ELECTRONIC PUBLISHING, VOL. 1(2), 79–96 (SEPTEMBER 1988)

Page description languages: development, implementation and standardization

A. L. OAKLEY
Department of Chemistry
Portsmouth Polytechnic
Portsmouth, PO1 2DT
UK

A. C. NORRIS
Department of Physical Sciences and Scientific Computing
South Bank Polytechnic
London, SE1 0AA
UK

SUMMARY

Advances in laser technology have facilitated the development of printers which accept input in the form of pages rather than the lines characteristic of impact printers. Concurrently, page description languages (PDLs) have been designed to facilitate the integration of complex text and graphics for printing. PDLs are described in a variety of ways and the paper starts with a rationalization of these viewpoints. It then traces the development of PDLs, describes their main characteristics, and looks at their relationship with laser printers. A survey of current implementations is then followed by an analysis of the relationship of these languages to other schemes for the description of printed images. Finally, the paper considers the requirements for a PDL standard.

KEY WORDS Page description languages Laser printers RIP Standards

INTRODUCTION

Non-impact laser printing, image processing and page description languages (PDLs) have all brought far greater flexibility and versatility to the graphics capabilities of computer publishing systems[1]. These techniques allow a complex mixture of text and graphics to be assembled on a page and printed at near camera-ready quality. Costs are decreasing rapidly and the ‘desktop publishing revolution’ is particularly beneficial where there are low-volume print runs or output is subject to rapid change, e.g. operator manuals. Previously, unit costs and publication times were much too high in such situations.

This paper describes the nature of PDLs and their relationship to laser printers. An overview is then given of the main PDLs, especially from a PC-based perspective, and they are compared to other approaches for the description of printed documents. Finally, the desirability and requirements of a PDL standard are discussed.

WHAT IS A PDL?

A page description language is variously described as:

- An interface between composition software and the output device[2].
- A program to communicate a description of a document from a composition system to a printing system[3].
- A software tool for composing graphical images on a page (including e.g. font, rotations, scaling, graphics)[4].

0894–3982/88/020079–18$09.00
Received 5 November 1987
Revised 26 March 1988

- A software solution to the transfer of logical (see later) and structural information from host processors to printers with sufficient abstraction to enable device independence[5].
- A method of communication between computers and printers providing sophisticated graphics, page layout and quality typesetting[6].
- A method for specifying exactly what a page should look like so that the same description can be rendered on a screen, laser printer or phototypesetter[6].
- A means of providing an on-screen page composition facility[1].
- A printer control language based on the page[1].

There are two main threads in these definitions of PDL function:

(i) *communication* of an already formatted page or document description from a composition system to a printing system;
(ii) the ability to *specify* images on an output medium (e.g. screen or page).

Arguably, those languages performing only the first of these functions are ‘printer control’ languagues and the term PDL should be reserved for languages encompassing both functions. Also, distinctions can be made between whole document description and page description languages and also between high-level procedural page description languages and low-level languages. For the purposes of this discussion, however, a loose definition of PDL is adopted; a PDL is taken to be primarily a printer control language based on the whole page.

The diversity of PDL definitions is best understood by looking at their evolution.

Development of PDLs

A screen-dump is the simplest way to transfer an image from a computer screen to paper; one dot (pixel) on the screen results in one dot on the printer and hence the printed image resolution is limited to that of the screen. Typical printer resolutions are currently 240 dots per inch (dpi) for dot-matrix, 300 dpi for laser and 1250–2500 dpi for phototypesetters, compared to 72–100 dpi for a typical ‘high-resolution’ graphics screen. Clearly, a screen-dump cannot fully utilize printer potential.

This potential is more readily achieved using a printer driver. For dot-matrix and daisywheel printers, manufacturers usually supply a set of codes which constitute a crude programming language. Typically, every instruction consists of a command character, usually *escape*, followed by codes peculiar to the printer. Clearly, drivers are neither user-friendly nor independent of the device they control, although they may allow extra control of output such as justification and kerning. An additional limitation is that drivers are line rather than page oriented. A page orientation is especially important for mixed text and graphics.

A further development, command-stream image description languages[7], describe images via commands to the controllers that generate the image: they are more compact and more easily edited than driver programs. They are also potentially more device independent, e.g. they use commands to move the printhead to line endpoints without the need to know device resolution. Most command-stream languages were developed from

XCRIBL (1972 Carnegie-Mellon). They can drive pen plotters and daisywheel and Imagen laser printers but they are still line oriented. Command-stream languages may be procedure based but they lack the control statements of a full programming language.

Only relatively recently has it become possible to achieve printed output which combines the typographical quality of typesetters and the graphics power of CAD–plotter systems. This integration has been made feasible by the availability of low-cost printer memory which can store a full page bit map before the image is translated into ink on paper. The evolution of page description languages has been crucial to this move away from line-oriented printers.

Some characteristics of PDLs

Different PDLs have different design priorities, as discussed later in this paper, but the procedural PDLs are currently the most fashionable. Warnock and Sproull developed the idea of using a procedural page description language with Interpress[8]. A procedural page description is a program written in a graphics programming language, which when executed, creates the intended page image. The language can be used to write loops and conditionals, to define and call functions, and to pass parameters such as the values of variables stored in the printer. An interpreter or compiler is required to translate the high-level commands into machine instructions[7].

Output devices vary not only in the codes used to drive them but also in their co-ordinate systems for addressing individual pixels and in their resolution and aspect ratios. To be able to drive a range of output devices, i.e. to be *device independent*, some PDLs describe pages of text and graphics in terms of abstract graphical entities (primitives) such as line, circle etc., rather than in terms of device pixels. They delay conversion to device specific code until the last possible moment such as when the page description is interpreted. The construction of a page description and the interpretation of that description may occur at different places and times. Whilst the interpreter may work on a range of printers, it will also be customised to exploit a given printer.

