記　　号　　処　　理　74－1
プログラミング－言語・基礎・実践－　17－1
（１９９４．　６．　３）

複数のページ記述言語(PDL)に対応する処理系の実現

森田雅夫，小野澤雄二，堀　素史，伊知地宏，東方聖朝
富士ゼロックス株式会社
システムコミュニケーション研究所

ページ記述言語は，強力な描画能力を持つプログラミング言語であり，文書をプリンタなどに送るために主として用いられる。代表的なページ記述言語にはInterpress*1, PostScript*2それにSPDLがある。我々はこれらの言語間の類似性に着目し，PDL処理系を各言語に共通な部分と各言語に固有な部分に分割して，言語固有な部分の追加で，複数のページ記述言語に対応できる処理系を実現した。独立した複数の処理系を持つ場合に比べてコード量を削減できる。本論文は，この処理系の設計，実装およびその評価について述べる。

Design and Implementation of Multilingual Decomposer for Page Description Language

Masao Morita, Yuji Onozawa, Motofumi Hori, Hiroshi Ichiji, Masato Toho
Systems & Communications Laboratory
Fuji Xerox Co., Ltd.

A page description language is a programming language with strong functions for drawing and is mainly used to represent documents sent to printers. Widely known examples of page description languages are Interpress*1, PostScript*2, and SPDL. We focused on the commonality of the page description languages, and designed and implemented a multilingual decomposer that consists of a common part for multiple languages and a language dependent part for each language. This decomposer can support multiple page description languages with smaller program size. This paper describes the design, implementation and evaluation of this decomposer.

*1　Interpressは米国Xerox Corp.の登録商標である。
*2　PostScriptは米国Adobe Systems Inc.の登録商標である。

1. 緒言

ネットワーク上のワークステーションやパーソナルコンピュータで作成された文書は，ネットワークを通してプリンタに送られ，出力される。さまざまな種類のワークステーションとプリンタがネットワークに接続されるが，文書はプリンタにもワークステーションにも依存しない標準的なデータ形式でプリンタへ送られる。標準的なデータ形式を用いることにより，ワークステーションのプリンタドライバの共通化とプリンタのネットワーク接続性の向上が実現されている。この標準的なデータ形式はページ記述言語(Page Description Language; PDL)と呼ばれる一種のプログラミング言語である。

一方，プリンタの出力データ形式は通常，ラスタ画像である。そのため，プリンタ側ではページ記述から出力画像への変換処理が必要となる。この処理はプリントの速度や画質に直接影響するためプリンタにとって重要である。この処理を行うシステムをデコンポーザと呼ぶ。

多くのページ記述言語があるが，代表的なものとしては，Xerox 社のInterpress[1)2)] と Adobe Systems 社のPostScript[3)4)5)] がある。また，国際標準 Standard Page Description Language (SPDL) を作る活動が ISO で行われており，1993年9月に DIS が承認された[6)]。しかし，現在は，まだページ記述言語が統一されていないため，複数種類のページ記述言語に対応するためには複数のプリンタが必要となっている。

これを解決する試みとしては，各言語に対応する複数のデコンポーザを一つのプリンタが持つことや，ある言語を別の言語に変換することが行われている。後者の例としては，Interpress のオペレータをPostScript で記述することにより，Interpress によるページ記述を PostScript に変換した例がある[7)]。

しかし，前者には比較的大規模なプログラムであるデコンポーザを対応するページ記述言語の数だけ持たなければならないという問題があり，後者には変換されるページ記述の処理時間が増加するという問題があった。

我々は，ページ記述言語が類似した言語構造，描画能力を持つことに着目し，デコンポーザの処理を言語間で共通の部分と，各言語に依存する部分に分けたデコンポーザを研究，開発した。共通部と言語依存部の分離により，言語依存部の追加だけで新しいページ記述言語に対応できる。このデコンポーザは，コードの共通化によるデコンポーザのプログラム量の削減，新しいページ記述言語への拡張性，新たな出力デバイスへの対応のしやすさ，フルカラーのサポートを狙いとしている。本論文では，デコンポーザの設計，実装，および PostScript デコンポーザとしての性能評価について述べる。

