SEAMAIL

Newsletter from Software Engineers Association

Vol. 12, Number 2 November, 1999

目　次

ソフトウェア技術者協会
Software Engineers Asociation

ソフトウェア技術者協会(SEA)は，ソフトウェアハウス，コンピュータメーカ，計算センタ，エンドユーザ，大学，研究所など，それぞれ異なった環境に置かれているソフトウェア技術者または研究者が，そうした社会組織の壁を越えて，各自の経験や技術を自由に交流しあうための「場」として，1985 年12 月に設立されました．

その主な活動は，機関誌SEAMAIL の発行，支部および研究分科会の運営，セミナー／ワークショップ／シンポジウムなどのイベントの開催，および内外の関係諸団体との交流です．発足当初約200 人にすぎなかった会員数もその後飛躍的に増加し，現在，北は北海道から南は沖縄まで，500 余名を越えるメンバーを擁するにいたりました．法人賛助会員も20 社を数えます．支部は，東京以外に，関西，横浜，長野，名古屋，九州，広島，東北の各地区で設立されており，その他の地域でも設立準備をしています．分科会は，東京，関西，名古屋で，それぞれいくつかが活動しており，その他の支部でも，月例会やフォーラムが定期的に開催されています．

「現在のソフトウェア界における最大の課題は，技術移転の促進である」といわれています．これまでわが国には，そのための適切な社会的メカニズムが欠けていたように思われます．SEA は，そうした欠落を補うべく，これからますます活発な活動を展開して行きたいと考えています．いままで日本にはなかったこの新しいプロフェッショナル・ソサイエティの発展のために，ぜひとも，あなたのお力を貸してください．

ＳＥＡＭＡＩＬ Ｖｏｌ．１２，Ｎｏ．２ １９９９年１１月８日発行
編集人 岸田孝一
発行人 ソフトウェア技術者協会（ＳＥＡ）
〒１６０-０００４ 東京都新宿区四谷３－１２ 丸正ビル５Ｆ
T: 03-3356-1077 F: 03-3356-1072 sea@sea.or.jp
印刷所 有限会社錦正社 〒１３０-００１３ 東京都墨田区錦糸町４－３－１４
定価 ５００円 （禁無断転載）

編集部から

☆

とりあえず，Vol.12 - No.2 は船便にならずにすみました．やれやれ．

☆☆

巻頭の長編論文は，CORBA を用いた開発に関係されている方々にとってはきわめて有用な情報だと思います．御活用ください．

☆☆☆

前号に引き続いて，山崎さんからは，Formal Language を用いたアルゴリズム・デザインの美しい例題を提供していただきました．途中で何度かバグがみつかったと原稿の差し替えがありました！もしまだ残っていたとしたら，発見者には，編集部から賞品をさしあげます．

☆☆☆☆

奈良先端大学院の花川さん他の論文は，プロジェクト・プランニングのためのシミュレータの解説です．実際に現場で使って見たいと思われる方は，どうぞ花川さん宛にe-mail で御連絡を！

☆☆☆☆☆

次に載っているのは，巻頭のCORBA 論文を御紹介くださった熊谷さんが書かれたこの夏のヨーロッパ出張の紀行随筆です．まだ続きがあるとかで，それは次号に．

☆☆☆☆☆☆

来年のソフトウェア・シンポジウム 2000 の論文募集も始まりました．ふるってご応募ください．

☆☆☆☆☆☆☆

リフレクションを利用した CORBAのアプリケーション開発環境

藤枝 和宏 (北陸先端科学技術大学院大学)
東田雅宏 (PFU)

1 はじめに

CORBA(Common Object Request Broker Architecture) は、OMG(Object Management Group) によって策定された分散オブジェクト指向環境の標準規格である [1]。CORBA の ORB(Object Request Broker) を利用することで、ネットワークを介してサービスを提供するサーバを分散オブジェクトとして実装し、クライアントは分散オブジェクトのメソッド (CORBA ではオペレーションと呼ばれる) を実行する形でサービスを利用することが可能になる。

分散オブジェクトの実装やそれを呼び出すクライアントは、オブジェクト指向モデルを採用していない言語を含め、さまざまなプログラミング言語で記述可能である。オブジェクト指向プログラミング言語では、分散オブジェクトの実装をその言語のクラスとして記述し、クライアントからは分散オブジェクトをその言語のオブジェクトして操作することが可能である。

しかし、CORBA のオブジェクトモデルは C++を参考に設計されているため、他のプログラミング言語で CORBA のアプリケーションを記述する際には、多重継承のモデル、型システム、名前空間などがうまく対応せずプログラムの記述が難しくなる傾向がある。また、分散オブジェクトをプログラミング言語のオブジェクトと対応づけるために、IDL コンパイラによって生成されるスタブとスケルトンと呼ばれるプログラムコードを利用する必要があるため、開発手順が煩雑になっている。

本報告書ではリフレクションというプログラミング言語にプログラム自身やプログラムの処理系を操作するプログラムを記述可能にする概念を利用して、分散オブジェクトとプログラミング言語を対応付ける際のプログラミングや開発手順を簡略化する方法について述べる。また、リフレクションを利用することで、分散オブジェクトの実装や操作のために用意されている CORBA の API の実行手順を簡略化する方法についても述べる。

本報告書では、まず 2 節で CORBA の概要について述べ、次に CORBA のアプリケーション開発における問題点を明らかにするために、2 節で CORBA のアプリケーション開発の手順を述べ、4 節でその問題点について議論する。5 節では、問題を解決する方法として、CORBA のインターフェイスリポジトリとリフレクションの概念と利用した CORBA のアプリケーション開発環境について述べる。6 節では、CORBA のアプリケーション開発を容易にすることを目的として行われている関連研究や製品について紹介する。そして、7 節で本報告書のまとめと今後の課題について述べる。

2 CORBA の概要

CORBA(Common Object Request Broker Architecture) は、OMG(Object Management Group) によって策定された分散オブジェクト指向環境の標準規格である [1]。CORBA の規格は、分散オブジェクトを実現する Object Request Broker(ORB) を中心に、さまざまな分散オブジェクトに共通して必要となるサービスをまとめた Common Object Services および Common Facilities からなる。

CORBA の ORB を利用することで、ネットワークを介してサービスを提供するサーバを分散オブジェクトとして実装し、クライアントから分散オブジェクトのメソッド (CORBA ではオペレーションと呼ばれる) を実行することで、サービスを利用することが可能になる。

> CORBA の ORB のことを「分散オブジェクト間の通信をサポートするシステム」と紹介している文献を見かけることがあるかもしれないが、この記述は現状では正しくない。CORBA では、サーバである分散オブジェクトと、それを利用するクライアントは明確に区別されており、分散オブジェクトがさらに別の分散オブジェクトと通信する方法は規格には示されていない。

Common Object Services は、さまざまな分散オブジェクトに共通して必要となる基本的なサービスを集めたものである。各サービスについては CORBAservices[2] で定められている。この規格には、オブジェクトの位置と名前の対応を管理する Naming Service、非同期通信をサポートする Event Service、通信路の暗号化をサポートする Security Service、トランザクションをサポートする Transaction Service など 15 のサービスが含まれているが、実際に実装されて広く利用されているのは、例示した 4 つ程度である。

Common Facilities は、多くのアプリケーションで共通して必要となるが必須ではないサービス、およびアプリケーションの分野に依存するサービスを集めたものである。CORBAfacilities[3] でその概要が定められている。

[1] は主に ORB について定めたものであり、分散オブジェクトのオブジェクトモデル、オブジェクトのインターフェイスを記述する IDL(Interface Description Language)、分散オブジェクトの実装と操作に必要な API、および IDL と API の各プログラミング言語 (C 言語、C++、Java、Ada、COBOL、SmallTalk) へマッピングが示されている。

ORB を利用することで、分散オブジェクトのオペレーションを、プログラミング言語のオブジェクトのメソッド (あるいは手続き) と同じ方法で呼び出すことができる。さらに、オペレーションの引数が複雑なデータ型の場合でも、対応するプログラミング言語のデータ型をそのまま用いることができる。この際用いられる、分散オブジェクトのメソッドを呼び出すネットワークプロトコルや、複雑なデータ型のネットワーク上での表現形式は ORB の実装に隠蔽されているので、アプリケーションプログラマは意識する必要がない。

さまざまなプログラミング言語から利用できることも、CORBA の大きな特徴であり、CORBA のサポートしている、どのプログラミング言語で実装した分散オブジェクトも、他のプログラミング言語から利用することが可能である。また、CORBA の規格には含まれていないが、Perl、Tcl、Python といったスクリプト言語にも CORBA を利用するためのパッケージが存在する。これらの言語から C++や Java などで実装された分散オブジェクトを操作することが可能であり、逆にこれらのスクリプト言語で分散オブジェクトを実装することも可能である。

分散オブジェクトをプログラミング言語のオブジェクトと同様に操作するためには、分散オブジェクトと通常のオブジェクトの間を仲介するスタブと呼ばれるプログラムが必要である。CORBA のアプリケーションを作成する際には、分散オブジェクトのインターフェイスを記述した IDL ファイルからスタブを生成し、アプリケーションに適切に取り込む必要がある。この手順が入るために CORBA のアプリケーション開発はいくらか複雑になっている。

また、分散オブジェクトを実装する際には、IDL で定義したオブジェクトのインターフェイスを満たすように、プログラミング言語のオブジェクトを実装しなければならない。CORBA のオブジェクトモデルとプログラミング言語のオブジェクトモデルとの間にオブジェクトモデルの差がある場合もあり、通常のオブジェクトを実装する場合と比べてプログラミングが難しくなる。

次節では、これらの問題点についてより詳細に検討するために、CORBA のアプリケーションの開発手順を説明する。

3 CORBA アプリケーションの開発手順

CORBA のアプリケーション開発は、おおむね以下の手順で行われる。

- IDL ファイルの記述
- IDL コンパイラを用いたスタブの生成
- サーバの記述
- クライアントの記述
- ビルド

以下各手順について具体的に説明する。

3.1 IDL ファイルの作成

まず、IDL(Interface Description Language) を用いて、分散オブジェクトのインターフェイスを記述したファイルを作成する。IDL で記述したインターフェイスの例を図 1 に示す。IDL は C++を元に設計されているので概観は C++によく似ている。この例で用いられている module というキーワードは C++の namespace に対応する。これは、名前空間を分離するために用いられる。C++の class に相当するのは interface である。このブロックの中で分散オブジェクトのインターフェイスを記述する。

```
/* Copyright (c) FUJITSU LIMITED 1998 */

module ODdemo{
  interface   calculator{
    exception ZEROPARAM {};
    struct result {
      long    add_result;
      long    subtract_result;
      long    multiple_result;
      float   divide_result;
    };
    result calculate(in long a, in long b)
    raises (ZEROPARAM);
  };
};
```

図 1: calc.idl

インターフェイスとして記述するのは、分散オブジェクトの持つ属性[1]とオペレーションの宣言である。属性の型およびオペレーションの引数、返値の型として、数値や文字列などの基本型の他に、構造体や配列などの構造を持ったデータ型や、他の分散オブジェクトを指すオブジェクトリファレンス型を指定することができる。

構造を持ったデータ型を利用する場合には、事前にその定義を記述する必要がある。また、オブジェクトリファレンス型を利用する場合には、そのリファレンス型が指すオブジェクトのインターフェイスも記述しておく必要がある。C++のクラスのように、インターフェイスは複数のインターフェイスを継承することができるが、その場合には継承するインターフェイスも記述しておかなければならない。

IDL ファイルは C++と同じプリプロセッサで処理されることが規格で定められているので、#include を用いて定義を複数のファイルにわけて記述することもできる。ただし、プリプロセッサを通った後のファイルは、必要な定義をすべて含んでいなければならない。

3.2 IDL コンパイラを用いたスタブの生成

IDL ファイルを作成したら、CORBA の実装に付属している IDL コンパイラにかけてスタブを生成する。IDL コンパイラの名前やコンパイルした結果生成されるファイルの種類は、対象とするプログラミング言語や CORBA の実装によって異なっている。

基本的には、IDL コンパイラは IDL ファイルで定義されたインターフェイスに対応するスタブとスケルトンを生成する。スタブはプログラミング言語のオブジェクトと ORB の間を仲介するプログラムである。スタブには IDL で定義された構造体などのデータ型に対応する、プログラミング言語のデータ型の定義も含まれる。スケルトンは ORB と分散オブジェクトの実装の間を仲介するプログラムであり、分散オブジェクトを実装する際に必要となる。

CORBA の実装の一つである ObjectDirector[4] では、図 1 を calc.idl というファイル名で保存して、C++を対象として IDL コンパイラを実行すると以下のファイルが生成される。

- calc.h

  calc.idl で定義されている calculator インターフェイス、ZEROPARAM 例外、result 構造体に対応する C++のデータ型の宣言が含まれている。このファイルは分散オブジェクトを実装するサーバと、それを利用するクライアント両方に必要である。

- calc_c++.cpp

  calculator インターフェイスのオブジェクトリファレンス型を扱うための calculator クラス、ZEROPARAM 例外に対応する ZEROPARAM クラスの実装が含まれている。このファイルはサーバとクライアント両方に必要である。

- calc_stub.cpp

[1]例題には含まれていない

calculator インターフェイスのスタブクラス_st_ODdemo_calculator[2]の実装が含まれている。このクラスは calculator クラスの実装に隠蔽されているので、アプリケーションプログラムから参照されることはない。このファイルはクライアントにのみ必要である。

- calc_skel.cpp

  calculator インターフェイスのスケルトンクラス_sk_ODdemo_calculator の実装が含まれている。このファイルはサーバにのみ必要である。

- calc_cdr.h と calc_cdr.c

  calc.idl で定義されているデータ型をネットワーク上の表現形式 (CDR — Common Data Represenation) と相互に変換するための関数の宣言と定義が含まれている。クライアントとサーバの両方に必要である。

ObjectDirector では分散オブジェクトの実装 (サーバ) に必要な要素、クライアントに必要な要素、共通して必要な要素を完全に別のファイルに分離しているため、他の実装と比較してファイルの数が多くなっている。

ここで注意してほしいのは、IDL ファイルに対応してファイルが生成されることである。IDL ファイルが他の IDL ファイルをインクルードしている場合には、インクルードされた内容もすべて同じファイルに出力される。定義されているインターフェイスやデータ型が多い場合には、各ファイルに出力される内容が非常に多くなる。

ObjectDirector を含むほとんどの実装では、インクルードされた IDL ファイルの内容を出力しないようにするオプションが、IDL コンパイラに用意されている。このオプションを用いることで、生成されるファイルを小さくすることができるが、インクルードされる IDL ファイルについても IDL コンパイラを実行する必要がある。

Java を対象とする場合には、IDL コンパイラは IDL ファイルの名前とは関係なく、Java の慣習に従いクラスごとにファイルを生成する。IDL コンパイラによって生成される Java のクラスには、インターフェイスに対応するスタブクラスとスケルトンクラス、データ型や例外に対応するクラスの他に、すべてのデータ型について Helper クラスと Holder クラスが含まれる。Helper クラスはデータ型に関する CORBA の API を実装したクラスで、Holder クラスは引数の参照渡しに用いられるクラスである。たとえば、calc.idl を Java に対応した IDL コンパイラでコンパイルすると 14 個のファイルが生成されることになる。

[2]規格では namespace が使えない C++の処理系では、_でモジュール名をつなげた名前に変換することになっている。

3.3 サーバの記述

3.3.1 分散オブジェクトの実装

まず、IDL ファイルで定義されたインターフェイスを実装する分散オブジェクトのクラス定義を作成する。ObjectBroker では、C++の場合には IDL コンパイラがクラスの宣言を自動的に生成するので、メソッドの実装を記述するだけでよい。図 1 の場合には、分散オブジェクトを実装するクラスは ODdemo_calculator_impl として calc.h で宣言されているので、このクラスの result オペレーションの実装を記述すればよい (図 2)。

```
/* Copyright (c) FUJITSU LIMITED 1998 */
ODdemo::calculator::result
ODdemo_calculator_impl::calculate(
  CORBA::Long a,
  CORBA::Long b,
  CORBA::Environment & )
  throw( CORBA::Exception )
{
  if( b == 0 ){
    ODdemo::calculator::ZEROPARAM  *exc =
        new ODdemo::calculator::ZEROPARAM();
    throw( *exc );
  }
  ODdemo::calculator::result res;
  res.add_result = a+b;
  res.subtract_result = a-b;
  res.multiple_result = a*b;
  res.divide_result = (CORBA::Float)a/b;
  return res;
}
```

図 2: result オペレーションの実装

インターフェイスが属性を持つ場合には、属性の値を設定/参照する属性と同じ名前を持つメソッドを定

義しなければならない。たとえば、IDL ファイルで属性が attribute long x として定義されている場合には、参照するメソッドとして CORBA::long x() を設定するメソッドとして void x(CORBA::long) を定義する。これらのメソッドが参照するクラスのメンバは、IDL コンパイラが生成したクラスの宣言に (たとえば CORBA::Long _x として) 含まれているので、それを利用する。

分散オブジェクトを実装するクラスはスケルトンクラス (図 2 の場合には _sk_ODdemo_calculator) を継承しなければならない。ObjectDirector の場合には IDL コンパイラが生成したクラス宣言でクラス継承が指定されているので、特に気にする必要はない。CORBA では共通するベースクラスは共有させなければならないので、継承するクラスには必ず virtual キーワードを指定しなければならないが、これも ObjectDirector の場合には IDL コンパイラによって指定される。ObjectBroker 以外の CORBA の実装では、IDL コンパイラの生成したスケルトンクラスを参考に、以上の点に注意しつつクラス宣言から記述しなければならない。

メソッドの実装が宣言と対応しない、メソッドを実装し忘れているといった類いのミスは、IDL コンパイラが生成したヘッダファイルに基づいて C++のコンパイラによって検出される。

