# I－19 オブジェクト指向プログラミングによる開水路網のモデル化と不定流の再帰的計算

# Recursive Modeling and Unsteady Flow Computation of Open Channel Networks with Object Oriented Programming

池田裕一[1]

Hirokazu Ikeda

開水路網における不定流を陰解法で計算するシステムを、Java言語によるオブジェクト指向プログラミングで構築した。その中核は、開水路計算を行う「開水路オブジェクト」と、その上下流端や分合流点の処理を行う「節点オブジェクト」、それらを組み合わせた「開水路網オブジェクト」である。「開水路網オブジェクト」の中にまた別の「開水路網オブジェクト」をもつ再帰的構造をとることも可能で、オブジェクト複合体としての再利用性も高い。そしてオブジェクトの再帰的構造を巡りながら、計算や出力などの処理が進められるようにした。これによって、プログラムの可読性や再利用性が向上し、開水路網計算の効率的プログラミングが可能となった。

Computation system of unsteady flow in open channel network with implicit method was established with Object Oriented Programming in Java language. The system has some objects; "River Channel Object" calculates the unsteady flow, "River Node Object" represents the boundary condition (upstream end, confluence, and so on), and "River Object" contains these two kinds of objects. "River Object" can also contain another "River Object", that is, the open channel network can have recursive structure. This means high re-usability of objects, which was created formerly. Navigating the recursive structure, the system computes the "River Channel" flow and operates some output routines. The program code is easy to understand and reuse, then, effective programming of open channel network was enabled.

**キーワード**： 開水路網、不定流、再帰的構造、オブジェクト指向プログラミング、デザインパターン

***Keywords*** ： open channel network, unsteady flow, recursive structure, object oriented programming, design pattern

## 1．はじめに

一昔前までは、土木工学分野で計算機の利用というと、数値計算が唯一の用途であった。ところが最近の情報関連分野の発展と社会需要の多様化により、コンピュータの用途は数値計算はもとより、表計算、データベース、Webアプリケーションなど、ずいぶんと多彩になってきた。

そのような情勢の中で、Java は注目すべきコンピュータ言語のひとつである。インターネットで結ばれた大規模な分散処理系の構築から、パソコン単位のアプリケーション、そして携帯端末での小規模アプリケーションまで、多様なプラットフォームでの多彩なアプリケーションの構築をサポートし得る。そして、最近になって広く普及してきたオブジェクト指向プログラミング(OOP)もサポートしている。

Fortran など従来の言語は構造化(あるいは機能詳細化)プログラミングと呼ばれる。これはプログラムに要求されている「機能」全体をまず大まかに分解し(たとえば入力，計算，出力など)，それをさらに細かく分解していく手法である．これに対して，OOP は，プログラムを動作させたい世界がどのような「もの(オブジェクト)」で構成されるか検討していく手法で，人間の認識により近いアプローチで，複雑なシステムを扱うことができる．また、プログラムコードの再利用が容易ともいわれている。

ここで取り上げる1次元開水路不定流は、支配方程式自体はさほど複雑なものではない。ただし、開水路全体を計算していくなかで、上下流端の(外部)境界条件はもとより、堰やゲート、逆サイフォン、横越流など、さまざまな内部境界条件[1)]を処理しなければならない。また、河川の合流や分流も扱って河川網を計算するなどと考えると、システム全体はかなり複雑なものになる。

著者は前報[2)]において、さまざまな内部境界条件を有する1次元開水路不定流を陽解法で計算する場合について、OOP を適用し、プログラムの可読性とプログラムコードの再利用性が向上することを示した。

そこで本研究では、次の段階として、開水路の分流や合流を有する開水路網の不定流計算にOOPを適用し、Java

1：正会員 博士(工学) 宇都宮大学大学院 助教授 工学研究科エネルギー環境科学専攻
(〒321-8585 栃木県宇都宮市陽東7－1－2、Tel :028-689-6215, E-mail : ikeda@cc.utsunomiya-u.ac.jp)

言語による実装を試みた。その際に、プログラミングの効率を向上させるために、以前に作成した開水路網のオブジェクトを再利用して、もっと大きな開水路網を構築できるような仕組みがあれば便利である。いわば開水路網の入れ子状態である。こうした再帰的構造の構築と処理の方法については、OOPにおけるデザインパターンを活用することにした。

## 2．問題の定式化と計算方法

### （1） 支配方程式

1次元開水路における不定流の連続式および運動方程式は次の2式である.

$$\frac{\partial A}{\partial t}+\frac{\partial Q}{\partial x}=0 \qquad (1)$$

$$\frac{\partial Q}{\partial t}+\frac{\partial vQ}{\partial x}+gA\frac{\partial z_s}{\partial x}+gA\frac{n^2|v|v}{R^{4/3}}=0 \qquad (2)$$

ここに, $x$:流下方向座標, $t$:時間, $A$:断面積, $Q$:流量, $v$:流速, $z_s$:水位, $R$:径深, $n$:粗度係数である。

これらの方程式の数値計算方法は枚挙に暇がないが、ここでは参考文献3)に説明されているスタッガード格子上の陽解法を参考にしながら、陰解法での計算方法を組み立てることにした。すなわち、文献1)では対象とする変数の時間微分項を現在時刻の値を用いて計算するのに対して、今回の陰解法では、時間刻み $\Delta t$ 後の値を用いて計算することにした。この方法では、連立方程式を解いたり収束計算を行ったりなど計算に時間はかかるが、計算の安定性が良好であるなどの利点もある。

### （2） 境界条件

本研究では、境界条件としては、上流端と下流端そして分合流点の3種類を考えることにする。

開水路の流端では流量を次式のように与える[4]。

$$Q_0(t)=Q_b+(Q_p-Q_b)\left\{\frac{t}{t_p}\exp\left(1-\frac{t}{t_p}\right)\right\}^C \qquad \cdots(4)$$

ここに, $Q_b$ :基底流量, $Q_p$ :ピーク流量, $t_p$ :ピーク時刻、$C$ :洪水波形の形状を示す定数である.

