2　シミュレーションモデル・ソフトウェアの共通基盤の構築に向けて

2.1 共通基盤－「汎用型シミュレーションモデル」とは

“汎用型シミュレーションモデル”とは、当該分野のシミュレーションをあまねく行うことができる単一の標準的、総合的なモデル・ソフトウェアを意味するのではない。そうではなくて、本章の標題にあるように、シミュレーションモデル・ソフトウェアの開発や管理、更新、利用、活用などにかかわる共通基盤の構築を目指すものである。具体的には、データ交換・連携インターフェースの機能を有し、異なる解析モデル・ソフトの接続が可能となる流域水循環解析モデル・ソフトのフレームワーク（モデル・ソフトを構築するためのシステム（仕組み））があり、その下で様々なモデル・ソフト群が目的に応じて連動する状態を支えるものと捉えることとする。この“汎用型シミュレーションモデル”によって、様々なユーザーが水文・水理・水質現象に関わる種々のモデル・ソフトを任意に組み合わせて、必要な入力データを効率よく処理しつつ、求める計算を実行できる環境を得ることができる（図 2-1 参照）。

このような定義の「汎用型シミュレーションモデル」は、モデル・ソフトそのものではなく、モデリングのシステムを共通化しようとする考えに立つ。今後作られるモデル・ソフトがニーズ、技術の進展や対象とする現象の拡がりに応じて様々なものになっていく可能性、独立したプロセスごとにモデル・ソフトを開発し進化させ、あるいは評価され、その時々の目的に応じてそれらを組み合わせて行く方式の持つ合理性（「要素モデル」の組み合わせ）、様々なセクターがモデル・ソフト（要素モデル）の開発に参画するインセンティブを持てるようにすることの重要性などに鑑みて、このような「汎用型シミュレーションモデル」が今後のモデル・ソフトのあり方を考える際の有力なひな形になっていくものと考えられる。

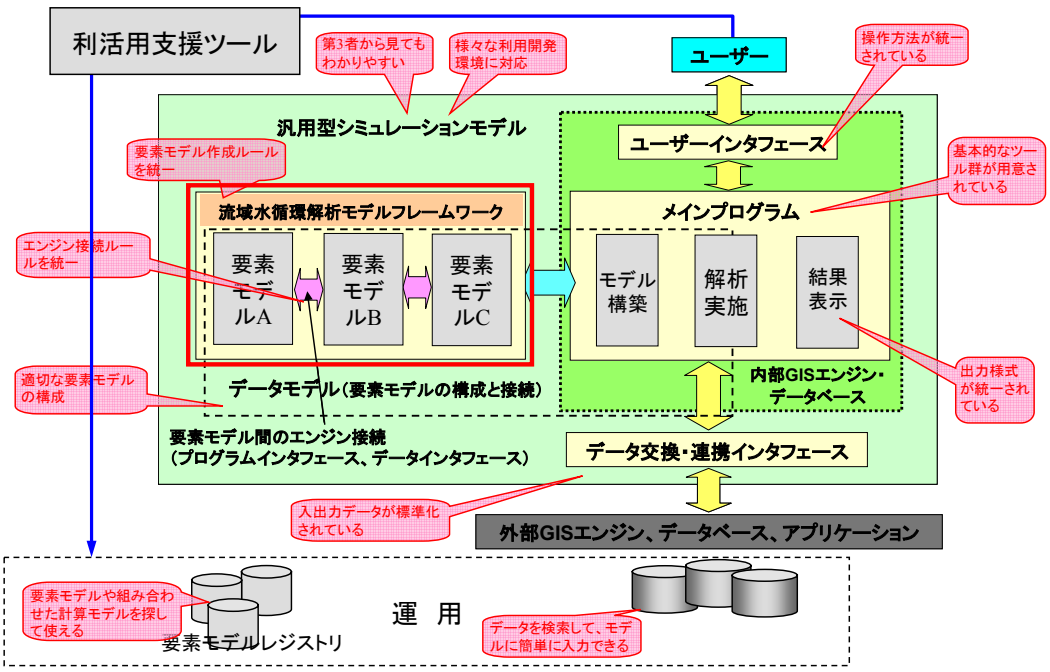


図 2-1　モデル・ソフト群の共通基盤としての汎用型シミュレーションモデル

もちろん、概念的にはこのような方式が良いとしても、実現性について具体的な見通しが無ければ議論にとどまる。以下では、「汎用型シミュレーションモデル」を実現する方法を検討した内容を１つの提案として示し、モデル・ソフトの共通基盤づくりに向けた具体的動きにつなげることをねらう。なお、モデリングのシステムの共通化が重要であっても、その下で動く個々のモデル・ソフト（要素モデル）が具体のニーズに直接応えるツールであることには変わりはなく、本章の内容は、個々のモデル・ソフト（要素モデル）の存在を前提する。また、「汎用型シミュレーションモデル」を構築する際に、その下で動くモデル・ソフト（要素モデル）を想定する必要があり、その意味でも、代表的なモデル・ソフト（群）の存在は「汎用型シミュレーションモデル」と車の両輪をなすと言える。さらに、既存のモデル・ソフトの資産を生かすことも重要な検討対象となる。

汎用型シミュレーションモデルの構築に向けて、大きく(1)データ構造の標準化、(2)インターフェースの標準化、(3)プログラム構造の標準化を満たすことが求められる。そのつながりのイメージを図 2-2に示す。①～⑩のそれぞれの要求事項は、「1)流域水循環解析モデルフレームワーク」、「2)データ構造標準化」、「3)河川 GIS・アプリケーションインターフェース」、「4)ユーザーインターフェース」、「5)運用、維持管理の枠組み」の中の事項として位置づけられる。なお、要素モデルに関する詳細な説明は 2.6.7 にて行う。

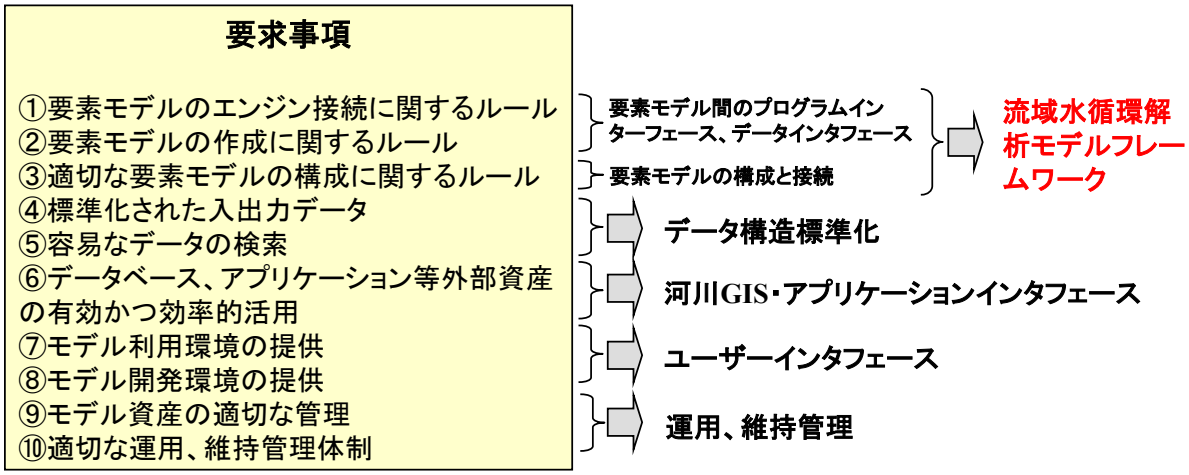


図 2-2　汎用型シミュレーションモデル

2.2 データ構造の標準化

現在、日本における河川に関する電子データの標準化状況は、図 2-3に示すようにデータの電子化、公表は進んでいるが（例：水文水質データベース、水路網データガイドライン）、水理・水文・水質シミュレーションに関するデータの公表は進んでおらず、データは存在しているものの、多種多様のデータ構造であり、それらは散在し、統一的に管理されていない。

これらの問題を解決するには、利用者が必要とするデータについて、標準化ガイドラインを策定し、それらを運用するルール、仕組みを構築する必要がある。

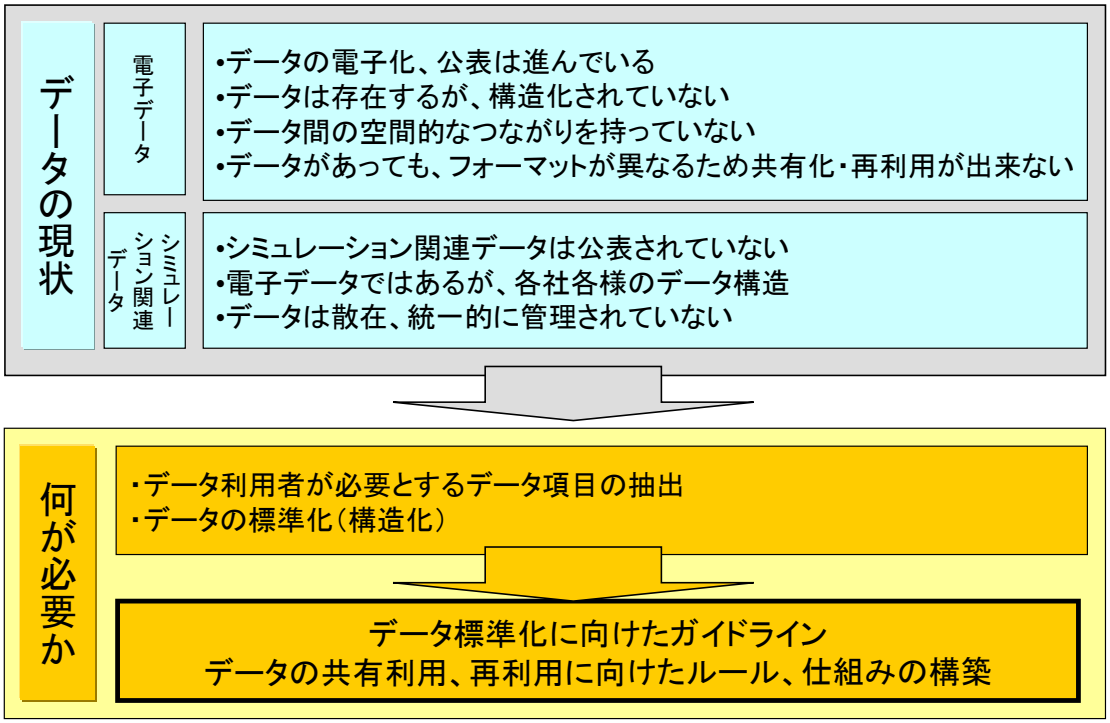


図 2-3　データの現状と標準化の必要性

データの構造化、標準化について、以下のように定義する。

データの構造化	各々のデータについて、その定義を明確にするとともに、データ間の関係を誰にでも分かるように整理し、
データの標準化	データの構造化における共通ルール、その運用に向けたガイドライン等を作成すること

図 2-4　データの構造化と標準化の定義

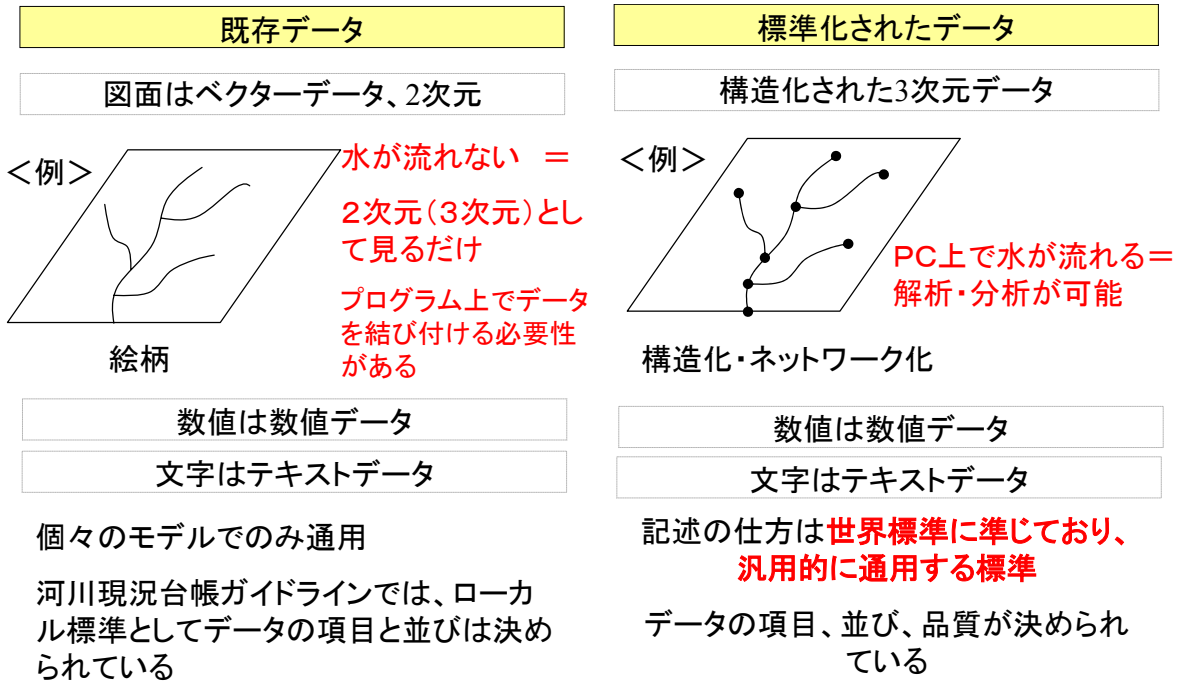


図 2-5　既存データと標準化されたデータの違い

これまで、標準化データとして作成されてきたデータは、例えば、河川を表現する線データがあったとしても、絵として河川が表現されているのみで、“意味”として河川を表現しているわけではなく、また、河川・水路の分合流についても、そこに分流、合流の意味が付与されているわけではないので、既存データを用いて、水理解析を行う際には、改めて、河川・水路としての、あるいは分合流の意味を付与しなければならない。

これでは、標準化データとして定義、整備されたとしても、実際の利用に供する際には様々な加工を施さなければならず、利活用の範囲は限定されてしまう。また、同じデータを利用しても、解析や検討の結果は異なるものとなってしまう。

これらの理由から、河川に関するデータの構造化・標準化では、河川・水路の意味、分合流の意味を付与されたデータ、すなわち解析等の実利用に供することが可能なデータを整備する方針で作業を進めており[2)]、具体的には、次図に示す内容を持ったデータを整備している。

既存データ	標準化されたデータ
図面はベクターデータ、2次元	構造化された3次元データ
■ （X,Y）座標が入ったベクトル	■ （X,Y,Z）座標が入ったベクトル ■ 水路間の接続関係、接続構造 ■ 接続の条件（例えば、操作規則） ■ 水路の持っている属性（管理者、設置年月日・・・）

