博士論文

モデルベースシステムズエンジニアリングを活用した

宇宙機搭載自律プランナーの構築手法に関する研究

Development of Onboard Planner for Autonomous Spacecraft

Using Model-Based Systems Engineering

東京大学大学院工学系研究科航空宇宙工学専攻
学籍番号　37-167064
中島　晋太郎

指導教官　船瀬龍准教授

2018 年 11 月 30 日　提出

論文要旨

近年，軌道上に打ちあがる宇宙機の数が地球周回・深宇宙問わず増加してきており，宇宙機に自律機能を搭載する必要性が高まっている．本研究における自律機能は，「宇宙機の運用を宇宙機上で自動化する機能」としている．自律機能として，宇宙機の振る舞いの全パターンをルールとして事前に定める「If-then ルールベース」の自律機能と，宇宙機と外部環境のモデルを持ち，そのモデルをもとにどのような行動をすべきかを自ら探索する「モデルベース」の自律機能がある．未知の状況にも柔軟な対応ができるというカバレッジの観点で，「モデルベース」の自律機能である「宇宙機搭載自律プランナー」を構築することを目指すが，探索に利用するモデルの構築は属人的になっているとともに，検証が難しいという問題がある．宇宙機搭載自律プランナーに利用するモデルには，宇宙機の振る舞いに関する内容が記述されるが，その内容は衛星の設計時に定まるはずである．そこで，宇宙機の設計情報を正確に記述し，その情報をもとにプランナーに用いるモデルを構築することで，モデル構築の属人性や検証の必要性を低減することを目指している．

本研究では，人間によって行われる運用の一部を宇宙機側で自動化するというモチベーションをもとに宇宙機搭載自律プランナーの目的を定めた．宇宙機の運用内容を「打ち上げ後の初期の搭載機器動作確認」「宇宙機の軌道解析・決定」「異常検知・対処」「ミッション運用」の四項目に大別し，このうち後半の二項目を自動化することを本研究で構築する自律プランナーの目的とした．具体的には，「運用目標が与えられたときに，自動計画アルゴリズムを用いて，運用目標を達成できる方策を自ら探索し，実行できること」と「ミッション運用を行う際に，異常が発生した場合であっても，それに対処をしたうえで運用目的の達成を目指すこと」の二つと定義した．続いて，この目的を達成するための宇宙機搭載自律プランナーの構成要素を，「Planning Engine」「Low Level Controller」「Operator」「State Monitoring」「Executor」の五つに整理した．これらの構成要素を実装するのに必要な情報について議論を進めていくと，「ドメインモデルの記述内容」と「Low Level Controller に関する情報」，「ドメインモデルと Low Level Controller に関する情報の関係性」に集約されることを導いた．後者の自由度は非常に限られているため，前者に関して深く議論を進めることで，宇宙機の自動計画問題を記述するためには PDDL 2.1 以降のプランニング記述言語を用いればよいことを示した．PDDL 2.1 以降で宇宙機の自動計画問題を記述することにより，自動計画問題を解くためのアルゴリズムは既存のものを利用でき，アルゴリズムの検証が不要となる．さらに，搭載ソフトウェア・アーキテクチャの一つである Command Centric Architecture（C2A）を Low Level Controller に用いることによって，テレメトリ・コマンドのデータベースの記述方法を示すだけでなく，テレメトリ・コマンドとドメインモデルの関係性のシステマティックな記述方法を提案している．

続いて，宇宙機の運用と自動計画アルゴリズムの相似性から，宇宙機の振る舞いを記述するドメインモデルの内容は宇宙機の設計情報，特にコマンド・テレメトリの情報と深い関連

性があることを整理した．この関連性から，宇宙機のドメインモデルを正確に記述するためには，コマンド・テレメトリを含めた設計情報が正確であることが重要であると考え，コマンド・テレメトリの設計情報を正確に構成するためにモデルベースシステムズエンジニアリング（MBSE）のプロセスを利用することとした．MBSE のプロセスによりシステムレベルの抽象的な要求をもとにトレーサビリティを取りつつ搭載コンポーネントの設計を行うことができ，搭載コンポーネントの設計からコマンド・テレメトリの設計情報が定まる．また，抽象的なモデルの段階で設計の妥当性確認を行うことができるだけでなく，製造した結果からもモデルの修正を行うことよって，検証された設計モデルの構築が可能である．MBSE のプロセスを進める中で自然と描かれるコンポーネントの状態遷移図やシーケンス図によって，コマンドやテレメトリの設計に関する情報が抽出できることが示唆された．この結果をもとに，本研究では，MBSE のプロセスによって自然と記述される状態遷移図やシーケンス図といったダイアグラムをもとに，宇宙機のドメインモデルである PDDL ファイルを自動生成することによって，宇宙機の検証された設計情報をもとに自律プランナーの中心をなすドメインモデルを構成するという一連のプロセスを提案している．

本研究では，提案したプロセスを東京大学と JAXA が中心となって開発を進めている，EQUULEUS という宇宙機に適用し，実際の宇宙機搭載計算機で実験を行った．MBSE を進めていく上での Methodology の例として MagicGrid を採用し，ダイアグラムの生成を行った．コンポーネントの状態遷移図に，状態遷移の詳細な条件という付加情報を加えることで，状態遷移図からコンポーネントのドメインモデルを構成できることを確認した．続いて，各コンポーネントのドメインモデルを統合して宇宙機のドメインモデルを構成し，自動計画実験を行ったところ，抽象的なミッション目的を達成する方策を自動で探索できるだけでなく，簡易な故障に対しても対処を行えることを確認できた．最後に，実際の宇宙機に搭載する計算機を利用して自律プランナー全体の検証を行い，設計モデルをもとに構成した自律プランナーによって，「異常検知・対処」「ミッション運用」の自動化が行えることを確認した．

以上の成果は，これまで検証プロセスが明確でなく，信頼性が低かった自動計画アルゴリズムベースの自律機能をより簡易に，かつ信頼性を保って構築することを可能にしており，より高度な自律機能を持った宇宙システムを実現させる礎となるものである．

目次

図目次

表目次

第1章 序論

1.1 研究背景

1.1.1 超小型衛星の広がりによる軌道上衛星数の増加

近年, CubeSat[1]をはじめとする超小型衛星が実用的になり, 産学官問わず様々な団体が超小型衛星を利用したミッションを計画し, 実施している. 図 1.1 に, SpaceWorks による, 重量が 1 kg から 50 kg の超小型衛星の打ち上げ数予測[2]を示す. 2014 年から 2016 年にかけて打ち上げ数が落ち込んでいるものの, 2013 年以降は 100 機以上の超小型衛星が軌道上に打ち上げられており, 今後も打ち上げ数が増加していくことが予測されている.

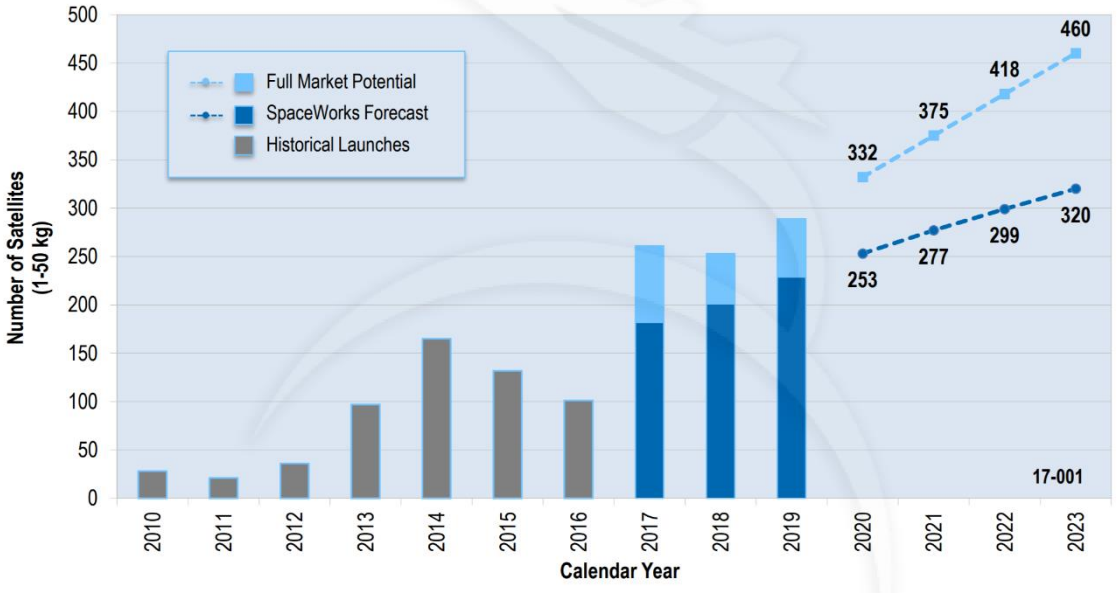

図 1.1 重量が 1 kg から 50 kg である超小型衛星の市場予測[2]

超小型衛星の利用法として特に注目を集めているのが, 多数の人工衛星を利用した衛星コンステレーションによるミッションである. ビジネスにおける具体例としては, Axelspace による地球観測網構築[3], Planet Labs による地球観測網構築[4], 少し衛星サイズが大きくなるが OneWeb による衛星通信[5]などが挙げられる. これらのミッションに使用される衛星とその数をまとめたものを表 1.1 に示す. 表 1.1 に示される 3 ミッションのみであっても, 利用する衛星数の合計は 1000 機を超えるため, 将来のミッションも考慮すると軌道上の衛星数が爆発的に増加することが予測できる.

表 1.1 衛星コンステレーションの例

企業名	衛星サイズ	衛星数	軌道	ミッション
Axelspace	50 kg	50 機	LEO	地球観測
Planet	3U	200 機以上	LEO	地球観測
OneWeb	150 kg	900 機	LEO	衛星通信

地球周回軌道において衛星コンステレーションを利用したミッションが広がる一方で，深宇宙においても超小型衛星の利用が広がりつつある．2014 年 12 月 3 日に打ち上げられたはやぶさ 2 [6]においては，PROCYON[7]をはじめとする 3 機の小型副ペイロード[8]が相乗りで打ち上げられた．現在計画されている深宇宙への打ち上げ機会としては，SLS EM-1 ロケットによる 13 機の 6U (1U = 10 cm × 10 cm × 10cm) CubeSat の相乗り[9]がある．今後，SLS EM-2 ロケットにおいても同様の相乗りを実施する可能性が示唆されている[9].

以上のように，地球周回・深宇宙問わず，超小型衛星を利用したミッションが広がってきており，将来的に軌道上で運用される宇宙機の数が大きく増加していくことが予想されている．

1.1.2 宇宙機の運用と将来的な問題点

軌道上の宇宙機は，地上にある運用局との間で電波を介してデータのやり取りを行い，目的を達成するための運用をなされる．運用の概念図を図 1.2 に示す．宇宙機は，搭載している各機器やソフトウェアの状態，取得した各種データをテレメトリデータとして生成する．生成されたテレメトリデータは電波を介し，宇宙機のアンテナから宇宙空間を伝搬して地上にあるアンテナへと送られる．地上のアンテナで受信した電波を復調し，得られたデータを運用局で運用者が確認する．運用者は，得られたデータをもとに次の目的を達成するための運用計画（コマンド列）を作成する．作成されたコマンドデータは，テレメトリデータとは逆の順序で地上局から軌道上の宇宙機に送信され，宇宙機が実行する．このように，宇宙機を運用するためには，受信機能だけでなく送信機能を持った地上局が必要である．

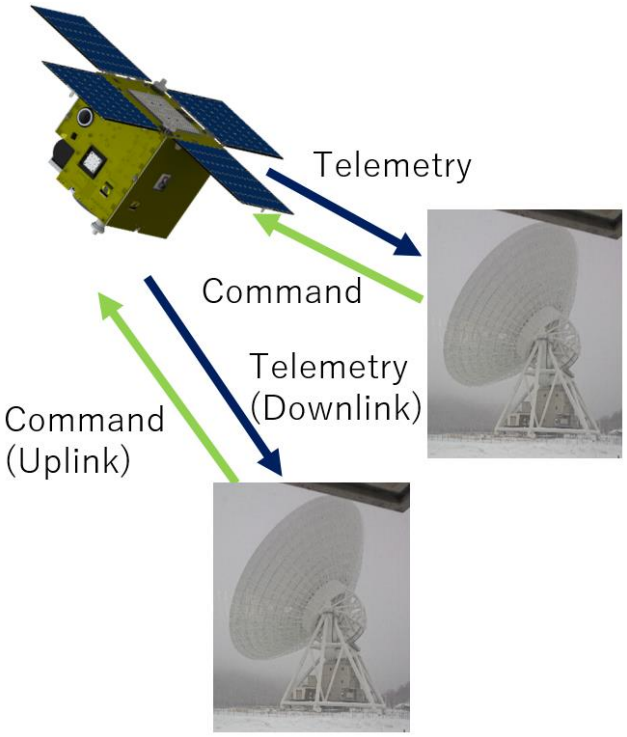

図 1.2 運用の概念図

表 1.2 PROCYON の運用計画表（2014 年 12 月分）

日時				DOY	運用時間 局	時間	JST 開始	JST 終了	UT 開始	UT 終了	主たる運用項目
		3	水	337	UDSC	5:24	20:45	2:09	11:45	17:09	バス健全性確認
		4	木	338	UDSC	5:21	20:53	2:14	11:53	17:14	バス健全性確認
		5	金	339	UDSC	5:23	20:51	2:14	11:51	17:14	バス健全性確認
		6	土	340	USC34	7:00	20:15	3:15	10:29	17:30	STT ON
		7	日	341	USC34	7:00	20:15	3:15	11:15	18:15	LAICA 動作確認
		8	月	342	USC34	7:00	20:10	3:10	11:10	18:10	STT動作確認
		9	火	343	USC34	7:00	20:10	3:10	11:10	18:10	MGA地球指向
		10	水	344	USC34	7:00	20:10	3:10	11:10	18:10	レンジング
		11	木	345	USC34	7:00	20:05	3:05	11:05	18:05	VLBI+MGA　動作確認
		12	金	346							
		13	土	347							
		14	日	348							
		15	月	349	USC34	7:00	19:50	2:50	10:50	17:50	ITU動作確認
		16	火	350	USC34	7:00	19:45	2:45	10:45	17:45	ITU動作確認
		17	水	351	USC34	7:00	19:40	2:40	10:40	17:40	LAICA状態確認
		18	木	352	USC34	6:40	20:30	3:10	11:30	18:10	LAICA高圧ON
		19	金	353							
		20	土	354							
		21	日	355							
		22	月	356	USC34	7:00	19:15	2:15	10:15	17:15	MGA地球指向
		23	火	357	USC34	7:00	19:15	2:15	10:15	17:15	DR再生
		24	水	358	USC34	7:00	19:10	2:10	10:10	17:10	LAICA撮影
		25	木	359	USC34	7:00	19:05	2:05	10:05	17:05	STT異常検知ロジック動作確認
		26	金	360							
		27	土	361							
		28	日	362	USC34	7:00	19:05	2:05	10:05	17:05	ITU動作確認
		29	月	363	USC34	7:00	18:45	1:45	9:45	16:45	LAICA撮影
		30	火	364	USC34	7:00	18:40	1:40	9:40	16:40	LAICA撮影(校正)
		31	水	365							

ここで，宇宙機の運用内容を整理する．表 1.2 に，PROCYON が打ち上げられた 2014 年 12 月の運用計画表を示す．この運用計画は，中長期的な運用目標をもとに各運用日における運用目標を定めることで作成されている．この運用計画表と実際の運用報告書，運用計画コマンドファイルをもとに，PROCYON の運用項目の整理を試みたところ，運用項目を以下のように 4 種類に大別できた．

- 打ち上げ後の初期の搭載機器動作確認
  - 姿勢系機器（STT）動作確認
  - 推進系機器（ITU）動作確認
  - 通信系機器（MGA）動作確認
- 宇宙機の軌道解析・決定
  - レンジング
  - VLBI 観測
- 異常検知・対処
  - 運用中健全性確認
  - 非可視中データ確認
  - ソフトウェア異常対処
- ミッション運用
  - 観測機器（LAICA）での撮影
  - VLBI 観測

こうした運用は地上局アンテナを用いて実施される．1.1.2 項で述べたように，将来的に軌道上の宇宙機数が増加していくことが予想されるが，そのすべてに対して地上局アンテナを用いて運用を行おうとすると，様々な問題が発生することが予想できる．

一つ目の問題点として，運用に必要な各種コストの増加が挙げられる．地上局と軌道上の宇宙機との間の通信は，特にコマンドにおいては一対一で実施されることを想定している．そのため占有率や周波数など，何らかの問題で既存の地上局アンテナを使用できない場合，新規に地上局アンテナを建設する必要に迫られ，地上局設備の建設・維持コストが発生する．また，電波を利用して通信を実施するため，運用時には少なくとも無線従事者をアサインする必要がある．そのうえ，宇宙機の状況に応じた運用を実施するために，運用担当者もアサインしなければならない．したがって，運用人員のコストも発生する．

二つ目の問題として，深宇宙地上局の利用性の問題が挙げられる．深宇宙探査機との通信においては，通信距離が非常に大きくなってくるため，搭載通信機だけでなく地上局の性能が高いことが重要な要素になっている．日本で深宇宙探査機の運用に用いられているアンテナは主に臼田局と内之浦局の二つのみであり，アメリカの DSN（Deep Space Network）

においても運用で使用されているアンテナは 14 基のみである．地球周回衛星だけでなく深宇宙へ航行する宇宙機の数が増加すると，このような数少ない深宇宙局ですべての宇宙機の運用を行うことが難しくなってくると考えられる．

1.1.3 宇宙機運用の課題に対する解決策

前項では，宇宙機運用の概要を説明したうえで，宇宙機運用で将来的に生じうる課題を述べた．前項で提示した課題を解決するには，いくつかの手段があるので，それを紹介する．

一つ目の手段としては，世界各地に散らばる地上局を互いにシェアし，有効活用することである．日本で行われた取り組みの例としては，UNISEC（大学宇宙工学コンソーシアム）で執り行われている，GSN（Ground Station Network）ワーキンググループの活動がある[10][11]．GSN では，日本各地にある地上局間のネットワーク運用システムを構築することを目指して活動しており，実際に東京大学においては PRISM 衛星[12]の運用において，東京大学構内の地上局に加え，電気通信大学の地上局を併用して運用を実施したことがある．他にも，Infostellar 社[13]による地上局共有プラットフォームや，Libre Space Fundation 社によるオープンソースの地上局ネットワークである SatNOGS[14]などが同様の事例として挙げられる．これらの手法は，地球周回衛星の運用に対しては非常に強力な手段となるが，各国の宇宙機関が中心となって建設した，数少ない大型の地上局アンテナを使用せざるを得ない深宇宙探査機に対しては有効な解決策とならない．

二つ目の手段としては，一つの地上局で複数の宇宙機の運用を実施することが考えられる．過去に実施された例としては，はやぶさ 2 の運用時における，PROCYON のテレメトリ電波の同時受信実験[15]が挙げられる．また，今後検討されているものとして，SLS EM-1 ロケットで打ち上げられる 13 機の CubeSat の運用を DSN で実施する場合に，アンテナ 1 基あたりに 4 機程度の宇宙機を割り当て，運用時間内では 4 機すべてのテレメトリデータを取得しつつ，コマンド運用に関しては 1 機ずつ時間を分けて行う，という計画[16]がある．このような，一つの地上局を複数機でシェアして運用する際の問題点としては，シェアする宇宙機同士がアンテナの Same Beam に存在しなければならないという制約条件がある上に，コマンド運用は一度に 1 機に対してしか実施できない，ということが挙げられる．

三つ目の手段としては，運用の自動化である．前項で述べた通り，宇宙機の運用は地上局アンテナと宇宙機の間で電波を介した無線通信によってデータのやり取りを行い，実施される．そのため，宇宙機の運用を自動化する場合，地上局側において運用に関する処理を自動化するケースと，宇宙機側において運用に関する処理を自動化するケースの二通りが考えられる．表 1.3 に，宇宙機側において自動化を実施する場合と，地上局側で自動化を実施する場合の比較を簡単に行った結果を示す．宇宙機に搭載できる計算機は，打ち上げ重量や利用できる電力，周囲の熱環境，放射線環境等の理由で，地上局設備に設置できる計算機に比べて性能が低くなる．一方で，地上局ベースで自動化を行おうとした場合には，利用する

データにおいて二つの問題点が挙げられる．一つは，宇宙機から地上局まで電波が到達するまでに伝搬遅延がある上に，通信が地上局からの可視時間に限られるという「リアルタイム性」の問題である．もう一つの問題は，一般的な宇宙機の運用においては，通信速度と可視時間の制約により，宇宙機上で生成されるデータをすべて地上局にダウンリンクすることはできないという点である．そのため，実際の運用においては，重要なデータを抜粋したり，サンプリング周期を落としたりといった処理を実施した後のデータを地上局にダウンリンクする．つまり，地上局には一部のデータしかダウンリンクされない，ということである．

表 1.3 宇宙機側での自動化と地上局側での自動化の比較

	計算リソース	リアルタイム性	情報量
宇宙機側での自動化	×（低性能）	**〇（伝搬遅延なし）**	**〇（全ての生データ）**
地上局側での自動化	**〇（高性能）**	×（伝搬遅延あり）	×（抜粋されたデータ）

宇宙機の運用においては，表 1.3 の項目の中でリアルタイム性が非常に重要となる局面が多数存在する．それが顕著に出た例として，火星探査機「のぞみ」の地球スイングバイ運用[17]が挙げられる．のぞみにおいては，地球スイングバイ時の非可視時に発生した推進系トラブルにすぐ対応できなかったことで，最終的には火星周回軌道への投入を断念することになってしまった．

本研究においては，将来的な宇宙機運用の課題を解決するために，上記の「リアルタイム性の重要性」の重要性を鑑み，計算リソースが限られることを理解したうえで，宇宙機側で運用に関する処理を自動化することを目指す．「宇宙機側での運用に関する自動処理」を，以降では簡単に「自律機能」と呼ぶことにする．

1.2 宇宙機搭載自律機能に関する先行研究

宇宙機に自律機能を搭載する試みは，以前から多く行われてきている．本節では，宇宙機に自律機能を搭載する試みをいくつか紹介していく．

1.2.1 If-then ルールベースの自律機能

多くの宇宙機に搭載されている自律機能としては，If-then ルールベースの自動化が挙げられる．すなわち，宇宙機自身で判断する場合分けを事前に定めておき，打ち上げ後は事前に定められた場合分けのルールに沿って宇宙機が判断し，行動する，というものである．具体例として，PROCYON に搭載された自律機能[18]を二つ紹介する．

一つ目は，ガス漏れ監視・対処機能である．これは，推進系の一部圧力を監視し，異常な圧力値が観測されるとガス漏れと判断して上流の遮断弁を閉じる等の対処を実施する機能であり，図 1.3 で示されるような動作イメージである．これを，If-then ルールで書き下すと，以下のようになる．

If [圧力値 ＞ 閾値] then [バルブ閉止]

二つ目は，Under Voltage Control（UVC）機能である．これは，姿勢異常などの理由によって電力収支が取れない期間が継続した際に，バッテリーを過放電から保護するために対応を行う機能である．これを，If-then ルールで書き下すと，以下のようになる．

If [バッテリー電圧 ＜ 閾値] then [太陽指向制御]

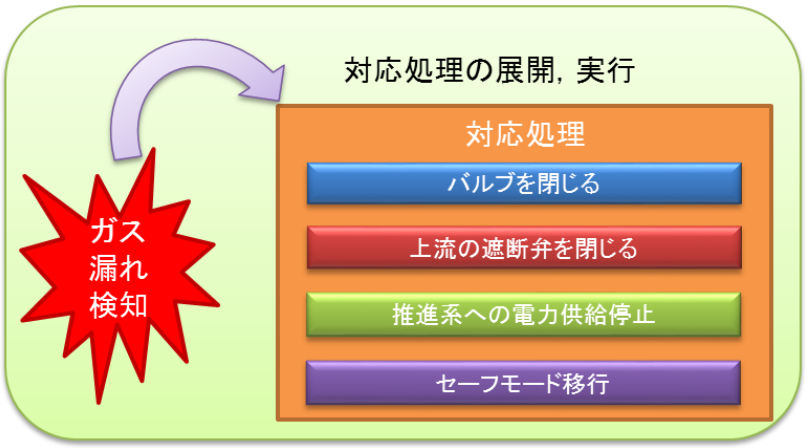

図 1.3 PROCYON における If-then ルールベース自律機能イメージ[19]

1.2.2 モデルベースの自律機能

If-then ルールベースではない自律化としては，自動計画アルゴリズムを用いたモデルベースの自律機能を挙げることができる．その一つの例が，NASA の深宇宙探査機 Deep Space 1 [20] に搭載された Remote Agent [21]である．Remote Agent の実験では，目標遂行のコマンド，閉ループでの計画遂行，宇宙機の状態推定・故障検知，閉ループでのモデルベース故障診断・修復，修復不能な故障に際しての再プランニング，システムレベルでの故障保護などの機能が実証された．具体的には，推進系のバルブシステムに関して，模擬故障の検知・修復を行った．

他にも自動計画アルゴリズムを用いたモデルベースの自律機能に関する研究はあり，DESTINY ミッションで福島らが提案していた知能化技術[22]や NASA-JPL の火星ローバーミッションで提案されている Resilient Spacecraft Executive (RSE)[23]，NASA の地球観測衛星 EO-1 において実証された CASPER[24]など，NASA を中心として様々な自律計画アルゴリズムを利用したモデルベースの自律機能の提案及び実証がなされている[25][26][27].

1.2.3　宇宙機以外の自律機能の例

宇宙機以外の自律機能の例として，ディジタルゲームにおける意思決定アルゴリズムの全体像を，文献[28]を参考にして，宇宙機とディジタルゲームを比較しながら簡単に説明する．ここでディジタルゲームの例をあげている理由は，宇宙機のシミュレータの仕組みがディジタルゲームとよく似ているため，ディジタルゲームにおけるキャラクタの自律機能は宇宙機の自律機能に対する良い参考となると考えたためである．キャラクタの内部と外部を区別し，情報のモデル化をしたもの（エージェントアーキテクチャ）を図 1.4 に示す．

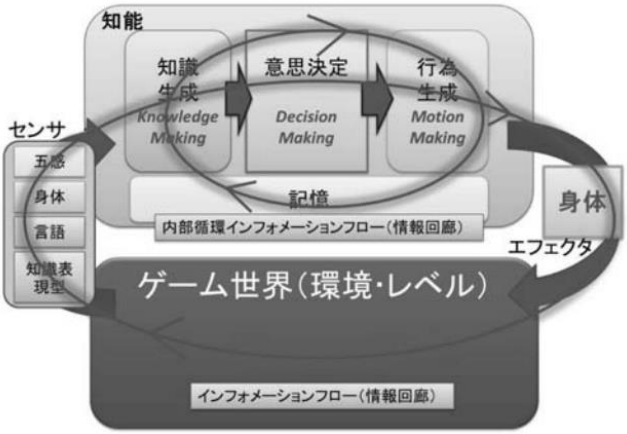

図 1.4 エージェントアーキテクチャとインフォメーションフロー[28]

これと比較する対象として，SILS(Software-in-the-Loop-Simulator)モデルを抽象化したブロック図を示す．

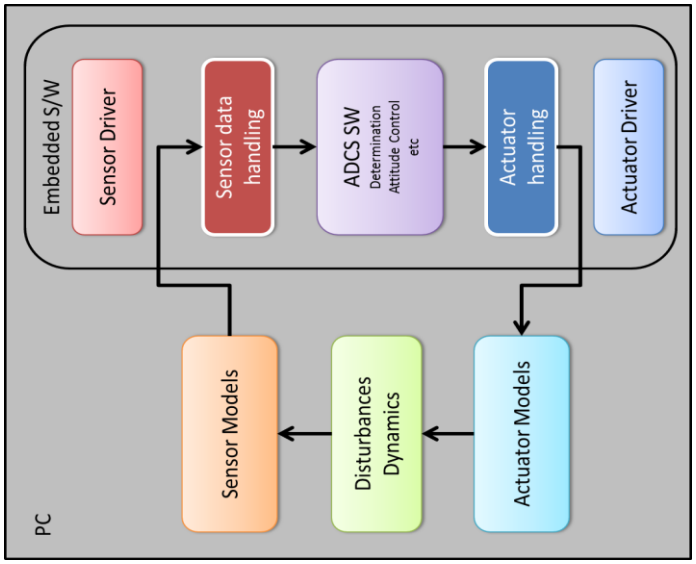

図 1.5 SILS モデル[29]

これらを比較すると，エージェントアーキテクチャと SILS は，ほぼ同じ形で記述されていることになる．すなわち，環境部→センサ部→搭載 SW（知能）部→アクチュエータ（エフェクタ）部→環境部，… といったように情報が流れるモデルとなる．このモデルの具体例として，C4 アーキテクチャ[30]等がある．結局のところ，センサやエフェクタはあくまで「環境部とのやり取りのモデル化」でしかないため，AI 分野として重要なのは知能部である．知能は，認識，意思決定，行為生成という主に 3 つの役割を持つ．基本となるのは，自分が現在の環境下でどのような行動をなしえるのかを「認識」すること（Affordance）である．そのうえで，なしえる行動の中から最適な行為を選択するという「意思決定」を行い，決定された行為を状況に合わせてチューニングすること（たとえば，対象の一・高さに応じてモーションを修正する）が「行為生成」である．また，これらの情報伝達において，各 3 モジュールにおける瞬間の状態を記憶に蓄積することもある．認識は，外界の情報と自己の状態を整理し，足りない情報を推論によって補完することにより，意思決定のための情報を整理することが役割である．そのために，記憶を時系列的に積み重ねておき，そのデータをデータマイニングすることで未来の状態を予想する，などのことをする．意思決定に関してはこの後詳細で述べる．行為生成は，意思決定の出力から実際のモーションを生成する過程であり，たとえばトルクを出力するためにスラスタのバルブを開ける，などがあるだろう．

ディジタルゲームの AI と宇宙機の AI で大きく違う点は，前者はゲームシステムや環境自体にも AI 技術を適用することが多い点である．後者においてゲームシステムや環境はある意味「神」に近いようなものである．

キャラクタの意思決定でよく使われる意思決定アルゴリズムは以下の 7 つである．

- ルールベース AI
  - If…then ルールによって記述される AI．ルールを複数持ち，優先度がつけられている場合が多い．

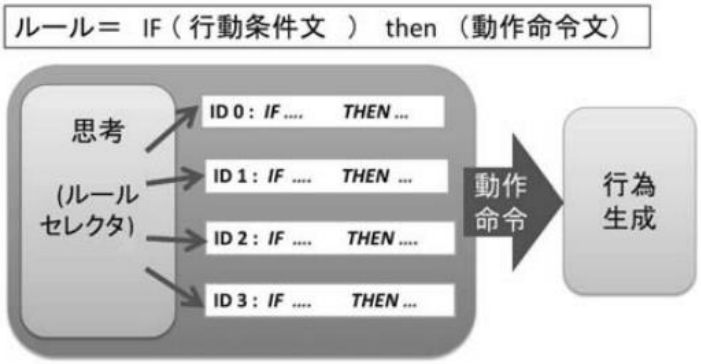

図 1.6 ルールベース AI[28]

- ステートベース AI
  - 状態を基に，状態間の遷移を条件によって決定する AI．キャラクタは状態で定義された行動を行う．複数の状態を遷移状態で結んだシステムを「ステートマシン」という．

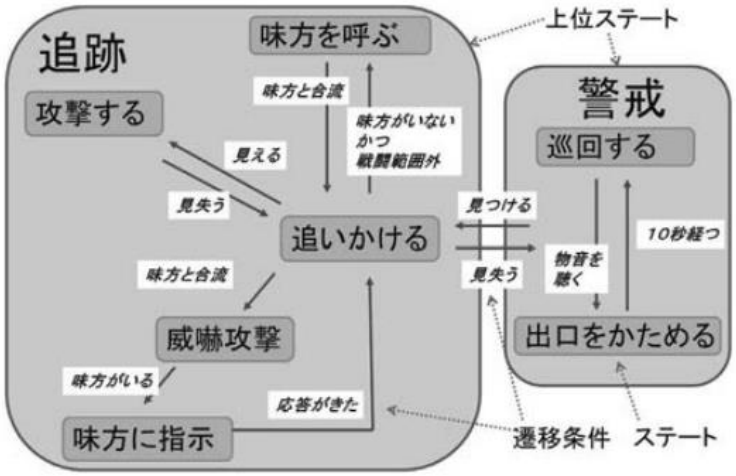

図 1.7 階層型ステートマシン[28]

- ビヘイビアベース AI
  - キャラクタの動作を基本に施行を組み立てること．近年ではビヘイビアツリーという手法が主流になり，ステートマシンに代わってよく使われる手法となっている．

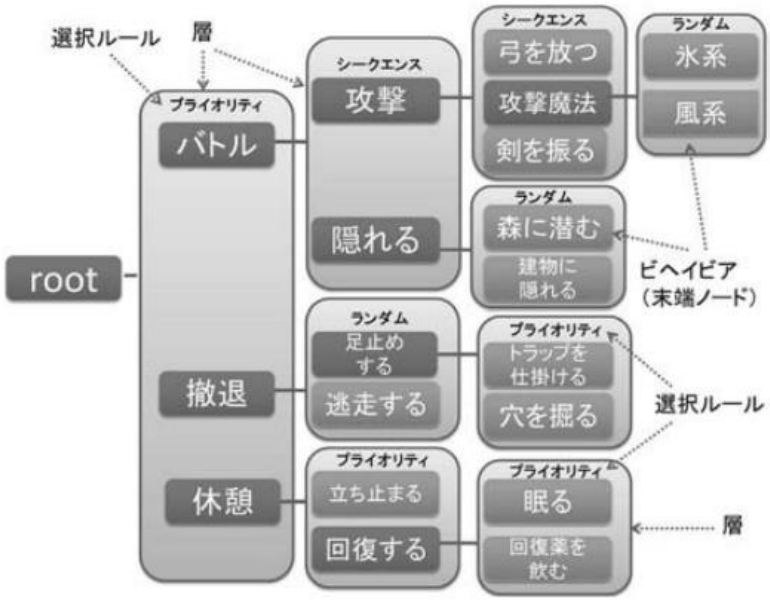

図 1.8 ビヘイビアツリー[28]

- ゴールベース AI
  - 最初にゴールを決定し，それを達成するプロセスを考える，という AI．ゲームでよく使われる手法としては，ゴール指向型アクションプランニング（GOAP）[31]と階層型ゴール指向プランニングがある．前者は，ゴールと初期条件の間をアクションの連鎖で繋いでいく方法で，後者は大きなゴールに対して，スクリプトなどに

よってより小さなゴールへと分解を繰り返し，最終的に単純なコマンドまで分解する方法.

