Strategies for Document Optimization in Digital Publishing*

Felipe Rech Meneguzzi
Leonardo Luceiro Meirelles
Fernando Tarlá Martins Mano
Centro de Pesquisa em Software Embarcado
6800 Ipiranga Avenue
Porto Alegre, Brazil
{felipe,meirelles,fernando}@cpts.pucrs.br

Joao Batista de Souza Oliveira
Ana Cristina Benso da Silva
Faculdade de Informática / PUCRS
6800 Ipiranga Avenue
Porto Alegre, Brazil
{oliveira,benso}@inf.pucrs.br

ABSTRACT

Recent advances in digital press technology have enabled the creation of high-quality personalized documents, with the potential of generating an entire batch of one-of-a-kind documents. Even though digital presses are capable of printing such document sets as fast as they would print regular press jobs, raster image processing might possibly be performed for every different page in the job. Such process demands a large computational effort and it is therefore interesting to gather repeated images that are used throughout all documents and rasterize them as few times as possible. Moreover, performing such process separately from document production in the publishing workflow allows optimization to be performed prior to final printing, thus allowing it to take press hardware specifics into account, and reducing the time taken for it to produce the final output. This paper describes techniques to perform this task using PPML as the document description language, as well as the main issues concerning this kind of document optimization. Several gathering policies are described along with explanatory examples. We also provide and discuss experimental data supporting the use of such strategies.

Categories and Subject Descriptors

J.7 [**Computer Applications**]: Computers in other Systems—*Publishing*; I.7.2 [**Computing Methodologies**]: Document and Text Processing—*Document Preparation, Digital Publishing, Variable Data Printing*

General Terms

Variable Information Documents, Digital Press

Keywords

PPML, Variable Data Printing, Personalized Printing

*This work was (partially) developed in collaboration with HP Brazil R&D

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DocEng'04, October 28–30, 2004, Milwaukee, Wisconsin, USA.
Copyright 2004 ACM 1-58113-938-1/04/0010 ...$5.00.

1. INTRODUCTION

The advent of digital presses capable of producing high-quality full-color printouts has made possible the personalization of printed documents to a degree that has only been previously possible in digital content [10]. In a typical press production workflow, digital content must be transformed into a set of master plates used by the press to generate a large number of identical pages and such process has a high setup cost that is justified by the number of copies produced from the set of press plates. Master "plates" on digital presses exist only logically in the memory of the printing device, thus eliminating the costs associated with the creation of physical plates, and potentially allowing every page in a digital press job to be unique [1, 2]. In order to take advantage of such possibility, new standards are being developed for the definition of variable data documents intended for high-quality printing. One such standard is the *Personalized Print Markup Language* or PPML [2, 8], which allows the specification of large print jobs intended to be processed by digital presses.

Even though a digital press does not involve the creation of physical plates, digital page descriptions must be converted into high resolution raster data used to control the placement of ink over the target media in a process called *raster image processing* or *RIPing*. RIPing is computationally expensive, and thus represents a bottleneck in personalized printing. Such a bottleneck exists mainly because in a traditional print job a rasterized page can be cached and used to produce many physical pages, whereas in personalized printing every page might be unique thus reducing the usefulness of page caching, as in this case the press needs more time to process the raster images than to print a page. Though entirely identical pages are not likely to occur, some elements, like company logos or greeting texts, tend to recur in a print job. Therefore, caching elements smaller than a page is an approach to deal with such bottleneck.

The PPML standard provides constructs for specifying reusable content in a document enabling the document producer to state which elements recur within a job. It is also up to the producer to assess the utility of specifying certain elements as reusable, hence an efficient selection of reusable elements is a key factor in overcoming the RIP bottleneck in personalized documents intended for digital press publishing. In this paper we propose a series of strategies for the optimization of personalized documents. These strategies are based on the analysis of the document set and the selection of reusable elements aiming at maximizing cache utilization

at the consumer press and reducing the data transfer to the press.

This paper is organized as follows: Section 2 contains related work that sets the context for the remaining of this paper; Section 3 details the research regarding document optimization that underpins this work; Section 4 describes the implementation developed to verify the feasibility of the proposed strategies, and finally, Section 5 summarizes the paper while pointing out future research directions.

2. BACKGROUND

This section contains a summary of the most important material related to the work developed in this article. Section 2.1 briefly describes the concept of variable data documents and printing, whereas Section 2.2 describes the parts of the PPML standard specification that are relevant for this work.

2.1 Variable Data Documents

Document sets containing elements that are common across multiple document as well as elements containing variable information are called *Variable Data/Information Documents*, which are also known as *Personalized Documents* [10]. Examples of such documents are phone bills, bank statements or directed marketing catalogs, which contain individualized information like the customer's name, address, a list of transactions or a list of individual product offers. A concrete example of variable data document usage was the 2000 Olympic games in Sydney, where the competition results were published using such technology [4].

