国立天文台報　第7巻, 41－55（2004）

JASMINEシミュレーター
－フレームワークの構築－

山田良透*1，上田誠治*2，桑原　立*3，矢野太平，郷田直輝

（2004年3月31日受理）

JASMINE Simulator －Construction of Framework－

Yoshiyuki YAMADA*1, Seiji UEDA*, Takashi KUWABARA*3,
Taihei YANO, and Naoteru GOUDA

Abstract

JASMINE is an abbreviation of Japan Astrometry Satellite Mis-sion for INfrared Exploration currently planned at National As-toronomical Observatory of Japan. JASMINE stands at a stage where its basic design will be determined in a few years. Then it is very important for JASMINE to simulate the data stream generated by the astrometric fields in order to support investi-gations of accuracy, sampling strategy, data compression, data analysis, scientific performances, etc. It is found that the new soft-ware technologies of Object Oriented methodologies with Unified Modeling Language are ideal for the simulation system of JASMINE (JASMINE Simulator). In this paper, we briefly introduce some concepts of such technologies and explain the framework of the JASMINE Simulator which is constructed by new technolo-gies. We believe that these technologies are useful also for other future big projects of astronomical research.

1. はじめに

国立天文台を中心として，人工衛星を用いたプロジェクトである赤外線位置天文観測衛星（JASMINE）[1, 2] が計画されている．人工衛星を用いた場合，地上のプロジェクトとは異なるいくつかの特徴がある．

・装置の運用期間は搭載機器の機械的寿命や放射線による劣化などのため，地上観測機器に比べて極端に短い．この期間内に，効率的な観測を行う必要がある．
・一度装置を宇宙に打ち上げてしまうと，予めプログラムで想定されている機器の調整以外は不可能である．何か予想外の不具合が発生しても，現場で修正することは不可能である．
・天文学者以外に，ロケット・衛星などの専門家とのディスカッションが必要となる．
・一部の機器は地上環境では完全な予備実験が困難である．

このため，天文学・観測装置・ロケット・衛星の要求をクリアーにして，あらかじめ予想できることは万全の準備を行うとともに，予想外のことが起こった場合も対処しやすいようにしておく必要がある．このため，計算機を用いて，衛星の軌道から観測，データ転送にいたるまでのシミュレーションが有効となる．JASMINEと同様の位置天文観測衛星プロジェクトに，ESAのGAIAやNASAのSIMがあるが，GAIAにはSimulator Working Group（SWG）があり，またSIMでもこういったシミュレーションが行われており，その重要性が認識されている[3]．

ところで，シミュレーションといえば，従来でも装置開発において，また理論研究においても盛んに行われてきた手法である．従来のシミュレーションや数値計算では，計算性能がだけが重視されてきた．一方，ここ10年から20年のソフトウエ

*1 京都大学
*2 総合研究大学院大学
*3 新潟大学

ア工学では，ソフトウエアをどうしたら「より保守性が高く」「より再利用性が高く」できるかということが研究されてきた．

例えば人工衛星の場合，上で述べたように，シミュレーションシステムに関係する知識は天文学以外に装置，ロケットのような輸送系，姿勢制御系や通信，電力などの衛星システムまで多岐にわたる．一人の研究者がこれらのすべてにわたって精通することは不可能である．シミュレーションの一部にはこういった分野の専門知識を組み込む必要が生じる．そこで，個別の研究者は部分的にコード開発を行い，これを「再利用」して全体を構築するというスタイル，共同開発が必要となる．

そこで，ソフトウエアの共同開発に関しては，ここ 10 年フリーソフトウエアや企業でのソフトウエア開発などの現場において，さまざまな蓄積ができた．こういったものを積極的に導入して，効率よい共同開発を行ってゆく必要がある．天文学において，ソフトウエア工学で進歩した技術の導入は不十分であり，今後積極的に行ってゆくべきだという認識は，GAIA Simulator グループの認識と一致している[4]．JASMINE において，このようなソフトウエア工学における新しい技術を導入してゆく開発スタイルが必要となってくる事情は，プロジェクト自身の規模が時間的にも人数的にも大きくなることによる．今後の天文学におけるビッグプロジェクトでは，同様な技術導入が必要になると予想される．

そこで，本論文では，プログラミング言語や開発手法に関する最近の情報科学の進歩に関して説明するとともに，こういった技術を導入したJASMINE Simulator の開発におけるフレームワークの構築方法について述べる．

本論文は，以下のように構成される．まず第 2 章でJASMINE Simulator において導入した情報技術の簡単な解説を行う．この章は，以後の章で用いられる用語や概念の定義を与えることを目的とする．以後の章は，実際のJASMINE Simulator のフレームワーク（骨組み）部分の開発に関して，分析（第 3 章），設計（第 4 章），実装（第 5 章）の順番に解説し，最後に第 6 章でまとめを記述する．

2. 情報科学の手法の導入

JASMINE Simulator はプログラムの規模も大きくなり，またプログラムの一部は実際のデータ解析に利用することも視野に入れており，時間的にも長期にわたるソフトウエアプロジェクトとなる．このため，プログラミング言語のみならず，開発手法やソフトウエアの保守・管理の方法についても，以下に示すようなここ 10 年で発展してきた最新のソフトウエア技術を導入する必要がある．

JASMINE Simulator は，銀河モデル，観測装置，衛星システム，解析手法から得られるサイエンスに至るまでの，観測にかかわる全体をまとめてシミュレーションすることによってその相互の関連を明らかにすることを目的としている．従ってSimulator 全体は巨大なものとなるが，その中身は小さな部品から構成される．それぞれは，従来の科学シミュレーションや装置シミュレーションなどで作られているような部品である．従来でも，こういった部品を少数組み合わせて行った計算もあった．しかしながら，こういった部品を多数組み合わせる際には，ある部品と他の部品の独立性を高めることが重要となる．例えば，観測装置にかかわるプログラムを修正したとしても，その効果が銀河モデルから解析手法のモデルまで，プログラム全体に波及するのでは，このような大きなプログラムの保守や開発が困難となる．

このような細かい部品を寄せ集めて，しかも部品同士の独立性を高めることによって，プログラムの開発や保守を容易にしようという考え方が，オブジェクト指向と呼ばれる考え方である．オブジェクト指向では，プログラムは「オブジェクト」と呼ばれる単位で構築される．オブジェクトは，例えば光学系・検出器など，現実の「もの」を計算機の上に適切にモデル化したものである．JASMINE の場合，銀河モデル・光学系・検出器・データ処理系などがオブジェクトになる．システム全体としてはこれらは複雑に絡み合うが，全体をまとめてプログラムすることはいたずらに問題を複雑にする．光学系のプログラムを考える時は光学系のことだけを，検出器のプログラムを考える時には検出器のことだけを考えられれば，問題を多少単純化することができる．これが，問題を「オブジェクト」に分割することの意義となる．

また，こういった装置の動作自体は複雑であるが，外から見れば

銀河モデル　光を発するもの（天体からの光）
光学系　光が入ってきて光が出てくるもの
検出器　光が入ってきて電気信号が出てくるもの
データ処理系　検出器の電気信号を数値に変換するもの

というような役割を持つものとして単純化でき

る．こういった装置を組み合わせてサイエンスを行なうためのシステムを考える時，ある装置の他の装置に対する影響は多くの場合単純化できる．考慮すべき影響は，次のような点だけである．つまり，個別の装置の誤差がシステム全体の性能にどのように影響するのかということ，衛星システムの場合は電力，実装の位置などによる熱や振動の影響と言った少数の影響を考慮すれば良いはずである．従って，複雑なプログラムの中身のうち外に対してオープンでなければならない要素は限定されるであろう．オブジェクト指向では，外部プログラムから変更可能なもの，外部プログラムから参照できるけれども変更できないもの，外プログラムから参照できないものを区別することによって，オブジェクト（部品）同士の独立性を高めることが可能である．

