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

2005－SE－150（9）
2005／11／29

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

川 澄 成 章[†]　野 口 和 義[††]
畠 山 正 行[††]　荒 木 俊 郎[††]

本研究の目的は記述言語 OONJ の分析記述を元に対象世界を計算機で駆動させるための設計記述言語を設計することである．そのため，OONJ と同じ基盤に立つオブジェクト指向設計記述言語 ODDJ と，ドメインユーザによる追加／修正／変換作業を支援する ODDJ 記述環境を開発した．ODDJ では，OONJ 記述から計算機で利用できる情報を抽出し，不足している情報を記述環境を利用してドメインユーザに追加させる．そのための ODDJ 構造化記述規則を設計した．特にオブジェクトの初期化や駆動制御記述用の記述規則が重要である．また ODDJ 記述環境は OONJ のものと類似性を持たせた設計で開発した．現時点で ODDJ の設計は一定の水準まで完成し，記述環境の開発途中である．また水の大気循環の OONJ 記述を元に ODDJ 記述を作成し，更にその ODDJ 記述からプログラミング言語記述に変換できることを検証できた．結論としては，OOJ における設計段階の記述言語の果たすべき機能は現時点でもある程度は実現できているであろうと評価している．今後の課題は，更に一貫記述例を作成し，下流の実装記述言語や記述環境と連携して検証を行うことである．

A Design of Object-oriented Design Description Japanese ODDJ and Development of It's Description Environment

Shigeaki Kawasumi [†], Kazuyoshi Noguchi [†],
Masayuki Hatakeyama [††] and Toshirou Araki [††]

This study aims at designing a design description language to drive/execute the descriptions in the computer system. To attain the aim, we have designed the Object-oriented Design Description Japanese ODDJ based on the same paradigm of the OONJ, and have developed the ODDJ description environments to support the user works for the addition/modification/transformation of the design descriptions. The ODDJ structured description rules have been designed, especially for the initializations of the objects and the drive control descriptions. The ODDJ description environment has been developed based on the same design paradigm of the OONJ. At present, the design of the ODDJ has already completed at a certain level, and the description environments are now in developing. We have made up the ODDJ descriptions for the atmospheric circulations of the water, and succeeded in transforming this ODDJ descriptions into the programming language descriptions. As the conclusion, the ODDJ has already realized the functions of the design description language on some level. The future works are to verify the consistency of the integrated descriptions in collaboration with the programming description language and its description environments.

1. はじめに

我々はドメインユーザ (以降，DU) を対象とした OOJ[1] という自然日本語 (NJ) をベースとした記述言語系とその記述環境を含むプログラム開発環境の研究を行っている．DU は OOJ と記述環境を用いることで，オブジェクト指向パラダイムに基づいて分析・設計・実装 (以降，“・”は「または，あるいは」の意味で用いる．) を行い，最終的にプログラミング言語 (Programming Language 以降，PL) 記述を得る事ができる．OOJ は現在，分析段階の言語 OONJ[2)6)]，設計段階の言語 ODDJ(Object-oriented Design Description Japanese) そして実装段階の言語 OEDJ[3] と OBF[4] で構成される．

OONJ とは DU が着目した対象世界について分析記述するための言語である．ODDJ とその記述環境では，OONJ を用いて作成した OONJ 記述を利用して最終的に PL 記述に変換するために必要な情報を対象世界から抽出する．さらに PL 記述に変換するために属性の型や初期値等を記述し，計算機で実行し結果を

† 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, Ibaraki University

得るための設計を行う．最後に下流の OEDJ・OBF では特定の PL 記述に変換するために ODDJ 記述から PL 記述に変換する．

OOJ では，設計段階の言語である ODDJ の研究が未着手であり，各段階における記述変換の処理が確立されていなかった．そこで本論文では，DU 向けに OONJ 記述を利用したオブジェクト指向設計記述言語 ODDJ とその記述環境を設計する．

ODDJ の狙いとしては，OONJ 記述の要素を ODDJ 記述の要素として利用することで DU の ODDJ 記述に要する負担を軽減し，対象世界を計算機で駆動させるために，ODDJ の段階で必要な計算機の情報を追加する．さらに下流の実装言語に対して，特定の PL 記述表現に偏らない情報を提供することで，最終的に DU の設計／実装 (以降, “ ／ ”は「かつ」の意味で用いる) にかかる負担を軽減することである．

2. 設計記述言語 ODDJ の要求分析

ODDJ は設計段階の記述言語であり，分析言語記述と実装言語記述の中間に位置し，対象世界を計算機で駆動させる設計を行うことが目的である．

上流の OONJ では，DU の対象世界を自然日本語と OOSF[5] を用いて記述する．自然日本語文をそのまま PL 記述に変換するためには日本語に制約を加える方法があるが，これは OONJ の性質に反するので行わない．そこで，ODDJ では，OOSF を用いて表現されたオブジェクトと振舞い，属性や相互関係といった要素を用いて，計算機におけるオブジェクトとその処理，変数と処理呼び出し等の要素へと変換／構築する．なお ODDJ における変数の処理には計算式を用いる．また ODDJ 記述の作成に足りない情報については DU に入力してもらうことを義務とする．

下流の実装記述言語では，特定のプログラミング言語に基づいた表現をするので，ODDJ では一般的な計算機の情報を追加する．

3. 設計記述言語 ODDJ の設計

3.1 OONJ 記述の利用

ODDJ では DU の設計にかかる負担を軽減するために，まず分析段階で作成した OONJ 記述を読み込む．OONJ 記述ではオブジェクトを表すフレームと属性や振舞いを表すスロット／サブスロットの集約階層構造を情報として保持しているので，これらの情報を計算機におけるオブジェクトとその属性と振舞いという要素として抽出する．

自然日本語を用いて作成した記述は，計算機を用いて PL 記述に一意に変換することは困難である．これは日本語の曖昧さを計算機で処理できないことによる．ODDJ では設計記述から PL 記述への自動変換を可能にするため，一意に PL 記述に変換できる情報のみを記述する．よって，PL 記述への自動変換が困難である自然日本語文で表現される要素は取り扱わない．

