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

2008－SE－159 (8)
2008／3／17

オブジェクト指向設計記述言語ODDJの汎化階層構成への拡張とその実装

片野 克紀 † 池田 陽祐 †† 畠山 正行 ††

オブジェクト指向 (OO) 設計記述言語 ODDJ は，分析記述言語 OONJ と実装記述言語 OPDJ と連携して分析から実装までのモデリング/プログラミング過程を一貫した OO 記述で実現し，プログラム生成までの大幅な負担減少を狙う「記述言語系」の一端を担う言語である．従来の ODDJ の仕様や記述を支援するためのエディタの機能および特性には幾つかの問題点が残っていた．本論文では，言語仕様に汎化階層構成 (いわゆる，クラス階層) を導入し，ODDJ をエディタで記述支援するために定義した XML で書かれた DTD の仕様の改訂を行うと共に，ODDJ 自体の仕様 (構造化記述規則) の拡張を行った．その結果，OONJ や OPDJ との連携機能が強化され，エディタの機能の改善，汎化階層の概念もエディタで使用可能になった．

Extentions and implementations of generalization hierarchy constitutions on Object-oriented Design Description Japanese ODDJ

Katsunori Katano ,† Yousuke Ikeda †† and Masayuki Hatakeyama ††

The Object-oriented Design Description Japanese ODDJ is one of the description language series that are constituted with the analysis description language OONJ, ODDJ and the implementation description language OPDJ. The ODDJ cooperates with the OONJ and the OPDJ, this series of description languages realize an integrally consistent OO descriptions through the modeling/programming processes from the analysis stage up to the implementation stage. These processes are to be realized some significant reduction of the program development load of the users. Some problems have been left in the existing ODDJ language specifications and in the description support editor. In the present paper, we have introduced the generalization hierarchy constitutions in the language specifications that are so called the class hierarchy, and we have revised the DTD specifications based on the XML for the ODDJ description support by the editor in the computer system. We have also extended the specifications of the ODDJ itself, especially of the structured description rules. As the results, the cooperation functions with the OONJ and the OPDJ have been upgraded, the functions of editor has been improved, and the concept of the generalization hierarchy have been applied and realized in the ODDJ editor.

1. はじめに

オブジェクト指向設計記述言語 ODDJ[1] は我々の研究グループで開発してきた記述言語系 OOJ[2] の中の，設計段階に当たる記述言語である．この記述言語系はドメインユーザ (以降，DU と略す) といわれる非情報系の特定分野の専門家で，プログラムを作成し，シミュレーション等を実行することをそのドメインのプロフェッショナルの仕事の一部として利用する人達のために構築してきた．

この記述言語系は分析段階の OONJ[3][4]，設計段階の本 ODDJ，実装段階の OPDJ[5] がある並行して開発されてきている．しかし，ODDJ には幾つかの問題点の解決が持ち越されてきている．それは以下のようなものである．

(1) 開発の上流工程にある分析段階記述言語 OONJ と，開発の下流工程にある実装段階 OPDJ との「連携」が必ずしも十分ではなく，OONJ から ODDJ への変換，また ODDJ から OPDJ への変換が必ずしも十分な機能を果たしていない．

(2) 必要な機能であるべき汎化階層構成，Java などのオブジェクト指向プログラミング言語 (以降，OOPL と略す) ならばクラスとクラス階層，継承などの概念とその機能の実装，が未だ実現されていない．

† 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

これ等はODDJ当初の段階ではやむを得ない一面も存在した．それ故に現状のODDJでは「1クラス1インスタンス」と見なす特殊な設定で構築してきた．しかし汎化階層の概念を導入することは，DUの高い要求から見ても特に必要であるし，現代的な言語(記述言語)としても必要な機能であることには変わりはない．また，OONJとOPDJでは既に準備が出来ているのに，ODDJだけが実装しないのは問題であるということもあって，汎化階層を設計・実装する．

2. ODDJの改良設計

2.1 記述系言語系OOJの概略

OONJ，ODDJ，OPDJの3つの記述言語を連携させ，OONJ記述をODDJエディタで編集し，それをOPDJエディタに持ち込んで編集するなどの作業過程の中において，一部は手作業による改訂も行うことで，幾つかのOOPL(Java)のプログラムを生成することが出来た．しかし，それらのプログラムの機能は903行に上る「水の大気循環」というタイトルの飛び抜けて大きな記述例を除けば，簡単な計算など限定的なものである．

