GearsOSの分散ファイルシステムの設計

一木貴裕[,a)]　河野 真治[,b)]

概要：本研究室では gear というプログラミング概念を持つ, 分散フレームワーク Christie を開発している. Christie はノード同士が Datagear と呼ばれる変数データを送信しあうことにより, 簡潔に分散プログラムの記述を行うことができる. この Christie の仕組みを, 同様に本研究室が開発している GearsOS に組み込み, ファイルシステムを構築したい. GearsOS はノーマルレベルとメタレベルを分けて記述できる Continuation based C(CbC) で構成されており、Christie と近い仕様をもつ.

Designing a Distributed File System for GearsOS

1. GearsOS のファイルシステムの開発

当研究室では OS の信頼性の検証を目的とした OS である GearsOS を開発している. GearsOS はユーザレベルとメタレベルを分離して記述が行える言語である Continuation based C(以下 CbC) で記述されており, Gear というプログラミング概念を持つ.

GearsOS は現在開発途上であるため, 現在は言語フレームワークとしてしか動作しない。OS として起動するためにこれから実装が必要な機能が多く存在しており, その中の一つとして分散ファイルシステムが挙げられる. GearsOS の分散ファイルシステムを構成するために、当研究室が開発している分散フレームワーク Christie の仕組みを用いようと考えた.

Chrsitie は GearsOS のもつ Gear という概念とよく似た, 別の Gear というプログラミング概念を持っており, DataGear と呼ばれる変数データを接続されたノード同士が送信しあうことで分散処理を簡潔に記述することができる. DataGear は指定された型と名前を持つ key に対応しており, プログラムが必要な key にデータが揃ってから初めてプログラムが処理される. また, Chrisite は Topology-Manager と呼ばれる機能を持っており, 任意の形でノード同士の配線を行い Topology を形成する機能を持っている.

[1] 情報処理学会
IPSJ, Chiyoda, Tokyo 101–0062, Japan
[†1] 現在，琉球大学理工学研究科情報工学専攻
Presently with Johoshori Uniersity
[a)] ikki-tkhr@cr.ie.u-ryukyu.ac.jp
[b)] kono@ie.u-ryukyu.ac.jp

2. 現代のファイルシステムについて

3. Continuation based C

GearsOS は C 言語の下位言語である Continuation based C を用いて記述されている. CbC は関数呼び出しでなく, 継続を導入しており, スタック領域を用いず jmp 命令でコード間を移動することにより軽量な継続を実現している. CbC ではこの継続を用いて for 文などのループの代わりに再起呼び出しを行う. 実際の OS やアプリケーションを記述する際には GCC または LLVM/clang の CbC 実装を用いる.

CbC では関数の代わりに CodeGear という単位でプログラミングを行う. CodeGear は`__code` で宣言を行い, 各 CodeGear は DataGear と呼ばれる変数データを入力として受け取り, その結果を別の DataGear に書き込む. 特に入力の DataGeat を InputDataGear, 出力される DataGear を OutputDataGear と呼ぶ. CodeGear と DataGear の関係図を図 1 に示す. CodeGear は関数呼び出しのスタックを持たないため, 一度 CodeGear を遷移すると元の処理に戻ってくることができない.

CbC コードの例をソースコード 1 に示す.

```
void syscall(void)
#include <stdio.h>
__code CG2(){
  int i = 10;
  printf("i␣=␣%d\n", i);
}
```

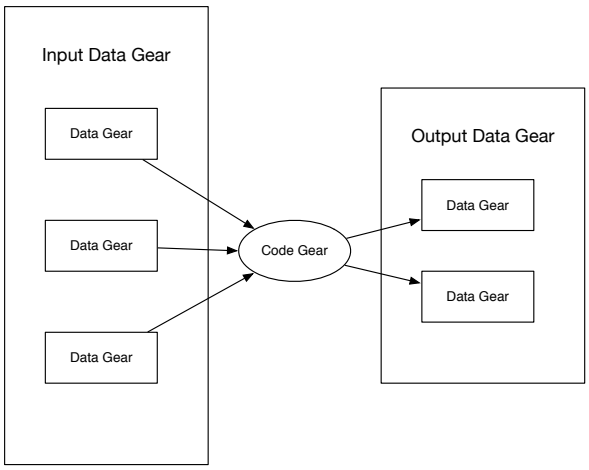

図 1: CodeGear と入出力の関係図

```
__code CG1(){
  printf("Hello\n");
  goto CG2();
}

int main(){
  goto CG1();
}
```

