Endless Documents : A Publication as a Continual Function[♦]

John Lumley Roger Gimson and Owen Rees
Digital Media Systems Laboratory
HP Laboratories Bristol

HPL-2007-111 (R.1)
21st September, 2007[*]

XML; XSLT; SVG; document construction; functional programming

Variable data documents can be considered as functions of their bindings to values. The Document Description Framework (DDF) treats documents in this manner, using XSLT semantics to describe document functionality and a variety of related mechanisms to support layout, reference and so forth. But the result of evaluation of a function could itself be a function: can variable data documents behave likewise? We show that documents can be treated as simple *continuations* within that framework with minor modifications. We demonstrate this on a perpetual diary.

[♦] ACM Symposium on Document Engineering 2007, 28-31 Aug.2007 Winnipeg, Canada

[*] Internal Accession Date Only

© Copyright 2007 ACM Approved for External Publication

Endless Documents: A Publication as a Continual Function

John Lumley
Hewlett-Packard Laboratories
Filton Road, Stoke Gifford
BRISTOL BS34 8QZ, U.K.
john.lumley@hp.com

Roger Gimson
Hewlett-Packard Laboratories
Filton Road, Stoke Gifford
BRISTOL BS34 8QZ, U.K.
roger.gimson@hp.com

Owen Rees
Hewlett-Packard Laboratories
Filton Road, Stoke Gifford
BRISTOL BS34 8QZ, U.K.
owen.rees@hp.com

ABSTRACT

Variable data documents can be considered as functions of their bindings to values. The Document Description Framework (DDF) treats documents in this manner, using XSLT semantics to describe document functionality and a variety of related mechanisms to support layout, reference and so forth. But the result of evaluation of a function could itself be a function: can variable data documents behave likewise? We show that documents can be treated as simple *continuations* within that framework with minor modifications. We demonstrate this on a perpetual diary.

Categories and Subject Descriptors

I.7.2**[Computing Methodologies]**: Document Preparation — *desktop publishing, format and notation, languages and systems, markup languages, scripting languages*

General Terms: Languages

Keywords: XSLT, SVG, Document construction, Functional programming

1. INTRODUCTION & MOTIVATION

The *Document Description Framework*[1] (DDF), is an architecture for variable content documents based on separation of data, logical structure and presentation, and a view of the document as an extensible function. Using an XML tree as the main syntactic representation, constructional semantics are supported by sections of XSLT and an SVG-based geometric presentation by a hierarchical tree of layout instructions[2]. A suite of support tools implements the evaluation of such sets of such documents on variable bindings, managing the creation, merging, binding, evaluation and observation of the results into final printable forms, such as PDF.

Viewing a DDF document as effectively a function (or fragments of a functional program) has been very useful in engineering solutions to several documentation problems by making separation of different scopes and roles relatively easy and robust. But we were always aware that we were only scratching the surface of some of the possibilities. One of the types of document we were curious about was that of a 'continual' document, that is one that may continue and grow as new information is added, *but for which it is meaningful, and useful, to observe its presentation content at stages during its binding 'life'.* Examples of such documents are:

- A diary where entries are added in a sequential order and where the bulk, but not all, of the new data appears at the 'end' of the major sequence.
- A patient's simple medical record. As new data is added, from a variety of sources, content can be added at several parts of the document. For example a temperature reading may add to a chart *and* a table. Lab tests might add both a complete new page and an additional entry in an earlier summary table.

We built such a document within our framework, specifically one for which presentation is generated at each stage of data binding *only* on the consequences of the new data. To put it differently, we build as much of the document 'presentation' (including layout) as possible, as early as possible, leaving large sections that will remain invariant for the remainder of the lifetime of the document, including the binding of 'new' data. In this paper we report on how we can define and process a document which *can be continual.*

We use the term 'continuation' loosely here, to describe both an indication that variable data is 'to be continued..' and also program sections that support the continual functionality.

