

RDoc を用いた Fortran90/95 プログラム のドキュメント生成

北海道大学 理学研究科
地球惑星科学専攻
地球流体力学研究室 D1
森川 靖大



(大) 目標

- 可読性・可変性に優れた大気大循環モデル (GCM) を作る
 - 可読性: ソースコードの読み書きが簡単
 - 可変性: 物理過程の交換や分離, 力学過程の変更が簡単

- こんな GCM にできると良いな・・・
 - お手軽に動かせる GCM
 - カスタマイズが簡単にできる GCM

可読性・可変性に必要なのは・・・？

■ ソースコードの書き方の工夫

- 例: 気象庁コーディングルール, AGCM5, FMS マニュアル, GFD Dennou Club dcmode1 コーディングルール

■ プログラムの構造の工夫

- 例: FMS, DCPAM (spml, gt4f90io, ISPACK)

■ ドキュメントの整備の工夫

はじめに

- **ドキュメントの重要性**
 - 第三者への提供 (プログラムの利用)
 - 開発や保守の効率化 (プログラムの改変)

- **Fortran による数値モデルのドキュメント**
 - 数理、離散化ドキュメント :: TeX
 - ◆ 数式の記述に最適

 - リファレンスマニュアル :: HTML
 - ◆ Web からの参照に最適、ハイパーリンクが便利

リファレンスマニュアルの作成

■ 何が厄介かというところ...

- プログラムとマニュアルを別々に用意するのは面倒
- プログラムで書いたものをもう一度書くのは面倒
- Java や Ruby では、JavaDoc や RDoc があるが、Fortran の規格にはドキュメント生成機能は無い...

■ 過去に行われた工夫

- XML を用いた FMS (GFDL) の試み
- RD を用いた我々のこれまでの試み

■ そこで

- RDoc を用いた Fortran ソースコードの自動解析

XML を用いた FMS (GFDL) の試み

6 / 25

■ Fortran95 に XML のドキュメントを埋め込む

- FMS (Flexible Modeling System: GFDL) で試行
- FMS 謹製のツールで HTML へ変換

Fortran95 ソースコード

```
module module_name_mod
! <OVERVIEW>
!   module_name_mod の概要
! </OVERVIEW>
implicit none
private
public :: module_name_init, module_name_end
! <SUBROUTINE NAME="module_name_init">
! <OVERVIEW>
!   モジュールの初期化
! </OVERVIEW>
! <TEMPLATE>
!   module_name_init (inchar, outint)
! </TEMPLATE>
! <IN NAME="inchar" TYPE="character" >
!   文字型の入力変数
! </IN>
! <OUT NAME="outint" TYPE="integer" >
!   整数型の出力変数
! </OUT>
subroutine module_name_init(inchar, outint)
character(*)      , intent(in) :: inchar
integer(INTKIND), intent(out) :: outint
end subroutine module_name_init
! </SUBROUTINE>
end module module_name_mod
```

```
! <SUBROUTINE NAME="module_name_init">
! <OVERVIEW>
!   モジュールの初期化
! </OVERVIEW>
! <TEMPLATE>
!   module_name_init (inchar, outint)
! </TEMPLATE>
!
! </SUBROUTINE>
```

- ◆ XML でコメントを記述する
- ◆ このコメント部分を抜き出し、XML から HTML マニュアルを作成

■ Fortran95 に XML のドキュメントを埋め込む

- FMS (Flexible Modeling System: GFDL) で試行
- FMS 謹製のツールで HTML へ変換
- 利点
 - ◆ XML の利用により、構造的なマニュアルを作成することが出来る
- 欠点
 - ◆ XML のタグで Fortran コードが汚くなる
 - ◆ 手続き名や引数をコードとマニュアルで 2 度書く必要あり

RD を用いた我々のこれまでの試み

■ Fortran95 に RD 形式のコメントを埋め込む

- RD を用いることでタグが簡潔に
- rdtool によって RD を HTML に変換

Fortran95 ソースコード

```

! =begin
! = Module module_name_mod : Sample module
! * Developers: Yasuhiro Morikawa
! == Overview
! module_name_mod の概要

module module_name_mod
  implicit none

  ! == Public Interface
  private
  public :: module_name_init, module_name_end

  ! == Procedure Interface
  ! == Subroutine module_name_init : モジュールの初期化
  ! NAMELIST を入力し、グローバル変数を allocate する。
  subroutine module_name_init(inchar, outint, inoutdata)
    ! == Input
    character(*) , intent(in) :: inchar
    ! == Output
    integer(INTKIND), intent(out) :: outint
  end subroutine module_name_init
! =end
end module module_name_mod

```

```

! =begin
:
! =end

```

```

! = Module ..
! * Develop..
! == Overview

```

```

! =begin
:
subroutine ..
char.., int..
:
! =end

```

! =begin ~ ! =end の部分をマニュアルに反映

コメントには RD の文法を利用。見出しやリストは “=” や “*” で簡潔に表現

引用仕様 (引数の名前と型など) に関するソースコードをそのままマニュアルへ

RD を用いた我々のこれまでの試み

■ Fortran95 に RD 形式のコメントを埋め込む

● 利点

- ◆ ソースとマニュアルとで 2 度書く手間を軽減
- ◆ XML タグと比較してソースが汚れない

● 欠点

- ◆ モジュール間の依存関係を解釈不能
 - ▶ 複数のファイルを同時に解析しないと不可能
- ◆ ソース内のタグ (“=begin” 等) はやっぱりそれなりに汚い