Code 1: CbC の例題

4. CbC を用いた OS の記述

CodeGear の遷移はノーマルレベルから見ると単純に CodeGear が DataGear を Input, Output をのみ繰り返し, コードブロックを移動しているように見える. CodeGear が別の DataGear に遷移する際の DataGear との関係性を図 2 に示す. ノーマルレベルでは DataGear を受け取った CodeGear を実行, 実行結果を DataGear に書き込み別の CodeGear に継続していると見える.

しかし, 実際には CodeGear から別の CodeGear への遷移にはデータの整合性の確認などのメタ計算が必要となる. コード間の遷移に必要となるメタ計算は, MetaCodeGear と呼ばれる CodeGear ごとに実装された CodeGear で行う. MetaCodeGear で参照される DataGear を MetaDataGear 呼び, また, CodeGear の直前に実行される MetaCodeGear を StubCodeGear と呼ぶ. これら Meta 計算部分を含めた CodeGear の遷移と DataGear の関係性を図示すると図 2 の下段の形に表せる. CordGear の実行前後に実行される MetaCodeGear や入出力の DataGear を MetaDagaGear から取り出すなどのメタ計算が加わる.

MetaCodeGear は詳細な処理の変更や, スクリプトに問題がある場合を除き, プログラマが直接実装する必要がなく, GearsOS が持つ Perl スクリプトにより, GearsOS がビルドされる際に生成される.

CodeGear の遷移に重要な役割を持つ MetaDataGear と

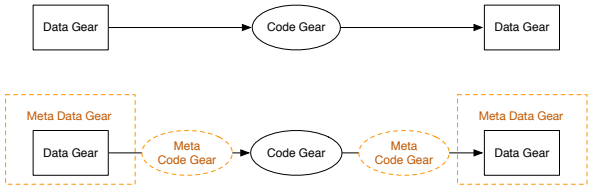

図 2: CodeGear と MetaCodeGear の関係図

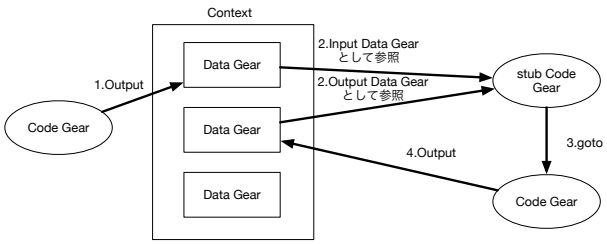

図 3: Context を介した CodeGear の継続

して context が存在する。context は遷移先の CodeGear と MetaDataGear の紐付けや, 計算に必要な DataGear の保存や管理を行う. 加えて context は処理に必要になる CodeGear の番号と MetaCodeGear の対応表や, DataGear の格納場所を持つ. context と各データ構造の役割を図 3 に示す. 計算に必要なデータ構造と処理を持つデータ構造であることから, context は従来の OS のプロセスに相当し, ユーザープログラムごとに context が存在している.

5. 分散フレームワーク Christie

Christie は当研究室で開発されている java 言語で記述された, 分散フレームワークである. Christie は CbC と似ているが異なる仕様を持つ Gear というプログラミング概念を持つ.

- `CodeGear`（以下 CG）
- `DataGear`（以下 DG）
- `CodeGearManager`（以下 CGM）
- `DataGearManager`（以下 DGM）

CodeGear はクラスやスレッドに相当する. DataGear は変数データであり, CodeGear 内で java のアノテーションを用いて記述する.

DataGear は Key と必ず対応しており, CodeGear 内の全ての Key に DataGear が揃った際に初めて CodeGear が動作するという仕組みになっている.

CodeGearManager はいわゆるノードに相当し, CodeGear, DataGear, DataGearManager を管理する. 複数の CodeGearManager 同士が配線され, DataGear を送信し合うことで分散処理を実現している.