3.3.2 main ルーチンの記述

分散オブジェクトの実装を作成したら、そのインスタンスを提供する main ルーチンを記述する。main ルーチンでは、ORB を初期化し、分散オブジェクトを実装するクラスのインスタンスを作成し、ネームサーバにそのインスタンスを登録して、リクエストを待ち受けるメインループを実行する API を呼び出せばよい。この一連の処理を行う際に呼び出す API は CORBA の実装によって異なっている。

3.4 クライアントの記述

クライアント側で分散オブジェクトを利用する際には、ORB を初期化した後に、分散オブジェクトのオブジェクトリファレンスを取得し、そのメソッドを呼び出せばよい。

オブジェクトリファレンスを取得する方法はおおむね以下の 3 通りである。

- NamingService を利用する方法

  NamingService のオブジェクトリファレンスを resolve_initial_reference により取得し、NamingService に問い合わせてオブジェクトリファレンスを取得する。

- Parsistent Object Reference を利用する方法 (ベンダ依存)

  ホスト名、ポート番号、オブジェクトキーから、オブジェクトリファレンスを生成する。VisiBroker[5] でサポートされている。

- Stringified IOR(Interoperable Object Reference) を利用する

  オブジェクトリファレンスを文字列化 (stringify) したものをファイルなどを経由して読み込み、string_to_object によりオブジェクトリファレンスに変換する。

取得したオブジェクトリファレンスを用いて、分散オブジェクトのメソッドを呼び出す方法は二つある。一つは、スタブを利用した Static Invocation Interface(SII) であり、もう一つはスタブを利用しない Dynamic Invocation Interface(DII) である。

SII ではスタブクラスを利用することで、分散オブジェクトのメソッドを通常のオブジェクトと同じメソッド呼び出しの構文を用いて実行することができる。SII を実行する場合には、上記の方法で得たオブジェクトリファレンスは任意のインターフェイスの分散オブジェクトを表す CORBA::Object を指しているので、スタブクラス (Java の場合には Helper クラス) の持つnarrow メソッドを用いて、オブジェクトリファレンスをそのスタブクラスを指すものに変換する必要がある。型を変換した後は、IDL に記述されたとおりに引数を指定してスタブクラスのメソッドを呼び出せばよい。エラーが起きた場合には例外が発生するので、try-catch 構文を用いて例外を捕捉しエラー処理を行うことができる。

DII の場合は、スタブクラスを必要としないが、メソッド呼び出しとはまったく異なった形式で行われる。

- request オブジェクトの生成
- request オブジェクトにメソッド名、引数の型と値の組みを設定する

- request オブジェクトの invoke() メソッドを呼ぶ
- exception が発生していないか調べる (try-catch 構文は使えない)

DII を用いて文字列を引数に持つ printf メソッドを呼び出す例を図 3 に示す。printf メソッドを呼び出すだけですむ SII と比べるとかなり煩雑である。

```
CORBA_Request_var req =
    obj->_request("printf");
req->add_in_arg() <<=
    (const char *)"Hello World!!\n";
req->invoke();
CORBA_Exception
    *exp = req->env()->exception();
if (exp) {
    cerr << program_name
        << ": dynamic invocation failed."
        << endl;
    return 1;
}
```

図 3: DII の例

3.5 ビルド

まず、IDL コンパイラが生成したソースファイルをすべてコンパイルし、サーバとクライアントのソースファイルもコンパイルする。これは Java と C++のどちらの場合にも必要な作業である。C++の場合には、サーバとクライアントそれぞれに必要なオブジェクトをリンクして実行ファイルを生成する必要がある。

先の、calc.idl の例では、クライアントにはクライアントのソースコードをコンパイルしたオブジェクトに加えて、calc_c++.o、calc_stub.o、calc_cdr.o が必要であり、サーバにはサーバのメイン、分散オブジェクトの実装のオブジェクトに加えて、calc_c++.o、calc_skel.o、calc_cdr.o が必要である。

Java の場合には、必要なクラスファイルを自動的に読み込むので、必要なクラスファイルの組み合わせを明示的に指定する必要はない。ただし、実際に運用する際には必要なクラスファイルをサーバを実行するホストとクライアントを実行するホストに適切に配置する必要がある。クライアントをアプレットとして実行する場合には、クライアントに必要なクラスファイルをアプレットからロードできる場所に配置しなければならない。

4 CORBA のアプリケーション開発における問題点

本節では、CORBA のアプリケーションを開発する過程において、開発者の負担を必要以上に大きくしていると考えられるいくつかの問題点を論じる。

4.1 サーバの記述に関する問題点

サーバを記述するための API である BOA(Basic Object Adaptor) の規格がんだったために、ベンダごとに BOA の実装、利用法がまちまちになっている。また、マルチスレッドなど scalability の高いサーバを書くために必要な要素がまったく考慮されていないため、ベンダ独自の API で解決している。結果として、サーバのコードは特定のベンダの実装に依存せざるをえず、移植性がまったくない。たとえば、リクエストが届いたときに新しいスレッドを生成して処理するようなサーバを実装しようとするだけで、ベンダ独自の API を利用しなければならない。

この問題を解決するために、POA(Portable Object Adaptor) という新しい規格が策定されたが、大手ベンダの実装ではサポートされていない。POA は、いろんなベンダが独自 API で実現してきたことを、ほとんどすべて実現しているため、規格が非常に大きくなっており、利用するのが非常に難しい。

4.2 インターフェイスに関する情報の管理

IDL ファイルで定義された内容を管理するためのリポジトリとして、Interface Repository(IR) が用意されている。IDL ファイルの内容を IR に格納するツールは、たいていの実装で提供されている。しかし、IR に格納された内容からスタブを生成するツールが提供されていることはまれで、CORBA のアプリケーション開発に IR を利用することが難しくなっている。

IR をアプリケーション開発に利用する上では、IR に格納されている情報を更新したときに、参照する側

からはそれがいつ更新されたのかわからないのも問題である。

IDL で定義を記述する際に、バージョン番号を定義ごとに付けることができるが、番号を手で振らなければならないため一貫性が保てなくなる可能性がある。また、IR ではバージョン番号の違う定義はまったく別の定義として扱われるため、参照する側からはバージョンを追跡することができない。

IR に格納された情報に基づいてリモートオブジェクトを操作するには、Dynamic Invocation Interface(DII) を用いなければならないため、プログラミングの手間が大きくなる。

Interface Repository にもスタブにも頼らずに、リモートオブジェクトに関する情報を取得する方法は、かなり制限されている。オブジェクトリファレンスの実装には、そのオブジェクトの型をあらわすフィールドが用意されているが、このフィールドは空でもかまわないことになっている。Any 型で用いられる、CORBA のデータ型をあらわす TypeCode というオブジェクトには、そのデータ型の構造がすべて含まれているが、名前のフィールドは空でもかまわないことになっている。そのため、構造体では各メンバの型は調べることができるが、そのメンバがどういう名前で定義されているかを調べることができない場合がある。

4.3 オブジェクトモデルに関する問題点

分散オブジェクトの操作や実装は、ほとんどの場合プログラミング言語のオブジェクトと同じ構文を用いて記述することができる。しかし、分散オブジェクトとプログラミング言語のオブジェクトの間のオブジェクトモデルの差や、プログラミング言語の仕様上の限界などから、分散オブジェクト固有の記述が必要になることが多い。そのような場合には、プログラミング言語の構文要素が利用できないため、プログラミングの量が多くなってしまう。

4.3.1 クライアント側の問題

- 型キャストの問題

  NamingService の検索結果や`string_to_object`操作の返値は、任意のインターフェイスの分散オブジェクトをあらわす`CORBA.Object`型である。この返値を用いて分散オブジェクトのオペレーションを呼び出すには、Java や C++ではそのスタブクラスに型キャストしなければならない。

  この際に、Java の持つ型キャストや C++の dynamic_cast は利用することができない。スタブクラス (Java の場合 Helper クラス) のクラスメソッドとして定義さている`_narrow`というメソッドを使う必要がある。

  Microsoft の Visual J++では Java VM に手を入れて、COM のインターフェイスを完全に Java のインターフェイスとして見せることで、型キャストを可能にしている [6]。

- null オブジェクトの扱い

  nil オブジェクト (CORBA::OBJECT_NIL) の概念は、Java では null にマップされているが、C++の場合には NULL ポインタに対応づけられていない。nil オブジェクトの検査には_is_nil メソッドを使う必要があり、nil オブジェクトが必要なときは_nil クラスメソッドで取得しなければならない。

- オブジェクトリファレンスの比較

  同じリモートオブジェクトのスタブオブジェクトが、同じオブジェクトであるという保証はないので、プログラミング言語の持つオブジェクトアイデンティティの概念は利用することができない。すべてのスタブクラスには`_is_equivalent`メソッドが用意されており、このメソッドを使って同一性を判断しなければならない。

- スタブオブジェクトに関する制約

  スタブオブジェクトが不要になった時点で分散オブジェクトとの接続を切る必要があるので、C++ではスタブオブジェクトの複製や削除は`_duplicate`、`_release`というメソッドを用いなければならない。

- 参照渡しに関する制約

  IDL で operation を定義する際に、値を受け取る引数を指定することができる。C++では引数で直接値を受け取ることができるので問題はない。Java ではそれができないため、その型の値を保持

するHolderクラスのインスタンスを介して値をやりとりする必要がある。HolderクラスはIDLコンパイラによってインターフェイスやユーザ定義型ごとに生成される。

- 名前空間の制約

  IDLの名前空間のスキーマはC++をベースに設計されているので、他の言語にうまく対応しないことがある。Javaではインターフェイスもユーザ定義型もすべてクラスで実装されるため、IDLファイルでinterface文の中でユーザ定義型が定義されていると、クラスの中でクラスを定義することが必要になる。しかし、規格策定当時のJavaは入れ子のクラス定義をサポートしていなかったので、インターフェイスに対応するpackageを生成して、その中でユーザ定義型に対応するクラスを定義するようになっている。そのため、インターフェイスの中で定義されたユーザ定義型を利用する際のプログラミングが、極めて不自然になっている。

4.3.2 サーバ側の問題

- 属性の実装

  IDLで定義されたインターフェイスの持つ属性は、代入用と参照用の二つのメソッドで実装しなければならない。各言語の属性の概念をそのまま用いることはできない。

- 継承の実装

  IDLで定義されたインターフェイスを実装するクラスを定義する際に、インターフェイスが別のインターフェイスを継承している場合には、スケルトンクラスの他に、継承しているインターフェイスを実装しているクラスも継承する必要がある。Javaの場合は多重継承を利用することができないので、継承しているインターフェイスのオペレーションは、そのインターフェイスを実装しているクラスのインスタンスへ委譲する形で実装を記述しなければならない。インターフェイスが多重継承している場合は、委譲を使って実装しなければならないだけでなく、重複しているベースクラスが共有されるように実装しなければならない。

4.4 各APIの問題点

- 遅延同期呼び出し

  CORBAではonewayを指定したオペレーション以外は、呼び出すとサーバからオペレーションの返値が戻るまで待つ同期呼び出しである。返値を待たずに別の処理を行い、後で結果を受け取る遅延同期(Deferred Synchronous)呼び出しもサポートされているが、これを利用するにはDIIを用いなければならないため、プログラミングの手間が非常に大きくなる。

  現在策定中のCORBAの新しい非同期呼び出しの規格[7]では、IDLコンパイラがスタブを生成する際にインタフェースの持つオペレーションごとに、同期、遅延同期、非同期の3種類のメソッド定義を生成させることで、通常のメソッド呼び出しと同じ方法で遅延同期呼び出しを実行することが可能になっている。

  しかし、この方式では、そのオペレーションで本当に遅延同期呼び出しが行われるかどうかにかかわらず、すべてのオペレーションについて遅延同期呼び出しのメソッド定義を生成する必要があるため、スタブが非常に大きくなるという欠点がある。

4.5 ビルドに関する問題点

IDLファイルから生成されたスタブやスケルトンのソースコードをコンパイルして、サーバとクライアントにそれぞれ必要なオブジェクトをリンクする作業は、通常はmakeを利用して行う。

複数のIDLファイルを用いる場合には、生成されるファイルの数が多くなり、IDLファイルと生成されるファイルの依存関係が増え、アプリケーションとファイルの依存関係も複雑になるため、Makefileを記述する作業はかなり煩雑である。サーバやクライアントにリンクしなければならないオブジェクトは、クライアントでは利用する分散オブジェクト、サーバでは提供する分散オブジェクトによって変化するため、それをMakefileに追随させる作業も必要になる。

C++の場合にはIDLファイルを一つにまとめることで、生成されるファイルの数を減らし、Makefileの記述を簡略化することができる。しかし、先に述べた

ように、IDL ファイルで定義されいてる内容が多くなると、生成されるファイルのサイズが大きくなり、生成されたファイルやそれを取り込んでいるアプリケーションのコンパイルにかかるコストが大きくなる。また、IDL ファイルが修正されると、生成されるファイル全体が上書きされるため、再コンパイルにかかる時間も長くなる。そこで、IDL ファイルは`#include`を利用してインターフェイスを単位に分割し、IDL コンパイラを起動するときには`#include`した内容を生成しないオプションを指定することで、出力されるファイルを分割する必要がある。

Java の場合には、IDL コンパイラは実際に利用するかどうかにかかわらず、IDL ファイルで定義されたインターフェイスやデータ型ごとに、その型に対応するクラスの他に、Helper クラスと Holder クラスを生成するので、非常にファイルの数が多くなる。生成されたファイルをコンパイルする作業を make を利用して行う場合には、Makefile に生成されたファイルの中から実際に利用するものをすべて指定しなければならない。IDL ファイルの内容が変更されると、生成されるファイルの名前が変化することもあり、その場合には Makefile を修正する必要がある。

Java では実行時に必要なクラスファイルが自動的に読み込まれるので、C++のようにクライアントとサーバに必要なオブジェクトを指定する必要はないが、実際に稼働させるときにはサーバのホストとクライアントのホストに必要なクラスファイルを配布する必要がある。クラスファイルの数は非常に多いためこの作業は非常に煩雑である。

Java の場合には、依存するファイルを自動的にコンパイルする機能を持った IBM の jikes[3]という Java コンパイラや、Inprise の JBuilder に備わっている関連するファイルを JAR(Java Archive) ファイルにまとめる機能を利用することで、上で述べた問題点はかなり改善することができる。

5 Interface Repository とリフレクションを利用した開発環境

CORBA の開発手順の中で、比較的手間のかかる IDL の記述、スタブ/スケルトンの生成、ビルドの手順を、Interface Repository(IR) を利用して簡略化する方法を考える。

5.1 IR を利用した IDL コンパイラ

IDL ファイルには定義するインターフェイスとそのインターフェイスの利用するすべてのユーザ定義型の定義が含まれていなければならない。このために構成管理の手間が増えることについては先に述べた通りである。

この問題を解決するために IR を利用することを考える。まず、IDL コンパイラは IDL ファイルから直接スタブやスケルトンを生成するのではなく、IDL ファイルで定義されている内容を IR に書き込むだけにする。必要なファイルをすべて IDL コンパイラに掛けて完全な情報が IR に格納された段階で、IR の内容に基づいてスタブやスケルトンを生成するようにすれば、一つの IDL ファイルにすべての定義を記述したり、`#include`を用いて定義を取り込むファイルを指定したりする必要がなくなる。格納された情報に基づいて、完全な定義を格納した IDL ファイルを生成するツールを用意すれば他の CORBA の実装との互換性も損なわれることもない。

この環境の元で、分散オブジェクトを実装したり利用したりする際に、IDL コンパイラの生成したスケルトンやスタブを用いるのではなく、IR に格納された情報をプログラムから直接取り込むことが可能になれば開発手順がかなり簡略化できる。これを実現する方法を述べる前に、まずスタブやスケルトンの必要性について考察する。

5.2 なぜスタブやスケルトンが必要なのか

スタブやスケルトンは IDL ファイルに記述された内容をプログラムに取り込むために用いられる。スタブやスケルトンを用いずに実行時に IDL ファイルの情報を取り込みながら動作するようにプログラムを記述するのは、プログラミング言語上の制約から非常に難しくなっている。

先に延べたようにスタブには、IDL で定義されたユーザ定義型に対応するプログラミング言語のデータ型の定義が含まれている。静的な型システムを持つ言語で

[3]http://www.research.ibm.com/jikes/

は、プログラムのコンパイル時にデータ型を定義したプログラムが必要になる。

ユーザ定義型を Any 型で扱う際にもスタブが必要となる。ユーザ定義型と Any 型の間の相互変換を行う際に、C++ではスタブで定義されている<</>>演算子が Java ではスタブで定義されている Helper クラスの `insert/extract` メソッドが用いられる。スタブを利用しない場合には、ユーザ定義型ごとに Any 型との相互変換を記述しなければならない。

スタブには、SII を実行するために必要な分散オブジェクトに対応するローカルなクラス (スタブクラス) の定義が含まれている。スタブクラスには、ローカルオブジェクトへのメソッド呼び出しを分散オブジェクトに対するリクエストに変換する操作が含まれている。このクラスを利用しない場合には、分散オブジェクトのオペレーションを呼ぶたびに、リクエストに変換する操作を DII を利用して記述しなければならない。

スケルトンには、クライアントからのリクエストを分散オブジェクトを実装したクラスのメソッド呼び出しに変換するクラス (スケルトンクラス) が含まれている。このクラスには、リクエストに含まれているオペレーションの名前を元に、それを実装しているメソッドを呼び出す操作が含まれている。スケルトンを用いない場合には、DSI を利用してインターフェイスごとにこの操作を記述しなければならない。

スタブには、プログラミング言語で用いられるデータ型とネットワーク中を流れる表現形式— CDR (Common Data Representation) を相互に変換する操作が含まれている。スタブを利用しない DII や DSI を用いたプログラミングでは、Any 型を介してデータを扱うので CDR を意識する必要はないが、Any 型とユーザ定義のデータ型の相互変換は必要である。