2. ページ記述言語

ページ記述言語はプリンタで出力されるページを表現するためのプログラミング言語であり，一般的なプログラミング言語の機能に加えて描画機能を持つ。一般的なプログラミング言語機能としては実数，配列などのデータ型とそれに対する演算を与えるオペレータ，それにループ，条件分岐，手続きなどの実行制御を持つ。プログラミング言語としての特徴は，演算子の後置記法(逆ポーランド記法)を採用していること，スタック型言語であること，型宣言がないことなどが挙げられる。描画データとしては，ベクタ図形，文字，ラスタ画像などを持ち，その演算としては，例えば座標変換，色の指定などがある。

Interpress，PostScript および SPDL について言語仕様を比較した結果の要約を表1に示す。次節以降，これらの言語間の相違について述べる。

2.1 データ型

各ページ記述言語のデータ型を表2に示す。これらの言語のデータ型は基本的には同じで，共通して持っているデータ型に対する演算もほぼ共通しているが，以下のような違いがある。

まず Interpress は，ヌル型と辞書型を持っていない。ヌル型は不定値を表すのに用いられるもので大し

表1 各PDLの比較

	Interpress	PostScript	SPDL
文書構造	あり	なし	なし
データ型と属性	11種類(なし)	17種類(アクセス、実行)	13種類(アクセス、実行)
描画属性	25種類	22種類	21種類
実行形式	直接実行	間接実行	間接実行
変数記憶領域	フレーム(最低50個)	辞書，配列(任意個)	辞書，配列(任意個)
変数退避／復帰	手続きに同期 フレームと描画属性が対象	オペレータで行う 辞書，配列，描画属性が対象	オペレータで行う 辞書と描画属性が対象
描画要素	線図形，塗り潰し図形，文字， ラスタ画像	線図形，塗り潰し図形，文字， ラスタ画像	線図形，塗り潰し図形，文字， ラスタ画像
フォント形式	FIS	Type0,Type1,Type3等	SPDLの仕様に基づく
デフォルト座標系	左下原点 1m単位	左下原点 1/72インチ単位	左下原点 1mm単位

て重要ではないが，辞書型はPostScriptとSPDLでは重要なデータ型である。これは，キーと値の対が並んだ表であり，キーによる値の参照が行える。連想配列として使える他に，変数領域，あるいは構造体などの複雑なデータを表現するために用いられる。

Interpressには，論理型と文字列型もなく，論理型は，整数の0と1で，文字列は整数の配列として表される。また，Interpressの配列は書き込み禁止であり，読み書きできる領域としてはフレームと呼ばれる大きさ固定の配列が一つあるだけである。

また，PostScriptとSPDLでは一部のデータ型にアクセス属性および実行可能属性が付けられているが，Interpressにはない。

逆に，PostScriptとSPDLは描画に関連するデータを，SPDLのPath型を除いて，独立したデータ型としては持っていない。これらは，表2に示したとおり，辞書型，配列型などを用いて表現されるか，あるいは処理系の内部変数となっていて，ページ記述では直接表現されない。

2.2 プログラムの実行

PostScriptとSPDLは，辞書のスタックを持っている。これらの辞書にキーと値の対を登録することにより，変数や手続きを定義できる。オペレータの実行は，この辞書のスタックを参照して行われる間接実行である。すなわち，デコンポーザは，実行すべき識別子に出会うと，まずそれをキーとして辞書を検索して値を取得する。そしてその値を実行する。したがって，オペレータ名と同じ名前の手続きを定義すれば，オペレータ名を再定義できる。

Interpressではオペレータ名は予約語で，定義を変更できない。実行すべき識別子がオペレータ名であると，直ちにそのオペレータが実行される。これは，直接実行と呼ばれる。

2.3 メモリ管理機構

ページ記述言語は，データの退避と回復の機能を持っているが，データの記憶領域の取り方とその退避回復の方法がInterpressと他の二つの言語では異なっている。

Interpressではフレームと呼ばれる固定長の領域にのみデータを記憶でき，ページ記述から一度に見えるフレームは一つだけである。PostScriptとSPDLは，辞書や配列にデータを記憶でき，その個数や大きさは可変である。