3.2 設計段階で追加する要素

ODDJ では対象世界を計算機で駆動させるための設計を行う．OONJ は計算機に関する情報を一切記述しないので，計算機の要素は ODDJ の段階で追加／記述する．例えば，OONJ では属性の名前が記述されている．ODDJ において計算機で扱う属性はデータ型／属性名などが記述する必要があるので，OONJ 記述から抽出した属性にデータ型を追加する．また，OONJ 記述から抽出した振舞い記述には振舞い名が記述されているので，ODDJ ではこの振舞い記述に戻り値型／仮引数などを追加する．

3.3 計算式の汎用表現

本論文では，汎用の指す意味を「特定のプログラミング言語に依存しない」と定義する．ODDJ で記述する計算式は汎用表現であり，オブジェクトの持つ属性に対する処理を記述する．汎用表現の計算式を記述することにより，それぞれの PL 記述に容易に自動変換することができる．

3.4 対象世界の駆動設計

ODDJ では対象世界を計算機で実行するための設計を行う．OONJ 記述では計算機の情報を扱わないため，計算機特有な記述を加える．ODDJ ではオブジェクトは自然に「動く」ことはないので,「駆動させる」必要がある．そこで，ODDJ では対象世界を計算機で駆動する情報として，初期値の設定に加えて，変数の初期化手順やプログラムの開始位置を指定するように設計する必要がある．

3.5 数値シミュレーションの初期値設定

計算機で対象世界を実行するためには，オブジェクトの持つ属性の初期値を設定する必要がある．OONJ ではこの仕組みとして初期状況記述と境界条件記述という特別なフレームを記述している．しかし，属性の型が宣言されていない事に加え，計算式の表現は DU の任意で行われるため，PL 記述に自動変換することは困難である．ODDJ では属性の型を宣言し，汎用表現の計算式を記述することによって，PL 記述に自動変換できるように設計する．

4. 設計記述言語 ODDJ の詳細

ODDJ の説明の参考にするために，**図 1** と**図 2** の水の大気循環の ODDJ 記述例を先に引用する．詳細な記述例の説明は 7 章で改めて行う．

4.1 ODDJ の構造化規則

ODDJ の構造化規則を**表 6**，**表 7**，**表 8**，**表 9** に示す．これらの構造化規則は OOSF を基盤とし，設計段階の ODDJ に適した形に設計した．OONJ を利用した DU は OOSF を用いる表現を知っているため，ODDJ でも同じ表現形式を取り入れた．表 6 は ODDJ

におけるフレームやスロット等の基本的な要素の集約階層構造の構造化規則を示している．表 7 は特定の目的を持ったフレームの定義である．表 8 は特定スロットの定義である．OONJ では，フレームやスロットの構造化規則が多く用意されている．これはスロットの持つ意味を視覚的に表現するためであるが，ODDJ では計算式と制御構文の組み合わせによる表現以外は不要なため，表 7 や表 8 のように OONJ の記述規則[5),6)]よりも規則が少なくなっている．表 9 は ODDJ における式とデータ型の定義である．**表 1** は各表における文法定義記号である．

表 1 ODDJ 構造記述規則文法定義記号
Table 1 ODDJ Symbols of structured rules

記号	定義内容	記号	定義内容
::=	左辺を右辺と定義する	( )	グループ化
	非終端記号	a?	a または空
“ a ”	終端記号	a+	a を 1 個以上繰返す
≪≫	終端記号	a*	a を 0 個以上繰返す
a | b	a または b	[a]	a は 0 個または 1 個

4.2 情報保存形式の策定

ODDJ では独自の構造化規則を定めている．しかし，これらの規則は表現のための規則である．OOJ では，情報を交換するための情報保存形式として XML[7)] を採用している．しかし，XML 自体は要素や属性の記述方法を定めていないので，語彙や要素構成を定義する仕組みとして DTD[7)] を採用する．

表 5 に ODDJ の DTD を示す．ODDJ では，計算機で駆動させるために必要な要素を下流の実装言語で一意特定できるように保存する．そのため，DTD は各要素の必要な情報を保持するように設計した．

フレームはその役割を一意に識別できるように設計した．またオブジェクトの振舞いにおいても，その役割を一意に識別できるように要素ごとに設計した．フレームの名前や相互関係を示す要素なども意味を識別できるように設計した．計算式はその構成要素である演算子や項の情報を持つように設計した．

4.3 ファセット記号

ODDJ 記述を構成する要素にファセット記号を割り当てる．ファセット記号を付加することで，要素の持つ意味の解釈をサポートする．ODDJ では，OONJ で用いられているファセット記号と区別するため，ファセット記号を dfn(Design Facet Number) とする．

4.4 ODDJ で取り扱う要素種類

ODDJ で扱う要素種類を**表 2** に示す．これらの要素は対象世界を計算機で実行するための設計記述に必要な要素であり，特定の PL 表現に依存しない．**表 3** は，OONJ の要素と対応する ODDJ の要素を比べた表である．表 2 及び表 3 において，“ – ”で表示されている要素は，他の言語には要素が存在するが，自身には存在しない事を表す．

表 2 ODDJ 要素種類表
Table 2 ODDJ elements kind list

dfn1 フレーム	**dfn3 処理**
dfn1.1 オブジェクト	dfn3.1 内部処理
dfn1.2 –	dfn3.2 メソッド呼び出し
dfn1.3 –	dfn3.3 メソッド総称文
dfn1.4 –	dfn3.4 if 文
dfn1.5 定数定義	dfn3.5 –
dfn1.6 ライブラリ	dfn3.6 switch 文
dfn1.7 初期設定	dfn3.7 while 文
dfn1.7.1 初期状況記述	dfn3.8 –
dfn1.7.2 境界条件記述	**dfn4 相互関連**
dfn1.8 駆動制御記述	**dfn5 定数定義**
dfn1.8.1 main	dfn5.1 定数定義
dfn1.8.2 駆動シナリオ	dfn5.2 単位定義
dfn2 属性	**dfn6 ライブラリ**
dfn2.1 参照属性	dfn6.1 ライブラリ記述
dfn2.2 実引数	**dfn7 初期設定**
dfn2.3 仮引数	dfn7.1 初期状況記述
dfn2.4 –	dfn7.2 境界条件記述
dfn2.5 局所変数	**dfn8 駆動制御**
dfn2.6 –	dfn8.1 main
dfn2.8 共有変数	dfn8.2 駆動シナリオ

