社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－SE－150（11）
2005／11／29

オブジェクトベースFortranの設計とそのトランスレータの開発

齋藤正樹[†]　畠山正行[††]

要約: 一貫記述言語系 OOJ の一環として, 実装段階の記述言語であるオブジェクトベース Fortran (OBF) の設計と, 設計段階の記述言語である ODDJ の記述から OBF の記述に変換する ODDJ/OBF トランスレータの開発を行った. OBF は Fortran 90 を基にオブジェクトベースの概念に準拠するための記述ルール, 制約を加えた言語であり, OBF 記述は既存の Fortran 90 コンパイラによって処理することが可能である. 本トランスレータにより ODDJ 記述の一部の自動変換が既に実現している. 現時点では ODDJ 記述からの変換に問題は生じておらず, 引き続いてトランスレータの開発を続けている. 今後は変換の実証例を増やすとともに, OOJ に自動分散化システム ADS を組み込み, ドメインユーザにとって有用な開発環境の構築を目指す.

A Design of Object-based Fortran and Development of its Translator

Masaki Saito [†] and Masayuki Hatakeyama[††]

Abstract: We have designed a description language Object-based Fortran (OBF) at the programming stage in the integrated language system OOJ, and developed the ODDJ/OBF translator to transform the design descriptions of the ODDJ into the OBF descriptions. The description rules of the OBF pursuant to the Object-based paradigm with some constraint rules have been designed based on the Fortran 90. The OBF descriptions can be compiled using the ordinary Fortran 90 compiler. Using this translator, the automatic transformations of the part of the ODDJ descriptions have already been realized. At present, no problems have been found during the transformation processes from the ODDJ into OBF. The future works are to make more description examples, to built in the automatic distribution system ADS in the OOJ, and finally to aim at establishing the valuable development environments for the Domain Users.

1. はじめに

我々の研究グループでは非情報系の科学技術分野の専門家であるドメインユーザ (Domain User, 以降 **DU** と略) を対象としたシミュレーション開発支援を目的として, 日本語を用いたオブジェクト指向によるモデリングとその記述についての研究が行われている. これはシミュレーション開発を分析, 設計, 実装の 3 段階に分けて扱い, それぞれの段階に対応する記述言語—一貫記述言語系 OOJ [1)]—とそれを支援する開発環境によって構成されている.

これまで分析段階の記述言語としてオブジェクト指向自然日本語記述言語 OONJ [2),3)], 設計段階の記述言語としてオブジェクト指向設計記述言語 ODDJ [4)] が提案されている. 本稿では残る実装段階の記述言語としてオブジェクトベース Fortran (Object-based Fortran, 以降 **OBF** と略) の設計について, また実装段階の開発支援環境として ODDJ 記述から OBF 記述に変換する ODDJ/OBF トランスレータの開発について述べる.

実装段階に OBF と ODDJ/OBF トランスレータを用意することで, DU が既存のシミュレーション開発手法から OOJ による開発手法にスムーズに移行し, 最小限の負担でシミュレーションを開発することのできる環境を提供することを目指す.

† 茨城大学大学院理工学研究科
Graduate School of Science and Engineering, Ibaraki University
†† 茨城大学工学部情報工学科
Department of Computer and Information Sciences, Faculty of Engineering, Ibaraki University

2. OBF の設計方針

2.1 OBF の要件

OOJ では DU の想定として 8 つの特徴を挙げている[1)] が, この中で実装段階に関わるものとして以下の点に着目した.

- 中小規模（数千〜数万行）の試行錯誤や改訂の多い自家用プログラムを必要に迫られ個人か少人数で一貫自主開発. そのアルゴリズムは複雑なことが多く, 別言語での書き換え等は嫌がる

- 常用 PL☆（多くは Fortran）と常用自然言語（母国語）は十分に駆使し, 知識は前提に出来る. 未知の新規 PL （特に異型の OOPL）は避ける

これらの特徴から, DU は常用のプログラミング言語を持っており, それを用いてシミュレーションを開発した経験があること, また未知のプログラミング言語を新たに習得することはもちろん, 一度記述したプログラムを別言語へ書き換えることを避ける傾向があるということが分かる. このことから, OOJ の実装段階が DU に対してもっとも負荷の少なく, 受け入れやすいものとなるよう, 以下のような目標を掲げた.

- DU が常用しているプログラミング言語, あるいはそれに近い言語による記述が得られること
- 設計段階で作成した ODDJ 記述から, DU の手を借りることなく上記言語による記述に変換が可能であること

この 2 つの目標を満たすための言語には何が必要であるかを検討した.

2.2 DU の常用言語との互換性

まずシミュレーションのためのプログラミング言語として DU がどういった言語を求めているのかということが重要である. DU が常用している計算機システムで扱うことができない, あるいはパフォーマンスを発揮できないような言語であっては DU に対しての魅力に乏しい. ひいては OOJ の DU への普及を阻害する要因と成りかねない. とはいっても, 最初からすべての DU の要求を満たすことは困難である. そこで当面は DU として数値流体力学 (CFD) 分野においてシミュレーションを行うユーザを想定することとした.

CFD のシミュレーションでは C++[5] や Java[6] といったオブジェクト指向言語も使用されているが, 未だに C や Fortran といった手続き型言語も広く使用されている. 特に Fortran は長期間の使用実績があり, 過去に作成した膨大なライブラリ資産が蓄積されている. 近年では, JAXA が CFD の共通基盤プログラムとして開発を行っている UPACS[7] も対象言語としては Fortran を採用しており, 現在においても主流言語の一つであると考えられる. こうした点から実装段階の記述言語としては Fortran をベースとして用いることとした.