In the case of bills and statements, which usually are graphically simple, a variable data document is composed of static text and variable "fields" are laid out to hold customer specific information extracted from a database. These documents are usually generated based on a template designed by a document expert resulting in documents with a nearly identical layout with very well defined static and dynamic areas. More complex documents such as personalized product catalogs or reports containing variable-sized high-quality pictures require different processes to be generated [5, 6] and thus have static and dynamic areas that are harder to recognize.

2.2 Personalized Printing Markup Language

The Personalized Print Markup Language (PPML) [8] was defined by the Printing On Demand Initiative (PODI) in order to provide the means to enable variable data printing for high quality documents. PPML defines a number of print job optimization features, such as reuse of printing objects and job ticketing.

To describe a set of dynamic documents in PPML, usually a database containing the data for different instances of documents is used in conjunction with some kind of template language describing how the document should look and where the different data instances should be placed inside it. There are PPML producers that handle this process.

It is interesting to notice that the PPML producer may or may not declare as reusable elements (text blocks, images, or other graphical constructs) that are repeated throughout the document set, although PPML provides such facilities. Thus, it is reasonable to propose a smaller, less complex and independent process that takes a PPML document and analyzes it, rearranging elements and declaring as reusable the ones with the higher processing requirements, ultimately producing a functionally equivalent PPML output that has a smaller rasterization time on the press. By doing so, it will be able to optimize documents produced by any PPML generator, as a preprocessing step before each document reaches the printer. As a side effect, PPML producers can be made simpler because they may be oblivious to the specifics of the consumers and in turn to document optimization, because this will be made at a later stage. In any case, PPML producers have greater difficulties when optimizing documents, as they would need to have all the document structure in memory to achieve full optimization, and use more memory still to produce the actual PPML.

To decide which elements should be declared reusable and which ones should be used only once, several policies can be proposed. Some of these policies will be described in the next section.

A PPML document is composed of several elements, some of which are shown in Figure 1 with the basic tags required in the definition of a simple document. The root XML element is the `PPML` tag. It encompasses all elements contained within a PPML data set. The PPML tag contains one or more `Document_Set` tags [8].

A `Document_Set` is a group of `Document` tags that are to be treated as a unit, or processed in a single printing job. It is merely a grouping mechanism used to hold multiple instances of similar documents or a set of documents intended for a single person. The `Document_Set` tag is optional within a PPML document and is a synonym for `Job` [8].

The `Document` element represents the binding of layout information (a template) and a record of data from some data set (a database). The `Document` element occurs only within a `Document_Set` element. `Page` elements delimit the content of individual pages in a `Document`. They appear only within `Document` elements [8].

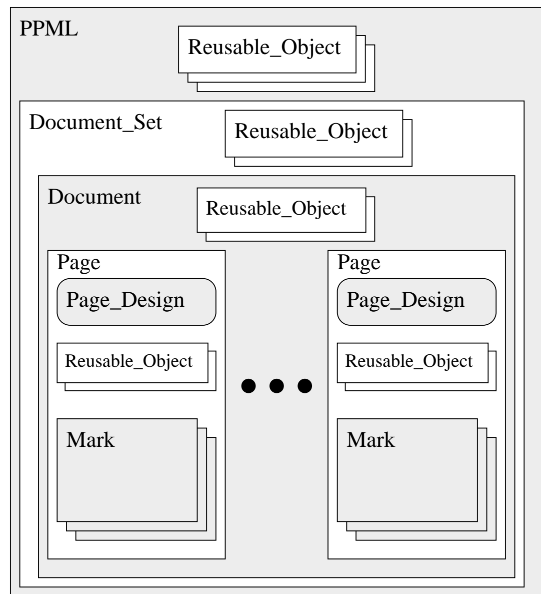

Figure 1: General organization of a PPML document.

PPML does not provide any constructs for describing ac-

tual graphic content, it instead adopts a series of pre-existing standards such as PostScript and Scalable Vector Graphics [9, 7]. The actual graphic descriptions are organized within Object elements, which contain Source elements to specify the type of content being used in Internal_Data or External_Data elements. Conversely, a Mark may refer to content declared in a Reusable_Object (Figure 2). Reusable Objects represent pieces of content that the consumer is instructed to store once processed in order to optimize the printing process by caching recurrent elements in a job.

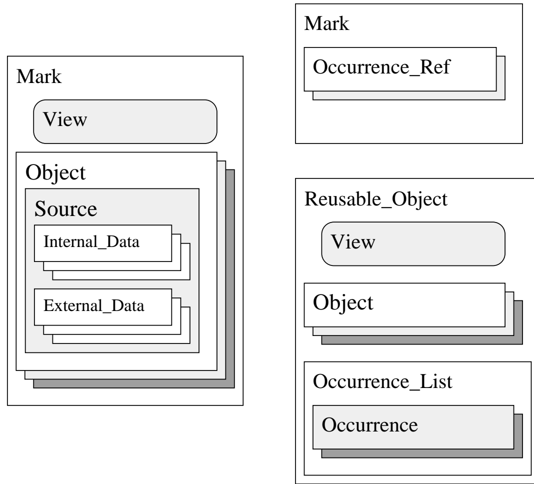

Figure 2: General organization of PPML Marks and Reusable Objects.