表 3 OONJ と ODDJ における要素比較
Table 3 Comparisons of OONJ elements with ODDJ's

OONJ	ODDJ
オブジェクト (モノ)	オブジェクト
定数定義	定数定義
–	ライブラリ
初期状況記述	初期状況記述
境界条件記述	境界条件記述
シナリオ	–
対象世界スクリプチャ	–
–	駆動シナリオ
–	main フレーム
スロット	メソッド
共有属性	共有変数
スロット内属性	局所変数
振舞い付置属性	仮引数
mp 送信付置属性	実引数
mp 受信付置属性	仮引数

4.5 属性と振舞い記述

4.5.1 属性の記述

OONJ 記述で，属性は属性名のみが記述されている．これは対象世界に計算機の情報が一切存在しないからである．計算機で属性を扱う場合には，すべて変数となり，データとして扱われることになるのでデータ型や変数名，初期値等を記述する必要がある．下流の OEDJ や OBF が ODDJ 記述を PL 記述に自動変換するため，ODDJ では変数名に加え，変数のデータ型・初期値・アクセス制限修飾子を記述する．

4.5.2 振舞い記述

OONJ における振舞いは，ODDJ ではメソッドとして認識する．ODDJ におけるメソッドとは，スロット内に記述したデータに対する処理の纏まりと定義する．対象世界を計算機で駆動させるため，メソッドは

戻り値型を持ち，引数やアクセス制限等の情報も合わせて記述する．下流の実装言語ではメソッド記述から必要な情報を抽出し，特定の PL 記述を生成する．

4.5.3 変数とメソッドにおけるデータ型

表 4 は，ODDJ で記述する変数の型とメソッドの戻り値型を，OEDJ が PL 記述変換の対象としている Java，さらに OBF がプログラミング言語変換の対象としている Fortran90 とを比較した表である．下流の OEDJ と OBF では対応する型の名前が違うが，これはそれぞれの言語がプログラミング言語特有の表現をするためである．ODDJ では表 4 における用語を計算機のデータ型として定義する．

表 4 各言語における型の比較
Table 4 Comparisons of types in each language

	ODDJ	Java	Fotran90
変数	整数型	int	integer
	実数型	double	real
	文字列型	String	character(len = ?)
	文字型	char	character
	論理型	boolean	logical
メソッド戻り値型	void	void	–
	整数型	int	integer
	実数型	double	real
	文字列型	String	character(len = ?)
	文字型	char	character
	論理型	boolean	logical

4.5.4 変数とメソッドにおけるアクセス制限

ODDJ では変数・メソッドのアクセス制限修飾子は，他のフレームから直接参照可能である“ 共有 ”と，他のフレームからは参照することができない“ 私有 ”のどちらかを記述する．

4.6 計 算 式

下流の OEDJ や OBF では，ODDJ 記述を PL 記述に自動変換することを視野に入れているため，計算式においても PL 記述に一意に変換可能でなくてはならない．自動変換を行うため，式の表現には制約が必要である．計算式には制約が必要であり，表 9 に示したとおりである．計算式を用いることで，データに対する演算や制御構文の条件式を表現できる．また，演算子は表 9 の (102) から (106) に示す記号のみとする．これらの定義については，PL 記述を作成する際に必要である記号のみに注目し，定義した．

4.7 制 御 構 文

ODDJ では計算式に加え，制御構文を用いてデータの処理を記述する．下流の実装言語記述環境では，ODDJ 記述をある程度自動変換するため制御構文もその表現に制限がなくてはならない．制御構文として条件分岐のために if 文と switch 文を，反復処理のために while 文を用意した．ODDJ では制御構文をこれら 3 つに制限した．これらの制御構文における条件分岐判定文や反復条件判定文といった計算式は，DTD に従い保存し下流に提供する．

4.8 定 数 定 義

定数定義とは，ODDJ 記述内で使用する定数または単位を，一括して定義するためのフレームである．構造化規則は表 7 における (33) に示す通りである．変数名に対しては，その値を示す計算式を記述する．複雑な式を単位／定数として纏め，ODDJ 記述内で共通に使用することができるので，記述作成の際に DU の負担を軽減する効果がある．

4.9 初期状況記述・境界条件記述

初期状況記述は各フレーム内に存在する共有変数の初期値を記述する．境界条件記述は対象世界における境界条件の設定を行うための記述である．OONJ の段階でこれら 2 つの記述は作成するが，これらの表現は計算機で動作する際に必ずしも最適な表現であるとは言えない．そのため，対象世界を計算機で動作させるために必要な変数の初期値を，計算機で実行することを想定した値に設定／記述する必要がある．図 2 における 23 番フレームは初期状況記述である．このように各変数を値で初期化するための式を記述する．

5. 駆動制御設計

駆動制御設計とは，ODDJ を用いて構成する対象世界を計算機で実行させるために必要な記述の総称である．ODDJ 記述は PL 記述に自動変換するため，汎用表現で駆動制御設計を行う．駆動制御記述では，各々のフレームに対して初期状況記述や境界条件記述を適用する手順を指定し，計算機で実行するための開始スロットを指定する．駆動制御のために，初期状況記述と境界条件記述に加え，main フレームと駆動シナリオが必要である．

5.1 main フレーム

main フレームは PL 記述に変換された際にプログラム処理の開始部分に相当する．図 2 に示した 28 番フレームが main フレームである．main フレームの役割は駆動シナリオフレームのメソッドを起動することである．

5.2 駆動シナリオ

駆動シナリオは，対象世界の変数初期化や振舞いの開始位置を指定などの駆動制御をするためのフレームである．図 2 に示した 27 番フレームがこの記述に相当する．このフレーム内の 4 番目のスロットでは初期状況記述／境界条件記述を適用し，実行結果の出力／対象世界の処理開始の制御を行っている．同じフレーム内の 5 番目のスロットでは，ODDJ 記述が処理を開始するためのスロットの指定と引数の制御を行っている．駆動シナリオの役割は対象世界と計算機における処理を連携することである．

1 dfn1.1 太陽

1	dfn2.8 放射熱量Q r	実数型 私有
2	dfn2.8 現在の時刻 t	整数型 私有
3	dfn2.8 円周率 PI	実数型 私有
4	dfn3.3 熱放射する (整数型 t)	void 共有
	dfn3.4 | if $7 \leq t$	
	dfn3.4 | then	
	dfn3.4 | if $t \leq 17$	
	dfn3.4 | then	
	dfn3.1 | Qr = $\sin(PI * (t-6)/12)$	
	dfn3.4 | else	
	dfn3.1 | Q r =0	
	dfn3.2 | 熱吸収する (Q r *1/3)	
	≫ 5:海上空大気 (高度 0)[8]	
	...	