Interpressでは，データ領域の退避と回復は手続きの実行時に自動的に行われるが，他の二つの言語では任意の時にオペレータによって実行できる。

2.4 画像記述能力

描画要素とそれに対する操作は三つの言語とも基本的には同様であり，イメージングモデルも描画能力もほぼ同等である。描画要素としては，いずれも2次元のベクタ図形，文字およびラスタ画像を持つ。その属性や可能な操作もほぼ類似している。

違いとしては，PostScriptとSPDLが文字の輪郭形

表2 各PDLのデータ型

	Interpress	PostScript	SPDL
論理型	(整数型で表現)	boolean	Boolean
整数型	Number	integer	Integer
実数型	Number	real	Real
識別子型	Identifier	name	Identifier
演算子型	Operator	operator	Operator
配列型	Vector	array, packed array	Vector Reference
文字列型	(配列で表現)	string	Octet String Reference
辞書型	-	dictionary	Dictionary Reference
ヌル型	-	null	Null
マーク型	Mark	mark	Mark
ベクタ図形	Trajectory, Outline, Clipper	(内部状態)	Path Reference
描画状態	(内部状態)	graphics state	(内部状態)
フォント	Font	(辞書型で表現)	(辞書型で表現)
座標変換行列	Transformation	(配列型で表現)	(配列型で表現)
色	Color	(辞書型で表現)	(辞書型で表現)
ラスタ画像	Pixel Array	(文字列型で表現)	(文字列型で表現)
手続き	Body	(配列型で表現)	(配列型で表現)
セーブ型	(自動的に行われる)	save	Saved State Reference
外部入出力	(自動的に行われる)	file	Stream Object

状を，ベクタ図形として取り出せること，色変換処理，中間調処理などのパラメータを制御できることなどがある。その他の違いとしては，フォントデータの形式や座標系の単位などがある。

2.5 文書構造

ページ記述言語で書かれたプログラムの実行によって，デコンポーザの実行環境 (スタックの内容，描画パラメータの値など) は変化する。この実行環境は，プログラム実行の開始時や各ページの処理開始時に初期設定する必要がある。文書構造は，主としてこの初期設定や処理内容を規定するために用いられる。

PostScript は文書構造を持っていない。したがって，各ページの処理開始時に再初期設定が行われないため，PostScript ではページごとの独立性が保証されない。

Interpress の文書構造は図1に示すようにブロックとボディから構成される。ブロックは，0 個以上のブロックと1つ以上のボディから構成され，ボディが文書構造の最小単位である。ブロックの最初のボディをプリアンブル (図1中の太枠部分) と呼び，それに続くボディをページボディと呼んで区別している。プリアンブルには環境初期設定などの記述が，ページボディには1ページ分のページ記述がある。Interpressでは，各ページボディを実行する前に，プリアンブルを実行するので，このプリアンブルによってページごとの初期設定が行える。

SPDLの文書構造を，図2に示す。他のピクチャに含まれていないピクチャ (図2中の太枠部分) がページを表現している。つまり，SPDLではページより小さい文書構造単位が存在する。SPDLの文書構造は複雑なため，文書構造処理部と文書内容処理部を分離して処理するモデルが規格案に示されている。

3. デコンポーザの設計

3.1 全体構成

前章の検討に基づいて，以下の設計方針を立てた。

(1) デコンポーザを，各ページ記述言語処理系が共通に使う部分と言語に依存する部分に分ける。

(2) 文書構造の解釈，プログラムの字句の切り出しとその解釈実行系は言語によって大きく異なるため言語依存部として言語ごとに作成する。

(3) ページ記述言語に必要なデータ型は，共通しているものが多く，また共通部分と言語依存部のインタフェースとして用いるので共通部分で提供し，言語間のメモリ管理方法の違いを吸収する機構を設ける。

