第2005-08号

XMLによる河川網の記述とオブジェクト指向プログラミングおよびデータ流通への活用

宇都宮大学　工学研究科エネルギー環境科学専攻
助教授　池田　裕一

平成１９年９月

助成研究者紹介

池田　裕一
現職：宇都宮大学大学院工学研究科准教授（博士（工学））
主な著書：パソコンで解く水理学（丸善，1990）
水圏の環境（共著，東京電機大学出版局，1998）
土木技術者のための Excel 入門（共著，森北出版，1998）
土木技術者のための Excel 活用（共著，森北出版，1999）
土木技術者のための Visual Basic 活用（共著，森北出版，2001）

ＸＭＬによる河川網の記述とオブジェクト指向プログラミング およびデータ流通への活用

目　次

第 1 章

序論

１－１　研究の背景

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。

そのような情勢の下，汎用的なデータ記述言語として，XML（eXtensible Markup Language）が急速に普及している[1)]．XMLにより記述したデータファイルは，人間とコンピュータの両方が理解し処理するのに適しており，スタンドアロン環境のみならず，インターネットを介してデータを交換する際にも利用されている．特にXMLを核として活用する周辺技術の発展が目覚しく，今後はXMLなくしての情報処理はありえないといえる．

また，この 10 年で発展してきた注目すべきコンピュータ言語として、Java を挙げることができる[2)]．Java言語は，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。

さらに Java 言語については，XML 技術にもよく対応できるように，さまざまなプロジェクトが立ち上がっている．その成果を用いることで，XML を用いたアプリケーションを構築する時間と手間を，飛躍的に効率化することが可能になっている．

一方で，今日の河川に関する業務には、限られた対象区間だけでなく中小河川も含めた水系全体，すなわち河川網を対象とした状況把握や予測計算が求められてきている。水系全体での土砂管理はもちろんのこと，生態系や周辺と土地利用など，さまざまな階層のデータやモデル化が必要になっている．

それには、水系全体で共通的なデータフォーマット、しかも河川や構造物，周辺土地利用など，多彩で階層的なデータ構造とそれらの多様な相互関係を記

述できるものが必要である。このようなデータ記述には，XML が適しているものと考えられる．

そして、それらのデータを活用するアプリケーションの開発がまた重要となる．アプリケーションの種類としては，データを読み込んでさまざまな目的に応じた処理を行い結果を出力するもの，データそのもののメンテナンスを行うものなどがある．いずれにしても，アプリケーション（あるいはその一部）のメンテナンスや再利用が容易で，各所に分散するデータを効率的に活用できるようなものが要請されるところである．

１－２　研究の目的と構成

本研究では，XML により河川網を記述し，それを用いて 1 次元不定流計算を行うアプリケーションを Java 言語で開発することを最終的な目的とする。

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[3)] 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。そこで，河川網のデータの記述の仕方と，それを読み込んで計算処理する方法とそのプログラミングを，簡便で効率的なものとしたい．これにはデータやプログラムのメンテナンスも含まれる．

そのため本研究では，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 言語による実装を試みる。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する。そして，OOP により，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できるようにする．

またデータの記述には XML を活用し，水路網の入れ子構造を簡便に記述する方法を検討する．XML で記述されたデータを Java プログラムで活用するために，さまざまなプロジェクトでプログラムの開発が行われている．その成果を有効利用することで，アプリケーション構築の効率化を図ることにする．

本報告書の構成は以下のようになる．

第 2 章では，河川網の入れ子状態を再帰的に処理する手法を OOP のデザインパターンを利用して検討し，XML 技術を利用して入れ子状態を記述する方法を解説する．

第 3 章では，河川網の不定流計算を再帰的に行うアプリケーションを構築する．計算の基礎方程式を示し，アプリケーションの内部構成を設計し，そのコーディングした成果を利用して，実際に計算を実施する例を示す．ただし，河川網のデータはプログラム内に記述されたままである．

第 4 章では，河川網のデータを XML ファイルとして記述し，それを読み込んで不定流計算を行う方法について述べる．まず，XML ファイルから河川網をコンピュータ上に生成する方法を示し，つぎに，計算結果を XML ファイルとして出力する方法を述べる．そして，こうした XML ファイルを多目的に活用・流通する方法について，考察を加える．

第 5 章では，全体のまとめと今後の課題について述べる．

第２章

河川網の再帰的記述

２－１　河川網の再帰的構成

図２－１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，これら開水路と境界点という２種の要素によって開水路網を構成してやればよい。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ことになる.

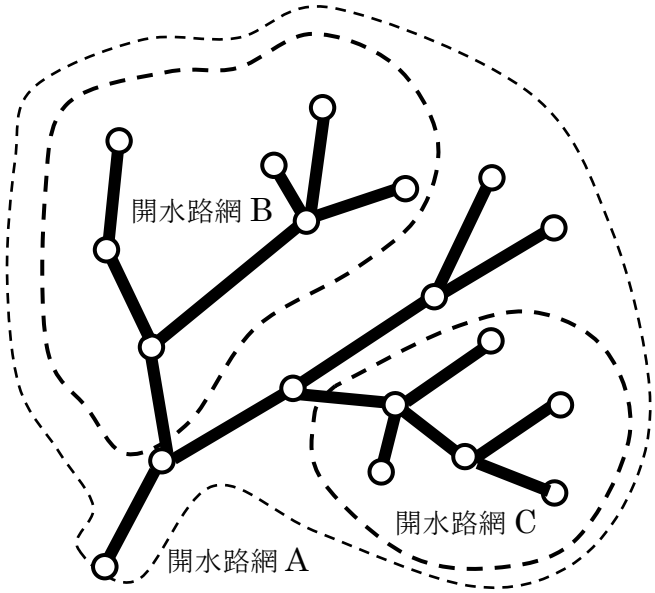


図２－１　開水路と境界点のみで構成した開水路網

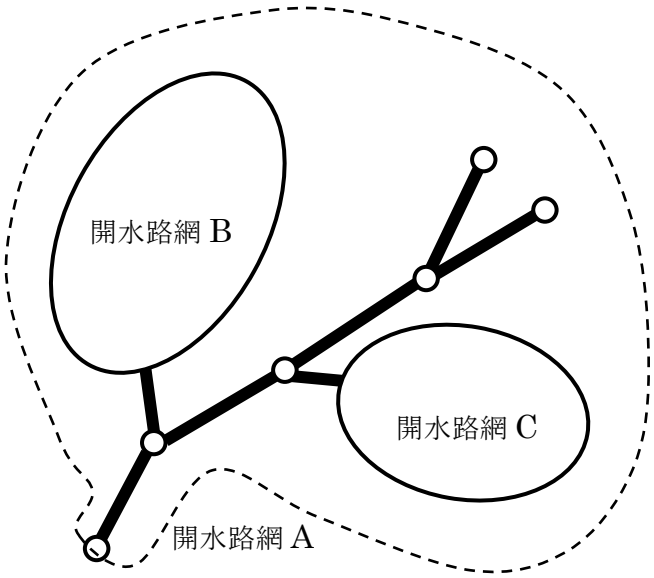


図２－２　入れ子状にした開水路網の構成

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２―１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２－２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，若干の工夫が必要となる。というのは，プログラミング言語において，同種のデータを効率よく処理するには配列を用いるのが一般的である．ところが，Fortran などオブジェクト指向プログラミングをサポートしない言語では，配列の要素として処理できるのは，数値データあるいはせいぜい構造体の類でしかない．しかも構造体のデータ構成は通常は固定されており，今回対象とするような任意の小規模開水路網を表現することは非常に困難である．したがって，こうした言語ではどうしても図２－１の構成で開水路網を処理するプログラムを記述することになる．

また開水路ごとにさまざまな境界条件を設定する必要があるので，開水路の計算を担当するサブプログラムでは，境界条件の種類に応じて計算方法を変える必要があり，プログラム上では IF 文などで計算方法を切り替えることになる．また境界条件に付随するパラメータの管理は， COMMON ブロックであれ引数であれ，条件の種類ごとに異なってくる．すると，新たな境界条件の与え方（オプション）が追加されるたびにプログラムの核心部分に関わる手直しが必要となり，しかも手直しされるたびにプログラムの核心部分そのものが複雑化してしまう．

以上のような問題は，オブジェクト指向プログラミング（OOP）[4]では容易に解決することが出来る．すなわち「もの」を示すオブジェクトの構成内容には任意性が許されており，当然それの配列（あるいはコレクションという集合体）を処理するプログラムを記述するのは容易なことである．

また，汎用的な「もの」に対して，それより下位の具体的な「もの」をいく種類も用意することが可能である．そして，必要に応じて汎用的な「もの」の名前で具体的な「もの」を割り当て，さまざまな処理を行わせる（多態性[4]という）ことにより，オプションの追加を容易なものにできる．

さらに，これまでのプログラミングの実績を分析して，効率のよい「もの」の構成の仕方と処理の方法を，デザインパターンとして提案されており，それを手本とすることで，複雑で大規模なアプリケーションであっても，メンテナンスが容易で再利用性の高いシステムを効率よく構築することが可能である．

そこで次節では，図２－２の河川網構成を処理する方法を，OOP のデザインパターンを利用して，検討することにする．

２－２　デザインパターンの活用

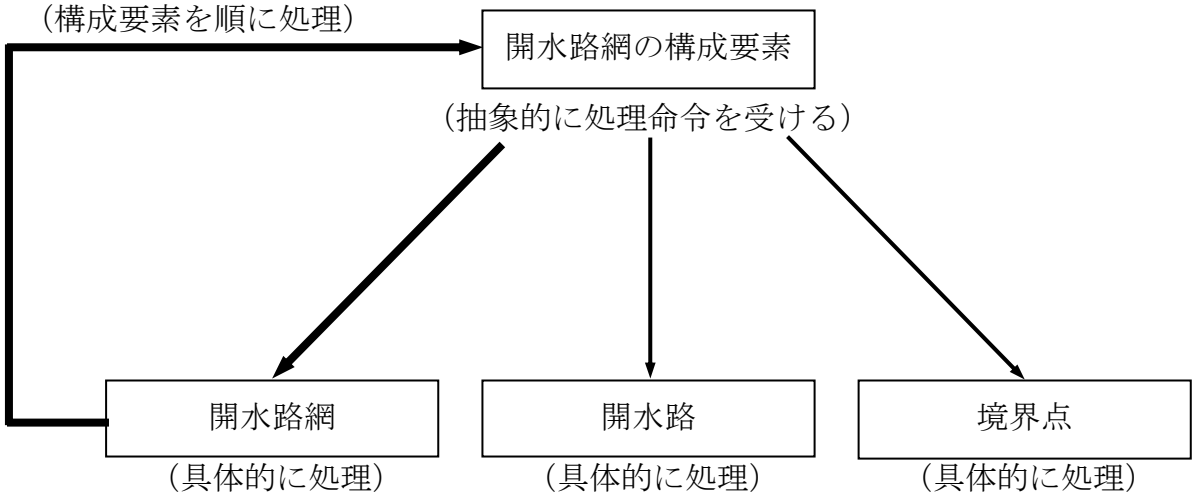


図２―３　開水路網の再帰的処理

オブジェクト指向プログラミング（OOP）において，図２－２のような河川網構成を処理する場合の「もの」の成り立ちと働きは，図２－３のように表すことが出来る．

まず，開水路網，開水路，境界点の３者はともに，「開水路網の構成要素」ということができる．いいかえれば，開水路網は「開水路網の構成要素」がいくつか集まって構成されていることになる。

すると，ある開水路網の計算処理を行う際には，その開水路網がもつ「開水路網の構成要素」を順にめぐっていくことになる。順に処理していく際に，それが実は開水路や境界点であったり，開水路網であったりするので，それに応じた処理を行うことになる．

実体が開水路や境界点である場合には，それらに割り当てられた処理を実行させればよい．実体が開水路網（図２－２の開水路網 B のような）であれば，それがもつ「開水路網の構成要素」をまた順にめぐって処理を行う。その「開水路網の構成要素」のなかに開水路網がある場合には，それがもつ「開水路網の構成要素」をまた順にめぐって処理・・・・・・，というように（図２－３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「開水路網の構成要素」の個別の処理については，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[2)]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[5]，図２－３のような処理の方法は「Compositeパターン」と呼ばれている。

また，これら個別の要素に対して，一時的な処理たとえば一定時間ごとの計算結果の出力を行いたい場合は少なくない．ただし，構成要素の種類に応じて処理内容は異なってくるだろうし，出力フォーマットも状況に応じて使い分ける必要も予想される．そのような処理は，「河川網の構成要素」の個別のプログラム単位にコーディングしないで，「オプション処理をするもの」として，別のプログラム単位としてコーディングするほうが賢明であろう．

そして，さまざまな状況に応じた「オプション処理をするもの」が，図２－３の再帰的構成に沿って「河川網の構成要素」を順に渡り歩きながら，個別の要素に応じた処理を行っていくようにすればよい．このような対処の仕方は，「Visitor パターン」と呼ばれている．

以上述べたような「Composite パターン」と「Visitor パターン」を，開水路網の不定流計算について，具体的にどのようにコーディングするかは，第 3 章において説明する．

２－３　XML の活用

XML（eXtensible Markup Language）は，データ記述形式の一つで，タグと呼ばれる情報をデータの中に埋め込むものである[6]. XML形式のファイルは，テキストエディタなどでも作成が可能であり，タグを見ればデータの内容がすぐわかるので，人間がそのまま扱いやすい．しかもタグを使う側が定義して使うことができるので，さまざまなデータを柔軟に表すことができる．

また，XMLファイルは基本的にはテキストファイルであり，XMLの文法が簡潔なので，プログラム上からも扱いやすくなっている．さらにXMLファイルをJava言語で簡便に扱うためのさまざまなプロジェクトが立ち上がっており，そこで開発されたプログラム群を利用することで, XMLファイルを扱うアプリケーションを効率的に構築することができる[6].