The graphic primitives are less bulky to transmit than bit maps and may be described in terms of user-defined natural units (‘real world’ co-ordinates) rather than device-dependent co-ordinates. Also, a library of procedures that draw commonly used primitive-based images can be developed. Where typefaces are represented as mathematical outlines instead of bit maps of the individual character shapes, geometric manipulation is facilitated and text is more readily integrated with pictures since the text itself is a graphic.

Page composition is normally achieved with an interactive page layout program such as Ventura (Xerox) which sends its PDL output directly to the printer without user intervention. However, some high-level PDLs can be used to program a page directly without composition software. This feature affords particular flexibility as discussed later in this paper.

Finally, the ability to produce an entire document description as opposed to a purely page description is important in a commercial environment where there are requirements to define fonts needed for whole documents or for intelligent spooling systems which can control page size, for example.

Since the PDL is often used in conjunction with a laser printer it is helpful to have some knowledge of these devices.

LASER PRINTERS, CONTROLLERS AND RASTER IMAGE PROCESSORS (RIPS)

The first commercially available laser printers appeared in the mid 1970s but were text oriented and very expensive (>$500,000). Canon introduced one of the first cheap laser printers (LBP 10) in 1979, its cost being an order of magnitude less than that of similar machines[9]. Since then performance has increased whilst prices have plummeted. Many available printers rely on either a Canon or Ricoh 'print engine'[10] which determines dot resolution, page speed, paper handling and life span. The other main laser printer component is the controller which uses command sequences to control the print engine. Controllers may be 'dumb', i.e. rely on host processing of documents, or 'intelligent' i.e. a processor chip is housed within the printer with local memory.

A few suppliers offer controllers on more than one print engine and therefore printers may be installed on a price/performance basis and documents interchanged between different print engines without reformatting.

In some systems fonts may be user-defined or down-loaded from a host computer whilst in others they are either embedded in ROM (read only memory) on the controller or in a ROM plug-in cartridge. The latter approach may necessitate the purchase of unnecessary fonts whilst the former approach may result in duplication of fonts or inconsistent font versions. Also, the time taken to download a set of fonts, probably in the order of minutes, can be a drawback.

The first laser printers, based on the Canon print engine, supported down-loadable fonts but they lacked adequate graphics capabilities. In 1981, Imagen introduced the 'page cell-based' laser printer. A page was considered to be subdivided into cells each of which could be composed independently, thus avoiding representing a whole page at once and conserving costly memory. However, such printers could still not print complex graphics.

Computer systems traditionally send page description data to impact printers in character code and at a transfer rate determined by the host[1]. Similarly early laser printer controllers typically stored one line of characters at a time. Unlike impact printers, however, modern laser printers require 'dot-stream' information at video speeds (more than one million pixels per second). The page description must therefore be decoded and translated into instructions for the printer and once printing has commenced the supply of information must be continuous. Also, these printers store a whole page bit map before printing and approximately 1.1 Mbytes of memory are needed to represent an A4 page as dots at 300 dpi. Hence, local memory and processing power are required. Falling memory prices have spurred the processing of increasingly complex images so that some recent laser printers have become more intelligent than their host computers[3].

In principle the processor and memory may be:

(i) built into the printer;
(ii) located in a separate Raster Image Processor (RIP) unit interfaced to the printer;
(iii) installed as a RIP on an interface board for a personal computer (PC) slot;
(iv) those of the host microcomputer system running the PDL interpreter as normal software.

The first of these alternatives is probably the most common approach at present. Where the second method is used, the input to the RIP, probably in the form of a PDL, is translated by a ROM-based interpreter into a page bit map held in local random access memory (RAM). The data are then output line by line through the video-output section and the raster signal is sent to the print engine to control the laser which places the dots on the paper. The entire operation, including the timing of data input, the composition of the bit image and output to the printer, is under the control of an on-board processor. Such processing of whole-page bit maps enables the printing of complex graphics and integrated text composed of proportionally spaced fonts with full kerning etc.

Each character of a font may be stored in ROM either as a fully formed bit map or as a mathematically defined outline. A printer controller normally deals only with characters and bit maps; the ability to deal with vectors and mathematically defined shapes generally resides in a RIP. When a character is to be printed it is selected from font memory and, if in outline form, it is translated into a bit map. The character bit map is then written into the page buffer bit map and the buffer contents are eventually output to create the complete page. Typically, the ROM containing both PDL interpreter and font definitions has a capacity of 1 Mbyte and the RAM has a minimum of 1.5 Mbytes, consisting of 1.1 Mbytes for an A4 page bit map and the remainder for temporary storage etc.

It is extremely difficult to edit a bit-mapped description. Bit maps cannot be scaled because resolution is lost and aliasing or gaps will appear. Thus, for characters stored directly in bit-map form, a separate bit map must be held for each typeface and point size. These image descriptions therefore take an enormous amount of storage space and are machine specific.

In contrast, where a font is stored in outline form, attributes such as size, resolution, colour are specified as parameters and computed for each page. This method is therefore more processor and time intensive but allows font size and resolution independence as well as geometric transformations such as rotation. An outline font description is much more compact than a bit map and so a greater variety of fonts may be stored in font ROM.

The RIP unloads the task of creating dot-stream information from the host processor and also performs control tasks such as page merges. For example, a form layout may be stored in memory and variable data may be sent from the host and the form and data merged. This avoids the need to produce, store or load pre-printed forms[11].

Naturally, the sophistication of RIPs is reflected in their cost which can be a substantial fraction of the whole printer cost.

RIP design is currently an area of much research not only to decrease costs but to improve performance. For example, memory requirements can be reduced by dynamic memory allocation[12], a ‘blitter’ (a ROM for bit block transfer) may be used to speed the time-intensive page bit map creation, and a second page can be composed whilst the first is being printed. Also, parallel operation transputers[13] may be used to speed the processing of outline fonts and high-speed interfaces employed to accelerate the transfer of data from host computer to printer[11]. It is also possible to have a generic RIP with more than one interpreter to process a range of PDLs and control any print-engine[14]. Other developments in page printer technologies and some recent alternatives to laser printers are reviewed in a paper by Cook[15].

