国立天文台報　第10巻, 1－22（2007）

Nano-JASMINE シミュレータ

山田良透[*1]京都大学，酒匂信匡[*2]東京大学，初鳥陽一[*2]，片岸秀明[*3]南カリフォルニア大学，
田中崇資[*2]，稲守孝哉[*2]，山内雅浩[*2]，矢野太平，郷田直輝

（2006年10月31日受理）

Nano-JASMINE Simulator

Yoshiyuki YAMADA, Nobutada SAKO, Yoichi HATSUTORI, Hideaki KATAGISHI, Takashi TANAKA,
Takaya INATORI, Masahiro YAMAGUCHI, Taihei YANO, Naoteru GOUDA

Abstract

JASMINE is an abbreviation of Japan Astrometry Satellite Mission for INfrared Exploration currently planned at National Astoronomical Observatory of Japan. We are also planning to launch Nano-JASMINE in about 2008, which is nano-size technical demonstrator with about 10kg weight. Now we implements beta version of the software for end-to-end simulation of Nano-JASMINE. In this paper, we report the modelling and the implementation of the simulator. We also report that we prove possibility of attitude control using mission data as the first application of the simulator. We adopt object oriented methodologies for giving applicability for JASMINE and other space missions.

１．はじめに

「JASMINE」は，口径75cm程度の望遠鏡による位置天文衛星観測プロジェクトである．一方我々は，技術実証衛星として「Nano-JASMINE」を計画している．このNano-JASMINEは，日本で初めての人工衛星による位置天文観測である．ESAが1989年に打ち上げたHIPPARCOSは光電管を用いた観測を行ったが，これに続くESAの計画はCCDのTDIモードを用いた観測を行うことになっている．Nano-JASMINEは，これに先駆けて世界で始めて衛星に搭載したCCDのTDIモードを用いた，位置天文観測を行う．また，現在海外で計画されている衛星による位置天文観測は全て可視光によるものだが，日本で計画されているJASMINEは赤外線（z-バンド，中心波長0.9μm）を用いる点が特徴的である．Nano-JASMINEは，JASMINEに搭載予定のzバンドのCCDの小型のものを搭載する点から，zバンドCCDの衛星上での特性の検証ともなる．また，我々はJASMINEとNano-JASMINEの二つの計画を同時に進行させながら，装置のスペックなどを検討している．Nano-JASMINEが検討どおりのスペックを出すことが出来るかどうかは，JASMINEの検討に不足した点がないことを確認するためにも重要である．

Nano-JASMINE Simulatorは，このNano-JASMINE計画のために，搭載される観測機器と衛星システムの双方に渡る幅広い機能を，物理量の幅広いダイナミックレンジに渡り同時にシミュレーションすることにより，Nano-JASMINEの動作を検証し，解析手法を含めた衛星設計全体に反映し，ミッションフィージビリティーを確認することを目的とするソフトウエアである．さらに，解析手法を確立し，衛星が打ちあがったときの解析ソフトウエアとしてすぐに使用可能なものを予め構築するため，観測模擬データを作成したり，可能なサイエンスの評価に役立てる役割も持つ．また，ソフトウエアのシミュレーターを手元に用意しておくことで，衛星打ち上げ後の予期しない不具合に迅速に対応するためのツールとしても用いるこ

*1 京都大学（Kyoto University）
*2 東京大学（The University of Tokyo）
*3 南カリフォルニア大学（University of Southern California）

とが出来る．

我々のシミュレーター構築は，オブジェクト指向技術を用いることにより，必要なパーツをモデル化して組み合わせている．このため，単独星，望遠鏡，検出器，オンボードコンピューターなどに相当する基本的なソフトウエア部品を作っている．天文観測は一般に望遠鏡と検出器を用いて行われ，これらは他のミッションにも利用可能である．即ち，他のミッションにおいて望遠鏡やCCD以外のミッション独自の装置を使用する場合でも，独自の装置のモデル化を適切に行ってソフトウエア部品を追加することによって，シミュレーター自身の基本構造と既存のソフトウエア部品はそのまま適用可能であり，幅広いミッションに適用可能なシミュレーターとなる．

我々は，兼ねてより「JASMINE」のためのシミュレーションシステムの検討を行なってきた[1,2]．JASMINE Simulatorの骨格は数値の依存性を非巡回有向グラフ（DAG）で記述し，このグラフを利用してシミュレーションを行うものとして報告されたが，本論文で報告されるNano-JASMINE Simulatorは，基本的にこのソフトウエアの枠組みを踏襲している．本実装では，観測に関連する装置の状態変化を引き起こす原因である光子やノイズなどを「イベント」という形で抽象化し，グラフ上に「イベント」を流す点が，以前報告されたJASMINE Simulatorからソフトウエアの枠組みに機能が追加された部分である．イベントという具体物をグラフ上に流すため，ノードの結合構造はDAGではなくイベントのソース側に開いた木構造に制限される．また，以前のSimulatorでは抽象的に扱われていた観測装置の具体的な実装を行った．これにより，Nano-JASMINE Simulator は衛星システムや搭載機器に対応するいくつかのクラスを置き換えることにより，親ミッションであるJASMINEのSimulator としても使うことが出来るものとして設計されている．

本論文では，このSimulatorの設計・実装に関わるソフトウエア的な議論と，実際に衛星設計のための使用が開始されているのでその結果と使用例の報告を行なう．ソフトウエアに関する議論では，全体的なソフトウエア設計上の枠組に加え，天文，特に位置天文観測をサポートするためのソフトウエア部品の具体的な設計と実装，衛星バス部の実装について述べる．使用例としては，ミッション画像を用いた衛星姿勢制御について紹介する．

本論文の構成は，以下のとおりである．§2にNano-JASMINE Simulatorの設計及び実装，§3にコードコンベンションなどの若干のソフトウエア製品の説明，§4に動作例として姿勢制御系とミッション望遠鏡の協調作業のシミュレーション，§5にまとめを記述する．

2．設計および実装

（Nano-）JASMINE Simulatorでは，観測に関わる装置の状態変化を起こす事象を「イベント」として抽象化している．ここで，イベントとは次のようなものである．

- 天体が発する光子
- TDI 信号
- シミュレーション終了信号
- 視野更新信号
- 検出器により変換された電気信号（イベントとしては未実装）
- 光学系や検出器に影響を与える放射線などのノイズ（未実装）

Simulatorの基本設計は，連続するイベントを時間順序に従って処理してゆくものである．光学系は天体が発生する「光子」イベントを監視し，イベントが発生すると回折や反射などの動作を行ない，検出器は光学系が処理した「光子」イベントが検出器に到着するのを監視し，露出を行なう．また同時に検出器はTDI信号を監視し，信号を受けるとTDI動作を行なうといった具合である．ここで光学系から監視される天体は視野に含まれる天体に限ってよく，そのため，視野を適切なタイミングで変更し，視野に含まれる天体のリストを更新しているのが視野更新信号の役割となる．この他にも，観測に影響を与える宇宙線やさまざまなノイズは，イベントとしてこの枠組の中で取り扱うことが可能である．なお，衛星の軌道・姿勢制御擾乱はイベントとしての取り扱いも可能だが，光子に比べて時間スケールが非常に長いこと，および衛星システム側とミッション側の開発の独立性を確保することから，このシミュレーターに含まれるイベント処理とは独立に考慮することとした．

現在のシミュレータの構造は，擬似的なイベント駆動型であるが，厳密な意味でイベント駆動型ではない．いわゆるイベント駆動型は外部から与えられるイベントでプログラムが駆動されるものだが，現実装ではイベント発生源とイベント処理機構を分離しておらず，シミュレーター自身がイベントを発生させながらこのイベントを処理して

いるため，厳密な意味でのイベント駆動型とは異なる実装となっている．

2.1　イベントとイベント処理機構

ミッション系では，天体が発する光子の方向やエネルギーなどの属性を，光学系が反射・屈折・回折などにより変更し，検出器がこれを受け取ってA/D変換を行うことにより最終的に数値に変換することを模擬する機構を実装する．このため，光子やこれが変換された電気信号などを「イベント」として抽象化し，これを光学系・検出器・オンボードコンピューターなどイベントを処理する装置を順次通過するというモデル化を行なう．同じ光子（あるいはその副産物）が同じ装置を二度通過することは無いと仮定する．この場合，イベント処理装置どうしはソース側に開いた木構造で結ばれている「ノード」として記述することが可能となる．本シミュレーターでは，グラフがDAGで書けているという仮定のもとで，各処理装置ではイベントを発生時刻の順序に処理することが可能であるような処理を行なう．

ミッション部は，天体から放出される光子を望遠鏡や検出器で検出する機構の実装，およびそれに伴う擾乱や誤差の取り扱いである．この実装において，光子および擾乱を「イベント」として抽象化し，望遠鏡や検出器を「イベント処理装置」として抽象化した．ここで，イベント処理装置の役割は，上流で発生したイベントを受け取り，処理し，下流のイベント処理装置に引き渡すことであり，この一連の流れのなかで引き渡されるものがイベントである．

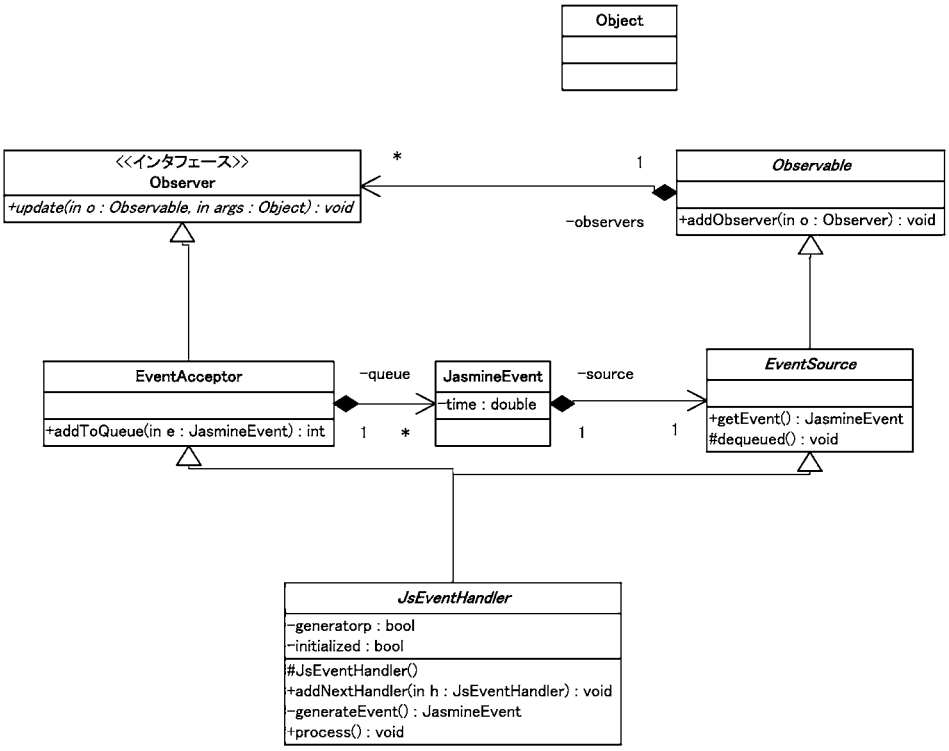

図 1．イベント処理機構のクラス図。

EventAcceptorクラスはイベントのキューイングと時刻順序への整列を、EventSourceクラスは要求に応じたイベントの引き渡しを担当する。これら両者の特性を継承したJaEventHandlerクラスが継承され、光学系や検出器などのイベント処理装置を構成する。