■ ソースコード自動解析の必要性

■ 何ぞや？

- Ruby で書かれたソースコードからドキュメントを自動生成する Ruby の標準ライブラリ

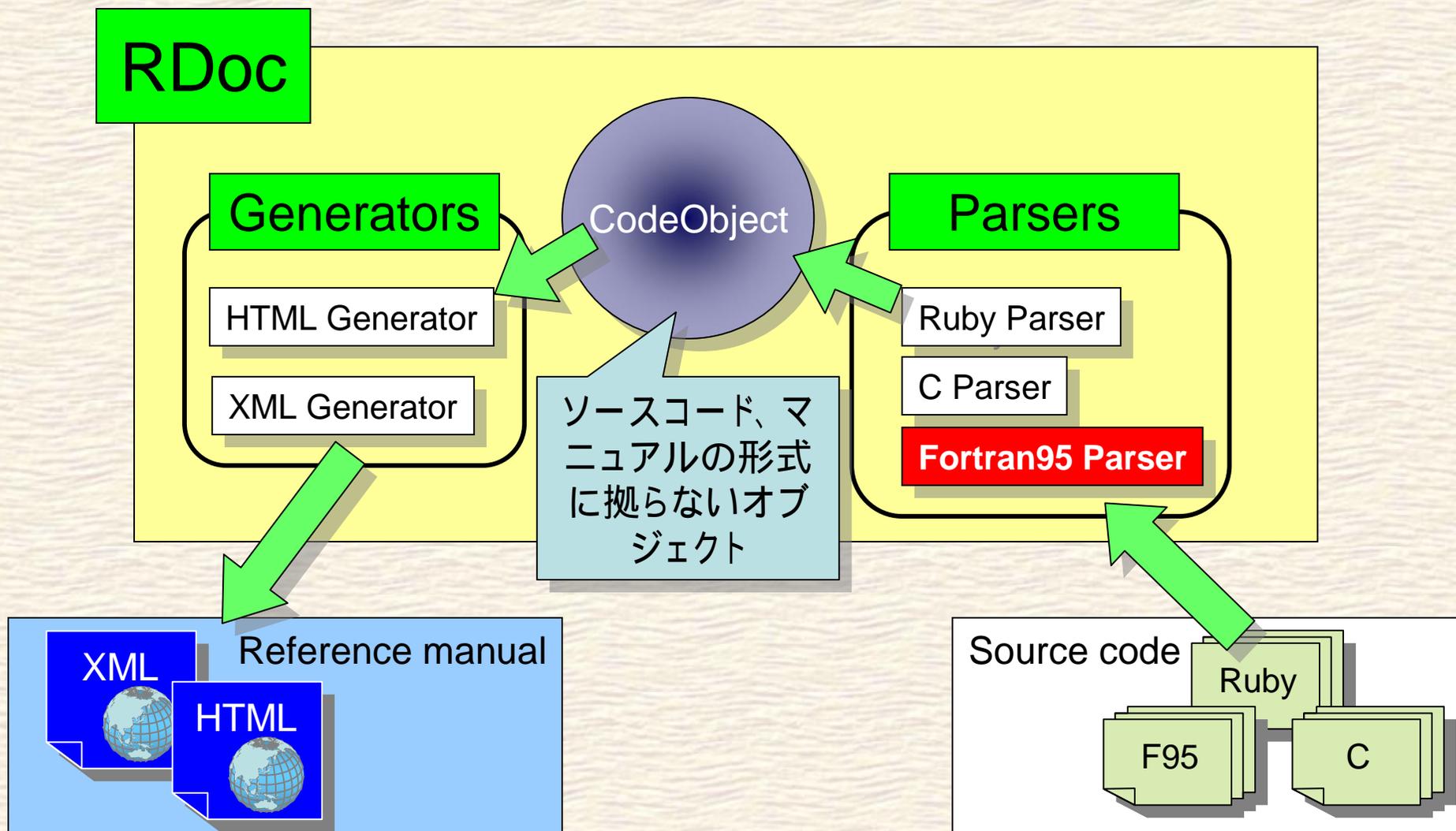
■ Fortran90/95 との関係は？

- ソースコード解析機構とマニュアル生成機構が分離しているため、他の言語で書かれたソースコードも解析可能
- 標準で C および **Fortran95** 用の解析機構が付属

■ 利点は？

- RD の利点 (マニュアルのソースへの埋め込み) を引継ぎ、欠点 (タグの簡素化、依存関係の解釈) を克服

RDoc によるドキュメント生成の流れ



■ Fortran90/95 の文法を解釈

- RDoc のタグは RD よりもさらに簡潔
- 別ファイル内のモジュール等へ自動的にリンク作成

Fortran95 ソースコード

```
!= Module module_name_mod : Sample module
! Authors:: Yasuhiro MORIKAWA
!
! This module depends base_mod module
!
module sample_mod
  use base_mod
  implicit none
  private
  public :: sample_init, sample_end, Const

  real(8) :: Const = 3.14

  subroutine sample_init(inchar, outint)
    character(*) , intent(in) :: inchar
    integer(INTKIND), intent(out):: outint
  end subroutine sample_init

  subroutine sample_end(err)
    logical, intent(inout) :: err
  end subroutine sample_end

end module sample_mod
```

```
!= Module ..
! Authors:: ..
```

```
use base_mod
:
module module_..
:
subroutine mo..
:
end subrou..
end module modu..
```

RDoc のタグは “=” や “::” で表記 (RD に近い文法)

module 文, use 文, subroutine 文を解釈。別ファイルのモジュールへのリンクを自動で作成

Fortran95 Parser (オリジナル版)