XML そのものは簡潔であるが，簡潔であるがゆえに，さまざまな周辺技術が提案・標準化され, 運用されている（消滅していくものもある）. そのなかに，XML ファイルをリンクする機能がある．これはファイルの一部を別のファイル（またはその一部）と関連付けるものである．

たとえば, 河川網 A の構成の仕方を記述する XML ファイル（図２－４の「河川網 A.xml」）のなかに，河川網 B や C に関する内容を具体的に記述するかわ

りに，それぞれの河川網の構成を記述した XML ファイル（同図の「河川網 B.xml」，「河川網 C.xml」）へのリンクを記しておけば，小規模な河川網から段階的に河川網のデータを記述することができる．つまり図２－３のような構成要素の関係を XML ファイルの構成そのものを用いて表すことが可能になる．

XML ファイルを用いて河川網のデータの記述および，それを読み書きするアプリケーションの構築については，第 4 章において説明する．

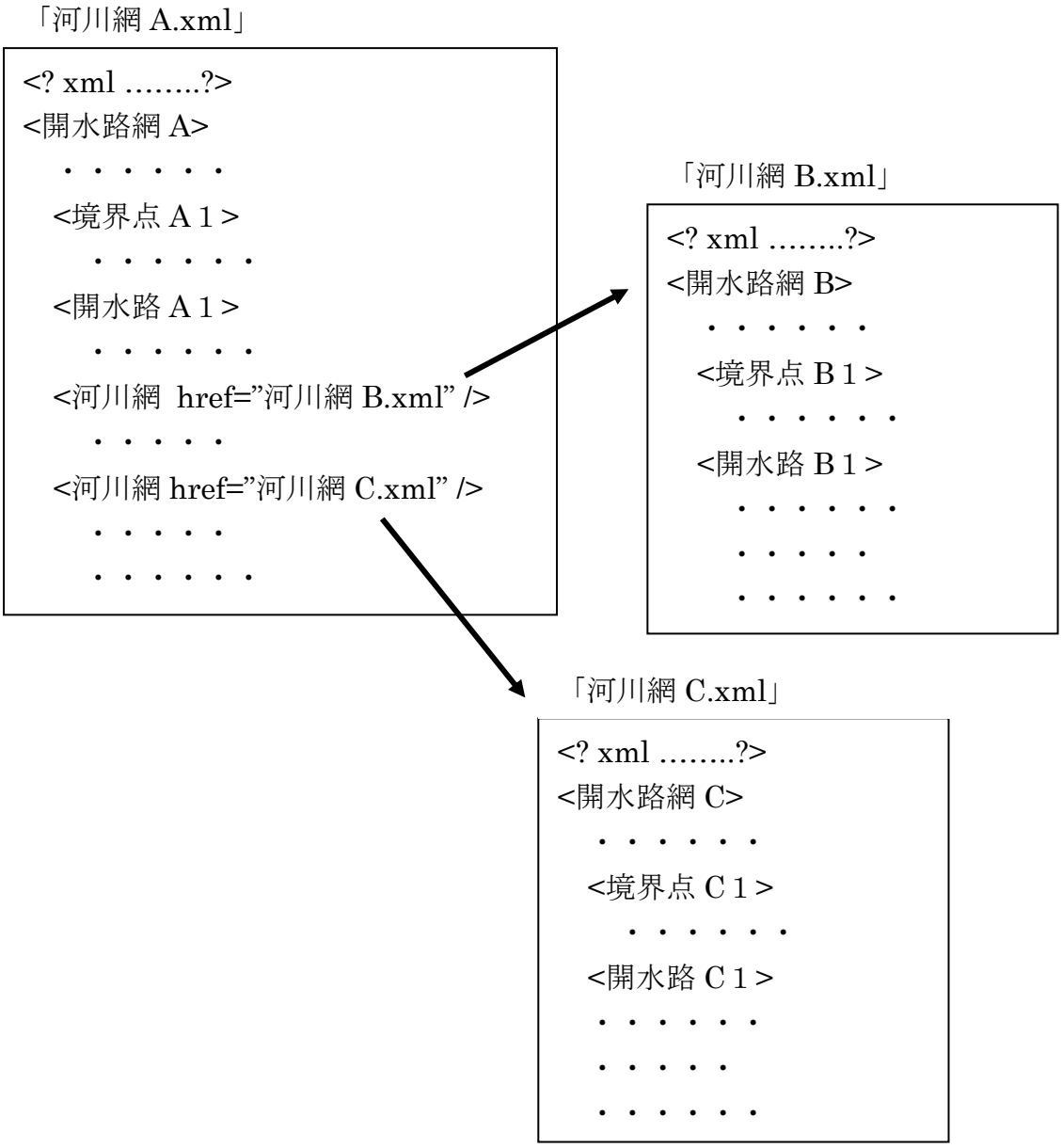


図２－４　XML のリンク機能を用いた河川網データの入れ子状態の概念図
（あくまで模式的なものあり，タグの記述は厳密ではない）

第３章

河川網不定流の
再帰的計算プログラム

３－１　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[8].

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

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

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 8)に説明されている陽解法を参考にした．ただし，文献 8）では一様勾配の一様断面水路を対象にしているのに対し，ここでは勾配・断面形とも非一様な場合にも対応できるようにする．

（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。

開水路の上流端では流量を次式のように与える[8]。

$$Q_0(t)=Q_b+(Q_p-Q_b)\left\{\frac{t-t_0}{t_p}\exp\left(1-\frac{t-t_0}{t_p}\right)\right\}^{C_p} \quad \cdots (4)$$

ここに，Q_b：基底流量，Q_p：ピーク流量，t_0：洪水開始時刻，t_p：ピーク時刻，C_p：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。

（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。

３－２　クラスの設計

本節では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。

（1）クラスとその視覚化

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。またクラスをもとに実際のオブジェクトを生成したものを「インスタンス」とよぶ.

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インスタンスはすべて，同一種類の（インターフェースを持つ）インスタンスとして対処される。

以上のようなクラスやインターフェースの種類や相互関係は視覚化すると理解しやすい．そのための標準仕様はUML（Unified Modeling Language）[9] と呼ばれている．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。クラス図では，クラスや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３－１に示す通りである。

つまり白抜きの△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

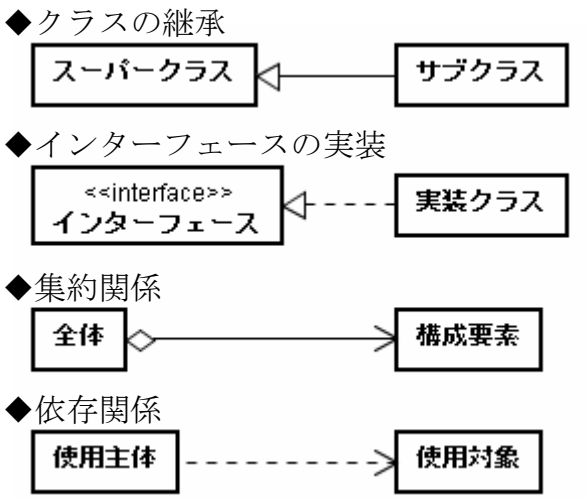


図３－１　クラス図の記号

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。その両端に数字等が記して，多重度を表す場合がある．多重度とは，線で結ばれた片側のクラスのインスタンスひとつが，もう一方のクラスのインスタンスのいくつに結びついているかを示すものである．「1 以上」を表すには，「1..*」などと記す．

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インスタンスが矢印の先にあるクラスのインスタンスを一時的に使用することを意味している．

（2）基本的クラス構成

本研究における開水路網モデルの基本的クラスの構成を図３－２に示す．また同図に示されたクラス等の主要なメソッドを表３―１に示す。以下，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。

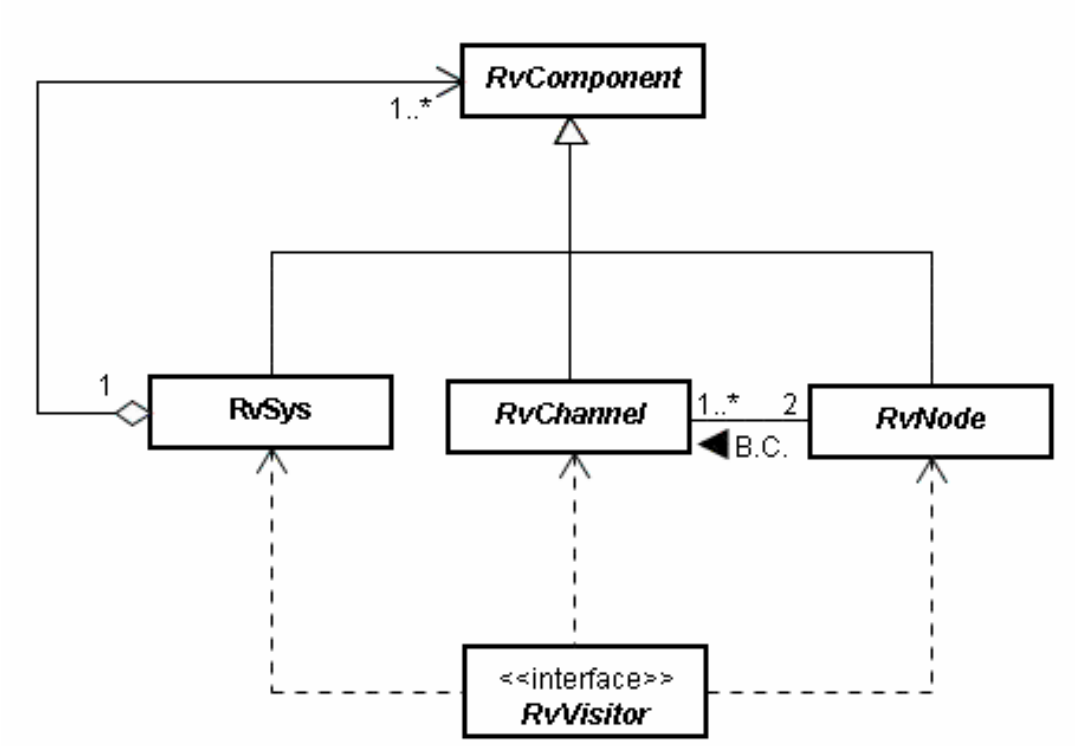


図３－２　基本的なクラス等の構成

表３―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）

■RvComponent クラス：「開水路網の構成要素」を表す。

メソッド名	引　数	内　容
init	初期時刻，初期化済みチェック用フラグ	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 進める計算を行う.
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。
accept	RvVisitor	RvVisitor インターフェースを実装したインスタンスを受け入れて，そのインスタンスにさまざまな処理を実行させる.。

■RvSys クラス：開水路網を表す。

init,	RvComponent と同じ	開水路網のなかの RvComponent インスタンスを順にめぐって処理を行う。
calc		
renew		
accept	RvVisitor	RvVisitor の実装インスタンスの visit メソッドを自分自身（RvSys）を引数にして呼び出す。
add	RvComponent	RvComponent インスタンスを開水路網の構成要素に加える。
remove	RvComponent	RvComponent インスタンスを開水路網の構成要素から除外する。
getRvComponents	なし	開水路網の中の RvComponent インスタンスの集合体を返す.
getRvComponent	RvComponent の名前	名前に対応する RvComponent インスタンスを返す。
setDownNode	RvNode の名前, RvChannel の名前	RvNode を RvChannel の下流端条件として設定する.
setUpNode	RvNode の名前, RvChannel の名前	RvNode を RvChannel の上流端条件として設定する.