下流端では自然境界条件、すなわち式(2)の移流項(左辺第1・2項)をゼロとおいて、水面勾配から流量を求めることにする。これは簡単な差分式で計算可能である。ただし、プログラムコードでは、他の方法(たとえば水の時系列を直接与えるなど)による設定方法にも容易に切り替えられるよう工夫することにする。

分合流点では、接続された水路での水位がともに等しく、また合流する流量の和と分流側のそれが等しくなるようにする。これを数値計算で実現するために、次のようにした。まず、分流側開水路の上流端について、連続式(1)の差分方程式を導くと、

$$h_{k0}=h_{k0}^0+\frac{\Delta t}{B_k\Delta x_k}(Q_{k1}-Q_{k0}) \qquad (3)$$

となる。ここにhは水深、Bはhに対応する水面幅である。下添字kは分流する水路の番号、下添字0および1は上流端およびそこから距離刻み $\Delta x$ 下流での値を意味する。上添字0が現在時刻での値、上添字なしは $\Delta t$ 後の値を意味する。もし境界条件が満たされていれば、$h_{k0}$ はkによらず同一の値 $h_0$ になるはずである。このときk番目の分流水路への分流流量は、式(3)より

$$Q_{k0}=Q_{k1}+\frac{B_k\Delta x_k}{\Delta t}(h_0-h_{k0}^0) \qquad (4)$$

のように得られる。このすべての値を合計したものが、合流してくる流量の和の等しいことから、結局

$$h_0=h_0^0+\frac{\sum_j Q_j-\sum_k Q_{k1}}{\sum_k B_k\Delta x_k}\Delta t \qquad (5)$$

となる(ここで下添字jは合流する水路の番号)。つまり、式(5)により分合流点の水深の近似値を求め、それを式(4)に代入して分流流量の配分を決める。その流量と式(3)を用いて水路ごとの分合流点の水深が求められるが、その差が許容誤差範囲内となるまで、以上の計算を繰り返すものとする。

### （3） 初期条件

初期条件の設定にもさまざまな方法がある。ここでは最も単純なもののひとつとして、流量は水路の上流端で与える初期流量に、水深はそれに対応する等流水深にする。ただし、プログラムコードでは、他の方法による設定方法にも容易に切り替えられるよう工夫することにする。

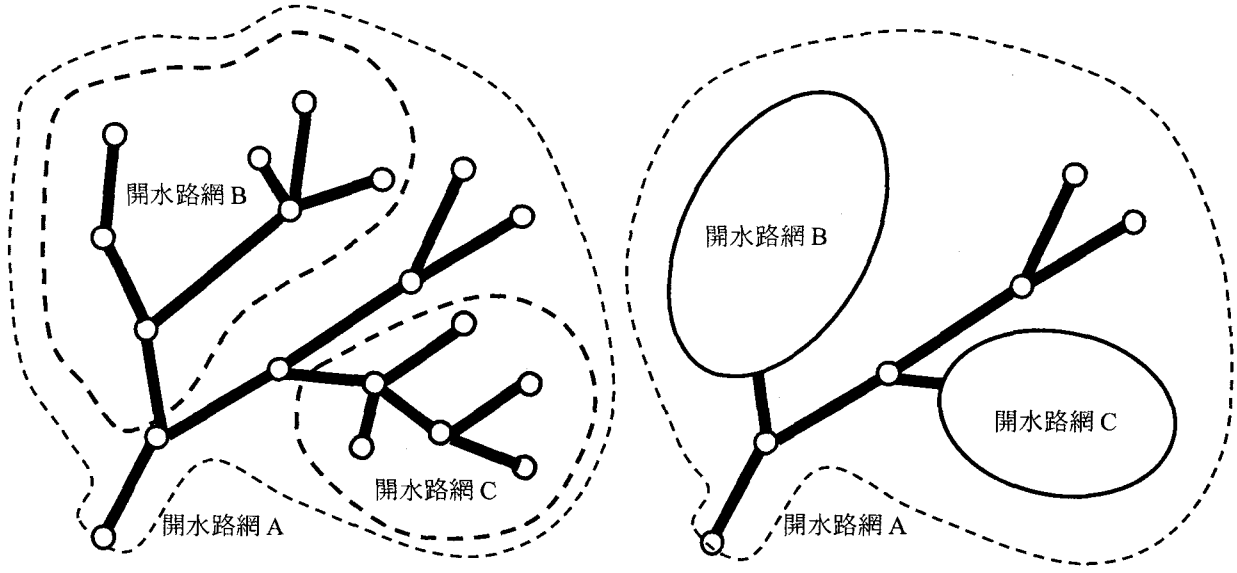
## 3．開水路網の再帰的処理

図－1は開水路網の構成の仕方の例を示したものである。図中, 太い実線はひとつの開水路を, 白丸は種々の境界点(下流端, 上流端, 分合流点など)を表している。

単純に考えれば, この2種の要素によって開水路網を構成してやればよい。(**図—1**(a))。計算処理においては, 個々の開水路や境界点を順にめぐっていけばよい。それぞれの計算処理は前章に述べたとおりである。

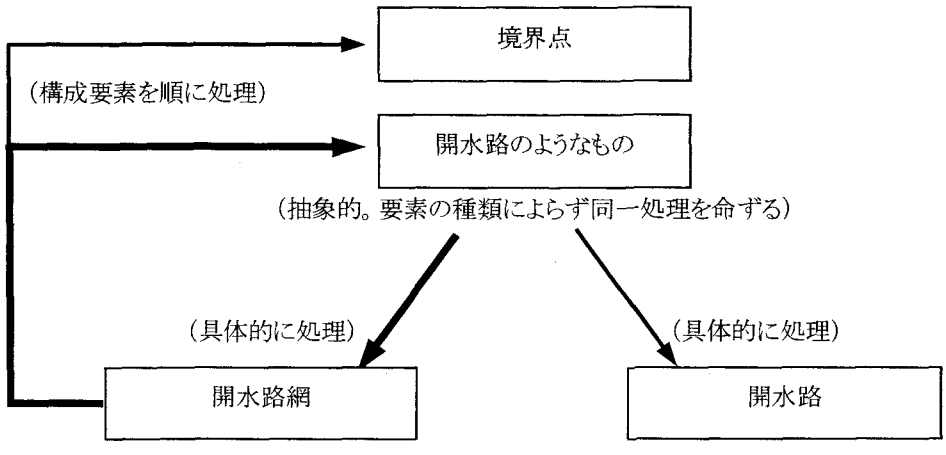
とはいえ, 開水路網の規模が大きくなれば, 開水路と境界点の接続状態を記述するのが煩雑になる。そこで, **図—1**(a)の開水路網Aのなかにたとえば開水路網BとCが含まれているものと見て, **図－1**(b)のような入れ子構成とすれば, 小さな開水路網から段階的に大きな開水路網を構築できる。開水路網を記述するデータの見通しもよいだろう。