One of the main utilities of the PPML format lies in the context of digital printing presses, where large scale printing jobs must be handled by the printing hardware.

3. DOCUMENT OPTIMIZATION

Considering the goals set forth in the introduction, we propose in this section a series of steps used to transform a valid PPML document into an equivalent PPML document with a different organization. The transformed document should produce the same graphical output when processed by a consumer, but its internal organization is optimized to maximize processing speed and throughput in a digital press. In order to define the transformation process we propose a simple problem model, which includes practical considerations regarding cache size, object types, sizes and relative processing effort. Such a model should be flexible enough to be changed according to different needs or printing environments.

3.1 Situation Model and Problem Statement

A PPML document is composed of several logical subdivisions like Document_Set, Document and Page, which ultimately encapsulate Marks that contain the actual content defined within Objects or Occurrence_Refs to Reusable_Objects. Therefore, a PPML-consuming press produces the raster image of a page through the composition of raster images generated from the content-holding elements in the marks of a given PPML Page. When one such element is a regular Object, the corresponding raster image must be generated, whereas when the element is a Reusable_Object, the corresponding raster image is recovered from a raster image cache, thus avoiding the overhead of reprocessing a recurring element. The size of the raster image cache is naturally limited, and in order to take full advantage of caching, all the reusable objects used in every scope of a PPML Document_Set should fit into the available cache size.

Ultimately, the process of converting a PPML document into graphical output can be described in terms of processing a sequence of content elements, each of which will either be processed by a raster image processor or be taken directly from a limited-capacity cache repository, thus requiring considerably less computational effort than rasterizing the element anew. Moreover, such a process will be most efficient when the distribution of reusable and non-reusable objects in the document results in minimum processing time by the consumer, in this case a digital printing press.

Therefore, we assume a generic consumer with an amount M of cache memory that can be used to hold reusable objects, and M is known in advance by the optimization algorithm. An object is an individual file included as external data into the document or the raw data declared within a document prior to any PPML transformation operations (*e.g.* Rotation, Translation, ...). In addition, a PPML document set can be seen as a sequence of objects $A_1, \ldots, A_n$ with possibly repeated objects, that is, $A_i = A_j$ for $i \neq j$. In fact, the repetition of objects is a condition for their reuse, because if no objects are repeated there is no need for them to be cached. An important aspect of the caching strategies defined in our proposal is that, unlike caching strategies used in Operating Systems [12], the processing tool has prior knowledge of all the occurrences of a given object, thus our strategies can actually aim at achieving optimum cache use. Moreover, we assume that the information about the amount $\text{Size}(A_i)$ of memory used to store object A_i in the cache after any preprocessing due to its format (EPS, GIF, etc.) is also available. One might argue that this process would ultimately require an entire document to be loaded into memory in order for any optimization to take place, thus barring optimization for very large transactional print jobs. On the other hand, this process can be used in a finer level of granularity with regard to the PPML document hierarchy, *i.e.* optimization might be performed over a fixed set of pages or documents within a job. This process could then be used to attain locally optimized pieces of a job.

The need for keeping objects in the cache changes over time as the document is processed. For example, if an object is used in the first ten pages of a 300-page high-quality personalized report, there is no need for keeping it in memory after page 10 is processed. Therefore, it is necessary to describe different instants in time when the document is being processed. Considering that a PPML document is a sequence of objects $A_1, \ldots, A_n$, it is natural to define instant t as immediately after object A_t has been processed. Conversely, the instant before processing starts is instant 0 and the instant after all processing is done is instant n. The contents of the cache memory also change over time, thus we may define the cache at an instant t as a set C_t containing objects. Clearly, at instant 0 the cache C_0 is empty.

Considering the situation thus described, we propose the following problem:

(Reusable Object Optimization). *Let $\mathcal{J}$ be a print job containing a sequence $A_1, \ldots, A_n$ of content-bearing objects,*

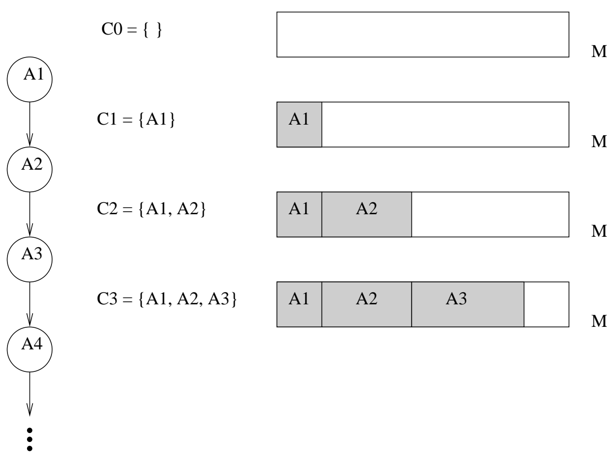

Figure 3: Evolution of a reusable object cache over time.