2.3 Fortran とオブジェクト指向

次に ODDJ 記述から Fortran の記述に変換するために何が必要であるかを検討した. OOJ の分析, 設計はオブジェクト指向に基づいており, そこで作成した記述をそのまま手続き型言語である Fortran に対応づけることは困難である. オブジェクト指向特有の概念であるオブジェクトと, オブジェクトの持つメソッドと属性— OONJ, ODDJ ではフレームとスロットに相当—を扱うためには Fortran に拡張を加えることが必要である.

Fortran にオブジェクト指向の概念を導入することについては既にいくつかの研究がなされている[8),9)]. これらの研究は確かにオブジェクト指向におけるカプセル化や継承, 多態性といった要素を導入することに成功しているが, 元となった Fortran のプログラミングスタイルとは大きく異なっているため, DU に可読性の低下, 記述の煩雑化といった弊害が生じることが予想される. 本研究では Fortran でオブジェクト指向の概念を扱うことを目的とするのではなく, あくまで OOJ の一翼を担い, DU にとって扱いやすい言語であることに主眼を置き, Fortran への拡張を OOJ で求める必要最小限のものとすることにした.

具体的には次節以降で説明するが, OOJ の持つ特徴的な概念であるオブジェクトベース（Object-based, 以降 **OB** と略）[1] に基づいているため, オブジェクトの概念とカプセル化, オブジェクト間のメッセージパッシングのみを実現しており, 継承や多態性といった要素は導入していない. また, オブジェクトの生成や消滅といったことを DU が考慮する必要はない. Fortran にこうした拡張を施すためには FORTAN 77 の仕様では不足しており, Fortran 90 をベースとすることにした. 前述の UPACS も Fortran 90 を対象としており, DU の常用言語という観点からも問題ないものと考えられる.

このようにして, 2 つの要件を満たすため Fortran 90 に OB に基づく拡張を施した言語 OBF を設計した.

2.4 OOJ の実装段階における記述言語

OOJ の実装段階における記述言語としては, 本稿で提案する OBF とともに, オブジェクト指向実装記述言語 OEDJ[10] がある. この 2 つの言語は設計段階の ODDJ 記述からプログラミング言語記述へ変換する方式が大きく異なっている. OBF を用いた方式では ODDJ 記述から後述する ODDJ/OBF トランスレータによって OBF 記述を生成する.

一方, OEDJ を用いた方式では ODDJ 記述からトランスレータによって OEDJ 記述を生成し, 生成した OEDJ 記述の追記, 修正を OEDJ エディタを用いて行い, そこから更に別のトランスレータによって既存のプログラミング言語（現在は Java を対象）の記述を生成する. OEDJ は特定のプログラミング言語に依存しない実装記述を行うための言語として位置づけられている.

OBF を用いた方式の長所としては, 新たな記述言語を習得する必要がなく, ODDJ 記述からの書き換え作業を手動で行う必要がないため実装段階での DU の負担が少ないという点が挙げられる. 一方短所としては, DU の手助けなしで自動変換するという性格

☆ Programming Language

上, ODDJ 記述に高い完成度が要求され, 設計段階での DU の負担が増加するという点が予測される.

3. OBF の設計

3.1 オブジェクトベースの実現

手続き型言語である Fortran 90 を OB の言語として扱うためには, OB の持つ概念を Fortran 90 の持つ機能に写像する必要がある. Rumbaugh は非オブジェクト指向プログラミング言語でオブジェクト指向を実装するためには, 以下の手順が必要であるとしている [11].

(1) クラスをデータ構造に翻訳する
(2) 引数をメソッドに渡す
(3) オブジェクトに記憶域を割り当てる
(4) 継承をデータ構造を使って実装する
(5) メソッドの決定機構を実装する
(6) 関連を実装する
(7) 並行性を処理する
(8) クラスの内部詳細をカプセル化する

これはクラスベースのオブジェクト指向を実装しようとするものであり, OB を実装するために上記手順を再検討した. OB は前述の通り, オブジェクトとカプセル化, オブジェクト間のメッセージパッシングという条件を満たすものであり, クラスベース特有のクラスやインスタンス, 継承, 多態性といった概念は存在しないため, それらに関する項目は考慮しない. 以下に上記手順の各項目に対して検討した結果を示す.

(1) クラスやインスタンスといった概念は存在しないため, オブジェクトの持つ属性をデータ構造に翻訳する（**属性のデータ構造への翻訳**）
(2) オブジェクトの持つ振る舞いをメソッドとして実装する（**メソッドの実装**）
(3) オブジェクトが対象世界に存在するための記憶域を割り当てる（**オブジェクトの記憶域の割り当て**）
(4) 継承の概念は存在しないため考慮しない
(5) 多態性の概念は存在しないため考慮しない
(6) オブジェクト間のメッセージパッシングを実現するため関連を実装する（**関連の実装**）
(7) 並行性はサポートしないため考慮しない
(8) オブジェクトの属性とメソッドをカプセル化する（**カプセル化の実装**）

OBF の設計をこの手順の項目 (1), (2), (3), (6), (8) に従って行った. 以降それぞれの項目ごとに説明する.

