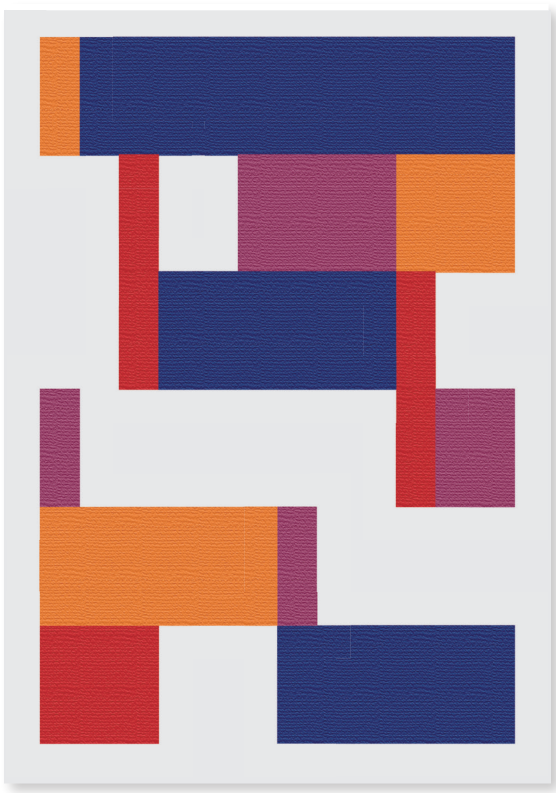
On Modernizing Variable Data Printing (VDP)

Transforming VDP Into a Media That Supports Fine-Grained, Unconstrained, and Highly Creative Variability

JACOB AIZIKOWITZ



Contents

APPENDIX I
FEW OBSERVATIONS ABOUT THE FUTURE OF MODERN VDP AND TOOLS SUPPORTING IT

APPENDIX II
CLOSING NOTES, FACTS, CLARIFICATIONS

Abstract

This paper examines the software developments that modernized Variable Data Printing (VDP), transforming it into a first-class media, same as digital, for individualized communications applications. The resulting discipline is Modern VDP, and it enabled print to play, together with digital, in Marketing Automation and Customer Experience Management (CXM), which are large and fast-growing markets. The key to becoming such first-class media was Modern VDP's enablement of fine-grained, unconstrained, creative variability combined with its online access to data and scripting to compute variability.

This paper weaves together the needs, the challenges, the innovations, and the developments that envisioned Modern VDP and brought it to life.

Introduction

The essence of modernizing Variable Data Printing (VDP) was the vision to enable the design, authoring, and production of individually relevant and engaging variable data print documents. We envisioned these documents to support fine-grained and unconstrained variability, which meant allowing a document to include parts of any type, dimension, orientation, or location, that are variable. We felt that such variability would bring VDP to support creative and engaging individualized communications.

The value of the vision was high, as the businesses that mastered Modern VDP got access to large markets, especially Marketing Automation and Customer Experience Management (CXM) (see **On CXM, SXM, and Customer Journeys *on page 43***). And for the digital printing industry, it provided a much-needed stream of color pages.

However, implementing the vision was challenging. Modern VDP's rich and unconstrained variability implied complex, hence lengthy computations when preparing documents for print. This performance barrier risked the viability of Modern VDP; therefore, overcoming it was critical for materializing the vision.

The heart of this paper is about analyzing the performance barrier and explaining the solutions that eliminated it. Solutions included inventing new abstractions for documents and print definition languages, developing software to implement them, and using networked computers to deliver scalable performance gains.

This paper reflects on the experience with VDP that my colleagues and I had while working at Scitex (1996-1999) and XMPie (1999-2019). Naturally, the paper's point of view is undeniably biased; however, our experience was broad. It covered working on a broad spectrum of related disciplines, including HW, and it included extensive collaboration with small and large industry players who, like us, aspired to modernize VDP. Moreover, our experience grew from a clean start, challenging the incumbent discipline (Legacy VDP) rather than gradually evolving it. Altogether, these characteristics of our experience make the approach we took, the challenges we faced, and the solutions we developed universally relevant; hence this paper.

Paper's Outline

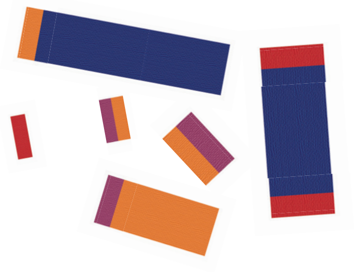
The paper starts by presenting the Printing and Desktop Publishing fundamentals, familiarizing the reader with the basis upon which VDP's modernization grew. The paper will then zoom in to discuss VDP and its modernization. Since modernizing VDP embraced the principles of desktop publishing, the discussion will cover the creation of a new document abstraction for variable data documents and the design of a Variable PDL (VPDL) – a new Page Definition Language (PDL) for VDP jobs. The discussion then focuses on printing this new document type, which is all about translating such a document to a stream of instructions in the VPDL. Finally, the paper closes by presenting the use of concurrent processing across networked servers to deliver ultra-performance gains for the print process.

Driven by reviewers' feedback, I added two Appendix chapters. The first one outlines some insights about What's Next with Modern VDP. The second one covers few background key concepts or dives deeper into some technologies.

Guiding Comments to the Reader

The heart of this paper is about technology. As such, certain parts might be overly technical for some readers while being precisely suitable for others. While I could not avoid a technical analysis, I pushed some of it to an Appendix. Nevertheless, a section may still appear overly technical for some readers. I recommend that the reader will skip either to the section's summary or to the next section in such a case.

In the paper, I sometimes use "we" or "our," although I am the author. I use these to reflect on the team efforts behind critical issues addressed by this paper.



Desktop Publishing – The Foundation

A printed document, such as an annual report or a product brochure, starts as a digital desktop document that a person creates interactively using a document application, such as Adobe InDesign. Then, it results in a printed document through the process outlined in **Figure 1**.

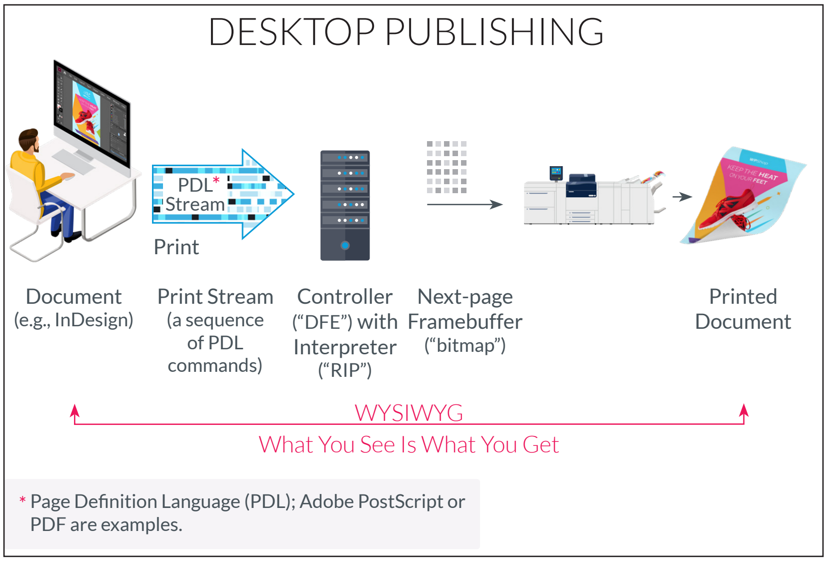

DESKTOP PUBLISHING
PDL* Stream
Print
Document (e.g., InDesign)
Print Stream (a sequence of PDL commands)
Controller ("DFE") with Interpreter ("RIP")
Next-page Framebuffer ("bitmap")
Printed Document
WYSIWYG
What You See Is What You Get
* Page Definition Language (PDL); Adobe PostScript or PDF are examples.

Figure 1: From a Document to Printed Pages, The Desktop Publishing Way

When instructed to print the document, the application – or the system's print driver – generates a sequence of instructions in a PDL, e.g., Adobe's PostScript, which specifies the document's content and appearance in print and communicates it to the printer's Controller; this sequence is known as a Print Stream.[1] The Interpreter is the Controller's subsystem that processes print streams, interpreting the PDL instructions and generating, for each page, a bitmap that faithfully resembles the specified page.[2] Finally, in coordination with its Interpreter, the Controller writes these bitmaps into the printer's page memory ("Framebuffer"), instructing the printer to render the image in the Framebuffer onto a printed page. This process results in printed pages that appear identical to the document's pages on the bitmapped display.

The above end-to-end process is the essence of implementing **W**hat **Y**ou **S**ee (on the computer display) **I**s **W**hat **Y**ou **G**et (on paper) (**WYSIWYG**), which demystified printing, allowing the "originators" – authors, designers, creative professionals – to control the publishing process from their desktop computers. So, WYSIWYG – the heart of desktop publishing – democratized publishing, giving control to the <u>many</u> (originators) and taking it away from the <u>few</u> (print magicians). For further details, please read **Desktop Publishing on page 57**.

While elegant and straightforward, the above process must perform within the real-time constraints imposed by the printer's speed of printing pages. These constraints arise because the Controller must finish writing the Framebuffer of the next page before the print engine starts printing it. As a result, the Interpreter must generate the bitmap of the next page before the printer finishes printing the current page. Adhering to these real-time

[1] In practice, the head of the stream may be communicated to the printer while its tail is being generated.