such that repeated objects may exist, that is, it is possible that $A_i = A_j$ *for* $i \neq j$*, and let* $\mathcal{P}_M$ *be a process capable of caching reusable objects in a cache of size* M*. The cost of processing object* A_i *is* $\text{Raster}(A_i)$*, and let* $\text{Raster}(\mathcal{J})$ *be the total cost of processing* $\mathcal{J}$*. The problem of Reusable Object Optimization consists in redefining* $\mathcal{J}$ *as a new job* $\mathcal{J}'$ *containing reusable objects* $R_1, \ldots, R_m$ *and a sequence of objects* $A'_1, \ldots, A'_n$ *such that* A'_i *is either a regular content object with its normal processing cost or a reference to a reusable object* R_j *with negligible processing cost, and such that* $\text{Raster}(\mathcal{J}')$ *is as low as possible.*

3.2 Optimization Strategies

Considering the problem defined in Section 3.1, we propose an optimization process based on the transformation of an existing PPML structure into a functionally equivalent one that declares some of its content in reusable objects. Reusable objects are selected according to a simplified simulation of the consumer process. Such simulation aims at selecting a set of cache objects resulting in the least possible net processing effort by the consumer while not exceeding the total amount of physical memory in the press and thus not incurring in performance losses due to memory swapping in the press. We define the strategy for selecting reusable objects according to Algorithm 1.

Algorithm 1 implicitly uses two auxiliary functions, a priority evaluation function that measures the importance of keeping an object in cache at any given time (to help decide what can be thrown away from the cache to make room for another object) and a cache replacement policy that selects objects in the cache for replacement, making decisions based on these priorities. These functions will be detailed further in the following sections.

3.2.1 Priority Evaluation

To select the most useful candidates for reusability, it is necessary to assign some kind of priority to every object within a PPML document set. Such a priority should compute the payoff of keeping an object in cache with respect to other objects in the same document set. The main components that define such payoff are the size of the resulting raster image, as well as the frequency with which this raster

Algorithm 1 Reusable object selector.

```
Require: Cache is empty
for all objects A_i in the document set do
  if A_i is already in the cache then
    if A_i was used only once before then
      Declare A_i reusable {It appears at least twice}
    else if A_i is already declared as reusable then
      A_i was already declared reusable, so refer to it.
    end if
  end if
  if A_i is not in the cache then
    if there is no room for A_i then
      Find objects to be removed
    end if
    Load A_i and use it
  end if
  if A_i is not used later then
    Remove A_i from the cache
  end if
  Re-evaluate priorities in the cache {An instant has elapsed}
end for
```

image is used in the document set. Therefore we define a function $\text{Pri}_t(A_i)$ that expresses the priority for holding object A_i in the cache at instant t. Function $\text{Pri}_t(A_i)$ is non-negative and changes value as time elapses, describing the importance of keeping object A_i in the cache at a given instant t. The definition of $\text{Pri}_t(A_i)$ is affected by the following considerations:

1. If object A_i is in the cache at instant t but is not used afterwards, it is clear that $\text{Pri}_{t'}(A_i) = 0$ for $t' \geq t$;

2. If object A_i is in the cache at instant t and is also the next object to be used, it should have $\text{Pri}_t(A_i)$ as high as possible, in order not to leave the cache at that instant. That is, the next object to be used should never be removed from the cache;

3. If two equal-sized objects A_i and A_j are in the cache at a given instant t and A_i is used sooner than A_j, then we should have $\text{Pri}_t(A_i) > \text{Pri}_t(A_j)$;

4. If objects A_i and A_j have different sizes and are both in the cache at a given instant t, it is necessary to look further ahead in the document to check when one of them will be used again and adjust the priorities accordingly. For example, if A_i is a 10 MB image and A_j is a 1 MB image, there are several situations to be handled:

   (a) If there are fewer than 10 occurrences of A_j before the next occurrence of A_i, then it is better if A_j leaves the cache as reloading it less than ten times requires less effort than reloading A_i once;

   (b) If there are exactly 10 occurrences of A_j before the next occurrence of A_i, then both objects are equivalent with respect to data transfer from auxiliary memory;

   (c) If there are more than 10 occurrences of A_j before the next occurrence of A_i, then it is better if A_i is removed from the cache to be reloaded later, when needed;

5. If two objects A_i and A_j are of different types (*e.g.* TIFF and SVG files) and are in the cache at a given instant t, priority should be higher for the file that needs more preprocessing when being reloaded into the cache;

To provide a priority function Pri() that satisfies those requirements, we propose

$$\text{Pri}_t(A_i) = \frac{\text{Weight}(A_i) * \text{Size}(A_i)}{\text{Distance}(A_i, t)}$$

where

- $\text{Size}(A_i)$ returns the amount of memory needed to store object A_i after any preprocessing. As a first approximation, it can be obtained from the PPML `CLIP_RECT` directive, that contains information about the total area used by the element.
- $\text{Distance}(A_i, t)$ returns how many objects different from A_i are used from instant t to the next occurrence of A_i. Thus, this is a measure of how far ahead the object will be necessary, and as this distance increases, keeping it in the cache becomes less desirable.