DataGearManager は DG を管理しているもので変数プールに相当し, CodeGearManager の持っている DataGear の key と put されたデータの全てを所持している. DataGearManager は Local と Remote に区分することができ, Local-

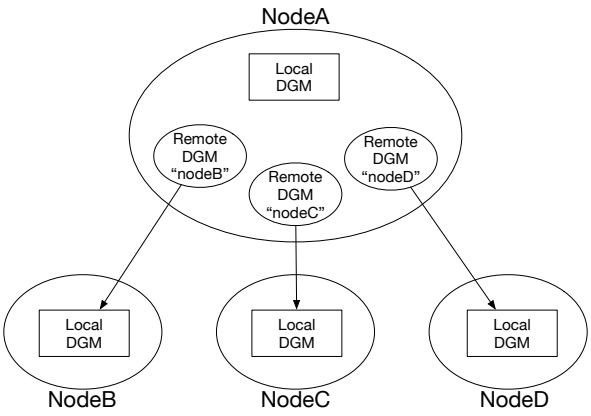

図 4: RemoteDataGear と接続ノードの関係図

DataGearManager は CodeGearManager 自身が所持する DataGear(key) のプールであり, Local に put することにより自身の持つ key に DataGear を送ることができる. 対する RemoteDataGearManager は CodeGearManager が配線されている別の CodeGearManager が持つ DataGear のプールである. つまり, 任意の接続された RemoteDataGear に DataGear を put すると対応したノードが持つ key に DataGear が送信される. RemoteDataGear に DataGear を put する処理が分散処理の肝となっている. RemoteDataGear の仕組みを図 4 に示す.

Christie の要となる DataGear の key は java のアノテーション機能が使われている. アノテーションには以下の 4 つが存在する.

Take 先頭の DG を読み込み，その DG を削除する. DG が複数ある場合，この動作を用いる.

Peek 先頭の DG を読み込むが, DG が削除されない. そのため，特に操作をしない場合は同じデータを参照し続ける.

TakeFrom(Remote DGM name) Take と似ているが，Remote DGM name を指定することで，その接続先 (Remote) の DGM から Take 操作を行える.

PeekFrom(Remote DGM name) Peek と似ているが，Remote DGM name を指定することで，その接続先 (Remote) の DGM から Peek 操作を行える.

コード 2, 3 は Christie で記述した Hello World のプログラムである. ユーザープログラムは StartCodeGear クラスを継承したクラス (コード 2) から開始する. CodeGearManager はポート番号を指定した上で creatCGM メソッドを呼び出すことにより生成される. 生成された CodeGearManager は CGM 名.setup にて CGM に処理させたいスレッド, つまり CodeGear を持たせることができる.

コード 3 は HeloWorldCodeGear の記述である. HelloWorldCodeGear では key: helloWorld に put された文字列を print 出力するという単純な処理を記述している. CGM 名.getLocalDGM().put(”Keyname”, 変数データ) にて key に変数データを紐付け (put し), CodeGear に設定されている全ての key がデータを受け取った際に初めて CodeGear は処理される. HelloWorldCodeGear では String 型の helloWorld という key が Take 型で設定されている.

以下の HelloWorld プログラムを実行した際の流れを説明する. まずポート 10000 番の CodeGearManager を生成し, HelloWorldCodeGear を setup させる. この時点では必要な key(key 名: helloWorld) にデータが揃っていないので CodeGear は実行されない. cgm.getLocalDGM().put(”helloWorld”,”hello”); にて helloWorldkey に文字列”hello”を put すると, HelloWorldCodeGear に必要な DataGear が揃い, print 表示が行われる. プログラム中では key:helloWorld への put は文字列”hello”と”world”の二回が行われ, print 出力結果は hello world と表示される.

```
public class StartHelloWorld extends StartCodeGear {

    public StartHelloWorld(CodeGearManager cgm) {
        super(cgm);
    }

    public static void main(String[] args){
        CodeGearManager cgm = createCGM(10000);
        cgm.setup(new HelloWorldCodeGear());
        cgm.getLocalDGM().put("helloWorld","hello");
        cgm.getLocalDGM().put("helloWorld","world");
    }
}
```

Code 2: Christie における CGM と CG の setup

```
public class HelloWorldCodeGear extends CodeGear {
    @Take
    String helloWorld;

    @Override
    protected void run(CodeGearManager cgm) {
        System.out.print(helloWorld + "␣");
        cgm.setup(new HelloWorldCodeGear());
    }
}
```

Code 3: HelloWorldCodeGear

Christie には Topology を形成するための機能 TopologyManager が備わっている. Topology に参加するノードに対して名前を与え, 必要とあればノード間の配線を行う.