この機構の抽象クラスの基本構造は，図１に示すとおりである．この機構の最上位クラスはObserver およびObservableであり，これはGoF[3,4]のObserverパターンの応用である．設計上は，イベントの処理はその発生だけを担当するEventSourceクラスと，イベントを受け付けて処理を行うEventAcceptorクラスからなる．EventAcceptorはObserver，EventSourceはObservableである．多くのイベント処理装置は，その両者を多重継承したJsEventHandlerクラスのサブクラスとなる．javaが多重継承を認めない仕様からEventAcceptorクラスは実装されず，必要

なメソッドや属性はJsEventHandlerクラスの中で実装されている．実際のイベントの受渡し順序は，シミュレーション開始時にJsEventHandlerクラスの連鎖を定義することで決められる．

イベント取り扱いの設計方針として，イベント処理装置の相互関連に関する取り扱いおよびキューイングについては図１に示すようにJsEventHandlerクラスおよびJasmineEventクラスで完結し，イベントの詳細な取り扱いについてはそれぞれのサブクラスにまかせる設計となっている．これにより，新たな種類のイベントを追加しても，Simulatorの全体構造に影響を与えること無くソフトウエアが拡張可能となる．

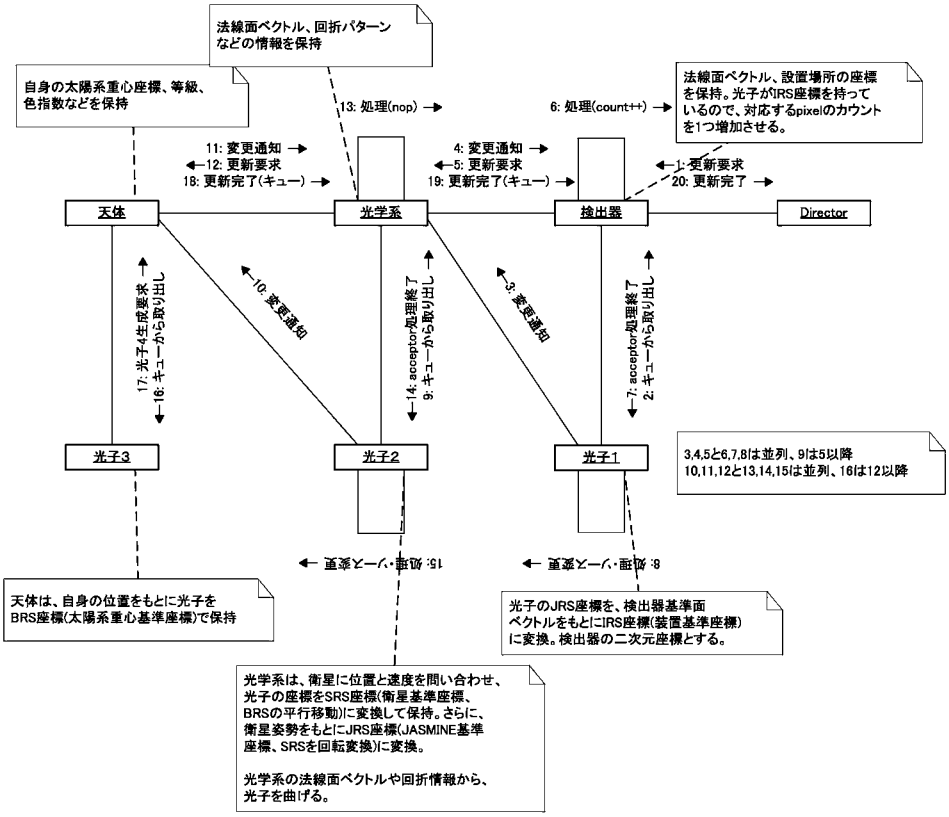

図２：イベント処理機構の協調図。例では、天体、光学系、検出器の３種類のイベント処理装置の間に順次光子「イベント」が受け渡されて行く様子を示している。「天体」の位置はBRS座標系(太陽系重心基準座標系)で保持されていて、「光子」が「天体」に保持されている時その位置はBRS座標系で与えられる。「天体」が視野に入ると「天体」は「光子」を出す。これが「光学系」に捉えられると、「光学系」は「衛星」に位置を問い合わせ、「光子」の座標をSRS座標系(衛星基準座標系、BRS座標の並行移動) へ、さらに衛星姿勢を問い合わせてJRS座標系(JASMINE基準座標系、SRS座標系を回転変換したもの)に変換する。「光学系」は、自身が保持する法線面ベクトルと解説パターンの情報をもとに、JRS座標系で光子の方向と位置を変化させる。その「光子」が「検出器」に捉えられると、「検出器」が保持する法線面ベクトルをもとにIRS座標系(検出器基準座標系)に変更し、「光子」があたる「検出器」面上の座標値を計算し、対応するpixel値を1増加させる。座標系の記述については2.2章を参照。順次イベントを処理するため、最終段の「Director」はイベントを順次取り出して行く役割を持つ。

JsEventHandlerクラスのサブクラスとしては，天体，光学系（ビーム混合鏡，望遠鏡），検出器，検出器制御装置，データ読取装置などが実装される．検出器制御信号および視野変更信号は，衛星姿勢と連動するため，このタイミングを衛星バス部から取得したり，バス部からタイミングを制御する必要がある．また，画像データは最後のデータ読取装置からバス部へ引き渡される．これらは，信号の上流側をObservable，下流側をObserverとして順次結合され，上流で発生したイベントが順次下流に引き継がれてゆく．この様子を図２に示す．

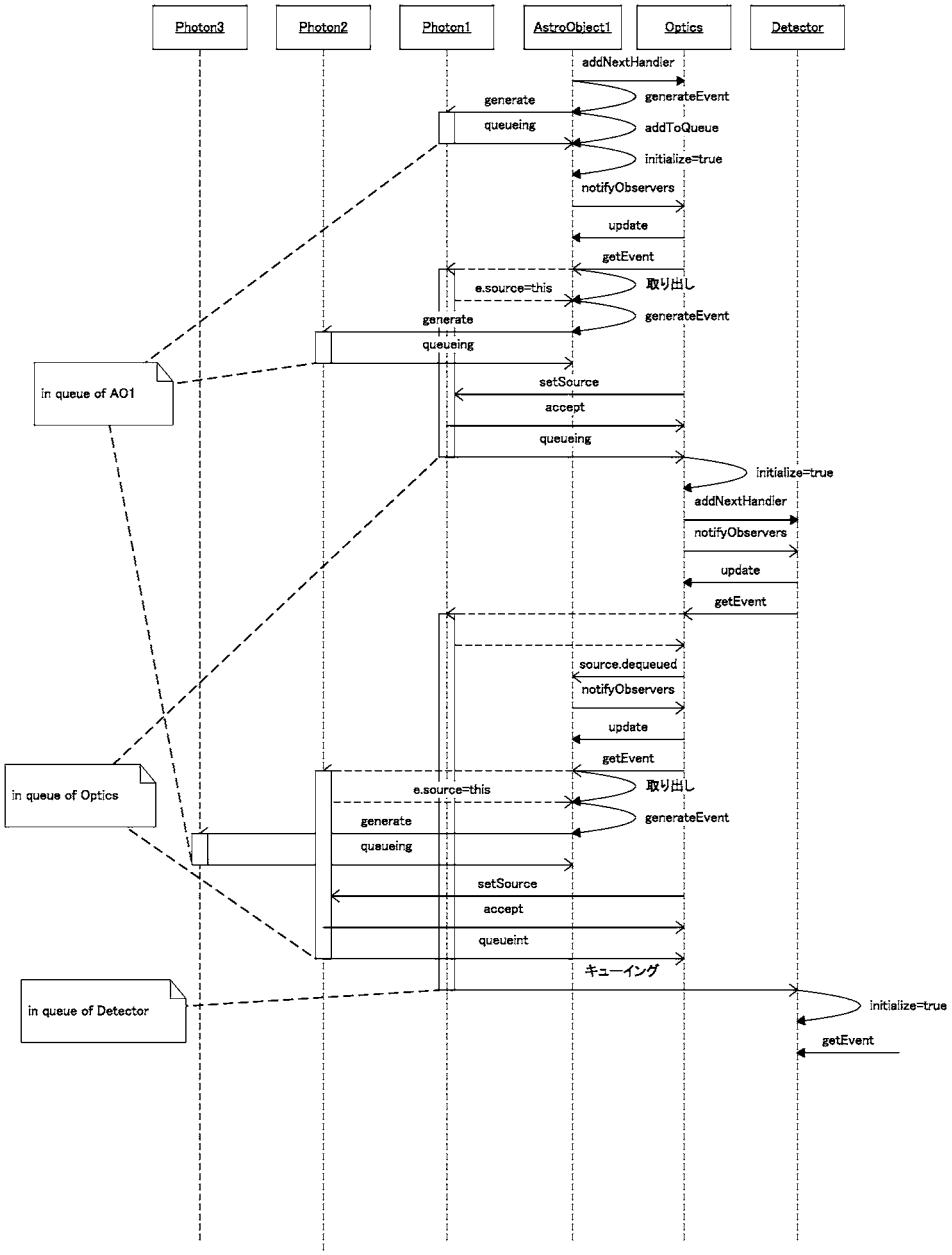

図 3 ：イベント処理機構の、図 2 に対応するシーケンス図。「AstroObject」が「天体」に、「Optics」が「光学系」に、「Detector」が「検出器」に対応する。

イベントの処理順序を時間順序と整列するため，イベント処理装置は上流の複数のイベント処理装置からのイベントを蓄え，下流装置からの要求があれば時間順序でこれを引き渡すためのキューを実装している．また，自身のキューが消費されると，必要なイベントを上流にとりに行く．キューイングの流れは，図 3 に示す通りである．

イベントとイベント処理装置の関係は，イベントがacceptor，イベント処理機構がVisitorとなっている．これは，イベントの受け渡し自体はJasmineEventおよびJsEventHandlerクラスのレベルで行いたいが，イベントとイベント処理装置の双方の型の解決後に実際の処理の内容が決まるため，型解決が必要となるためである．実装は，以下のようになり，JasmineEventクラスの具象クラスが処理装置のprocessメソッドを呼ぶ際に，自身の型を通知する，GoF[3,4]のVisitorパターンの応用となっている．以下に実例を示す．

```
class JsEventHandler{
  abstract protected void process（ConcreteEvent e）;
```

```
}

abstract class JasmineEvent{
  abstract void accept（JsEventHandler h）;
}
```

```
class ConcreteEvent{
  void accept（JsEventHandler h）{ h.process
（this）; }
}
```

