A Dataflow Graph Based Approach to Web Application Development

DAVID BRÜLL, BJÖRN SCHWARZER, SEBASTIAN OSCHATZ,
Meso - digital media systems design
D-60329 Frankfurt
http://vvvv.meso.net

Prof. Dr. ARND STEINMETZ
h_da - University of Applied Science ,Darmstadt
Dept. of Media
D-64295 Darmstadt,
http://www.media.h-da.de

bruell@meso.net, schwarzer@meso.net, oschatz@meso.net, steinmetz@media.h-da.de

Abstract: - This paper introduces a methodology for developing web applications within a dataflow graph based environment. In the first section we shortly introduce the programming environment we have extended for this work. The consecutive discussion focuses on the advantages of handling web resources by detaching them from the file system and leads to a conclusion for the implementation of the web-server components into a dataflow graph. We also present a dataflow based method for generating markup content and its style allocation.

Key-Words: - visual, dataflow, web, application, development, HTTP, HTML, XML, vvvv

1 Introduction

Since the web has grown to a worldwide accepted platform for presenting, exchanging and consuming knowledge, goods and services, people are subjected to software development more than ever. Standards for directing the increasing possibilities of web technologies emerge in shorter intervals. Those standards mostly attend conventional methods of web application development like code based imperative approaches. Editing in WYSIWYG mode is the most famous complement to this evolution. Although, by trying to skip the code's abstraction, a basic understanding of the underlying processes is lost. In this paper we present an application of web standards with the aid of a visual dataflow programming paradigm.

Our work should be seen as an experimental research project and is not meant to completely substitute traditional techniques. In our opinion today's development standards are subjected to firmly adopted programming methods, which hinders us to look beyond certain patterns. Our theory indicates the advantages of the abstract visualization of today's web standards.

2 Related Work

There have been many attempts to proof the advantages of visual programming approaches. Nan C. Shu [1] approved the cognitive benefits of visual programming and Margret Burnet [2] classified the different visual paradigms. These contributions are proven to be still useful to today's development situation as in different fields the dataflow paradigm is constantly established. As an example Max/MSP[3] can be listed for the area of multimedia applications and LABView[4] for the field of laboratory automation.

Although the research in the area of dataflow based web application development is not very old, there have been a few works similar to our approach such as the VIPERS [5] system and the Lixto transformation server [6]. The Lixto TS e.g. provides an XML based system for designing a so called *infopipe* to build a data aggregation step by step. The processing model of the Lixto TS contains four component types for extracting, integrating, transforming and delivering information. The most important difference of both, VIPERS as well as Lixto TS to our own approach is the actual purpose for using the systems. While our approach shows how to program and provide a dynamic web application during runtime, VIPERS and Lixto TS are more suitable to preprocess programs. Developers usually use scripting languages like PHP and Perl to add dynamics to their static HTML pages. Our system shows, that a dataflow graph is able to provide the same flexibility without resorting to additional scripting languages.

3 The Environment

For the demonstration of our approach, we use vvvv [7], a visual programming environment whose core metaphor

is a dataflow graph consisting of nodes and links which can be manipulated via an intuitive graphical interface during runtime.

vvvv is designed to facilitate the handling of large media environments with physical interfaces, real-time motion graphics, audio and video that can interact with many users simultaneously. Furthermore vvvv includes a wide range of possibilities to manipulate strings in general as well as special functionalities for handling XML data strings [8]. A build in HTTP server enables the supply of web resources directly out of the graph [8].

Each node within a graph corresponds to a particular procedure whose input parameters are given by the upper edge "input-pins" of the node. The procedures result occurs on the lower "output-pins". Each output pin can be linked to an arbitrary number of input pins of other nodes. vvvv's graphical user interface prevents the developer from connecting nodes of different data types. In textual programming environments this feature is often left to the compiler and is a second step after authoring source files. The computation of the graph occurs in real-time, by splitting the time into calculation frames. With each calculation frame vvvv evaluates the complete graph. Therefore each node requests upstream nodes to evaluate their outputs. Various caching strategies avoid duplicate calculations in one frame. This method ensures the availability of every procedures result for the current frame.

A node can either be supplied by the environment or can be a vvvv graph itself (Sub-graph). Thus every graph can be at any other graph's disposal as a node.

4 Dynamic and File-less

Our approach shows a server side operation which is detached from the file system. By requesting a resource from the server, the corresponding string representation of that resource is held in memory and can be manipulated by the dataflow graph in real-time. The HTTP server itself receives the final string of the graph's calculation and sends it back to the browser.