  This represents a clear advantage of the PPML processor over the press, as the press makes decisions about moving objects to and from memory without knowledge of what will come next. Considering that the algorithm has information available about the complete document, it can make better decisions.
- $\text{Weight}(A_i)$ is a weighting factor that describes how much effort is required to rasterize file A_i, depending on its type. By changing the values for different files, one can increase the priority of files whose rasterization requires more effort. As a first approximation $\text{Weight}(A_i)$ can be taken as 1 for all file types, though in a real situation these values must be tuned accordingly.

Unlike traditional caching algorithms, which put significant importance in the access frequency of cached elements, the proposed priority function emphasizes the adjustment of a *weight* value associated with format of such an object. This shift in prioritization reflects the multiplicity of graphic formats currently processed by printing hardware as well as the capabilities of specific platforms to deal with such formats. For instance, even though Postscript and SVG are similar formats in terms of capabilities and associated processing cost, many press platforms have dedicated hardware to interpret Postscript, resulting in significantly different processing times between these formats.

3.2.2 Caching Policies

The second main component of the optimization strategy described in this work is a cache replacement policy that determines which objects are to be discarded from cache in order to allow new ones to be used in the printing process.

An important aspect of our use of cache replacement policies is that, unlike the caching strategies used in Operating Systems or Memory Architectures [12], complete information regarding future usage of cached elements is available. Such information is incorporated into the priority function of Section 3.2.1 and is taken into account in the three policies defined below, used to decide which objects are to be taken out of the cache to provide room for a new object A_i.

First fit: simply searches the current cache objects from lowest priority to highest, and the first one whose size allows A_i to fit in is taken out. If no one is found, the object with smallest priority is deleted and the process is repeated;

Sum fit: starts by removing objects from lowest priority to highest until there is enough memory to hold A_i;

Sum fit with recovery: analyzes objects from lowest priority to highest, marking them for removal until there is enough memory to hold A_i. After enough objects are selected, the list of marked objects is checked for objects that may remain in the cache;

As an example, suppose that we need 40 MB from the cache, but only 5 MB are free. When trying to find another 35 MB, we mark objects with lowest priority and sizes 10 (adding to 15 MB), 20 (adding to 35 MB) and 40 MB (adding to 75 MB) to leave the cache. By doing so we would have 75 MB free to store the 40 MB object, which is too much as there will still be 35 MB free. We then go *backwards* in the list (this assures that objects with higher priority will be considered first) and determine which objects can be kept. In this case, the objects with 10 and 20 MB will be kept and only the 40 MB object will leave the cache.

As there are several policies that may be used to decide what to take out of the cache, the proposal is very simple: to read in the entire document and simulate the effect of all proposals, measuring the amount of data moved into the cache by each one of them. The one that moves less data is used to generate the final PPML output. Moreover, as we simply have to simulate the different policies and decide which one is the best, additional policies can be easily incorporated into this model.

3.3 Dealing with PPML constraints

One of the problems that had to be solved in this approach was the fact that to ensure proper cache utilization it is sometimes necessary for objects to be explicitly deallocated. Unfortunately PPML does not provide such an instruction, and objects can only be loaded *into* the cache, but not taken out. Though these objects may have scopes associated to them, PPML provides only a limited range of scopes and their granularity is not fine enough for our intended usage model. Therefore, without an explicit deallocation operator we are only able to load new objects, while the press handles older objects with its own algorithms, writing them to auxiliary memory and possibly making worse decisions concerning the use of memory.

It is interesting to point out that reusable objects can have an associated weight attribute, so that the PPML producer can give a hint about what objects should be kept in the cache and which ones have less priority. However, *the press might not take into account* such information, which is provided merely as a hint.

An alternate approach to deal with this issue would be the creation of a separate PPML job containing only reusable objects. This document would be sent to the press to allow it to rasterize the reusable objects prior to the processing of the actual printable document. Using this approach, when the actual document intended for printing is sent to

Size	Distribution
1-10 MB	5%
11-30 MB	10%
31-50 MB	20%
51-100 MB	30%
101-150 MB	20%
151-200 MB	15%

Table 1: Distribution of object sizes in the test

the press, the objects sent beforehand would already be processed, thus speeding up the printing process.