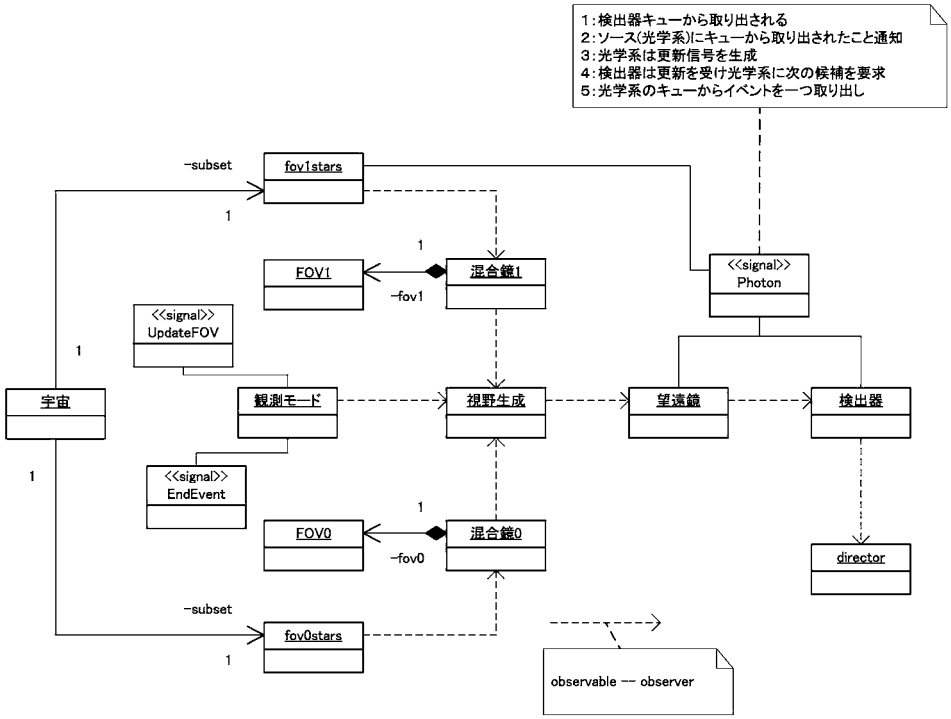

図4：イベント処理機構のオブジェクト図。Nano-JASMINEをシミュレートする場合のイベント処理装置オブジェクトの関係を示す。破線で示されるのがJsEventHandlerクラスのObserver-Observable関係であり、それぞれのオブジェクトは担当するいくつかのオブジェクトと関連を持ち、処理を行なう。視野が二つあっても、このような結びつけ方をすればDAGの構造を壊さずに実装可能である。

鏡が光子を反射し，検出器が光子を蓄積するなどの処理装置個別の処理は，イベント処理装置のprocessメソッド内で実装される．処理装置が処理しないイベントに対するprocessメソッドは，空メソッドとなる．新たな種類のイベントを追加する場合は，JasmineEvent クラスの子クラスとして新たなイベントを担当するクラスを追加し，JsEventHandler クラスに抽象メソッドまたは空メソッドを追加しすればよい．Nano-JASMINE Simulatorでは，図 4 のようにイベント処理装置を配置する．

シミュレーションをコントロールするためのいくつかのイベントを発生するため，「観測モード」と呼ばれる処理装置を作成する．観測モードのイベント処理装置は，衛星のノミナルな回転などの情報から視野更新信号を，シミュレーションを行なう総時間の情報からシミュレーション終了信号を生成する．シミュレーション終了信号を検出すると，全ての処理装置のObserver-Observable関係は解消される．星（天体）はPhotonを生成する．CCDはTDI信号を生成し，Photonを検出する．光学系はPhotonを反射，回折する役割を持つ．オンボード処理装置は，TDI信号を検出すると検出器から 1 カラムを読みだし，オンボード処理装置に蓄積する．

将来は，星は光子を出すクラスと位置通知イベントを出すクラスに分けられるであろう．光子レベルのシミュレーションは，現実をより忠実に反映している反面，計算コストががかかるため，ミッション全体をシミュレーションすることは難しい．そこで，発生するイベントを位置通知に置き換えることにより，ミッション全体のシミュレーションに対応する．

装置擾乱は，時間スケールが大きくことなるため，本実装ではイベントとして扱わないこととした．また，検出器や光学系・OBCに対する放射線

イベントの影響を取り扱う場合，これらもイベントとして追加される．

2.2 座標

ミッション機器は，自身の姿勢情報や位置情報を得るため，衛星システム側の情報を要求する．衛星システム側は，要求に応じてMatrixの形で位置情報と姿勢情報を返却する．

光子が光学系と検出器を通過する時間は非常に短いので，観測機器毎に衛星システムに値を要求することはオーバーヘッドを上げる．ベンチ等の熱変動と姿勢情報は独立に計算できるはずである．また，相対的な位置関係が重要であり，この量の精度に着目すれば，姿勢情報と熱変動は独立に返却されることが望ましい．

座標系は3次元ベクトルとして，座標変換は3次元直交行列として表現され，天体観測のための衛星シミュレーションに用いる座標の種類は非常に多い．天体の位置を表す太陽系重心座標，衛星の軌道に座標を平行移動した衛星基準座標，衛星機軸に回転変換したJASMINE座標，検出器面など装置面に射影された装置座標などがこれにあたる．行列やベクトルの演算を利用するためNational Institute of Standards and Technologyが提供するMatrix クラス（math.nist.gov/javanumerics/jama/）を利用しているが，それぞれの座標がどの座標であるかを認識し，また演算を安全に行うためには，型付けが便利である．このために，それぞれの座標クラスや変換行列のクラスはMatrixクラスを継承している．

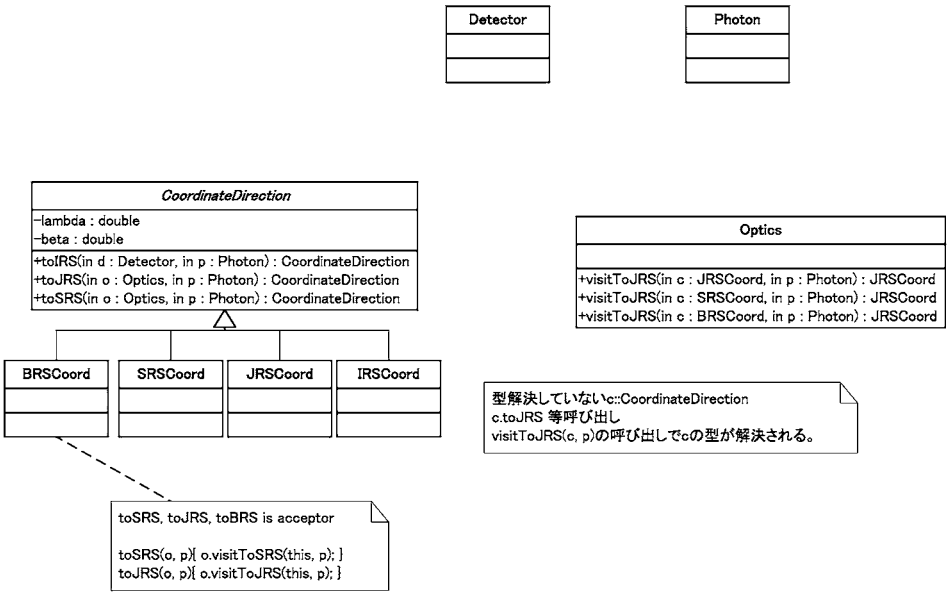

図5：座標取り扱いに関するクラス図。座標系は抽象クラスCoordinateDirectionクラスに括られており、一方反射や回折などを行なう光学系クラスなどは具象クラスのVisitorとして実装され、visitメソッドを持つ。

PhotonがJsEventHandlerのacceptorとして作用する部分では，Photonの方向の変化を扱わなければならない．本来なら，Photonクラス内で

newC = c.toJRS（o, this）;
setCoord（newC）;

としたいところだが，Photonクラスのacceptメソッドは抽象イベント処理装置型であるため，光学系特有のパラメータを持つtoJRS関数の特性は使えない．processメソッドに引き渡された後，JsEventHandler::OpticsからJasmineEvent::Photon.setCoord（）を呼ぶ方法も考えられるが，この場合setCoord（）関数は外部から呼び出される必要があり，publicにする必要がある．setCoord（）をpackage privateにするにはPhotonをopticsパッケージに移動する方法もあるが，他のvisitor/acceptorの安全性と矛盾するのでこの解決は別の危険性を生み出すだけである．座標系をreplaceするためのメソッドを用意してOptics::processメソッド から呼ぶ方法は，replace なら座標の視点を変えただけで意味は変えないからpublicにしても安全である．しかし，同様に反射・回折でも座標系の変更が必要となる．従って，

いずれにしても座標を変更する方法が必要となる．そこで，Photon にdiffractやreflect関数を用意し，Opticsからこれを呼び出す仕様を採用するのが，安全性と利便性から最も適切である．

そこで，座標系の変換に図 5 のようにVisitor[3,4]を適用する．Photonは方向を持っているが，これを抽象化するのがCoordinateDirection クラスである．実際には，BRS（太陽系重心）座標から軌道情報を用いてSRS （衛星重心）座標へ，更に姿勢情報を用いてJRS（JASMINE基準）座標へ，さらに焦点距離情報を用いてIRS（装置基準）座標へ変換される．これらの変更は必ずこの順番に行われるが，今Photonが保持する座標が何であるかを保障するものはCoordinate Directionの具象型だけである．この型を解決するため，visitorパターンを用いている．Opticsがvisitor，座標がacceptorとなる．この場合は，visitor側は階層化される必要は無い．

なお，年周視差がゼロである場合のシミュレーションが多用されるため，このクラスに実装する関数は年周視差がゼロの場合（距離無限大）に特異にならないよう配慮するべきである. このため，星の等級も実視等級を保持する．

BRS Coordinateは，位置天文パラメータを用いて以下のように与えられる．

$$(b,l,\pi)=\left(\lambda+t\mu_{\lambda},\beta+t\mu_{\beta},\pi\right)$$

SRS CoodinateはBRSCoordを衛星軌道に平行移動した座標．現在光行差は未実装だが，このクラスで対応すべきである．衛星の位置をb，BRS座標の単位ベクトルをcとすると，SRS座標の三次元座標は

$$\frac{\pi}{\mathrm{AU}}b-c$$

を規格化したものとなる．

SRSCoordに衛星の姿勢を考慮して直交変換したもの．衛星姿勢を現す行列を$\mathbf{M}$，SRS座標の単位ベクトルを$\boldsymbol{c}$として，JRS座標は

$$\mathbf{M}\boldsymbol{c}$$

で与えられる．

JRSCoordを装置面基準に直交変換したものを，IRS座標として用意する．

これらの他，三次元直交変換を表すクラス，相互に変換する行列クラス，各座標に対応したベクトルのクラス，位置のクラスなどを用意し，型解決で変換を可能にしておく．

2.3　天体および宇宙

天体および宇宙は，compositeパターンにより実装される．Leafクラスとして，現在SingleStarが実装されている．binaryやNEOなどを表すクラスを追加実装する予定である．