ただし, このような構成で計算処理を進めるには, 若干




(a) 開水路と境界点のみで構成　　(b) 開水路網を入れ子にして構成

**図—1**　開水路網の入れ子構成



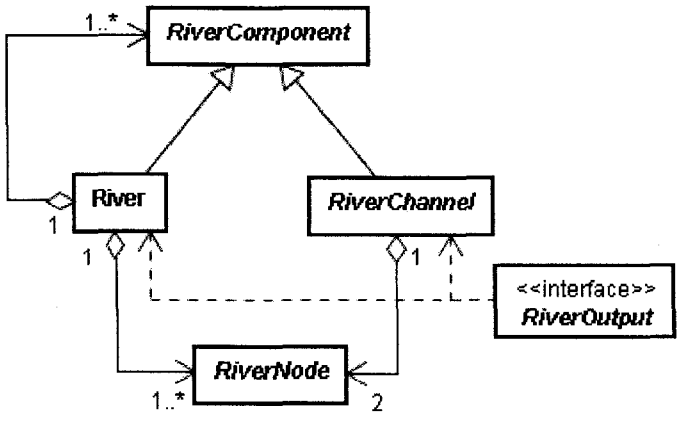

**図—2**　開水路網の再帰的処理

の工夫が必要となる。**図—2**にその概要を示す。まず，開水路網も開水路と同様に境界点にそのまま接続できることにする。その点では，両者は同じ働きをするものといえる。そこで，両者を包含するもの(概念)を「開水路のようなもの」と呼ぶことにする。つまり開水路網は，境界点と「開水路のようなもの」で構成されていることになる。

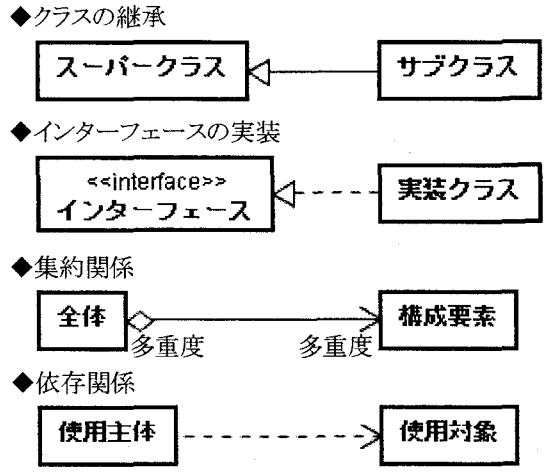
次に，ある開水路網の計算処理を行う際には，その構成要素である境界点と「開水路のようなもの」を順にめぐっていくことになる。境界点については，以前と同じように処理すればよい。ただし，たとえば**図—1**(b)の開水路網Aの計算処理を行う際には，開水路網BやCに含まれる境界点の処理はしない。

「開水路のようなもの」の処理を順に行う場合，それが開水路であれば，以前と同じように開水路の計算処理を行う。それが開水路網であれば，それを構成する境界点と「開水路のようなもの」を順にめぐって処理を行う。その「開水路のようなもの」のなかに開水路網がる場合には，それを構成する境界点と「開水路のようなもの」を順にめぐって処理・・・・・・，というように(**図—2**の太線の矢印)，開水路網の構造をたどって再帰的に処理をしていくことになる。

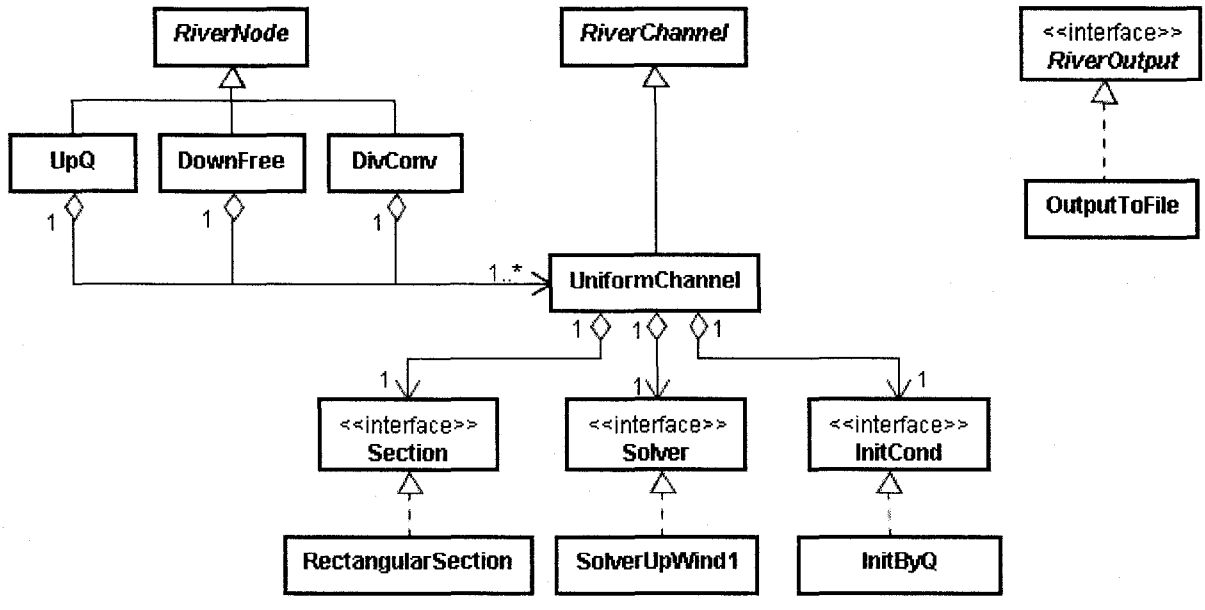
こうした「開水路のようなもの」の処理については，単純な条件分岐を用いても実現は可能であるが，後述のようにOOPにおけるクラスの継承と多態性[5]を活用すればコーディングが極めて容易であり，OOPでは頻繁に採用される処理の仕方(パターン)である。OOPでは，再利用や機能拡張がしやすい典型的なプログラミングの仕方を「デザインパターン」として整理しており[6]，今回の方法はCompositeパターンと呼ばれている。




図—3 開水路網モデルの基本的構成




図—4 クラス図の記号




図—5 開水路網不定流計算モデルの実装状況

## 4．クラスの設計

前章で説明した開水路網モデルを OOP で実現するためのクラスとインターフェースの構成について説明する。対象モデルにおけるクラスとインターフェースの構成を検討しておくことは、プログラミング全体を容易にするとともに、再利用性や拡張性を高めるうえでも重要である。

クラスとはオブジェクトの設計図としてのプログラム単位である。オブジェクトは実際世界の「もの」や「こと」を表すものであり，さまざまなデータを保有し、さまざまな動作をさせることが可能である。Java 言語ではそれぞれをフィールドおよびメソッドと呼ぶ。オブジェクトが有するフィールドとメソッドをまとめて記述したプログラム単位が「クラス」である。