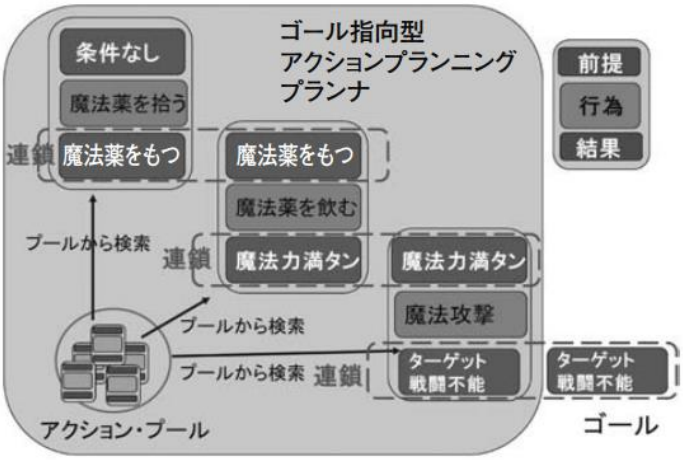

図 1.9 ゴール指向型アクションプランニング（GOAP）[28]

- ユーティリティベース AI
  - それぞれの行動の効用を評価式から評価し，最高点を経た行動を選択する，という AI．効用関数を設定することで，キャラクタに個性を持たせることもできる．
- タスクベース AI
  - キャラクタの行動の単位「タスク」を定めて，その単位によって行動を組み上げる仕組み．タスクは，それ自身の定義とタスク間の結合ルールが定められている．
- シミュレーションベース AI
  - モデルを基に，いくつかのランダムな組み合わせで計算して最良の結果を選ぶ AI．単純なロジックで解けないようなものに使う．

このように，ディジタルゲームでは多様な意思決定アルゴリズムを用いて，キャラクタを動作させている．宇宙機における意思決定アルゴリズムや自律機能と異なる点としては，ディジタルゲームにおいてはすべての情報がディジタルゲーム内の世界に閉じるのに対し，宇宙機ではハードウェアが絡み，実世界との情報のやり取りが発生するとともに，実世界の情報すべてを宇宙機が獲得できることは基本的にない，という違いがある．しかしながら，シミュレーション環境という観点で見ると，宇宙機のシミュレータとディジタルゲームはよく似ているといえよう．

1.3 従来の宇宙機に搭載された自律機能の課題

前節では，宇宙機の自律機能として，大きく分けて「If-then ルールベースの自律機能」と「自動計画アルゴリズムを用いたモデルベースの自律機能」の二つがあることを述べた．本節では，これらの自律機能の課題に関して述べる．

まず, If-then ルールベースの自律機能の特徴をまとめる. If-then ルールベースの自律機能においては，宇宙機の挙動は事前に定めたルールの場合分けによって定まる．そのため，宇宙機の挙動を事前に予測することができ，自律機能の検証が容易であるという利点がある．この「検証が容易である」という点は，非修理系である宇宙機において非常に大きな利点となるため，多くの宇宙機でこの If-then ルールベースの自律機能が利用されている．一方で，事前に定めた 1 対 1 対応のルールの上でしか宇宙機が動けないということは，事前に予測できなかった不測の事態に対応できないという欠点にもつながる．この欠点に対しては，従来の宇宙機では主に次に述べる二つの対策を行う．一つ目の対策は，綿密な検討を行い，可能な限り抜けがなく，適切な対応ができるルールを作り上げることである．この場合，検討時間が増加して設計に必要な時間が増加してしまったり，場合分けの数が増えることでシークエンスが複雑になってしまって検証が難しくなってしまったり，という悪影響がある．二つ目の対策は，「セーフモード」と呼ばれる，宇宙機の安全を確保できる状態を用意して，不測の事態が発生しても人間の運用を待てるようにする，ということである.「セーフモード」から通常の運用状態に戻すためには人間の運用が必要になってしまうため，天体へのフライバイのようなタイムクリティカルなイベント時に「セーフモード」に入ってしまうとミッションの失敗のような取り返しのつかない事態になってしまう可能性がある．

次に，自動計画アルゴリズムを用いたモデルベースの自律機能の特徴をまとめる．モデルベースの自律機能の最大の利点は，未知の状況に対応できる柔軟性があるという点である．Remote Agent の例では，推進系のモデルを持っておくことで，バルブ故障時の対応ルールを事前に定めずとも，バルブ故障への対応を行えることが実証されている．その一方で，自動計画アルゴリズムを用いたモデルベースの自律機能全体の検証が難しいという欠点が挙げられる．特に，自動計画の探索材料となるモデルの構築が属人的で，その検証が難しいことが大きな欠点だと考えている．このモデル検証に関連した研究として，Domain Analysis[32] や Model Checking[33] が挙げられるが，前者は自動計画の効率化に焦点が当たっており，後者は構築したモデルの文法チェックに焦点が当たっている．そのため，モデルと実機の整合性をどう検証するか，という実用上重要な点に焦点が当てられていないのが現状だと考えている．

1.4 研究目的

従来の自律機能の課題として，If-then ルールベースの自律機能に対しては未知の状況に対応できないこと，自動計画アルゴリズムを用いたモデルベースの自律機能に対してはモデル検証の難しさというものがある．宇宙機の運用を自動化するうえで，未知の状況に対するカバレッジの観点は非常に重要だと考えられるため，モデルベースの自律機能を宇宙機に搭載するほうが望ましいと考えられるが，その課題はモデル構築とその検証である．以降，宇宙機に搭載される自動計画アルゴリズムを用いたモデルベースの自律機能を，「宇宙機搭載自律プランナー」と呼ぶ．本研究は，宇宙機搭載自律プランナーを構築するうえでの課題を解決することを目指す．

従来は，設計者が自身の知識をもとに，宇宙機搭載自律プランナーに使用する宇宙機のモデルを構築していた．本来は，その設計者の知識は宇宙機の設計情報に基づいているはずである．そこで，宇宙機の設計情報をデータベース化・モデル化し，その検証を行うことによって「検証された設計情報」というものを作成し，その情報をベースに宇宙機搭載自律プランナーに使用する宇宙機のモデルを構築することで，属人性を排除するともにモデルの検証も行えるのではないかと考えた．

つまり，研究の目的としては以下のようになる．まず，何らかの手法を用いて設計情報をデータベース化・モデル化し，その情報を検証する．そうしてできた「検証された設計情報」から宇宙機搭載自律プランナーの構築に必要な情報を自動的に生成することで，衛星の設計から自律機能の構築までをシームレスに行えるような手法を確立することを目指す．本研究の概要を，図 1.10 に示す．

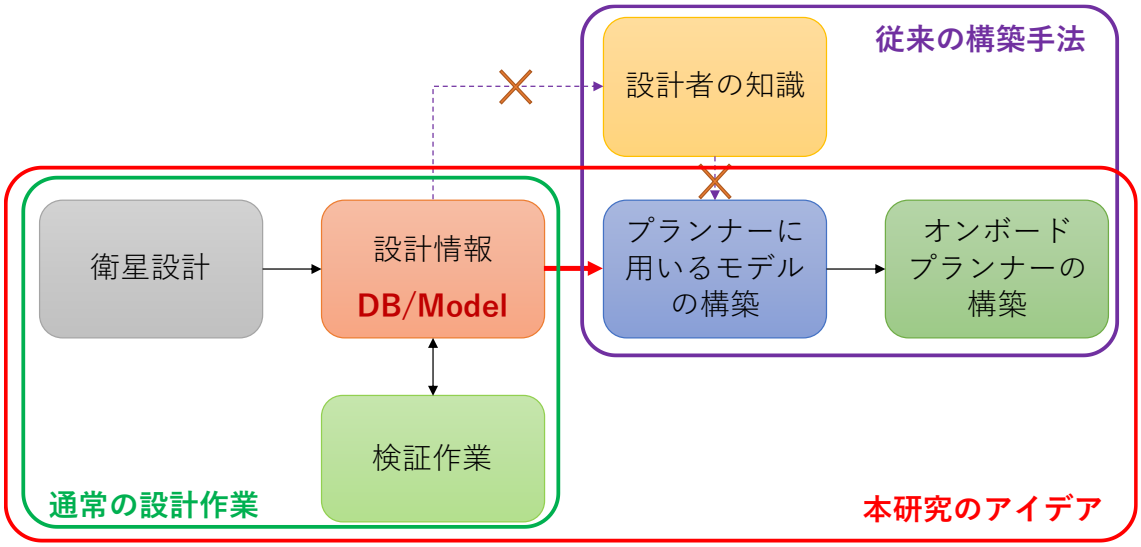

図 1.10 本研究の概要

1.5 論文の構成

本章では，宇宙機において自律機能が必要となってきた背景と，従来の宇宙機で行われてきた自律機能の紹介とその課題の提示を行い，本研究の目的を述べてきた．これ以降では，この目的を達成するための研究内容に関して述べていく．

第 2 章では，本研究において構築する宇宙機搭載自律プランナーに関して整理を行う．まず，宇宙機搭載自律プランナーの目的を整理し，宇宙機搭載自律プランナーに要求される機能とそれを達成するための構成要素，それぞれの構成要素に必要なモデル及び情報をまとめる．第 3 章では，設計情報を可視化しながらシステムの設計を行う「モデルベースシステムズエンジニアリング」の概要を紹介するとともに，宇宙機への適用例を簡単に招待する．また，構築された宇宙機の設計情報と宇宙機搭載自律プランナーがどのような関係になるのかを整理する．第 4 章では，第 3 章までに整理された情報をもとに，宇宙機の設計から宇宙機搭載自律プランナーの構築までをシームレスに行える手法を提案する．第 5 章では，提案した手法を実際の宇宙機に適用することにより，手法の妥当性を検証する．第 5 章では，第 4 章までで述べた内容をまとめるとともに，副次的な効果や今後の発展性を提示する．

第2章 宇宙機搭載自律プランナーの構成要素

本章では，構築すべき宇宙機搭載自律プランナーに関して議論を進める．まず，宇宙機搭載自律プランナーの目的を整理して，プランナーに要求される機能を整理する．次に，要求される機能を達成するための自律プランナーの各構成要素と全体像に関して議論を進め，各構成要素に必要な機能と必要なモデル・情報に関して整理を進める．続いて，各構成要素に必要なモデルと情報から重要な項目を抽出し，宇宙機搭載自律プランナーを構成するために必要な情報が特定のモデル・情報に集約できることを確認する．最後に，集約されたモデル・情報の記述方法を定め，本研究の目的を達成するための宇宙機搭載自律プランナーの構成方法について議論を行ったうえで，実機での実装方針に関して議論を行うこととする．

2.1 宇宙機搭載自律プランナーの目的

本節では宇宙機搭載自律プランナーの目的を整理し，必要な機能を整理する．第 1 章で整理したように，宇宙機搭載自律プランナーは運用の処理を宇宙機上において自動化するものである．1.1.2 項でまとめた通り，宇宙機の運用は大まかに以下の 4 つに大別できる．

- 打ち上げ後の初期の搭載機器動作確認
- 宇宙機の軌道解析・決定
- 異常検知・対処
- ミッション運用

このうち，「初期の搭載機器確認」に必要な期間が運用期間の全体に対して占める割合は低く，この運用項目を自動化するメリットは他の 3 項目を自動化するメリットに比べて少ないと考える．近年では，日本に限っても三菱電機の DS2000[35]や日本電気の NEXTAR[36]など，大型衛星でのコスト削減を目指した標準バスシステムの採用が進んでおり，超小型衛星ではほどよしプロジェクト[37]によって搭載機器のサプライチェーンの構築が進むなど，衛星バスシステムの標準化が進んできている．それに加えて，コンステレーションミッションにおいては同一設計の衛星を大量生産することが多くなると考えられる．このように衛星バスシステムの標準化が進むと，新規の不具合が発生しづらくなるとともに，初期のチェックアウト手順の洗練化・簡略化が進むと考えられる．このような理由から，打ち上げ後の初期の搭載機器動作確認の運用を自動化するメリットは他の 3 項目を自動化するメリットに比べて小さいと考えられる．

「宇宙機の軌道解析・決定」の自動化は，運用で人間が実施する項目を自動化すればよいわけではなく，ダイナミクスなどの検討が必要であり，他の 3 項目と質が異なる．具体的な研究例としては，X 線パルサー航法による自律軌道決定[38]や，衛星間通信を用いた自律軌道決定[39]，光学情報を用いた自律軌道決定[40]などになる．こういった自律軌道決定を行うためには，専用の搭載機器が必要になることが多い．そのため，「宇宙機の軌道解析・決

定」の運用項目は，本研究において宇宙機上での自動化を行う運用項目には含まないこととする．

したがって，本研究においては，「異常検知・対処」「ミッション運用」の二つを自動で行えるような機能を，宇宙機搭載自律プランナーが実現することを目指すことになる．

上記で述べた機能に関して，明示的に示されていない内容について述べていく．異常検知・対処に関して考えると，異常の内容によっては運用目的の達成ができない場合があることが推測できる．例えば，カメラを用いて天体を撮像することが運用目的であった場合，観測に利用するカメラが永久故障してしまうと，運用目的を達成することができなくなる．また，カメラ自体が故障していなくても，カメラに電源を供給する経路である回路や電線に永久故障が発生してしまうと，冗長機能が存在しない限り故障への対処はできない．このように，永久故障が発生した際に，その冗長機能が実装されていなければ，対処によるリカバーを行うことができない．このような事態に対して，本研究における搭載自律プランナーによる対処は期待しないこととする．ただし，搭載自律プランナーを用いた打ち上げ前の試験やシミュレーションによって，このような脆弱性を見つけ，対処を実装できることが期待できる．

他の明示されていない内容として，対処できる時間スケールが挙げられる．図 2.1 に，ロボティクス分野において，Planning と Control における時間スケールを検討した図を示す．Planning と Control は別の周波数を担当しており，Planning は 1000 秒オーダー以上の領域，Control は 1 秒オーダー以下の領域を主に担当する．人工衛星の実運用に当てはめると，Control は姿勢制御やヒーターを用いた温度制御などに相当し，Planning は観測機器による運用計画や，軌道制御の計画などが相当する．このように，Control と Planning では扱う時間スケールが異なり，宇宙機搭載自律プランナーは「運用計画の宇宙機上における自動化」という目的から Planning を担当すると考えられる．したがって，宇宙機搭載自律プランナーによって対応する時間スケールは，少なくとも Control よりも長い時間スケール，本研究では 100 秒オーダーほどを想定すればよいことになる．

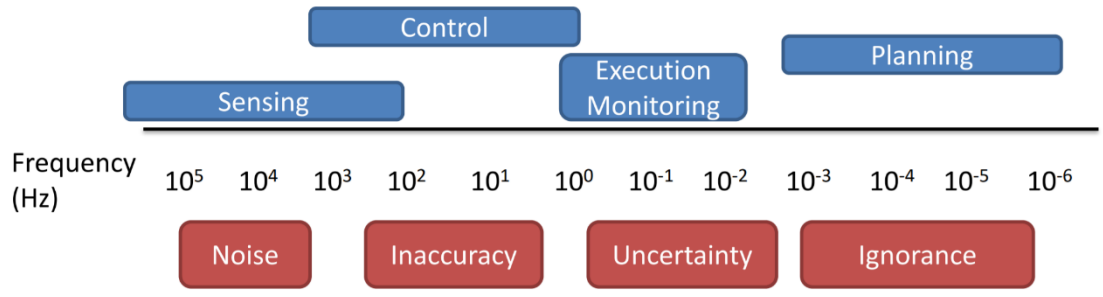

図 2.1 Planning と Control の扱う時間スケール[34]

以上を踏まえて，本研究において宇宙機搭載自律プランナーが満たすべき機能は以下のようになる．

- 【ミッション運用】運用目標が与えられたときに，自動計画アルゴリズムを用いて，運用目標を達成できる方策を自ら探索し，実行できること．
  - 例：「月の写真を撮れ」という目標が与えられたとき，「姿勢制御～観測機器 ON～撮影」のような計画を計算し，実行する．
- 【異常検知・対処】ミッション運用を行う際に，異常が発生した場合であっても，それに対処をしたうえで運用目的の達成を目指すこと．
  - 例：観測機器がフリーズしたときに，「観測機器の OFF→ON」のような計画を計算し，実行する．
- 【対応する時間スケール】100 秒オーダーの自動計画・実行を行う．それより短い時間スケールの事象に対しては，自律プランナー外の制御システムで対応を行う．

2.2 構成要素の抽出

前節でまとめた宇宙機搭載自律プランナーの機能を達成するために，自律プランナーに必要な要素をまとめていく．従来の宇宙システムの知識と，本研究における目的とを加味したうえで，順を追って構成要素の整理を行う．

2.2.1 Planning Engine

まずは，自動計画アルゴリズムが動作するモジュールを Planning Engine と名付ける．本研究における宇宙機搭載自律プランナーは自動計画アルゴリズムを用いてタスクプランニングを実施するため，このモジュールは必須である．Planning Engine が実施する役割を簡潔に述べると以下のとおりである．

- 現在の状態から目標の状態に到達するまでの方策を，自動計画アルゴリズムを用いて探索する

2.2.2 Low Level Controller

本研究では，従来の宇宙機搭載計算機上のソフトウェアが実施していた役割を果たすモジュールを，Low Level Controller と呼ぶことにする．2.1 節で述べていた，Control を担当するモジュールとなる．Low Level Controller の機能は宇宙機の構成次第に依存するため，ここでは宇宙機の例として PROCYON を採用して，具体的な機能を紹介する．Low Level Controller が実施する役割としては主に以下のものが挙げられる．

- 地上局からのコマンドを解釈し，実行する

- 地上局に送信するテレメトリデータのパケットを生成し，送信する
- 宇宙機に搭載された機器を操作する
- 宇宙機に搭載された機器から，データを受け取る
- 姿勢センサの情報をもとに姿勢決定を行う
- 与えられた目標の姿勢状態へ制御する
- 事前に定められたルールに則り，異常検知と対処を実行する

ここまでの二項で，宇宙機搭載自律プランナーの目的を達成するために必須のモジュールと，従来の宇宙機に搭載されていたモジュールに名前を与え，整理した．続いて，ここまで述べてきたモジュールが動作するために必要な機能を整理していく．

2.2.3 Operator

Planning Engine の機能は，「現在の状態から，目標の状態に到達するまでの方策を探索する」というものである．この機能から，Planning Engine の入力として「現在の状態」「目標の状態」があり，出力として「方策」があることが推測できる．このうち，Operator は Planning Engine に「目標の状態」を与えるモジュールになる．Operator という名称は，運用者の英訳から与えている．

本研究における宇宙機搭載自律プランナー全体の目的を考慮すると，Operator の役割はあくまで「Planning Engine に目標の状態を与える」となる．そのため，本研究においては実際の運用者が Operator の役割を果たしても問題ない．一方，ミッション目的を宇宙機自身で設定するような高度な自律機能を構築する場合は，Operator に「ミッション目的を設定する」という機能を与えることになる．

2.2.4 Executor

Planning Engine が導き出した方策を宇宙機が実行するためには，方策を解釈して，各機器を操作する必要がある．方策を解釈する機能は従来の宇宙機搭載ソフトウェアである Low Level Controller には存在しないため，宇宙機搭載自律プランナーはこの機能を持つモジュールを新たに持つ必要がある．これを，Executor と名付けている．

従来の宇宙機の運用においては，運用者はコマンドを Low Level Controller に解釈させ，搭載機器の操作を含めた各種運用を実施する．すなわち，Low Level Controller の入力はコマンドである．したがって，Executor が果たすべき役割を端的に述べると以下のようになる．

- Planning Engine が導き出した方策をコマンド列に変換し，Low Level Controller に入力する

2.2.5 State Monitor

Planning Engine が導き出した方策が正しく実行されているかを確認するためには，宇宙機搭載機器の状態をもとに方策で到達すべき状態になっているかを判断する必要がある．Low Level Controller が保持している宇宙機状態と，Planning Engine が期待している宇宙機状態では，構成する情報に差がある可能性がある．従来の運用では，運用者は一つもしくは複数のテレメトリデータをもとに，宇宙機の状態を確認して，運用を進めていく．したがって，State Monitor では，Low Level Controller から出力されるテレメトリデータをもとに，現在の状態が運用計画の通りであるかを確認する必要がある．したがって，以下のような機能が必要となる．

- Planning Engine が出力した方策をシミュレーションし，宇宙機の状態がどのように遷移していくかを計算する
- テレメトリデータをもとに，現在の状態に問題がないかを確認する

2.2.6 全体像の整理

ここまで述べてきた構成要素をまとめると，図 2.2 のようになる．2.1 節でまとめた目的を達成するうえで必要なモジュールは 2.2.1 項から 2.2.5 項で述べた合計 5 個となるが，2.2.2 項で述べたように Low Level Controller は従来の搭載ソフトウェアが果たしていた役割を担うモジュールであり，Operator は人間の運用者であってもよい．そのため，Planning Engine と State Monitor，Executor の 3 つのモジュールをまとめてオンボードプランナーと呼ぶこととする．

図 2.2 に示される全体像において，宇宙機搭載自律プランナーがどのように動作をするかを簡単に説明する．まず，Operator（運用者）が運用目的を定め，Planning Engine にその運用目的を与える．Planning Engine は，初期条件（現在の状態）から与えられた運用目的を達成するための方策を，自動計画アルゴリズムを用いて探索する．探索した結果は，Executor によって Low Level Controller が解釈できるコマンド列へと変換され，Low Level Controller に渡されて実行される．Low Level Controller はコマンド列を実行しつつ，定期的にテレメトリデータを State Monitor に送信する．State Monitor は，初期条件と Planning Engine が導き出した方策をもとに，方策が正しく実行された場合に，宇宙機の状態がどう遷移していくかをシミュレーションする．このシミュレーション結果と，Low Level Controller から送られてきたテレメトリデータを照合し，異常が発生していないかを確認する．もし異常が発生していた場合，異常が発生している現在の状態から本来の運用目的を

達成するための方策を探索するために，初期条件として現在の（異常が発生している）状態，目標として当初の運用目的を Planning Engine に与えて方策の再探索を実施する．この探索によって導かれた方策は，現在発生している異常に対処をしたうえで運用目的を達成できる方策であるため，再度 Executor を通して Low Level Controller に方策のコマンド列を送り，方策を実行する．このような動作を，運用目的が達成されるまで実施することになる．

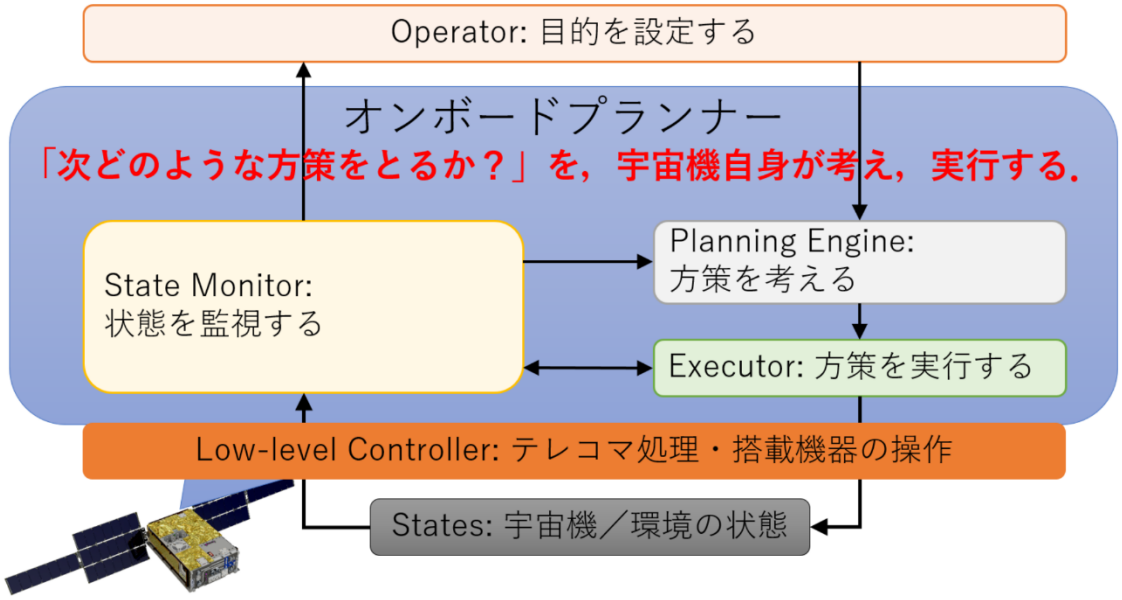

図 2.2 宇宙機搭載自律プランナーの概略

2.2.7 従来の宇宙機搭載自律プランナーの構成との比較

前項まででは，本研究における宇宙機搭載自律プランナーの目的をもとに，必要なモジュールとその機能を導き出した．本項では，導き出された宇宙機搭載自律プランナーの構成を従来の宇宙機搭載自律プランナーの構成と比較し，図 2.2 の構成の妥当性に関して検証を行う．

従来の宇宙機搭載自律プランナーである，Remote Agent との比較を行う．Deep Space 1 における，Remote Agent を含めた搭載ソフトウェアシステム全体のブロック図を図 2.3 に示す．また，Remote Agent 内における Planner / Scheduler の部分を抜粋した部分を図 2.4 に示す．まず，本研究における Planning Engine に当たる部分は Remote Agent における Planner / Scheduler に該当すると考えられ，本研究における Low Level Controller は Real-Time Control にあたると考えられる．これらに関しては，自動計画のアルゴリズムが動作すること，ハードウェアの制御や地上局との通信のために必須の機能があることの二点から，同様の機能を持つモジュールが存在することには納得ができる．また，本研究における

Operator に当たる部分は，Remote Agent では Mission Manager に当たる．本研究では Operator をモジュールとして定義はしているものの，実際には人間の運用者が担当するとしているのに対し，Remote Agent では，その部分も宇宙機に搭載している，という違いがあるものの，モジュールとしては共通して存在している．続いて，本研究における Executor の機能は，Remote Agent では Smart Executive として実現されている．両者の違いとして，前者は単に方策をコマンド列に変換しているだけなのに対し，後者は方策の実行状況のモニタなどの機能が実現されているという違いがある．この点から見ても，本研究における State Monitor の機能は，Remote Agent では Monitors, Mode Identification and Reconfiguration, Smart Executive のそれぞれに分けて実装されている．Remote Agent は，Livingstone[41] を利用したモデルベースの状態推定・再構成といった機能も持っている．本研究では，宇宙機の動作モデルを設計情報から生成して，それに沿った状態推定や方策の探索を行うことで，宇宙機の動作モデルを宇宙機上で組み替えるような高度な柔軟性は確保できないものの，堅実な動作が期待できる自律プランナーを構成することとしているため，Livingstone のような高度な状態推定機は実装していない．以上より，本研究で構成する宇宙機搭載自律プランナーと Remote Agent を比較した場合，状態推定に関する高度な機能以外の機能は共通しているといえ，目的に合った必要十分な機能を抽出できていると考えられる．

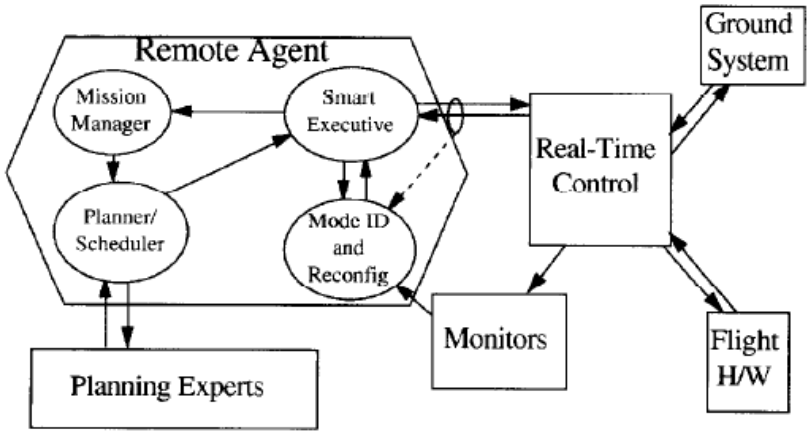

図 2.3 Remote Agent architecture embedded within flight software[21]

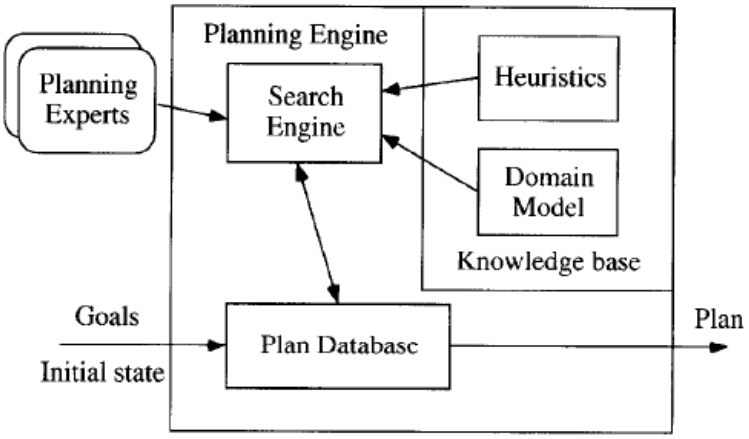

図 2.4 Planning and Scheduling architecture diagram[21]

2.3 宇宙機搭載自律プランナー構成に必要なモデルと情報

前節までで，宇宙機搭載自律プランナーを構成する要素を整理した．本節では，整理された各要素を実装するうえで必要な情報や，利用するモデルとその記述方法に関して議論を行う．

2.3.1 各構成要素で必要なモデルと情報

まず，整理された各構成要素を実装するうえで必要な情報とモデルをまとめていく．

2.3.1.1. Planning Engine

Planning Engine は，自動計画のアルゴリズムが実行されるモジュールである．そのため，利用するアルゴリズムによって必要なモデルや情報が変わるように思える．しかし，本モジュールはあくまで自動計画問題を解くものであるため，必要なモデルや情報を一般化すると，自動計画問題を記述するための内容と一致することがわかる．したがって，Planning Engine に必要なモデル・情報としては,「宇宙機の挙動を記述するドメインモデル」となる．このドメインモデルの具体的な記述内容は使用する自動計画アルゴリズムによって異なるが，基本的には宇宙機が行う行為である action の集合と宇宙機がとりうる状態 state の集合，それらの付随情報となる．

2.3.1.2. Low Level Controller

Low Level Controller は，搭載機器（ハードウェア）を操作する機能や，地上局との間の通信を行うためのデータ処理機能，姿勢・温度制御などといった機能を持つ，従来の搭載ソ

フトウェアに当たる部分である．この中で宇宙機搭載自律プランナーに関係する部分のみを考慮すると，宇宙機に搭載されているハードウェアを操作するために必要な機能が主となることがわかる．すなわち，地上局からコマンドを受信し，そのコマンドデータに応じて搭載機器を操作する機能と，搭載機器の動作状況を示すデータを搭載機器から受信し，テレメトリデータとして地上局に送信する機能である．また，地上局と通信できていない間であっても，宇宙機は自身が生存するための動作や，ミッションを達成するための動作を行わなければならない．そのタイミングを制御するための時刻情報も必要となると考えられる．

2.3.1.3. Operator

Operator は，端的に述べると運用目的を Planning Engine へ伝える機能である．したがって，抽象的な運用目的を，Planning Engine が解釈できる形式に置き換え，Planning Engine に渡す必要がある．したがって，Operator は運用目的を，Planning Engine のドメインモデルにおける state の集合として定義し, Planning Engine に渡してやる必要がある．

2.3.1.4. Executor

Executor は，Planning Engine が導き出した方策を解釈し，Low Level Controller が解釈できるようなコマンド列に変換する．Planning Engine が導き出す方策は，Planning Engine のドメインモデルにおける action が時系列に並んだものである．したがって，Executor が持つべき情報は，ドメインモデルにおけるにおける時刻と Low Level controller における時刻との関係性と，ドメインモデルにおける action と Low Level Controller におけるコマンドの関係性の二つとなる．

2.3.1.5. State Monitor

State Monitor は，Low Level Controller から出力されるテレメトリデータをもとに，現在の状態が運用計画の通りであるかを確認するモジュールである．この機能を実現するためには，Planning Engine が導き出した方策によって，宇宙機の状態がどのように変わっていくかという情報が必要となる．この情報を保持するためには，方策をシミュレーションする機能が必要となるが，このシミュレーションを行うためには，方策内の各 action によって，state がどのように変化していくかという情報が必要となる．これは，宇宙機の挙動を記述したドメインモデルそのものである．したがって，State Monitor にはドメインモデルの情報が必要となる．また，宇宙機の状態は，Low Level Controller においてはテレメトリデータとしてあらわされるが，Planning Engine ではドメインモデルにおける state としてあらわされる．そのため，State Monitor では，テレメトリデータとドメインモデルにおける state との関係性を情報として保持しておく必要がある．

2.3.1.6. まとめ

ここまで述べてきた，宇宙機搭載自律プランナーの構成要素ごとに必要なモデルと情報を箇条書きで整理すると以下のようになる．

- Planning Engine
  - 宇宙機の挙動を記述するドメインモデル
    - action と state の集合
    - 上記の付随情報
- Low Level Controller
  - 搭載機器を操作するためのコマンド
  - 搭載機器の状態を確認するためのテレメトリ
  - 時刻情報
- Operator
  - Planning Engine に与える state の集合（Goal）
- State Monitoring
  - 宇宙機の挙動を記述するドメインモデル
  - Low Level Controller におけるテレメトリデータと，Planning Engine のドメインにおける state の関係
- Executor
  - Low Level Controller におけるコマンドと，Planning Engine のドメインにおける action の関係
  - Planning Engine から出力された方策における時間と，Low Level Controller における時間の関係

整理された内容から，宇宙機搭載自律プランナーを構成するために必要な情報は，Planning Engine におけるドメインモデルと，Low Level Controller を構成するコマンド・テレメトリ・時刻情報，それらの関係性という三項目に集約されることがわかる．以後，ドメインモデルの記述内容と Low Level Controller に関する情報のそれぞれに関して具体的な議論を進めることで，宇宙機搭載自律プランナーを実装するために必要な情報を明確にしていく．

2.3.2 ドメインモデルの記述方法

ドメインモデルは主に action と state から構成されるが，具体的な内容は解きたい自動計画の問題の複雑度によって異なる．また，ドメインモデルがどのように記述されるかは，自動計画問題を解くアルゴリズムによって異なる．そのため，まずは本研究において解こうと考えている自動計画問題とその複雑度に関して整理を行い，ドメインモデルに含むべき内容に関して議論を行う．続いて，自動計画問題を解くアルゴリズムが扱えるようなドメインモデルの記述方法に関して検討を行い，本研究においてドメインモデルをどのように記述すればよいかを整理していく．

2.3.2.1. 自動計画問題の複雑度

自動計画問題の形式として，多くの仮定を置くことで解きやすい形としているものが，古典的プランニング問題である．古典的プランニング問題では考慮されていないが，実問題では扱うべき項目と，宇宙機におけるプランニング問題で考慮すべき項目を，Ghallab らが比較している[47]．文献[47] (p.74) において示されている，古典的プランニングでは考慮されていないが実問題（Operational Models）では考慮すべき項目は以下の通り．

- Dynamic Environment.: 環境は，静的(static)である必要はない．つまり，実問題では，Actor の動作によらない，外因性のイベントも扱う．
- Imperfect information.: 実際には，actor が全部の状態変数に関して現在の値を知っていることや，外部世界が変わる中でこの情報を保てることはまれである．実問題では，actor が何を知っている・知らないかや，どのように必要な情報を獲得するかに関して扱う必要がある．
- Overlapping actions.: 動作を完了するには時間がかかるし，複数の動作が並行して行われることもある．動作が Overlap することを扱うため，複数の動作が並行して行われることを含める．
- Nondeterminism.: 外因性のイベントによる干渉等により，動作後の出力が複数の可能性を持つことがある．Actor は動作によって実際に起きた出力をシステマチックに観測する必要がある．実問題では，このための「観測」を扱う必要がある．
- Hierarchy.: actor は階層的に構成されていることがよくあるため，実問題では階層的な動作の思考を体系づけたり代表したりする方法を考慮する必要がある．
- Discrete and Continuous Variables.: Actor は離散量と連続量の両方を扱う必要に迫られる場合がある．

ここまで述べてきた 6 つの項目に関して，宇宙機のプランニング問題では考慮すべきか

否かを議論する. Dynamic Environment は, 宇宙機の動作に依らない外部イベントであり, 放射線によるシングルイベントが起きた場合など, コマンドで意図したこととは別の事象が発生することにあたる. これは宇宙機では頻繁に発生する可能性がある事象であるだけでなく, そういった事象への対処を行うことも本研究で構築する宇宙機搭載自律プランナーの目的の一つであることから, Dynamic Environment を宇宙機搭載自律プランナーで扱う必要がある. Planning Engine で Dynamic Environment を扱うのは, 方策を立てる段階で外因性のイベントが発生することを考慮したい場合である. 今回の宇宙機搭載自律プランナーにおいては, 不測の事態が発生していることを検知したうえで, その事象への対処を目指せればよく, 対象とする時間スケールも 100 秒以上のオーダーである. したがって, Planning Engine で考慮をするのではなく, State Monitor で扱えばよいと考えている.

次に, Imperfect Information に関して検討をする. 宇宙機が状態変数のすべてを把握しているわけではない, ということがこの項目であるが, 人間の運用者が実施する運用において, 運用者はテレメトリデータをもとに宇宙機の状態を推定したうえで, 目的を達成できるような運用手順を作成する. 場合によっては, 電波の受信強度やドップラーなど, テレメトリデータに原理的に含むことができないものも含め, テレメトリデータに含まれていない様々な情報を利用して運用を行うこともある. しかし, 人間がテレメトリデータをもとに運用を行うという前提に立つと, 運用に必要な情報はテレメトリデータに含まれていなければならないということになる. したがって, 本研究の宇宙機搭載自律プランナーにおいては, Imperfect Information を扱わないこととする.