宇宙モデルは天体の集約として表現される．現在，天体のLeafクラスは光子を出す．しかし，光子レベルでのシミュレーションは実際の観測の数倍の時間を要するため，大円一周程度のシミュレーションは可能でもミッション全体にわたるシミュレーションは不可能である．そのため，個別の光子ではなく光子の個数と平均位置，分散だけを示すようなLeafクラスを新たに実装することが望ましい．そのため，集約の生成にはbuilderにより実装される．

宇宙モデルに対して，走査・検索やカタログ生成などは，天体クラスのvisitorとして実装される．

天体は，光子イベントあるは位置通知イベントのソースとしての意味を持つ．JsEventSourceの直系サブクラスとなる．光子イベントを持つか位置通知イベントを持つかは，シミュレーションの粒度により選択される．

宇宙は天体のcollectionクラスとして実装される．構造を表現するクラスの構造は図 6 上に，構造を生成するクラスの構造は図 6 の下に示す．

collectionクラスである宇宙モデルの生成は，BuilderおよびFactoryを用いて実装する（図 6 左）．天体は，異なる種類のイベントに対して異なる具象クラスを持つ．従って，Factoryクラスは Abstract Factory[3,4] で実装する．

走査はVisitor[3,4] クラスが担当する（図 6 下）．光子イベントを用いる場合に視野に入るかどうかを判定する方法，および位置通知イベントを用いる場合に観測時刻を返す方法は，Visitorが内部クラスとして実装する．

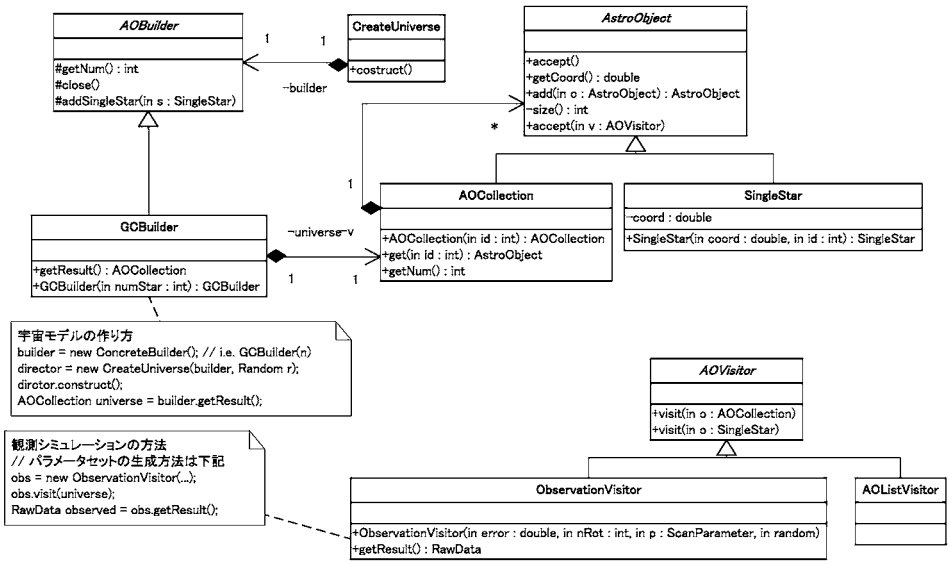

図６：天体を表すクラス。個別の天体として、現在SingleStarクラスが実装されている。宇宙モデルや視野内に入る星などの天体の集約クラスとして、AOCollectionクラスが実装されている。これは、Compositeパターンの応用である。これらを生成するためのBuilderくらすおよび処理を行なうためのVisitorクラスが別途存在する。

２.４　定数

定数は，インターフェース内でstatic finalで記述されている．

２.５　入出力

天文データの入出力として，fitsフォーマットを実装している．現実装ではsimple fitsのみに対応しているが，設計上はnormal fitsへの拡張を考慮している．

また，複数種類のシミュレーションを実行したり，それぞれのシミュレーション独自のパラメータを入力するためのGUI，画像を画面に出力する機構を実装する．

FITSを扱うクラスは，構造を表すクラス（図７下にクラス図を示す）とこれを生成するためのクラス（図７上にクラス図を示す）からなる．

２.６　視野

実際には，観測にかからない多くの方向に光子が放出されている．しかし，これらを全てシミュレーションすることは非現実的で，見えるものだけを予めselectする必要がある．

まず，装置側からみた視野は，装置の視野情報を用いて観測対象となる星を選択する形で行われる．

逆に星側から見た装置は，自ら放出する光子の頻度を決めるため，視野の大きさに関する情報を知っている必要がある．

２.７　パラメータ

ここで言うパラメータは，解析・推定されるものである．パラメータセットと推定手法を組み合わせて一つのクラスとして実装する．推定アルゴリズムを樹柔軟に変更するため，アルゴリズムについてはStrategy[3,4]パターンを適用するべきである．

２.８　ミッション系とバス系の結合

ミッション系からバス系を制御する必要は無いが，シミュレーションにおいてはミッション系は衛星姿勢を知る必要がある．そこで，バス系の衛星はAbstractSatelliteクラスを継承した具象クラスを持ち，ミッション系から姿勢情報などを得るためのメソッドを実装する必要がある．

一方バス系からはミッション系に対する制御信号が送られる．これは，実際の衛星の場合はミッションOBCを介して行われるものである．このクラスはObdhクラスで実装される．

即ち，ミッション系側はObdhクラス，バス系はAbstractSatelliteクラスがFacade[3,4]になって

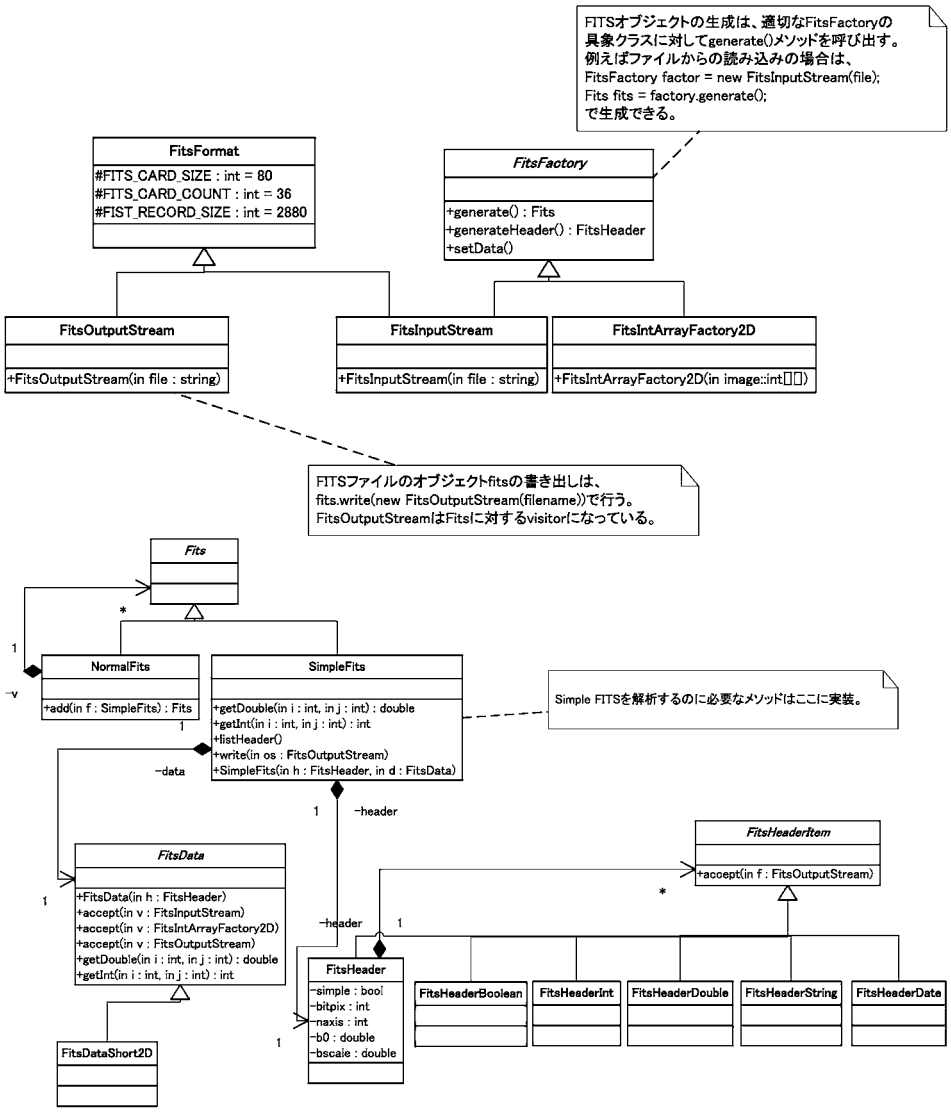

図７：FITSの生成を行なうクラス（上）と構造を表すクラス（下）から成る。

いる．双方が相手の具象クラスを持たなければならないので，次の手順でこれを行う．

１．バス系のクラスをインスタンス化する．

２．ミッション系のクラスのインスタンス化で，バス系クラスを引き渡す．

３．バス系クラスのインスタンスsatにおいて，sat.setObdh（o::Obdh） を呼び出すことにより，バス系にミッション系のFacadeクラス情報を与える．

２.９　バス部実装機能

Simulatorバス部の実装機能について述べる．Simulatorは衛星のフィージビリティー確認だけでなく，衛星打ち上げ後の挙動解析，不具合対策にも用いることを想定しているため，一般的な衛星の姿勢運動，軌道運動だけでなく，各種コンポーネントレベルまで詳細に記述する必要がある．図８にSimulatorバス部に実装される機能のシステムブロック図を示す．Simulator内部では図８のシステムブロック図に記されたクラスごとに実装されている．

２.９.１　衛星の軌道運動

衛星の軌道運動はOrbitクラスとして処理される．OrbitクラスはBRS（太陽系重心）座標を基準として記述されており，地球の扁平性に基づく軌道運動学を取り扱う．ミッション系から位置情報の要求があった場合には，軌道六要素により定まる衛星の位置情報を，インターフェイスを介してミッション系に返却する．また，解析手法の確