(4) 描画機能は，どの言語でもほぼ同じであるので，全て共通部分が提供する。

図3は，上記の方針に基づいて設計したデコンポーザのブロック図である。共通部を四つのサブモジュールに分けたので，デコンポーザは以下の五つのモジュールで構成される。

(1) 解釈実行モジュール (言語依存部)

ページ記述言語の文書構造と文書内容を解釈実行し，言語データモジュールと描画モジュールを呼び出して，必要な演算，描画を行う。

(2) 言語データモジュール

ページ記述言語の基本データ型とその演算を解釈実行モジュールに提供する。データの操作とメモリ管理が主な機能である。

(3) 描画モジュール

デバイスから独立した描画処理を行う。ベクタ図形，文字，およびラスタ画像などを表現するデータとその操作を解釈実行モジュールに提供する。

(4) フォントモジュール

文字を出力するために，アウトラインフォントデータの展開，文字の輪郭抽出およびフォントデータの管理機能を提供する。また，高速化のためにフォントのキャッシングを行う。

(5) 出力変換モジュール

描画要素データを，共通のデータ構造に変換し，次にこのデータ構造から各出力デバイスに適した出力データ形式に変換する。

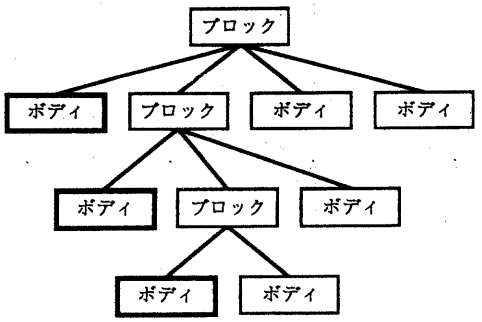


図1 Interpressの文書構造

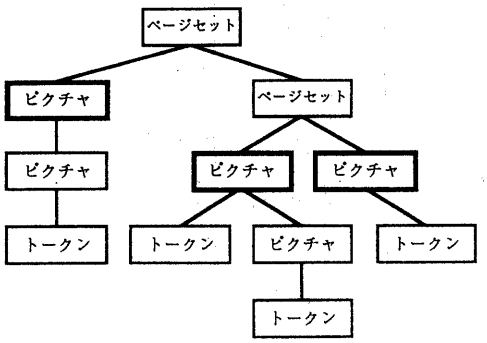


図2 SPDLの文書構造

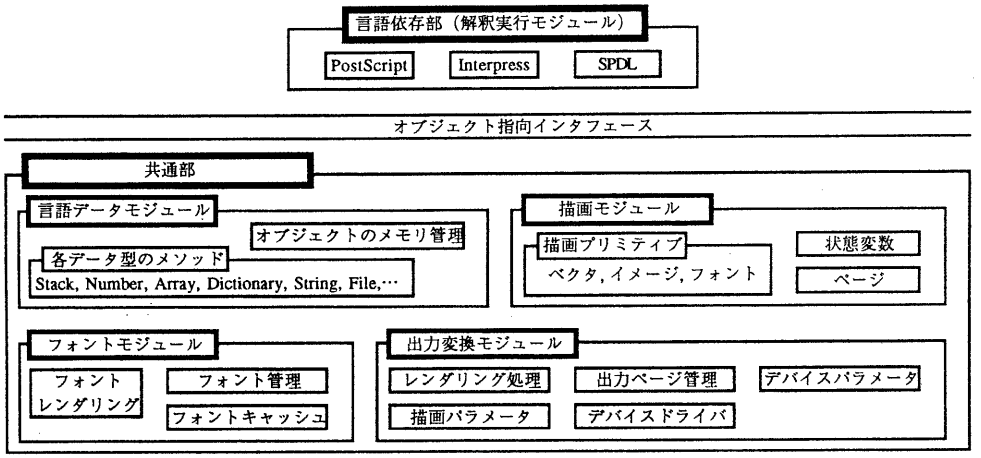


図3　システムのブロック図

3.2　オブジェクト指向インタフェース

3.2.1 オブジェクトの設計

各ページ記述言語は表2に示したデータ型を持っており，それらに対する操作はオペレータとしてそれぞれの言語に定義されている。これらのデータ型およびオペレータを検討した結果，本システムでは，オブジェクト指向[8) 9)]の概念を用いてページ記述言語のデータを表現し，これを言語依存部と共通部のインタフェースに用いることにした。理由は，以下の通りである。