表―1　（つづき）

■RvChannel クラス：開水路を表す。

init *calc* *renew*	RvComponent と同じ	具体的な処理内容は，サブクラスに委ねる。
accept	RvVisitor	RvVisitor の実装インスタンスの visit メソッドを自分自身（RvChannel）を引数にして呼び出す。
setDownNode,	RvNode	RvNode をこの開水路の下流端条件として設定する.
setUpNode	RvNode	RvNode をこの開水路の上流端条件として設定する.
getX, getZ, getH, 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。

■RvNode クラス：開水路網の境界点を表す。

init *calc* *renew*	RvComponent と同じ	具体的な処理内容は，サブクラスに委ねる。

■RvVisitor インターフェース：開水路網構成をたどった処理の呼び出し方。

visit	RvSys	RvSys の getRvComponents メソッドにより RvComponent のリストを入手し，順に処理する。
visit	RvChannel	RvChannel について所定の処理を行う.
visit	RvNode	RvNode について所定の処理を行う.

(a)RvComponent クラス

これは「開水路網の構成要素」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，境界点を同一視する。すなわちあるRvComponent インスタンスは，その内実は開水路網を表す RvSys インスタンスでも，開水路を表す RvChannel インスタンスでも，境界点を表す RbNode インスタンスでもよい。そしてその RvComponent インスタンス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)RvSys クラス

RvComponent のサブクラスのひとつで，開水路網を表す。図３－２に示されるように，１つの RvSys インスタンスは１つ以上（複数）の RvComponent インスタンスで構成されており，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。

RvSys クラスは，RvComponent 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２－３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。

getRvComponents メソッドは，他のインスタンスがその開水路網の構成要素を順にめぐって処理する際に，RvComponent のリストを提供するものである。

(c)RvChannel クラス

RvComponent 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。上下流端に境界条件の計算を担当する RvNode インスタンスを設定するメソッド（setUpNode、setDownNode）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)RvNode クラス

境界条件を表す抽象クラスで、RvChannel 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 RvChannel 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計算を実行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)RvVisitor インターフェース

河川網の中を順に巡って，構成要素に応じた処理を行わせるためのインター

フェース。RvComponent のサブクラスである RvSys，RvChannel，RvNode クラスには，RvVisitor 実装インスタンスを受け入れるための accept メソッドが用意されている．各クラスの accept メソッドでは，受け入れた RvVisitor 実装インスタンスの RvVisitor 実装クラスでは，それぞれのインスタンスを引数とする visit メソッドを，自分自身を引数として呼び出すようになっている．

visit メソッドには RvSys，RvChannel，RvNode それぞれのインスタンスを引数とするものが用意されており（それが RvVisitor インターフェースの定義になっている），インスタンスの種類に応じて異なる動作をするようにコーディングすることが可能である．

RvChannel や RvNode のインスタンスを引数とするものは，呼び出し元の RvChannel インスタンスに関する操作を行う。一方、RvSys インスタンスを引数とするものは，呼び出し元の RvSys インスタンスに収納されている RvComponent インスタンスのリストを入手して，それを順にめぐって accept メソッドを呼ぶようにする。その RvComponent インスタンスが RvSys，RvChannel，RvNode それぞれのインスタンスを同一視しているものなので，結局，図２－３に示したような仕組みで，開水路網の構造を順にたどって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[5]。こうした操作は、RvComponentクラスのinit、calc、renewメソッドのように、サブクラスで実装してもよいのだが，動作が固定化してしまうし、計算部分とそれ以外のさまざまな操作を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。

（３）具体的な処理の実装

本研究では，先に述べた基本的なクラス等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３－３参照）．以下，これらのクラス等について順に説明していく．

(a) RvNode クラスのサブクラス

前節で述べた３種の境界条件を処理するものとして上流端の UpDicharge クラス、下流端の DownFree クラス、分合流点の DivCon クラスをコーディングした。

(b)RvChannel クラスのサブクラス

本研究で回は勾配・断面形とも非一様な開水路の計算を行うクラス RvNonUniChannel をコーディングした。このクラスでは，計算に必要な各断

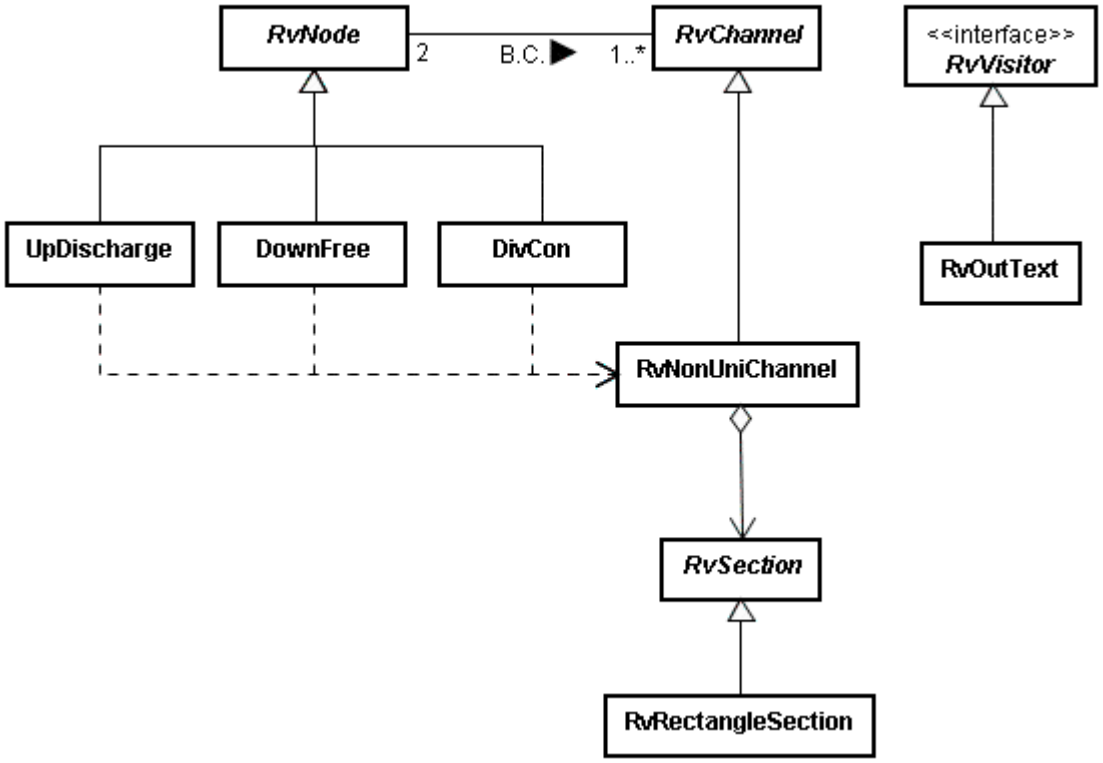


図３－３　開水路網不定流計算モデルの実装クラス

面の性状を，RvSection（抽象）クラスを通して取得することにしてある．

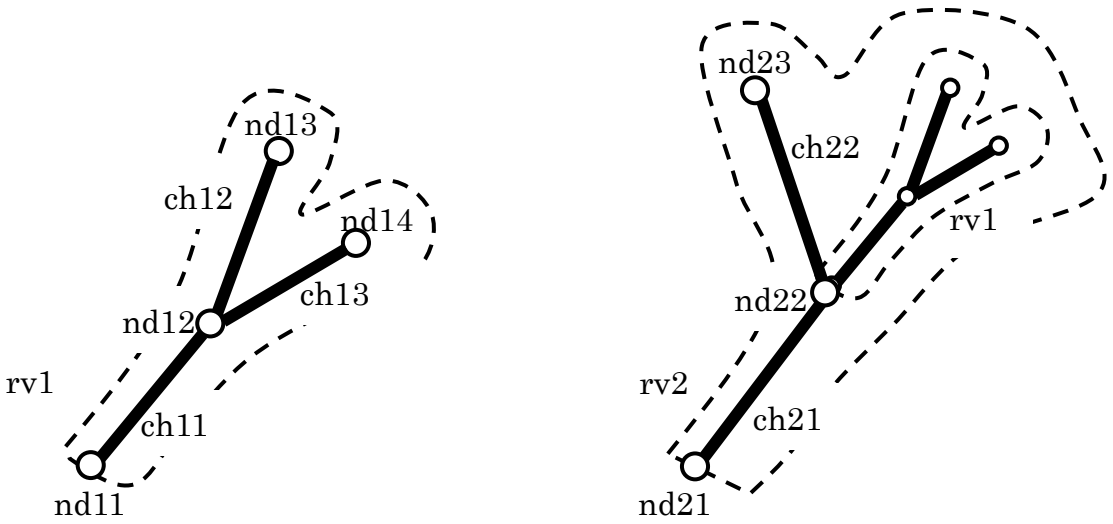
(c)RvSection 抽象クラスとその実装クラス

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 RvRectangularSection クラスをコーディングした。

(d) RvVisitor インターフェースの実装

ここでは、開水路網の構造を順にたどりながら，計算により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 RvOutCSV クラスをコーディングした。

３－３　プログラミング例



(a) Case1　　(b) Case2

図３－４　計算対象とした開水路網

本節では，前節で述べたクラスを利用して，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例を示す．ここでは**図３－４**に示す２つの開水路網を計算対象とする．すなわち

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。

図３－４には RvSys、RvChannel、RvNode それぞれのインスタンスの名称が示されている。ただし，ここでは簡単のため，RvChannel インスタンスは表３－２に示すように、勾配倍と断面形は各水路で一様としている．また表３－３には UpDischarge インスタンスそれぞれの仕様（記号は式(4)を参照）を示す.

表３―２　RvNonUniChannel インスタンスの仕様

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

※粗度係数はすべて 0.03 とした。

表３－３　UpDischarge インスタンスの仕様

名称	Q_b （m³/s）	Q_p（m³/s）	t_0 （s）	t_p （s）	C_p
nd13	60	600	9*60*60	12*60*60	20
nd14	40	400	9*60*60	12*60*60	20
nd23	100	1000	9*60*60	24*60*60	20

リスト３－１に、Case1 のプログラムの中核部分を示す。このリストではまず、下流端と合流点および２つの上流端の RvNode インスタンスを生成している。次に RvNonUniChannel インスタンス ch11、ch12、ch13 を生成してその上下流端の境界条件となる RvNode インスタンスを設定している。そして RvSys インスタンス rv1 を生成して、これにこれまで生成したインスタンスを収納している。これで計算自体の枠組みである RvSys インスタンスが完成する。

あとはこのインスタンス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 RvOutCSV インスタンス rvOut を生成し、init メソッドで rv1 全体の初期条件を設定しておく。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、calc メソッドで時間ステップを進める計算をしてから、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、rvOut を accept して CSV ファイルに出力する。

リスト３－１　Case1 のプログラムの中核部分（全リストは巻末付録参照）

```
DownFree nd11 = new DownFree(各データ);
DivCon nd12 = new DivCon(各データ);
UpDischarge nd13 = new UpDischarge(各データ);
UpDischarge nd14 = new UpDischarge(各データ);

RvNonUniChannel ch11 = new RvNonUniChannel(各データ);
ch11.setDownNode(nd11);
ch11.setUpNode(nd12);

RvNonUniChannel ch12 = new RvNonUniChannel(各データ);
```

```
ch12.setDownNode(nd12);
ch12.setUpNode(nd13);

RvNonUniChannel ch13 = new RvNonUniChannel(各データ);
ch13.setDownNode(nd12);
ch13.setUpNode(nd14);

RvSys rv1 = new RvSys(各データ);
rv1.add(ch11);
rv1.add(ch12);
rv1.add(ch13);
rv1.add(nd11);
rv1.add(nd12);
rv1.add(nd13);
rv1.add(nd14);



    t = 0;
    rv1.init(t, !initFlag);
    rv1.renew();
    rv1.accept(rvOut.setTime(t));
for (int i = 0; i < Nt; i++){
    t = i * dt;
    rv1.calc(t, dt);
    rv1.renew();
    if ((i+1)%Mt == 0){
        t = (i+1) * dt;
        rv1.accept(rvOut.setTime(t));
    }
}
```

リスト３－１を見ると，**図３－４**(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いることがわかる．前節のようなクラスをいったんコーディングしてしまえば、それを利用して開水路網を構成するプログラムをコーディングするのは，比較的直感的に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、RvSys インスタンスがまた別の RvSys インスタンスを収納できることを有効に活用したい。

リスト３－２は、Case２のプログラムで最上位の RvSys インスタンス rv2 を生成する部分を示したものである。

まず、rv1 以外の RvNode インスタンスと RvChannel インスタンスを生成させる。つぎに以前に作成した rv1 を生成させるコード(MyRvSys1 クラスの getRvSys メソッドとしておく)を用いて、河川網 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、rv1 の中の開水路 ch11 の下流端の境界条件を，新たに合流することになる nd22 と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 RvSys インスタンス rv2 に収納するだけである。

rv1 はそのままの形で rv2 に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。また rv1 を生成させるコードの中身はまったく変更する必要がないので，その詳細を知る必要はなく，再利用性が高い.