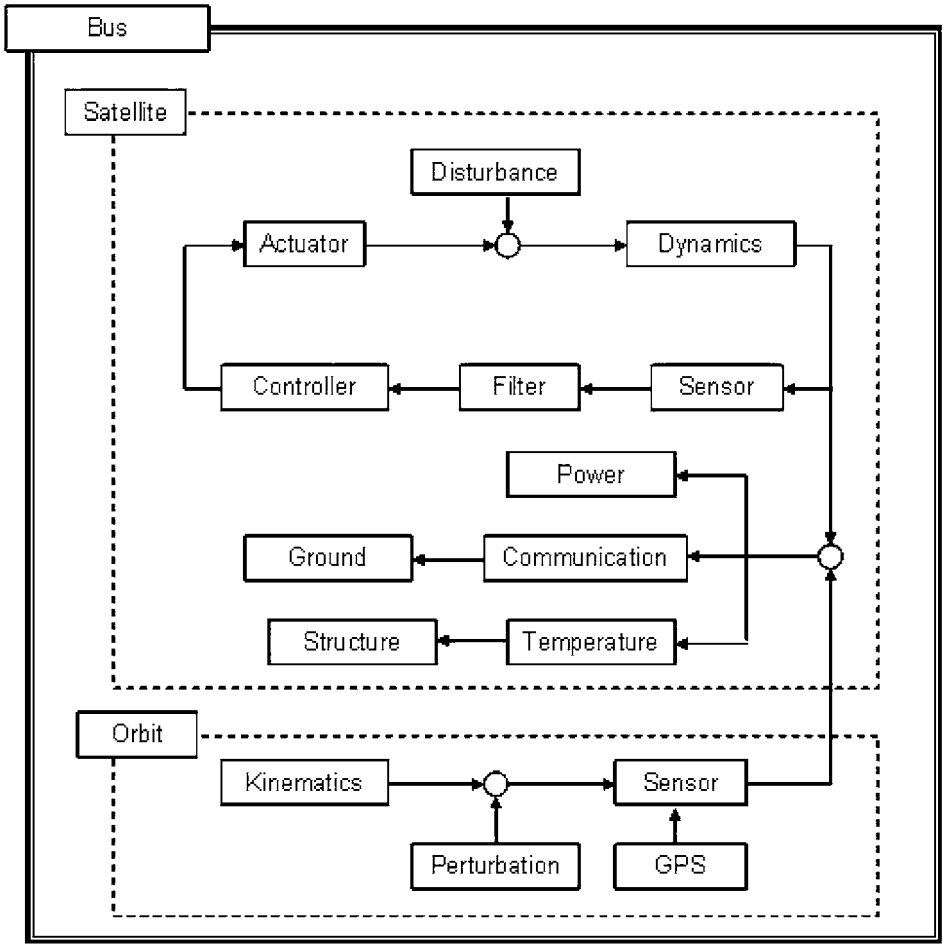

図８：Simulatorバス部のシステムブロック図

立や観測模擬データの作成のために，GPS受信機の動作を模擬した位置情報取得クラスが実装される．

2.9.2　衛星の姿勢運動

Satelliteクラスは衛星の動力学と運動学に関する部分を処理する．衛星の状態量は衛星姿勢角を示すQuaternionと角速度を用いて記述されており，それぞれの微分方程式を数値積分によって解くことで状態量の更新を行う．

2.9.3　衛星コンポーネント

衛星コンポーネントはセンサであるSTT（Star Tracker）やアクチュエータのRW（Reaction Wheel）といったハードウェアと，OBC（OnBoard Computer）によるソフトウェア処理部分からなり，それぞれの機能ごとのクラスに分けられ実装されている．まず，センサクラスには地球センサ，ジャイロ，地球センサ，磁気センサ，恒星センサ，太陽センサが実装されており，それぞれのコンポーネントに関して精度や更新周期などのパラメータを設定できるようになっている．フィージビリティスタディの際には搭載を考えているセンサのパラメータを代入することで，搭載の可否を決定することができる．つぎに，フィルタクラスでは各種センサより得られる情報から，より正確に衛星の状態量を推定するために，最尤推定法の一つであるカルマンフィルタによる姿勢推定フィルタが実装される．コントローラ，アクチュエータクラスでは，フィルタクラスによる状態量推定結果を用いて制御トルクの指令値の計算を行い，アクチュエータの運動学，伝達特性を考慮した上でSatelliteクラスへ制御トルクを返す機能が実装されている．Nano-JASMINEのように極めて高い姿勢安定度が要求される衛星ではセンサやアクチュエータの選定だけではなく制御則も十分に検討する必要があり，コントローラクラス内部を書き換えることで各種制御則の適応の可能性について検討することができる．最後に，電源，通信，地上局，温度，構造については，衛星－地

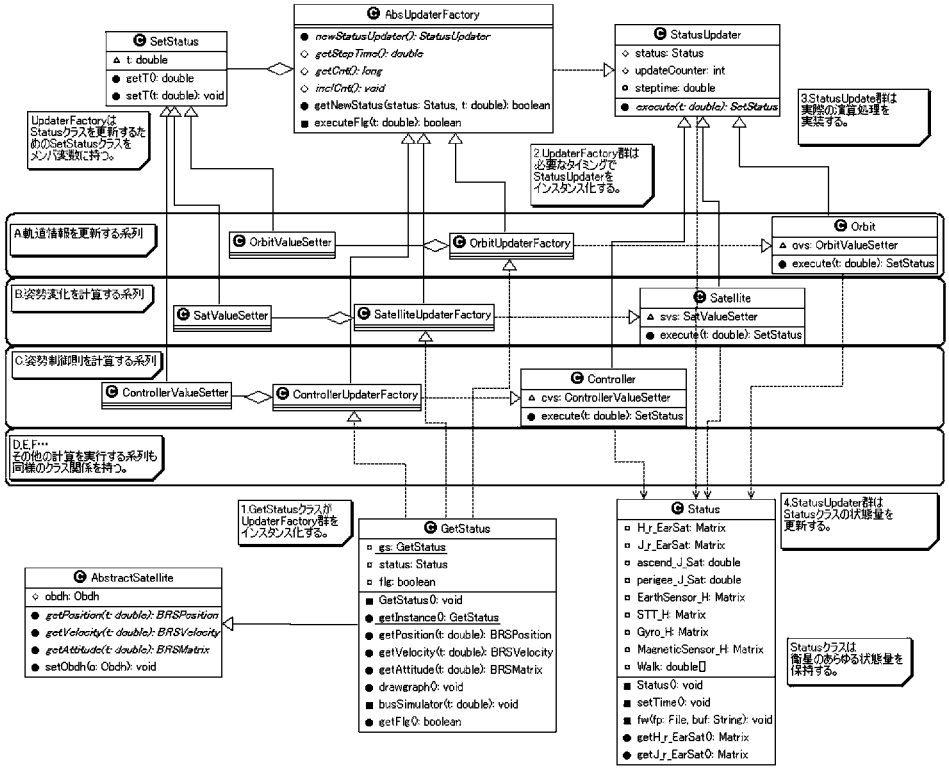

図９：Simulatorバス部のUML図

球－太陽の幾何学的関係から，通信可能時間の算出，衛星内部の温度状態の時間変化，電力収支の確認を行う昨日が実装される．これらのクラスの実装により，打ち上げ－運用－解析といったプロジェクト一連の流れをシステムレベルで詳細に検討可能なSimulatorとなり，概念検討のみならず，観測模擬データの作成や打ち上げ後の不具合対策にも使用可能な汎用的Simulatorとして用いることができる．

２.10　バス部実装

２.10.１　バス部実装概要

Nano-JASMINEシミュレータにおいてバスシステム側のシミュレータは次の機能を担う．

・衛星の運動方程式を解くことで，ミッション側から要求された時刻における衛星の位置情報，速度情報，姿勢情報をミッション側シミュレータに渡す．

・衛星に搭載されるADCS機器をモデル化し，各種センサー出力から衛星の状態量を推定し，各種アクチュエータを用いて衛星の姿勢や温度を制御する過程を模擬する．

ミッション側シミュレータとバスシステム側シミュレータは開発の独立性を高めるために，両者の間のインターフェイスは最小限にとどめてある．また，ここでNano-JASMIN用に開発されたシミュレータの基本構造を将来JASMINEシミュレータに容易に転用できるようにするため，JAVAのオブジェクト指向性を利用して汎用性のあるフレームワークを構築することが望ましい．

２.10.２　クラス関係

ParameterクラスとStatusクラス

Parameterクラスはシミュレーションに必要となる基本定数（物理定数や衛星の質量など，シミュレーションを通じて値が変化しないもの）を主にpublic finalのアクセス修飾子で記述している．一方Statusクラスはシミュレーションを通して絶えず変化する値（衛星の姿勢や温度などの状態量）を保持し，これらをカプセル化している．これら２つのクラスはともにSingleton[3,4]を実装したコンストラクタを持ち，それぞれシミュレーション全体を通じてオブジェクトは常に１つだけ存在する．Parameterオブジェクト内の変数をfinalで宣言することにより，これを利用するクラスが不用意に定数の値を変更する事を防いでいる．同様にStatusオブジェクト内の変数をprivateで宣言す

ることでカプセル化し，一定の手続きに従わない限り状態量の更新が許されないようにすることで，意図しない箇所から値が変更されることを防いでいる．これらのオブジェクトをSingletonにすることにより，次のメリットがあると考えられる．

・状態量を保持したオブジェクトが複数存在すると，状態量を変更する際にすべてのオブジェクトにアクセスしなければならず，間違いが発生しやすくなる．オブジェクトが唯一であることを保証することでこういった間違いをなくすことができる．
・ともに大きなメモリ領域を必要とするオブジェクトであるため，シミュレーション中に何度も生成と消滅を繰り返すことがあると計算負荷になる可能性がある．

SetStatusクラス

カプセル化されたStatusクラス内の状態量を更新するためには更新後の状態量をSetStatus型のオブジェクトに格納して，これをStatusクラスに渡す必要がある．実際には状態量の種類ごとに，抽象クラスSetStatusを継承したOrbitValueSetter（軌道に関する状態量の更新値を保持）やPowerValueSetter（電力に関する状態量の更新値を保持）などにまとめられ，Statusクラスへ渡される．すなわちSetStatusクラスはフレームワークでOrbitValueSetterやPowerValueSetterは実装であるといえる．

OrbitValueSetterやPowerValueSetterなどのSetStatusクラスを継承したオブジェクト群は，以下に述べるStatusUpdaterクラスを継承したクラス群（それぞれ軌道に関する状態量や電力に関する状態量などを計算するモジュールである）によって生成される．

StatusUpdaterクラス

本シミュレータでは，あらゆる変数（姿勢の真値を表す変数群，温度を表す変数群，各種センサー出力を表す変数群，姿勢推定値を表す変数群など）はStatusオブジェクトに保持されており，これらを更新する一連の流れはすべて同じ枠組みの中で行われる．