In summary, the general approach in current desktop publishing involves document editors and composition software on a host computer which produces a document file[9]. The document file is then translated into the PDL for the target printer. The PDL commands are passed to an interpreter held on a RIP, which then translates the commands into the pulses of the laser in the printer, causing toner to adhere to paper to produce printed images.

Survey of current PDLs

This section presents a survey of the most important PDLs available at the time of writing. It describes and compares their features and indicates their principal advantages and disadvantages. Naturally, certain features of a given language will have both advantages and disadvantages depending upon both the application and the user's perception.

PostScript (Adobe Systems Inc)

PostScript has its origins in research carried out by Xerox in the late 1970s at their Palo Alto Research Centre. The first PostScript-based laser printer, the Apple LaserWriter, became available in the spring of 1985.

This page description language is rapidly becoming the *de facto* standard, partly because it was designed and promoted as such by Adobe and partly because it was the first high-level procedural PDL generally available. It is also very powerful and flexible[4].

PostScript is supported by many packages and printers, e.g. Microsoft WORD on the IBM PC with output to the Apple Laserwriter and Linotype typesetters (100 and 300 series), and page composition software such as Pagemaker (Aldus) and Ventura (Xerox). Typesetting software such as $\TeX$ and Micro$\TeX$ (Addison-Wesley), and JustText (MacEurope) also interface to PostScript. IBM recently adopted PostScript for their desktop publishing systems.

PostScript is in two parts; the Programming Language (to which most articles relate) and the Interpreter. A PostScript image or page description is a program expressed in the PostScript language[6]. These programs are created, transmitted and interpreted in the form of ASCII source text which is largely device-independent. The page description is a sequence of high-level commands, variables, strings and procedures[7]. The interpreter executes the program by manipulating a stack of procedure calls which manage other stacks containing, for example, operands and dictionaries of variable and font names. Graphics states are held in global variables and describe ink colour, transformations, current position etc. The graphics themselves are normally handled as vectors and splines although bit maps may be specified.

As execution proceeds the interpreter's painting or imaging functions use the interpreter's graphics state variables to calculate where dots should be placed on a page and set the corresponding bits of a page bit map stored in the printer controller memory. In the program source text a complete image description is set up before the command is given to draw and a complete page description is set up before the command is given to print a page. Thus, at interpretation, the output function instructs the printing mechanism to process the raster image and mark the paper with the ink dots only when the page is complete.

The language comes with four font families as standard—Times-Roman, Helvetica, Courier and Symbol—and each family comprises normal, italic, bold and italic-bold typefaces. The symbol font contains Greek and mathematical symbols. Also, expert users can construct their own fonts.

Each character in a typeface is defined as a PostScript procedure which is executed to produce the character's shape. The different font sizes are created by scaling this outline description held in one point size. Proprietary algorithms take account of the conventional typographical changes in appearance of each character at different point sizes. The scaled font is then converted to a bit map and stored in a temporary 'cache' memory. Characters are then acessed from this cache. This approach is device independent, conserves memory, since only one copy of a typeface is stored, allows sharp resolution at any point size and avoids time-consuming recalculation except when the typeface or size are changed. The amount of memory assigned to the cache can be customized to balance storage, speed and image resolution[16].

PostScript has enjoyed favourable reviews and is the best known of the current PDLs. The main advantages and disadvantages are summarized below[17,18]. Not all the following features are exclusive to PostScript and certain features, such as ASCII encoding, have both positive and negative points.

Advantages

- Source code is readable; many commands are obvious.
- Text and pictures are all graphic objects.
- Powerful program control including normal constructs such as loops, procedures and conditionals and unusual features such as dictionaries (associative tables). For example, when a procedure is defined its name is added to a dictionary with the procedure body as the meaning.
- There is a powerful range of operators to e.g. scale one or both axes, rotate, oblique, kern, orientate strings, justify, define line thicknesses and styles etc as well as carry out mathematical operations.
- Fonts may be modified or user-defined using splines or bit maps.
- One PostScript font master will generate the same character on any raster laser printer, at any point size or orientation.
- A word, letter or image may be placed anywhere on a page.
- A compact character format significantly reduces font storage problems in printers (from 100 bytes per character; a complete alphabet of 209 characters with special symbols and diacritics for international language typesetting is stored in 15–20 Kbytes).
- The programmer can control the temporary memory store where scaled characters can be held for reuse without recalculation.
- Half-toning is available to produce the illusion of gray levels[19], and colour images.
- A page can be represented as a set of procedures and hence libraries of procedures may facilitate construction of a compact page description.
- Sections of a page can be geometrically transformed independently of each other.
- A page image can be redefined as a figure; the page description is named as an operator and used to achieve effects such as nested pictures or several copies of this 'page' on one page[7].
- Since no printing occurs until a page in printer memory is complete, the host is freed for other processing and it is also possible to alter the page before it is printed.
- PostScript source code can be automatically generated by a range of software such as CAD, word-processing and page make-up packages. It may also be created directly using a word-processor.

- The same source code drives a range of printers and typesetters and can also drive a screen, providing these devices incorporate a PostScript interpreter.
- Error handling is comprehensive; there is type and range checking and error messages are quite specific[19].
- The use of a powerful and human-readable procedural language opens up many creative possibilities (see below).

Disadvantages

- The language is stack oriented and the resulting postfix notation makes it difficult to read and write mathematical expressions.
- Some terse commands diminish readability.
- It is easy to write inefficient programs.
- The time taken for syntax interpretion and for processing produces slow printing rates, maybe several hours for complex graphics[9].
- Use of ASCII rather than more compact code slows communications (see comments on DDL below).
- A purely page description rather than a document description is produced.
- There are no printing instructions such as paper colour etc. Adobe have recognized this failing and have introduced a PostScript ‘comment convention’. A PostScript programmer can indicate these parameters as comments, which are ignored by the PostScript interpreter, but are actioned by a separate interpreter which in turn, ignores the normal PostScript commands as if they were comments.
- The number of gray levels is too low, giving too coarse a result[19].
- There is no facility to make a PostScript program modular, hence source files can be very long and difficult to maintain[19].
- Costs are significant, e.g. licensing fees for the printer-based interpreter are high and the price of a PostScript screen controller is much greater than for other display adapters.