ここまでに挙げてきた、スタブが存在しない場合に必要になるプログラムを、オペレーションごと、インターフェイスごと、データ型ごとに記述するのではなく、それぞれの要素に依存しない単一のプログラムで記述することは、プログラミング言語の制約から一般には不可能である。

これを実現するには、実行時にデータ型の構造を調べる操作や、その名前をあらわす文字列を使ってメソッドを呼び出す操作を記述できなければならない。さらに、IDL で定義されたユーザ定義型をプログラムに取り込んだり、分散オブジェクトに対応するクラス定義を取り込むには、実行時に新しいデータ型を生成する操作を記述できる必要がある。

これらの操作を記述する能力は Linguistic Reflection として知られている [8]。Linguistic Reflection を持つプログラミング言語はいくつか存在する。例えば、Java 1.1 以降で利用できる Reflection API は Linguistic Reflection の一部を提供するものである。Perl や Python といったスクリプト言語や、Common Lisp などの Lisp 系の言語では完全な Linguistic Reflection を提供している。

そこで、完全な Linguistic Reflection を提供しているプログラミング言語 Python を利用して、スタブやスケルトンを必要としない CORBA の開発環境を実現する方法を考えてみる。

5.3 Python と Fnorb

5.3.1 Python とは

Python はインタプリタ型で対話的に実行可能なオブジェクト指向言語である [9] 。

クラスが型のひとつとして存在し、どのクラスに属さない関数を定義することができるという点で、Python は純粋なオブジェクト指向言語ではない。また、一般的なオブジェクト指向言語とは異なり、リストや連想配列といった高水準のデータ構造をクラスライブラリではなく、言語に組み込まれたデータ型としてサポートしている。これらのデータ型を操作するための簡便な構文も用意されており、複雑な処理を非常に短いプログラムで記述することができる。

クラスは多重継承をサポートしており、親クラス間でベースクラスが重複している場合には、必ずマージされる仕様になっている。これは CORBA の IDL の仕様と同じである。また、クラスは独立した名前空間を持ち、その内部でさらにクラスを定義することができる。さらに、モジュールという機構で名前空間の分離をサポートしている。Python のプログラムを格納したファイルが一つのモジュールであり、ディレクトリを利用して階層的なモジュールを構成することができる。オブジェクトモデルや名前空間については、CORBA の IDL との親和性が高いと言える。

5.3.2 Pythonの提供するリフレクション能力

Linguistic Reflection Python は Linguistic Reflection が可能な言語である。Linguistic Reflection はプログラム自身に関するプログラミングを可能にする能力、つまり、プログラムが定義しているデータ型や関数やクラスを調べる、変更する、さらにそれらを新たに生成することができる能力を指している [8]。

Python ではクラスもオブジェクトであり、クラスオブジェクトの持つ属性にアクセスすることで、クラス定義の内容を調べることができる。クラスで定義されているメソッドや属性は、名前と定義を対応付ける連想配列として格納されており、クラスオブジェクトの `__dict__`属性でこの連想配列にアクセスすることができる。連想配列の内容は書き換え可能なので、新たに属性やメソッドを追加することが可能である。また、クラスの名前、親クラスの一覧は、それぞれ`__name__`、`__bases__` という属性でアクセスすることができる。これらの属性の値は変更することができないので、クラス名を変更したり、親クラスを追加/削除することはできない。

関数やクラス定義を新たに生成する際には、文字列の形で生成した任意のプログラムを実行できる exec 文を利用する。文字列をプログラムとして実行可能な言語はさほど珍しくはないが、Python の exec 文は exec 文が実行される名前空間を自由に設定できるという点で少し風変わりである。Python ではクラスを定義する文を普通に実行すると、実行した名前空間にクラスが定義されてしまう。この機能を利用して既存のクラスや既存のモジュールを名前空間として指定することで、クラスの中でクラスを定義する、モジュールにクラス定義を追加するといった操作が可能になる。これは CORBA のスタブを実行時に生成する際に必ず必要になる機能である。

新しい関数やクラスは exec 文で生成することができるが、モジュールは Python のプログラムを格納したファイルを import 文で取り込むことによって生成されるので、これから定義を追加するためのまったく新しいモジュールを exec 文で生成することはできない。その代わりに、Python では import 文の実装に使われている関数群を imp モジュールを介してプログラムから利用できるようにしており、その中の `new_module()` 関数を利用して新しいモジュールを生成することができる。

Bihavioural Reflection Linguistic Reflection は Reflection によって実現されることの一部を取り上げたものである。Reflection 自体は本来、「計算システムが、自分自身の構成や計算過程に関する計算を行うこと」と定義されている [10]。本来の意味で Reflection が可能なプログラミング言語では、プログラムそのものではなくプログラムの実行環境をプログラムから観測したり、変更することが可能になっている。プログラム自身を調べたり変更したりする能力は、その結果として自然に得られるものである。

実際に Reflection の利用例を見てみると、プログラム自身を調べたり変更したりする能力だけで十分であったり、むしろその方が重要であることも多い。そこで [8] では、この能力を本来の意味の Reflection と区別して Linguistic Reflection と呼び、プログラムの実行環境に関する Reflection を Behavioral Reflection と呼んでいる。

Python では Behavioral Reflection も可能である。プログラムの実行状態を表す、名前空間とスタックフレームはそれぞれ Python の連想配列と Frame オブジェクトとして表現されており、組み込み関数を利用してプログラムから現在の名前空間とスタックフレームを参照することができる。また、インタープリタの動作を変更する方法として、クラスに関する振る舞いと import 文の実装を変更する方法が提供されている。

クラスの振る舞いの変更は、特殊な名前のメソッドを定義することで行われる。たとえば、クラスを定義する際に`__getattr__`と`__setattr__`メソッドを定義すると、そのクラスのインスタンスに対して定義されていない属性の参照やメソッド呼び出しが行われたときのインタプリタの振る舞いを記述することができる。さらにクラスについては、クラスを定義する、つまりクラスオブジェクトを生成するときの動作を変更する方法も用意されている [11]。

import 文はその実装が組み込み関数`__import__`として公開されており、この組み込み関数の定義を置き換えることで、import 文の振る舞いを変更することができる。新しい`__import__`文の実装に必要な関数もすべて imp モジュールを介して利用することができる。

これらの機構を利用すると、import 文にモジュール名ではなく URL を指定できるようにして、URL が

指定された場合にはネットワークを介してモジュールを読み込むといった機能を実現することもできる。

5.3.3 Fnorb

Python に対応した CORBA の実装としては、CRC for Distributed Sytems Technology によって開発された Fnorb[12] がある。Fnorb は Python 用の CORBA 2.0 の実装であり、Python の値を CDR(Common Data Representation) に変換する部分と、IDL のパーザは C 言語で記述されているが、他はすべて Python で記述されている。DII や DSI を含む CORBA 2.0 で規定されている API はすべて実装されており、Interface Repository および NamingService の Python による実装も含まれている。

5.4 スタブ/スケルトンの自動生成の実現

最初に、分散オブジェクトを利用するクライアント側の開発手順を IR とリフレクションを利用して簡略化する方法を考える。

5.4.1 スタブの自動生成と取り込み

CORBA では、分散オブジェクトのオブジェクトリファレンスの実装方法として IOR(Interoperable Object Reference) と呼ばれる方式を規定している。この IOR の最初のフィールドには、分散オブジェクトのインターフェイスを表す Type ID が格納されることになっている。規格では Type ID として具体的にどのような値が格納されるかは規定されていない上に、空にしておくことも認められているが、ほとんどの CORBA の実装で Type ID としてインターフェイスの Repository ID が用いられている。

したがって、クライアントが NamingService などから入手したオブジェクトリファレンス (IOR) をプログラミング言語のオブジェクトに変換する前に、Repository ID を用いて IR に問い合わせることで、インターフェイスの操作に必要なスタブを生成するために必要な情報が得られる。Python では実行時に文字列として生成したプログラムをコンパイルして、名前空間に取り込むことができるので、この段階でスタブを生成して取り込むことで、プログラマの手をまったく煩わせずに必要なスタブをクライアントに取り込むことができる。

オブジェクトリファレンスを入手した段階で、スタブとして必要な定義をすべて生成して取り込むこともできるが、実際に分散オブジェクトのオペレーションを呼び出すまで必要ではない定義も多い。分散オブジェクトのオペレーションを呼び出す前に本質的に必要なのは、オペレーションの引数として用いられているユーザ定義型の情報だけである。そこで、最初はユーザ定義型の情報だけを取り込んで、実際にオペレーションを呼び出したときに、そのオペレーションの実行に必要な定義を生成して取り込むというアプローチも可能である。このアプローチを用いると、最初にオペレーションを呼ぶときのコストが大きくなるが、多くのオペレーションを持つインターフェイスのごく一部をテストする場合には、こちらのアプローチを選択することでスタブの生成コストを小さくすることができる。

CORBA Messaging 前に述べたように deffered synchronous operation は現状では DII でしか実行できないため、プログラミングの手間が大きくなっている。この問題を解決する CORBA Messaging という規格では、インターフェイスの持つすべてのオペレーションについて、スタブを生成するときに通常の SII 用のメソッドおよび、二つの非同期通信モデル用のメソッドの 3 種類を生成することで、SII でも非同期通信ができるようになっている。しかし、この規格に基づいてそのまま実装してしまうと、スタブが非常に大きくなるという問題が生じる。

現状では CORBA Messaging の実装はまだ存在しないが、実装されるときにはおそらく IDL コンパイラのオプションとして CORBA Messaging のサポートの有無を指示できるようにしてスタブのサイズが不必要に大きくなるのを押さえるようになるだろう。現状でも十分煩雑なスタブの管理がますます煩雑になるのだ。

この問題を回避する方法として、先にのべた実際にそのオペレーションが使われたときに初めてスタブを生成するアプローチを用いることができる。

Repository ID が利用できない場合について IOR に Type ID が格納されている場合には、自動的に取り込む情報を決定することができるが、Type ID が空の場合や正確な情報が格納されない場合には、明示的に

スタブを取り込む操作を記述する必要がある。その際に、従来のようにIDLコンパイラでスタブを生成してそれをimport文で取り込むのではなく、利用するインターフェイスのRepository IDを提示すると、必要な定義をIRから自動的に生成して取り込むAPIを提供するというアプローチが考えられる。

Pythonではプログラムにモジュールを取り込むimport文の実装がPytonの関数として公開されているので、この関数の定義を入れ替えることにより、import文の動作をPython自身を用いてカスタマイズすることができる。この機構を利用すると、importの対象としてRepository IDが指定された場合には、IRから必要な情報を取り出してスタブを生成して取り込むようにimport文の動作をカスタマイズすることができる。

この方法には、プログラマはPytonのモジュールと同じ感覚で、インターフェイスのRepository IDを扱うことができるという利点がある。また、IRに格納された定義を直接扱うことができるので、ファイル単位でIDLの定義を扱うことから生じる不便さからも解放される。

5.4.2 スケルトン自動生成の導入方法

IDLコンパイラによって生成されたスケルトンは、分散オブジェクトを実装するクラスの上位クラスとして用いられている。こにIRを利用したスケルトンの自動生成を導入するために、上位クラスとしてスケルトンクラスを指定する代わりに、実装するインターフェイスのRepository IDを指定できるようにするというのも一つのアプローチである。Pythonではクラスが定義されるときの動作を、やはりPython自身でカスタマイズすることができるので、これを実現することは可能である。

もう一つのアプローチとしては、スケルトンクラスを指定せずにインターフェイスを実装するクラスを定義して、定義したクラスと実装するインターフェイスのRepository IDを指定するとスケルトンクラスを自動的に生成して、それを組み込んだ新しいクラス定義を生成するAPIを提供する方法が考えられる。Pythonではクラスもまたオブジェクトであり、クラスオブジェクトを操作することでクラス定義の内容を操作することができるので、調べたクラスとスケルトンクラスを組み合わせて、新しいクラスを生成することができる。

このアプローチでは、その過程でインターフェイスを実装しているクラスの実装ミスをIRの内容に基づいて検査することができるという利点がある。Pythonは動的な型を持つ言語なので、実際に運用するまでわからないミスも多いのだが、単なる実装し忘れ程度は検出可能であるし、属性に対応するアクセスメソッド(`_get_`属性名 / `set_`属性名)の実装ミス程度は十分検出可能である。

このアプローチのもう一つの利点は、ほとんど同じ仕様で名前だけが異なる複数のインターフェイスを実装する際に、すべてのインターフェイスの実装に同じクラス定義を用いることができる点である。従来の実装方法では、インターフェイスを実装するクラスには、そのインターフェイスに対応するスケルトンクラスを継承させなければならないため、同じクラス定義を複数のインターフェイスの実装に用いることはできないが、このアプローチではスケルトンクラスを指定する必要がないので、それが可能になる。

リフレクションを利用した属性の実装 Fnorbではインターフェイスの持つ属性は`_get_`属性名 と `set_`属性名 のメソッドにより実装することになっている。しかし、Pythonではオブジェクトの属性にアクセスされたときの振る舞いをカスタマイズするための機構が用意されているので、それを利用してインターフェイスの属性をそのままPythonの属性と対応させることも可能である。

クライアント側では、オブジェクトの属性の読み書きに対応して、その属性に対応する分散オブジェクトのget/setオペレーションを呼び出すように属性の振る舞いをカスタマイズすればよい。

サーバ側では、`_get_`属性名 / `set_`属性名が呼ばれたときに、もしそれが実装されていなければ、そのまま属性名の部分を使ってオブジェクトの属性にアクセスするという仕組みを、属性アクセスに関するカスタマイズを利用して実現することができる。

5.5 リフレクションを利用したAPI

リフレクションはスタブやスケルトンを生成する手間を減らすだけでなく、CORBAの複雑なAPIをより簡単に利用できるようにすることも可能である。特に、型の情報を操作するCORBAのAPIについては、

リフレクションを利用することでプログラミングを簡略化できる余地が多い。

5.5.1 型キャスト

先にのべた通り、C++や Java におけるオブジェクトリファレンスを取得する API の返り値は CORBA::Object 型であり、オペレーションを実行する前にスタブクラスの_narrow メソッドを用いて、スタブクラスのオブジェクトに変換しなければならない。

Fnorb では、オブジェクトリファレンスを取得する API が、IOR に含まれている Repository ID を元に、その Repository ID に対応するスタブクラスのスタブクラスのオブジェクトを返すので、プログラマが明示的に型の変換を記述する必要がない。これは、あらかじめ Repository ID とスタブクラスの対応表を用意しておいて、IOR の Repository ID に対応するクラスのインスタンスを生成することで実現されている。これを実現する際には、リフレクションを提供するプログラミング言語の持つ、1) クラスがオブジェクトとして扱える、2) 実行時に任意のクラスのインスタンスを生成できる、という二つの性質が用いられている。

5.5.2 Dynamic Any API

CORBA には、オブジェクトリファレンスを含む任意のデータ型を扱うことができる Any 型が用意されている。Any 型は、CORBA のデータ型を表現する TypeCode と、そのデータ型の値を表すバイト列を組にしたものである。

TypeCode とバイト列の対応を正しく扱うために、Any 型に値を格納したり、値を取り出したりする際には、C++の場合には Any クラスと値の型の間に定義された<</>>演算子を、Java の場合には値の型の Helper クラスを使用する。Any 型でユーザ定義のデータ型の値を扱う際には、そのデータ型に関する<</>>演算子の定義や Helper クラスの定義が必要なのでスタブを取り込む必要がある。

CORBA 2.2 では、スタブを用いずに Any 型を扱うことができる DynAny という API が定められている[4]。この API を利用するとプログラミング言語の型システムに依存せずに、Any 型に格納されている値を扱うことができる。たとえば、Any 型に格納されている任意の構造体の name メンバを取り出すプログラムは図 4 のように記述することができる。

```
CORBA::Any any_val;
CORBA::String_var name;
...
CORBA::DynAny_ptr dyn_any =
    orb->create_dyn_any(any_val);
CORBA::DynStruct_ptr dyn_struct =
    DynStruct::_narrow(dyn_any);
CORBA::release(dyn_any);

// 構造体のメンバを調べる
CORBA::Boolean found = False;
do {
    CORBA::String_var member_name =
       dyn_struct->current_member_name();
    found = !strcmp(member_name, "name");
} while (!found && !dyn_struct->next());

if (found) {
    CORBA::DynAny_var dyn_member =
       dyn_struct->current_component();
    name = dyn_member->get_string();
}
```

図 4: DynAny の例

Python では動的に型を生成することができるので、Fnorb では分散オブジェクトから Any 型の値を受け取る際に、Any 型に格納された TypeCode を元に対応する Python のデータ型を生成して、値をその型に変換して返すようになっている。この機構により、スタブがあるなしに関わらず、図 4 と同じ動作をするプログラムは Fnorb では一行で記述することができる。

```
name = any_val.value().name
```

DynAny には Any 型に格納されたユーザ定義型をスタブなしで解釈するための API だけではなく、スタブなしでユーザ定義型の値を Any 型に格納するための API も用意されている。たとえば、以下の IDL で定義された構造体の値をスタブなしで生成するプログラムは、図 5 のように記述することができる。

```
struct MyStruct {
    long member1;
    boolean member2;
}
```

[4] ObjectDirector では DynAny は利用できない

```
using CORBA;
StructMemberSeq mems(2);
Any any_val;

// struct MyStruct を表す TypeCode を生成する
mems[0].name = string_dup("member1");
mems[0].type =
  TypeCode::_duplicate(CORBA::_tc_long);
mems[1].name = string_dup("member2");
mems[1].type =
  TypeCode::_duplicate(CORBA::_tc_boolean);
CORBA::TypeCode_var struct_tc =
  orb->create_struct_tc("IDL:MyStruct:1.0",
                        "MyStruct", mems);
// DynAny を利用して struct MyStruct 型の値を
// 生成する
DynStruct_ptr dyn_struct =
  orb->create_dyn_struct(struct_tc);
dyn_struct->insert_long(10000);
dyn_struct->insert_boolean(true);
// Any 型に変換する
any_val = dyn_struct->to_any();
```