たとえばこれらの変数のうち軌道情報に関する変数群を更新したい場合，抽象クラスStatusUpdaterクラスを継承したOrbitオブジェクトを用いる．StatusUpdaterは抽象メソッドexecute () を持っているのでOrbitクラスもexecute () メソッドをオーバーライドしなければならない．実際の軌道要素を計算するロジックはこのexecute（）メソッドに記述する．また抽象メソッドexecute（）は抽象クラスSetStatusを返り値にとるように指定してあるので，Orbitクラスでオーバーライドしたexecute（）メソッドもSetStatusクラスを継承した何らかのオブジェクトを返さなければならない．このときにOrbitのexecute（）メソッドでは，SetStatusクラスを継承したOrbitValueSetterオブジェクトを生成し，軌道要素の計算結果を格納して返すようにする．

同様に衛星の姿勢に関する変数群を更新する場合にはSatelliteクラスのexecute（）メソッドを用い，計算結果をSatValueSetterオブジェクトに格納して返すようにする．他のあらゆる種類の変数についてもこの更新方式で統一する．このように変数の更新方式を統一することで次のメリットがあると考えられる．

・プログラムの再利用性を高める．
　各変数群を更新する計算モジュール（Orbit，Satellite，Power，etc.）を実行し変数の更新を行うのはAbsUpdaterFactoryクラスであるが，AbsUpdaterFactoryクラス内の記述ではその計算モジュールの実態にかかわらず，その上位クラスのみを用いて「StatusUpdaterオブジェクトを生成してexecute（）メソッドを実行し，返ってきたSetStatusオブジェクトをStatusオブジェクトに渡す」といった抽象的な記述が可能となる．これによりあらゆる状態量更新の手続きで同じコードを共有することができ，この部分での間違いをなくすことができる．
・さまざまな実装方法が可能である．
　StatusUpdater，SetStatusなどの抽象クラスをフレームワークと見ることによって，シミュレータに新たな変数群を加える場合にも，これらの抽象クラスを継承することによって，信頼性の高い既存のコードを利用することができる．このような拡張性はNano-JASMINEシミュレータをJASMINEシミュレータに拡張する場合などに有効であると考えられる．

AbsUpdaterFactoryクラス

上述のOrbitオブジェクトやSatelliteオブジェクトなどを生成し計算モジュールであるexecute（）メソッドを実行するのは，抽象クラスAbsUpdaterFactoryを継承したOrbitUpdaterFactory

やSatelliteUpdaterFactoryである．こういった具象クラスの共通部分を上位クラスであるAbsUpdaterFactoryに記述することで，上述のように「StatusUpdaterオブジェクトを生成してexecute（）メソッドを実行し，返ってきたSetStatusオブジェクトをStatusオブジェクトに渡す」といった抽象的な記述が可能となる．これがAbsUpdaterFactoryクラスの第一の機能である．

AbsUpdaterFactoryクラスのもう一つの機能は，Orbitオブジェクトや Satelliteオブジェクトなど個々の計算モジュールを実行するかどうかの判定である．ミッション側では光子を一つのイベントとしているが，これはバスシステム側のシミュレータとしてはイベントの間隔が短すぎる．したがってバスシステム側のシミュレータはイベントごとに毎回計算モジュールを起動せず，ミッション側のシミュレータが要求してきた時刻がある程度進んだ段階で次のステップを計算することにする．各計算モジュールの実行ステップ間隔を何秒にするのが適切かについてはモジュールごとに異なるので，AbsUpdaterFactoryの具象クラスであるOrbitUpdaterFactoryやSatelliteUpdaterFactoryにsteptimeとして記述する．バスシステム側の計算実行間隔はミッション側のイベント発生間隔よりもかなり長いが，要求されてきた時刻における状態量を算出する場合には，その時刻の前後のステップでの計算結果を直線近時で内挿した値を返すことにしてある．この機能を実現するために，Statusは過去の数ステップの状態量を記憶しておくStatusHistory型のオブジェクトを保持している．このオブジェクトのlinearlize（）メソッドにより内挿計算が実現される．

実際に計算モジュールを実行するのは抽象クラスAbsUpdaterFactoryを継承したOrbitUpdaterFactoryやSatelliteUpdaterFactoryなどであるが，以上の機能は各計算モジュールで共通なので，上位のAbsUpdaterFactoryクラスに記述してある．

GetStatusクラス

これがミッション側にとってバスシステム側にアクセスする唯一のインターフェイスとなるクラスである．ミッション側シミュレータはこのクラスをインスタンス化したオブジェクトを常に保持しており，メンバ関数であるgetPosition（double t），getVelocity（double t），getAttitude（double t）により，それぞれ衛星の位置情報，速度情報，姿勢情報を取得する．それぞれの関数ではまずはじめに引数として渡された時刻から判断し，その時刻の状態量をミッション側に返すために必要な最小限のステップだけ計算ステップを進める．上述のようにこのときのステップ幅は，各計算モジュールごとに異なっている．この後，各計算ステップの間を直線近似で線形補完して，任意の時刻に対する状態量をミッション側に返している．

さらにGetStatusクラスはミッション側のAbstractSatelliteクラスを継承しているので，ミッション機器の状態量も参照することができる．Nano-JASMINEではミッション系のCCDデータをもとにバスシステム側が姿勢を制御するので，このことは特にバスシステム側がミッション系の保持する星像データを参照するときに利用される．

2.10.3　処理の流れ

以上はバスシステム側シミュレータの各クラスの機能について説明であるが，ここでは具体的にデータ処理の流れを示す．

1．まず，バスシステム側のシミュレータは，ミッション側シミュレータによりGetStatusクラスの3関数getPosition（double t），getVelocity（double t），getAttitude（double t）を呼ばれるところから始まる．
2．各計算モジュールに対応するUpdaterFactoryをインスタンス化する．すなわちOrbitUpdaterFactory，SatelliteUpdaterFactoryなどを生成する．これらのメンバ関数であるgetNewStatus（）を時刻を引数に実行することで，その時刻の状態量をミッション側に返すために必要な最小限のステップだけ計算ステップを進める．具体的には，バスシステム側のシミュレーションが，ミッション側から指定された時刻まで達していないとき，getNewStatus（）はtrueを返し続けるので，この返り値がfalseになったことで計算が完了したことを判断している．
3．要求された時刻まで既に計算が完了している場合，getNewStatus（）は，ただfalseを返すだけであるが，完了していない場合，getNewStatus（）の内部で相当する計算モジュール（Orbit,Satelliteなど）を生成し，結果をStatusに格納する．
4．Statusに格納されているすべての変数群が最新の状態になったら，StatusHistoryの

linearlize（）メソッドを実行して，線形補完された状態量を返す．

2.10.4　各計算モジュールの概要

Comm　通信に関する状態量を更新する．
Controller　姿勢制御命令に関する状態量を更新する．
Disturbance　衛星の姿勢外乱に関する状態量を更新する．
Filter　衛星の状態量を推定するカルマンフィルターに関する状態量を更新する．
Orbit　軌道に関する状態量を更新する．
Power　電力に関する状態量を更新する．
Satellite　姿勢に関する状態量を更新する．
SunSensor,STT,MagSensor,Gyro　それぞれ太陽センサ，スタートラッカ，磁気センサ，ジャイロに関する状態量を更新する．
SunShine　日照に関する状態量を更新する．
Temperature　温度に関する状態量を更新する．

3．コード開発上の既約など

3.1　動作環境

Java 1.5以上およびantの環境に対応している．また，Eclipseにも対応している．これらが動作する環境では動作する．行列演算のためのJamaライブラリについては，提供されているライブラリはJava 1.5対応だが，ソースファイルをツリーに組み込むことでJava 1.4にも対応出来るよう配慮する．一部の関数の使用法がJava 1.3には未対応である．Java 1.4については可能な限り対応する方針で開発をすすめる．

現在確認されている動作環境は，Windows XP Professional SP2上のJava 1.5，およびDebian Linux上のJava 1.4，いずれもIntel CPU環境での動作を確認している．Eclipseに関してはは Eclipse 3.1のWindows XP Professional上での動作を確認している．

3.2　コードコンベンションその他

日本語（非Latin 1文字）を用いる場合は，utf-8を用いるものとする．

Sun Java Coding Conventions（http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html）に従う．

トップディレクトリーにはsrcディレクトリー，build.xmlファイル，ChangeLogファイルを置く．docにはドキュメントを，libには結合するライブラリを置く．ThermalLibは熱関係のファイルを保管する．また，report, doc,
build, META-INF, dist, CVS, .settingsの名称は，ant・cvs・Eclipseが使用するディレクトリー名称なので，ユーザーがこの名称でファイルまたはディレクトリーを作成してはいけない．パッケージ名は，srcディレクトリーをトップにした名前に統一する．

物理量を表現する短精度および倍精度実数型変数の値は，表示のためなど特別の理由が無い限りSI 単位系での値であると仮定する．また，角度は特に理由が無い限りradianとする．これに関して例外を設ける場合は，ソースコード中のコメントに必ず使用した単位系を，理由とともに明記する．

3.3　評価，今後の課題，展望

・検出器のノイズなどの適切なモデル化
・現実的なPSFの導入や，視野の場所によるPSF形状の違いなど，光学系の詳細なモデル化
・視野内に入る天体のサーチの高速化⇒天体を階層化
・全ミッションスケジュールのシミュレーションに向け，位置通知イベントを持つ星の実装
・ObservationModeをミッション部全体のbuilder[3,4]クラスにする．
・複数の検出器を同時に搭載する場合に対応した検出器クラスの階層化
・解析手法を組み込む
・熱構造モデルの実装

シミュレータの高速化

現状，計算時間が実時間の約100倍掛かっている．この原因として以下の項目が考えられる．

・シミュレータ内のいたるところでJama.Matrixクラスが多用されているため．Matrixの初期化に計算時間が割かれている．
・バスシステム側シミュレータは新たなステップを計算するかどうかを判断するときにAbsUpdaterFactoryを継承した個々のUpdaterFactoryをインスタンス化しなければならない．ミッション側のイベントの発生す

る間隔は，バスシステム側の計算ステップ間隔よりもかなり短いため，ほとんどの場合新たな計算を実行する必要がない．にもかかわらず毎回UpdaterFactoryをインスタンス化するのは大きなオーバーヘッドになっていると考えられる．

１度に必要なシミュレーション時間はバス側で200分，ミッション側では数年である．そのため，バス向けには引き続き現在の精度のシミュレータをハードウェアとソフトウェアの面から高速化することで対応し，ミッション向けには簡易高速版の開発を行う．

シミュレータを開発しつつ，Nano-JASMINEの概念設計に適用することで，頑健な制御則の開発，衛星パラメータの感度解析などの設計作業が行われている．

４． 動作例

本シミュレーターはミッション全体に渡り多くの目的に使用される予定だが，緊急性を要する課題として，ミッション画像を用いた姿勢制御の可能性を検証する問題がある．位置天文観測は，通常衛星の姿勢を知るために使われる恒星の位置そのものを測定するミッションであり，そのためには衛星に通常搭載されるセンサーよりはるかに高精度の姿勢安定性が要求される．このレベルの姿勢擾乱を検出できる装置として最も有効なのは，ミッション側の望遠鏡である．そこで，ミッション側の望遠鏡の画像を処理することで，衛星姿勢のノミナル値からのずれを推定し，微弱な制御をかけて，ミッションが目標とする姿勢に落しこんで行く手法を考えているが，この方法が本当にうまく行くかどうかについて検証を行なう必要があった．