もちろん従来型のプログラム言語（FORTRANなど）でも，気を付けてプログラムすればこう言ったことが全て可能である．例えば，上述の「変更できない」ものは，「変更しない」ように気を付けてプログラミングすれば良い．即ち，言語仕様の上でのさまざまな機能は，これ無しでは原理的にプログラミング不可能であることを意味するものではない．あくまでもプログラミングをよりやりやすくするためのものである．しかし，言語仕様のなかで巨大なプログラムを開発する場合に必要な機能が与えられていることが，プログラミングの効率を高める．オブジェクト指向言語には長年の歴史があり，その中でさまざまな蓄積（第2.2 章参照）もある．こういった蓄積が，従来型の言語ではなくオブジェクト指向の枠組で語られてきていることも，オブジェクト指向を利用するメリットである．Simulator 構築の説明に必要なオブジェクト指向の概念は，第 2.1 章で説明する．現在は，C++やruby など多数のオブジェクト指向言語がある．この中で，我々は第 5 章で示す理由よりJAVA 言語を選択する．

オブジェクト指向プログラミングでは，部品同士の独立性を高めることが重要であると述べた．しかし，部品同士がまったく関連しないわけではなく，プログラム上必要最低限の関連は持っている．オブジェクト指向プログラムを設計する際には，こういったこまかな部品同士の関連や状態の変化などを把握しておくことが重要である．このために開発された言語が UML（Unified Modeling Languate：統一モデリング言語）である．UMLは，部品同士の関連のしかたや動作を図で示すものである．オブジェクト指向では，プログラムを細かな部品に分けることが多く，部品の種類や数が多くなる．これらの関連のしかたや動作をプログラムのソースコードを読んで理解することは困難な作業となる．そこで，図を用いることで，プログラムの全体像を把握することが容易となる．また，図に表示するクラスを適切に限定することで様々な詳細度でプログラムを把握することを可能にし，プログラムの部品同士の関連性が適切に考慮されているかなど，プログラムの保守性や再利用性についての検討も可能になる．JASMINE Simulator も大規模なプログラムとなり，この設計を説明するにはUML が必要なので，これを第2.2 章で解説する．また，オブジェクト指向は「再利用性の向上」や「保守性の向上」などをその意義としてうたっているが，オブジェクト指向言語を用いたからといってそういった目標が達成されるわけではない．オブジェクト指向を使って再利用性や保守性を高めるためのノーハウの蓄積がパターンと呼ばれる．これを，あわせて第 2.2 章で解説する．

JASMINE Simulator は，共同開発である．共同開発においては，開発途上の変化し続けるコードを開発者の間で共有する技術が必要となる．JASMINE では，このコード共有技術として，オープンソースの世界で広く用いられている技術であるCVS を用いているが，これを第2.3 章で紹介する．

開発者においてもユーザーにおいてもプログラムに関する文書化は必須である．計算コードが開発途上で変化しつつある場合，文書とコードが整合しないことがしばしば起こる．そこで，JASMINE において用いた文書化技術を第 2.4 章で解説する．また，重い処理を分割して複数の計算機で実行するための分散処理技術として CORBA を導入したので，これを第 2.5 章で紹介する．

2.1　オブジェクト指向技術

フリーソフトウエアや企業のソフトウエア開発で進められてきた蓄積の一つは，オブジェクト指向技術の発展である．オブジェクト指向とは，プログラムを「オブジェクト」と呼ばれるもので構成する考え方である．そもそもオブジェクト指向は実際のもの（装置や組織，システムなど）をシミュレーションするプログラムを開発する手段として発生した．シミュレーションすべき対象の構成要素として，基本的な単位となるものを，計算機の上で「オブジェクト」と呼ばれるもので表現し，「オブジェクト」同士が情報をやりとりすることでシミュレーションを構成しようというの

が，基本的な考え方である．

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．

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．

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．

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．

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**抽象クラス**と呼ぶ．

オブジェクト指向の重要な性質は，以下の 4 点にまとめられる．

抽象化　対象を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．

巨大なシミュレーションプログラムを開発した経験がある研究者なら，部品の独立性を高めたり，再利用性を高める試みをした経験があるはずである．FORTRAN プログラムなら，メインプログラムだけで組まれたプログラムを適切にサブルーチ

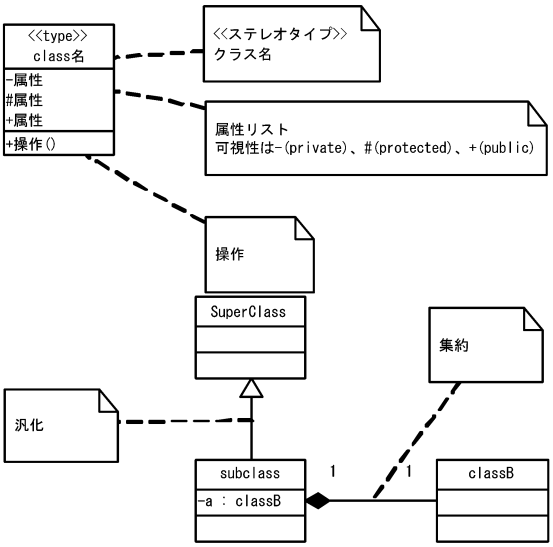

図 1. クラス図の例：クラスは三つの箱で表される．上段がクラスの名前と，記述しておくと便利なクラスの特性（ステレオタイプ），中断が属性，下段がメソッドを表す．クラス同士の関係は，矢印付きの線で表す．可視性は，自身以外のクラスから参照・変更できないものをprivate ，継承された子クラスと自身以外のクラスから参照・変更できないものをprotected ，自身以外の任意のクラスから参照・変更可能なものをpublic と呼ぶ．参照できるが変更できないようにするためには，属性をprivate にして参照するメソッドをpublic で定義することで実現する．継承されるクラスは，子クラスから親クラスへ白抜き三角の矢印を付けて表す．また，別のクラスをクラスの属性として持つ場合，線の根本に菱形を付けて表す．菱形が図のように黒塗の場合は線の根もと（菱形のついている側）オブジェクトが消滅すると線の先のオブジェクトも同時に消滅することを表し，白抜きの場合は根もとのオブジェクトが消滅しても先のオブジェクトは消滅しないことを表す．線の上に付けられた数字は多重度を表す．すなわち，図では根もとのオブジェクトが一つに対して先のオブジェクトが一つ存在する．数が定まっている場合は数字を，複数で数が不定である場合は数のかわりに*印を記述する．図では属性欄にclassB のインスタンス名が書かれているが，属性欄に書かずに線の上にインスタンス名を書くことも出来る．

ンに分割するというような作業である．情報科学の分野では，このレベルの再利用性を高める試みを，「構造化プログラミング」という名前で呼ぶ．オブジェクト指向は，構造化プログラミングの後に誕生した技術であり，構造化プログラミングで不足している部分を補っている．例えば，FORTRAN 90 やC では属性を「構造体」という形で一括りにできるが属性とメソッドの組を作ることができない．すなわち，FORTRAN 90 やC でプログラミングを行う場合，上であげたオブジェクト指向の重要な性質である 4 点，「抽象化」「カプセル化」「継承」「多相性」のうち，「抽象化」だけが可能である．一方他の 3 点の概念は，言語仕様に含まれていない．カプセル化は計算コードとデータを局在化する効果があり，継承や多相性は「構造体」の定義をより意味論的な側面から構築しやすくする効果があるという意味で，オブジェクト指向言語の導入はFORTRAN やC に比べて優れていると言える．一方で，オブジェクト指向は計算機に対して非常に重い処理を要求する．コードの実行効率という意味においてオブジェクト指向言語が従来言語に劣るとしても，コードの開発を含めた全体の時間的な短縮となることは間違えない．さらに，実行速度の問題はハードウエアの進展により時間とともに解消してゆくと予想される．