リスト３－２　Case2 において，rv1 を再利用して rv2 を生成するコード
（全リストは巻末付録参照）

```
//河川網の構成要素の生成
DownFree nd21 = new DownFree(各データ);
DivCon nd22 = new DivCon(各データ);
UpDischarge nd23 = new UpDischarge(各データ);

RvNonUniChannel ch21 = new RvNonUniChannel(各データ);
ch21.setDownNode(nd21);
ch21.setUpNode(nd22);

RvNonUniChannel ch22 = new RvNonUniChannel(各データ);
ch22.setUpNode(nd23);
ch22.setDownNode(nd22);

//既存の河川網生成クラスの再利用
RvSys rv1 = MyRvSys1.getRvSys();
//ここで生成している河川網との接続を設定する.
RvChannel rv1ch11 = (RvChannel)rv1.getRvComponent("rv1:ch11");
```

```
rv1.setDownNode(“nd22”, “rv1:ch11”);

//ここで生成すべき河川網に構成要素を追加する.
RvSys rv2 = new RvSys(各データ);
rv2.add(ch21);
rv2.add(ch22);
rv2.add(nd21);
rv2.add(nd22);
rv2.add(nd23);
rv2.add(rv1);
```

このような RvSys インスタンスの作り方は、単に以前作成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。

第 4 章

XML による河川網の記述と活用

前章では，河川網を入れ子状に構成し，不定流計算を再帰的に実施できるように，クラスの設計およびコーディングを行った．その結果，河川網の個々の構成要素インスタンスを生成して，それらの関連を設定し，不定流計算を実施するプログラムを，きわめて直感的に構築することができた．

ただしのままでは、具体的な開水路網を生成するためには，あくまでプログラムコードを書かなくてはならず，そのためには 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。

そこで本章では，XML の活用方法を検討する。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。さらにさまざまなプロジェクトにより，Java 言語で XML を扱う際のさまざまな作業を効率化できるようなプログラムが提供されている．本章では，こうしたプロジェクトの成果も活用して，簡便にアプリケーションを構築する手法も説明する．

４－１　XML による河川網の記述方式

本節では，開水路網の入れ子状の構成を，XML を用いて記述する手法について検討する．

（１）他の XML ファイルを引用する手法

XML にはファイルをリンクする機能がある．これはファイルの一部を別のファイル（またはその一部）と関連付けるものである．この機能を用いれば，河川網の入れ子状態を容易に記述することができる．

そのためのXMLの仕様としてXLink[10]がある．これはWebページを記述するHTML言語のリンク機能を拡張したものである (詳細は他書[1] を参照のこと)．ただし，XLinkは現時点では仕様の策定が熟してはおらず，その機能を十分に活用できる段階には至っていない．自然消滅の危機にあるという見方さえある[11]．XLinkはXMLファイルをはじめ，さまざまな種類のファイルとのリンクを設定するものでありながら，その仕様を実現する環境すなわち，XLinkに対応したXMLブラウザの普及もままならない状況である．

これに対して，XInclude[12]という仕様が実用・普及の段階に入っている．これは 2006 年 11 月に改訂仕様が確定したもので，Java言語でXMLを扱うための仕組みの中でもすでに対応が済んでいるものである．

XInclude を利用するには，利用対象となる要素範囲の属性として，名前空間「http://www.w3.org/2001/XInclude」を設定する必要がある．そのうえで「inclde タグ」とその「href 属性」でリンク対象の XML ファイル（あるいはその一部）を指定する．たとえば，2 つのファイル「sample1.xml」と「sample2.xml」の内容が，

sample1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns:xi="http://www.w3.org/2001/XInclude">
    <xi:include href="sample2.xml"/>
</components>
```

sample2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<component name="foo1">
    "Foo1"
</component>
```

であった場合，XML 処理系には，

```
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns:xi="http://www.w3.org/2001/XInclude">
    <component name="foo1">
        "Foo1"
    </component>
</components>
```

と同じものとして処理される．

XIncludeではファイル全体でなく，ファイルの一部を組み込むことも可能である．その場合には，href属性においてファイル名に加えてXPointer[13]などで範囲指定も行う必要がある．

（2）XML による河川網の記述様式

前節で検討したように，河川網の入れ子構成を記述するには，XInclude を活用すれば必要十分である．そこで，次の要領で河川網を XML を用いて記述することにする．

(1) 河川網を示す xml ファイルのルート要素は<RvSys>要素とする．
(2) <RvSys>要素内には，必ず以下の 3 つの要素を配する．
  - <name>要素：河川網の名称
  - <RvComponents>要素：河川網の構成要素
  - <RvConnections>要素：河川網の構成要素の接続状況
(3) <RvComponents>要素内には，以下のいずれかの要素を配して，それぞれの構成要素の特性を記述する．
  - <RvDownFree>要素
  - <RvUpDischarge>要素
  - <RvDivCon>要素
  - <RvNonUniChannel>要素
  - <RvSys>要素
(4) <RvDownFree>，<RvUpDischarge>，<RvDivCon>要素では，必ず name 属性にて境界点の名称を定義する．
(5) さらに<RvUpDischarge>要素では，必ず qb，qp，to，tp，cp 属性にて，それぞれQ_b，Q_p，t_0，t_p，C_pの値を設定する．
(6) <RvNonUniChannel>要素内には，必ず以下の２つの要素を配する．
  - <name>要素：開水路の名称
  - <sections>要素：開水路を構成する断面ごとのデータ
(7) <sections>要素の内容としては，本研究では前章にて RvRectangularSection クラスをコーディングしているので，<RvRectangularSection>要素を必要な数だけ（実際の断面の文だけ）配するものとする．
(8) <RvRectangularSection>要素では，name，x，bedLevel，width，rugh 属性にて，それぞれ断面名称，流下距離，河床高，水路幅，粗度係数を設定する．

(9) <RvConnections>要素には，<RvComponents>要素内に書かれた開水路と開水路網（のなかの開水路）に接続する境界点を指定するために，以下の要素を配する．

・ <DownNode>要素：下流端に設定する境界点を指定する．
・ <UpNode>要素：上流端に設定する境界点を指定する．

(10) <DownNode>要素と<UpNode>要素では，chName 属性にて開水路名称を指定し，ndName 属性にて境界点名称を指定する．

以上のような規則を記述するには，文章として記述するだけでなく，XML関連の仕様として決められているものもある．代表的な仕様としてはDTD (Document Type Definition)[6]，XML Schema[6]，RELAX NG[14]などがある．こうした汎用的な仕様に準じて規則を記述しておけば，実際に河川網を記述した文書に誤りがないか，XMLファイルの攻勢を自動的にチェックすることができる．ただし，それぞれの方式に一長一短があるので，本研究では，上述のような文章による記述で済ますことにした．

（3）XML による河川網の記述例

前章の図３－４の Case2 にあるような河川網 rv2 を記述する xml は以下のようになる．

rv2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<RvSys xmlns:xi="http://www.w3.org/2001/XInclude">
    <name>rv2</name>

    <RvComponents>
        <RvDownFree name="nd21" />
        <RvDivCon name="nd22" />
        <RvUpDischarge name="nd23" qb="100" qp="1000"
            t0="32400" tp="86400" cp="20" />

        <RvNonUniChannel>
            <name>ch21</name>
```

```
            <xi:include href="ch21_sections.xml" />
        </RvNonUniChannel>

        <RvNonUniChannel>
            <name>ch22</name>
            <xi:include href="ch22_sections.xml" />
        </RvNonUniChannel>

        <xi:include href="rv1.xml" />
    </RvComponents>

    <RvConnections>
        <DownNode chName="ch21" ndName="nd21" />
        <UpNode   chName="ch21" ndName="nd22" />

        <DownNode chName="ch22" ndName="nd22" />
        <UpNode   chName="ch22" ndName="nd23" />

        <DownNode chName="rv1:ch11" ndName="nd22" />
    </RvConnections>

</RvSys>
```

ここで，rv1.xml は以下のようになる．

rv1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<RvSys xmlns:xi="http://www.w3.org/2001/XInclude">
    <name>rv1</name>

    <RvComponents>
```

```
    <RvDownFree name="nd11" />
    <RvDivCon name="nd12" />
    <RvUpDischarge name="nd13" qb="60" qp="600"
       t0="32400" tp="64800" cp="20" />
    <RvUpDischarge name="nd14" qb="40" qp="400"
       t0="32400" tp="64800" cp="20" />

    <RvNonUniChannel>
       <name>ch11</name>
       <xi:include href="ch11_sections.xml" />
    </RvNonUniChannel>

    <RvNonUniChannel>
       <name>ch12</name>
       <xi:include href="ch12_sections.xml" />
    </RvNonUniChannel>

    <RvNonUniChannel>
       <name>ch13</name>
       <xi:include href="ch13_sections.xml" />
    </RvNonUniChannel>
</RvComponents>

<RvConnections>
    <DownNode chName="ch11" ndName="nd11" />
    <UpNode   chName="ch11" ndName="nd12" />

    <DownNode chName="ch12" ndName="nd12" />
    <UpNode   chName="ch12" ndName="nd13" />

    <DownNode chName="ch13" ndName="nd12" />
```

```
        <UpNode   chName="ch13" ndName="nd14" />
    </RvConnections>

</RvSys>
```

また「ch○○_sections.xml」というファイルにはそれぞれの開水路の断面のデータが，以下のような形式で記述されている．

ch○○_sections.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<sections>
    <RvRectangularSection name="ST40" x="0"
        bedLevel="50.000" width="200" rough="0.03" />
  . . . . .
  . . . . . .
<sections>
```

４－２　XML ファイルからの河川網インスタンスの生成

（１）Jakarta Commons Digester の活用

前節で規定されたルールにしたがって，系統的に河川網の特性を記述することができるようになった．つぎは，この XML ファイルにしたがって，河川網インスタンスを Java プログラム上で自動的に生成するような仕組みを用意する必要がある．

これには，本報告書で再三再四述べているように，さまざまなプロジェクトがあり，Java 言語上で XML を簡便に扱えるようになっている．

もっとも基本的なものは DOM（Document Object Model）と SAX（Simple API for XML）であろう．DOM は XML ファイルに記述された木構造をそのままインスタンス化したものである．DOM の関連クラスを用いることにより，Java 言語上で，XML ファイルの隅々まで，データを参照することができる．SAX は XML ファイルを頭から順に読み込みながら，要素の開始や終了を見つけたりテキストが出現したりしたときに，その状況に応じて，さまざまな処理ができるようにしたイベントドリブン形式の処理系を構築するものである．

いずれにしても，DOM や SAX をそのまま用いて，XML ファイルのデータから河川網インスタンスを自動的に生成するようなプログラムを最初からコーディングするのは極めて困難である．ただし，こうしたニーズは非常に高く，実際，いくつかのプロジェクトや仕様（とその実装）によって，簡便に作業できる環境が整ってきている．

XMLファイルとインスタンスとの相互変換はデータバインディング技術と呼ばれ，いくつかの方式が提案されている[7)15)16)]．本研究ではその中でもJakarta Commonsプロジェクト[17)]のサブプロジェクトであるDigester[18)]とBetwikt[19)]の成果を活用して，アプリケーションを構築することにする．DigesterはXMLファイルからインスタンスを自動生成するクラスを提供し，Betwixtは単純なXML→インスタンス変換と簡便なインスタンス→XML変換のツールを提供する．

本研究では，河川網を記述した XML ファイルから河川網インスタンスを生成するには Digester を活用し，計算結果を XML ファイルとして出力するには Betwixt を使うことにする．

（2）Digester ルールの作成

Digester では，前節で示したような XML ファイルからインスタンスを生成する規則を，やはり XML ファイルで記述する．本研究で扱う河川網 XML ファイルを河川網インスタンスに変換するためのルールは，以下のように書くことができる．

RuleOfRvSys.xml：河川網インスタンスを生成させるルール