図10：制御前の星像イメージ。衛星搭載の標準的なセンサーの情報だけから姿勢制御を行なった場合、衛星姿勢は十分な精度が得られず、回転速度や軸方向がずれているために、斜めに伸びたようなイメージとなる。

衛星に搭載するセンサーで追い込んだ衛星の運動状態では，図10の状態の画像が得られる．回転速度や回転軸がノミナル値と若干ずれているため，

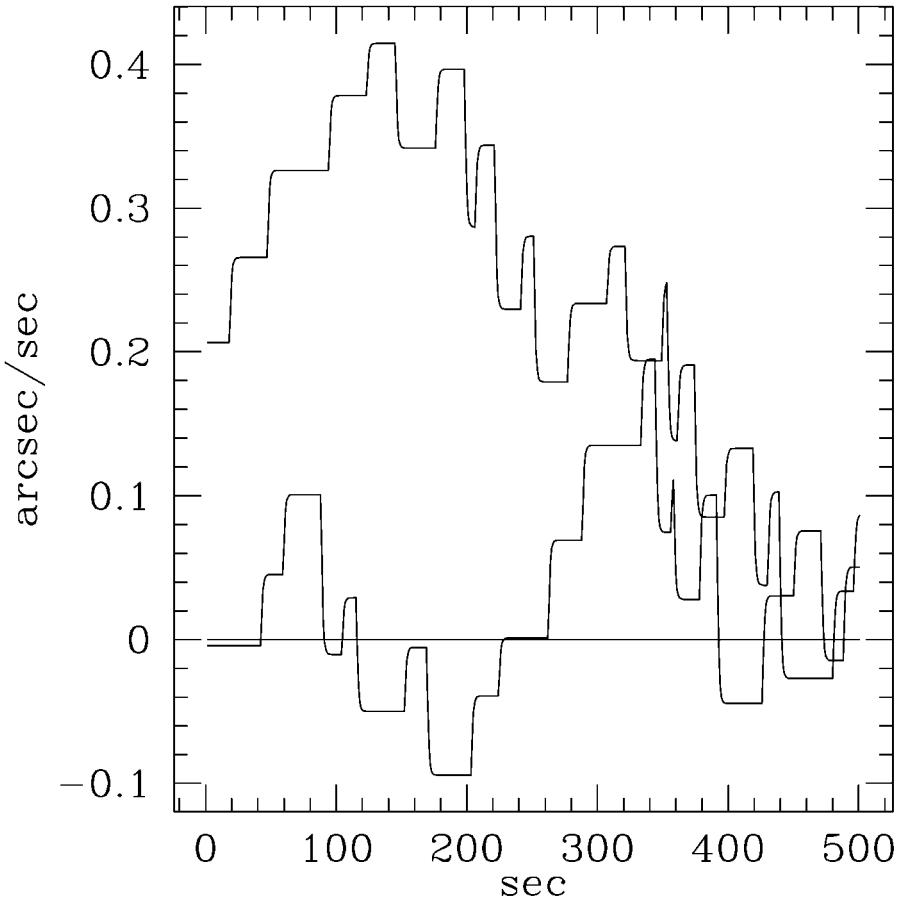

図11：制御の様子。軸の動く速度のノミナル値からのずれを示す。上がx、下がy、ゼロで示された直線は回転速度成分zを示す。回転速度成分は、衛生姿勢制御ではなくTDI速度の変更で制御されるため、この図では常にノミナル値であるように示されている。xおよびy成分は、微小なトルクを発生して制御を行なう。画像からは、ずれの絶対値は推定可能だが符号は推定不可能なため、制御を行なった結果を見て制御を行なった方向が正しいかどうかを判定し、さらに制御を続ける。約500秒程度の間に制御が完了し、撮像に十分な姿勢精度に達していることが分かる。

星像は図のように歪んだものとなる．

そこで，この歪みを推定し，ずれを定量化して，制御をかける．制御の様子は図11のように，微弱なトルクを発生することで衛星の回転軸や回転速度を制御する．得られている画像は約８秒程度積分されたものであり，画像からずれを推定する手法をとる限り，ずれの絶対量を推定する可能性はあるが，ずれの符号を知ることは出来ない．すなわち，例えば星像が画面の左下から右上に延びている場合，星が画面の左下から右上へ移動したのか右上から左下に移動したのかを知ることは出来ない．そこで，本制御では，最初にずれの符号を仮定して制御を行ない，この結果画像がより良質のものになるなら同じ方向へ制御をかけ，画像が更に悪化するならあらためて逆符号の制御をかけることにより，制御を行なっている．この例では最初の符号推定が誤っていたため，図11では一度悪い方向に推移し，後にゼロ付近に収束している．この数値実験では，符号情報が不定のまま制御を行なっても，500秒程度でほぼ期待通りの姿勢に追い込むことが出来ることを示している．

その結果，図12のような画像が得られ，ミッション画像を用いた姿勢制御が可能であることが，このシミュレーターを用いて検証された．

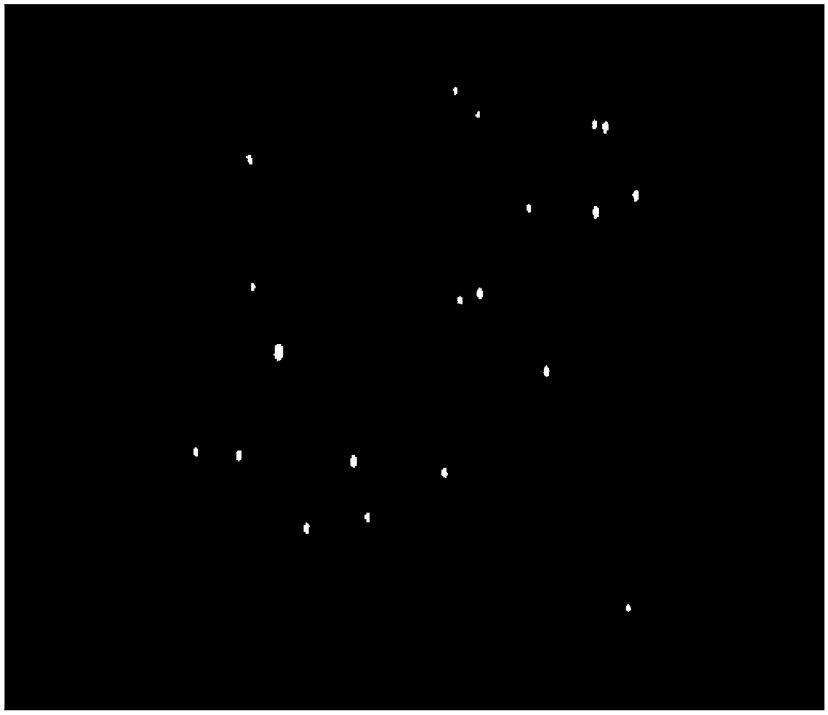

図12：制御後の星像イメージ。最初の斜めに伸びたイメージから、期待するおよそ$5¥times9$pixel程度に広がった、まっすぐなイメージが得られる。

5．まとめ

我々は，従来のJASMINE Simulatorの抽象的枠組みを発展させて，より具体的な観測を模擬するためのNano-JASMINE Simulatorを開発した．観測装置の状態を変化させる要因として，光子だけでなくさまざまな制御信号やノイズなどを「イベント」という抽象化で統一的に扱うことで，観測の模擬が容易になった点が，本シミュレーターの新しい点である．Nano-JASMINEの特徴として，衛星搭載姿勢センサーでは十分な姿勢安定度を達成できず，ミッション画像を用いることが必要となる．この「ミッション画像の処理による衛星姿勢制御」の可能性はミッションフィージビリティーに係わる重要な点で，この検証をシミュレーターを用いて行ったので，このシミュレーターの動作例として示した．

Nano-JASMINEは，2008年の打ち上げを目指している．このため，現在の１万秒程度までのシミュレーションに加えて，約1.5年のミッション全期間のデータを集めて十分な精度を出せるかどうかの検証まで含めて，早急に行う必要が在る．そこで，今後はより現実的な誤差要因を採り入れたシミュレーションシステムに改良し，全体の解析精度の評価，予期しない問題が発生した場合の対処方法などの検討に役立てられるよう，改良される予定である．

付録Ａ　用語集

ここでは，本稿に用いられている専門用語に関して，カテゴリーに分けて簡単な解説を与える．

Ａ.１　情報技術・データ解析技術

プログラミングのスタイルや技法，プログラミングのためのプロジェクトにおける既約などに関する用語は以下のとおりである．

イベント駆動型プログラム　起動するとともにイベントを待機し，起こったイベントにしたがって処理を行なうプログラミングスタイルを言う．通常のGUIプログラミングはマウスやキーボードからのイベントを処理するイベント駆動型で書かれる．

キューイング　送信すべきデータをいったん保管しておき，受信側の処理の完了を待たずに次の処理を行なう処理方式．

コードコンベンション　タブの数とか改行の方法，大文字と小文字の使い分けなど，コードを見

やすくするために規約を設けることがある．この既約を言う．

JAVA　オブジェクト指向言語の一つ．

ant　JAVAの自動ビルド機能を提供する．apacheグループのオープンソースソフトウェア．

Eclipse　JAVAの統合開発環境の一つ．オープンソースソフトウエア．

Jama　National Institute of Standards and Technologyが提供する，行列演算用のJavaライブラリ

Windows XP　OSの一種，MicroSoft社の商標．

Debian Linux　OSの一種

utf-8　文字コードの一種．世界中の多くの文字種を計算機上で統一的に表現しようとするUnicodeという動きがあり，この枠組の中で日本語を16bit で表現するような表現の一種．日本語には従来からShift_JIS，EUC-JP，ISO-2022-JP（JISコード）など多くの文字コードがあるが，そういったものの一種だと考えて良い．utf-8はJavaで標準でサポートされている．

非Latin-1文字　アルファベットにウムラウトやアクサンターギューがついた文字などを集めたISO-8859-1に規定される文字セットをLatin-1文字と言う．欧米系の文字は，アスキーコードとLatin-1文字を合わせてすべて1バイトで記述できる．日本語を含む，この枠に収まらない文字種を総称して，非Latin-1と呼ぶ．

cvs　concurent versions systemの略，サーバークライアント形式で動作するバージョン管理システムのオープンソースソフトウエア．

データ解析やデータフローの表示方法については，以下の用語が使われている．

カルマンフィルター　フィルターの一種. 1960年，カルマン氏により提唱された．

フィルター　時系列観測データからの状態パラメータ推定において，現在までの観測データをもとに状態パラメータを推定する方法．現在と離れた過去のある時点以前の観測データから現在の状態を推定する方法をprediction，現在を含む過去から未来までの観測データを元に現在の状態推定する方法をsmoothingと呼び，状態推定はprediction，filtering，smoothinの三つに分類される．