Interpress (Xerox)

Interpress was used internally by Xerox before it was publicly released in 1984. Although very similar to PostScript with origins in the same environment[20,21], it is less widely implemented[20].

Both PostScript and Interpress assume that the printer contains an interpreter which translates the page description program to print a page. Like PostScript, Interpress integrates text and graphics and allows the co-ordinate system to be user-defined. However, a complete implementation and language description of PostScript are readily available whereas the implementation of Interpress is limited and, at the time of writing, documentation can only be obtained by special order from Xerox[7]. Nonetheless, whilst Adobe charges high licensing fees for their PostScript interpreter, Interpress is free of any of such fees. Also, Xerox have a policy of open architecture[21].

The design philosophy for several PDLs is that since the printing process must be transparent to the system user, a PDL is intended for creation and interpretation by software only and hence Interpress, unlike PostScript, encodes commands in binary format for compactness and speed. It is therefore more difficult to write programs in Interpress than in PostScript. However, utility programs exist to translate the binary code to and from characters thus aiding debugging and maintenance by programmers.

A further distinction concerns access to variables which is by address with Interpress and by high-level variable names in PostScript. Direct addressing is more efficient since no look-up tables are referenced when the program is interpreted but it is much harder to program. Programming in PostScript is also easier because its error handling and recovery are more clearly defined.

The two languages also differ in their treatment of character information. The Interpress standard does not refer to fonts whereas PostScript has a powerful font strategy using mathematically defined outlines. In Interpress, characters are not generally bit-mapped, although bit-mapped images are supported for 300 dpi printers. Although much memory is required for the bit maps, computational time is rapid and very high quality output can be achieved. Because of its speed Interpress may perform better than PostScript with very high-speed printers.

Interpress is a tokenized language within which documents are represented in a tightly encoded form. Each variable is preceded by a length field so that the interpreter knows in advance how many data the current token needs to read from the input stream. This technique facilitates speedy and reliable transmission and printing of documents over a network without the need for recomposition[9]. Hence, Interpress is better suited than PostScript for network applications although Interpress files are unsuitable for transmission over character-protocol lines. In contrast, PostScript program files are intended for transmission over character-protocol lines although they can also be transmitted over Ethernet. A PostScript file is generally larger than the corresponding Interpress file[20].

Unlike PostScript, Interpress is well-suited to multipage document preparation and allows rapid repagination[4,20]. An Interpress file consists of a series of independently executed 'bodies' so that commands for one page will not affect others and pages can be printed in any order. A PostScript file achieves similar results by the optional use of two operators (SAVE and RESTORE). Again, unlike PostScript, Interpress can issue handling instructions, e.g. how paper is to be stacked in the printer, which are useful for high throughput jobs[21].

In summary, Interpress and PostScript are based on quite different philosophies[20]; the former offers powerful but rigidly prescribed facilities whereas the latter allows the user more freedom and creativity, and a rope to hang himself!

DDL (Document Description Language) (Imagen)

DDL[1] developed from imPRESS (see below) and Imagen are developing a plug-in RIP for the IBM PC to drive a Hewlett-Packard Laserjet printer. This will be of interest to the numerous existing users who already have the Laserjet's proprietry printer control language (PCL), which has DDL compatibility. In future, DDL may be supported by Pagemaker.

Arguably, DDL has the potential to rival PostScript. In some areas the designers of DDL sought to avoid what they regarded as the bad design features of both PostScript

[1] Not to be confused with general document description languages to be discussed later.

and Interpress. For example, DDL can produce both a user-readable and a tightly encoded form of source code. The former allows rapid application development whilst the latter allows fast host to printer communication. Also, DDL addresses the processing of full documents, has high performance, and a high degree of functionality including the graphics richness of PostScript.

imPRESS (Imagen Corporation)

imPRESS is a comparatively old (1982), low-level language with a non-character code[1]. It has limited macros, a restricted set of graphics commands and no geometric transformations apart from linear scaling of a whole page. Most objects are bit-mapped and the language is page rather than document oriented. However, there is a separate printer control language[22].

imPRESS was designed to drive early laser printers with relatively little or no onboard memory and, in use, tied up the host. Nonetheless, imPRESS is still used by Interleaf on a number of workstations and it has been a standard in the documentation and scientific and engineering markets. Also printing is faster than for PostScript. However, it is incompatible with PostScript and is likely to be superseded by DDL.

ACE (ASCII Coded Escapement Language) (Chelgraph (UK))

ACE is a low-level assembler-like language producing unreadable code in which escape characters are embedded between text characters. The character positions defined by these escape codes need not be in device pixels thus allowing the language a measure of device independence. The positioning is also relative e.g. to other characters. The main advantage of this relative approach is the resulting direct control of justification, kerning and letter spacing. Also, fonts can be stored as outlines, geometric transformations are possible and there is a good set of graphics primitives[22,23].

However, ACE has no ‘macro’ facilities and no procedural programming constructs. This is because ACE was designed mainly as an interface and therefore leaves all formatting to the composition software[23]. The question of pages or whole documents is therefore not relevant. ACE is very fast and quite powerful but portability and flexibility have been sacrificed.

The language was specifically designed for the company’s RIP which has a modular design allowing configuration to almost any raster device. Thus the ACE language depends on the Chelgraph RIP to control a raster device whereas the PostScript language depends on its interpreter. Currently, there are more raster output devices with a PostScript interpreter than with a Chelgraph RIP. However, ACE has been interfaced to a range of typesetting languagues such as Linotype’s CORA V, and word processors such as Microsoft WORD. Both the Digitek (Itex Graphix) and Mentor 512 (GB Techniques) typesetters are compatible with ACE.

Harris[23] points out that a large, more complex PDL probably requires more memory and processing power and suggests that it is likely that some device features will not be fully utilized. He argues for simplicity. “ACE provides . . . a simple set of basic operations which allow the higher layers of document preparation software to access device facilities through a consistent interface.”