TopologyManager の Topology 形成方法として静的 Topology と動的 Topology がある. 静的 Topology はプログラマが任意の形の Topology とノードの配線を dot ファイルに記述し, TopologyManager に参照させることで自由な形の Topology が形成できる. 現時点では静的 Topology での Topology 形成は dot ファイルに記述した参加ノード数に実際に参加するノードの数が達していない場

合, 動作しないという制約が存在している. 動的 Topology は参加を表明したノードに対し, 自動的にノード同士の配線を行う. 例えば Tree を構成する場合, 参加したノードから順番に root から近い役割を与える.

6. 論文の構成

6.1 表題・著者名等

表題，著者名とその所属，および概要を前述のコマンドや環境により**和文と英文の双方について**定義した後，\maketitle によって出力する.

6.1.1 表題

表題は，\title および \etitle で定義した表題はセンタリングされる．文字数の多いものについては，適宜 \\ を挿入して改行する.

6.1.2 著者名・所属

各著者の所属を第一著者から順に \affiliate を用いてラベル（第 1 引数）を付けながら定義すると，脚注に番号を付けて所属が出力される．なお，複数の著者が同じ所属である場合には，一度定義するだけで良い.

現在の所属は \paffiliate を用い，同様にラベル，所属先を記述する．所属先には自動で「現在」，\\の改行で「Presently with」が挿入される．著者名は \author で定義する．各著者名の直後に，英文著者名，所属ラベルとメールアドレスを記入する．著者が複数の場合は \author を繰り返すことで，2 人，3 人，...と増えていく．現在の所属や，複数の所属先を追加する場合には，所属ラベルをカンマで区切り，追加すればよい.

また，メールアドレス部分は省略が可能である.

6.1.3 概要

和文の概要は abstract 環境の中に，英文の概要は eabstract 環境の中に，それぞれ記述する.

6.2 本文

6.2.1 見出し

節や小節の見出しには \section, \subsection, \subsubsection, \paragraph といったコマンドを使用する.

「定義」,「定理」などについては，\newtheorem で適宜環境を宣言し，その環境を用いて記述する.

6.2.2 行送り

2 段組を採用しており，左右の段で行の基準線の位置が一致することを原則としている．また，節見出しなど，行の間隔を他よりたくさんとった方が読みやすい場所では，この原則を守るようにスタイルファイルが自動的にスペースを挿入する．したがって本文中では \vspace や \vskip を用いたスペースの調整を行なわないようにすること.

6.2.3 フォントサイズ

フォントサイズは，スタイルファイルによって自動的に設定されるため，基本的には著者が自分でフォントサイズを変更する必要はない.

6.2.4 句読点

句点には全角の「.」，読点には全角の「,」を用いる．ただし英文中や数式中で「.」や「,」を使う場合には，半角文字を使う．「。」や「、」は使わない.

6.2.5 全角文字と半角文字

全角文字と半角文字の両方にある文字は次のように使い分ける.

(1) 括弧は全角の「(」と「)」を用いる．但し，英文の概

要，図表見出し，書誌データでは半角の「(」と「)」を用いる．
(2) 英数字，空白，記号類は半角文字を用いる．ただし，句読点に関しては，前項で述べたような例外がある．
(3) カタカナは全角文字を用いる．
(4) 引用符では開きと閉じを区別する．開きには ‘‘ を用い，閉じには’’ を用いる．

6.2.6 箇条書

箇条書に関する形式を特に定めていない．場合に応じて標準的な enumerate, itemize, description の環境を用いてよい．

6.2.7 脚注

脚注は \footnote コマンドを使って書くと，ページ単位に[*1]や[*2]のような参照記号とともに脚注が生成される．なお，ページ内に複数の脚注がある場合，参照記号は LaTeX を 2 回実行しないと正しくならないことに注意されたい．