またインターフェースとはメソッドの特徴だけを決めたものであり，これにしたがって作成されたオブジェクトはすべて，同一種類の（インターフェースを持つ）オブジェクトとして対処される。

本研究における開水路網モデルの基本的構成を**図—3**に，その詳細な実装状況を**図－5**に示す。また**図—3**に示されたクラス等の主要なメソッドを**表—1**に示す。

**図—3,5**は UML（Unified Modeling Language）[7]により視覚化したものである。UMLはシステムを視覚化したり文書化したりするための標準的な仕様のひとつである。図中，

表—1 開水路網モデルの基本的クラスの主要なメソッド
(斜体字のメソッドは，サブクラスで具体的処理を記述する)

■RiverComponent クラス:「開水路のようなもの」を表す。

| メソッド名 | 引 数 | 内 容 |
|---|---|---|
| *init* | 初期化チェック用フラグ | 初期条件を設定する。 |
| *calc* | 時刻 t, 時間刻み dt | 時刻 t から時間刻み dt 進める計算を行い，計算誤差を返す。 |
| *renew* | なし | 現在時刻の変数の値を新しい時刻の値に更新する。 |
| *output* | RiverOutput | RiverOutput オブジェクトを用いて計算結果を出力する。 |

■River クラス:開水路網を表す。

| | | |
|---|---|---|
| init,<br>calc<br>renew | RiverComponent と同じ | 境界点と「開水路のようなもの」を順にめぐって処理を行う。 |
| output | RiverOutput | RiverOutput の output メソッドを自分自身(River)を引数にして呼び出す。 |
| iterator | なし | RiverComponent オブジェクトの反復処理用リストを返す。 |
| add | RiverComponent あるいは RiverNode | 開水路網の構成要素に加える。 |
| remove | RiverComponent あるいは RiverNode | 開水路網の構成要素から除外する。 |
| getRiverComponent | RiverComponent の名前 | 名前に対応する RiverComponent オブジェクトを返す。 |
| getRiverNode | RiverNode の名前 | 名前に対応する RiverNode オブジェクトを返す。 |

■RiverChannel クラス:開水路を表す。

| | | |
|---|---|---|
| *init*<br>*calc*<br>*renew* | RiverComponent と同じ | RiverComponent と同じ。サブクラスに委ねる。 |
| output | RiverOutput | RiverOutput の output メソッドを自分自身(RiverChannel)を引数にして呼び出す。 |
| setDownNode, setUpNode | RiverNode | 開水路の上流端あるいは下流端の境界点を定める。 |
| getDownNode, getUpNode | なし | 開水路の上流端あるいは下流端の境界点を返す。 |
| getX, getZ, getH,<br>getA, getQ, getV | なし | 縦断座標 X を示す配列と，それに対応した河床高 Z, 水深 H, 断面積 A, 流量 A, 流速 V を与える配列を返す。 |

■RiverNode クラス:開水路網の境界点を表す。

| | | |
|---|---|---|
| *init*<br>*calc*<br>*renew* | RiverComponent と同じ | RiverComponent と同じ。サブクラスに委ねる。 |

■RiverOutput インターフェース:開水路網および開水路の出力処理の呼び出し方を定義する。

| | | |
|---|---|---|
| setTime | 時刻t | 出力する時刻を設定して，自分自身(RiverOutput)を返す。 |
| output | River | River の iterator メソッドにより「開水路のようなもの」の反復リストを入手し，順に処理する。 |
| output | RiverChannel | RiverChannel から X,Z,H,A,Q,V の配列を入手し，出力する。 |
| close | なし | 出力動作をすべて止める。 |

クラスあるいはインターフェースは長方形で表現されており，それをつなぐ線の意味は**図—4**に示す通りである。

白抜きの△がついた実線の矢印は，あるクラスとそれを継承して機能を追加・修正したサブクラスを結び付けている。メソッドの内容が具体的に記述されていないクラス(抽象クラス)の名前は斜体字で示される。そのメソッドの具体的内容はサブクラスに委ねられる。白抜きの△がついた破線の矢印はインターフェースとそれを実装したクラスを結びつけるものである。

白抜きのひし形がついた矢印は「集約」の関係を示すもので，ひし形のついているほうが集約する主体であり，矢印のついているほうが集約される構成要素である。両端の数字は多重度を表しており，たとえば**図－3**では，RiverChannel オブジェクト1つは RiverNode オブジェクトを2つもつことを表している。「1..*」などは「1 以上」という意味である。また破線の矢印は依存関係あるいは使用関係とよばれるもので，矢印の元にあるクラスのオブジェクトが矢印の先にあるクラスのオブジェクトを一時的に使用することを意味している，

### (1)基本的クラス構成

開水路網モデルの基本的なクラス等(**図－3**)を順に説明していく。

#### a) RiverComponent クラス

これは「開水路のようなもの」を表す抽象クラスで，このクラスによってサブクラスの開水路網と開水路を同一視する。すなわちあるRiverComponentオブジェクトは，その内実はRiverオブジェクトでもRiverChannelオブジェクトでもよい。そしてその RiverComponent オブジェクトのメソッドが呼び出された場合，その内実によってメソッドの動作は異なってくる(これをOOPにおける多態性という)。

#### b) River クラス

RiverComponent のサブクラスのひとつで，開水路網を表す。**図－3**に示されるように，1つのRiverオブジェクトは1 つ以上(複数)の RiverComponent オブジェクトとRiverNodeオブジェクト(境界点)で構成されており，それらはaddメソッドやremoveメソッドで加えたり除外したりする。

Riverクラスは，RiverComponentクラスとは集約と継承の関係で結ばれてループを形成している。これは，**図—2**で示した開水路網の再帰的処理をそのままクラス図にしたものともいえる。

iterator メソッドは他のオブジェクトが構成要素を順にめぐって処理する際に，RiverComponent のリストを提供するもので，ある。

#### c) RiverChannel クラス

RiverComponent のサブクラスのひとつで、開水路単区間のデータと計算処理をまとめたクラスである。上下流端に境界点(RiverNode オブジェクト)を設定したり(setUpNode、setDownNode)，それを取り出したり(getUpNode、getDownNode)するメソッドをもつ。

開水路の形態や計算方法は多岐にわたるので，このクラスではinit、calc、renewメソッドは具体的に記述せず，実装はサブクラスに委ねることにする。

#### d) RiverNode クラス

境界条件を表す抽象クラスで、RiverChannel が計算をする際の上下流端の処理を受け持つ。この部分をRiverChannelから分離することによって、さまざまな境界条件に対応したオブジェクトを生成させることが可能になる。いろいろな境界条件を設定できるように，具体的な処理はサブクラスに委ねるようになっている。