13 / 25

■ Fortran90/95 の文法を解釈

- RDoc のタグは RD よりもさらに簡潔
- 別ファイル内のモジュール等へ自動的にリンク作成

Fortran95 ソースコード

```
!= Module module_name_mod : Sample module
! Authors:: Yasuhiro MORIKAWA
!
! This module depends base_mod module
!
module sample_mod
  use base_mod
  implicit none
  private
  public :: sample_init, sample_end, Const

  real(8) :: Const = 3.14

  subroutine sample_init(inchar, outint)
    character(*) , intent(in) :: inchar
    integer(INTKIND), intent(out):: outint
  end subroutine sample_init

  subroutine sample_end(err)
    logical, intent(inout) :: err
  end subroutine sample_end

end module sample_mod
```

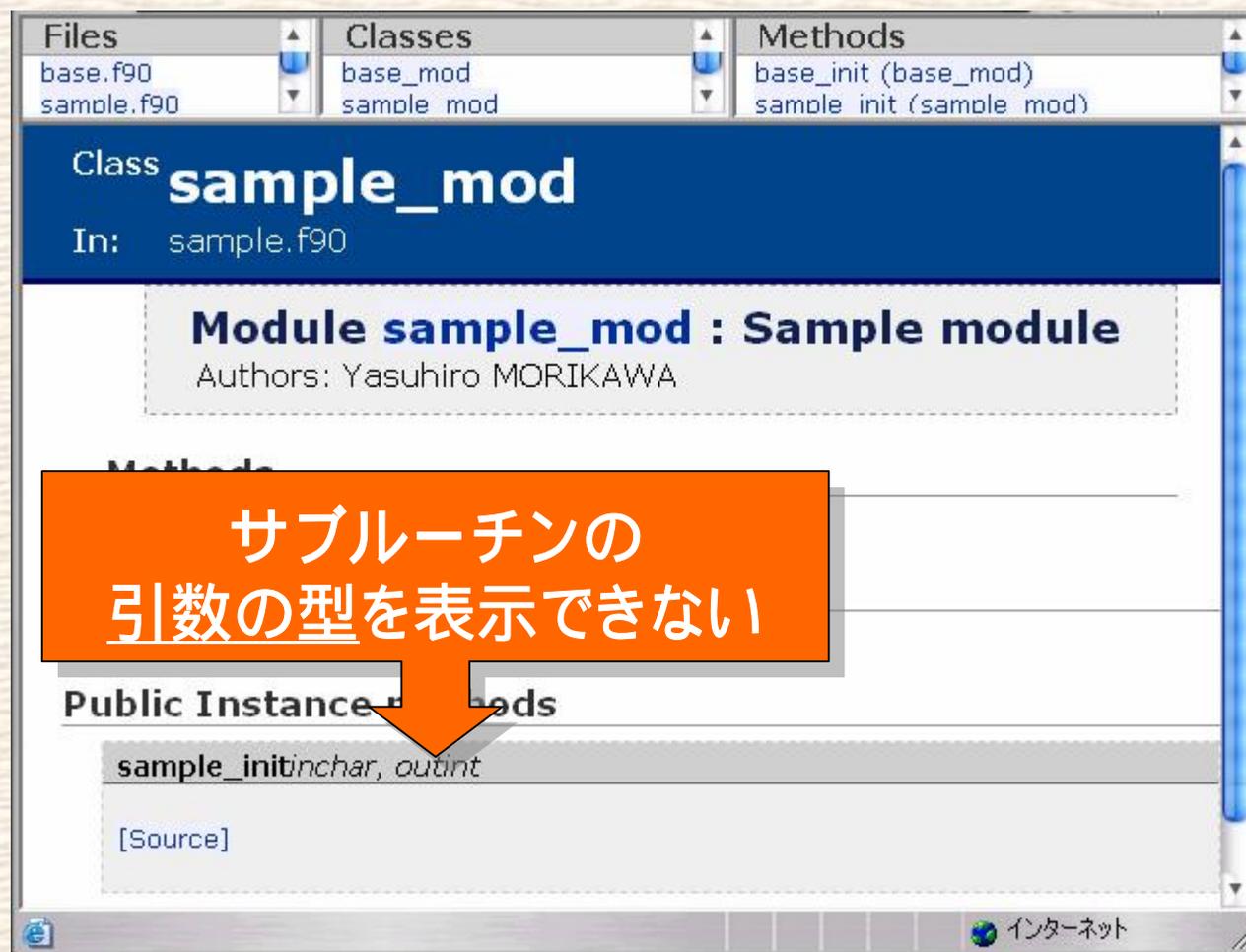
Rdoc

HTML のリファレンスマニュアル

Files	Classes	Methods
sample.f90 base.f90	sample_mod base_mod	sample_init sample_end
Class <code>sample_mod</code> In: <code>sample.f90</code>		
Module <code>sample_mod</code> : Sample module Authors: Yasuhiro MORIKAWA This module depends base_mod module		
Methods sample_init sample_end		
Included Modules base_mod		
Public instance methods		
<code>sample_init</code> (<i>inchar, outint</i>) [source]		
<code>sample_end</code> (<i>err</i>) [source]		

ハイパーリンク

■ 解析機能の不足



■ 解析機能の不足

The screenshot shows a web-based interface for a Fortran95 parser. At the top, there are three panels: 'Files' (listing base.f90 and sample.f90), 'Classes' (listing base_mod and sample_mod), and 'Methods' (listing base_init (base_mod) and sample_init (sample_mod)). Below these is a detailed view for the 'Class sample_mod' located in 'sample.f90'. It includes a 'Module sample_mod : Sample module' section with the author 'Yasuhiro MORIKAWA'. Under 'Methods', 'sample_init' is listed. Below that, 'Included Modules' shows 'base_mod'. Under 'Public Instance methods', 'sample_init(char, outint)' is listed with a '[Source]' link. An orange callout box with an arrow points to the '[Source]' link, containing the text: サブルーチンの解説コメントを表示できない (Cannot display explanatory comments for subroutines).