Although the proposed strategy can be considered complete from a theoretical point of view, there is still room for improvement: the current model does not consider the precise preprocessing effort required for a given object. If such information was available, the PPML processor would be able to identify when an object having a very expensive preprocessing will be disposed of and preserve it for later use. This would allow the press to store such an object in some kind of secondary storage if the cost of writing it and reading it anew is less than the cost of reading it and rasterizing it again. Furthermore, the cost of rendering an object depends not only on its format but also *on its content*, so that it is common to find objects described in the same format but having drastically different rendering times depending on the kind of instructions contained in the file as is the case with PDF, SVG and PS. Thus, a more precise estimation of the rendering time would require the analysis of the content of each object, which is a clearly hopeless task, and examining the size of the object would not be more effective. For example, a PS file with just a few hundred bytes might put a printer into an infinite loop.

Nevertheless, a limited but straightforward estimation of rendering times may be made by timing the rendering of the objects themselves, so that we have approximate data on the rendering times and may assume that these times will be repeated when an object is rendered again. By doing so, relative rendering costs can be approximated.

4. IMPLEMENTATION

The optimization strategies described in Section 3 were used in a series of prototype implementations aimed at verifying the characteristics of such strategies. Initially the proposed model was tested in a simulated environment, whose results are summarized in Section 4.1, and later the entire model was used in the implementation of a PPML processing tool, described in Section 4.2.

4.1 Caching Strategies Simulation

To verify the viability of implementing a preprocessor using the model described in Section 3 and to benchmark its advantages with respect to the suggested optimization approaches, a number of experiments were performed. Such experiments were conducted using a simplified representation of a PPML document on which the proposed optimization model operated. The cache utilization policies were implemented within the evaluation tool.

The experiments were made with documents consisting of 300 instances of 50 randomly chosen objects, each requiring the same amount of rendering effort. Object sizes were distributed according to the classes of Table 1.

Policy	Avg. Transfer	Amount of reuse
No Cache	24227MB	0%
First Fit	18036MB	25.55%
Sum Fit	15685MB	35.26%
Sum Fit w/ Rec.	15410MB	36.39%
Infinite Cache	4158MB	82.84%

Table 2: Average data transfer and amount of reuse.

We begin by taking into account two extreme situations regarding cache size (Figure 4 shows the amount of data read in and rendered for each situation):

- **Infinite Cache Size:** considers a press where memory allocation in the cache is always successful. Therefore, whenever an object is needed, it is taken from disk, rendered and stored for further use.
- **No Cache:** considers a press model in which every object required for printing must be transferred from disk and rasterized again. This situation is equivalent to a document with no reusable objects at all.

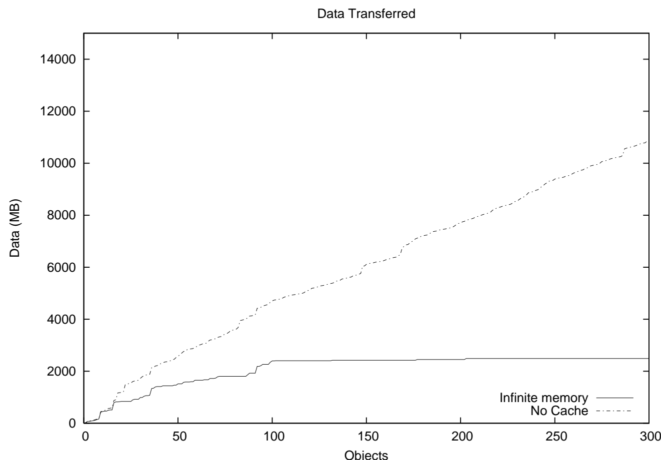

Figure 4: Data transfer extremes.

These situations represent the boundaries beyond which no reusability policy can possibly be situated. Considering the upper extreme, no reusability policy will perform worse than the *No Cache* curve. Conversely, no reusability policy can perform better than the *Infinity* curve. A clear conclusion that can be reached considering these situations is that the closer a policy is to the *Infinity* curve, the better it is.

The evaluation data was used to test the proposed policies in a simulated consumer press incorporating a 600 MB cache, the results being shown in Table 2 and the amount of data transferred for each policy in Figure 5. One sees that there is a reduction of at least 2 GB in transferred data when the cache is used. Thus, evaluation of the proposed cache utilization policies has shown that a PPML document optimized using the First Fit policy would require significantly less effort by the press than the same document without reusable objects. Also considering the proposed test data, the Sum Fit policy performed better than First Fit with its reconsidering variation performing slightly better.