#### e) RiverOutput インターフェース

計算結果を出力するメソッドを規定するインターフェース。その output メソッドは、RiverChannel オブジェクトを引数にするものと River オブジェクトを引数にするものでは，以下のように動作が異なる。

RiverChannel オブジェトを引数とするものは，呼び出し元の RiverChannel オブジェクトが保有するデータを出力する操作を行う。一方、River オブジェトを引数とするものは，呼び出し元の River オブジェクトに収納されているRiverComponent オブジェクトの反復リストを入手して，それを順にめぐって output メソッドを呼ぶようにする。そのRiverCimponentオブジェクトがRiverChannelとRiverオブジェクトを同一視しているものなので，開水路網の構造をたどって出力処理を続けていくことができる。

これはデザインパターンのうちのVisitorパターンを活用したものである[7]。こうした操作は、RiverComponent クラスの init、calc、renew メソッドのように、River クラスとRiverChannel クラスとに分けて実装してもよいのだが、動作が固定化してしまうし、計算部分と入出力操作はなるべく分けておくようにしたほうが、いろいろなアプリケーションで再利用するのに便利であろう。

### (2) 具体的な処理の実装

本研究では，前節で検討したクラスの具体的な処理をサブクラスにより次のように実装した(**図—5**参照)

#### a) RiverNode クラスのサブクラス

第2章に述べた3種の境界条件を処理するものとして上流端のUpQクラス、下流端のDownFreeクラス、分合流点のDivConvクラスをコーディングした。

#### b) RiverChannel クラスのサブクラス

今回は単純に，勾配と断面形状が一様な開水路の計算を行うクラス UniformChannel をコーディングした。このクラスでは，断面性状，不定流の計算方法，初期条件の設定方法を記述することになるが，なるべく拡張性を持たせたいことから，この3種類の機能をそれぞれ，Section，Solver，InitialCondition というインターフェースを通して実装することにした。

#### c) Section インターフェースとその実装クラス

断面性状を提供するインターフェースで，水深に対応した断面積、径深、摩擦係数等を計算するメソッドを規定する。等流状態での水深や流量を求めるメソッドも含ませることにする。本研究では，幅広長方形断面を扱うWideRectangularSectionクラスをコーディングした。

#### d) Solver インターフェースとその実装

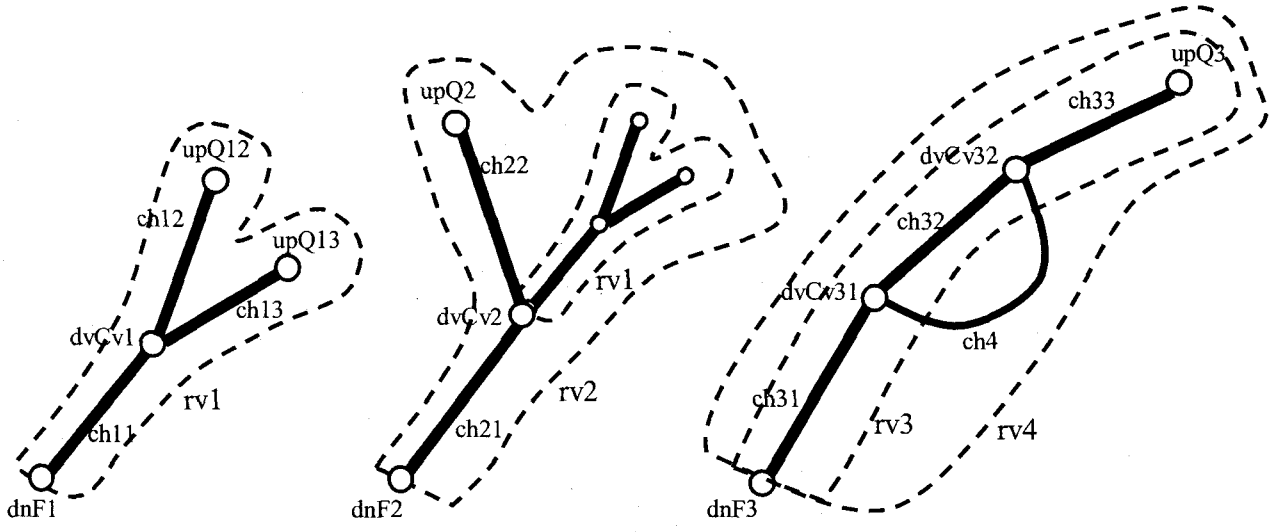
一様な開水路の不定流計算を受け持つメソッドを規定する。本研究では，1 次風上差分による計算を実施するものとしてSolverUpWind1クラスをコーディングした。

#### e) InitialCondition インターフェースとその実装

一様な開水路の初期条件を設定するメソッドを規定する。今回はその実装クラスとして、与えられた流量に応じた等流水深を設定するInitByUniQクラスをコーディングした。

#### f) RiverOutput インターフェースの実装

今回は単純に、開水路網の構造をたどりながら得られた開水路のデータを，CSV 形式でファイルに書き出すOutputToFileクラスをコーディングした。




(a) Case1　　(b) Case2　　(c) Case3

**図—6**　計算対象とした開水路網

**表—2**　UniformChannel オブジェクトの仕様

| 名称 | 長さ(km) | 幅(m) | 河床勾配 |
|---|---|---|---|
| ch11 | 30 | 100 | 1/1000 |
| ch12 | 20 | 100 | 1/1000 |
| ch13 | 25 | 50 | 1/1000 |
| ch21 | 60 | 200 | 1/2000 |
| ch22 | 40 | 200 | 1/2000 |
| ch31 | 40 | 200 | 1/2000 |
| ch32 | 30 | 100 | 1/2000 |
| ch33 | 30 | 200 | 1/2000 |
| ch4 | 20 | 200 | 3/4000 |

※粗度係数はすべて 0.03 とした。

**表—3**　UpQ オブジェクトの仕様

| 名称 | Qb | Qp | Tp | C |
|---|---|---|---|---|
| upQ12 | 60 | 600 | 12 | 20 |
| upQ13 | 40 | 400 | 9 | 20 |
| upQ2 | 100 | 1000 | 15 | 20 |
| upQ3 | 200 | 2000 | 15 | 20 |

## 5．プログラミング例と計算結果

具体的な開水路網について不定流計算を行うプログラムを、先述のクラスを利用してコーディングし、その計算結果を見ることにする。ここでは以下に示す3つの開水路網を計算対象とする(**図—6** 参照)。