2 dfn1.1 海

1	dfn2.8 面積S	実数型 私有
2	dfn2.8 比熱C	実数型 私有
3	dfn2.8 表面温度T s	実数型 私有
4	dfn2.8 表面質量M s	実数型 私有
5	dfn2.8 放射熱量Q r	実数型 私有
6	dfn2.8 吸収熱量Q a	実数型 私有
7	dfn2.8 保有熱量Q	実数型 私有
8	dfn3.3 熱吸収する（実数型 Q a）	void 共有
	dfn3.1 | Q=Q+Q a*S	
	dfn3.2 | 温度が上昇する	≫ [9]
9	dfn3.3 温度が上昇する	void 私有
	dfn3.1 | T s=T s+Q/(M s*C)	
	dfn3.2 | 蒸発する	≫ [10]
10	dfn3.3 蒸発する	void 私有
	dfn2.5 | 温度 (高度 0) T z 実数型	
	dfn2.5 | 蒸発量 e 実数型	
	dfn3.2 | 温度を取得する	
	≫ 5:海上空大気 (高度 0)[7]	
	dfn3.1 | T z = 結果	
	dfn3.1 | e=1.155*(T s-T z)	
	dfn3.2 | 水を受ける (e)	
	≫ 8:海上空雲 (高度 1)[3]	
	dfn3.2 | 熱放射する	≫ [11]
11	dfn3.3 熱放射する	void 私有
	dfn3.1 | Q r=$\epsilon * \sigma$ *(T s+0.948)^4	
	dfn3.2 | 海から熱吸収する (Q r)	
	≫ 5:海上空大気 (高度 0)[2]	
	dfn3.2 | 温度が低下する	≫ [12]
12	dfn3.3 温度が低下する	void 私有
	dfn3.1 | T s=T s-Q r/(M s*C)	
13	dfn3.3 海へ流出する	void 私有
14	dfn3.3 雨/雪を受ける（実数型 F a）	void 私有
15	dfn3.3 大気から熱吸収する（実数型 Q a）	void 共有
	dfn3.1 | Q=Q a*S	
16	dfn3.3 表面温度を設定する（実数型 temp）	void 共有
	dfn3.1 | Ts = temp	
17	dfn3.3 保有熱量を設定する（実数型 temp）	void 共有
	dfn3.1 | Q = temp	
18	dfn3.3 比熱を設定する（実数型 temp）	void 共有
	dfn3.1 | C = temp	
	...	

図 1 水の大気循環 オブジェクト
Fig. 1 ODDJ example of water flow objects

23 dfn1.7.1 初期状況記述

1	dfn3.3 初期状況記述を適用する	void 共有
	dfn3.2 海の構成を設定する	≫ [2]
	dfn3.2 川の構成を設定する	≫ [3]
	dfn3.2 大地の構成を設定する	≫ [4]
	dfn3.2 海上空大気 (高度 0) の構成を設定する	≫ [5]
	dfn3.2 川上空大気 (高度 0) の構成を設定する	≫ [6]
	dfn3.2 大地上空大気 (高度 0) の構成を設定する	≫ [7]
	...	
2	dfn3.3 海の構成を設定する	void 私有
	dfn3.1 | 表面温度T s =1.2	
	dfn3.1 | 保有熱量Q=0	
	dfn3.2 | 表面温度を設定する	≫ 2:海 [16]
	dfn2.2 | 表面温度 Ts	
	dfn3.2 | 保有熱量を設定する	≫ 2:海 [17]
	dfn2.2 | 保有熱量 Q	
3	dfn3.3 川の構成を設定する	void 私有
	dfn3.1 | 表面温度T s = 1.2	
	dfn3.1 | 質量M=20	
	dfn3.1 | 保有熱量Q=0	
	dfn3.1 | 流入量F i =1	
	dfn3.1 | 流出量F o =1	
	dfn3.2 | 表面温度を設定する	≫ 3:川 [11]
	dfn2.2 | 表面温度 Ts	
	dfn3.2 | 質量を設定する	≫ 3:川 [12]
	dfn2.2 | 質量 M	
	...	

28 dfn1.8.1 main

1	dfn3.3 プログラム駆動開始	void 共有
	dfn3.2 | 駆動制御を行う	≫ 27:駆動シナリオ [4]

27 dfn1.8.2 駆動シナリオ

1	dfn2.8 計算ステップ step	整数型 私有
2	dfn2.8 最大計算回数 max	整数型 私有
3	dfn2.8 現時刻 t	実数型 私有
4	dfn3.3 駆動制御を行う	void 共有
	dfn3.2 | 初期状況記述を適用する	
	≫ 24:初期状況記述 [1]	
	dfn3.2 | 境界条件記述を適用する	
	≫ 25:境界条件記述 [1]	
	dfn3.7 | while step < max	
	dfn3.4 | if step % 10 == 0	
	dfn3.4 | then	
	dfn3.2 | 計算結果を出力する	≫ [6]
	dfn3.2 | 駆動シナリオを実行する	≫ [5]
	dfn3.1 | step = step + 1	
5	dfn3.3 駆動シナリオを実行する ()	void 私有
	dfn3.1 | t = step % 24	
	dfn3.4 | if t ==24	
	dfn3.4 | then	
	dfn3.1 | t =0	
	dfn3.2 | 熱放射する (t)	≫ 1:太陽 [3]
	dfn3.2 | 川へ流出する	≫ 3:大地 [13]
	dfn3.1 | t = t + d t	
	...	