3.2 属性のデータ構造への翻訳

Fortran 90 では **type 構文**を用いて複数の型のデータから複合データを構造型として定義することが可能である. そこで, 構造型の各構成データをオブジェクトの持つ属性に対応づけることにした. 構造型名にはオブジェクト名に “_type” というサフィックスを追加する. **図 1** にオブジェクトの属性定義構文を示す.

```
! 属性
type オブジェクト名_type
  属性型 :: 属性名
  属性型 :: 属性名
  .....
end type オブジェクト名_type
```

図 1 構造型による属性定義
Fig. 1 Attribute definition by structure type

3.3 メソッドの実装

メソッドは Fortran 90 のサブルーチンまたは関数を用いて定義する. メソッド名については “オブジェクト名_” というプレフィックスを付加することで, プログラム全体を通して一意な名称となるようにした.

メソッドに渡す引数は, Fortran 90 の通常のサブルーチンや関数の引数の扱いと同様にすべて参照渡しとする. また, 多くのオブジェクト指向プログラミング言語では, メソッドは対象オブジェクトを指定する self や this といった暗黙的な引数が存在していることが多く, 非オブジェクト指向言語でオブジェクト指向を実現しようとするとこの種の引数を明示的に持つことが必要になるが, OBF ではこの仕組みを用いることはない. メソッドは対象オブジェクトを指定する引数を持たずとも, 暗黙的にオブジェクトと関連づけられる. **図 2** にオブジェクトのメソッド定義構文を示す.

```
! サブルーチンによるメソッド定義
subroutine オブジェクト名_メソッド名 (仮引数リスト)
  .....
end subroutine オブジェクト名_メソッド名

! 関数によるメソッド定義
型 function オブジェクト名_メソッド名 (仮引数リスト)
  .....
end function オブジェクト名_メソッド名
```

図 2 サブルーチンと関数によるメソッド定義
Fig. 2 Method definition by subroutine and function

3.4 オブジェクトの記憶域の割り当て

OBF ではオブジェクトへの記憶域の割り当てを, 自身を表す特別な変数を持つことで静的に行うことにした. これは OB に基づいて, DU がオブジェクトの生成や消滅といったことに対して注意を払うことなく, オブジェクトが対象世界に最初から存在するものとして扱えるようにするためである. 自身を表す変数名にはオブジェクト名を指定する. メソッドから自身の持つ属性の値を取得, 設定する際にはこの変数を経由し

て行うことになる．**図 3** にオブジェクト自身を表す変数定義構文を示す．

```
! 自身を表す変数
type（オブジェクト名_type）:: オブジェクト名
```

図 3 オブジェクト自身を表す変数定義
Fig. 3 Variable definition for object own self

3.5 関連の実装

後述するカプセル化とともに説明する．

3.6 カプセル化の実装

Fortran 90 では **module 構文**を用いて変数宣言とセットにしてサブルーチンや関数を定義することが可能である．そこで, オブジェクトの属性に対応する構造型, メソッドに対応するサブルーチンと関数をまとめて 1 つのモジュールとして定義することでカプセル化を実現する．モジュール名にはオブジェクト名に “_object” というサフィックスを追加する．更にモジュール内で前述のオブジェクト自身を表す変数を宣言することで, モジュールを OB の概念の基でのオブジェクトとして扱うことにする．このようにして定義した OBF におけるオブジェクトを**オブジェクトモジュール**と呼称する．

オブジェクト間の関連はこのオブジェクトモジュール間の関連に相当し, Fortran 90 ではモジュールから別のモジュールへの参照は **use 宣言文**によって行うことが可能である．そこでオブジェクトモジュール内で関連するオブジェクトモジュールへの use 宣言を行うことによってオブジェクト間の関連を実現する．**図 4** にオブジェクトモジュールの定義構文を示す．

3.7 OBF と Fortran 90

OBF は Fortran 90 に OB に基づく拡張を加えた言語であり, それは図 4 に示したオブジェクトモジュール定義に集約されている．オブジェクトモジュールにはオブジェクトとカプセル化, 関連が実装されており, オブジェクトモジュール間の相互作用としてプログラムを実装することが可能である．別の見方をすれば, オブジェクトモジュールの存在を除けば, 既存の Fortran 90 と同一の言語と見なすことができる．OBF と Fortran 90 の関係を**図 5** に示す．

4. ODDJ から OBF への変換

ODDJ は OOSF[12] に基づく構造化規則を採用しており, 記述は複数のフレームとそのフレームを構成する複数のスロットによって成り立っている．フレームには以下のような種類があり, これらの ODDJ 要素

```
! オブジェクトモジュール
module オブジェクト名_object
  ! 関連
  use 関連するオブジェクトモジュール名

  ! 属性
  type オブジェクト名_type
    .....
  end type オブジェクト名_type

  ! 自身を表す変数
  type（オブジェクト名_type）:: オブジェクト名

  ! メソッド
  contains
  subroutine オブジェクト名_メソッド名（仮引数リスト）
    .....
  end subroutine オブジェクト名_メソッド名

  型 function オブジェクト名_メソッド名（仮引数リスト）
    .....
  end function オブジェクト名_メソッド名
end module オブジェクト名_object
```

図 4 モジュールによるオブジェクト定義
Fig. 4 Object definition by module

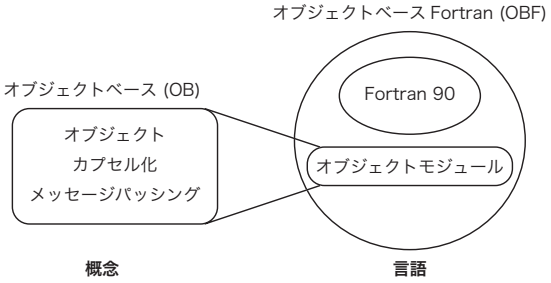

図 5 OBF と Fortran 90 の関係
Fig. 5 Relation between OBF and Fortran 90

が OBF にどのように対応づけられるかを述べる．

- オブジェクトフレーム
- 定数定義フレーム
- 初期設定フレーム
  - 初期状況フレーム
  - 境界条件フレーム
- 駆動制御記述フレーム
  - main フレーム
  - 駆動シナリオフレーム
- ライブラリフレーム

4.1 オブジェクトフレーム

オブジェクトフレームは対象世界に存在するオブジェクトについて記述したフレームである．**図 6** にオブジェクトフレームを構成するスロットと, スロット

に含まれる主な要素を示す.