Case1：2本の開水路が合流したもの。

Case2：Case1の開水路網がさらにもう 1 本の開水路と合流しているもの。

Case3：1 本の開水路が途中で分岐し、それがまた下流で合流しているもの。

**図—6** には RiverChannel、River、RiverNode それぞれのオブジェクトの名称が示されている。また**表—2**、3には それぞれの RiverChannel オブジェクトと UpQ オブジェクトの仕様が示されている。また本研究では，各オブジェクトの生成や計算に必要なパラメターはすべてプログラムリスト上に書き込む形になっている。後述するリスト上では「各データ」とある部分や省略された部分に書かれている。

**リスト—1** に、Case1 のプログラムの中核部分を示す。はじめの4行では、下流端と合流点、2つの上流端のオブジェクトを生成している。次に UniformChannel オブジェクト"ch11"、"ch12"、"ch13"を生成しながらその上下流端の境界条件となる RiverNode オブジェクトを設定している。そして River オブジェクト rv1 を生成して、これにこれまで生成したオブジェクトを収納していく。これで計算自体の枠組みである River オブジェクトが完成する。

あとはこのオブジェクトを用いて計算を進めていくことになる。そのまえに計算結果をファイル出力する OutputToFile オブジェクト"rvOut"を生成し、init メソッドで rv1 全体の初期条件を瀬呈しておく。つぎに与えられた計算時刻ステップ数 Nt だけ時間進行計算を実施する。

各時間ステップでは、与えられたは収束計算回数 Nit を上限に反復計算を行い、その誤差(変数 eps、calc メソッドの戻り値)が許容範囲内(変数 epsA)となれば、その時間ステップの計算を終了し、renew メソッドで旧時刻の値を更新する。そして決められた時間間隔 Mt ごとに、計算結果を output メソッドで出力する。

**図—6** (a)に示した計算対象の有様と、プログラムコードがよく対応しており、いったんクラスをコーディングしてしまえば、それを再利用して開水路網を構成するのは直感的

```
RiverNode dnF1 = new DownFree("dnF1", 各データ・・・・・・)
RiverNode dvCv1 = new DivConv("dvCv1",各データ・・・・・・)
RiverNode upQ12 = new UpQ("upQ12",各データ・・・・・・)
RiverNode upQ13 = new UpQ("upQ13",各データ・・・・・・)

UniformChannel ch11 = new UniformChannel("ch11",各データ・・・・・・)
ch11.setUpNode(dvCv1);
ch11.setDownNode(dnF1); //

UniformChannel ch12 = new UniformChannel("ch12",各データ・・・・・・)
ch12.setUpNode(upQ12);
ch12.setDownNode(dvCv1);

UniformChannel ch13 = new UniformChannel("ch13",各データ・・・・・・)
ch13.setUpNode(upQ13);
ch13.setDownNode(dvCv1);

River rv1 = new River("rv1");
rv1.add(upQ12);
rv1.add(upQ13);
rv1.add(dvCv1);
rv1.add(dnF1);
rv1.add(ch11);
rv1.add(ch12);
rv1.add(ch13);

RiverOutput rvOut1 = new OutputToFile("Case1.csv");

rv1.init(true);
rv.output(rvOut1.setTime(0));

for(i = 0; i < Nt; i++){
   t = i * dt;
   for (iit = 0; iit < Nit; iit++) {
      eps = rv1.calc(t, dt);
      if (eps <= epsA) break;
```

**リスト－１** Case1 のプログラムの中核部分

にコーディングできることがわかる。

ただしそれでも、比較的大きな開水路網を1から組み立てようとすると、ずいぶんな手間になることは避けられない。そこで、River オブジェクトがまた別の River オブジェクトを収納できることを有効に活用したい。

**リスト－２**は、Case2のプログラムで最上位の River オブジェクト"rv2"を生成する部分を示したものである。まず、"rv1"以外の RivrNode オブジェクトと RiverChannle オブジェクトを生成させる。つぎに以前に作成した"rv1"を生成させるコード(createRv1 メソッドとしておく)を用いて、"rv1"を生成する。そして"rv1"を新しい開水路網に接続させるために、下流端の自由境界条件オブジェクト"dnF1"を取り出して"rv1"から廃棄し、合流することになるRiverChannel オブジェクト"ch11"を取り出して、その下流端に今回新しく生成した分合流点オブジェクト"dvCv2"を設定してやればよい。あとはすべての構成メンバーを新しく生成した River オブジェクト"rv2"に収納するだけである。"rv1"はそのままの形で収納すればよいので、プログラムコードの可読性は良好なままである。

このような River オブジェクトの作り方は、単に以前作成

```
RiverNode dnF2 = new DownFree("dnF2",各データ・・・・・・)
RiverNode dvCv2 = new DivConv("dvCv2",各データ・・・・・・)
RiverNode upQ2 = new UpQ("upQ2",各データ・・・・・・)

UniformChannel ch21 = new UniformChannel("ch21",各データ・・・・・・)
ch21.setUpNode(dvCv2);
ch21.setDownNode(dnF2);

UniformChannel ch22 = new UniformChannel("ch22",各データ・・・・・・)
ch22.setUpNode(upQ2);
ch22.setDownNode(dvCv2);

River rv1 = createRv1();
RiverNode dnF1 = rv1.getRiverNode("dnF1");
rv1.remove(dnF1);
RiverChannel ch11 = (RiverChannel)rv1.getRiverComponent("ch11");
ch11.setDownNode(dvCv2);

River rv2 = new River("rv2");
rv2.add(rv1);
rv2.add(upQ2);
rv2.add(dnF2);
rv2.add(dvCv2);
rv2.add(ch21);
rv2.add(ch22);
```