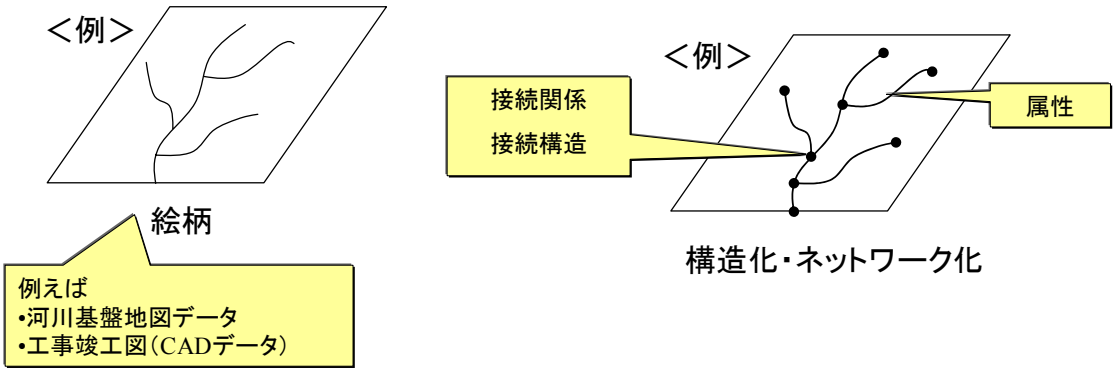
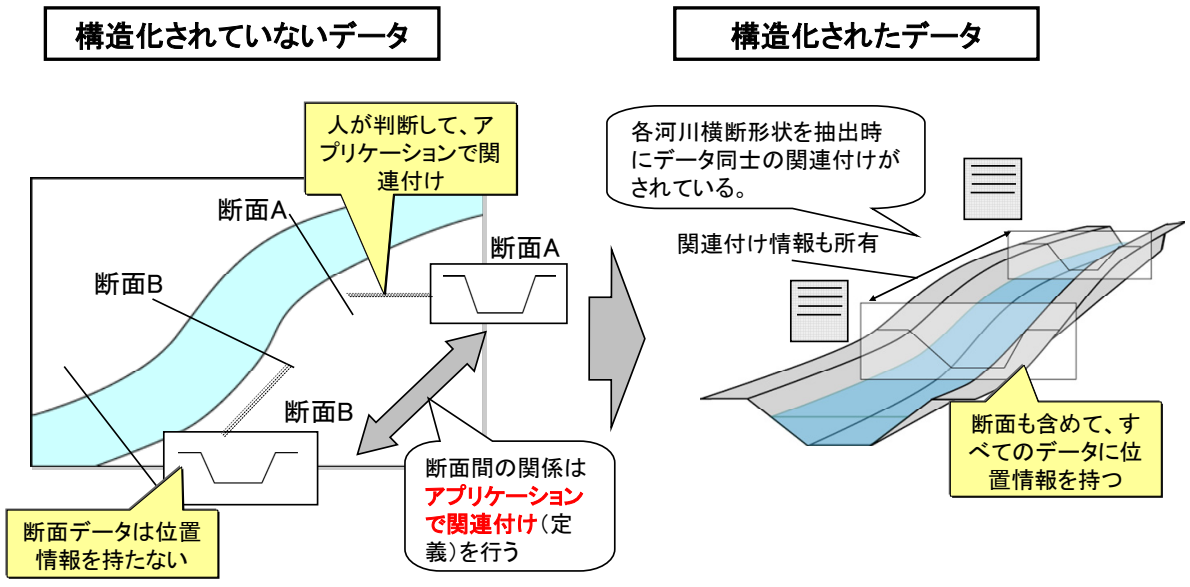


図 2-6　既存データと標準化データの内容



データ量が少ないので、関連付け等をすべてアプリケーションで対応	データが関連付け等の情報も持っており、アプリケーションの負荷が少ない

図 2-7　構造化データのイメージ

また、データの標準化を進めることで、同じ解析を行う別のプログラム間でのデータの共有利用が実現し、これまで、大きな時間と作業量を費やしていたデータ作成業務、データ作成・加工作業が減少することが期待されている。

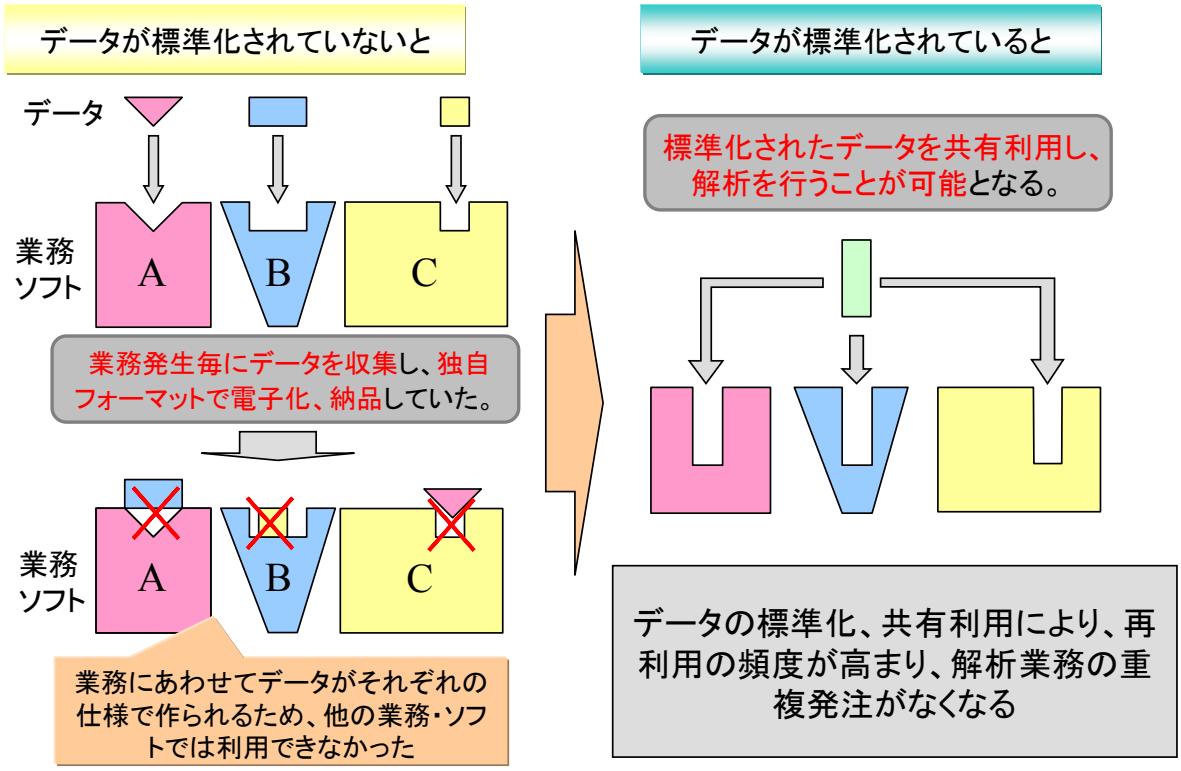


図 2-8　標準化データによるデータの共有利用

2.3 インターフェースの標準化

2.3.1 インターフェースの標準化とは

ここでのインターフェースとは、下図に示すシミュレーションモデルの全体図において、データベースとアプリケーション間のデータの送受信にかかわるものを指し、降雨予測、流出計算、河道計算等、さまざまな水理・水文・水質シミュレーションモデルと、それらが共通して利用するデータを接続する仕様、関数、ルールを定めることである。

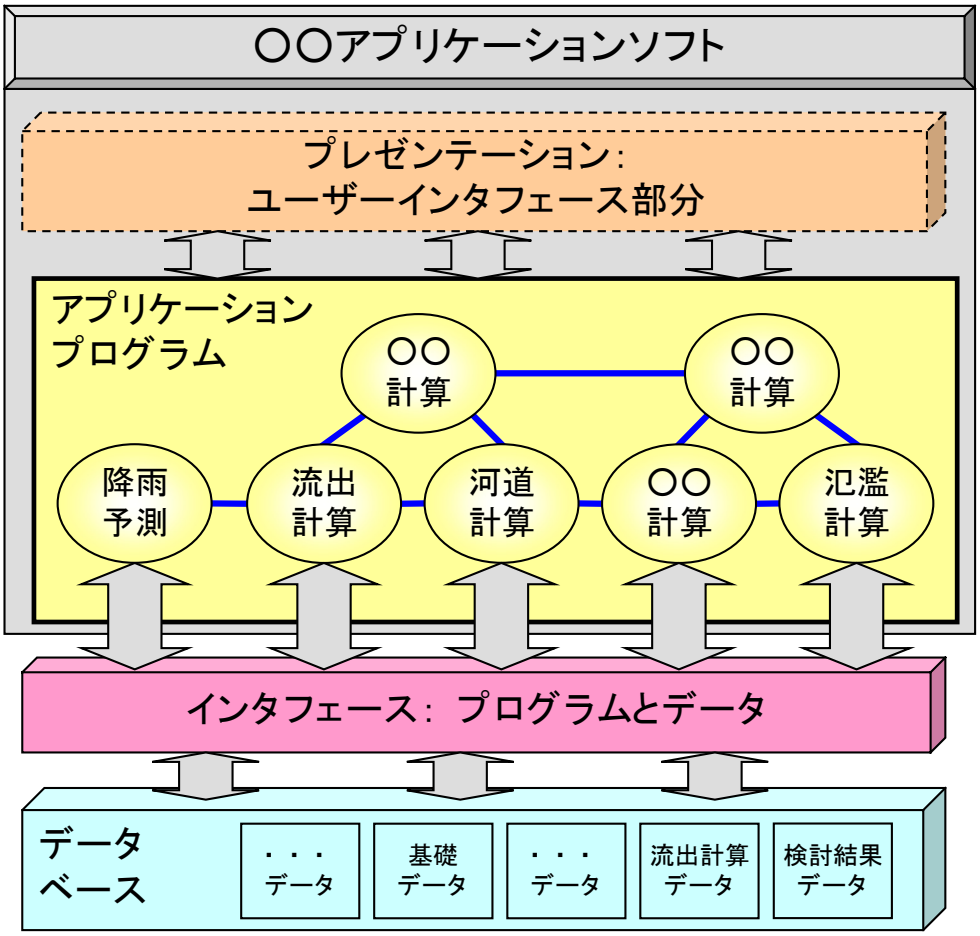


図 2-9　インターフェースの構成

2.3.2 インターフェース標準化の方向

河川局では、現在、図 2-10 に示す範囲でのインターフェースの標準化を進めている[2)]。

汎用型シミュレーションモデルにおけるインターフェースにおいても、この仕様を基本に策定を進めるべきであるが、水理・水文・水質シミュレーションモデルに関する必要な関数仕様等については議論が十分でないため、この標準仕様を参考に必要なインターフェースについても、議論・検討していく必要がある。

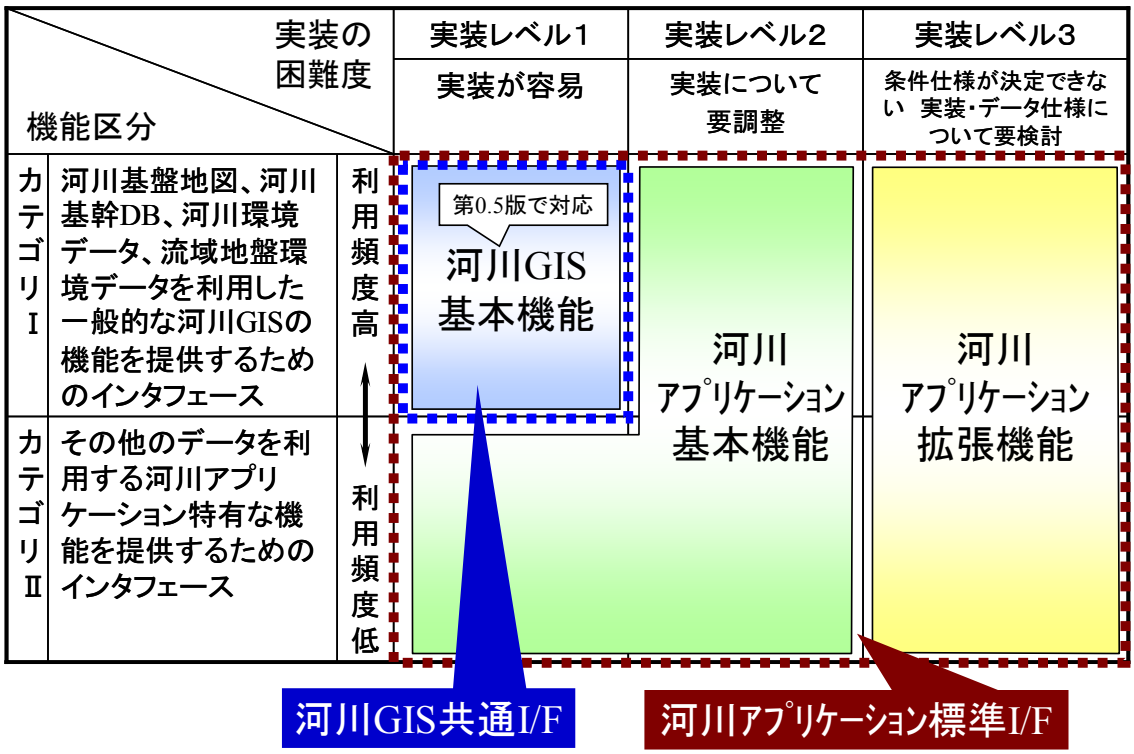


図 2-10　河川 GIS・アプリケーション標準インターフェースの区分

2.4 プログラム構造の標準化

2.4.1 プログラム構造の標準化とは

シミュレーションモデルの開発はこれまで個々の主体により行われ、他者より優れたソフトウェアを構築することが、研究者や企業にとっての開発インセンティブとなってきた。このため、開発に係わるノウハウは共有化されることなく、多種多様なモデルが構築されてきたと言える。汎用性に富み、性能の優れたソフトウェアを提供していくためには、開発に係わる技術情報の共有化による開発効率の向上や、統一基準に基づく個々のプログラムモジュールの組み合わせ、相互補完が考えられる。

前述のように、汎用型シミュレーションモデルとは、データ構造、インターフェース、アプリケーションプログラムのそれぞれが標準化されたものである。プログラム構造の標準化ならびにデータ構造の標準化、インターフェースの標準化によって、

- 要素モデルが容易に変更できる。
- 要素モデルを組み合わせて、規模の大きいモデルを構築できる。
- 流域水物質循環のシミュレーションモデルの開発の効率が向上する。
- より流域の課題解決に向けたソリューションを提供できる。
- インターフェースの作成に労力をかけずに、要素モデルの開発に注力できる。