2.2 ODDJの改良設計の必要性

記述例の生成が必ずしも順調に進行しなかった理由は設計段階であるODDJにおいて，記述情報保存規則であるDTDと構造記述規則が十分に対応しておらずOONJとODDJ,そしてODDJとOPDJとの連携が不十分であったからである．

2.3 ODDJの改良設計の検討・変更

そこでODDJのDTDと構造記述規則を見直しを検討した．検討項目は以下のようであった．

(1) 1クラス1インスタンスの概念の拡張．これは汎化階層構成の導入にあたる．
(2) OPDJとの区分の曖昧さ
(3) エディタのGUIの共通化
(4) DTDの再設計

1クラス1インスタンスの概念の拡張

OOJ全体として1クラス1インスタンスが採用されている．これはDUにオブジェクト指向を意識させないためであった．しかし1クラス1インスタンスではプロトタイプベースに似た動きをする．本論文では特に，未実装であった汎化階層構造を取り入れることで記述の拡張性の向上を図った．

OPDJとの区分の曖昧さ

ODDJは特定のOOPLに依存しない形式でなくてはならない．しかし，汎用なOOPLの機能とは何かという問題があり，ODDJで記述するべき情報とOPDJで記述するべき情報を決定する必要があった．これを考慮して互いの作業領域を改定した．

エディタのGUIの共通化

ODDJは記述を作成する支援を行うためにODDJエディタを提供している．しかし，分析段階で使用するOONJの記述エディタとGUI，使用法が異なっていたためDUは複数のエディタの使用法を学ばなければならなかった．そこでDUの負担を減らすためにODDJエディタの仕様をOONJエディタに近づくように変更した．

ODDJのDTDの再設計

以前のDTDでは定義に不備があり構造記述規則を守れず，OONJ，OPDJとの連携を阻害していた．例として以前のDTDでは，構造記述規則によって定められたデータ型の制限などを定義していなかった．そのため意図しない情報が入りOPDJ記述への変換を妨げる可能性があった．そこで新しいDTDでは構造記述規則に基づいてデータ型やアクセス制約などを定義し，意図しないデータが入れないようにした．

3. 汎化階層構成のODDJへの導入

3.1 ODDJの汎化階層構成の設計

OONJにおいては汎化階層構造は自由に記述する事ができる．これは分析段階であるOONJにおいてDUの発想を妨げないためである．しかし,汎化階層構造はOOPLによって異なるため,ODDJでは汎化階層構造を厳密に定める必要がある．OOPLの汎化階層としては,多重継承を許可しているもの，単純継承のみ，多重継承の代わりに代替手段を用意するものあるが，ソフトウェア開発の非専門家であるDUを考慮し，単純継承をODDJの仕様とした．

3.2 ODDJの汎化階層構成の実装の概略

フレーム，スロットにはIDが与えられている．汎化階層は任意のフレームの下で存在しており，DUが対象のフレームとの汎化/特化を選択したとき相手のフレームのIDと汎化，特化どちらかの種類を持つことでフレーム間の汎化階層を表現する．相手のフレー

表 1 ODDJ 要素種類表
Table 1 ODDJ element kinds list

dfn1 フレーム	**dfn3 処理**
dfn1.1 オブジェクト	dfn3.1 内部処理メソッド | return 文
dfn1.2 初期状況	dfn3.1.1 if 文
dfn1.3 境界条件	dfn3.1.2 switch 文
dfn1.4 駆動シナリオ	dfn3.1.3 while 文
dfn1.5 対象世界全体共有変数	dfn3.2 呼び出しメソッド
	dfn3.2.1 メソッド呼び出し
	dfn3.2.2 マクロ呼び出し
	dfn3.3 (総称メソッド | 条件設定メソッド)
	dfn4 相互関連
	dfn4.2 集約
	dfn4.3 汎化 | 特化
	dfn4.4 類型化 | 実値化
dfn2 属性	**dfn5 定数定義**
dfn2.1 –	**dfn6 ODDJ 関数**
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 –	**dfn8 駆動制御**
dfn2.7 アクセス制約	dfn8.1 駆動シナリオ
dfn2.8 変数	–