[2] A bitmap represents an array of digital representation of pixels. It may have one bit per pixel, reflecting an on/off semantics, or say thirty-two bits per pixel, every eight represent the intensity of a color component (e.g., CMYK).

requirements demands that either the interpretation speed be higher than the speed at which the print engine emits pages or that an Interpret-first Print-later (**IFPL**) method be used.[3] Regretfully, as we will see in the sections below, IFPL is not applicable for variable data printing.

[3] Implementing IFPL requires a sophisticated internal buffering mechanism, allowing to implement IFPL in chunks.

Modernizing VDP

A VDP job represents a set of printed documents, each individualized to one recipient.[4] Taking the lead from Desktop Publishing, a VDP job needs to start as a document on the desktop that describes a set of documents – all the individualized documents of the VDP job. Printing such a document requires generating a print stream in a VPDL. Moreover, the Modern VDP vision demands that the VPDL and the document support rich, fine-grained, and unconstrained variability.

Exploring VDP's modernization, we will start with the language challenge and follow with the document challenge. Then we will tie these together in the section about printing Modern VDP.

■ The Emergence of Elements-Based VPDLs (E-VPDL)

As discussed, a VDP stream specifies the printing of the individualized documents of the VDP job. Therefore, a straightforward VDP stream would be the concatenation of the regular print streams of the job's individualized documents. But unfortunately, a concatenation-style VDP stream is impossible performance-wise. Its large size – directly correlated to the typically large size (e.g., number of recipients) of VDP jobs – prevents

[4] Recipients are usually individuals, but a recipient might be other things, such as an indication of a version, a product type for a brochure, or a branch of a restaurant chain.

interpreting the whole stream before printing, leaving **Interpreting While Printing** as the only viable method.[5] But Interpreting While Printing requires interpreting faster than the speed at which the print-engine emits pages, which is impossible to guarantee for concatenation-style VDP stream. This is because such a stream is no different from a stream specifying a random collection of unrelated documents.

Since the individual documents of a VDP job are not a random collection of documents, there must be a better representation of a VDP stream. The key insight was that the individual documents of a VDP job share content and the content's presentation, which means that an item that is part of a document for one recipient may also be a part of another recipient's document. This insight implied that an Interpreter would be able to process such repeating part once, cache the resulting bitmap, and reuse it whenever that part would appear again.

But how would an Interpreter identify a section in the VDP stream that represents such a part? We addressed this question by defining a new VPDL with language constructs for explicitly specifying such potentially repeating items. I would add that there are probably other algorithmic alternatives, but further research is needed to reveal and assess them. However, this paper will focus on the "being explicit" approach for specifying parts of the VDP stream that are likely to repeat.

The VPDL we envisioned (1997) had explicit constructs for specifying items that may repeat and for reusing such items. We introduced *Elements* – a language construct – for defining and using items that may repeat. We use the term **Elements-based VPDLs** (**E-VPDL**) to refer to this new flavor of VPDLs.

We envisioned that an Element would support two operations: *Defining* it and *Placing* it. Defining an Element gives it a name and associates it with a sequence of instructions, e.g., in PostScript, which specifies how to render

[5] The stream's size directly reflects the size of the VDP jobs, which can be 10,000 records long and, in many cases, 100,000 records long.

it in print. And Placing an Element is "rendering" the Element on the page at the specified location. "Rendering" is tricky. The Interpreter can either reuse a bitmap it prepared for the Element when it processed the Element's definition or avoid preparing the bitmap at definition time and reinterpret the definition's PDL upon each "render." While results would be identical, reusing the bitmap might be better performance-wise.

The leading Element-based languages were Scitex VPS, PPML (created by a consortium of companies; started as an XML encoding of VPS), and Adobe's PDF/VT, which became the standard. For further context on Element-based VPDL, examine the section **On Elements-based VPDLs on page 47**.

In summary, E-VPDLs and similarly flavored VPDLs opened the door for developers – of Controllers, Interpreters, and hardware accelerators – to create solutions for efficient and fast processing of Modern VDP jobs.

The Emergence of Dynamic Documents

Given a VPDL, one can create a VDP stream by programming in the VPDL. However, like what happened in Desktop Publishing (see **Desktop Publishing *on page 57***), VDP shifted from programming the print stream to designing the document and generating the print stream automatically (see **Figure 2**). Our work supported this shift by (a) defining *Dynamic Document* as an abstraction for a variable data document and (b) developing applications that supported the creation, editing, and printing of Dynamic Documents.

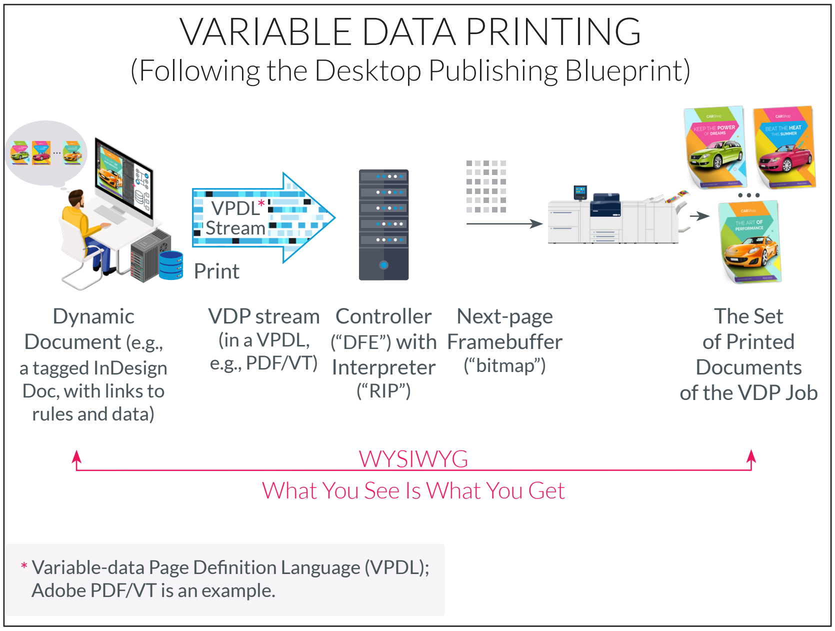

VARIABLE DATA PRINTING
(Following the Desktop Publishing Blueprint)
VPDL* Stream
Print
Dynamic Document (e.g., a tagged InDesign Doc, with links to rules and data)
VDP stream (in a VPDL, e.g., PDF/VT)
Controller ("DFE") with Interpreter ("RIP")
Next-page Framebuffer ("bitmap")
The Set of Printed Documents of the VDP Job
WYSIWYG
What You See Is What You Get
* Variable-data Page Definition Language (VPDL); Adobe PDF/VT is an example.

Figure 2: Variable Data Printing

We defined a *Dynamic Document* to be the binding of a *dynamic template*, a *set of rules*, and *data sets* (see **Figure 3**). A *dynamic template* is a regular document with *tagged* design objects, and any design object can be tagged. A tagged design object is a *Dynamic Object* – its value, say a picture or a text string, is calculated dynamically per recipient, using the rules indicated by the object's tag. The *rules* are specified in a scripting language, and they range from assigning a value from a data field to scripts or SQL queries that compute values from data. One data set is the *Recipients List*, which has one record per recipient.

Conceptually, a Dynamic Document represents a set of *Instance Documents*, where each Instance Document is a regular document individualized for a unique recipient; individualization is by assigning values computed for that recipient for all Dynamic Objects of the dynamic template (see **Figure 3**).

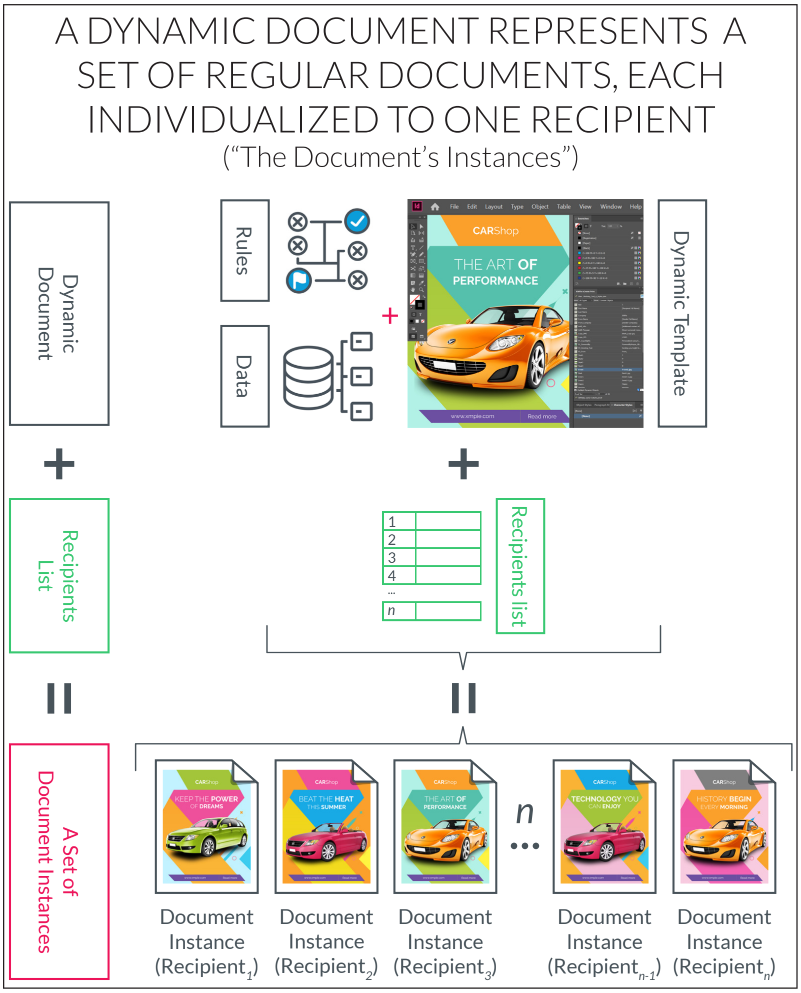

A DYNAMIC DOCUMENT REPRESENTS A SET OF REGULAR DOCUMENTS, EACH INDIVIDUALIZED TO ONE RECIPIENT
("The Document's Instances")
Dynamic Document
Rules
Data
Dynamic Template
CARShop
THE ART OF PERFORMANCE
Recipients List
Recipients list
1
2
3
4
...
n
A Set of Document Instances
KEEP THE POWER OF DREAMS
BEAT THE HEAT THIS SUMMER
THE ART OF PERFORMANCE
TECHNOLOGY YOU CAN ENJOY
HISTORY BEGIN EVERY MORNING
n
Document Instance (Recipient$_1$)
Document Instance (Recipient$_2$)
Document Instance (Recipient$_3$)
Document Instance (Recipient$_{n-1}$)
Document Instance (Recipient$_n$)