図 5: DynAny の例 2

この API を Python の属性に値を代入するときの振る舞いをカスタマイズするためのメソッド __setattr__ を利用して簡略化すると、図 6 のように記述することが可能になる。

```
dyn_struct = orb.create_dyn_struct()
dyn_struct.member1 = CORBA.long(10000);
dyn_struct.member2 = CORBA.boolean(true);
any_val = dyn_struct.to_any();
```

図 6: リフレクションを利用して簡略化した DynAny

この簡略化した DynAny API では、create_dyn_any() メソッドによって DynStruct のインスタンスが生成されて、そのインスタンスの属性を変更するときに、あらかじめ DynStruct クラスで定義しておいた __setattr__ が呼び出される。この __setattr__ は、代入された属性の名前と値を引数として呼び出されるので、渡された名前と値の型から構造体の型を表す TypeCode を生成しつつ、インスタンスに新しい属性と値を挿入して構造体型の値を作成していく仕組みになっている。

6 関連研究

6.1 LuaORB

LuaORB は Lua というリフレクションを提供しているプログラミング言語を CORBA に対応させたものである [13]。LuaORB は、存在しないメソッドが呼ばれたときのプログラミング言語の振る舞いを、リフレクションを利用してカスタマイズして、分散オブジェクトのオペレーション実行させる形で実装されている。

ただし、著者の CORBA に関する理解が甘く設計上の欠陥がある。メソッドを呼び出すときに Interface Repository にアクセスしてインターフェイスに関する情報を取得するため、引数にユーザ定義型を取るオペレーションを扱うことができない。また、DII に相当する API を提供していないので、特定のインターフェイスに依存しないアプリケーションを記述することができない。

6.2 CorbaScript

CorbaScript は CORBA のアプリケーションを記述するために新たに設計されたオブジェクト指向スクリプト言語である [14]。オブジェクトモデルや名前空間のモデルは Python によく似ている。言語仕様はそのものは普通のオブジェクト指向言語だが、処理系に CORBA のアプリケーション開発を簡略化するための機構が組み込まれている。

CorbaScript の処理系では、プログラミング言語の名前空間と IR の名前空間が結合されており、プログラミング言語側で未定義の名前にアクセスすると、その名前に対応する IR に格納された情報が自動的にプログラムに取り込まれる。分散オブジェクトの実装する際には、インターフェイスの名前とそれを実装するクラスのインスタンスの対応を指定するだけでよい。したがって、事前にスタブやスケルトンを生成したり、それを取り込む操作を記述する必要は全くない。

欠点としては、まったく新しい言語であるため、既存のクラスライブラリを利用できないことや、言語を覚える学習コストが大きいことが挙げられる。

6.3 ILU

ILU(Inter-Languange Unification)[15] は、様々なプログラミング言語で記述されたモジュール間のインタラクションを可能にすることを目的として開発された、CORBA の ORB と同様な機能を提供するシステムである。最近では、モジュールの仕様記述に CORBA の IDL を利用可能にしたり、通信プロトコルとして CORBA の IIOP を利用可能にするなど、CORBA2.0 に準拠する方向に進んでいる。

ILU は Python に対応しており、Python の import の機構をカスタマイズすることで、スタブを取り込む際に import 文でスタブのモジュールではなく、IDL ファイルを指定することが可能になっている。import 文に IDL ファイルを指定すると、IDL コンパイラを起動してスタブを生成しそれを取り込むところまで、カスタマイズされた import 機構が自動的に処理する。

この機構によって軽減されるのは IDL コンパイラを起動する手間だけである。IDL ファイルに構文誤りがある場合には例外が発生して実行が停止するので、改めて IDL コンパイラを起動してエラー検査を行う必要がある。先に述べた開発手順からすると、プログラムを適切に配置して起動する段階から、IDL ファイルの記述の段階まで差し戻されることになり、かえって手間が増えることになる。

6.4 Caffeine

Caffeine は、サーバクライアント共に Java を使用することを前提に CORBA のアプリケーション開発を容易にするための環境であり、VisiBroker for Java[5] に含まれている。

この環境では、分散オブジェクトのサポートするインターフェイスを IDL ではなく、Java の `interface` を用いて記述する。記述したインターフェイスを Java のコンパイラでコンパイルして、生成されたバイトコードを元に `java2iiop` というツールでスタブやスケルトンを生成することができる。クライアントとサーバの実装方法は通常の CORBA のアプリケーション開発と同様である。

分散オブジェクトのオペレーションで用いることができるデータ型は、基本的には CORBA のデータ型に対応づけることができるものだけである。そのため、データ型として利用できる Java のクラスは、以下の制限を満たすものだけである。

- `final` かつ `public`
- 実装の継承 (`extend`) を使っていない
- 属性がすべて `public`

CORBA との互換性は失われるが、上記の制限を満たさないクラスをデータ型として利用することもできる。その場合には、そのクラスのインスタンスは Java の serializer によってバイト列に変換され、CORBA のバイト列 (Octet Sequence) として扱われる。この形式を正しく扱うには、サーバクライアントともに Caffeine を利用する必要がある。

サーバとクライアントが両方とも Java の場合には RMI という選択肢もある。Caffeine の場合は、Java 以外の言語で書かれた分散オブジェクトと Caffeine を利用した分散オブジェクトの両方を扱うことができる他に、VisiBroker for Java 用の CORBAservices を利用できるという利点がある。

Caffeine を利用することで、既存の Java のクラスを分散オブジェクト化する際に、IDL ファイルを記述する手間を減らすことができるが、開発手順そのものが簡略化されるわけではない。

7 おわりに

本報告書では、最初に CORBA のアプリケーション開発の手順を概説し、その問題点として

- サーバを記述するための API の不備
- IDL で定義された情報を管理するインターフェイスリポジトリの不備
- CORBA のオブジェクトモデルと、プログラミング言語のオブジェクトモデルのずれ
- 遅延同期呼び出しの煩雑さ
- ビルドの手順の煩雑さ

について指摘した。

本報告書ではこれらの問題点のうち、最後のビルドの手順の煩雑さに着目し、この問題を解決するために、

インターフェイスリポジトリとリフレクションの可能なプログラミング言語を利用して、IDL ファイルをすべてインターフェイスリポジトリを中心に管理し、スタブやスケルトンの生成と取り込みはアプリケーションの実行時に自動的に行われる開発環境の構成方法について論じた。

さらに、この開発環境を前提に、リフレクションの可能なプログラミング言語持つ、プログラミング言語の振る舞いをカスタマイズする機能を用いて、遅延同期呼び出しの API をスタブの肥大化を招くことなく簡略化する方法や、スタブを用いずにユーザ定義型を扱う DynAny API を簡略化する方法について論じた。

本報告書で述べた内容では、リフレクションの可能なプログラミング言語の持つ、動的にアプリケーションの振る舞いを変更できる特性が十分生かし切れていない。今後はこの点について検討を進めていきたい。たとえば、本報告書で述べた環境と、動的に振る舞いを変更できる特性を活用することで、IDL ファイルの小規模な変更に対して、アプリケーションの実行を止めずに、かつアプリケーションを変更することなく、IDL ファイルの変更を吸収する仕組みを実現することは可能なはずである。また、オペレーションの実行される頻度に応じて、生成されるスタブの性質を変更する、たとえば速度優先、サイズ優先といったことも可能になるはずである。

本報告書では、リフレクションを利用した開発環境を Python を用いて実現することについて述べてきたが、Java を用いて同様の環境を実現することについても検討を進めていきたいと考えている。Java は Python と比較すると限定されてはいるが、リフレクションの能力を持っているので、この報告書で述べたスタブとスケルトンの自動生成レベルであれば、十分実現可能であると予想している。

参考文献

[1] Object Management Group. *The Common Object Request Broker: Architecture and Specification Revision 2.2*, February 1998. formal/98-02-01.

[2] Object Management Group. *CORBAservices : Common Object Services Specification*, November 1997. formal/98-07-05.

[3] Object Management Group. *Common Facilities Archiecture*, November 1995. formal/97-06-15.

[4] 富士通株式会社. FUJITSU ObjectDirector プログラミングガイド, February 1998.

[5] Inprise Corporation. *VisiBroker for Java 3.3 Programmer's Guide*, 1998.

[6] Microsoft Corporation. *Microsoft SDK for Java 3.1*, 1998. http://www.microsoft.com/java/sdk/31/.

[7] Object Management Group. *CORBA Messaging*, May 1998. OMG TC Document orbos/98-05-06.

[8] Graham Kirby, Ron Morrison, and David Stemple. Linguistic reflection in java. *Software: Practice and Experience*, Vol. 28, No. 10, pp. 1045–1077, 1998.

[9] Python language website. http://www.python.org.

[10] 渡部卓雄. チュートリアル リフレクション. コンピュータソフトウェア, Vol. 11, No. 3, pp. 5–14, May 1994.

[11] Guido van Rossum. Metaclasses in python 1.5. http://www.python.org/doc/essays/metaclasses/.

[12] Martin Chilvers. *fnorb version 1.0.* CRC for Distributed Systems Technology, February 1999. http://www.dstc.edu.au/Fnorb.

[13] Roberto Ierusalimschy, Renato Cerqueira, and Noemi Rodriguez. Using reflexivity to interface with CORBA. In *IEEE International Conference on Computer Languages (ICCL'98)*, May 1998.

[14] Laboratoire d'Informatique Fondamentale de Lille and Object-Oriented Concepts, Inc. *OMG CORBA Scripting Language RFP Revised Submission*, December 1998. OMG TC Document orbos/98-12-09.

[15] Bill Janssen, Mike Spreitzer, Dan Larner, and Chris Jacobi. *ILU 2.0alpha14 Reference Manual.* Xerox Corporation, 1998. ftp://ftp.parc.xerox.com/pub/ilu/2.0a14/manual-html/manual_toc.html.

SD'96 ワークショップの宿題

山崎利治

はじめに

話は旧聞も旧聞であるが SD' 9 6 ワークショップに出席した．そこでの課題の一つに,「グラフの最短路を求める算法を ML で書け」があった．この集会は実に素晴らしく，場所は富士を見ながらの温泉であり, 酒は熊谷さんが用意の加賀ものであり, 全く完璧であった．したがって呑むのに忙しく，課題を考える余裕はなかった．

しかし，伊藤さんがわたしのノートパソコンでも Haskell を使えるようにしてくださったので遅れ馳せながらプログラムしてみようと思い立った．ML と Haskell は詳しくは違うがおなじような関数型言語だから大目に見てほしい．以下は不出来なプログラマの奮闘記でご笑覧ください．

1 最短路問題

周知の問題ながらやはりそれを述べた方がいいだろう．そうなら，いっそ形式的な記法 Z に頼ってみよう．Z と関数型プログラミングを同時に勉強しようというつもりである．ごく普通に提示することにしてグラフや路などの術語を用意する．グラフの辺集合型 *Edges* と点集合型 *Vertices* を与集合型としてまず定義しておく．以下順にスキーマを提示する．スキーマ *Graph* は自己閉路も多重辺ももつ（有向）グラフである．

$[Edges, Vertices]$

Graph

$V : \mathbb{F}(Vertices)$
$E : \mathbb{F}(Edges)$
$from, to : E \to V$
$w : E \to \mathbb{N}$

Z の記述単位はスキーマであり, 上のような E 形の箱で書く．箱は抽象データ型あるいはオブジェクト指向のクラスの定義である. 箱の真ん中の横線の上, 宣言部に状態やその上の演算の名前と型を書く．横線の下, 公理部にはそれらの性質や制約条件を書くことになっている．スキーマは宣言の他に型や述語としても使う．

Graph には点集合 V と辺集合 E と写像 *from*,*to*,*w* がある．$from, to : E \to V$ は $from, to$ が E から V への写像（関数）であることを,$w : E \to \mathbb{N}$ は w が E から自然数 $\mathbb{N}$ への写像であることを示している．$from, to, w$ はグラフの辺に対してそれぞれその始点, 終点, 辺の長さを与える関数である．w を重み関数ともいう．このスキーマは公理部になにも書いていない例である．

つぎの一連のスキーマは以降の進行のための補助的な定義である．

*Graph*1

Graph
$\_adjv\_ : V \leftrightarrow V$
$\_adje\_ : E \leftrightarrow E$

$adjv = to \circ from^{\sim}$
$adje = from^{\sim} \circ to$

$V \leftrightarrow V$ は V と V の直積の部分集合, つまり, V と V との関係を表す．*adjv*,*adje* はそれぞれ点と点が，辺と辺が隣接していることを表す関係である．隣接とはつぎのような意味である．u *adjv* v は u を始点，v を終点とする辺 e が存在すること，また,e *adje* f は e の終点と f の始点が同一であることを意味している．$from^{\sim}$ は *from* の逆関係を表している．

$W == \mathbb{P}\,\mathrm{seq}\,Edges$

$\mathrm{seq}\,X$ は集合 X の要素を 0 個以上並べた有限列全体を表す．$\mathbb{P}\,X$ は集合 X の部分集合の全体である．W は辺列の集合で, グラフ上の路を定義するための準備である．以下に出現する $\vdash$ 形のスキーマは公理記述で, 大域的変数の導入のためのに用いる．

$$\_ \subseteq \_ : Graph \leftrightarrow Graph$$

$$\begin{aligned}
&\forall g1, g2 : Graph \bullet \\
&\quad g1 \subseteq g2 \Leftrightarrow \\
&\qquad g1.V \subseteq g2.V \land g1.E \subseteq g2.E \land \\
&\qquad g1.from = (g1.E) \lhd g2.from \land \\
&\qquad g1.to = (g1.E) \lhd g2.to
\end{aligned}$$

これは部分グラフを定義している．$R \lhd f$ は関数 f の定義域を R に制限したものである．

$$walk : Graph \to W$$

$$\begin{aligned}
&\forall g : Graph;\ \tau : \mathrm{seq}\, Edges \bullet \\
&\quad \tau \in walk\ g \Leftrightarrow \\
&\qquad \mathrm{ran}\,\tau \subseteq g.E \land \\
&\qquad \forall i : \mathrm{dom}\,\tau \bullet \\
&\qquad\quad i \neq \#\tau \Rightarrow \tau(i)\ adje\ \tau(i+1)
\end{aligned}$$

これはグラフ上の歩道の集合を定義したものである．歩道とは辺列で，辺列中のすべての隣り合う２辺は隣接している（点を共有する）ものである．

$$closed : Graph \to W$$

$$\begin{aligned}
&\forall g : Graph;\ \tau : \mathrm{seq}\, Edges \bullet \\
&\tau \in closed\ g \Leftrightarrow \\
&\quad \tau \in walk\ g \land from(head\,\tau) = to(last\,\tau) \\
&\quad \#\tau > 0
\end{aligned}$$

これはグラフの閉じた歩道の集合を定義する．閉じた歩道は少なくとも 1 辺を含むものとする．$head\,\tau, last\,\tau$ は列 τ の最先頭要素，最後尾要素を指す．

$$elementary : Graph \to W$$

$$\begin{aligned}
&\forall g : Graph;\ \tau : \mathrm{seq}\, Edges \bullet \\
&\tau \in elementary\ g \Leftrightarrow \\
&\quad \tau \in walk\ g \land \#\,\mathrm{ran}\,\tau = \#\tau
\end{aligned}$$

グラフの基本歩道を定義するが，基本歩道とは歩道をつくる辺も点もすべて異なるものをいう．

$$circuit : Graph \to W$$

$$\begin{aligned}
&\forall g : Graph \bullet \\
&circuit\ g = closed\ g \cap elementary\ g
\end{aligned}$$

閉路の定義である．

$$acyclic : \mathbb{P}\, Graph$$

$$\begin{aligned}
&\forall g : Graph \bullet \\
&g \in acyclic \Leftrightarrow circuit\ g = \emptyset
\end{aligned}$$

非巡回グラフの定義である．

$$connected : \mathbb{P}\, Graph$$

$$\begin{aligned}
&\forall g : Graph \bullet \\
&g \in connected \Leftrightarrow \\
&\quad \forall u, v : g.V \bullet \exists p : walk\ g \bullet \\
&\qquad \{u, v\} \subseteq \mathrm{ran}\, p
\end{aligned}$$

連結グラフを定義している．これであとで使う木がやっと定義できる．連結かつ非巡回なグラフを木という．

$$tree \mathrel{\widehat{=}} acyclic \cap connected$$

もう一つスキーマを書く．課題を述べるための最後である．$path(u, v)$ は点 u から v への歩道集合を与える．

Path

$$\begin{aligned}
&Graph1 \\
&path : V \times V \nrightarrow W \\
&\hline
&\forall (u, v) : V \times V;\ \tau : \mathrm{seq}\, Edges \bullet \\
&\quad \tau \in path(u, v) \Leftrightarrow \\
&\qquad \tau \in walk(\theta Graph) \land \\
&\qquad u = from(head\,\tau) \land v = to(last\,\tau)
\end{aligned}$$

スキーマの中に *Graph* とあるのはこのスキーマにスキーマ *Graph* を取り込んだもの，すなわち *Path* の中に *Graph* の宣言部と公理部をそれぞれ書き込んだものである．S をスキーマ名とするとき θS は S の束縛を意味する．たとえば $\theta Graph.V$ は $g : Graph \bullet g.V$ を意味している．

以上で最短路問題（一始点問題）が提示できる．グラフが与えられたとき，特定の点 $r?$ からすべての点への最短距離を与える関数 $s!$: $V \to \mathbb{N}$ を求めよという問題である．スキーマ *ShortestPathProbrem* がそれである．