2. A DOCUMENT WITH 'CONTINUATIONS'

Our challenge is to define a document which is even *capable* of accreting new information and adding to its display at each subsequent step of data binding, and ensure that the processing machinery can evaluate this correctly. We made experimental design choices which will restrict the types of document definable:

- 'Continuation' is indicated actively in the binding of data (by embedded 'to be continued..' markers) rather than passively (e. g. the absence of 'stop' signals or process-step parameters.)
- Accretion of additional data *does not modify* elements already bound and processed. For example the presentation of a statement that is an existential qualification over data ('None of these accounts are overdrawn'), would require removal or modification if additional data conflicts with this.
- 'Continued' data is substantially the same type as that already bound - the continuation implies 'and more of the same'.
- The continuation is at the 'end' of the data stream.

2.1. An Eager Diary

We'll show the approach with a simple diary, to which new 'days' are bound in succession. The pictures are the actual presentations resulting from the progressive evaluation of this diary within a DDF processing workflow. This diary contains three sections

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, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DocEng'07, August 28-31, 2007, Winnipeg, Manitoba, Canada.
Copyright 2007 ACM 978-1-59593-776-6/07/0008...$5.00.

- A title page, with a summary of each day contained in the diary.
- A page for each completed day, containing data for that day, gleaned from newly bound input.
- A 'work-hours' chart showing the working time for each completed day in the week.

The variable data to be bound is a simple XML structure of several `<day/>`s with simple descriptions as shown in Figure 1.

```
 <diary>
   <day date="Mon" start="9" end="17">Got up</day>
   <day date="Tues" start="10" end="17"
 >Went to work</day>
   <day date="Wed" start="8" end="18">Travelled</day>
   <contd/>
 </diary>
```

Figure 1. An example of some days for the diary

We use `<contd/>` as the specific marker: the document anticipates that it will be the last child of an entry. The diary is bound progressively to three different sets of days in a workflow (Figure 2). The 'empty' diary (actually a template), contains boilerplate content and necessary program to respond to variable binding. We then evaluate this document over the data for several days with a continuation (setting one-off parameters such as page size), yielding a result document *which is both presentable, and still a 'function'.*

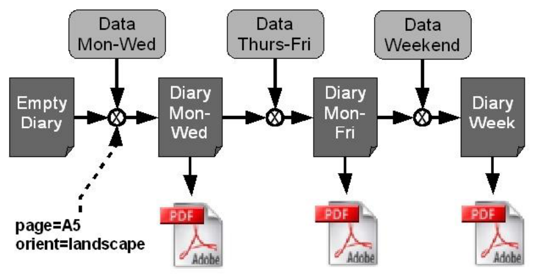

Figure 2. Process graph for 3-stage bindings to a diary

After the first binding we have content for Monday to Wednesday appearing in all three sections with a presentation shown in Figure 3. (We can display visible markers where the continuations sit, though ordinarily there is no presentation at these points.)

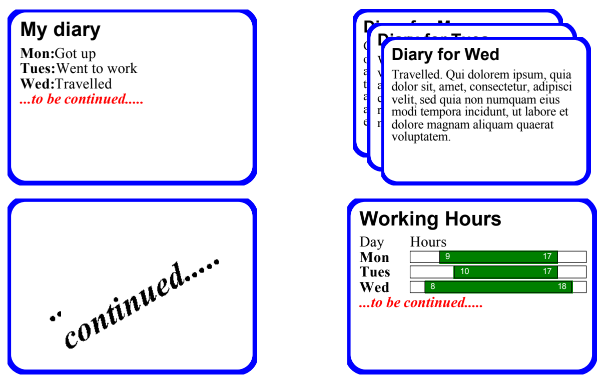

Figure 3. Pages of the diary after its first stage of binding

There are now three continuations – in the summary, between 'day' pages (displayed as a separate page here) and at the foot of the working-hours diagram. At each of these points in the document's presentation description there is now new program code that will consume new content and add new presentation.

We can now proceed to bind data for the second part of the week: these program sections now operate, generating material for Thursday and Friday (but *not* Monday to Wednesday as that presentation already exists in the document at this stage). Since we claimed the week still continued in the new data this processing leaves additional continuation program points, as shown in Figure 4.