プログラム構造の標準化において、重要な概念となるのが次項で説明する標準フレームワークである。

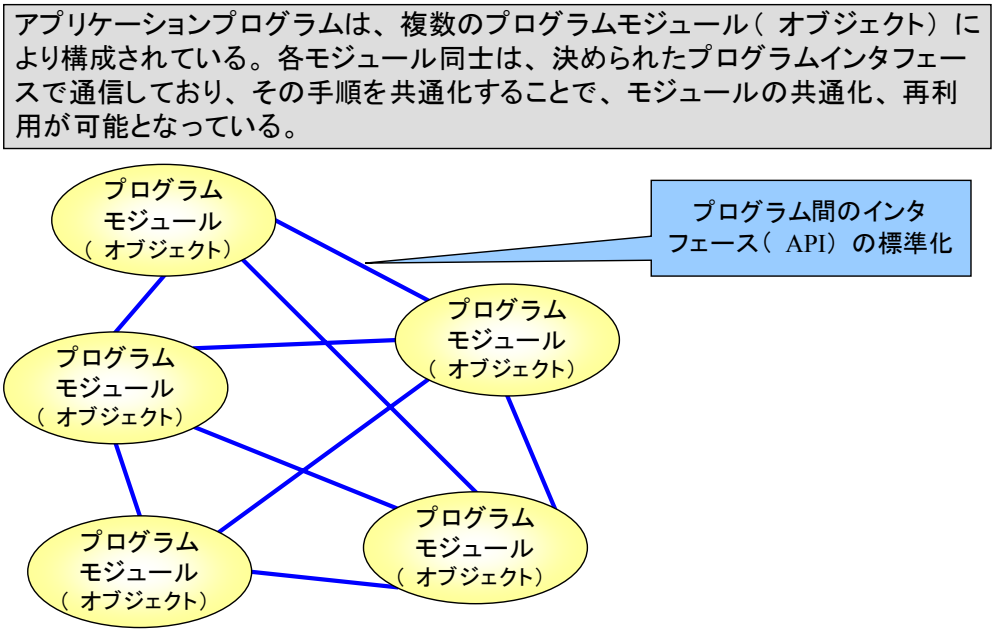


図2-11　プログラム構造の標準化

2.4.2 標準フレームワークとは

標準フレームワークとは、汎用型シミュレーションモデルを構成するコアエンジンのシステム部分のことを指し、その機能を下記の 3 点に要約する。

- 標準フレームワークとは、モデルそのものではなく、モデルを構築するためのシステム（仕組み）のことである。

- 標準フレームワークの仕様に基づいて、種々の要素モデルを構築することが出来る。
- それぞれの要素モデルを自由自在に相互接続し、複合的な物理現象をシミュレートする全体系モデルを構築することが出来る。

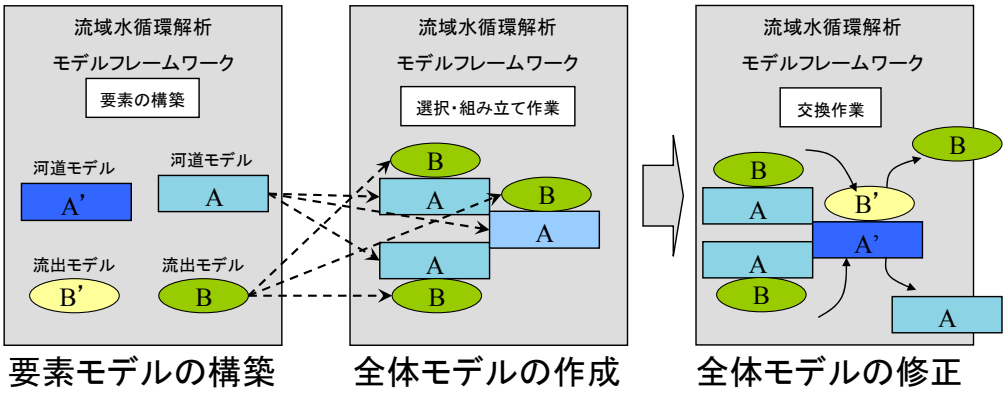


図 2-12　標準フレームワークのイメージ～流域水循環解析モデルを例に～[3)]

標準フレームワークの仕様に基づいて、要素モデルを構築することが出来る仕組みが整っていることで、様々な開発者が流域水循環に関係する要素モデルを多種多様にコーディング（ソースコードを作成すること）することが可能である。それぞれの要素モデルの相互接続の規定（プログラムインターフェース、データインターフェース）も予め定められていることから、自由自在に要素モデルを相互接続し、複合的な物理現象をシミュレートする全体系モデルを構築することも可能である。種々の物理現象を説明する要素モデルが多いほど、その中から取捨選択して、ある流域の水循環解析にカスタマイズした全体系モデルの構築が可能と期待できる。また、ある全体系モデルを構成している要素モデルを別の要素モデルに交換して応答を見てみたいといったニーズに容易に答えることが可能となる。

2.4.3 標準フレームワークのコンセプト

“標準フレームワーク”とは、データ交換・連携インターフェースの機能を有し、異なる解析モデルの接続を可能とし、複合的な流域水循環解析を実現するためのモデルフレームワークである。すなわち、この標準フレームワークによって、様々なユーザーが水文・水理現象に関わる種々のモデルを任意に組み合わせて、必要な入力データを効率よく処理しつつ、求める計算を実行できる環境を得ることができる。

また、今後必要となる水工系以外の解析プログラムと連携させる複合的な解析に対応させるべきものでなければならない。

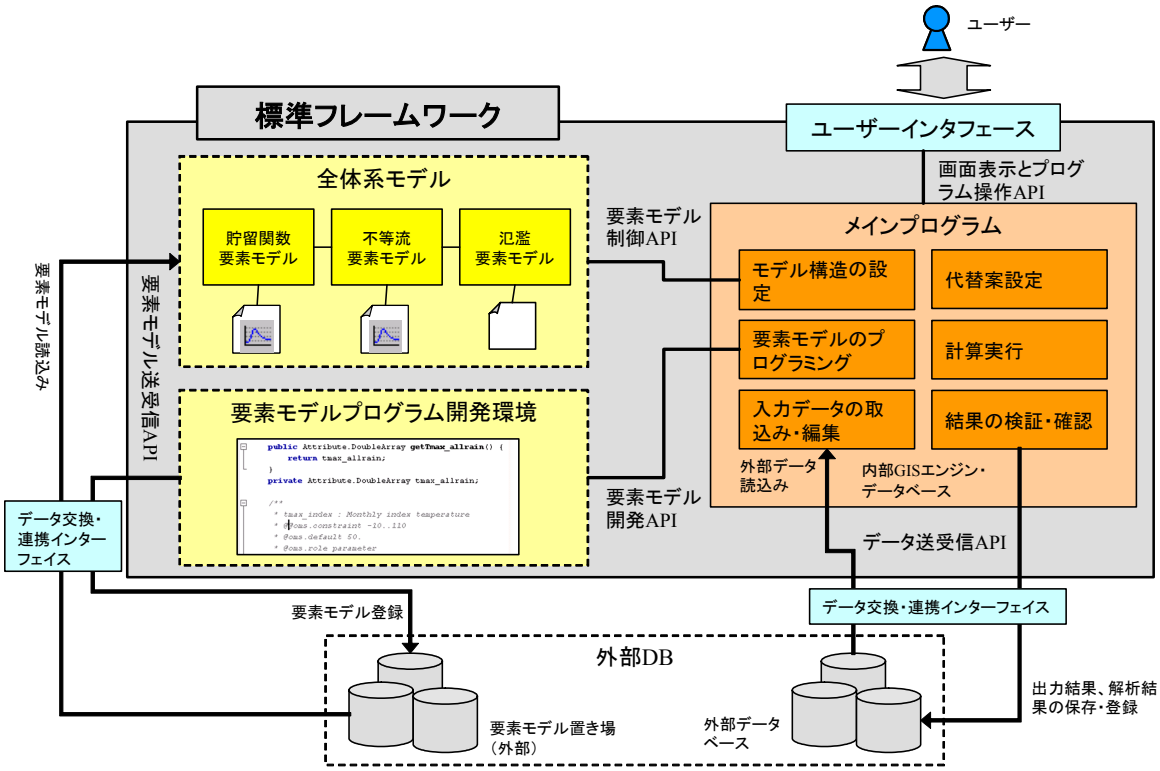


図 2-13　標準フレームワークの機能イメージ図

2.4.4 標準フレームワークの開発効果

標準フレームワークを開発することによって得られると期待される効果の要点をまとめると以下の項目が挙げられる。

- ■ 異なった環境で作成されたソフトが流通し、競争が始まるので
  - 研究が活性化する
  - 水循環・水環境行政が高度化でき、新しい河川や水利用に関する施策が提案される
  - 学生や技術者が直接ソフトを操作できるので、技術力が向上する
- ■ ソフトの重複開発が避けられ、メンテナンスコストが下がり、行政機関とコンサルタンツ、研究機関すべてで、生産性が向上する
- ■ ソフトの品質がオープンになるので、国民間の政策合意形成に寄与される
- ■ アジアの特徴を表現でき、外国との競争力が高まる

以下、標準フレームワークの開発に向けて、プログラミング言語について検討する。

2.5 標準フレームワークの開発言語

2.5.1 標準フレームワークの開発言語の検討

ここでは、標準フレームワークの開発言語について検討する。これまでも述べてきたように標準フレームワークは水理・水文・水質解析の共通基盤となるフレームワークであり、完成されたモデル・ソフトウェアを開発するわけではない。また、様々な技術レベルのユーザーによる利用を想定したものであり、フレームワークに自ら要素モデルを加えるなどして利用するユーザーもいるだろう。従って、フレームワークを開発するプログラム言語が適しているかについて調査検討しておく必要がある。

2.5.2 オブジェクト指向型言語の概要

オブジェクト指向とは、ソースコードをクラスと呼ばれる単位に分割し、それらを統合することによりプログラムを構築することであり、こういった方法でプログラミングする言語のことをオブジェクト指向型言語という。代表的なオブジェクト指向型言語としては、C++、C#、Java が挙げられる。手続き型言語にはないオブジェクト指向型言語の特徴として

・プログラムの作成
・プログラムの再利用
・プログラムの保守

のしやすさが挙げられる。表 2-1 は、オブジェクト指向型言語と手続き型言語の違いを説明したものである。オブジェクト指向型プログラムでは、「あたかも、オブジェクトが自分で何でも知っていて、自分で何でも自律的に行動するようにプログラミングする」ので、「クラスオブジェクトのユーザーは、そのオブジェクトに命令する（メッセージを送る）だけでOK」である。手続き型言語が、「よちよち歩きの子供を手取り足取り世話するイメージ」であるのに対し、オブジェクト指向型言語のクラスオブジェクトは、「自律的に行動できる大人というイメージ」である。

表 2-1　手続き型言語とオブジェクト指向型言語

プログラム技法	説明	主な言語
手続き型言語 (procedural language)	処理を必要最小限の手続き単位に分解・作成し、これを必要とする処理に基づいて組み立てていく手続き指向の言語	FORTRAN　C COBOL Pascal　etc.
オブジェクト指向型言語 (object oriented language)	プログラムをオブジェクト（物）の単位としてとらえ、オブジェクトの相互的な作用によって処理を行う、オブジェクト指向の言語	Smalltalk　C++ Java Visual Basic　etc.

2.5.3 オブジェクト指向型プログラムの重要な概念

オブジェクト指向型プログラムが、プログラムの作成、プログラムの再利用、プログラムの保守を容易にするのは、先の述べたようにプログラムの要素がオブジェクトとして自律的に行動することにあるが、どのようにしてこのようなプログラミングが実現されるのか。ここではオブジェクト指向型プログラムの重要な概念を説明する。

①クラス、インスタンス

クラスとは共通の特性（属性、振る舞い）を持つオブジェクトの集合である。モデリングシステムにおいては、クラスとはあるデータ群とそれらを操作する関数群を一つのパッケージにまとめたものを指す。このデータ群をデータメンバ、関数群をメンバ関数と呼ぶ。要素モデルは状態量やパラメータといったデータ群を、水文特性を表現する数式群で操作することによって計算を行うので、状態量やパラメータをデータメンバ、数式群をメンバ関数とすれば、要素モデルをクラスで表現することができる。

クラスは様々なプログラムで共通に利用することができる。例えば、プリントアウト機能は、あらゆるソフトウェアで必要な機能である。こういったとき、プリントアウトとする機能を実装したクラスがあれば、様々なソフトウェアを開発するときにこのクラスを自分でコーディングすることなく、共通して利用すればよい。また、高度なプログラミング技術がなくても、他者が実装した高度なクラスを自分のプログラムに組み込むことも可能で、インターネット上にも無償で高性能なクラスが公開されていることもある。

このように、ソースコードを機能ごとに分割してプログラミングすることにより、一つのプログラムを構築するのに複数のプログラマがそれぞれクラスを作成するという作業分担ができるため、ソフトウェア開発の生産性が向上される。また、別のプログラムに以前作成したクラスを再利用することもできる。

②メッセージ

メッセージとは、オブジェクトに対する指示や設定のことである。すなわちオブジェクトに対して、プロパティを設定したり、メソッドを動かすように指示することをメッセージという。このメッセージを送ることを”メッセージパッシング”という。

図 2-14 の例では、オブジェクトに「ベルを止める」というメッセージパッシングをしているが、プロパティそのものを設定する、たとえば現在時刻を設定する、目覚まし時刻を設定するなどのプロパティの設定ももちろんメッセージパッシングである。ちなみにベルを止めるというメッセージパッシングは、実生活では目覚まし時計のボタンを押すという行為になる。

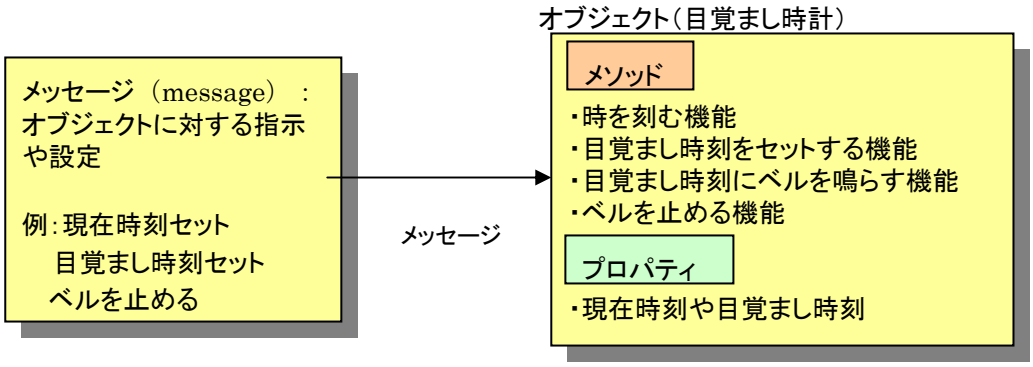


図 2-14　メッセージの概念

③継承（インヘリタンス）

現在、大規模なソフトウェアやプログラムはほぼオブジェクト指向型言語を用いて作成されているといってもよく、プログラミング言語の主役となっている。その理由は、クラスを共通利用できるという点に留まらない。