**リストー２** rv1 を再利用して rv2 を生成するコード

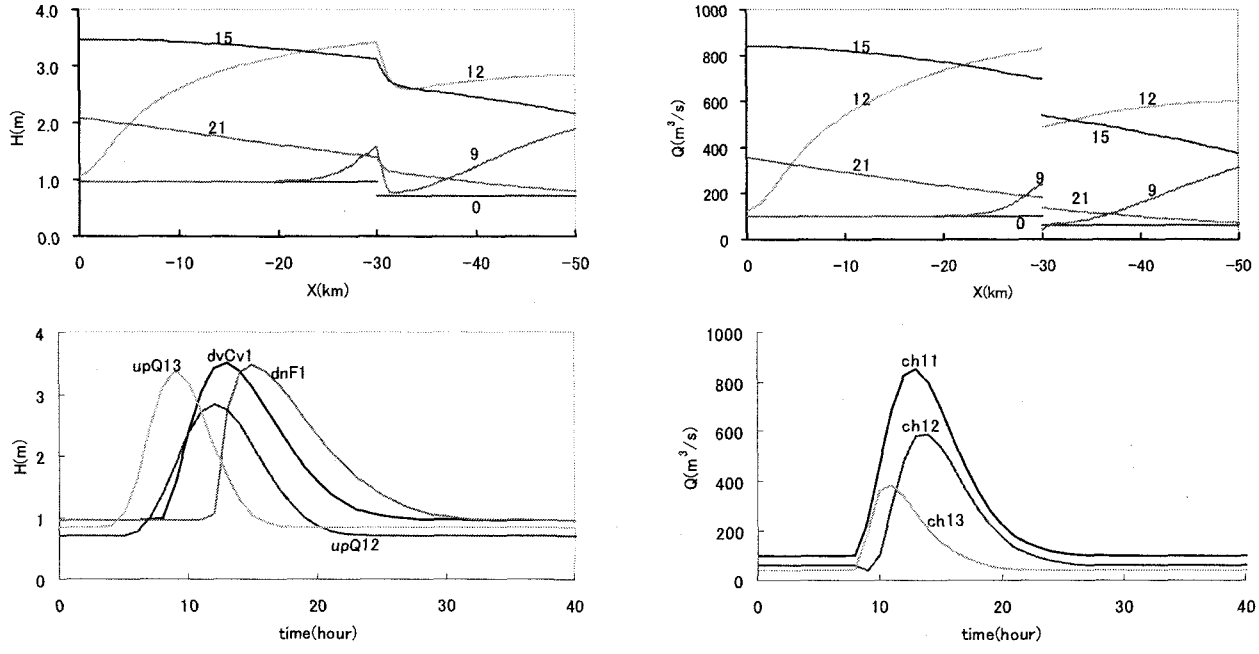
```
River rv3 = createRv3();
RiverNode dvCv31 = rv3.getRiverNode("dvCv31");
RiverNode dvCv32 = rv3.getRiverNode("dvCv32");

UniformChannel ch4 = new UniformChannel("ch4", 各データ・・・・・・)
ch4.setUpNode(dvCv32);
ch4.setDownNode(dvCv31);    //

River rv4 = new River("rv4");
rv4.add(rv3);
rv4.add(ch4);
```

**リストー３** rv3 を再利用して rv4 を生成するコード

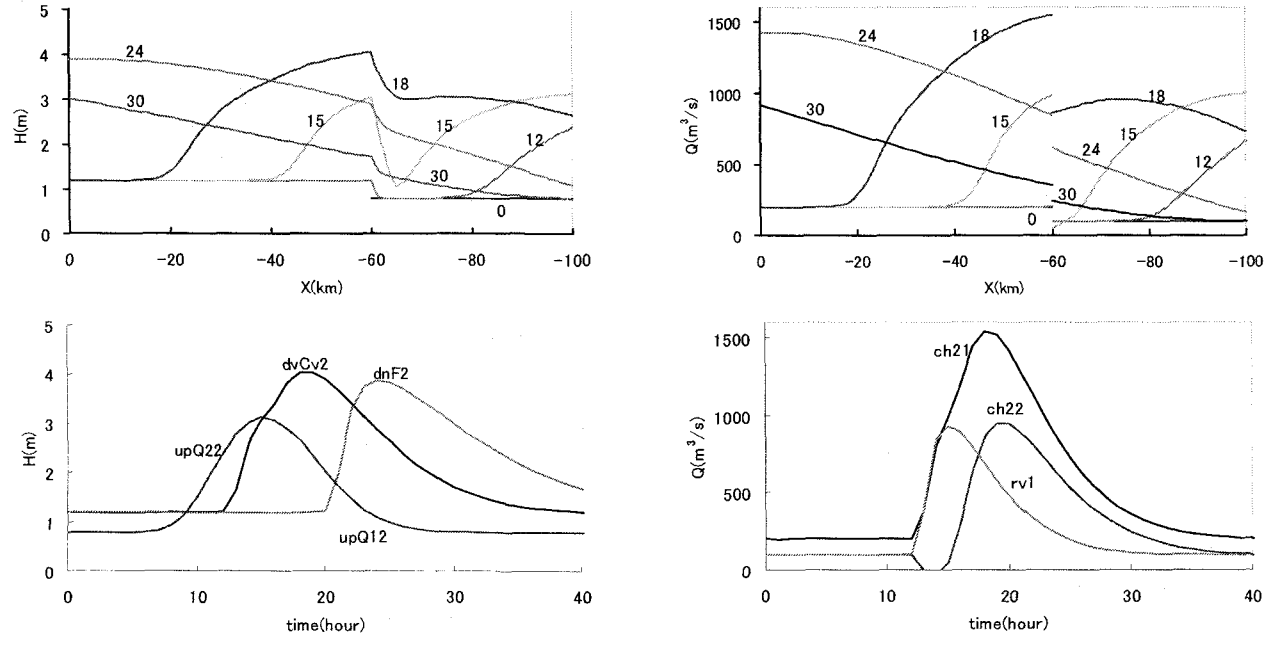
したプログラムを再利用するだけでなく、大規模な開水路網を構成しようとするときに、それをいくつかの部分開水路網に分けてコーディングする際にも有効になるものと考えられる。これは今後のアプリケーション開発や分散コンピューティングへの応用にも、示唆を含むものであろう。

**リストー３**は、Case３において River オブジェクト"rv4"を生成する部分を示したものである。

**図－７**に Case1の計算結果を示す。縦軸の H は水深を、Q は流量を表す。横軸の X は水路 ch11 から ch12 へと沿って計った距離であり、下流端をゼロとし、上流側に行くほどマイナスになる。また距離との関係を示した図中の数字は時刻(単位:時間)を意味している。横軸 time は時刻