```
<?xml version="1.0" encoding="UTF-8"?>
<digester-rules>

   <pattern value="RvSys">
      <object-create-rule
         classname="RvSys" />
      <bean-property-setter-rule pattern="name" />
   </pattern>

   <pattern value="*/RvComponents">

      <pattern value="RvSys">
         <object-create-rule classname="RvSys" />
         <bean-property-setter-rule pattern="name" />
         <set-next-rule methodname="add" />
      </pattern>

      <pattern value="RvUpDischarge">
         <object-create-rule classname="RvUpDischarge" />
         <set-properties-rule />
         <set-next-rule methodname="add" />
      </pattern>
```

```
    <pattern value="RvDownFree">
       <object-create-rule classname="RvDownFree" />
       <set-properties-rule />
       <set-next-rule methodname="add" />
    </pattern>

    <pattern value="RvDivCon">
       <object-create-rule classname="RvDivCon" />
       <set-properties-rule />
       <set-next-rule methodname="add" />
    </pattern>

    <pattern value="RvNonUniChannel">
       <object-create-rule classname="RvNonUniChannel" />
       <bean-property-setter-rule pattern="name" />
       <pattern value="sections">
          <pattern value="RvRectangularSection">
             <object-create-rule
                classname=" RvRectangularSection" />
             <set-properties-rule />
             <set-next-rule methodname="addSection" />
          </pattern>
       </pattern>
       <set-next-rule methodname="add" />
    </pattern>

</pattern>
```

```
    <pattern value="*/RvConnections">

        <pattern value="UpNode">
            <call-method-rule methodname="setUpNode"
                paramcount="2" />
            <call-param-rule paramnumber="0" attrname="ndName" />
            <call-param-rule paramnumber="1" attrname="chName" />
        </pattern>

        <pattern value="DownNode">
            <call-method-rule methodname="setDownNode"
                paramcount="2" />
            <call-param-rule paramnumber="0" attrname="ndName" />
            <call-param-rule paramnumber="1" attrname="chName" />
        </pattern>

    </pattern>

</digester-rules>
```

（3）Digester を用いた河川網インスタンスの生成

リスト４－１は, Digester を用いて, 前章の Case2 の河川網インスタンスを生成させるコードの抜粋である.

リスト４－１：河川網インスタンスを生成させるプログラムコードの例

```
import java.io.FileInputStream;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.apache.commons.digester.Digester;
import org.apache.commons.digester.RuleSet;
import org.apache.commons.digester.xmlrules.FromXmlRuleSet;
import org.xml.sax.InputSource;

SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setNamespaceAware(true);
factory.setXIncludeAware(true);
SAXParser parser = factory.newSAXParser();

Digester digester = new Digester(parser);

InputSource rulesSource
 = new InputSource(new FileInputStream(“RuleOfRvSys.xml”));
RuleSet rulesSet = new FromXmlRuleSet(rulesSource);
digester.addRuleSet(rulesSet);

InputSource dataSource
 = new InputSource(new FileInputStream(“rv2.xml”));
rv = (RvSys)digester.parse(dataSource);
```

リスト４－１では, しかるべきパッケージのクラスをインポートしたうえで, まず，SAX で XML ファイルを読み込むための parser を生成している．この

ときに，名前空間や XInclude をサポートして動作するように設定しておく．つぎに，この parser を用いて XML ファイルを処理する Digester インスタンス（変数 digester）を生成する．これによって，Digester は XInclude でまとめられたファイルを，ひと続きのファイルとして処理してくれることになる．

また，今回の河川網 XML ファイルを河川網インスタンスに変換するルールを digester に設定する．このためには，変換ルールを記述した XML ファイル（”RuleOfRvSys.xml”）を FromXmlRuleSet クラスに渡して，RuleSet インスタンスを生成し，これを digester に変換ルールとして追加する．

そのうえで，digester に河川網 XML ファイル（“rv2.xml”）を読み込ませれば，所望のインスタンスが生成される．ただし，digester が生成するインスタンスは Object 型になっているので，これを RvSys クラスへキャストするのを忘れてはならない．

以上のように， XML ファイルでデータを記述し，そのデータからインスタンスを生成するルールを XML で記述してやれば，それらを用いてインスタンスを生成するのは，いつも決まった手順となる．この手続きをたとえば「RvSysFactory」などと名づけたクラスに記述しておけば，非常に便利なことになろう．

たとえば，河川網の構成が変更された場合には，河川網 XML ファイルを修正するだけでよい．また，河川網の構成要素のオプションが追加された場合には，そのオプションに関するクラスのコーディングのみを行い，XML 変換ルールにそのことを書き加えれば済む．プログラムの中核部分に「IF 文」を追加したり，「RvSysFactory」を書き換える必要はない．

アプリケーションのユーザー側から見れば，河川網の解析を行いたいときは，もはや Java 言語を習得する必要はない．極めて初歩的な XML の文法さえ知っておれば，河川網 XML ファイルを記述することは十分可能である．

さらに河川網のデータが XML で記述されていることにより，このデータそのものをメンテナンスするアプリケーションの構築も，見通しがよいものになろう．XML によりデータを記述し，これをプログラムから独立させることで，可能性はさまざまに広がっていく．

４－３　不定流計算結果の XML ファイルへの出力

（１）Betwixt の XML マッピングファイル

Betwixt プロジェクトのツールを利用すると，JavaBean から XML，XML から JavaBean への変換を容易に行えるようになる．後者についてきめ細かなルールが必要な場合には Digester を利用することになるが, 単純なルールであるならば，Betwixt も利用の候補に挙がる．また前者については，Digester では対応していないので，Betwixt に頼ることになる．

Betwixt のツールは, 対象とする JavaBean のコードを解析して, JavaBean が有するデータを自動的に適切な XML 形式にして出力したり, XML ファイルから適切なデータを JavaBean に読み込ませたりすることができる．

ただし，「Betwixt にすべてお任せ」では，必要ないデータまで出力したり，所望の出力フォーマットでなかったり，多少不便な思いをすることがある．

そのような場合，Betwixt では，マッピングファイルを用いて XML 書式のカスタマイズができるようになっている．マッピングファイルは「出力対象のクラス名.betwixt」というファイル名にして，クラスパスに置けばよいとされている．

本研究における河川網とその構成要素のマッピングファイルは，たとえば次のようなものが考えられる．

RvSys.betwxt

```
<?xml version="1.0" encoding="UTF-8"?>
<info primitiveTypes="element">
   <element name="RvSYs">
      <attribute name="name" property="fullName" />

      <element name="RvComponents">
         <element property="rvComponents" />
      </element>

   </element>
</info>
```

RvNonUniChannel.betwxt

```
<?xml version="1.0" encoding="UTF-8"?>
<info primitiveTypes="element">
   <element name="RvNonUniChannel">
      <attribute name="name" property="fullName" />

      <element name="sections">
         <element property="section"/>
      </element>
   </element>
</info>
```

RvRectangularSection.betwxt

```
<?xml version="1.0" encoding="UTF-8"?>
<info primitiveTypes="element">
   <element name="Section">
      <attribute name="name" property="name" />
      <attribute name="X" property="x" />
      <attribute name="Zb" property="averageBedLevel" />
      <attribute name="Zs" property="waterLevel" />
      <attribute name="H" property="depth" />
      <attribute name="A" property="area" />
      <attribute name="Q" property="discharge" />
      <attribute name="V" property="velocity" />
   </element>
</info>
```

RvDownFree.betwxt ， RvUpDischarge.betwxt ， RvDivCon.betwxt

```
<?xml version="1.0" encoding="UTF-8"?>
<info primitiveTypes="element">
</info>
```

これらのマッピングによれば，RvSys インスタンスの構成要素のデータを XML 形式で順に出力していく際には，RvNonUniChannel のデータを RvNonUniChannel.betwxt や RvRectangularSection.betwxt に記述されたフ

ォーマットで出力し，RvDownFree.，RvUpDischarge，RvDivCon インスタンスについては何も出力しない，極めてシンプルな出力形式となる．

ただしこれでは，計算時刻は出力されないままである．出力されている計算結果が，時刻 t がいくつのときのものなのか，出力結果ファイルに明示されなくては意味がない．とはいえ，以上に挙げたクラスには，もともと時刻を収納する変数がないので，当然といえば当然である．

これを解決するには 2 通りの方法がある．一つには，これまでのクラスに時刻を収納する変数とそれにアクセスするメソッドを新たに追加することである．たとえばそれを最上位の RvComponent クラスにコーディングすれば，そのすべてのサブクラスで，時刻出力の問題は解決する．

もう一つの方法は, 時刻用の変数と最上位の RvSys インスタンスをフィールドにもつ新しいクラスをコーディングすることである．この方法では，もともとのクラスには変更を行わないで済むことが, 利点となっている. 将来的にも，さまざまな付加情報を出力する状況が予想される場合には，そのつど中核的なクラスを変更するよりは，オプションとして新たなクラスでラッピングすることも，ひとつの選択肢といえる．このような処置の仕方は，オブジェクト指向プログラミングの「デザインパターン」でいう「Bridge パターン」の一種といえる．

本研究においては後者の方法をとることにする．具体的には，時刻用の変数と最上位の RvSys インスタンスを合わせて計算結果とするために, 次のような RvOutXMLSysResult クラスを用いる．

リスト４－２：計算結果出力のための RvOutXMLSysResult クラス

```
public class RvOutXMLSysResult {
    private double time;
    private RvSys rvSys;

    public void setTime(double time) {this.time = time;}
    public double getTime() {return time;}
    public double getHour() {return time/3600;}

    public void setRvSys(RvSys rv) {this.rvSys = rv;}
    public RvSys getRvSys() {return rvSys;}
}
```

そして，このクラスに対する Betwixt のマッピングファイルを，

RvOutXMLSysResult.betwxt

```
<?xml version="1.0" encoding="UTF-8"?>
<info primitiveTypes="element">
   <element name="Result">
      <attribute name="time" property="time" />
      <attribute name="hour" property="hour" />
      <addDefaults />
   </element>
</info>
```

のようにすれば，秒単位表示および時間単位表示の時刻を属性とした<Result>要素が出力される．この<Result>要素のなかに，その時刻での RvSys インスタンスのデータすべてがマッピングされた通りに出力されることになる．

（2）Betwixt を用いた計算結果の XML 形式での出力

本研究では，計算結果を XML 形式で出力するための RvOutXMLSys クラスをコーディングした．その中核部分をリスト４－３に示す．

リスト４－３：計算結果を出力する RvOutXMLSys クラス

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.PrintWriter;
import org.apache.commons.betwixt.io.BeanWriter;

public class RvOutXMLSys {

   private String fileName;
   private PrintWriter printWriter;
   private FileWriter fileWriter;
   private BeanWriter beanWriter;
   private RvOutXMLSysResult result = new RvOutXMLSysResult();
```

```
    private void openFile(){
        fileWriter = new FileWriter(fileName);
        printWriter = new PrintWriter(new BufferedWriter(fileWriter));
        printWriter.println("<?xml version='1.0' encoding='UTF-8'?>");
        printWriter.println("<Results>");

        beanWriter = new BeanWriter(printWriter);
        beanWriter.getBindingConfiguration().setMapIDs(false);
        beanWriter.enablePrettyPrint();
        beanWriter.setIndent("¥t");
    }

    public void write(double t, RvSys rv) {
        result.setTime(t);
        result.setRvSys(rv);
     beanWriter.write(result);
    }

    public void closeFile(){
        printWriter.println("</Results>");
        printWriter.close();
    }
}
```

この RvOutXMLSys クラスでは，openFile メソッドにおいて所定のファイルを開き，XML 宣言と<Results>要素の開始タグを書き込む．<Results>要素は，この XML ファイルのルート要素であり，各時刻の出力結果である<Result>要素（これは一般に複数個出力される）を包み込むものである．このメソッドではさらに，Betwixt の BeanWriter インスタンスである beanWriter を生成させて，主種の設定を行っている．

計算をしながらの出力は RvOutXMLSys クラスの write メソッドを用いる．このメソッドでは，RvOutXMLSysResult インスタンスの result に時刻と RvSys インスタンスをセットして，それを beanWriter の write メソッドに渡して，<Result>要素を出力させる．

計算が終了した段階で, RvOutXMLSys クラスの closeFile メソッドをよび, <Results>要素の終了タグを書き込むとともに，ファイルを閉じる.

以上のクラスを用いて, 前章の Case2 の計算を行うプログラム例をリスト４－４に示す. これら RvOutXMLSys および Case2XML のリストをみてわかるように，河川網の入れ子状態のデータを出力するのに，もはや Visitor パターンを利用していない．これは，Betwixt の BeanWriter インスタンスの出力をコントロールするマッピングファイルに，すでに河川網構成要素の再帰的構造が織り込まれているためである.