図 2 水の大気循環 初期状況記述
Fig. 2 ODDJ example of water flow objects

6. ODDJ 記述環境

6.1 設 計 方 針

ODDJ の記述環境として以下の方針を設ける．

共通のデータフォーマット

OOJ 共通のデータフォーマットとして XML を採用している．OOJ の各言語記述環境では XML で表現された各言語記述から必要な情報を抽出し，利用する．ODDJ の記述環境では，XML で表現された OONJ 記述から情報を利用するため，XML を読み取ることができ，かつ表 5 に示した DTD で構成する XML 形式で情報を保存する．

分かりやすいインターフェース

ODDJ の構造化規則は表 6〜9 に示した通り，合計で 106 個である．これらをすべて覚えて ODDJ 記述を作成するのでは，DU の負担が大きい．複雑な構造化規則をある程度知らなくても，ODDJ 記述を作成できるようなインターフェースが望ましい．このような構造化規則は ODDJ の記述環境でなるべく吸収し，DU の負担軽減を図る．

6.2 ODDJ エディタ

ODDJ 記述環境として ODDJ エディタを開発した．

ODDJ エディタの機能概要

ODDJ エディタは表 6〜9 の構造化規則に従い ODDJ 記述の表示を行う．DU は任意の要素に対し，情報の追加・削除・変更を行うことができる．ODDJ エディタは DU の要求を受けて，DTD に従い XML 形式の ODDJ 記述をファイルに保存する．ODDJ エディタの機能は，以下である．

(1) OONJ 記述 (XML 形式) の読み込み
(2) ODDJ 記述の (一部) 自動生成
(3) フレームの追加・削除・編集
(4) スロットの追加・削除・編集
(5) サブスロットの追加・削除・編集
(6) 初期状況記述作成
(7) 境界条件記述作成
(8) 駆動制御記述の作成
(9) ODDJ 記述 (XML 形式) の読み込み・保存
(10) 計算式の構造化

画面の説明

図 3 にその実行画面を示す．画面左の枠は，対象世界のオブジェクト一覧を表示する．DU はこれらのオブジェクトのうち，編集したいオブジェクトを選択し，メソッドや変数を追加する．画面中央背面の枠は，DU が作成した ODDJ 記述を表示する領域である．画面右下のウィンドウは，スロット編集のためのダイアログである．DU は追加・削除・変更等のメニューを選んで，必要な要素や情報を入力する．

スロット編集方法

図 3 における画面右下のウィンドウを見てわかるとおり，DU は追加したい文をメニューから選択する．ODDJ エディタでは DU の要求に対して対話形式で情報を入力してもらうことにより要素の追加を行うことができる．このように ODDJ エディタから要素の構成に必要な情報を DU に要求することにより，構造化規則を極力吸収している．

ODDJ 記述の自動生成

ODDJ エディタでは，OONJ 記述を読み込み ODDJ 記述を自動生成する．OONJ 記述から要素を抽出する際の，要素の変換規則は表 3 に示した．

7. ODDJ 記述例とその比較

7.1 ODDJ 記述例の概要

水の大気循環の“ 海 ”と“ 雲 ”フレームは図 1 に示した．また，ODDJ 記述の処理開始位置を示す main フレームと対象世界の初期化と駆動制御を示した駆動シナリオ，初期状況記述を図 2 に示した．これらの 5 つのフレームは完成した ODDJ 記述の一部である．この記述例の作成においては，ODDJ 記述のための記述環境を用いず手作業で変換／記述作成を行った．

7.2 PL 記述への変換と比較

水の大気循環の ODDJ 記述を元に，手作業で Java プログラムに変換した．その際，変数の値を出力するために Java 特有の情報を追加する必要があったが，そのほかの要素については Java 言語に対応する要素に変換できた．また，Java プログラムを実行し，妥当な計算結果を得ることができた．水の大気循環の記述例は Java プログラムにのみ変換しただけだが，ODDJ 記述から変換する際に追加が必要な情報は，変数の出力に関する記述のみであった．このことから，Java プログラムへの記述変換のための設計がある程度達成できていることがいえる．

7.3 OONJ 記述との比較

水の大気循環の OONJ 記述[2] と対応する ODDJ 記述である図 1 を比較すると，変数においては PL 記述変換に必要な変数の型，アクセス制限修飾子が追加されていることが分かる．さらに，メソッドに必要な引数や戻り値型，アクセス制限修飾子を追加している．また OONJ 記述において自然日本語文で表現されている文が，ODDJ 記述では計算式で表現されている．また，図 2 に示した駆動制御記述フレームは，変数・計算式・制御構文などで表現している．これは ODDJ に特有な (特徴的な) フレームである．

8. ODDJ エディタの評価

ODDJ エディタを利用して，水の大気循環の OONJ 記述を読み込んだ．ODDJ エディタでは OONJ 記述から ODDJ 記述の要素として利用可能な情報を抽出し，記述の雛型を作成する．作成した ODDJ 記述に存在するオブジェクトは，図 3 に示した画面左の枠に

表 5 ODDJ における DTD スキーマ
Table 5 DTD Schema in ODDJ

ルート要素

```
<!ELEMENT doc (name,type,frame-obj+,frame-init,frame-boundary,
```

オブジェクト

```
frame-const*,frame-scenario,frame-main,frame-library*)>
<!-- クラス dfn1.1 -->
<!ELEMENT frame-obj (name,p-name,slot-attr*, slot-method+)>
<!ATTLIST frame-obj id CDATA #REQUIRED>
<!-- 定数定義 dfn1.5 -->
<!ELEMENT frame-const (name,slot-attr*,slot-method+)>
<!ATTLIST frame-const id CDATA #REQUIRED>
<!-- ライブラリフレーム dfn1.6 -->
<!ELEMENT frame-library (name,p-name,slot-attr*,slot-method+)>
<!ATTLIST frame-library id CDATA #REQUIRED>
<!-- 初期状況記述 dfn1.7.1 -->
<!ELEMENT frame-init (name,p-name,slot-attr*,slot-method+)>
<!ATTLIST frame-init id CDATA #REQUIRED>
<!-- 境界条件記述 dfn1.7.2 -->
<!ELEMENT frame-boundary (name,p-name,slot-attr*,slot-method+)>
<!ATTLIST frame-boundary id CDATA #REQUIRED>
<!-- main dfn1.8.1 -->
<!ELEMENT frame-main (name,p-name,slot-attr*,slot-method+)>
<!ATTLIST frame-main id CDATA #REQUIRED>
<!-- 駆動シナリオ dfn1.8.2 -->
<!ELEMENT frame-scenario (name,p-name,slot-attr*,slot-method+)>
<!ATTLIST frame-scenario id CDATA #REQUIRED>
```