EXPRESS (CSC International)

This was one of the first languages to have an interpreter resident in a printer. It was intended for word processor operators and therefore did not require programming techniques[10]. It uses English language mnemonics to control text and graphics formatting and may be used in conjunction with page-formatting commands from other languages.

Like imPRESS (see above), EXPRESS was used to drive early laser printers with relatively little or no on-board memory[3].

RIPrint (Interleaf Corp)

Like ACE, RIPrint was developed to take special advantage of a particular RIP[9] and the RIP itself is designed to include device and language independence. Thus, several PDL interpreters can be either resident on, or down-loaded to, the RIP so that documents can be directed toward many existing and future printers. RIPrint was also designed to be resolution dependent in order to gain speed; bit maps are down-loaded directly thus avoiding time-consuming translation of mathematical descriptions.

The RIP software is divided into the low-level RIP system interface and the high-level PDL interpreters. The interface provides communications and print engine interface services, memory management and access to functions such as blitter control. The PDL interpreters can then be hardware independent. This modular approach also eases ports to new RIP hardware and adaptation to new print engines.

RIPrint includes a range of graphics commands as well as controls for the printer. The commands are binary, not ASCII, and their compactness allows significantly faster communications compared to languages such as PostScript. The processing of whole documents is not addressed.

The paper so far has focused on the implementation aspects of page description languages. It is useful, however, to place these languages within the context of other schemes for the description of printed documents. This framework will clarify the role of PDLs in document preparation and lead to a consideration of the requirements of a PDL standard.

SCHEMES FOR THE DESCRIPTION OF PRINTED IMAGES

The advantage of viewing a document as a logical abstraction is that the description is free of implementation details such as typefaces, linewidths, data structures etc. The document can be regarded as a collection of logical objects which include, for example, the whole document itself, chapters, pages, paragraphs, lines and characters, as well as title, abstract etc. In contrast, layout describes how the objects are positioned and styled on a given page. Thus, a logo is centred at the top of a page; a heading is typeset with a particular font. Appropriate layout therefore visualises and underlines the logical structure which itself enhances comprehension of the content. Both logical and layout levels are hierarchical and apply to the same document content; they provide different but complementary views; they are autonomous but related.

Far from being a purely theoretical distinction the separation of structure and layout has important practical consequences. For example, an author can specify logical document components which may then be formatted automatically according to a ‘style’ sheet which determines fonts, position of headings etc. The layout of a document is then readily changed simply by changing the style sheet and can be restyled for viewing on different output devices. Documents, rather than individual pages, are also easily processed. However, where structure and layout are not distinguished, the changes to layout are only possible by changing the markup throughout the whole document. This disadvantage is offset partly by greater flexibility and control since there is no reliance on a limited set of pre-defined logical objects.

There are several computer-based models for the description of printed images. Some are best described as 'data structures' (e.g. ODA described below) whilst others are languages which are either executed directly (e.g. PDLs and the typesetting languages) or indirectly (e.g. SGML). The notion of a separate logical structure is least evident in the directly executable languages. Clearly, our previous discussion shows that PDLs are concerned only with layout and do not separate logical structure. To set the PDLs in their place in relation to the rest of the formatting world we shall briefly describe the evolution of ideas in this field by looking at the more important models and implementations.

Not surprisingly, the earlier languages mimic the traditional method of markup in which formatting commands are embedded in the text. Examples of such typesetting languages include TEX[24,25] and TROFF[24,26]. JustText[27] is a more recent addition designed to take advantage of PostScript. Whole documents may be processed with considerable control of page layout but there is no clear and full notion of an abstract, logical document structure quite separate from the layout.

These languages were originally intended for straightforward composition and to work with traditional text-oriented printers. Although very powerful, they are difficult to use, especially for the novice or casual user[28]. The language is often complicated and the juxtaposition of content and formatting commands makes either difficult to read. Also, since the whole document with the embedded commands must be processed, the system is batch oriented resulting in considerable delays between entering the document description and seeing the final result. This inhibits experimentation with different layouts and also makes the detection and correction of errors rather tedious. However, such systems do not rely on graphics devices and they also allow greater device independence. Also, there is the recent development of preview screens which allow non-interactive viewing of output on-screen, thus partially offsetting some of the disadvantages of delayed output. Additionally, such systems allow considerable control and very high quality output.

Although a batch implementation like TEX, SCRIBE[24] is an early attempt to define a 'declarative' markup language in which logical document structure is better discerned. It includes notions of chapter, page, paragraph etc and allows document types to be held in a modifiable database. Consequently, documents may be reformatted using style sheets with different fonts, centred headings etc.

At a higher level of abstraction, the Office Document Architecture (ODA) is an ISO (International Standards Organisation) standard[29,30] defined to avoid the proliferation of incompatible formal document representations. It is more of a model for document structure than a language and is not actually implemented. ODA takes account of representations in areas such as communications, mixed media (e.g. voice, video) and text and graphics. ODA specifies parallel descriptions of logical and layout structures for documents and further subdivides these into generic and specific structures. It defines formal relationships between levels and in particular relates logical and layout structures.

Implementations may be ODA subsystems. For example, it is possible to create ODA systems which address layout only or logical structure only. Unfortunately, these systems may then be incompatible descriptions of the same document! Other disadvantages of ODA include its size and problems in reflecting changes simultaneously at logical and layout levels.

The Standard Generalised Markup Language (SGML)[30,31] may be considered as an ODA subsystem. SGML results from accumulated experience in the devlopment of markup languages and follows from earlier work on GML. Like SCRIBE, it is a declarative markup language designed for users to enter markup ‘tags’ into a document. These tags define logical structure only and can lead to different effects on different devices. The tags can even be redefined. Unlike the typesetting languages, there is no provision to define low-level details. The translation from the SGML logical structure to the fully formatted document could be via a typesetting language such as TEX or a high-level procedural PDL such as PostScript.

Both interactive and non-interactive software for generating SGML documents are beginning to appear. Interactive versions are of particular interest in view of the shift in recent years towards ‘user-friendly’ software based on windows, icons, mice and pop-down menus.