Figure 3: Dynamic Document and its Instances

The XMPie software is an example of implementing Dynamic Documents. Its plug-in to Adobe InDesign desktop software enables creating InDesign-based *dynamic templates*.

By defining the variability of a document in terms of its design objects, we introduced ***unconstrained, fine-grained variability***, breaking away from the rigidity of structured business documents. *Moreover, Object-based variability resembled variability in web pages, enabling the bridging of print and digital media for individualized communications.*

Printing Dynamic Documents

Printing a Dynamic Document amounts to printing its Instance Documents. However, rather than explicitly creating and printing Instance Documents, one for each recipient, the process needs to use the E-VPDL to generate a compact VDP stream that encodes the printing of all of these documents. We use the term *composition* to refer to such a process, and the term Composer refers to the process that performs composition.

A key challenge of the composition process is determining the object instances that may repeat and, hence, define Elements for these (and use these Elements when such object instances reappear). Let us zoom into the composition process to understand this challenge and ways to resolve it.

Composition

The *Composer* scans the Recipients List, and for each recipient, it processes the dynamic template. The process instantiates the dynamic objects, generates the needed E-VPDL, and appends it to the VDP stream (see **Figure 4**).

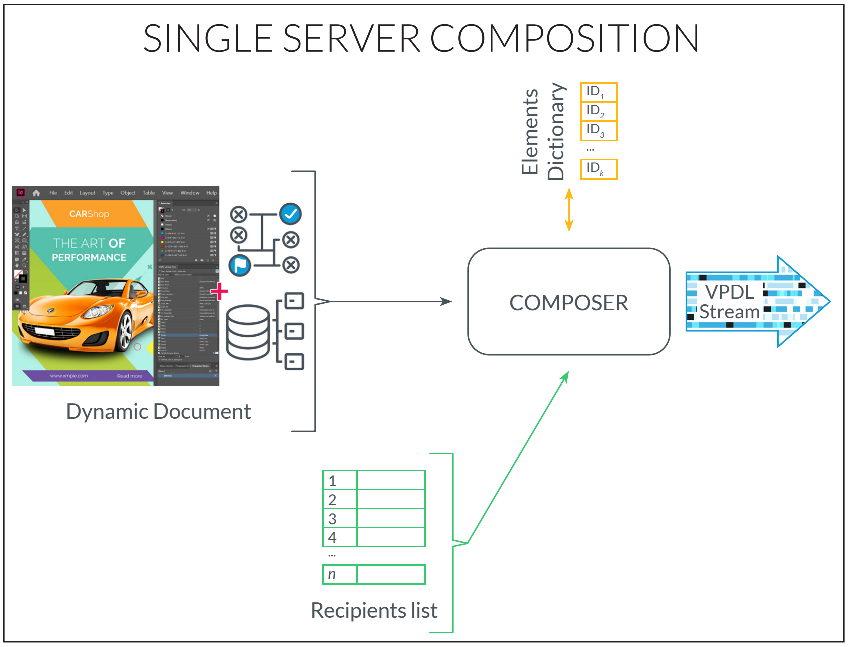


Figure 4: Single Server Composition

In doing that, the Composer needs to distinguish the Instances that may repeat, hence worthy of enabling caching a bitmap of their rendering. Accordingly, we use the term *Caching-worthy* to refer to such Instances.

For Caching-worthy Instances, the Composer emits a *DefineElement* upon the Instance's first appearance and then a *PlaceElement* for every appearance (first and repeat). For all other Instances, the Composer emits regular PDL.

Determining the Caching-worthiness of an object instance is tricky and still open for further research.The XMPie Composer's method decides the Caching-worthiness of Instances of a Dynamic Object based on the object's type. It is based on classifying certain design-objects types – e.g., a graphic frame – as *Reusable* and all others as *Unique*. All Instances of Reusable objects are considered Caching-worthy, and all others are not. In addition, static Objects (i.e., non-tagged design objects) are also Reusable (their content and appearance are not changing, but they reappear for every recipient).

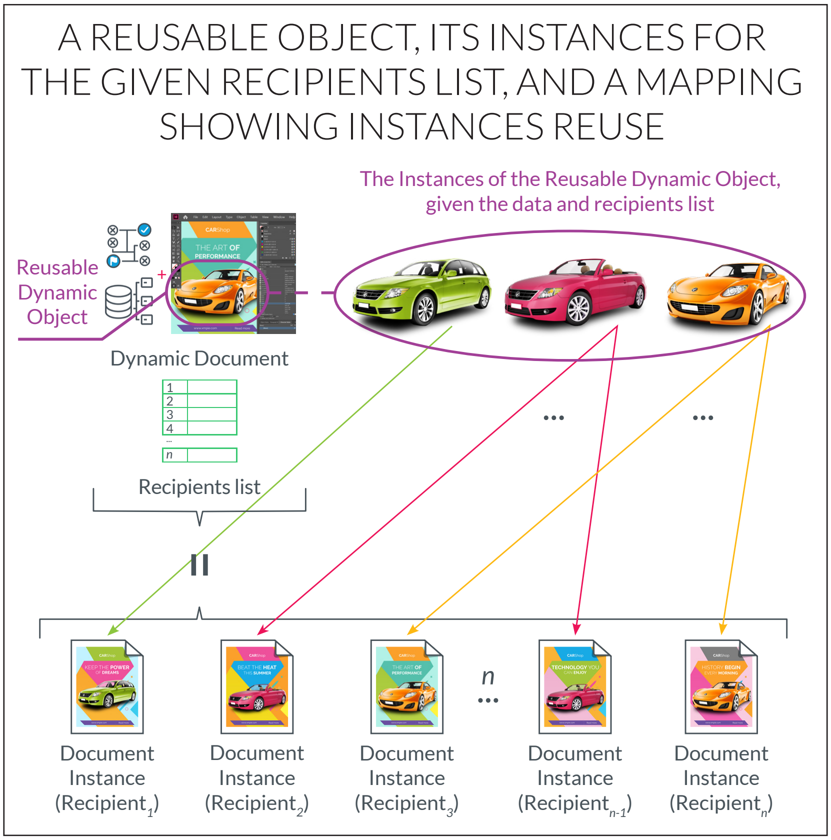

A REUSABLE OBJECT, ITS INSTANCES FOR THE GIVEN RECIPIENTS LIST, AND A MAPPING SHOWING INSTANCES REUSE
The Instances of the Reusable Dynamic Object, given the data and recipients list
Reusable Dynamic Object
Dynamic Document
1
2
3
4
n
Recipients list
n
Document Instance (Recipient$_1$)
Document Instance (Recipient$_2$)
Document Instance (Recipient$_3$)
Document Instance (Recipient$_{n-1}$)
Document Instance (Recipient$_n$)

Figure 5: A Reusable Dynamic Object, Its Instances, and their Reuse

While simple to explain and implement, determining Caching-worthiness based on the object's type is not precise, as it may lead to mistakes, such as caching an Instance that will not reappear (see the Green Car in **Figure 5**). There is room for further research, but since such errors do not affect correctness and their negative impact on performance is low, it is practically acceptable to use them.

Generating the E-VPDL for Caching-Worthy Instances

Handling a Caching-worthy Instance requires giving it a unique name – *Instance ID or ID* – that will be the same each time the Instance appears and then, for each appearance, using the name to determine whether the Instance appeared before. Hence, upon instantiating a Reusable Object, the Composer will compute the *Instance ID* and check whether this is the Instance's *First* or *Repeat* appearance.

For that, the Composer uses a dictionary – The Elements Dictionary – where it manages the names of all Instances that it saw. For example, consider an Instance *X* of a Reusable Object. If the Composer does not find *X* in the Elements Dictionary, then it is the first appearance of *X*, and otherwise, it's a repeat appearance.

<u>Upon a first</u>, the Composer will add *X* to the Elements Dictionary, generate the PDL instructions for rendering *X*, generate a DefineElement command and append it and a PlaceElement, to the VDP stream. <u>Upon a repeat</u>, the Composer just appends a PlaceElement to the VDP stream.