メソッド

```
<!-- メソッド呼び出し文 dfn3.2 -->
<!ELEMENT slot-mp (class-rels,method-rels, actual-argument*)>
<!ATTLIST slot-mp id CDATA #REQUIRED>
<!ATTLIST slot-mp depth CDATA #IMPLIED>
<!-- メソッドスロット dfn3.3 -->
<!ELEMENT slot-method (name,p-name,type,access,dummy-argument*,
(slot-switch | slot-equation | slot-if | slot-while | slot-mp |
 slot-attr | slot-return)*)>
<!ATTLIST slot-method id CDATA #REQUIRED>
<!-- 属性スロット dfn2.x -->
<!ELEMENT slot-attr (variable)>
<!ATTLIST slot-attr id CDATA #REQUIRED>
<!-- if 文スロット dfn3.4 -->
<!ELEMENT slot-if (expression, then,else?)>
<!ATTLIST slot-if id CDATA #REQUIRED>
<!ATTLIST slot-if depth CDATA #IMPLIED>
<!-- then 文 dfn3.4 -->
<!ELEMENT then (slot-switch | slot-equation | slot-if |
 slot-while | slot-mp | slot-attr)*>
<!ATTLIST then depth CDATA #IMPLIED>
<!-- else 文 dfn3.4 -->
<!ELEMENT else (slot-switch | slot-equation | slot-if |
 slot-while | slot-mp | slot-attr)*>
<!ATTLIST else depth CDATA #IMPLIED>
<!-- 式スロット dfn3.1 -->
<!ELEMENT slot-equation (calc)>
<!ATTLIST slot-equation id CDATA #REQUIRED>
<!ATTLIST slot-equation depth CDATA #IMPLIED>
<!-- switch 文 dfn3.6 -->
<!ELEMENT slot-switch (expression, case+, default?)>
<!ATTLIST slot-switch id CDATA #REQUIRED>
<!ATTLIST slot-switch depth CDATA #IMPLIED>
<!-- case 文 dfn3.6 -->
<!ELEMENT case ((case-char | case-number), (slot-switch |
 slot-equation | slot-if | slot-while | slot-mp | slot-attr)*)>
<!ATTLIST case id CDATA #REQUIRED>
<!ATTLIST case depth CDATA #IMPLIED>
<!-- default 文 dfn3.6 -->
<!ELEMENT default (slot-switch | slot-equation | slot-if |
 slot-while | slot-mp | slot-attr)*>
<!ATTLIST default id CDATA #REQUIRED>
<!ATTLIST default depth CDATA #IMPLIED>
<!-- while 文 dfn3.7 -->
<!ELEMENT slot-while (expression, (slot-switch | slot-equation |
 slot-if | slot-while | slot-mp | slot-attr)*)>
<!ATTLIST slot-while id CDATA #REQUIRED>
<!ATTLIST slot-while depth CDATA #IMPLIED>
<!-- return 文 dfn3.1 -->
<!ELEMENT slot-return (name | expression)>
<!ATTLIST slot-return id CDATA #REQUIRED>
<!ATTLIST slot-return depth CDATA #IMPLIED>
```

一般要素

```
<!-- 数字 -->
<!ELEMENT case-number (#PCDATA)>
<!-- 文字値 -->
<!ELEMENT case-char (#PCDATA)>
<!-- 仮引数 -->
<!ELEMENT dummy-argument (name,type,comment?)>
<!-- 実引数 -->
<!ELEMENT actual-argument ((expression | name),comment?)>
<!-- 相互関係先 -->
<!ELEMENT class-rels (#PCDATA)>
<!ATTLIST class-rels id CDATA #REQUIRED>
<!-- 相互関係先 CDATA -->
<!ELEMENT method-rels (#PCDATA)>
<!ATTLIST method-rels id CDATA #REQUIRED>
<!-- コメント -->
<!ELEMENT variable (name,access,type,comment)>
<!ATTLIST variable belong CDATA #IMPLIED>
<!-- 名前 -->
<!ELEMENT name (#PCDATA)>
<!-- プログラム変換名 -->
<!ELEMENT p-name (#PCDATA)>
<!-- データ型 -->
<!ELEMENT type (#PCDATA)>
<!-- アクセス制限修飾子 -->
<!ELEMENT access (#PCDATA)>
<!-- コメント -->
<!ELEMENT comment (#PCDATA)>
<!-- 所属 -->
<!ELEMENT belong (#PCDATA)>
<!-- class 指定タグ -->
<!ELEMENT class (#PCDATA)>
```

計算式

```
<!-- 式 -->
<!ELEMENT calc (target?,expression,nj)>
<!-- 左辺 -->
<!ELEMENT target (term)>
<!-- 右辺または式 -->
<!ELEMENT expression (section,section?,nj?)>
<!ATTLIST expression operator CDATA #REQUIRED>
<!-- 項 -->
<!ELEMENT section (term | method | expression)>
<!-- 変数項 -->
<!ELEMENT term (double | int | char | string | boolean)>
<!-- 自然日本語文 -->
<!ELEMENT nj (#PCDATA)>
<!-- 関数名 -->
<!ELEMENT method (class-rels,method-rels,term*)>
```

データ型

```
<!-- 実数 -->
<!ELEMENT double (#PCDATA)>
<!ATTLIST double ary CDATA #IMPLIED>
<!-- 整数 -->
<!ELEMENT int (#PCDATA)>
<!ATTLIST int ary CDATA #IMPLIED>
<!-- 文字列 -->
<!ELEMENT string (#PCDATA)>
<!ATTLIST string ary CDATA #IMPLIED>
<!-- 文字 -->
<!ELEMENT char (#PCDATA)>
<!ATTLIST char ary CDATA #IMPLIED>
<!-- 論理値 -->
<!ELEMENT boolean (#PCDATA)>
<!ATTLIST boolean ary CDATA #IMPLIED>
```