ムも同様に ID と汎化階層の種類を取得し，双方が汎化階層を他のフレームとの汎化階層の構成を監視する事で，多重継承を禁止している．

4. 改良された設計記述言語 ODDJ の詳細

以前の仕様を旧仕様，本研究の仕様を新仕様として以下を説明する．

4.1 ODDJ で取り扱う要素種類

ODDJ で扱う要素種類を**表 1** 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，特定のプログラミング言語 (以降，PL と略) 表現に依存しない．表 1 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．

4.2 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．

4.3 OONJ 記述からの変換

ODDJ はまず分析段階である OONJ 記述から対象世界の構造を取得し，ODDJ 記述の雛型を作成する．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．

旧仕様では計算機に理解できない形で記述されている OONJ 記述は削除された．新仕様では対象世界の流れや振舞いの説明をする情報として、計算機が理解する必要の無い注釈の要素として抽出する．

4.4 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 では属性の名前のみが記述されている．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とアクセス制約を追加する．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や仮引数，アクセス制約などを追加する．ODDJ において，属性や振舞いで記述できる型を**表 2** に示す．旧仕様では属性にアクセス制約を行えなかったが新仕様では記述できる．

4.5 ODDJ の構造化記述規則

ODDJ の新仕様の構造化記述規則の一部を**表 3** に示す．**表 4** は表 3 における文法定義記号である．これらの構造化規則は OOSF[6] を基盤とし，設計段階の

表 2 記述可能なデータ型
Table 2 describable date type

変数	メソッド
	void
整数型	整数型
浮動小数点型	浮動小数点型
文字列型	文字列型
文字型	文字型
論理型	論理型

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 3 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．これらの構造化記述規則の他に特定の目的を持ったフレーム，特定スロット，式の構造化記述規則が存在する．表 3 の (1)〜(14) のように，旧仕様よりも変数や定数，引数のデータ型やアクセス制約，配列などのデータ構造を詳細に決定している．これにより数の表現力が上がった．

4.6 計算式の追加

OONJ では自由な分析のために DU が自身の方式で計算式を記述する事ができるため任意性が高く，計算機が理解できる形式に一意に変換する事ができない．ODDJ では，計算機が理解できるように計算式の表現に構造記述規則による制約を加える必要がある．

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．

既に OONJ で記述されている制御構文については，その構造は ODDJ 記述に変換されているため条件式のみを追加すればよい．

4.8 初期状況記述・境界条件記述

ODDJ ではオブジェクトの持つ属性を初期化するために 2 種類の記述を用いる．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，属性のデータ型が定義されていなかったり，日本語で記述されているなど計算機が理解できる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．

4.9 駆動シナリオ

駆動シナリオとは，作成したフレームを用いて計算手順を記述する対象世界を管理するフレームである．駆動シナリオ記述では，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 main スロットを記述する．旧仕様では main スロットは独立したフレームであったが，プログラムの開始場所という機能しか有していないため，新仕様では駆動シナリオにスロットとして取り込んだ．

4.10 実装記述言語への転送

プログラム処理において必要な記述が書けたら OPDJ に記述データを XML 形式で送る．特定の PL に由来する要素は実装記述言語で記述されるように分けられた．新仕様によって実装記述言語との対応関係がより緊密となったため，連携しやすくなった．

5. ODDJ 記述環境

5.1 データフォーマット

OOJ では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 ODDJ の構造規則に則った XML 形式で情報を保存する．

5.2 ODDJ エディタの機能概要

ODDJ エディタの機能は，以下である．

(1) OONJ 記述 (XML 形式) の ODDJ 記述への変換
(2) フレームの追加・削除・編集
(3) スロットの追加・削除・編集
(4) サブスロットの追加・削除・編集
(5) XSLT による ODDJ 記述のブラウザ表示
(6) ODDJ 記述 (XML 形式) の読み込み・保存
(7) ODDJ 記述の検証