続いて, Overlapping actions に関して議論を行う. Overlapping actions では, 各 action が完了するまでに時間を要するという効果と, 複数の action を並列に実行できるという拡張が行われる. PROCYON での運用時の経験から, 実際の宇宙機の運用では運用時間を効率的に使用するために, 複数の運用を同時並行で行う場合があることがわかっている. 例えば, 特定の星を観測するというミッションを行うという目的の運用時では, まず姿勢変更を開始し, 姿勢制御が収束するまでの間に観測機器のパラメータを設定したり, 非可視中のデータを保持しているデータレコーダーのダウンリンクをしたりなど, 可能な運用を並行して実施した. こういった, 実際の宇宙機運用で行うような運用時間の効率化を実現するために, 本研究の宇宙機搭載自律プランナーでは Overlapping actions を扱うことにする. 特に, 運用計画を立案する部分である Planning Engine でこの項目を扱うことが必要であると考えられる.

Nondeterminism は, 動作の結果が複数の状態を持ちうるということである. この項目は, 不確定性が多いシステムに対しては有効だと考えられるが, 宇宙機においては一つのコマンドに対して期待される結果は一つに定まることがほとんどであり, 期待される結果が得られないのは機器故障等の不具合による不測の事態である場合が多い. 不具合等による action の失敗は, Dynamic Environment として扱われることを踏まえると, 本研究の宇宙機搭載自律プランナーでは Nondeterminism を考慮する必要性は薄いといえる.

Hierarchy を考慮することで，action を階層化することにより，探索時間の短縮等の効果が得られる．本研究のように宇宙機に自動計画アルゴリズムを搭載する場合，計算機の能力が限られているため，階層化によって計算量を減らすことは特に重要であると考えられる．しかし，action の数は多くても宇宙機に用意されるコマンドの数と同程度になること，階層化によって複数の action を固めると，不測の事態へ対応するまでの時間が長くなるとともに柔軟性が薄れることの二点から，本研究の宇宙機搭載自律プランナーでは階層化を実施しないことにする．代わりに，action を Low Level Controller における複数のコマンドの塊とすることを許可することで，簡易的な階層化を実現することとする．

最後に，Discrete and Continuous Variables に関して検討をする．古典的プランニング問題ではすべての状態変数が離散量であったが，実際の宇宙機においては連続量の状態変数は多数存在する．特に，バッテリー残量や推進剤の残量，電力収支の状況など，運用計画を立てる時に考慮をすべき連続量は多数存在することが考えらえる．したがって，本研究における宇宙機搭載自律プランナーの中でも，Planning Engine において本項目を考慮することが必要であると考えられる．

以上で述べた議論から，古典的プランニング問題においては仮定を置くことで考慮をしていない項目のうち，どの項目を宇宙機搭載自律プランナーで考慮すべきかを表 2.1 にまとめた．Planning Engine では Overlapping actions と Discrete and Continuous Variables を考慮する必要があり，自律プランナーの他のモジュールでは Dynamic Environments を考慮する必要がある．

表 2.1 古典的プランニング問題で仮定が入っている項目と宇宙機のプランニング問題で考慮すべき項目の比較

項目	概要	自律プランナーでの必要性
Dynamic Environment	宇宙機の動作によらない，外因性のイベント	State Monitor で対応
Imperfect Information	宇宙機が状態変数の情報をすべて保持しているわけではない	State Monitor で一部対応
Overlapping actions	動作にかかる時間，動作の並列性	**Planning Engine で考慮**
Nondeterminism	動作後の出力が複数の状態を持つ	×
Hierarchy	動作の階層化	Executor と Low Level Controller で一部対応
Discrete and Continuous Variables	離散量と連続量の状態変数	**Planning Engine で考慮**

2.3.2.2. プランニング問題の記述方法

表 2.1 にまとめている通り，Planning Engine では Overlapping actions と Discrete and Continuous Variables の二点を扱う必要がある．その一方で，使用する Planning Engine のアルゴリズムに応じて，扱える問題の複雑度は異なっている．プランニング問題をどのように記述するかを決定するためには，本研究で利用する Planning Engine を決定したうえでその Planning Engine に合わせた記述方法を採用するか，汎用的に利用できるプランニング問題の記述方法を採用したうえでその方法で記述されたプランニング問題を解ける Planning Engine を採用するかのいずれかを実施する必要がある．プランニング問題の記述方法として汎用的に利用できるものを検討したところ，Planning Domain Description Language (PDDL)[48] が利用できることがわかった．

PDDL は，Drew McDermott らによって開発されたプランニング問題記述言語であり，自動計画問題におけるドメインモデルの記述と，解くべき問題の記述の二つに大別される．PDDL に対応した Planning Engine では，これら二つの PDDL ファイルを入力することで，入力したドメインにおける，入力した問題の解である方策を出力する．PDDL は，より実用的な問題を扱うために徐々に拡張が進められており，PDDL のバージョンによって扱える問題の複雑度が異なる．表 2.1 にまとめた項目について，PDDL の各バージョンでどの項目を扱えるかを比較した結果を表 2.2 に示す．比較した結果より，本研究において構築を行う宇宙機搭載自律プランナーの Planning Engine は，PDDL 2.1 以上を扱うことができるものを採用すればよいことがわかる．

表 2.2 PDDL の各バージョンで扱える項目の比較

項目	PEでの必要性	PDDL 1.2	PDDL 2.1	PDDL +	PDDL 2.2	PDDL 3.0	PDDL 3.1
Dynamic Environment	×	×	×	○	○	○	○
Imperfect Information	×	×	×	×	×	×	×
Overlapping actions	○	×	○	○	○	○	○
Nondeterminism	×	×	×	×	×	×	×
Hierarchy	×	×	×	×	△	△	△
Discrete and Continuous Variables	○	×	○	○	○	○	○

2.3.3 Low Level Controller を構成するための情報

前項までは，宇宙機搭載自律プランナーを構成するために特に重要な二つの要素のうちの一つであるドメインモデルの記述方法に関して議論を行った．本項では，もう一つの重要な要素である，Low Level Controller を構成するのに必要な要素に関して議論を行う．特に，Low Level Controller は従来の，自律プランナーが実装されないときの搭載ソフトウェアに当たる部分であり，搭載している機器（ハードウェア）と強い結びつきがあるため，宇宙機によって大きく実装内容が異なると考えらえる．そのため，規格がすでに存在している項目については，それを採用することを前提とする．

2.3.1 項でまとめた通り，宇宙機搭載自律プランナーを構成するために Low Level Controller に必要な情報は，コマンド・テレメトリと，時刻情報の二つである．前者に関しては，CCSDS (The Consultative Committee for Space Data Systems）で規定されている規格に則ることでパケットの構成は定まる．後者に関しては，搭載計算機の内部でカウントアップする時刻を使うのが一般的である．場合によっては，UTC 等の時刻情報を利用することもあるが，UTC を使う場合でも，実際には搭載計算機の内部処理でカウントアップした時刻に変換することが多いため，時刻の規格としては OBC 内部でカウントアップした時刻を用いればよい．

2.4 各機能の実装方針

ここまで，宇宙機搭載自律プランナーの各モジュールの機能の詳細と，その構成に必要な情報に関して議論を進めてきた．本節では，宇宙機搭載自律プランナーを実装するにあたっての方針に関して議論を行う．

まず，自動計画アルゴリズムが動作する Planning Engine に関しては，PDDL 2.1 以上のバージョンに対応している既存のもの，特にオープンソース化されている自動計画アルゴリズムを利用することが望ましいと考えられる．オープンソース化されている自動計画アルゴリズムは，開発者以外の多数のユーザーが利用することで，不具合の洗い出しや改善などが行われる．その結果，より洗練されて検証されたアルゴリズム・ソースコードとなっていることが期待される．

続いて，Low Level Controller に関して議論を行う．宇宙機搭載自律プランナーを構築するうえで Low Level Controller を実装する際に，新たな機能や情報を要求されているわけではなく，宇宙機で一般的に必要な時刻管理の機能やテレメトリ・コマンドの情報が定まっていればよい．したがって，実装するうえで制約条件を定めることはないが，本研究においては Command Centric Architecture (C2A) [43][44]を利用することを推奨する．C2A はコマンドを中心に宇宙機の搭載ソフトウェアを構成するアーキテクチャである．C2A を本研究

において採用するメリットとしては，テレメトリ・コマンドに関して搭載ソフトウェアのソースコードと地上局のデータベースとの間での齟齬が発生しないような共通のデータベースを構築できる仕組みがあること[45][46]や，再構成能力に優れていること，コマンドによって宇宙機の機能が構成されているため，Executor とのインターフェースが定めやすいことなどが挙げられる．

最後に，残りの 3 モジュールに関して議論を行う．まず，Operator に関しては，人間である運用者が担当してもよいとしている．Executor や State Monitor に関しては，一般的な宇宙機のテレコマ情報と PDDL の変換を行う機能となる．したがって，一度 State Monitor と Executor のテンプレートを構築してしまえば，その後も使いまわせると考えられる．特に，Low Level Controller に C2A を利用することにより，Executor や State Monitor を構築するために必要なデータベースをシステマティックに記述できるようになる．図 2.5 に Executor を構成するデータベースの例を，図 2.6 に State Monitor を構成するデータベースの例を示す．図 2.6 に示している State Monitor を構成するデータベースにおいては，C2A 上のテレメトリ情報からドメインモデルの state に変換するための変換則を新たに記述してやる必要があるが，それ以外の情報はドメインモデルもしくは C2A のテレメトリデータベースに記述されているものとなる．図 2.5 に示している Executor を構成するデータベースの例は，ドメインモデルの情報と C2A のコマンドデータベースから構成されている．このように，Low Level Controller に C2A を用いることによって Executor や State Monitor を構成するデータベースをシステマティックに記述でき，データベースを読み込む State Monitor や Executor は再利用できる形で実装することが可能になる．

図 2.5 Executor を構成するデータベースの例

図 2.6 State Monitor を構成するデータベースの例

2.5 本章のまとめ

本章では，人間によって行われる運用の一部を宇宙機側で自動化するというモチベーションをもとに，宇宙機搭載自律プランナーの目的を「運用目標が与えられたときに，自動計画アルゴリズムを用いて，運用目標を達成できる方策を自ら探索し，実行できること」と「ミッション運用を行う際に，異常が発生した場合であっても，それに対処をしたうえで運用目的の達成を目指すこと」の二つと定義した．続いて，この目的を達成するための宇宙機搭載自律プランナーの構成要素として，「Planning Engine」「Low Level Controller」「Operator」「State Monitoring」「Executor」の五つに整理した．これらの構成要素を実装するのに必要な情報について議論を進めていくと，「ドメインモデルの記述内容」と「Low Level Controller に関する情報」に集約されることを述べた．前者に関して深く議論を進めることで，宇宙機の自動計画問題を記述するためには PDDL 2.1 以降のようなプランニング記述言語を用いればよいことを示した．さらに，搭載ソフトウェア・アーキテクチャの一つである C2A を Low Level Controller に用いることによって，テレメトリ・コマンドのデータベースの記述方法を示すだけでなく，テレメトリ・コマンドとドメインモデルの関係性をシステマティックに記述できることを示した．

第3章 設計情報から自律プランナーを構築する手法

本章では，第 2 章で整理した宇宙機搭載自律プランナーをどのように構築するかを議論する．第 1 章では，宇宙機搭載自律プランナーを構築するうえでの課題はモデル構築の属人性と検証の難しさにあると述べた．これを解決するために，まずは第 2 章で整理された宇宙機搭載自律プランナーの各要素に必要な情報・モデルと宇宙機の設計情報の関連性を議論し，設計情報から搭載自律プランナーをシームレスに生成できる可能性を探る，そのうえで，モデルベースシステムズエンジニアリングを利用して宇宙機の設計情報を体系的にモデル化し，抽象的なモデルの段階で検証可能な，宇宙機の設計モデルを構築する．続いて，設計モデルを構築するうえで書き下していく各ダイアグラムの一部が，宇宙機のドメインモデルと深い関連があることを整理し，宇宙機の設計モデルからドメインモデルを自動生成することが可能か議論する．最後に，本章での議論をまとめ，宇宙機の設計モデルからドメインモデルをシームレスに構築する手法を提案する．

3.1 ドメインモデルと設計情報の関連性

本研究で構築を目指す宇宙機搭載自律プランナーは，運用の一部を宇宙機上で自動化することをモチベーションとしている．そのため，宇宙機搭載自律プランナーを構築するうえで重要となるドメインモデルに必要な情報は，従来の宇宙機において人間が実施している運用で利用している情報に包含されるのではないか，と推測できる．そこで，従来の宇宙機において人間が実施している運用と，宇宙機搭載自律プランナーの自動計画アルゴリズムによって行われる処理をそれぞれ整理し，その類似性について議論を行う．

まず，人間が実施している運用に関して整理する．どのような運用項目を・どのような運用手順で実施するかということは各プロジェクトによって大きく異なることが予想される．例えば，大学が開発して運用している超小型衛星と，政府が中心となって進めていると考えられる情報収集衛星，JAXA が開発して運用する大型の深宇宙探査機では，利用できる地上側のインフラや運用者のスキル，行うミッションとその重要度などが大きく異なることが予想される．そこで本節では，運用手順の自動化をほとんど行っておらず筆者が実際に運用に携わった PROCYON における運用手順をもとに，人間が実施する運用の手順を整理することにする．

宇宙機の運用は，宇宙機が地上局アンテナの可視範囲に入り，電波を受信できる状態（Acquisition of Signal, AOS）になってから始まる．最初に，地上局から宇宙機へ電波を送信し始め，周波数のスイープ作業，電波の変調作業等を行うことでコマンドを送信できるようにする．このコマンド送信準備作業と並行して，はじめに取得できたテレメトリデータを確認し，異常が発生していないかを確認する．異常が発生している場合は，異常への対処や，

異常の詳細の特定のために非可視時間のテレメトリデータの取得などといった運用を実施する．異常が発生していないようなときは，各運用日ごとに設定された運用項目を順番に実施していく．例えば，ミッション望遠鏡で地球を撮影するというミッションがあるときには，「撮像姿勢への姿勢マヌーバ」「望遠鏡の設定」「画像撮影」「画像ダウンリンク」などといった運用項目を順番にこなしていく．それぞれの運用項目は多数のコマンドで成立しており，姿勢マヌーバは「AOS 時の姿勢に戻るタイムライン（時刻指定）コマンドを仕込んでおく」「AOS 時のアンテナ設定に戻るタイムライン（時刻指定）コマンドを仕込んでおく」「AOS 時の姿勢から，撮像姿勢に移行するコマンド群を送信し，姿勢マヌーバを行う」「撮像姿勢時に最適なアンテナ設定に変更する」といった内容を，一つ一つコマンドに落とし込んでいる．そして，一つ一つのコマンドを送信するごとに，コマンドの実行結果が正しいかをテレメトリデータで確認していく．PROCYON の場合は深宇宙を航行していたため，伝搬遅延時間が大きく，即時にコマンドの実行結果を確認することができなかったが，逐次テレメトリを見てコマンドの実行結果を確認しながら運用を実施していた．図 3.1 に，実際の運用手順書（コマンドファイル）の一部を抜粋している．それぞれのコマンドに対して確認事項が設けられていることがわかる．

```
0547 . #-----------------------------------------↓
0548 . # DRのパーティション3(汎用パーティション設定)↓
0549 . # 3MBytes弱空ける↓
0550 . #-----------------------------------------↓
0551 . # DRのテレメトリを1秒1フレームで下ろすように設定↓
0552 . OBC_RT.Cmd_BCT_SET_BLOCK_POSITION 161 0↓
0553 . # □cmd_op.wfdでBCのポインタがBLK:161, CMD:0になっていることを確認↓
0554 . #↓
0555 . OBC_BL.Cmd_GENERATE_TLM 1 0x40 0x0c 1↓
0556 . # □cmd_op.wfdでBCのポインタがBLK:161, CMD:1になっていることを確認↓
0557 . # □cmd_op.wfdでBCの登録内容がID:0x0013, TI:1になっていることを確認↓
0558 . # □DR.wfdの表示内容が更新されることを確認↓
0559 . #↓
0560 . OBC_RT.Cmd_DR_SET_PARAMS 2 0x0AFC0000  0x0BD00000 0 #パーティション2を削り，パーティション3を空ける
0561 . # □DR.wfdでパーティション2のBeginが0x0AFC0000，Endが0x0BD00000になることを確認↓
0562 . #↓
0563 . OBC_RT.Cmd_DR_SET_RP 3 0x0BD00000 #RPをパーティション先頭に↓
0564 . # □DR.wfdでパーティション3のRPが0x0BD00000になることを確認↓
0565 . #↓
0566 . OBC_RT.Cmd_DR_SET_WP 3 0x0BD00000 #WPをパーティション先頭に↓
0567 . # □DR.wfdでパーティション3のWPが0x0BD00000になることを確認↓
0568 . #↓
0569 . OBC_RT.Cmd_DR_SET_PARAMS 3 0x0BD00000  0x0BFF0000 0 #新たにパーティション3を作成(約3MBytes)↓
0570 . # □DR.wfdでパーティション3のBeginが0x0BD00000，Endが0x0BFF0000になることを確認↓
0571 . #↓
0572 . OBC_RT.Cmd_DR_SET_WRITE_ENABLE_FLAG 3 1 #パーティション3を書き込み有効に↓
0573 . # □DR.wfdでパーティション3のWRITE_ENABLEがENABLEになることを確認↓
0574 . #↓
0575 . OBC_RT.Cmd_BCT_SET_BLOCK_POSITION 161 0↓
0576 . # □cmd_op.wfdでBCのポインタがBLK:161, CMD:0になっていることを確認↓
0577 . #↓
0578 . OBC_BL.Cmd_NOP 1 ↓
0579 . # □cmd_op.wfdでBCのポインタがBLK:161, CMD:1になっていることを確認↓
0580 . # □cmd_op.wfdでBCの登録内容がID:0x0000, TI:0になっていることを確認↓
```

図 3.1 運用手順の例（PROCYON 2014/12/17 運用手順書より一部編集し，抜粋）

以上より，宇宙機の運用は複数の運用項目からなり，それぞれの運用項目に対して多数のコマンドが設定されていることがわかる．そして，各コマンドに対して，実行結果を確認し，

問題がなければ次の運用項目に進んでいく，という流れになっている．この流れを簡易的にまとめると，人間が実施している運用は以下のように抽象化できる．

- 人間による運用では，宇宙機からダウンリンクされてきたテレメトリデータをもとに，人間が現在の宇宙機の状態を推定し，運用目標を達成するための運用手順としてコマンド列を作成し，宇宙機にアップリンクを行って，宇宙機にコマンド列を実行させる．

次に，宇宙機搭載自律プランナーの自動計画アルゴリズムによって行われる処理を整理する．これは，プランニング問題が与えられた際に，自動計画アルゴリズムがどのような動作をするかということなので，以下のようにまとめることができる．

- 現在の state から目標（Goal）の state に到達するための方策（action 列）を，自動計画のアルゴリズムを用いて探索する．

ここまで，人間によって行われる宇宙機の運用と，宇宙機搭載自律プランナーの自動計画アルゴリズムによって行われる処理を整理した．これを簡単にまとめたものを図 3.2 に示す．この比較によって，人間の運用と宇宙機搭載自律プランナーの自動計画アルゴリズムとの間に相似性があることがわかる．すなわち，人間の運用におけるテレメトリデータは自動計画アルゴリズムにおける state に対応し，人間の運用における抽象的な運用目標は自動計画アルゴリズムにおける Goal に対応し，人間の運用における運用手順のコマンド列は自動計画アルゴリズムにおける方策の action 列に対応する，ということである．

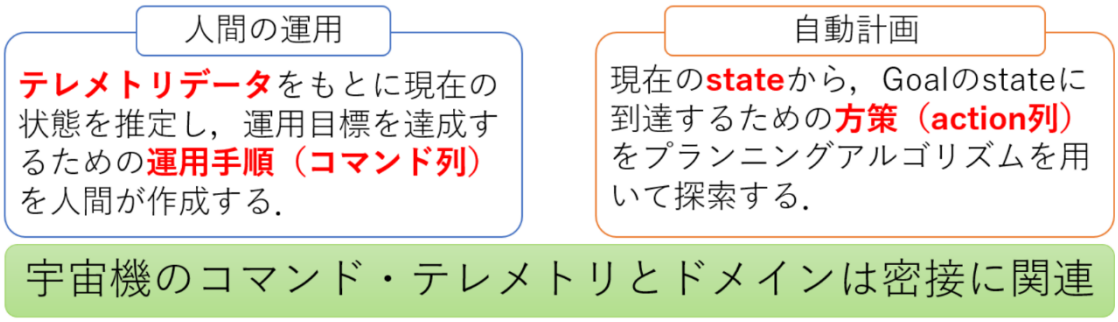

図 3.2 人間の運用と自動計画アルゴリズムの比較

第 1 章で，自動計画アルゴリズムにいて自律機能を実現する場合の課題は，利用するドメインモデルの構築の属人性とその検証方法であると考え，それを解決するために，宇宙機の設計情報から宇宙機搭載自律プランナーをシームレスに構築することを目指すと述べた．宇宙機の設計情報をもとに宇宙機搭載自律プランナーの構築に必要なドメインモデルを構築しようと考えた場合，図 3.2 で示したような相似性が重要となる．つまり，宇宙機の設計情報に含まれるコマンドとテレメトリという要素と，宇宙機搭載自律プランナーに利用するドメインモデルを構成する action と state という要素が密接に関連しているため，設計情報をもとにドメインモデルを構築するためには，以下の二点が重要となる．

① コマンド・テレメトリとドメインの対応関係が正しいこと
② 情報源であるコマンド・テレメトリが過不足なく設計され，実機との対応関係が正しいこと

この関係性を簡単に図 3.3 に示している．①は「Executor や State Monitor を構成するデータベースが正しいこと」と言い換えることができ，2.4 節で述べたデータベースのシステマティックな記述方法を利用することにより，このデータベースを設計情報から自然と生成できれば解決できると考えられる．②については，ステークホルダーからの要求を満たすテレメトリ・コマンドが用意できれば良いと考えられる．まず，ステークホルダーからの要求をトレースしながらコンポーネントレベルの設計要求が整理され，この要求をもとに設計・製造されることで，コンポーネントがステークホルダーからの要求を満たす，十分な機能を持つことになる．この機能を操作するのに必要なテレメトリ・コンドが用意されることにより，ステークホルダーからの要求を満たした機能を操作するのに十分な，過不足ないテレメトリ・コマンドが設計できる．このテレメトリ・コマンドが実際の宇宙機に実装されることで，②が達成できると考えられる．すなわち，ステークホルダーからの要求をコンポーネントの設計要求に正確にトレースし，製造されたコンポーネントが要求を満たしているか検証することが②の達成に重要だと考えられる．

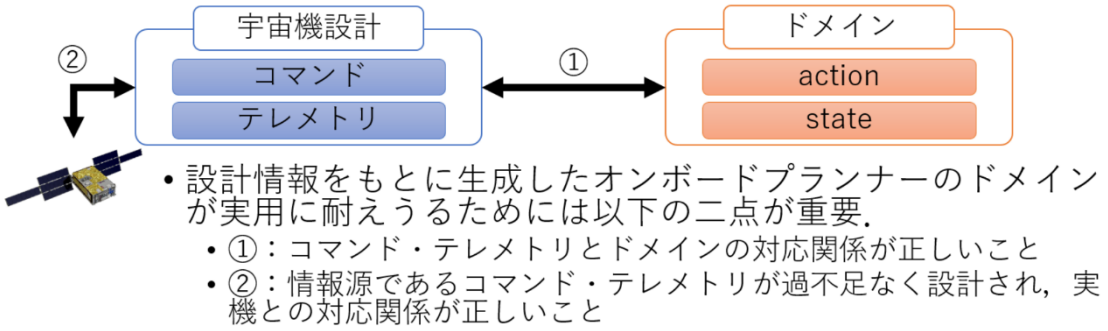

図 3.3 宇宙機の設計情報とドメインモデルのリンクと重要な点

3.2 モデルベースシステムズエンジニアリング

システムへの要求を図示し，要求のトレーサビリティを取りながら各サブシステムの設計を行える手法が，モデルベースシステムズエンジニアリング（Model-based Systems Engineering, MBSE）である．この MBSE のプロセスを利用することにより，システムレベルからコンポーネントレベルにわたる各抽象度にわたって要求のトレーサビリティを保ちながら設計を行うことで，各ステークホルダーが提示した要求が満たされるシステムを設計できる．また，MBSE のプロセスを通して設計情報をダイアグラムとして図示し，ダイア

グラム間のトレーサビリティも保つことにより，情報構造体としての設計モデルを構築していく．このような設計モデルを構成すると，様々なツールを用いることにより，抽象的な設計モデルのレベルで設計の妥当性検証を行うことが可能になる．本節では，このようにして構成した設計モデルと自動計画アルゴリズムにおけるドメインモデルの関連性を議論し，設計モデルからドメインモデルを構成することを検討する．

3.2.1 モデルベースシステムズエンジニアリングの概要

本項では，情報処理推進機構が公開している「モデルベースシステムズエンジニアリング導入の手引き」[52]に則って，MBSE の概要を紹介する．

まず，MBSE のもとになっているシステムズエンジニアリングとは，「システムの開発を成功裏に実現するための複数の分野にまたがるアプローチ及び手段」である[50][52]．システムズエンジニアリングプロセスの基本的な定義は，IEEE 1220-2005[53] として標準化されている．システムズエンジニアリングのプロセスは図 3.4 に示すような二元 V 字モデルで示すことができる．この二元 V 字モデルは，垂直方向にはアーキテクチャの複雑度を示すアーキテクチャ V，水平方向には各アーキテクチャにおける開発のプロセスを示すエンティティ V という二次元の V 字モデルから構成されている．図 3.5 に，エンティティ V の概要を示す．

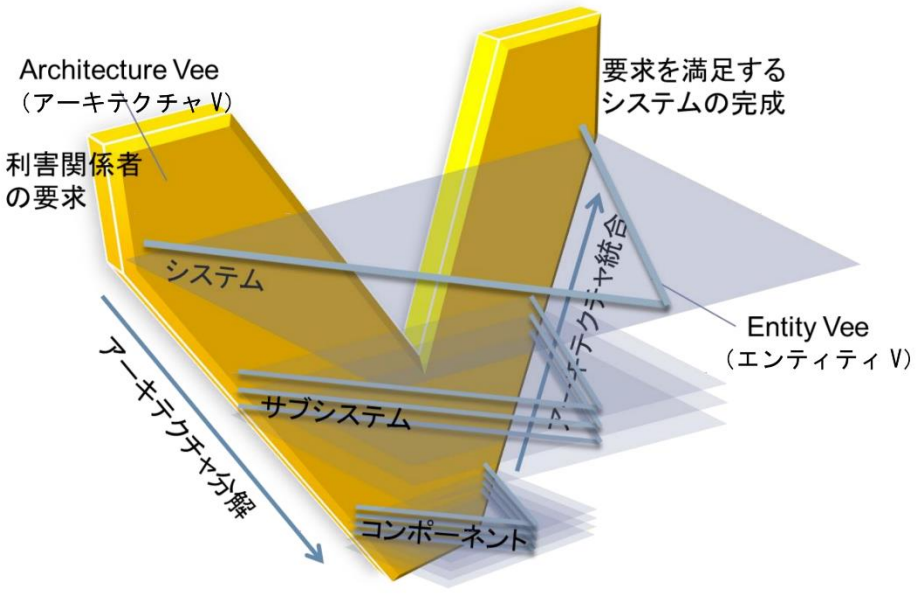

図 3.4 二元 V 字モデル[52][58]

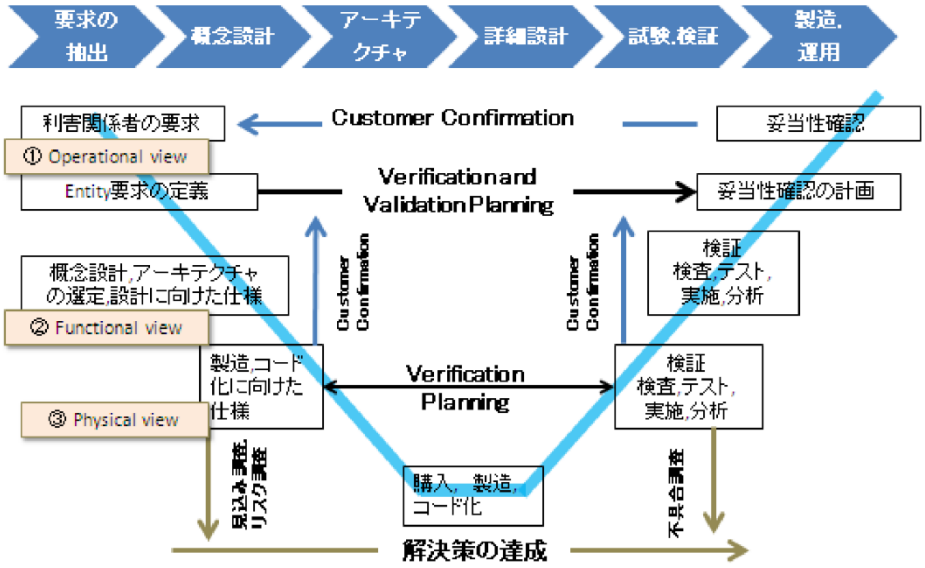

図 3.5 システムズエンジニアリングにおけるエンティティ V の概要[52]

二元 V 字モデルによって規定されるシステムズエンジニアリングの流れは以下のとおりである．まず，システムレベルという複雑で抽象的なアーキテクチャの要求をサブシステム・コンポーネントといったアーキテクチャレベルにまでブレークダウンし，この要求をもとにコンポーネントの設計・製造を行う．製造された各コンポーネントは，コンポーネントのアーキテクチャレベルで仕様を満たしているか確認されるとともに，より抽象度の高い要求を満たしているかも確認される．コンポーネントレベルでの確認が完了すると，コンポーネントを統合してサブシステムを構成し，サブシステムレベルでの確認作業を実施する．サブシステムレベルでの確認作業が完了すると，サブシステムを統合してシステムを構成し，システムレベルでの検証作業を行う．システムレベルでの検証が完了すると，当初のシステム要求を満たすシステムが完成することになる．

MBSE は，上述した「システムズエンジニアリングのプロセスにおいて，開発の対象となる製品の要求とアーキテクチャを，モデルを用いて整理することで開発の複雑さを管理，低減するもの」である[52]．図 3.6 に，エンティティ V における各設計段階において使用するモデリング言語を示している．開発チームにおいて要求の導出やまとめを行う上で，自然言語ではなく SysML (Systems Modeling Language) のような共通言語を用いて要求をモデル化することにより，要求が変更されたことによる影響範囲の管理や，システム設計の妥当性確認や検証のプロセス等も行えるようになる．ここでいうモデルは，様々な要求と設計情報，それらの関係性を記述した情報構造体のようなものである．その情報構造体を人間が理解するために，様々なダイアグラムを図示する．具体的なダイアグラムは，3.2.2 項で紹介する．

MBSE は，航空宇宙分野に限らず，複雑で高い信頼性が求められるシステムを開発する

際に用いられている．航空宇宙分野においては，Boeing 社が MBSE の適用を積極的に行っている[54]．また，CubeSat 開発の標準化に向け，CubeSat MBSE Reference Model [55]というものも提唱されている．日本においては，JAXA の創発考房[56]において MBSE の活用を進めており，その一部は SALTS（つばめ）の開発にも適用された[57]．

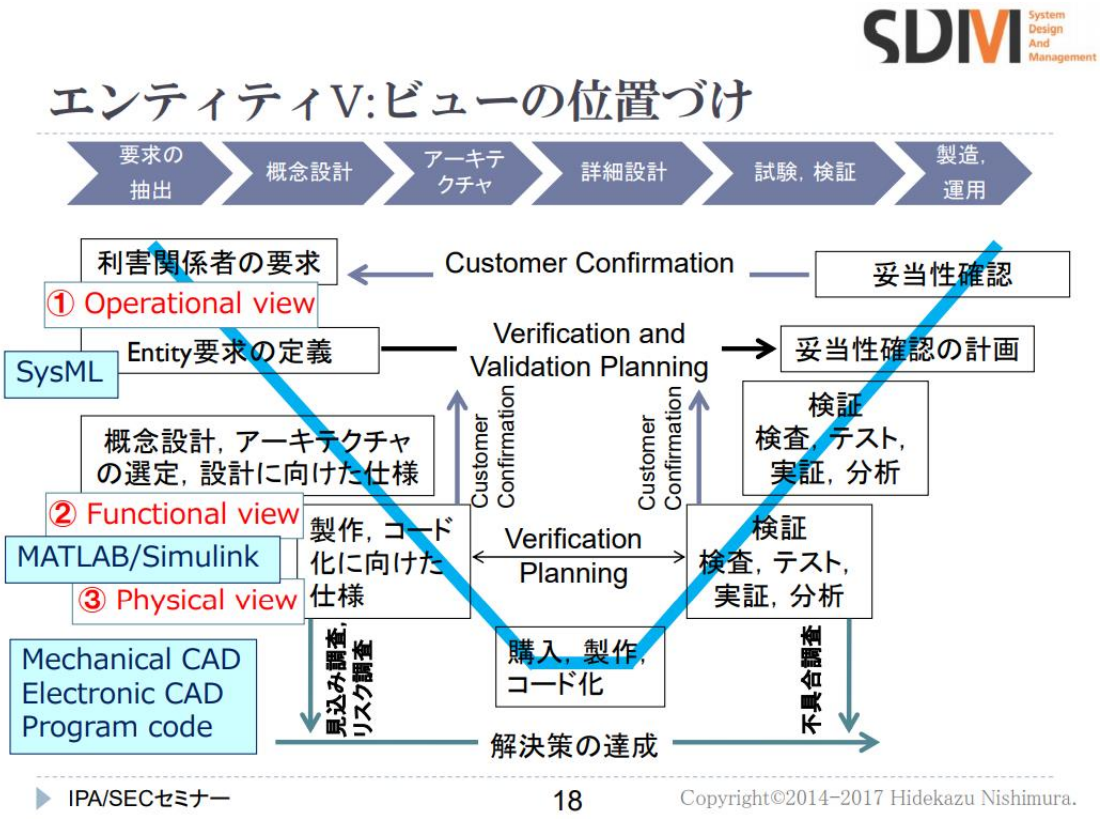

図 3.6 MBSE におけるエンティティ V の概要[51]

3.2.2 モデルベースシステムズエンジニアリングで作成する図（ダイアグラム）

3.2.1 項では，MBSE の概要をまとめた．本項では，MBSE のプロセス上で作成する何種類かの図（ダイアグラム）を，情報処理推進機構が公開している「モデルベースシステムズエンジニアリング導入の手引き」[52]で示されているエレベータシステムの例を用いて紹介する．

まずは，開発すべきシステムをブラックボックスとして考え，システムが持つべき機能やそのコンテクストを整理していく．ユーザーがシステムを利用する際の具体的な利用方法やシステムの振る舞いを整理することで．システムがどのような機能を持つべきか，という

ものを記述するのが図 3.7 に示すような「ユースケース図」である．続いて，システムが利用されるコンテクストを定めるとともに，各コンテクストにおけるシステム内外の構成要素を整理するのが図 3.8 に示す「ブロック定義図」である．これらの図によってブラックボックスとして考えられたシステムのスコープ（範囲）が定まってくるだけでなく，システムの運用概念もモデル化される．モデル化された運用概念をもとに，何らかの分類軸に従って要求を整理するのが図 3.9 に示す要求図である．要求は様々な抽象度からなっているとともに，整理するための分類軸も多様である．全ての要求が出そろってから整理を開始することは難しいため，一つ分類軸を定めたうえで，手元にある要求から徐々に整理を行うことが重要である．