オブジェクト指向型言語には、「継承」という機能が含まれている。継承とは新たなクラスを作成する際に既存のクラスの機能を引き継ぐことである。例えば、図 2-15 に示すようにある機能を持ったクラス（親クラス）から新たなクラス（子クラス）を作成すると、親クラスが持っている機能は全て子クラスが受け継ぐことになり、追加したい機能だけを子クラスに実装すれば、わざわざ子クラスに必要な全ての機能をコーディングすることなく子クラスは「親クラスの機能+追加した機能」を持つことができる。このように継承機能を有するにより、新たにクラスを作成することも容易になる。

例えば、目覚まし時計、腕時計、大時計という子クラスは、時計という親クラスのメソッドとプロパティを自動的に継承した上で、それぞれの子クラス独自のメソッドとプロパティを定義することになる。

水文モデルで言えば、流域流出モデルという親クラスは、雨から流量を算出するというメソッドを持ち、面積というプロパティを有する。その下の子クラスである貯留関数法では、貯留方程式というメソッドと流出率、パラメータ K,P,Tl というプロパティを有することになる。

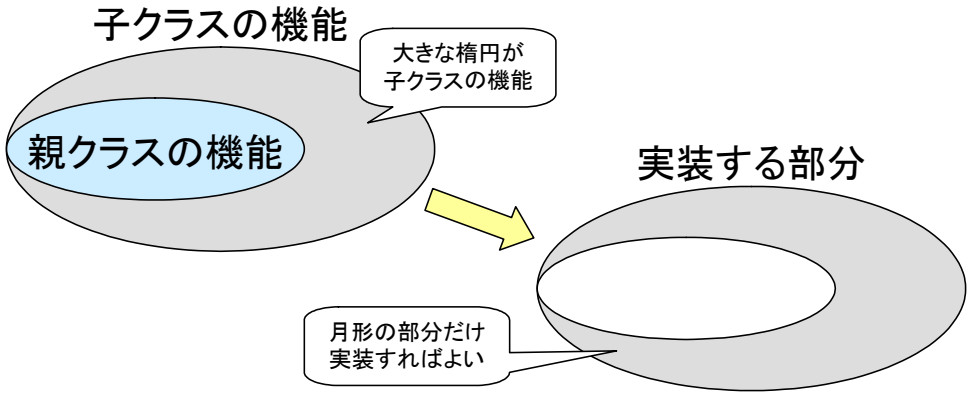
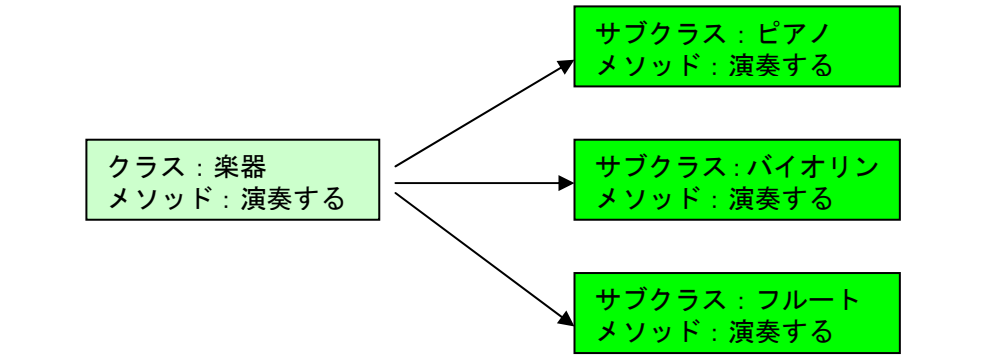


図 2-15　クラスの継承

④多態性（ポリモフィズム）

２つのクラス A，B が全く同じ機能を備える場合には継承を利用すればよいことがわかった。しかし、機能によっては、A，B ともにその機能を備えることがわかっていても、その内容までは同一に定義できないものもある。例えば、全ての要素モデルがその水文特性に応じて数理計算を行う機能を備えるのはわかっているが、その内容は当然ながらそれぞれ異なる。しかし、『水文学的な計算を行う』ということに関しては共通であり、その手続きを統一することで利便性・汎用性を高めることができる。

このように、クラスのメンバ関数の使用方法のみ（メソッド）を規格化し、その操作により実現される内容（メソッドの方法）を個々の派生クラスごとに定義できることを多態性という。下図の例で、スーパークラスの「楽器」で「演奏する」というメソッドが定義されているが、このメソッドは、ピアノクラスではピアノの演奏を、バイオリンクラスではバイオリンの演奏をするという意味を持つ。例えば、流域流出モデルで、雨から流量への変換を行うというメソッドを定め、それを「Calculate」という名前にすれば、貯留関数や kinematic wave のクラスで具体的に計算式を定義すればよいことになる。



「演奏する」というメソッドを継承しているが、それぞれのサブクラスで「演奏する」メソッドの方法を独自に定義できる

図 2-16　多態性の概念

2.6 既存のフレームワークの調査

前節まで説明してきたモデル構造の標準化を意識した水理・水文に関するフレームワークの開発が進められてきている。例えば、「標準フレームワーク」の概念に近いシステムとして、欧米で OpenMI(欧州関係機関が推進する HarmonIT プロジェクト)、MMS（米国地質調査所（USGS）が推進するプロジェクト）、OMS（米国農務省（USDA）が推進するプロジェクト）として開発されてきているが（詳細は３章で説明）、日本においても京都大学椎葉研究室で開発・保守・管理されている「OHyMoS（Object-oriented Hydrological Modeling System・オハイモス）」がある。本節では、「OHyMoS」の調査結果を整理する。

2.6.1 OHyMoS とは

OHyMoS とは、Object-oriented Hydrological Modeling System の略であり、オブジェクト指向型水文モデリングシステムである。その概要は、「水理・水文モデルの共通基盤を開発し、その基盤上でモデルを開発させ、モデル同士の接続を可能にすることにより、流域規模の統合型水文解析を構築するための水理水文モデル構築システム」である（図 2-17 参照）。

OHyMoS のような標準フレームワークや標準的な要素モデルを広く公開し、誰もが利用することができる仕組みがあれば、他機関が開発した要素モデルを取り入れ、新たなモデルを構築することもできるし、自身で作成した要素モデルと結合させることもできる。また、数値計算の技術を持たない者であっても、公開されている要素モデル同士を接続し、解析モデルを構築することができるため、水理・水文計算を行うことができる。

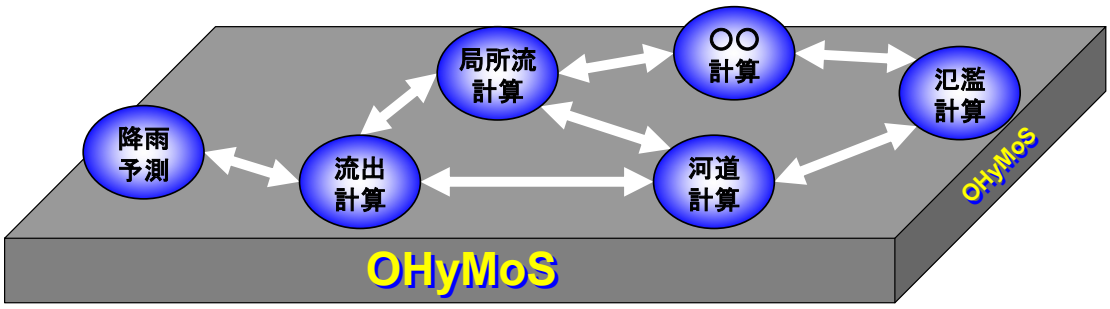


図 2-17　OHyMoS の概念図

2.6.2 開発の経緯

OHyMoS は京都大学の椎葉研究室で開発されたものであり、開発の経緯は次のようである。現在、OHyMoS はバージョン 5 まで開発されており、オブジェクト指向型言語である C++ で記述されている。また、web との親和性が高い Java（Java もオブジェクト指向型言語）で記述された OHyMoSJ も開発されており、バージョン 2 まで開発されている。OHyMoS、OHyMoSJ 共に、web サイト（http://hywr.kuciv.kyoto-u.ac.jp/ohymos.html）でソースコードから公開されており、いくつかのサンプルプログラム（シンプルな数値解析モデルが主）と共に OHyMoS および OHyMoSJ をダウンロードすることができる。

・1991-2 年：原型の開発

・1993-4 年：現在のシステムがほぼ完成
・2003 年：Java 版 OHyMoS-J の開発

2.6.3 OHyMoS の基本的な概念

OHyMoS の基本概念は図 2-18 のとおりであり、要素モデルを複数個接続することで、全体系モデルを構成し、個々のモデルは交換可能とする。こうすることで、他の要素モデルのコードまで、解読しなくても、要素モデルの付加や交換が可能になるのである。しかし、このようなモデル構成にするためには、A,B,C,D の各要素モデルが独立していることが必要となる。B の動作に A が必要なら、A を D に入れ替えることができない。このように個々の要素モデルを独立とするためには、FORTRAN などの手続き型言語では難しく、個々の要素モデルをオブジェクト（物）として取り扱うオブジェクト指向型言語を用いることになる。

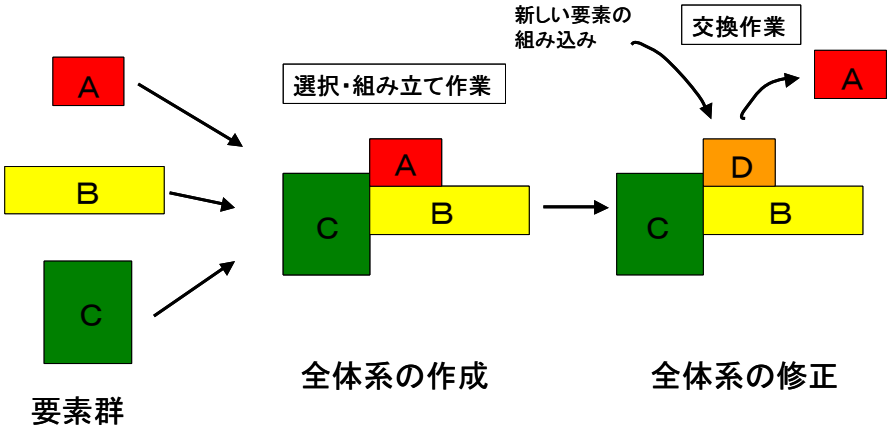


図 2-18　OHyMoS の基本的な概念

2.6.4 OHyMoS の代表的なクラス

OHyMoS を一言で表すならば、クラスライブラリー（同じクラスを様々なプロジェクトで利用できるようにしたもの）であると言うことができる。解析モデルに必要なプログラムを予め C++もしくは Java のクラスとして用意している。OHyMoS が予め用意しているクラスは、モデルのエンジン部分を構成するクラスや、データの交換、ファイルの読み込み、ファイルの吐き出し、要素モデル間のデータの交換などの役割を持つクラスなどである。

OHyMoS のシステムと要素モデルの関係をコンピュータの世界に例えるならば、OS とアプリケーションの関係で例えることができる。コンピュータで稼働させるアプリケーションは共通の OS 上で動かすことを前提に作られているからこそ、アプリケーション間のデータの交換などを行うことができるし、同じアプリケーションを異なるコンピュータで利用することができる。つまり、各々の要素モデルは OHyMoS という仕組みの上で動くように作ることによって、要素モデルの共通利用や、モデル間の情報のやり取りを可能にすることが OHyMoS の基本的な概念の一つである「モデル構成の共通基盤」としての機能である。

また、様々な要素モデルの交換・追加・削除が容易に行うことができることも OHyMoS の特徴の一つであり、他機関が作ったモデル同士であっても、OHyMoS の仕様を満たしてい

る要素モデルであれば、自由に組み合わせることができ、さまざまな水理水文モデルを構築することができる。

要素モデルの新規開発や既往モデルのカスタマイズを行うことも可能で、要素モデルを新規開発する場合であっても、ユーザーは特段プログラミング言語（C++、Java）に精通している必要がないような仕組みが考えられている。

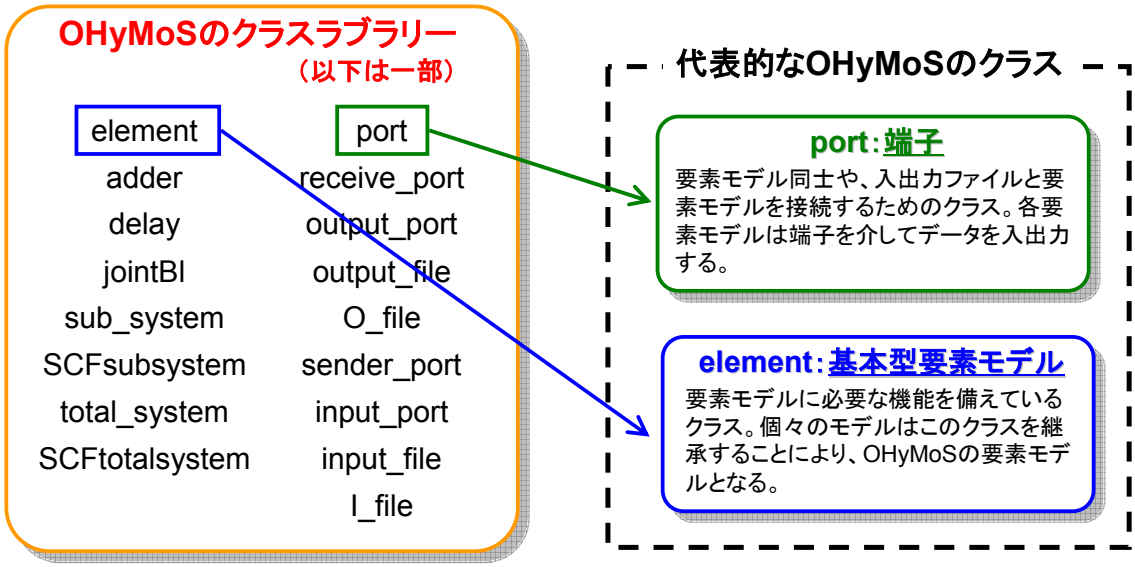


図 2-19　OHyMoS の代表的なクラス

2.6.5 OHyMoS の基本構造

上述したように、OHyMoS では要素モデルの機能を共通の機能と固有の機能に分離し、共通の機能を基本型要素モデルとする。すなわち基本型要素モデルをひとつのベースクラスとする。基本型要素とは、パラメータ値の設定、状態量の初期化、計算時間の更新、時系列データ入力機能などである。次に、各要素モデルは、この基本型要素モデルを継承して、独自機能のプロパティとメソッドを定義することにより作成されることになる（図 2-20 参照）。