2.2 UML とパターン

このようなオブジェクト指向の概念，「クラス」や「オブジェクト」を用いたプログラムの設計において，共通の言語（図式）が作られている．これは，UML（統一モデリング言語，Unified Modeling Language）と呼ばれている．継承や委譲（あるクラスのインスタンスが別のクラスの属性となっている）のような，クラス同士の関係を図式的に表すものである．プログラムの全体像を視覚的にとらえることができるため，細部にとらわれずに短時間で全体を把握することができ，複数の開発者がプログラムの設計について議論するためにUML が活用される．JASMINE Simulator の規模のプログラムになると，UML 図無しに開発するのは難しい．UML の解説書は多いが，例えば具志堅他[5] などがある．UML には全部で 9 種類の図があるが，良く使われる図として，ここでは二種類の図を紹介する．クラス図（図 1）は，

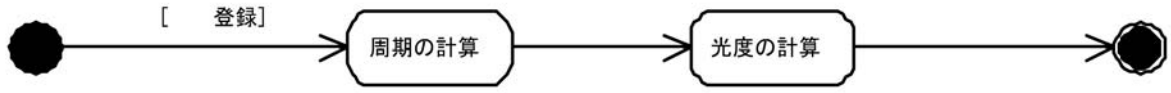

図 2. 状態図の例：オブジェクトの状態の変化を表す．始状態を黒丸で，終状態は黒丸の周りを白丸で囲ったもので表す．途中にいくつかの状態を持つ場合，その状態と状態間を遷移する理由（どのメソッドが呼ばれた場合等）を矢印で示す．図では始状態から「登録」というメソッドが呼ばれると「周期の計算」という状態に移ることを表す．

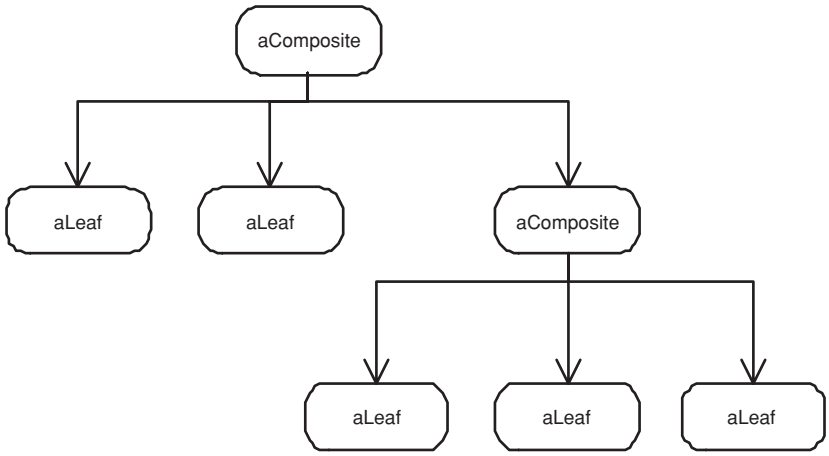

図 3. オブジェクトの階層構造．aComposite および aLeaf と書かれたものは，図 4 に書かれる Composite クラスおよび Leaf クラスのインスタンスである．aComposite のオブジェクトは aLeaf あるいは aComposite オブジェクトを任意の個数持つことが出来る．aLeaf オブジェクトは以下にオブジェクトを持たない．これにより，木構造の階層が生成できる．

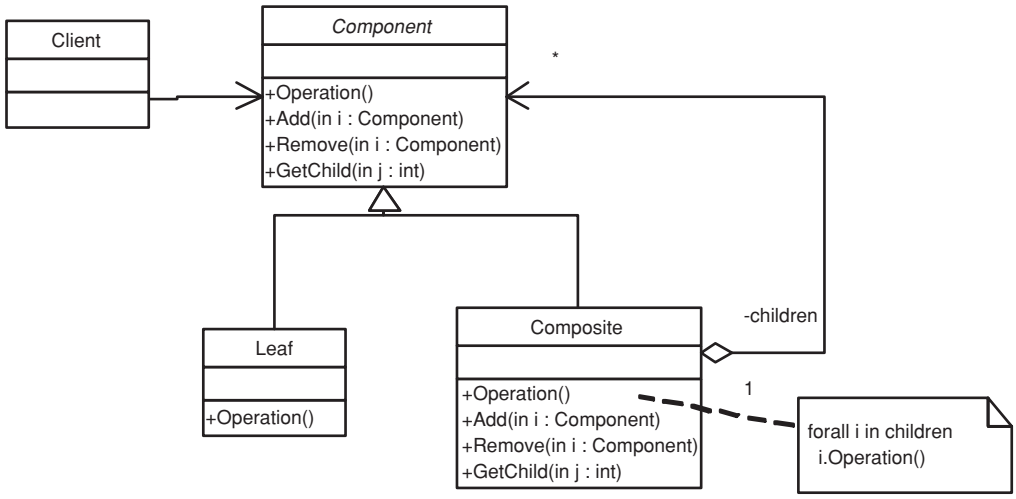

図 4. 図3 で示すオブジェクトの階層構造を実現するためには，この図に示すようなクラス構造を用意すれば良い．Composite クラスは Component クラスを複数持つことが出来，children という名前でアクセスできる．例えば衛星全体の装置を全て Component という親クラスからの継承で用意し，電力や重量の計算を行なうメソッドを親クラスで定義しておく．Leaf クラスは自信の消費電力や重量を返す．Composite クラスは配下にある Component クラスでの電力や重量の計算を行ない，これを足し算したものを返すように設計しておくと，クライアントからは階層の任意のレベルでこのメソッドを呼ぶことにより，部分あるいは全体の消費電力や重量を計算することが出来る．

クラスの属性とメソッドの名前を示すとともに，クラス同士の継承やあるクラスが別のクラスを属性として持つ等の関係を示す図である．状態図（図２）は，ある一つのクラスに注目して，クラスが持ちうる状態を列挙し，ある状態から別の状態へ移行する条件(呼び出される関数の名前)を示す図である．フローチャートはアクティビティー図という名前で UML の 9 種類の図のうちの一つとして存在している．この他，クラスではなくオブジェクトがどのように構築されているかを見るためのオブジェクト図，複数のクラスの状態変化の時間的な変化に着目した状態遷移図などが，よく使われる UML 図である．こういった図を編集する機能を有するソフトウエアを，CASE（Computer Aided System Engineering）ツールと呼ぶ．CASE ツールでは，さらに UML 図からプログラムのテンプレート(クラス宣言文のほか，属性名と関数の宣言文)を生成したり，逆に既存のプログラムから UML 図を生成したりする機能を持つ．

言語がサポートしている抽象化やカプセル化の機能だけでは，再利用性や独立性が高いプログラムは書けない．あるクラスの中の少しの変更が別のクラスにも影響し，それが波及して結局コード全体に影響を及ぼしてしまう場合がある．こういうコードを「クラスの結合性が高い」コードと呼び，逆に「クラスの結合性を下げる」ことが，保守性と再利用性を向上する．プログラム上のある種の目的を与えた時に，クラスやオブジェクトをどのように組み合わせれば便利なのかをまとめたものに，デザインパターンがある．Gamma et al. 1995 [6] の中には，目的毎に整理された 23 のパターン（クラスやオブジェクトの組合せ方）が掲載されており，同じ目的を実現するための他の解決方法と比較して，そのメリット・デメリットや実際にどういったプログラムのどういう部分で利用されているかという経験がきれいに整理されている．パターンは，オブジェクト指向がメリットであると唱っているオブジェクトの再利用性や独立性を最大限に引き出すための，いわば経験を分析し，まとめたものとして，有用である．