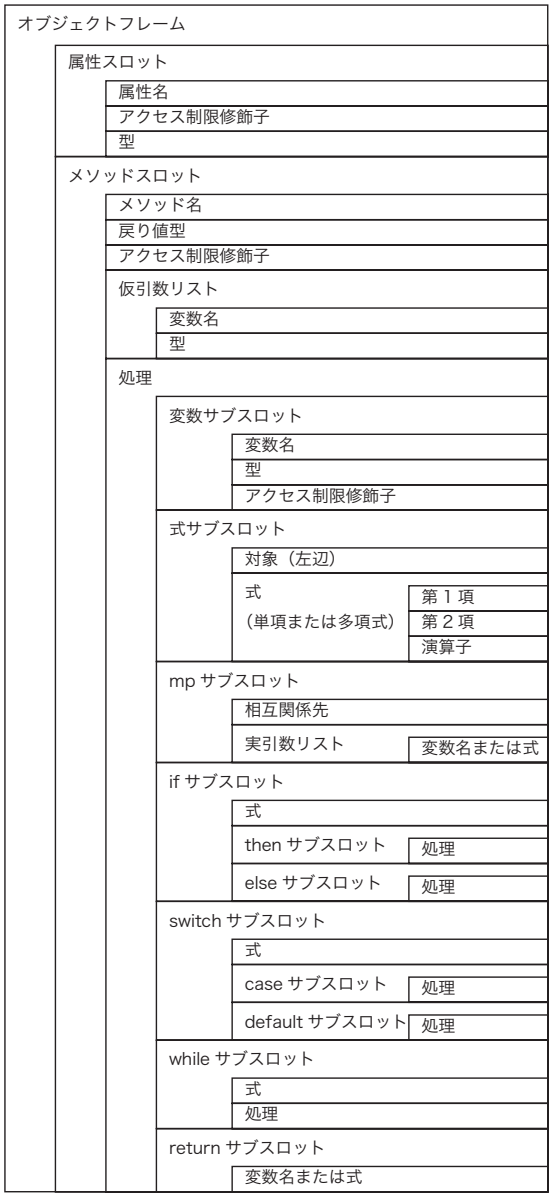

図 6 オブジェクトフレームの構成
Fig. 6 Constitution of ODDJ object frame

オブジェクトフレームは OBF のオブジェクトモジュールに 1 対 1 で変換される. オブジェクトフレームの属性スロット, メソッドスロットと OBF の構文との対応を**表 1**, **表 2** に示す. また, 両スロット内の属性や変数, メソッドの戻り値型と OBF の型との対応を**表 3** に示す.

4.2 定数定義フレーム

定数定義フレームは DU が定義した定数や単位を記述するためのフレームである. フレームを構成するスロットは対象世界全体で共通して用いられる値や式に相当する.

OBF では定数定義フレームをプログラムの共有変数として扱う. 共有変数はモジュールを用いて定義し, これを**定数定義モジュール**と呼称する. プログラムを構成する全てのモジュールは定数定義モジュールを use 宣言する. **図 7** に定数定義モジュールを示す.

表 1 属性スロットと OBF 構文の対応
Table 1 Correspondence of attribute slot to OBF syntax

ODDJ	OBF
属性名	構造型の成分名
アクセス制限修飾子	指定無し
型	構造型の成分の型

表 2 メソッドスロットと OBF 構文の対応
Table 2 Correspondence of method slot to OBF syntax

ODDJ	OBF
メソッド名	サブルーチン名または関数名
戻り値型	関数名の型
アクセス制限修飾子	指定無し
仮引数リスト	サブルーチンまたは関数の仮引数リスト
変数サブスロット	変数宣言
式サブスロット	代入文
mp サブスロット	サブルーチンまたは関数の呼び出し
if サブスロット	if 構文
switch サブスロット	select case 構文
while サブスロット	while 構文
return サブスロット	関数名の代入文

表 3 ODDJ と OBF の型の対応
Table 3 Correspondence of ODDJ types to OBF types

ODDJ	OBF
整数型	integer
実数型	real
文字型	character
文字列型	character[len=文字列長]
論理型	logical
void	指定無し

```
! 定数定義モジュール
module obf_constant_numbers
  型, parameter :: 定数名 = 定数値または定数式
  .....
end module obf_constant_numbers
```