■ 解析機能の不足

The screenshot shows a web-based interface for a Fortran95 parser. At the top, there are three panes: 'Files' (base.f90, sample.f90), 'Classes' (base_mod, sample_mod), and 'Methods' (base_init (base_mod), sample_init (sample_mod)). Below these is a section for 'Class sample_mod' with 'In: sample.f90'. A dashed box contains 'Module sample_mod : Sample module' and 'Authors: Yasuhiro MORIKAWA'. Under 'Methods', 'sample_init' is listed. 'Included Modules' shows 'base_mod'. 'Public Instance methods' is underlined in red, with 'sample_init(inchar, outint)' listed below it. An orange arrow points from a text box to this section. The text box contains the text: 'public, private を 区別できない (全て public 扱いになる)'. At the bottom, there is a '[Source]' link and a status bar with 'インターネット'.

■ 解析機能の不足

The screenshot shows a Fortran95 IDE interface. At the top, there are three panes: 'Files' (listing base.f90 and sample.f90), 'Classes' (listing base_mod and sample_mod), and 'Methods' (listing base_init (base_mod) and sample_init (sample_mod)). Below these is a detailed view of the 'Class sample_mod' from 'sample.f90'. The view includes sections for 'Module sa...', 'Methods' (listing sample_init), 'Included Modules' (listing base_mod), and 'Public Instance me...'. An orange callout box is overlaid on the right side of the IDE, containing a list of features that cannot be expressed in Fortran95. The callout box has a large arrow pointing downwards towards the source code area.

Class **sample_mod**
In: sample.f90

Module sa
Authors: Yasuh

Methods
sample_init

Included Modules
base_mod

Public Instance me
sample_init(inchar, ou
[Source]

インターネット

- そもそも表現できない要素いろいろ
- ◆ 関数 (function 文)
 - ◆ モジュールが公開する変数, 定数
 - ◆ 構造体 (type 文)
 - ◆ 利用者定義演算子 (operator)
 - ◆ 利用者定義代入 (assignment)
 - ◆ 総称手続き (interface 文)
- etc ...

問題点の解決に向けて

■ 開発者・メンテナに連絡

- 森川 「こうなると嬉しいのだけど」と連絡
- 開発者 「メンテナさんに連絡しておいたよ」
- メンテナ 「... (返事来ず)」
 - » 後に連絡がつき、以降で紹介する改良版のパッチを送付.
 - » 2005/02/16 現在, まだコミットはされていない模様

■ しょうがないので (?) 自力で改造

- 見た感じで足りないと思うところから五月雨的に解析機能追加

Fortran95 Parser (強化版)

アドレス(D) http://shadow.vir/~morikawa/rdoc_sample/doc/

Files	Classes	Methods
base.f90 sample.f90	base_mod sample_mod	TYPE_A (sample_mod) base_init (base_mod) const (sample_mod)

Class **sample_mod**
In: sample.f90

Derived Types: TYPE_A

Public Instance Methods

TYPE_A()
Derived Type :
counter : integer
: 構造体内部の変数の解説
構造体の解説

構造体の解析
要素、要素の型、解説の表示

Fortran95 Parser (強化版)

20 / 25

```
const()
Constant :
const = 3.14 : real(8), parameter
          : 公開定数
```

```
sample_fun
Function :
```

定数の解析
型、初期値、解説の表示

[\[Source\]](#)

```
sample_init( inchar, outint )
Subroutine :
inchar : character(*) , intent(in)
          : 入力変数
outint : integer(INTKIND), intent(out)
          : 出力変数
```

初期化サブルーチン

[\[Source\]](#)

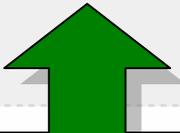
Fortran95 Parser (強化版)

```
const()
Constant :
const = 3.14 : real(8), parameter
          : 公開定数
```

```
sample_func( log ) result(res)
Function :
res :      logical
          : 論理型の返り値
log :     logical, intent(in)
          : 論理型入力変数
```

関数

[\[Source\]](#)



関数の解析
解説の表示

初期化サブルーチン

[\[Source\]](#)

Fortran95 Parser (強化版)

22 / 25

const()

Constant :

```
const = 3.14 : real(8), parameter  
           : 公開定数
```

sample_func(log) result(res)

Function :

```
res :      logical  
       : 論理型の返り値  
  
log :      logical, intent(in)  
       : 論理型入力変数
```

関数

[\[Source\]](#)

sample_init(inchar, outint)

Subroutine :

```
inchar :    character(*), intent(in)  
           : 入力変数  
  
outint :    integer(INTKIND), intent(out)  
           : 出力変数
```

初期化サブルーチン

[\[Source\]](#)

引数の解析
型、解説の表示

Fortran95 Parser (強化版)

23 / 25

関数

[Source]

sample_init(inchar, outint)

Subroutine :

inchar : character(*), intent(in)
: 入力変数

out int : integer(INTKIND), intent(out)
: 出力変数

初期化サブルーチン

[Source]

Private Instance methods

internal()

Variable :

internal : integer, save
: 非公開変数

[Validate]

public, private
を区別

Fortran95 Parser (強化版)

24 / 25

関数

[Source]

sample_init(inchar, outint)