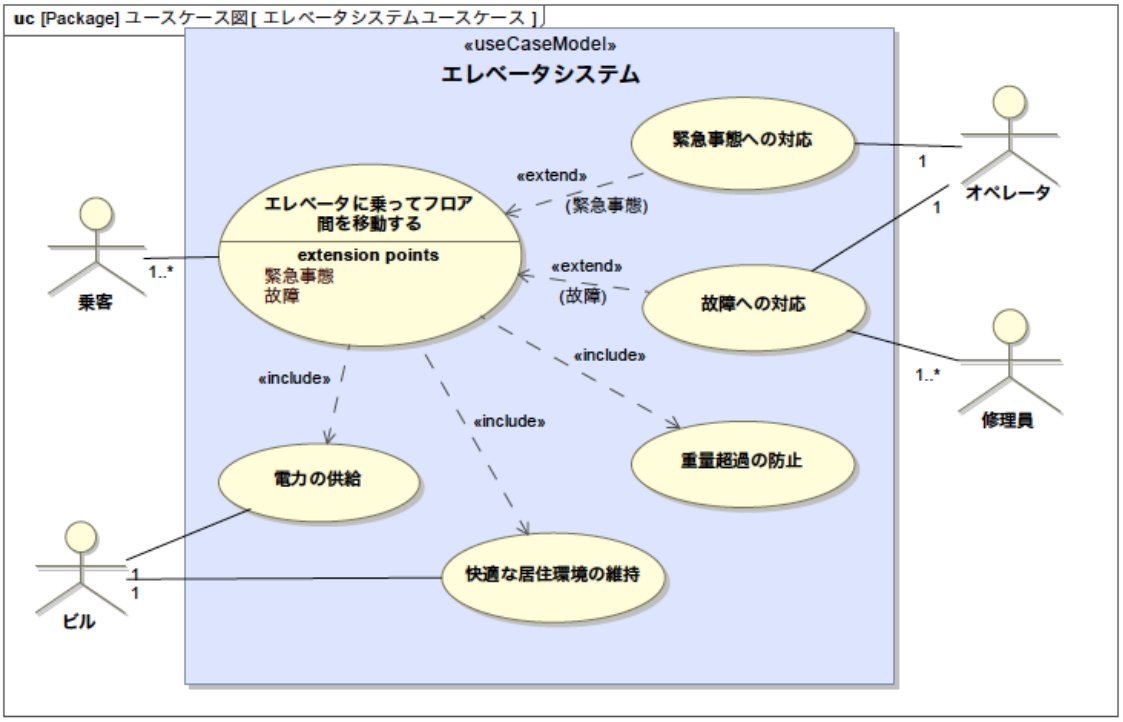

図 3.7 エレベータのコンテキストレベルのユースケース図例[52]

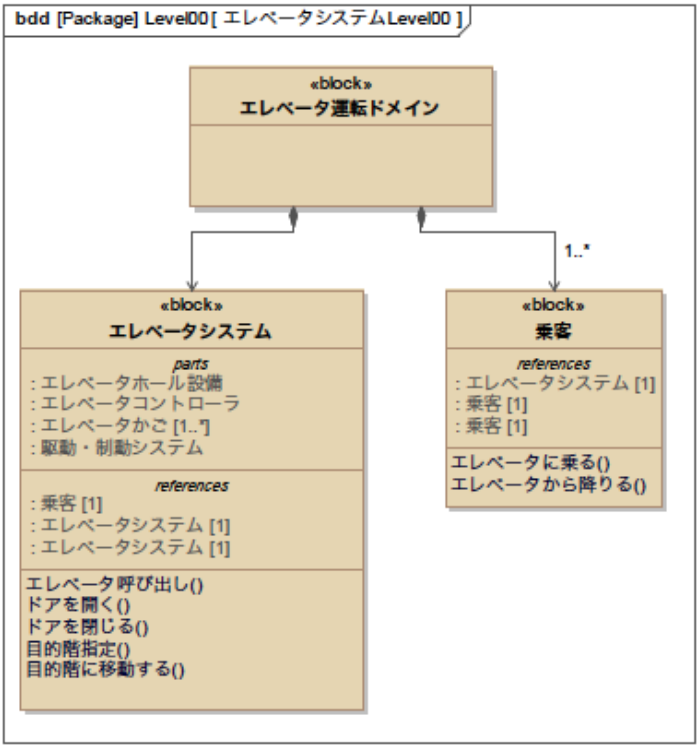

図 3.8 エレベータのコンテキストレベルのブロック定義図例[52]

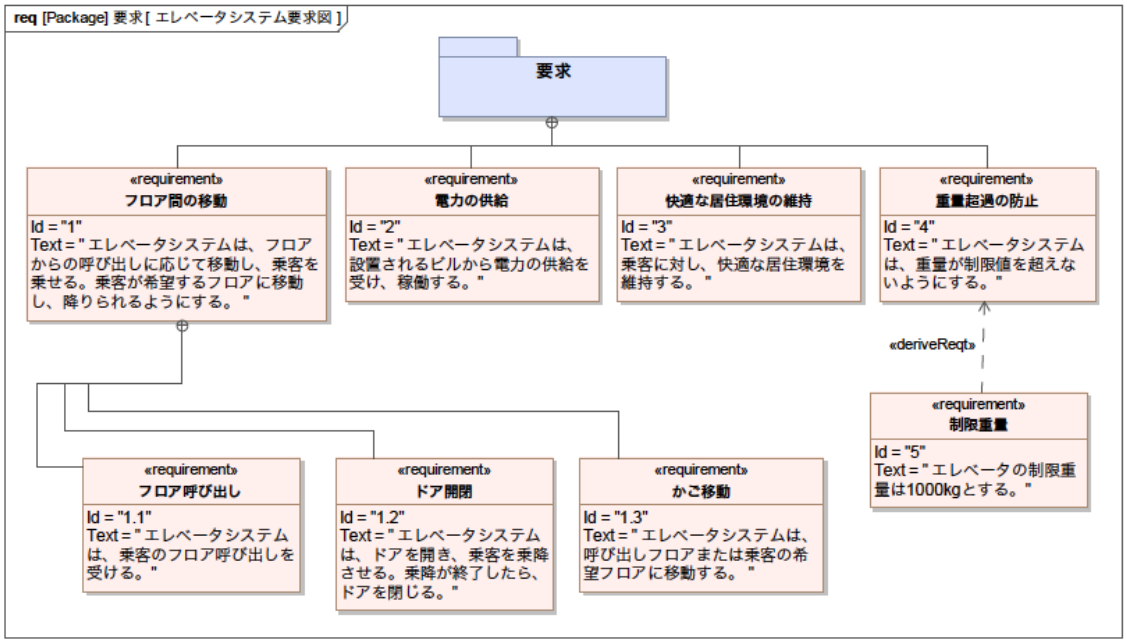

図 3.9 エレベータの要求図例[52]

ここまで作成してきたシステムレベルのユースケース図・ブロック定義図・要求図によって，システムのスコープが明確にされてきた．以降では，システムが提供するサービスや機能を詳細化していく．まずは，ユースケース図によって整理されたユースケースを，図 3.10 に示すようなシーケンス図を描くことによって具体化する．シーケンス図では振る舞いを具体的にモデル化し，振る舞いの順番と入力・出力を見極める．続いて，具体的なシーケンス図をもとに，図 3.11 のような状態機械図（状態遷移図とも呼ぶ）を描画する．状態機械図は，システムの普遍的な振る舞いを示したものになっている．

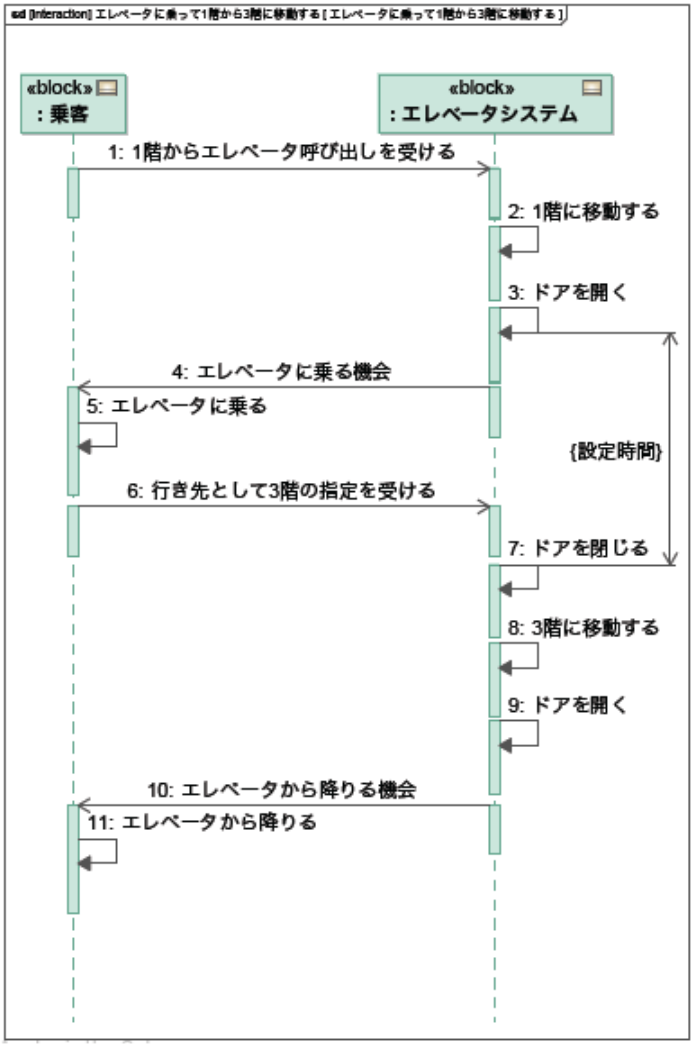

図 3.10 エレベータのシーケンス図例[52]

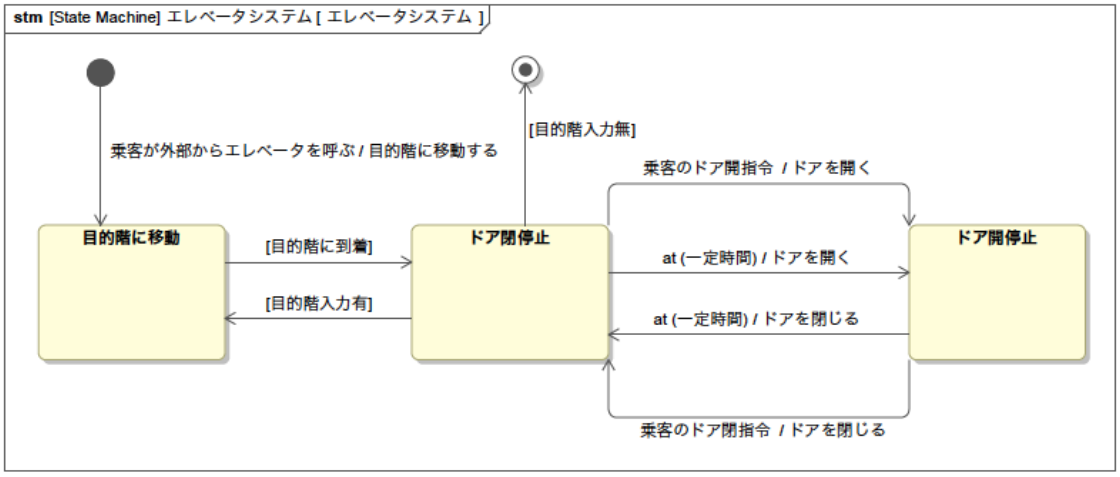

図 3.11 エレベータの状態機械図例[52]

ここまでの図では，システムをブラックボックスとして定義している．このブラックボックスをサブシステム・コンポーネントといった要素に分解することにより，システムを詳細化して開発を進めやすくする必要がある．詳細化のプロセスを進めるうえで，サブシステム分解したブロック定義図やインターフェースを定義する内部ブロック図，サブシステムごとの振る舞いを整理するアクティビティ図など，様々な図を書くことになる．こうした図の描画は，システムレベルで要求をまとめるために描いてきた図を，要求のトレーサビリティを取りながら詳細化していく，という作業に当たる．本論では具体的な作図を第 4 章の実際の宇宙機の例で示すため，本項では詳細な説明を割愛することにする．

3.2.3 モデルベースシステムズエンジニアリングで作成される図とドメインモデルの関係性

3.2.2 項でまとめた通り，MBSE のプロセスを進めるうえでシステムのシーケンス図（図 3.10）や状態機械図（図 3.11）などを描くことになる．また，システムを詳細化していくプロセスにおいても，サブシステムやコンポーネントのシーケンス図及び状態機械図を描くことになる．これらのダイアグラムは，各システム・サブシステムの動作を規定するものであり，システムの振る舞いを記述するドメインモデルと深い関係があると推測できる．本項では，3.2.2 項のエレベータシステムでの例を利用して，ドメインモデルと設計情報の関係性を整理していき，宇宙機におけるドメインモデルと設計情報の関係性を検討する．

まず，ドメインモデルを構成する要素は，state と action の二つに大別できる．このうち，システムがとりうる状態を示す state に関する情報は，図 3.11 の状態機械図に含まれてい

ると考えられる．具体的には「目的階に移動」「ドア閉停止」「ドア開停止」の三つが state に関する情報だといえる．これら三つの情報をさらに詳細化すると，「目的階に移動」は「外部から入力された目的階に移動している」となり，「ドア閉停止」は「ドアが閉まっていて，かつ階の移動が停止している」，「ドア開停止」は「ドアが開いていて，かつ階の移動が停止している」となる．詳細化された情報を整理すると，状態変数やパラメータとして「目的階」「エレベータシステムの移動状態」「ドアの開閉状態」があるとわかる．この状態変数やパラメータがまさに state に関する情報であり，状態機械図から抽出できることがわかる．一方で，action に関する情報は，図 3.11 の状態機械図だけでなく図 3.10 のシーケンス図にも含まれていると考えられる．各図における矢印が action の具体的な内容にあたるだけでなく，状態機械図には action の実行条件に相当する「状態の遷移条件」が記述されており，シーケンス図には各 action に要する「時間」に関する情報が記述されている．以上のことから，ドメインモデルに関連する情報はシーケンス図及び状態機械図に集約されていると考えられる．

続いて，宇宙機における設計情報とドメイン情報に関して検討する．MBSE のプロセスを宇宙機に適用することで，システム・サブシステム・コンポーネントといった様々なアーキテクチャレベルにおける状態遷移図とシーケンス図が作成される．エレベータの例で示した通り，状態遷移図やシーケンス図にはドメインモデルに関する情報が集約されている．ここで，状態遷移図やシーケンス図は，対象とする要素の機能を整理するためのダイアグラムである．一方，宇宙機の操作はコマンド・テレメトリを介して行うことから，コマンド・テレメトリは宇宙機の機能を規定するものであるといえる．そうすると，コマンド・テレメトリの設計情報も状態遷移図やシーケンス図に集約できることがわかる．2.3 節でまとめた通り，宇宙機の自律プランナーを構築するために必要な情報は，ドメインモデルとコマンド・テレメトリの情報，それらの関係性である．ドメインモデルとコマンド・テレメトリの情報は宇宙機の機能という観点から，MBSE のプロセスによって状態遷移図やシーケンス図といった形で整理できる．あとは，これらの関係性もダイアグラムもしくはその背後の情報として記述してやることにより，自律プランナーに必要な情報をそろえることができる．

以上から，MBSE のプロセスで自然と作成される図に追加で情報を付加してやることにより，自律プランナーを構成するために必要な情報を導き出せることがわかる．具体的な手法に関しては，第 4 章で具体例を示しながら説明することとする．

3.3 宇宙機搭載自律プランナー構築手法の提案

ここまで，宇宙機の設計情報と宇宙機搭載自律プランナーに利用するドメインモデルの相似性と，モデルベースシステムズエンジニアリングの概要に関して議論を行ってきた．モデルベースシステムズエンジニアリングによって設計情報をモデル化でき，その検証が設

計と開発の各フェーズで順を追って実施される．これにより，検証された「信頼できる設計情報モデル」が構築できることになる．3.2.4 項で議論した通り，宇宙機搭載自律プランナーに利用するドメインモデルを構築するのに必要な情報は，宇宙機の設計モデルにおいて各機器の状態遷移図やそれに付随する情報にまとまっているため，この情報からドメインモデルをシームレスに構築できると考えられる．ドメインモデルは，第 2 章での議論より PDDL 2.1 以降の形式で作成することになり，この PDDL ファイルを Planning Engine に入力することで，運用目標を達成するための方策を自動計画アルゴリズムによって探索することができる．また，State Monitor や Executor の構築に必要な情報は，ドメインモデルと Low Level Controller におけるテレコマの関係を記述したものであり，これも PDDL ファイルを自動生成するのに必要な情報が定まっていれば十分である．

以上より，本研究において提案する宇宙機搭載自律プランナーの構築方法の概要を図 3.12 に示す．その流れは以下のとおりである．まず，宇宙機の設計を，MBSE のプロセスを利用して実施する．MBSE のプロセスにより，ステークホルダーの要求からコンポーネントの設計まで要求のトレーサビリティをとって行い，「信頼できる」設計情報の構築を行う．この設計情報をもとに各コンポーネントを製造し，試験を行って，MBSE で構築した設計モデルが当初の要求を満たしていることを確認するとともに，設計モデルが実機の機能を正しく記述できていることを確認する．こうして構築できた「検証された設計モデル」をもとに，ドメインモデルやデータベースを構築し，宇宙機のオンボードプランナーを構築する．こうして構築されたオンボードプランナーを，想定される運用の複数ケースに対して試験し，妥当性を確認する．

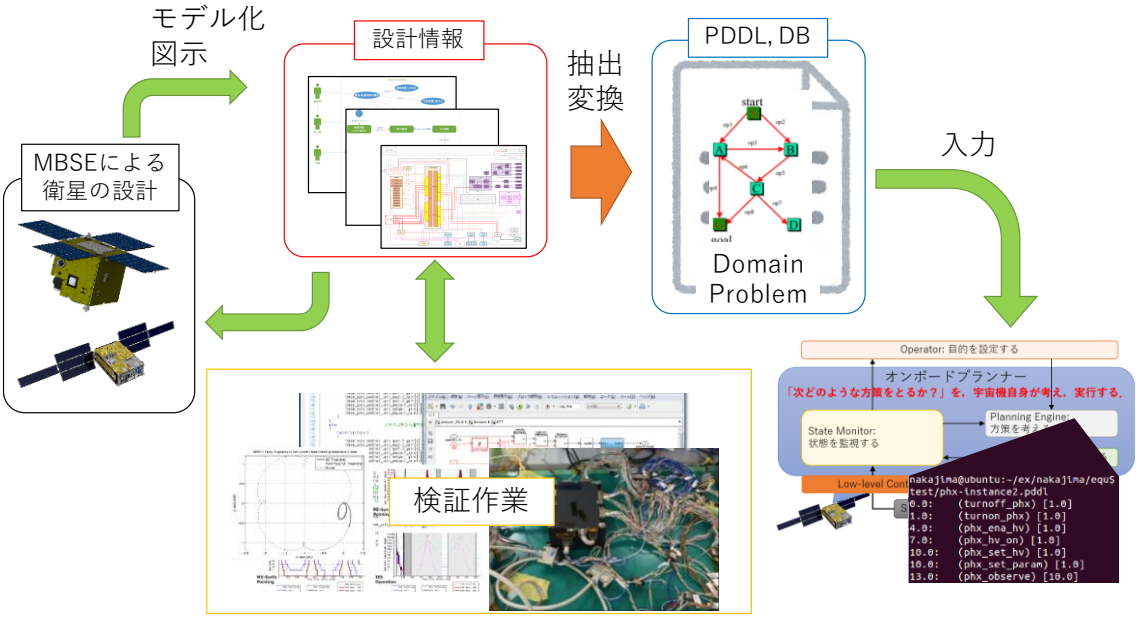

図 3.12 提案手法の全体像

3.4 本章のまとめ

本章では，宇宙機の運用と自動計画アルゴリズムの相似性から，宇宙機の振る舞いを記述するドメインモデルの内容は宇宙機の設計情報，特にコマンド・テレメトリの情報と深い関連性があることを述べた．この関連性から，宇宙機のドメインモデルを正確に記述するためには，コマンド・テレメトリの設計情報が正確であることが重要であると考え，コマンド・テレメトリの設計情報を正確に構成するために MBSE のプロセスを利用することとした．MBSE のプロセスによりシステムレベルの抽象的な要求をもとにトレーサビリティを取りつつ搭載コンポーネントの設計を行うことができ，搭載コンポーネントの設計からコマンド・テレメトリの設計情報が定まる．また，抽象的なモデルの段階で設計の妥当性確認を行うことができるだけでなく，製造した結果からもモデルの修正を行うことよって，検証された設計モデルの構築が可能である．特にコンポーネントの状態遷移図やシーケンス図を書く中で，コマンドやテレメトリの設計に関する情報が抽出できることが示唆された．結果として，MBSE のプロセスによって自然と記述される状態遷移図やシーケンス図といったダイアグラムをもとに，宇宙機のドメインモデルである PDDL ファイルを自動生成することによって，宇宙機の検証された設計情報をもとに自律プランナーの中心をなすドメインモデルを構成することができると導かれた．この「MBSE プロセスを進めることで記述されるダイアグラムをもとに，自律プランナーを構成するドメインモデル・データベースを抽出する」という一連のプロセスが，本研究によって提案する手法となる．

第4章EQUULEUS のエンジニアリングモデルを用いた実験

本章では，第 3 章で提案した宇宙機搭載自律プランナーの構築方法を，実際の宇宙機に適用することで有効性を検証する．今回適用する宇宙機は，東京大学と JAXA が中心となって開発を進めている，EQUULEUS[60]である．

4.1 地球・月系ラグランジュ点探査 CubeSat EQUULEUS とは

EQUULEUS は，NASA が 2019 年に打ち上げる予定の SLS EM-1 ロケットによって打ち上げられる 13 機の CubeSat の中の一つであり，6U (約 10 cm × 20 cm × 30 cm)，14kg 以内のサイズで地球・月系のラグランジュ点である EML2 に航行し，様々なミッションを行う．EQUULEUS のミッションイメージを図 4.1 に示す．また，EQUULEUS の EM(Engineering Model)の写真を図 4.2 に示す．EQUULEUS のミッションは，以下の三つである．

- ミッション 1（工学）：太陽-地球-月系における軌道操作技術
  - 地球-月のラグランジュ点周りの周期軌道へ飛行することで，深宇宙港を基点とした将来の探査ミッションシナリオを先行実証する．
- ミッション 2（理学）：地球磁気圏プラズマ撮像
  - 地球から離れたポイントから磁気圏プラズマの全体像を観測し，ERG と共にジオスペースの包括的理解を目指す．
- ミッション 3（理学）：Cis-lunar 空間における固体天体分布の把握
  - 以下の 2 つの手段のいずれかにより，Cis-lunar 空間における固体天体分布に関する知見を得る．
    - EM L2 点から月の裏側の発光現象を常時観測し，小サイズのメテオロイドの月面衝突フラックスを明らかにする．
    - ダスト検知器により月周辺のダスト環境をモニタする．

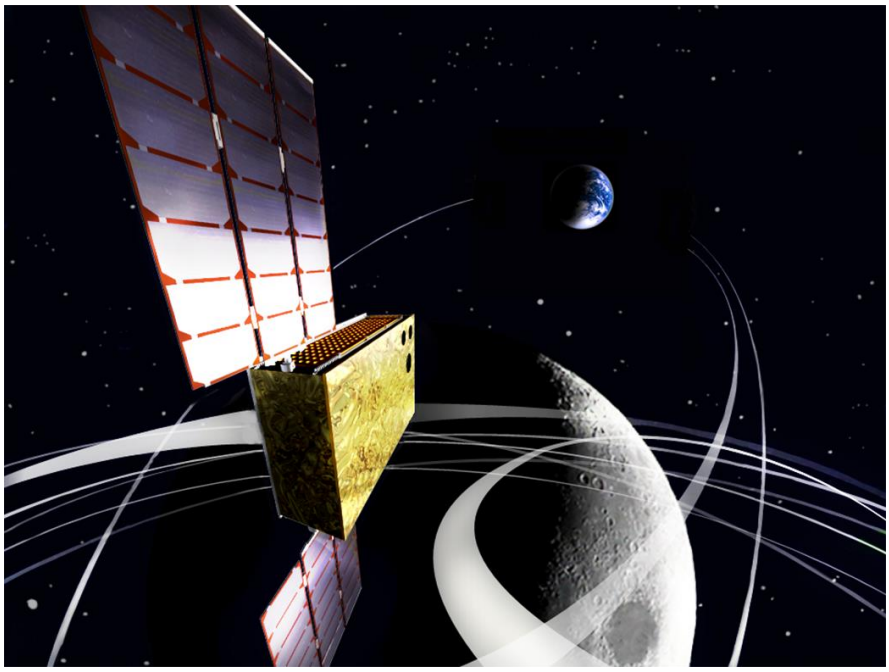

図 4.1 EQUULEUS ミッションイメージ

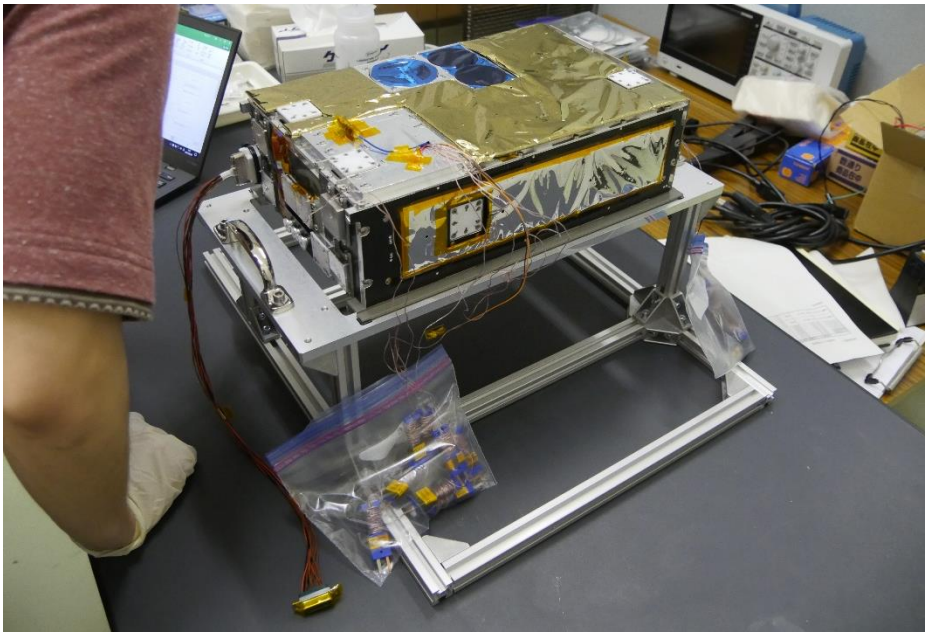

図 4.2 EQUULEUS EM 外観写真

4.2 EQUULEUS における宇宙機搭載自律プランナーの実装方針

2.4 節で述べた通り，本研究ではオープンソースで公開されている自動計画アルゴリズムを Planning Engine として利用することを想定している．今回，Planning Engine として SMTPlan+[49] を利用することとする．SMTPlan+ は，PDDL 2.1 の発展形である PDDL+[66] に対応した自動計画アルゴリズムであり，GitHub 上でソースコードが公開されている．Low Level Controller に関しては，東京大学の超小型人工衛星の搭載ソフトウェアに使用されている C2A[43][44] というアーキテクチャに則ることとする．C2A はコマンドを中心として搭載ソフトウェアを構成するアーキテクチャであり，再利用性や再構成能力に優れている．さらにその副次的な効果として，コマンド・テレメトリのデータベースを作成し，搭載ソフトウェアコードや地上局で利用するデータを自動生成することができる [45][46].

EQUULEUS は，2018 年 11 月時点で，フライトモデルの総合試験を開始しており，設計が確定している．搭載ソフトウェアに関しては，従来の搭載ソフトウェア（本研究における Low Level Controller 部分）のみを想定しており，この前提で搭載計算機（OBC）の設計を行っている．そのため，Planning Engine を中心とした宇宙機搭載自律プランナーを動作させるために，OBC とは別に計算機を用意し，そちらにオンボードプランナー部を実装することとする．ただし，前述した EQUULEUS 特有の事情を考慮せずとも，宇宙機において自律プランナーのような自律機能を搭載する場合，Low Level Controller が実装される計算機とは別に自律機能専用の計算機を搭載することが望ましいと考えている．この理由としては，万が一自律機能が暴走した際でも自律機能が搭載された計算機の電源を切ることによって，Low Level Controller と自律機能を切り離し，Low Level Controller のみで生存できるためである．図 4.3 に，EQUULEUS OBC フライトモデルの写真を示す．基板は二枚構成となっており，その大きさとしては，約 10 cm × 10cm のフットプリントに，二枚合計の厚さが 1.5 cm 程度になっている．

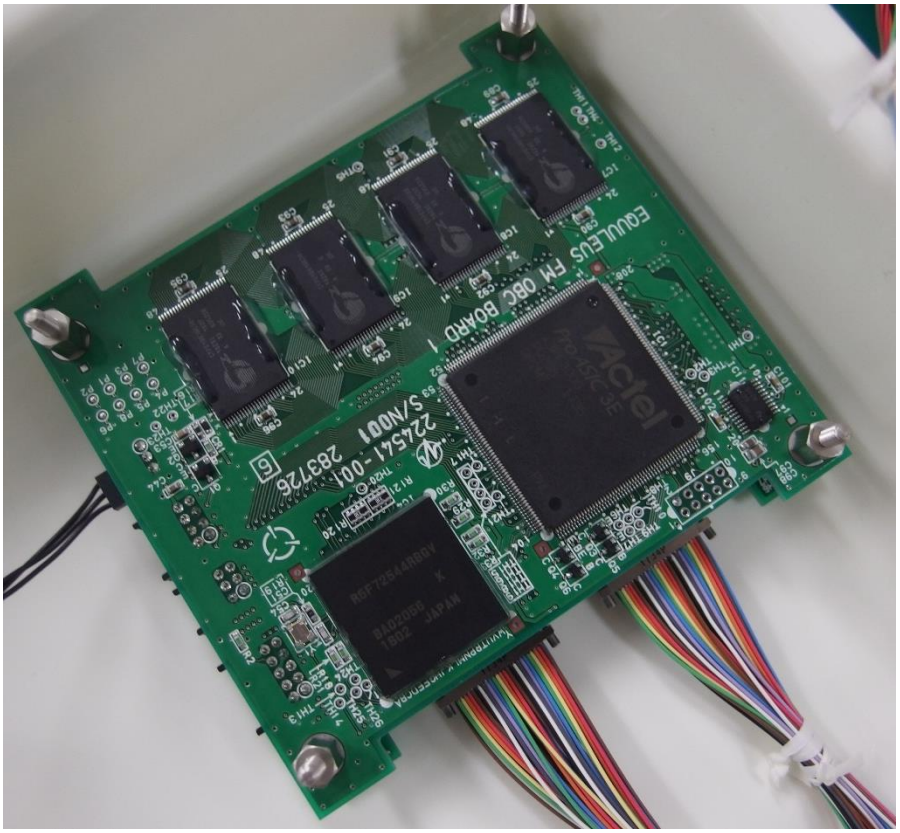

図 4.3 EQUULEUS OBC (FM)の写真

OBC とは別の計算機を選定するうえで以下の事項を考慮する.

- CubeSat に搭載が可能な程度に小型であり，オンボードプランナーが動作する程度に高性能である
  - Linux，特に Ubuntu が動作するのが望ましい
- 軌道上での利用が検討されている

この事項が満たされる計算機として，本研究では Raspberry Pi 3 model B+[62] を利用することとする．Raspberry Pi 3 model B+ の外観図を図 4.4 に示す．Raspberry Pi 3 model B+ は，Ubuntu を動作させることができ，OBC と通信するうえで必要なシリアル通信のポートや USB ポートが複数用意されており，性能としては十分である．また，Raspberry Pi 3 model B は，東京工業大学が開発している Deep Learning を用いた姿勢センサ[61]で採用されている．Raspberry Pi 3 model B + と Raspberry Pi 3 model B は異なるモデルで，前者は後者に対して高性能化・熱設計の向上などがなされている．異なるモデルではあるものの，同シリーズの計算機であることから，今後この計算機もしくはその後継機種が宇宙利用される可能性は高いと考えている.