Considered from an ideological point of view, this method results into a better perception of what a web server does, because the string can be gripped in every state of the calculation. This leads to new opportunities of debugging web applications, since in conventional methods the developer depends completely on the processors support. Hence, the dataflow programming paradigm itself enables better debugging in an application. Furthermore, the string based approach of handling web resources provides unskilled developers with an understanding of the interaction between the application and the server by reducing the server's activity to its core role. By passing data through the file system we loose information about path and extension of a resource. As this information is needed by the server in order to send the corresponding MIME-type back to the client, it has to be manually re-assigned on the application level, which, on the other hand increases the flexibility of the content administration. Of course, it is also possible to build a graph which itself accesses the file system and extract the missing information and sets the appropriate properties automatically. This would then reflect the usual server behavior.

Resources with MIME-types that differ from *text*, e.g. images, are handled within vvvv as binary strings. In order to make those resources visible to human eyes, vvvv provides nodes for transforming binary strings to images and vice versa. Since vvvv is designed to solve tasks involving media like video and 3d animation, it is accordingly suitable for image transformations.

5 The Dataflow Setup

Within the dataflow paradigm, it is not the execution order of operations that has to be determined, but the dependence between nodes. Therefore, the order of operations execution results implicitly from the way nodes are being linked to each other [9]. Technologically the node's arrangement is irrelevant as long as their linkage is correct. Simply the cognitive interpretation of a dataflow graph is much easier if all nodes were arranged in a linear fashion.

Detractors of visual programming often review the bad readability of dataflow diagrams. As we know from experience, this effect can be avoided by finding the right abstraction for the processes. By designing new nodes for a dataflow graph it is required to optimize the given functionalities to the developer's perception to ensure efficient cognition of the dataflow.

Hence, for this work it is also essential to implement the processing of a web server request according to that pattern. Every activity starts with a HTTP request to the server and results in the delivery of the corresponding resource to the client. Fig.1 shows a straightforward implementation of the web server as one operation node which expects the extraditable resource as its input and delivers request header information as its output. This implementation causes recursive linking and overlapping connections if the resource properties depend to the request parameters. To avoid confusing diagrams it is important to reduce the necessity of recursive linking.

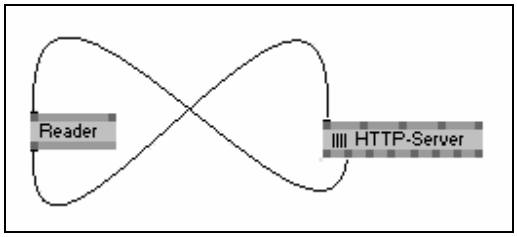

Fig. 1: Recursive linking

As a logical conclusion one has to split the server operations into two nodes: The HTTP receiver node and the HTTP server node. The HTTP receiver node reads all parameters of the browsers request and provides them at its output pins. The HTTP server node on the other hand accepts the final resource as a string at its input pin(s) and delivers it to the client.

A graph of any given processing purpose can now be linked between the HTTP receiver and the HTTP server node.

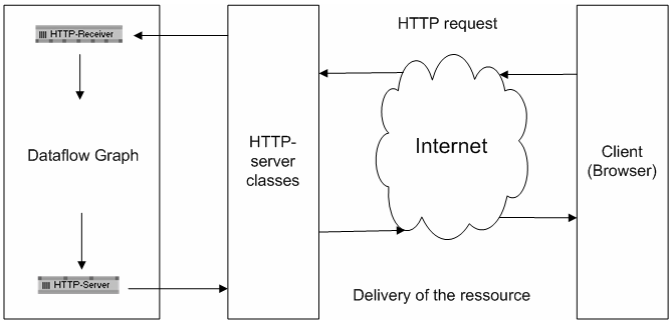

Fig. 2: The process of a HTTP request

6 A simple dynamic scenario

The following example demonstrates the setup mentioned in the previous section. The scenario in Fig. 3 shows the dataflow graph for the initialization of a simple user login verification. The graph provides a HTML page with a simple form included.

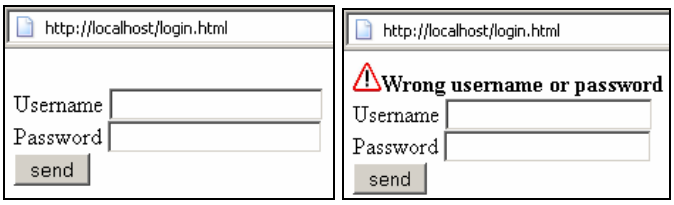

Fig. 3: A simple HTML form.