Subroutine :

inchar : character(*) , intent(in)
: 入力変数

変数の解析
型、初期値、解説の表示

Private Instance methods

internal()

Variable :

internal : integer, save
: 非公開変数

[Validate]

インターネット

まとめ

- RDoc はかなり便利
- RDoc の Fortran95 Parser を改良
 - Fortran90/95 ソースコードからリファレンスマニュアルを自動生成
- 強化版パッチの公開アドレス
 - <http://www.gfd-dennou.org/library/dcmode1>
 - ◆ パッチを当てたパッケージも配布してます
 - ◆ コメントの書法等の解説あります
- 使用例
 - <http://www.gfd-dennou.org/library/gtool4>
 - <http://www.gfd-dennou.org/library/dcpam>

参考資料

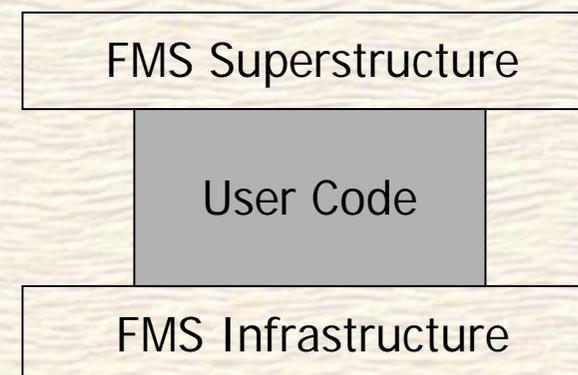
- 数値モデリングプロジェクト **dcmodel**
 - <http://www.gfd-dennou.org/library/dcmodel/>
- オブジェクト指向スクリプト言語 **Ruby**
 - <http://www.ruby-lang.org>
- **Fortran からドキュメントを自動生成するツール**
 - f90tohtml
 - ◆ <http://mensch.org/f90tohtml/>
 - f90doc
 - ◆ <http://theory.lcs.mit.edu/~edemaine/f90doc/>
- 惑星大気モデル **DCPAM**
 - <http://www.gfd-dennou.org/library/dcpam/>
- **FMS (Flexible Modeling System)**
 - <http://www.gfdl.noaa.gov/~fms/>
- **The FMS Manual**
 - <http://www.gfdl.noaa.gov/~vb/FMSManual/>
- 気象庁 **Fortran コーディングルール**
 - <http://www.mri-jma.go.jp/Project/mrinpd/coderule.html>

