあるものの，シミュレータ上で姿勢制御ソフトウェアの動作を簡易に確認することができるという特徴を持つ．一方の HILS は計算機自体の挙動まで含めて検証が行える点が長所であるが，検証は実時間となり SILS のような簡便さが失われるという欠点を持つ．このため，通常は，SILS で姿勢制御ソフトウェアのロジックを確認した後で HILS を用いて実際の計算機上で姿勢制御ソフトウェアが動作するかを確認するという流れで検証が実施される．

SILS と HILS を用いた検証では，SILS と HILS で動作させる姿勢制御ソフトウェアが実際の搭載ソフトウェアと同一であることが重要となる．これが異なると SILS や HILS を用いて検証する意義が大幅に損なわれる．C2A では搭載ソフトウェアの各機能をデータの流れに沿って分類し，それぞれを独立したアプリケーションとして階層的に実装することで，衛星搭載ソフトウェアのうち，搭載計算機特有の機能に依存する箇所が明確に分離されるため，SILS 環境の構築が容易になることが期待できる．これも C2A という枠組みを用いた効果の一つである．

第 7 章

C2A の適用結果

本章では，ここまで述べてきた C2A を実際の超小型衛星の搭載ソフトウェアに適用した結果について述べる．

本研究では C2A を「UNIFORM-1」,「ほどよし 3 号機」,「ほどよし 4 号機」,「PROCYON」の 4 宇宙機の搭載ソフトウェアに適用し，実際に開発と運用を行った．

本章では，まずこれら 4 宇宙機の概要を紹介し，その後これらの宇宙機の搭載ソフトウェア開発に C2A を適用した結果を

- 再利用性
- 軌道上再構成能力
- 副次的な効果
- 開発期間の短縮

の観点で整理し，C2A の効果について議論する．

7.1　適用した超小型衛星

本研究では，C2A を「UNIFORM-1」,「ほどよし 3 号機」,「ほどよし 4 号機」,「PROCYON」の 4 宇宙機の搭載ソフトウェアに適用して開発し，運用を実施している．図 7.1 はこれらの宇宙機を打ち上げ日順に示したものである．

UNIFORM-1 は文部科学省の超小型衛星研究開発事業「日本主導の超小型衛星網 UNIFORM の基盤技術研究開発と海外への教育貢献」プロジェクト (通称: UNIFORM プロジェクト) として開発された超小型衛星である．UNIFORM-1 のミッションは森林火災の早期検知であり，熱赤外カメラと可視カメラを搭載して地球観測を実施する．本プロジェクトは 2010 年 10 月に開始され，開発した衛星は 2014 年 5 月 24 日に H-IIA ロケットによって ALOS-2 打ち上げの相乗り超小型衛星として打ち上げられた [55]．打ち上げ後の初期運用を経て，現在はミッション運用を進めている [56]．

ほどよし 3 号機とほどよし 4 号機は，東京大学を中心とした内閣府最先端研究開発支援プログラ

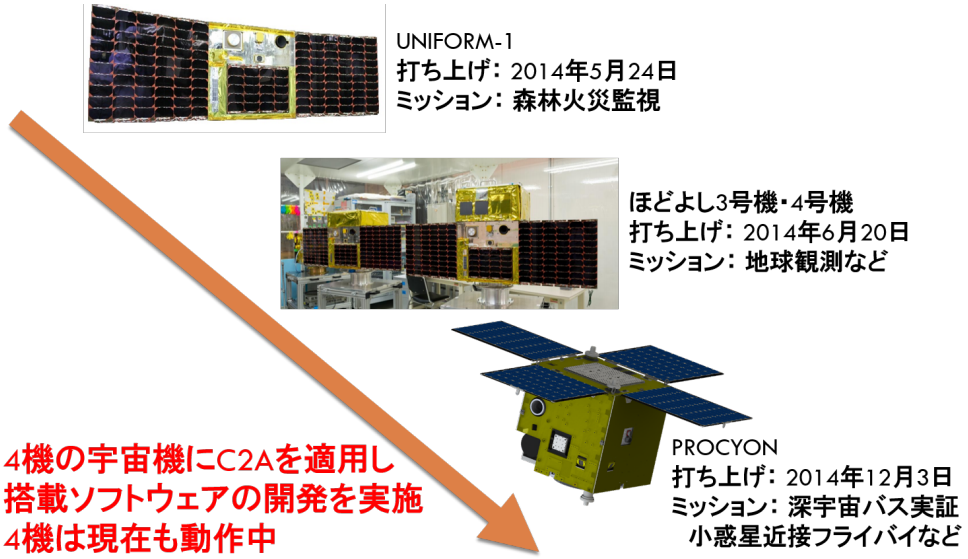


図 7.1　提案フレームワークを搭載した超小型衛星

ム「日本発のほどよし信頼性工学を導入した超小型衛星による新しい宇宙開発・利用パラダイムの構築」(通称: ほどよしプロジェクト) を通じて開発された超小型衛星である．両衛星は同一のバス構成を基本として同時並行に開発が実施された [59]．衛星のミッションはカメラによる地球観測を中心に，Store & Forward, 機器搭載スペースなど多岐に渡る．本プロジェクトは 2011 年 6 月に開始され，開発した 2 機の衛星は 2014 年 6 月 20 日にドニエプルロケットで同時に打ち上げられた [57][58][60]．2 機の衛星は初期運用を経て，現在は各種ミッションを実施している [61][62][63][64][65]．

PROCYON は上記 3 機の地球周回衛星とは異なり，地球重力圏を脱出して深宇宙へ向かう超小型深宇宙探査機である．PROCYON のミッションは超小型宇宙機による深宇宙探査技術の実証であり，深宇宙探査用の各種バス技術の実証と，小惑星への近接フライバイ撮影を目指している [18]．本宇宙機の開発は東京大学と宇宙航空研究開発機構・宇宙科学研究所を中心に 2013 年 9 月から開始され，2014 年 12 月 13 日に H-IIA ロケットによってはやぶさ 2 の打ち上げの相乗りペイロードとして打ち上げられた．現在探査機は既に惑星間軌道にあり，初期運用を進めている [66]．

以下ではこれらの宇宙機に C2A を適用した結果についてまとめる．

7.2　再利用の実例

本節では，C2A を適用して実際に複数衛星の搭載ソフトウェアを開発した結果を再利用性の観点で整理する．

7.2.1　ほどよし 3 号機とほどよし 4 号機のソフトウェア比較

「ほどよし 3 号機」と「ほどよし 4 号機」は C2A を適用して同時並行に搭載ソフトウェアの開発を行った衛星である．

「ほどよし 3 号機」と「ほどよし 4 号機」は衛星の基本機能を担うバス部と衛星固有のミッション部を明確に分離した設計が行われ，ハードウェア構成としては両衛星で共通のバス部を採用し，その上にそれぞれに異なるミッション機器を搭載している．図 7.2 に両衛星のシステム構成を示す[58]．

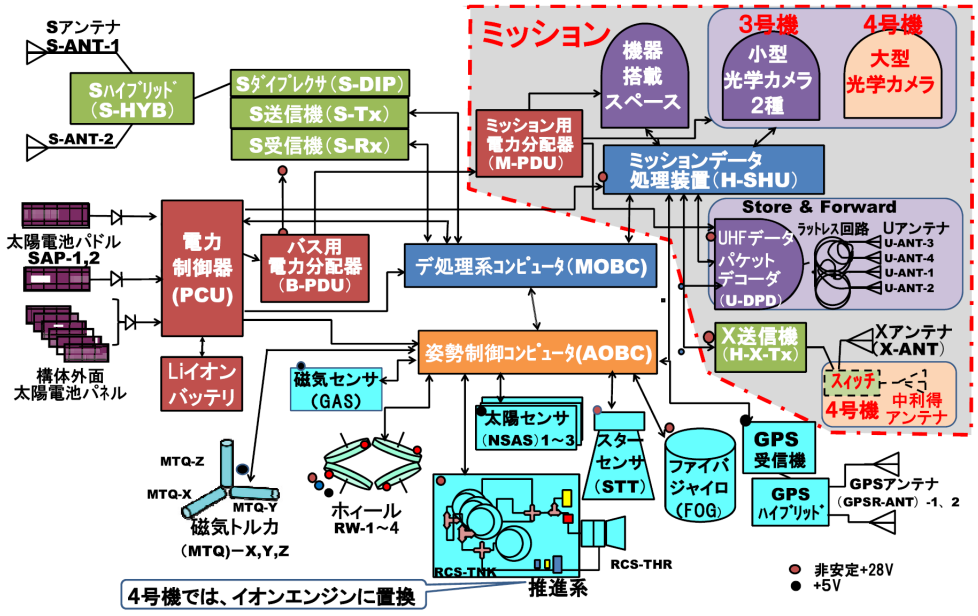


図 7.2　ほどよし 3 号機とほどよし 4 号機のシステム構成 [58]

C2A を適用して開発した両衛星の搭載ソフトウェアのコア機能とアプリケーションそれぞれを比較した結果を以下にまとめる．

コア機能の比較

C2A は衛星の搭載ソフトウェアが普遍的に有する機能を必須機能と補助機能からなるコア機能として抽出・実装し，衛星毎に固有の機能をアプリケーションの形で実装する．各衛星に固有の機能はコア機能からアプリケーションを呼び出すことで実現する仕組みとなっている．このため C2A を適用して開発した両衛星の搭載ソフトウェアは，コア機能の部分は両衛星で同一となり，差異が生じるのはアプリケーションに限定されるはずである．

表 7.1 にほどよし 3 号機とほどよし 4 号機のコア機能の比較結果を示す．C2A を構成する補助機能と必須機能について，4 章にまとめた，「必要な処理」，「必要なコマンド」，「必要なテレメト

リ」,「衛星毎に調整するパラメータ」それぞれの変更の有無をまとめたものである.

表 7.1　ほどよし 3 号機とほどよし 4 号機の C2A コア機能の比較

		処理	コマンド	テレメトリ	パラメータ
補助機能	時刻管理機能	同一	同一	同一	同一
	アプリケーション管理機能	同一	同一	同一	同一
	イベント記録機能	同一	同一	同一	同一
必須機能	コマンド処理機能	同一	同一	同一	同一
	モード管理機能	同一	同一	同一	同一
	イベント対応機能	同一	同一	同一	同一

実際に開発を行った「ほどよし 3 号機」と「ほどよし 4 号機」の搭載ソフトウェアでは，C2A のコア機能部分については両衛星で完全に同一のソースコードを共有しており，表 7.1 においては全ての項目が「同一」となっている.

以上の結果は，C2A のコア機能が衛星に共通の機能を実現するのに十分な内容を備えており，C2A のコア機能自身に高い再利用性があることを示していると考えられる.

衛星毎に調整するパラメータを含め，両衛星のコア機能が同一の構成となった理由には，両衛星の開発が同時に行われ，バス部の構成を共有しているという点が大きく影響していると考えられる．両衛星の構成が非常に似通っているため，同一の搭載ソフトウェアを適用できた可能性も考えられるが，これについては続く PROCYON との比較において追加の議論を行う.

アプリケーションの比較

C2A ではアプリケーションを実装する際に，5.3 節で示した通り，アプリケーションをデータの流れに沿って分割して実装することで，ハードウェア構成の差異に応じて発生する搭載ソフトウェアの差異を各アプリケーションに局所化することが可能となっている．このため，バス部のハードウェア構成を共有し，ミッション部のみにハードウェアの違いがある両衛星の搭載ソフトウェアでは，アプリケーションに生じる変更もハードウェアの変更箇所と明確に対応する箇所に限定されるはずである.

ほどよし 3 号機とほどよし 4 号機のアプリケーションを比較した結果として，両者の変更がアプリケーションの削除と追加のみであり，内容を変更したアプリケーションが存在しない点が挙げられる．ほどよし 4 号機の搭載ソフトウェアを基準にすると，削除が行われたアプリケーションはほどよし 3 号機固有のミッション機器と対応しており，逆に追加が行われたアプリケーションはほどよし 4 号機固有のミッション機器に対応している．両衛星はミッション機器以外のバス部の機器を共有しており，バス部の機器に対応するアプリケーションには変更が行われていない.

以上の結果は，アプリケーションの違いをミッション部の違いというハードウェア的な差異に関連する部分のみに限定し，両衛星の間で，同一の機器については同一のアプリケーションを利用す

るというアプリケーションの再利用が行えていることを示している．

これは C2A という明確な枠組みの元でソフトウェアを構成したことにより得られた効果である．

7.2.2　ほどよし衛星と PROCYON のソフトウェア比較

本節では，C2A を適用したほどよし衛星と PROCYON との間で搭載ソフトウェアを比較し，その結果について議論する．

図 7.3 に PROCYON のシステム構成図を示す [18]．

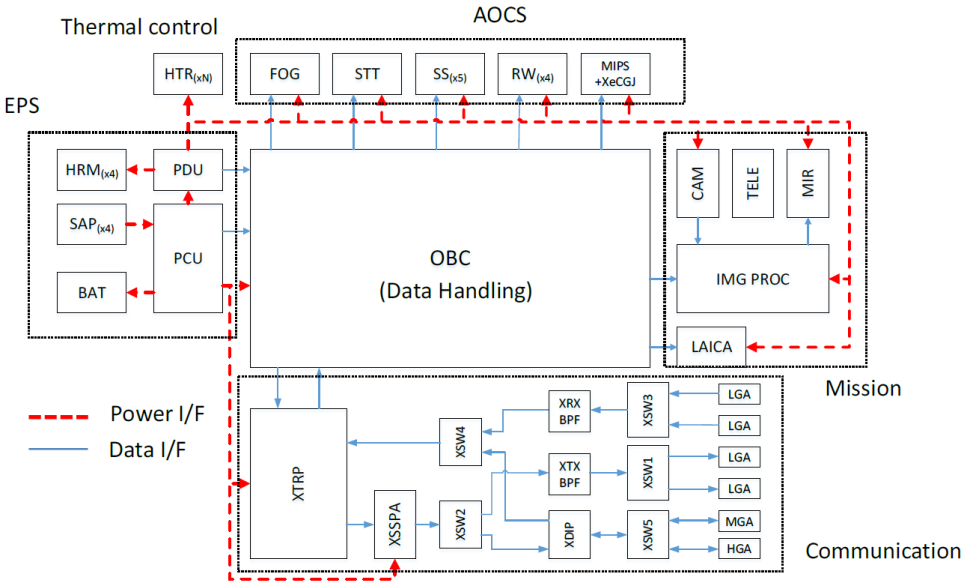


図 7.3　PROCYON のシステム構成 [18]

PROCYON の開発期間は約 1 年という非常に短い期間であり，この超短期開発を実現するために探査機のバス部はほどよし 3 号機・4 号機で開発・利用された機器を最大限活用して開発を実施した．電源系，姿勢制御系，そして搭載計算機については可能な限り，ほどよし衛星と同一の機器を採用している．深宇宙探査機として地球周回衛星とは異なる機器が必要となった通信系と PROCYON 独自の部分であるミッション系については新規に開発を行っている．

機器単位では，ほどよし 3 号機・4 号機と PROCYON の間で共通の機器は多いものの，宇宙機としてのシステム構成には大きな違いがある．図 7.2 と図 7.3 の比較から読み取れる通り，ほどよし 3 号機・4 号機ではデータ処理を担当する計算機 (MOBC) と姿勢制御を担当する計算機 (AOBC) が別個に搭載されていたのに対し，PROCYON では搭載計算機 (OBC)1 機のみの搭載となっている．このため，PROCYON では C2A に基づく搭載ソフトウェアが，ほどよし 3 号機・4 号機では担当していなかった姿勢制御関係の機能も担当する必要が生じている．

このようにハードウェア構成の大きく異なる 2 種類の宇宙機に，同一の C2A を適用した結果を比較することは，C2A に基づく搭載ソフトウェアの再利用性を評価する上で極めて有益であると

考えられる．

コア機能の比較

C2A の必須機能と補助機能についての比較結果を表 7.2 にまとめる．

表 7.2　ほどよし 3 号機と PROCYON の C2A コア機能の比較

		処理	コマンド	テレメトリ	パラメータ
補助機能	時刻管理機能	同一	同一	同一	変更
	アプリケーション管理機能	同一	同一	同一	変更
	イベント記録機能	同一	変更	同一	変更
必須機能	コマンド処理機能	同一	同一	同一	変更
	モード管理機能	同一	同一	同一	変更
	イベント対応機能	変更	同一	同一	変更

まず着目すべき点として，ほどよし衛星と PROCYON との間に必須機能や補助機能の新規追加や削除が存在しない点が挙げられる．過去の衛星から必須機能を洗い出して整理し，パターン化することで導き出した C2A のコア機能が地球周回衛星と地球重力圏を脱出する深宇宙探査機という性質の異なる両宇宙機に機能の追加・削除なしに適用できたという事実は，本アーキテクチャの妥当性を強く支持する結果と言える．

次に，必須機能と補助機能のうち変更のあった箇所について考察する．

表 7.2 より，C2A コア機能の全てに対してパラメータの変更が実施されたことが読み取れる．この変更はほどよし 3 号機と PROCYON との間のハードウェア構成の違いによって生じたものである．ほどよし 3 号機には姿勢制御を専門に担当する計算機である AOBC が存在し，姿勢制御機器の多くが AOBC に接続されていたのに対し，PROCYON では探査機に搭載する計算機を OBC 一台に集約し，ほどよし 3 号機では AOBC が担当していた姿勢制御機能も OBC が担当している．これにともなって PROCYON では，モード維持に必要な処理を全て実行する時間を確保するために 1 サイクルの時間変更が必要となり，実装が必要になるアプリケーションの数，モードの数，イベントの数等が大きく増加した．これらがパラメータの変更として現れている．

その他の変更点としてはイベント記録機能のコマンドとイベント対応機能の処理に関するものが存在している．イベント記録機能については PROCYON でほどよし 3 号機には用意していなかった，擬似的にイベントを追加するコマンドを用意した．これは主に地上試験においてイベント対処機能の確認を簡単に実現するための機能追加である．また，イベント処理機能自体の変更として，ほどよし 3 号機では顕在化しなかったバグの修正を実施している．

これら 2 つを除けば，変更箇所は予め変更を想定して用意したパラメータ変更のみにとどまっており，新規にコア機能として追加が必要となった機能も存在しないことから，性質の大きく異る 2 種類の宇宙機で C2A という同一のアーキテクチャを共有できていると言える．

アプリケーションの比較

各衛星毎の機能を実現するアプリケーション部分の比較結果を図 7.4 に示す．

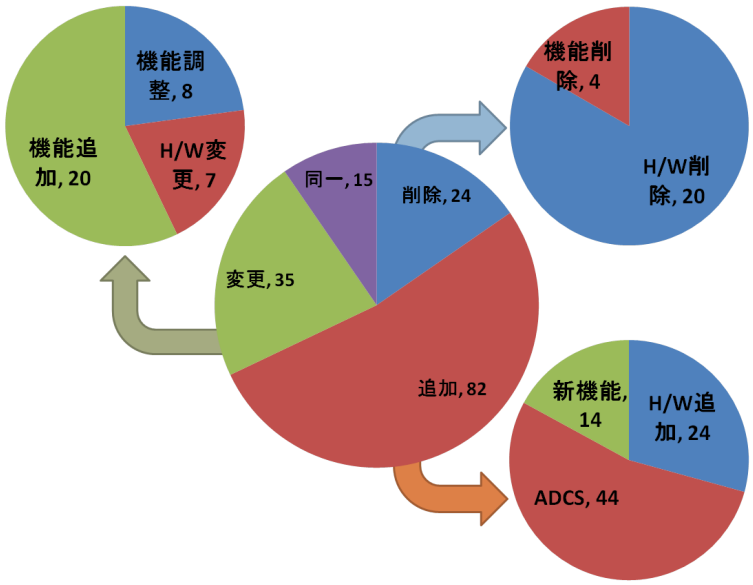


図 7.4　ほどよし衛星と PROCYON の変更箇所比較 (アプリケーション部分)

先ほどの必須機能と補助機能の比較結果と異なり，アプリケーション部分については「削除」・「追加」が存在し，その割合は「変更」・「同一」の箇所に対して大きくなっている．

まず「削除」と「追加」の部分に着目すると，これらの変更の大部分が衛星のハードウェア部分の変更に由来することが言える．削除と追加それぞれにについてより詳しい考察を以下に述べる．

「削除」の部分に関してはほどよし衛星のミッション機器に対応するアプリケーションが大半を占めている．これはほどよし衛星のミッション機器が PORCYON に搭載されないことを考えれば当然の結果である．また，ミッション機器以外の項目に関しても，ほどよし衛星でバス系機器を構成していたもののうち，PROCYON には搭載されていない磁気トルカに関係するアプリケーションが取り除かれた他，ほどよしでは独立して搭載した姿勢系計算機と主計算機を PROCYON では 1 つの搭載計算機にまとめた結果，姿勢系計算機とのデータ送受に関係するアプリケーションが削除されている．

「追加」の部分に関しては PROCYON で新たに搭載された小惑星撮影用のカメラや VLBI 送信機といったミッション機器や，X バンドトランスポンダや姿勢制御スラスタと言ったバス機器による H/W 追加が一定の割合を占めている．これについてもこれらの機器がほどよし衛星には搭載されていなかったことを考えればアプリケーションの追加は当然の結果といえる．加えて，「削除」の点で触れた通り，PROCYON ではほどよし衛星と異なり，ほどよし衛星では AOBC が担って