図 4.4 Raspberry Pi 3 model B+ 外観図[62]

OBC と Raspberry Pi 3 model B+ （以後，Raspberry Pi と呼称する）を組み合わせた場合の宇宙機搭載自律プランナーの構成に関して検討を行う. 前述した通り, Planning Engine を動作させるために Raspberry Pi を用意しているので，Planning Engine は Raspberry Pi に実装することとする．また，OBC は搭載機器と通信をする Low Level Controller の役割に特化した計算機であるため, Low Level Controller は OBC に実装することとする. Operator に関しては，人間がその役割を果たすこととするため，宇宙機に搭載される計算機に実装する必要性はない．したがって，State Monitor と Executor をそれぞれどちらの計算機に実装するかが検討項目となる．ここで，Low Level Controller が OBC に実装されるのは，自律機能が実装されていない従来の一般的な宇宙機と同様であることから，他の機能を Raspberry Pi に実装することで，従来の搭載ソフトウェア実装の枠組みから逸脱することなく，宇宙機の搭載自律プランナーを新規に実装することができる．このことを考慮すると，Executor と State Monitor は Raspberry Pi に実装することが望ましいといえる．

以上から，宇宙機搭載自律プランナーは以下の図 4.5 のような構成となる．Raspberry Pi には Planning Engine，State Monitor および Executor が実装され，OBC には Low Level Controller として C2A が実装される．この二つのボードは，地上局とやり取りする際と同様のテレメトリ・コマンドのパケットレベルでデータの共有を行う．Operator は，運用者として人間が担当することとする．

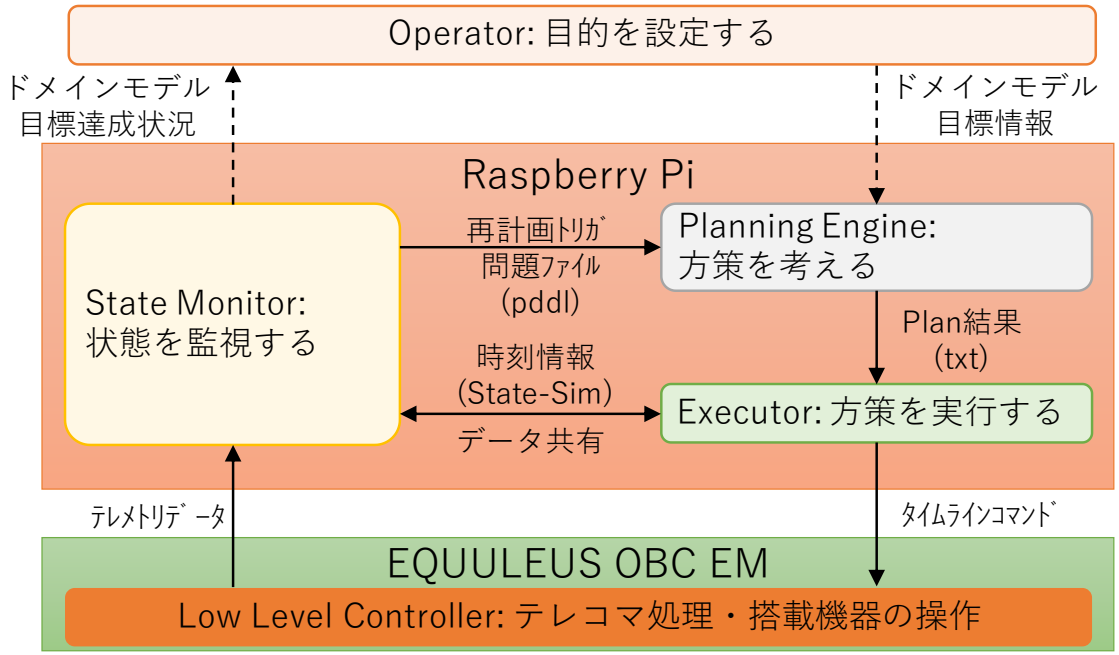

図 4.5 EQUULEUS EM における宇宙機搭載自律プランナーの構成図

4.3 モデルベースシステムズエンジニアリングによる宇宙機モデルの構築

4.3.1 MagicGrid の利用

MBSE を進めていく上での Methodology として，いくつかの手法（ガイドライン）が存在する．本研究では，一つの例として No Magic 社が提唱している MagicGrid[63]を採用し，要求のトレーサビリティを取りながら設計モデルの構築を行っていく．MagicGrid は MBSE の一般的な Methodology を示したものであり，その全体フローを図 4.6 に示す．MagicGrid においては，縦方向にはダイアグラムで対象とする抽象度，横方向には記述する要素が並べられ，各ダイアグラムを描く順番とそれらにおける要求の対応関係が整理されている．4.3.2 から 4.3.12 までの各項では，この MagicGrid に則って各ダイアグラムを作成している．

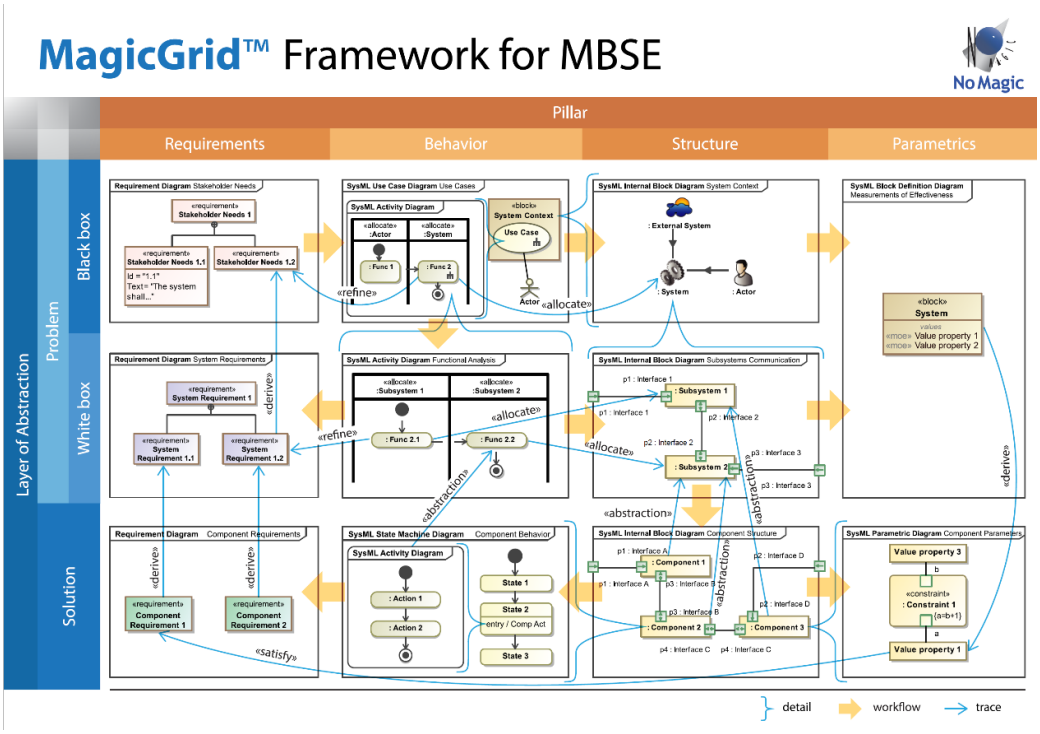

図 4.6 MagicGrid の全体像[63]

4.3.2 Stakeholder Needs の分析

まず，Stakeholder Needs の分析を実施する．この項目においては，ステークホルダーの要求や，規制，原則，内部のガイドラインを記述する．この中には，政府の規制（＝法律など）も加わることになる．EQUULEUS の例では，主に以下の要求となった．

- ミッション目的を達成する（ミッション担当者）
- SLS EM-1 ロケットによって打ち上げられる（ロンチャー，システム担当者）
- 既存の地上局で運用できる（システム担当者）
- 法的な要求を満たす（政府）

また，4.1 節に記述した通り，EQUULEUS には 3 つのミッション目的がある．これらは，要求 1 の「ミッション目的を達成する」の要求をブレークダウンしたものになるため，その仮想に記述されることになる．このようにして整理し，作成された Stakeholder Needs の一部を抜粋したものを図 4.7 に示す．

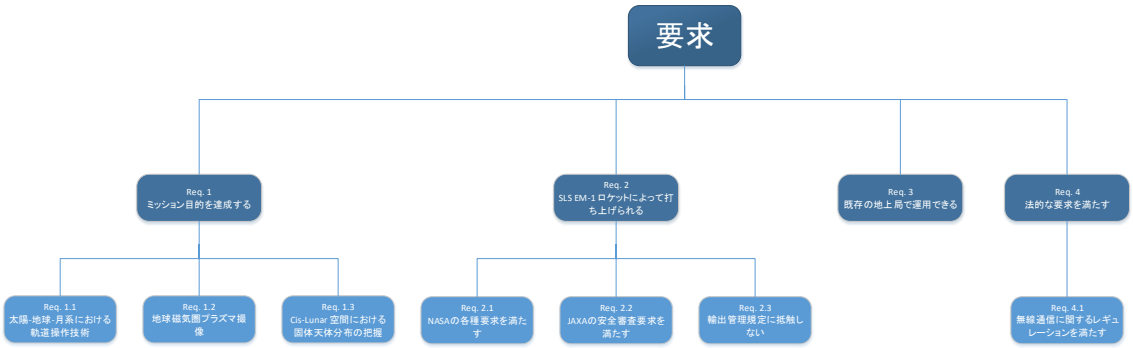

図 4.7 Stakeholder Needs の抜粋例

4.3.3 Use Case 解析

ユーザーからの機能的な要求を書き下すために，ユーザー視点でシステムの使われ方を記述するのが Use Case 解析である．ユーザーを Actor という形で描き，各 Actor がシステムの持つべき機能とどのように結びつき，ユーザーがシステムをどのように利用するかを記述していく．この Use Case 図を作成する際には，このシステムが利用されるコンテクストに留意する必要性がある．Use Case 図において書き下された機能は，Stakeholder Needs を詳細化したものになっている．図 4.8 に EQUULEUS における Use Case 図の一部を抜粋したものを示しているが，ここで示している「観測・軌道制御の運用」は Stakeholder Needs で示した要求 1 と対応している．

EQUULEUS の例では，主に以下の Use Case を検討した．

- 地上試験
- ロケット搭載中
- 打ち上げ時
- ロケットからの分離時
- 運用時

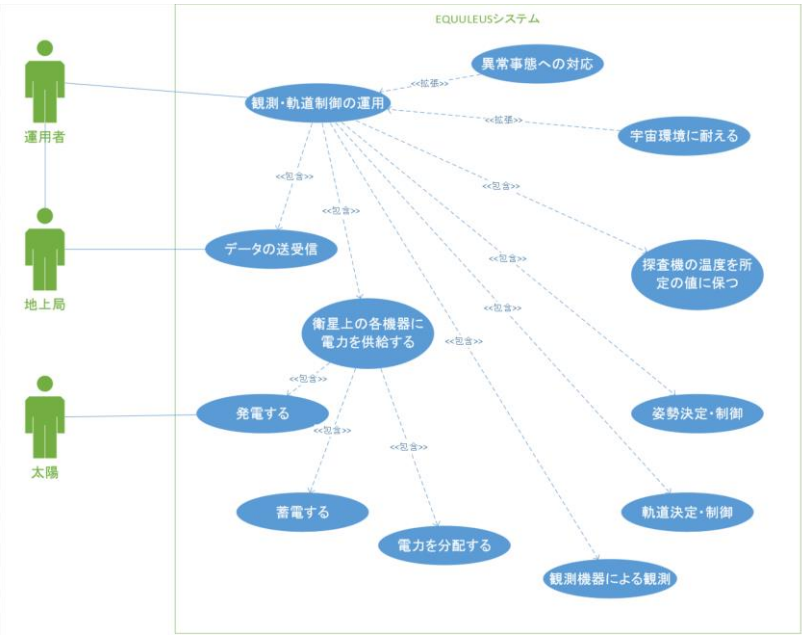

図 4.8 Use Case 図の抜粋例；運用時

4.3.4 System Context の整理

このシステムが Actor や外部環境・内部環境とどのように相互作用しているかを記述するのが，System Context の整理である．この System Context を示した図は，Use Case を記述するときに留意した，利用されるときのコンテクストごとに作成される．

EQUULEUS では，以下のコンテクストを考慮した．カッコ内は，対応する Use Case を記述している．

- 地上試験中（地上試験）
- ロケット搭載中（ロケット搭載中，打ち上げ時）
- 軌道上（ロケットからの分離時，運用時）

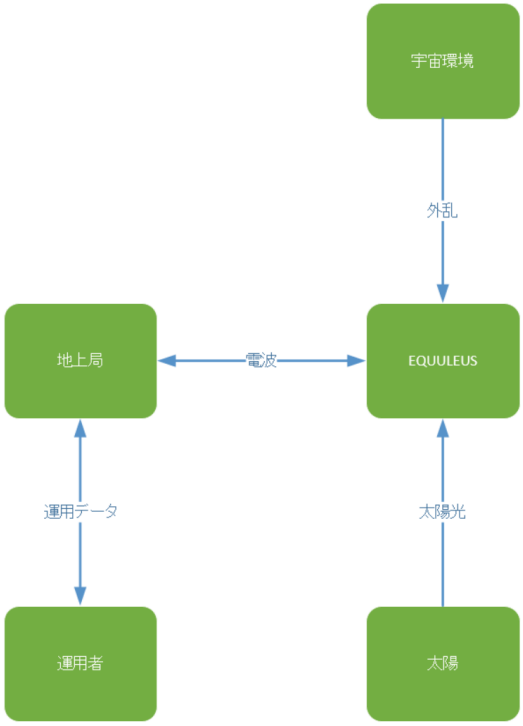

図 4.9 System Context の抜粋例：軌道上

4.3.5 Measurements of Effectiveness および Measurements of Performance の整理

機能として明示されないような，Stakeholder Needs や目的などを，定量的に記述するのが，Measurements of Effectiveness（MoE）の整理である．また，MoE をさらにブレークダウンし，MoE を達成するために必要なシステムの性能を整理したものが Measurements of Performance（MoP）の整理である．MagicGird のフローにおいては MoE の整理を求められているが，MoP の整理まで踏み込む方がより設計に必要な情報を可視化できると考え，本項では MoP の整理まで実施している．EQUUELUS では，主に以下のようなものが該当し，一部を抜粋した表を表 4.1 に示している．

- バッテリーの容量や充放電レート
- 姿勢決定・制御精度
- 推進系の推力

表 4.1 Measurements of Performance の抜粋例

項目	単位
ミッション期間	[yr]
宇宙機重量	[kg]
慣性テンソル	[kg･mm2]
SAP発電量	[W]
バッテリー容量	[Wh]
バッテリー充電速度	[C]
バッテリー放電速度	[C]
DVスラスタ推力	[mN]
RCSスラスタ推力	[mN]
比推力	[s]
RW出力トルク	[mNm]
RW保有角運動量	[mNms]
姿勢絶対指向	[deg]
姿勢安定度	[deg @time]
絶対姿勢決定	[deg]
相対姿勢決定	[deg]
コマンドビットレート	[bps]

4.3.6 System Requirements の整理

Stakeholder Needs をもとに，システムへの要求を整理していく．本項における要求は，Goals と Objectives の二つからなる．Goals は抽象度の高い，達成したい内容を記述し，Objectives は Goals を達成するための具体的な方策を定義する．EQUULEUS での例を一つ挙げると，

- Goals: 軌道操作技術の実証を行う．
- Obj#1: 軌道操作の機能として，水を推進剤とする推進系を持つ．
- Obj#2: 軌道決定の機能として，レンジング機能を持つ．
- Obj#3: 軌道決定を行うために，日本の地上局からの可視時間が一日一回以上ある軌道に投入される．

といったものになる．この Goal は，Stakeholder Needs の要求 1.1 に対応している．

4.3.7 必要機能の分析（Functional Analysis）

Use Case 図の作成時に抽出された機能をさらに詳細化し，システムが持つべき機能を解析するのが，必要機能の分析（Functional Analysis）である．ここでは，機能を詳細化したうえで，どのようなサブシステムが必要かを整理することが重要になる．記述された機能は，System Requirements として整理された各要求を詳細化したものとなる．EQUULEUS での一例を以下に示す．カッコ内は，機能が属するサブシステムであり，以下の箇条書きで記述した機能をダイアグラムで描画したものを図 4.10 に示している．

- 月観測機能を詳細化
  - 姿勢制御（電源，姿勢）
  - 電力確保（電源）
  - 観測機器起動（電源，ミッション）
  - 観測パラメータ設定（ミッション）
  - 観測（ミッション）
  - データ転送（CDH，ミッション）
  - ダウンリンク（CDH，通信）

実際には，上記機能をさらに詳細化した Functional Analysis を行い，必要のサブシステムの抽出を行っている．

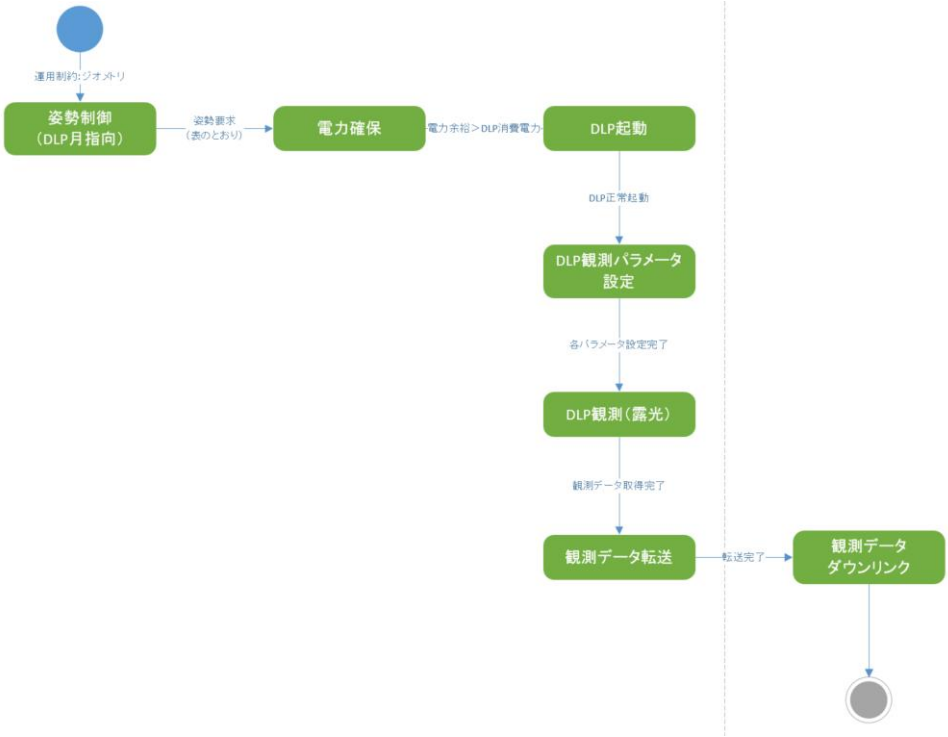

図 4.10 Functional Analysis の抜粋例：月観測の Activity 図

4.3.8 Logical Subsystems Communication の整理

Functional Analysis で定義されたサブシステムをもとに，サブシステムをもとに，各サブシステム間のインターフェースを定義・整理するのが，Logical Subsystems Communication の整理である．この段階で，ICD (Interface Control Document)が作成でき，各サブシステムへの設計要求がより具体的なものになってくる．例として，EQUULEUS のサブシステムレベルのブロック図を図 4.11 に示す．

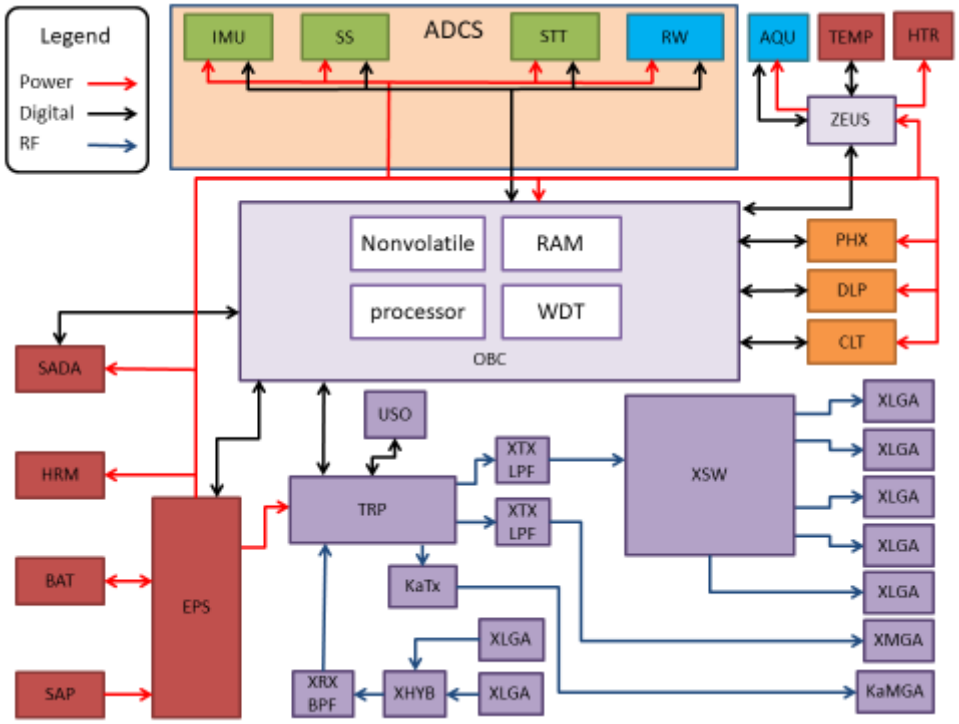

図 4.11 EQUULEUS のサブシステムブロック図

4.3.9 Component Structure の整理

Logical Subsystems Communication の内容に関して，サブシステム内をコンポーネントごとに分割し，詳細化するのが Component Structure の整理である．EQUULEUS では，サブシステム内に含まれるコンポーネントの数が少ないため，コンポーネントレベルでブロック図を作成してもその複雑性は許容できると考えたため，全コンポーネントの間のインターフェースを一つのダイアグラムで記述することとした．実際のブロック図を図 4.12 に示す．

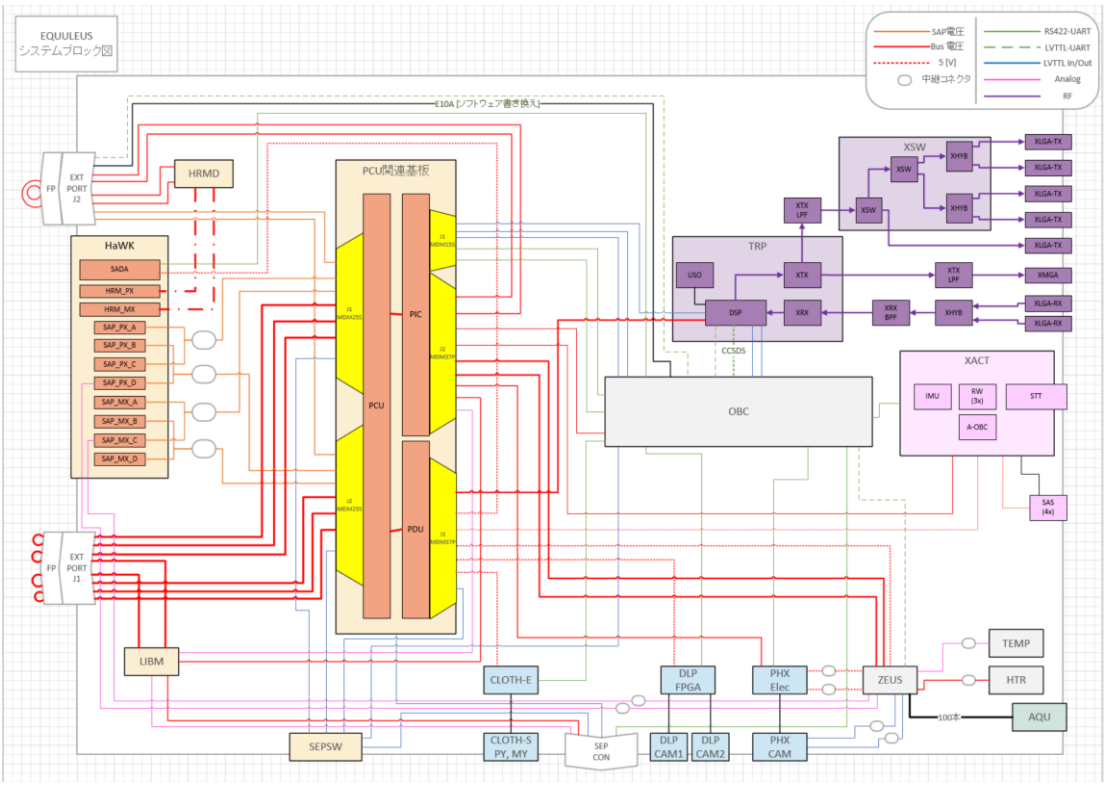

図 4.12 EQUULEUS の詳細ブロック図

4.3.10 Physical Characteristics の整理

MoE や MoP の部分で定義された各種変数と，コンポーネントの関係を記述するのが Physical Charasteristics の整理である．EQUULEUS の例をいくつか示すと，

- 各機器の消費電力の和は，太陽電池の発電量を下回っているか
- 重量の和は 14kg 以下か
- 総重量と推力の関係から，既定の加速度を満たしているか

などがこれにあたり，各コンポーネントはこれらの制約条件を満たすように設計されることになる．

4.3.11 Component Behavior の定義

各コンポーネントの状態遷移図と，その状態遷移に含まれる action を記述するのが Component Behavior の定義である．この図を作成する際には，状態遷移図の各 state と，Functional Analysis の機能とのトレーサビリティに留意する必要がある．EQUULEUS の例として，地球磁気圏プラズマ撮像を行うための観測機器 PHOENIX（PHX）の状態遷移図の一部を図 4.13 に示す．

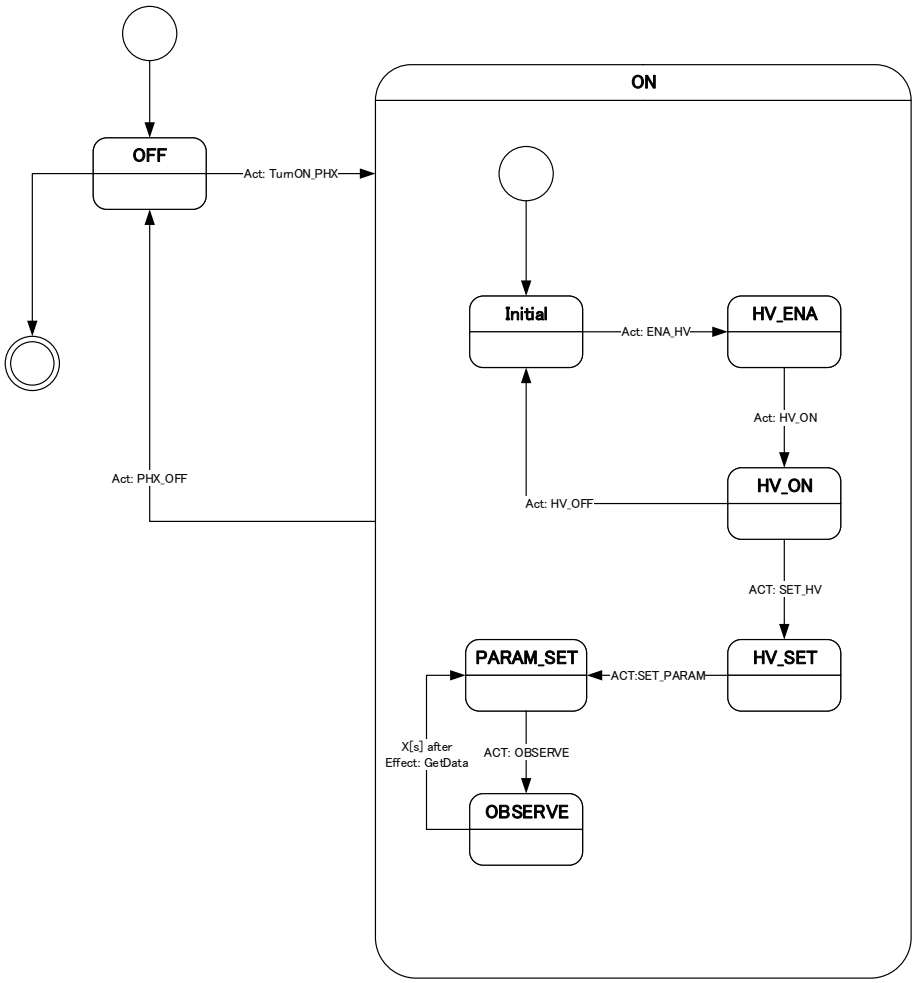

図 4.13 Component Behavior の抜粋例：観測機器の状態遷移図

4.3.12 Component Requirements の整理

ここまで作成してきた各図をもとに，コンポーネントへの要求を整理し，製造するコンポーネントへの設計要求をまとめる．具体的にまとめる内容は，Component Structure で定義されたインターフェース情報，MoE や Physical Characteristics で定義された設計制約，Functional Analysis や Component Behavior で定義された機能などがある，各機器への設計要求を簡単にまとめた表の一部を表 4.2 に示している．

表 4.2 Component Requirements の抜粋例

機器	機能
（XACT）	姿勢系資料参照．RW/IRU/STTを搭載＋RCS搭載
AQU	5.5.1 節 Figure 5 のようなRCS配置とする．
AQU	推力4mN，比推力70s，推進剤タンク容量1200cc　以上を満たす．
AQU	（システム要求以外はない？）
BAT	HTRを持つ
BAT	温度計を持つ
CDH	姿勢異常が発生してから1秒以内にシャッターへの通電を行う．
CLT	（観測する＋テレコマ機能くらい）
COMM	受信機能，送信機能を持つ
COMM	（計装損失や安定度などは機能に入らない？）
DLP	（同上．各種パラメータ設定，撮像などなど）
HYB(RX)	2つのアンテナで受信した電波は合成される
LGA-RX	PY, MY の二面に搭載される．
LGA-TX	PY, MY, PZ, MZ の四面に搭載される．
LGA-TX	MX面にも搭載される．
MGA-TX	PY面にMGAを搭載する
OBC	2GBのNAND Flash を搭載する．
OBC	OBC，PCUにLVTTL規格のリセット信号を送信できる
OBC	CCSDSとコンパチ
OBC	各機器と，UART規格で各機器と通信
OBC	50MIPS以上の処理性能のCPU？
PCU	起動に対して2-inhibit 以上持つ
PCU	過電流検知機能を持つ
PCU	過電流シャットダウン機能を持つ
PCU	OBCからの信号に応じて，全機器の電源を落とす
PCU	TRPからの信号に応じて，全機器の電源を落とす
PCU	OBCからの信号に応じて，OBCのみの電源を落とす
PCU	各機器の電源をOFF/ONできる
PCU	12V（非安定），5V（安定）の二種類の電源を作る
PCU	SAPから電力を引き出す
PCU	BATを充放電する
PCU	各機器の電源をON/OFFする
PCU	MPPT機能を持つ
PCU／OBC	電波放出・SAP展開までに15秒以上待てるシステムとする
PHX	通電を始めて4秒以内にシャッターが閉じる

4.3.13 検証計画の作成方法

ここまで，MagicGrid に則って設計要求をまとめてきた．本項では，実際に製造されたコンポーネントが，コンポーネントやシステムへの要求を達成しているかを検証するための検証項目をどのように作成するかを議論する．上記要求のうち，システム要求は 4.3.6(System Requirements)，コンポーネント要求は 4.3.12 (Component Requirements) によって定義されている．そのため，この二項によって定義された要求項目を縦に並べることで，検証すべき項目に漏れはなくなると考えられる．そのうえで，その各項目に対して検証方法を設定すれば，漏れのない検証計画が作成できる．このようにして作成した検証計画を表 4.3 に示す．

表 4.3 検証計画の例（電源系機器）

	機能試験		環境試験(単体)		環境試験(システム)			
検証項目	単体試験	統合システム試験	恒温槽試験	振動試験	恒温槽試験	熱真空試験	振動試験	衝撃試験
KSW・FPの機能確認	○	○	○	○	○	○	○	○
各供給CHのON/OFF制御	○	○	○	○	○	○	○	○
MPPT制御回路の機能確認	○	○		○	○	○		
過充電制御回路の機能確認	○	○		○	○	○		
タイマー起動	○	○	○	○	○	○	○	○
リセット機能	○	○	○	○	○	○	○	○
過電流シャットダウン機能	○	○						
過放電保護機能	○	○	○	○	○	○	○	○
コンポ起動時の消費電力確認		○			○	○	○	○
コンポ突入電流測定・トリップ電流値との整合性確認		○						
DCDCの排熱パス健全性確認						○		
基板間基板の構造的健全性確認				○			○	○
分離スイッチの構造的健全性確認							○	○

4.3.14 モデル化された設計情報とドメインのリンク

ここまで，設計情報とその検証方法を整理してきた．MBSE に則って作成された設計情報をもとに製造し，製造されたコンポーネントを検証方法に従って確認していくことで，当初の Stakeholder Needs が達成されるようなコンポーネントが製造され，その設計情報が各ダイアグラムやその関係性という形でモデル化される．もちろん，実際に製造することによって予期せぬ設計変更や制約条件の負荷が加わる場合もあるが，その場合は設計モデル

を実機に合わせる形で修正し，要求から逸脱していないかを確かめる，といったようにイタレーションをおこなうことになる．

実際に作成された設計モデルをもとに，Planning Engine で利用するドメインモデルをどのように生成するかを議論していく．まず，宇宙機の動作を記述するドメインモデルに必要な情報は，各コンポーネントがとりうる状態（state）と実行できる action，各 action を実行するのに必要な条件やその効果といった付加情報などが挙げられる．このうち，コンポーネントがとりうる状態は，Component Behavior における状態遷移図の構成要素として定義されている．具体的には，状態遷移図で記述される各状態である．また，action は，各コンポーネントの Component Behavior における Activity 図や状態遷移図の構成要素として定義される．具体的には，状態遷移図において各状態をつなぐ矢印という形で記述されている．その一方で，action の付加情報は明確に記述すべき図は決まっていない．一部，実行条件を状態遷移図等に記述することがあるが，実行にかかる時間や消費するリソースなどの情報まで明確に記述することはほとんどない．そこで，他の二項目は Component Behavior にまとまっていることから，action の付加情報に関しても Component Behavior に付加情報として載せることが望ましいと考えられる．言い換えると，Component Behavior に action の付加情報を記述することで，ドメインモデルの構築に必要な情報を Component Behavior にそろえることができると考えられる．EQUULEUS における実験例では，Component Behavior を一度 Excel ファイルに変換し，PDDL ファイルの形式へと変換している．図 4.13 の PHX に関する Component Behavior をもとに作成した，PDDL ファイルへの変換を行う前の Excel ファイル形式となっているドメインモデルの記述例を表 4.4 に示している．

表 4.4 ドメインの記述例：観測機器

Name	Parameters	Duration	Condition	effect	Code
TurnON_PHX		1	at start (not (sw_phx))	and (at start (sw_phx))	(:durative-action TurnON_PHX□arameters ()□uration (
TurnOFF_PHX		1	at start (sw_phx)	and (at start (not (sw_p	(:durative-action TurnOFF_PHX□arameters ()□uration
PHX_ENA_HV		1	and (over all (cur_phx)) (	at end (phx_hv_ena)	(:durative-action PHX_ENA_HV□arameters ()□uration
PHX_HV_ON		1	and (over all (cur_phx)) (	and (at end (phx_hv_on	(:durative-action PHX_HV_ON□arameters ()□uration (
PHX_HV_OFF		1	and (over all (cur_phx)) (	and (at end (not (phx_h	(:durative-action PHX_HV_OFF□arameters ()□uration
PHX_SET_HV		1	and (over all (cur_phx)) (	at end (phx_hv_param)	(:durative-action PHX_SET_HV□arameters ()□uration
PHX_SET_PARAM		1	and (over all (cur_phx)) (	at end (phx_obs_param	(:durative-action PHX_SET_PARAM□arameters ()□ura
PHX_OBSERVE		10	and (over all (cur_phx)) (	at end (phx_obs_data)	(:durative-action PHX_OBSERVE□arameters ()□uratio

本論文で示す例においては，各ダイアグラムを SysML に沿った形式ではなく，Microsoft Visio を用いて UML ベースで図示している．そのため，Component Behavior に action の付加情報を手動で書いてしまっている．しかし，本来はこの情報も設計モデルから自動生成・抽出することが望ましいと考えられる．最初に Component Behavior を整理した際には記述されないような action の付加情報が，どのように設計モデルから自動生成・抽出できるかを，具体的な action を例にして考察する．

姿勢系統合機器のバス電源 ON という action に関して考察を行う．この action の実行条件には，姿勢系統合機器の 5V が ON であることが必要，というものがある．この条件は，姿勢系統合機器のバス電源はリアクションホイールの駆動電源であるため，制御系の電源である 5V 系が供給されていなければ正常動作を期待できない，ということから加わった条件である．すなわち，この条件は Stakeholder Needs を達成するためにシステムから与えられた設計要求ではないが，コンポーネントのメーカーが設計し，製造した結果の仕様書に書かれる項目である．このような仕様変更が生じた場合，仕様書に新たに加わった項目を，設計モデルに反映させる必要が生じると考えられる．結果的に，仕様書の内容を，Component Behavior や関連した設計モデルにフィードバックするような手順が加わることになると考えられる．Component Behavior からドメインモデルを自動生成できるためには，このようなフィードバックの手順を支援するツールや仕組みを構築することが必要だと考えられるが，このツールが存在すればドメインモデルの自動生成は可能であると結論付けられる．

4.4 State Monitor, Executor を構成するためのデータベース作成

前節では，MBSE の手順に則って設計モデルを構築し，Planning Engine に利用するドメインモデルを Component Behavior から自動生成できる見込みを立てた．本節では，自律プランナー上の他のモジュールである State Monitor, Executor を構成するためのデータベースを作成する方法に関して記述する．

4.4.1 Executor を構成するためのデータベース作成

Executor に必要な情報は，ドメインモデルにおけるアクション名と，Low Level Controller における具体的なコマンドの関係である．この例を表 4.5 に示す．表 4.5 においては，この関係性を csv 形式で記述しており，この csv ファイルを自律プランナーの起動時に読み込むようにしている．こうすることで，csv ファイルを入れ替えることによって軌道上での再構成をおこなうことができるようにしている．

今回，この csv ファイルにおけるコマンド側の情報は，C2A で制作されるコマンドデータベースと共通フォーマットになっている．また，ブロックコマンド（複数のコマンドの塊）の使用を許可している．これは，再構成能力の確保と，簡易的な階層化の実施のためである．結果的に，表 4.5 のデータベースはドメイン情報（action 名）と C2A におけるコマンドデータベース合成したものであるため，自動生成可能であると考えられる．

表 4.5 Executor を構成するデータベース例の抜粋

Action Name	Command Name	APID	Code	Num Para	P1_Type	P1_Value	P2_Type	P2_Value	P3_Type	P3_Value	Descriptio	Note
clt_send_data	Cmd_CLOTH	OBC	0x00A8	2	uint8_t	83	uint8_t	170			地上局から	第一引数：
clt_set_param	Cmd_TLCD_DEPLOY_BLOCK	OBC	0x0012	2	uint8_t	1	uint8_t	246			BLC展開（	240あたり
data_downlink	Cmd_TLCD_DEPLOY_BLOCK	OBC	0x0012	2	uint8_t	2	uint8_t	77			BLC展開（	77はHKダ
dlp_observe_cam	Cmd_DLP_SHIFT_SDRAM_MEMORY	OBC	0x0087	1	uint8_t	0					SDRAMの	画像保存先
dlp_observe_lif	Cmd_DLP_SHIFT_SDRAM_MEMORY	OBC	0x0087	1	uint8_t	0					SDRAMの	画像保存先
dlp_send_data_cam	Cmd_DLP_SEND_DATA_OBC	OBC	0x0086	1	uint8_t	0					SDRAMの	仮置き
dlp_send_data_lif	Cmd_DLP_SEND_DATA_OBC	OBC	0x0086	1	uint8_t	33					SDRAMの	仮置き
dlp_set_mode_cam	Cmd_DLP_SET_MODE	OBC	0x0082	1	uint8_t	1					モードを変	0:LIF, 1:As
dlp_set_mode_lif	Cmd_DLP_SET_MODE	OBC	0x0082	1	uint8_t	0					モードを変	0:LIF, 1:As
dlp_set_param_cam	Cmd_TLCD_DEPLOY_BLOCK	OBC	0x0012	2	uint8_t	1	uint8_t	242			BLC展開（	240あたり
dlp_set_param_lif	Cmd_TLCD_DEPLOY_BLOCK	OBC	0x0012	2	uint8_t	1	uint8_t	243			BLC展開（	240あたり
phx_close_shutter	Cmd_ZEUS_PHOENIX_CLOSE	OBC	0x00AE	0							PHOENIXシャッター	
phx_ena_hv	Cmd_PHOENIX_HV_ENABLE	OBC	0x0099	0							HV　ENA	
phx_hv_off	Cmd_PHOENIX_HV_OFF	OBC	0x009B	0							HV　OFF	
phx_hv_on	Cmd_PHOENIX_HV_ON	OBC	0x009A	0							HV　ON	
phx_observe	Cmd_PHOENIX_START_ACC	OBC	0x008F	1	uint8_t	3					積分スター	3に書き込
phx_open_shutter	Cmd_ZEUS_PHOENIX_OPEN	OBC	0x00AD	0							PHOENIXシャッター	
phx_send_data	Cmd_PHOENIX_READ_FLASH_MEMORY	OBC	0x00A6	1	uint8_t	3					フラッシュ	3からよみ
phx_set_hv	Cmd_PHOENIX_SET_HV_VALUE	OBC	0x0098	1	uint8_t	1					HV値セッ	安全のため
phx_set_param	Cmd_TLCD_DEPLOY_BLOCK	OBC	0x0012	2	uint8_t	1	uint8_t	240			BLC展開（	240あたり