**図－7　Case1の計算結果**



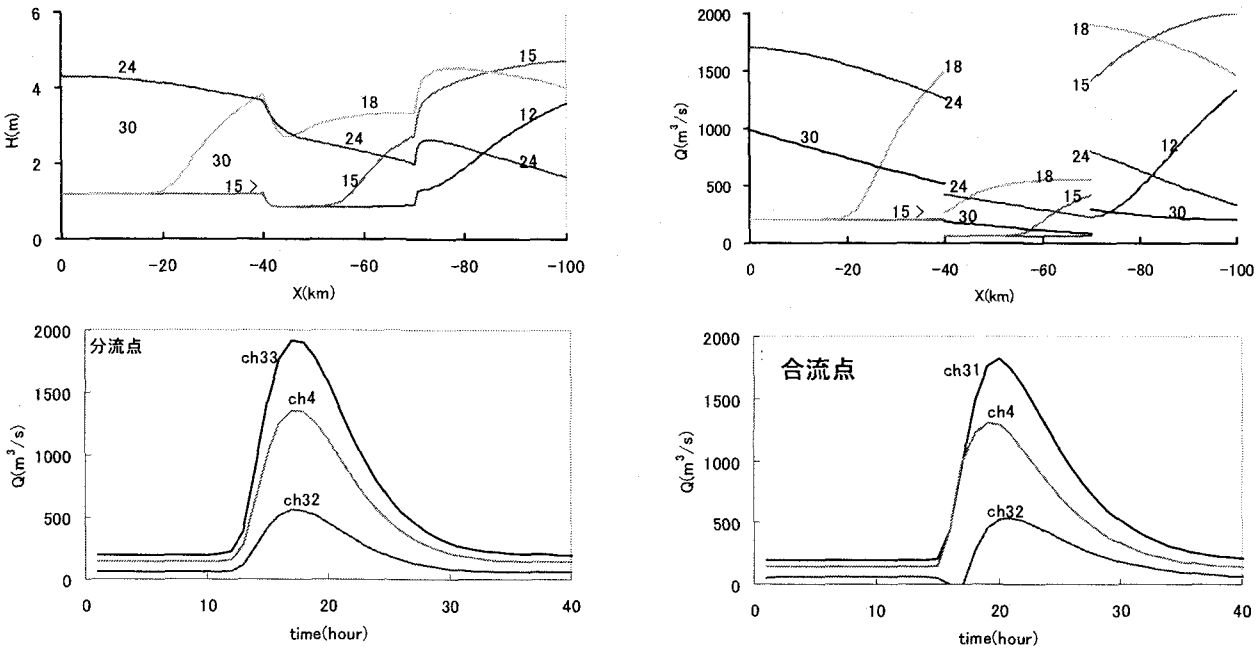

**図－8　Case2の計算結果**

(単位:時間)である。またその図の記号は、どの地点での値かを示している(**図－6**参照)。

この図を見ると、合流点から下流で水深も流量も増加している。特に、時刻9時間でのch12の合流直前での水深変化は著しく、そのため合流する流量が若干減少するという現象が見られる(Q-t図)。

**図—8**は Case2 の計算結果である。今度は距離 X はch21から ch22へと沿って測った距離である。これを見ると、やはり合流直前での水深変化が著しく、時刻12時間付近では、若干の逆流(流量が負)になる時間帯も見られる。

**図—9**は Case3の計算結果である。ここでは距離 X は ch31、ch32、ch33 と沿って計った距離である。この計算例では ch32 の水路幅が狭いので、本来なら他の水路よりも水深が増加するはずであるが、ch32の上流で幅の広い水路ch4に分岐している効果により、他の水路よりに比べて水深、流量ともに減少している。




**図－９** Case3の計算結果

ただし、合流する「－40km」地点での水深変化が著しく、特に時刻 16 時間前後では、ch32 からの流量が負になる状態が見られる。

以上のように，本研究におけるモデル化は，さまざまな開水路網の状況に応じて，直感的で可読性に優れたプログラムリストを作成でき，それでいて計算結果は開水路網の複雑な挙動を十分に再現できるものといえる。

## ６．おわりに

本研究では、開水路網の不定流計算をオブジェクト指向プログラミング（OOP）の手法によりモデル化し、Java 言語で実装した。その際に、OOP のデザインパターンである Composite パターンを用いて、開水路網の中にまた別の開水路網を有するような再帰的構造の扱いを簡便にした。また Visitor パターンを用いて、再利用する際の入出力操作などを、開水路網の構造を巡りながら順々に実行していく仕組みを組み込んだ。

その結果、プログラム上で開水路網を構成するのが極めて直感的で可読性のよいものになることを示すことができた。再帰的構造を有しているので、以前に作成した開水路網オブジェクトをそのまま再利用することができ，大規模な開水路網を構成する際には，分担して段階的にコーディングしていくことが可能となった。これによって，アプリケーションの開発効率も向上すると期待できる。

また，本研究では数多くのメソッドを記述することになったが，それらはまず，クラスというくくりで整理されている。さらにそのクラスは，**図―3**に示す基本的なクラスと**図―5**に示す詳細な実装を記述した部分とに分けられており，実装部分を交換・修正することで，さまざまな状況に対応できることになる。今後，開発が進んでさらに多数のクラスやメソッドが作成されても，ライブラリ化は容易なものといえよう。

しかしこのままでは、あくまでプログラムコードを書かなくては、開水路網を生成して計算を行うことはできず，それには Java 言語の習得が必要になる。もっと敷居の低い仕組みを提供したり，インターネット環境下で活用したりすれば，さらに利用機会を増やして再利用の恩恵を発揮することができよう。

これにはたとえば，XML の活用が挙げられる。XML で開水路網に関するさまざまなデータ項目をわかりやすい名前で定義することにより，Java 言語を理解していなくても，容易に開水路網の構成を記述することが可能になる。また XML のリンク機能を利用することで，開水路網を記述した複数の XML 文書を入れ子状態にすることができる。それらの XML 文書を順に読み込んで，プログラム上で自動的に開水路網オブジェクトを生成させるのである。Java 言語は XML との親和性も高いので，こうしたシステムの実現は難しくないだろう。

また開水路網を記述した XML 文書が複数のコンピュータ内に分散している場合には，インターネットを通じてその内容を入手して動作させることも，インターネットとも親和性の高い Java 言語ならば可能であろう。さらにサーバー上でサーブレットとして動作させ，Web ブラウザからコントロールすることも考えられる。

本研究では，オブジェクト指向プログラミングを利用して開水路網の計算処理を再帰的に行うことに主眼を置いて

いた。それをJava言語で実装したことによって，さまざまな利用形態への発展性も手にしたと考えている。今後はそうした方面の検討を進めていくつもりである。

**参考文献**


1) 伊藤良栄:Preissmann型陰差分法における内部境界条件の実用的・安定的計算法、農業土木学会論文集 168、pp9-18、1993.
2) 池田裕一:内部境界条件を有する開水路不定流計算のオブジェクト指向プログラミング、土木情報利用技術講演集、第29巻、2004.
3) 細田尚:常流・射流混在流れ、水理公式集プログラム例題集【例題 2-9】, 土木学会水理委員会, 2002.
4) 池田裕一:不定流計算, 水理公式集プログラム例題集【例題 2-2】, 土木学会水理委員会, 2002.
5) たとえば, 高橋麻奈:やさしい Java（第2版）, ソフトバンクパブリッシング、551p.、2002.
6) 結城浩:Java 言語で学ぶデザインパターン入門、ソフトバンクパブリッシング、480p.、2001.
7) 浅海智晴:UML & Java オブジェクト指向開発（入門編）, ピアソンエデュケーション, 240p., 2002.


（2005.5.20 受付）