いた機能をほどよしの MOBC に統合して単一の OBC としたため，これに伴って姿勢制御系の処理がアプリケーションとして大幅に追加されている点も特徴である．これらの他に，地球周回衛星であるほどよし衛星では使用せず，深宇宙探査機の PROCYON で必要となった機能が追加されている．

次に，「同一」と「変更」の箇所に着目すると，アーキテクチャ部分とは異なり，アプリケーションではほどよしのものをそのまま使用できたものは少なく，多くの部分で変更が加えられたことが読み取れる．

「変更」が行われた箇所の内訳には必須機能と補助機能の部分で述べたようなパラメータ調整や PROCYON でのハードウェア変更に伴う部分も一定数存在するものの，半数以上が機能の追加となっている．これはほどよし衛星で用意されていた機能を PROCYON で使用するにあたり，深宇宙探査という特殊性や運用時の利便性を考えより高機能化が求められたことによる．具体的な例を挙げると衛星の非可視時のデータを蓄積するデータレコーダ機能はほどよし衛星では単一の領域に全てのデータを蓄積していたものが PROCYON では記録領域を複数に分割して管理し，データの種別毎に蓄積先を振り分けるといった動作ができるよう機能が追加された．

このように，ほどよし衛星と PROCYON との間にはアプリケーションに多くの違いが生じたが，衛星の機能をアプリケーションとして実装したことによってここまでで説明してきた通り，その違いを明確に説明することが可能となっている．これは本フレームワークによって衛星が実現すべき各機能が明確に区分されて実装された事により搭載ソフトウェア全体が見通しの良い形に整理されたことを示している．また，ほどよし衛星と PROCYON とで共通の機能を実現するアプリケーションについては PROCYON で使用するにあたって機能追加は多かったものの，完全に新規の開発ではなく，ほどよしの各機能を引き継いだ形での開発となっており，各機能を個別のアプリケーションとして整理したことによって再利用性が高まったといえる．

7.3 軌道上再構成の実例

本節では，C2A を適用した衛星の運用の過程で，C2A の柔軟な軌道上再構成能力が活用された事例を紹介する．

7.3.1 ブロックコマンド編集による軌道上再構成

モード維持処理の再構成

C2A のモード管理機能はモードを維持のために周期的に実行するアプリケーションをブロックコマンドの形で管理している．このブロックコマンドの内容を編集することで，モード維持に必要となるアプリケーションの追加・削除や実行タイミングの調整が可能である．

実際の運用でこのモード維持処理のブロックコマンド編集が必要となった例として，PROCYON のミッション機器と搭載計算機 (以下 OBC) との通信処理を担うアプリケーションの実行時間変動問題への対処が挙げられる．

PROCYON はミッション機器として小惑星の撮影を行うカメラを搭載している．このカメラは OBC と接続され，両者の間でコマンドとテレメトリのやりとりを行っている．今回ここで例として取り上げる問題は，このカメラから OBC へのテレメトリ受信で生じた問題である．

カメラから OBC に転送するテレメトリには，カメラ自体の状態を示すデータ量の少ないテレメトリと，カメラで撮影した画像データを含むデータ量の大きなテレメトリの 2 種類が存在する．カメラからのテレメトリ受信を行うアプリケーションの実行時間は受信するテレメトリのデータ量に依存しており，データ量の少ないテレメトリを受信する際にはその実行時間が後続のアプリケーションの実行開始前に収まっているものの，データ量の多いテレメトリの受信時にはその実行時間が伸び，後続のアプリケーションの実行開始時刻を超える場合が存在することが問題となった．

カメラのテレメトリ受信を行うアプリケーションの実行時間が後続するアプリケーションの実行開始時刻を超えてしまう場合，後続するアプリケーションの実行開始時刻は本来設定した時刻より遅れる事となり，カメラから OBC へのテレメトリのデータ量の大小によって，1 周期の中で実行されるアプリケーションの実行タイミングが異なってしまう．このような状態は周期的に実行している姿勢制御などの動作に悪影響を与える可能性があり，対処を行う必要が生じた．

この問題は，新規開発要素であったカメラ単体の開発が想定より遅れたために，カメラを PROCYON 全体と統合した構成での試験が十分に実施できなかったことが影響し，ロケット側へ PROCYON を引き渡す直前のソフトウェア開発の最終段階で発覚した．この時点では既に搭載ソフトウェアの最終動作確認がある程度進んでおり，ソフトウェアを修正して再度確認する時間的余裕もなかったことから打ち上げ後に軌道上で対処を行う方針となった．

カメラを動作させるモードでは，1 周期の中で実行するアプリケーションの合計実行時間は 1 周期の時間全体に比べて小さく余裕があったため，ブロックコマンドの編集では，カメラからのテレメトリ受信アプリケーション以降に実行するアプリケーションの実行開始時間を初期の設定より後ろにずらして設定し，カメラからのテレメトリデータ量の大小でテレメトリ受信アプリケーションの処理時間が変動しても確実に処理が完了する時間を確保することで，問題を回避する．本論文執筆時点では，地上試験で動作を確認しただけの状態であるが，近日中に実機に対し同様の再構成を実施する予定である．

この例は C2A が衛星の必須機能をブロックコマンドに集約したことで実現される柔軟な再構成能力を示す具体例と言える．

イベント対処の再構成

C2A のイベント対処機能は，イベント発生に応じて呼び出す処理をブロックコマンドの形で管理している．このブロックコマンドの内容を編集することで，イベント対処に必要な処理の追加・削除や実行タイミングの調整が可能である．

実際の運用でイベント対処のブロックコマンド編集が必要となった例として，ほどよし 3 号機と 4 号機でのソフトウェア UVC の対処変更が挙げられる．UVC とは Under Voltage Control の略称で，衛星のバッテリ電圧が想定を下回った場合のバッテリ電圧回復操作を意味する．ソフトウェア UVC とはソフトウェアによって実行される電圧回復操作である．

ほどよし 3 号機・4 号機は衛星の姿勢制御状態を中心にモードが構成されており，衛星起動後は，「無制御モード」，「スピン太陽指向モード」，「3 軸太陽指向モード」の 3 段階を経て地球指向の姿勢を基本とする様々な「ミッションモード」で各ミッションが実施される．ソフトウェア UVC としては 3 軸太陽指向モード以降のモードでバッテリ電圧の降下が発生した場合は衛星のモードをスピン太陽指向モードに設定する対処がイベント処理機能に設定されていた．これは，姿勢制御機器の多くが新規の開発機器であった背景もあり，なるべく少ない姿勢制御機器で電力を確保しようという思想に基づく設定であった．

しかし，打ち上げ後の軌道上で搭載している太陽センサが太陽だけでなく，日照時の地球アルベドにも反応し，地球方向を太陽方向として誤出力するという不具合があり，この影響で UVC 発生時に電力確保のためにスピン太陽指向モードに遷移すると，衛星が主発電面を地球方向に向けてスピン安定するという事象が発生した．ほどよし衛星の軌道では日照期間中の地球方向はほぼ反太陽方向であり，主発電面の裏側には太陽電池がなかったことから，衛星はスピン安定後ほとんど発電能力を失ってスピン軸を反太陽方向に固定した状態で電力枯渇状態に陥った．衛星は日照期間中は僅かに発生する電力で起動するものの，バッテリ充電に十分な電力を獲得できず日陰期間中は電力収支が成立しない状態となったため，運用が日照期間中のみに制限され，復帰運用に非常な困難が伴う事態となった．

上記の経緯から，ほどよし 3 号機・4 号機ではこの不具合から復帰後に UVC の対処を規定するブロックコマンドの内容を書き換え，遷移するモードをスピン太陽指向モードから，無制御モードへ変更する再構成を実施した．これは反太陽指向で姿勢が安定するよりも，姿勢制御を行わずに姿勢制御機器も全て電源を OFF にして，衛星の姿勢がランダムに変化する状態とした方が発電量が確保でき安全であるという判断を反映したものである．この変更後，同様の事象は回避され，反太陽指向による電力枯渇状態は発生していない．

この例も前述のモード維持処理の変更と合わせて，必須機能をブロックコマンドに集約して実現する C2A によってもたらされる柔軟な再構成能力が活用された具体例といえる．

7.3.2　定義テーブル変更による軌道上再構成

新モードの追加

C2A ではモード処理定義テーブルとモード遷移定義テーブルを打ち上げ後に変更することで，新たなモード遷移の追加や新たなモードの追加といったモード定義の変更を行うことが可能である．

実際にこの再構成能力を活用した例として，前述の反太陽指向状態からの回復操作が挙げられる．ほどよし 3 号機・4 号機では前述の反太陽指向でのスピン安定の発生後，この状態から復帰させるために衛星のスピン角速度を落として衛星の姿勢を意図的に不安定にするデスピンの操作を行う必要が生じた．

通常状態であれば用意していた姿勢制御モードを使うことでデスピンを実施することが可能であったが，不具合発生時は衛星は主発電面を反太陽方向に向けて安定しているため日照期間中でも発生電力が不足しており，多数の姿勢制御機器を用いた姿勢制御が行えない状態であった．

発生電力が不足している状態でもデスピンを実行するためには，使用する姿勢制御機器の少ない，なるべく消費電力の低い状態で姿勢制御を行う必要があり，ほどよし 3 号機・4 号機ではこの制約を満たす制御方式として B-Dot 制御を用いた角速度制御が対策として考えられた．

B-Dot 制御は地磁気センサと MTQ の 2 つの機器だけでデスピンを行う制御方式であるが，本制御方式は打ち上げ時点では AOBC に実装はされていたものの使用する予定はなく，衛星のモードとしては用意していないものであった．このため，地上からモード維持処理定義テーブルとモード遷移定義テーブルの書き換えを行い，新たに「B-Dot モード」を定義することでこの対処を実現した．

具体的には，まず B-Dot モードで必要となるモード維持処理をブロックコマンドに定義して，モード維持定義テーブルに登録し，その後，無制御モードから B-Dot モードへ遷移する遷移処理とその逆の遷移処理をブロックコマンドに定義してモード遷移定義テーブルに登録して事前に存在しなかった B-Dot モードを定義し，遷移を可能とした．

定義した B-Dot モードで運用を行うことで衛星のデスピンに成功し，衛星は無事回復した．

この例は軌道上でモード遷移を変更できる C2A の柔軟な再構成能力が活用された具体例といえる．

7.3.3 メモリ部分書き換えによる軌道上再構成

アプリケーション単位でのメモリ書き換え

C2A では衛星固有の機能がアプリケーションという明確な単位で区分けされて実現されているため，衛星固有の機能追加を C2A のコア機能に変更を加えず，アプリケーション単位で実施することが可能である．

C2A を適用した衛星の運用において，実際にこのアプリケーション単位での再構成を行った例として，PROCYON の姿勢決定系のアプリケーション変更が挙げられる．

PROCYON では 3 軸姿勢制御を実施する際に自身の姿勢を決定するセンサーとしてスタートラッカ (以下 STT) を用いているが，打ち上げ後の機器動作確認でこの STT が不正確な姿勢決定値を出力する場合があることが判明した．打ち上げ前に用意したアプリケーションでも，STT の出力する姿勢決定値が不正確な場合を考慮しこのような値を取り除く処理を組み込んではいたものの，実際に出力される姿勢決定値の中で不正確なものの一部はこの処理を通過してしまうものであり，結果として，衛星の姿勢決定が異常をきたし問題となった．

この問題を解決するため，PROCYON では STT の出力する姿勢決定値に対して新たにより厳格な確認を行って不正確な値を取り除く処理を加えたアプリケーションを実装して地上で試験を行った後，このアプリケーションのみを PROCYON の OBC のメモリに追加して，このアプリケーションを用いて姿勢制御を実施するよう搭載ソフトウェアの軌道上再構成を実施した．

具体的には，アプリケーションを OBC のメモリに追加した後に，追加したアプリケーションをアプリケーション管理機能のアプリケーション定義テーブル，コマンド定義テーブル，テレメトリ定義テーブルに登録して C2A のコア機能から呼び出し可能な状態とし，3 軸姿勢制御を行うモード

維持処理に対応するブロックコマンドを書き換えることで，STT に関連するアプリケーションを新規に追加したアプリケーションに置き換える作業を実施した．この再構築によって PROCYON は安定した 3 軸姿勢制御が可能となった．

この例は，C2A によって衛星固有の機能がアプリケーションという明確な単位に区分して実装されることで得られた柔軟な再構成能力の具体例といえる．C2A のような枠組みが存在しない場合，単機能の追加を行う場合であっても，その影響が及ぶ範囲を限定できず，結果として大規模なメモリ書き換えが必要になると考えられ，このような対処は容易に実現できなかったと考えられる．

7.4 副次的効果の実例

ここまでで，衛星の搭載ソフトウェアの開発に C2A を適用することで実現した高い再利用性と柔軟な軌道上再構成能力について，具体例を挙げて整理した．

本節では C2A の狙いである再利用性と軌道上再構成能力の強化以外に C2A を衛星の搭載ソフトウェア開発に適用することで得られた副次的な効果について具体例を挙げて整理する．

7.4.1 開発者同士の意思疎通の円滑化

C2A を衛星の搭載ソフトウェア開発に適用した効果として開発者同士の意思疎通が円滑になった点が挙げられる．

まず，開発チーム内での意思疎通について PROCYON の開発チームの例を挙げることができる．PROCYON の搭載ソフトウェア開発では，通信系や電源系といったサブシステム単位や，より細かな単位で開発者を割り当て，複数人の開発者で同時並行にアプリケーションの開発を行った．

開発にあたっては，まず開発者に C2A のコア機能の内容，アプリケーションの位置づけ，アプリケーションをコア機能から呼び出す方法など，C2A の仕組みを説明した上で，開発者ごとに担当するアプリケーションを開発し，統合を実施した．

搭載ソフトウェアの開発に C2A という枠組みが存在したことで，各開発者がアプリケーションとして開発すべき内容が明確化されたことに加え，複数の開発者でアプリケーションの実装について議論を行う場合も C2A の枠組みが議論を行う上で共通の言語として機能し，円滑に開発を進めることができた．また，アプリケーションの開発だけでなく，それらを統合して確認する各種の試験においても，不具合が発生した場合の原因調査の過程や，原因が確定した後の対策の過程でも，C2A の枠組みによって開発者全員の意思疎通が円滑に実施された．

次に，複数の開発拠点で搭載ソフトウェアの開発を実施した際に C2A がもたらした効果として，東京大学と東京理科大学との搭載ソフトウェアの共同開発を挙げることができる．

C2A を適用した 4 衛星の搭載ソフトウェア開発は，筆者の所属する東京大学 中須賀・船瀬研究室だけで実施したものではなく，東京理科大学の木村研究室と共同で実施したものである．特に，以降の節で述べる，コマンド・テレメトリ追加の自動化や SILS の構築は両研究室の密な連携に

よって実現された．

複数の拠点で開発を行う際には，単一の拠点での開発以上に開発者間の意思疎通が重要となる．C2A という明確な枠組みによって，両拠点の間で作業の役割分担が明確となり円滑に開発・統合が進められたことは，C2A によってもたらされた大きな効果といえる．

加えて，C2A を用いて複数の衛星搭載ソフトウェアを開発することで，過去の衛星の搭載ソフトウェアの理解や開発者との意思疎通が円滑となった点も C2A による効果として挙げられる．

具体的には，PROCYON の開発ではそれ以前に開発が行われた UNIFORM-1 やほどよし 3 号機・4 号機の搭載ソフトウェアの内容や動作を PROCYON の開発者が確認したり，過去の開発者と議論を行ったりといったような複数衛星間での意思疎通も円滑に行われた．仮に C2A が存在せず，これらの衛星の搭載ソフトウェアが独立に開発されたとするとこのような効果を得ることは困難であったと予想され，これも C2A という枠組みによってもたらされた効果である．

7.4.2 開発作業のパターン化

C2A を衛星の搭載ソフトウェア開発に適用した効果として，開発者が実施すべき作業手順がパターン化され理解が容易となり，開発を効率的に進められた点が挙げられる．

C2A 上で搭載ソフトウェアに機能を追加する手順はモード関連の作業でも，イベント処理関連の作業であっても，

1. 追加する機能に対応するアプリケーションの作成
2. 作成したアプリケーションのアプリケーション管理機能への登録
3. 追加したアプリケーションを呼び出すブロックコマンドの作成
4. 定義テーブルへのブロックコマンドの登録

という一定のパターンに沿って実現できる．これは C2A が全ての必須機能をブロックコマンドに集約して実現したことで得られた効果である．

このような作業手順のパターン化は，特に，複数の開発者で開発を実施した PROCYON の搭載ソフトウェア開発において，

- 開発者が C2A の使い方を容易に理解できる
- 開発作業が明瞭で効率的に作業できる

といった効果をもたらした．

C2A は必須機能の動作を全てブロックコマンドに集約して実現しているため，PROCYON で初めて C2A に触れる開発者であっても，動作の中心となるブロックコマンドの仕組みと，各必須機能がブロックコマンドをどのように呼び出すかを理解するだけでその動作を把握することができた．また，C2A 上で機能を追加する手順がパターン化されることで，開発者は作業をミスや出戻りなく実施することができた．このような効果は PROCYON の搭載ソフトウェア開発の円滑化に大きく貢献している．

上記に加え，C2A による作業のパターン化としては，アプリケーションの開発作業自体のパターン化を挙げることができる．

C2A では開発者が開発したアプリケーションをコア機能から呼び出すためには，開発したアプリケーションをアプリケーション管理機能に登録する必要があり，この登録を実現するため，アプリケーション管理機能はアプリケーションが備えるべき要素を規定している．このように，アプリケーションが備えるべき要素が明確に規定されたことで，C2A に基いて開発された衛星の搭載ソフトウェアでは，各アプリケーションの実装であるソースコードの記述がパターン化された．

これは開発者に対して，

- 実装作業を効率的に進められる
- 他人が作成したアプリケーションを容易に理解できる
- 過去の衛星のアプリケーションを容易に理解できる

といった効果をもたらした．

開発者は担当するアプリケーションを実装する際に，アプリケーション管理機能によって定められたインターフェースを満たすソースコードを作成すればよく，効率的に作業を進められた．また，ソースコードがパターン化される事によって，複数人で開発を進める場合にも自分以外の開発者が作成したソースコードを理解することが容易になり，同一衛星だけでなく，過去に C2A で開発されたアプリケーションの内容を理解することも可能となる．これらは特にほどよし 3 号機・4 号機の後に開発が行われた PROCYON において大きな効果を発揮した．

7.4.3 開発作業の自動化

前節では，C2A を衛星の搭載ソフトウェア開発に適用した効果として，各種作業のパターン化を取り上げた．C2A を適用して開発した 4 機の衛星ではこのパターン化の効果を活かして，一部搭載ソフトウェアのソースコード生成が自動化されている [68)67)]．これも C2A によって得られた効果の一つである．

前節でも述べた通り，C2A では各衛星毎に実装するアプリケーションはアプリケーション管理機能に登録された上でコア機能から呼び出される．この登録を可能にするため，アプリケーション管理機能はアプリケーションが備えるべき要素を規定している．アプリケーションを登録するためのインターフェイスが明確化されているので，これに沿ってアプリケーションを実装してしまえば，登録作業自体は単純な作業となっている．

C2A を適用した 4 衛星では，このアプリケーション管理機能へのアプリケーション登録作業のうち，特にコマンド定義テーブルとテレメトリ定義テーブルへの登録作業を自動化する仕組みを構築し，活用している．

コマンドとテレメトリの登録作業を自動化した理由としては，前述したように C2A によって搭載ソフトウェア側での作業がパターン化され単純な作業となったことに加え，衛星運用に必要となる地上局との連携の観点が挙げられる．

衛星のコマンドとテレメトリはともに衛星とそれを運用する地上局との間でやりとりを行うデータであり，このやりとりを実現するためには衛星と地上局の間でコマンドとテレメトリの定義が共有されていなければならない．例えば，衛星側に新しいコマンドを追加した場合でも，この追加したコマンドの情報を地上側に反映しなければ，地上側は衛星にコマンドを送出することができず，追加したコマンドを利用することはできない．衛星搭載ソフトウェア開発の過程では，この両者を常に一致させることが非常に重要となる．

C2A を適用して開発を行った 4 衛星では，この地上局との連携も含めたコマンドとテレメトリ追加作業の自動化を実施している．図 7.5 に具体的な処理の流れを示す．この自動化ではまず，衛星のテレメトリとコマンドの定義をソースコードとして記述するのではなく，定義内容をより人間が理解しやすく編集も簡単な形式の定義シートとして記述する．定義シートの形式としては，ここでは Excel による表形式を採用している．衛星側でコマンド・テレメトリを定義するソースコードと地上側で運用ソフトが読み込むコマンドテレメトリ定義はこの定義シートからそれぞれ専用の変換ソフトウェアによって生成される．衛星側と地上側どちらも共通の定義シートを元に各種ファイルを生成することで，両者の内容が確実に一致する環境を構築している．