(1) ページ記述言語のデータ型が高度な機能を持った複雑なものである。

(2) 変数領域として用いられるスタックや配列の要素には型の概念がなく，任意の型が混在することができること。すなわち，すべてのデータ型を同等に取り扱える必要がある。

(3) オペレータのオーバーローディングがあり，一種のオペレータが複数種のデータ型に対して定義されているものがある。したがって，すべてのデータ型を統一的に扱える必要がある。

オブジェクトの種類は，以下の通りとした。

(1) 単純オブジェクト

整数型：	Integer	実数型：	Real
識別子型：	Name	演算子型：	Operator
ヌル型：	Null	マーク型：	Mark

(2) 複合オブジェクト

配列型：	Array	文字列型：	String
辞書型：	Dictionary	スタック型：	Stack
ベクタ図形：	Path	ラスタ画像：	Image
外部入出力：	File	色：	Color
座標変換行列：	Matrix		

単純オブジェクトは，値を短い固定長で表せるオブジェクトであり，複合オブジェクトは内部構造を持つような複雑なオブジェクトで，値を表現するのに可変長の領域を必要とするオブジェクトである。

3.2.2 オブジェクトの実現

本システムにおけるオブジェクトは，固定長の構造体Packetで表現した(図4)。Packetは，オブジェクトの型，値およびその他の属性を保持する。単純オブジェクトの場合は，値のフィールドには値そのものが入り，複合オブジェクトの場合は本体を指すポインタを値フィールドに入れ，本体は別の領域に格納する。配列，辞書，スタックなどはこのPacketの並びとして実現する。

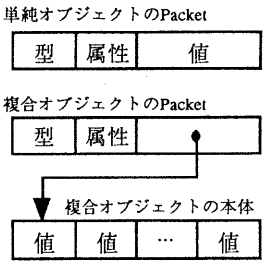


図4　オブジェクトのデータ構造

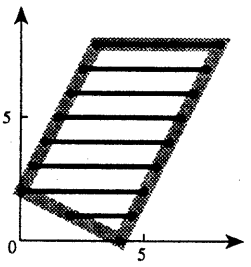
Packetの型のフィールドには，上記のオブジェクト型を表す値が入る。属性のフィールドには，PostScriptとSPDLのデータ属性(アクセスと実行可能)とメモリ管理用の属性が入る。後者は，解釈実行モジュールと言語データモジュールが使用する。

オブジェクトに対する操作であるメソッドは，関数を用いて実現する。関数名をメソッド名として使用し，操作対象となるオブジェクトを第一引数として取り，その他の引数は第二引数以降に取ることとした。

一般のデータ型に対応するオブジェクトとそのメソッドは言語データモジュールが提供し，描画に関するオブジェクトとそのメソッドは描画モジュールが持つ。

3.3 エッジリスト

出力変換モジュールは描画要素オブジェクトを出力装置のデータ形式に変換する。この処理はレンダリング[10]と呼ばれている。本システムのレンダリング処理では，ベクタ図形，文字，ラスタ画像など全ての描画要素を，まず共通のデータ構造に変換することにより，その後の処理の共通化を図った[11](図5)。



	始点	終点	色の増分	色値ポインタ
8	3.0	8.0	0	p1
7	2.5	7.5	0	p1
6	2.0	7.0	0	p1
5	1.5	6.5	0	p1
4	1.0	6.0	0	p1
3	0.5	5.5	0	p1
2	0.0	5.0	0	p1
1	2.0	4.5	0	p1
0	4.0	4.0	0	p1