図 3 ODDJ エディター実行画面
Fig. 3 ODDJ editor GUI window in execution

一覧が表示される．DU はオブジェクトを選択することで，画面中央の枠に構造化規則に従った ODDJ 記述が表示される．またオブジェクトに対して変数・メソッドの追加・削除・変更を行うことができるため，ODDJ 記述の構造と内容を ODDJ エディタで確認しながら編集を行うことができる．また，スロット内の編集方法は専用の編集ウィンドウを用いて行うが，図 3 に示すように要素を追加する際には，どのような文を入力するか選択するだけでよいことがわかる．

現在，ODDJ エディタは開発途中であるが，上記の手順で ODDJ 記述を作成することができる．ODDJ エディタは DU の要求に対して，要素の構成に必要な情報を要求することで対話形式で処理を進める．この機能により ODDJ の構造化規則を極力吸収している．

9. 今後の予定

現在，一貫した記述は水の大気循環のみである．そのため，ODDJ 記述をいくつか作成し PL 記述に変換することで，ODDJ 記述から PL 記述に変換するために必要な情報が十分であるか検証する必要がある．よって今後は水の大気循環において OBF への変換を行っていくと共に，いくつか一貫記述例を作成し，下流の実装言語記述環境と連携して検証を行っていく．

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，第 150 回 SE 研究会報告, 2005-OOSE-150，pp.23-26，(2005)．
2) 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，第 150 回 SE 研究会報告, 2005-OOSE-150, pp.23-26，(2005)．
3) 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，第 150 回 SE 研究会報告, 2005-OOSE-150, pp.23-26，(2005)．
4) 齋藤正樹，畠山正行，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，第 150 回 SE 研究会報告, 2005-OOSE-150，pp.23-26, (2005)．
5) 畠山正行，オブジェクト指向自然日本語構造化フレーム OOSF の設計と表現技法，シミュレーション学会誌，**22**-4，195/209, Dec., (2004)．
6) 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 の設計とその記述環境, 情報処理学会第 102 回 HPC 研究会報告, 2005-HPC-102，13/22，(2005)．
7) http://www.w3.org/TR/2004/REC-xml-20040204/

表 6 ODDJ 構造記述規則 (1/4) 一般/階層構造記述規則, 対象世界共通要素記述規則
Table 6 ODDJ structured description rules (1/4)

(1) **ODDJ 記述**	::=	フレーム+ 定数定義* 初期状況記述 境界条件記述 main 駆動シナリオ ライブラリ*
(2) **フレーム**	::=	『フレームヘッダ 【スロット +】』
(3) フレームヘッダ	::=	「 フレーム番号 sp “ dfn1.1 ” sp フレーム名 」
(4) フレーム番号	::=	≪ ODDJ 記述内一連番号≫
(5) フレーム名	::=	NJ
(6) **スロット**	::=	メソッドスロット	変数スロット
(7) **メソッドスロット**	::=	「スロット総称文」戻り値型 アクセス制約修飾子 lf (mr サブスロット)+
(8) スロット総称文	::=	「 スロット番号 sp “ dfn3.3 ” sp NJ [“ (” 仮引数* “) ”]
(9) スロット番号	::=	≪フレーム内一連番号≫
(10) **変数スロット**	::=	「 スロット番号 sp “ dfn2.8 ” sp 変数部 データ型 “ 私有 ” 」
(11) **変数サブスロット**	::=	「 行番号 sp “ dfn2.5 ” sp 変数部 データ型 」
(12) 変数部	::=	(変数名)
(13) **サブスロット**	::=	サブスロット記号 (if 文 | swicth 文 | while 文 | 式 | メソッド呼び出し文 | 変数サブスロット | return 文)*
(14) サブスロット記号	::=	sp 行番号 sp ファセット記号 mr2 sp 階層表記線
(15) 階層表記線	::=	“	”
(16) ファセット記号	::=	“ dfn ”≪ファセット番号≫
(17) 仮引数	::=	データ型 変数
(18) 実引数	::=	変数名
(19) アクセス制約修飾子	::=	“ 私有 ”	“ 共有 ”
(20) 戻り値型	::=	データ型	“ void ”
(21) 変数	::=	変数名 配列要素?
(22) 変数名	::=	≪属性名≫	≪一時変数名≫	≪仮引数名≫
(23) 配列要素	::=	“ [” 要素番号 “] ”
(24) 要素番号	::=	算術式 //算術式のデータ型は整数型
(25) 行番号	::=	- ≪行番号 (スロット内一連番号) ≫
(26) lf	::=	≪改行して次の行の先頭へ戻る≫
(27) rt	::=	≪右詰め≫
(28) sp	::=	≪任意一定幅空白≫
(29) 『』	::=	≪フレームを構成し，内部に記述を収める≫
(30) 【】	::=	≪スロットを構成し，内部に記述を収める≫
(31) 「」	::=	≪行を構成し，内部に記述を収める≫
(32) mr	::=	≪直上の行に 階層表記線 があれば, その位置まで右に移動する. ≫

表 7 ODDJ 構造記述規則 (2/4) 特定フレーム構造記述規則
Table 7 ODDJ structured description rules (2/4)