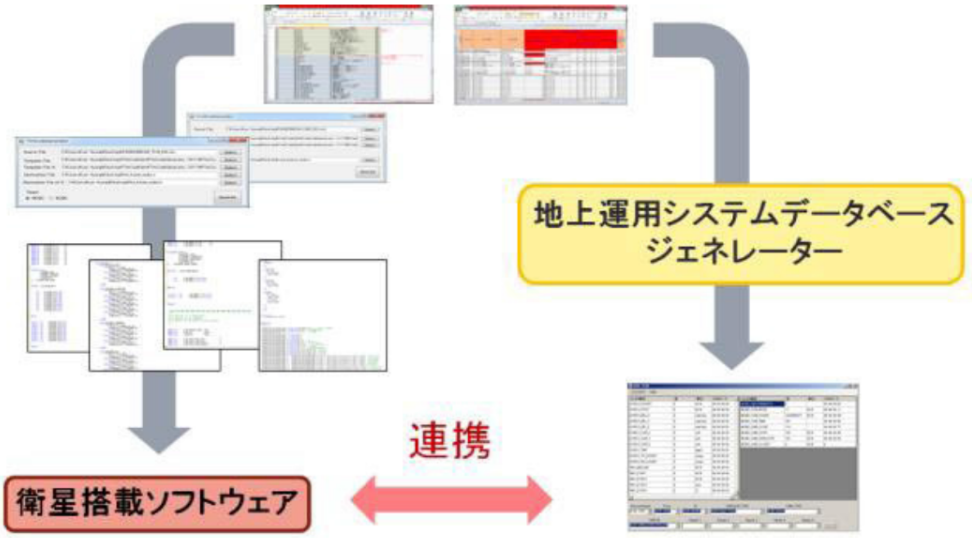
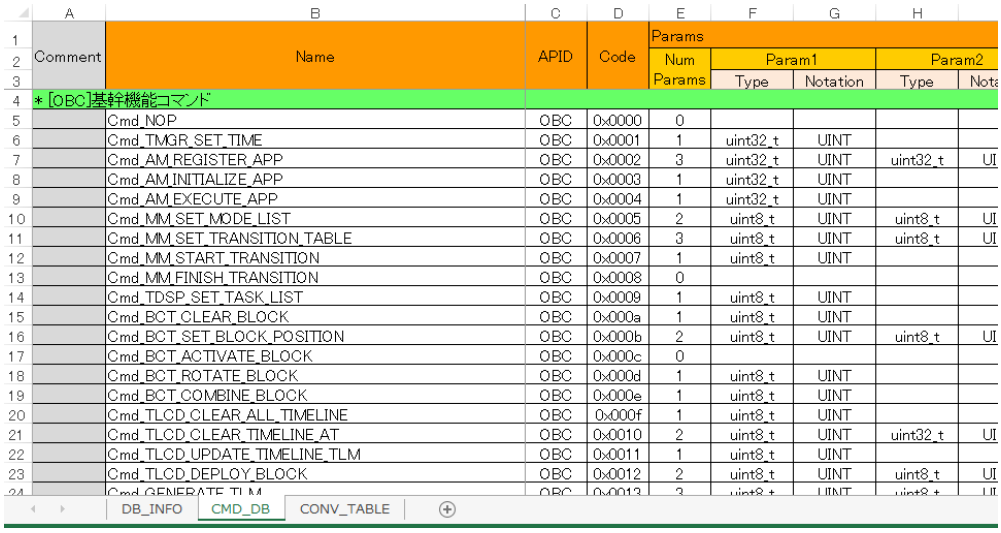


図 7.5　テレメトリ・コマンド定形コード自動生成の流れ [68)]

図 7.6，図 7.7 に PROCYON の開発に使用した定義シートを具体例として示す．

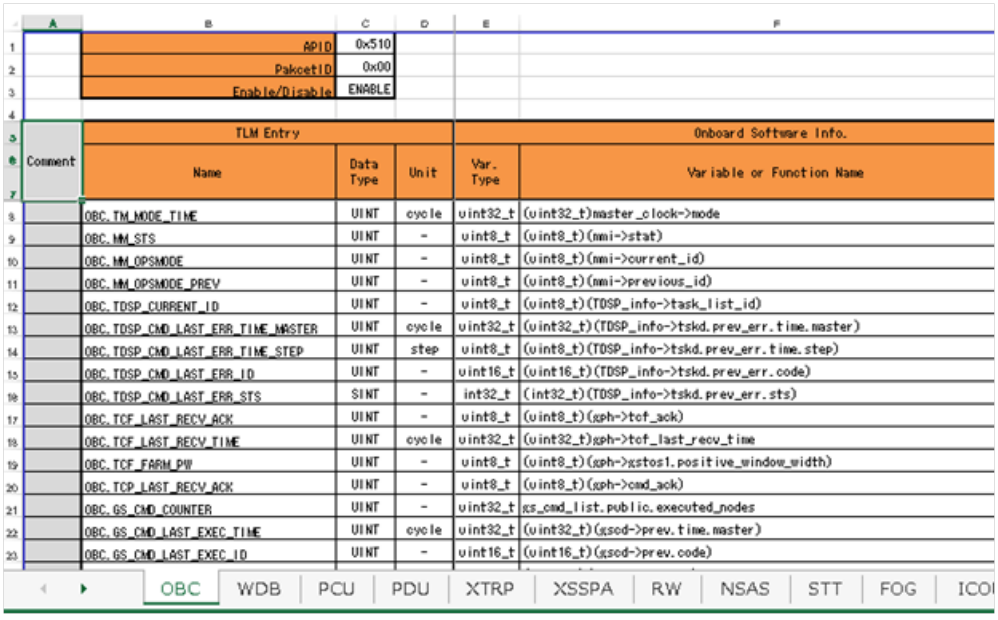
この自動化は，単純に衛星へのコマンドとテレメトリの追加・削除や変更が容易になるだけでなく地上側との内容同期も確実となり，開発者の不注意による作業ミスを取り除くことができ，開発の効率化に大きく貢献した．これも C2A によって搭載ソフトウェアの構成が明確化されたことで得られた効果である．



	Comment	Name	APID	Code	Params				
					Num Params	Param1		Param2	
						Type	Notation	Type	Nota
4	* [OBC]基幹機能コマンド								
5		Cmd_NOP	OBC	0x0000	0				
6		Cmd_TMGR_SET_TIME	OBC	0x0001	1	uint32_t	UINT		
7		Cmd_AM_REGISTER_APP	OBC	0x0002	3	uint32_t	UINT	uint32_t	UI
8		Cmd_AM_INITIALIZE_APP	OBC	0x0003	1	uint32_t	UINT		
9		Cmd_AM_EXECUTE_APP	OBC	0x0004	1	uint32_t	UINT		
10		Cmd_MM_SET_MODE_LIST	OBC	0x0005	2	uint8_t	UINT	uint8_t	UI
11		Cmd_MM_SET_TRANSITION_TABLE	OBC	0x0006	3	uint8_t	UINT	uint8_t	UI
12		Cmd_MM_START_TRANSITION	OBC	0x0007	1	uint8_t	UINT		
13		Cmd_MM_FINISH_TRANSITION	OBC	0x0008	0				
14		Cmd_TDSP_SET_TASK_LIST	OBC	0x0009	1	uint8_t	UINT		
15		Cmd_BCT_CLEAR_BLOCK	OBC	0x000a	1	uint8_t	UINT		
16		Cmd_BCT_SET_BLOCK_POSITION	OBC	0x000b	2	uint8_t	UINT	uint8_t	UI
17		Cmd_BCT_ACTIVATE_BLOCK	OBC	0x000c	0				
18		Cmd_BCT_ROTATE_BLOCK	OBC	0x000d	1	uint8_t	UINT		
19		Cmd_BCT_COMBINE_BLOCK	OBC	0x000e	1	uint8_t	UINT		
20		Cmd_TLCD_CLEAR_ALL_TIMELINE	OBC	0x000f	1	uint8_t	UINT		
21		Cmd_TLCD_CLEAR_TIMELINE_AT	OBC	0x0010	2	uint8_t	UINT	uint32_t	UI
22		Cmd_TLCD_UPDATE_TIMELINE_TLM	OBC	0x0011	1	uint8_t	UINT		
23		Cmd_TLCD_DEPLOY_BLOCK	OBC	0x0012	2	uint8_t	UINT	uint8_t	UI
24		Cmd_GENERATE_TLM	OBC	0x0013	3	uint8_t	UINT	uint8_t	UI

DB_INFO | CMD_DB | CONV_TABLE

図 7.6　コマンド定義シート (PROCYON の例)



APID	0x510	
PakcetID	0x00	
Enable/Disable	ENABLE	

Comment	TLM Entry			Onboard Software Info.	
	Name	Data Type	Unit	Var. Type	Variable or Function Name
	OBC_TM_MODE_TIME	UINT	cycle	uint32_t	(uint32_t)master_clock->mode
	OBC_MM_STS	UINT	-	uint8_t	(uint8_t)(mmi->stat)
	OBC_MM_OPSMODE	UINT	-	uint8_t	(uint8_t)(mmi->current_id)
	OBC_MM_OPSMODE_PREV	UINT	-	uint8_t	(uint8_t)(mmi->previous_id)
	OBC_TDSP_CURRENT_ID	UINT	-	uint8_t	(uint8_t)(TDSP_info->task_list_id)
	OBC_TDSP_CMD_LAST_ERR_TIME_MASTER	UINT	cycle	uint32_t	(uint32_t)(TDSP_info->tskd.prev_err.time.master)
	OBC_TDSP_CMD_LAST_ERR_TIME_STEP	UINT	step	uint8_t	(uint8_t)(TDSP_info->tskd.prev_err.time.step)
	OBC_TDSP_CMD_LAST_ERR_ID	UINT	-	uint16_t	(uint16_t)(TDSP_info->tskd.prev_err.code)
	OBC_TDSP_CMD_LAST_ERR_STS	SINT	-	int32_t	(int32_t)(TDSP_info->tskd.prev_err.sts)
	OBC_TCF_LAST_RECV_ACK	UINT	-	uint8_t	(uint8_t)(xph->tcf_ack)
	OBC_TCF_LAST_RECV_TIME	UINT	cycle	uint32_t	(uint32_t)xph->tcf_last_recv_time
	OBC_TCF_FARM_PW	UINT	-	uint8_t	(uint8_t)(xph->xstos1.positive_window_width)
	OBC_TCP_LAST_RECV_ACK	UINT	-	uint8_t	(uint8_t)(xph->cmd_ack)
	OBC_GS_CMD_COUNTER	UINT	-	uint32_t	gs_cmd_list.public.executed_nodes
	OBC_GS_CMD_LAST_EXEC_TIME	UINT	cycle	uint32_t	(uint32_t)(xsod->prev.time.master)
	OBC_GS_CMD_LAST_EXEC_ID	UINT	-	uint16_t	(uint16_t)(xsod->prev.code)

OBC | WDB | PCU | PDU | XTRP | XSSPA | RW | NSAS | STT | FOG | ICO

図 7.7　テレメトリ定義シート (PROCYON の例)

7.4.4　SILS 環境の構築

C2A を適用して搭載ソフトウェアを開発することで得られた効果として，Software In the Loop Simulator(以下 SILS) を用いた姿勢系ソフトウェアの検証が挙げられる．

C2A では衛星に実装する各アプリケーションのハードウェア依存性を局所化するために，デー

タの流れに沿ってアプリケーションを分割して実装している．これによって衛星毎に生じるハードウェア構成が個々のアプリケーションに実際に局所化された点については，再利用性の評価の項ですでに述べた．

アプリケーションをデータの流れにそって分割して実装することによって，衛星のハードウェアに依存するアプリケーションがより高度な処理を実行するアプリケーショと明確に分離したことで，再利用性が高まったことに加えてハードウェアに依存するアプリケーションのみを PC 上で模擬することで容易に SILS 環境を構築することが可能となった．構築した SILS 環境では図 7.8 に示す通り，実際に OBC 上で動作するのと同一のアプリケーションを用いて姿勢決定や制御量計算を実行することが可能であり，これらのアプリケーションの検証を確実に行うことが出来る．

PROCYON では SILS を活用することで姿勢系の搭載ソフトウェア検証を効率的に実施することが可能となり，これは開発の効率化に大きく貢献した．実際に運用で得られた制御結果も地上試験の結果を高い確度で一致しており [69]，このような環境を構築して試験を実施することの有効性を示している．

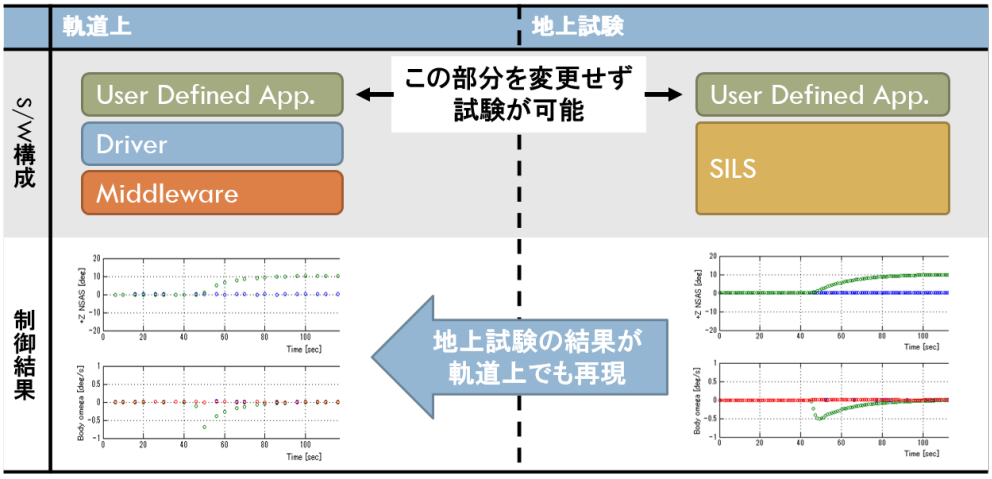


図 7.8　SILS を用いた搭載ソフトウェアの検証 [69]

このような SILS 環境が容易に構築出来た点も C2A の明確な枠組みによってもたらされた効果の一つである．

7.5　開発期間短縮の実例

前節まででは，C2A を適用して搭載ソフトウェア開発を行うことで得られる様々な効果について具体例を示しながら説明を行ってきた．本節では C2A によって得られるこれらの効果が総合的に作用した結果得られた効果として，搭載ソフトウェアの開発期間の短期化について具体例を示す．

超小型深宇宙探査機 PROCYON は UNIFORM-1，ほどよし 3 号機・4 号機の搭載ソフトウェア開発を通じて構築された C2A を最初から適用して搭載ソフトウェアを開発した初めての超小型

宇宙機である．3 機の衛星の開発を経て，導出・体系化された C2A に基づいて実際の衛星・探査機の搭載ソフトウェアを開発することで，開発期間の短期化についてどのような効果が得られるかを評価するという観点で PROCYON の開発は非常に良いサンプルと考えられる．

図 7.9 に PROCYON の開発の大まかな流れをまとめる．PROCYON では搭載ソフトウェアの開発にバージョン管理システムの一種である Subversion を利用している．この図は Subversion に記録された PROCYON の開発履歴を元に作成したものである．

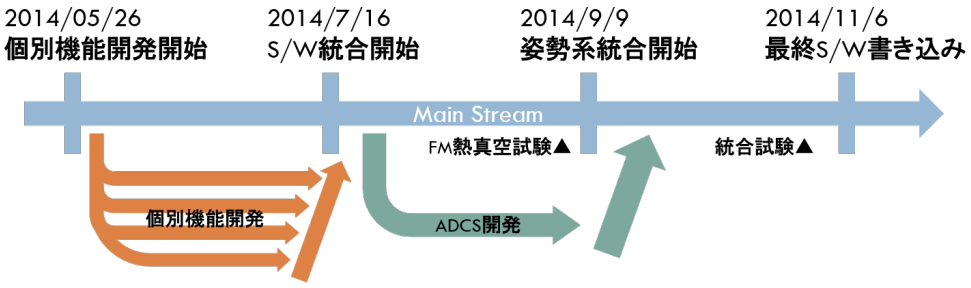


図 7.9　PROCYON のソフトウェア開発実績の概要

PROCYON の搭載ソフトウェア開発は 2014 年 5 月 26 日に始まり，探査機をロケット側に引き渡す直前の 2014 年 11 月 6 日に終了した．開発期間はおおよそ 5.5 ヶ月である．

搭載ソフトウェアの開発は

- 個別機能開発・統合 (約 1.5 ヶ月)
- 統合機能試験と姿勢系開発・統合 (約 2 ヶ月)
- 最終統合試験 (約 2 ヶ月)

の 3 段階で実施された．

個別機能開発とは，衛星に搭載される各機器と OBC との間の通信を実現するアプリケーションの開発であり，ここで各機器と OBC を一対一で接続した動作確認が行える環境を整え，その後，各機器のアプリケーションを統合した．

個別機能の統合が完了した後は，統合機能試験と姿勢系開発を平行して実施した．統合機能試験では，個別機能を統合した搭載ソフトウェアを用いて熱真空試験などの各種試験を実施し，各機器の動作確認や個別機能のバグ出しを実施し，姿勢系開発では，個別機能として実装した各アプリケーションを利用して姿勢決定や姿勢制御を実施するアプリケーションの開発を行った．

統合機能試験が終了した段階で，平行して開発を行っていた姿勢系のアプリケーションを統合し，実際に探査機に搭載するソフトウェアを構成し，最終統合試験として打ち上げ直後の動作確認や運用期間中の動作確認などを実施し，ロケット側へ引き渡した．

これまで開発が行われた超小型衛星において，搭載ソフトウェア開発にどの程度の期間を要したかを議論することは難しいが，当研究室が過去に開発した衛星に関しては，開発に数年程度の期間を要しており，これに対して 5.5 ヶ月という PROCYON の搭載ソフトウェア開発期間は十分に短

いと言える．このような短期間での搭載ソフトウェア開発が可能となった要因を以下に整理する．

まず，搭載ソフトウェアの開発開始時点で C2A という明確な枠組みが存在していたことが開発期間の短縮に大きく貢献していると考えられる．C2A が存在しない場合は C2A が提供する搭載ソフトウェアのコア機能に対応する機能を独自に用意する必要が生じる．コア機能は衛星の挙動を実現するのに必要な機能であるから，これが用意できるまでは衛星の個々の動作を実現するソフトウェアの開発を進めることは難しく，開発期間が長期化する大きな要因となる．

次に，C2A が提供するコア機能が十分に整理されている点も開発期間の短縮に貢献していると考えられる．衛星の必須機能から共有するパターンを抽出し，それをブロックコマンドという形に集約して実装した C2A のコア機能はその動きを理解しやすく，アプリケーションの開発者が作業を容易に進めることができる．C2A が存在せず，必須機能が個別に用意された場合はこのような特性は得られず開発期間が長期化する可能性が考えられる．

また，C2A を用いて搭載ソフトウェアの開発を行う場合に衛星固有の機能がアプリケーションという形に整理される点も重要である．開発をアプリケーション単位で行えることで複数開発者での作業の分担が可能になり，その後の統合も円滑に実施できる．PROCYON の搭載ソフトウェア開発において，個別機能の開発を同時並行に実施して統合した過程や，姿勢系の開発を統合機能試験と平行して実施して統合した過程は全体の開発期間短縮に大きく寄与している．C2A が存在しない場合，作業の分担・統合が円滑に行えず開発期間が長期化する可能性が考えられる．

C2A によって搭載ソフトウェアの検証が容易となっている点も開発期間の短期化に寄与している．C2A が明確な枠組みを提供することで，搭載ソフトウェアの動作を理解することが容易となり，また再現性も高いことから不具合発生時の原因究明や対処が容易となる．加えて，ソフトウェアの構成が整理されることで SILS や HILS といった検証環境の構築も容易となる．

最後に，C2A によって柔軟な軌道上再構成能力が獲得された点も開発期間の短期化に寄与している．柔軟な再構成能力によって PROCYON でのモード維持処理の再構築の例のように，地上で時間を確保できなかった要素を軌道上で修正する手段が存在することで，限られた期間で開発を完了させる場合の優先度設定に柔軟性を与えることが可能となる．

PROCYON はその開発期間がおよそ 1 年というこれまでの超小型衛星・探査機の中でも短いものであったが，C2A を適用して搭載ソフトウェアの開発を実施したことにより開発を 5.5 ヶ月という短期間で行えたことは C2A の非常に大きな成果であり，C2A の優位性を示している．

7.6　本研究の適用範囲

本研究では超小型衛星が直面している問題を起点とし，これを搭載ソフトウェア面から解決する手法として C2A を提案している．前節まででは，C2A を実際の超小型衛星の搭載ソフトウェア開発に適用した結果を評価し，提案手法が超小型衛星の問題解決に有効であることを示した．本節ではこれらの結果を元に，C2A を超小型衛星以外の範囲へ適用することが可能か，また適用することに意味があるかを中心に本研究の適用範囲について議論する．