Figure 6 shows how a Composer transforms a Dynamic Document into an E-VPDL stream. At the head of the VDP stream, the reader can see the DefineElement and PlaceElement pairs that resulted from the Composer's seeing the relevant Instances the first time as it was processing the first three recipients. Later in the stream, it reuses these Elements upon seeing the Red and Orange Instances again.

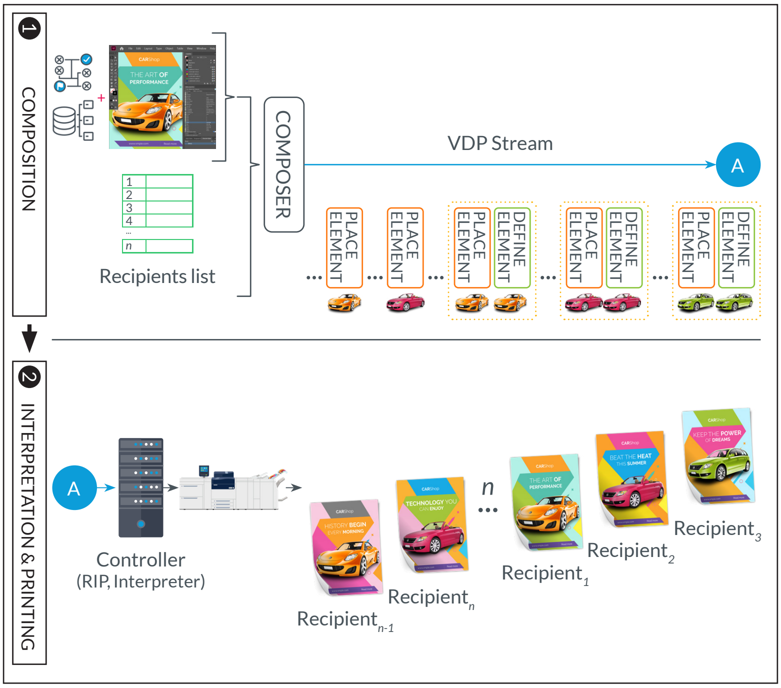


Figure 6: The Composer's Process and How Elements and Reusable Objects Play

Please note that defining an Element is just a recommendation to the Interpreter to process the PDL commands in the definition and cache the resulting bitmap. Similarly, the placement of an Element is a recommendation to use the cached bitmapped rather than reinterpreting the PDL commands in the definition. Whatever the Interpreter's choice, including avoiding caching and, instead, using reinterpretation, the results on paper must be the same.

Modern VDP – A Short Summary

Defining *object-based Dynamic Documents* and *Element-based VPDLs*, and developing the technologies for printing Dynamic Documents, were cornerstones in transforming VDP. The use of Elements was critical for enabling fine-grained unconstrained variability while also addressing the performance challenges:

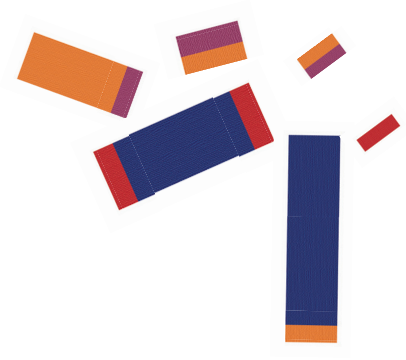
- Composers could save time by avoiding regenerating the PDL for recurring Instances,
- Interpreters' performance could be improved by caching and reusing page bitmaps, avoiding reinterpretation.

Together, these materially removed the performance challenges and made Modern VDP a viable media for marketing and customer experience initiatives, along with digital media.

The fast evolution of color digital printing – color and image quality, speed, substrates, and workflow – is linked with Modern VDP. On the one hand, these modern digital color presses would have little use without Modern VDP's stream of pages that require these presses' top quality. But, on the other hand, without the print qualities provided by these digital color presses, Modern VDP would have been rendered useless.

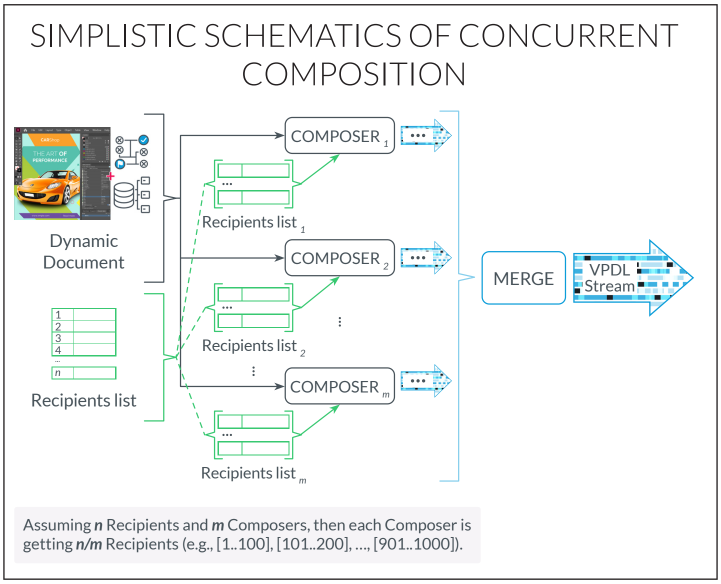
Interestingly, the shift to object-based variability of documents, combined with enabling online access to data in databases and powerful scripting for the variability computing rules, opened the door for bridging print and digital media personalization. Thus, XMPie's early vision of cross-media personalization became real and practical due to inventing Dynamic Documents and everything around it, as discussed above.

In the next section, we will cover Concurrent Composition, closing this exploration of Modern VDP. The performance boost from implementing composition across multiple collaborating networked servers can be game-changing, especially considering the abundance of resources offered with cloud computing.



Concurrent Composition for Modern VDP

A simple method for concurrently composing a Dynamic Document is to break the specified Recipients List into sub-lists and run a Composer per sub-list (see **Figure 7**).



Assuming ***n*** Recipients and ***m*** Composers, then each Composer is getting ***n/m*** Recipients (e.g., [1..100], [101..200], ..., [901..1000]).

Figure 7: Simplistic Schematics of Concurrent Composition

While simple, easy to implement, and highly scalable, it requires adding a merge phase to combine the resulting sub-job VDP streams into one VDP stream for the job. Adding a merge phase will introduce delays, but by running the merge process and the Composers in a pipeline fashion, such delays will be minimized. Furthermore, in cases where the print order does not need to follow the order of records in the Recipient's List, merging is essentially concatenating the sub-job VDP streams in the order they arrive from their Composers, hence a no-delay phase.

The main challenge of the above method for concurrent composition is managing access to the Elements Dictionary. Composers need the dictionary, and they will either share a global dictionary or work a local replica individually. Whatever is the choice, it must satisfy two requirements: <u>minimize redundant definitions of Elements</u>, and <u>minimize delays due to processes coordination</u>.

Let us examine the two alternatives – Shared Global Dictionary and Dictionary Per Composer.

The Shared Dictionary Approach

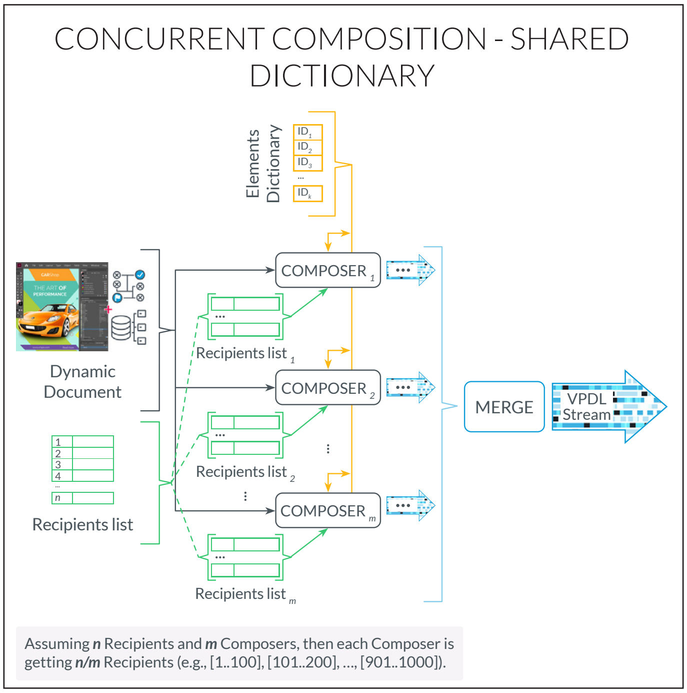
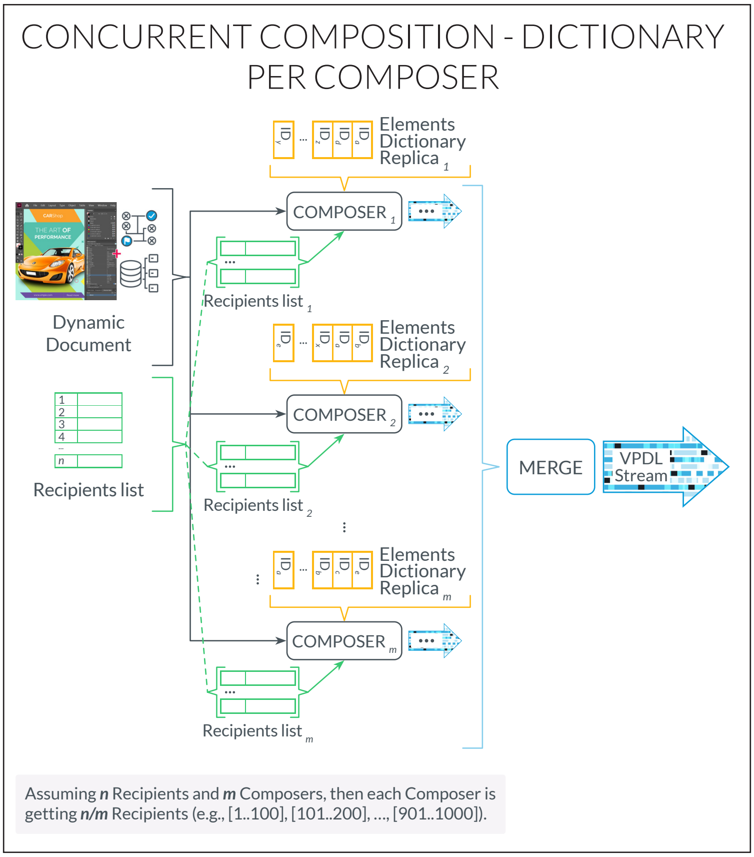