これらの中で新しく追加された機能は OONJ 記述の ODDJ 記述への変換と ODDJ 記述の検証である．OONJ 記述の ODDJ 記述への変換とは指定し

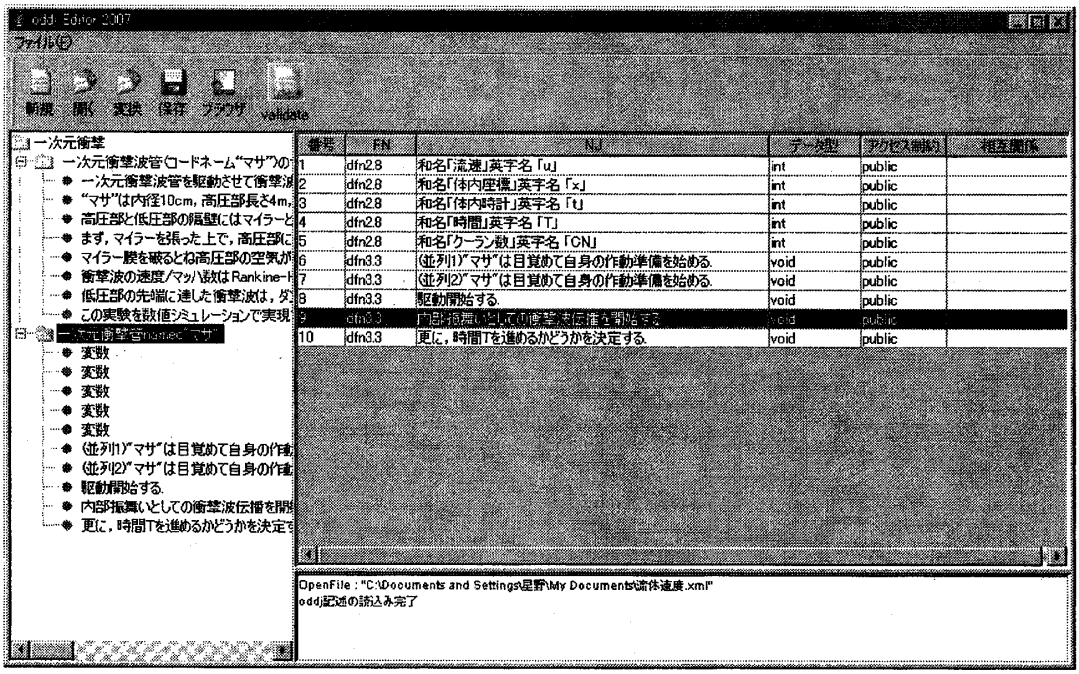

図 1 ODDJ エディタ画面例
Fig. 1 an ODDJ Editor display example

た OONJ 記述を ODDJ 記述に変換する機能である．ODDJ 記述の検証とは ODDJ 記述を DTD を用いて構造記述規則にしたがっているか検証する機能である．

5.3 ODDJ エディタの画面の構成例

図 1 にその実行中の画面を示す．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．画面中央背面の枠には，DU が作成した ODDJ 記述が表示され．DU はサブスロットを選択して計算式や条件式，制御構造を記述する．

以下にエディタの操作法を示す．

5.3.1 ドキュメント階層

図 1 の画面の左枠のツリービューから一番上のフォルダアイコンを右クリックする．メニューが現れるのでドキュメントの編集，フレームの追加の項目から目的の機能を選ぶ．ODDJ エディタが対話形式で次の行動を指示するので，それに従い作業を行う事で目的の機能を果たす．

5.3.2 フレーム階層

図 1 の画面の左枠のツリービューからドキュメント階層下の対象のフォルダアイコンを右クリックする．メニューが現れるのでフレームの削除・追加・編集，変数・スロットの追加の項目から目的の機能を選び作業を行う．

5.3.3 スロット・変数階層

図 1 の画面の左枠のツリービューからフレーム階層下の対象の丸アイコンを右クリックする．メニューが現れるのでスロットの追加・編集，スロット/変数の削除の項目から目的の機能を選び作業を行う．

5.3.4 サブスロット階層

図 1 の画面の左枠のツリービューから対象のサブスロットが所属するスロットを左クリックする．すると右の枠の画面にスロットに所属するサブスロット一覧が表示されるので対象のサブスロットを選択し，メニューから目的の機能を選び作業を行う．

5.3.5 OONJ 記述の ODDJ 記述への変換

図 1 の画面の上に並んでいるアイコンの中から変換と表示されているアイコンを選択する．するとファイル選択ダイアログが表示されるので目的の OONJ 記述データを選択する．選択された OONJ 記述データが OONJ 構造記述規則に従っていれば，ODDJ 記述データに変換され，エディタに表示される．

5.3.6 ODDJ 記述の検証

図 1 の画面の上に並んでいるアイコンの中から validate と表示されているアイコンを選択する．するとODDJ 記述データの検証が行われ，ODDJ 記述に不備があった場合下の枠の画面にエラーメッセージが表示される．