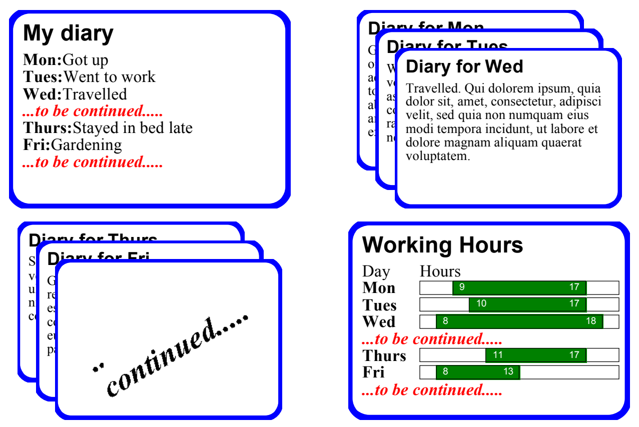

Figure 4. Diary pages after its second stage of binding

The third stage completes it with the weekend (without a continuation), giving a final result shown in Figure 5. This has no remaining program and will not alter with the addition of any further data.

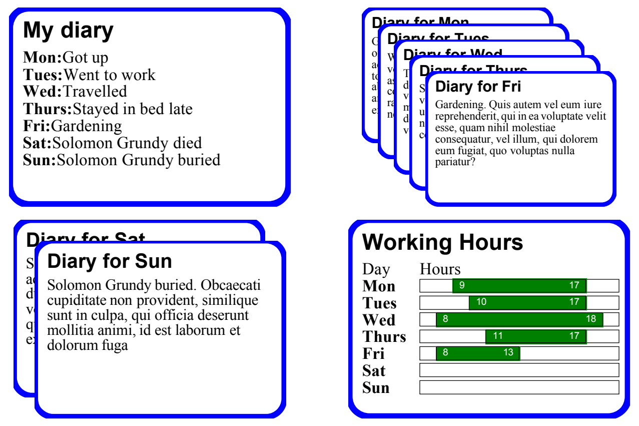
My diary
Mon:Got up
Tues:Went to work
Wed:Travelled
Thurs:Stayed in bed late
Fri:Gardening
Sat:Solomon Grundy died
Sun:Solomon Grundy buried
Diary for Mon
Diary for Tues
Diary for Wed
Diary for Thurs
Diary for Fri
Gardening. Quis autem vel eum iure reprehenderit, qui in ea voluptate velit esse, quam nihil molestiae consequatur, vel illum, qui dolorem eum fugiat, quo voluptas nulla pariatur?
Diary for Sat
Diary for Sun
Solomon Grundy buried. Obcaecati cupiditate non provident, similique sunt in culpa, qui officia deserunt mollitia animi, id est laborum et dolorum fuga
Working Hours
Day
Hours
Mon
Tues
Wed
Thurs
Fri
Sat
Sun
9
17
10
17
8
18
11
17
8
13

Figure 5. Diary pages after its third and final stage of binding

3. DEFINING THE DOCUMENT

To enable the document to have this ability to self-continue as a function if required, it must have extra capability in two broad areas:

- necessary (possibly modified new) program should appear in the result document and
- program and layout constructions behave appropriately when continuations appear in the variable data to which they respond.

Eventually we'd like this to be declared as simply as possible, for example by adding a 'can be continued' declaration to some description of the anticipated data ('*diary/day* continuable') and automatic analysis will add such capability. In this paper we add the necessary features 'by hand' to template documents to achieve these results. Our approach is to add declarative annotations to program elements describing lower-level requirements and then (as the program is itself XML) modify results according to these annotations.

3.1. Propagating Program

If our document has program sections that builds the day 'page', then these elements must be mapped forward into the result document, so it can be used for days to come. In DDF documents most program elements are `<xsl:template/>` trees. Because we evaluate the effect of binding a DDF document on data by compiling the document into an XSLT program (which is then run with the data set as input), we can mark program elements we want 'copied through': the compiler will produce appropriate 'code-producing' code. Simple attributes (`@ddf:retain`) identify such elements.