R^+ は関係 R の推移的閉包である．関係 R が推移的というのは $(x, y) \in R$ かつ $(y, z) \in R$ ならば $(x, z) \in R$ となるときにいい，R を含む最小の推移的関係を R の推移的閉包という．また，$r?$ は演算 *ShortestPathProbrem* に対する入力であり，$s!$ は出力であることを意味する Z の約束ごとである．

ShortestPathProbrem

$$
\begin{array}{|l}
\hline
Graph1 \\
largeno : \mathbb{N} \\
r? : V \\
s! : V \to \mathbb{N} \\
\hline
largeno > \Sigma[E]\ w\ E \\
\forall v : V \bullet \\
\quad v = r? \Rightarrow s!(v) = 0 \\
\quad v \neq r? \wedge \neg\ (r?\ adjv^{+}\ v) \Rightarrow \\
\qquad s!(v) = largeno \\
\quad v \neq r? \wedge r?\ adjv^{+}\ v \Rightarrow \\
\qquad s!(v) = min\{p : path(r?, v) \bullet \\
\qquad\qquad \Sigma[E]\ w\ (\mathrm{ran}\, p)\} \\
\hline
\end{array}
$$

ここでΣは汎用スキーマとしてつぎのように定義している．参照するときには実引数を与える．

$$
\begin{array}{|l}
\hline
[X] \\
\Sigma : (X \to \mathbb{N}) \to \mathbb{F}\, X \to \mathbb{N} \\
\hline
\forall f : X \to \mathbb{N};\ A : \mathbb{F}\, X;\ a : X \setminus A \bullet \\
\quad \Sigma\, f\, \emptyset = 0 \\
\quad \Sigma\, f\, (\{a\} \cup A) = f(a) + \Sigma\, f\, A \\
\hline
\end{array}
$$

$X \setminus A$は集合差（Xの要素でAに属さないものの全体）を表す．

2 貪欲系

さて，周知のダイクストラによる最短路算法は貪欲算法であった．この算法自体きわめて単純明快であるが，ここでは一般的な貪欲算法の復習からはじめて愚鈍にプログラムしたい．貪欲算法は多くの最適化問題に役立つ簡単な算法であるが，どんな問題に適用できるのかについては明確ではない．貪欲系 (greedoid) はこれをはっきりさせる一つの手がかりになる．

貪欲系は行列系 (matroid) の弟分であるが，早速定義しよう．つぎのスキーマは引数をもち，貪欲系を定義する．Xは勝手な型の集合である．利用に際しては引数に適切な集合を与えて具体化することになる．有限集合 X と，X の部分集合族 Fs が二条件 $G1, G2$ をみたすとき，(X, Fs) を貪欲系というわけである．貪欲系がさらに遺伝的，すなわち，Fs に属する任意の F の勝手な部分集合 Y がまた Fs に属するとき行列系と呼んだ．

貪欲系ではつねに $\emptyset \in Fs$ である．Fs の要素を可能集合 (feasible set) といい，極大可能集合を基底 (basic set) という．スキーマの $fcont$ は可能可拡大要素集合 (feasible continuation) のことで，可能集合に対して，それに追加するとまた可能集合になるような要素の全体を意味する．d はあとで考える重み関数であるが便宜上ここで定義しておく．

また，$comp$ は貪欲系の重み関数の性質で，この性質をもつ目的関数の最小値問題が貪欲算法によって解けるというものである．貪欲算法の実行中のある時点においてxの選択が最適となるなら，それ以降の任意の時点においても，そこでもxが選択可能ならその選択が最適になるという性質である．

Greedoid$[X]$

$$
\begin{array}{|l}
\hline
Fs : \mathbb{F}(\mathbb{F}\, X) \\
fcont : Fs \to \mathbb{F}\, X \\
basic : \mathbb{F}\, Fs \\
comp : \mathbb{P}(Fs \to \mathbb{N}) \\
d : Fs \to \mathbb{N} \\
\hline
(G1)\ \forall F : Fs \bullet F \neq \emptyset \Rightarrow \\
\qquad \exists x : X \bullet x \in F \wedge F \setminus \{x\} \in Fs \\
(G2)\ \forall F1, F2 : Fs \bullet \\
\qquad \#F1 = \#F2 + 1 \Rightarrow \\
\qquad \exists x : X \bullet x \in F1 \setminus F2 \wedge \\
\qquad\quad F2 \cup \{x\} \in Fs \\
\forall F : Fs \bullet fcont(F) = \\
\qquad \{x : X \setminus F \mid F \cup \{x\} \in Fs\} \\
\forall F : Fs \bullet F \in basic \Leftrightarrow \\
\quad \forall x : X \bullet x \in X \setminus F \Rightarrow F \cup \{x\} \notin Fs \\
\forall d : Fs \to \mathbb{N} \bullet d \in comp \Leftrightarrow \\
\quad \forall F1, F2 : Fs;\ x \in X \bullet \\
\qquad [F1 \subseteq F2 \wedge x \in X \setminus F2 \wedge \\
\qquad F1 \cup \{x\} \in Fs \wedge F2 \cup \{x\} \in Fs \wedge \\
\qquad (\forall y : X \bullet y \in fcont(F1) \Rightarrow \\
\qquad d(F1 \cup \{y\}) \geq d(F1 \cup \{x\}))] \\
\qquad \Rightarrow \forall z : X \bullet (z \in fcont(F2) \Rightarrow \\
\qquad d(F2 \cup \{z\}) \geq d(F2 \cup \{x\})) \\
\hline
\end{array}
$$

貪欲系の一例に根付木がある．グラフ (V,E) を一つ固定し $r \in V$ とする．r を含むグラフの部分木を考える．部分木を辺集合とみなし，グラフの部分木全体を T_r と書けば，(E, T_r) は貪欲系になる．

$X \neq \emptyset$ を木とすれば，この木は r と異なる葉 v を少なくとも一個はもっている．e を v を含む辺とすると $X \setminus \{v\} \in T_r$ だから $(G1)$ が成立する．

二つの木 X, Y で $\#X > \#Y$ とする．そうすると X に含まれ Y に含まれない辺 e が存在する．したがって e の端点で X に属し Y に属さない v も存在する．いま X の根 r から v への唯一の路を考えれば，この上には Y に属さない根から一番近い辺 f がある．そこでは $Y \cup \{f\} \in T_r$．すなわち $(G2)$ が成立する．

最短路問題を意図しているからもう少し条件の付いた貪欲系を考える．つぎは局所半順序貪欲系 (local poset greedoid) である．$G3$ が余分に付いた条件である．

LocPosGrd[X]

$Greedoid[X]$
$ep : Fs \to \mathbb{F}\, X$
$route : \mathbb{F}\, Fs$
$\rho : X \times Fs \nrightarrow Fs$

$(G3) \forall F1, F2, F3 : Fs \bullet$
$F1 \subseteq F3 \land F2 \subseteq F3 \Rightarrow$
$F1 \cup F2 \in Fs \land F1 \cap F2 \in Fs$
$\forall F : Fs;\ x : X \bullet x \in ep(F) \Leftrightarrow$
$x \in F \land F \setminus \{x\} \in Fs$
$\forall F : Fs \bullet$
$F \in route \Leftrightarrow \exists_1 x : F \bullet ep(F) = \{x\}$
$\forall x : X;\ F : Fs \bullet x \in F \Rightarrow$
$\forall G : Fs \bullet [\rho(x, F) = G \Leftrightarrow$
$G \in route \land G \subseteq F \land$
$ep(G) = \{x\}]$

$ep(F)$ は可能集合 F の端点 (end point) の集合である．可能集合の端点とは，それを取り去っても可能集合となるよう要素のことである．端点を使って道 (route) を定義する．

可能集合 F が道であるとは F の端点が単集合になるときにいう．可能集合 F 上での F の要素 x への道が $\rho(x, F)$ である．これらの用語の意味はグラフの根付木全体のつくる貪欲系の例で明らかだろう．

3 最小値問題と貪欲算法

貪欲系 (X, Fs) 上で目的関数 $d : Fs \to \mathbb{N}$ の最小値問題

$$min\{F : Fs \mid F \in basic \bullet d(F)\}$$

は，d が許容 $comp$ なら貪欲算法によって解ける．この詳細 [BZ] は省略するが，算法はつぎのスキーマに示すものである．可能集合として空集合から出発し目的関数に照らし要素を選択，これが基底集合になるまで反復する算法である．

貪欲算法の謂れは $find$ にある．つまり，常に局所的な判断に基づく選択が大局的にも最適になる（貪欲選択性）というものである．$(\mathbf{let}\ x == E_1 \bullet E)$ は E に出現する x を E_1 に置換した値を指す．またここに出てくる関数 $loop$ はつぎのようなものである．$loop\ f\ (x, y)$ は $(x, y)(x, y$ ともに集合) に対して f を y が空集合になるまで反復適用するという関数で，f が y の濃度をその適用ごとに減少させることが重要である．

Greedy[X]

$Greedoid[X]$
$find : Fs \nrightarrow X$
$step : Fs \times \mathbb{F}\, X \nrightarrow Fs \times \mathbb{F}\, X$
$greedy : Fs$

$\forall F : Fs;\ x : X \bullet$
$fcont(F) \neq \emptyset \Rightarrow$
$x = find(F) \Leftrightarrow$
$\forall y : fcont(F) \bullet$
$d(F \cup \{x\}) \leq d(F \cup \{y\})$
$\forall (F, Y) : Fs \times \mathbb{F}\, X \bullet$
$Y = fcont(F) \land Y \neq \emptyset \Rightarrow$
$step(F, Y) =$
$(\mathbf{let}\ F1 == F \cup \{find(F)\} \bullet$
$(F1, fcont(F1))$
$greedy = loop[Fs, \mathbb{F}\, X]\ step\ (\emptyset, fcont(\emptyset))$

[X, Y]

$loop : (X \times Y \to X \times Y) \to X \times Y \to X$

$\forall f : X \times Y \to X \times Y;\ (x, y) : X \times Y \bullet$
$loop\ f\ (x, y) = \mathbf{if}\ y = \emptyset\ \mathbf{then}\ x$
$\mathbf{else}\ loop\ f\ (f(x, y))$

この算法を命令型で書けばつぎのようになる．

PROCEDURE *Greedy*(X; Fs; d) : Fs
PROCEDURE *find*($a : Fs$) : X
PRE $fcont(a) \neq \emptyset$
POST $find(a) \in fcont(a) \land$
$d(a \cup \{find(a)\}) =$
$min\,\{y : fcont(a) \bullet d(a \cup \{y\})\}$
PROCEDURE *fcont*($a : Fs$) : $\mathbb{F}\, X$
POST $fcont(a) =$
$\{x : X \setminus a \mid a \cup \{x\} \in Fs\}$

```
BEGIN
  F := ∅;
  WHILE  fcont(F) ≠ ∅
  DO
    x := find(F);
    F := F ∪ {x}
  OD
  return(F)
END
```

4 分岐貪欲系と最短路問題

先に例として挙げたグラフ上の根付木がつくる貪欲系を分岐貪欲系 (branching greedoid) と呼んでいる．これは局所半順序貪欲系でもある．最短路問題はこの局所半順序貪欲系の重み関数 d に対する最小値問題である．この重み関数はグラフの重み関数 w から構成する．すなわち, グラフの重みを辺の長さとみて根から木の各点への距離の総和を与える関数 d がそれである．

$$d(F) = \sum_{x \in F} (\sum_{y \in \rho(x,F)} w(y))$$

そうするとこの関数は許容である．

$d \in comp$ を主張するためにはつぎをいえばよい．

前提：

$$F_1, F_2 \in Fs;\ \ F_1 \subseteq F_2;$$

$$x \in fcont(F_1) \cap fcont(F_2);$$

$$\forall y \in fcont(F_1) \Rightarrow d(F_1 \cup \{x\}) \leq d(F_1 \cup \{y\})$$

を仮定してつぎが導ければよい．

結論：

$$\forall z \in fcont(F_2) \Rightarrow d(F_2 \cup \{x\}) \leq d(F_2 \cup \{z\})$$

一般に $F \cup \{x\} \in Fs$ なら,

$$d(F \cup \{x\}) = d(F) + d(\rho(x, F \cup \{x\}))$$

が成立する．$\rho(x, F \cup \{x\})$ は可能集合 $F \cup \{x\}$ 上の x への道であった．また $z \in fcont(F_2)$ とし $y \in \rho(z, F_2)$ で $y \notin F_1$ とすれば,

$$\rho(y, F_1 \cup \{y\}) \subseteq \rho(z, F_2) \qquad \clubsuit$$

である．ところで,

$$d(F_1 \cup \{x\}) = d(F_1) + d(\rho(x, F_1 \cup \{x\}))$$

$$d(F_1 \cup \{y\}) = d(F_1) + d(\rho(y, F_1 \cup \{y\}))$$

であるからつぎが成立する．

$$\begin{aligned} d(\rho(x, F_1 \cup \{x\})) &\leq d(\rho(y, F_1 \cup \{y\})) \\ &\leq d(\rho(z, F_2 \cup \{z\})) \end{aligned}$$

はじめの不等式は直前の二式と前提から, またつぎの不等式は $\clubsuit$ から導ける．つぎに,$z \in fcont(F_2)$ に対して,

$$\begin{aligned} d(F_2 \cup \{z\}) &= d(F_2) + d(\rho(z, F_2 \cup \{z\})) \\ &\geq d(F_2) + d(\rho(x, F_1 \cup \{x\})) \\ &= d(F_2 \cup \{x\}) \end{aligned}$$

すなわち $d \in comp$ である．

以上は, 根から各点にいたる最短路を含む極大木が存在し，この木が分岐貪欲系上で d を最小にし, 逆に d を最小にする任意の極大木は根から各点への最短路を含むという事実の反映にすぎない．

ここで分岐貪欲系をスキーマに整理しておこう．

BrGrd

$g : Graph$
$lpg : LocPosGrd[g.E]$
$r : g.V$

$lpg.Fs =$
$\quad \{t : tree \mid t \subseteq g \wedge r \in t.V \bullet t.E\}$
$\forall F : lpg.Fs \bullet$
$\quad lpg.d(F) =$
$\quad\quad \sum_{x \in F} (\sum_{y \in \rho(x,F)} g.w(y))$

算法 *Greedy* の *Greedoid* を *BrGrd* として *ShortestPathProblem* に適用しプログラム化したいが, このためにはひと工夫必要である．

5 ダイクストラ算法へ

最短路問題に対する貪欲系は木を考えればいい．その要素は辺であった．それを点に着目した算法に変形したい．つぎがその試みである．展開の便宜上補助的な定義を与える．

Aux

$$
\begin{array}{l}
Graph1 \\
cut, ens : \mathbb{F}\, V \to \mathbb{F}\, E \\
nes : \mathbb{F}\, E \to \mathbb{F}\, V \\
nns : \mathbb{F}\, V \\
\hline
\forall v : V;\ e : E;\ U : \mathbb{F}\, V;\ F : \mathbb{F}\, E \bullet \\
\{e \in cut\ U \Leftrightarrow \\
\quad from\ e \in U \wedge to\ e \in (V \setminus U) \wedge \\
e \in ens\ U \Leftrightarrow \{from\ e, to\ e\} \subseteq U \wedge \\
v \in nes\ F \Leftrightarrow \exists e : F \bullet v \in \{from\ e, to\ e\} \\
\wedge\ v \in nns\ U \Leftrightarrow \\
\quad \exists u : U \bullet u\ adjv\ v
\end{array}
$$

関数 *cut* は, 点集合 U に対して U と $V \setminus U$ にそれぞれ端点をもつ辺の集合 (cut) を与える. *ens* は U の二点を両端点とする辺の集合を与える (edges in the node set). *nes* は辺集合 F に対してその辺の端点全体をあたえる (nodes in the edge set). *nns* は点集合 U に対して U の点と隣接する点全体を与える (neighbor nodes of the node set).

ここらでグラフの具体的な表現を考えたほうがいいようなので点に対して隣接点集合を与える関数を定義する.