また場合によっては，脚注をつけた位置と脚注本体とを別の段に置く方がよいこともある．この場合には，\footnotemark コマンドや \footnotetext コマンドを使って対処していただきたい．

なお，脚注番号は論文内で通し番号で出力される．

6.2.8 Overfull と Underfull

組版時には overfull を起こさないことを原則としている．従って，まず提出するソースが著者の環境で overfull を起こさないように，文章を工夫するなどの最善の努力を払っていただきたい．但し，flushleft 環境，\\, \linebreak などによる両端揃えをしない形での overfull の回避は，できるだけ避けていただきたい．また著者の執筆時点では発生しない overfull が，組版時の環境では発生することもある．このような事態をできるだけ回避するために，文中の長い数式や \verb を避ける，パラグラフの先頭付近では長い英単語を使用しない，などの注意を払うようにして頂きたい．

6.3 数式

6.3.1 本文中の数式

本文中の数式は $ と $, \(と \), あるいは math 環境のいずれで囲んでもよい．

6.3.2 別組の数式

別組数式 (displayed math) については $$ と $$ は使用せずに，\[と \] で囲むか，displaymath, equation, eqnarray のいずれかの環境を用いる．これらは

$$\Delta_l = \sum_{i=l|1}^{L} \delta_{pi} \qquad (1)$$

のように，センタリングではなく固定字下げで数式を出力し，かつ背が高い数式による行送りの乱れを吸収する機能がある．

[*1] 脚注の例.
[*2] 二つめの脚注.

```
\begin{figure}[tb]
  <図本体の指定>
\caption{<和文見出し>}
\ecaption{<英文見出し>}
\label{ ... }
\end{figure}
```

図 5: 1 段幅の図
Fig. 5 Single column figure with caption explicitly broken by \\.

6.3.3 eqnarray 環境

互いに関連する別組の数式が 2 行以上連続して現れる場合には，単に\[と \]，あるいは \begin{equation} と\end{equation} で囲った数式を書き並べるのではなく，\begin{eqnarray} と \end{eqnarray} を使って，等号（あるいは不等号）の位置で縦揃えを行なった方が読みやすい．

6.3.4 数式のフォント

LaTeX が標準的にサポートしているもの以外の特殊な数式用フォントは，できるだけ使わないようにされたい．どうしても使用しなければならない場合には，その旨申し出て頂くとともに，組版工程に深く関与して頂くこともあることに留意されたい．

6.4 図

1 段の幅におさまる図は，**図 5** の形式で指定する．位置の指定に h は使わない．また，図の下に和文と英文の双方の見出しを，\caption と \ecaption で指定する．文字数が多い見出しはは自動的に改行して最大幅の行を基準にセンタリングするが，見出しが 2 行になる場合には適宜 \\ を挿入して改行したほうが良い結果となることがしばしばある（図 5 の英文見出しを参照）．図の参照は \figref{<ラベル>} を用いて行なう．

また紙面スペースの節約のために，1 つの figure（または table）環境の中に複数の図表を並べて表示したい場合には，**図 6** と **表 1** のように個々の図表と各々の \caption/\ecaption を minipage 環境に入れることで実現できる．なお図と表が混在する場合，minipage 環境の中で\CaptionType{figure} あるいは \CaptionType{table} を指定すれば，外側の環境が figure であっても table であっても指定された見出しが得られる．

2 段の幅にまたがる図は，**図 7** の形式で指定する．位置の指定は t しか使えない．

図の中身では本文と違い，どのような大きさのフォントを使用しても構わない（図 7 参照）．また図の中身として，encapsulate された PostScript ファイル（いわゆる EPS ファイル）を読み込むこともできる．読み込みのため

```
\begin{minipage}[t]%
  {0.5\columnwidth}
\CaptionType{table}
\caption{ ...}
\ecaption{ ...}
\label{ ...}
\makebox[\textwidth][c]{%
\begin{tabular}[t]{lcr}
\hline\hline
left&center&right\\\hline
L1&C1&R1\\
L2&C2&R2\\\hline
\end{tabular}}
\end{minipage}
```

図 6: 表 1 の中身
Fig. 6 Contents of Table 1.