4.4.2 State Monitor を構成するためのデータベース作成

State Monitor を構成するために必要な情報は，ドメインモデルにおける state と，Low Level Controller におけるテレメトリとの関係性である．EQUULEUS において製作した例を表 4.6 に示す．

表 4.6 State Monitor を構成するデータベース例の抜粋

ID	State Variables	type	LLC_TLM	operand	value	size	pos	TLMList	TLMGetInfo
0	sw_xact_bus	uint8_t	(uint8_t)pcu1->rx_sts.sw_status_1.	=	1	1	13	uint8_t sw_xact_	EndianConv(&tli
1	sw_xact_5v	uint8_t	(uint8_t)pcu1->rx_sts.sw_status_3.	=	1	1	14	uint8_t sw_xact_	EndianConv(&tli
2	vol_xact_bus	uint8_t	pcu1->rx_sts.xact_bus_voltage	>	20	1	15	uint8_t vol_xact_	EndianConv(&tli
3	cur_xact_bus	uint8_t	pcu1->rx_sts.xact_bus_current	>	1	1	16	uint8_t cur_xact_	EndianConv(&tli
4	vol_xact_5v	uint8_t	pcu1->rx_sts.xact_ctrl_5v_voltage	>	20	1	17	uint8_t vol_xact_	EndianConv(&tli
5	cur_xact_5v	uint8_t	pcu1->rx_sts.xact_ctrl_5v_current	>	1	1	18	uint8_t cur_xact_	EndianConv(&tli
6	sw_zeus_bus	uint8_t	(uint8_t)pcu1->rx_sts.sw_status_1.	=	1	1	19	uint8_t sw_zeus_	EndianConv(&tli
7	sw_zeus_5v	uint8_t	(uint8_t)pcu1->rx_sts.sw_status_3.	=	1	1	20	uint8_t sw_zeus_	EndianConv(&tli
8	vol_zeus_bus	uint8_t	pcu1->rx_sts.prop_bus_voltage	>	20	1	21	uint8_t vol_zeus_	EndianConv(&tli
9	cur_zeus_bus	uint8_t	pcu1->rx_sts.prop_bus_current	>	1	1	22	uint8_t cur_zeus	EndianConv(&tli
10	vol_zeus_5v	uint8_t	pcu1->rx_sts.prop_5v_voltage	>	20	1	23	uint8_t vol_zeus_	EndianConv(&tli
11	cur_zeus_5v	uint8_t	pcu1->rx_sts.prop_5v_current	>	1	1	24	uint8_t cur_zeus	EndianConv(&tli
12	sw_phx	uint8_t	(uint8_t)pcu1->rx_sts.sw_status_1.	=	1	1	25	uint8_t sw_phx;	EndianConv(&tli
13	vol_phx	uint8_t	pcu1->rx_sts.phoenix_bus_voltage	>	20	1	26	uint8_t vol_phx;	EndianConv(&tli
14	cur_phx	uint8_t	pcu1->rx_sts.phoenix_bus_current	>	1	1	27	uint8_t cur_phx;	EndianConv(&tli

表 4.6 のうち，テレメトリの型と定義（ソースコード）の列は，C2A で制作されるテレメトリデータベースと共通フォーマットになっている．EQUULEUS で構築したドメインモデルにおいては，state variable は true/false の二値のみ持つことになっているため，state variable に対応するテレメトリと比較演算子，比較する値を記述することで，true/false を判別できるようにしている．しかし，この形式で記述しようとした場合に，記述できない state variable が存在してしまうことが分かった．例えば，「姿勢収束」という state variable

を定義した場合，姿勢誤差角のノルムが十分小さいことや，姿勢系の指令トルクが外乱を打ち消す程度の大きさであることといった，複数のテレメトリ情報をもとに判断することになる．今回のデータベースフォーマットでは，state variable 一つに対してテレメトリを一つに限定した書き方になっているため,「姿勢収束」の判別式が記述できなくなってしまう．しかし，これは実際に人間が運用するうえで，判断に利用するテレメトリが複数にまたがっていることを示しており，この判断を機上で実行したほうが人間の運用が楽になると推測できる．すなわち，設計へのフィードバックとして,「姿勢収束」を示す新たなテレメトリを定義するとよいと示せたことになる. したがって, 表 4.6 のフォーマットで State Monitor の構成に必要な情報を集約できる．

以上から，State Monitor のデータベースに関しても，ドメイン情報と C2A のデータベースから生成できていることがわかる．したがって，Executor や State Monitor を構成するために必要なデータベースは，すべてドメイン情報と C2A のデータベースから生成できることが分かった．ドメイン情報は MBSE プロセスで生成したモデルから抽出できることが分かっており，C2A のデータベースは搭載ソフトウェアの設計をするうえで自然と出来上がるものであることから，各関係性をモデルに記述するだけで，このデータベースは自然と出来上がるといえる．

4.5 自動計画アルゴリズムを用いた自動計画実験

本節では，4.2 節で紹介した SMTPlan+ という Planning Engine を用いて，4.3 節で構成した宇宙機のドメインモデルの検証を行う．まず，機器単体のモデルという簡単なドメインモデルを用いて自動計画実験を行い，自動計画アルゴリズムの妥当性や，機器レベルでのドメインモデルの妥当性を確認する．続いて，複数の機器・サブシステムのドメインモデルを統合して宇宙機のドメインモデルを構成して自動計画実験を行い，実際の運用に近い問題を解くことが可能かを確認するとともに，その計算時間が許容できるかを確認する．

本節のシミュレーションは，表 4.7 に示す計算機に SMTPlan+の動作環境を構築して実施した．

表 4.7 Planning Engine の動作環境

OS	Ubuntu 16.04.5 LTS (Windows Subsystem for Linux)
CPU	Intel Core i7-7500U 2.70 GHz
RAM	16 GB

4.5.1 機器単体モデルを用いた自動計画実験

本節では，機器単体として PHOENIX(PHX)という観測機器を題材とし，自動計画の実験を行う．PHX の機器レベルでの状態遷移図は，4.3.11 項の図 4.13 に示されている．PHX は電源を ON されると，初期状態として Initial へと遷移する．PHX で撮像を行うためには，専用の高圧電源を用いる必要がある．しかし，高圧電源は真空度によっては放電の危険性があるため，何らかのミスによって高圧部が予期せぬ動作をしないように，高圧 Enable のフラグを上げるコマンドと高圧の ON コマンドを連続して受信しないと高圧電源が ON されないようになっている．また，高圧電源を ON した後も，観測するための最適な電圧に設定する必要があり，高圧部の設定以外にも露光時間・閾値設定など，様々な観測パラメータの設定を行う必要がある．高圧部分以外のパラメータ設定は，「観測パラメータの設定」という抽象的なアクションとしてモデル化し，バスシステムの担当者が理解しやすいようにしている．

この状態遷移図をもとに，ドメインモデルの pddl ファイルを生成した．生成したドメインファイルのうち，アクションの一つを以下の図 4.14 に示す．各アクションは，アクションの名前と実行に要する時間，実行条件と実行することによる効果，これらの条件を記述するために使用するパラメータから構成されている．今回の記述では，state を抽象化していないため，parameters の行は利用していない．

```
(:durative-action PHX_HV_ON↓
 :parameters ()↓
 :duration (= ?duration 1)↓
 :condition (and (over all (cur_phx)) (over all (vol_phx)) (at start (phx_hv_ena)))↓
 :effect (and (at end (phx_hv_on)) (at end (not (phx_hv_ena))) (at end (not (phx_hv_param))))↓
)↓
```

図 4.14 PHX_HV_ON のアクション記述例

このようなアクションの記述や，存在する状態一覧，ドメインの複雑度（リソースシミュレーションをするか，条件に or 文を許可するか，など）によって，ドメインの PDDL ファイルは構成される．このように生成したドメインモデルの PDDL ファイルが正しいかを，問題の PDDL ファイルを作成して確認する．特に，ノミナルの観測計画が実施できるか，異常発生時にも対処を行えるかの 2 点に関して確認を行うこととする．

4.5.1.1. 問題 1 ：ノミナルの観測計画

本問題では，作成したドメインモデルの検証を行うとともに，宇宙機搭載自律プランナーの目的の一つである「ミッション運用を行うこと」が達成できるかの確認を，簡素なドメインである機器単体で実施することを目的としている．解くべき問題は，以下の通り．

- 初期状態：PHX が OFF
- 目標：観測データ取得完了

上記問題を，SMTPlan+で解いた結果を図 4.15 に示す．

図 4.15 PHX におけるノミナル自動観測計画

図 4.15 より，図 4.13 の状態遷移図から予想されるとおりに観測計画が探索されることで分かる．そして，これは直感とも一致しており，人間が考える観測計画とも一致していることがわかる．この結果から，本研究の手法で生成されたドメインファイルと SMTPlan+を組み合わせることで，ノミナルの観測計画を導き出すことができることが分かった．つまり，機器単体レベルではあるが，本研究で構築を目指す宇宙機搭載自律プランナーの目的の一つである「ミッション運用の実施」が行えることが確認できた．

4.5.1.2. 問題 2 ：異常発生後，観測までの計画

本問題では，宇宙機搭載自律プランナーの目的の一つである「異常に対処をしたうえでミッション目的の達成を目指す」が達成できるかの確認を行う．解くべき問題は，以下の通り．

- 初期状態：PHX が過電流シャットダウンされた状態
  - スイッチ状態は ON だが，電流値・電圧値は 0 の状態
- 目標：観測データ取得完了

上記問題を，SMTPlan+ で解いた結果を図 4.16 に示す．

図 4.16 PHX が過電流シャットダウンにかかった後の自動観測計画

図 4.16 と図 4.15 を見比べると，図 4.16 に示されている結果は，図 4.15 の最初に PHX の電源 OFF の action が入っていることがわかる．すなわち，と電源 OFF を行ったうえで，本来の観測計画を実行した．この観測計画は，人間が考える（かつ，PROCYON 時代にも実行した）運用手順と一致している．したがって，異常に対処をしたうえで本来の観測を実施する，という運用手順を自動計画アルゴリズムによって導き出すことができていることが確認できた．

4.5.2 宇宙機システム全体のモデルを用いた自動計画実験

続いて，観測機器単体ではなく，複数の機器・サブシステムを組み合わせて，宇宙機システム全体に相当するようなドメインモデルを構築する．この複雑なドメインモデルを用いて，実際の宇宙機の運用に限りなく近い問題を解くことを目指す．今後，EM の実機で試験を行う際に，常温・常圧下で行ってはいけないアクションが存在する．そのため，実機で実験を行うことを念頭に置いたうえで，本研究の目的である「ミッション運用を達成できる自律機能」が構築できているかを確認するために，以下のような機器・サブシステムに関するドメインモデルを統合した．

- ドメインファイルとして統合した機能
  - 電源系
    - PCU：各機器の ON/OFF，電流，電圧
    - SADA：太陽指向の状態
  - ミッション系
    - DLP：LIF 撮影，普通のカメラの撮影のモードと，それらに対するパラメータ設定・観測実施・データ転送
    - CLT：パラメータ設定，データ転送
    - PHX：シャッター開閉，パラメータ設定，高圧，観測実施，データ転送
  - 姿勢系
    - 太陽指向・月指向・ターゲット指向・セーフモード

- CDH 系
  - OBC：メモリの占有状況
- 通信系
  - 送信 ON/OFF，リセット

4.5.2.1. 問題 3 ：複数サブシステムにまたがる観測計画

問題 1 で行った観測計画を，複数のサブシステムの情報が含まれるドメインをもとに実施する．

- 初期状態：全機器 OFF（衛星起動直後と同等）
- 目標：PHX 観測データ取得完了

上記問題を，SMTPlan+ で解いた結果を図 4.17 に示す．また，図 4.17 の結果をタイムチャートとして示したものを表 4.8 に示す．

```
0.0:    (turnon_xact_5v) [1.0]
0.0:    (turnon_phx) [1.0]
1.0:    (turnon_xact_bus) [1.0]
2.0:    (turnon_zeus_5v) [1.0]
2.0:    (turnon_sada) [1.0]
2.0:    (phx_ena_hv) [1.0]
3.0:    (xact_set_att_target) [10.0]
12.0:   (phx_hv_on) [1.0]
13.0:   (sada_rotate2sun) [3.0]
13.0:   (phx_open_shutter) [4.0]
16.0:   (phx_set_hv) [1.0]
16.0:   (phx_set_param) [1.0]
19.0:   (phx_observe) [15.0]
Solved 10:      0.875000 seconds
Total time:     1.875000 seconds
nakajima-ubuntu16@NKJMNOTE:~/ex/equ$
```

図 4.17 EQUULEUS における，初期状態から PHX 観測までの自動計画結果

表 4.8 問題 3 の計画結果タイムチャート

t[s]	Thread1	Thread2	Thread3
0	turnon_xact_5v	turnon_phx	
1	turnon_xact_bus		
2	turnon_sada	phx_ena_hv	turnon_zeus_5v
3	xact_set_att_target		
4			
~			
~			
11			
12		phx_hv_on	
13	sada_rotate2sun	phx_open_shutter	
14			
15			
16	phx_set_param		phx_set_hv
17			
18			
19	phx_observe		
~			
~			
33			
34			

図 4.17 の結果を解釈しやすくすると，以下のようになる．

① 時刻 0.0 [s] に，姿勢決定・制御統合機器（XACT）の 5V 系の電源を ON にする．5V 系は，姿勢決定系と内蔵計算機の電源電圧である．
② 時刻 0.0 [s] に，観測機器（PHX）の電源を ON にする．
③ 時刻 1.0 [s] に，XACT のバス電圧系の電源を ON にする．バス電圧系は，リアクションホイールの駆動電源である．
④ 時刻 2.0 [s] に，推進系制御基板（ZEUS）の 5V 系の電源を ON にする．この電源は，ZEUS 上のマイコンや，PHX のシャッター駆動回路に利用される電源である．
⑤ 時刻 2.0 [s] に，太陽電池パドルを回転させるモーター駆動回路（SADA）の電源を ON にする．
⑥ 時刻 2.0 [s] に，PHX の高圧電源を Enable にする．PHX の高圧電源は観測時に利用されるが，安全のため，高圧電源 Enable コマンドを受け取った後に高圧電源 ON コマンドを受信する必要がある．
⑦ 時刻 3.0 [s] に，XACT を利用して地球観測姿勢に遷移を開始する．
⑧ 時刻 12.0 [s] に，PHX の高圧電源を ON する．
⑨ 時刻 13.0 [s] に，SADA を利用して太陽電池パドルの回転を開始し，太陽電池が太陽に正対するようにする．
⑩ 時刻 13.0 [s] に，PHX のシャッターを開き始める．
⑪ 時刻 16.0 [s] に，PHX の高圧電源の電圧値を設定する．
⑫ 時刻 16.0 [s] に，PHX の観測関連のパラメータ設定をする．具体的には，露光時間やゲインなどを想定している．
⑬ 時刻 19.0 [s] に，観測を開始する．

この結果は，人間が考える，効率的な観測計画に近い出力結果である，具体的には，時間がかかる姿勢制御を先に実施し，並行して高圧電源の準備作業を開始し，姿勢制御が完了した段階で発電量の確保・シャッター開放という「機体固定座標における太陽方向が確定しないと実施できない運用項目」を実施し，その裏で並行して観測パラメータ設定を行い，観測を開始する，という手順になった．

4.5.2.2. 問題 4：複数サブシステムにまたがる，異常発生後の観測計画

問題 2 で行った観測計画を，複数のサブシステムの情報が含まれるドメインをもとに実施する．

- 初期状態：PHX 過電流シャットダウン状態
- 目標：PHX 観測データ取得完了

上記問題を，SMTPlan+ で解いた結果を図 4.18 に示す．

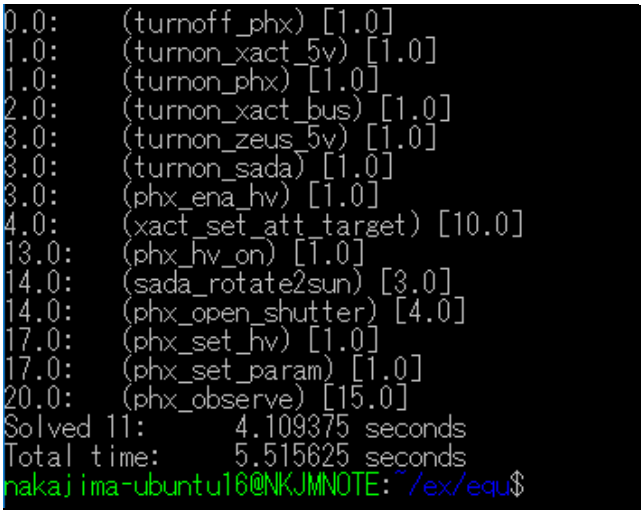

図 4.18 EQUULEUS における，PHX 過電流シャットダウン後から PHX 観測までの自動計画結果

図 4.18 の結果は，図 4.17 の結果の最初に「PHX 電源 OFF」が加わったものである．すなわち，電源 OFF を行ったうえで，本来の観測計画を実行したという結果になった．これは，問題 2 の結果と同様に．過電流シャットダウンに正しく対処を行ったうえで観測を行うという計画結果が得られた，また，この問題は，探索した方策が 14 個の action から成り立つというやや複雑な問題であったが，計算時間は数秒オーダーに収まっており，計算リソースがより制限される宇宙機搭載計算機でも実時間での計算が可能だと考えられる．

4.5.2.3. 他の観測機器を用いた観測計画

ここまで，観測機器として PHX に関する自動計画問題を解いてきた．ドメインに組み込んだ他の機器モデルの確認を行うために，別の観測機器である DELPHINUS（DLP）で月面衝突閃光を観測する自動計画問題を解くことにする．具体的な問題設定は以下の通り．

- 初期状態：全機器 OFF（衛星起動直後と同等）
- 目標：DLP で，Lunar Impact Flash 観測データ取得完了

上記問題を，SMTPlan+ で解いた結果を図 4.19 に示す．また，図 4.19 の結果をタイムチャートとして示したものを表 4.9 に示す．

```
0.0:    (turnon_xact_5v) [1.0]
0.0:    (turnon_dlp) [1.0]
1.0:    (turnon_xact_bus) [1.0]
2.0:    (turnon_sada) [1.0]
2.0:    (dlp_set_mode_lif) [1.0]
3.0:    (xact_set_att_moon) [10.0]
12.0:   (dlp_set_param_lif) [1.0]
13.0:   (sada_rotate2sun) [3.0]
18.0:   (dlp_observe_lif) [5.0]
Solved 9:       0.406250 seconds
Total time:     1.156250 seconds
nakajima-ubuntu16@NKJMNOTE:~/ex/equ$
```

図 4.19 EQUULEUS における，初期状態から LIF 観測までの自動計画結果

図 4.19 の結果を解釈しやすくすると，以下のようになる．

① 時刻 0.0 [s] に，姿勢決定・制御統合機器（XACT）の 5V 系の電源を ON にする．5V 系は，姿勢決定系と内蔵計算機の電源電圧である．
② 時刻 0.0 [s] に，観測機器（DLP）の電源を ON にする．
③ 時刻 1.0 [s] に，XACT のバス電圧系の電源を ON にする..
④ 時刻 2.0 [s] に，太陽電池パドルを回転させるモーター駆動回路（SADA）の電源を ON にする．
⑤ 時刻 2.0 [s] に，DLP の観測モードを月面衝突閃光観測モード（LIF モード）に設定する．
⑥ 時刻 3.0 [s] に，XACT を利用して月観測姿勢に遷移を開始する．
⑦ 時刻 12.0 [s] に，DLP の観測パラメータを月面衝突閃光に合わせた値に設定する．
⑧ 時刻 13.0 [s] に，SADA を利用して太陽電池パドルの回転を開始し，太陽電池が太陽に正対するようにする．
⑨ 時刻 18.0 [s] に，観測を開始する．

この結果から，DLP に関しても，問題なく人間による観測計画と同様の結果が得られることが分かった．

表 4.9 問題 5 の計画結果タイムチャート

t[s]	Thread1	Thread2
0	turnon_xact_5v	turnon_dlp
1	turnon_xact_bus	
2	turnon_sada	dlp_set_mode_lif
3	xact_set_att_moon	
4		
~		
~		
11		
12		dlp_set_param_lif
13	sada_rotate2sun	
14		
15		
16		
17		
18	dlp_observe_lif	
19		
~		
~		
22		
23		

4.6 EQUULEUS 搭載計算機を用いた自動計画実験

本節では，これまで述べてきた宇宙機搭載自律プランナーに関して，EQUULEUS に搭載される計算機（OBC）の実機を用いて行った実験の結果を紹介する．まず，実機を用いた試験の目的を整理していく．前節まで，Planning Engine を用いてドメインモデルの検証を行った．その一方，自律プランナー上の他のモジュールである State Monitor, Executor の検証を実施できていない．そのため，これら 2 モジュールの機能確認を実施する．Executor に関しては，Planning Engine が導き出した方策（Plan）を Executor が読み出す部分や，読み出した方策を具体的なコマンドに変換する部分の検証を実施する．State Monitor に関しては，ドメインモデルの情報と Plan の情報を併せることにより，宇宙機がどの時刻にどんな状態であれば正常化のシミュレーションを行う部分や，このシミュレーション結果とテレメトリ情報を照らし合わせて異常を検知する部分，異常を検知した際に異常に対処をしたうえで当初の目的を達成できるような方策を探索するための再計画を実施するための機能などの確認を行う．この再計画機能の確認を行うために，OBC 上のメモリを書き換えて故障を模擬したテレメトリデータを生成する．また，ハードウェア依存部分の確認として，エンディアンの影響やシリアル通信機能など，シミュレータのみでは確認できない部分の確認を実施する．

4.6.1 試験コンフィグレーション

EQUULEUS の OBC EM を用いた試験時の写真を図 4.20 に示す．この試験時の物理的な試験コンフィギュレーションを図 4.21 に，ソフトウェア的な試験コンフィギュレーションを図 4.22 にそれぞれ示す．Raspberry Pi と OBC は，RS422-UART によるシリアル通信でデータの共有を行う．OBC は，Raspberry Pi と通信するのとは異なる RS422-UART の通信ポートを経由し，模擬地上局とテレメトリ・コマンドの通信を行う．PC と Raspberry Pi は Ethernet 経由で接続され，自律プランナーの実行状況が出力される．

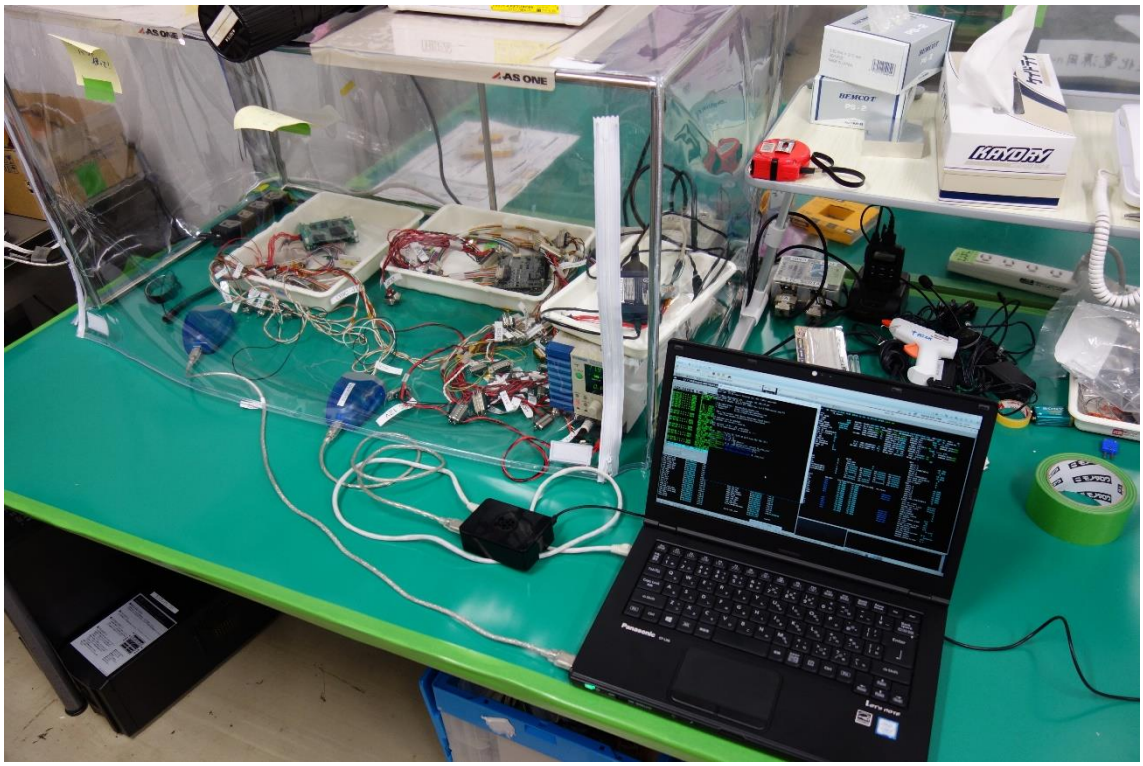

図 4.20 OBC EM 試験時の写真

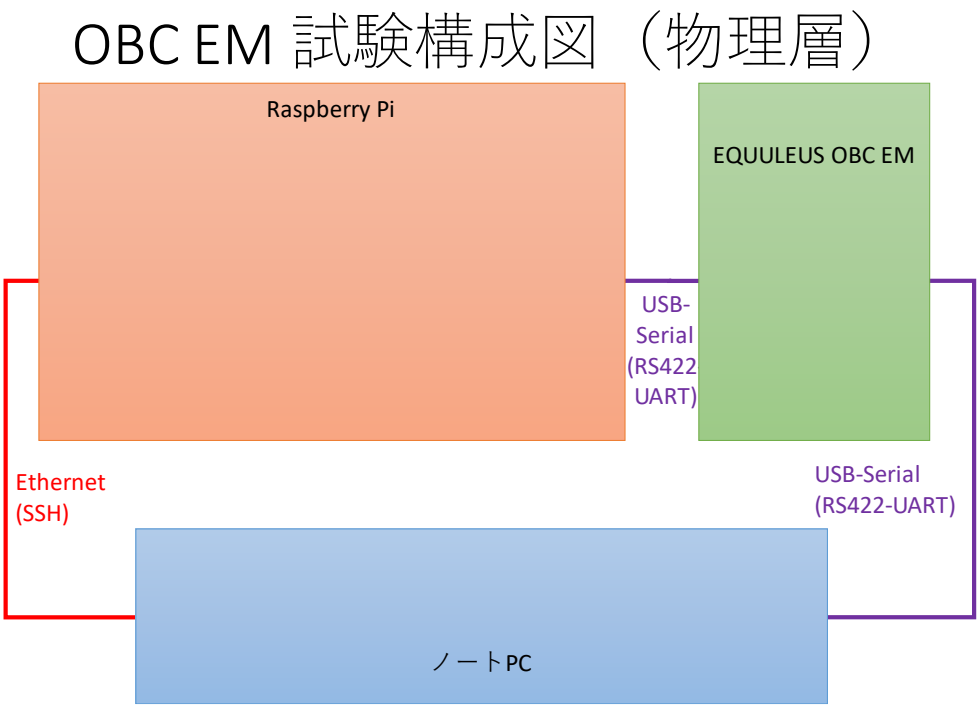

図 4.21 OBC EM 試験の物理層ブロック図

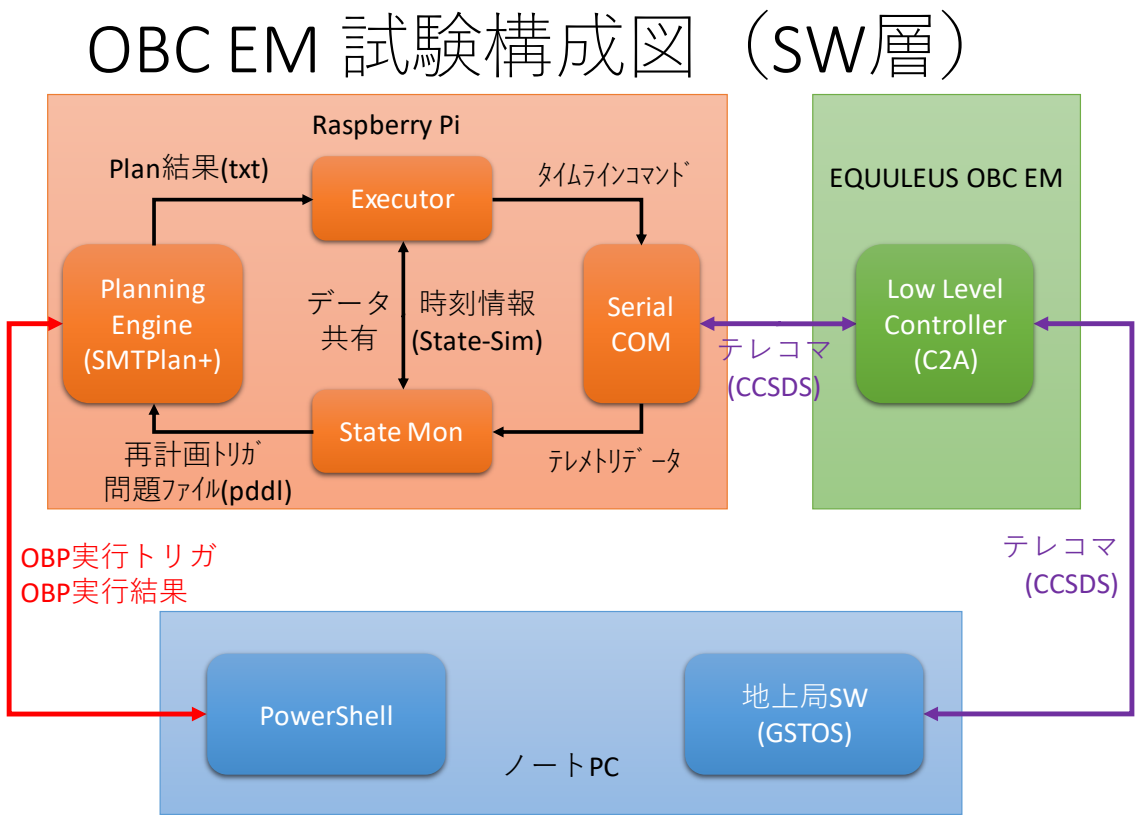

図 4.22 OBC EM 試験のソフトウェア層ブロック図

4.6.2 実験結果

自律プランナーの動作順に従って，各機能の確認結果を説明していく．まず，Planning Engine を用いて問題 3 を解き，計画結果のテキストファイルを生成しておく．このテキストファイルを Executor が読み出し，Low Level Controller が解釈できるタイムラインコマンドへの変換を実施し，Low Level Controller に送信する．図 4.23 に，Low Level Controller が保持したタイムラインコマンドのリストを模擬地上局にダウンリンクした結果を示す．今回，Plan 上の時刻 0 が OBC 時刻の 2000 に相当するようにオフセットをかけている．また，コマンドの実行時刻が重複すると Low Level Controller が異常判定をしてコマンドを受信しないため，100ms ずつ実行時刻をずらしている．図 4.23 と図 4.17 より，図 4.17 で導かれた方策の各 action がタイムラインコマンドに変換され，Low Level Controller に保持されていることが確認できた．また，電源系制御基板関連のコマンドに関しては，実機で正しく実行されることも確認できている．

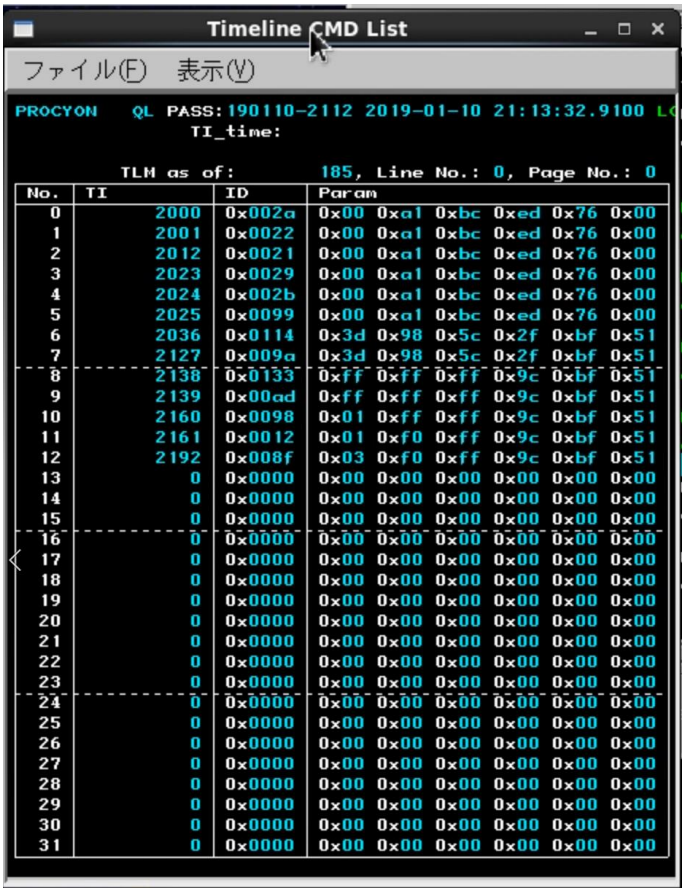

図 4.23 Low Level Controller による Plan の読み込み結果

続いて，State Monitor の動作を確認していく．まず，異常検知を行うためには，Plan のシミュレーションを行い，どの時刻に・どんな状態にあることが正常か，という情報を作成する必要がある．この情報は，Plan において前提としている初期状態から，各アクションを指定された時刻に実施した場合に，満たすべき条件と，アクションによってもたらされる効果の両方を考慮することによって導き出すことができる．本論文において EQUULEUS での実験のために実装した State Monitor においては，今回のドメインモデルにおいて各 state variable が true/false の二値しか持たないことを利用し，各アクションの開始時刻・終了時刻において各 state variable が二値のうちどちらの値を持つか，どう変化するか，その時刻に考慮すべきか，という情報を表形式で保持することにしている．この表の情報をもとに，与えられた OBC 時刻において正しいはずの state variable を導くことになる．さらに，表 4.6 のような State Monitor を構成するデータベースをもとに，Low Level Controller のテレメトリ情報とドメインモデルにおける state の変換を行って，Plan のシミュレーション結果と現在の状態の比較を実施し，異常が発生しているか否かの判別を行っている．図 4.24 に判別結果が正常であった際の自律プランナーの出力結果を示し，図 4.25 に異常が発生し

ていた場合における自律プランナーの出力結果を示している.