まず，超小型衛星に比較的近い領域である中・大型衛星の開発に C2A を適用する場合について

考える．本研究では実際に中・大型衛星の搭載ソフトウェアに対して C2A を適用した評価は実施していないため，超小型衛星の場合のような結果に基づく議論を行うことは不可能であるが，提案手法を用いてこれらの衛星の搭載ソフトウェアを開発することは可能と考えている．この理由として，超小型衛星と中・大型衛星において搭載ソフトウェアが実現している機能自体に大きな違いが無い点と，2 章で議論した中・大型衛星で行われている搭載ソフトウェアに対する取り組みと本研究が提案する手法とは競合するものではなく，共存が可能な点が挙げられる．ただし，地上での試験・検証が厳密になるに従い，本手法の利点は薄れていくと予想される．C2A の利点は搭載ソフトウェアに柔軟な軌道上再構成能力が付加される点にあるが，この能力が必要となる背景には地上で試験・検証を実施すると短期・低コストの利点を活かせない超小型衛星の特性と，地上での試験・検証期間を十分に確保できない場合が多い超小型衛星の現状がある．中・大型衛星のミッションの中には，例えば有人ミッションのような，軌道上での不具合発生が即時にミッション失敗につながるものも存在する．このような場合には地上での確実な試験・検証が必要であり，C2A を搭載ソフトウェア開発に適用しても利点は少ないであろう．

次に，衛星ではない地上の組み込みシステムの開発に C2A を適用する場合を考える．これに関しては以下に述べる通り，本研究の利点を必要とする分野は少ないように感じられる．C2A は打ち上げ後は基本的に動作を停止することができない衛星に対して，全体を停止することなく実施出来る動的な再構成能力を提供している．一方地上では，ソフトウェアの更新が必要であれば一旦システムを停止して書き換えを行い，その後再び動作を開始させる静的な再構成を容易に選択できる場合が多い．このため仮に C2A を適用して開発を行ったとしても，その利点が活かされる状況は少ないものと予想される．

以上のような C2A の特性を踏まえると，C2A は長期にわたって動作を継続し，随時更新・調整を実施するような用途においてその有効性が発揮されるものと考えられる．

第 8 章

結論と今後の課題

8.1 結論

8.1.1 本研究が提案した手法

近年，特に実用超小型衛星において衛星の高信頼化の要求が高まり，短期・低コスト開発という超小型衛星の特徴を維持することが難しくなってきている．この要求と特徴とを両立させるためには「高い再利用性」と「柔軟な軌道上再構成能力」の 2 つが重要であり，本研究ではこれらを衛星搭載ソフトウェアの側から実現する手法を提案した．

8.1.2 本研究がもたらした利点

衛星搭載ソフトウェアの構成を明確化し，過去の衛星で使用された実績のあるソフトウェアの再利用が可能となる環境を整えることで，開発期間の大幅な短縮と信頼性の向上を実現した．また，軌道上再構成能力の強化によって搭載ソフトウェアに柔軟性をもたらすことで，軌道上で生じる不測の事態への対応能力を強化したことに加え，従来地上で行ってきた試験の一部を軌道上で実施するといった方針を積極的に検討することが可能となる環境を構築した．特に打ち上げ機会が限定されている超小型衛星では，本研究を適用することでもたらされる地上開発期間の短縮は大きな利点である．

8.1.3 本研究が解決した課題

再利用性と軌道上再構成能力の高い搭載ソフトウェアは以下の 3 つの性質を備える必要がある．

1. **体系性** - 地上での開発を明確な枠組みに沿って行えること
2. **柔軟性** - 軌道上再構成を地上と同様の手順・粒度で行えること
3. **局所性** - 衛星毎の差異を特定の箇所に限定できること

本研究では上記の性質を備えた搭載ソフトウェアを実現するため，以下に示す 3 つの課題を設定し，Command Centric Architecture (C2A) を導出することでこれらを解決した．

- 衛星の必須機能と再構成能力とを融合させる仕組みの導出
- 必須機能を実現するアーキテクチャの構築
- アーキテクチャに基づく開発方針の整理

また，本研究では導出した C2A を実際の超小型衛星 4 機の搭載ソフトウェアに適用して開発・運用を実施し，その結果の評価を行っている．

8.1.4 衛星の必須機能と再構成能力とを融合させる仕組みの導出

衛星の動きを実現する必須機能は「コマンド処理機能」,「モード管理機能」,「イベント処理機能」の 3 機能で構成され，これらは全て「予め用意された処理の塊を展開し，決まったタイミングで実行する」という同一のパターンを共有している．軌道上再構成能力とは衛星の動きを軌道上で変更する能力であり，この柔軟性を高めるためには，このパターンに基づき「処理の塊」と「実行のタイミング」を軌道上でも地上と同一の手順・粒度で編集可能にする一貫した仕組みを用意することが重要となる．

論文中では以上の観点にもとづいて，3 つの必須機能全てを「コマンド処理機能」，特に「ブロックコマンド」に集約することで衛星の振る舞いを軌道上でも柔軟に変更可能とする仕組みを提案した．

必須機能と抽出したパターンの妥当性については，提案した仕組みを実際に備えた超小型衛星の開発とその運用を通じて問題ないことを確認している．

8.1.5 必須機能を実現するアーキテクチャの構築

概念的で抽象度の高い「仕組み」を衛星の搭載ソフトウェアとして具体化するにあたって必要となる補助的な機能を整理し，提案した仕組みに基づく必須機能と合わせて搭載ソフトウェアの枠組みを Command Centric Architecture (C2A) として明確化した．

明確化の過程では各機能が実現すべき処理を規定するとともに，様々な衛星に C2A を適用する前提に立ち，処理の細部を調整して各衛星毎の違いを吸収するパラメータについても議論を行っている．

明確化した C2A については，実際に複数の衛星・探査機に適用した結果を通じて，宇宙機毎に機能の追加や変更を実施すること無く，パラメータの調整のみで異なる衛星へ適用できることを確認している．

8.1.6　アーキテクチャに基づく開発方針の整理

提案する C2A を個々の衛星の搭載ソフトウェア開発に適用する場合を念頭に「安全性」「軌道上再構成能力」「再利用性」を高める上で考慮すべき点について議論した．

軌道上再構成能力の強化は衛星の柔軟性を高めると同時に，誤った操作によって衛星を再起不能とするリスクも高めている．衛星の安全性を確保するためには，C2A を搭載する計算機の外部に独立して計算機のリセットを実施できるハードウェアの用意が必須である．

C2A が提供する柔軟な軌道上再構成能力を最大限活用するためには，予め搭載ソフトウェアに余裕を確保しておくことが重要となる．C2A での余裕とは各種定義テーブルの余裕であり，これらを用意しておくことが重要となる．

搭載ソフトウェアの再構成能力が強化されることで，衛星の機能を柔軟に組み替えることが可能となるが，個々の衛星を超えて複数衛星でソフトウェアの再利用を実現するためには提案アーキテクチャ上で衛星の機能を実装する際に，データの流れに対応させて機能を分割することが必要である．

8.2　今後の課題

8.2.1　アプリケーション管理の自動化

本研究で提案した C2A によって，衛星が実行する個々の処理はアプリケーションの形で明確化され，異なる衛星で同様の処理を行う場合にはアプリケーションを再利用することが可能となった．

このように再利用の枠組みは整ったといえるものの，過去の衛星で使用したアプリケーションを別の衛星で再利用しようとした場合，その作業は現状では人力に頼っており，場合によっては非常に煩雑なものとなっている．

作業が煩雑化するのは，アプリケーション同士に依存関係が存在する場合である．例えば，太陽指向を行うアプリケーションを再利用する場合を例に取ると，太陽指向の計算を行うアプリケーションを動作させるためには，それ単体だけでなく，センサの値を取得するアプリケーションや，アクチュエータに指令を送るアプリケーション等の関連するアプリケーションも合わせて用意する必要がある．

今後，多数の衛星で効率的に再利用を行っていくためには多くのソフトウェアで利用されているパッケージ管理機能のような，アプリケーションを一元管理し，その依存関係を自動的に解決するような仕組みが必要であろう．

8.2.2　搭載ソフトウェア生成の自動化

C2A が搭載ソフトウェアの構成を明確化したことによって搭載ソフトウェアの多くの部分がソースコードのレベルではパターン化された．論文中でも述べた通り，既に一部についてはパター

ン化したコードの自動生成が実現されているが，まだ限定的である．

パターン化した部分については，可能な限り自動生成を行うことで単純なミスを軽減しより効率的に開発を進めることができる．このような環境を引き続き構築していくことが重要である．

8.2.3 搭載ソフトウェア調整の自動化

コード生成だけでなく，搭載ソフトウェアの動作調整についても自動化が望まれる．

アーキテクチャを適用して複数の衛星を開発する過程で得た知見として，通常ユーザーがアプリケーションを実行する場合に明確に指定出来るのはアプリケーション自身の実行頻度と各アプリケーションの実行順序だけであり，それぞれのアプリケーションの実行に要する時間は予測できない場合が多い．通常は逆に予め各アプリケーションを実行する時間枠を全体が 1 周期に収まるように配分し，それに収まるようアプリケーションを開発し，どうしても収まらない場合は他のアプリケーションと時間配分を調整するといった方針が取られるが，これは非常に煩雑な作業となる．

アプリケーションの実行タイミングは軌道上再構成能力を用いて変更できるパラメータであり，ユーザーが指定する周期・順序で各アプリケーションが動作するよう実行タイミングを調整する作業は自動化できる．このようなソフトウェアの動作調整の自動化もアーキテクチャに基づく開発を効率化する上で重要である．

謝辞

まず初めに，本研究の指導教官である中須賀真一教授に感謝致します．中須賀教授には本研究だけでなく，研究室に所属した学部 4 年から実に 6 年間に渡り常に丁寧な指導を頂きました．本研究においても直接的な指導だけではなく，「ほどよし 3 号機」「ほどよし 4 号機」といった実際の超小型衛星の開発に参加する機会を与えて頂き，研究内容の実証も含め非常に貴重な経験を積むことが出来ました．

次に，副査として貴重な時間を割いて本論文の指導をしてくださった堀浩一教授，船瀬龍准教授，矢入健久准教授，東京理科大学の木村真一教授に感謝致します．副査の先生方には特に論文をまとめる過程を中心に様々な観点から建設的な指摘を頂きました．また，船瀬准教授，木村准教授には本研究に関連する超小型衛星の開発プロジェクトを通じて，研究と直接は関係しない部分も含め多くの助言を頂きました．

博士課程では，東京大学大学院工学系研究科博士課程学生特別リサーチ・アシスタント (SEUT) の援助を頂きました．感謝致します．

あまりに人数が多く，それぞれの名前を挙げることは叶いませんが，「UNIFORM-1」「ほどよし 3 号機」「ほどよし 4 号機」のプロジェクトの皆様に感謝致します．

「PROCYON」の開発をはじめとする多くのプロジェクト，研究に関連する議論，日々の生活など多くの面で支えてくれた研究室の皆様にも感謝致します．

最後に，ここまでの学生生活をサポートしてくれた家族・親戚に心から感謝致します．

参考文献

1) 中村揚介, 平子敬一, “小型実証衛星 1 型の研究開発”, IEICE Technical Report SANE2007-42 (2007-6) pp.5-9, 2007
2) Anthony Shao, Elizabeth A. Koltz, et al., “Quantifying the Cost Reduction Potential for Earth Observation Satellites”, 12th Reinventing Space Conference, 2014
3) 中須賀真一, “超小型衛星コンソーシアムによる新しい宇宙開発・利用のパラダイムを目指して”, IEICE Technical Report SANE2010 − 49（2010 − 06）, 2010
4) 舟根司, 佐々木史記, et al., “CubeSat-XI(サイ) の運用成果と東大における今後の超小型衛星計画について”, 日本機械学会 2004 年度年次大会講演論文集 pp.469-467, 2004
5) 田中利樹, 中須賀真一, “超小型リモートセンシング衛星 PRISM(ひとみ) の開発と運用”, 日本機械学会 2009 年度年次大会講演論文集 pp.249-250, 2009
6) 稲守考哉, 酒匂信匡, et al., “超小型天文位置衛星 Nano-JASMINE のバスシステムについて”, 第 58 回宇宙科学技術連合講演会講演集 JSASS-2014-4210, 2014
7) 石井亮介, 佐原宏典, “超小型衛星のミッション成否分析に基づく現状把握と将来予測”, 日本航空宇宙学会論文集 Vol.60 No.2 pp.65-73, 2012
8) 石井亮介, 佐原宏典, “統計的手法を用いた大学衛星の現状分析”, JSASS-2012-4221, 2012
9) 宇宙航空研究開発機構, “平成 20 年度夏季打ち上げの H-IIA ロケットに相乗りする小型副衛星の選定結果について”, JAXA プレスリリース http://www.jaxa.jp/press/2007/05/20070516_sac_smallsat.pdf, 2007
10) 宇宙航空研究開発機構, “PLANET-C に相乗りする小型副衛星の選定結果について”, JAXA プレスリリース http://www.jaxa.jp/press/2008/07/20080709_sac_sat.pdf, 2008
11) 宇宙航空研究開発機構, “平成 23 年度打ち上げの H-IIA ロケットに相乗りする小型副衛星の選定結果について”, JAXA プレスリリース http://www.jaxa.jp/press/2010/10/20101006_sac_subpayload.pdf, 2010
12) 宇宙航空研究開発機構, “GPM 相乗り公募小型副衛星の選定結果について”, JAXA プレスリリース http://www.jaxa.jp/press/2011/12/20111214_sac_subpayload.pdf, 2011
13) 宇宙航空研究開発機構, “ALOS-2 相乗り公募小型副衛星の選定結果について”, JAXA プレスリリース http://www.jaxa.jp/press/2012/03/20120328_sac_smallsat.pdf, 2012
14) 宇宙航空研究開発機構, “はやぶさ 2 相乗り公募小型副ペイロードの選定について”, JAXA プ

レスリリース http://aerospacebiz.jaxa.jp/jp/ainori/data/topics_130913.pdf, 2013
15) 宇宙航空研究開発機構, “ASTRO-H 相乗り公募超小型衛星の選定・契約について”, JAXA プレスリリース http://aerospacebiz.jaxa.jp/jp/topics/data/140827.pdf, 2014
16) 中須賀真一, 鶴田佳宏, “ほどよしプロジェクトの成果をベースにした超小型衛星のビジョン”, 第 58 回宇宙科学技術連合講演会 JSASS-2014-4034, 2014
17) Mengu Cho, Hirokazu Masui, Toru Hatamura, Koichi Date, Shigekatsu Horii, Shoichi Obata, “Overview of Nano-satellite Environmental Tests Standardization Project: Test Campaign and Standard Draft”, 26th Annual AIAA/USU Conference on Small Satellites SSC12-VII-10, 2012
18) 船瀬龍, 川勝康弘, PROCYON プロジェクトチーム, “はやぶさ 2 相乗り超小型深宇宙探査機 PROCYON の開発状況”, 第 58 回 宇宙科学技術連合講演会 JSASS-2014-4472, 2014
19) “TOPPERS プロジェクト プロジェクトの紹介と参加のお誘い”, TOPPERS プロジェクト https://www.toppers.jp/docs/intro-invite.pdf, 2013
20) “Wind River Systems 社 Web ページ”, http://www.windriver.com/products/vxworks/
21) “RTAI - the RealTime Application Interface for Linux Web ページ”, https://www.rtai.org/
22) “AUTOSAR Web ページ”, http://www.autosar.org/
23) 徳田 昭雄, 田村 太一, “車載ソフトウェアの標準化と AUTOSAR の動向”, 立命館経営学 Vol.45 No.5 pp.153-169, 2007
24) Paul J. Prisaznuk, “ARINC 653 ROLE IN INTEGRATED MODULAR AVIONICS (IMA)”, 27th Digital Avionics Systems Conference, 2008
25) 平山芳和, 井上禎一郎, et al., “人工衛星搭載ソフトウェア開発へのモデルベース開発の適用”, JSASS-2012-4483, 2012
26) The European Cooperation for Space Standardization, “SpaceWire - Links, nodes, routers and networks”, ECSS-E-ST-50-12C, 2008
27) 高橋忠幸, 能町正治, et al., “SpaceWire にもとづく衛星アーキテクチャ”, 第 57 回宇宙科学技術連合講演会講演集 JSASS-2013-4072, 2013
28) Jim Lyke, Don Fronterhouse, et al., “Space Plug-and-Play Avionics”, 3rd Responsive Space Conference RS3-2005-5001, 2005
29) 石井宏宗, 小林高士, et al., “利用衛星を目指した SpaceWire の高信頼化設計”, 第 58 回宇宙科学技術連合講演会講演集 JSASS-2014-4361, 2014
30) 入門朋子, 石濱直樹, et al., “宇宙機搭載リアルタイム OS の高信頼性化”, 第 55 回宇宙科学技術連合講演会講演集 JSASS-2011-4287, 2011
31) 石川拓也, 本田晋也, 高田広章, “静的なメモリ配置を行うメモリ保護機能を持ったリアルタイム OS”, コンピュータソフトウェア Vol.29 No.4 pp.161-181, 2011
32) 佐藤伸子, 石濱直樹, et al., “宇宙機搭載用リアルタイム OS に適用した高信頼化技術のハンドブック化”, 組み込みシステムシンポジウム 2011 pp.25-1-25-7, 2011

33) The European Cooperation for Space Standardization, “SpaceWire Remote memory access protocol”, ECSS-E-ST-50-52C, 2010
34) 高田広章, 高田光隆, et al., “SpaceWire のリアルタイム性保証手法と SpaceWire OS”, 第 56 回宇宙科学技術連合講演会講演集 JSASS-2012-4117, 2012
35) 平原大地, 笠原希仁, et al., “SpaceWire 統合型計算機の概要と開発ステータス”, 第 57 回宇宙科学技術連合講演会講演集 JSASS-2013-4074, 2013
36) Takahiro Yamada, “Generic Software for Spacecraft Testing and Operations Based on a spacecraft Model”, SpaceOps 2006 Conference AIAA 2006-5529, 2006
37) Takahiro Yamada, “Standardization of Spacecraft and Ground System Based on a Spacecraft Functional Model”, SpaceOps 2008 Conference AIAA 2008-3423, 2008
38) Takahiro Yamada, “Functional Model of Spacecraft (FMS)”, GSTOS 201-0.9, 2011
39) Takahiro Yamada, “Spacecraft Monitor and Control Protocol (SMCP)”, GSTOS 200-0.13, 2011
40) 山田隆弘, 松崎恵一, “衛星情報ベース 2 定義 (案)”, GSTOS 300-0.9, 2009
41) Takahiro Yamada, “Generic Software for Spacecraft Testing and Operations Based on a spacecraft Model”, AIAA 2006-5529, 2006
42) Takahiro Yamada, “Standardization of Spacecraft and Ground System Based on a Spacecrafpt Functional Model”, AIAA 2008-3423, 2008
43) 清水健介, “軌道上での姿勢システムの構築を前提とした衛星設計手法に関する研究”, 東京大学 平成 24 年度博士論文, 2012
44) 宇宙航空研究開発機構, “ソフトウェア開発標準 (宇宙機用)”, JERG-2-600A, 2011
45) 日笠元裕, 朝比奈寛之, 児玉義和, “JAXA ソフトウェア開発標準の活用”, 第 55 回宇宙科学技術連合講演会講演集 JSASS-2011-4290, 2011
46) 朝比奈寛之, 金子達哉, 宮本祐子, “改善指向のプロセスアセスメント技術”, 第 55 回宇宙科学技術連合講演会講演集 JSASS-2011-4291, 2011
47) 宇宙航空研究開発機構, “IV&V ガイドブック 【虎の巻】”, JAXA-SP-12-016, 2013
48) 川口真司, 早川浩司, 梅田浩貴, “ソースコード評価における指摘の水平展開”, 第 55 回宇宙科学技術連合講演会講演集 JSASS-2011-4281, 2011
49) 大熊浩示, 松本充広, 梅田浩貴, “IV&V 評価における状態遷移図の上位・下位間の整合性の確認方法”, 第 55 回宇宙科学技術連合講演会講演集 JSASS-2011-4282, 2011
50) 植田泰士, 有川善久, et al., “陸域観測技術衛星 2 号 (ALOS-2) におけるソフトウェア品質向上活動”, 第 55 回宇宙科学技術連合講演会講演集 JSASS-2011-4285, 2011
51) 大島武, 萩野慎二, 川口淳一郎, “小惑星探査機「はやぶさ」のシステム設計と自動化自律化機能”, 第 47 回宇宙科学技術連合講演会講演集 pp.1155-1158, 2003
52) 梅里真弘, 岡橋隆一, “小型ソーラ電力セイル「IKAROS」の開発”, NEC 技報 Vol.64 No.1 pp.46-49, 2011
53) Sotaro Kobayashi, Jun’ichi Takisawa, Shinichi Nakasuka, Shinichi Kimura, “Software De-

velopment Framework for Small Satellite On-board Computers", 29th ISTS, 2013
54) 小林宗太郎, 木村真一, "マルチプラットフォームに対応した小型衛星標準搭載ソフトウェアフレームワーク", 第 57 回宇宙科学技術連合講演会講演集 JSASS-2013-4727, 2013
55) Shusaku Yamaura, Seiko Shirasaka, et al., "UNIFORM-1: First Micro-Satellite of Forest Fire Monitoring Constellation Project", 28th Annual AIAA/USU Conference on Small Satellites SSC14-VI-2, 2014
56) 北海道大学, "UNIFORM-1 衛星で御嶽山噴火の観測に成功", 北海道大学 プレスリリース http://www.hokudai.ac.jp/news/141006_cris_pr.pdf, 2014
57) 間瀬一郎, 松井正安, et al., "ほどよし 3 号機、4 号機の開発状況", 第 56 回 宇宙科学技術連合講演会講演集 JSASS-2012-4479, 2012
58) 間瀬一郎, 松井正安, et al., "ほどよし 3 号機、4 号機の開発状況", 第 57 回 宇宙科学技術連合講演会講演集 JSASS-2013-4690, 2013
59) 鶴田佳宏, 中須賀真一, et al., "ほどよし 3・4 号機のシステム設計と運用成果", 第 58 回 宇宙科学技術連合講演会講演集 JSASS-2014-4218, 2014
60) 間瀬一郎, 松井正安, et al., "ほどよし 3 号機, 4 号機の軌道上初期性能", 第 58 回 宇宙科学技術連合講演会講演集 JSASS-2014-4218, 2014
61) 青柳賢英, 岩崎晃, et al., "ほどよし 3 号機, 4 号機地球観測ミッションの初期結果", 第 58 回 宇宙科学技術連合講演会講演集 JSASS-2014-4120, 2014
62) 松本健, 松井正安, et al., "ほどよし 3, 4 号機のデータ蓄積中継 (S&F) 実証実験概要", 第 58 回 宇宙科学技術連合講演会講演集 JSASS-2014-4112, 2014
63) 田中利樹, 中須賀真一, et al., "ほどよし 3 号 4 号 機器搭載スペースミッション初期運用成果の報告", 第 58 回 宇宙科学技術連合講演会講演集 JSASS-2014-4127, 2014
64) 山田紘平, 長野方星, et al., "超小型衛星用蓄熱パネルの開発とほどよし 4 号機による実証", 第 58 回 宇宙科学技術連合講演会講演集 JSASS-2014-4416, 2014
65) 田中康平, 鶴田佳宏, et al., "「ほどよし」衛星の技術実証モジュールを利用した新規蓄電池の軌道上実証", 第 58 回 宇宙科学技術連合講演会講演集 JSASS-2014-4518, 2014
66) 宇宙航空研究開発機構, "超小型深宇宙探査機「PROCYON(プロキオン)」の飛行状況について", JAXA プレスリリース http://www.jaxa.jp/press/2014/12/20141204_procyon_j.html, 2014
67) Tran Ngoc Lan Huong, Sotaro Kobayashi, Jun'ichi Takisawa, Shinichi Nakasuka, Shinichi Kimura, "Automatic Document Based Program Code Generation System for On-Board Software", 29th ISTS, 2013
68) Tran Ngoc Lan Huong, 小林宗太郎，et al., "搭載ソフトウェアと運用データベースの文書ベース連動プログラミングと文書構造に関する考察", 第 57 回 宇宙科学技術連合講演会講演集 JSASS-2013-4728, 2013
69) 五十里哲, 中谷俊洋, et al., "PROCYON 姿勢制御系システムの開発と運用状況", 第 15 回宇宙科学シンポジウム ポスター講演 P-247, 2015