図5　エッジリストの例

この共通のデータ構造をエッジリストと呼び，図5に示すように，出力画像の主走査線ごとの色の変化点を示す2つの座標値(始点,終点)および色情報から構成される。色情報は，さらに，色データポインタと増分の二つの要素で表現され，エッジリストをデバイスの描画バッファメモリに最終的に展開する時に参照する。色データポインタは，そのエッジリストの始点の色データを表す。始点の次の画素の色は，色データポインタに増分を加えた値の指す色データである。したがって，色が変化しない場合は増分は0である。また，ラスタ画像などの場合には，拡大率に合わせた値を増分に設定しておくことによって，画素ごとの色データを指し示すことができる。これにより，通常のランレングスによる表現と違って，ラスタ画像データを微小なエッジリストの集合として表す必要がなくなる。

この方式の利点は以下の通りである。

(1) 図形と文字中心の文書の場合，ページメモリに直接展開する方式と比べて，大幅にバッファメモリを削減できる。

(2) データ構造の共通化により，レンダリング処理が共通化でき，プログラムサイズを削減できる。

(3) 座標情報(外接形状)と色情報が分離されているため，レンダリング処理の最終段階まで色情報を扱う必要がない。この結果，白黒からフルカラーの対応のための変更点が少ない。さらに，クリッピング後まで外接形状だけを処理すれば良いので，ラスタ画像データの処理が高速化される。

(4) フォントも直接エッジリストへ展開するため，従来のビットマップで処理する方式よりも，色付けの処理が高速である。また，大きな文字を出力する場合，フォントキャッシュ用のメモリを削減できる。

4. 結果

4.1 実装

本デコンポーザはC言語で記述し，Argoss*3ワークステーション(SPARCstation*4)上に実装した。現在サポートしている出力先は，以下の4種類である。

(1) X Window
(2) 昇華型熱転写プリンタSC-60
(3) ゼログラフィ方式カラープリンタA color
(4) Sunラスタファイル形式

各モジュールのプログラムサイズを表3に示す。

表3　各モジュールのサイズ

	Source (1000行)	Source (Kbyte)	Object (Kbyte)
共通（ヘッダ）	1.5	47.7	-
解釈実行モジュール	35.4	1034.0	512
言語データモジュール	10.6	325.6	122
描画モジュール	16.2	480.7	147
フォントモジュール	20.9	626.5	242
出力処理モジュール	13.4	386.0	164
合計	98.0	2900.5	1187

表3から分かるように，デコポーザのソースプログラムの約2/3が共通化できている。なお，解釈実行モジュールのサイズはPostScript用のものである。

PostScript解釈実行モジュールは，現在，PostScript Level 1の約90%のオペレータを実装しており，本デコンポーザは，PostScriptで記述されたほとんどのプログラムを実行，出力可能である。これは，本デコンポーザの共通部が十分な描画能力を持っていることを示している。

4.2 性能評価

作成したデコンポーザをPostScriptデコンポーザとして性能評価した。実行時間と使用したメモリ量を，複数の評価用のサンプルPostScriptプログラムについて測定し，実行時間については他の処理系とも比較した。測定に用いた環境は次の通りである。

マシン	Argoss 5250 (SPARCstation 2 相当)
主記憶	48MB (スワップ 110MB)
Operating System	SunOS 4.1.3
Window System	日本語OpenWindows 2.0.1D
出力サイズ	A4 (210mm× 297mm)
出力解像度	72dpi (2.83 dot/mm)

PostScriptで記述できる画像は，単純な線画，複雑な塗りつぶしを繰り返すもの，文字だけで構成される文書，文書と図形が組み合わさったものなど多種多様である。PostScriptプログラムを収集し，以下の7種類

*3 Argossは富士ゼロックスの登録商標である。

*4 SPARC stationは米国SPARC International Inc.の商標である。

に分類し，それぞれから性能評価用サンプルを選んだ。プログラムの分類は各サンプルのPostScriptオペレータの実行回数に基づいて行った。

(1) 線画(線だけで構成される図形)
(2) ペイント(線と領域の塗りつぶし)
(3) グラデーション(連続する領域を色を変えながら塗りつぶす)
(4) カラー画像(デジタル画像)
(5) 欧文文章(欧文文字だけの文書)
(6) 和文文章(和文文字だけの文書)
(7) 組み合せ文書(文字，図形からなる文書)