CONCURRENT COMPOSITION - SHARED DICTIONARY
Elements Dictionary
ID1
ID2
ID3
...
IDk
Dynamic Document
Recipients list
COMPOSER 1
COMPOSER 2
COMPOSER m
Recipients list 1
Recipients list 2
Recipients list m
MERGE
VPDL Stream
Assuming n Recipients and m Composers, then each Composer is getting n/m Recipients (e.g., [1..100], [101..200], ..., [901..1000]).

Figure 8: Concurrent Composition – The Shared Dictionary Approach

A straightforward approach for managing the Elements Dictionary is to require Composers to share one global dictionary (see **Figure 8**). Such sharing would guarantee no redundant definitions, but it will not satisfy the requirement to minimize delays due to process coordination. Composers must wait until granted exclusive access before inserting new *ID*s.

The Dictionary Per Composer Approach

The Dictionary Per Composer approach allows Composers to manage a local replica (copy) of the dictionary (see **Figure 9**), eliminating process coordination delays.


CONCURRENT COMPOSITION - DICTIONARY PER COMPOSER
Elements Dictionary Replica 1
COMPOSER 1
Recipients list 1
Dynamic Document
Elements Dictionary Replica 2
COMPOSER 2
Recipients list 2
Recipients list
Elements Dictionary Replica m
COMPOSER m
Recipients list m
MERGE
VPDL Stream

Assuming ***n*** Recipients and ***m*** Composers, then each Composer is getting ***n/m*** Recipients (e.g., [1..100], [101..200], …, [901..1000]).

Figure 9: Concurrent Composition – The Dictionary Per Composer Approach

However, the "minimize redundant definitions" requirement is not satisfied. Such redundancies would happen when several different Composers instantiate a Reusable Object to the same value and, not finding that value in their local dictionaries, consider this the first appearance.[6] As a result, each Composer will create and append a DefineElement command to the stream it generates. Since it is the same

[6] Different individuals may, for example, own the same car, and in a communication related to a person's car the Dynamic Object reflecting the owned car will be instantiated to the same value for different records in the recipients list.

value for all of them, these Element Definitions will be identical, hence redundant.

Is Concurrent Composition Stuck?

Examining the above two approaches, we see that eliminating process-coordination delays and eliminating redundant Definitions are "in conflict." Indeed they are, but insights gained from the Distributed Computing discipline of Computer Science showed us a way out of this seemingly deadlock situation.

Computer Science to the Rescue

We realized that our *Dictionary Per Composer* approach is an example of Computer Science's *Replication Management* problem – the Elements Dictionary is the distributed service, and the local dictionaries are the replicas. Composers use the service to *look up* or *insert* an *ID*. Examining the research, we found a proven method, known as Gossiping, for managing replicas while eliminating delays and minimizing – <u>not eliminating (!)</u> – redundancies.

The generic idea behind Gossiping is that at each replica:

- A background process updates its peers ("gossips") on its replica's state changes
- A background process listens to such updates ("gossip messages"), and if it "hears" state changes that are not reflected in its replica's state, it will update the replica
- The main algorithm does not wait for these background processes.

While Gossiping does not introduce process coordination delays, it minimizes replicas' states differences but does not eliminate them. Hence, only systems with a main algorithm that does not depend on replicas states being identical at all times can leverage Gossiping.

Well, it turned out that our Dictionary Per Composer was precisely such a system. Since the only state-changing operation is inserting an *ID* into the dictionary, the only "error" from not finding an Instance, say *X*, in a local dictionary when *X* is already in other dictionaries is a redundant definition for *X*. But since redundant definitions do not cause correctness problems, Dictionary Per Composer could work with Gossiping.[7]

Dictionary Per Composer with Gossiping

We augment each Composer with two processes: a *Listener* and a *Talker* (see **Figure 10**). Whenever a Composer generates a new Element, it triggers its Talker, which then, in parallel to the Composer continuing its process, gossips – "broadcasts" to the Listeners of all other Composers – the *ID* of the new Element. Listeners are waiting for gossip messages from Talkers. When a Listener hears a gossip with an *ID*, say *X*, the Listener checks whether *X* is already in its dictionary. If it is, it's old news, and the Listener ignores it. Otherwise, the Listener adds *X* to its dictionary, which will prevent its Composer from treating an appearance of *X* as a first appearance, avoiding a redundant definition.

Closing Thoughts on Replication and Gossiping

Gossiping-enhanced Dictionary Per Composer is a scalable solution that fits the massive parallelism enabled by today's cloud computing systems. Realizing that Dictionary Per Composer is an instance of Distributed Systems Replication Management and further understanding that it can sustain local dictionaries not being identical at all times triggered augmenting it with Gossiping. The key for any concurrent composition method is to use replicas and avoid replicas' lock-step synchrony. See **On Gossiping *on page 49*** for further analysis of Gossiping's applicability criteria.

[7] Redundant definitions define the exact same appearance on paper, hence processing one or the other will not affect the result.

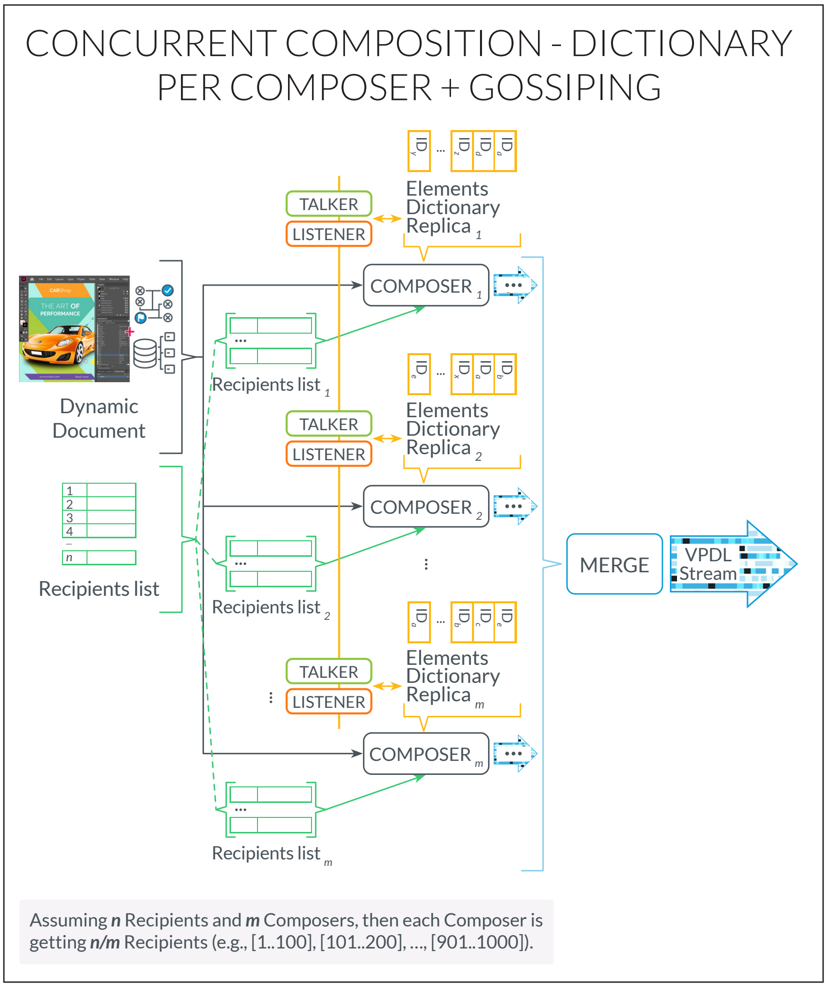
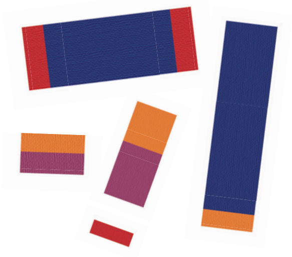

CONCURRENT COMPOSITION - DICTIONARY PER COMPOSER + GOSSIPING
Dynamic Document
Recipients list
1
2
3
4
...
n
TALKER
LISTENER
Elements Dictionary Replica 1
COMPOSER 1
Recipients list 1
Elements Dictionary Replica 2
COMPOSER 2
Recipients list 2
Elements Dictionary Replica m
COMPOSER m
Recipients list m
MERGE
VPDL Stream
Assuming n Recipients and m Composers, then each Composer is getting n/m Recipients (e.g., [1..100], [101..200], ..., [901..1000]).