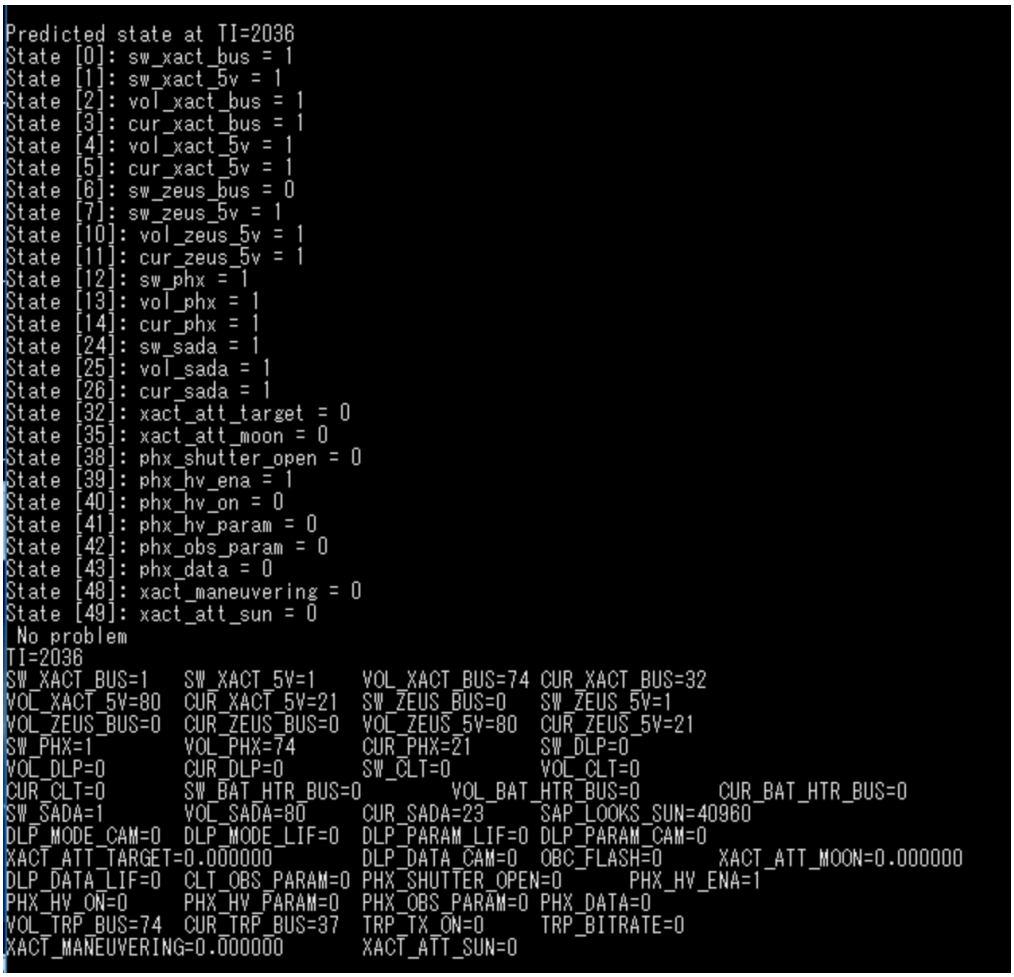

図 4.24 実際のテレメトリ情報と，方策による予想状態の照合結果（正常時）

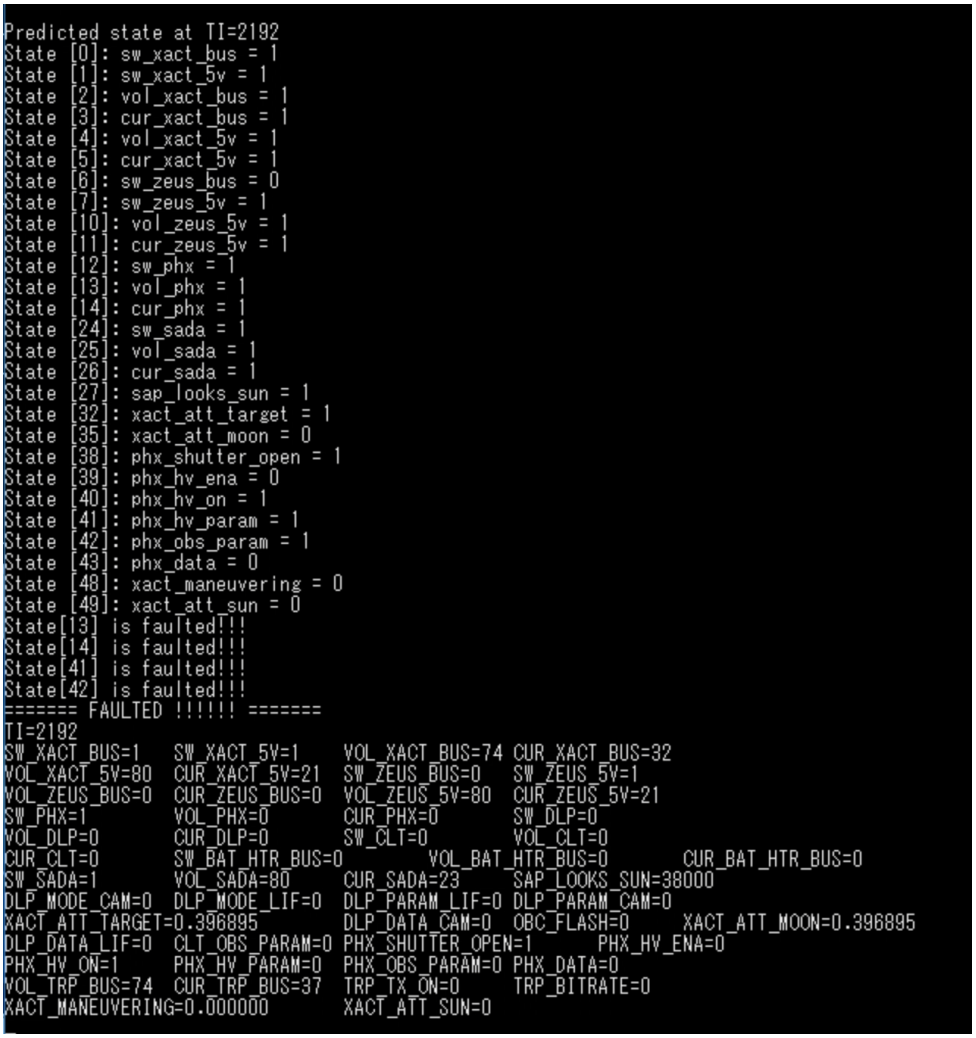

図 4.25 実際のテレメトリ情報と，方策による予想状態の照合結果（異常発生時）

最後に，異常が発生していた際の対応に関して確認を行う．図 4.25 のように異常を検知した際に，State Monitor は異常への対処を行ったうえで当初の目標を達成できる方策を探索するための計画問題を作成する．すなわち，「異常が発生している現在の状態」を初期状態とし，「当初から与えられている目標」をそのまま目標とした，PDDL の問題ファイルを作成する．この問題ファイルを Planning Engine に渡し，最初の計画を探索した際と同じドメインモデルを用いて計画問題を解く．こうして導かれた方策が，現在発生している異常に対処を行ったうえで当初の運用目標を達成できるものとなる．実際に，再計画を行った際の自律プランナーの出力結果を図 4.26 に示す．今回の実験では，問題 3 を解いた結果の方策のうち，phx_set_hv というアクションを実施した際に異常が発生したことを模擬している．具体的には，phx_hv_set を実行した直後に，観測機器である PHX に過電流が一時的に流れ，過電流シャットダウンされたという事象を模擬している．図 4.17 を見ると，phx_set_hv 以降に実行されるはずだった方策はすべて PHX に関するものであることがわかる．結果的

に，phx_hv_set 実行時に過電流シャットダウンが発生したあと PHX での観測を行うまでの計画問題は，問題 2 と一致することがわかる．図 4.16 と図 4.25 の出力結果を比較すると，導き出された方策が一致していることがわかる．このことから，異常に対処を行ったうえで目標を達成するための自動計画問題が正しく生成されていることが確認できた．あとは，導き出された方策を再度 Executor に渡すことで，本項の最初に述べたのと同様の手順で方策が実行されていくことになる．

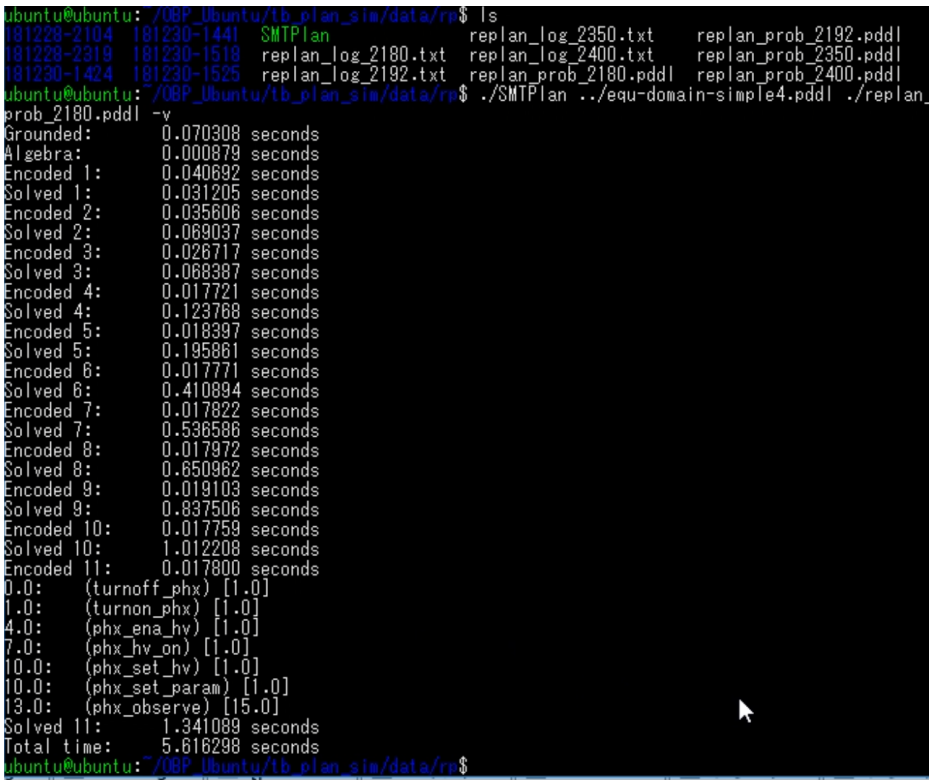

図 4.26 故障に対応する再計画結果

以上までの手順によって確認できた結果をまとめると，以下のとおり．

- Executor
  - Planning Engine が出力した Plan を読み込み，タイムラインコマンドの列に変換することを確認した．
  - 生成したタイムラインコマンド列を，Low Level Controller で正しく実行できることを確認した．
- State Monitor
  - ドメインと問題の PDDL ファイル，Planning Engine が出力した Plan から，state がどのように変遷するかのシミュレーションを行えることを確認した．

- state が正しく変遷していない場合に，現在の state から本来の Goal を達成するための問題を作成できることを確認した．

上記で確認できた内容より，MBSE で生成したモデルから生成した PDDL ファイル・データベースをもとに，自律プランナーが生成できることが確認できた．

4.7 本章のまとめ

本章では，第 3 章で提案したプロセスを EQUULEUS という宇宙機に適用し，実際の宇宙機搭載計算機で実験を行った．MBSE を進めていく上での Methodology の例として MagicGrid を採用し，ダイアグラムの生成を行った．コンポーネントの状態遷移図に，状態遷移の詳細な条件という付加情報を加えることで，状態遷移図からコンポーネントのドメインモデルを構成できることを確認した．続いて，各コンポーネントのドメインモデルを統合して宇宙機のドメインモデルを構成し，自動計画実験を行ったところ，抽象的なミッション目的を達成する方策を自動で探索できるだけでなく，簡易な故障に対しても対処を行えることを確認できた．最後に，実際の宇宙機に搭載する計算機を利用して自律プランナー全体の検証を行い，設計モデルをもとに構成した自律プランナーによって，「異常検知・対処」「ミッション運用」の自動化が行えることを確認した．

第5章結論

5.1 本研究の適用範囲

本節では，本研究の位置づけと適用できる範囲を示し，今後の発展につなげていくための情報を整理していく．まず，本研究は，宇宙機の運用の一部を機上で自動化するために，自動計画アルゴリズムを用いた自律プランナーを構築することを目指していた．そのうえで，モデルの構築・検証に関する課題を解決するために，MBSE を用いて宇宙機の正確な設計情報を構築し，構築した設計モデルから自律プランナーに用いるドメインモデルを生成する，という手順を示した．すなわち，本研究は「宇宙機の設計を本来やるべき手順である MBSE に沿って実施することで，検証された宇宙機の設計情報を構築でき，さらに自然と自律プランナーを構築できる」という枠組みを示し，この手法を宇宙機に適用したものであるといえる．本研究で提案している手法は宇宙機特有の手順は多くないため，宇宙機ではない他のシステムでも同様に適用できると考えられる．

本研究で構築する自律プランナーは，モデルベースの自動計画アルゴリズムが動作している．また，利用するモデルは宇宙機の設計段階で構築していき，試験を進めながら洗練化させていくものであるが，軌道上で動的にモデルの変更を行うことは想定していない．これは，動的なモデル変更を実施するとその検証が困難になることが主因となっている．その結果，本研究で構築する自律プランナーは，検証された設計情報に基づいたドメインモデルに記述された挙動の通りに動作するため，予期せぬ行動を起こしづらいということになる．この点は，システムの安全性という観点からみると，暴走しづらい自律システムであると評価できる．その一方でこの安全性は，機器故障などの影響によって地上試験時と軌道上で宇宙機の動作モデルが変わってしまった場合に，宇宙機の自律プランナーで対応できなくなってしまうという欠点にもなる．

動的なモデル変更に対応していないことに加えて，本研究で構築した自律プランナーはモデルベースであるため，モデルに記述できる内容に依存して自律プランナーで対応できる問題の複雑度が変わってしまう．例えば，第 4 章で示した EQUULEUS での例においては，宇宙機のドメインモデルを PDDL 2.1 で記述していた．そのため，action の結果が複数の可能性に分かれるような不確定性を持つドメインモデルを扱うことができていない．また，宇宙機のドメインモデルは宇宙機に搭載されている機器の機能を示す状態遷移図ベースで構成されているため，機器に依存しないような複雑な物理モデルや，複数のモーダルにまたがるようなモデルを組み込むことができていない．熱ひずみによって光学系観測機器のミスアラインメントが変わってしまうことや，機器の電源の ON にすることで周囲の温度が変わることなどは，機器やサブシステムの状態遷移図に記述されるような内容ではない．しかし，人間の運用では，このような複雑なモデルを考慮しながら運用計画を立てることがあるため，複数のモーダルにまたがるような複雑なモデルを組み込むことが可能に

なると，より高度で柔軟性を持った自律機能を構築できると期待できる．

5.2 副次的な効果

本研究で提案した手法を実際の宇宙機に適用した結果，様々な副次的な効果を得ることができた．それらに関して，簡単に紹介を行う．

5.2.1 設計との相互作用

まずは，本研究の手順で自律プランナーを構築することにより，設計へのフィードバックが行えることが副次的効果の一つとしてあげられる．もちろん，MBSE のプロセスを実施することによる効果もあるが，4.4.2 項では，State Monitor を構成する際のデータベースを作成する際に，運用に便利なテレメトリが足りないことを指摘できる例を示した．同様に，Executor のデータベースを作成する際に，マクロ化すべきコマンド群を指摘できることも考えられる．このように，Low Level Controller のテレメトリ・コマンドというソフトウェア設計へのフィードバックを行うことが可能になっている．加えて，本論文で示した EQUULEUS の例では，Low Level Controller と他のモジュールを別の計算機に搭載している．モジュール間での情報共有のために，二つの計算機間で通信を行い，テレメトリデータ・コマンドデータによる情報共有を行う必要性がある．この情報共有で用いるテレメトリデータは自律プランナーが動作するために必要な情報になるが，このテレメトリ情報は運用に必要な情報を凝縮したデータであると考えられる．これはすなわち，ハウスキーピング（HK）データと同様の情報になる．この情報共有で用いるテレメトリデータは State Monitor を構成する際の変換データベースによって定義されている．つまり，State Monitor を構成する際のデータベースを作成することで，自然と HK データのパケット構成が導かれるといえる．なお，この HK データは生存確認に特化した目的の HK データではなく，ミッションの実施を目的とした HK データとなる．

また，自律プランナーを構築した後，様々なパターンで自律プランナーを動作させることによって設計へのフィードバックを導きだすこともできる．特に，様々な異常を模擬してシミュレーションを実施し，対応可能な異常と対応不可能な異常を判別することができる．対応不可能な異常を発見できると，その異常の発生確率や影響，対応コストを検討したうえで，機器冗長や機能冗長などといったハードウェアおよびソフトウェア設計へのフィードバックを行うことが可能となる．

5.2.2 人間による運用手順の洗練化

設計との相互作用とは別の副次的な効果としては，人間が行う運用手順の洗練化が挙げられる．本研究で自律プランナーを構築したモチベーションは運用の一部を自動化することであったため，人間の運用手順に立ち戻っていることが本末転倒に感じられるだろう．しかし，自律機能の利用が大きく広がるまでは，人間による運用を基本としつつ，その補助として自律機能を利用する，といった宇宙機の運用スタイルがとられることになると考えている．言い換えると，人間の運用が介在しないような宇宙機システムが広がるまでの間は，その機会は減れども人間が運用する機会はなくならないだろう，ということである．人間による運用がなくならないとしても，自律プランナーは運用のコストを減らすことに貢献できる．加えて，自律プランナーは目的関数を最適化するような運用計画を計算機が探索するため，人間が運用手順をすべて設計する必要性はなく，少なくとも初案を計算機が導き出すことが可能である．本研究で示した例でも，時間最適な観測計画が導かれており，導かれた観測計画をそのまま利用することも可能であるし，この観測計画をベースとして議論を重ね，運用手順の洗練化を行うこともできる．また，本研究の自律プランナーによって異常に対応するコンティンジェンシープランも探索することができる．このように，運用手順を作成するうえでは，自律プランナーを利用することが有効であると考えられる．

5.3 今後の課題と発展性

本研究では，設計から自律プランナーを構築するプロセスの提案を行ったが，このプロセスに改善を加えることにより，本研究のさらなる発展を見込むことができる．本節では，プロセスの改善例と，それによる発展性に関して紹介する．

5.3.1 複雑な計画問題への対応

2.3.2 項においては，宇宙機の自動計画問題は PDDL 2.1 以上のプランニング問題記述言語を用いればよいとしていた．これは，宇宙機の計算リソースが限られていることを考慮して，宇宙機の運用計画問題を記述するために最低限必要な内容が記述できるような記述方式を利用していたためであり，例えば一つの action に対して複数の出力結果があるような不確定性を持つ計画問題などを扱うことはできない．そのため，PDDL 2.1 では記述できないような，より複雑な問題に対応できるようにすることが課題の一つであると考えられる．

この解決方法としては，より新しいバージョンの PDDL によってドメインモデルと計画問題を記述するだけでなく，利用したバージョンの PDDL を扱えるような自動計画アルゴリズムを採用すればよいと考えらえる．新しいバージョンの PDDL を利用するだけでなく，

派生形の PDDL を利用することでも，より複雑な問題を圧開けるようになる．具体例としては，Probabilistic PDDL (PPDDL) [64] を利用することで前述した不確定性を扱えることになる．近年，International Planning Competition (IPC) [65] が二年に一回開催されており，より新しいバージョンの PDDL や派生形の PDDL を扱えるようなアルゴリズムの開発が競って行われている．このようなコミュニティで改良されるプランニング問題記述言語と自動計画アルゴリズムを用いることで，さらに複雑な問題を扱うことができると期待される．

5.3.2 打ち上げ後のドメインモデル差し替え

5.1 節の議論の中で，軌道上で動的にモデルの変更を行うことは想定していないとしていた．これを実施できるようにすることが，さらに高度な自律機能を実現するための課題であるといえる．ドメインモデルを変更する手法として，最も簡易なものは人間によって差し替えを実施することである．まず，ドメインモデルは PDDL ファイルで記述されている．ドメインモデルの PDDL ファイルに関する具体的な記述内容は付録として示しているが，ここで重要なのは PDDL ファイルがテキストファイル形式であることである．すなわち，Planning Engine の input となる PDDL ファイルを手動で差し替えてやることが可能であり，これによってドメインモデルを変更することができるということである．ドメインモデルを変更した場合，Executor や State Monitor を構成するデータベースにも変更が生じる場合がある．本論文で示した例における実装では，このデータベースもテキストファイルベースの形式にしており，データベースの差し替えを想定している．したがって，ドメインモデルの PDDL ファイルを差し替えることに加えて，これらのデータベースも手動で差し替えることにより，軌道上で動作モデルの変化があった場合でも対応できると考えられる．

ここまで人間の手動によるモデル変更方法に関して議論を行ったが，宇宙機の自律プランナー上でモデルを動的に変更することも望まれる．自律的なモデル差し替えに関する例として，強化学習と同様の機能を追加することが挙げられる．自律プランナーの State Monitor では方策のシミュレーションを実施しているため，方策内のどのアクションが失敗したかを検知することができる．そこで，失敗したアクションを選びづらくするためにアクションの実行コストを上昇させることで，永久故障した機器を何度も動作させようとする，といった状況から自動で脱することが可能になると期待できる．

5.3.3 方策の柔軟性確保

本研究の自律プランナーによって導かれる方策は，一本道の方策となっている．そのため，異常が起きた際は方策内で対処を実施するのではなく，State Monitor が異常を検知して元の方策の実行を停止したうえで，異常に対処できる新たな方策を作成する，ということになっている．この方式では，方策を実行する際に条件分岐を織り込むことができず，条件分岐のうちの一通りにしか対応できないため，柔軟性に欠ける側面がある．そのため，方策を実行する際に条件分岐を織り込むことで，より高度な自律機能の実現が可能であると期待できる．

このような条件分岐を実現する一つの手段として，宇宙機の運用をコマンドベースではなくスクリプトベースにすることが挙げられる．コマンドベースでの運用は，指定された時刻に指定された処理が行われるだけの，前述した一本道の運用になっている．これを抽象的なスクリプトベースにすれば，スクリプト内に様々な条件分岐を織り込むことが可能になる．スクリプトエンジンを用いた宇宙機の例として，DESTINY における自律運用実験[22]が挙げられる．軌道上での実験は実現しなかったものの，javascript を利用したスクリプトエンジンが搭載されており，軌道上での動的計画変更を実現できるような搭載ソフトウェアフレームワークの実現を目指していた．このように，スクリプトエンジンを搭載することにより，条件分岐を含んだ方策の実行が可能になると期待できる．ただし，スクリプトエンジンを搭載する場合，Low Level Controller のフレームワークが変わることになるため，本研究で示した手法に変更が生じる可能性があることに留意をしておきたい．

5.3.4 復元が難しい異常や，解がない計画問題に対する対処

軌道上では，機器が一時的に使えなくなるような異常だけではなく，永久故障のような異常も発生する可能性がある．永久故障のように復元が難しい異常が発生した場合，それまで利用していた宇宙機の動作モデルでは目的を達成できなくなってしまう可能性がある．また，動作モデルの変化や問題の複雑度といった理由より，計画問題の解が実時間で導けない場合も考えられる．こういった場合への対処を行うことも，自律プランナーを実用化するうえでは重要な課題である．

動作モデルの変更に関しては，5.3.2 項で述べたようにドメインモデルを差し替えることで対処が行えると考えられる．解が実時間で導けない場合への対症療法としては，計算時間に制限をかけて，計算が完了しない場合は現在の状態を保つか宇宙機が生存できるノミナル状態に遷移する，といったものが挙げられる．解が実時間で導けるが，成功しない方策を何度も試してしまうような場合に関しても，再計画の回数に制限を与え，制限回数を超えた場合は現在の状態を保つか宇宙機が生存できるノミナル状態に遷移する，といった対処が行えると考えられる．

5.3.5 本研究で利用する MBSE プロセスの改善

本研究では，MBSE のプロセスを手動で実施することを想定しており，支援ツールを利用することは想定していなかった．そのため，action の付加情報を手動で Component Behavior に追加する必要が生じていた．SysML によるダイアグラムの作成を支援するツールを用いることにより，action の付加情報を自動で追加できる可能性が 4.3.14 項で示唆されている．また，本研究で構築したドメインモデルでは，state は連続量ではなく離散値になっている．これは，Component Behavior の状態遷移図からドメインモデルを構築していることが主因になっていると考えられる．そのため，state が連続量も扱えるようにするためには，MBSE のプロセスを改善・拡張する必要があると考えられる．

また，本研究では MBSE のガイドラインとして，MagicGrid を利用していた．MagicGrid は一般的な MBSE におけるガイドラインであり，様々なシステムに利用できる反面プロセスの最適化は行われていない．そのため， CubeSat MBSE Reference Model [55]やなどの iSAT での例[70]のように，MagicGrid ではなく宇宙機特有の MBSE プロセスを利用することで，ドメインモデルの記述内容をより宇宙機の自律プランナーに適したものにできると考えられる．ただし，宇宙機特有の MBSE プロセスの本研究への適用は，宇宙機以外のシステムへの適用が難しくなることの裏返しでもあることを考慮する必要がある．

本論文で示した例では，一般的な MBSE のプロセスをすべて実施することを想定している．しかし，特に大学で開発するような超小型衛星を考えると，スケジュールと人的リソースが厳しく制限されている場合が多く，MBSE のプロセスをすべて通すことが難しい可能性がある．そのため，衛星の複雑度・スケジュールに応じた MBSE プロセスの最適化を行い，最適化された MBSE プロセスにおいても本研究の手法が適用できることを示すことにより，より多くの宇宙機で本研究の手法が適用できると考えられる．

5.3.6 本研究を衛星の初期設計から適用

本論文では，実際の宇宙機に適用した例として EQUULEUS を用いた場合の結果を示している．4.1 節で述べた通り，EQUULEUS は本研究を適用した時点で設計がほぼ固まっていただけでなく，エンジニアリングモデルを用いた試験も一通り終了していた．そのため，MBSE のプロセスを設計当初から適用したわけではなく，実際に適用したかのようなシミュレーションを行ったのみになっていた．そのため，MBSE のプロセスを通した設計と検証のイタレーション回数は少なくなっていることに加えて，Component Behavior の定義を作成する際に設計者の事前知識・ノウハウが反映されてしまっている．そのため，新しい人工衛星・宇宙機に本研究のプロセスを適用した場合と EQUULEUS での実証例では，様々な面で違いが生じることが考えられる．そのため，新しい人工衛星に本研究のプロセスを適用し，プロセスの修正・洗練化を実施することが望ましいと考えられる．

5.4 本研究の成果

今後，特に超小型衛星の利用がさらに広がることにより，地球周回・深宇宙問わず，軌道上の衛星数が膨大になることが想定されている．そのため，宇宙機の運用コストの増大や，深宇宙用の地上局の利用性の問題など，様々な問題が発生することが考えられる．このような問題を解決するために，本研究では運用を宇宙機上で自動化するというアプローチをとり，自動計画アルゴリズムを用いた自律プランナーの構築を行った．モデルベースの自律プランナーを構築するうえでの従来の課題として，モデル構築の属人性と検証の難しさが挙げられる．これを解決する手法を提案するために，宇宙機搭載の自律プランナーに必要とされる機能と情報・モデルを整理した．そのうえで，モデルベースシステムズエンジニアリングを用いて，設計情報を検証したうえでモデル化することにより，設計情報から自律プランナーで利用するモデルを構築する手法を提案した．主に機器の状態遷移モデルを利用することで，自律プランナーの自動計画アルゴリズムで用いるドメインモデルが生成できることを示した．この手法を実際の宇宙機である EQUULEUS に適用し，自律プランナーの構築手法の実証を行った．

このように，本研究はモデルベースシステムズエンジニアリングを用いて宇宙機の設計を実施することにより，自然と自律プランナーが構築できるという全体の枠組みを示したものである．また，構築された自律プランナーは，検証された設計情報から生成したドメインモデルに基づいているため，堅牢な動作をするものとなっており，非修理系である宇宙機においても利用しやすいと考えている．

本研究の手法は，宇宙機の軌道や大きさに依らず適用可能なものであり，宇宙機に自律機能を搭載するうえで重要な課題である信頼性の問題を解決するものであると考えられる．本研究の成果は，宇宙機に自動計画アルゴリズムを用いた自律プランナーを搭載するための標準となりうる手法を示したものであり，高度な自律機能を搭載した宇宙機を実現するための敷居を大きく下げることにつながるものであるといえる．

補遺A. SMTPlan+

A.1. SMTPlan+ の概要

SMTPlan+ [49] は，Kings College London の研究グループによって提案された PDDL+[66] に対応した Planner である．PDDL+ は，離散量と連続量のハイブリッドダイナミクスをモデル化するために，PDDL 2.1 から拡張されたプランニング記述言語である．PDDL+では，process や event といった要素を導入することにより，連続的な状態量の変化を柔軟に記述できるようにしている．SMTPlan 以前の Planner では，PDDL＋による拡張の一部しか対応していないことがほとんどであった．例えば，process が線形変化のみに限定されていたり，event を扱うことができなかったり，といった制限があった．SMTPlan+ では，PDDL+を標準的な SMT(Satisfiability Modulo Theory)に変換し，theory of quantifier-free nonlinear arithmetic (QF_NRA)を用いて解いている．すなわち，複雑な PDDL モデルをうまく標準的なソルバが存在する SMT にエンコーディングし，ソルバで解いている．

SMT は，背景理論付き SAT(SATisfiability Problem)と呼ばれる．SAT は論理式の充足可能性問題であり，「論理変数を含む命題論理式が与えられたとき，それを真にする（＝充足する）割り当てがあるか？」という問題を解くものである[67]．このような問題を解くための素朴なアルゴリズムとしては，各論理変数の true/false で探索木を作り，グラフ探索アルゴリズムで探索する，といったものが考えられる．SMT は，この SAT に述語論理を加えたものとなっている．すなわち，SAT は論理変数の true/false と and/or による組み合わせで記述さるのに対し，SMT は式も含めた形での true/false を記述できるようにしたものとなっている．イメージとしては，探索木を作るときに，式を解くための理論ソルバ（例えば，算術ソルバ）と組み合わせているものとなる．

SMTPlan+ではまず, Happning という要素を導入している. Happning は, action や event, process などの開始・終了時刻 t に対して，実際に状態量に変化が加わるまでに微小時間 ε が生じるという要素であり，これを用いることで時間的な action を「action 開始という Happning」「action 終了という Happning」「action 中という process」という三つに分割している．これにより，離散的な変化は Happning，連続的な変化は process という形でハイブリッドダイナミクスを記述できている．Happning を導入することにより，PDDL+のドメインモデルは Happning の境界集合として記述され，SMT の定式化に落とし込むことができている. PDDL+の問題ファイルも同様に SMT の定式化に落とし込むことができるが，SMT の定式化に落とし込む際に様々な制約を与えている．こうして SMT に落とし込んだ後は，Microsoft Research が中心となって開発した z3 というソルバ[68]を用いて SMT の解を求めている．

A.2. SMTPlan+ の動作環境構築方法

SMTPlan+ は z3 ソルバーをはじめとする様々なライブラリを利用している．そのため，依存関係が複雑になっているだけでなく，特定のバージョンを利用する必要があるライブラリも存在している．こうしたライブラリの組み合わせも考慮しつつ，SMTPlan+動作環境の構築方法を説明していく．

まず，開発者が動作確認している環境は以下の通りである[69].

- Ubuntu 14.04
- libboost-dev1.55 (and libboost-thread1.55-dev)
- Cmake 3.2.2
- gcc 4.8.4
- piranha (hash: 7e420423106923f700b72c07e24d1909d33898b4)

これらをもとに，開発環境を整えていく．

筆者の方では，Ubuntu 14.04，16.04，18.04 の 3 バージョンで動作確認を行っている．OS のバージョンによって利用するライブラリのバージョンが異なる場合があるが，その都度説明を追加する．以下の手順は，筆者が動作環境を構築した際の物であり，必要十分な手順を示しているものではないことに留意されたい．

まず，Ubuntu のパッケージとして提供されているソフトウェアのインストールを行う．インストールすべきパッケージは以下の通り．

- Ubuntu のバージョンに依らずインストールすべきもの
  - Git，build-essential，vim，python，python-dev，flex，libbz2-dev
- 特定のバージョンでインストールすべきもの
  - Ubuntu 14.04: なし
  - Ubuntu 16.04: libboost-all-dev
  - Ubuntu 18.04: libboost1.62-all-dev

以上のパッケージは，「sudo apt install [ライブラリ名]」のコマンドでインストールできる．続いて，パッケージとして提供されていないライブラリや，特定のバージョンが必要なライブラリ等をインストールする．各ソフトウェアの詳細なインストール方法は，各ソフトウェアの Read me や，ウェブサイトなどに掲載されている．必要なソフトウェアは以下の通り．

- Cmake (ver. 3.12.1)
  - https://cmake.org/files/v3.12/cmake-3.12.1.tar.gz
- Boost (ver. 1.55.0) ：Ubuntu 14.04 のみ．
  - http://sourceforge.net/projects/boost/files/boost/1.55.0/boost_1_55_0.tar.bz2
- GMP (ver. 6.1.2)
  - https://gmplib.org/download/gmp/gmp-6.1.2.tar.bz2

- MPFR (ver. 4.0.1)
  - https://www.mpfr.org/mpfr-current/mpfr-4.0.1.tar.bz2

次に，GitHub で公開されているソフトウェアをインストールしていく．GitHub で公開されているソフトウェアの中で最初にインストールすべきものは，「MP++ (https://github.com/bluescarni/mppp)」である．次に，Z3Prover をインストールすることになるが，利用している CPU によっては gcc のライブラリが不足し，コンパイルが失敗することがある．具体的には，Raspberry Pi にインストールされる gcc では一部のライブラリが足りていない．そこで，gcc のライブラリを，別の CPU にインストールされた gcc からコピーすることにする. 筆者は仮想マシン上の Ubuntu 16.04 に gcc をインストールし，gcc ライブラリ内の include フォルダ内のファイルを，Raspberry Pi の gcc ライブラリ内の include フォルダにコピーした.

Gcc 内のライブラリを修正したら，残りのソフトウェアのインストールを行う．インストールするソフトウェアは以下の通り.

- Z3Prover
  - https://github.com/Z3Prover/z3
- Piranha
  - https://github.com/bluescarni/piranha.git
  - かならず,『7e420423106923f700b72c07e24d1909d33898b4』のバージョンをインストールすること.

ここまでで必要なライブラリ類は揃うので，最後に SMTPlan+ 本体をインストールするする．GitHub レポジトリ（https://github.com/KCL-Planning/SMTPlan）から最新版をダウンロードし，ReadMe に書かれている通りにインストール作業を進める．ここで，Cmake コマンドを実行した後に，コンパイル設定を変更する必要がある．具体的には，『SMTPlan/SMTPlan/build/CMakeFiles/SMTPlan.dir/link.txt』において,「-pthread」というオプションを追加してやる必要がある．記入場所としては,「-rdynamic」と書かれている部分を,「-pthread　-rdynamic」と置き換えてやればよい．最後に，make コマンドでコンパイルしてやれば，SMTPlan+ のインストールは完了である.

参考文献

[1] Heidt, Hank, et al. “CubeSat: A new generation of picosatellite for education and industry low-cost space experimentation,” Proceedings of the 14th Annual AIAA/USU Conference on Small Satellites, 2000, SSC00-V-5.

[2] SpaceWorks Enterprise, Inc., “2017 Nano/Microsatellite Marcket Forecast, 7th Edition,” [Online], Available: http://www.spaceworkscommercial.com/wp-content/uploads/2018/01/SpaceWorks_Nano_Microsatellite_Market_Forecast_2017.pdf . [Accessed 2018/10/17]

[3] Axelspace Corporation, “新時代のインフラ、AxelGlobe,” [Online], Available: https://www.axelspace.com/axelglobe/ . [Accessed 2018/10/17]

[4] Planet Labs, Inc., “Our approach,” [Online], Available: https://www.planet.com/company/approach/ .[Accessed 2018/10/17]

[5] OneWeb, “Home,” [Online], Available: http://www.oneweb.world/ .[Accessed 2018/10/17]

[6] Y. Tsuda, et al. "Trajectory Design for Japanese New Asteroid Sample Return Mission Hayabusa-2." 23rd International Symposium on Space Flight Dynamics. Vol. 1501. 2012.

[7] Funase, R., et al., “One-year Deep Space Flight Results of the World’s First Full-scale 50-kg-class Deep Space Probe PROCYON and Its Future Prospects,” Proceedings of the 30th Annual AIAA/USU Conference on Small Satellites, Logan, Utah, 2016, SSC16-III-05.

[8] 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”, [Online], Available: http://www8.cao.go.jp/space/comittee/kagaku-dai7/siryou2.pdf . [Accessed 2018/10/17]

[9] Robinson, K., Schorr, A., and Smith, D., “NASA's Space Launch System: Opportunities for Small Satellites to Deep Space Destinations,” Proceedings of the 32nd Annual AIAA/USU Conference on Small Satellites, Logan, Utah, 2018, SSC18-IX-02

[10] 坂本祐二, et al. "UNISEC/GSN ワーキンググループによる大学衛星のための地上局ネットワークシステムの開発および活動." 電子情報通信学会技術研究報告. SAT, 衛星通信 107.497 (2008): 39-44.