図 7 定数定義モジュール
Fig. 7 Constant number definition module

4.3 初期設定フレーム

初期設定フレームは初期状況フレームと境界条件フレームから構成される. 初期状況フレームはオブジェクトフレームの初期状況を, 境界条件フレームはオブ

ジェクトフレームの境界条件をそれぞれ設定するためのフレームである．この 2 つのフレームの記述によりオブジェクトフレームの属性の初期化が行われる．

OBF では初期状況フレームと境界条件フレームを役割をそのままにモジュールとして定義し，それぞれ**初期状況モジュール**，**境界条件モジュール**と呼称する．**図 8** に初期状況モジュールと境界条件モジュールを示す．

```
! 初期状況モジュール
module obf_initial_condition
  ! オブジェクトモジュールとの関連
  use オブジェクトモジュール名
  ! 定数定義モジュールとの関連
  use obf__constant_numbers

  .....
end module obf_initial_condition

! 境界条件モジュール
module obf_boundary_condition
  ! オブジェクトモジュールとの関連
  use オブジェクトモジュール名
  ! 定数定義モジュールとの関連
  use obf__constant_numbers

  .....
end module obf_boundary_condition
```

図 8 初期状況モジュールと境界条件モジュール
Fig. 8 Initial condition module and boundary condition module

4.4 駆動制御記述フレーム

駆動制御フレームは main フレームと駆動シナリオフレームから構成される．main フレームは駆動シナリオフレームをどのように起動するか記述するためのフレームで，駆動シナリオフレームは対象世界の駆動を制御するためのフレームである．この 2 つのフレームの記述によりシミュレーションが駆動される．

OBF では main フレームと駆動シナリオフレームを役割をそのままにそれぞれ主プログラム，モジュールとして定義し，それぞれ**main プログラム**，**駆動シナリオモジュール**と呼称する．**図 9** に main プログラムと駆動シナリオモジュールを示す．

図 10 にシミュレーションの駆動の流れを示す．main プログラムが駆動シナリオモジュールの開始メソッドを起動した後，駆動シナリオモジュールが初期状況モジュールと境界条件モジュールを用いてオブジェクトモジュールの属性を初期化し，オブジェクトモジュールのメソッドを起動することでシミュレーションを駆動する．

```
! main プログラム
program obf_main
  ! 駆動シナリオモジュールとの関連
  use obf_drive_scenario

  ! 開始メソッドの起動
  call obf_drive_scenario_開始メソッド
end

! 駆動シナリオモジュール
module obf_drive_scenario
  ! 初期設定モジュールとの関連
  use obf_initial_condition
  use obf_boundary_condition
  ! 定数定義モジュールとの関連
  use obf_constant_numbers

  ! 開始メソッド
  subroutine obf_drive_scenario_開始メソッド
    .....
  end subroutine obf_drive_scenario_開始メソッド

  .....
end module obf_drive_scenario
```

図 9 main プログラムと駆動シナリオモジュール
Fig. 9 Main program and drive scenario module

4.5 ライブラリフレーム

ライブラリフレームはファイル入出力や数学関数といった汎用的な処理を記述するためのフレームであり，シミュレーションの開発には不可欠なものである．ライブラリフレームは設計段階での仕様が不明確であるため，変換についても厳密には考慮されていない．

4.6 ODDJ 記述と OBF 記述の対応

これまで述べた ODDJ 記述を構成する各フレームと OBF との対応を**表 4** に示す．ODDJ 記述から OBF 記述への変換に際して新たに追加する情報はないが，ODDJ 記述の持つ情報すべてを利用しているわけではない．例えば，オブジェクトフレーム内の属性スロット，メソッドスロットが持つアクセス制限修飾子は OBF 記述に対応づけられていない．

表 4 ODDJ のフレーム要素と OBF の対応
Table 4 Correspondence of ODDJ frame to OBF program

ODDJ	OBF
オブジェクトフレーム	オブジェクトモジュール
定数定義フレーム	定数定義モジュール
初期状況フレーム	初期状況モジュール
境界条件フレーム	境界条件モジュール
main フレーム	main プログラム
駆動シナリオフレーム	駆動シナリオモジュール
ライブラリフレーム	未定