Nbr

$$
\begin{array}{l}
Graph1 \\
nbr : V \to \mathbb{F}\, V \\
\hline
\forall u, v : V \bullet \\
\quad v \in nbr\ u \Leftrightarrow u\ adjv\ v
\end{array}
$$

貪欲算法に現れた関数で辺を引数にしているものを, 上で定義した補助関数を使って辺の引数に変更したい. *fcont* と *find* である. 辺を要素とした可能集合 F^e を点集合 F^v に $F^v = nes\ F^e$ として変換する. そうすると, つぎのように考えることができる.

$$
\begin{array}{c}
e \in fcont(F^e) \\
\Updownarrow \\
F^e = \emptyset \Rightarrow from\ e = r \wedge \\
F^e \neq \emptyset \Rightarrow e \in cut \circ nes\ F^e
\end{array}
$$

であるが, これがつぎのようになる.

$$
\begin{array}{c}
v \in fcont(F^v) \\
\Updownarrow \\
F^v = \emptyset \Rightarrow v = r \wedge \\
F^v \neq \emptyset \Rightarrow v \in (nes \circ cut\ F^v) \setminus \\
\qquad nes \circ ens \circ cut(F^v)
\end{array}
$$

$$
\begin{array}{c}
e = find(F^e) \\
\Updownarrow \\
e \in fcont(F^e) \wedge \\
\textbf{let}\ \Delta = \lambda f \bullet s!(from\ f) + w(f) \bullet \\
\Delta(e) = min\{f : E \mid f \in fcont(F^e) \bullet \Delta(f)\}
\end{array}
$$

は, またつぎのようになる（$\lambda x \bullet e$ は x を変数とし値 e をとる関数式である）.

$$
\begin{array}{c}
v = find(F^v) \\
\Updownarrow \\
v \in V \setminus F^v \wedge \\
s!(v) = min\{u : V \mid u \notin F^v \bullet s!(u)\}
\end{array}
$$

ここでつぎの関数を定義すればスキーマ *Dijkstra* がえられる（$d \lhd f$ は関数 f の定義域を $\mathrm{dom}\, f \setminus d$ に制限したものである）.

$$
\begin{array}{c}
update : V \times [V \to \mathbb{N}] \times [V^2 \to \mathbb{N}] \\
\qquad \to [V \to \mathbb{N}] \\
update(v, s, w) == (\{v\} \lhd s) \oplus \\
\{u : V \mid u \in nbr\ v \bullet u \mapsto \\
min(s(v) + w(u, v), s(u))\}
\end{array}
$$

スキーマのようなダイクストラ算法ができれば, これは集合を列に有限写像を対に書き直せば直ちに Haskell で書き下せる. それを付録に示す. 小さなプログラムなのでモジュールを使わなかった. また, 計算量も考えていない. モジュールやヒープなどを本来は使うべきである. この意味で未完成であるが取り敢えずの関数型プログラムの見本になるかと思ったのでお目に掛ける. 関数 *nbr* のような安易なものではなく, グラフの表現を真面目に考え, グラフ入力プログラムを工夫しなければならないが, 関数型プログラムのためのデータ構造についていい教科書 [CO],[RL] が出版されているのでこのような改良版の作成は易しいだろう.

$$
\begin{array}{l}
MIN : (X \to \mathbb{N}) \to (X \to \mathbb{N}) \\
\hline
\forall f : X \to \mathbb{N} \bullet \\
\quad \mathrm{dom}\, f \neq \emptyset \Rightarrow \\
\quad MIN\ f = \{x\} \lhd f \Leftrightarrow f(x) = min\,(\mathrm{ran}\, f)
\end{array}
$$

Dijkstra

$$
\begin{array}{l}
Nbr \\
r? : V \\
s! : V \to \mathbb{N} \\
largeno : \mathbb{N} \\
step : (V \to \mathbb{N})^2 \to (V \to \mathbb{N})^2 \\
initial : (V \to \mathbb{N})^2 \\
update : V^2 \times \mathbb{N}^2 \to \mathbb{N} \\
\hline
largeno > \#E * max\ w(\!|\ E\ |\!) \\
\forall (a,b,c,d) : V^2 \times \mathbb{N}^2 \bullet \\
a\ adjv\ b \Rightarrow \\
\quad update(a,b,c,d) = \\
\qquad \textbf{let}\ e == \mu\, e1 : E \bullet \\
\qquad \{from\ e1, to\ e1\} = \{a,b\} \bullet \\
\qquad min(c, d + w(e)) \\
\forall (f,g),(ff,gg) : (V \to \mathbb{N})^2 \bullet \\
\mathrm{dom}\, f \cup \mathrm{dom}\, g = V \land \\
\mathrm{dom}\, f \cap \mathrm{dom}\, g = \emptyset \Rightarrow \\
\quad step(f,g) = (ff,gg) \Leftrightarrow \\
\qquad \textbf{let}\ \{m \mapsto gm\} == MIN\ g; \\
\qquad\quad xs = ((\mathrm{dom}\, g) \setminus \{m\}) \cap nbr(m) \bullet \\
\qquad ff = f \oplus \{m \mapsto gm\} \land \\
\qquad gg = (\{m\} \lhd g) \oplus \\
\qquad \{x \in xs \bullet x \mapsto update(m, x, g(x), gm)\} \\
initial = (\emptyset, (\lambda\, x : V \bullet \\
\qquad \textbf{if}\ x = r?\ \textbf{then}\ 0\ \textbf{else}\ largeno)) \\
s! = loop\ step\ initial
\end{array}
$$

6 反省と言い訳

大変ごちゃごちゃしてしまった．ダイクストラ算法をもっと素直にプログラムすればよかったと思う．それにしてもこんなに長くなるとは想像しなかった．不明を恥じるだけだが言い訳も聞いてください．

1. 貪欲算法をよく理解したかった．
2. そのために行列系や貪欲系に親しみたかった．
3. 形式的仕様記述言語 Z を使ってみたかった．1，2のために厳密な記法は有効であるに違いない．
4. 算法展開とプログラム設計との関係を検討するために一般的な算法から特殊な算法を導出したかった．
5. バードたち [BM] の算法導出計算における重要な原理である thinning と貪欲系との関係を調べてみたかった（まだ理解できていない）．

おなじ課題を玉井先生が CafeOBJ で扱われている [TT]．そこでは課題をグラフ上の不動点問題として定式化し,CafeOBJ の直接実行によって解を得,また反復法の仕様を与えそれによる解も得られている．是非参考にしてください．

SD'96 ワークショップに出席させてくださった岸田孝一さん熊谷章さん，課題を提示し,忘れていたプログラミングを思い出させてくださった野呂昌満先生佐原伸さん伊藤昌夫さんらの世話役の皆さん,ghc を使えるようにしてくださった伊藤昌夫さんに心からお礼を申し上げます．ありがとうございました．

参考文献

[RB] R. Bird, *Introduction to Functional Programming using Haskell.* 2nd ed. Prentice Hall, 1998.

[BM] R. Bird and O. de Moor, *Algebra of Programming.* Prentice Hall, 1997.

[BZ] A. Björner and G. M. Ziegler, *Introduction to Greedoids.* in Matroid Application (ed. N. White). Cambridge, 1992.

[CLR] T. H. Cormen, et al., *Introduction to Algorithms.* MIT Press, 1990. 浅野哲夫他訳，アルゴリズムイントロダクション．（3冊） 近代科学社， 1995.

[CO] C. Okasaki, *Purely Functional Data Structures.* Cambridge, 1998.

[RL] F. Rabhi and G. Lapalme,*Algorithms: A functional programming approach.* Addison-Wesley, 1999.

[JMS] J. M. Spivey, *The Z Notation A Reference Manual.* 2nd ed., Prentice Hall, 1992.

[TT] T. Tamai, A unified approach to a class of fixed-point probrems on graphs by CafeOBJ. Proc. of the CafeOBJ Symp. '98. 1998.

[ST] S. Thompson, *The Craft of Functional Programming.* Addison-Wesley, 1996.

付録

最短路問題プログラム

```
module Main(main) where

vs = [1,2,3,4,5,6]
r = 1
nbr :: Int -> [(Int,Int)]
nbr 1 = [(2,7),(3,6)]
nbr 2 = [(1,7),(3,4),(4,2),(5,2)]
nbr 3 = [(1,6),(2,4),(4,2),(5,5)]
nbr 4 = [(2,2),(3,2),(5,3),(6,1)]
nbr 5 = [(2,2),(3,5),(4,3),(6,4)]
nbr 6 = [(4,1),(5,4)]

isin :: Int -> [(Int,Int)] -> (Bool,Int)
isin u []                              = (False,0)
isin u ((y1,y2):ys)   | u ==y1         = (True,y2)
                      | otherwise      = isin u ys

miN :: [(Int,Int)] -> (Int,Int)
miN = foldl1 lt    where
        lt (x1,x2) (y1,y2) | x2 < y2       = (x1,x2)
                           | otherwise     = (y1,y2)

delete :: (Eq a) => a -> [a] -> [a]
delete x xs = filter p xs     where
              p y = y /= x

loop :: (([a],[a]) -> ([a],[a])) -> ([a],[a]) -> [a]
loop f (xs,[])    = xs
loop f (xs,ys)    = loop f (f (xs,ys))

update :: (Int,Int) -> (Int,Int) -> (Int,Int)
update (x1,x2) (y1,y2) =
  let (b1,b2) = isin y1 (nbr x1)
  in if b1 then (y1,(min y2 (x2+b2))) else (y1,y2)

step :: ([a],[a]) -> ([a],[a])
step (xs,ys) =   let u  = miN ys
                 in (u:xs,(map (update u) (delete u ys)))

vv :: [(Int,Int)]
vv = map f vs        where
          f x | x == r         = (x,0)
              | otherwise      = (x,9999)
```

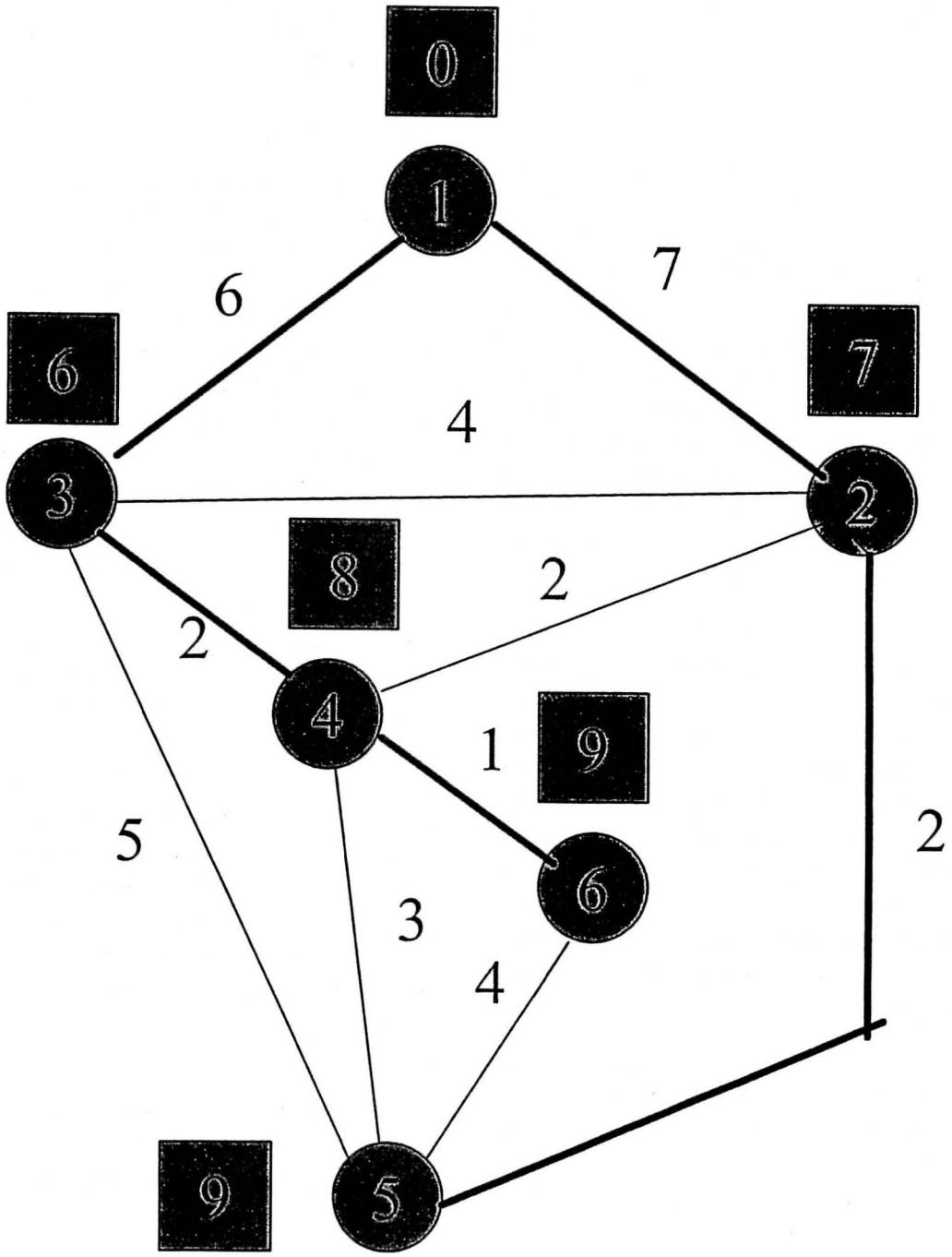
```
s :: [(Int,Int)]
s = loop step ([],vv)

main = print s
```

実行結果

```
[(5,9),(6,9),(4,8),(2,7),(3,6),(1,0)]
```

例題　次の図のような（無向）グラフに対してプログラムしている．丸印は点，そのなかの数は点番号，辺に付けた数は辺の長さ，箱の中の数は点１からの距離をそれぞれ表している．太い辺は極大根付木を示す．



ソフトウェア開発プロジェクト立案時のシミュレーション

花川 典子 松本 健一 鳥居 宏次
(奈良先端科学技術大学院大学)
E-mail {noriko-h,matumoto,torii}@is.aist-nara.ac.jp

1. はじめに

プロジェクト当初に立案されるソフトウェア開発プロジェクト計画の精度は，プロジェクトの不確定要素や管理者の経験に左右される．つまり，プロジェクト計画は，開発すべきシステムの詳細化がどのくらい進んでいるか，また，管理者が類似プロジェクト経験をどれだけ持っているか，に影響される．多くの管理者が，プロジェクト計画の精度に不安を抱えながら，プロジェクトを開始している[5]．

計画立案時にプロジェクトをシミュレーションすることには，さまざまな利点がある．シミュレーションとは，プロジェクトの状況，たとえば，ソフトウェアの種類や規模，開発者の特性などを入力情報とし，シミュレーションモデルにもとづいて，プロジェクトの進捗を計算し擬似実行することである．その結果，プロジェクトの将来，すなわち，納期や進捗を予測することができ，プロジェクト計画立案時の有効な判断材料を管理者に提供することができる．たとえば，プロジェクトに新技術のJAVA言語を適用するか，または，従来のC 言語で開発するかの判断を迫られるとき，両者をシミュレーションすることで，納期などの制約を検証することができる．

大学の研究成果でも，さまざまなソフトウェア開発シミュレーションモデルが提案されている[1][3][4]．それぞれのシミュレーションモデルは，特定の問題，たとえば開発者の学習による生産性の変化や平行作業による工数増加を予測するシミュレータとして，実行環境と共に提供されている．これらのモデルは，大学内の実験や企業の実績データを用いて，シミュレーション結果の妥当性を示している．しかし，シミュレータを実際に利用するためには，シミュレーションモデルを理解のために数学的知識を要求され，また同時に，シミュレーションモデルの入力値を決めるため煩雑な分析作業を必要とする．

また，市販されているプロジェクト管理のシミュレーションパッケージソフトウエアの多くは，プロジェクト管理者教育を目的としたもの[8]，または，プロジェクト管理用のデータ蓄積と蓄積データ分析中心であり[6][7]，現在計画中であるプロジェクトの予測のためのシミュレーションは難しい．

本稿では，ソフトウェア開発プロジェクト立案時にプロジェクトを擬似実行できるシミュレータを紹介する．このシミュレータは，開発現場のプロジェクト管理者の知識と論理的なシミュレーションモデルにもとづいて作成されている．その意味で管理者にとって使い易いシミュレータとなっている．つまり，通常の管理業務で使用する値，たとえば生産性などをシミュレータに与えるだけで，実プロジェクトをシミュレーションできる．モデルの理解や入力値を求めるための分析作業を必要としない．

また，このシミュレータは，論理的なシミュレーションモデルにもとづいて作成されているので，あらゆる状況のプロジェクトの擬似実行が可能となる．もし，シミュレータがノウハウの集まりにもとづいて作られているとすれば，可能なシミュレーションは，そのノウハウに該当する特定のプロジェクトに限定される．プロジェクト管理教育用パッケージソフトウェアのシミュレーションの多くが，これに該当する．このシミュレータで利用するモデルは，習熟を考慮したソフトウエア開発シミュレーションモデルとなっている（詳細は[1] を参照）．

さらに，プロジェクト管理者の知識と論理的なシミュレーションモデルにもとづく本シミュレータは，ブリッジシステムと呼ばれる環境の中で作成された．ブリッジシステムとは，大学の研究成果であるさまざまなモデルをソフトウェア開発現場に役立つツールへ変換するためのプラットフォームである．今回の開発にあたって，プロジェクト管理者の知識は，ブリッジシステム上で幅広くWeb アンケートによって収集された．したがって，このシミュレータでは，特定の企業やプロジェクトではなく，一般性の高いシミュレーション結果を求めることができる．反対に，プロジェクト管理知識を特定の企業やプロジェクトの過去の実績から得ることによって，特定の企業やプロジェクトに特化したシミュレーション結果を得ることも可能である．

以降，２章でブリッジシステムの簡単な紹介を行い，３章では作成したシミュレータ例の紹介と実行例を示し，４章でまとめと今後の課題について述べる．

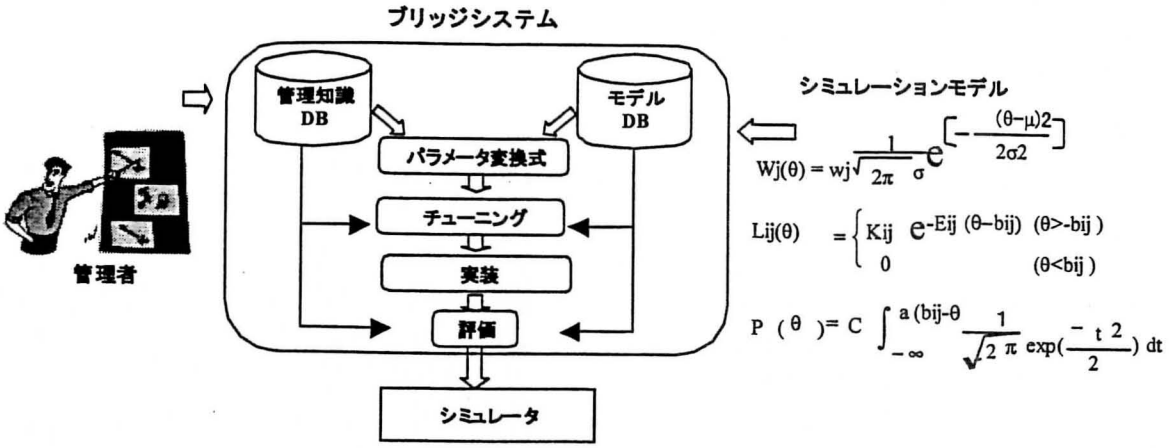


図1 ブリッジシステム

2. ブリッジシステム

図1にブリッジシステムの概要を示す．ブリッジシステムとは，モデルの情報とプロジェクト現場の情報をもとに学術的なモデルを実際のプロジェクトで利用可能なツールへ変換するためのプラットフォームである．ブリッジシステムは2つのデータベースと4つのステップから構成されている．以下それぞれについて説明する（詳細は[2] を参照）．