リスト４－４：XML を活用した不定流計算の例

```
import jp.ac.uuwem.rvsys.xml.RvOutCSV;
import jp.ac.uuwem.rvsys.xml.RvOutXML;
import jp.ac.uuwem.rvsys.xml.RvSys;
import jp.ac.uuwem.rvsys.xml.RvSysFactory;

public class Case2XML {

   public static void main(String[] args) {

      RvSys rv = RvSysFactory.createRvSys("rv2.xml", "RuleOfRvSys.xml");
      RvOutXMLSys outXML = new RvOutXMLSys();
      outXML.setFileName(“case2.xml”);
      outXML.openFile ();

      double dt = 60;
      int Nt = 1 * 60 * 48;
      int Mt = 1 * 60 * 3;

      double t = 0;
      rv.init(t, false);
      rv.renew();
      outXML.write(t, rv);

      for (int i = 0; i < Nt; i++){
         t = i * dt;
```

```
            rv.calc(t, dt);
            rv.renew();
            if ((i+1)%Mt == 0){
                t = (i+1) * dt;
                outXML.write(t, rv);
            }
        }

        outXML.closeFile();
    }

}
```

また，これによって出力された XML ファイルの例（抜粋）を以下に示す.

Case2.xml（抜粋）

```
<?xml version='1.0' encoding='UTF-8'?>
<Results>
    <Result time="0.0" hour="0.0">
        <RvSys name="rv2:rv2">
            <RvComponents>
                <RvSYs name="rv2:rv1">
                    <RvComponents>
                        <RvNonUniChannel name="rv1:ch11">
                            <sections>
                                <Section name="ST30" X=・・・Zs=・・・Q=・・・/>
                                <Section name="ST29" X=・・・Zs=・・・Q=・・・/>
                                ・・・・・・
                            </sections>
                        </RvNonUniChannel>
                        <RvNonUniChannel name="rv1:ch12">
                                ・・・・・・
                                ・・・・・・
             </RvNonUniChannel>
             <RvNonUniChannel name="rv1:ch13">
```

```
                ・・・・・・
            ・・・・・・
            </RvNonUniChannel>
                 </RvComponents>
              </RvSYs>

        <RvNonUniChannel name="rv2:ch21">
                 <sections>
                    <Section name="ST60" X=・・・Zs=・・・Q=・・・/>
                    <Section name="ST59" X=・・・Zs=・・・Q=・・・/>
                      ・・・・・・
                 </sections>
              </RvNonUniChannel>
              <RvNonUniChannel name="rv2:ch22">
                     ・・・・・・
                     ・・・・・・
              </RvNonUniChannel>
           </RvComponents>
        </RvSys>
    </Result>

    <Result time="10800.0" hour="3.0">
           ・・・・・・
           ・・・・・・
           ・・・・・・
    </Result>
       ・・・・・・
       ・・・・・・
       ・・・・・・
</Results>
```

この XML ファイルの内容を見ると，どのデータが何を表しているか，非常にわかりやすい．また，この XML ファイルをもとにして，計算結果を収めた RvOutXMLSysResult インスタンスの集合体を生成することも可能なことに気づく．そうすると，河川網の入れ子構造をたどりながら，計算結果を取り扱うアプリケーションを容易に構築することができる．

第５章

総　括

５－１　結論

本研究で得られた知見は以下の通りである.

（１）河川網を小規模な河川網の入れ子構造として構成するために，オブジェクト指向プログラミング（以下，OOP）のデザインパターンにおける Composite パターンが有効であることがわかった.

（２）また開水路網の入れ子構造を順に巡りながら，さまざまな処理を行う場合には，デザインパターンの Visitor パターンを活用すればよいことがわかった.

（３）OOP の継承と多態性を活用して, 河川網の不定流計算プログラムが簡潔に直感的にコーディングできることを示した．すなわち開水路インスタンスを生成して，その上下流端に境界条件としての境界点インスタンスをセットし，それらを開水路網の構成要素として加えていけばよい.

（４）しかも小規模な開水路網を生成するコードを手直しせずに，より大きな開水路網に組み込むことができるので，プログラムコードの再利用あるいは段階的で効率的な開水路網の構築が可能であることを示した.

（５）XML により河川網の入れ子構造を記述するには，XInclude を用いると見通しがよくなることを示した．そして本研究で扱う開水路網を記述する XML 規則を示した.

（６）Jakarta Commons Digester を活用することにより，XML ファイルから河川網インスタンスを生成するプログラムを簡潔にコーディングできることを示した.

（７）河川網の構成要素のデータとそのデータからインスタンスを生成するルールを，ともに XML 形式のファイルとして，プログラム本体から独立させることで，アプリケーションのメンテナンスやオプションの追加が容易になることを示した.

（８）Jakarta Commons Betwixt を活用することにより，河川網の不定流の計算結果を XML 形式で出力するプログラムを簡潔にコーディングできることを示した.

（９）計算結果の出力形式をマッピングファイルを用いてカスタマイズすることにより, 河川網の入れ子構造を反映させながら, 必要なデータを XML ファイルに出力することが出来た．これにより，計算結果をもとに，もとの河川網と同じ入れ子構造を有する新たなインスタンスを生成して，さまざまな処理を実行するアプリケーションを容易に構築できるであろうことを考察した.

５－２　今後の課題

本研究では，開水路網での不定流計算を扱ったが，他の計算処理にも本研究の手法が有効であろうことは予想できる．ただし，その場合に，さまざまな機能をどのような形で追加・実装していけば，メンテナンスや再利用性が向上するのか，たとえばデザインパターンでいう Bridge パターンをうまく適用することも視野に入れて，検討していくべきであろう．

また，本研究で示した開水路網の記述ルールは，ここだけのローカルなものであり，今後，実用面での発展を期するためには，このようなデータ記述形式を，さまざまな局面で活用できるようにしていかなければならない．

たとえば河川関連データについて，既存の XML 記述規則がすでにあるならば，それとの整合性を取りながら，データ活用の幅を広げていくのが肝要かと思われる．

既存の規則としては，財団法人 河川情報センターが 2004 年に策定した「統一河川情報システム ＸＭＬスキーマ定義書」[20]が挙げられる．これはさまざまな水系，観測所位置，観測データなどを，XML形式によって統一的に記述しようというものである．

今後は，こうしたより影響力の大きなルールに則って，データを受け取り，データを発信していくことが求められていくものといえる．

参考文献

1） たとえば，池田 実・小野寺 尚希：まるごと図解 最新 XML がわかる，技術評論社，269ｐ.，2000.

2） たとえば，藤田 一郎：まるごと図解 最新 Java がわかる，技術評論社，270ｐ.，1999.

3） 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.

4） たとえば，高橋麻奈：やさしい Java （第３版），ソフトバンククリエイティブ，576p.，2005.

5） たとえば，結城浩：増補改訂版 Java 言語で学ぶデザインパターン入門，ソフトバンククリエイティブ，484p.，2004.

6） たとえば，高橋麻奈：やさしい XML（第２版），ソフトバンククリエイティブ，469p.，2005.

7） Brett McLaughlin （須賀 祐治・寺田 美穂子　訳）：Java&XML 第 2 版，オライリー・ジャパン，551p.，2002.

8） 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.

9） 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.

10） XML Linking Language (XLink) Version 1.0, W3C Recommendation 27 June 2001.（ http://www.w3.org/TR/2001/REC-xlink-20010627/ ）

11） たとえば，村田 真：XLink の死，http://ch05250.kitaguni.tv/e132904.html（2007 年 8 月 31 日現在）.

12） XML Inclusions (XInclude) Version 1.0 (Second Edition), W3C Recommendation 15 November 2006.

（ http://www.w3.org/TR/2006/REC-xinclude-20061115/ ）

13） XPointer Framework, W3C Recommendation 25 March 2003.（ http://www.w3.org/TR/2003/REC-xptr-framework-20030325/ ）

14） たとえば，http://www.relaxng.org/ .

15） たとえば，上川伸彦・寺嶋剛：Java プログラマーに贈る XML 超入門，JavaWorld，August，pp.181－198，2004.

16） T. M. O’Brien（株式会社テクノロジックアート　訳）: Jakarta Commons クックブック，オライリー・ジャパン，399p.，2005.

17） http://commons.apache.org/

18） http://commons.apache.org/digester/

19） http://commons.apache.org/betwixt/

20） 財団法人 河川情報センター: 統一河川情報システム ＸＭＬスキーマ定義書 （ Ver.1.1 ）， 2004 .（ た と え ば， http://www3.river.go.jp/IDC/guideline/RiverXMLschema/XMLschema.pdf ）