図 2-21 にモデルの構成イメージを示す。モデルは要素モデル、部分系モデル、全体系モデルという 3 つの概念を持つ。要素モデルは個々の水文要素に対応するモデルである。部分系モデルは、要素モデルを幾つか統合したモデルである。例えば、土中の水理モデルは中間流と地下水流という 2 つの要素モデルで構成されるが、これら 2 つの要素モデルをあらかじめ接続し、部分系モデルとする。全体系モデルは、これら要素モデルと部分系モデルで構成されるものであるが、全体系モデルとしては、ユーザーとの対話作業、ファイルとのデータの授受、要素モデル・部分系モデルに対する計算実行命令という 3 つの機能を持つ。

また、要素モデル間、要素モデルと部分系モデル間、及び部分系モデル間のデータのやりとりは全て端子で行われ、これら端子の基本型要素も作成されている。

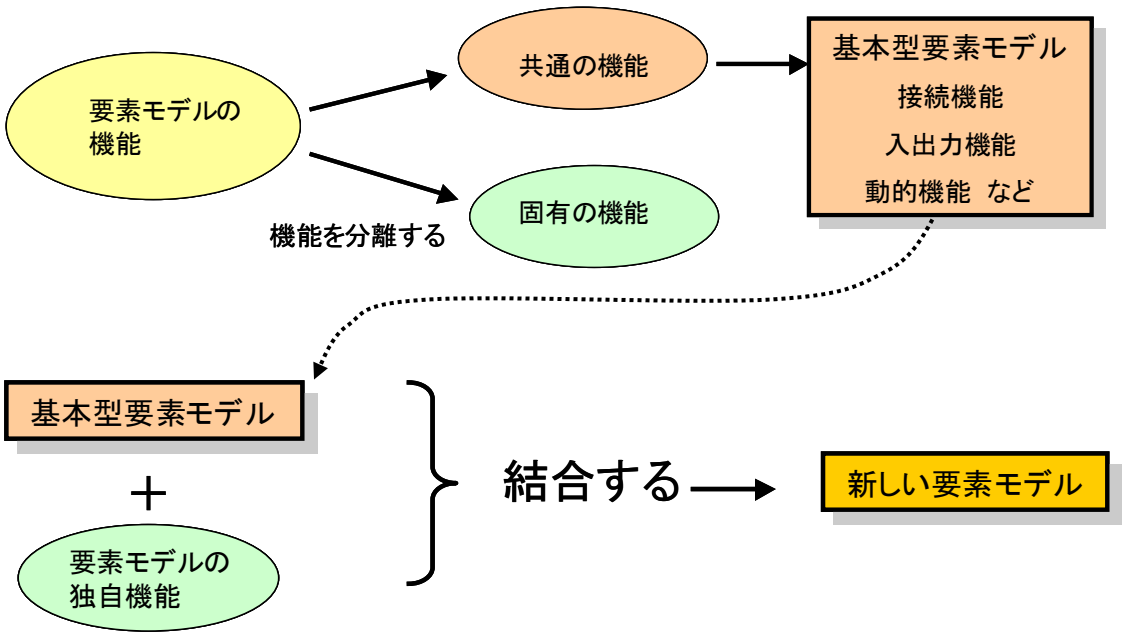


図 2-20　OHyMoS の基本構造

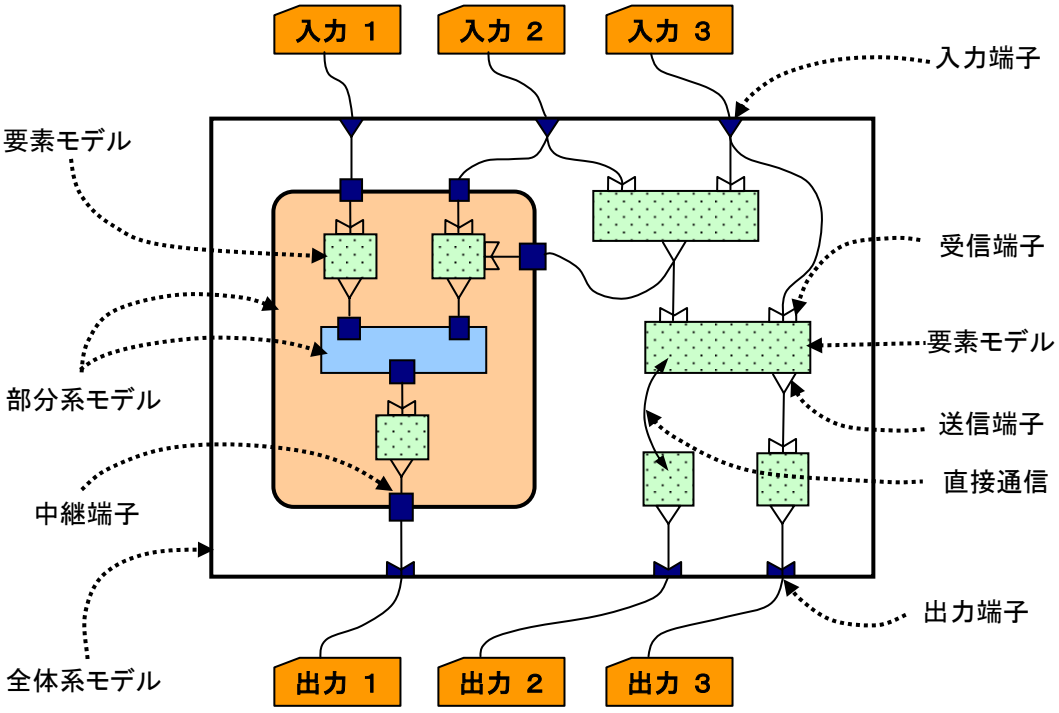


図 2-21　OHyMoS のモデル構成 [3),4)]

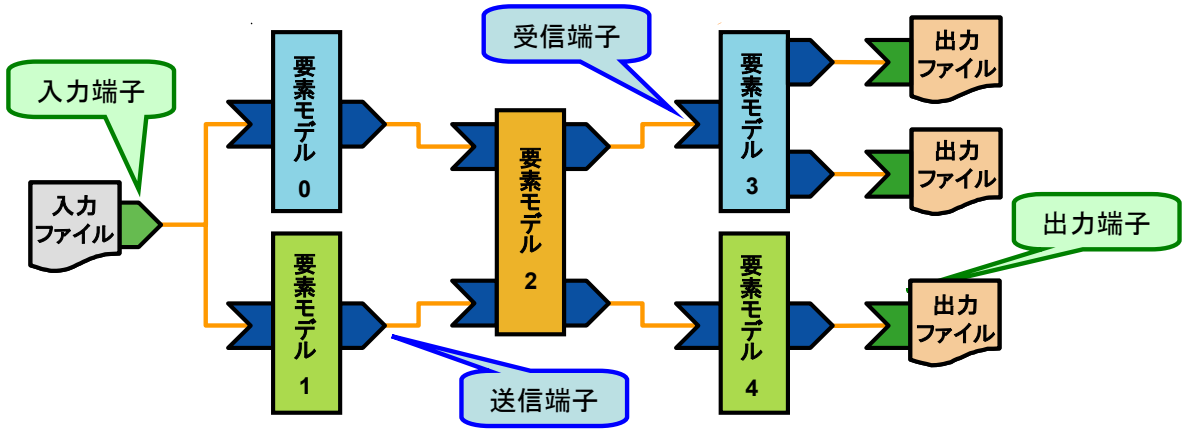
2.6.6 データの受け渡し

OHyMoS では、要素モデル間のデータの交換に『端子』という考え方を導入しており（端子のクラスが用意されている）、各要素モデルは端子を介してデータの授受を行う。端子には入力端子、出力端子、送信端子、受信端子の 4 種類があり（4 つのクラスが用意されている）、入力端子は入力ファイルに、出力端子は出力ファイルに、送信端子・受信端子は要素モデルに付属する。

各端子には固有の認識番号が割り当てられ、その認識番号を利用して端子同士のつながり、つまり要素モデルや入出力ファイルのつながりを定義する。後述の構造定義ファイルにおいて、固有の認識記号を用い、端子同士のつながりを定義することにより要素モデルの接続関係を決める。

要素モデルは受信端子からデータを受け取り、送信端子でデータを送信することになる。受信したデータを変数に代入し、ある変数の値を送信端子から送るが、その変数を決めるのは、要素モデルのプログラムでユーザーが決める。端子から送信される情報は必ずしも流量だけではなく、流速、水深といった情報も送ることができる。

図 2-22 は、3 種類の要素モデルを全部で 5 つつなげ、入力ファイルから出力ファイルに至るまでの過程を示したイメージ図である。



1. 入力ファイルの情報が要素モデル 0,1 に送信される
2. 要素モデル 0,1 が計算を行う
3. 要素モデル 0,1 の計算結果がそれぞれ要素モデル 2 に送信される
4. 要素モデル 2 が計算を行う
5. 要素モデル 2 の計算結果がそれぞれ要素モデル 3,4 に送信される
6. 要素モデル 3,4 の計算結果がそれぞれの出力ファイルに送信される

図 2-22　要素モデルと端子のイメージ

2.6.7 要素モデル

(1) 基本型要素モデル

OHyMoS では基本型要素モデルというクラスを用意している。基本型要素モデルは OHyMoS 上で稼働するための必要な機能を予め備えており（必要な関数が実装済みである）、

この基本型要素モデルというクラスを継承することにより、OHyMoS の要素モデルとしての機能を持たすことができる。概念図を図 2-23 に示す。

要素モデルの構築にあたり、ユーザーが実装しなければならないのは、1 ステップ分の計算プログラムと、送受信端子の定義、パラメータ値・初期値の設定などの関数である。一般的に数値計算モデルを構築しようとすると、計算部分とは直接関係のない入出力データに関するプログラミングや、画面表示のプログラミングなどに時間と手間がかかってしまうことが多いが、OHyMoS では前述の通り、基本的な機能は予め準備してあるので、ユーザーは計算の本質部分のプログラミングに集中することができ、モデル開発にかかる時間や手間を削減することができる。

ユーザーは実装しなければならない関数の中身を記述することで、要素モデルを開発するが、その関数名が明らかになっているため、どの関数の中身を実装すればよいかが明確になっている。例えば、1 ステップ分の計算プログラムを実装する場合、calculate()という関数の中身を実装すればよく、計算スキームを記述すべき箇所がユーザーにとって明確になっている。受信端子の定義は受け取ったデータをどの変数に代入するのかを決め、送信端子の定義は計算によって求められた値のうち、どの値を接続先の要素モデルへ送信するかを決める。送受信端子の数に制限はなく、一つの要素モデルにいくつの送受信端子を作ってもよい。パラメータ値・初期値の設定はそれぞれ、set_parameter()、set_initialstate()という関数を実装する。

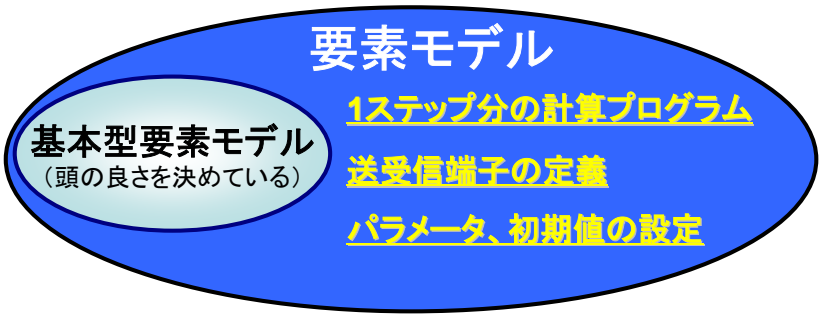


図 2-23　要素モデルと基本型要素モデルのイメージ

⑵ 要素モデルの独立性とボトムアップコントロール

OHyMoS の特徴は要素モデルの交換・追加・削除が自由に行えることにあるが、要素モデルは他の要素モデルから独立していることが求められる。つまり、A モデルを削除したら B モデルが動かないとか、C モデルを追加したら D モデルが動かないということがないような要素モデルを構築しなければならない。

OHyMoS は要素モデルの自立性というものを重要視しており、メインルーチンを置かないボトムアップコントロール型のモデリングシステムである。OHyMoS の基本コンセプトはあくまで水理水文解析の共通基盤を用意することであり、要素モデルは他のモデルに依存していないこと（他のモデルがないと計算することができないということのない）が求められる。

OHyMoS では、要素モデル自らが周りの要素モデルの計算状況を踏まえながら計算を進めていくため、ある要素モデルを交換・追加・削除したとしても、その状況に応じて計算を進める。メインルーチンを置かないことのメリットは、モデル構成を変えたとしても、プログ

ラミングの中身を変更する必要がなく、次節で後述する構造定義ファイルというシステムとは外部にあるテキストファイルを書き換えることにより対応できることにある。

手続き型言語で書かれている従来型の数値計算モデルは要素モデルをサブルーチン化しておき、メインルーチンからトップダウン的に、それらの要素モデルをサブルーチンとして計算の手順をコントロールしているため、要素となるコンポーネントを交換・追加・削除した場合、メインルーチンを更新する必要がある。一方、OHyMoS の場合、メインルーチンからの指令によって要素モデルの計算をコントロールするのではなく、それぞれの要素モデルが接続先の要素モデルとの情報のやり取りを通して、要素モデル自身が自らの計算手順をコントロールする。メインルーチンの仕事は、要素モデル間のつながりを定義することだけで、計算手順は要素モデル自身が判断し、計算を進めていく。

OHyMoS では計算するためのデータを受信している要素モデルから計算を行い、接続先の要素モデルにデータを送信する、という手順を取るため、必ずしも時間ステップごとに各要素モデルが時間的に横並びに計算を進めていくわけではない。例えば、河道網の流出解析をした場合、必然的に上流側の要素モデルから計算が進められ、最後に最下流の要素モデルの計算が行われることになる。上流からの流入量が決まらない限り、下流の要素モデルが計算できないからである。