5.3.7 ODDJ 記述の XSLT による表示

図 1 の画面の上に並んでいるアイコンの中からブラウザと表示されているアイコンを選択する．するとODDJ 記述データである XML ファイルが XSLT を通して図 2 の形式でインターネットエクスプローラに表示される．

5.4 新仕様に対応した ODDJ エディタの評価

以前のエディタと比べ以下のことがわかった．

(1) 構造記述規則と DTD を見直したことにより，以前より OONJ 記述から ODDJ 記述への変換効率が上がり，記述する作業量が減少
(2) 配列を扱えるようになったり，変数にアクセス制約を明示的に与えられるなど記述力が向上
(3) ODDJ 記述を検証する機能により誤記によるプログラム変換時のミスを防ぐことが出来る
(4) OONJ エディタに GUI や操作性をあわせたことにより ODDJ エディタを使用する負担が減少

以上のことから ODDJ エディタの性能が上がったといえる．

6. ODDJ 記述例とその評価

ODDJ の記述例として Lax_Wendroff 法 (差分法) を用いた流体速度計算を**図 2** に示す．この記述例は OONJ 記述情報を ODDJ エディタを用いて変換，計算式の追加などを行い作成された．より具体的に作業を示すと，ODDJ 記述に変換されたのはフレーム，スロット，変数，制御構造，メソッド呼び出しである．これらのデータで OONJ 記述で記述されないデータ型やアクセス制約を追加，制御構造の条件式や計算式，初期状況記述，駆動シナリオを追加した．この記述例は OPDJ によって Java プログラムに変換でき，正しい結果を返した．これより，ODDJ の記述は正しく Java プログラムに変換できた．ただし，DU が入力した部分の一部は OONJ 記述から抽出できる情報であったので，この点は改善の余地がある．

7. まとめと今後の課題

本研究の一連の作業により，旧仕様の ODDJ と比べて以下の点が改良された．

(1) ODDJ 構造記述規則と DTD との対応
(2) ODDJ エディタの使いやすさ
(3) OONJ 記述からの ODDJ 記述への変換効率
(4) ODDJ の記述力

以上のことを踏まえると，

(1) 本研究の狙いの第 1 に挙げた点，すなわち，開発の上流工程の OONJ と，下流工程の OPDJ との連携は現仕様においてはかなり改善されたと結論できる．

(2) それに伴って，DU の使い勝手が大幅に向上した．例えば，OONJ と ODDJ のエディタの仕様，特にユーザインタフェースが近い仕様になったことなどが挙げられる．

(3) 汎化階層構成 (クラス，クラス階層，継承) については，ODDJ の DTD とエディタに実装されて使える状態ではある．ただし，それが論文の構造記述規則に対しては幾つかの問題点が解決していないゆえに，反映されておらず，今後の課題を残している．

参 考 文 献

1) 川澄成章, 野口和義, 畠山正行, 荒木俊郎: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, 第 150 回 SE 研究会報告,2005-SE-150, Nov.29,(2005).
2) 畠山正行：オブジェクト指向一環記述言語 OOJ の構成とその概念設計－オブジェクトベース一貫相似性過程の方法論をベースとして, ドメインユーザにも使いこなせるオブジェクト指向技術の提案－, 第 150 回 SE 研究会報告,2005-SE-150, Nov.29,(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, 第 150 回 SE 研究会報告,2005-SE-150, Nov.29,(2005).
4) 畠山正行, 松本賢人：オブジェクト指向分析記述言語 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).
5) 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-SE-150, Nov.29,(2005).
6) 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).

1 dfn1.1 一次元衝撃波管「マサ」

1	dfn3.3	初期条件取得		
		(実数型 pu[ ]) (実数型 px[ ]) (実数型 pt)	void	共有
	dfn3.1	| u[ ]= pu[ ]		
	dfn3.1	| x[ ]=px[ ]		
	dfn3.1	| t=pt		
	dfn3.2	| 計算開始 (t) >> 一次元衝撃波管「マサ」		
2	dfn3.3	計算開始 (実数型 t)	void	私有
	dfn3.1.1	| if (t>=0)		
	dfn3.1.1	| then		
	dfn3.2	| Lax_Wendroff 計算 >> 一次元衝撃波管「マサ」		
3	dfn3.3	Lax_Wendroff 計算	void	私有
	dfn2.8	離散点速度 (予測値) u3[20]	実数型	私有
	dfn2.8	離散点速度 (修正値) UN[20]	実数型	私有
	dfn2.8	離散空間座標パラメータ J	整数型	私有
	dfn3.1	| J=0		
	dfn3.1.3	| while (J<20)		
	dfn3.1	| u3[J]=0.5*((1.0-CN)*u[J+1]+(1.0+CN)*u[J])		
	dfn3.1	| J=J+1		
		...		
4	dfn3.3	継続判定	void	私有
	dfn3.1	| t=t+dt		
	dfn3.2	| 駆動制御を行う 2 (t) >> 駆動シナリオ		