付　録

主要なプログラムコード

```java
// 河川網構成要素の基底クラス
package jp.ac.uuwem.rvsys.xml;

public abstract class RvComponent implements Comparable<RvComponent> {

    protected final double gr=9.8;
    protected int calcOrder = 0;
    protected RvSys rvSys;
    protected String name;
    protected boolean initFlag=true;

    public RvComponent(RvSys rv, String n, int o){
        this.rvSys = rv;
        this.name = n;
        this.calcOrder = o;
    }

    public int compareTo(RvComponent another) {
        return  this.calcOrder - another.getCalcOrder();
    }

    public void setRvSys(RvSys rv) {this.rvSys = rv;}
    public void setName(String name) {this.name = name;}
    public RvSys getRvSys() {return rvSys;}
    public String getName() {return name;}
    public int getCalcOrder() {return calcOrder;}

    public String getFullName() {return getFullName(this.name);}
    public String getFullName(String name) {
        if (name.contains(":")){
            return name;
        }else{
            return rvSys.getName() + ":" + name;
        }
    }

    public boolean init(double initTime, boolean initFlag){
        if (this.initFlag != initFlag){
            this.initFlag = initFlag;
            init(initTime);
        }
        return initFlag;
    }
    protected abstract void init(double initTime);

    public abstract RvComponent getRvComponent(String fullName);
    public abstract void calc(double nowTime, double dt);
    public abstract void renew();
    public abstract void accept(RvVisitor v);
}
```

```
// 河川網クラス
package jp.ac.uuwem.rvsys.xml;

import java.util.ArrayList;
import java.util.Collections;

public class RvSys extends RvComponent {

	private ArrayList<RvComponent>
		rvComponents = new ArrayList<RvComponent>();

	public RvSys(RvSys rv, String name){super(rv, name, 1);}
	public RvSys(String name){this(null, name); this.rvSys = this;}
	public RvSys(){this(null, null); this.rvSys = this;}

	public void add(RvComponent rc){
		rvComponents.add(rc);
		rc.setRvSys(this);
	}

	public void remove(RvComponent rc){rvComponents.remove(rc);}

	public void remove(String fullName){
		RvComponent rc = getRvComponent(fullName);
		if (rc == null) return;
		rvComponents.remove(rc);
	}

	public ArrayList<RvComponent> getRvComponents(){return rvComponents;}

	public RvComponent getRvComponent(String fullName){
		if ((this.getFullName()).equals(fullName)) return this;

		RvComponent rc;
		for (RvComponent c: rvComponents){
			rc = c.getRvComponent(fullName);
			if (rc != null) return rc;
		}

		return null;
	}

	public void setUpNode(String ndName, String chName){
		//RvNode and RvChannel belong to this RvSys or below.
		RvChannel ch = (RvChannel)getRvComponent(getFullName(chName));
		RvNode nd = (RvNode)getRvComponent(getFullName(ndName));
		ch.setUpNode(nd);
	}
```

```
    public void setDownNode(String ndName, String chName){
     //RvNode and RvChannel belong to this RvSys or below.
     RvChannel ch = (RvChannel)getRvComponent(getFullName(chName));
     RvNode nd = (RvNode)getRvComponent(getFullName(ndName));
     ch.setDownNode(nd);
     }

     @Override
     public void calc(double nowTime, double dt) {
          for (RvComponent rc: rvComponents){
               rc.calc(nowTime, dt);
          }
     }

     @Override
     protected void init(double t) {
          Collections.sort(rvComponents);
          for (RvComponent rc: rvComponents){
               rc.init(t, initFlag);
          }
     }

     @Override
     public void renew() {
          for (RvComponent rc:rvComponents){
               rc.renew();
          }
     }

     @Override
     public void accept(RvVisitor v) {v.visit(this);}

}
```

```
// 水路計算の基底クラス
package jp.ac.uuwem.rvsys.xml;

public abstract class RvChannel extends RvComponent {

	protected RvNode downNode = null;
	protected RvNode upNode = null;
	protected int Nx;
	protected double[] X, Zmin, Zavg;
	protected double[] Zs, H, A, Q, V;

	public RvChannel(RvSys rv, String name){super(rv, name, 2);}
	public RvChannel(String name){this(null, name);}
	public RvChannel(){this(null, null);}
	public RvChannel(RvSys rv, String name, int Nx){
		this(rv, name);
		this.Nx = Nx;
		initBaseArrays();
	}

	protected void initBaseArrays(){
		X =new double[Nx+1];
		Zmin =new double[Nx+1];
		Zavg =new double[Nx+1];
		Zs =new double[Nx+1];
		H =new double[Nx+1];
		A =new double[Nx+1];
		Q =new double[Nx+1];
		V =new double[Nx+1];
	}

	public int getNx() {return Nx;}
	public double[] getX() {return X;}
	public double[] getZs() {return Zs;}
	public double[] getZmin() {return Zmin;}
	public double[] getZavg() {return Zavg;}
	public double[] getH() {return H;}
	public double[] getA() {return A;}
	public double[] getQ() {return Q;}
	public double[] getV() {return V;}

	public RvNode getDownNode(){return downNode;}
	public RvNode getUpNode(){return upNode;}

	public void setDownNode(RvNode downNode){
		if (this.downNode != null) removeDownNode();
		this.downNode = downNode;
		downNode.setUpChannel(this);
	}
```

```
    public void setUpNode(RvNode upNode){
        if (this.upNode != null) removeUpNode();
        this.upNode = upNode;
        upNode.setDownChannel(this);
    }

    public void removeDownNode(){
        downNode.removeUpChannel(this);
        this.downNode = null;
    }
    public void removeUpNode(){
        upNode.removeDownChannel(this);
        this.upNode = null;
    }

    @Override
    public void accept(RvVisitor v) {v.visit(this);}

    @Override
    public RvComponent getRvComponent(String fullName) {
        if ((this.getFullName()).equals(fullName)) return this;
        return null;
    }
}
```

```
// 境界条件設定の基底クラス
package jp.ac.uuwem.rvsys.xml;

public abstract class RvNode extends RvComponent {

	public RvNode(RvSys rv, String name){super(rv, name, 3);}
	public RvNode(String name){this(null, name);}
	public RvNode(){this(null, null);}

	public abstract void setDownChannel(RvChannel ch);
	public abstract void setUpChannel(RvChannel ch);
	public abstract void removeDownChannel(RvChannel ch);
	public abstract void removeUpChannel(RvChannel ch);

	public RvChannel getDownChannel(){return null;}
	public RvChannel getUpChannel(){return null;}

	@Override
	public void renew() {}

	@Override
	public RvComponent getRvComponent(String fullName) {
		if ((this.getFullName()).equals(fullName)) return this;
		return null;
	}

	@Override
	public void accept(RvVisitor v) {v.visit(this);}
}
```

```
// 断面特性の基底クラス
package jp.ac.uuwem.rvsys.xml;

import static java.lang.Math.signum;;

public abstract class RvSection    implements Comparable<RvSection> {

	protected double waterLevel=0, minimumBedLevel=0, averageBedLevel=0;
	protected double depth=0, area=0, waterWidth=0, hydraulicRadius=0;
	protected double discharge=0, velocity=0, fricCoeff=0;
	protected double x=0;
	protected String name=null;

	protected abstract void calcSectionByH();
	protected abstract void calcSectionByA();
	protected abstract void calcSectionByQ();
	protected abstract void calcSectionByAQ();
	protected abstract void calcSectionByHQ();
	public abstract void calcUniByQ(double Q, double slope);
	public abstract void calcUniByH(double H, double slope);
	public abstract void calcUniByZs(double Zs, double slope);

	public int compareTo(RvSection another) {
		return    (int)signum(this.getX() - another.getX());
	}

	public String getName() {return name;}
	public double getX() {return x;}
	public double getWaterLevel() {return waterLevel;}
	public double getMinimumBedLevel() {return minimumBedLevel;}
	public double getAverageBedLevel() {return averageBedLevel;}
	public double getDepth() {return depth;}
	public double getArea() {return area;}
	public double getWaterWidth() {return waterWidth;}
	public double getHydraulicRadius() {return hydraulicRadius;}
	public double getDiscarge() {return discharge;}
	public double getVelocity() {return velocity;}
	public double getFricCoeff() {return fricCoeff;}

	public void setName(String name) {this.name = name;}
	public void setX(double x) {this.x = x;}
	public void setWaterLevel(double waterLevel) {
		this.waterLevel = waterLevel;
		depth = waterLevel - minimumBedLevel;
		calcSectionByH();
	}
	public void setDepth(double depth) {
		this.depth = depth;
		calcSectionByH();
	}
```

```
    public void setArea(double area) {
        this.area = area;
        calcSectionByA();
    }
    public void setDiscarge(double discharge) {
        this.discharge = discharge;
        calcSectionByQ();
    }
    public void setAreaAndDischarge(double area, double discharge) {
        this.area = area;
        this.discharge = discharge;
        calcSectionByAQ();
    }
    public void setDepthAndDischarge(double depth, double discharge) {
        this.depth = depth;
        this.discharge = discharge;
        calcSectionByHQ();
    }


}
```

```
// 河川網のなかに入り込んで作業をするためのインターフェース
package jp.ac.uuwem.rvsys.xml;

public interface RvVisitor {
	public abstract void visit(RvSys sys);
	public abstract void visit(RvNode node);
	public abstract void visit(RvChannel ch);
}
```

```
// テキストファイルへの出力
package jp.ac.uuwem.rvsys.xml;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.PrintWriter;

public class RvOutCSV implements RvVisitor {

	private String myName = "RvOutText";
	private String fileName;
	private PrintWriter printWriter = null;
	private FileWriter fileWriter = null;
	private double time;

	private RvOutCSV(){} //引数なしのコンストラクタはうけつけない.

	public RvOutCSV(String fileName){
		this();
		setFileName(fileName);
	}

	private void announceError(String s, Exception e){
		System.out.println(" Error " + myName + ":" + s);
		e.printStackTrace();
	}

	public RvOutCSV setTime(double time){
		this.time = time;
		return this;
	}

	public void setFileName(String fileName){
		this.fileName = fileName;
		openFile();
	}

	public void closeFile(){
		if (printWriter == null) return;
		try {
			printWriter.close();
		} catch (Exception e) {
			announceError("closeFile", e);
		}
	}
```

```
private void openFile(){
        closeFile();
        try{
                fileWriter = new FileWriter(fileName);
                printWriter = new PrintWriter(new BufferedWriter(fileWriter));
                printWriter.print("RvName,");
                printWriter.print("ChName,");
                printWriter.print("Nx,");
                printWriter.print("i,");
                printWriter.print("time(sec),");
                printWriter.print("time(hour),");
                printWriter.print("X,");
                printWriter.print("Zmin,");
                printWriter.print("Zs,");
                printWriter.print("H,");
                printWriter.print("A,");
                printWriter.print("Q,");
                printWriter.print("V,");
                printWriter.println();

        } catch (Exception e) {
                announceError("openFile", e);
        }
}

@Override
public void visit(RvChannel ch) {
        String rvName = ch.getRvSys().getFullName();
        String chName = ch.getFullName();
        double[] X = ch.getX();
        double[] Zmin = ch.getZmin();
        double[] Zs = ch.getZs();
        double[] H = ch.getH();
        double[] A = ch.getA();
        double[] Q = ch.getQ();
        double[] V = ch.getV();
        double timeHour = time/3600;

        try {
                for (int i = 0; i < X.length; i++) {
                        printWriter.print(rvName);
                        printWriter.print("," + chName);
                        printWriter.print("," + X.length);
                        printWriter.print("," + i);
                        printWriter.print("," + time);
                        printWriter.print("," + timeHour);
                        printWriter.print("," + X[i]);
                        printWriter.print("," + Zmin[i]);
                        printWriter.print("," + Zs[i]);
                        printWriter.print("," + H[i]);
```

```
                    printWriter.print("," + A[i]);
                    printWriter.print("," + Q[i]);
                    printWriter.print("," + V[i]);
                    printWriter.println();
                }
                printWriter.println();
            } catch (Exception e) {
                announceError("visitRvChannel", e);
            }
        }

        @Override
        public void visit(RvSys rv) {
            try {
                printWriter.println("**********");
                for(RvComponent rc: rv.getRvComponents()){
                    rc.accept(this);
                }
                printWriter.println();
            } catch (Exception e) {
                announceError("visitRvSys", e);
            }
        }

        @Override
        public void visit(RvNode node) {}

}
```

```
// XMLファイルへの出力
package jp.ac.uuwem.rvsys.xml;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.PrintWriter;
import org.apache.commons.betwixt.io.BeanWriter;

public class RvOutXMLSys {

	private String myName = "RvOutXML";
	private String fileName;
	private PrintWriter printWriter = null;
	private FileWriter fileWriter = null;
	private BeanWriter beanWriter = null;
	private RvOutXMLSysResult result = new RvOutXMLSysResult();

	private RvOutXMLSys(){} //引数なしのコンストラクタはうけつけない.

	public RvOutXMLSys(String fileName){
		this();
		setFileName(fileName);
	}

	private void announceError(String s, Exception e){
		System.out.println(" Error " + myName + ":" + s);
		e.printStackTrace();
	}

	public RvOutXMLSys setTime(double time){
		result.setTime(time);
		return this;
	}

	public void setFileName(String fileName){
		this.fileName = fileName;
		openFile();
	}
	public void closeFile(){
		if (printWriter == null) return;
		try {
			printWriter.println("</Results>");
			printWriter.close();
		} catch (Exception e) {
			announceError("closeFile", e);
		}
	}
```

```
    private void openFile(){
        closeFile();
        try{
            fileWriter = new FileWriter(fileName);
            printWriter = new PrintWriter(new BufferedWriter(fileWriter));
            printWriter.println("<?xml version='1.0' encoding='UTF-8'?>");
            printWriter.println("<Results>");

            beanWriter = new BeanWriter(printWriter);
           beanWriter.getBindingConfiguration().setMapIDs(false);
            beanWriter.enablePrettyPrint();
           beanWriter.setIndent("¥t");
        } catch (Exception e) {
            announceError("openFile", e);
        }
    }

    public void write(double t, RvSys rv) {
        result.setTime(t);
        result.setRvSys(rv);
        try {
            beanWriter.write(result);
            printWriter.println();
        } catch (Exception e) {
            announceError("write", e);
        }
    }


}
```

```
//計算結果bean
package jp.ac.uuwem.rvsys.xml;

public class RvOutXMLSysResult {
	private double time;
	private RvSys rvSys;

	public RvOutXMLSysResult(){}
	public RvOutXMLSysResult(double t, RvSys rv){
		this();
		setTime(t);
		setRvSys(rv);
	}

	public void setTime(double time) {this.time = time;}
	public double getTime() {return time;}
	public double getHour() {return time/3600;}

	public void setRvSys(RvSys rv) {this.rvSys = rv;}
	public RvSys getRvSys() {return rvSys;}


}
```

```
//XMLからの河川網の生成
package jp.ac.uuwem.rvsys.xml;

import java.io.FileInputStream;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.apache.commons.digester.Digester;
import org.apache.commons.digester.RuleSet;
import org.apache.commons.digester.xmlrules.FromXmlRuleSet;
import org.xml.sax.InputSource;

public class RvSysFactory {

	public static RvSys createRvSys(String targetFileName, String ruleFileName){
		RvSys rv=null;

		try{
			SAXParserFactory factory = SAXParserFactory.newInstance();
			factory.setNamespaceAware(true);
			factory.setXIncludeAware(true);
			SAXParser parser = factory.newSAXParser();

			Digester digester = new Digester(parser);

			InputSource rulesSource =
					new InputSource(new FileInputStream(ruleFileName));
			RuleSet rulesSet = new FromXmlRuleSet(rulesSource);
			digester.addRuleSet(rulesSet);

			InputSource dataSource =
					 new InputSource(new FileInputStream(targetFileName));
			rv = (RvSys)digester.parse(dataSource);
		}catch (Exception e){
			 e.printStackTrace();
		}

		return rv;
	}
}
```

```
//Case1の計算（開水路網をコーディング）
import jp.ac.uuwem.rvsys.text.DivCon;
import jp.ac.uuwem.rvsys.text.DownFree;
import jp.ac.uuwem.rvsys.text.RvNonUniChannel;
import jp.ac.uuwem.rvsys.text.RvOutText;
import jp.ac.uuwem.rvsys.text.RvRectangleSection;
import jp.ac.uuwem.rvsys.text.RvSys;
import jp.ac.uuwem.rvsys.text.UpDischarge;


public class Case1a {

	public static void main(String[] args) {
		//RvSysインスタンスの生成
		String sysName = "rv1";

		double manning = 0.03;

		double xL11 = 30 *1000;
		double B11 = 100;
		double slope11 = 1. / 1000.;
		double z011 = xL11 * slope11;

		double xL12 = 20 *1000;
		double B12 = 100;
		double slope12 = 1. / 1000.;
		double z012 = z011 + xL12 * slope11;
		double Qb12 = 60;
		double Qp12 = 600;
		double T012 = 0;
		double Tp12 = 12 *60*60;
		double Cp12 = 20;

		double xL13 = 25 *1000;
		double B13 = 50;
		double slope13 = 1. / 1000.;
		double z013 = z011 + xL13 * slope11;
		double Qb13 = 40;
		double Qp13 = 400;
		double T013 = 0;
		double Tp13 = 9 *60*60;
		double Cp13 = 20;

		int Nx = 50;
		RvSys rv1 = new RvSys(sysName);

		RvNonUniChannel ch11 = new RvNonUniChannel(sysName, "ch11", Nx);
		DownFree dnF11 = new DownFree(sysName, "dnF11");
		DivCon dvCn11 = new DivCon(sysName, "dvCn11");
		ch11.setDownNode(dnF11);
```

```
ch11.setUpNode(dvCn11);
        double x, zb, dx;
        dx = xL11 / Nx;
for (int i = 0; i <= Nx; i++) {
        x = i * dx;
        zb = z011 - slope11 * x;
        ch11.set(i, x, new RvRectangleSection(zb, B11, manning));
}

RvNonUniChannel ch12 = new RvNonUniChannel(sysName, "ch12", Nx);
UpDischarge upQ12 =
         new UpDischarge(sysName, "upQ12", Qb12, Qp12, T012, Tp12, Cp12);
ch12.setUpNode(upQ12);
ch12.setDownNode(dvCn11);
        dx = xL12 / Nx;
for (int i = 0; i <= Nx; i++) {
        x = i * dx;
        zb = z012 - slope12 * x;
        ch12.set(i, x, new RvRectangleSection(zb, B12, manning));
}

RvNonUniChannel ch13 = new RvNonUniChannel(sysName, "ch13", Nx);
UpDischarge upQ13 =
        new UpDischarge(sysName, "upQ13", Qb13, Qp13, T013, Tp13, Cp13);
ch13.setUpNode(upQ13);
ch13.setDownNode(dvCn11);
        dx = xL13 / Nx;
for (int i = 0; i <= Nx; i++) {
        x = i * dx;
        zb = z013 - slope13 * x;
        ch13.set(i, x, new RvRectangleSection(zb, B13, manning));
}

rv1.add(ch11);
rv1.add(dnF11);
rv1.add(dvCn11);
rv1.add(ch12);
rv1.add(upQ12);
rv1.add(ch13);
rv1.add(upQ13);

//計算結果をテキストファイルに出力するためのインスタンスを生成
RvOutText rvOut = new RvOutText("Case01.csv");

//不定流計算の実行
double t;
double dt = 10;
int Nt = 6 * 60 * 48;
int Mt = 6 * 60 * 3;
boolean initFlag = true;
```

```
            t = 0;
            rv1.init(t, !initFlag);
            rv1.renew();
            rv1.accept(rvOut.setTime(t));
        for (int i = 0; i < Nt; i++){
            t = i * dt;
            rv1.calc(t, dt);
            rv1.renew();
            if ((i+1)%Mt == 0){
                t = (i+1) * dt;
                System.out.println("Time(hour):" + t/3600);
                rv1.accept(rvOut.setTime(t));
            }
        }
        rvOut.closeFile();
        System.out.println("finished.");

    }

}
```

```
// Case2において，rv1を再利用してrv2を生成するコード
import jp.ac.uuwem.rvsys.text.DivCon;
import jp.ac.uuwem.rvsys.text.DownFree;
import jp.ac.uuwem.rvsys.text.RvChannel;
import jp.ac.uuwem.rvsys.text.RvNonUniChannel;
import jp.ac.uuwem.rvsys.text.RvRectangleSection;
import jp.ac.uuwem.rvsys.text.RvSys;
import jp.ac.uuwem.rvsys.text.UpDischarge;


public class MyRvSys2 {

	private static RvSys myRvSys = createRvSys();

	private MyRvSys2(){}
	public static RvSys getRvSys(){return myRvSys;}

	private static RvSys createRvSys(){
		//ここで生成する新たな河川網
		String sysName = "rv2";
		RvSys rv2 = new RvSys(sysName);

		//河川網の構成要素の生成
		double manning = 0.03;

		double xL21 = 60 *1000;
		double B21 = 200;
		double slope21 = 1. / 2000.;
		double z021 = 0.;

		double xL22 = 40 *1000;
		double B22 = 200;
		double slope22 = 1. / 2000.;
		double z022 = z021 + xL22 * slope21;
		double Qb22 = 100;
		double Qp22 = 1000;
		double T022 = 0;
		double Tp22 = 15 *60*60;
		double Cp22 = 20;

		int Nx = 50;

		RvNonUniChannel ch21 = new RvNonUniChannel(sysName, "ch21", Nx);
		DownFree dnF21 = new DownFree(sysName, "dnF21");
		DivCon dvCn21 = new DivCon(sysName, "dvCn21");
		ch21.setDownNode(dnF21);
		ch21.setUpNode(dvCn21);
			double x, zb, dx;
			dx = xL21 / Nx;
		for (int i = 0; i <= Nx; i++) {
```

```
            x = i * dx;
            zb = z021 - slope21 * x;
            ch21.set(i, x, new RvRectangleSection(zb, B21, manning));
        }

        RvNonUniChannel ch22 = new RvNonUniChannel(sysName, "ch22", Nx);
        UpDischarge upQ22 =
              new UpDischarge(sysName, "upQ22", Qb22, Qp22, T022, Tp22, Cp22);
        ch22.setUpNode(upQ22);
        ch22.setDownNode(dvCn21);
            dx = xL22 / Nx;
        for (int i = 0; i <= Nx; i++) {
            x = i * dx;
            zb = z022 - slope22 * x;
            ch22.set(i, x, new RvRectangleSection(zb, B22, manning));
        }

        //河川網への構成要素の追加
        rv2.add(ch21);
        rv2.add(dnF21);
        rv2.add(dvCn21);
        rv2.add(ch22);
        rv2.add(upQ22);

        //既存の河川網生成クラスの再利用
        RvSys rv1 = MyRvSys1.getRvSys();
        //この河川網クラスはここで生成している河川網に所属することになる.
        rv1.setSysName(sysName);
        //ここで生成している河川網との接続を設定する.
        RvChannel rv1ch11 = (RvChannel)rv1.getRvComponent("rv1:ch11");
        rv1ch11.setDownNode(dvCn21);
        //ここで生成している河川網に追加する.
        rv2.add(rv1);

        return rv2;
    }

}
```

```
//XMLファイルによる河川網の生成と計算結果のXML出力
import jp.ac.uuwem.rvsys.xml.RvOutCSV;
import jp.ac.uuwem.rvsys.xml.RvOutXMLSys;
import jp.ac.uuwem.rvsys.xml.RvSys;
import jp.ac.uuwem.rvsys.xml.RvSysFactory;


public class Case2XML {

	public static void main(String[] args) {

		RvSys rv = RvSysFactory.createRvSys("rv2.xml", "RuleOfRvSys.xml");
		if (rv == null){
			System.out.println("rv is null.");
			return;
		}
		RvOutCSV outCSV = new RvOutCSV("Case2.csv");
		RvOutXMLSys outXML = new RvOutXMLSys("Case2.xml");

		double dt = 60;
		int Nt = 1 * 60 * 48;
		int Mt = 1 * 60 * 3;
		boolean initFlag = true;

		double t = 0;
		rv.init(t, !initFlag);
		rv.renew();
		rv.accept(outCSV.setTime(t));
		outXML.write(t, rv);

		for (int i = 0; i < Nt; i++){
			t = i * dt;
			rv.calc(t, dt);
			rv.renew();
			if ((i+1)%Mt == 0){
				t = (i+1) * dt;
				rv.accept(outCSV.setTime(t));
				outXML.write(t, rv);
				System.out.println("Time(hour):" + t/3600);
			}
		}
		outCSV.closeFile();
		outXML.closeFile();
		System.out.println("finished.");

	}

}
```