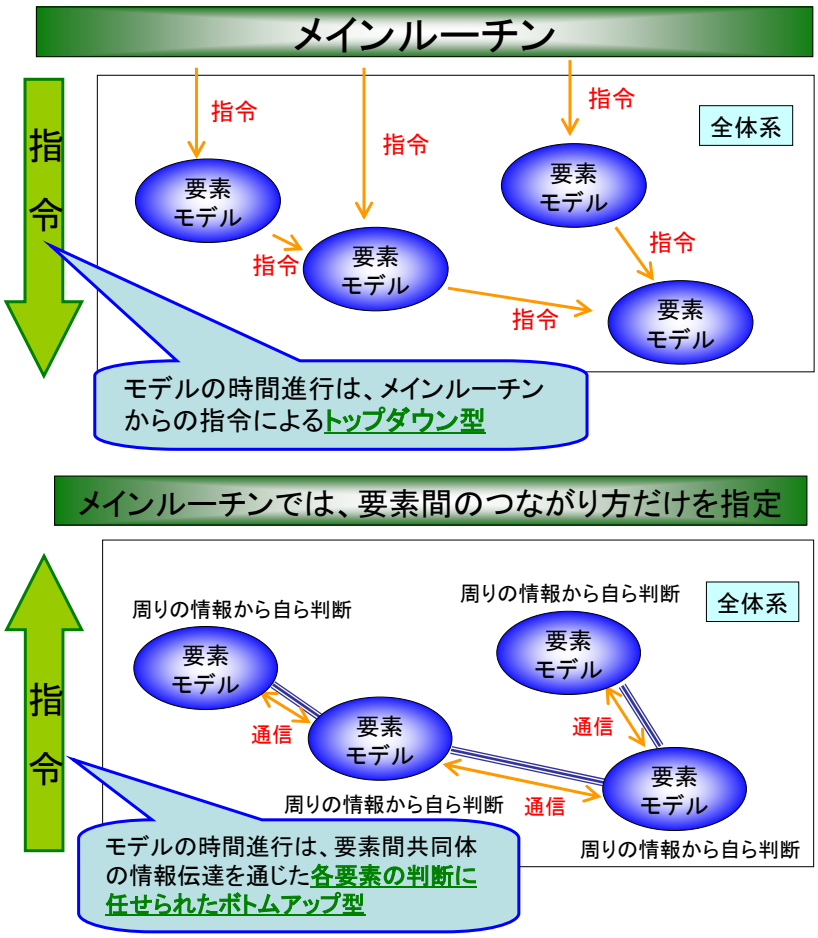


図 2-24　トップダウンコントロールとボトムアップコントロールの比較

2.6.8 構造定義ファイル

構造定義ファイル（Structure Configuration File）とは、OHyMoS バージョン 3.0 以上（OHyMoSJ はすべてのバージョンで対応）で使うことのできるモデルの構成を定義するためのテキストファイルであり、ユーザーはこの構造定義ファイルを作成することにより、解析モデルに組み込む要素モデルの選択、要素モデルの接続関係の定義、入出力ファイルの選択を行い、解析モデルの構成を決定する。従って構造定義ファイルは解析モデルの全容を決定するファイルであり、OHyMoS による解析モデル構築の核となるファイルである。

構造定義ファイルでは、入力ファイル、要素モデル、出力ファイルそれぞれに認識番号を付与して ID 化し、その ID を用いて要素モデルのつながりを定義する。

構造定義ファイルで定義する項目は以下の通りである。

- Number part
  - 入力ファイルの総数
  - 要素モデルの総数
  - 反復計算を行う要素モデルの総数
  - 反復計算を行うセットの総数
  - 出力ファイルの総数
- Input Ports
  - 入力ファイルの登録（ID を付与）
- Components
  - 要素モデルの登録（ID を付与）
- Output Ports
  - 出力ファイルの登録（ID を付与）
- Connect Ports
  - 入力ファイルと要素モデルのつながりを定義（ID を利用）
  - 要素モデル同士のつながりを定義（ID を利用）
  - 要素モデルと出力ファイルのつながりを定義（ID を利用）

構造定義ファイルはプログラミングソースのファイルではないので、ファイル作成にあたり、プログラミング言語に関する知識は必要なく、簡単な記号と数字を打ち込むだけで作成が可能である。よって、プログラミングに関する技術がなく、自ら数値解析モデルの開発を行うことができないユーザーでも、OHyMoS を用いれば数値解析を行うことができる。

また、構造定義ファイルは OHyMoS システムの外側にあるファイル（ソースではない）なので、構造定義ファイルを更新しても、再コンパイルや、システムの再構築などをすることなく、容易に様々な計算パターンを試すことができる。

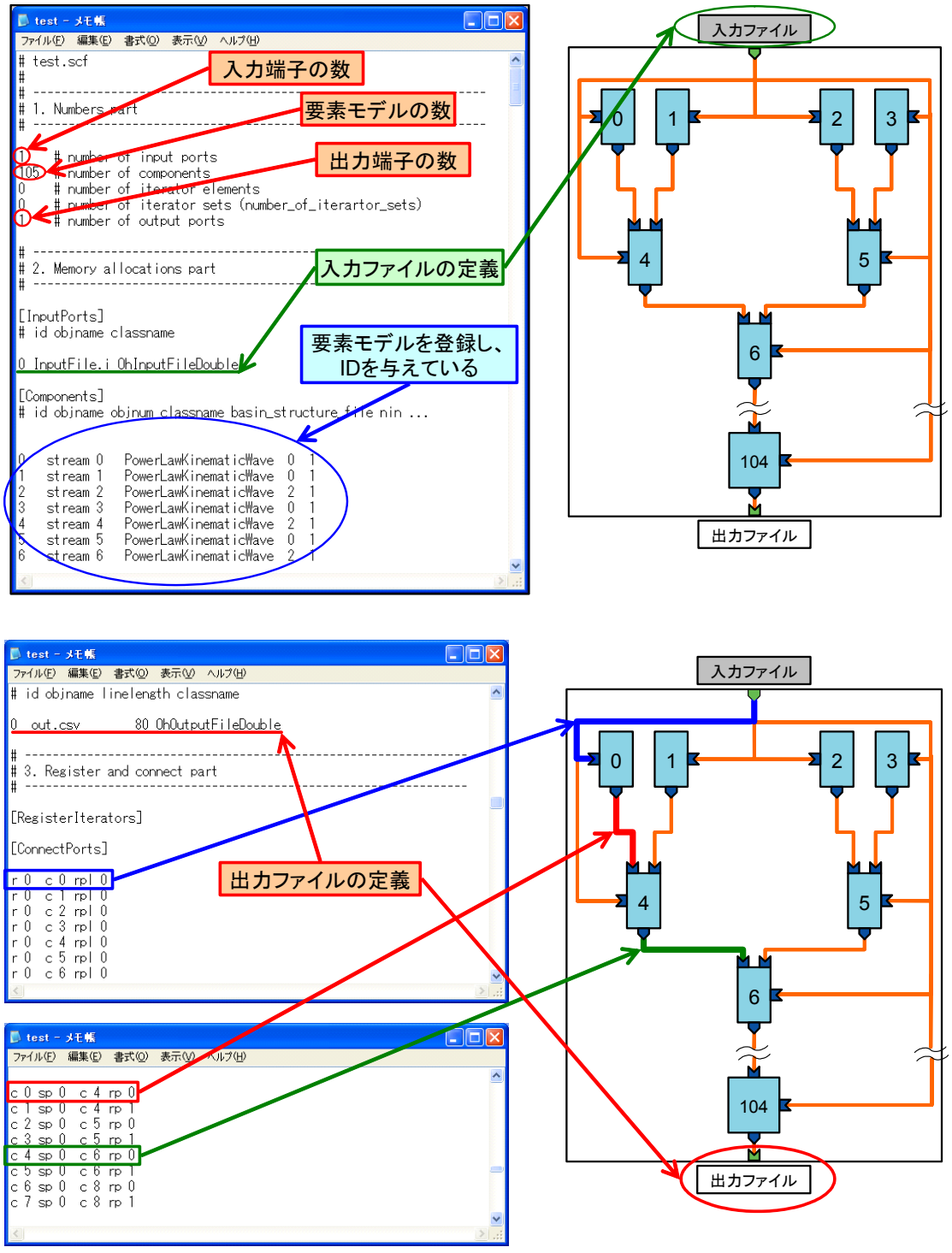


図 2-25　構造定義ファイルの概要

2.6.9 コンパイルとクラスライブラリー

OHyMoS ではソースコードをコンパイルし、クラスライブラリーを生成して解析を行うが、OHyMoS と OHyMoSJ では、コンパイル手法が異なり、クラスライブラリーの性質も異なる。ここでは、それぞれのコンパイル法とクラスライブラリーについて説明する。

(1) OHyMoS（C++版）

はじめに、OHyMoS 本体（要素モデルは含まない）のソースコードをコンパイルし、クラスライブラリーのファイル（サンプルでは ohymos.lib、以下同様）を生成する。その後、要素モデルのソースファイルを含むディレクトリで実行ファイル（ohymos.exe）を生成し、この実行ファイルをコマンドツールから実行することにより、OHyMoS を稼働させる。

(2) OHyMoSJ（Java 版）

一方、OHyMoSJ では OHyMoSJ 本体のソースコードと要素モデルのソースコードを同じディレクトリで管理し、OHyMoSJ の本体と要素モデルのソースファイルを一度にコンパイルし、クラスライブラリー化を行う（ohymosj.jar ファイルを生成）。OHyMoSJ のクラスライブラリーは jar ファイルと呼ばれ、中間ファイルであるクラスファイル（.class）を圧縮したファイルであり、OHyMoSJ では jar ファイルをクラスライブラリーと考える。OHyMoSJ 実行の際はこの jar ファイルを呼び出すことによって解析を行う。OHyMoSJ の実行は、OHyMoS 同様、コマンドツールを用いる。

新規要素モデルを追加しようとしたとき、Java のソースファイルを同ディレクトリの中に置き、再コンパイルを行うと、新規要素モデルを含んだクラスライブラリーを生成することができる。この際、新規要素モデルソースファイル以外のものに関してはソースファイルである Java ファイルは必要ない。必要なのは class ファイルと呼ばれる抽象クラスのファイルである。class ファイルはコードを見ることができないファイルであるため、要素モデルを公開したとしてもプログラムソースを公開せずにモデルだけ公開することができる。コンパイルし、ライブラリーを生成することに関しては、OHyMoSJ の容易に扱うことができると考えられる。

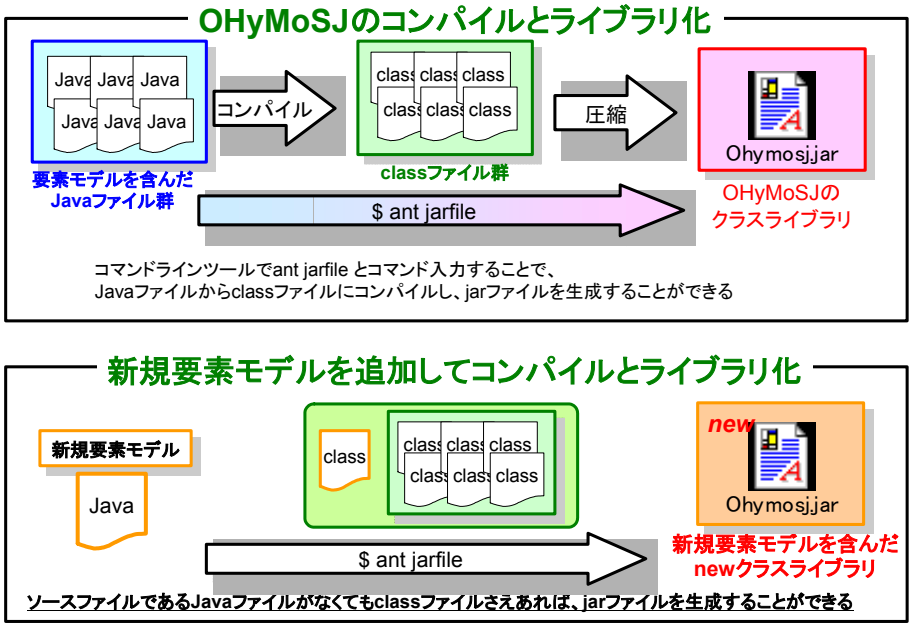


図 2-26　OHyMoSJ のコンパイル

2.6.10 ユーザーによるモデルの作成方法 [3)]

OHyMoS では、基本型要素モデルのクラスを継承すれば、以下のような方法で要素モデルが作成できる。

① 基本型要素モデルを継承する。
② 受信端子・送信端子を定義する。
③ 受信端子・送信端子を登録する関数を定義する。
④ パラメータ・状態量を定義する。
⑤ パラメータを設定する関数、状態量を初期化する関数を定義する。
⑥ 初期の送信を行う関数を定義する。
⑦ 次の計算のためのタイムステップを算出する関数を定義する。
⑧ 1 ステップ分の計算を実行するか判断する関数を定義する。
⑨ 1 ステップ分の計算・送信を行う関数を定義する。
⑩ 初期化後の一連の計算作業を行う関数を定義する。

2.6.11 まとめ

OHyMoS の特徴をまとめると、以下のようである。

- 要素モデルを組み合わせて全体系モデルを構成するようになっている。
- 水文系モデルの要素モデルとして普通に要求されるような動作、パラメータの設定、初期値の設定、計算時間の更新などについては標準化されている。データの入出力機能は、新規にユーザーが定義するデータ型に対しても対応できるようになっている。
- 要素モデルは、個々に独自の時間単位で計算を進めることができる。
- ある要素モデルの出力データを他の要素モデルが入力データとして利用するという関係は、端子によるデータ授受の関係としてモデル化されている。
- 端子によるデータ授受では対応できない場合に対応するため、要素モデル同士が直接データを交換する直接通信の機能がある。
- 流出系の最終状態をファイルに書き出し、後で計算を再開することができる。

2.7 標準フレームワークの開発に向けて

2.7.1 標準フレームワークの目指すもの

以下に、標準フレームワークの開発から究極的な目標到達までの道程のイメージを示す。このような道筋については様々な議論があり、これに拘泥するものではないが、「何を目指して」という部分はとても重要であり、その検討を本格的に行うためにも，こうしたイメージを出してみることが大事と考えられる。