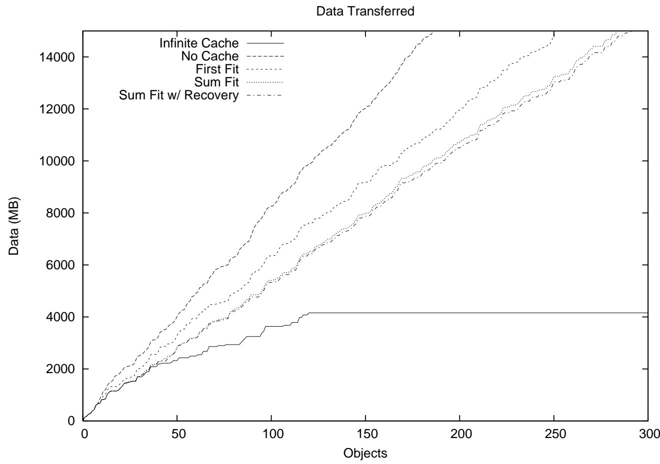

Figure 5: Data transfer for the replacement policies.

4.2 The Cruncher: A PPML processing tool

Once the optimization hypothesis was verified in the preliminary experiments, a tool was designed intended to perform the proposed processes on actual PPML documents. Such tool was dubbed the **PPML Cruncher** and its initial goal was to analyze PPML input, identify reusable objects and rearrange its content so that potentially reusable objects would be declared as such whenever such re-declaration was appropriate.

4.2.1 Plug-in Architecture

The **Cruncher** architecture was designed to allow the incorporation of new plug-in modules intended to execute arbitrary algorithms upon PPML files. Such architecture is centered around an abstract model of PPML elements designed using the *Composite Pattern* [3]. The classes that comprise this model, in turn, are designed to accept *Visitor* classes [3, 11] that implement the pluggable algorithms.

Using the proposed architecture, the analysis of a PPML document according to the policies defined in this work would be implemented as a plug-in within the main program. The reorganization of the PPML tree would be a separate plug-in, thus allowing different usage possibilities for the reusable object elements of the PPML standards.

4.2.2 Cruncher Implementation

From an implementation point of view, a parser module based on *Apache Commons Digester* has been put together in order to parse a subset of the PPML specification. Moreover, the plug-ins responsible for analyzing a PPML document set according to the First Fit and Sum Fit policies have been implemented, and evaluated using real PPML documents.

The current implementation of the Cruncher architecture is capable of a series of additional pre-processing refinements during the optimization process, such as:

1. Using a pre-processing plug-in **Cruncher** is able to recognize repeated objects and mark them as such for the reuse policies to consider. It is also possible to use a pre-processing module capable of identifying specific reuse instructions provided by other tools, such as an XSL-FO-driven producer [5, 6], which can use some kind of markup to identify static and dynamic elements gathered while processing the stylesheet and data. In such setting, a **Cruncher** plug-in is be able to take advantage of application-dependent information to identify reusable objects;

2. If an input file already contains reusable objects, a pre-processing module "expands" the file so that the reusability features can deal with a uniform document representation, not having to cope with already existing reusable object details and scopes;

3. Considering its architecture, the **Cruncher** is able to use different strategies and methods for PPML document manipulation, as well as being able to consider press implementation particularities.

Policy	RO Decls.	RO Refs.	Data Transf.
No Cache	0	0	5432.58MB
Best Fit	1369	4578	4922.15MB
Sum Fit	1600	5383	4573.68MB
Sum Fit w Rec.	2047	6867	4553.98MB

Table 3: Total data transfer, Reusable Object (RO) declarations and references.

4.3 Results and Evaluation

An implementation of the **Cruncher** was used in the evaluation of the proposed algorithms, as well as the scalability of our optimization strategy. The documents used for the evaluated benchmarks contained approximately 48000 graphical objects comprising a mixture of vector graphics (SVG), and bitmaps (JPEG and PNG), 30% of which were potentially reusable, *i.e.* objects occurring more than once within the job. After being processed by the **Cruncher** for a cache size of 500MB the data transfer statistics of Figure 6 were obtained.

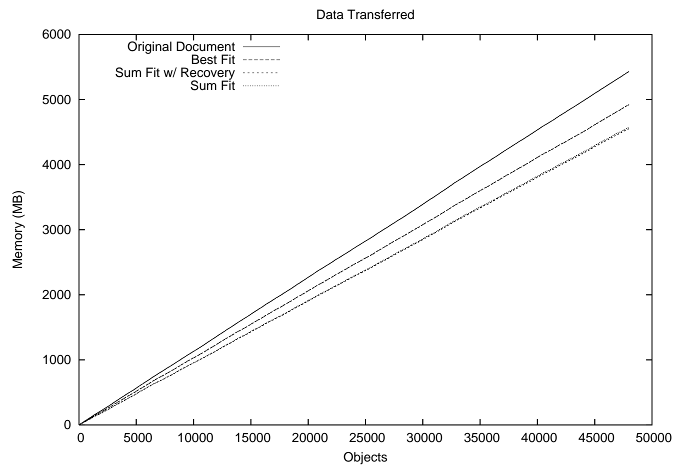

Figure 6: Data transfer for the implemented algorithms.