付録

以降に超小型深宇宙探査機「PROCYON」の開発で用いた C2A のマニュアルを付録として収録する．

本マニュアルは実際の搭載ソフトウェアのソースコードとそれに記入したコメントを元に，ソースコードドキュメンテーションツールである Doxygen を利用して変換・作成したものである．

より詳細な内容について情報が必要な場合は「東京大学 工学系研究科 航空宇宙工学専攻 中須賀・船瀬研究室」 (E-mail: nlab_info@space.t.u-tokyo.ac.jp) へ連絡のこと．

Command Centric Architecture Reference Manual

Generated by Doxygen 1.8.9.1

Mon Dec 1 2014

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 AMInfo Struct Reference

Data Fields

- **AppInfo ais** [**AM_MAX_APPS**]
- int **page_no**

3.1.1 Detailed Description

Parameters

AppManager の情報を格納する構造体	

3.1.2 Field Documentation

3.1.2.1 AppInfo ais[AM_MAX_APPS]

Parameters

アプリ登録テーブル	

3.1.2.2 int page_no

Parameters

テレメトリ生成ページ番号	

The documentation for this struct was generated from the following file:

- Core/ApplicationManager/**AppManager.h**

3.2 AnomalyCode Struct Reference

Data Fields

- unsigned int **group**
- unsigned int **local**

3.2.1 Detailed Description

アノマリコードはアノマリの発生源を識別する「グループID」と発生源で発 生したアノマリーの種別を識別する「ローカルID」の組で表現する。

例えばAnomalyLogger が記録リストにこれ以上アノマリーを記録できないと いうアノマリーを発生させる場合、アノマリーコードの (グループID, ロー カルID) は (AL_ANOMALY_LOGGER, AL_FULL) となる。

3.2.2 Field Documentation

3.2.2.1 unsigned int group

グループID

3.2.2.2 unsigned int local

ローカルID

The documentation for this struct was generated from the following file:

- Core/AnomalyLogger/**AnomalyLogger.h**

3.3 AnomalyLogger Struct Reference

Data Fields

- size_t **counter**
- size_t **header**
- int **page_no**
- **AnomalyRecord records** [**AL_RECORD_MAX**]

3.3.1 Detailed Description

AnomalyLogger の各種情報を一元管理する構造体。アノマリーの記録状況と して、記録したアノマリーの総数、現在の記録リストの先頭位置、記録リス ト本体を保持している。また、テレメトリ生成用のページ番号を保持する。

AnomalyLogger はアノマリーを連長圧縮して記録するため、総アノマリー記 録数と記録リスト先頭位置が通常一致しないことに注意すること。

3.3.2 Field Documentation

3.3.2.1 size_t counter

総アノマリー記録数

3.3.2.2 size_t header

記録リスト先頭位置

3.3.2.3 int page_no

テレメトリ生成ページ番号

3.3.2.4 AnomalyRecord records[AL_RECORD_MAX]

アノマリ記録リスト

The documentation for this struct was generated from the following file:

- Core/AnomalyLogger/**AnomalyLogger.h**

3.4 AnomalyRecord Struct Reference

Data Fields

- **AnomalyCode code**
- size_t **run_length**
- **OBCTime time**

3.4.1 Detailed Description

AnomalyLogger はアノマリーを、記録時刻、アノマリーコード、連続発生回 数の 3 要素で記録する。

AnomalyLogger は同一のアノマリーコードが連続発生した場合に記録リスト が埋め尽くされる事態を防ぐため、アノマリーコードを連長圧縮する機能を 備える。記録情報の連続発生回数は、この連長に対応する。

3.4.2 Field Documentation

3.4.2.1 AnomalyCode code

アノマリーコード

3.4.2.2 size_t run_length

連続発生回数

3.4.2.3 OBCTime time

記録時刻

The documentation for this struct was generated from the following file:

- Core/AnomalyLogger/**AnomalyLogger.h**

3.5 AppInfo Struct Reference

Data Fields

- void(∗ **entry_point**)(void)
- void(∗ **initializer**)(void)
- **step_t max**
- **step_t min**
- const char ∗ **name**
- **step_t prev**

3.5.1 Detailed Description

利用者が明示的に指定する情報として、アプリケーションの名称・初期化処 理・実処理に関する情報を格納する。

上記に加えてアプリケーション実行時にTaskDispatcher が自動収集するアプ リケーションの最新・最短・最長の処理時間を格納する。

3.5.2 Field Documentation

3.5.2.1 void(∗ entry_point) (void)

実処理関数へのポインタ

3.5.2.2 void(∗ initializer) (void)

初期化処理関数へのポインタ

3.5.2.3 step_t max

過去最長の処理時間情報

3.5.2.4 step_t min

過去最短の処理時間情報

3.5.2.5 const char∗ name

アプリケーション名

3.5.2.6 step_t prev

最新の処理時間情報

The documentation for this struct was generated from the following file:

- Core/ApplicationManager/**AppInfo.h**

3.6 ModeManagerInfo Struct Reference

Data Fields

- MD_ModeID **current_id**
- size_t **mode_list** [MODE_MAX]
- MD_ModeID **previous_id**
- **MM_Status stat**
- size_t **transition_table** [MODE_MAX][MODE_MAX]

3.6.1 Field Documentation

3.6.1.1 MD_ModeID current_id

現行モード番号

3.6.1.2 size_t mode_list[MODE_MAX]

タスクリストテーブル

3.6.1.3 MD_ModeID previous_id

直前モード番号

3.6.1.4 MM_Status stat

モード遷移処理状態

3.6.1.5 size_t transition_table[MODE_MAX][MODE_MAX]

モード遷移テーブル

The documentation for this struct was generated from the following file:

- Core/ModeManager/**ModeManager.h**

3.7 OBCTime Struct Reference

Data Fields

- **cycle_t master**
- **cycle_t mode**
- **step_t step**

3.7.1 Detailed Description

マスターサイクル、モードサイクル、ステップで構成されるOBC 時刻情報を 格納する。

3.7.2 Field Documentation

3.7.2.1 cycle_t master

マスターサイクル

3.7.2.2 cycle_t mode

モードサイクル

3.7.2.3 step_t step

ステップ

The documentation for this struct was generated from the following file:

- Core/TimeManager/**OBCTime.h**

3.8 TDSP_Info Struct Reference

Data Fields

- **cycle_t activated_at**
- size_t **task_list_id**
- CDIS **tskd**

3.8.1 Detailed Description

タスク実行用のコマンドキュー、展開するタスクリストID、直近のタスクリ スト展開サイクルを格納する。

3.8.2 Field Documentation

3.8.2.1 cycle_t activated_at

直近タスクリスト展開サイクル

3.8.2.2 size_t task_list_id

タスクリストID

3.8.2.3 CDIS tskd

タスク実行用コマンドキュー

The documentation for this struct was generated from the following file:

- Core/TaskManager/**TaskDispatcher.h**

Chapter 4

File Documentation

4.1 Core/AnomalyLogger/AnomalyLogger.c File Reference

Functions

- void **AL_add_anomaly** (unsigned int group, unsigned int local)
- void **AL_clear** (void)
- const **AnomalyRecord** * **AL_get_latest_record** (void)
- const **AnomalyRecord** * **AL_get_record** (size_t pos)
- void **AL_initialize** (void)
- int **Cmd_AL_ADD_ANOMALY** (const CTCP *packet)
- int **Cmd_AL_CLEAR_LIST** (const CTCP *packet)
- int **Cmd_AL_SET_PAGE_FOR_TLM** (const CTCP *packet)

Variables

- const **AnomalyLogger** * **al**

4.1.1 Function Documentation

4.1.1.1 void AL_add_anomaly (unsigned int *group*, unsigned int *local*)

アノマリー情報記録関数

アノマリー記録リストに引数で指定したアノマリーID をもつアノマリー情報 を記録する。搭載プログラム内部で各種アノマリーを記録する際に使用する。

搭載ソフトウェア内部ではコマンド形式ではなく本関数を用いてアノマリー の登録を行うことを想定している。

外部とのインターフェースをコマンドのみに集約するという通常のコア機能 の実装方針と異なり、アノマリー記録機能をコマンドだけでなく通常関数と しても用意している理由はアノマリーの記録がコマンド機能の初期化が完了 する以前のコア機能初期化処理や各種アプリケーション初期化処理などでも 発生する可能性があり、搭載ソフトウェア内部ではコマンドとしての機能実 装だけでは不十分なためである。

Parameters

group	記録アノマリーのグループID

local	記録アノマリーのローカルID

4.1.1.2 void AL_clear (void)

アノマリー記録リストクリア関数

アノマリー記録リストに登録されたアノマリー情報をすべて消去し、初期化 直後と同様の状態にクリアする。記録リスト中のすべてのアノマリー情報は アノマリーID, 記録時刻、連長がすべて 0 に設定される。

この関数はAnomalyLogger 以外ではAnomalyHandler アプリケーションの状態 クリア機能で呼び出されることのみを想定しており、他の場所での呼び出し は想定されていない。

4.1.1.3 const AnomalyRecord* AL_get_latest_record (void)

最新アノマリー記録取得関数

アノマリー記録リストに記録された最新のアノマリー情報を指すポインタを 返す。

アノマリーが記録リストに記録されていない場合は初期化された先頭要素へ のポインタを返し、この場合の (グループID, ローカルID) は (0, 0) となる。

Returns

最新アノマリー情報へのポインタ

4.1.1.4 const AnomalyRecord* AL_get_record (size_t *pos*)

アノマリー情報取得関数

アノマリー記録リストの指定された位置に記録されたアノマリー情報を指す ポインタを返す。

直接アノマリー記録リストの配列にアクセスする場合との違いは指定位置の 確認機能にあり、指定された位置が、記録リストの先頭位置を超え、まだ記 録が行われていない位置だった場合は異常と判定しNULL ポインタを返す。

Parameters

pos	取得するアノマリー情報の位置

Returns

指定した位置に記録されたアノマリー情報へのポインタ

4.1.1.5 void AL_initialize (void)

AnomalyLogger 初期化関数

AnomalyLogger の状態初期化を行う。初期化により、記録アノマリー総数は 0 となり、アノマリー記録位置は先頭 (位置 0) に設定される。また、記録リス ト中のすべてのアノマリー情報はアノマリーID, 記録時刻、連長がすべて 0 に設定されたアノマリー情報でクリアされる。

4.1.1.6 int Cmd_AL_ADD_ANOMALY (const CTCP * *packet*)

アノマリー情報記録コマンド

アノマリー記録リストにパラメータで指定したアノマリーID をもつアノマリー 情報を記録するコマンド。パラメータはアノマリーのグローバルID とローカ ルID の 2 つ。

パラメータ長が不正な場合は処理は打ち切られ、アノマリーの記録は行われ ない。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常

4.1.1.7 int Cmd_AL_CLEAR_LIST (const CTCP * *packet*)

アノマリー記録リストクリアコマンド

アノマリー記録リストに登録されたアノマリー情報をすべて消去し、初期化 直後と同様の状態にクリアするコマンド。パラメータは無い。

このコマンドの実行により、記録リスト中のすべてのアノマリー情報はアノ マリーID, 記録時刻、連長がすべて 0 に設定される。

パラメータ長が不正な場合は処理は打ち切られ、記録リストのクリアは行わ れない。

AnomalyHandler アプリケーションによるアノマリー対応を有効化している場 合、このコマンドを利用してアノマリー記録リストをクリアすると、AnomalyHnadler 側が記録しているアノマリー発生状況とAnomalyLoggerの記 録リストの状態に不整合が生じる。この状態を避けるため、 AnomalyHandler 使用時はAnomalyHandler側に実装されているクリアコマンド を利用してAnomalyLogger 側の情報をクリアすべきである。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常

4.1.1.8 int Cmd_AL_SET_PAGE_FOR_TLM (const CTCP * *packet*)

アノマリー記録リストページ番号設定コマンド

アノマリー記録リストをテレメトリで確認する際のページ番号を設定するコ マンド。パラメータは指定するページ番号。

パラメータ長が不正な場合、パラメータで指定されたページ番号不正な場合 は実行異常を返し、この場合ページ番号は変化しない。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常

4.1.2 Variable Documentation

4.1.2.1 const AnomalyLogger* al

AnomalyLogger 状態変数

4.2 Core/AnomalyLogger/AnomalyLogger.h File Reference

Data Structures

- struct **AnomalyCode**
- struct **AnomalyLogger**
- struct **AnomalyRecord**

Macros

- #define **AL_RECORD_MAX** (**AL_TLM_PAGE_SIZE**∗**AL_TLM_PAGE_MAX**)
- #define **AL_TLM_PAGE_MAX** (4)
- #define **AL_TLM_PAGE_SIZE** (32)

Enumerations

- enum **AL_ACK** { **AL_SUCCESS**, **AL_FULL** }
- enum **AL_CORE_GROUP** {
  AL_ANOMALY_LOGGER, **AL_MODE_MANAGER**, **AL_TASK_MANAGER**, **AL_TASK_DISPATCHER**,
  AL_APP_MANAGER, **AL_CORE_GROUP_MAX** }

Functions

- void **AL_add_anomaly** (unsigned int group, unsigned int local)
- void **AL_clear** (void)
- const **AnomalyRecord** ∗ **AL_get_latest_record** (void)
- const **AnomalyRecord** ∗ **AL_get_record** (size_t pos)
- void **AL_initialize** (void)
- int **Cmd_AL_ADD_ANOMALY** (const CTCP ∗packet)
- int **Cmd_AL_CLEAR_LIST** (const CTCP ∗packet)
- int **Cmd_AL_SET_PAGE_FOR_TLM** (const CTCP ∗packet)

Variables

- const **AnomalyLogger** ∗ **al**

4.2.1 Macro Definition Documentation

4.2.1.1 #define AL_RECORD_MAX (AL_TLM_PAGE_SIZE∗AL_TLM_PAGE_MAX)

最大アノマリー記録数

AnomalyLogger の管理するアノマリー記録リストに記録できるアノマリーの 最大数を決める定数。1 ページあたりのアノマリー数 (AL_TLM_PAGE_SIZE) と 記録リストの総ページ数 (AL_TLM_PAGE_MAX) の積。

4.2.1.2 #define AL_TLM_PAGE_MAX (4)

アノマリ記録リストのページ数

AnomalyLogger が管理するアノマリー記録リストのページ数。

4.2.1.3 #define AL_TLM_PAGE_SIZE (32)

1 ページあたりのアノマリー数

AnomalyLogger が管理するアノマリー記録リストの確認テレメトリ 1 ページに 格納するアノマリー数。

アノマリ記録リストの内容確認テレメトリは記録リスト全体から必要情報が 含まれた部分を効率的に取得するため、記録リストをページと呼ぶ固定サイ ズに分割し、ページ指定でテレメトリパケットを生成する実装となっている。

1 ページあたりのアノマリー数は 1 ページがテレメトリパケットのサイズ上限 を超えない範囲に設定する必要がある点に注意すること。

4.2.2 Enumeration Type Documentation

4.2.2.1 enum AL_ACK

AnomalyLogger の処理結果定数

Enumerator

AL_SUCCESS 異常なし
AL_FULL 登録余裕なし

4.2.2.2 enum AL_CORE_GROUP

Core 機能が登録するアノマリーのグループID 定数

Enumerator

AL_ANOMALY_LOGGER **AnomalyLogger** (p. 133).
AL_MODE_MANAGER ModeManager.
AL_TASK_MANAGER 使われていない!
AL_TASK_DISPATCHER TaskDispatcher.
AL_APP_MANAGER AppManager.
AL_CORE_GROUP_MAX Core 機能以外のグループID の開始値

4.2.3 Function Documentation

4.2.3.1 void AL_add_anomaly (unsigned int *group*, unsigned int *local*)

アノマリー情報記録関数

アノマリー記録リストに引数で指定したアノマリーID をもつアノマリー情報 を記録する。搭載プログラム内部で各種アノマリーを記録する際に使用する。

搭載ソフトウェア内部ではコマンド形式ではなく本関数を用いてアノマリー の登録を行うことを想定している。

外部とのインターフェースをコマンドのみに集約するという通常のコア機能 の実装方針と異なり、アノマリー記録機能をコマンドだけでなく通常関数と しても用意している理由はアノマリーの記録がコマンド機能の初期化が完了 する以前のコア機能初期化処理や各種アプリケーション初期化処理などでも 発生する可能性があり、搭載ソフトウェア内部ではコマンドとしての機能実 装だけでは不十分なためである。

Parameters

group	記録アノマリーのグループID
local	記録アノマリーのローカルID

4.2.3.2 void AL_clear (void)

アノマリー記録リストクリア関数

アノマリー記録リストに登録されたアノマリー情報をすべて消去し、初期化 直後と同様の状態にクリアする。記録リスト中のすべてのアノマリー情報は アノマリーID, 記録時刻、連長がすべて 0 に設定される。

この関数はAnomalyLogger 以外ではAnomalyHandler アプリケーションの状態 クリア機能で呼び出されることのみを想定しており、他の場所での呼び出し は想定されていない。

4.2.3.3 const AnomalyRecord* AL_get_latest_record (void)

最新アノマリー記録取得関数

アノマリー記録リストに記録された最新のアノマリー情報を指すポインタを 返す。

アノマリーが記録リストに記録されていない場合は初期化された先頭要素へ のポインタを返し、この場合の (グループID, ローカルID) は (0, 0) となる。

Returns

最新アノマリー情報へのポインタ

4.2.3.4 const AnomalyRecord* AL_get_record (size_t *pos*)

アノマリー情報取得関数

アノマリー記録リストの指定された位置に記録されたアノマリー情報を指す ポインタを返す。

直接アノマリー記録リストの配列にアクセスする場合との違いは指定位置の 確認機能にあり、指定された位置が、記録リストの先頭位置を超え、まだ記 録が行われていない位置だった場合は異常と判定しNULL ポインタを返す。

Parameters

pos	取得するアノマリー情報の位置

Returns

指定した位置に記録されたアノマリー情報へのポインタ

4.2.3.5 void AL_initialize (void)

AnomalyLogger 初期化関数

AnomalyLogger の状態初期化を行う。初期化により、記録アノマリー総数は 0 となり、アノマリー記録位置は先頭 (位置 0) に設定される。また、記録リス ト中のすべてのアノマリー情報はアノマリーID, 記録時刻、連長がすべて 0 に設定されたアノマリー情報でクリアされる。