Figure 10: Dictionary Per Composer with Gossiping

On Merge Anomalies in Concurrent Composition

Before closing this section, it is essential to note that avoiding redundant definitions may result in a merged VDP stream that uses an Element before defining it. Such an anomaly would happen when a Composer sees an Instance, say *X*, the first time but finds *X* in its dictionary; hence, considers this a repeat appearance of *X* and issues just a PlaceElement *X* into the VDP stream it generates. If, in the merged VDP stream for the job, this stream would appear ahead of the stream generated by the Composer that saw *X* first and has the DefineElement *X* command in it. Please check **On Merge and Avoiding Anomalies** on page 51 for a detailed explanation and ways to solve these.



Summary

Modernizing VDP applied the principles of desktop publishing to the VDP world. Giving control to the professionals who design and author documents – the originators – was one fundamental principle of desktop publishing. Applying it to VDP required defining a new document abstraction – Dynamic Document – and providing software for creating, editing, and printing such documents. In addition, the need to satisfy various performance requirements in printing triggered the development of Elements-based VPDLs, composition processes that map Dynamic Documents into E-VPDL streams, and Controllers/ Interpreters that process such streams efficiently.[8] The high-end performance-improvement development was Concurrent Composition with a distributed implementation of the Elements Dictionary service that used local dictionaries and Gossiping to help minimize differences between replicas.

Through its E-VPDL, Modern VDP established an ecosystem allowing developers of applications, controllers/interpreters, and print technology to play together, which accelerated Modern VDP growth and expanded its reach.

[8] Controllers or Interpreters are not covered in this paper but were part of the experience of my colleagues and I at Scitex.

Moreover, Modern VDP's creative variability, together with its use of online data and rules, made individualized print media – paper, packages, labels, physical products, garments, and wearables – a first-class media, exactly like digital. And this first-class status enabled reaching the large and fast-growing markets of Marketing Automation, Customer Experience Management (CXM), and, to a degree, Service Experience Management (SXM).

On Innovations and Customers

An interesting takeaway from exploring how Modern VDP evolved is that its key innovations and developments were not a response to explicit requirements from customers. Instead, these emerged from observations on where technology could lead and why getting there would be valuable for the market. So, for example, there was no requirement for creating an Element-based VPDL as an enabler for variability-rich VDP jobs. Likewise, there was no requirement to create a new document abstraction for graphically rich and creative-friendly documents.

Interestingly, while customers and digital printing-press vendors could imagine fully variable color pages coming out of these presses, they did not formulate what is needed upstream to feed these presses with content that will leverage their capabilities.

To summarize, sometimes customers are not aware of what's possible technology-wise, hence do not formulate a need and do not ask for a solution. Therefore, innovators must internalize that asking customers what they need/want and using their answers as specifications for a future product will not lead to breakthrough, deep technology innovations. Therefore, customers' input and answers are essential for context but should not be a literal specification.

On Modern VDP and Digital Color Printing Synergy

Advancements in digital color printing influenced the evolution of Modern VDP. For example, there was no point in originating sophisticated, engaging, and highly creative variable data documents without the ability to render them in high-quality print. At the same time, without Modern VDP's demand for high-quality digital color print, the use-case of the advanced digital color presses would have been weak.

Acknowledgments

My experience at EFI (1989-1993), Scitex (1996-2000), and XMPie (2000-2019) is the source of ideas, direction, solutions, insights, and industry collaborations that are the foundations for this paper. My XMPie co-founders, Israel Roth and Reuven Sherwin, who worked with me at Scitex before founding XMPie, were intimate partners for analyzing and addressing numerous issues covered by this paper. Others at XMPie contributed to the innovations and developments covered in this paper.

Reuven Sherwin, Idan Youval, Zvika Leibovich, Israel Roth, and David Baldaro gave me extensive feedback, and it sent me back to the writing desk several times.

My colleagues, Andy and Julie Plata, the co-leaders of OutputlinksCG, encouraged me to embark on the road that led to this paper, and they helped with feedback. They also helped by engaging Harvey Levenson, Professor Emeritus at Cal-Poly (San Luis Obispo), University of Houston Professor Jerry Waite, and the industry thought leader Helene Blanchette who gave their feedback. Another colleague and a well-recognized industry analyst, Cary Sherburne, gave me extensive feedback and suggested adding the What's Next section. Finally, Frank Romano, Professor Emeritus at RIT and the driving force behind the Museum of Printing, gave me constructive suggestions.

My friend and colleague, Jeroen Van Druenen, one of XMPie's top customers and the President of its users' group (XUG), gave me feedback that influenced qualitative business observations in the paper.

Ayelet Szabo-Melamed was instrumental in helping publish this paper, including engaging Natalie Broyer for creating the graphics, and David Baldaro for handling all aspects of professionally publishing it. Natalie took my sketches and skillfully brought them to life in the Figures of this paper.

APPENDIX I

Few Observations About the Future of Modern VDP and Tools Supporting It

Where Do We Go from Here – What's Next

In this section, I leave the description and analysis of the rear-view mirror and switch to discuss a few front view items. Some are those enabled by Modern VDP, some are new technologies that may advance Modern VDP, and some are threats to its continued progress.

The enabling items are all around Modern VDP's fit with digital media and its impact on creating rich and engaging customer experiences. The new technologies are the abundance of Machine Learning (ML) and Artificial Intelligence (AI) technologies that are likely to open new horizons for Modern VDP impact or enhance its implementation algorithms. The threat that I see is what I perceive as a re-emerging gap between the data technologies used by Modern VDP and those used by today's digital media.

Modern VDP and Interactive Print

These days, the print piece, which might be a label, a package, a brochure, a statement, or a garment, is "just a touchpoint." In other words, beyond the intended function of the printed object, the brand would want to use it as a touchpoint in a customer journey to keep a relationship with the customer. Hence, these days, the desire to use print as an entry point to a digital dialogue (via a QR code or some form of Augmented Reality) is everywhere. And in a world where individualization is mainstream (massive customization, personalization, massive versioning), the print piece and the touchpoints it leads to, or the touchpoints that lead to it, need to project a holistic individually-relevant experience. Such an experience is like an ongoing dialogue between the brand and the individual.

The only way to achieve such continuity and consistency is when the print personalization is driven on-demand and online, with fine granularity, and by the data and rules that serve the digital media. But, of course, Digital was "born" to work like that, and, in order to play, print must embrace these workflows and architectures.

Modern VDP and its data and rules models provide what's needed for print to play. But unfortunately, the Legacy VDP discipline, which preprocesses the data, preparing it to the performance requirements of its composition process, breaks the linkage with digital and renders print an archaic media.

▪ Print Must Play Equally with Digital Media

Many in Graphic Communications (GC), including professionals in the printing business, analysts, thought leaders, and providers of Hardware and Software technologies, repeatedly explain the benefits of print in engaging prospects or customers.[9] Moreover, in this digital-first era, the promoters of print, in their desire to keep print alive, emphasize the power of print when combined with or supporting digital. The problem with these promotions is that they ignore the challenge of integrating print in a digital world.[10] The way I see it, print's use will be low for as

9 prospects or customers also extend to citizens, employees, or partners.

10 It is clearly a marketing or creative communications challenge; its just that few recognize the depth of the technology challenge and its criticality to enabling the communications or engagemen vision.

long as its personalization methods – interacting with data and rules – will be vastly different from those practiced in the digital-media world.

As discussed above, Modern VDP gives print the needed data and rules models, enabling it to play together with digital, as needed.

The Role of AI

Using AI is a broad issue. There are at least two flavors of it related to Modern VDP. The first one is internal, implementation-related. For example, AI could play a role in helping determine what is reusable and what is not—even going to the degree of what's reusable with a high-enough reusability count and what is reusable but not worth the effort involved in caching and reusing.

The second dimension for using AI is much broader, and it is external in nature. It is all about defining, creating, and executing journeys. At its core, this challenge is a journey builder challenge. And with Modern VDP, customer journeys can include print and digital touchpoints, where each touchpoint is highly creative and individualized. While such journeys are more effective than the non-personalized, digital-only, or print-only journeys, they are more complex to plan and design. And this is where ML and AI can enter and help. Even with print-only touchpoints, the sophistication in creative and variability that Modern VDP can bring to such touchpoints may become a source of the need for AI to help define and design these documents.

Print and Digital's Data Models are Drifting Apart

In this paper and many other occasions, I stated that the data and rules model used for Modern VDP was identical to the one used for digital media. At the time of founding XMPie (1999-2000), this meant using SQL with live access to databases and scripting for computing the variability per recipient as an integral part of the composition process. Most if not all other solutions for VDP at that time used an offline model for working with data. "Offline" means that these models relied on a preprocessing step – not part of the composition – that processed the data in databases, creating one table with all the relevant data for the VDP job.[11] That table is offline because it's a snapshot of the data when it was computed, and it is stale beyond that.

Without online models for data and rules, such as those implemented in the XMPie software, the results of a person interacting with a website and updating information will not show in a follow-up print or even PDF piece. By now, it should be clear that the professionals of the digital media world cannot accept such "behavior."

The problem we are facing today is that Modern VDP's data and rules models appear to have opened a gap relative to the models used by digital media. For example, supporting SQL queries or external computations in your data and rules model is now an old-fashioned approach. These days, methods and subsystems that abstract away from the detail of SQL or scripting, such as Meta's GraphQL and React, are the norm in the digital world. Therefore, for Modern VDP to stay relevant, it must embrace such methods. I will add here that adopting these new methods is not just for the intrinsic benefits of these state-of-the-art methods. Print must also recognize that the professionals who plan and implement digital campaigns will hesitate to integrate print into their work if print requires archaic processes.