Attempts are also being made to combine the advantages of interactive and batch markup systems[28,32].

The recent development of interactive editor/formatters has been rapid. Pagemaker was one of the first serious contenders in the desktop publishing sphere but gives little evidence of any separation of logical and layout levels. There is no option to use markup codes in text and no style sheets. There is no provision for whole documents and pages are processed one at a time. However, other recent products, notably Ventura, do allow embedded codes, provide style sheets and they process whole documents. Whilst these composition programs deal with screen representations, it is the PDLs which communicate with the printers.

The differences in approach of these schemes reflect not only the degree of abstraction but also the differing sets of activities addressed as summarised in Figure 1.

All schemes apart from the PDLs are to some extent concerned with composition. However, whilst certain high-level procedural PDLs such as PostScript can be used for page composition, most PDLs are concerned only with events after formatting is

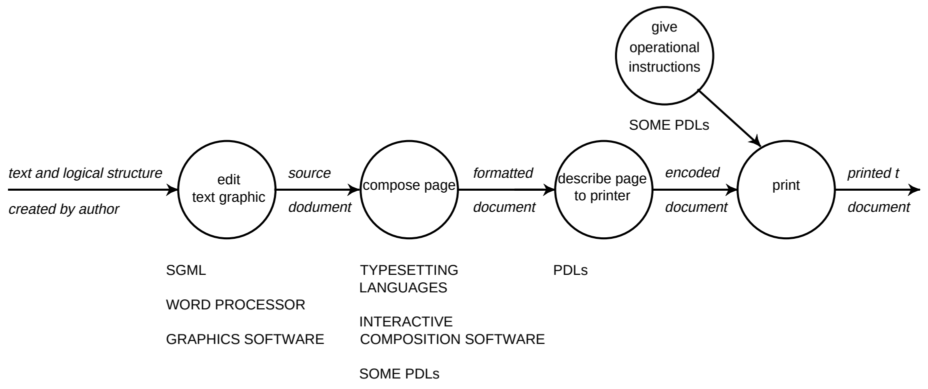

Figure 1. Activities to produce a printed page and related document description schemes

complete. A question thus posed by the PostScript type of PDL is to what extent a standard should encompass the page or document composition function and take account of logical structure.

TOWARDS A PDL STANDARD

Although as yet there is no agreed common standard, PostScript's rapid rise as a *de facto* standard for PDLs is due to two reasons. Firstly, it was an attempt to design a standard for the representation of a printed page[5] and, secondly, it was the first commercial implementation to approach market needs. However, in the longer term, there are other serious contenders such as DDL, ACE and Interpress.

Currently, the International Standards Organisation (ISO) is involved in the definition of a standard for page description languages and therefore it is pertinent to consider if a standard is desirable at all and if so, to clarify the ideal characteristics of a standard, taking existing implementations into account.

An ideal standard would allow perfect device independence and compatibility of systems. This ideal has been sought, but not achieved, in other areas. For example, with graphics application software standards such as GKS it it has been found that the great disparity of hardware devices makes true device independence virtually impossible[33]. Experience also suggests that it is extremely difficult to define one all-encompassing yet manageable standard for all time. Again, GKS was not designed for interactive graphics on microcomputers[34] whereas the later PHIGS standard does address this issue. Further, the domination of a standard may hinder innovation or lead to use in inappropriate situations.

Similar reservations are reflected in the debate as to whether a standard for PDLs is desirable at all. One argument against a standard is that PDLs need to vary greatly since each language will fill a niche, depending on application, system configuration, number of users, desired output etc[4]. Another objection[23] is that the principal reason for having a PDL is ease of interfacing and other features such as device independence are secondary. Indeed, adherence to an abstract standard may prevent full advantage being taken of the features of a given output device. For example, a RIP design with some device dependence can lead to clear advantages such as greater data compression and transmission rates. Thus, the information needed to drive a 2000 dpi typesetter need not be transmitted to a 300 dpi printer; faster transmission is especially advantageous when the communications channel is slow (e.g. Appletalk for the Apple Laserwriter). Some device recognition would also allow fonts to be targeted for particular printer types and 'hand tuned' for legibility and style e.g. the appearance of characters can differ significantly depending on whether a 'write-to-black' or 'write-to-white' printer drum approach is used[9].

The main argument in favour of a standard, however, is that it minimizes the proliferation of incompatible hardware and software, saves costs and facilitates the exchange of information. It therefore seems better to propose a standard and accept that practical considerations will disallow true device independence. With PostScript, for example, the software has been designed for raster rather then character- or vector-based technologies. A change in technology could therefore make the *de facto* standard redundant.

Given that complete device independence is an unobtainable goal, a proposed standard can take account of the possibility that device dependence can be minimized by transferring device-specific features to other parts of the computer-printer interface. For example, printer independence can be achieved using a generic RIP instead of a device-independent language. This is the approach associated with the RIPs using ACE and RIPrint and also a RIP developed by KMW (USA) which, it is claimed, will work with a range of PDLs as well as printers[14].

However, there are definite advantages to maintaining device independence in the PDL itself. For example, the output device independence adopted by PostScript has allowed Sun Micro-Systems to develop a machine-independent windowing system called NeWS (Network/extensible Window system) incorporating an extended PostScript interpreter for each window. The display for a window is transmitted as a PostScript program and will enable a preview facility without waiting for a printer. The printer itself would receive the same PostScript program.

Turning from device dependence to language features, the essential requirements of a PDL are that it should describe both text and graphics and should translate easily into printer instructions. If the language is in character form it is easier to program and adapt, but there is no benefit to the system user since the composition-to-printout translation should be transparent. Also, character transmission is slower than for more compact codes.

Arguably, document formatting is the responsibility of the interactive composition software and page description languages which can be used to program a page directly are unecessarily powerful. In contrast, languages such as ACE leave all formatting to the composition software. There is then less need for a language in human-readable form and more compact code can be used.