付録

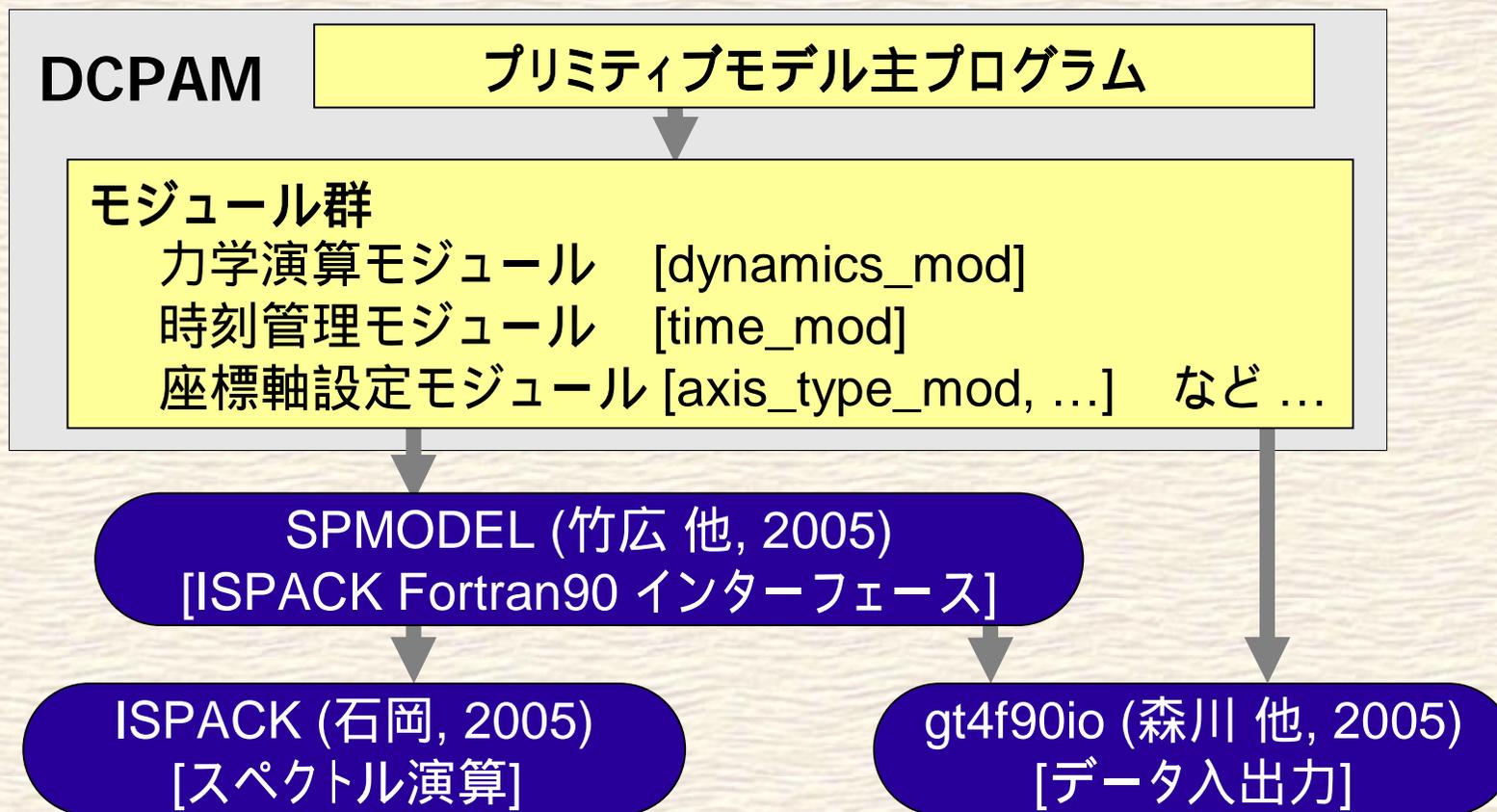


AGCM5, FMS

- **AGCM5** (沼口, 1992; SWAMP Project, 1998; <http://www.gfd-dennou.org/arch/agcm5>)
 - FORTRAN77 で可読性向上の工夫
 - ◆ 変数命名規則
 - ◆ プログラム書法を工夫
 - ◆ リファレンスマニュアルを整備
 - FORTRAN77 の制約大
- **FMS** (Flexible Modeling System; Geophysical Fluid Dynamics Laboratory , 2005)
 - モデルの可変性向上の工夫
 - ◆ Infrastructure: データI/O, 並列化, 機種依存性の吸収をサポート
 - ◆ Superstructure: 大気・海洋・氷床モデルの結合をサポート
 - User Code の可読性・可変性は？



- サブルーチンや関数は必ずモジュールにまとめることで、個々のプログラムの着脱を容易に
- 初期化の手順の統一により交換・分離を容易に
 - 必ず `module_init` というサブルーチンにより初期化 (`module` は各モジュール名)
- ファイル名、モジュール名、サブルーチン名の命名規則 [The FMS Manual (Baraji, 2002)]
 - ファイル名 : `module.f90`, 例: `dynamics.f90`
 - モジュール名 : `module_mod`, 例: `dynamics_mod`
 - サブルーチン名: `module_init`, 例: `dynamics_init`



本研究のモデル (DCPAM) の構造

XML を用いた FMS (GFDL) の試み

■ Fortran95 に XML のドキュメントを埋め込む

- FMS (Flexible Modeling System: GFDL) で試行
- FMS 特製のツールで HTML へ変換

Fortran95 ソースコード

HTML のリファレンスマニュアル

```

module module_name_mod
! <OVERVIEW>
!  module_name_mod の概要
! </OVERVIEW>
implicit none
private
public :: module_name_init, module_name_end
! <SUBROUTINE NAME="module_name_init">
! <OVERVIEW>
!   モジュールの初期化
! </OVERVIEW>
! <TEMPLATE>
!   module_name_init (inchar, outint)
! </TEMPLATE>
! <IN NAME="inchar" TYPE="character" >
!   文字型の入力変数
! </IN>
! <OUT NAME="outint" TYPE="integer" >
!   整数型の出力変数
! </OUT>
subroutine module_name_init(inchar, outint)
character(*)      , intent(in) :: inchar
integer(INTKIND), intent(out) :: outint
end subroutine module_name_init
! </SUBROUTINE>
end module module_name_mod
  
```

Module module_name_mod

OVERVIEW

module_name_end の概要

PUBLIC INTERFACE

[module_name_init:](#)

モジュールの初期化

PUBLIC ROUTINES

a. module_name_init

call module_name_init (inchar, outint)

INPUT

inchar 文字型の入力変数
[character]

OUTPUT

outint 整数型の出力変数
[integer]

RD を用いた我々のこれまでの試み

32 / 25

- Fortran95 に RD 形式のコメントを埋め込む
 - rdtool によって RD を HTML に変換

Fortran95 ソースコード

```
! =begin
! = Module module_name_mod : Sample module
! * Developers: Yasuhiro Morikawa
! = Overview
! module_name_mod の概要

module module_name_mod
  implicit none

  ! = Public Interface
  private
  public :: module_name_init, module_name_end

  ! = Procedure Interface
  ! == Subroutine module_name_init : モジュールの初期化
  ! NAMELIST を入力し、グローバル変数を allocate する。
  subroutine module_name_init(inchar, outint, inoutdata)
    ! == Input
    character(*) , intent(in) :: inchar
    ! == Output
    integer(INTKIND), intent(out) :: outint
  end subroutine module_name_init
end module module_name_mod
! =end
```

rdtool

HTML のリファレンスマニュアル

Module module_name_mod : Sample module

• Developers: Morikawa Yasuhiro

Overview

module_name_mod の概要

Public Interface

```
private
public :: module_name_init, module_name_end ! subroutines
```

Procedure Interface

Subroutine module_name_init : モジュールの初期化

NAMELIST を入力し、グローバル変数を allocate する。

```
subroutine module_name_init(inchar, outint, inoutdata, inoutdb)
```