例えば，図３のようなオブジェクトの階層的な構造が必要になったとしよう．衛星では，ある対象が幾つかの部品の組合せで出来ている．衛星自身は「光学系」や「検出器」という部品の組合せであるが，さらに「光学系」は複数の鏡・架台・支柱などを構成要素として持つ．「検出器」は複数のCCD 検出器やそれを処理するためのA/D 変換器などの回路から構成される．そうすると，複数の部品の組合せであることを表す「Composite」なクラス（集約クラス）と，これ以上分割できない「Leaf」なクラスを用意することで，この構造が実現できる．具体的には，図4 のようなクラスの構造を用意すれば良い．これは，Composite パターンと呼ばれるパターンの適用例である．composite パターンは，children を複数持つだけでなく，図のようなツリー構造を作ったり変更したりするために Add や Removeというメソッドを持つ．この図の中の Operation というメソッドは，Composite クラスではツリーを走査して自身の子である Component クラスの Operation メソッドを順次呼び出して適切な処理をするように作られるものである．

例えば，電力や重量など全体の総和に相当する量を計算するような場合に用いることもできるし，ツリーを走査して適切な Component だけを取り出すなど，一般にツリーを走査することが必要になる操作の実装に使われる．例えば電力や重量の計算などは，各部品の合計として計算される．Composite クラスでは，配下にある Leaf クラスの電力や重量を計算した上で，その総合計を返すようにしておけば良い．これにより，例えば設計により鏡が一つ増えたとか，検出器が二つ減ったとかいうことは，階層クラスからオブジェクト構造が適切に生成されていれば，重量や電力の計算の部分に何ら変更を加える必要はない．

FORTRAN などの従来型言語では，このような集約をあらわすために配列を用いる．しかし，上で示すような集約オブジェクトを作ると，配列としての機能，即ち何番目という番号を指定してデータを受け渡しする以外に，順番に走査したり，キーで検索したり，新たに要素を追加したり(末尾だけでなく途中にでも)削除したりといった機能をもたせることが出来る．

JASMINE シミュレーターにおいてオブジェクト指向性が重要である理由は，複数の分野の専門家があつまって，大規模なコードを，共同で開発するところにある．適切なカプセル化によって，自分の専門外の部分には変更を加えずに，自分の専門の部分のコードの修正だけを行なって行くことが出来るような仕組みが望ましい．この意味で，オブジェクト指向の利点を最大限度に生かすため，随所にパターンを適用している．

2.3 CVS

コードの共有技術も進歩している．異なる研究

機関に所属する複数の研究者によって，コンピューターープログラムを共同開発する場合，ネットワークを利用して，複数の開発者がコードに機能の追加やバグの修正などの変更を加えて行き，それが整合的に行なわれる必要がある．企業では企業毎に独立のシステムを利用している場合も多いようだが，フリーソフトウエアの世界で標準的に利用されているのが，cvs と呼ばれるバージョン管理システムである．我々は，京都大学に設置した計算機の上に cvs と呼ばれるサーバープログラムを走らせて，コードを共有している．

2.4 文書化

また，我々は JAVA 言語での開発を行なっているが，文書化は JAVA 附属の javadoc と呼ばれるプログラムによって行なうようにしている．これは，プログラムの中にコメントとして説明を書き込むことによって，プログラムの説明文書をプログラムと一緒に保管する形式である．マニュアルなどの文章を別の文章として管理する手法はあり得るが，プログラムの更新と文章の更新を同期することが難しく，すぐにやくにたたない文章となってしまう．これを避けるために，プログラムの中に説明文書を入れることが行なわれている．C++ではdoc++と呼ばれるツールがある．

2.5 分散オブジェクト

分散処理には，国立天文台等の研究所に設置された並列計算機などの上で利用可能なものもあるので，多くの天文研究者には馴染み深いものである．しかしながら，これらは直接コードの中で並列計算を行なう部位を指定して，例えば一つのループ計算に含まれる数行のソースコードを，ループを分割していくつかの CPU で実行するというように用いられる．分割して個別の CPU に渡される処理量が比較的小さいので，「粒度の小さい分散処理」と呼ばれる．この場合，複数の CPU の間をつなぐ通信は高速なものを仮定しており，専用の並列計算機が無ければパフォーマンスは出せない場合が多い．

一方，オブジェクトという単位で並列化を行なうための，分散オブジェクト技術がある．並列計算機の上の並列化は，プログラム言語のコンパイラが計算をどの CPU に割り振るかを判断するけれども，分散オブジェクト技術では使用する言語に依存しない通信手段だけを定めている．並列計算機はもとより，ネットワーク上に分散する複数のコンピューターでの並列などにも対応する柔軟性がある．また，言語に依存しないので，別の言語で書かれたプログラム同士を結合するためにも使うことができる．

3 JASMINE Simulator の機能の分析

前述の通り，JASMINE と同様の位置天文観測では，GAIA が同じような計画をもって，Simulator Working Group(SWG)を立ち上げている．

GAIA では，まずシミュレーションすべき対象を「衛星システム」「光学系」「検出器」「観測対象である宇宙」などのコンポーネントに分類した．GAIA SWG が示しているクラス図を見ると，これらの対象は抽象クラスで定義され，検討の進展に従って各コンポーネントの具象クラスの詳細度を上げるような形の設計を行っているようである．これらの関連のしかたは実際の観測における情報の流れに沿って予め規定している．即ち，銀河モデルの中にある星が光を発し，この光が光学系を通過し，検出器で電気信号に変換され，テレメトリ・データに変換され，生データが生成される．光学系と検出器の姿勢を決めるために，衛星モデルが利用される．しかしながら，JASMINE で検討を進めたところ，このような枠組みでは JASMINE の検討を十分に支援できないことが分かっ

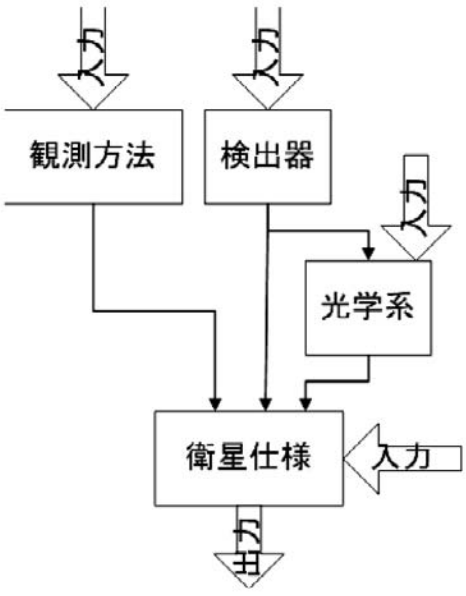

図 5. JASMINE Simulator の仕様決定の流れ．仕様決定では，検出器の pixel サイズをもとに最適な光学系の焦点距離を決める．実際の観測における情報の流れは光学系を通過した光が検出器に落ちるので，仕様決定における情報の流れと観測シミュレーションの情報の流れは反対向きになる．

てきた．

JASMINEの現段階で最も重要な作業は，JASMINE 自体の仕様を決めることである．図5 に，仕様決定の流れを示した[7]．一例をとると，現段階の検討では JASMINE の光学系の焦点距離は検出器の pixel サイズと光学系の口径と観測波長から決めている．口径と焦点距離は光学系の属性，pixel サイズは検出器の属性と考えるのが自然だが，光学系の属性の一つである焦点距離を決めるために，光学系自身のほかの属性（口径）との整合性だけでなく，検出器の属性（ pixel サイズ）が必要となる．これは，実際の観測の時の情報の流れの上流に位置するものの属性値を決めるために，下流に位置するものの属性値が必要となる場合があることを意味している．実際の観測における情報の流れに従ってプログラムが動作する GAIA Simulator のような設計では，このような場合には対応できない．