However, programming directly in a high-level procedural language such as PostScript opens up many, sometimes novel, possibilities. JustText commands for example, can be intermingled with PostScript commands providing a very powerful combination for high-quality composition and typesetting[27]. Also, images may be created from mathematical descriptions even where the end result is not known in advance. For example, Pelli[19] uses PostScript to produce patterns to help in his study of the human visual system.

Despite its power, PostScript does not have the logical structures, data structures, programming constructs or speed to create interactive formatting software. Since PostScript can drive screen output as well as a printer, it is possible to handle the interactive aspects with a language such as C and to program the screen drawing with PostScript, although speed of drawing is still a problem. However, it is conceivable that a high-level language could be designed that is not only a scheme for the description of page images but may also be a full programming language, powerful and fast enough to create interactive graphics so that interactive formatting and printing would be essentially the same operation based on one language.

The need for compact code appears to conflict with the idea of a high-level language such as PostScript which is easier to program, to adapt and to use creatively. However, it is possible for a language to use both forms, as in DDL. Another possibility is that a page description could be generated in an intermediate language sufficiently generic to be intelligible to various PDLs and optimised for data compression[9]. This would be particularly beneficial in a networked system with various printers using different PDL interpreters.

Yet another issue of language design is the degree of allowed programmer control, as reflected in the different approaches of Interpress and PostScript.

An important consideration in defining a standard is how it relates to existing standards in other areas. In particular, a PDL standard should be closely compatible with the ANSI virtual device interface (VDI) standard which aims to provide a device-independent way to control graphics hardware. Compatibility with other graphics standards such as GKS and PHIGS is also of concern. However, consideration of other standards has the problem that they may change!

The relationship of a possible PDL standard to other relevant standards is shown in Figure 2.

Font representation is another key issue. The speed trade-offs between accurate bit-map and flexible outline-font strategies are likely to diminish with technological advances. For example, processing of outlines may be greatly accelerated with the use of transputers[13]. However, the resolution advantages of the outline font strategy would remain.

Several other criteria can be advanced as part of a PDL standard and these are listed below together with a summary of the main requirements already discussed. A PDL standard[2,5] should:

- be robust in the face of technological change;
- minimise device dependence by describing a page at a high conceptual level, yet allow advantage to be taken of device-specific features. These combined goals may not necessarily prove incompatible in practice;
- provide user control over certain device-dependent features;
- be portable across a range of machines;
- interface to a range of composition packages;
- integrate text and graphics, including scanned images and photographs;
- take account of other standards, e.g. SGML, GKS, PHIGS, ODA etc;
- allow easy translation from PDL to printer codes;
- have compact code to reduce memory requirements and increase transmission speeds;
- have the ability to process whole documents of any length;
- have the ability to include printing instructions, e.g. print on both sides of paper;
- be modular to assist maintenance and development;
- have a flexible font strategy;
- be cost-effective.

Perhaps these criteria could most readily be met by a multi-level standard. The interfacing level would address the device independence, transmission and printing aspects. The page description level would concern layout and the integration of text and graphics whilst other layers would deal with programming features, font description etc. Whilst the idea of an absolute standard is perhaps inappropriate, not allowing for creativity and progress, it is worth defining a ‘standard’ as a guideline to prevent chaotic proliferation of incompatible systems.

REFERENCES

1. C. King, ‘Page Description Data’, *Systems International*, **13**(12), 41–42 (1985).
2. K. Lang and T. Lang, ‘As You like It’, *Personal Computer World*, **10**(3), 130–134 (1987).
3. K. Hanks, ‘Intelligent Composition’, *Systems International*, **14**(6), 29–31 (1986).
4. S. L. Herring, ‘What To Know About Page Description Languages’, *Canada Data Systems*, **18**(6), 72–76 (1986).

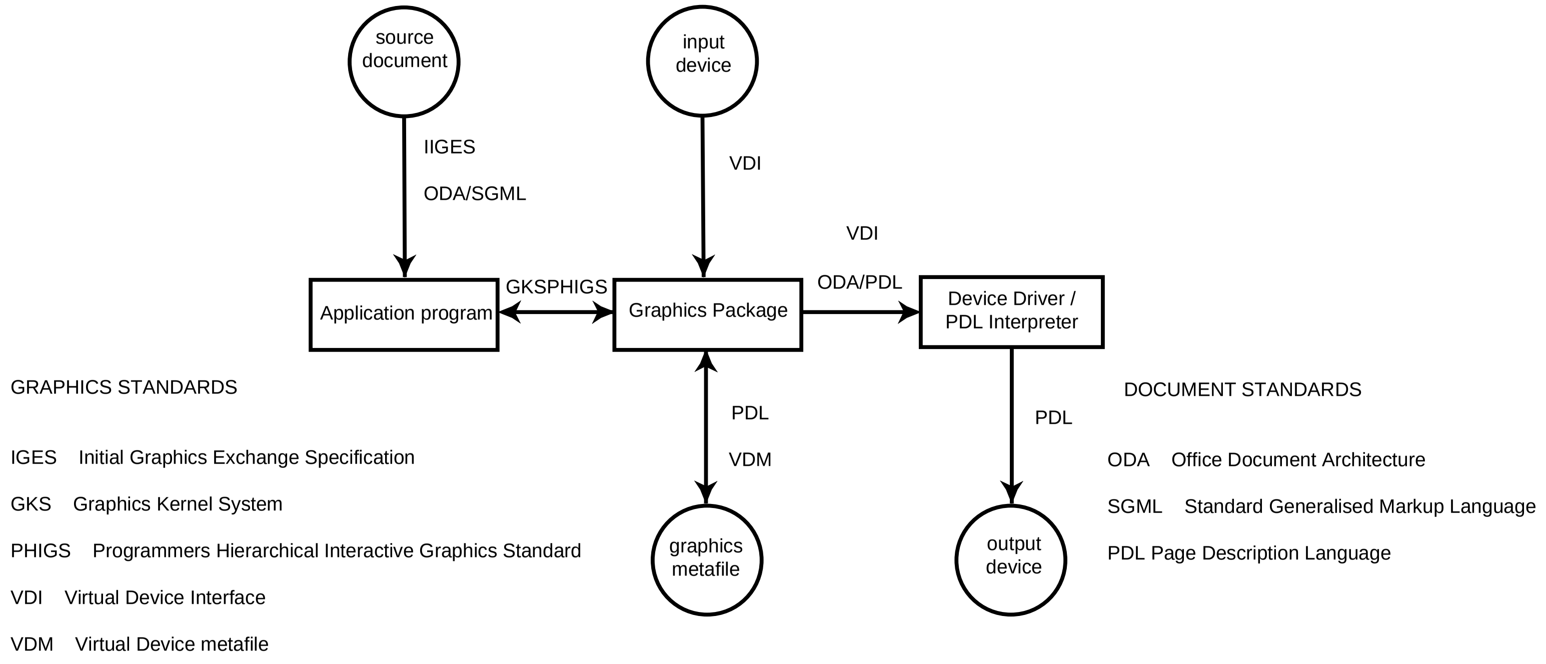
source document
IIGES
ODA/SGML
input device
VDI
Application program
GKSPHIGS
Graphics Package
VDI
ODA/PDL
Device Driver / PDL Interpreter
PDL
VDM
graphics metafile
PDL
output device
GRAPHICS STANDARDS
IGES Initial Graphics Exchange Specification
GKS Graphics Kernel System
PHIGS Programmers Hierarchical Interactive Graphics Standard
VDI Virtual Device Interface
VDM Virtual Device metafile
DOCUMENT STANDARDS
ODA Office Document Architecture
SGML Standard Generalised Markup Language
PDL Page Description Language