4.2.3.6 int Cmd_AL_ADD_ANOMALY (const CTCP * *packet*)

アノマリー情報記録コマンド

アノマリー記録リストにパラメータで指定したアノマリーID をもつアノマリー 情報を記録するコマンド。パラメータはアノマリーのグローバルID とローカ ルID の 2 つ。

パラメータ長が不正な場合は処理は打ち切られ、アノマリーの記録は行われ ない。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常

4.2.3.7 int Cmd_AL_CLEAR_LIST (const CTCP * *packet*)

アノマリー記録リストクリアコマンド

アノマリー記録リストに登録されたアノマリー情報をすべて消去し、初期化 直後と同様の状態にクリアするコマンド。パラメータは無い。

このコマンドの実行により、記録リスト中のすべてのアノマリー情報はアノ マリーID, 記録時刻、連長がすべて 0 に設定される。

パラメータ長が不正な場合は処理は打ち切られ、記録リストのクリアは行わ れない。

AnomalyHandler アプリケーションによるアノマリー対応を有効化している場 合、このコマンドを利用してアノマリー記録リストをクリアすると、AnomalyHnadler 側が記録しているアノマリー発生状況とAnomalyLoggerの記 録リストの状態に不整合が生じる。この状態を避けるため、 AnomalyHandler 使用時はAnomalyHandler側に実装されているクリアコマンド を利用してAnomalyLogger 側の情報をクリアすべきである。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常

4.2.3.8 int Cmd_AL_SET_PAGE_FOR_TLM (const CTCP * *packet*)

アノマリー記録リストページ番号設定コマンド

アノマリー記録リストをテレメトリで確認する際のページ番号を設定するコ マンド。パラメータは指定するページ番号。

パラメータ長が不正な場合、パラメータで指定されたページ番号不正な場合 は実行異常を返し、この場合ページ番号は変化しない。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常

4.2.4 Variable Documentation

4.2.4.1 const AnomalyLogger* al

AnomalyLogger 状態変数

4.3 Core/ApplicationManager/AppInfo.c File Reference

Functions

- **AppInfo create_app_info** (const char *name, void(*initializer)(void), void(*entry_point)(void))

4.3.1 Function Documentation

4.3.1.1 AppInfo create_app_info (const char * *name*, void(*)(void) *initializer*, void(*)(void) *entry_point*)

AppInfo オブジェクト生成関数

引数で指定したアプリケーション名称、初期化処理関数、実処理関数を設定 したAppInfo オブジェクトを生成する。

指定情報の設定の他に、実行時間情報の初期化も実施する。この初期化によ り最新と最長の処理時間情報は 0 に、最短の処理時間情報は 0xffffffff に設 定される。最長と最短の大小関係が逆転した値に設定されるのは初期化時の みであり、この情報を使うことでアプリケーションが実行されたかどうかを 判定できる。

Parameters

name	アプリケーション名
initializer	初期化関数へのポインタ
entry_point	実処理関数へのポインタ

Returns

指定情報が設定されたAppInfo オブジェクト

4.4 Core/ApplicationManager/AppInfo.h File Reference

Data Structures

- struct **AppInfo**

Functions

- **AppInfo create_app_info** (const char *name, void(*initializer)(void), void(*entry_point)(void))

4.4.1 Function Documentation

4.4.1.1 AppInfo create_app_info (const char * *name*, void(*)(void) *initializer*, void(*)(void) *entry_point*)

AppInfo オブジェクト生成関数

引数で指定したアプリケーション名称、初期化処理関数、実処理関数を設定 したAppInfo オブジェクトを生成する。

指定情報の設定の他に、実行時間情報の初期化も実施する。この初期化によ り最新と最長の処理時間情報は 0 に、最短の処理時間情報は 0xffffffff に設 定される。最長と最短の大小関係が逆転した値に設定されるのは初期化時の みであり、この情報を使うことでアプリケーションが実行されたかどうかを 判定できる。

Parameters

name	アプリケーション名
initializer	初期化関数へのポインタ
entry_point	実処理関数へのポインタ

Returns

指定情報が設定されたAppInfo オブジェクト

4.5 Core/ApplicationManager/AppManager.c File Reference

Functions

- void **AM_initialize** (void)
- void **AM_initialize_all_apps** (void)
- **AM_ACK AM_register_ai** (size_t id, const **AppInfo** ∗ai)
- int **Cmd_AM_EXECUTE_APP** (const CTCP ∗packet)
- int **Cmd_AM_INITIALIZE_APP** (const CTCP ∗packet)
- int **Cmd_AM_REGISTER_APP** (const CTCP ∗packet)
- int **Cmd_AM_SET_PAGE_FOR_TLM** (const CTCP ∗packet)

Variables

- const **AMInfo** ∗ **ami**

4.5.1 Function Documentation

4.5.1.1 void AM_initialize (void)

AppManager 初期化関数

アプリ登録テーブルの内容をすべてNOP アプリで初期化し、ページ番号を 0 に 初期化する。

4.5.1.2 void AM_initialize_all_apps (void)

登録済みアプリ一括初期化関数

アプリ登録テーブルに登録されている全アプリの初期化処理をアプリID が小 さいものから順に呼び出す。

プログラム実行開始前に登録されている全アプリの初期化を行うため用意さ れた関数であり、実行開始後の呼び出しは想定していない。

4.5.1.3 AM_ACK AM_register_ai (size_t *id,* const AppInfo ∗ *ai*)

アプリ情報登録関数

指定されたアプリID に指定されたアプリ情報を登録する。

この関数はプログラム初期化処理での呼び出しを想定しており、異常検知を 容易にするため、異常発生時はアノマリーを登録する。登録される異常は指 定されたアプリID が登録可能最大数を超えている場合で、登録されるアノマ リーコードはAM_INVALID となる。この場合登録操作は行われず、登録テーブ ルの内容は変化しない。

Parameters

id	登録先アプリID
ai	登録情報を保持したAppInfo オブジェクトへのポインタ

Return values

AM_SUCCESS	正常終了
AM_INVALID_ID	指定アプリID が登録最大数以上

4.5.1.4 int Cmd_AM_EXECUTE_APP (const CTCP * *packet*)

アプリ実行コマンド

登録されているアプリの実処理を呼び出すコマンド。パラメータはアプリ ID。

パラメータ長が不正な場合、アプリID が不正な場合、アプリID は正常だが初 期化処理が登録されていない場合は初期化処理を行わず実行異常を返す。

このコマンドはTaskList の中で各種アプリを呼び出すために使用することを 主目的に実装されている。通常コマンドとして呼び出しは可能だが、その場 合は呼び出すアプリの処理時間に注意すること。通常コマンドとして呼び出 す場合、その実行時間上限はコマンド実行処理に割り当てられた処理時間に 制約されるため、それを超える処理時間のアプリを実行した場合は処理時間 超過が発生することになる。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常
AM_INVALID_ID	アプリID 異常
AM_NOT_REGISTERD	初期化処理未登録

4.5.1.5 int Cmd_AM_INITIALIZE_APP (const CTCP * *packet*)

アプリ初期化コマンド

登録されているアプリの初期化処理を呼び出すコマンド。パラメータはアプ リID。

パラメータ長が不正な場合、アプリID が不正な場合、アプリID は正常だが初 期化処理が登録されていない場合は初期化処理を行わず実行異常を返す。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常
AM_INVALID_ID	アプリID 異常
AM_NOT_REGISTERD	初期化処理未登録

4.5.1.6 int Cmd_AM_REGISTER_APP (const CTCP * *packet*)

アプリ登録コマンド

アプリ登録テーブルにアプリ情報を登録するコマンド。パラメータはアプリ ID, 初期化関数へのポインタ、実処理関数へのポインタ。

指定したアプリID に対応するアプリ情報として、初期化関数と実処理関数を 設定する。このコマンドにより登録されるアプリ名称はすべて"SPECIAL"と なる。

パラメータ長が不正な場合、アプリID が不正な場合は登録処理を行わず実行 異常を返す。この場合登録テーブルの内容は変化しない。

このコマンドは軌道上で部分再プロによりアプリケーションを追加する場合 を想定して用意したコマンドであり、通常の運用で使用することはない。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常
AM_INVALID_ID	アプリID 異常

4.5.1.7 int Cmd_AM_SET_PAGE_FOR_TLM (const CTCP ∗ *packet*)

アプリ登録テーブルページ番号設定コマンド

アプリ登録テーブルをテレメトリで確認する際のページ番号を設定するコマ ンド。パラメータは指定するページ番号。

パラメータ長が不正な場合、パラメータで指定されたページ番号不正な場合 は実行異常を返し、この場合ページ番号は変化しない。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常
CCP_EXEC_ILLEGAL_P↩ ARAMETER	ページ番号異常

4.5.2 Variable Documentation

4.5.2.1 const AMInfo∗ ami

AppManager 状態変数

4.6 Core/ApplicationManager/AppManager.h File Reference

Data Structures

- struct **AMInfo**

Macros

- #define **AM_MAX_APPS** (**AM_TLM_PAGE_SIZE**∗**AM_TLM_PAGE_MAX**)
- #define **AM_TLM_PAGE_MAX** (6)
- #define **AM_TLM_PAGE_SIZE** (32)

Enumerations

- enum **AM_ACK** { **AM_SUCCESS**, **AM_INVALID_ID**, **AM_NOT_REGISTERED**, **AM_UNKNOWN** }

Functions

- void **AM_initialize** (void)
- void **AM_initialize_all_apps** (void)
- **AM_ACK AM_register_ai** (size_t id, const **AppInfo** ∗ai)
- int **Cmd_AM_EXECUTE_APP** (const CTCP ∗packet)
- int **Cmd_AM_INITIALIZE_APP** (const CTCP ∗packet)
- int **Cmd_AM_REGISTER_APP** (const CTCP ∗packet)
- int **Cmd_AM_SET_PAGE_FOR_TLM** (const CTCP ∗packet)

Variables

- const **AMInfo** ∗ **ami**

4.6.1 Macro Definition Documentation

4.6.1.1 #define AM_MAX_APPS (AM_TLM_PAGE_SIZE∗AM_TLM_PAGE_MAX)

最大アプリ登録数

AppManager の管理するアプリ登録テーブルに登録できるアプリケーションの 最大数を決める定数。1 ページあたりのアプリ情報数 (AM_TLM_PAGE_SIZE) と 登録テーブルのページ数 (AM_TLM_PAGE_MAX) の積。

4.6.1.2 #define AM_TLM_PAGE_MAX (6)

アプリ登録テーブルのページ数

AppManager が管理するアプリ登録テーブルのページ数を決める定数。

4.6.1.3 #define AM_TLM_PAGE_SIZE (32)

1 ページあたりのアプリ登録情報数

AppManager が管理するアプリ登録テーブルの内容確認テレメトリの 1 ページ に格納するアプリケーション情報の数を決める定数。

アプリ登録テーブルの内容確認テレメトリは登録テーブル全体から必要情報 が含まれた部分を効率的に取得するため、登録テーブルをページと呼ぶ固定 サイズに分割し、ページ指定でテレメトリパケットを生成する実装となって いる。

1 ページあたりのアプリ情報数は 1 ページがテレメトリパケットのサイズ上限 を超えない範囲に設定する必要がある点に注意すること。

4.6.2 Enumeration Type Documentation

4.6.2.1 enum AM_ACK

AppManager の処理結果定数

Enumerator

AM_SUCCESS 異常なし

AM_INVALID_ID アプリID 異常

AM_NOT_REGISTERED 登録情報なし

AM_UNKNOWN 想定外異常

4.6.3 Function Documentation

4.6.3.1 void AM_initialize (void)

AppManager 初期化関数

アプリ登録テーブルの内容をすべてNOP アプリで初期化し、ページ番号を 0 に 初期化する。

4.6.3.2 void AM_initialize_all_apps (void)

登録済みアプリ一括初期化関数

アプリ登録テーブルに登録されている全アプリの初期化処理をアプリID が小 さいものから順に呼び出す。

プログラム実行開始前に登録されている全アプリの初期化を行うため用意さ れた関数であり、実行開始後の呼び出しは想定していない。

4.6.3.3 AM_ACK AM_register_ai (size_t *id*, const AppInfo * *ai*)

アプリ情報登録関数

指定されたアプリID に指定されたアプリ情報を登録する。

この関数はプログラム初期化処理での呼び出しを想定しており、異常検知を 容易にするため、異常発生時はアノマリーを登録する。登録される異常は指 定されたアプリID が登録可能最大数を超えている場合で、登録されるアノマ リーコードはAM_INVALID となる。この場合登録操作は行われず、登録テーブ ルの内容は変化しない。

Parameters

id	登録先アプリID
ai	登録情報を保持したAppInfo オブジェクトへのポインタ

Return values

AM_SUCCESS	正常終了
AM_INVALID_ID	指定アプリID が登録最大数以上

4.6.3.4 int Cmd_AM_EXECUTE_APP (const CTCP * *packet*)

アプリ実行コマンド

登録されているアプリの実処理を呼び出すコマンド。パラメータはアプリ ID。

パラメータ長が不正な場合、アプリID が不正な場合、アプリID は正常だが初 期化処理が登録されていない場合は初期化処理を行わず実行異常を返す。

このコマンドはTaskList の中で各種アプリを呼び出すために使用することを 主目的に実装されている。通常コマンドとして呼び出しは可能だが、その場 合は呼び出すアプリの処理時間に注意すること。通常コマンドとして呼び出 す場合、その実行時間上限はコマンド実行処理に割り当てられた処理時間に 制約されるため、それを超える処理時間のアプリを実行した場合は処理時間 超過が発生することになる。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常
AM_INVALID_ID	アプリID 異常
AM_NOT_REGISTERD	初期化処理未登録

4.6.3.5 int Cmd_AM_INITIALIZE_APP (const CTCP * *packet*)

アプリ初期化コマンド

登録されているアプリの初期化処理を呼び出すコマンド。パラメータはアプ リID。

パラメータ長が不正な場合、アプリID が不正な場合、アプリID は正常だが初 期化処理が登録されていない場合は初期化処理を行わず実行異常を返す。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常
AM_INVALID_ID	アプリID 異常
AM_NOT_REGISTERD	初期化処理未登録

4.6.3.6 int Cmd_AM_REGISTER_APP (const CTCP * *packet*)

アプリ登録コマンド

アプリ登録テーブルにアプリ情報を登録するコマンド。パラメータはアプリ ID, 初期化関数へのポインタ、実処理関数へのポインタ。

指定したアプリID に対応するアプリ情報として、初期化関数と実処理関数を 設定する。このコマンドにより登録されるアプリ名称はすべて"SPECIAL"と なる。

パラメータ長が不正な場合、アプリID が不正な場合は登録処理を行わず実行 異常を返す。この場合登録テーブルの内容は変化しない。

このコマンドは軌道上で部分再プロによりアプリケーションを追加する場合 を想定して用意したコマンドであり、通常の運用で使用することはない。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常
AM_INVALID_ID	アプリID 異常

4.6.3.7 int Cmd_AM_SET_PAGE_FOR_TLM (const CTCP * *packet*)

アプリ登録テーブルページ番号設定コマンド

アプリ登録テーブルをテレメトリで確認する際のページ番号を設定するコマ ンド。パラメータは指定するページ番号。

パラメータ長が不正な場合、パラメータで指定されたページ番号不正な場合 は実行異常を返し、この場合ページ番号は変化しない。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常
CCP_EXEC_ILLEGAL_P↩ ARAMETER	ページ番号異常

4.6.4 Variable Documentation

4.6.4.1 const AMInfo* ami

AppManager 状態変数

4.7 Core/ModeManager/ModeManager.c File Reference

Functions

- int **Cmd_MM_FINISH_TRANSITION** (const CTCP *packet)
- int **Cmd_MM_SET_MODE_LIST** (const CTCP *packet)
- int **Cmd_MM_SET_TRANSITION_TABLE** (const CTCP *packet)
- int **Cmd_MM_START_TRANSITION** (const CTCP *packet)
- void **MM_initialize** (void)

Variables

- const **ModeManagerInfo** * **mmi**

4.7.1 Function Documentation

4.7.1.1 int Cmd_MM_FINISH_TRANSITION (const CTCP * *packet*)

モード遷移処理終了コマンド

モード遷移処理終了をModeManager に通知するコマンド。このコマンドにパ ラメータはない。

このコマンドによりModeManager の状態は遷移中から遷移完了に変更される。 モード遷移処理を記述するブロックコマンドは最後にこのコマンドの呼び出 しが必須となる。地上局からこのコマンドを直接発行する状況は通常想定さ れないはず。

このコマンドは自動モード遷移などオンボードで発行される可能性が高い。 このため、遷移失敗時の自律対応をサポートする目的で異常発生時はアノマ リを登録する。モード遷移処理実行中以外でこのコマンドが呼び出された場 合はMM_NOT_IN_PROGRESS を、遷移先モードのタスクリスト設定に失敗した場 合はMM_TL_LOAD_FAILED をそれぞれ登録する。

遷移先モードのタスクリスト設定に失敗した場合は直前モードのタスクリス トを維持する。この場合もモード遷移処理自体は完了とし、ModeManager の 状態は遷移中から遷移完了に変更される。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
MM_NOT_IN_PROGRESS	モード遷移処理実行中以外での呼び出し

4.7.1.2 int Cmd_MM_SET_MODE_LIST (const CTCP * *packet*)

タスクリストテーブル設定コマンド

各モードで実行するタスクリストに対応するブロックコマンド番号を設定す るコマンド。モード番号とブロックコマンド番号をパラメータに取る。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常
MM_BAD_ID	モード番号異常
MM_BAD_BC_INDEX	タスクリスト番号がブロックコマンド番号の範囲外
MM_INACTIVE_ID	タスクリスト番号に対応するブロックコマンドが不活状態

4.7.1.3 int Cmd_MM_SET_TRANSITION_TABLE (const CTCP * *packet*)

モード遷移テーブル設定コマンド

モード遷移時に呼び出す遷移処理に対応するブロックコマンド番号を設定す るコマンド。遷移元モード番号、遷移先モード番号、ブロックコマンド番号 をパラメータに取る。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常

MM_BAD_ID	モード番号異常
MM_BAD_BC_INDEX	タスクリスト番号がブロックコマンド番号の範囲外
MM_INACTIVE_ID	タスクリスト番号に対応するブロックコマンドが不活状態

4.7.1.4 int Cmd_MM_START_TRANSITION (const CTCP * *packet*)

モード遷移処理開始コマンド

現在のモードから指定されたモードへの遷移開始をModeManager に通知する コマンド。遷移先モード番号をパラメータに取る。

このコマンドによりModeManager の状態は遷移完了から遷移中となり、遷移 テーブルで定義されたブロックコマンドが展開され遷移処理が始まる。

このコマンドは自動モード遷移などオンボードで発行される可能性が高い。 このため、遷移失敗時の自律対応をサポートする目的で異常発生時はアノマ リを登録する。遷移先に指定されたモード番号が存在しない場合は MM_BAD_ID を、モード遷移処理実行中に遷移開始を指令した場合は MM_OVERWRITE を、モード遷移テーブル上で許可されないモード遷移が指令さ れた場合はMM_ILLEGAL_MOVE をそれぞれ登録する。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常
MM_BAD_ID	モード番号異常
MM_OVERWRITE	モード遷移処理中の呼び出し
MM_ILLEGAL_MOVE	遷移が許可されないモードへの遷移指令

4.7.1.5 void MM_initialize (void)

ModeManager 初期化関数

ModeDefinisions.h で定義されるタスクリストテーブル、モード遷移テーブ ルの読み込みと各種状態変数の初期設定し、初期シーケンスを開始する。

各種状態変数の初期値は、遷移状態はMM_IN_PROGRESS, 直前モードと現行モー ドはともにINITIAL。

モード遷移テーブルでINITIAL モード以外からINITIAL モードへの遷移を不許 可に設定することで、運用時 INITIAL モードであれば確実に衛星起動直後と 判断できるようになるため、この設定を推奨する。

4.7.2 Variable Documentation

4.7.2.1 const ModeManagerInfo* mmi

ModeManager 状態変数

4.8 Core/ModeManager/ModeManager.h File Reference

Data Structures

- struct **ModeManagerInfo**

Enumerations

- enum **MM_ACK** {
  MM_SUCCESS, **MM_BAD_ID**, **MM_BAD_CYCLE**, **MM_BAD_BC_INDEX**,
  MM_INACTIVE_BLOCK, **MM_OVERWRITE**, **MM_ILLEGAL_MOVE**, **MM_NOT_IN_PROGRESS**,
  MM_TL_LOAD_FAILED }
- enum **MM_Status** { **MM_FINISHED**, **MM_IN_PROGRESS** }

Functions

- int **Cmd_MM_FINISH_TRANSITION** (const CTCP ∗packet)
- int **Cmd_MM_SET_MODE_LIST** (const CTCP ∗packet)
- int **Cmd_MM_SET_TRANSITION_TABLE** (const CTCP ∗packet)
- int **Cmd_MM_START_TRANSITION** (const CTCP ∗packet)
- void **MM_initialize** (void)

Variables

- const **ModeManagerInfo** ∗ **mmi**

4.8.1 Enumeration Type Documentation

4.8.1.1 enum MM_ACK

ModeManager の処理結果定数

Enumerator

MM_SUCCESS 異常なし

MM_BAD_ID 不正なモード番号

MM_BAD_CYCLE 使われていない!