表 1: 図 6 で作成した表
Table 1 A table built by Fig. 6.

left	center	right
L1	C1	R1
L2	C2	R2

表 2: 表の例
Table 2 An Example of Table.

	column1	column2	column3
row1	item 1,1	item 2,1	—
row2	—	item 2,2	item 3,2
row3	item 1,3	item 2,3	item 3,3
row4	item 1,4	item 2,4	item 3,4

には，プリアンブルで

```
\usepackage{graphicx}
```

を行った上で，`\includegraphics` コマンドを図を埋め込む箇所に置き，その引数にファイル名（など）を指定する．

6.5 表

表の罫線はなるべく少なくするのが，仕上がりをすっきりさせるコツである．罫線をつける場合には，一番上の罫線には二重線を使い，左右の端には縦の罫線をつけない（**表 2**）．表中のフォントサイズのデフォルトは`\footnotesize` である．

また，表の上に和文と英文の双方の見出しを，`\caption` と `\ecaption` で指定する．表の参照は `\tabref{<ラベル>}` を用いて行なう．

6.6 参考文献・謝辞

6.6.1 参考文献の参照

本文中で参考文献を参照する場合には`\cite` を使用する．参照されたラベルは自動的にソートされ，`[]` でそれぞれ区切られる．

文献 `\cite{companion,okumura}` は LaTeX の総合的な解説書である．

と書くと；

文献 [1], [2] は LaTeX の総合的な解説書である．

が得られる．

6.6.2 参考文献リスト

参考文献リストには，原則として本文中で引用した文献のみを列挙する．順序は参照順あるいは第一著者の苗字のアルファベット順とする．文献リストは BiBTeX と `ipsjunsrt.bst`（参照順）または `ipsjsort.bst`（アルファベット順）を用いて作り，`\bibliograhpystyle` と`\bibliography` コマンドにより利用することが出来る．これらを用いれば，規定の体裁にあったものができるので，できるだけ利用していただきたい．また製版用のファイル群には`.bib` ファイルではなく`.bbl` ファイルを必ず含めることに注意されたい．一方，何らかの理由で thebibliography 環境で文献リストを「手作り」しなければならない場合は，このガイドの参考文献リストを注意深く見て，そのスタイルにしたがっていただきたい．

6.6.3 謝辞

謝辞がある場合には，参考文献リストの直前に置き，`acknowledgment` 環境の中に入れる．

7. 論文内容に関する指針

論文の内容について，論文誌ジャーナル編集委員会で作成した「べからず集」を以下に示す．投稿前のチェックリストとして利用頂きたい．これ以外にも，査読者用，メタ査読者用の「べからず集」[8] も公開しているので，参照されたい．また，作文技術に関する [3], [4], [5], [6] のような書籍も参考になる．

7.1 書き方の基本

- □ 研究の新規性，有用性，信頼性が読者に伝わるように記述する．
- □ 読み手に，読みやすい文章を心がける（内容が前後する，背景・課題の設定が不明瞭などは読者にとって負担）．
- □ 解決すべき問題が汎用化（一般的に記述）されていないのは再考を要する（XX 大学の問題という記述に終始）．あるいは，（単に「作りました」だけで）解決すべき問題そのものの記述がないのは再考を要する．
- □ 結論が明確に記されていない，または，範囲，限界，問題点などの指摘が適切ではない，または，結論が内

```
\begin{figure*}[t]
  <図本体の指定>
\caption{<和文見出し>}
\ecaption{<英文見出し>}
\label{ ... }
\end{figure*}
```

図 7: 2 段幅の図
Fig. 7 Double column figure.

容にそったものではないものは再考を要する.
□ 科学技術論文として不適当な表現や，分かりにくい表現があるのは再考を要する.
□ 極端な口語体や，長文の連続などは再考を要する.
□ 章，節のたて方，全体の構成等が適切でない文章は再考を要する.
□ 文中の文脈から推測しないと内容の把握が困難な論文にしない.
□ 説明に飛躍した点があり，仮説等の説明が十分ではないのは再考を要する.
□ 説明に冗長な点，逆に簡単すぎる点があるのは再考を要する.
□ 未定義語を減らす.