2 dfn1.5 初期状況記述

1	dfn3.3	初期条件起動	void	共有
	dfn3.2	| マサの初期条件設定 >> 初期状況記述		
2	dfn3.3	マサの初期条件設定	void	私有
	dfn2.8	離散化パラメータ i	整数型	私有
	dfn3.1	| i=0		
	dfn3.1	| t=0.0		

3 dfn1.1 対象世界全体共有属性

1	dfn2.8	流速 u[21]	実数型	共有
2	dfn2.8	体内座標 x[21]	実数型	共有
3	dfn2.8	体内時計 t	実数型	共有
		...		
9	dfn3.3	一次元衝撃波流れの対象世界の定数定義を行う	void	共有
	dfn3.1	| dx=0.1		
	dfn3.1	| dt=0.2		
	dfn3.1	| CN=0.2		
	dfn3.1	| M=2.0		

4 dfn1.1 駆動シナリオ

1	dfn3.3	データ入力	void	私有
	dfn3.2	| 実数入力 (n) >> 標準入力		
	dfn3.2	| 駆動制御を行う 1 (n) >> 駆動シナリオ		
2	dfn3.3	駆動制御を行う 1 (実数型 n)	void	共有
	dfn3.2	| 初期条件起動 >> 初期状況記述		
3	dfn3.3	駆動制御を行う 2 (実数型 t)	void	共有
	dfn3.1.1	| if (t<0.4)		
	dfn3.1.1	| then		
	dfn3.2	| 計算開始 (t) >> 一次元衝撃波管「マサ」		
4	dfn3.3	データ出力 (実数型 u[ ])	void	私有
	dfn3.2	| 実数出力 (u[ ]) >> 標準出力		
5	dfn3.3	プログラム駆動開始	void	共有
	dfn3.2	| 標準入力 >> 駆動シナリオ		

図 2 ODDJ 記述例：一次元衝撃波菅「マサ」

Fig. 2 ODDJ description exsample : one-dimentional shock tube "masa"

表 3 ODDJ 構造記述規則 （一般/階層構造記述規則, 対象世界共通要素記述規則）
Table 3 ODDJ structured description rules