Some program elements may need to be carried forward in an *evaluated* state as their dependencies might only appear in an early workflow stage. For example, the diary may be parameterised in terms of a standard page size (*A4, A5, Letter*) and orientation (*portrait, landscape*): this is set externally at the start (Figure 2) and not subsequently. In this case the `<xsl:variable/>` elements describing page dimensions should be carried forward as bound static values and not computations. Again the compiler can detect the annotation and generate code to build a suitable result program element. (A compiler that can analyse the internal dependency graphs for variables in the source could determine this need and arrange for complete propagation of a binding.)

3.2. Responding To Continuation

We have to construct the 'program' of the document so it can respond correctly to 'and some more later' appearing in the input. For this we must tackle three issues:

- Detecting the presence of the continuation. This could either be done at 'top-level' or distributed to points where decisions are taken based on bound data.
- Arranging for these decision points to add calls to future processing when encountering continuation.
- Defining program and layout operations to respond correctly.

Firstly we must detect the continuation and when one appears add declarations for eventual reprocessing (i.e. code to trigger new processing). If the document is built in an XSLT 'push' form, with templates matching suitable data instances and patterns, then we can simply add templates to match continuations, which contain code for future processing where we would ordinarily add content. These templates operate wherever continuations are detected in incoming data and leave program to generate new content subsequently at appropriate places in the presentation. In our example three places in the presentation processed 'day' data (summary, page-per-day and worksheet) - each of these responded correctly to the continuation marker appearing in its input.

For 'pull-mode' XSLT forms (controlled iteration over data) specific alternative control flows need to be added . For simple iterations this is relatively straightforward, but complex requirements (such as processing the last day specially) may make this difficult.

3.2.1. Partial Geometric Layout

Ultimately the final document presentation is graphical and geometric, and in the case of DDF a (large) SVG-based tree. This may contain sections of program that will be executed later, generated as a consequence of continuation. As such program is written in XSLT and clearly distinguishable, we ignore such sections in 'observing' the final result, when we produce a graphical rendition for a browser or a PDF document.

Program sections that define geometric layouts, which in DDF are combinator functions over sets of children, present much more of a challenge. A detailed knowledge of the semantics of the combination are required to design how (or even whether) partial sections can be evaluated.

For example a uni-directional 'flow' of a sequence of parts can (by associativity) be replaced by a flow of partial flows; completely bound flows can be evaluated and treated subsequently as an atomic piece. (Macdonald [3] describes some of the problems involved in geometric layout of partially bound assemblies, focussing on exploiting invariances such as this)

But something as simple as a self-sizing table can't be completely broken down into independent sub-tables, as the size of cells can depend upon the size of new-cells 'yet to be bound'. So in these cases we may have to resort to speculative evaluation, providing test choices to determine whether pre-evaluated or re-evaluated alternatives should be used.

In our diary we've managed to produce output avoiding speculation, by making each day's main section a separate page and using simple flows where continuation occurs *within* a page. Elements on the summary page are placed within a simple flow and the workflow table is actually a flow of several 'constant width' tables.

4. STATUS AND FUTURE DEVELOPMENT

It is *possible* to define and process certain types of continually active documents within DDF, as the example in this paper shows. To take this further we need to examine i) a model for supporting speculative evaluation, ii) approaches to partial evaluation of geometric layouts, iii) adding fuller analysis of internal dependencies, such that the consequences of bindings can be propagated as far as possible and iv) how such models can be generated automatically from 'non-continual' document samples. The approach will need to be refined in terms of theoretical functional concepts such as closures.

5. REFERENCES

[1] Lumley, J., Gimson, R. and Rees, O. A Framework for Structure, Layout & Function in Documents. In *Proceedings of the 2005 ACM symposium on Document engineering*. 2005.

[2] Lumley, J., Gimson, R. and Rees, O. Extensible Layout in Functional Documents. In *Digital Publishing, Proc. of SPIE-IS&T Electronic Imaging, Vol 6076*. 2006.

[3] Macdonald, A., Brailsford, D. and Lumley, J. Evaluating Invariances in Document Layout Functions. In *Proceedings of the 2006 ACM symposium on Document engineering*. 2006.