そこで，実際の情報の流れに沿っている場合にも，その逆の場合にも情報の受渡しを可能にするようにするため，入口と出口が定義されていて中身に関係ない情報受渡しの手段を提供することが必要となる．このため，検出器や光学系などのコンポーネントをまとめて扱えるような「モデル」という抽象概念を導入する．こういった枠組を作った後に，検出器や光学系，データ転送，衛星軌道や振動擾乱源などをモデル化して，コンポーネントとして作り上げれば良いことになる．この抽象化によって，もちろん情報の流れに沿った通常の評価を行なうことは可能だし，それ以外の問題にも対応する柔軟なシミュレーションの枠組を構築することが可能となる．

大規模シミュレーションなどを行った経験があるユーザーは，例えば AVS などのグラフィックソフトウエアと同様の設計といえば理解しやすいかも知れない．等高線を書く，ベクトル場を表示するなど，さまざまなコンポーネントが用意され，これらのコンポーネントの間に情報を流す仕組みが用意されている．従って，一見 AVS などのソフトウエアをベースにしてそのプラグインを組み立てればよいように思われる．実際こういった大規模な作図ソフトウエアは，ユーザーによってプラグインを作る手段が示されていて，ユーザーが機能を拡張することが出来る．しかしながら，通常のグラフィックソフトウエアで可能なことは，これらの作図用「コンポーネント」を拡張することである．データ自信は拡張できない．メッシュの上に与えられたスカラー場やベクトル場のデータ，時系列データ，スカラーパラメータなどをコンポーネント間で流すことは可能だが，予め用意されたグラフィック用のデータ形式とは異なる情報を流すことは出来ない．この制限は，JASMINE Simulator を AVS の拡張として構築することを難しくしている．

このため，我々はまずモデルの間に情報を流すというレベルの抽象化を行ったソフトウエアを構築することとする．

その上で，モデルを具体化したものとして JASMINE 用のコンポーネント，検出器や光学系・衛星システムなどを構築してゆく．利用方法は，AVS などの使用方法に慣れているユーザーならほとんど迷うことは無いだろう．

4. JASMINE Simulator の設計 －フレームワークの構築

4.1 プログラミングにおける設計の位置付け

前の章で，JASMINE Simulator が持つべき機能について分析した．この章では，JASMINE Simulator の設計について議論する．現代のソフトウエアの開発は「モデリング」,「設計」,「実装」の順で行なわれる．出来上がったものを見て，問題点を洗い出して更に「設計」,「実装」のプロセスが繰り返され，より洗練されたソフトウエアとなる．情報科学の方法論は，実際のもの作りの経験をプログラム開発にや焼き直して整理している．もの作りで言えば,「モデリング」はポンチ絵を書く段階,「設計」は設計図面を書く段階,「実装」は実際に材料を加工してものを作る段階にそれぞれ相当する．

モデリングのポイントは，機能を明確化することにある．前の章で行なったように，機能を分析して必要な部品を洗い出したところで，おおよその「モデリング」が完了していると考えられる．実装はプログラムを一行一行書いてゆく作業に相当し，従来的なプログラミングではモデリングが出来たら実装に取り掛かる．当然，実装では同じ機能を実現するのに複数の手段をとり得る．例えば，繰返しを表現するのにループを用いるか再帰を用いるかというような判断が随所に現れる．従来的なプログラミングの経験から考えると,「モデリング」と「実装」の間に入っている「設計」は新しい概念である．端的に言うならば，設計段階で考えるべきことは，ループを用いるか再帰を用いるかなどの実装の詳細に依存しないレベルでの，全体の組み立てかたを考える作業である．多

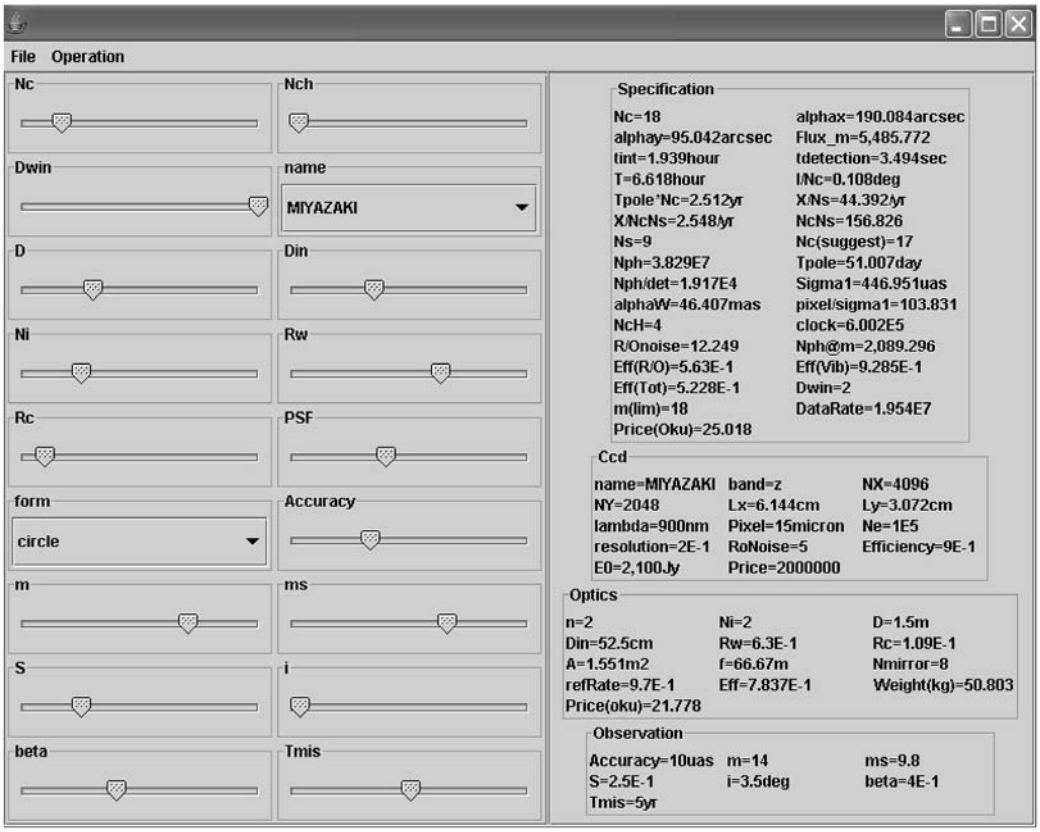

図 6. JASMINE Simulator の機能を検討するためのプロトタイプ．これは，「CCD 」「光学系」「観測手法」「仕様計算独自のモデル」の 4 つの「モデル」から構築されている．それぞれのモデルの持つ属性値が，右側のウインドー内に四角で囲って表示されている．それぞれの「モデル」の一部の属性は他の「モデル」より与えられる．また一部の属性は，右側のウインドーに示されるスライダー部品や選択部品により，ユーザーが設定できる．依存性グラフは，このバージョンでは表示されていない．

少不正確だが大胆な説明をするなら，「モデリング」は機能の分析・明確化の作業であり，「設計」は必要なクラスを選定してそれらのクラス同士の関係をつける作業であり，「実装」はクラスの中身を作りこんで行く作業となる．以下では，具体的に JASMINE Simulator の設計を見て行くことにする．