(1) ＜**対象世界**＞	::=	＜フレーム＞ (＜ lf ＞＜フレーム＞)$^{+}$ ＜対象世界全体駆動フレーム群＞
(2) ＜**フレーム**＞	::=	＜フレームヘッダスロット＞ (【＜ lf ＞＜フレーム内共有変数スロット＞】)* (【＜ lf ＞＜スロット＞】)$^{+}$
(3) 【 】	::=	≪矩形のフレーム実線枠内へのスロット収容記号≫
(4) ＜フレーム番号＞	::=	≪フレーム単位の一連番号で，通常は 1 から始まる昇順の整数≫
(5) ＜フレーム名＞	::=	≪オブジェクト名≫
(6) ＜ファセット記号＞	::=	「ODDJ 計算機世界の要素比較表」を参照
(7) ＜**スロット**＞	::=	≪短冊形のスロット実線外枠線≫≪＜スロット記述＞≫
(8) ＜スロット記述＞	::=	＜スロット総称文＞“(”＜仮引数＞ (“,”＜仮引数＞)* “)”＜ rt ＞＜型制約＞ (＜ lf ＞ (＜サブスロット＞ | ＜変数宣言サブスロット＞ | ＜コメント＞))$^{+}$ ＜ return 文＞$^{?}$
(9) ≪≫	::=	≪短冊形のスロット実線外枠内への収容を指示する記号≫
(10) ＜スロット番号＞	::=	≪スロット単位の一連番号で，通常は 1 から始まる昇順の整数≫
(11) ＜フレームヘッダスロット＞	::=	＜フレーム番号＞＜ sp ＞＜ファセット記号 (dfn1.1) ＞＜ sp ＞＜フレーム名＞
(12) ＜フレーム内共有変数スロット＞	::=	＜スロット番号＞＜ sp ＞＜ファセット記号 (dfn2.8) ＞＜ sp ＞＜数＞
(13) ＜**サブスロット**＞	::=	＜ sp ＞＜ファセット記号＞＜ sp ＞＜階層構造記述子＞
(14) ＜スロット総称文＞	::=	＜スロット番号＞＜ sp ＞＜ファセット記号 (dfn3.3) ＞＜ sp ＞≪ NJ 単位文≫
(15) ＜変数宣言サブスロット＞	::=	＜ファセット記号 (dfn2.8) ＞＜ sp ＞＜数＞
(16) ＜数＞	::=	＜数名称＞＜ sp ＞＜数構造＞＜ sp ＞＜数処理＞
(17) ＜数名称＞	::=	≪ NJ 動詞，動名詞 (対象世界内名称) ≫
(18) ＜数構造＞	::=	(＜数構造記号名＞＜数構造化定義＞ | ＜変数＞ | ＜定数＞)
(19) ＜数構造記号名＞	::=	≪アルファベット (A～z) で始まる英数字列≫
(20) ＜数構造化定義＞	::=	＜一次元配列＞	＜二次元配列＞
(21) ＜一次元配列＞	::=	“[”＜数の数量＞“]”
(22) ＜二次元配列＞	::=	“[”＜数の数量＞“]”“[”＜数の数量＞“]”
(23) ＜変数＞	::=	≪アルファベット (A～z) と“_”で始まる英数字列≫
(24) ＜定数＞	::=	≪アルファベット (A～z) と“_”で始まる英数字列 (実値必須) ≫
(25) ＜数の数量＞	::=	(＜整数＞	＜整数変数＞	＜算術式＞	＜“ ”(定義配列全体を表す) ＞)
(26) ＜数処理＞	::=	＜アクセス制約＞＜ sp ＞＜データ型＞ [＜ sp ＞≪処理メソッド名≫][＜ sp ＞＜数構造 (実値) ＞]
(27) ＜アクセス制約＞	::=	≪ private ≫	≪ public ≫
(28) ＜データ型＞	::=	(＜算術型＞ | ＜論理型＞ | ＜文字型＞ | ＜文字列型＞)
(29) ＜仮引数＞	::=	＜型制約＞＜数構造記号名＞
(30) ＜実引数＞	::=	＜型制約＞＜数構造記号名＞
(31) ＜型制約＞	::=	＜アクセス制約＞＜ sp ＞ (＜データ型＞ |“ void ”)
(32) ＜ rt ＞	::=	≪その右側の要素右詰め (右寄せ) 記述する．≫
(33) ＜ lf ＞	::=	≪次の行またはスロットに移り，その左先頭に戻ることを示す記号≫
(34) ＜ sp ＞	::=	≪任意一定幅空白≫
(35) ＜ mr ＞	::=	≪直上の行に“|”があればそこまで右シフト，なければそのまま．≫
(36) ＜階層構造記述子＞	::=	＜ mr ＞“|”
(37) ＜メソッド呼び出し＞	::=	＜ sp ＞＜メソッド呼び出し名＞＜ sp ＞“(”＜実引数＞ (“,”＜実引数＞)* “)”
(38) ＜対象フレーム＞	::=	＜フレーム名＞
(39) ＜対象スロット＞	::=	＜スロット総称文 (≪ NJ 単位文≫のみ) ＞
(40) ＜メソッド呼び出し名＞	::=	＜対象フレーム＞“≫”＜対象スロット＞

表 4 構造記述規則の表記法一覧表
Table 4 expression table of structured description rules

(0) アングルブラケット＜ … ＞，またはギュメ (ギルメット) ≪ … ≫は要素を表す．
(1) ＜ … ＞は別の内部構成要素に再帰展開すべき/可能な要素であることを示す．
(2) ≪ … ≫は内部要素の再帰展開は完了し，対象世界からモデリングし/記述すべき要素であることを示す．
(3) ＜ … ＞^{n} は n 個 (n=1 は省略可) の要素並び
(4) ＜ … ＞$^{+n}$ は n 個 (n=1 は省略可) 以上の要素の並び．
(5) ＜ … ＞* は 0 個以上の要素並び
(6) ＜ … ＞$^{?}$ は 0 個または 1 個 (無し/有り) の要素
(7) ＜ … ＞$^{++}$ は条件を満たす全要素を挙げることを指す．
(8) “|” はこの記号の左右にある要素の選択を示す．