[11] 東北大学, “UNISEC GSN-WG - UNISEC GSN-WG | Members Page”, [Online], Available: http://www.astro.mech.tohoku.ac.jp/~gsn/jp/ [Accessed 2018/11/27]

[12] 田中利樹, 小松満仁, 金相均, 兪逸淵, 草川靖大, 稲守孝哉, 佐藤友紀, 須崎祐多, 清水健介, 三川祥典, 中須賀真一：超小型リモートセンシング衛星「PRISM」の開発, 講演番号 2C16, 第 52 回宇宙科学技術連合講演会, 淡路島, 2008 年 11 月.

[13] Infostellar Inc., “Infostellar”, [Online], Available: https://www.infostellar.net/ [Accessed 2018/11/27]

[14] White, Daniel J.; Giannelos, Ioannis; Zissimatos, Agisilaos; Kosmas, Eleytherios; Papadeas, Dimitrios; Papadeas, Pierros; Papamathaiou, Matthaios; Roussos, Nikolaos; Tsiligiannis, Vasileios; and Charitopoulos, Ioannis, "SatNOGS: Satellite Networked Open Ground Station" (2015). Engineering Faculty Publications. Paper 40. http://scholar.valpo.edu/engineering_fac_pub/40

[15] 伊藤 大智, 冨木 淳史, 福島 洋介, 小林 雄太, 川勝 康弘, 藤澤 健太, 川端 洋輔, 中島 晋太郎, 船瀬 龍, PROCYON チーム, “大学連携を用いた超小型深宇宙探査機の低コスト地上運用システム”, 第 60 回 宇宙科学技術連合講演会, 1G03, 函館, 2016 年 9 月

[16] Hackett, Timothy M., Mark Johnston, and Sven G. Bilen. "Spacecraft Block Scheduling for NASA’s Deep Space Network." 2018 SpaceOps Conference. 2018.

[17] 宇宙開発委員会, “第 18 号化学衛星（PLANET-B）「のぞみ」の火星周回軌道への投入失敗に係る原因究明および今後の対策について”, 2004 年 6 月 25 日.

[18] 船瀬龍, 滝澤潤一, "超小型深宇宙探査機 PROCYON（プロキオン）のミッションと自動化・自律化技術について (< 特集> 宇宙に挑む人工知能技術)." 人工知能: 人工知能学会誌 29.4 (2014): 344-349.

[19] 中島晋太郎, 木村真一, 中須賀真一, 川勝康弘, 船瀬龍, “柔軟な再構成能力を有する衛星搭載ソフトウェアアーキテクチャと PROCYON での実証事例”, 第 59 回 宇宙科学技術連合講演会, 1G14, 鹿児島, 2015 年 10 月

[20] Rayman, Marc D. "The successful conclusion of the Deep Space 1 Mission: important results without a flashy title." Space Technology 23.2 (2003): 185-196.

[21] Muscettola, Nicola, et al. "Remote agent: To boldly go where no AI system has gone before." Artificial Intelligence 103.1 (1998): 5-47.

[22] 福島洋介. 知能化の敷居を下げる DESTINY: スクリプトでの自律運用 (< 特集> 宇宙に挑む人工知能技術). 人工知能: 人工知能学会誌, 2014, 29.4: 335-343.

[23] McGhan, Catharine LR, et al. "A risk-aware architecture for resilient spacecraft operations." Aerospace Conference, 2015 IEEE. IEEE, 2015.

[24] Chien, S., et al.: The EO-1 Autonomous Science Agent, Proc. 3rd Int. Joint Conf. on Autonomous Agents and Multiagent Systems, pp. 420-427（2004）

[25] Gaines, D., Rabideau, G., Doran, G., Schaffer, S., Wong, V., Vasavada, A., Anderson, R., “Expressing Campaign Intent to Increase Productivity of Planetary Exploration Rovers”, ICAPS Workshop on Planning and Robotics, 2017

[26] Balzano, V. and Isaacs, J. C.: Event-driven James Webb Space Telescope Operations, SpaceOps Conf. Proc., AIAA-2006-5747（2006）

[27] Obata, T., Nakasuka, S., Aoyanagi, Y., Matsumoto, T., and Shirasaka, S., “On-Orbit

Demonstrations of Robust Autonomous Operations on Cubesat," Proceedings of the 32nd Annual AIAA/USU Conference on Small Satellites, Logan, Utah, 2018, SSC18-WKX-02.

[28] 三宅陽一郎. "ディジタルゲームにおける人工知能技術の応用の現在 (< 特集> エンターテイメントにおける AI)." 人工知能: 人工知能学会誌 30.1 (2015): 45-64.

[29] 五十里哲，et al.,"超小型深宇宙探査機 PROCYON の姿勢制御系開発と初期運用結果", 第 2 回計測自動制御学会制御部門マルチシンポジウム，663-4，2015 年

[30] Downie, Robert Burke Damian Isla Marc, and Yuri Ivanov Bruce Blumberg. "Creature smarts: The art and architecture of a virtual brain." (2001).

[31] Orkin, J.: Applying goal-oriented action planning to games. In: AI Game Programming Wisdom II. Charles River Media (2003)

[32] Doherty, Patrick, and Jonas Kvarnstram. "TALplanner: A temporal logic-based planner." AI Magazine 22.3 (2001): 95.

[33] Penix, J., Pecheur, C., & Havelund, K. (1998, December). Using model checking to validate AI planner domain models. In Proceedings of the 23rd Annual Software Engineering Workshop, NASA Goddard.

[34] Michael Cashmore, Luca Iocchi, Daniele Magazzeni, "AI Planning for Robotics and Human-Robot Interaction", Tutorials at The 27th International Conference on Automated Planning and Scheduling, Pittsburgh, USA, 2017.

[35] Seko, Hiromi, et al. "Development of the DS2000 Platform for GEO/HEO Satellites." 21st International Communications Satellite Systems Conference and Exhibit. 2003.

[36] Hihara, Hiroki, Toshiaki Ogawa, and Kenji Kitade. "NEXTAR: small satellite bus based on SpaceWire deterministic implementation." Int. SpaceWire Conf. 2011.

[37] 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

[38] Kevin D. Anderson and Darryll J. Pines. "Methods of Pulse Phase Tracking for X-ray Pulsar Based Spacecraft Navigation using Low Flux Pulsars", SpaceOps 2014 Conference, SpaceOps Conferences, (AIAA 2014-1858)

[39] Hill, K., and Born, G. H., "Autonomous Interplanetary Orbit Determination Using Satellite-to-Satellite Tracking," Journal of Guidance, Control, and Dynamics, Vol. 30, No. 3, May-June 2007, pp. 679-686.

[40] 川端洋輔，"深宇宙における光学情報を用いた宇宙機の自律軌道決定について"，東京大学新領域創成科学研究科博士課程学位論文，2017

[41] Williams, B. C., & Nayak, P. P. (1996, August). A model-based approach to reactive self-configuring systems. In Proceedings of the national conference on artificial intelligence (pp. 971-978).

[42] M. Ghallab, D. Nau, and P. Traverso, Automated Planning: Theory and Practice. Morgan Kaufmann, 2004.

[43] 滝澤潤一, “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”, 東京大学工学系研究科博士課程学位論文, 2015

[44] Shintaro NAKAJIMA, Ryu FUNASE, Shin'ichi NAKASUKA, Satoshi IKARI, Masashi TOMOOKA, Yoshihide AOYANAGI, “Command Centric Architecture (C2A): Satellite Software Architecture with a Flexible Reconfiguration Capability” , 67th International Astronautical Congress, IAC-17,D1,2,11,x40875, Adelaide, Australia, September 2017

[45] Tran Ngoc Lan Huong, Sotaro Kobayashi, Jun'ichi Takisawa, Shinichi Nakasuka, ShinichiKimura, "Automatic Document Based Program Code Generation System for On-Board Software", 29th ISTS, 2013

[46] 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

[47] Malik Ghallab, Dana Nau and Paolo Terverso, "Automated Planning and Acting", Cambridge University Press, August 2016

[48] McDermott, Drew, et al. "PDDL-the planning domain definition language." (1998).

[49] Michael Cashmore, Maria Fox, Derek Long and Daniele Magazzeni, “A Compilation of the Full PDDL+ Language into SMT,” Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS-16), 2016

[50] Systems Engineering Handbook, A Guide for System Life Cycle Process and Activities, Ver.3.2, International Council on Systems Engineering, (2010)

[51] 西村 秀和, “システムズエンジニアリング・MBSE 概要”, [Online], Available: https://sec.ipa.go.jp/users/seminar/seminar_tokyo_20151210-01.pdf, 2013

[52] 情報処理推進機構, “モデルベースシステムズエンジニアリング 導入の手引き”, https://www.ipa.go.jp/files/000033609.pdf

[53] “IEEE Standard for Application and Management of the Systems Engineering Process”, IEEE Std 1220-2005, pp c1-66, 2007

[54] Malone, Robert, et al. "Insights from Large Scale Model Based Systems Engineering at Boeing." INCOSE International Symposium. Vol. 26. No. 1. 2016.

[55] Kaslow, David, et al. "A Model-Based Systems Engineering (MBSE) approach for defining the behaviors of CubeSats." Aerospace Conference, 2017 IEEE. IEEE, 2017.

[56] 宇宙航空研究開発機構, “宇宙システムの概念検討およびシステム技術の研究”, [Online], Available: http://www.kenkai.jaxa.jp/research/system/system.html [Accessed 2018/11/27]

[57] 野田篤司, “「つばめ」（SLATS）の概念検討 ～アイデアから立ち上げ～”, 8th UNISEC Space Takumi Conference, UNISEC 2018-005, 2018

[58] Kevin Forsberg, Hal Mooz, Howard Cotterman, Visualizing Project Management, Third Edition, John Wiley & Sons, Inc., (2005)

[59] Yamada, Takahiro. "Standardization of Spacecraft and Ground Systems Based on a Spacecraft Functional Model." SpaceOps 2008 Conference. 2008.

[60] Ryu Funase et al., Flight Model Design and Development Status of the Earth-Moon lagrange Point Exploration Cubesat EQUULEUS Onboard SLS EM-1, Proceedings of the AIAA/USU Conference on Small Satellites (2018), SSC18-VII-05.

[61] Sho Koizumi et al., “Development of Attitude Sensor using Deep Learning”, Proceedings of the AIAA/USU Conference on Small Satellites (2018), SSC18-WKVII-01.

[62] Raspberry Pi Foundation, “Featured products”, [Online], Available: https://www.raspberrypi.org/products/, [Accessed 2018/11/27]

[63] No magic, “MagicGrid Quick Reference Guide”, [Online], Available: https://www.nomagic.com/component/phocadownload/category/1-quick-reference-guides?download=8:magicgrid-quick-reference-guide, 2017

[64] YOUNES, Håkan LS; LITTMAN, Michael L. PPDDL1. 0: An extension to PDDL for expressing planning domains with probabilistic effects. Techn. Rep. CMU-CS-04-162, 2004.

[65] The International Conference on Automated Planning and Scheduling, “ICAPS Competitions”, [Online], Available: http://www.icaps-conference.org/index.php/Main/Competitions, [Accessed 2018/11/27]

[66] Fox, M., and Long, D. 2006. Modelling mixed discretecontinuous domains for planning. Jorunal of Artificial Intelligence Research (JAIR) 27:235–297.

[67] 酒井政裕, “SAT/SMT ソルバの仕組み”, [Online], Available: https://www.slideshare.net/sakai/satsmt, [Accessed 2018/11/27]

[68] De Moura, Leonardo, and Nikolaj Bjørner. "Z3: An efficient SMT solver." International conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, Berlin, Heidelberg, 2008.

[69] Kings College London, “ Issue using make command · Issue #5 · KCL-Planning/SMTPlan”, [Online], Available: https://github.com/KCL-Planning/SMTPlan/issues/5, [Accessed 2018/11/27]

[70] Walker, Lloyd, and Dale Thomas. "Integrated system modeling in SysML for Small

Satellites." AIAA Modeling and Simulation Technologies Conference. 2017.

謝辞

まず，指導教官であり本論文の主査を担当してくださった船瀬龍准教授には，学部 4 年生からの 6 年間お世話になりました．お忙しい中，昼夜問わず研究や進路に関する様々なアドバイスをいただいたともに，実際に打ち上げられる超小型深宇宙探査機に関するプロジェクトに参加する機会を設けていただきました．これらの経験は私の中でかけがえのないものです．本当にありがとうございました．

次に，副査である中須賀真一教授には学部 3 年生からの 7 年間お世話になりました．模擬衛星である CanSat を作る機会を提供していただき，本格的な宇宙機の研究・開発に携わるきっかけを作ってくださいました．研究室に所属した後は，研究に関することや進路に関することなど，様々なアドバイスをいただきました．深く感謝を申し上げます．

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，矢入健久准教授，慶應義塾大学の白坂成功教授に心から感謝を申し上げます．本論文に関して，様々な視点から建設的なアドバイスをいただきました．

中須賀・船瀬研究室のメンバーにも，感謝を述べたいと思います．特に，先輩である五十里哲助教や尾崎直哉博士には，プロジェクトや研究に関して相談をさせていただいただけでなく，研究室活動の進め方など，本当に多くの議論をし，アドバイスもいただきました．本当にありがとうございました．また，小畑俊裕氏や髙橋亮平君をはじめとする研究グループのメンバーには，研究に関する多くの建設的なアドバイスをいただきました．

PROCYON および EQUULEUS，Nano-JASMINE を始めとする各種プロジェクトにおいても，多くを学ばせて頂きました．特に，PROCYON と EQUULEUS のプロジェクトマネージャーである船瀬龍准教授や，Nano-JASMINE のプロジェクトマネージャーである酒匂信匡博士には大変お世話になりました．参加させていただいたプロジェクトの中でも，概念検討から打ち上げ後の運用まで，衛星開発のフェーズの多くを経験できた PROCYON プロジェクトは非常に大きな存在であり，本研究の着想は PROCYON プロジェクトでの運用経験から得たものです．また，プロジェクトの立ち上げ段階からフライトモデルの設計まで携わった EQUULEUS プロジェクトに関しては，実機を利用して試験をさせていただきました．各プロジェクトメンバーの皆様，本当にありがとうございました．

博士課程 1 年の半年間においては，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．深く感謝をしております．

博士課程 1 年の 9 月から博士課程 3 年までの 2 年半にわたり，リーダー博士人材育成基金（LDPP）の支援をいただきました．月々の奨学金による経済的なサポートだけでなく，研究費の支援をいただきました．さらには，LDPP の採用者やご参加企業との交流会を開催していただき，航空宇宙分野にとどまらない多角的なフィードバックと刺激をいただきました．非常に深く感謝をしております．

最後に，博士課程までの進学を経済的・精神的にサポートしてくださった父 義之，母 喜

久子，姉 小春をはじめとする家族に感謝を伝えたいと思います．ありがとうございました．

他にも，博士課程で研究を進めるうえで，非常に多くの方に助けていただきました．全ての方のお名前を挙げることはできませんが，この場をお借りして深く感謝を申し上げたいと思います．

付録

付録として，本研究で作成し，シミュレーションや実機試験に利用した pddl ファイルを添付する．特に重要なものとして，4.5.2 項で利用した「観測機器周りの機器・サブシステムを統合したドメインモデル」の PDDL ファイルおよび自動計画問題の PDDL ファイルと，4.6.2 項において模擬故障を検知した際に作成された自動計画問題の PDDL ファイルを添付する．

観測機器周りの機器・サブシステムを統合したドメインモデル (equ-domain-simple4.pddl)

```
(define (domain EQU-SIMPLE)

(:requirements
 :strips
 :durative-actions
 :negative-preconditions
)

(:predicates
   (sw_xact_bus)
   (sw_xact_5v)
   (vol_xact_bus)
   (cur_xact_bus)
   (vol_xact_5v)
   (cur_xact_5v)
   (sw_zeus_bus)
   (sw_zeus_5v)
   (vol_zeus_bus)
   (cur_zeus_bus)
   (vol_zeus_5v)
   (cur_zeus_5v)
   (sw_phx)
```

(vol_phx)

(cur_phx)

(sw_dlp)

(vol_dlp)

(cur_dlp)

(sw_clt)

(vol_clt)

(cur_clt)

(sw_bat_htr_bus)

(vol_bat_htr_bus)

(cur_bat_htr_bus)

(sw_sada)

(vol_sada)

(cur_sada)

(sap_looks_sun)

(dlp_mode_cam)

(dlp_mode_lif)

(dlp_param_lif)

(dlp_param_cam)

(xact_att_target)

(dlp_data_cam)

(obc_flash)

(xact_att_moon)

(dlp_data_lif)

(clt_obs_param)

(phx_shutter_open)

(phx_hv_ena)

(phx_hv_on)

(phx_hv_param)

(phx_obs_param)

(phx_data)

(vol_trp_bus)

(cur_trp_bus)

(trp_tx_on)

(trp_bitrate)

(xact_maneuvering)

```
    (xact_att_sun)
  )
  (:durative-action turnon_xact_bus
   :parameters ()
   :duration (= ?duration 1)
   :condition (and (at start (not (sw_xact_bus))) (at start (sw_xact_5v)))
   :effect (and (at start (sw_xact_bus)) (at end (vol_xact_bus)) (at end (cur_xact_bus)) (at end (not (xact_att_moon)))
(at end (not (xact_att_sun))) (at end (not (xact_att_target))) (at end (not (xact_maneuvering))))
  )

  (:durative-action turnoff_xact_bus
   :parameters ()
   :duration (= ?duration 1)
   :condition (and (at start (sw_xact_bus)))
   :effect (and (at start (not (sw_xact_bus))) (at end (not (vol_xact_bus))) (at end (not (cur_xact_bus))) (at end (not
(xact_att_moon))) (at end (not (xact_att_sun))) (at end (not (xact_att_target))) (at end (not (xact_maneuvering))))
  )

  (:durative-action turnon_xact_5v
   :parameters ()
   :duration (= ?duration 1)
   :condition (and (at start (not (sw_xact_bus))) (at start (not (sw_xact_5v))))
   :effect (and (at start (sw_xact_5v)) (at end (vol_xact_5v)) (at end (cur_xact_5v)) (at end (not (xact_att_moon))) (at
end (not (xact_att_sun))) (at end (not (xact_att_target))) (at end (not (xact_maneuvering))))
  )

  (:durative-action turnoff_xact_5v
   :parameters ()
   :duration (= ?duration 1)
   :condition (and (at start (sw_xact_5v)) (at start (not (sw_xact_bus))))
   :effect (and (at start (not (sw_xact_5v))) (at end (not (vol_xact_5v))) (at end (not (cur_xact_5v))) (at end (not
(xact_att_moon))) (at end (not (xact_att_sun))) (at end (not (xact_att_target))) (at end (not (xact_maneuvering))))
  )

  (:durative-action turnon_zeus_bus
   :parameters ()
```

```
  :duration (= ?duration 1)
  :condition (and (at start (not (sw_zeus_bus))) (at start (sw_zeus_5v)))
  :effect (and (at start (sw_zeus_bus)) (at end (vol_zeus_bus)) (at end (cur_zeus_bus)))
 )

 (:durative-action turnoff_zeus_bus
  :parameters ()
  :duration (= ?duration 1)
  :condition (and (at start (sw_zeus_bus)))
  :effect (and (at start (not (sw_zeus_bus))) (at end (not (vol_zeus_bus))) (at end (not (cur_zeus_bus))))
 )

 (:durative-action turnon_zeus_5v
  :parameters ()
  :duration (= ?duration 1)
  :condition (and (at start (not (sw_zeus_bus))) (at start (not (sw_zeus_5v))))
  :effect (and (at start (sw_zeus_5v)) (at end (vol_zeus_5v)) (at end (cur_zeus_5v)))
 )

 (:durative-action turnoff_zeus_5v
  :parameters ()
  :duration (= ?duration 1)
  :condition (and (at start (sw_zeus_5v)) (at start (not (sw_zeus_bus))))
  :effect (and (at start (not (sw_zeus_5v))) (at end (not (vol_zeus_5v))) (at end (not (cur_zeus_5v))))
 )

 (:durative-action turnon_phx
  :parameters ()
  :duration (= ?duration 1)
  :condition (and (at start (not (sw_phx))))
  :effect (and (at start (sw_phx)) (at end (vol_phx)) (at end (cur_phx)) (at end (not (phx_data))) (at end (not
(phx_hv_ena))) (at end (not (phx_hv_on))) (at end (not (phx_hv_param))) (at end (not (phx_obs_param))))
 )

 (:durative-action turnoff_phx
  :parameters ()
```

```
    :duration (= ?duration 1)
    :condition (and (at start (sw_phx)))
    :effect (and (at start (not (sw_phx))) (at end (not (vol_phx))) (at end (not (cur_phx))) (at end (not (phx_data))) (at end (not (phx_hv_ena))) (at end (not (phx_hv_on))) (at end (not (phx_hv_param))) (at end (not (phx_obs_param))))
  )

  (:durative-action turnon_dlp
    :parameters ()
    :duration (= ?duration 1)
    :condition (and (at start (not (sw_dlp))))
    :effect (and (at start (sw_dlp)) (at end (vol_dlp)) (at end (cur_dlp)) (at end (not (dlp_data_cam))) (at end (not (dlp_data_lif))) (at end (not (dlp_mode_cam))) (at end (not (dlp_mode_lif))) (at end (not (dlp_param_cam))) (at end (not (dlp_param_lif))))
  )

  (:durative-action turnoff_dlp
    :parameters ()
    :duration (= ?duration 1)
    :condition (and (at start (sw_dlp)))
    :effect (and (at start (not (sw_dlp))) (at end (not (vol_dlp))) (at end (not (cur_dlp))) (at end (not (dlp_data_cam))) (at end (not (dlp_data_lif))) (at end (not (dlp_mode_cam))) (at end (not (dlp_mode_lif))) (at end (not (dlp_param_cam))) (at end (not (dlp_param_lif))))
  )

  (:durative-action turnon_clt
    :parameters ()
    :duration (= ?duration 1)
    :condition (and (at start (not (sw_clt))))
    :effect (and (at start (sw_clt)) (at end (vol_clt)) (at end (cur_clt)) (at end (not (clt_obs_param))))
  )

  (:durative-action turnoff_clt
    :parameters ()
    :duration (= ?duration 1)
    :condition (and (at start (sw_clt)))
    :effect (and (at start (not (sw_clt))) (at end (not (vol_clt))) (at end (not (cur_clt))) (at end (not (clt_obs_param))))
```

```
)

(:durative-action turnon_bat_htr_bus
  :parameters ()
  :duration (= ?duration 1)
  :condition (and (at start (not (sw_bat_htr_bus))))
  :effect (and (at start (sw_bat_htr_bus)) (at end (vol_bat_htr_bus)) (at end (cur_bat_htr_bus)))
)

(:durative-action turnoff_bat_htr_bus
  :parameters ()
  :duration (= ?duration 1)
  :condition (and (at start (sw_bat_htr_bus)))
  :effect (and (at start (not (sw_bat_htr_bus))) (at end (not (vol_bat_htr_bus))) (at end (not (cur_bat_htr_bus))))
)

(:durative-action turnon_sada
  :parameters ()
  :duration (= ?duration 1)
  :condition (and (at start (not (sw_sada))))
  :effect (and (at start (sw_sada)) (at end (vol_sada)) (at end (cur_sada)))
)

(:durative-action turnoff_sada
  :parameters ()
  :duration (= ?duration 1)
  :condition (and (at start (sw_sada)))
  :effect (and (at start (not (sw_sada))) (at end (not (vol_sada))) (at end (not (cur_sada))))
)

(:durative-action sada_rotate2sun
  :parameters ()
  :duration (= ?duration 3)
  :condition (and (over all (vol_sada)) (over all (cur_sada)) (at start (not (sap_looks_sun))))
  :effect (and (at end (sap_looks_sun)))
)
```

```
  (:durative-action dlp_set_mode_cam
   :parameters ()
   :duration (= ?duration 1)
   :condition (and (over all (vol_dlp)) (over all (cur_dlp)))
   :effect (and (at end (dlp_mode_cam)) (at start (not (dlp_mode_lif))) (at end (not (dlp_mode_lif))))
  )

  (:durative-action dlp_set_param_cam
   :parameters ()
   :duration (= ?duration 1)
   :condition (and (over all (vol_dlp)) (over all (cur_dlp)) (over all (dlp_mode_cam)))
   :effect (and (at end (not (dlp_param_lif))) (at end (dlp_param_cam)))
  )

  (:durative-action dlp_observe_cam
   :parameters ()
   :duration (= ?duration 5)
   :condition (and (over all (vol_dlp)) (over all (cur_dlp)) (over all (dlp_mode_cam)) (over all (dlp_param_cam)) (over
all (xact_att_target)) (over all (sap_looks_sun)))
   :effect (and (at end (dlp_data_cam)))
  )

  (:durative-action dlp_send_data_cam
   :parameters ()
   :duration (= ?duration 7)
   :condition (and (over all (vol_dlp)) (over all (cur_dlp)) (over all (dlp_data_cam)) (over all (not (obc_flash))))
   :effect (and (at end (obc_flash)))
  )

  (:durative-action dlp_set_mode_lif
   :parameters ()
   :duration (= ?duration 1)
   :condition (and (over all (vol_dlp)) (over all (cur_dlp)))
   :effect (and (at end (dlp_mode_lif)) (at start (not (dlp_mode_cam))) (at end (not (dlp_mode_cam))))
  )
```

```
  (:durative-action dlp_set_param_lif
   :parameters ()
   :duration (= ?duration 1)
   :condition (and (over all (vol_dlp)) (over all (cur_dlp)) (over all (dlp_mode_lif)))
   :effect (and (at end (not (dlp_param_cam))) (at end (dlp_param_lif)))
  )

  (:durative-action dlp_observe_lif
   :parameters ()
   :duration (= ?duration 5)
   :condition (and (over all (vol_dlp)) (over all (cur_dlp)) (over all (dlp_mode_lif)) (over all (dlp_param_lif)) (over all
(xact_att_moon)) (over all (sap_looks_sun)))
   :effect (and (at end (dlp_data_lif)))
  )

  (:durative-action dlp_send_data_lif
   :parameters ()
   :duration (= ?duration 7)
   :condition (and (over all (vol_dlp)) (over all (cur_dlp)) (over all (dlp_data_lif)) (over all (not (obc_flash))))
   :effect (and (at end (obc_flash)))
  )

  (:durative-action clt_set_param
   :parameters ()
   :duration (= ?duration 1)
   :condition (and (over all (vol_clt)) (over all (cur_clt)))
   :effect (and (at end (clt_obs_param)))
  )

  (:durative-action clt_send_data
   :parameters ()
   :duration (= ?duration 2)
   :condition (and (over all (vol_clt)) (over all (cur_clt)) (over all (clt_obs_param)) (over all (not (obc_flash))))
   :effect (and (at end (obc_flash)))
  )
```

```
(:durative-action phx_open_shutter
  :parameters ()
  :duration (= ?duration 4)
  :condition (and (over all (vol_zeus_5v)) (over all (cur_zeus_5v)))
  :effect (and (at end (phx_shutter_open)))
)

(:durative-action phx_close_shutter
  :parameters ()
  :duration (= ?duration 4)
  :condition (and (over all (vol_zeus_5v)) (over all (vol_zeus_5v)))
  :effect (and (at end (not (phx_shutter_open))))
)

(:durative-action phx_ena_hv
  :parameters ()
  :duration (= ?duration 1)
  :condition (and (over all (vol_phx)) (over all (cur_phx)) (at start (not (phx_hv_ena))))
  :effect (and (at end (phx_hv_ena)))
)

(:durative-action phx_hv_on
  :parameters ()
  :duration (= ?duration 1)
  :condition (and (over all (vol_phx)) (over all (cur_phx)) (at start (phx_hv_ena)))
  :effect (and (at end (not (phx_hv_ena))) (at end (phx_hv_on)) (at end (not (phx_hv_param))))
)

(:durative-action phx_hv_off
  :parameters ()
  :duration (= ?duration 1)
  :condition (and (over all (vol_phx)) (over all (cur_phx)))
  :effect (and (at end (not (phx_hv_on))) (at end (not (phx_hv_ena))) (at end (not (phx_hv_param))))
)
```

```
  (:durative-action phx_set_hv
   :parameters ()
   :duration (= ?duration 1)
   :condition (and (over all (vol_phx)) (over all (cur_phx)) (over all (phx_hv_on)))
   :effect (and (at end (phx_hv_param)))
  )

  (:durative-action phx_set_param
   :parameters ()
   :duration (= ?duration 1)
   :condition (and (over all (cur_phx)) (over all (vol_phx)))
   :effect (and (at end (phx_obs_param)))
  )

  (:durative-action phx_observe
   :parameters ()
   :duration (= ?duration 15)
   :condition (and (over all (vol_phx)) (over all (cur_phx)) (over all (phx_hv_on)) (over all (phx_hv_param)) (over all
(phx_shutter_open)) (at start (phx_obs_param)) (over all (xact_att_target)) (over all (sap_looks_sun)))
   :effect (and (at end (phx_data)))
  )

  (:durative-action phx_send_data
   :parameters ()
   :duration (= ?duration 6)
   :condition (and (over all (vol_phx)) (over all (cur_phx)) (over all (phx_data)) (over all (not (obc_flash))))
   :effect (and (at end (obc_flash)))
  )

  (:durative-action data_downlink
   :parameters ()
   :duration (= ?duration 1)
   :condition (and (over all (vol_trp_bus)) (over all (cur_trp_bus)) (over all (trp_tx_on)) (over all (trp_bitrate)) (at
start (obc_flash)))
   :effect (and (at end (not (obc_flash))))
  )
```

```
  (:durative-action xact_set_att_target
    :parameters ()
    :duration (= ?duration 10)
    :condition (and (over all (vol_xact_bus)) (over all (cur_xact_bus)) (over all (vol_xact_5v)) (over all (cur_xact_5v)) (over all (not (phx_shutter_open))) (at start (not (xact_maneuvering))))
    :effect (and (at start (xact_maneuvering)) (at end (not (xact_maneuvering))) (at end (xact_att_target)) (at start (not (xact_att_moon))) (at start (not (xact_att_sun))) (at start (not (sap_looks_sun))) (at end (not (sap_looks_sun))))
  )

  (:durative-action xact_set_att_moon
    :parameters ()
    :duration (= ?duration 10)
    :condition (and (over all (vol_xact_bus)) (over all (cur_xact_bus)) (over all (vol_xact_5v)) (over all (cur_xact_5v)) (over all (not (phx_shutter_open))) (at start (not (xact_maneuvering))))
    :effect (and (at start (xact_maneuvering)) (at end (not (xact_maneuvering))) (at end (xact_att_moon)) (at start (not (xact_att_target))) (at start (not (xact_att_sun))) (at start (not (sap_looks_sun))) (at end (not (sap_looks_sun))))
  )

  (:durative-action xact_set_att_sun
    :parameters ()
    :duration (= ?duration 10)
    :condition (and (over all (vol_xact_bus)) (over all (cur_xact_bus)) (over all (vol_xact_5v)) (over all (cur_xact_5v)) (over all (not (phx_shutter_open))) (at start (not (xact_maneuvering))))
    :effect (and (at start (xact_maneuvering)) (at end (not (xact_maneuvering))) (at end (xact_att_sun)) (at start (not (xact_att_moon))) (at start (not (xact_att_target))) (at start (not (sap_looks_sun))) (at end (sap_looks_sun)))
  )

  (:durative-action xact_set_att_free
    :parameters ()
    :duration (= ?duration 2)
    :condition (and (over all (vol_xact_bus)) (over all (cur_xact_bus)) (over all (vol_xact_5v)) (over all (cur_xact_5v)) (over all (not (phx_shutter_open))))
    :effect (and (at start (not (xact_maneuvering))) (at end (not (xact_maneuvering))) (at start (not (xact_att_sun))) (at start (not (xact_att_moon))) (at start (not (xact_att_target))) (at start (not (sap_looks_sun))))
  )
```

```
(:durative-action trp_tx_on
  :parameters ()
  :duration (= ?duration 1)
  :condition (and (over all (vol_trp_bus)) (over all (cur_trp_bus)))
  :effect (and (at end (trp_tx_on)))
)

(:durative-action trp_tx_off
  :parameters ()
  :duration (= ?duration 1)
  :condition (and (over all (vol_trp_bus)) (over all (cur_trp_bus)))
  :effect (and (at end (not (trp_tx_on))))
)

(:durative-action trp_set_bitrate
  :parameters ()
  :duration (= ?duration 1)
  :condition (and (over all (vol_trp_bus)) (over all (cur_trp_bus)))
  :effect (and (at end (trp_bitrate)))
)

(:durative-action trp_reconfig
  :parameters ()
  :duration (= ?duration 2)
  :condition (and (over all (vol_trp_bus)) (over all (cur_trp_bus)))
  :effect (and (at end (not (trp_tx_on))) (at end (not (trp_bitrate))))
)


)
```

自動計画問題の PDDL ファイル（phx.pddl）

```
(define (problem EQU_TEST)
(:domain EQU-SIMPLE)

(:init
   (not (sw_xact_bus))
   (not (sw_xact_5v))
   (not (vol_xact_bus))
   (not (cur_xact_bus))
   (not (vol_xact_5v))
   (not (cur_xact_5v))
   (not (sw_zeus_bus))
   (not (sw_zeus_5v))
   (not (vol_zeus_bus))
   (not (cur_zeus_bus))
   (not (vol_zeus_5v))
   (not (cur_zeus_5v))
   (not (sw_phx))
   (not (vol_phx))
   (not (cur_phx))
   (not (sw_dlp))
   (not (vol_dlp))
   (not (cur_dlp))
   (not (sw_clt))
   (not (vol_clt))
   (not (cur_clt))
   (not (sw_bat_htr_bus))
   (not (vol_bat_htr_bus))
   (not (cur_bat_htr_bus))
   (not (sw_sada))
   (not (vol_sada))
   (not (cur_sada))
   (not (sap_looks_sun))
```

```
   (not (dlp_mode_cam))
   (not (dlp_mode_lif))
   (not (dlp_param_cam))
   (not (xact_att_target))
   (not (dlp_data_cam))
   (not (obc_flash))
   (not (dlp_param_lif))
   (not (xact_att_moon))
   (not (dlp_data_lif))
   (not (clt_obs_param))
   (not (phx_shutter_open))
   (not (phx_hv_ena))
   (not (phx_hv_on))
   (not (phx_hv_param))
   (not (phx_obs_param))
   (not (phx_data))
   (not (vol_trp_bus))
   (not (cur_trp_bus))
   (not (trp_tx_on))
   (not (trp_bitrate))
   (not (xact_maneuvering))
   (not (xact_att_sun))
)


(:goal (and
   (phx_data)
))

)
```

模擬故障を検知した際に作成された自動計画問題の PDDL ファイル（replan_prob_2180.pddl）

```
(define (problem EQU_TEST)
(:domain EQU-SIMPLE)

(:init
   (sw_xact_bus)
   (sw_xact_5v)
   (vol_xact_bus)
   (cur_xact_bus)
   (not (vol_xact_5v))
   (not (cur_xact_5v))
   (not (sw_zeus_bus))
   (sw_zeus_5v)
   (not (vol_zeus_bus))
   (not (cur_zeus_bus))
   (vol_zeus_5v)
   (cur_zeus_5v)
   (sw_phx)
   (not (vol_phx))
   (not (cur_phx))
   (not (sw_dlp))
   (not (vol_dlp))
   (not (cur_dlp))
   (not (sw_clt))
   (not (vol_clt))
   (not (cur_clt))
   (not (sw_bat_htr_bus))
   (not (vol_bat_htr_bus))
   (not (cur_bat_htr_bus))
   (sw_sada)
   (vol_sada)
```

```
    (cur_sada)
    (sap_looks_sun)
    (not (dlp_mode_cam))
    (dlp_mode_lif)
    (dlp_param_lif)
    (not (dlp_param_cam))
    (xact_att_target)
    (not (dlp_data_cam))
    (not (obc_flash))
    (not (xact_att_moon))
    (not (dlp_data_lif))
    (not (clt_obs_param))
    (phx_shutter_open)
    (not (phx_hv_ena))
    (phx_hv_on)
    (not (phx_hv_param))
    (not (phx_obs_param))
    (not (phx_data))
    (vol_trp_bus)
    (cur_trp_bus)
    (not (trp_tx_on))
    (not (trp_bitrate))
    (not (xact_maneuvering))
    (not (xact_att_sun))
)


(:goal (and
    (phx_data)
))

)
```