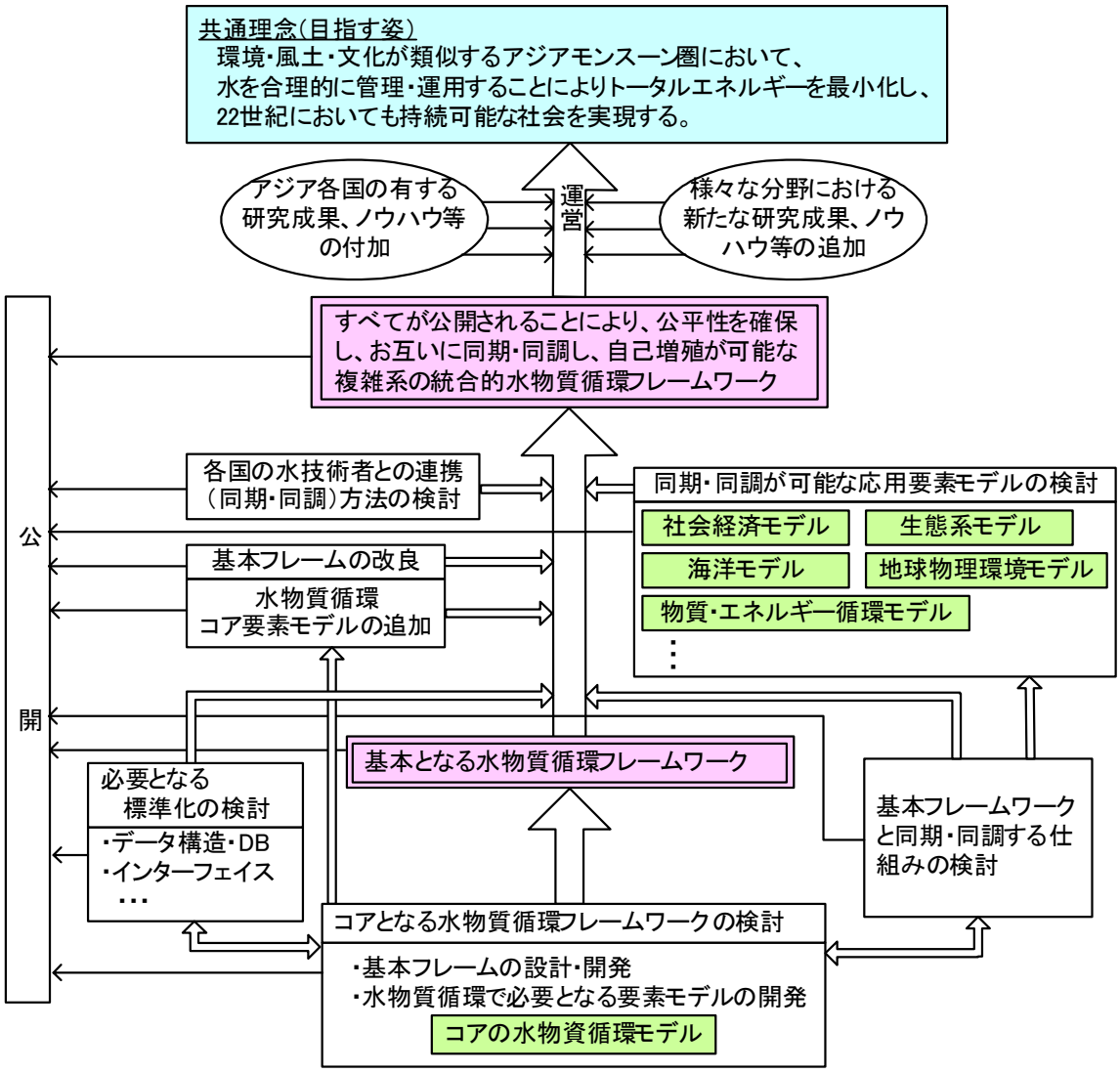


図 2-27　理想形までの道程のイメージ

2.7.2 標準フレームワークと既存シミュレーションモデルとの関係

ここでは、標準フレームワークと既存のシミュレーションモデルとの関係について記す。

(1) 標準フレームワークと既存シミュレーションモデルの共存

標準フレームワークはシミュレーションモデルの共通基盤であり、いわばコンピュータのOSのような位置づけである。標準フレームワークを作成したとしても、既存のシミュレーションモデルやプログラムの存在を否定するものではなく、共存していける関係にある。図 2-28に示すように、標準フレームワークと既存モデルの機能はお互いに参照・フィードバックしあうことにより、より良いものへと共存共栄することができる。

また、大学などで作成されるモデルに関しては、標準フレームワークの流れに乗らないものも想定されるが、そういったモデルの存在を否定するものではない。

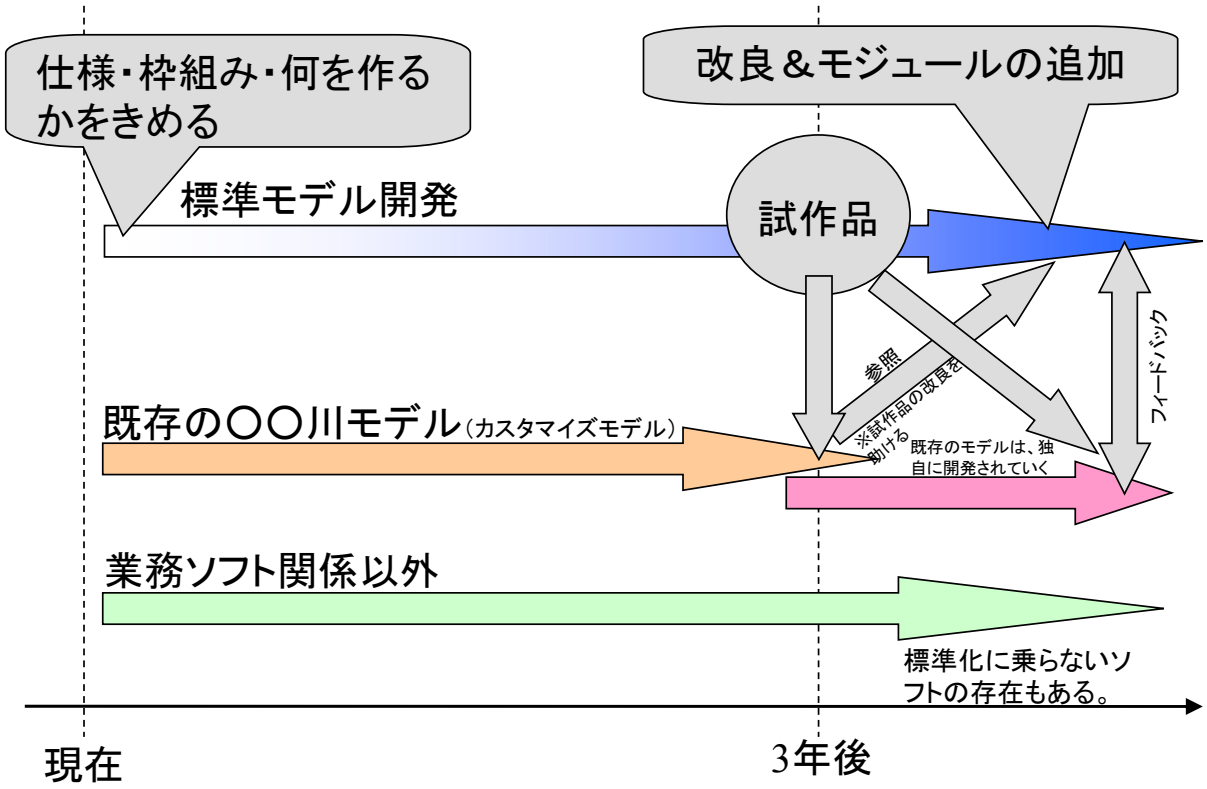


図 2-28　標準フレームワークと既存モデルの関係

(2) 既存モデルの標準フレームワークへの取り込み

標準フレームワークはシミュレーションモデルの共通基盤であり、シミュレーションモデルそのものではない。従って、開発を目指す標準フレームワークでは、これまで作成してきた解析プログラムも利用できるものとすることが資源の活用という点から重要な点と考えられる。

これまで多くの水理・水文・水質解析プログラムは Fortran で作成されており、これをそのままの形でフレームワーク上に乗せることはできない。フレームワークに対応した言語で作成するかもしくは、ラッピングと呼ばれる技術を導入するかのどちらかと考えられる。

ラッピングに関する技術的な情報は本報では詳細には論じないが、ラッピングとはある言語で書かれたプログラムを他の言語で書かれたプログラムと連動させるための技術である。つまり、ソースコードに触れずに実現できる技術ではなく、ラッピングと呼ばれる処理を実装する必要があり、相応のプログラミング力が要求されると考えられ、容易な作業ではないと想定されるが、新たに再プログラミングするよりは、手間がかからないと言われている。しかし、ラッピングは万能ではなく、相性の悪い言語も存在すると言われている。

2.7.3 標準フレームワークの開発言語

ここでは、標準フレームワークの開発言語について検討する。これまでも述べてきたように標準フレームワークは水理・水文・水質解析の共通基盤となるフレームワークであり、完成された解析ソフトウェアを開発するわけではない。また、様々な技術レベルのユーザーによる利用を想定したものであり、フレームワークに自ら要素モデルを加えるなどして利用するユーザーもいるだろう。従って、フレームワークを開発する言語について、その特徴について調査検討しておく必要があり、それぞれ比較しておく必要がある。

2.5 で説明した通り、オブジェクト指向型言語の特徴の中で象徴的なクラスという概念は機能の追加の容易性という利点を持っている。これは標準フレームワークの実現に不可欠な要素である。その理由について以下に記す。

(a)様々な機能を分割して作成

標準フレームワークには解析を実行する以外にも様々な機能が必要となる。例えば、データを入出力する機能、グラフを描画する機能等であるが、これらをクラスの単位で分割して作成しておくことにより、バージョンアップの際に機能を向上させるクラスだけを書き直せばよい。また、Windows で利用するとした場合、既に公開されているクラスをそのまま使えばいいし、データベースと連携するためのクラスなどもあるため、データベースエンジンごとに対応するクラスを持っておけば、様々なデータベースを利用することが可能となる。

(b)要素モデルの作成が容易

標準フレームワーク上で稼働する要素モデルを作成するには、標準フレームワークの仕様に則ってプログラミングする必要がある。つまり要素モデルを実装する場合、最低限必ず実装しなければいけない機能がある。こういったプログラム作成にある程度の縛りがある場合は、継承の機能が有効利用できる。予め要素モデルの親クラスとなるクラスに最低限必ず実装しなければいけない機能を実装しておけば、子クラスは何もしなくてもその機能を持つことができるため、ユーザーは自分が追加する機能、すなわち計算部分だけを実装すればよいと言うことになり、要素モデル実装の生産性が格段に向上する。

この考え方は OHyMoS・OHyMoSJ ですでに取り入れられており、基本型要素モデルという親クラスが実装されてある。

⑵ 候補となる言語の比較

ここでは、標準フレームワークの開発言語の候補となるプログラミング言語についてそれぞれの特徴を挙げて比較する。

比較する言語は前述の検討を踏まえオブジェクト指向型言語に絞った。候補として検討する言語は、C++、C#、Java の３種類とする。この３言語は様々なシーンで使われている言語で、メジャーなオブジェクト指向型言語のトップ３といってよい。ユーザー自身も実装することができる標準フレームワークの言語としては極力メジャーなものを利用する方が望ましいことは論を持たない。比較結果を表 2-2に示す。

表 2-2　言語の比較

比較項目	.NET Framework（C#）	C++	Java
概要	.NETはMicrosoft社が開発したアプリケーション開発環境。プログラミング言語ではない。 C#は.NET対応のプログラミング言語。	C言語をオブジェクト指向型言語として拡張した言語。近年最もよく使われている商用言語の一つといわれている。	Sun Microsystems社が開発したプログラミング言語であり、開発環境でもある。
開発・管理体制	Microsoftの商品であり、1社で開発・管理を行っている。	様々なベンダーがコンパイラや、開発環境のGUIを開発している。最も普及しているのはMicrosoft社の「Visual C++」	開発はSun Microsystems社であるが、その後は様々なユーザによって言語が成長していった。
特定の組織の影響力	Microsoft社製の商品であるため、Microsoft社の影響を直接受ける。Microsoft社が開発を辞めたら、そこで終了する。	C#ほどではないが、最も普及しているコンパイラがMicrosoft社のVisual C++であるため、Microsoft社の影響を受けるかもしれない。	Sun Microsystems社が開発したが、言語に対して影響力は少ない。既にSunの手を離れてコミュニティによって開発されており、突然なくならない。
対応OS	基本的にWindows対応だが、今後Windows外でも利用できるようになる可能性もある。WindowsOSが持っている機能を活用しやすい。	どの環境でも動くが、プログラムを動かすOSごとに合わせたプログラミングが必要。例えば、Linuxで開発したプログラムがWindowsで走らないことがある。	すべてのOSで利用できる。特定のOSで特にパフォーマンスを発揮するということはない。
クラスに関して	Windows上で利用できるGUIなどが初めから備えられており、.NET対応言語であれば共通して利用することができる。	異なる環境（OSやコンパイラ）でコンパイルされたクラスを共通に利用することはできないことがある。	世界中で作成された多種多様のクラスが存在する。無償で手に入るものが多い。
公開されているクラスの品質	Microsoftが管理しているものに関しては、品質は保証されていると考えて良い。	公開されているクラスライブラリが少ないと言われている。また、ヘッダファイルはソース提供しなければならない。	個人作成のものに関しては品質保証されているとは限らない。
過去の資産の活用	過去の資産をそのまま使うことはできない。 ラッピング、再プログラミングが必要。	C++で書かれているものであっても、コンパイラなどが違うと、そのまま利用できない可能性がある。	過去の資産をそのまま使うことはできない。 ラッピング、再プログラミングが必要。
文法の分かりやすさ	機能が多く、やや煩雑と言われる。	C言語が解る者には分かりやすいと言われるが、全般的に複雑というのが一般的。	シンプルな文法構造になっており、理解しやすいと言われている。
プログラミング	機能が多く、複雑な処理でも比較的簡易に実装できる。	他言語と比べて、プログラミングが複雑といわれ、実装に時間がかかるといわれる。	文法がシンプルな分、複雑な処理には回りくどいプログラミングが必要。
実行環境	.NET framework上で、Windowsで実行できる。	開発時の環境以外では、そのまま実行できないことがある。	Java Virtual Machine上で、OSに依らず実行できる。
DBとの親和性	DBとの親和性は高いと言われている。特にSQL ServerはMicrosoft社の製品であり、親和性は高い。	Microsoft社のコンパイラとDBであれば、親和性は高い。	データベースにアクセスできるAPIを実装することができる（JDBC）
対応言語に関して	.NET対応言語は幾つかある。 ・C#、C++.NET、VB.NET、など。 C++とC++.NET、VBとVB.NETは異なる言語	C++そのものが、言語である。	Javaそのものが、言語である。
水理解析に用いられている頻度	現在はほとんど使われていないと考えられるが、C++、VBからの移行がスムーズになれば増加すると考えられる	C#、Javaよりは利用されていると考えられる。C言語ユーザがC++に移行して利用している可能性がある。	現在はあまり使われていないと考えられるが、今後は徐々に増えていくと考えられる