4.2 JASMINE Simulator の全体設計

JASMINE Simulator のフレームワークを構築するため，まず前章で示した仕様計算を行うための，具体的なプログラムを作成してみる．我々が行っている仕様検討の流れは，光学系のパラメータとして主鏡口径・PSF の特性によるいくつかの数値，検出器のパラメータとして pixel サイズや pixel 数・full well electron の数・感度など，観測手法のパラメータとして天球上で実際に観測を行うエリアやミッション時間など，サイエンスのパラメータとして目標精度など，約 20 ほどのパラメータを与え，最終的には必要な検出器の数を計算する．手で入力する 20 ほどのパラメータについては，ユーザーが柔軟に値を変更できるようにしたい．また，最終結果としての検出器の数だけでなく，計算の途中で光学系の焦点距離や検出器が見込む天球上の角度など，重要なパラメータで他のパラメータから決められる数値があるので，これらの数値をあわせて表示したい．

こういった要求を満たすように作ったテストプログラムの起動画面を，図 6 に示す．ユーザーの入力値変更を支援するために，左側にスライダー（スライドして数値を変更する）やコンボボックス（リストで示された数値や文字列などをマウスで選択する）と呼ばれる入力ウィジェット（入力を支援するための Graphic User Interface 部品）を配置し，結果は右側に表示される．この程度の計算であれば，入力値のどれか一つでも変更を受ければ全体を再計算しても，それほど重い計算とはならない．しかしながら，検討が複雑化して計算量が増えてくると，変更されたパラメータによって影響を受ける式だけを再計算するような機構が望ましい．さらに，新たな検討課題が生じた場合に，同じようなプログラムをいちいち作るのでは，

間違えが混入しやすい．できるだけ再利用可能な形でプログラムを構築してゆくことが望ましい．

再利用できるまとまりとして，具体的なものに対応させて検出器・光学系などで括っておくことが，計算内容をイメージしやすい．例えば光学系をとると，口径・焦点距離・扱う波長・回折限界などの属性がある．これらのパラメータのすべてが独立であるわけではない．仕様検討においては，口径は利用者が入力可能なパラメータであり，波長は観測要求から決まるパラメータである．このほか，PSF に由来するパラメータを 2 つ導入しているが，これらは本来光学系のデザインが決まれば計算可能なものである．しかしながら，この計算は複雑であるため，現在は手で入力するパラメータとして扱っている．これらに対して，回折限界は口径と波長の二つのパラメータから決められるものであり，焦点距離は，検出器の pixel サイズと観測要求（回折限界が 2pixel となる）で決めているため，pixel サイズを与えれば焦点距離を自動的に計算したい．

プログラム全体の整合性を考えた場合は，次のような要求がる．もし光学系の口径が変更された場合，光学系の焦点距離という属性を利用するほかの部品に対しては，焦点距離を自身で再計算した後に，焦点距離が変更されたことを通知する必要がある．

計算の中で光学系という部品の入力と出力に注目すると，シミュレーターの全体設計の中で，シミュレーションに必要な部品が満たすべき条件は以下のようになる．

- 入力可能な属性が複数個有る
- 入力は手で行われることもあるし，他の部品から行われることもある．
- 自身で整合性を確保できる，他の属性がある
- いずれの属性も，他から参照される場合がある．
- 入力可能な属性の一部が変更を受けた場合，自身の整合性を確保した後に，他の部品に属性の変更を通知する必要がある．

「機能の分析」の章でも見たように，検出器や光学系など JASMINE Simulator に必要な部品同士の属性値の依存性は検討項目ごとに異なるため，これに柔軟に対応するためにこれらの部品に共通の抽象物「モデル」を導入する．上で見た項目は，この「モデル」が持つべ共通の条件であり，「モデル」を具体化した光学系や検出器といったものは，入力可能なものは何か，属性として持つべきものは何か，属性値同士の関係はどのように規定されるかといった点を具体化することにより，「モデル」を拡張する．

ここで，光学系とか検出器とかといった内容を限定しない「モデル」のレベルの抽象クラスを作る意義は，入力や出力を管理し，依存性グラフ（点を線で結んだものをグラフと呼ぶ）を描くレベルにおいて必要かつ十分な機能を抽出することにある．実際の計算には具体的な検出器や光学系といった情報が必要だが，依存性グラフの中では「ここでこのタイミングで必要な計算をする」ということが分かっていれば十分であって，その計算の中身が何であるかは知らなくて良い．そこで，抽象レベルの「モデル」をあらわすクラスでは呼び出される関数(メソッド)の名前や型だけが定義されていて，その実態は具体的なクラスの中で定義される．

今の例は第ゼロ次的な単純な「仕様計算」に限定されているが，実際には様々な検討を行ってゆかなければならない．そこで，Simulator 全体としては，あらかじめ用意された部品の属性値の依存性を示す「グラフ」を編集する手段が必要となる．ここで考えるべきグラフは依存性を示すのであるから A から B へというような方向性を持っており，さらにループを持たないものに限定してよい（情報科学の言葉では，有向非巡回グラフ-Directed Acyclic Graph-と呼び，DAG と略す）．編集する機能に加え，構築したグラフを保存，あるいは読み込む手段も必要となる．

更に，図6 の例では入力用の部品としてのスライダーやコンボボックス，出力部品として文字を表示する機能を提供している．これらは，依存性グラフの始点および終端に位置しなければならないという点を除けば，検出器や光学系と同じように依存性グラフの中に現れて，「モデル」の一種と考えてることができる．出力部品はこの例では文字の表示だけだが，検討のある段階において複数のパラメータの関係を示す折れ線グラフが必要になったり，CCD 画像等に対応する画像が必要になる可能性もあり，こういったものへの拡張可能性を残さなければならない．

さて，上で見た機能を実際にクラス図として構築してみる．こういったある程度汎用性を意識したなライブラリを構築する場合，他のライブラリと組み合わせたときに名前が競合しないように，クラスの名称にはある種の規則を作るのが一般的である（name space convention と呼ぶ）．我々は，クラスの名称の前二文字が必ず Js（Jasmine Simulator の意味)であることとしている．クラス