This page has the URL *login.html* assigned. Once the server receives a HTTP post requesting that URL (e.g. a user clicked the submit button), the HTTP receiver node reads and then outputs all transmitted HTTP header information including the typed username and password. The node *chkLoginData* is a sub-graph which verifies the incoming login information e.g. against a database query. Depending to the resulting boolean value, the graph selects as output either an error or a welcome page which will be served back to the client by the HTTP server node.

As one can see in Fig. 3, the error page contains an image which is also provided as a binary string on the bottom of the graph. The general flexibility becomes even more obvious by considering the welcome page to be dynamically concatenated, too.

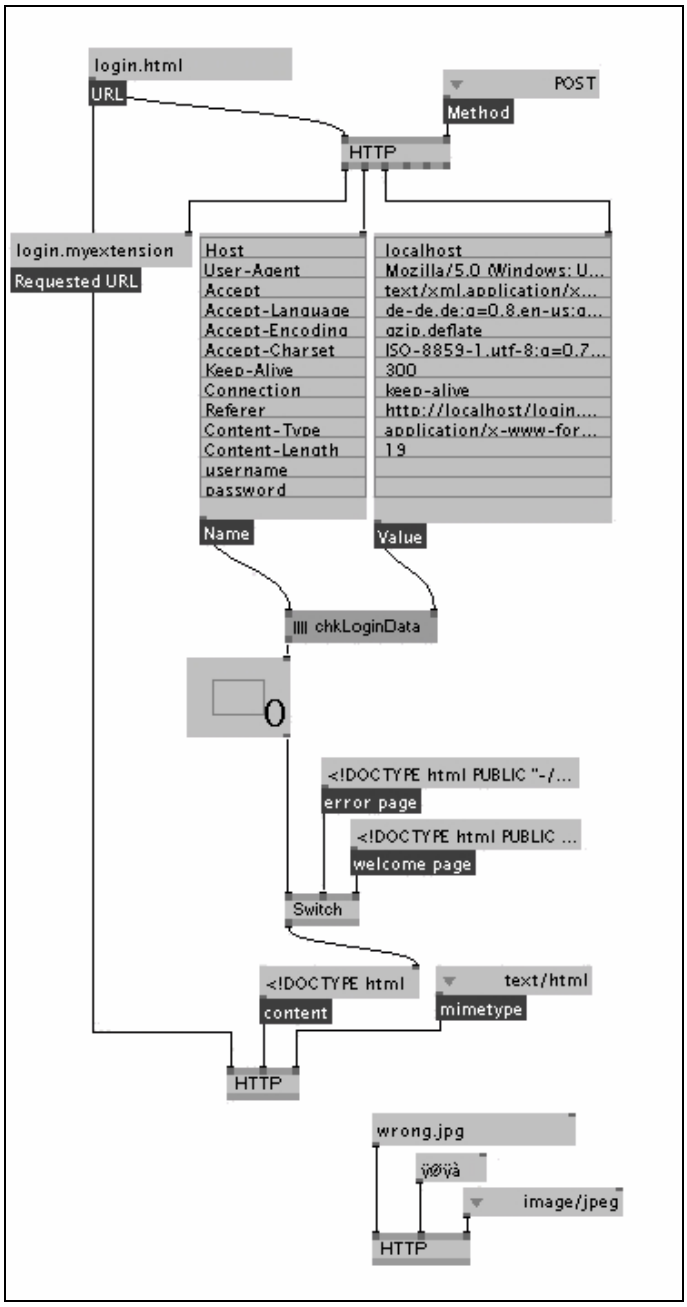

Fig. 4: The login verification graph.

Another aspect of the flexible HTTP adoption is the possibility to use a multitude of as many HTTP receiver nodes as needed. Furthermore it is possible to restrict their output by adjusting the *URL* and *HTTP Method* it listens to. Parameters, which result from requesting an URL like *http://localhost/login.html?username=MyUsername&password=MyPassword* will be listed the same way as the graph in Fig. 3 shows. Since the HTTP receiver outputs the complete requested URL it is also possible to define a special URL syntax and let the graph parse the necessary information.

7 Dataflow Based Markup Generation

All information transmitted during a web request consists of strings that match a certain pattern. Even a HTML page is a string that matches a pattern. In order to dynamically generate or manipulate these patterns, generic string operations are necessary. vvvv provides a wide range of common string operations in form of special nodes e.g. for concatenation, reduction, search and replacement of string tokens. In general it would be possible to generate HTML markup with these operations. However, it turned out to be extremely impractical as every single tag and their corresponding attributes have to be typed in manually.