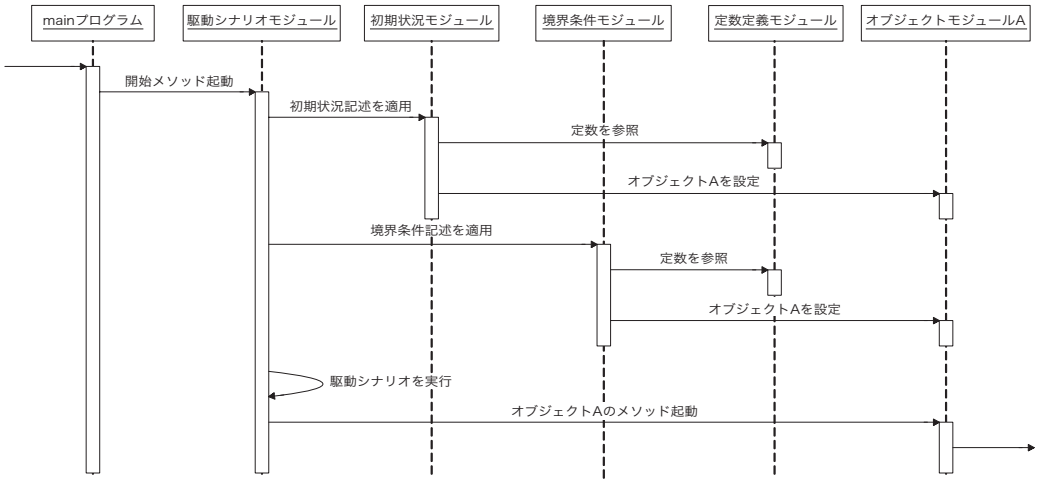

図 10 シミュレーション駆動の流れ
Fig. 10 Simulation drive sequence

5. ODDJ/OBF トランスレータ

5.1 トランスレータの機能仕様

4 節で述べた ODDJ と OBF の変換規則に基づいて ODDJ 記述から OBF 記述に変換する ODDJ/OBF トランスレータの開発を行った. ODDJ/OBF トランスレータは CUI のプログラムで, 設計段階で作成された ODDJ の XML 文書を読み込み, それを変換規則に従って OBF 記述に変換し, OBF 記述をファイル出力するというものである. トランスレータの変換処理に DU が関わることはなく, 唯一トランスレータ起動時に ODDJ 記述をファイル指定するのみである. 指定された記述が ODDJ の XML スキーマ [4] に適合していない場合, エラーを出力し変換処理を中断する. **図 11** に ODDJ/OBF トランスレータの実行コマンドを示す.

```
usage: oddjt [options] input-file
options:
    -h, –help :  ヘルプメッセージを表示する.
    -v, –version :  ODDJ/OBF トランスレータのバージョン情報を表示する.
    -o, –output-path :  変換した OBF 記述の出力先を指定する. 指定しない場合は読み込んだ ODDJ 記述と同じ場所に出力する.
    -p, –programming-language :  ODDJ 記述から変換するプログラミング言語を指定する. 現在のところ OBF のみ指定可能.
    -f, –frame :  変換するフレーム番号を指定する. 指定しない場合は ODDJ 記述全体を変換する.
```

図 11 ODDJ/OBF トランスレータの実行コマンド
Fig. 11 Execution command of ODDJ/OBF translator

5.2 トランスレータの内部仕様

図 12 に ODDJ/OBF トランスレータのクラス図を示す. クラスの設計は, ODDJ 記述のデータ構造の取り扱いと OBF 記述への変換というトランスレータの持つ 2 つの機能に着目して行った.

ODDJ 記述のデータ構造を取り扱うため, ODDJ 記述のデータフォーマットである XML モデルにデータバインドするクラスを用意した. 具体的にはオブジェクトフレームと対応する COddjObjectFrame クラス（図左上）, 属性スロットと対応する COddjAttributeSlot クラス (図中央上) , メソッドスロットと対応する COddjMethodSlot クラス（図左中央）といったものである. この他, 両スロットが持つサブスロットについても同様のクラスが存在している. XML モデルの木構造はこれらのクラスの集約構造に対応づけている.

記述の変換処理は CObfTranslateVisitor クラス (図中央上）に実装した. Visitor パターンを用いてデータ構造と変換処理を分離しているのは, 今後トランスレータに OBF 以外の言語への変換処理を実装する際にデータ構造を扱うクラスを修正することなく機能を拡張できるようにするためである.

5.3 トランスレータの開発状況

ODDJ/OBF トランスレータはいまだ開発途上であり, 表 4 で示した ODDJ 記述と OBF 記述の対応のうちオブジェクトフレームからオブジェクトモジュールへの変換機能は実装されているが, その他のフレームの変換機能は現在実装中である.

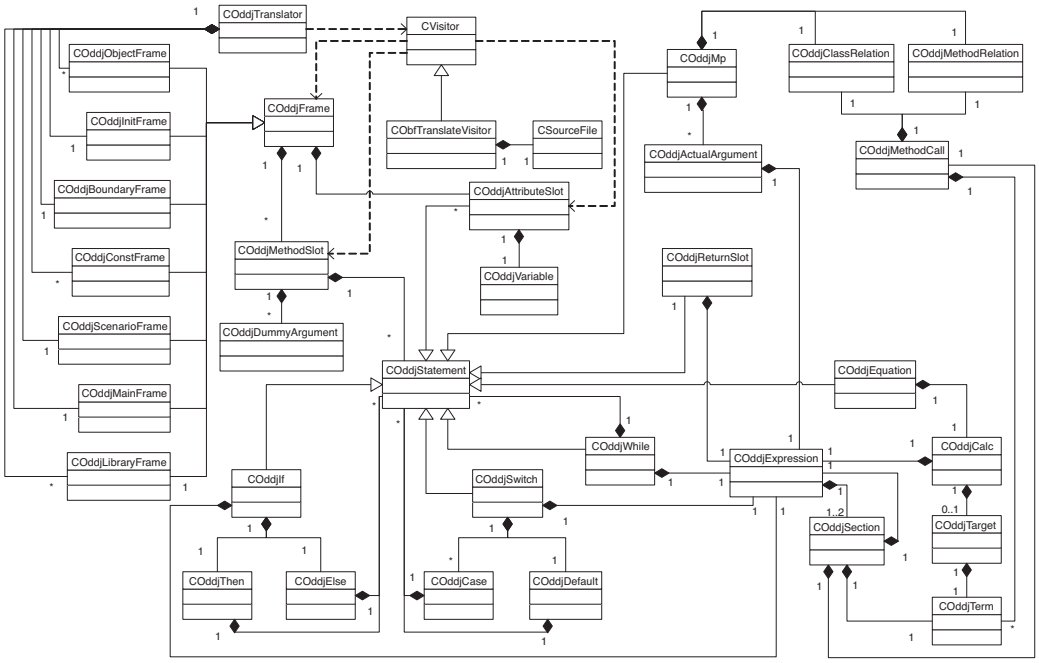

図 12 ODDJ/OBF トランスレータのクラス図
Fig. 12 Class diagram of ODDJ/OBF translator

6. 記 述 例

OBF を用いて記述したシミュレーションプログラムの記述例を示す. 対象は OOJ の一貫記述例として作成した “水の大気循環” のシミュレーションである. **図 13** にその一部である “海” オブジェクトモジュールについての記述を示す.