Input

```
character(*) , intent(in) :: inchar ! Input Character
```

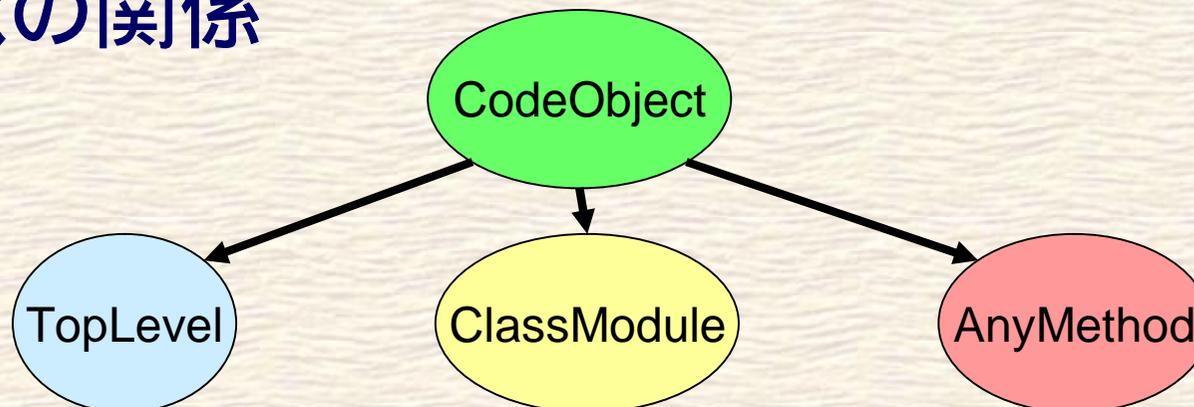
Output

```
integer(INTKIND), intent(out) :: outint ! Output Integer
```

■ ソースコード内の情報を階層的に保持するためのオブジェクト

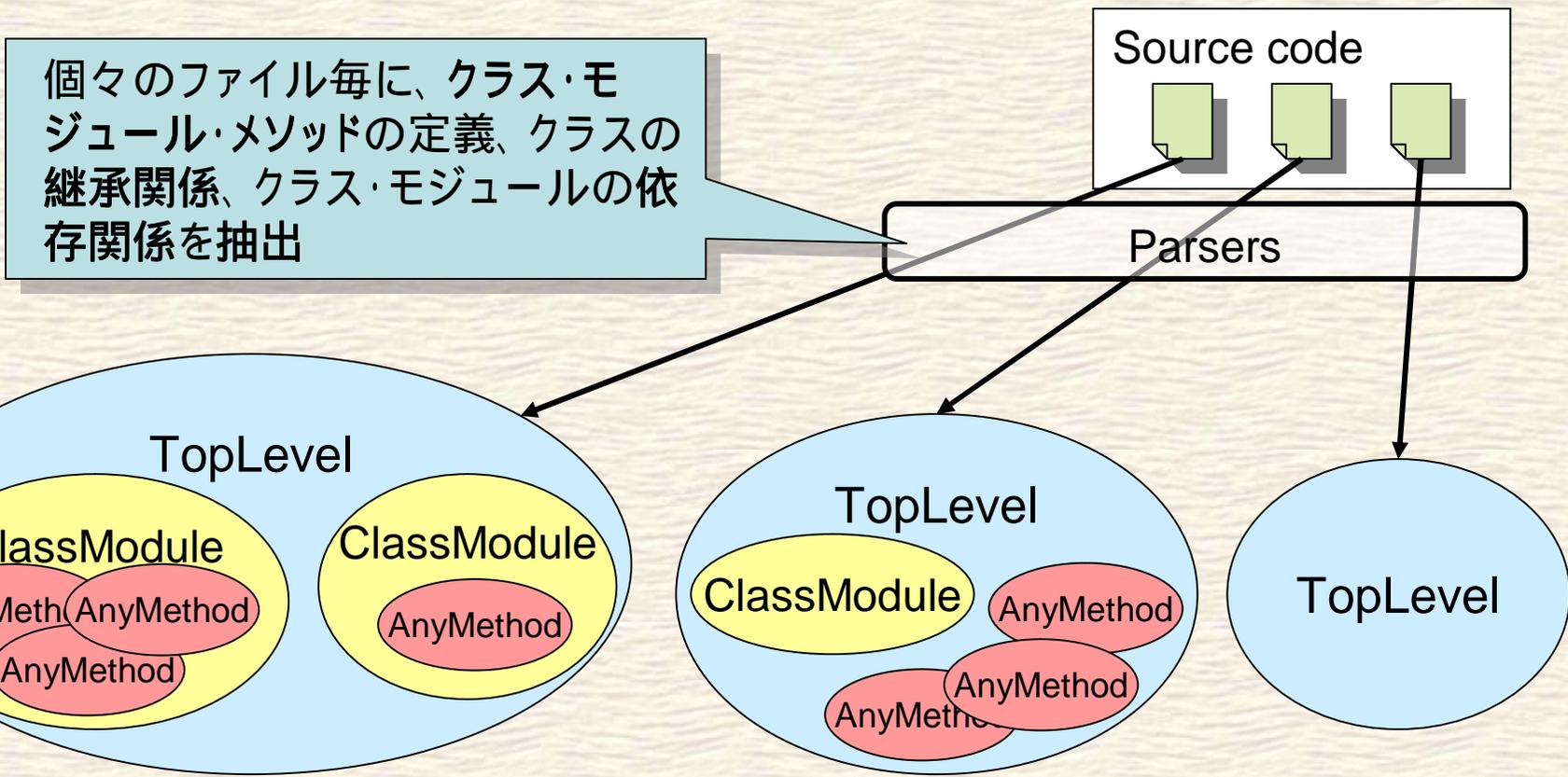
- CodeObject : 共通する情報
- TopLevel : ファイルの情報
- ClassModule : クラス、モジュールの情報
- AnyMethod : メソッドの情報