An analysis of Figure 6 reveals that most of the preliminary suspicions regarding document performance were cor-

roborated by the evaluation data. More specifically, that optimization using a simple reuse policy such as Best Fit is significantly better than no optimization at all. Moreover the two more advanced policies offer significant improvements over Best Fit. Nevertheless, the performance gap between Sum Fit and Sum Fit with Recovery for actual PPML documents has proven to be narrower than for toy examples. The performance deviation between these two algorithms is so slight as to be imperceptible in the curves of Figure 6, while the data in which these curves are based (shown in Table 3) shows that the concrete difference is approximately 20MB for a 5.3GB-large job (approximately 0.4%). Such a small performance difference in this situation is attributed to the small size of the objects whose priority value allows them to be retained in cache during the recovery step of the third policy. More precisely, the recovery step allows a very large number of additional objects, that would otherwise be disposed by the Sum Fit policy, to be kept in memory by "squeezing" back into the cache, as evidenced by Table 3. Even though approximately 400 additional objects are declared as reusable, resulting in more than 1500 further disposable objects being replaced in the document, these objects are very small in comparison to others that remain longer in the cache.

5. CONCLUDING REMARKS

Considering the work developed so far, it seems clear that the rational use of the reusability features of any variable data document standard is of paramount importance to the performance of large-scale high-quality personalized printing. In this paper we have outlined the basic aspects of an optimization strategy based on a simulation of the processing of a job within a digital press. Nevertheless, there is still room for improvements in terms of other optimization strategies. Such improvements may come as refinements in our simulation-based optimization or as novel optimization strategies. An example of an alternate strategy for optimizing personalized documents would be the usage of genetic algorithms [10] as a means to iteratively refine the usage of reusable objects within the document.

The detection and organization of repeated elements outlined in this paper can also be useful for archiving purposes. In this type of application the strategies described for printing optimization would be analogous to compression algorithms in the sense that they identify recurring patterns within documents and index them for usage throughout the document instead of the actual content, ultimately reducing storage requirements for a given document. Such an archiving solution could be integrated into the digital publishing workflow using the Cruncher architecture, which would declare as reusable *every* object which appears multiple times when a document is archived. Later in the production flow, when such a document is retrieved from storage, Cruncher would re-expand the reusable objects prior to its optimization process targeting the document for a specific press hardware.

Work in the implementation of our basic Cruncher architecture is concluded, and future developments of such architecture will point out alternative investigation paths to our research. As of the submission of this paper, the authors were working on algorithms to automatically decide the equality of complex elements within a document set.

Acknowledgments

This work was (partially) developed in collaboration with HP Brazil R&D. The authors would like to thank Fabio Giannetti and Antony Wiley, our research partners from Hewlett-Packard Laboratories Bristol, whose ideas and participation were invaluable for this research to take place.

6. REFERENCES

[1] D. D. Bosschere. Book ticket files & imposition templates for variable data printing fundamentals for PPML. In *XML Europe 2000*, Paris, France, 2000. International Digital Enterprise Alliance.

[2] D. DeBronkart and P. Davis. PPML (personalized print markup language): A new XML-based industry standard print language. In *XML Europe 2000*, Paris, France, 2000. International Digital Enterprise Alliance.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley, 1994.

[4] T. Goodman. Case Study: Digital Publishing at the Olympic Games. In *Open Publish 2001*, 2001.

[5] W. E. Kimber. Using XSL Formatting Objects for Production-Quality Internationalized Document Printing. In *XML Europe 2003*, pages 1–20, London, England, 2003. International Digital Enterprise Alliance.

[6] K. Kreulich. Publishing Workflows with XSL-FO. In *XML Europe 2003*, pages 1–6, London, England, 2003. International Digital Enterprise Alliance.

[7] J. C. Mong and D. F. Brailsford. Using svg as the rendering model for structured and graphically complex Web material. In *Proceedings of the 2003 ACM symposium on Document engineering*, pages 88–91. ACM Press, 2003.

[8] P. PODI. Print markup language functional specification version 2.1, 2002. Extracted from http://www.podi.org/ at June 20th, 2003.

[9] S. Probets, J. Mong, D. Evans, and D. Brailsford. Vector graphics: from PostScript and Flash to SVG. In *Proceedings of the 2001 ACM Symposium on Document engineering*, pages 135–143. ACM Press, 2001.

[10] L. Purvis, S. Harrington, B. O'Sullivan, and E. C. Freuder. Creating personalized documents: an optimization approach. In *Proceedings of the 2003 ACM symposium on Document engineering*, pages 68–77. ACM Press, 2003.

[11] A. Shalloway and T. James R. *Design Patterms Explained: A New Perspective on Object-Oriented Design*. Addison Wesley, 2001.

[12] A. Tanenbaum. *Modern Operating Systems*. Prentice Hall, 2nd edition, 2001.