このプログラムは ODDJ の記述例[4] にある “海” オブジェクトフレームを手動で変換したものである. ODDJ と OBF の変換規則に従って, オブジェクトフレームは “sea_object” という名称のオブジェクトモジュールとして定義されている. モジュール内を見ていくと, 6 行目から 9 行目にかけて “海上空大気（高度 0）” と “海上空雲（高度 1）” の 2 つのオブジェクトモジュールと定数定義モジュールとの関連が use 宣言されている.

次に 12 行目から 21 行目にかけて構造型 “sea_type” によって属性が定義されている. 構造型の各成分は “海” フレームの 1〜7 番の属性スロットと対応するものである. 属性スロットの型は表 3 の対応に基づいて Fortran 90 の型に変換されている.

24 行目でオブジェクト自身を表す変数 “sea” を宣言している. 後述するメソッド定義の中でこの変数を経由してオブジェクトの属性を扱っていることが分かる.

29 行目から 116 行目にかけてサブルーチンによってメソッドが定義されている. これらは “海” フレームの 8〜20 番のメソッドスロットと対応するものである. サブルーチン内の記述は Fortran 90 の記述そのものであり, Fortran 90 を常用言語とする DU であれば容易に理解できるものとなっている. 属性と同様, メソッドスロットの仮引数の型や内部変数の型は表 3 の対応に基づいて Fortran 90 の型に変換されている.

また, 10 番スロット “蒸発する” と 11 番スロット “熱放射する” には “海上空大気（高度 0）” と “海上空雲（高度 1）” オブジェクトフレームへのメッセージパッシング（mp サブスロット）があり, これは関連で宣言したオブジェクトモジュールのサブルーチンまたは関数の呼び出しに変換されている. 11 番スロット内で放射熱量の計算に用いている変数 ϵ と σ は定数定義フレームで定義されたものを参照しており, これは定数定義モジュールで定義した共有変数である.

7. 現時点での評価

ODDJ/OBF トランスレータは開発途上にあり, 現在の OBF の仕様が OOJ の実装段階として十分なものであるか, また DU にとってどれだけの利点があるかといったことについては評価を下すことはできない. そこで, 6 節で示した “水の大気循環” シミュレーションの OBF 記述例と ODDJ 記述例[4], OEDJ 記述例[10] を比較した結果から分かることについて述べる.

```
! 海オブジェクトモジュール
module sea_object
  implicit none

  ! オブジェクトモジュールとの関連
  use air_sea0_object ! 海上空大気（高度 0）
  use cloud_sea_1_object ! 海上空雲（高度 1）
  ! 定数定義モジュールとの関連
  use obf_constant_numbers

  ! 属性
  type sea_type
    private
    real :: S   ! 面積 S
    real :: C   ! 比熱 C
    real :: Ts  ! 表面温度 Ts
    real :: Ms  ! 表面質量 Ms
    real :: Qr  ! 熱放射量 Qr
    real :: Qa  ! 吸収熱量 Qa
    real :: Q   ! 保有熱量 Q
  end type sea_type

  ! 自身を表す変数
  type (sea_type) :: sea

  ! メソッド
  contains
    ! 熱吸収する
    subroutine sea_absorb_heat(Qa)
      real :: Qa

      sea%Q = sea%Q + Qa * sea%S
      call sea_rise_in_temp
    end subroutine sea_absorb_heat

    ! 温度が上昇する
    subroutine sea_rise_in_temp
      sea%Ts = sea%Ts + sea%Q / (sea%Ms
        * sea%C)
      call sea_evaporate
    end subroutine sea_rise_in_temp

    ! 蒸発する
    subroutine sea_evaporate
      real :: Tz  ! 温度（高度 0)Tz
      real :: e   ! 蒸発量 e

      Tz = air_sea0_getTemp
      e = 1.155 * (sea%Ts - Tz)
      call cloud_sea1_receive_water(e)
      call sea_radiate_heat
    end subroutine sea_evaporate

    ! 熱放射する
    subroutine sea_radiate_heat
      sea%Qr = epsilon * (sigma * 4.18 * 3.6
        * 10.0**(-2)) * (sea%Ts + 0.948)**4
      call air_sea0_absorb_heat_sea(sea%Qr)
      call sea_drop_in_temp
    end subroutine sea_radiate_heat

    ! 温度が低下する
    subroutine sea_drop_in_temp
      sea%Ts = sea%Ts - sea%Qr / (sea%Ms
        * sea%C)
    end subroutine sea_drop_in_temp

    ! 海へ流出する
    subroutine sea_outflow
    end subroutine sea_outflow

    ! 雨/雪を受ける
    subroutine sea_receive_rain_snow
    end subroutine sea_receive_rain_snow

    ! 大気から熱吸収する
    subroutine sea_absorb_heat_air(Qa)
      real :: Qa

      sea%Q = Qa * sea%S
    end subroutine sea_absorb_heat_air

    ! 表面温度を設定する
    subroutine sea_setTs(temp)
      real :: temp

      sea%Ts = temp
    end subroutine sea_setTs

    ! 保有熱量を設定する
    subroutine sea_setQ(temp)
      real :: temp

      sea%Q = temp
    end subroutine sea_setQ

    ! 比熱を設定する
    subroutine sea_setC(temp)
      real :: temp

      sea%C = temp
    end subroutine sea_setC

    ! 表面質量を設定する
    subroutine sea_setMs(temp)
      real :: temp

      sea%Ms = temp
    end subroutine sea_setMs

    ! 面積を設定する
    subroutine sea_setS(temp)
      real :: temp

      sea%S = temp
    end subroutine sea_setS

end module sea_object
```