7.2 新規性と有効性を明確に示す
□ 在来研究との関連，研究の動機，ねらい等が明確に説明されていないのは再考を要する.
□ 既知／公知の技術が何であって，何を新しいアイデアとして提案しているのかが書かれていないのは再考を要する.
□ 十分な参考文献は新規性の主張に欠かせない.
□ 提案内容の説明が，概念的または抽象的な水準に終始していて，読者が提案内容を理解できない（それだけで新規性が感じられないもの）のは再考を要する.
□ 論文で提案した方法の有効性の主張がない，またはきわめて貧弱なのは再考を要する.

7.3 書き方に関する具体的な注意
□ 和文標題が内容を適切に表現していないのは再考を要する.
□ 英文標題が内容を適切に表現していない，または英語として適切でないのは再考を要する.
□ アブストラクトが主旨を適切に表現していない，または英文が適切ではないのは再考を要する.
□ 記号・略号等が周知のものでなく，または，用語が適切でなく，または，図・表の説明が適当ではないのは再考を要する.
□ 個人的あるいは非常に小さなグループ／企業だけで通用するような用語が特別な説明もなしに多用されているのは再考を要する.
□ 図表自体は十分に明確ではない，または誤りがあるのは再考を要する.
□ 図表が鮮明ではないのは再考を要する.
□ 図表が大きさ，縮尺の指定が適切でないのは再考を要する.

7.4 参考文献
□ 参考文献は 10 件以上必要（分野によっては 20 件以上，30 件以上という意見もある）.
□ 十分な参考文献は新規性の主張に欠かせない.
□ 適切な文献が引用されておらず，その数も適切ではないのは再考を要する.
□ 日本人によるしかるべき論文を引用することで日本人研究コミュニティの発展につながる.
□ 参考文献は自分のものばかりではだめ.

7.5 二重投稿
□ 二重投稿はしてはならない — ただし国際会議に採択された論文を著作権が問題にならないように投稿することは構わない.
□ 他の論文とまったく同じ図表を引用の明示なしに利用することは禁止.
□ 既発表の論文等との間に重複があるのは再考を要する.

7.6 他の人に読んでもらう
□ 投稿経験が少ない人は，採録された経験の豊富な人に校正してもらう.
□ 読者の立場から見て論理的な飛躍がないかに注意して記述する.

7.7 その他
□ 投稿前にチェックリストの各項目を満たしているか，必ず確認する.

8. おわりに

本稿では，A4 縦型 2 段組み用に変更したスタイルファ

イルを用いた論文のフォーマット方法と，論文誌ジャーナル編集委員会がまとめた「べからず集」に基づく論文の書き方を示した．内容的にまだ不十分の部分が多いため，意見，要望等を

editt@ipsj.or.jp

までお寄せ頂きたい．

謝辞 A4 横型に対するガイドを基に，本稿を作成した．クラスファイルの作成においては，京都大学の中島　浩氏にさまざまなご教示を頂き，さらに BiBTEX 関連ファイルの利用についても快諾頂いたことを深謝する．また，A4 横型に対するガイドを作成された当時の編集委員会の担当者に深謝する．

参考文献

[1] 奥村晴彦：改訂第 5 版 LATEX 2ε 美文書作成入門，技術評論社 (2010).
[2] Goossens, M., Mittelbach, F. and Samarin, A.: *The LaTeX Companion*, Addison Wesley, Reading, Massachusetts (1993).
[3] 木下是雄：理科系の作文技術，中公新書 (1981).
[4] Strunk W. J. and White E.B.: *The Elements of Style, Forth Edition*, Longman (2000).
[5] Blake G. and Bly R.W.: *The Elements of Technical Writing*, Longman (1993).
[6] Higham N.J.: *Handbook of Writing for the Mathematical Sciences*, SIAM (1998).
[7] 情報処理学会論文誌ジャーナル編集委員会：投稿者マニュアル (online)，入手先 ⟨http://www.ipsj.or.jp/journal/submit/manual/j_manual.html⟩ (2007.04.05).
[8] 情報処理学会論文誌ジャーナル編集委員会：べからず集 (online)，入手先 ⟨http://www.ipsj.or.jp/journal/manual/bekarazu.html⟩ (2011.09.15).