[11] In the very early days of VDP, a magnetic tape with the content of such table was used to transfer data to the print providers.

Solutions for Embracing Modern VDP

Years ago, one of the first implementations of Modern VDP was at Scitex, later to become Creo. Between Darwin, the plug-in to QuarkXPress upstream, the VPS E-VPDL, and the Controller now known as the Creo Controller, a Modern VDP offering came to market. Its primary deficiency was the way it worked with data and rules. It required data to be one table and used simplistic rules monolithically integrated within the application. These requirements implied an offline process dependent on preprocessing the data and extracting the relevant data into a one-table structure. And this offline nature of the process meant severe difficulties in integrating it with digital communications. Being a pure desktop solution, Darwin also lacked the scalability needed for large-scale deployments of variable data solutions.

Aside from Darwin, few applications added support for VPS – one of the first E-VPDLs – to their software. These included GMC (Quadient of today), Document Sciences, PrintShop Mail, and Pageflex. Unfortunately, I have no clarity about their data or rules-related functionality.

In founding XMPie, Modern VDP was implemented in full, and the methodologies for working with data and rules were the same ones used by the digital media world.[12] As a result, integrating print and digital in a cross-media campaign was straightforward, streamlined, and live (i.e., always online). In addition, the server-side offerings of the solution enabled resilience and scalability not seen before. The paper takes it further to point out the technologies that allow sizeable scalability, leveraging the abundance of cloud-based resources.

These days, other solutions, even those that specialized initially with Legacy VDP, like OpenText or Quadient, might be offering Modern VDP capabilities. Still, when examining a particular solution, it is crucial to validate that the solution's mechanisms for accessing and interacting with data and rules are online – not requiring a data preparation preprocessing before composition – precisely as the mechanisms used for digital.

12 Online access to real databases, not just a single table, and power scripting for the rules that compute variability.

About the Author

Jacob Aizikowitz is a Computer Science professional who has worked in Software Technology for Graphic Communications problems since graduating from the Technion in Israel in 1977.

After completing graduate studies at Cornell University's Computer Science department, with his fresh Ph.D., he joined the late Efi Arazi (the founder of Scitex) in founding Electronics for Imaging (EFII), where he led EFI's R&D and Engineering through 1992.

Returning to Israel in 1993, he joined IBM Research in Haifa, leading a few commercial initiatives blending imaging research and engineering. In 1996 Jacob joined Scitex, coming back to Graphic Communications. He led the development of Software and Hardware initiatives, including creating the Creo Controller, the VPS language for variable data printing, and Darwin, the applications for designing variable data documents.

Jacob founded XMPie in late 1999, and with his team, broke ground in cross-media, creative VDP, and related disciplines. He led XMPie in various roles to become a well-respected leading brand. In late 2006 Xerox acquired XMPie and asked Jacob to continue leading it as XMPie, A Xerox Company. In May 2019, Jacob retired, and he is now an advisor to entrepreneurs and a board member at the Accrediting Council for Collegiate Graphic Communications (ACCGC).

APPENDIX II

Closing Notes, Facts, Clarifications

On CXM, SXM, and Customer Journeys

Engaging an audience – customers, prospects, employees, citizens, or members of groups – is a key goal of enterprises of all sizes, municipalities, and governments.

In the past, it was done by classical media, such as newspaper advertising, newsletters, radio, or TV advertising.

The emergence of the Internet opened numerous other means for engaging audiences, including email and the World Wide Web. Then, on top of the Internet platform, the social media means of communications emerged and, yet again, added a plethora of new ways for interacting with and engaging audiences.

The ease of access – cellular-based or Wi-Fi – and the abundance of smartphones created a situation where the audience is "always-on." Moreover, individuals can control when and for what purpose they interact with an entity.

While personalization has been a part of the marketing practice since the emergence of Direct Marketing, it has become the norm in this digital, always-on era. Individuals now expect a high degree of individualization and high-quality design. Information alone – say a monthly summary of transactions with the phone company – is not enough. The individuals' experiences are now that the information and the relevancy are packaged with rich media, delivering a colorful, creative, and engaging experience.

And this is precisely where Customer Experience becomes the leading theme. The interaction with the individual is not one time anymore. It evolves, reflecting on the individual's reactions and the enterprise messaging goals. The individual's experience is like an individually-tailored dialogue with the brand, not one of an audience listening to a lecture by the brand. The individual relevancy can be in terms of content or presentation. It evolves over multiple steps, and each step must reflect choices or actions done in previous steps. These days, the business wisdom is that such dialogues will lead to more sales and higher loyalty.

Hence, the focus is on Customer Experience Management and the software tools and systems that enable creating and managing such experiences. The notion of Service Experience Management is similar. The only difference is that it focuses on service-related experiences, for example, a customer calling for service of its refrigerator. The individual's experience still needs to be relevant, clear, engaging, and positively reflect on the brand.

A Customer Journey is sometimes used to describe an individual's overall experience. For example, such a journey may start with the individual getting personalized direct mail. Then, as a reaction, the individual follows through QR code into an individually relevant landing site. And then, based on the individual's interaction with the site, the enterprise sends a follow-up email with a PDF attachment of some individually relevant material (beautifully presented, of course). Finally, after receiving the email with the PDF, the individual clicks a link, lands on a different website, and purchases a product or service. These steps are touchpoints – or stops/stations – in the customer journey. And CXM solutions allow designing and planning such journeys, for the whole audience, with automatic individualization.

Modern VDP is relevant to CXM and SXM because it enables creating print or PDF touchpoints that are timely, engaging, and individually relevant that fit well in the overall customer journey. Its individualization style and detail, combined with its online use of data and rules, works well with the experiences required for such journeys and the capabilities of the digital media touchpoints in such journeys.

On Elements-based VPDLs

In my writing, I followed the definition of Elements and their use as Israel Roth, Reuven Sherwin, and I developed it in VPS – one of the early, if not first, Element-based VPDLs (the late ‘90s while at Scitex). However, the observation that the documents defined by a VDP job share parts of any type, dimension, or location on the page, may have preceded ours. Thus, for example, we could find the notion of Forms in PostScript as a part of a print stream that could be reused. However, it was rarely used due to interpretation-complexities in deciding whether the item described by the Form is reusable as a cached bitmap.

The main difference that E-VPDL brought is the language design choice to be explicit about what may repeat – hence defined as an Element – and what not. In other words, an Element represents not just the repeating content, say, a car image, but also the transformation used to place it on the page. While potentially generating many Elements, that language design choice made the Interpreter’s task straightforward, focusing on implementing the explicit instructions for caching and reuse. However, it is important to remember, especially when AI and ML are flourishing and becoming accessible, that there may be other ways, requiring less precision in the VPDL and leaving the Interpreter more (intelligent) guesswork and optimization space.

In addition, please note that Elements in the VDP stream are just recommendations for caching and reuse. The Interpreter may decide, globally or case by case, to avoid the caching of the bitmaps that result from processing the Element Definitions and reinterpret these definitions upon the placement of Elements. The results on paper must be identical.

On Gossiping

For Gossiping to impact Composition, one needs two main characteristics. The first one is massive parallelism, i.e., many networked collaborating computers. Without it, the Shared Dictionary approach may work fine. It guarantees no redundant definitions, and, due to the small number of "competing" Composers, the chronic delays of this method will be minimal.

The second one is Dynamic Documents with extensive, creatively rich variability. It should be clear that Dynamic Documents with only a tiny number of Reusable Objects require a low frequency of accessing the Elements Dictionary (relative to the overall size of the whole job). Hence, almost nothing to gossip about, rendering Gossiping unnecessary for such a case.

Remember that (a) implementing Gossiping is easy, and (b) having it in the background does not damage performance. Hence, the "price to pay" for implementing Gossiping is minimal. In contrast, its benefit is priceless – a concurrent composition architecture that works well for any degree of Objects Reusability or any scale of parallelism.

On Merge and Avoiding Anomalies

In a concurrent composition architecture, the VDP stream that any Composer generates ("the Local VDP Stream") may have *PlaceElement* commands without having the necessary *DefineElement* commands. Such a situation can happen when a Composer sees an Instance, say *X*, the first time but finds *X* in the Elements Dictionary, indicating that another Composer found *X* already. As a result, the Composer will treat the appearance of *X* as a repeat appearance; hence it will not issue the DefineElement command and only issue the PlaceElement command. Such "anomaly" in a Local VDP stream is not a problem because these local VDP streams never go to an Interpreter. However, it is a problem when such an anomaly appears in the VDP stream for the whole job. And regretfully, due to merging decisions, the stream with the DefineElement for *X* may follow the stream with PlaceElement for *X*, causing the anomaly in the combined VDP stream. The merge process must prevent such anomalies.

Avoiding the *Place before Define* Anomaly

The simplest method for avoiding such anomalies is for the merge process to extract Element definitions from the local VDP streams and append them to the job's VDP stream head. Regretfully, this simple method is inefficient as it requires scanning the various local streams, extracting substreams from them, and moving a lot of content around.

A better approach is for Composers to create two streams – a Definitions-only stream and a Definitions-clean stream. Once done, each Composer will communicate to the merge process the two streams. Next, the merge process will create two streams: a Definitions-only VDP stream and a Definitions-clean VDP stream.[13] Finally, streaming the Definitions stream followed by the no-Definitions stream will eliminate the anomaly.