OBJECT ORIENTED PROGRAMMING OF UNSTEADY FLOW COMPUTATION IN OPEN CHANNEL NETWORK DESCRIBED WITH USING XML

Ikeda,H. [1]
[1] Utsunomiya University

Computation system of unsteady flow in open channel network, which is described with using XML, was developed with object oriented programming in Java language. An open channel network is described as a “river system” instance, which is composed of some kinds of instances; “river channels”, “river nodes” and another smaller “river systems”. Computation procedure for this nested recursive structure was coded with using “composite pattern” and “visitor pattern”, that enables to improve readability and re-usability of the program code. Conversion of XML data to Java-instances and *vice versa* was established effectively with using Jakarta Commons Digester and Betwixt, that means extensive possibility of these kind of applications with XML.
.

KEYWORDS**:** *XML, object oriented programming, composite pattern, visitor pattern, Jakarta Commons, Digester, Betwixt, open channel network, unsteady flow.*

様式－３－２

研　究　成　果　の　要　約

助成番号	助　成　研　究　名	研 究 者・所 属
第2005-8号	ＸＭＬによる河川網の記述とオブジェクト指向プログラミングおよびデータ流通への活用	池田　裕一 宇都宮大学大学院工学研究科

本研究は，XMLにより河川網を記述し，それを用いて1次元不定流計算を行うアプリケーションを，Java言語で開発することを目的としたものである．

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が，さまざまな外部あるいは内部の境界条件を扱い，河川の分合流も考慮して河川網を扱うことを考えると、アプリケーション全体はかなり複雑なものになる。

そこで，この問題にオブジェクト指向プログラミング（以下，OOP）を適用し，いわば開水路網の入れ子状態を扱う仕組みをも組み込むことにした．加えて，OOPにおけるデザインパターンの考え方を取り入れて，プログラムの可読性とプログラムコードの再利用性の向上を目指した．

また，最近よく普及しているデータ記述言語XMLを利用して，河川網の入れ子構造を簡便に記述する方法を検討した．

XMLで記述されたデータをJavaプログラムで活用するために，さまざまなプロジェクトでプログラムの開発が行われている．その成果を有効利用することで，アプリケーション構築の効率化を図ることにした．

その結果，以下の知見が得られた．

河川網を小規模な河川網の入れ子構造として構成するために，OOPのデザインパターンにおけるCompositeパターンが有効であることがわかった．また開水路網の入れ子構造を順に巡りながら，さまざまな処理を行う場合には，デザインパターンのVisitorパターンを活用すればよいことがわかった．

OOPの継承と多態性を活用して，河川網の不定流計算プログラムが簡潔に直感的にコーディングできることを示した．すなわち開水路インスタンスを生成して，その上下流端に境界条件としての境界点インスタンスをセットし，それらを開水路網の構成要素として加えていくようなコーディングを可能にした．

しかも小規模な開水路網を生成するコードを手直しせずに，より大きな開水路網に組み込むことができるので，プログラムコードの再利用あるいは段階的で効率的な開水路網の構築が可能であることを示した．

XMLにより河川網の入れ子構造を記述するには，XIncludeを用いると見通しがよくなることを示した．そして本研究で扱う開水路網を記述するXML規則を示した．

Jakarta Commons Digesterを活用することにより，XMLファイルから河川網インスタンスを生成するプログラムを簡潔にコーディングできることを示した．

河川網の構成要素のデータとそのデータからインスタンスを生成するルールを，ともにXML形式のファイルとして，プログラム本体から独立させることで，アプリケーションのメンテナンスやオプションの追加が容易になることを示した．

Jakarta Commons Betwixtを活用することにより，河川網の不定流の計算結果をXML形式で出力するプログラムを簡潔にコーディングできることを示した．

計算結果の出力形式をマッピングファイルを用いてカスタマイズすることにより，河川網の入れ子構造を反映させながら，必要なデータをXMLファイルに出力することが出来た．これにより，計算結果をもとに，もとの河川網と同じ入れ子構造を有する新たなインスタンスを生成して，さまざまな処理を実行するアプリケーションを容易に構築できるであろうことを考察した．

今後の課題としては，アプリケーションにさまざまな解析機能を追加していく場合の戦略として，OOPのBridgeパターンを検討する必要があること，今後のXMLデータの流通の発展のためには，影響力の大きなXMLデータ仕様との整合性を図りながら，活用の度合いを高めていくことが必要であることを指摘した．