(33) **定数定義**	::=	『定数定義フレームヘッダ 【定義スロット +】』
(34) 定数定義フレームヘッダ	::=	「 フレーム番号 sp “ dfn1.5 ” sp フレーム名 」
(35) 定数定義スロット	::=	「定数定義スロット総称文」 戻り値型 アクセス制約修飾子 lf (mr 定数定義式)+
(36) 定数定義式	::=	「 変数名 “ = ”式 」
(37) **ライブラリ**	::=	『ライブラリフレームヘッダ 【スロット +】』
(38) ライブラリフレームヘッダ	::=	「 フレーム番号 sp “ dfn1.6 ” sp フレーム名 」
(39) **初期状況記述**	::=	『初期状況フレームヘッダ 【初期設定起動スロット】 【初期設定スロット +】』
(40) 初期状況フレームヘッダ	::=	「 フレーム番号 sp “ dfn1.7.1 ” sp フレーム名 」
(41) 初期設定起動スロット	::=	「初期設定スロット総称文」 戻り値 アクセス制約修飾子 lf (mr サブスロット)+
(42) 初期設定スロット	::=	「定数定義スロット+」 「メソッド呼び出し+」
(43) **境界条件記述**	::=	『境界条件フレームヘッダ 【初期設定起動スロット】 【初期設定スロット +】』
(44) 境界条件フレームヘッダ	::=	「 フレーム番号 sp “ dfn1.7.2 ” sp フレーム名 」
(45) **main**	::=	『main フレームヘッダ 駆動開始スロット 』
(46) main フレームヘッダ	::=	「 フレーム番号 sp “ dfn1.8.1 ” sp フレーム名 」
(47) 駆動開始スロット	::=	駆動制御総称文 駆動対象起動文
(48) 駆動制御総称文	::=	「スロット番号 sp “ プログラム稼動開始 ”rt “ void ” “ 共有 ”」
(49) 駆動対象起動文	::=	サブスロット記号 “ 駆動制御を行う ” rt “ ≫メソッド呼び出し≫駆動シナリオ ” スロット番号
(50) **駆動シナリオ**	::=	『駆動シナリオフレームヘッダ 【スロット +】』
(51) 駆動シナリオフレームヘッダ	::=	「 フレーム番号 sp “ dfn1.8.2 ” sp “ 駆動シナリオ ” 」

表 8 ODDJ 構造記述規則 (3/4) 特定スロット構造記述規則
Table 8 ODDJ structured description rules (3/4)

(52) **メソッド呼び出し文**	::=	「(外部メソッド呼び出し | 内部メソッド呼び出し)」
		lf (「サブスロット記号 mr 実引数 [“ , ” 実引数]* 」)
(53) 外部メソッド呼び出し	::=	相手メソッド名 “ ≫メソッド呼び出し≫ ”フレーム名 [対象スロット番号]
(54) 内部メソッド呼び出し	::=	相手メソッド名 “ ≫メソッド呼び出し≫ ”[対象スロット番号]
(55) 対象スロット番号	::=	“ [” スロット一意特定識別子 [“ - ” 行番号] “] ”
(56) 相手メソッド名	::=	NJ
(57) **if 文**	::=	“ if ” sp if 条件文 lf “ if 文様式 ”分岐構造
(58) if 条件文	::=	“ (” 論理式 “) ”
(59) “ if 文様式 ”分岐構造	::=	then [lf else]
(60) then	::=	サブスロット記号 “ then ” (lf サブスロット)*
(61) else	::=	サブスロット記号 “ else ” (lf サブスロット)*
(62) **switch 文**	::=	“ switch ” sp switch 条件文 lf “ switch 文様式 ”分岐構造
(63) switch 条件文	::=	“ (” 式 “) ” //ただし式のデータ型は整数型か文字型である
(64) “ switch 文様式 ”分岐構造	::=	サブスロット記号 “ case ” sp case 条件文 “ : ”
		(lf サブスロット)+ [lf break])* [lf deafult]
(65) case 条件文	::=	整数値	文字値
(66) break	::=	サブスロット記号 “ break ”
(67) default	::=	サブスロット記号 “ default: ” (lf サブスロット形式記号)* [lf break]
(68) **while 文**	::=	サブスロット記号 “ while ” sp while 条件文 lf “ while 文様式 ”構造
(69) while 条件文	::=	“ (” 論理式 “) ”
(70) ”while 文様式”構造	::=	サブスロット記号 (lf サブスロット)+
(71) **return 文**	::=	サブスロット記号 sp “ return ” 戻り値
(72) 戻り値	::=	変数名

表 9 ODDJ 構造記述規則 (4/4) 式およびデータ型
Table 9 ODDJ structured description rules (4/4)

(73) 式	::=	算術式	論理式	関係式	文字式
(74) 算術式	::=	算術二項式	算術単項式	基本項
(75) 算術二項式	::=	項 算術二項演算子 項
(76) 算術単項式	::=	算術単項式 項
(77) 論理式	::=	論理二項式	論理単項式	基本項
(78) 論理二項式	::=	項 論理二項演算子 項
(79) 論理単項式	::=	論理単項演算子 項
(80) 関係式	::=	項 関係演算子 項
(81) 文字式	::=	文字二項式	基本項
(82) 文字二項式	::=	項 文字演算子 項
(83) 項	::=	基本項	“ (” 式 “) ”
(84) 基本項	::=	変数	リテラル
(85) リテラル	::=	数値	論理値	文字値
(86) NJ	::=	≪自然日本語文. ≫
(87) データ型	::=	基本データ型
(88) 基本データ型	::=	算術型	論理型	文字型
(89) 算術型	::=	“ 整数型 ”| “ 実数型 ”
(90) 整数型	::=	“ 整数型 ”
(91) 実数型	::=	“ 実数型 ”
(92) 論理型	::=	“ 論理型 ”
(93) 文字型	::=	“ 文字型 ”
(94) 数値	::=	整数値	実数値
(95) 整数値	::=	数字+
(96) 実数値	::=	数字+ “ . ” 数字+
(97) 論理値	::=	“ 真 ”	“ 偽 ”
(98) 数字	::=	“ 0 ”| “ 1 ”| “ 2 ”| “ 3 ”| “ 4 ”|
		“ 5 ”| “ 6 ”| “ 7 ”| “ 8 ”|“ 9 ”
(99) 文字値	::=	“ ’ ”≪英数字≫“ ’ ”
(100) 文字値	::=	“ ’ ”≪英数字≫“ ’ ”
(101) 注釈	::=	“ // ”≪任意の文字列≫
(102) 算術演算単項子	::=	“ - ”
(103) 算術二項演算子	::=	“ * ”	“ / ”	“ ＋ ”	“ - ”	“ ^ ”
(104) 関係演算子	::=	“ < ”|“ <= ”|“ > ”|“ >= ”|
	::=	“ == ”|“ != ”
(105) 論理単項演算子	::=	“ not ”
(106) 論理二項演算子	::=	“ and ”	“ or ”