■ 継承の関係

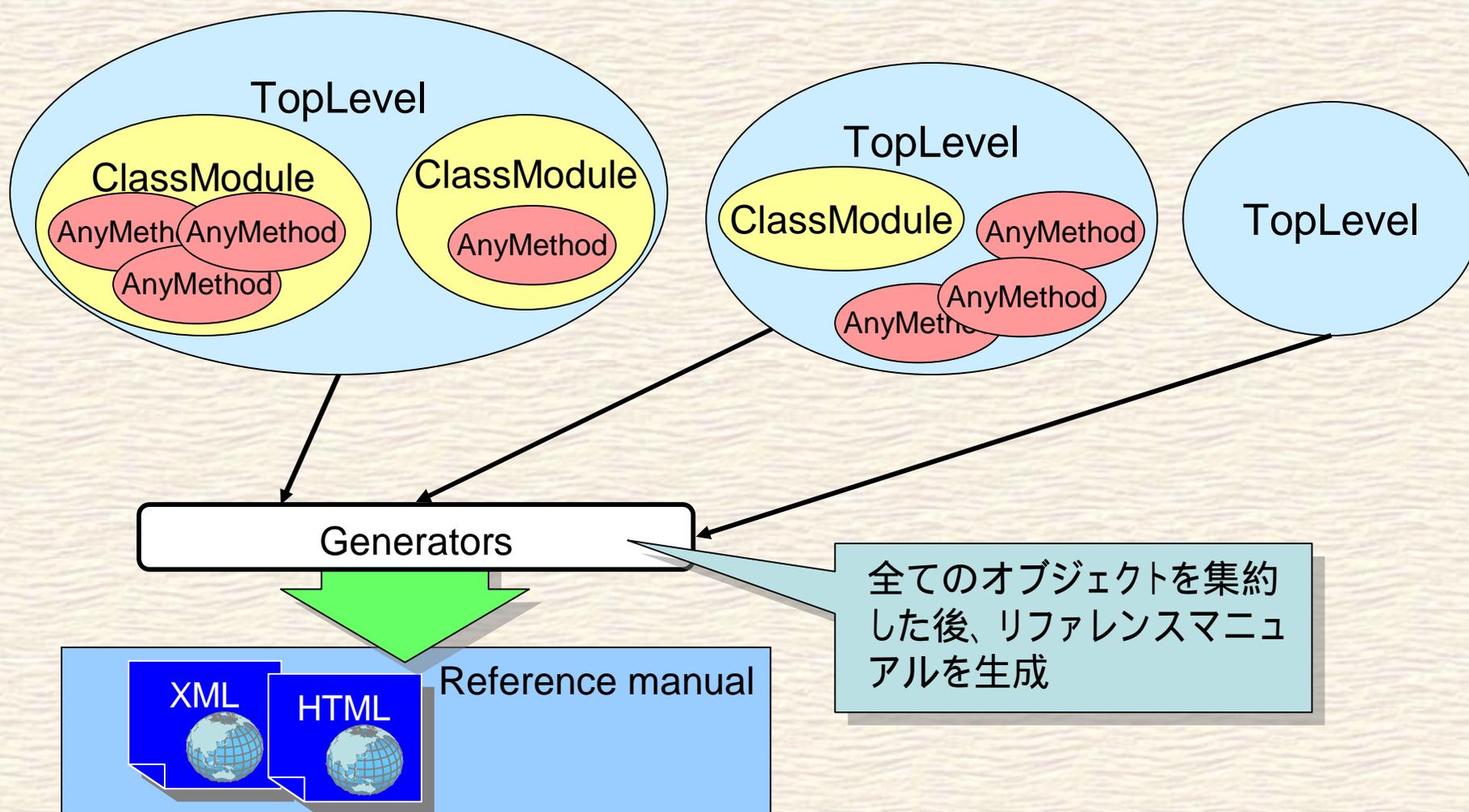


ソースコード CodeObject

- ソースコード内の情報を TopLevel, ClassModule, AnyMethod へ



■ HTML 等のドキュメントを生成



Fortran95 Parser (強化版)

■ 解析機能の強化の具体例

Fortran95 ソースコード

```
! = Module sample_mod : Sample module
! Authors:: Yasuhiro MORIKAWA
!
module sample_mod
  use base_mod
  !
  ! sample_mod の概要
  implicit none
  private
  public :: sample_init, sample_func, const, TYPE_A
  private:: internal

  type TYPE_A
    ! 構造体の解説
    integer :: counter ! 構造体内部の変数の解説
  end type TYPE_A

  real(8), parameter :: const = 3.14 ! 公開定数
  integer, save :: internal ! 非公開変数

  subroutine sample_init(inchar, outint)
    ! 初期化サブルーチン
    character(*) , intent(in) :: inchar ! 入力変数
    integer(INTKIND), intent(out):: outint ! 出力変数
  end subroutine sample_init

  function sample_func(log) result(res)
    ! 関数
    logical, intent(in) :: log ! 論理型入力変数
    logical :: res ! 論理型の戻り値
  end function sample_func
end module sample_mod
```

public:: samp...
private:: inte...

type TYPE_A ...

real(8), parame..
Integer, save..

! 初期化...

char.. ! 入力..
integer.. ! 出力..

function samp...
:
end function ..

公開要素と非公開要素の区別

構造体

公開定数、
公開変数

サブルーチン、
関数のコメント

引数の型、
コメント

関数

■ 解析機能の強化

- 解析可能になった要素のリスト
 - ◆ 関数 (function 文)
 - ◆ サブルーチンや関数の引数の型
 - ◆ モジュールが公開する変数, 定数
 - ◆ 構造体 (type 文)
 - ◆ NAMelist 文
 - ◆ 利用者定義演算子 (operator), 利用者定義代入 (assignment)
 - ◆ 上記要素のコメント文
 - ◆ 総称手続き (interface 文)
 - ◆ 公開要素と非公開要素との区別
 - ◆ 孫引きされている公開要素
- 大文字小文字の違いを無視