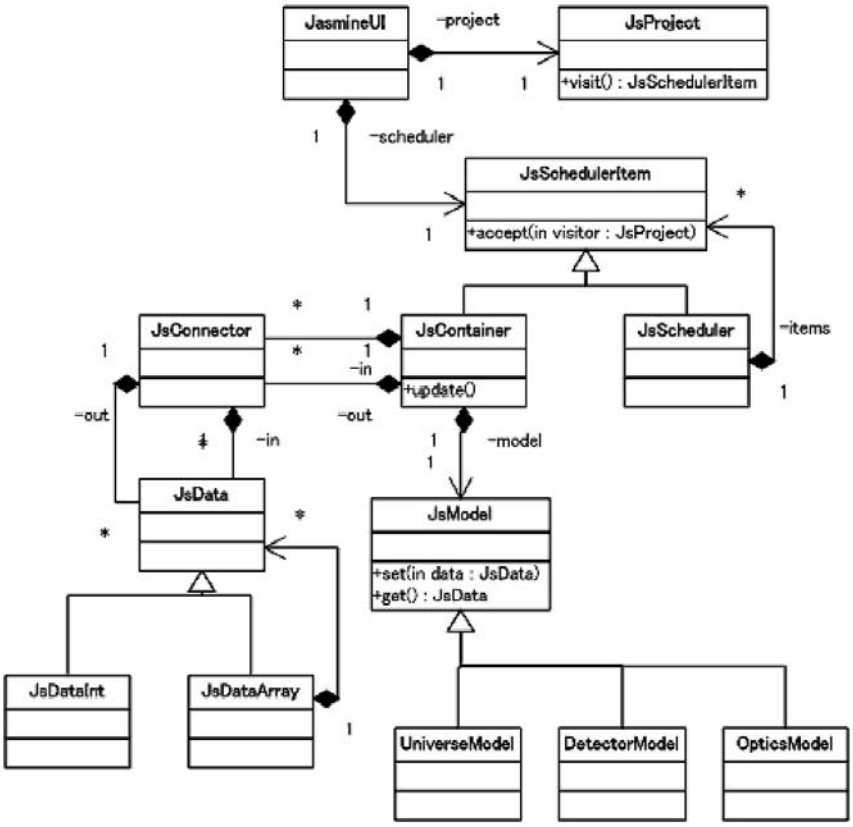

図 7. JASMINE Simulator のクラス図の一部．クラス図の書式は図1 とその caption を参照．ユーザーインターフェースが JasmineUI というクラスに治められている．JASMINE Simulator のユーザーインターフェースはユーザーインターフェースの形式によって JasmineCUI クラス及び JasmineGUI クラスとして実装したが，図では簡単のためにJasmineUI クラスとして示した．実際に計算を行なうための部品は JsModel クラスであるが，JsModel から入出力機能だけを適切に抜き出すため，JsModel は必ず JsContainer クラスの中に保持されるよう設計される．JsContainer クラスは Composite パターンで実装され，JsContainer クラスは JsScheduler クラスで集約される．JsModel は抽象クラスであり，UniverseModel，DetectorModel，OpticsModel など，それぞれの機能に応じた具象クラスがサブクラスとして構築される．Container 同士の接続には JsConnector クラスが用いられる．一つのJsContainer からは複数 JsContainer に対する接続がある場合が想定されるので，多重度は JsContainer が1 に対して JsConnector が複数である．JsConnector は接続だけを表し，実際にその中で交換されるデータの形式については別のクラス，JsData が担当する．JsData もそのデータ型によって，複数の具象サブクラスを持つことになる．Composite パターンに対する操作は別のクラスである JsProject から行なわれる．

名は内容を表すものが良いので，例えばモデルをあらわすクラス名は JsModel とするというように，Js の後ろにクラスの内容をあらわす英語名を付ける．この name space convention は，ライブラリ部分に適用される．即ち，main ルーチンとして起動時に呼び出されるユーザーインターフェースクラス（JasmineUI）や，汎用性を意識していないJASMINE 固有のクラス（ Universe Model, OpticsModel など）の名称に関しては，このルールに従っていない場合がある．

4.3 箱の設計―JsContainer/ JsScheduler/ JsSchedulerItem

JsModel クラスは実際に計算するためのある程度高機能なクラスである．他のモデルとの間でデータを授受するための機構や依存性通知機構は，抽象クラス JsModel で持ってもよいし，そういった機構を持ったクラスがJsModel を保持するという構築の仕方もある．一般に，プログラミングの柔軟性を確保するためには，クラスを継承するよりクラスのメンバーとしてクラスを持たせるほうが優れている場合が多い．そこで，Jasmine Simulator でもデータ授受機構や依存性通知機構は抽象クラス JsModel の機能として持たせるのではなく，JsModel を保持する別のクラスが持つように設計する．このクラスは「モデル」を入れる「箱」としての意味を持つので，JsContainer クラスと名づける．

実際のシミュレーションでは，この「箱」が複数あって，それらが依存性を表す「線」でつなが

図 8. JsModel の状態図．それぞれの属性が持つ値の違いにまで状態の違いと考えれば，JsModel は無数の状態を持つクラスである．しかしながら，JsModel の次の側面に注目すると，JsModel は二つの状態を持つクラスであると考えられる．すなわち，JsModel は外部から自身の属性の一部の変更を許し，変更が加えられれば一旦属性値が不整合状態となるが，この整合性を自身で回復する機能をもつクラスである．

れている．この「箱」のつながりは，依存性グラフの示す順番に従って順次計算を行ってゆく必要があり，また依存性グラフの構築においては「箱」を追加したり削除したり，あるいは「箱」同士を「線」でつなぐなどの操作が必要となる．つまり，複数の「箱」には構造を構築，保持，変更するなどの高度な機能が必要となり，集約クラスを作る必要がある．このクラスは，計算全体をスケジューリングするクラスであり，JsScheduler と名づける．ここに composite pattern を適用すると，集約クラスと Leaf クラスである「箱」の共通の親クラスが必要となるので，これを JsSchedulerItem と名づける．

4.4　線の設計―JsConnector―

「箱」同士を結ぶ「線」は，JsConnector クラスであらわされる．「箱」には入力（in）と出力（out）があり，それぞれが複数の「箱」につながる可能性もあるので，JsContainer のin 属性とout 属性がそれぞれJsConnector であり，JsContainer 一つに対して複数（図では*で表示）の JsConnector がつながる．一方 JsConnector 側から見れば，有向線分は終端を in とout（図ではこれに対応するin とout の名前は示していない）の二つ持つので，JsConnector の in 属性と out 属性にそれぞれ一つの JsContainer が接続される．

4.5　データの設計― JsData/ JsDataInt/ JsDataArray ―

この「線」は「データ」を流すための道である．「データ」の形式は，単純な数値であることもあるが，物理次元などの付加情報を持った数値，pixel データのような複数の数値の集合など，様々な形式を持つことが想定される．しかしながら，JsConnector はデータ形式にかかわらず，データを入力側から出力側へ運べばよい．そのため，JASMINE Simulator で扱うすべてのデータはJsData という抽象クラスの継承として構築することにする．

JASMINE Simulator で扱うデータは，整数，実数，文字列などの基本的なもの，実数に物理次元の属性を付与したものがある．また，時系列データや pixel などのアレイデータのように，これらの集約も考えられる．集約には，今まで何度か出てきた Composite パターンを使う．また，物理次元の付与には物理次元を取り扱うためのクラスを別に定義して，Decorator パターンを用いるのが便利である．（Decorator パターンの内容については[6]）

4.6　モデル（JsModel）の設計

実際に衛星を使った観測をシミュレーションするには，当然のことながら宇宙・光学系・検出器などの動作を表す細かな機能が必要となる．第3章でも述べたように，これらは抽象的な「モデル」を表すクラスのサブクラスとして実装される．言い換えれば，「モデル」を具体化したもの，「具体的なモデル」が必要となる．抽象的な「モデル」は，図 7 では JsModel として表され，具体的なモデルを表す UniverseModel クラス，DetectorModel クラス，OpticsModel クラスなどはこれを継承して作られる．JsModel はJsContainer に保持されていて，JsModel のレベルでは適切に情報を受け渡す手段だけを決めておく．他のモデルとのつながりは，JsContainer が JsConnector を使って処理する．従って，JsModel はJsContainer との情報のやりとりにだけ気を付ければ良く，他の JsModel とは直接つながらないので，他の JsModel の構造を知る必要はない．