MM_BAD_BC_INDEX 不正なブロックコマンド番号

MM_INACTIVE_BLOCK 不活ブロックの指定

MM_OVERWRITE モード遷移処理中のモード遷移指令呼び出し

MM_ILLEGAL_MOVE 不正なモード遷移先指定

MM_NOT_IN_PROGRESS モード遷移処理外での遷移終了処理呼び出し

MM_TL_LOAD_FAILED タスクリスト設定失敗

4.8.1.2 enum MM_Status

ModeManager の状態定義定数

Enumerator

MM_FINISHED モード遷移処理完了

MM_IN_PROGRESS モード遷移処理中

4.8.2 Function Documentation

4.8.2.1 int Cmd_MM_FINISH_TRANSITION (const CTCP * packet)

モード遷移処理終了コマンド

モード遷移処理終了をModeManager に通知するコマンド。このコマンドにパ ラメータはない。

このコマンドによりModeManager の状態は遷移中から遷移完了に変更される。 モード遷移処理を記述するブロックコマンドは最後にこのコマンドの呼び出 しが必須となる。地上局からこのコマンドを直接発行する状況は通常想定さ れないはず。

このコマンドは自動モード遷移などオンボードで発行される可能性が高い。 このため、遷移失敗時の自律対応をサポートする目的で異常発生時はアノマ リを登録する。モード遷移処理実行中以外でこのコマンドが呼び出された場 合はMM_NOT_IN_PROGRESS を、遷移先モードのタスクリスト設定に失敗した場 合はMM_TL_LOAD_FAILED をそれぞれ登録する。

遷移先モードのタスクリスト設定に失敗した場合は直前モードのタスクリス トを維持する。この場合もモード遷移処理自体は完了とし、ModeManager の 状態は遷移中から遷移完了に変更される。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
MM_NOT_IN_PROGRESS	モード遷移処理実行中以外での呼び出し

4.8.2.2 int Cmd_MM_SET_MODE_LIST (const CTCP * packet)

タスクリストテーブル設定コマンド

各モードで実行するタスクリストに対応するブロックコマンド番号を設定す るコマンド。モード番号とブロックコマンド番号をパラメータに取る。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常
MM_BAD_ID	モード番号異常
MM_BAD_BC_INDEX	タスクリスト番号がブロックコマンド番号の範囲外
MM_INACTIVE_ID	タスクリスト番号に対応するブロックコマンドが不活状態

4.8.2.3 int Cmd_MM_SET_TRANSITION_TABLE (const CTCP * packet)

モード遷移テーブル設定コマンド

モード遷移時に呼び出す遷移処理に対応するブロックコマンド番号を設定す るコマンド。遷移元モード番号、遷移先モード番号、ブロックコマンド番号 をパラメータに取る。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常
MM_BAD_ID	モード番号異常
MM_BAD_BC_INDEX	タスクリスト番号がブロックコマンド番号の範囲外
MM_INACTIVE_ID	タスクリスト番号に対応するブロックコマンドが不活状態

4.8.2.4 int Cmd_MM_START_TRANSITION (const CTCP * *packet*)

モード遷移処理開始コマンド

現在のモードから指定されたモードへの遷移開始をModeManager に通知する コマンド。遷移先モード番号をパラメータに取る。

このコマンドによりModeManager の状態は遷移完了から遷移中となり、遷移 テーブルで定義されたブロックコマンドが展開され遷移処理が始まる。

このコマンドは自動モード遷移などオンボードで発行される可能性が高い。 このため、遷移失敗時の自律対応をサポートする目的で異常発生時はアノマ リを登録する。遷移先に指定されたモード番号が存在しない場合は MM_BAD_ID を、モード遷移処理実行中に遷移開始を指令した場合は MM_OVERWRITE を、モード遷移テーブル上で許可されないモード遷移が指令さ れた場合はMM_ILLEGAL_MOVE をそれぞれ登録する。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常
MM_BAD_ID	モード番号異常
MM_OVERWRITE	モード遷移処理中の呼び出し
MM_ILLEGAL_MOVE	遷移が許可されないモードへの遷移指令

4.8.2.5 void MM_initialize (void)

ModeManager 初期化関数

ModeDefinisions.h で定義されるタスクリストテーブル、モード遷移テーブ ルの読み込みと各種状態変数の初期設定し、初期シーケンスを開始する。

各種状態変数の初期値は、遷移状態はMM_IN_PROGRESS, 直前モードと現行モー ドはともにINITIAL。

モード遷移テーブルでINITIAL モード以外からINITIAL モードへの遷移を不許 可に設定することで、運用時INITIAL モードであれば確実に衛星起動直後と 判断できるようになるため、この設定を推奨する。

4.8.3 Variable Documentation

4.8.3.1 const ModeManagerInfo* mmi

ModeManager 状態変数

4.9 Core/SDK.c File Reference

Functions

- void **sdk_interval** (VP_INT exinf)
- void **sdk_start** (VP_INT exinf)

4.9.1 Function Documentation

4.9.1.1 void sdk_interval (VP_INT *exinf*)

4.9.1.2 void sdk_start (VP_INT *exinf*)

4.10 Core/SDK.h File Reference

Functions

- void **sdk_interval** (VP_INT exinf)
- void **sdk_start** (VP_INT exinf)

4.10.1 Function Documentation

4.10.1.1 void sdk_interval (VP_INT *exinf*)

4.10.1.2 void sdk_start (VP_INT *exinf*)

4.11 Core/TaskManager/TaskDispatcher.c File Reference

Functions

- int **Cmd_TDSP_SET_TASK_LIST** (const CTCP ∗packet)
- **AppInfo print_tdsp_status** (void)
- void **TDSP_execute_pl_as_task_list** (void)
- void **TDSP_initialize** (void)
- void **TDSP_resync_internal_counter** (void)
- **TDSP_ACK TDSP_set_task_list_id** (size_t id)

Variables

- const **TDSP_Info** ∗ **TDSP_info**

4.11.1 Function Documentation

4.11.1.1 int Cmd_TDSP_SET_TASK_LIST (const CTCP ∗ *packet*)

タスクリストID 設定コマンド

運用者がTaskDispatcher に直接実行するタスクリストID を指定する場合に使 用するコマンド。パラメータはタスクリストID のみ。

パラメータで指定された値を無条件にタスクリストID として設定する実装と なっているが、ID がブロックコマンドID として有効か、指定されたブロック コマンドが展開可能状態かぐらいは確認するよう処理を追加すべき。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	このコマンドは必ず正常終了する

4.11.1.2 AppInfo print_tdsp_status (void)

TaskDispatcher 情報表示アプリ登録情報生成関数

設定されているタスクリストID と直近で発生したタスク実行異常の発生時刻 をデバッグコンソールに表示するアプリの登録情報を生成する。

このアプリはデバッグコンソールを利用したデバッグ用途を想定している。

Returns

アプリの実行情報を記録したAppInfo オブジェクト

4.11.1.3 void TDSP_execute_pl_as_task_list (void)

タスクリスト実行関数

タスクリストの展開によりタスク実行用のコマンドキューに蓄積されたコマ ンド列を関数呼び出し時点のステップ時刻に基づきタイムラインコマンドと して実行する。

呼び出し時点でコマンドキューが空の場合、TaskDispatcher に登録されてい るタスクリストID に対応するブロックコマンドを次サイクルの 0 ステップを 起点に展開する。

コマンドを実行しようとした時点で予定実行ステップが既に過ぎていた場合 はTDSP_STEP_OVERRUN アノマリーを登録する。この要因は直前に実行したコ マンド群の処理時間超過が考えられる。なお、アノマリーを記録した場合も コマンド実行は時刻超過の元で実施され、キャンセルされない。

実行したコマンドがCCP_EXEC_SUCCESS 以外の実行異常ステータスを返した場 合はTDSP_TASK_EXE↩C_FAILD アノマリーを登録する。

タスクリストを展開したサイクル時刻中にコマンドキューに蓄積されたコマ ンドの実行と次サイクル用のタスクリスト展開が完了しなかった場合は TDSP_CYCLE_OVERRUN アノマリーを登録する。この場合、この関数はコマンド キューを強制的に全クリアした上で、新規にタスクリストを展開する。ここ で展開されたコマンド群はこの異常が検出された次サイクルに実行されるこ とになる。

4.11.1.4 void TDSP_initialize (void)

TaskDispatcher 初期化関数

タスク実行用のタイムラインコマンドキューを初期化し、デフォルトのタス クリストBC_TL_INITIAL を展開する。

4.11.1.5 void TDSP_resync_internal_counter (void)

タスクリスト展開時刻強制同期関数

TaskDispatcher が内部に保持している、タスクリストを展開したサイクル時 刻の情報を強制的に関数呼び出し時点のサイクル時刻に一致させる。

この関数はOBC 時刻設定コマンドによって、OBC 時刻が不連続に変化した場合 にアノマリー発生を阻止する目的で呼び出すことを念頭に用意されており、 それ以外の呼び出しは想定していない。

4.11.1.6 TDSP_ACK TDSP_set_task_list_id (size_t *id*)

タスクリストID 設定関数

TaskDispatcher が毎サイクル実行するタスクリストのID を設定する。

本関数はタスクリストのID を設定するだけで、実行しているタスクリストに は即時に反映されない。この関数が呼び出されたサイクルの終わりまでは直 前に設定されていたタスクリストが有効であり、設定したタスクリストが実 行されるのは次サイクルからとなる。

Parameters

id	設定するタスクリストのID

Returns

この関数はTDSP_SUCCESS のみを返す。異常終了しない。

4.11.2 Variable Documentation

4.11.2.1 const TDSP_Info* TDSP_info

TaskDispatcher 状態変数

4.12 Core/TaskManager/TaskDispatcher.h File Reference

Data Structures

- struct **TDSP_Info**

Macros

- #define **TDSP_TASK_MAX** BCT_MAX_BLOCK_LENGTH

Enumerations

- enum **TDSP_ACK** {
  TDSP_SUCCESS, **TDSP_DEPLOY_FAILED**, **TDSP_CYCLE_OVERRUN**, **TDSP_STEP_OVERRUN**,
  TDSP_TASK_EXEC_FAILED, **TDSP_UNKNOWN** }

Functions

- int **Cmd_TDSP_SET_TASK_LIST** (const CTCP *packet)
- **AppInfo print_tdsp_status** (void)
- void **TDSP_execute_pl_as_task_list** (void)
- void **TDSP_initialize** (void)
- void **TDSP_resync_internal_counter** (void)
- **TDSP_ACK TDSP_set_task_list_id** (size_t id)

Variables

- const **TDSP_Info** * **TDSP_info**

4.12.1 Macro Definition Documentation

4.12.1.1 #define TDSP_TASK_MAX BCT_MAX_BLOCK_LENGTH

タスクリスト展開コマンドキューのコマンド蓄積上限値

タスクリストはブロックコマンドで表現されるため、全てのブロックコマン ドを問題なく登録できるよう、最大値として、ブロックコマンドの最大長を 指定している。

4.12.2 Enumeration Type Documentation

4.12.2.1 enum TDSP_ACK

TaskDispatcher の処理結果定数

Enumerator

TDSP_SUCCESS 異常なし

TDSP_DEPLOY_FAILED タスクリスト展開失敗

TDSP_CYCLE_OVERRUN サイクル中のタスクリスト実行完遂失敗

TDSP_STEP_OVERRUN タスク実行開始ステップ超過

TDSP_TASK_EXEC_FAILED タスク実行時異常

TDSP_UNKNOWN 想定外異常

4.12.3 Function Documentation

4.12.3.1 int Cmd_TDSP_SET_TASK_LIST (const CTCP * *packet*)

タスクリストID 設定コマンド

運用者がTaskDispatcher に直接実行するタスクリストID を指定する場合に使 用するコマンド。パラメータはタスクリストID のみ。

パラメータで指定された値を無条件にタスクリストID として設定する実装と なっているが、ID がブロックコマンドID として有効か、指定されたブロック コマンドが展開可能状態かぐらいは確認するよう処理を追加すべき。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	このコマンドは必ず正常終了する

4.12.3.2 AppInfo print_tdsp_status (void)

TaskDispatcher 情報表示アプリ登録情報生成関数

設定されているタスクリストID と直近で発生したタスク実行異常の発生時刻 をデバッグコンソールに表示するアプリの登録情報を生成する。

このアプリはデバッグコンソールを利用したデバッグ用途を想定している。

Returns

アプリの実行情報を記録したAppInfo オブジェクト

4.12.3.3 void TDSP_execute_pl_as_task_list (void)

タスクリスト実行関数

タスクリストの展開によりタスク実行用のコマンドキューに蓄積されたコマ ンド列を関数呼び出し時点のステップ時刻に基づきタイムラインコマンドと して実行する。

呼び出し時点でコマンドキューが空の場合、TaskDispatcher に登録されてい るタスクリストID に対応するブロックコマンドを次サイクルの 0 ステップを 起点に展開する。

コマンドを実行しようとした時点で予定実行ステップが既に過ぎていた場合 はTDSP_STEP_OVERRUN アノマリーを登録する。この要因は直前に実行したコ マンド群の処理時間超過が考えられる。なお、アノマリーを記録した場合も コマンド実行は時刻超過の元で実施され、キャンセルされない。

実行したコマンドがCCP_EXEC_SUCCESS 以外の実行異常ステータスを返した場 合はTDSP_TASK_EXE↩C_FAILD アノマリーを登録する。

タスクリストを展開したサイクル時刻中にコマンドキューに蓄積されたコマ ンドの実行と次サイクル用のタスクリスト展開が完了しなかった場合は TDSP_CYCLE_OVERRUN アノマリーを登録する。この場合、この関数はコマンド キューを強制的に全クリアした上で、新規にタスクリストを展開する。ここ で展開されたコマンド群はこの異常が検出された次サイクルに実行されるこ とになる。

4.12.3.4 void TDSP_initialize (void)

TaskDispatcher 初期化関数

タスク実行用のタイムラインコマンドキューを初期化し、デフォルトのタス クリストBC_TL_INITIAL を展開する。

4.12.3.5 void TDSP_resync_internal_counter (void)

タスクリスト展開時刻強制同期関数

TaskDispatcher が内部に保持している、タスクリストを展開したサイクル時 刻の情報を強制的に関数呼び出し時点のサイクル時刻に一致させる。

この関数はOBC 時刻設定コマンドによって、OBC 時刻が不連続に変化した場合 にアノマリー発生を阻止する目的で呼び出すことを念頭に用意されており、 それ以外の呼び出しは想定していない。

4.12.3.6 TDSP_ACK TDSP_set_task_list_id (size_t *id*)

タスクリストID 設定関数

TaskDispatcher が毎サイクル実行するタスクリストのID を設定する。

本関数はタスクリストのID を設定するだけで、実行しているタスクリストに は即時に反映されない。この関数が呼び出されたサイクルの終わりまでは直 前に設定されていたタスクリストが有効であり、設定したタスクリストが実 行されるのは次サイクルからとなる。

Parameters

id	設定するタスクリストのID

Returns

この関数はTDSP_SUCCESS のみを返す。異常終了しない。

4.12.4 Variable Documentation

4.12.4.1 const TDSP_Info* TDSP_info

TaskDispatcher 状態変数

4.13 Core/TimeManager/OBCTime.c File Reference

Functions

- void **OBCT_clear** (**OBCTime** ∗time)
- void **OBCT_count_up** (**OBCTime** ∗time)
- **OBCTime OBCT_create** (**cycle_t** master, **cycle_t** mode, **step_t** step)
- unsigned int **OBCT_cycle2sec** (**cycle_t** cycle)
- **OBCTime OBCT_diff** (const **OBCTime** ∗before, const **OBCTime** ∗after)
- unsigned int **OBCT_diff_in_msec** (const **OBCTime** ∗before, const **OBCTime** ∗after)
- float **OBCT_diff_in_sec** (const **OBCTime** ∗before, const **OBCTime** ∗after)
- **step_t OBCT_diff_in_step** (const **OBCTime** ∗before, const **OBCTime** ∗after)
- **cycle_t OBCT_get_master_cycle** (const **OBCTime** ∗time)
- unsigned int **OBCT_get_master_in_msec** (const **OBCTime** ∗time)
- float **OBCT_get_master_in_sec** (const **OBCTime** ∗time)
- **OBCTime OBCT_get_max** (void)
- **cycle_t OBCT_get_mode_cycle** (const **OBCTime** ∗time)
- unsigned int **OBCT_get_mode_in_msec** (const **OBCTime** ∗time)
- float **OBCT_get_mode_in_sec** (const **OBCTime** ∗time)
- **step_t OBCT_get_step** (const **OBCTime** ∗time)
- void **OBCT_print** (const **OBCTime** ∗time)
- **cycle_t OBCT_sec2cycle** (unsigned int sec)

4.13.1 Function Documentation

4.13.1.1 void OBCT_clear (OBCTime ∗ *time*)

OBCTime オブジェクトクリア関数

引数で渡されたOBCTime オブジェクトのマスターサイクル、モードサイクル、 ステップを全て 0 クリアする。

Parameters

time	クリア対象のOBCTime オブジェクトへのポインタ

4.13.1.2 void OBCT_count_up (OBCTime ∗ *time*)

OBCTime オブジェクトカウントアップ関数

引数で渡されたOBCTime オブジェクトの値を 1 ステップ進める。この処理には ステップからサイクルへの繰り上げ処理と繰り上がったサイクルが最大値を 超える場合のロールオーバ処理が含まれれる。

Parameters

time	カウントアップ対象のOBCTime オブジェクトへのポインタ

4.13.1.3 OBCTime OBCT_create (cycle_t *master*, cycle_t *mode*, step_t *step*)

OBCTime オブジェクト生成関数

引数で指定したマスターサイクル、モードサイクル、ステップを設定した OBCTime オブジェクトを生成する。

Parameters

master	マスターサイクル
mode	モードサイクル
step	ステップ

Returns

引数で指定された時刻情報を設定したOBCTime オブジェクト

4.13.1.4 unsigned int OBCT_cycle2sec (cycle_t *cycle*)

サイクルから秒への変換関数

引数で渡されたサイクル数を秒数に変換して返す。

Parameters

cycle	変換したいサイクル数

Returns

変換結果の秒数

4.13.1.5 OBCTime OBCT_diff (const OBCTime * *before*, const OBCTime * *after*)

OBC 時刻間差分取得関数 OBCTime 版

引数で渡された 2 つのOBCTime オブジェクト before, after が保持する時刻 の差分 after-before を計算し結果をOBCTime オブジェクトとして返す。

減算処理を行う際にロールオーバーに対応するような処理は実装していない。 引数の before が after より前の時刻であることを前提とした実装になってい るため、逆の前後関係で引数が指定された場合の結果は不正な値となるため 注意すること。

Parameters

before	引く数に相当するOBCTime オブジェクトへのポインタ
after	引かれる数に相当するOBCTime オブジェクトへのポインタ

Returns

差分を格納したOBCTime オブジェクト

4.13.1.6 unsigned int OBCT_diff_in_msec (const OBCTime * *before*, const OBCTime * *after*)

OBC 時刻間差分取得関数 ミリ秒版

引数で渡された 2 つのOBCTime オブジェクト before, after が保持する時刻 の差分 after-before を計算し結果をミリ秒で返す。

減算処理を行う際にロールオーバーに対応するような処理は実装していない。 引数の before が after より前の時刻であることを前提とした実装になってい るため、逆の前後関係で引数が指定された場合の結果は不正な値となるため 注意すること。

Parameters

before	引く数に相当するOBCTime オブジェクトへのポインタ
after	引かれる数に相当するOBCTime オブジェクトへのポインタ

Returns

ミリ秒で表現した差分値

4.13.1.7 float OBCT_diff_in_sec (const OBCTime * *before*, const OBCTime * *after*)

OBC 時刻間差分取得関数 実数秒版

引数で渡された 2 つのOBCTime オブジェクト before, after が保持する時刻 の差分 after-before を計算し結果を小数点以下に有意な値を含む実数秒で 返す。

減算処理を行う際にロールオーバーに対応するような処理は実装していない。 引数の before が after より前の時刻であることを前提とした実装になってい るため、逆の前後関係で引数が指定された場合の結果は不正な値となるため 注意すること。

Parameters

before	引く数に相当するOBCTime オブジェクトへのポインタ
after	引かれる数に相当するOBCTime オブジェクトへのポインタ

Returns

実数秒で表現した差分値

4.13.1.8 step_t OBCT_diff_in_step (const OBCTime * *before*, const OBCTime * *after*)

OBC 時刻間差分取得関数 ステップ版

引数で渡された 2 つのOBCTime オブジェクト before, after が保持する時刻 の差分 after-before を計算し結果をステップ数で返す。

減算処理を行う際にロールオーバーに対応するような処理は実装していない。 引数の before が after より前の時刻であることを前提とした実装になってい るため、逆の前後関係で引数が指定された場合の結果は不正な値となるため 注意すること。

Parameters

before	引く数に相当するOBCTime オブジェクトへのポインタ
after	引かれる数に相当するOBCTime オブジェクトへのポインタ

Returns

ステップで表現した差分値

4.13.1.9 cycle_t OBCT_get_master_cycle (const OBCTime * *time*)