Figure 2. Possible relationship of a PDL standard to other standards in document production

5. C. M. Geschke and J. S. Summers, *Software Printing Strategies*, Lasers in Graphics Proc. Conference on Electronic Publishing In The 80s, Vol 2, Dunn Technology, pp.425–432, 1984.
6. C. M. Geschke, ‘Describing Pages’, *Systems International*, **15**(6), 65 (1987).
7. B. Reid, *Procedural Page Description Languages*, Proceedings of the International Conference on Text Processing And Document Manipulation, Cambridge University Press, pp.214–223, 1986.
8. *Interpress Electronic Printing Standard*, Xerox Corporation, 1985, Xerox System Integration Standard, XSIG 048512, Version 3.0.
9. J. Barrett and K. Reistroffer, ‘Designing a Raster Image Processor’, *Byte*, **12**(5), 171–180 (1987).
10. G. Coldwell, ‘Evaluating Lasers’, *Systems International*, **15**, 2, 67–69 (1987).
11. P. Ellison, ‘Designing A High-speed Page Printer Controller’, *Byte*, **12**(9), 225–226 (1987).
12. B. Douglas, ‘Strip Buffer Versus Full Page Bit-map Imaging’, *Byte*, **12**(9), 229–230 (1987).
13. G. Holmes, ‘Transputer Control’, *Systems International*, **15**(2), 71–72 (1987).
14. H. Kevins, ‘Laser Control’, *Systems International*, **15**(4), 57–58 (1987).
15. R. Cook, ‘Page Printers’, *Byte*, **12**(9), 187–197 (1987).
16. J. Cavuoto, ‘Digitized Fonts in PostScript’, *Computer Graphics World*, **8**(9), 27–30 (1985).
17. *The Postscript Language Tutorial And Cookbook*, Addison-Wesley, Reading, 1985.
18. Adobe Systems Inc, *The Postscript Language Reference Manual*, Addison-Wesley, Reading, 1985.
19. D. G. Pelli, ‘Programming in PostScript’, *Byte*, **12**(5), 185–202 (1987).
20. B. Reid, ARPANET Laserlovers Distribution, *Postscript And Interpress: A Comparison*, 1 March 1985.
21. A. Bhushan and M. Plass, ‘The Interpress Page and Document Description Language’, *Computer*, 72–77 July (1986).
22. D. Brailsford, *Page Description Versus Document Description*, Lecture given to the Electronic Publishing Specialist Group (British Computer Society), 5 June 1987.
23. D. J. Harris, *An Approach To The Design Of a Page Description Language*, Proceedings of the International Conference on Text Processing And Document Manipulation, Cambridge University Presss, pp.58–64, 1986.
24. B. Tuthill, ‘Typesetting on the Unix System’, *Byte*, **8**(10), 253–262 (1983).
25. D. Knuth, *The TEX Book*, Addison-Wesley, Reading, 1984.
26. B. W. Kernighan, *A Typesetter-independent TROFF*, Computing Science Technical Report 97, Bell Laboratories, Murray Hill, N.J., 1982.
27. I. Phillips, ‘JustText’, *PCW Desktop Publishing Special*, Feb (1988).
28. D. D. Cowan and G De V Smit, *Combining Interactive Document Editing with Batch Document Formatting*, Proceedings of the International Conference on Text Processing And Document Manipulation, Cambridge University Press, 140–153, 1986.
29. G. Kronert, ‘International Standard For An Office Document Architecture Model’, *Journal of Information Science*, **10**, 69–78 (1985).
30. V. Joloboff, *Trends And Standards In Document Representation*, Proceedings of the International Conference on Text Processing And Document Manipulation, Cambridge University Press, pp.107–124, 1986.
31. ISO 8879, *Information Processing—Text And Office Systems—Standard Generalised Markup Language (SGML)*, Geneva ISO 1987.
Generalised Markup Surv. 14(3), 417–472, 1978.
32. G. Coray, R. Ingold and C. Vanoirbeck, *Formatting Structured Documents: Batch Versus Interactive?*, Proceedings of the International Conference on Text Processing And Document Manipulation, Cambridge University Press, pp.155–170, 1986.
33. R. F. Sproull, W R Sutherland and M K Ullner, *Device-independent Graphics*, McGraw-Hill, New York, pp.8–9, 1985.
34. E. Flerackers, *Scientific Programming With The Graphical Kernel System: Advantages And Disadvantages*, Conference on Scientific Computing and Automation, Amsterdam, 1987.