Since XHTML [10] is an expedient extension of HTML which follows the XML syntax [11], it opens up new possibilities for manipulating web pages. XSLT [12], which itself follows the XML syntax, provides powerful operations for transforming XML strings. The XML syntax, however, consist of a nested set of tags. As demonstrated by others, representing a XML-tree graphically is most natural [13]. Some achievements in the area of graphical XML representation have been query languages such as XML-GL [14], VQL [15], and Xing [16]. Hence, we are certain that markup generation with a dataflow environment is the next step to be taken. Following the principle of separating the content of a web resource from its presentation, we pursue the approach of supplying both, (X)HTML nodes for generating markup, and CSS nodes for style allocation.

.

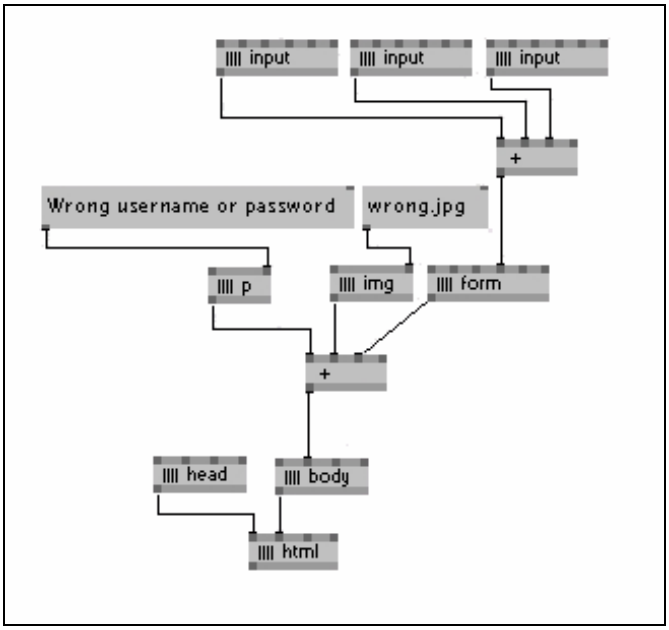

Fig. 5: A HTML tree concatenated by a dataflow graph

The simple HTML error page from the scenario in section 6 of this paper would be extended by the graph shown in Fig. 5.

CSS specifies a priority pattern to determine which style rules apply if more than one rule matches against a particular HTML element [17]. In the cascade, priorities are calculated and assigned to rules, so that the results are predictable. Writing CSS, however, requires the developer either to reconstruct given naming conventions or to define his own structure. In common use more complex projects lead the author to lose the overview of the CSS attributes. Often developers have to switch between multiple files in order to set the right order of precedence.

Our dataflow based approach frees the author from following the CSS cascading model. The dataflow model provides the correlation by visually linking the CSS nodes with the HTML nodes.