2.7.4 標準フレームワークの仕様の検討

(1) 要素モデルに関する仕様の検討

1) 要素モデルは独立したソースファイルで作成する

要素モデルは独立したソースファイル（クラス）で作成することが求められる。ユーザーが要素モデルを新規作成する場合、一つのファイルを作成すればよい。

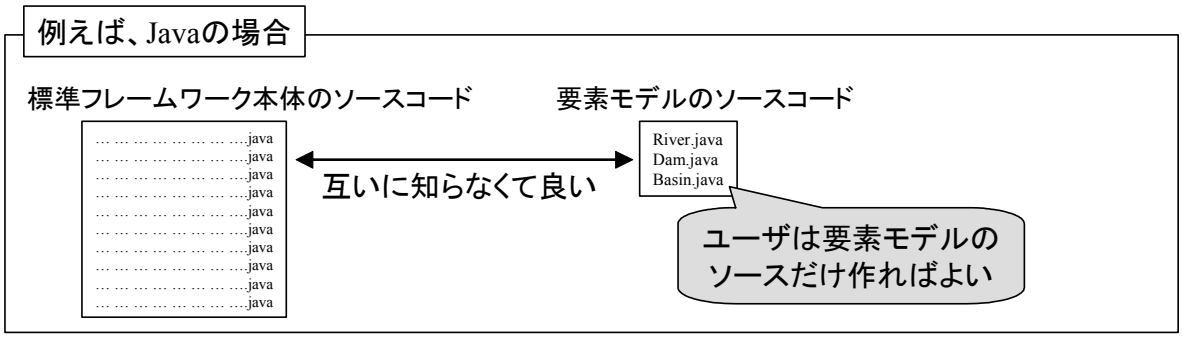


図 2-29　要素モデルは独立したファイルとするイメージ

2) 要素モデルは機能的に独立している

要素モデルは、他の要素モデルから独立していなければならない。接続する相手の要素モデルの中身を知らなくても接続できなければならない。

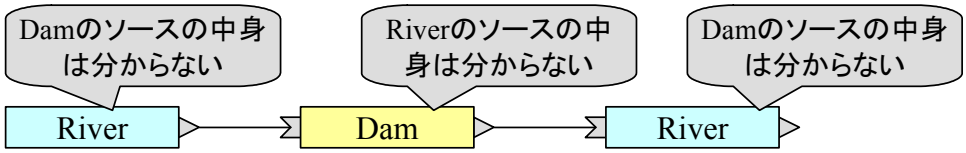


図 2-30　他のモデルの中身を知らなくて良いイメージ

3) 入力すべきデータ、出力するデータを明らかにする

要素モデルに入力すべきデータ項目、要素モデルが出力するデータ項目を明らかにしなければ、他の要素モデルと接続することができない。

4) パラメータ値はモデル外部から設定可能とする

パラメータ値はソースコード中で設定せずに、ソース外から設定できるようにする。内部で設定すると、条件を変えて解析を行うたびにソースコードを書き換える必要があるためである。

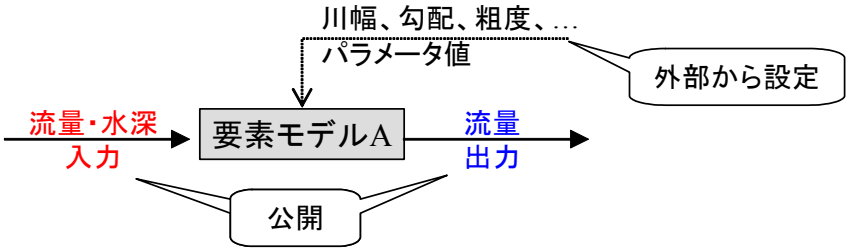


図 2-31　要素モデルの入出力データ項目の公開

5) 継承と多態性を持たせる

要素モデルのクラスを継承することにより、新たな要素モデルを作成できる。ある要素モデルをカスタマイズする場合、追加する機能だけを実装すれば、カスタマイズされた新たな要素モデルを容易に作成することができる。

⑵ 全体モデル構築に関する仕様の検討

1) 要素モデルをつなぎ合わせて全体モデルを構築できる

標準フレームワークの仕様に基づく要素モデルをつなぎ合わせることにより、全体モデルを構築できるものとする。

2) 要素モデルの交換・追加・削除ができる

要素モデルの組み合わせはユーザーが自由に設定できるものとし、要素モデルの交換・追加・削除も容易に可能なものとする。

3) 異なるΔtの要素モデルを接続することができる

異なるΔtの要素モデルを接続することができれば、要素モデル作成時にその要素モデルに最適なΔtを設定することができる。

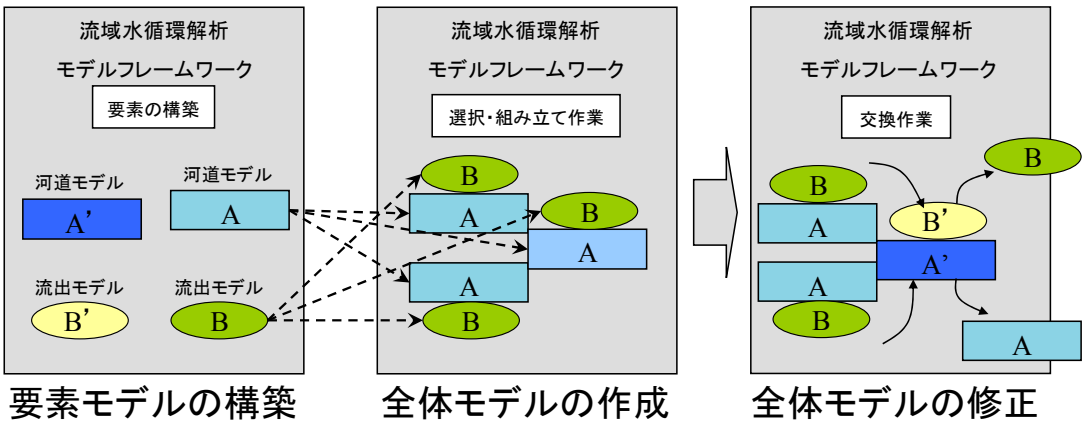


図 2-32　全体モデル構築のイメージ（図 2-12 の再掲）[3)]

⑶ データ交換端子に関する仕様の検討

1) 要素間データの授受、モデルへのデータの入出力は端子で行う

解析における要素モデル間データの授受は端子で行うものとする。解析に用いる入力データ（水理解析でいう外力：雨量、流量など）と、モデルから出力する出力データも端子で行う。

2) １次元、２次元、３次元の時系列データを取り扱える

水理・水文・水質に関する複雑な現象の解析にも対応する必要があり、多次元の時空間データを取り扱えるだけでなく、異なる次元のモデル同士の接続をも可能とする。また、水理・水文・水質だけでなく環境、生態系、経済などの河川、流域管理のための政策支援に資するための複合的な解析にも今後対応していく必要がある。

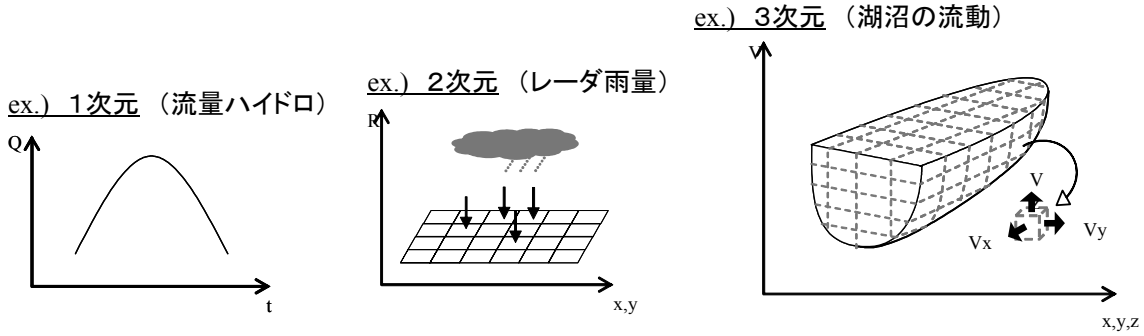


図 2-33　各次元データのイメージ

3) 単位系を規定する

端子を通じて入出力、送受信するデータの単位を規定する必要がある。要素モデルの中身を知ることなく利用できるという標準フレームワークのコンセプトを守るためには単位の統一は不可欠である。

⑷ 計算の進め方に関する仕様の検討

1) 各要素モデルが時間的に平行に進む

接続関係にある要素モデル同士が毎ステップごとにデータの授受を行いながら計算を進めていく。

2) 要素間の反復計算を行うことができる

１次元河道モデルと２次元氾濫解析モデルを相互接続するような水理解析においては、複数の要素の間で情報を交換しながら計算を反復する機能が必須である。

3) 計算の中断再開ができる

計算を途中で中断しても、中断した時点から再開できる機能を持たせることにより、長時間の解析が必要な場合でも、一時中断することができる。

(5) ユーザーインターフェースに関する仕様の検討

1) 様々なレベルのユーザーでも利用できる

水理解析に堪能なユーザーから、水理解析の知識を有さないユーザーまでストレスなく利用できるユーザーインターフェースを作成する必要があり、グラフィカルユーザーインターフェース（以下、GUI と表記）を開発する必要がある。

2) 要素モデルの接続が GUI 上で操作できる

要素モデルの接続を GUI 上の操作で設定できる必要がある。要素モデルの交換・追加・削除を容易に行えるものとする。

3) 入力データの選択ができる

入力データの設定やどの要素にデータを入力するかの設定について GUI 上の操作で行うことができる。

4) 結果をグラフ化できる

ハイドロなどの出力結果をグラフによる視覚化を GUI 上で行うことができる。

5) プロジェクト管理ができる

要素モデルを接続し、全体モデルを構築した状態や、パラメータの設定などの状態を記録できるプロジェクトファイルが作成できるものとする。また、プロジェクトファイルは異なる PC 間で互換性のあるものとする。

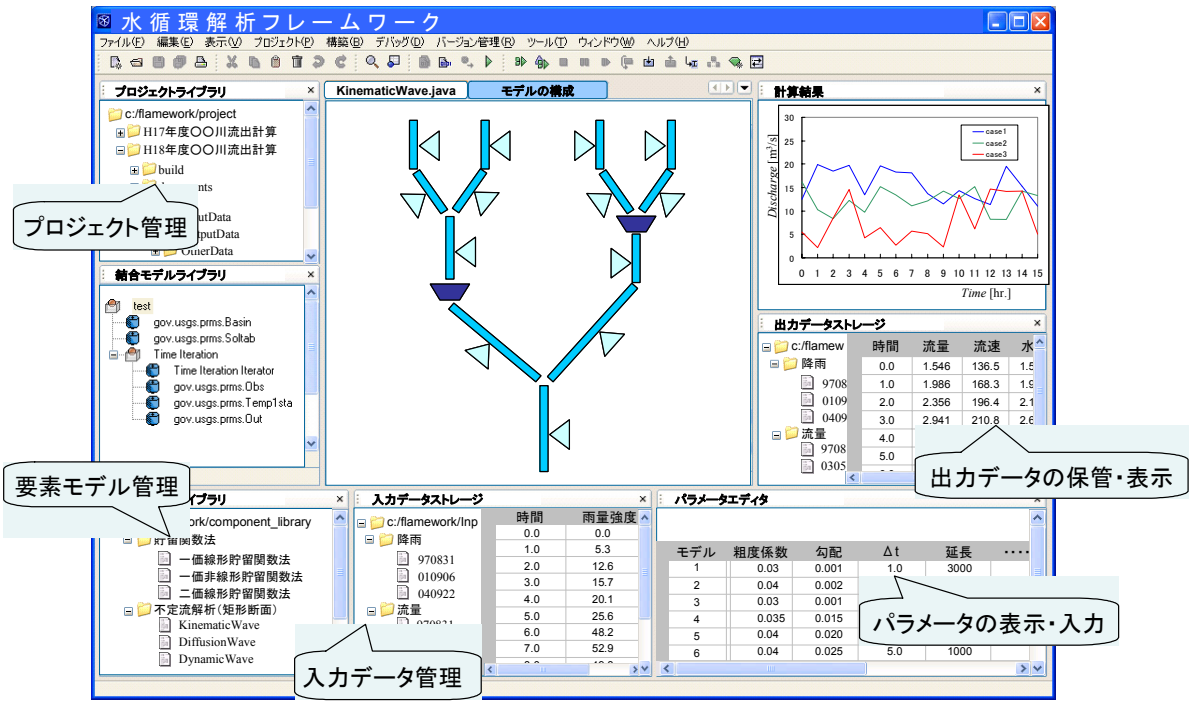


図 2-34　GUI のイメージ

(6) その他の機能に関する検討

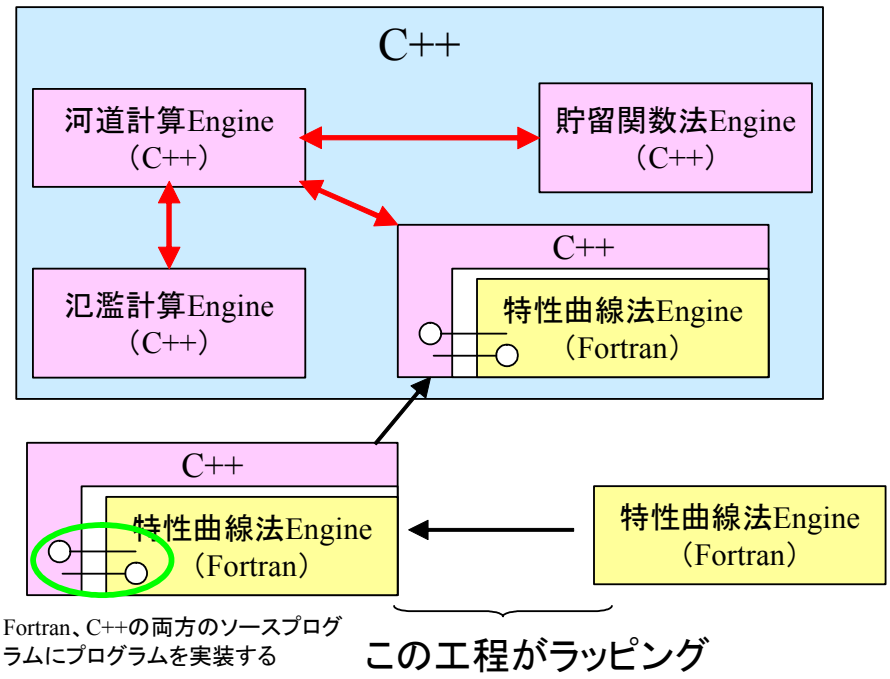
1) 汎用型シミュレーションモデルのセットアップが容易にできる

汎用型シミュレーションモデルの PC へのインストールを容易にできる必要がある。また、詳細な PC の設定をすることなく解析を実行できることが望ましい。

2) 既存のプログラムを利用することができる

これまで作成されてきた解析モデルのプログラムを標準フレームワーク上で利用できる必要がある。ただし、標準フレームワーク上で動くように加工する必要があり、その一つの手法としてラッピングがある。

以下にラッピングのイメージを示す。



（Fortran プログラムを C++でラッピング）

図 2-35　ラッピングのイメージ

3) データベースと連携できる

標準フレームワークからデータベースにアクセスできるものとする。その操作は GUI 上でできるものとする。