4.2.1 処理速度

本デコンポーザの処理速度を他の処理系と比較するために評価用プログラム内に時間を測定するPostScriptプログラムを加えて，実行時間を測定し，他の処理系と比較した。比較した処理系は，GhostScriptとOpenWindowsに付属のPageViewである。測定結果を表4に示す。

表4　処理系別実行時間（単位：秒）

	本システム	GhostScript Ver 2.2	PageView	本システム / GhostScript
線画	7.12	2.18	1.31	3.26倍
ペイント	19.45	5.01	2.38	3.88倍
グラデーション	1844.20	159.86	340.81	11.54倍
カラーイメージ	23.08	369.17	25.79	0.06倍
欧文文章	185.68	67.72	10.11	2.74倍
和文文章	113.64	48.25	-(注1)	2.36倍
混合文書	153.72	41.79	-(注1)	3.68倍

注1) 日本語フォントがないため測定せず

多くのテストプログラムについて，本システムの方がGhostScriptより遅くなっている。この原因を明らかにするために，モジュール別の実行時間を調べた。評価用サンプルプログラムを実行し，各モジュールごとの実行時間をCコンパイラのprofileオプションを用いて測定した。この結果を図6に示す。なお，図中で“システム”と記したのは，オペレーティングシステムなどの消費時間である。

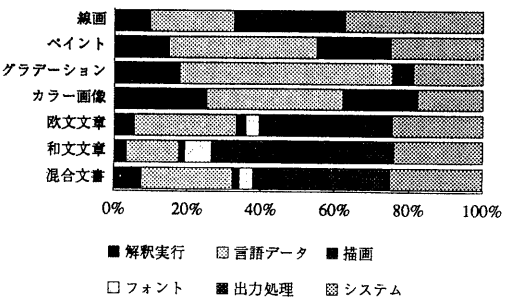


図6　実行時間の内訳

図6を見ると，言語データ部が大部分の処理時間を消費していることが分かる。さらに，詳しい検討の結果，これは，オブジェクトの実装方法，特にオブジェクトのメモリ管理に問題があることが分かった。

4.2.2 メモリ使用量

次に，各モジュールのメモリ管理ルーチンに測定機構を追加し，各モジュールごとに使用しているメモリを測定した。メモリの使用量の測定に用いた評価用サンプルプログラムは，実行時間の測定に用いたプログラムと同じものである。この結果を図7に示す。

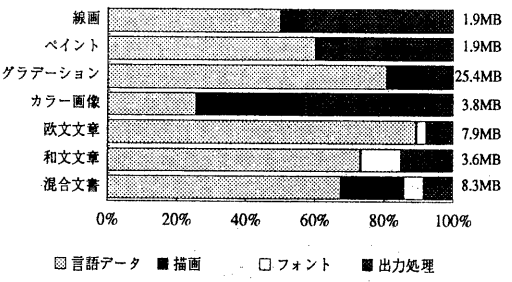


図7　メモリ使用量の割合

結果として，メモリ使用量についても言語データ部が多くを占めていることが分かる。複雑なプログラムになるほど言語データ部のメモリ消費が多くなっている。

4.3 言語データ部の改良

システムの解析により，言語データ部の問題が明らかになった。そこで，処理速度を向上させるために，次の方針で言語データ部の改良を行った。

(1) 言語データ部の提供する機能と外部インタフェースの見直し

言語データ部の提供する機能が低レベルすぎたため，関数の呼び出し回数が多かった。また，外部インタフェースの引数のほとんどがPacketであったため，データ変換に要する時間が大きかった。

(2) メモリ管理部の導入

各モジュールごとに行っていたメモリ管理を統合し，システム全体でのメモリ使用効率を上げる。

この方針に基づいて，言語データ部と言語解釈部をC++で再実装し，言語解釈部の核となる部分の速度を計測した。計測したのは，次の5種類のプログラムで，これは，ページ記述によく使われるものである。(2)を除いて，100万回のアクセスを行っている。