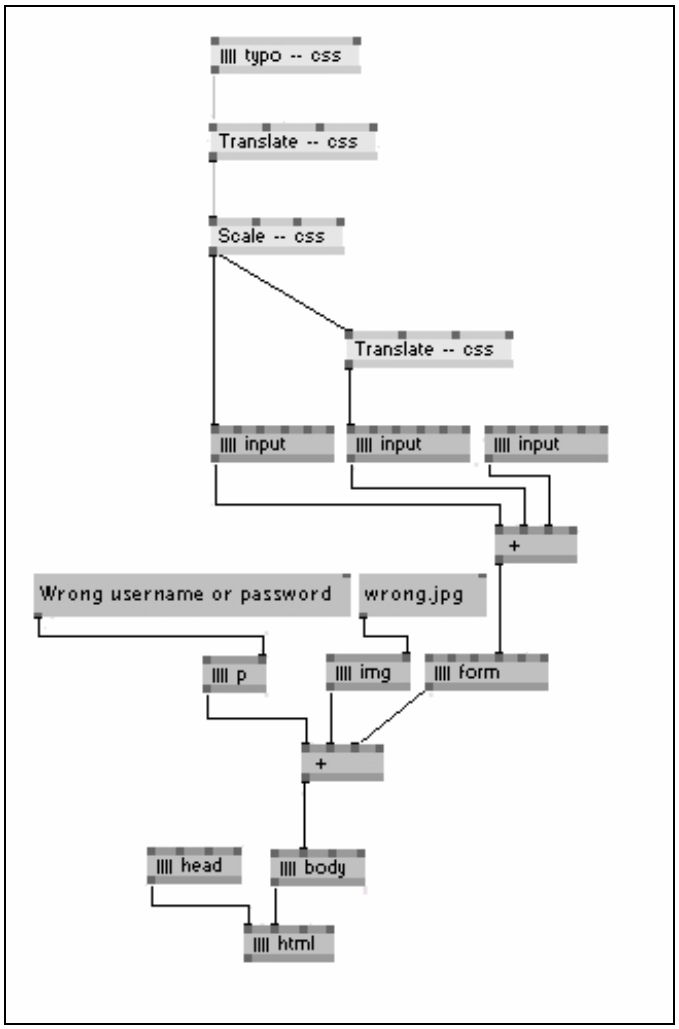

Fig. 6: A HTML tree with allocated CSS nodes

While the HTML nodes in Fig. 6 strictly follow the HTML elements name, the brighter CSS nodes are grouped to logical compositions of CSS properties. The HTML nodes functions are very simple. Each node encloses the incoming string in its equivalent html tags. The CSS nodes procedures are more complex. CSS nodes can inherit other CSS properties from connected CSS nodes above. Therefore internally the parameters are held in memory for every frame. A straightforward implementation would insert the CSS properties into the

html tags itself. Future implementations can move common elements automatically in the head section of the resulting HTML page.

8 Conclusion

In this paper, we have first introduced the dataflow based programming environment vvvv. We further explained the need for the right abstraction level by designing operation nodes for a dataflow graph. By breaking the server down to its two core functions (receiving requests and serving resources) we have ensured a logical dataflow perception without recusion. Starting from this basic setup we explained the resulting increased flexibility by going through small login verification scenario. Furthermore, we have presented an extension to XML markup nodes and CSS style allocation nodes. While the XML concatenation follows it own straight forward hierarchical conditions, we introduced the allocation of CSS statements in a new approach which passes on the CSS cascade priorities for the benefit of visual linking.

For the future we are also planning to implement nodes for XSLT stylesheet generation and expanding the server features to completely support HTTP 1.1.

We believe that dataflow based web development is a warrantable alternative to conventional methods as we have shown in this paper.

References:

[1] Nan C. Shu, *Visual Programming*, Van Nostrand Reinhold, 1988

[2] Margeret M. Burnett,, Marla J. Baker, A Classification System for Visual Programming, *Journal of Visual Languages and Computing Graphical Web-Server Programming*, No. 5, 1994, pp. 287-300.

[3] Cycling 74, *http://www.cycling74.com/products/maxmsp*, date: May 2006.

[4] National Instruments, *http://www.ni.com/labview/*, date: May 2006.

[5] Mauro Mosconi, Marco Porta, A visual approach to Internet Application Development, *Proceedings of the 8th International Conference of Human-Computer Interaction(HCT'99)*, Vol. 1, pp. 22-27, 1999

[6] Robert Baumgartner, Georg Gottlob, Marcus Herzog, Visual Programming of Web Data Aggregation Applications, *Eighteenth International Join Conference on Artificial Intelligence / Workshop on Information Integration on the web*, 1999

[7] Meso, official vvvv webpage, *http://vvvv.meso.net*, date: May 2006

[8] David Brüll and Björn Schwarzer, *Graphical Web-Server Programming*, Thesis, University of Applied Science Darmstadt, 2006

[9] Stefan Schiffer, *Visuelle Programmierung: Grundlagen und Einsatzmöglichkeiten*, Addison-Wesley-Longman, 1998

[10] W3C, XHTML 1.0, The Extensible HyperText Markup Language. W3C Recommendation, *http://www.w3.org/TR/2000/REC-xhtml1-20000126/*, 2000.

[11] W3C, XML 1.1, Extensible Markup Language. W3C Recommendation, *http://www.w3.org/TR/2004/REC-xml11-20040204/*, 2004.

[12] W3C, XSL Transformations (XSLT) Version 1.0. W3C Recommendation, *http://www.w3.org/TR/xslt*, 2004.

[13] Kang Zhang, Da-Qian Zhang, Yi Deng, A visual approach to XML Docuemtn Design and Transformation, *Proceedings of 2001 IEEEE Human-Centric Computing Language and Environments*, Stresa, Italy, 2001.

[14] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tancia, XML-GL: a graphical language for querying and representing XML documents, *Proc. 8th Int. World Wide Web Conf.*, 1999.

[15] K. Vadaparty, Y.A. Aslandogan, and G. Ozsoyoglu, Towards a unified visual database access, *Proc. ACM SIGMOD Conf. on Management of Data*, 357-366, 1993.

[16] M. Erwig, A Visual Language for XML, *Proc. 2000 IEEE Symp. on Visual Languages*, Seattle, USA, 47-54, 2000.

[17] W3C, Cascading Style Sheets, level 2, CSS2 Specification, *http://www.w3.org/TR/REC-CSS2/*, 1998.