DAG　Directed Acyclic graph，有向非巡回グラフの略．複数のものの関係をノードと線で表現する方法を，グラフと呼ぶ．このグラフの中で，線が向きを持っていて，かつこの向きにしたがって線をたどった時に巡回するものがないようなグラフをDAGと言う．

A.2　オブジェクト指向プログラミング

本稿では，「オブジェクト指向」技術を用いたコード開発を紹介している．オブジェクト指向の概念は，以下に示される．

オブジェクト　現実のものを計算機上で表現するもので，オブジェクト指向プログラミングではプログラムの基本要素となる．計算機の上でのオブジェクトの役割は，「状態」と「振舞い」をもつものと規定される．状態は，たとえば装置がONになっているとかOFFになっているとか，より複雑な装置では多数のモードを持っているうちのどのモードになっているかというようなことを表す．振舞いは，パルスが一つはいってくると「1」を，二つはいってくると「2」を返すというような，動作を表す概念である．

メソッド（操作）　オブジェクトの振舞いを表すもの．プログラム上で具体的なものの動作を表す場合，必要に応じて外から与えられた情報に基づいて自身の状態を変更したり，必要な計算を行なって値を返す必要がある．こういった動作は，プログラム上は関数を作成して行なう．

属性　オブジェクトの状態を表すもの．プログラム上は，変数に適切な値を保持することで，属性を表現する．例えば装置などの「もの」の大きさや重さ・位置，物性値などを表す属性がある．

クラス　実世界では，同じ種類の多くのオブジェクトが存在する．このオブジェクトの設計図，あるいは雛型をクラスと呼ぶ．クラスから見ると，個々のオブジェクトは「インスタンス」（実例）と呼び，雛型であるクラスからオブジェクトを作ることをインスタンス化と呼ぶ．例えばC言語やFortran 90では構造体と呼ばれる複数の変数を一括りにまとめて一つの変数のように扱う方法が使えるが，構造体の型定義がクラスに対応する．一方，プログラムの中には実際に同じ構造体に対応する複数の変数が存在するが，これらの各変数がオブジェクトに対応する．

抽象クラス　それ自身はインスタンスを作れず，このクラスから継承されたクラスだけがインスタンスを生成することが出来るような，特別のクラスを導入する方が，プログラムがきれいにまとまることがある．このような特別なクラス

を呼ぶ．

:: JasmineEvent::Photon.setCoord（）などと書いた場合，Photonクラスのインスタンスがメソッドを実行するが，実際にはPhotonクラスの親クラスであるJasmineEventクラスでそのメソッドを定義していることを表す．

また，オブジェクト指向言語とは言語仕様として以下の機能を持った言語を言い，これらを使って現実の系を計算機上に再現して行く．

抽象化　対象を計算機上のオブジェクトとして表現するのに必要なデータ（属性）を一括りにすること．例えば星を表現するのに必要な光度，色指数，半径，質量といったデータを一括りに扱って，「星」という概念を構築することができる．

カプセル化　属性とメソッドの組を作ることによって，あるデータを扱うための手続きがプログラム全体に散在することを防ぐ．また，オブジェクトの状態を変更するための内部的な手続きを隠して，外部からは決められた手段によってのみ変更可能であるようにすることで，情報を隠蔽する．例えば測定装置のスイッチを入れるた場合，内部的にはどの回路にどの順番で電流を流すといった複雑な手続きがあるが，外からは「スイッチを押す」ということしか見えないようにすることで，装置は使いやすくなる．装置を「使えない」状態から「使える」状態に変更するには，これで十分である．

継承　抽象化・カプセル化されたデータと操作の組の性質を，他の同様なデータと操作の組に受け継ぐ機構を持つこと．例えば，変光星は星の全ての性質を持っており，さらに変光周期などの変光星独自の性質を持つ．つまり，プログラミングの上では「変光星」は「星」を継承して作ることが出来る．「変光星」クラスは「光度を取得する」など「星」が通常持っているほとんどのメソッドを使うことが出来る．更に，「変光周期を取得する」など変光星独自のメソッドを追加することが出来る．通常の「星」と「変光星」で共通するメソッドは，これを呼び出す側は相手のオブジェクトが「星」のインスタンスか「変光星」のインスタンスかを意識する必要がない．

多相性　クラスが継承された場合に，同じ名前のメソッドで内部的に異なった処理を行なえるようにすること．例えば通常の星の光度は属性として持つ光度を返せば良いが，ある時刻に観測される変光星の光度は適当な関数によって計算されるはずである．このように，実際に内部で行なう処理は異なっているが，外から見れば，実際に処理の対象となっているオブジェクトは「変光星」なのかそうでない一般の「星」なのかをを知らなくても「光度を取得する」という同一の処理で行なうことができるような機構のことを，「多相性」と呼ぶ．Visitorパターンは，多相性を利用したパターンである．オブジェクト指向プログラムの開発においては，設計と実装は区別される．

設計　プログラム開発において，プログラムの動作や属性について，コードの記述のまえに，プログラミング言語の文法などと独立にプランを練る段階を経ることが効率的であるとする立場がある．この立場に立ち，プランを練る段階を設計と言う．この段階では，現実の系を計算機上で実現するにあたり，何をオブジェクトとして採用するか，オブジェクトの役割をどう割り振るか，オブジェクト同士の通信仕様をどう定めるかなどが検討される．

実装　プログラム開発において，実際にコードを記述する段階．

オブジェクト指向開発においては，オブジェクト同士あるいはクラス同士の関係を図に表すことによって，設計作業の効率化が行なわれている．1997年１月に，この図の統一した書式を定めたものがUML 1.0として発表された．UMLはUnified Modeling Languageの略である．最新バージョンは，2004年10月のUML2.0である．この定めにしたがって描かれた図をUML図と呼ぶ．UML図には，ユースケース図，クラス図，オブジェクト図，シーケンス図，コラボレーション図，アクティビティー図，状態図，コンポーネント図，配置図の９種類の図がある．

本稿で用いられているUML図は以下の４種類である．

クラス図　クラスどうしの継承や，クラスの属性が別のクラスである場合その関連性など，クラスのレベルで記述できるクラス同士の関連性を書く．「クラス」はプログラムを書けば決まるもので，実行時の状態により変化することはないことから，静的構造図とも呼ばれる．

オブジェクト図　実際の動作状況におけるオブジェクト同士の関連を記述する図．オブジェクトが実際にプログラム上に存在するか，どういった関連を持っているかは，プログラムの実行状況により変化するため，動的な図である．

シーケンス図　プログラムの実行順序に従って，

オブジェクト同士の通信の様子を示す図.

コラボレーション図（協調図） オブジェクト同士の通信の様子に注目した図．実行順序を示すことは出来ないが，シーケンス図に比べて多くのオブジェクトの間の関連を示すことが出来る．

また，オブジェクト指向の枠組の中で得られた経験は，「デザインパターン」と呼ばれる形でまとめられている．これに関連して，本稿に関連する以下の事項がある．

デザインパターン オブジェクト指向プログラミングにおいて，システムプログラムなどを分析して実際のプログラミングに有用なオブジェクトやクラスの使い方を選択したものを言う．Pattern Language of Program Design誌で現在でも多数のデザインパターンが提唱されているが，Gamma 3）らがまとめた23のパターンが有名. 著者らが自らを 4 人組と称したことから，GoF（Gang of four，4人組） パターンなどと呼ばれる．JASMINE Simulatorでもさまざまなパターンが用いられている．本稿では，以下のパターン名が現れている．AbstractFactory，Builder，Composite，Observer，Singleton，Strategy，Visitor．また，CompositeおよびLeafはCompositeパターンに現れる，ObserverとObservableはObserver パターンに現れる，VisitorとAcceptorはVisitorパターンに現れる，それぞれ特定の役割を持ったクラスを表すのに通常用いられる用語である．

A.3 クラス名，メソッド名

本稿で紹介している（Nano-）JASMINE Simulatorに現れる固有の概念を表す用語として，イベント，イベント処理装置がある．また，以下は本稿のシミュレーターで実装されているクラス名の一部である．AbstractSatellite（setObdh，getPosition，getVelocity，getAttitude はこのクラスのメソッド名），AbsUpdaterFactory（getNewStatusはこのクラスのメソッド名），BRSCoord，Comm，Controller，Disturbance，EventSource，EventAcceptor，Filter，GetStatus，Gyro，IRSCoord，JasmineEvent，JsEventHandler（processメソッドはこのクラスが持つvisitメソッド），JRSCoord，MagSensor，Obdh，ObservationMode，Optics，Orbit，OrbitUpdaterFactory，OrbitValueSette，Photon（diffractメソッドおよびreflectメソッドを持つ），Power，PowerValueSetter，Satellite，SatelliteUpdaterFactory，SetStatus，SingleStar，SRSCoord，Status，StatusHistory（linearlizeはこのクラスのメソッド名），StatusUpdater（executeはこのクラスのメソッド名），STT，SunSensor，SunShine，Temperature．Matrix は，Jama ライブラリに含まれるクラス名である．

A.4 人工衛星

人工衛星に関しては，以下の略語が用いられている．

ADCS Attitude Determination and Control System

GPS Global Positioning System

OBC オンボードコンピューター

RW Reaction Wheel．コマのような円盤の回転速度を変化させることによりトルクを発生する姿勢制御装置

STT Star Trackerの略．恒星を観測し，自身が保持している星表から衛星の姿勢を知る．

A.5 その他

その他，以下の用語が用いられている．

FITS 天文学で用いられる画像ファイルフォーマット

TDIモード CCDデバイスにおいては，読み出しのための転送は最終列のpixel 毎の読みだしと，一列を隣の列に転送する転送の二種類がある．このうち，列の転送のタイミングを自在に制御することにより，視野内を移動する物体の運動と同期させ，長時間露出を可能にする動作モード.

参考文献

1）山田良透, 上田誠治, 桑原立, 矢野太平, 郷田直輝 JASMINEシミュレーター―フレームワークの構築―国立天文台報 7， 3， 4， 41 （2004）

2）山田良透（依頼記事）赤外線位置天文観測衛星（JASMINE）計画に使われるオブジェクト指向技術情報処理45-12，1234 （2004）

3）Erich Gamma and Richard Helm and Ralph Johnson and John Vlissides Design Patterns Elements of Reusable Object-Oriented Software

Addison Wesey Longman, Inc., 1995
4）Erich Gamma and Richard Helm and Ralph Johnson and John Vlissides　オブジェクト指向における再利用のためのデザインパターン（改訂版）ソフトバンクパブリッシング，1999，本位田真一，吉田和樹監訳　ISBN4-7973-1112-6