マスターサイクル取得関数

引数で渡されたOBCTime オブジェクトが保持するマスターサイクル値を返す。 OBCTime オブジェクトのポインタだけが公開され内部変数値を直接取得でき ない場合向け。

Parameters

time	値取得対象のOBCTime オブジェクトへのポインタ

Returns

指定されたオブジェクトのマスターサイクル値

4.13.1.10 unsigned int OBCT_get_master_in_msec (const OBCTime * *time*)

マスター時刻取得関数 ミリ秒版

引数で渡されたOBCTime オブジェクトが保持するマスター時刻 (マスターサイ クルとステップの組み合わせ) をミリ秒に変換して返す。マスターサイクル だけでなくステップまで加味することに注意。

Parameters

time	変換対象のOBCTime オブジェクトへのポインタ

Returns

ミリ秒で表現したマスター時刻

4.13.1.11 float OBCT_get_master_in_sec (const OBCTime * *time*)

マスター時刻取得関数 実数秒版

引数で渡されたOBCTime オブジェクトが保持するマスター時刻 (マスターサイ クルとステップの組み合わせ) を小数点以下に有効な値を含む実数秒に変換 して返す。マスターサイクルだけでなくステップまで加味することに注意。

Parameters

time	変換対象のOBCTime オブジェクトへのポインタ

Returns

実数秒で表現したマスター時刻

4.13.1.12 OBCTime OBCT_get_max (void)

最大OBCTime オブジェクト生成関数

設定上取りうる最大時刻を設定したOBCTime オブジェクトを生成する。 比較などで時刻の最大値が必要な場合に利用する。

Returns

最大時刻が設定されたOBCTime オブジェクト

4.13.1.13 cycle_t OBCT_get_mode_cycle (const OBCTime * *time*)

モードサイクル取得関数

引数で渡されたOBCTime オブジェクトが保持するモードサイクル値を返す。 OBCTime オブジェクトのポインタだけが公開され内部変数値を直接取得でき ない場合向け。

Parameters

time	値取得対象のOBCTime オブジェクトへのポインタ

Returns

指定されたオブジェクトのモードサイクル値

4.13.1.14 unsigned int OBCT_get_mode_in_msec (const OBCTime * *time*)

モード時刻取得関数 ミリ秒版

引数で渡されたOBCTime オブジェクトが保持するモード時刻 (モードサイクル とステップの組み合わせ) をミリ秒に変換して返す。モードサイクルだけで なくステップまで加味することに注意。

Parameters

time	変換対象のOBCTime オブジェクトへのポインタ

Returns

ミリ秒で表現したモード時刻

4.13.1.15 float OBCT_get_mode_in_sec (const OBCTime * *time*)

モード時刻取得関数 実数秒版

引数で渡されたOBCTime オブジェクトが保持するモード時刻 (モードサイ クルとステップの組み合わせ) を小数点以下に有効な値を含む実数秒に変換 して返す。モードサイクルだけでなくステップまで加味することに注意。

Parameters

time	変換対象のOBCTime オブジェクトへのポインタ

Returns

実数秒で表現したモード時刻

4.13.1.16 step_t OBCT_get_step (const OBCTime * *time*)

ステップ取得関数

引数で渡されたOBCTime オブジェクトが保持するステップ値を返す。 OBCTime オブジェクトのポインタだけが公開され内部変数値を直接取得でき ない場合向け。

Parameters

time	値取得対象のOBCTime オブジェクトへのポインタ

Returns

指定されたオブジェクトのステップ値

4.13.1.17 void OBCT_print (const OBCTime * *time*)

OBCTime 情報出力関数

引数で渡されたOBCTime の情報をデバッグコンソールに出力する。地上での デバッグ作業での利用を想定している。

Parameters

time	出力対象のOBCTime オブジェクトへのポインタ

4.13.1.18 cycle_t OBCT_sec2cycle (unsigned int *sec*)

秒からサイクルへの変換関数

引数で渡された秒数をサイクル数に変換して返す。

Parameters

sec	変換したい秒数

Returns

変換結果のサイクル数

4.14 Core/TimeManager/OBCTime.h File Reference

Data Structures

- struct **OBCTime**

Macros

- #define **OBCT_CYCLES_PER_SEC** (1000/**OBCT_STEP_IN_MSEC**/**OBCT_STEPS_PER_CYCLE**)
- #define **OBCT_MAX_CYCLE** (0xfffffff0u)
- #define **OBCT_STEP_IN_MSEC** (1)
- #define **OBCT_STEPS_PER_CYCLE** (100)

Typedefs

- typedef unsigned int **cycle_t**
- typedef unsigned int **step_t**

Functions

- void **OBCT_clear** (**OBCTime** ∗time)
- void **OBCT_count_up** (**OBCTime** ∗time)
- **OBCTime OBCT_create** (**cycle_t** master, **cycle_t** mode, **step_t** step)
- unsigned int **OBCT_cycle2sec** (**cycle_t** cycle)
- **OBCTime OBCT_diff** (const **OBCTime** ∗before, const **OBCTime** ∗after)
- unsigned int **OBCT_diff_in_msec** (const **OBCTime** ∗before, const **OBCTime** ∗after)
- float **OBCT_diff_in_sec** (const **OBCTime** ∗before, const **OBCTime** ∗after)
- **step_t OBCT_diff_in_step** (const **OBCTime** ∗before, const **OBCTime** ∗after)
- **cycle_t OBCT_get_master_cycle** (const **OBCTime** ∗time)
- unsigned int **OBCT_get_master_in_msec** (const **OBCTime** ∗time)
- float **OBCT_get_master_in_sec** (const **OBCTime** ∗time)
- **OBCTime OBCT_get_max** (void)
- **cycle_t OBCT_get_mode_cycle** (const **OBCTime** ∗time)
- unsigned int **OBCT_get_mode_in_msec** (const **OBCTime** ∗time)
- float **OBCT_get_mode_in_sec** (const **OBCTime** ∗time)

- **step_t OBCT_get_step** (const **OBCTime** ∗time)
- void **OBCT_print** (const **OBCTime** ∗time)
- **cycle_t OBCT_sec2cycle** (unsigned int sec)

4.14.1 Macro Definition Documentation

4.14.1.1 #define OBCT_CYCLES_PER_SEC (1000/OBCT_STEP_IN_MSEC/OBCT_STEPS_PER_CYCLE)

1 秒あたりのサイクル数

4.14.1.2 #define OBCT_MAX_CYCLE (0xfffffff0u)

サイクル数の最大値

符号なし 32bit 値の上限ではなく、20 で割り切れる上限値を指定している。 ほどよしでの時刻定義が 20cycles/sec であり、秒単位を念頭においた剰余処 理でロールオーバ発生時に値の不連続が生じないよう配慮した。

本当は OBCT_CYCLES_PER_SEC の値に応じて上限値が調整されるべき。

4.14.1.3 #define OBCT_STEP_IN_MSEC (1)

1 ステップあたりのミリ秒数

4.14.1.4 #define OBCT_STEPS_PER_CYCLE (100)

1 サイクルあたりのステップ数

4.14.2 Typedef Documentation

4.14.2.1 typedef unsigned int cycle_t

サイクル数を表現する型

4.14.2.2 typedef unsigned int step_t

ステップ数を表現する型

4.14.3 Function Documentation

4.14.3.1 void OBCT_clear (OBCTime ∗ *time*)

OBCTime オブジェクトクリア関数

引数で渡されたOBCTime オブジェクトのマスターサイクル、モードサイクル、 ステップを全て 0 クリアする。

Parameters

time	クリア対象のOBCTime オブジェクトへのポインタ

4.14.3.2 void OBCT_count_up (OBCTime * *time*)

OBCTime オブジェクトカウントアップ関数

引数で渡されたOBCTime オブジェクトの値を 1 ステップ進める。この処理には ステップからサイクルへの繰り上げ処理と繰り上がったサイクルが最大値を 超える場合のロールオーバ処理が含まれれる。

Parameters

time	カウントアップ対象のOBCTime オブジェクトへのポインタ

4.14.3.3 OBCTime OBCT_create (cycle_t *master*, cycle_t *mode*, step_t *step*)

OBCTime オブジェクト生成関数

引数で指定したマスターサイクル、モードサイクル、ステップを設定した OBCTime オブジェクトを生成する。

Parameters

master	マスターサイクル
mode	モードサイクル
step	ステップ

Returns

引数で指定された時刻情報を設定したOBCTime オブジェクト

4.14.3.4 unsigned int OBCT_cycle2sec (cycle_t *cycle*)

サイクルから秒への変換関数

引数で渡されたサイクル数を秒数に変換して返す。

Parameters

cycle	変換したいサイクル数

Returns

変換結果の秒数

4.14.3.5 OBCTime OBCT_diff (const OBCTime * *before*, const OBCTime * *after*)

OBC 時刻間差分取得関数 OBCTime 版

引数で渡された 2 つのOBCTime オブジェクト before, after が保持する時刻 の差分 after-before を計算し結果をOBCTime オブジェクトとして返す。

減算処理を行う際にロールオーバーに対応するような処理は実装していない。 引数の before が after より前の時刻であることを前提とした実装になってい るため、逆の前後関係で引数が指定された場合の結果は不正な値となるため 注意すること。

Parameters

before	引く数に相当するOBCTime オブジェクトへのポインタ

after	引かれる数に相当するOBCTime オブジェクトへのポインタ

Returns

差分を格納したOBCTime オブジェクト

4.14.3.6 unsigned int OBCT_diff_in_msec (const OBCTime * *before*, const OBCTime * *after*)

OBC 時刻間差分取得関数 ミリ秒版

引数で渡された 2 つのOBCTime オブジェクト before, after が保持する時刻 の差分 after-before を計算し結果をミリ秒で返す。

減算処理を行う際にロールオーバーに対応するような処理は実装していない。 引数の before が after より前の時刻であることを前提とした実装になってい るため、逆の前後関係で引数が指定された場合の結果は不正な値となるため 注意すること。

Parameters

before	引く数に相当するOBCTime オブジェクトへのポインタ
after	引かれる数に相当するOBCTime オブジェクトへのポインタ

Returns

ミリ秒で表現した差分値

4.14.3.7 float OBCT_diff_in_sec (const OBCTime * *before*, const OBCTime * *after*)

OBC 時刻間差分取得関数 実数秒版

引数で渡された 2 つのOBCTime オブジェクト before, after が保持する時刻 の差分 after-before を計算し結果を小数点以下に有意な値を含む実数秒で 返す。

減算処理を行う際にロールオーバーに対応するような処理は実装していない。 引数の before が after より前の時刻であることを前提とした実装になってい るため、逆の前後関係で引数が指定された場合の結果は不正な値となるため 注意すること。

Parameters

before	引く数に相当するOBCTime オブジェクトへのポインタ
after	引かれる数に相当するOBCTime オブジェクトへのポインタ

Returns

実数秒で表現した差分値

4.14.3.8 step_t OBCT_diff_in_step (const OBCTime * *before*, const OBCTime * *after*)

OBC 時刻間差分取得関数 ステップ版

引数で渡された 2 つのOBCTime オブジェクト before, after が保持する時刻 の差分 after-before を計算し結果をステップ数で返す。

減算処理を行う際にロールオーバーに対応するような処理は実装していない。 引数の before が after より前の時刻であることを前提とした実装になってい るため、逆の前後関係で引数が指定された場合の結果は不正な値となるため 注意すること。

Parameters

before	引く数に相当するOBCTime オブジェクトへのポインタ
after	引かれる数に相当するOBCTime オブジェクトへのポインタ

Returns

ステップで表現した差分値

4.14.3.9 cycle_t OBCT_get_master_cycle (const OBCTime * *time*)

マスターサイクル取得関数

引数で渡されたOBCTime オブジェクトが保持するマスターサイクル値を返す。 OBCTime オブジェクトのポインタだけが公開され内部変数値を直接取得でき ない場合向け。

Parameters

time	値取得対象のOBCTime オブジェクトへのポインタ

Returns

指定されたオブジェクトのマスターサイクル値

4.14.3.10 unsigned int OBCT_get_master_in_msec (const OBCTime * *time*)

マスター時刻取得関数 ミリ秒版

引数で渡されたOBCTime オブジェクトが保持するマスター時刻 (マスターサイ クルとステップの組み合わせ) をミリ秒に変換して返す。マスターサイクル だけでなくステップまで加味することに注意。

Parameters

time	変換対象のOBCTime オブジェクトへのポインタ

Returns

ミリ秒で表現したマスター時刻

4.14.3.11 float OBCT_get_master_in_sec (const OBCTime * *time*)

マスター時刻取得関数 実数秒版

引数で渡されたOBCTime オブジェクトが保持するマスター時刻 (マスターサイ クルとステップの組み合わせ) を小数点以下に有効な値を含む実数秒に変換 して返す。マスターサイクルだけでなくステップまで加味することに注意。

Parameters

time	変換対象のOBCTime オブジェクトへのポインタ

Returns

実数秒で表現したマスター時刻

4.14.3.12 OBCTime OBCT_get_max (void)

最大OBCTime オブジェクト生成関数

設定上取りうる最大時刻を設定したOBCTime オブジェクトを生成する。 比較などで時刻の最大値が必要な場合に利用する。

Returns

最大時刻が設定されたOBCTime オブジェクト

4.14.3.13 cycle_t OBCT_get_mode_cycle (const OBCTime * *time*)

モードサイクル取得関数

引数で渡されたOBCTime オブジェクトが保持するモードサイクル値を返す。 OBCTime オブジェクトのポインタだけが公開され内部変数値を直接取得でき ない場合向け。

Parameters

time	値取得対象のOBCTime オブジェクトへのポインタ

Returns

指定されたオブジェクトのモードサイクル値

4.14.3.14 unsigned int OBCT_get_mode_in_msec (const OBCTime * *time*)

モード時刻取得関数 ミリ秒版

引数で渡されたOBCTime オブジェクトが保持するモード時刻 (モードサイクル とステップの組み合わせ) をミリ秒に変換して返す。モードサイクルだけで なくステップまで加味することに注意。

Parameters

time	変換対象のOBCTime オブジェクトへのポインタ

Returns

ミリ秒で表現したモード時刻

4.14.3.15 float OBCT_get_mode_in_sec (const OBCTime * *time*)

モード時刻取得関数 実数秒版

引数で渡されたOBCTime オブジェクトが保持するモード時刻 (モードサイ クルとステップの組み合わせ) を小数点以下に有効な値を含む実数秒に変換 して返す。モードサイクルだけでなくステップまで加味することに注意。

Parameters

time	変換対象のOBCTime オブジェクトへのポインタ

Returns

実数秒で表現したモード時刻

4.14.3.16 step_t OBCT_get_step (const OBCTime * time)

ステップ取得関数

引数で渡されたOBCTime オブジェクトが保持するステップ値を返す。 OBCTime オブジェクトのポインタだけが公開され内部変数値を直接取得でき ない場合向け。

Parameters

time	値取得対象のOBCTime オブジェクトへのポインタ

Returns

指定されたオブジェクトのステップ値

4.14.3.17 void OBCT_print (const OBCTime * time)

OBCTime 情報出力関数

引数で渡されたOBCTime の情報をデバッグコンソールに出力する。地上での デバッグ作業での利用を想定している。

Parameters

time	出力対象のOBCTime オブジェクトへのポインタ

4.14.3.18 cycle_t OBCT_sec2cycle (unsigned int sec)

秒からサイクルへの変換関数

引数で渡された秒数をサイクル数に変換して返す。

Parameters

sec	変換したい秒数

Returns

変換結果のサイクル数

4.15 Core/TimeManager/StopWatch.c File Reference

Macros

- #define **STW_MAX_RECORD_** (10)

Functions

- void **STW_lap** (const char *tag)
- void **STW_print** (void)
- void **STW_start** (void)

4.15.1 Macro Definition Documentation

4.15.1.1 #define STW_MAX_RECORD_ (10)

4.15.2 Function Documentation

4.15.2.1 void STW_lap (const char * *tag*)

4.15.2.2 void STW_print (void)

4.15.2.3 void STW_start (void)

4.16 Core/TimeManager/StopWatch.h File Reference

Functions

- void **STW_lap** (const char *tag)
- void **STW_print** (void)
- void **STW_start** (void)

4.16.1 Function Documentation

4.16.1.1 void STW_lap (const char * *tag*)

4.16.1.2 void STW_print (void)

4.16.1.3 void STW_start (void)

4.17 Core/TimeManager/TimeManager.c File Reference

Functions

- int **Cmd_TMGR_SET_TIME** (const CTCP *packet)
- void **TMGR_clear_mode_cycle** (void)
- void **TMGR_count_up** (void)
- unsigned int **TMGR_get_master_in_msec** (void)
- unsigned int **TMGR_get_mode_in_msec** (void)
- void **TMGR_init** (void)

Variables

- const **OBCTime** * **master_clock**

4.17.1 Function Documentation

4.17.1.1 int Cmd_TMGR_SET_TIME (const CTCP * *packet*)

マスターサイクル設定コマンド

マスターサイクルをパラメータで指定された値に設定する。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常
CCP_EXEC_ILLEGAL_P↩ ARAMETER	指定時刻が設定上限値以上

4.17.1.2 void TMGR_clear_mode_cycle (void)

モードサイクル初期化関数

モードサイクルを 0 にクリアする。 モード遷移時にModeManager から呼び出される。ModeManager からの呼び出し専用で、それ以外の箇所から呼び出すことは想定していない。

4.17.1.3 void TMGR_count_up (void)

システム時刻更新関数

システム時刻を 1step 進める。 計算機のタイマ割り込みによって呼び出される。 それ以外の箇所から呼び出すことは想定していない。

4.17.1.4 unsigned int TMGR_get_master_in_msec (void)

ミリ秒単位でのマスターサイクル取得関数

マスターサイクルをミリ秒単位に変換した結果を返す。 1 サイクルが 1 ミリ秒より大きい場合、変換結果はサイクルよりも 早い段階でロールオーバーすることになるため値の取扱いに注意すること。

Returns

ミリ秒単位に変換したマスターサイクル

4.17.1.5 unsigned int TMGR_get_mode_in_msec (void)

ミリ秒単位でのモードサイクル取得関数

モードサイクルをミリ秒単位に変換した結果を返す。 1 サイクルが 1 ミリ秒より大きい場合、変換結果はサイクルよりも 早い段階でロールオーバーすることになるため値の取扱いに注意すること。

Returns

ミリ秒単位に変換したモードサイクル

4.17.1.6 void TMGR_init (void)

システム時刻初期化関数

マスターサイクル、モードサイクル、ステップすべてを 0 に設定し、 他からの参照用に master_clock を設定する。 システム起動時の初期化処理で呼び出される。

4.17.2 Variable Documentation

4.17.2.1 const OBCTime* master_clock

システム時刻参照変数

4.18 Core/TimeManager/TimeManager.h File Reference

Functions

- int **Cmd_TMGR_SET_TIME** (const CTCP *packet)
- void **TMGR_clear_mode_cycle** (void)
- void **TMGR_count_up** (void)
- unsigned int **TMGR_get_master_in_msec** (void)
- unsigned int **TMGR_get_mode_in_msec** (void)
- void **TMGR_init** (void)

Variables

- const **OBCTime** * **master_clock**

4.18.1 Function Documentation

4.18.1.1 int Cmd_TMGR_SET_TIME (const CTCP * *packet*)

マスターサイクル設定コマンド

マスターサイクルをパラメータで指定された値に設定する。

Parameters

packet	コマンドパケット

Return values

CCP_EXEC_SUCCESS	正常終了
CCP_EXEC_ILLEGAL_L↩ ENGTH	パラメータ長異常
CCP_EXEC_ILLEGAL_P↩ ARAMETER	指定時刻が設定上限値以上

4.18.1.2 void TMGR_clear_mode_cycle (void)

モードサイクル初期化関数

モードサイクルを 0 にクリアする。 モード遷移時にModeManager から呼び出される。ModeManager からの呼び出し専用で、それ以外の箇所から呼び出すことは想定していない。

4.18.1.3 void TMGR_count_up (void)

システム時刻更新関数

システム時刻を 1step 進める。 計算機のタイマ割り込みによって呼び出される。 それ以外の箇所から呼び出すことは想定していない。

4.18.1.4 unsigned int TMGR_get_master_in_msec (void)

ミリ秒単位でのマスターサイクル取得関数

マスターサイクルをミリ秒単位に変換した結果を返す。 1 サイクルが 1 ミリ秒より大きい場合、変換結果はサイクルよりも 早い段階でロールオーバーすることになるため値の取扱いに注意すること。

Returns

ミリ秒単位に変換したマスターサイクル

4.18.1.5 unsigned int TMGR_get_mode_in_msec (void)

ミリ秒単位でのモードサイクル取得関数

モードサイクルをミリ秒単位に変換した結果を返す。 1 サイクルが 1 ミリ秒より大きい場合、変換結果はサイクルよりも 早い段階でロールオーバーすることになるため値の取扱いに注意すること。

Returns

ミリ秒単位に変換したモードサイクル

4.18.1.6 void TMGR_init (void)

システム時刻初期化関数

マスターサイクル、モードサイクル、ステップすべてを 0 に設定し、 他からの参照用に master_clock を設定する。 システム起動時の初期化処理で呼び出される。

4.18.2 Variable Documentation

4.18.2.1 const OBCTime* master_clock

システム時刻参照変数

Index