(1) スタックのpush, pop

1 popを100万回繰り返す。

(2) 名前の検索

各処理系が持つ共通の名前200個を1000回繰り返して検索する。

(3) 辞書への書き込み

大きさ1000の辞書へ異なる1000個のデータを1000回書き込む。

(4) 配列からの読み出し

大きさ1000の配列のすべての要素を1000回読み出す。

(5) 配列への書き込み

大きさ1000の配列のすべての要素に同じデータを1000回書き込む。

これらのプログラムの実行時間を測定し，他の処理系と比較した。この結果を図8に示す。

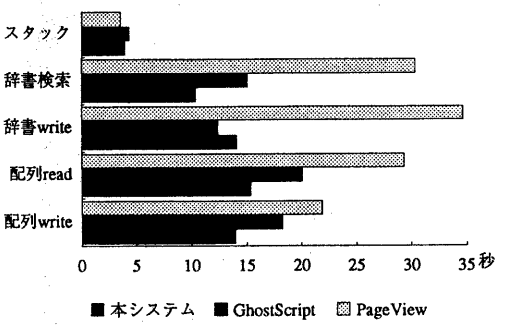


図8　改良後の測定データ

図8において，配列の読み出しより配列の書き込みのほうが速いのは，配列の読み出しによりスタック上にデータが置かれるため，それをpopしているためである。

辞書の書き込みを除いて，GhostScriptより早くすることができた。

5.むすび

複数のページ記述言語に対応するデコンポーザを設計，実装，評価した。デコンポーザのサブモジュールを複数の解釈実行系で共通化することにより，少ないプログラムサイズで複数のページ記述言語に対応できた。また，エッジリスト構造の採用により，新たな出力装置への対応は，エッジリストを出力装置に適したデータ形式に変換する部分だけでの変更ですむため，出力装置の追加も容易である。共通部における実装上の問題により処理速度が他の処理系に比較して遅かったが，問題点を解決した結果，高速な処理系を作成できた。

参考文献

1) Xerox: “Interpress Electronic Printing Standard, Version 3.0”, Xerox Corp. (1986).

2) Harrington, S.J. and Buckley, R.R.: “Interpress: the Source Book”, Brady (1988).
富士ゼロックス(株)訳: ”インタープレス”, 丸善 (1989).

3) Adobe Systems Inc. : “PostScript Reference Manual, SECOND EDITION”, Addison-Wesley (1990).
アドビシステムズジャパン監訳: “PostScript リファレンスマニュアル 第2版”, アスキー出版局 (1991).

4 Adobe Systems Inc.: “PostScript Language Program Design”, Addison-Wesley (1988).
松村邦仁訳: “PostScript プログラム・デザイン”, アスキー出版局 (1990).

5) Adobe Systems Inc.: “PostScript Tutorial and Cookbook”, Addison-Wesley (1985).
野中浩一訳: “PostScript チュートリアル&クックブック”, アスキー出版局 (1989).

6) ISO: “ISO/IEC DIS 10180, Information Processing - Text Communication - Standard Page Description Language”, ISO (1991).

7) 伊知地宏, 窪寺隆行, 森田雅夫: “ページ記述言語に置けるプログラム変換”, 情報処理学会プログラミングー言語・基礎・実践ー研究会資料 (1993).

8) Goldberg, A. and Robson, D.: “Smalltalk - the Language and its Implementation”, Addison-Wesley (1983).

9) Cox, B.: “Object Oriented Programming”, Addison-Wesley (1986).

10) David F.Rogers : “Procedural Elements for Computer Graphics”, McGraw-Hill Book Company (1985).
山口富士夫監修: “実践コンピュータグラフィックス 基礎手続きと応用”, 日刊工業新聞社(1987).

11) 堀素史, 森田雅夫, 小野沢雄二: “PDL処理系におけるベクターグラフィックス, 文字, ラスター画像データの共通ラスタライズ方式”, 情報処理学会夏のプログラミング・シンポジウム「可視化」報告集 (1993).