2.1 管理知識DB とモデルDB

2つのデータベースとは，管理知識DB とモデルDB とである．管理知識DB には管理用語や見積もり知識などが格納される．たとえば，管理者が通常使う用語，「生産性」や「納期」，「開発期間」などを定義して格納する．また，特定のプロジェクトの開発期間（実績データ，または，管理者の見積もり値）も，管理知識として格納する．モデルDB には，モデルの計算式やパラメータの意味などの情報が格納される．このモデルDB 作成作業には，モデルごとの深い理解が必要であるが，研究者がモデルDB を作成することで，プロジェクト管理者はモデルの理解という煩雑な作業から開放される．

特に，管理知識DB の作成には，プロジェクト管理者の知識，または，過去のプロジェクト実績データが活用される．もし，各企業の特殊な制約などを組み込みたいときには，過去のプロジェクト開発実績データを利用する．今回のシミュレータ作成では，特定の企業やプロジェクトに関するシミュレーションではなく，一般的なシミュレータ作成を意図したので，プロジェクト管理者の知識を幅広くWeb 上で収集し，統計処理を施して管理知識DB に格納した．具体的にはWeb 上のバーチャルプロジェクト（図2参照）に対する開発期間見積もり値を不特定多数のプロジェクト管理者たちに入力してもらった．個人差のために見積もり値にはばらつきがあるので，平均と分散より信頼区間を80 %として10 %以下と90 %以上を異常値として除外した残りを管理知識DB に蓄積した．

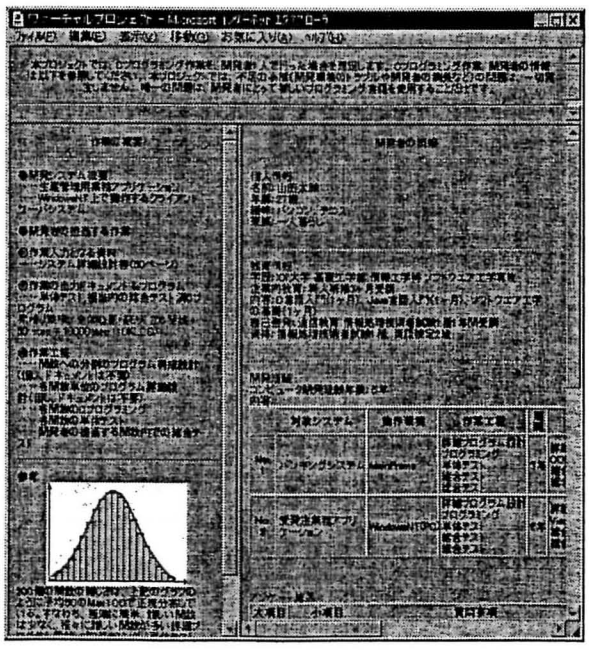


図 2 バーチャルプロジェクト

管理者	A	B	C	D	E	F	G	H	I
見積り期間(人月)	10	10	5	15	15	9	11	11	15

表1 バーチャルプロジェクト1の開発期間見積もり値

今回の管理知識蓄積にさいしては，78 人のプロジェクト管理者に見積もり入力を行ってもらい，62人の見積もり値を管理知識DB に格納した．具体的には，6つのバーチャルプロジェクトの開発期間見積もり値や開発者の特性の見積もり値などを，Web のア

ンケートシステムから入力していただいた．一部の結果を表１に示す．今回のシミュレータ作成ではこの値が管理知識となる．

2.2 ４つのステップ

パラメータ変換式作成，チューニング，モデルの実装と評価，の４ステップから構成される．まず，パラメータ変換式作成ステップでは，モデルDB のパラメータの意味と管理知識DB の管理用語からパラメータに関する質問を作成し，質問の回答値をモデルのパラメータ値へ変換する変換式を作成する．チューニングステップでは，前ステップで作成されたパラメータ変換式を，管理知識DB の開発期間見積もり値の一部と一致するように調整する．そして，実装ステップでは，モデルの式とパラメータ変換式をJAVA言語などで実装し，使用目的に合わせたユーザインタフェース部分も実装する．評価ステップでは，管理知識DB のチューニングで使用されなかった開発期間見積もり値を用いて，シミュレーションモデルの計算結果を評価する（詳細は[2] を参照）．評価の結果を図３に示す．図３の斜破線上にプロットされているならば，見積もり値とシミュレーション結果が一致することを意味する．χ２乗検定の結果，危険率５％で有意な差はないと判断できた[2] .

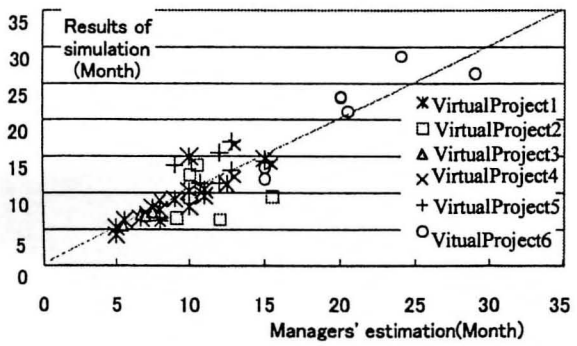


図３ バーチャルプロジェクト

3. シミュレータ

ブリッジシステムを適用して，習熟を考慮したソフトウェア開発シミュレーションモデルにもとづき，実プロジェクトのためのシミュレータを作成した例を，図４に示す．ブリッジシステム内のユーザインタフェース実装では，グラフ表示を行うことにしているが，もちろん使用目的に応じた計画表やバーチャートのユーザインタフェースも可能である．

また，ブリッジシステムプラットフォームでシミュレータを作成したことにより，シミュレータの入力項目が「生産性:KLOC/ 月」や「ソフトウエア開発規模:KLOC」などの，プロジェクト管理者にとって馴染みの深い入力項目内容と単位となった．シミュレーションモデルの計算式を理解し，パラメータの単位を気にしながら入力値を求めるよりは，はるかに容易にシミュレーションできるツールである．

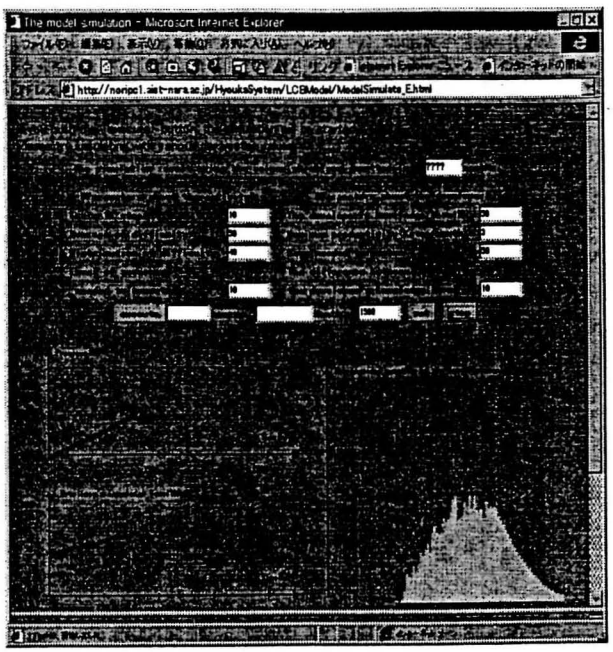


図 4　シミュレータ

このシミュレータを利用して２つの規模の異なるソフトウェアを３つの技術で開発した場合のシミュレーション例を図５，６に示す．シミュレーション例のグラフは前述のシミュレータの進捗グラフ部分に相当する．

3.1 10KLOC の詳細設計，製造，テスト作業

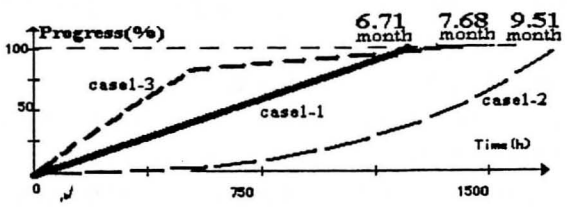


図５ 例１のシミュレーション

● **Case 1-1:** 従来技術

開発者が従来技術，つまり十分に習熟しているC言語や構造化技法を使って開発するプロジェクトを想定する．開発者の従来技術に対する知識は100% であり，生産性は1.5KLOC/ 月とする．

図５のCase 1-1 のラインは本プロジェクトのシミュレーション結果の進捗を示す．6.71 ヶ月で作業が終了し，進捗は一定の生産性で増加している．

● **Case 1-2:** 新技術

開発者にとって新技術であるコンポーネントウェアを使って開発するプロジェクトを想定する．開発者はコンポーネントを利用しての開発経験がなく，プロジェクトを実施するには学習が必要となる．ただし，学習後の生産性は従来技術より高い3.0KLOC/ 月を示す．

図５のCase 1-2 のラインは本プロジェクトのシミュレーション結果を示し，9.51 ヶ月で開発は終了する．前半では学習時間のために進捗が悪いが，習熟した後半には好転する．

● **Case 1-3: 習得が困難な技術**

習熟するために多くの学習や訓練を必要とする技術，たとえばオブジェクト指向開発技法で開発するプロジェクトを想定する．つまり，クラス図を作成することは比較的容易に習得できるが，再利用性の高いクラスを識別するためには多くの訓練が必要な技術である．開発者はクラス図を作成する知識をもっているが，再利用性の高いクラスを識別する能力には欠ける状態と想定し，また，再利用性の高いクラスを識別する技術を習得すると生産性が飛躍的に伸び，5.0KLOC を発揮できるプロジェクトと想定する．

図５のCase 1-3 のラインが本プロジェクトのシミュレーション結果を示し，開発終了までに7.68 ヶ月必要となる．前半では，開発者が既にもっている知識で作業が順調に進むが，中盤以降，知識不足と習得困難のために生産性が落ち込むことがわかる．

3.2 30KLOC の詳細設計，製造，テスト作業

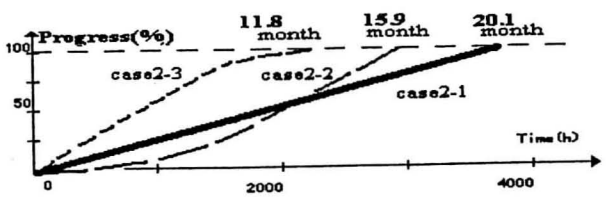


図６ 例２のシミュレーション

例２では，ソフトウェア規模が例１の３倍の30KLOC を想定する．他の条件は例１と同様とする．例２のシミュレーション結果を図６に示す．

Case 2-1(従来技術) では20.1 ヶ月，Case 2-2(新技術) では15.9 ヶ月，Case 2-3(習得困難な技術) では11.8 ヶ月それぞれ必要とする．

3.3 実行例の考察

例２において例１と最も異なる点は，Case 2-1(従来技術) の開発者が最も長い開発期間を必要とする点である．従来技術の低生産性の進捗に及ぼす影響が，学習に必要な時間の進捗に及ぼす影響よりも大きくなるためである．つまり，学習時間は規模にかかわらず一定であるが，作業時間は規模に影響される．結果として，全体の開発時間における学習時間の占める割合が小さくなり，学習が必要であるが高生産性を発揮できる技術の方が，全体の開発期間が短くなることを示している．このシミュレータでは，さまざまな技術を使った時のプロジェクトの規模による開発期間の相違を明確にすることができる．できるかぎり早くプロジェクトを終了する計画を作成するとき，管理者は，規模に応じて従来技術と新技術のどちらかを使えば早くプロジェクトが終了するかを知ることができ，技術選択判断の支援ができる．

4. まとめと今後の課題

ソフトウェア開発プロジェクトの計画立案時に管理者の判断支援ができるシミュレータの紹介をした．このシミュレータは，習熟を考慮したソフトウェア開発シミュレーションモデル[1] にもとづいて計算を行なっているので，新技術の導入を検討するのプロジェクトに有益な結果をもたらすことが期待できる．また，他のシミュレーションモデル[3][4] もそれぞれの特徴を生かすことで，実プロジェクトの計画立案時に役立つことができるであろう．

また本稿では，ブリッジシステムプラットフォーム上で論理的なシミュレーションモデルを実プロジェクトに有効なシミュレータに置き換えた．ブリッジシステムでは，管理知識DB が重要な役割を果たす．多くのプロジェクト管理者のWeb アンケート協力によって管理知識DB の作成ができ，ブリッジシステムを用いた本シミュレータ作成が可能となった．

今後は，ブリッジシステムの管理知識DB を企業のプロジェクト実績データにて作成する．これによって，企業の特性やプロジェクト固有の特殊性などを組み込んだプロジェクトシミュレーションが可能となるだろう．

謝辞

このシミュレータを作成するさいに重要な役割を果たすブリッジシステムの管理知識DB の構築にあたって，Web アンケートに協力していただいたプロジェクト管理者の方々に深く感謝いたします．

参考文献

[1] N. Hanakawa, S. Morisaki, K. Matumoto, “A Learning Curve Based Simulation Model for Software Development, ” Proceedings of 20th International Conference on Software

Engineering,pp.350-359,1998.

[2] 花川典子, 松本健一, 鳥居宏次, “学習収習熱を考慮したソフトウェア開発シミュレーションモデルの評価”, 電子情報通信学会技術研究報告書,KBSE98-28 ,pp.49-55,Nov.1998.

[3] H. Iida, J. Eijima, S. Yabe , K. Matsumoto, K. Torii, “Simulation model of overlapping development process based on progress of activities, ” Proceedings of 1996 Asia-Pacific Software Engineering Conference,pp.131-138,1996.

[4] S. Kusumoto, O. Mizuno, T. Kikuno, Y. Hirayama, Y. Takagi, K. Sakamoto, “A new software project simulator based on generalized stochastic, ” Proceedings of 19th International Conference on Software Engineering, pp.293-302,1997.

[5] 高橋宏士, “ソフトウエア工程管理技法”, ソフトウェア・リサーチ・センター,1991

[6] Knowledge Plan homepage, http://kke.co.jp/major/syskai/kp/kp-home.html

[7] Knowledge Pool homepage: http://www.flm.co.jp/japanese/m_contents/software/psim/1p.htm

[8] M1 for Windows homepage: http://www.mworld.or.jp/catalog/m1cata.html

ドナウ紀行

熊谷 章
(PFU)

7月6日

成田に午前8時頃に着く．NEX3号は満席であった．海外旅行をする人々が沢山いる．しかも，老若男女を問わない．日本が景気が悪いとはとても考えられない．Lufthans a にチェックインした．空いていて気持ちがよい．Igor がまだ来ないので，カウンターに近くに座って待つ．目の前を通る人々を見ながら，君の面影を探すが誰一人として似た人は通らない．

航空機の中で一悶着あった．座席を予約していたのに，物凄くデカイ奴が俺の座席を占有しているのだ．座ろうとするとスチュワデスを呼んだ．，ここは彼がが占有したので他の空いている座席に移って欲しいと彼女がのたまう．座席のはるか後ろを指差して向こうというのだ．頭に来たので，"Why should I change? I already reserved." と言ったら，OK，そこに座って下さいという．Igor も同じことを言われ，彼は最初素直に移動したが，俺が抗議して変わらなかったのを見て，彼も抗議した．そしたら，そのスチュワデスはIgor の抗議を受け入れた．その結果，そのデカイ奴は別の席に移動していった．当然であるが，世の中にはとんでもない奴がいるものだ．彼は，俺に一言，" 俺は足が長くて座るのが大変だから，こうしたのだ"，という言い訳をして去った．

機内では，"Semiotics for Beginners" を読む．結構，面白い．言語学と記号学の関係，メディア論，認識論，メタファ（隠喩）とメトニミ（換喩），記号と意味論，ソシュールとパースの記号論などなど，中々楽しむことができた．あとで，少し詳しくまとめたいと思っている．一昨日に作成した漢詩の推敲を行った．コンピュータとどこで接点を持つに至るかがキーでになると考え続ける．フィルムの一シーンを記号として捕らえている視点があるので，なんとなくスムーズに展開できそうだと，生来の楽観主義が顔を出す．

現地時間で午後三時ころFrankfurt に到着する．機長が挨拶で，夏のFrankfurt はベストシーズンだと力説した．しかし，外を見れば，凄い雨だし，暗くてとても良い季節だとは思えない．しかも，成田を出るときに，両替をしなかったのでお金がない．Sn ack Bar でビールも飲めない．仕方がないので，Igor と今夜から泊まるホテルにどうやっていくかを相談する．まず，メールからホテル名とアドレスを取り出す．二人共に名前も気にしていないことに愕然とする．なんと，名前は「ARCOTEL」らしい．これでは，まるで朝から晩まで酒を飲めと言っているようなものだ，と笑い合った．

飛行機が一時間位遅れているようだ．Frankfurt 発，Linz 行きである．乗客は20 名くらいで，皆ダークスーツを着ている．そして，皆がドイツ語を話している．それは当然のことであるのだが，とても不思議な気がした．Frankfurt 空港は工場のような作りである．いろいろなものがむき出しになっており，美的感覚がない．嫌いだ．「声なき声を聞き，形なき形を見る」の英訳を，"To hear a voice of silence and to see a shape of invisible" とし，Igor に見せたら簡単に，分かるよ，といわれた．少し，拍子抜けの感．

六時に，Linz に着く．雨は上がっている．タラップから降りたら，ゲートへは地べたを歩いて行く．まるで，中国の空港みたいだ．しかし，シンプルでよい．バッゲージクレームも殺風景で中国のそれとそっくりである．この表現は，別に中国を卑下しているから書くのでないことを断っておきたい．その証拠に，ここ20 年近くで30 回以上中国を訪問していることを示しておきたい．パスポートチェックも税関もなにもない．バッゲージを取り，歩いたらすぐ外に出てしまった．例によって，Igor は質問攻めに合い，暫く出て来ない．