図 13 “水の大気循環” における “海” オブジェクトの OBF 記述例

Fig. 13 OBF description example: “Sea” object in “atmospheric circulation of water”

7.1 ODDJ 記述例との比較

ODDJ 記述例と OBF 記述例の比較については 6 節で述べたとおりであり, オブジェクトフレームを変換規則に従ってオブジェクトモジュールに変換した. 変換時に DU が改めて情報を追加する必要はなく, このことからオブジェクトフレームの変換については ODDJ/OBF トランスレータを用いて自動変換することが可能であると考えられる.

7.2 OEDJ 記述例との比較

OBF 記述例と OEDJ 記述例を比較すると, オブジェクトの持つ属性やメソッドの記述にほとんど差がないことがわかる. 相違点としては, OBF では属性の値を取り扱うために自身を表す変数 “sea” を経由して行う点や, オブジェクト間の関連を明示的に宣言している点があるが, 言語の持つ記述能力については大差ないものといえる.

8. 今後の課題

8.1 ライブラリフレームの変換

ODDJ 記述を構成するライブラリフレームについての扱いがいまだ明確になっていない. ライブラリフレームの扱いについて設計段階と実装段階の双方で検討を行う予定である.

8.2 設計段階と実装段階の統合環境の開発

現在の ODDJ/OBF トランスレータはスタンドアロンで動作させることを想定して開発を行ったが, 将来的には ODDJ エディタと統合し, DU に対して ODDJ 記述の作成と ODDJ 記述から OBF 記述への変換をシームレスに行える環境を提供することを目指している. また, 分析段階の OONJ 記述も含め, 分析, 設計, 実装の全ての段階を統合した開発環境の構想もある.

8.3 一貫記述例の作成

“水の大気循環” 以外にも記述例を作成し, OBF の仕様が十分なものであることと, ODDJ/OBF トランスレータの動作が正常に行えることの検証を行う.

8.4 自動分散化システムの組み込み

DU の対象とする数値シミュレーション分野では計算規模（メモリ量, 計算量）の増大に伴って分散や並列計算を行うことが望まれるようになっているが, DU が通信処理や同期処理といった分散プログラムに特有の技術を習得し, 手動でプログラムの分散化を行うには手間と時間がかかり困難である. この問題の解決方法としては DU が作成した逐次型のプログラムを自動的に分散化することが考えられ, 我々の研究グループでは C++言語記述のプログラムを対象として自動分散化システム ADS (Automatic Disitribution System)[13] の開発が行われてきた.

DU にとって, より有用なシミュレーション開発環境となるよう, 自動分散化システムを OOJ に組み込むことを目指す. 具体的には設計段階と実装段階に手を加え, 設計段階では自動分散化に必要な情報の埋め込みを, 実装段階ではその情報を基に分散プログラムを出力することを検討している.

参 考 文 献

1) 畠山正行: オブジェクト指向一貫記述言語 OOJ の構成とその概念設計, 情報処理学会研究報告, 2005-SE-150 (2005).
2) 畠山正行, 松本賢人: オブジェクト指向自然日本語記述言語 OONJ の設計とその記述例, 情報処理学会研究報告, 2005-HPC-102, pp. 13–22 (2005).
3) 松本賢人, 畠山正行, 安藤宣晶: オブジェクト指向分析記述言語 OONJ の設計原理構築と記述環境開発, 情報処理学会研究報告, 2005-SE-150 (2005).
4) 川澄成章, 畠山正行, 野口和義: オブジェクト指向設計記述言語 ODDJ の設計とその記述環境の開発, 情報処理学会研究報告, 2005-SE-150 (2005).
5) 伊藤善規: 自律的な適応機能を導入した数値流体シミュレーション法の開発, 修士論文, 茨城大学大学院理工学研究科 (1999).
6) 峯村吉泰: Java による流体・熱運動の数値シミュレーション, 森北出版 (2001).
7) 宇宙航空研究開発機構: 他分野統合基盤ソフトウェア (UPACS), `http://www.ista.jaxa.jp/res/c02/upacs/index.html`.
8) Decyk, V. K., Norton, C. D. and Szymanski, B. K.: Expressing Object-Oriented Concepts in Fortran 90, *ACM Fortran Forum*, Vol.16, No.1, pp. 13–18 (1997).
9) Akin, E.: *Object-Oriented Programming Via Fortran 90/95*, Cambridge University Press (2003).
10) 加藤木和夫, 畠山正行: オブジェクト指向実装記述言語 OEDJ の記述環境およびトランスレータの開発, 情報処理学会研究報告, 2005-SE-150 (2005).
11) Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W.: オブジェクト指向方法論 OMT, トッパン (1992). 羽生田栄一 監訳.
12) 畠山正行: オブジェクト指向分析自然日本語構造化フレーム OOSF の設計と表現技法, 日本シミュレーション学会誌, Vol. 23, No. 4, pp. 308–325 (2004).
13) 上原均, 畠山正行: オブジェクト指向シミュレーションの自動分散化の実現と評価, 日本シミュレーション学会誌, Vol.17, No.4, pp.310–323 (1998).