As we close this section, it is worth noting that the VPS language supports a *job-id* in the VPS stream. The merge process can use this mechanism to avoid the literal merging of the sub-job VDP streams. Instead, it can feed the sub-job VDP streams to the Controller in the required order. The Interpreter will then process them as if they were parts of a VDP stream for the entire job and in the correct order.

Finally, I would add that early in 2019, XMPie was granted a patent that describes how Composers, merge processes, and Controllers/Interpreters can achieve even better performance by collaborating more intimately. The method breaks away from adhering to the "standard" E-VPDL streaming protocol, allowing Composers to communicate Elements directly to the Controller. As such, it enables the pipelining of composition and the Interpreter's processing of Elements Definitions to create the needed cached bitmaps. Furthermore, since Element Definitions are communicated directly to the Controller, the sub-job VDP streams have no Elements Definitions, eliminating the anomalies.

[13] The merging of the Definitions stream is easy since there is no need to preserve order.

Legacy VDP

The term "Legacy VDP" refers to the VDP practice that supports the printing in batches of individualized business documents, typically Transactional or Forms documents.

These business documents have a rigid and simple structure. Certain parts of such documents' pages are identical to all recipients, and other parts are unique per recipient. The rigid and plain design of these documents allows capturing the common parts as a page background that is identical to all pages of all recipients.[14] As an example, consider Forms. The graphics of the lines and text that define the Form's fields are a background.

Early phases of the Legacy VDP practice were based on first using classical offset or similar printing technology for printing the backgrounds on batches of pages. Then, using inkjet printing heads, print on these pages (also known as "shells"), in designated areas, the textual variable data. It should be clear that this practice was viable only because of business documents' rigid and simple structure. However, it should also be clear that

[14] At some point in the evolution of Legacy VDP, such backgrounds were refined to be backgrounds to all pages that are targeting a specific segment. This resulted in options to have few such backgorunds in a job, and switch the background when moving from one segment to another.

this method influenced communicators' vision about the type of documents possible with VDP. In a way, for the communicators, the implementation was given and non-negotiable; hence, they never challenged it and planned their communications to adhere to the Shells and Variable fields constraints.

Interestingly, the technologies for VDP kept adhering to the Shells and Variable fields concept, even when the printing technology evolved to full-page digital printing, where each page could be completely different from the page preceding it. This adherence was reflected in the protocols that these digital print devices used to support VDP, which aimed at mimicking the notions of Shells and Variable fields, "hiding" the true variability capabilities of the print device. Indeed, supporting such protocols made these digital print devices capable of supporting the existing world of VDP applications, hence immediately worthwhile. In addition, it allowed migration from Shells through offset and Variable fields via inkjet into full digital printing. However, at the same time, these protocols kept "hidden" the ability of every page to be completely different from the one preceding it.

The reality is that for the professionals upstream – authors, designers, and communicators – the world of VDP continued to be the rigid and structured one they were all accustomed to, completely disconnected from the true capabilities of digital printing. This gap became even more acute once digital color printing came to be (approximately 1995).

Only a few professionals in VDP challenged this status quo, wondering why VDP is still adhering to the Shells and Variable fields metaphors when the printing technology enables total variability everywhere on a page and across all pages. In Scitex, my team and I were among the challengers; teams at Adobe were also among the challengers. This paper is about why we challenged the status quo and how the solutions evolved, bringing the Modern VDP era.

In closing, the E-VPDL component of Modern VDP is a general-purpose approach for variability. One can specify the printing of Legacy VDP documents and enable the efficient processing of such specified print streams by using an E-VPDL. For example, a Form layout would be specified as an

Element in the E-VPDL stream, and it would be placed (i.e., PlaceElement) at the beginning of every page. Upon processing such an E-VPDL stream, the Interpreter would cache the interpreted Form design as a bitmap and reuse it with every page. So, the general-purpose variability of E-VPDLs would automatically adjust itself to the benefits of Master Page and Variable fields if the design calls for it. Yet, it will work just as well for a more sophisticated design, whereas the Legacy VDP approach cannot even support it.

Desktop Publishing

Few technologies that emerged around 1983 triggered the vision and implementation of desktop publishing:

1. Bitmap Displays (Xerox, Apple, and others),
2. Laser Printers, including Apple's LaserWriter
3. PostScript (Adobe), PostScript Interpreter in the LaserWriter (Adobe and Apple),
4. Page Layout Software that uses the bitmap display, and PostScript for printing (Aldus PageMaker)

The Bitmap displays enabled envisioning software that would support designing and authoring documents interactively, based on what the user sees on the screen. Such software aimed to provide a natural and intuitive experience to the originators of documents – authors and creative professionals. These Word Processing or Page Layout software applications invented the *interactive (digital) document experience*. The image on the screen mimicked the expected look of the printed page, and the applications' tools supported all sorts of interactive document-editing functions.

Printing the document required the application to translate its document representation into instructions for the printer.[15] Using PostScript guaranteed that the application would be able to define how to render in print anything that it enabled creating at the desktop. And a Laser Printer with a Controller and a PostScript interpreter could process such a specification and print pages that matched the bitmap display.

The hallmark "feature" of the above process was that what users saw on their screens, as they were creating and editing their documents, was what would appear in print. Hence, **W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et ("WYSIWYG"). And this characteristic became a synonym for desktop publishing.[16]

Desktop publishing was a major disruptor to the printing and publishing industries. It started small, and professionals considered it to be a toy. Eventually, in a classic Disruptive Innovation move, it took over all older methods (and players) and became the de facto standard in publishing.[17]

The glue that tied all desktop publishing components was the PostScript PDL. Its programming language power and device-independent imaging model made it the hub of an ecosystem. It opened the publishing industry to the developers of applications, controllers, interpreters, and print devices. It meant that application development could move forward without relying upon the progress of a printer or a Controller development. Similarly, developers of Controllers did not have to coordinate their actions with specific applications or features.

15 In all systems it was also possible for the applications to use the system's print driver, and that driver would generate the needed PDL, e.g., PostScript. Yet, the high-end applications, which are those we focus-on in this article, typically created the PostScript on their own.

16 For the first 10 years, desktop publishing did not support device-independent color. It was added during the early '90s.

17 The late Harvard Professor Clayton M. Christensen defined and researched Disruptive Innovation (https://hbr.org/2015/12/what-is-disruptive-innovation).

In closing, I would add that introducing PostScript was a disruptive innovation by itself. The incumbent leaders of the printing and graphic arts industries – Scitex, Hell, Crosfield – did not realize the tsunami effect that PostScript would bring. They perceived PostScript as a toy for personal publishing, not the professional level they mastered and owned. These leaders do not exist today, providing yet another example of the power of the Disruptive Innovation phenomena.

Glossary

Bitmap	A pixel array, having one or more bits per pixel.
Composer	A Composer is a process that performs composition – creating a VDP stream for printing a Dynamic Document.
Concatenation-style VDP stream	A VDP stream that is a concatenation of the regular PDL print-streams of the individual documents of the VDP job.
Controller	A Controller is a networked computer that manages a printer. It typically includes the Interpreter subsystem (see Interpreter).
CXM	Customer Experience Management.
Dynamic Document	A Dynamic Document is a document abstraction that represents variable data documents.
Element	An Element is a named group of instructions in the VPDL that represents a likely-to-repeat Instance of an object of the Dynamic Document.

Elements Dictionary	The Elements Dictionary is a dictionary where entries indicate the Instance IDs of Objects for which Element Definitions are already part of the VPDL stream.
E-VPDL	A Variable-data Print (Page) Definition Language ("VPDL") that supports Elements.
Framebuffer	The Framebuffer is an internal memory hardware of a printer that stores the image (bitmap) of the to-be-printed page.
Gossiping	Gossiping is an algorithm for managing replicas' states in implementations of services, where multiple networked servers collaborate to implement a service. Implementing the Elements Dictionary in the Dictionary per Composer method is an example of a distributed computing implementation of a service.
Instance ID	Instance ID is a unique name that a Composer assigns to an Instance of a Dynamic Object (i.e., the value of that Dynamic Object for a given recipient record).
Interpreter	The Interpreter is a Controller subsystem that processes input print streams and converts them to instructions and data to the printer, resulting in printed pages that match what the print streams specified.
Listener	The process that "listens" to gossip messages. It is part of the implementation of Gossiping in Dictionary per Composer.
Object-based variability	Every design object in a document can be variable. Typically, no restrictions on the design object's type, dimensions, or location.

PDF/VT	PDF/VT is Adobe's VPDL, which is now a standard.
PDL	PDL stands for a Page Definition Language (also Page Description Language, Print Description Language, or Print Definition Language, which are all the same).
PPML	Personalized Print Markup Language (an XML-based VPDL)
Recipients List	A list of records, each representing one recipient. A recipient typically represents an individual, but it may represent any individual target such as a product, a population segment, or a branch in a chain.
Rules	The logic for computing values for Dynamic Objects, given the data and the recipient record. For example, it could be selecting a value from a database field or a complex script that computes a value.
Talker	The process that sends gossip messages. It is part of the implementation of Gossiping in Dictionary per Composer.
VPDL	Variable-data Print Definition Language.
VPS	Variable Print Specification. One of the first VPDL; was defined and developed by Scitex.
WYSIWYG	What You See Is What You Get (see Desktop Publishing).