車を借りるためにAVIS を探す．空港前のバラックにそれはあった．六社あったが，開店はAVIS だけだった．予約をしていたので，われわれにためだけに開けていたことが一目瞭然．早速手続きに入ったが，さっぱり要領を得ない．最後には，コンピュータがうまく動作しないので手書きにするという．コンピュータメーカを訊ねたら，ICL だった．コンピュータが動作しないので，レンタカーの契約値段も分からないという．ツーリストから，契約時には必ず値段を確認するように言われたが，どうしようもない．白紙の契約書にサインしてしまった．仕方がないので，予約時の価格を彼に示したが果たして結果はどうなるか．BMW に乗る積もりでいたが，予約したクラスの車は一台しかなく，スバルだという．変えて貰おうとしたが，デスカウントが無くなることと面倒そうだったので，スバルにした．

宿泊するホテルの場所が分からないので，地図が欲しいといったら，少し探してからないという．チェンジマネーをしなかったので，地図も買えない．尤も，近くには店が一軒もなかった．概ねの道順を聞き，後は走ることにした．いろいろあったがどうにか「ARCOTEL」に着いた．表玄関が工事中のため裏からしか入れない．薄暗いバーを抜けてフロントに入る．7 時頃であるが，外は明るい．ホテルは，リバーサイド．美しき青きドナウである．川面の色はまっ茶色である．部屋はリバービューであり，停泊している船や河岸を散歩している人々を見ることができた．ゆったりとした，風景に浸り心が和まされていくのがよく分かった．Linz にまたやってきたという実感が湧いた．

ホテルで二万円を1900 オーストリアシリングに替え，遅い夕食を取るためにセントラル方向に向かう．河は大きな渦をたくさん作りながらたゆとうと流れている．遠くから見れば，そう速く見えないが，近くからだと10 ノットくらいのスピードがあることが分かる．水深は相当あるのだろう．50 メートル近い客船が岸の停泊している．レストランが中にある．セントラルの広場を抜け，とあるビヤーガーデンに入る．中庭にあり，木々がうっそうと繁っている素敵な場所である．ビールとソーセージなどを味わう．そんなに美味しくないが，雰囲気がとてもよい．頭上の木を見たら，お馴染みのマロニエ（栃）である．なんと，このビヤーガーデンは夜の12 時まで開店しているとのこと．

プラハの街並みに少し似た通りを歩く．ゴシック風の教会，嬌声が聞こえる酒場，怪しげな女性の身体をデフォルメした看板がある謎の店，明るすぎるショーウィンドウ，人気のない繁華街，老人たちのたまり場，自転車の少女，二両編成のバス，，，，．異邦人には皆面白くみえる．暗闇の中のドナウを見ながらホテルに戻る．波音一つ立てないドナウの流れは異常だと感じた．

7月7日

7時頃，ドナウ川の岸辺を散歩する．泊まっているホテルは，ARCOTELという名前で川岸にある．名前を聞いただけで酔っ払ってしまう．ドナウ川のここでの川幅は300ｍ位にみえる．もっとも，目測はいかにいい加減であるかは，今年の2月に関門海峡で試し済みだ．200か300位にしか見えなかった関門海峡が実は800ｍ近くあったのだ．向こう岸の人々，ジョギング，自転車，犬連れなどよく分かる．近くの岸には，河下りか河上がりのレストラン付きの豪華船が停泊している．

先ず，目に付いたのは燕が優雅に飛んでいる姿である．盛岡の中津川や北上川で見たものに様子が似ているが，姿形は4倍くらい大きい．水面スレスレに快速をとばして見事な滑走を見せてくれる．視野には合計で約２０羽位しかいない．川全体の雰囲気は，ドナウと北上はよく似ている．流域の様子が平らかであり，岸辺が広く沢山の樹木，例えば柳，白楊，楓などがあり，水流が多く，うねりと渦を生み出しながら湧き出るように流れている．今年の初春に見た関門海峡の流れにも似ている．河の流れは意外に早く５から１０ノットはありそうだ．大きく開かれた視野の中を，街を通り抜け，平野を通り抜け，ゆったりと揺るやかに水が流れている．心からのどかな気分だ．川面を渡ってくる夏の風も気持ちよい．水の色はカフェラテのような茶色で，決っして美しき青きドナウではない．

岸辺には沢山の草花がある．バラ，西洋ウド，アザミ，イタドリ，レンゲソウ，西洋シロツメクサ，西洋スカンポ，ヒヤシンスに似た美しい紫の花，ペンペンクサ，，，，，．数えあげればきりがないほどだ．北上川も同じであるが，人工的な堤防がなく，自然な感じの広い川原と更に樹木に覆われた広い堤が目に美しい景観を作り出している．その中に，野の花が自然に自生している．自然でいい．後で，知ったことだが，この河岸はすべて人工的に改修されたもののようだ．1935 - 1945年の間の工事を記念した石碑を近くで見つけることができた．盛岡の中津川のイメージを主題として習作した詩がここでも当てはまる．

小燕御風　熊太

江平楊柳渺緑波
小燕御風入菱荷
清渓倚岸相為侶
乾坤一碧竹枝歌

現代文の読み下しで書けば次のようなものだろう．

小燕，風にのる　熊太

江平らかにして，楊柳，緑波に渺（ビョウ）たり
小燕，風にのり，菱荷（リョウカ）に入る
清渓，岸に倚り，ともに侶と為す
乾坤，一碧，竹枝の歌が聞こえる

（注釈）竹枝：その土地の風俗人情を民謡風に詠じたもの

近くにMusic Pavilionがあり，折りしもオーストリアの民族音楽らしき曲が流れて来る．

リバーサイドホテル，ARCOTELから下流に30分位歩き戻る．帰りは岸辺ではなく，土堤の上にある散歩道を歩く．沢山の大きな木々に囲まれた美しい径である．楓，桜，スモモ（葉っぱの赤い20mくらいの木），バラ，木瓜，姫リンゴ，紫の実を付けているヒバのような葉をした木など沢山ある．中でも，気に入ったのが青リンゴ大の実を付けている木瓜である．3メートルくらいの茨になっており，大きな実を沢山付けている．田舎の庭の木瓜を思い出し，とても懐かしくその実に触れてみた．雨上がりの爽やかでひんやりとした官能的な感触が返ってきた．生きている証なのだろうか．また，ここらにはサクランボの木が多くあり，佐藤錦のような薄赤い実が10粒位かたまって，緑の葉陰の中に見えるのも美しい．

ヨハネスケプラー大学のRisc-Linz研究所でのミーティングを終え，夕方6時頃にまた同じドナウの辺りを一人で逍遥した．姫リンゴを摘み，朝心を通わせた木瓜の実を摘み，ドナウの岸辺に依り，そのたわわな流れに乗せてあげた．それらが，流れに沿い，ブダペストを抜け，海まで辿り着け，という思いを載せて流した．海に着けば何なのだろう，と思うが，なぜかそうしたい気分だ．流す前に，木瓜の実から優しい言葉が伝えられてきたような気がした．それは，木瓜を握っている感触から知れた．ボケ .... .

朝と違って夕方は，燕が２，３羽しか飛んでいない．夕方といっても，まだ太陽は30度から40度の高さにある．燕の代わりに，セキレイ，カモメ，カモたちが群れをなしている．よく見ていたら小魚を取っているようだ．カモメが風を受けて優雅に跳ぶ姿は流石に美しい．本で学んだ飛び方，水面スレスレに風の反作用を最大限に活かす方法，つまりダイナミックプログラミング方式で飛んでいる．その脇の岸辺で，30代位の二人連れが熱烈な抱擁とキッスを執拗に続けている．まさに，"love is feeling, love is touch" である．周りを見れば，朝と違って二人連れが多く歩いている．開放的な自然環境では，人は本来の姿に戻ることができるのだろう．一人岸辺に佇んで時の中に入り込む．

午後7時過ぎになったら，突然，川上の方から風が起き吹いてきた．薫風で気持ちが良い．手をつないだ40代の二人連れが，岸辺の石段に座りこの日記を書いている私の様子に興味を持ったらしく，珍しそうに覗き込み，笑顔を返して通り過ぎてゆく．夕陽が射して，川面に一筋の極楽浄土への光の途ができた．その赤い光の中を，輝く途の上をカモメたちが飛び交う．この光の途に入ってゆけば，桃源郷に行けるに違いないという確信が生まれる．そんな気分にさせてくれる何かが，ここドナウにある．

Preliminary Call for Papers

論文 / 報告募集

ソフトウェア・シンポジウム 2000

2000 年 6 月 21 日(水) ～ 23 日(金)　於: 金沢市文化ホール(石川県金沢市)

主 催: ソフトウェア技術者協会 (SEA)

協 賛 (予定)

日本ソフトウェア科学会 情報処理学会 情報サービス産業協会

電子情報通信学会ソフトウェアサイエンス研究専門委員会

ソフトウェア・シンポジウムは, これまで, ソフトウェアに関連する産学の技術者, 研究者, 管理者が一堂に会し, 発表や議論を通して, たがいの経験や研究成果ならびに最新の話題を共有するための貴重な場を提供して来ました.

20 世紀最後の今年, このシンポジウムは第20 回を迎えます. この区切りのよい年に, 既存の製品や技術動向の可否またはその評論にとどまらず, それらとソフトウェア工学の基礎技術とのリンクを見直して、中身のある議論を展開したいと考えます. すなわち, ソフトウェアエンジニアリングをもって新たな世紀の社会基盤に貢献する準備をととのえたいと考えます.

今回の会場は, 古い歴史を持つ北陸の文化都市です. その美しい環境の中で, ソフトウェア技術の現状をしっかりと見すえ, 実践的な, がしかし夢のある議論を展開しましょう. ふるって論文, 報告および企画のご応募をお願いいたします.

シンポジウムは以下の企画で構成する予定です.

- 論文発表および報告

  募集する論文/ 報告にはつぎの 3 つの範疇があります.

  - 研究論文：新規性のある技術を提案する論文.
  - 経験論文：既知の技術の適用経験において新たな知見を与える論文.
  - 事例報告：産業界での地道な研究・改善活動を発表するもの (論文執筆の必要はありません. 発表スライド原稿で審査します).
- チュートリアル
- パネルディスカッション
- ツールデモ

募集する企画のテーマは, ソフトウェア関連技術全般です. 以下に例示しますが, これに限りません.

・要求工学/ 企画計画技法	・開発支援ツール/ 環境(CASE)	・ビジネスプロセス/ モデリング	・品質管理
・ドメイン分析/ モデリング	・協調作業支援(CSCW)	・構成管理/ プロジェクト管理	・分析/ 設計技法
・プログラミング技法	・ソフトウェアプロセス	・プログラム解析/ テスト	・メトリクス
・保守/ リエンジニアリング	・ネットワーク, セキュリティ	・再利用技術/ コンポーネント	・人的要因/ 教育
・ソフトウェアアーキテクチャ	・GUI/ マルチメディア	・オブジェクト指向技術	・標準化

主要スケジュール：	**2000 年 1 月 31 日**	**論文/ 報告および企画応募締切**
	2000 年 3 月 31 日	**論文/ 報告および企画採否通知**
	2000 年 4 月 30 日	**論文/ 報告最終稿締切**

【シンポジウムスタッフ】

実行委員長：	落水 浩一郎 (JAIST)　新森 昭宏 (インテック・システム研究所)
プログラム委員長：	新田 稔 (新日鉄情報通信システム)　野呂 昌満 (南山大学)
プログラム委員：	人選中

【応募要領】

応募は電子投稿によるものとします. PS または PDF 形式で下記アドレスまでお送り下さい. 所定の要領で作成し, カバーシートを添付して下さい. チュートリアル, パネルディスカッション, ツールデモについては, 書式は問いませんが分量は良識の範囲内でお願いします(詳細は裏面を御覧ください) .

応募論文/ 報告送付先:	ss2000@iq.nanzan-u.ac.jp
応募に関する問い合わせ先:	新田 稔 (nitta@rd.enicom.co.jp)
	野呂 昌満 (masami@iq.nanzan-u.ac.jp)

【SS 2000 応募要領の詳細】

<<論文 / 報告>>: 応募論文/ 報告は未発表のものに限ります．他への重複投稿もご遠慮ください．

研究論文・経験論文作成要領

冒頭部に必ず要旨を記述して下さい．その他の書式は自由です．用紙は A4 サイズに限ります．分量は 5 ～ 10 ページ程度が目安です．できるだけ日本語で執筆して下さい（英文論文も受け付けますが，発表は日本語に限ります）

事例報告作成要領

発表スライド原稿を A4 紙に白黒印刷して下さい．発表時間は 15 分を予定していますので，スライド 10 枚程度が目安です．発表要旨（A4 用紙で 1 ページ以内）を添付してください．スライド原稿にも要点などを注記していただけば，査読者の理解が深まると思われます．

<<チュートリアル>>: チュートリアルの内容を記載したもの．

<<パネルディスカッション>>: 議論の主題，参加者，内容等を記載したもの．

<<ツールデモ>>: デモを行なうソフトウェアの内容，動作環境等を記載したもの．

応募論文/ 報告送付先: **ss2000@iq.nanzan-u.ac.jp**
応募に関する問い合わせ先: **新田 稔 (nitta@rd.enicom.co.jp) または 野呂 昌満 (masami@iq.nanzan-u.ac.jp)**

ソフトウェア・シンポジウム 2000 論文 / 報告 / 企画 応募カバーシート

氏名(筆頭著者，提案者): (ふりがな)
所属 (会社 / 大学 / 組織):
部門 (学部) :
役職:
住所: 〒( )
TEL (内線):
FAX:
E-mail:

応募項目(いずれかにチェックしてください)

☐ 論文 / 報告 (さらにいずれかをチェック) ☐ 研究論文 ☐ 経験論文 ☐ 事例報告
☐ チュートリアル
☐ パネルディスカッション
☐ ツールデモ

論文 / 報告 / 企画のタイトル:

共同執筆者 (提案者) : (もしいれば)

氏名: 所属:
氏名: 所属:
氏名: 所属:

キーワード (論文/ 報告のみ): 2 ～ 5 つのキーワードを挙げて下さい)

1:
2:
3:
4:
5:

ICS 2000
International Conference on Software :Theory and Practice

In cooperation with: IFIP TC2, United Nations University/IIST, Chinese Academy of Sciences/Institute of Software

The conference will provide a forum for professionals from academia, business and industry to share information, evaluate results and explore new ideas on all aspects of software development process including specification, design, implementation and validation of software systems. As the design of today's large and complex software requires intimate interactions between theory and practice, the conference especially welcomes papers on applying solid theoretical foundations of software in practice, and on innovative software practices that call for new theoretical development.

Topics of interest for paper submission

The conference will focus on a series of issues ranging from theory to the practice of software. The topics of the conference will include, but not limited to, the following:

General Topics in Software Engineering:
software architecture
software requirements engineering
software process models and methodology
software components and reuse
validation and verification
software quality and evaluation
formal methods and specifications
programming languages
domain analysis and design
tools and environments

Topics in cross-cutting and specific software domains:
network computing
internet/intranet applications
real-time, embedded systems
hybrid systems
concurrent and distributed systems
human-computer interfaces
system integration and frameworks
database management and data mining
multimedia systems and applications

Chair Persons

Conference Co-chairs:
Prof. Reino KURKI-SUONIO (Finland)
Prof. XU Jafu (China)

Program Committee Co-chairs:
Prof. FENG Yulin(China)
Prof. Marie-Claude GAUDEL (France)
Prof. David NOTKIN (USA)

Submission Guidelines

Paper Submission: Papers submitted should be in English. The congress will produce both CD-ROM proceedings and hardcopy ones by official IFIP publisher, Kluwer Academic Publisher. The papers in the proceeding are categorized in two types. Regular paper: up to 8 pages, $100 for each additional page, but for no more than 12 pages maximum. Short paper: up to 4 pages, more pages are not acceptable. To make sure paper review process as smooth as possible, authors are requested to submit their paper's title page (including author name, address, paper's title and abstract) electronically to the ICS contact person in Beijing (email: ics@wcc2000.org), and to submit their full papers in both electronic form and 3 copies of hardcopies to one of the Program Co-chairs listed below for review by deadline January 10, 2000. Fax submissions of the full paper are not acceptable. The electronic form should be in a required format and uploaded to an ftp site (further instruction will be provided through our website http://www.wcc2000.org).

Prof. Yulin FENG
Institute of Software
Chinese Academy of Sciences
PO. Box 8718
Beijing 100080, China
E-mail: feng@ox.ios.ac.cn

Dr. Marie-Claude GAUDEL
Laboratoire de Recherche en
Informatique, Bâtiment 490
Université Paris Sud
91405-ORSAY Cedex , France
E-mail: mcg@lri.fr

Prof. David NOTKIN
Department of Computer Science & Engineering
University of Washington
Box 352350 Seattle,
WA 98195-2350 USA
E-mail:notkin@cs.washington.edu

Call for Workshop, Tutorial Proposals: The conference Program Committee invites workshops and tutorial proposals for the ICS2000, to be held in Beijing, China. Tutorials will be held one day before conference, and be offered both on standard topics and on new and more advanced topics. Workshops will be held in parallel with the conference. Proposals should be received by one of the program chairs no later than January 10, 2000. Right now the workshop on the theme of "Software Industry in Developing Countries" has been set down in the conference, which is proposed by Prof. Chaochen ZHOU from UNU/IIST, Macau.

Schedule

- January 10, 2000: Full papers and proposals for workshops and tutorials due at one of the Program Co-chairs.
- March 10, 2000: Notification of acceptance.
- April 15, 2000: Final version of accepted papers due at Beijing

ソフトウェア技術者協会
〒160-0004　東京都 新宿区 四谷 3 - 12 丸正ビル 5F
Tel: 03 - 3356 - 1077　Fax: 03 - 3356 - 1072
E-mail: sea@sea.or.jp
URL: http://www.iijnet.or.jp/sea