また，複数の JsModel が情報を伝達するという視点で見ると，例えば光学系において口径や回折限界，焦点距離などの数値を任意に設定すること

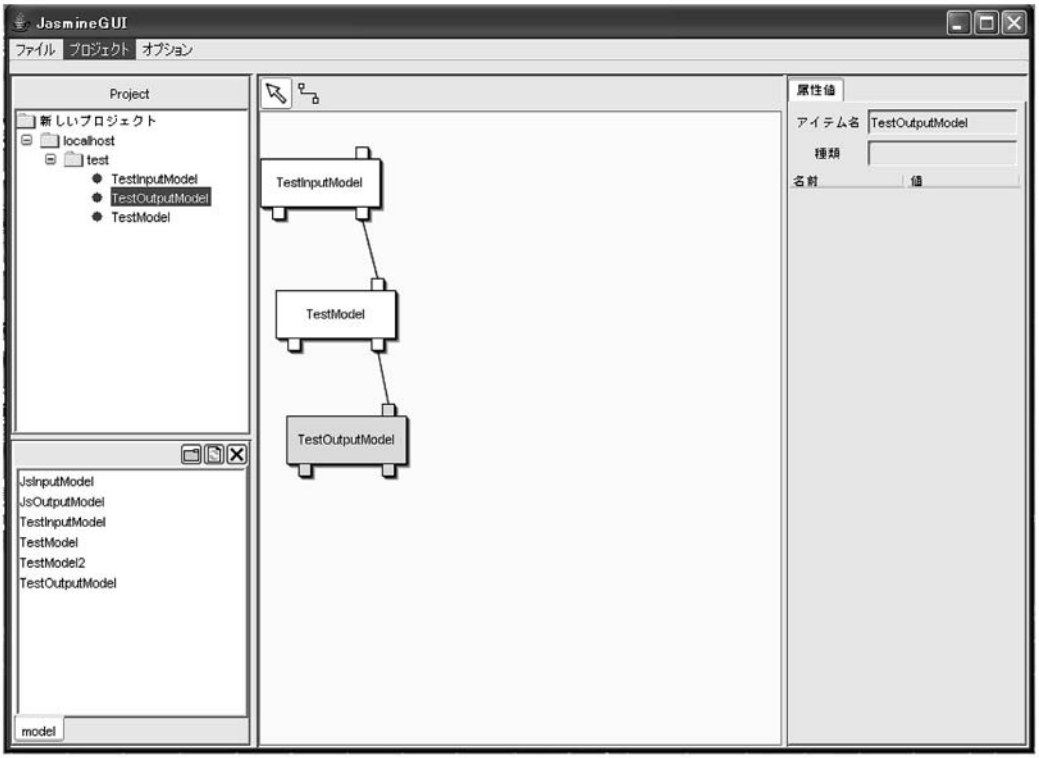

図 9. JASMINE Simulator の起動画面．中央に，依存性グラフが表示される．グラフの構築には，左側のウインドーにある部品一覧の表示が利用できる．ここから必要な部品を持ってきて，中央の画面に置く．選択された部品については，右側のウインドーで属性値を入力することが出来る．ツールバーには，以前に構築したシミュレーションのグラフを読み込んだり，編集した結果を保存したり，現在開いている依存性グラフから計算を実行するための機能が割り当てられている．

ができるし，こういった状態を表す値を外部から取得することもできる．JsModel は非常に多様な「状態」を持ち得る．一方，たとえば光学系の口径を変更すれば回折限界が変化する．属性一つ一つに対して個別に外部から変更可能な手段を提供するとすれば，一つの属性だけが変更されれば属性が不整合な状態が出現する．例えば外部から口径だけを変更されると，状態が不整合な状態になる．そこで，回折限界を自身によって変更することで，「整合」した状態に戻す必要がある．すなわち，外部との情報のやりとりという観点で見れば，多様な状態は，図 8 で示すように，状態を表す変数の値が「整合している」状態と「整合していない状態」の二つの状態に単純化できる．値を設定したり取得したりすることを許すのは,「整合している」状態の時だけであり,「整合していない」状態になれば自分自身で「整合している状態」に戻す機能が必要である．

値の整合性を確保する方法は JsModel のサブクラスである具体的なモデルのクラスの作成者しかわからない．従って，整合性を確保するための関数は，サブクラスの作成者が実装することになる．JsModel と JsContainer という二つのオブジェクトに分離したメリットは，この関数がどのようなタイミングで呼び出されるべきかは JsContainer が管理しているので，具体的なモデルのクラスの作成者はそのことを考える必要がないことである．外部から値を設定する操作は必ず JsContainer を介するので，JsContainer が値を設定した後から次に JsContainer が値を読み出す前までの間の適切なタイミングで，JsModel にたいして「値を整合させる」よう指示すれば良い．従って，光学系や検出器などを表す JsModel のサブクラスの作成者は，

・どのような値が外部のから設定あるいは参照されるのか
・値を整合するための手段

だけに集中し，それをどのように受け渡し，どのタイミングで実行するのかということを考えることから解放される．

5. JASMINE Simulator の実装

前の章でも説明したとおり，実装段階は設計に則して具体的にプログラムを構築する段階である．ここでは，必要な機能をどのような形で実現するかについて考える．

編集内容の保存形式としては XML を採用する．

編集された生成物（JASMINE Simulator ではプロジェクトと名付けている）は，「箱」と「線」のつながりを表していて，どういった機能を持った箱かは「箱」の属性として，流れるデータの種類は「線」の属性として位置付けることが出来る．こういった構造化されたデータを記述するのに，XML は非常に適している．

シミュレーションを制御するにはスライダーや選択ボックスといった入力機構が必要である．編集機構を実装すると，編集画面がある．値を入力する場合，編集画面の中の「箱」を選択すると，この箱に対応する入力可能な値が，適切な入力機構とともに表示されるというのが自然である．そこで，図 9 に示すように編集画面を中央に，値を設定する画面を画面右側に配置する．画面左側は，編集をサポートするために，多くの種類の「箱」が用意されている．出力機構は，必要な場合に新たに画面を開いて出力を行なうようにする．出力内容は，単純なテキストで示されるものもあれば，グラフや画像になる場合もある．

分散処理の実装にはいろいろ考えられるが，他のシステムとの互換性や標準化動向などを考慮して，CORBA を採用している．

こういった実装上の便を考慮して，開発言語は java を選択した．java を選択した理由は，

- 大きな計算に使用される標準的な言語の中では他の言語に比較して Objsect 指向性が優れていること，
- 実行プラットフォームを選ばないこと，
- 描画・ウィジェット・XML 処理・分散処理などに必要な仕組みが提供されている

である．

6. まとめ

今回我々は，光学系や検出器などと言ったシミュレーションに必要なコンポーネントを統一的に取り扱う機構，これらの複数のコンポーネントを組合せて依存性グラフを編集する機能，コンポーネントに必要なパラメータを与える機能，シミュレーションの実行を制御する機能，編集された依存性グラフを保存・読み出すための機能といった，JASMINE Simulator の骨格となる機能を持つプログラムを構築した．

今後は，実際のニーズに従って具体的な「箱」や具体的な「データ」の種類を増やして行くことで，より充実したシステムにして行くとともに，このツールを JASMINE の仕様検討に役立てて行きたいと考えている．現在の機能は，簡単な量の計算を行なって要求仕様を割り出すに留まっているが，実際にCCD 上の星像の様子を模擬したり，複雑な擾乱源・誤差要因を取り扱えるようにし，また複数のデータ解析の手法を比較検討することにより必要な観測回数を割り出すなど，実際のプロジェクトに即した機能を実装して行く必要がある．アウトプットも現在は数値を表示するだけだが，グラフなどのより視覚的なアウトプット形式をサポートする予定である．

7. 謝　辞

本研究では，IBM 東京基礎研究所の宮下尚氏にもアドバイザーとして加わって頂き，大変有益な意見を頂いた．また，この研究は，国立天文台共同研究及び文部科学省科学研究費補助金基盤研究（B）（2）（課題番号15340066），並びに東レ科学技術研究助成金によるサポートを受けたものである．

参考文献

1） N. Gouda et al: SPIE, 4850, 1161(2003)
2） http://www.jasmine-galaxy.org/index-j.html
3） W. O'Mullane and L. Lindegren: An Object-Oriented Framework for GAIA Data Processing *Baltic Astronomy* **8,** 57－72(1999).
4） X. Luri and C. Babusiaux: The GAIA Simulator: reference document, *GAIA-SWG-001* (2001).
5） 具志堅隆児,垣花一成,倉骨彰: “実践的 UML 入門－IIOSS で始める新世紀プログラミング”，株式会社ASCII (2002).
6） E. Gamma, R. Helm, R. Johnson and J. 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” (1995) (邦訳ソフトバンクパブリッシング (1999)).
7） JASMINE チーム:赤外線位置天文観測衛星(JASMINE)計画第一回検討報告書 (2003).